

А. А. ПЯРНПУУ

ПРОГРАММИРОВАНИЕ  
НА  
АЛГОРИТМИЧЕСКИХ  
ЯЗЫКАХ

ИЗДАНИЕ ВТОРОЕ,  
ПЕРЕРЕБОТАННОЕ И ДОПОЛНЕННОЕ

*Допущено Министерством  
высшего и среднего специального образования СССР  
в качестве учебного пособия  
для студентов высших технических учебных заведений*

64616



МОСКВА «НАУКА»  
ГЛАВНАЯ РЕДАКЦИЯ  
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ  
1983

ВАСИЛИЙ ПИКА

22.18

П 99

УДК 519.6

**Программирование на алгоритмических языках.** Пярнпуу А. А. Учебное пособие. Изд. 2-е перераб. и доп.— М.: Наука. Главная редакция физико-математической литературы, 1983. — 320 с.

В книге излагаются основы программирования на алгоритмических языках алгол-60, фортран IV и ПЛ/1. Дано полное описание эталонного языка алгол-60, расширенного варианта языка фортран IV и почти полного языка ПЛ/1. Специальная глава содержит общие сведения о развитии ЭВМ, совершенствовании алгоритмических языков и принципов решения задач на ЭВМ. Рассмотрены особенности входных языков некоторых широко используемых трансляторов для ЭВМ БЭСМ-6 и ЕС ЭВМ.

Книга предназначена студентам вузов, слушателям факультетов повышения квалификации и самостоятельно изучающим программирование на алгоритмических языках.

Рис. 9. Табл. 1. Библ. 20 назв.

П  $\frac{1702070000-165}{053(02)-83}$  83-83

© Издательство «Наука»,  
Главная редакция  
физико-математической  
литературы, 1978, 1983

## ОГЛАВЛЕНИЕ

Предисловие . . . . .	5
<b>Глава I. Общие сведения . . . . .</b>	<b>7</b>
§ 1. Электронные вычислительные машины . . . . .	7
§ 2. Решение задач на ЭВМ . . . . .	14
§ 3. Понятия алгоритма и программы . . . . .	17
§ 4. Алгоритмические языки . . . . .	20
§ 5. Трансляция и входные языки . . . . .	27
<b>Глава II. Алгол-60 . . . . .</b>	<b>30</b>
§ 1. Элементарные конструкции языка . . . . .	30
§ 2. Описание типа . . . . .	37
§ 3. Простые выражения . . . . .	39
§ 4. Организация программы . . . . .	43
§ 5. Ввод и вывод информации . . . . .	45
§ 6. Оператор присваивания . . . . .	46
§ 7. Оператор перехода . . . . .	47
§ 8. Условие. Условный оператор . . . . .	49
§ 9. Полный условный оператор . . . . .	51
§ 10. Условие в выражениях . . . . .	54
§ 11. Пустой и составной операторы . . . . .	57
§ 12. Переключатели . . . . .	58
§ 13. Оператор цикла . . . . .	60
§ 14. Блочная структура программы . . . . .	66
§ 15. Собственные величины . . . . .	69
§ 16. Комментарии . . . . .	71
§ 17. Процедуры . . . . .	72
§ 18. Процедура-функция . . . . .	80
§ 19. Процедура без параметров . . . . .	82
§ 20. Рекурсия в процедурах . . . . .	84
§ 21. Побочный эффект при вычислении функций . . . . .	86
§ 22. Примеры программ . . . . .	88
<b>Глава III. Фортран IV . . . . .</b>	<b>91</b>
§ 1. Элементы языка . . . . .	91
§ 2. Выражения . . . . .	97
§ 3. Структура программы . . . . .	101
§ 4. Операторы описания типа . . . . .	102
§ 5. Операторы присваивания . . . . .	107
§ 6. Правила записи операторов . . . . .	109
§ 7. Операторы перехода . . . . .	110
§ 8. Условные операторы управления . . . . .	113
§ 9. Оператор цикла . . . . .	116
§ 10. Операторы останова и окончания . . . . .	119
§ 11. Библиотечные подпрограммы . . . . .	120
§ 12. Оператор-функция . . . . .	121
§ 13. Подпрограммы-функции . . . . .	122

§ 14. Подпрограммы . . . . .	130
§ 15. Оператор ввода . . . . .	135
§ 16. Оператор описания общих блоков . . . . .	137
§ 17. Оператор присваивания начальных значений . . . . .	141
§ 18. Подпрограмма данных . . . . .	142
§ 19. Оператор эквивалентности . . . . .	143
§ 20. Операторы ввода и вывода . . . . .	146
§ 21. Оператор формирования списков . . . . .	156
§ 22. Оператор задания формата . . . . .	158
§ 23. Классификация операторов и наименований . . . . .	173
§ 24. Примеры программ . . . . .	174
<b>Глава IV. ПЛ/1 . . . . .</b>	<b>178</b>
§ 1. Данные . . . . .	178
§ 2. Идентификаторы . . . . .	181
§ 3. Операторы . . . . .	184
§ 4. Описание данных . . . . .	185
§ 5. Выражения . . . . .	197
§ 6. Примечания . . . . .	200
§ 7. Структура программы . . . . .	201
§ 8. Оператор присваивания . . . . .	202
§ 9. Операторы управления . . . . .	203
§ 10. Блоки . . . . .	208
§ 11. Встроенные функции . . . . .	215
§ 12. Обработка прерываний . . . . .	217
§ 13. Распределение памяти . . . . .	223
§ 14. Описатели файла . . . . .	227
§ 15. Подготовка файлов к вводу-выводу . . . . .	230
§ 16. Ввод и вывод данных . . . . .	231
§ 17. Операторы ввода-вывода потока . . . . .	233
§ 18. Операторы ввода-вывода записей . . . . .	242
§ 19. Пример программы . . . . .	246
<b>Глава V. Краткие сведения о некоторых входных языках . . . . .</b>	<b>248</b>
§ 1. Структура и функции операционной системы . . . . .	248
§ 2. Подмножества языка алгол-60 . . . . .	252
§ 3. Система БЭСМ-алгол . . . . .	253
§ 4. Алгол-60 ОС ЕС ЭВМ . . . . .	261
§ 5. Фортран IV ОС ЕС ЭВМ . . . . .	270
§ 6. Фортран для ЭВМ БЭСМ-6 . . . . .	273
§ 7. Фортран ДОС ЕС . . . . .	282
§ 8. Фортран-Дубна и фортран ДОС ЕС . . . . .	288
§ 9. Стандарт фортран 77 . . . . .	292
§ 10. Подмножество ПЛ/1 ОС ЕС ЭВМ . . . . .	303
§ 11. ПЛ/1 ДОС ЕС . . . . .	305
Приложение I. Стандартные функции и процедуры алгола-60 . . . . .	308
Приложение II. Библиотечные подпрограммы фортрана IV . . . . .	309
Приложение III. Встроенные функции ПЛ/1 . . . . .	312
Литература . . . . .	317
Предметный указатель . . . . .	318

## ПРЕДИСЛОВИЕ

Роль электронных вычислительных машин (ЭВМ) в развитии современной науки и практически всех отраслей народного хозяйства возрастает с каждым годом. Решение многих актуальных задач, относящихся к самым различным областям науки и техники, было бы невозможным без применения ЭВМ. В связи с этим увеличивается количество лиц, прибегающих в своей работе к помощи ЭВМ. Курс программирования и алгоритмических языков слушают студенты и аспиранты, он является обязательным на факультетах повышения квалификации специалистов народного хозяйства, большое число научных работников и инженеров изучает программирование и алгоритмические языки самостоятельно.

В основе данной книги лежит курс лекций, который регулярно читается автором на факультете повышения квалификации Московского авиационного института. Первоначальный вариант этих лекций изложен в книге автора «Программирование на алголе и фортране», опубликованной в 1978 г. В последние годы программа упомянутого курса претерпела существенные структурные изменения, главным образом, вследствие включения в программу языка ПЛ/1 и сосредоточения внимания на ЭВМ серии ЕС. В результате в настоящее время курс содержит описание алгоритмических языков алгол-60, фортран IV, ПЛ/1 и трансляторов с этих языков для ЭВМ БЭСМ-6 и ЕС ЭВМ.

В этой книге сохранена структура курса лекций, однако изложение материала дается более подробно. Включение в одну книгу описаний трех популярных языков программирования объясняется тем, что большинство программистов пользуется именно этими языками.

Первая глава является вводной и содержит общие сведения об электронных вычислительных машинах и программировании для них.

Вторая и третья главы книги содержат описание языков программирования высокого уровня алгол-60 и фортран IV, при этом учтены дополнительные возможности этих языков, которые возникли уже после официальных сообщений [1, 2]. Трансляторы большинства современных ЭВМ в той или иной степени учитывают такие изменения. В то же время материал этих глав не связывается ни с каким конкретным транслятором.

В четвертой главе изложен язык ПЛ/1 с учетом почти всех его особенностей, не включены лишь некоторые наиболее сложные и несущественные для первоначального знакомства с языком элементы (например, средства асинхронного выполнения программы), которые читатель может почерпнуть из [3].

По мере возможности в книге подчеркиваются особенности каждого из рассматриваемых языков, выделяются их общие свойства, а также принципиальное отличие. Такое сопоставление может помочь программисту сделать

выбор в пользу одного из этих языков при решении конкретной задачи. Кроме того, знание языков алгол-60, фортран IV и ПЛ/1 полезно при изучении других языков программирования, например кобола или алгола-68. Изложение языков дано независимо друг от друга, т.е. при чтении книги можно опустить любую из глав II, III или IV.

Пятая глава посвящена краткой характеристике некоторых широко используемых версий языков для ЭВМ БЭСМ-6 и ЕС ЭВМ. Назначение этой главы — указать особенности входных языков различных трансляторов. Предполагается, что приводимых здесь сведений будет достаточно при составлении программы для указанных вычислительных машин. Тем не менее перед выходом на ЭВМ будущему программисту необходимо ознакомиться с соответствующими инструкциями данного вычислительного центра.

Как правило, вводимые в книге понятия сопровождаются конкретными примерами. Выполнение упражнений, содержащихся почти в каждом параграфе, должно помочь лучшему усвоению прочитанного материала и способствовать приобретению практических навыков программирования.

В списке литературы даны названия лишь тех источников, которые вышли в последние годы и могут быть использованы как пособия при изучении языков программирования и знакомстве с трансляторами. В тексте приведено несколько наиболее удачных учебных примеров из этих книг.

Автор признателен всем, кто ему помогал в отборе материала для книги и прочитал рукопись. Их замечания и пожелания учтены при отработке содержания окончательного варианта книги.

*А. Пярнпуу*

### § 1. Электронные вычислительные машины

В настоящее время трудно назвать все те области человеческой деятельности, успех которых не был бы связан с использованием электронных вычислительных машин. Нельзя представить себе сегодня ни одного крупного исследования в области физики без ЭВМ. Исследования космического пространства своими грандиозными достижениями в значительной степени обязаны ЭВМ. Без электронных вычислительных машин немислимо управление современными технологическими комплексами. Список подобных примеров можно продолжить. Такое широкое распространение ЭВМ объясняется способностью машин выполнять длинные последовательности операций без вмешательства человека и с большой скоростью выдавать точные результаты, хранить большой объем информации.

Рождение ЭВМ диктовалось прежде всего потребностями физики и инженерных наук. Для того чтобы решить задачи бурно развивающейся науки и техники, надо было производить астрономический объем вычислений. Вот и появился электронный арифмометр, делающий тысячи арифметических операций в секунду и предназначенный для автоматической обработки информации.

В машину исходная информация может быть введена в виде физических величин (например, электрических напряжений, пропорциональных действительным измеряемым величинам). Результаты в этом случае обычно выводятся на экран осциллографа или непрерывно выводятся в виде плавной кривой на самопишущие приборы. Схема такой машины строится в соответствии с математической моделью явления, т. е. системой уравнений, описывающих изучаемый процесс, поэтому эти машины называются *моделирующими* или *аналоговыми* вычислительными машинами (АВМ). Такого типа ЭВМ являются машинами непрерывного действия.

Другой тип ЭВМ — *цифровые* вычислительные машины (ЦВМ), или машины дискретного действия. В машинах такого типа вся информация представляется в виде цифровых кодов. ЦВМ часто называют универсальными, понимая под этим термином широкую сферу их использования. Среди машин этого типа тем не менее выделяют подтипы мини- и микро-ЭВМ, предназначенные для вполне определенного круга решаемых задач.

В последнее время получили распространение аналого-цифровые вычислительные системы, в которые входят аналоговая и цифровая вычислительные машины, включенные для параллельной работы с помощью устройства связи.

*Электронная вычислительная машина* в этой книге понимается как комплекс технических средств, предназначенных для автоматической обработки информации, заданной в явном цифровом виде, т. е. предметом данной книги является *программирование* для цифровых вычислительных машин (универсальных ЭВМ). Описание процесса обработки информации, как и сама информация, сообщается машине программой (отсюда и понятие программирование), первоначально записанной на бумаге в виде некоторого текста на языке, называемом алгоритмическим.

Информация, которая подвергается обработке, программа, промежуточные и окончательные результаты обработки хранятся в памяти ЭВМ, или запоминающем устройстве. Устройство, в котором выполняется большинство операций, связанных с обработкой информации, в частности арифметические операции над числами, хранящимися в машине, называется арифметическим устройством. Для ввода информации в машину служит специальное устройство ввода, а для вывода результатов работы машины — устройство вывода. Для автоматического управления работой всех устройств ЭВМ в соответствии с заданной программой служит устройство управления.

Как правило, ЭВМ обладает несколькими запоминающими устройствами, среди которых имеется по крайней мере одно устройство, предназначенное для приема, хранения и выдачи информации для арифметического устройства, т. е. тесно связано с арифметическим устройством. Такое запоминающее устройство называется оперативным (внутренним) запоминающим устройством (ОЗУ) или оперативной (внутренней) памятью. Остальные запоминающие устройства называют внешней (периферийной) памятью или внешними запоминающими устройствами (ВЗУ). Объем внешних запоминающих устройств превосходит объем внутренней памяти, но время обращения к ОЗУ значительно меньше, чем к внешней памяти.

Информация из оперативного запоминающего устройства обычно извлекается определенными порциями, поэтому для удобства пользования ОЗУ разбивается на части — ячейки, содержащие вполне конкретное число разрядов, куда с помощью двоичных знаков (т. е. нулей или единиц) может быть записано машинное слово.

*Машинное слово* представляет собой записанный в ячейке при помощи двоичных знаков код числа или код команды. *Команда* — это инструкция или приказ машине о выполнении одной операции. Совокупность всех операций, которые может выполнить ЭВМ, называется *системой команд* (машинных кодов) этой машины.

Средства выполнения арифметических и логических операций, средства управления выполнением заданной программой последовательности действий (команд), средства обращения к оперативной памяти и организация обмена информацией между ОЗУ и ВЗУ объединяют единым понятием *процессор* (иногда центральный процессор, имея в виду, что ЭВМ может иметь несколько процессоров). Таким образом, процессор составляют арифметическое устройство и устройство управления. В некоторых моделях ЭВМ с процессором конструктивно объединено и оперативное запоминающее устройство, что дает основание считать его составной частью процессора, но логически это два различных типа устройств.

Каждая ЭВМ характеризуется некоторыми основными технико-экономическими показателями, среди которых можно выделить быстрдействие (некоторое среднее число производимых в секунду операций в центральном процессоре), объем оперативной памяти (количество информации, одновременно хранимой во внутренней памяти), надежность и стоимость. Наиболее важными из них, по крайней мере для пользователя, являются первые два, так как от этих показателей прежде всего зависит эффективность работы ЭВМ. Совершенно очевидно, что универсальные ЭВМ должны обладать достаточным быстрдействием и большой памятью, а ЭВМ, имеющие специальное назначение, не обязаны иметь оба эти показателя высокими. Например, в ряде экономических задач, где обрабатываются большие массивы данных при не слишком большом числе операций, важен большой объем оперативной памяти; для машин с большой библиотекой встроенных подпрограмм, встречающихся в какой-либо определенной отрасли, важно быстрдействие.

В зависимости от уровня инженерно-технических средств, использованных при изготовлении ЭВМ, технико-экономические показатели ЭВМ разных моделей могут быть существенно различными. ЭВМ отличаются друг от друга элементной базой, конструктивно-технологическим исполнением, логической организацией, программным обеспечением, техническими характеристиками, степенью доступа к ЭВМ со стороны пользователей и др. В процессе развития ЭВМ основной тенденцией всегда было стремление повысить эффективность использования машин, облегчить связь операторов с ЭВМ, стремление упростить подготовку программ решаемых задач, уменьшить трудоемкость процесса переложения условия задачи (алгоритма) на язык машины.

В зависимости от перечисленных и ряда других характеристик условно принято выделять *поколения* ЭВМ. Поскольку улучшение технико-экономических показателей в значительной степени зависит от используемых для построения ЭВМ электронных схем, то принято считать, что критерием поколения ЭВМ служит их элементная база.

Важной характеристикой ЭВМ является также ее *программное обеспечение*. Программным (математическим) обеспечением принято считать совокупность программ, которыми снабжена ЭВМ, таких, что позволяют без программирования решать некоторые задачи, выполняют ряд работ при программировании, а также обеспечивают выгодный режим работы машины.

Основным активным элементом ЭВМ первого поколения является электронная лампа. Появление машин первого поколения относится к пятидесятым годам нашего века. Типичные представители этого поколения из отечественных ЭВМ — это БЭСМ-1, Минск-1, Урал-1, Урал-2, Урал-4, М-1, М-3, БЭСМ-2, Стрела и др. Они были значительных размеров, потребляли большую мощность, имели невысокую надежность работы и слабое программное обеспечение. Быстрдействие их не превышало 2—3 тысяч операций в секунду, емкость оперативной памяти — 2К или 2048 машинных слов (1К = 1024) длиной 48 двоичных знаков. В 1958 г. появилась машина М-20 с памятью 4К и быстрдействием около 20 тысяч операций в секунду.

В машинах первого поколения были реализованы основные логические принципы построения машин и продемонстрированы возможности цифровой вычислительной техники.

В ЭВМ второго поколения элементной базой служат транзисторы. Появление полупроводниковых элементов в электронных схемах существенно увеличило емкость оперативной памяти, надежность и быстродействие ЭВМ. Уменьшились размеры, масса и потребляемая мощность.

С появлением машин второго поколения значительно расширилась сфера использования электронной вычислительной техники, главным образом за счет развития программного обеспечения. Появились также специализированные машины, например ЭВМ для решения экономических задач, для управления производственными процессами, системами передачи информации и т. д.

К ЭВМ второго поколения относятся Урал-14, Урал-16, Минск-22, Минск-32, БЭСМ-3, БЭСМ-4, М-220, М-222, БЭСМ-6, МИР-2, Наири, Рута и др. Первые четыре из названных используются главным образом для решения экономических задач, последние три относятся к подтипу мини-ЭВМ, остальные являются универсальными. ЭВМ БЭСМ-4, М-220, М-222 имеют быстродействие порядка 20—30 тысяч операций в секунду, а оперативную память — соответственно 8К, 16К и 32К.

Среди машин второго поколения особо выделяется БЭСМ-6, обладающая быстродействием около миллиона операций в секунду и оперативной памятью от 32К до 128К (в большинстве машин используется два сегмента памяти по 32К каждый).

Элементная база ЭВМ третьего поколения основана на микроэлектронике, и это поколение характеризуется широким применением интегральных схем. Интегральная схема — это логически законченный функциональный блок, соответствующий достаточно сложной транзисторной схеме. Размеры такого блока невелики, из них собираются более сложные узлы методом печатного монтажа. Благодаря интегральным схемам удалось существенно улучшить технико-эксплуатационные характеристики ЭВМ. Например, машины третьего поколения по сравнению с машинами второго поколения имеют больший объем оперативной памяти, увеличилось быстродействие, повысилась надежность, а потребляемая мощность, занимаемая площадь и масса уменьшились.

К машинам третьего поколения относятся все находящиеся сейчас в серийном производстве ЭВМ Единой системы (ЕС-1010, ЕС-1020, ЕС-1030, ЕС-1040, ЕС-1050, ЕС-1060 и несколько промежуточных модификаций — ЕС-1021 и др.) и агрегатная система средств вычислительной техники. ЭВМ ЕС разработаны и промышленно выпускаются совместными усилиями стран-членов СЭВ. Все машины Единой системы по принципам функционирования и взаимодействия основных элементов, по составу периферийного оборудования, которое может быть подключено к центральному процессору, представляют собой программно-совместимые ЭВМ, предназначенные для решения широкого круга научно-технических, экономических задач и использования в автоматизированных системах управления (АСУ).

ЭВМ ЕС-1010 разработана в ВНР и имеет быстродействие до 10 тысяч операций в секунду, а объем оперативной памяти — от 8К до 64К байт, байт состоит из восьми битов, где бит — единица информации, определяемая как количество информации, которое содержится в сообщении, состоящем из одного двоичного знака,

ЕС-1021 создана в СССР. Она имеет некоторые конструктивные особенности, так как проектировалась главным образом как машина для управления технологическими процессами. Быстродействие ее 15 тысяч операций в секунду, объем оперативной памяти — от 16К до 64К байт.

Машины ЕС-1010 и ЕС-1021 являются малыми моделями ЕС ЭВМ (мини-ЭВМ), все остальные — универсальные ЭВМ, причем с возрастанием номера модели, как правило, мощность машины растет и ее технико-экономические показатели улучшаются. В частности, модели ЕС-1050 и ЕС-1060 разработаны в СССР. Среднее быстродействие первой из них 500 тысяч операций в секунду, второй — 1,0—1,3 миллиона операций в секунду, объем оперативной памяти этих моделей соответственно: от 256К до 1204К байт и от 2048К до 8192К байт.

Все электронные вычислительные машины третьего поколения, помимо элементной базы, существенно отличаются от ЭВМ второго и первого поколений и другими характеристиками.

Прежде всего, ЭВМ третьего поколения оперируют с произвольной буквенно-цифровой информацией. Единичей адресации памяти является байт, в котором может храниться восьмизрядный двоичный код, представляющий собой либо две десятичные цифры, либо один алфавитный символ. В соответствии с этим изменилась система команд ЭВМ и появилась возможность каждую ячейку памяти полностью загрузить информацией. По этой же причине объем оперативной памяти ЭВМ третьего поколения указывается не в К машинных 48-разрядных словах, а в К байтах.

В машинах третьего поколения машинное слово стандартной длины содержит 4 байта. Рассматриваются также полуслова (длина 2 байта) и двойные слова (8 байт).

В отличие от ЭВМ первого и частично второго поколения, все основные устройства которых, такие, как центральный процессор и устройства ввода или вывода, работают последовательно, современные машины третьего поколения обладают возможностью параллельной работы устройств. Появились многопрограммные ЭВМ, которые, в отличие от однопрограммных машин, в которых программы выполняются только поочередно, могут одновременно реализовывать несколько программ (принцип мультипрограммирования) именно за счет организации параллельной работы основных устройств машины.

Принцип мультипрограммирования, совместимость систем программирования, т. е. систем методов и приемов обеспечения удобного и быстрого обмена информацией между человеком и машиной на уровне машинных кодов, высокий уровень технической стандартизации и унификации — все эти качества присущи ЕС ЭВМ.

Следствием реализации принципа мультипрограммирования является возможность работы в режиме разделения времени и в режиме диалога, при котором несколько пользователей, удаленных от ЭВМ на достаточно большие расстояния, могут общаться с ней непосредственно, оперативно и независимо друг от друга. Взаимодействие удаленных пользователей с ЭВМ осуществляется с помощью периферийной аппаратуры, подсоединенной к каналам связи через стандартную систему сопряжения. Благодаря этому центральный

процессор может работать с большим набором разнообразных периферийных устройств (например, экранных пультов управления и др.).

В настоящее время строятся ЭВМ четвертого поколения, основанные на применении больших интегральных схем, представляющих собой совокупность нескольких десятков электрических цепей, образованных в одном массиве полупроводникового материала (кремниевых пластин) и объединенных внутренними связями в единый функциональный блок. Высокая степень интеграции способствует увеличению плотности компоновки электронной аппаратуры, повышению ее надежности, что ведет к увеличению быстродействия ЭВМ и снижению ее стоимости. Все это оказывает существенное воздействие на логическую структуру (архитектуру) ЭВМ и на ее программное обеспечение. Более тесной становится связь структуры машины и ее программного обеспечения, особенно *операционной системы* (или монитора)—набора программ, которые организуют непрерывную работу машины без вмешательства человека.

ЭВМ четвертого поколения обеспечивают коммунальное использование вычислительных мощностей. Они связаны в единую вычислительную сеть и обеспечивают работу в режиме разделения времени. В СССР в настоящее время разрабатывается многопроцессорный вычислительный комплекс «Эльбрус». Рекордное быстродействие, достигаемое ЭВМ четвертого поколения, составляет несколько десятков миллионов операций в секунду, а объем оперативной памяти — до 16 тысяч К байт.

Для ЭВМ пятого поколения, которые разрабатываются в настоящее время, элементная база основывается на оптико-электронных элементах. Для этих машин носителями энергии служат не электроны, а фотоны, что значительно повышает скорость передачи сигналов, так что быстродействие ЭВМ может достигнуть сотен миллионов операций в секунду. Преобразование электрических сигналов в оптические в таких машинах осуществляется лазерами и светоналучающими диодами, используются световоды и фотоприемники. Дальнейшее развитие получает начавшийся еще в четвертом поколении процесс образования вычислительных систем, сращивания машин и вычислительных центров с системой связи. Меняется и представление о системе связи, которая в дальнейшем пользователю должна оказывать не только услуги передачи информации, но и ее хранения и обработки.

Элементная база ЭВМ последующих поколений просматривается менее уверенно, но по прогнозам в следующем поколении появятся машины с быстродействием до миллиарда операций в секунду. Эта скорость будет повышаться благодаря параллельной работе всех устройств, в том числе нескольких процессоров. По всей видимости, станет возможным осуществление параллельного преобразования информации типа той, которая представляется в виде голограмм с помощью систем лазерных элементов. Небезынтересными являются также прогнозы по разработке соответствующих «вычислительных сред» и ряд других, кажущихся сейчас фантастическими, проектов вычислительных машин.

К настоящему времени скорость выполнения операций в центральном процессоре уже достаточно высока, однако полное ее использование сдерживается в определенной степени медленностью периферийных устройств,

Правда, современные операционные системы с многими каналами ввода и вывода позволяют частично решить эту проблему. В то же время широкая область использования ЭВМ обеспечивается в значительной степени именно благодаря тому, что разработано много различных периферийных устройств. К таким устройствам, которые необходимы для использования машин независимо от областей их применения, относятся внешние запоминающие устройства на магнитных барабанах, лентах и дисках, традиционные устройства ввода и вывода информации через перфокарты и перфоленты, алфавитно-цифровые печатающие устройства (АЦПУ), электрифицированные пишущие машинки. Ряд устройств расширяет возможности использования машин в самых различных областях. Это устройства, обеспечивающие дистанционную обработку информации, например абонентские пункты для пользователей, устройства, облегчающие общение человека с машиной, например различные операторские пульты со средствами наглядного отображения алфавитно-цифровой и графической информации на электронно-лучевых трубках — дисплеи и др.

Внешние запоминающие устройства используются для хранения исходной информации, отдельных частей (или всей) выполняемой программы, вспомогательных программ, а также промежуточных и конечных результатов, если объем получаемой информации настолько велик, что не может разместиться в ОЗУ, или полученная в данный момент информация может понадобиться в дальнейшем. Современные ВЗУ имеют, как правило, емкость памяти, намного превосходящую емкость оперативной памяти ЭВМ. Так, магнитный барабан ЕС-5033 имеет емкость 5,6 Мбайт, скорость записи считывания 1200К байт в секунду, среднее время доступа к информации 21 мс, магнитная лента ЕС-5014 имеет емкость 40 Мбайт, скорость записи/считывания 126К байт в секунду, время разгона и останова ленты 5 мс, номинальная скорость движения ленты 2 метра в секунду. Стационарный пакет дисков ЕС-5051 имеет емкость 100 Мбайт, скорость записи/считывания 83,3К байт в секунду, среднее время доступа к информации 420 мс, а сменный пакет дисков ЕС-5050 имеет соответственно следующие параметры: 7,25 Мбайт, 156К байт в секунду, 90 мс. Базовый комплект ВЗУ для ЕС-1050 составляют 4 комплекта магнитных дисков и 16 магнитофонов.

Скорость считывания для устройства ввода с перфокарт ЕС-6012 составляет до 500 карт в минуту, для устройства ЕС-6013 — до 1200 карт в минуту, а для устройства ЕС-6014 — до 2000 карт в минуту. Алфавитно-цифровое печатающее устройство АЦПУ-128-6 печатает до 900 строк в минуту. Периферийные устройства ЭВМ постоянно совершенствуются.

Первоначально электронные вычислительные машины использовались только в тех сферах, где было необходимо выполнить большое количество арифметических операций, например при обработке бухгалтерских данных или при расчетах различных вариантов в отраслях техники, в промышленности. В последнем случае создание, например, нового технологического комплекса требует согласования многих взаимосвязанных, порой противоречивых условий. Машина помогает исследовать много вариантов в зависимости от исходных данных. Сопоставление различных по выбранному параметру (например, минимальным затратам на осуществление проекта) вариантов

позволяет выбрать оптимальный. Такой выбор тоже можно поручить машине. В научно-исследовательской работе с помощью ЭВМ могут быть исследованы методы решения математических задач, изучены свойства различных физических, биологических, экономических, социальных систем и других объектов, выявлены зависимости от конкретных факторов и параметров.

Благодаря ЭВМ эффективность работы математиков-вычислителей увеличилась в десятки тысяч раз. Это привело к новому качественному скачку. Стало возможным решать задачи, которые совсем недавно даже не ставились, однако стиль работы сначала во многом напоминал традиционный — как правило, рассматривалась конкретная, хорошо поставленная задача, и специалист-математик строил алгоритм ее решения. В то же время сложные проблемы трудно формализовать, т. е. свести к хорошо поставленной задаче. Кроме того, эффективность алгоритма требует вмешательства человека на различных этапах его реализации. Таким образом, вместо традиционного анализа задачи возникает система, куда входят человек, т. е. сам исследователь, ЭВМ как носитель и средство получения и хранения информации и система программного обеспечения — аппарат, позволяющий исследователю активно вмешиваться в процесс исследования. Такая система — это уже инструмент совершенно нового качества.

В связи с этим качественно изменился характер решения многих старых и новых проблем таких, как создание автоматизированных систем управления (АСУ), моделирование физических процессов, научное прогнозирование (например, ожидаемых экономических ситуаций по данным темпов производства и реализации продукции с учетом возможных действий партнеров и конкурентов) и другие задачи.

И, наконец, ЭВМ используются в так называемых невычислительных приложениях. Это — перевод с одного языка на другой, в том числе и трансляция (перевод с алгоритмического языка на язык конкретной машины), моделирование работы одной ЭВМ на другой (на этапе проектирования), моделирование «искусственного интеллекта» и др.

## § 2. Решение задач на ЭВМ

При решении любой задачи на электронной вычислительной машине предполагается, что некоторая информация подвергается обработке по предварительно составленной инструкции, называемой программой. Поэтому под решением задачи на ЭВМ подразумевается гораздо больший круг действий, чем только работа машины.

На первом этапе выбирается общий подход к решению, устанавливается, каким целям должно служить решение задачи и при каких условиях оно будет существовать. Здесь производится разбиение задачи на более мелкие, определяется последовательность их решения и выполняется ряд других действий, т. е. осуществляется общая постановка задачи. Например, задача изучения структуры ударной волны в газе требует четкого определения, каковы рассматриваемые параметры газа (плотность, температура и др.), какая будет исследована волна (в аэродинамической трубе или перед сверхзвуковым снарядом в атмосфере Земли) и т. д.

После этого этапа необходимо дать математическое описание задачи. Появление ЭВМ способствовало развитию приложений математики, что привело к математизации различных отраслей науки. Для того чтобы можно было интересующее нас явление подвергнуть математическому анализу, должны быть выполнены некоторые условия, должна существовать математическая теория явления, описывающая его закономерности в виде формул. Такой набор формул называют математической моделью явления. Лишь в очень простых и редких случаях математическая модель является в то же время и расчетной схемой, т. е. позволяет по имеющимся исходным данным получить требуемые результаты. Модель определяет искомые величины, как правило, неявно, в виде системы зависимостей, которым эти величины должны удовлетворять, но такая система зависимостей обычно неполна, она оставляет математику большую или меньшую свободу выбора. Выбор надо произвести так, чтобы получить наилучшее значение некоторого критерия. Указание этого критерия также входит в математическую модель.

Когда критерий выбран или написана полная система уравнений, определяющих исследуемое явление, задача становится уже чисто математической — задача математически поставлена.

Однако при решении многих практических задач приходится огрублять и упрощать их постановку, чтобы суметь воспользоваться теми методами решения, которые может предложить современная математика, или же разработать новый метод решения.

За математической постановкой и разработкой метода решения следует реализация выбранного метода на ЭВМ. Конечный результат предыдущих постановочных этапов — метод решения задачи — описан математическим языком, т. е. достаточно формально. Программа для ЭВМ — это также формальное описание того же метода. Казалось бы, превратить одно формальное описание в другое — дело несложное и тоже формальное, т. е. может быть выполнено механически. Это верно лишь до некоторой степени. Программист хорошо чувствует, насколько приблизительно и неполно описывают математики методы решения даже чисто вычислительных задач. Это связано с тем, что математик, как любой автор, рассчитывает на известный уровень знаний, опыта и интуиции своего читателя. Все эти качества присущи лишь человеку. Машина обладает тоже подобием знаний и опыта, но в совсем иной форме, чем человек. Для разговора с ЭВМ человек вынужден пользоваться или ее собственным языком, или в лучшем случае языком, значительно более формализованным, а следовательно, и более бедным, чем естественный человеческий язык, которым пользуются математики.

С другой стороны, язык вычислительной машины хотя и формален, но далеко не абстрактен. В языке ЭВМ, являющемся весьма конкретным, должны быть отражены принципиальные особенности машины, например отсутствие понятия непрерывности, конечность величин, определенные законы машинной арифметики. Так, арифметические операции в вычислительной машине, строго говоря, имеют не так уж много общего с математическими операциями над вещественными числами. ЭВМ выполняет приближенные операции над приближенными представлениями вещественных чисел, поскольку точно представить все вещественные числа в действительном мире конеч-

ных вычислительных машин невозможно. Определенные способы приближенного представления чисел и действий с ними встроены в конструкцию ЭВМ.

Итак, этап реализации метода решения содержит в себе исследование математической модели явления и переход от нее к расчетной схеме, включая анализ с оценкой погрешности результатов, собственно программирование — перевод схемы на язык машины, когда формируется последовательность операций, выполняемых машиной, и, наконец, контроль за выполнением программы машиной.

Процесс программирования состоит из двух частей: составления блок-схемы программы — графического изображения последовательности действий и написания программы на языке, который ЭВМ понимает непосредственно или с помощью транслятора, т. е. программы-переводчика.

Программу в простейшем случае можно определить как последовательность предложений, написанных на некотором произвольном языке и допускающую однозначность толкования. Например, последовательность предложений:

*заданы* ( $f_1 = 1, f_2 = 2, f_3 = 3$ );

*сложить* ( $f_1, f_2, S_2$ ); *сложить* ( $f_3, S_2, S_3$ );

есть программа вычисления суммы  $S_3 = f_1 + f_2 + f_3$ , написанная на языке, в котором слова *заданы*, *сложить*, цифры, буквы, скобки, запятая, точка с запятой и знак равенства машиной понимаются однозначно. Вопросы, связанные с составлением блок-схемы программы и уточнения понятия программы, будут рассмотрены в следующем параграфе.

Контроль за выполнением программы машиной включает в себя отладку программы и получение результатов. Отладка необходима, поскольку вероятность допустить ошибку при написании программы очень велика. Для обнаружения и устранения ошибок используется сама вычислительная машина. Под отладкой понимается не только устранение ошибок, допущенных при записи программы, но и процесс совершенствования (оптимизации) программы.

Последний этап решения задачи — обработка результатов. Продолжительность его зависит от конкретной задачи. Математик будет интерпретировать полученные данные в соответствии с поставленной целью и иногда все делать заново. В обработке результатов, так же как и в их получении, участвует и ЭВМ (рисует графики, выводит данные на телевизионный экран и т. д.).

Таким образом, в результате прохождения всех этапов решения задачи в машине реализуется некоторый процесс. Таковым может быть процесс, отражающий физическое явление, биологический процесс, процесс, происходящий в общественной сфере. При этом ЭВМ не освобождает пользователя от понимания сущности задачи, так как он должен уметь «поставить процесс на машину», что требует глубокого знания особенностей задачи. Это значит, что он должен научиться описывать процесс на языке математики, строить математическую модель.

И, наконец, можно сделать вывод, что программирование — это лишь один из этапов решения задачи на ЭВМ.

### § 3. Понятия алгоритма и программы

Процесс постановки задачи для вычислительной машины в определенной степени аналогичен постановке задачи человеку, производящему вычисления карандашом на бумаге. Однако если во втором случае это можно успешно сделать средствами обычного разговорного языка с привлечением тех понятий, которые используются в математической литературе, то для ЭВМ словесное описание оказывается громоздким, а главное, недостаточно четким и строгим.

Возникает необходимость в создании аппарата для полного, ясного и однозначного описания вычислительных процессов. Это описание должно содержать формулы, по которым происходит расчет, определять последовательность их применения, условия, при которых используется та или иная формула, а также указывать правила перехода от одной формулы к другой, от одной части вычислительного процесса к другой части.

После того как машиной получено задание, реализация вычислительного процесса внутри ЭВМ происходит автоматически без вмешательства человека. Аппарат для описания процесса должен содержать сведения, как поступить машине при любых обстоятельствах, которые могут возникнуть во время ее работы, т. е. заранее необходимо предусмотреть все возможные ситуации. Машина является аккуратным исполнителем задания, но она не может принять решение, если программист не дал ей соответствующие указания. ЭВМ необходимо указать характер и свойства тех или иных математических объектов — чисел, векторов, матриц и т. д., которые служат исходными данными для решения задачи или используются в ходе вычислений.

64616  
Таким образом, при оформлении задания машине возникает необходимость в точном и полном описании определенного вычислительного процесса или любой иной последовательности действий, выполняемых ЭВМ. Конструктивное описание, состоящее из конечного множества правил и определяющее процесс переработки данных, называется операционным правилом обработки или *алгоритмом*. Следовательно, алгоритм задает некоторый свод правил, описывающих процесс, протекающий во времени и определяющий последовательность действий для перехода от исходной ситуации к желаемому новому состоянию. Описание такого свода правил может быть выполнено на обычном языке, с помощью математических формул или же специальных символов.

Правила обработки информации для ЭВМ задаются в *программе* — последовательности предложений, написанных на некотором, произвольном, понятном ЭВМ языке, допускающей однозначность толкования и реализующей данный алгоритм, т. е. программа есть запись алгоритма для решения задачи на ЭВМ. Разработка алгоритма для решения любой задачи является наиболее ответственным и важным моментом, так как именно алгоритм определяет ту последовательность действий, которая выполняется машинной. Ошибки, допущенные при записи алгоритма, обычно приводят к неверному ходу вычислительного процесса и, следовательно, к неверному результату. Результат может быть верным, но получен не оптимальным путем, если используется алгоритм, не учитывающий индивидуальных особенностей задачи.

Основная задача при эффективном использовании ЭВМ — построение хорошего алгоритма.

Рассмотрим построение алгоритма и составление программы на простом примере вычисления значений  $x$ , удовлетворяющих квадратному уравнению  $ax^2 + bx + c = 0$ .

Для человека, изучившего методы нахождения корней квадратного уравнения, задача понятна при произвольных значениях коэффициентов  $a$ ,  $b$ ,  $c$ . Для ЭВМ необходимо расписать последовательность всех действий, причем нужно предусмотреть все возможные ситуации, возникающие вследствие тех или иных значений коэффициентов, которые полагаются вещественными, т. е. могут быть положительными, отрицательными или нулевыми.

Существуют различные методы решения этой задачи. В частности, можно пользоваться привычными формулами:

$$x_1 = -\frac{b}{2a} + \frac{\sqrt{b^2 - 4ac}}{2a}, \quad x_2 = -\frac{b}{2a} - \frac{\sqrt{b^2 - 4ac}}{2a},$$

которые будут лежать в основе алгоритма, так как они указывают последовательность действий при решении задачи. После подстановки в эти формулы числовых значений  $a$ ,  $b$ ,  $c$  производятся вычисления и получается решение в виде двух чисел. Остается предусмотреть исключительные ситуации, в противном случае составленная программа не будет полностью описывать вычислительный процесс, т. е. не будет универсальной.

Исключительными могут быть следующие ситуации:

1)  $a = 0$ . Уравнение перестает быть квадратным. Приведенные формулы неприменимы, и единственный корень линейного уравнения  $bx + c = 0$  вычисляется по формуле:  $x = -c/b$ . Случай  $b = 0$  (при  $a = 0$ ) требует, чтобы при этом и  $c = 0$ , т. е. уравнение вообще отсутствует и задачи как таковой нет. Однако, учитывая, что в большинстве практических задач входящие в них параметры суть результат физических измерений, отбрасывать случай, когда  $a = b = 0$ , не следует. Он может указывать, в частности, на сбой в процессе измерений или особенность системы при некоторой комбинации параметров и т. д.

2)  $b^2 - 4ac \leq 0$ . В случае равенства нулю дискриминанта уравнение имеет два совпадающих корня; если же подкоренное выражение отрицательное, то корни комплексные и вычислять квадратный корень следует из значения этого выражения с обратным знаком.

Блок-схема программы, т. е. графическое изображение последовательности действий при реализации алгоритма, составляется из элементов в виде геометрических фигур, имеющих по предварительному соглашению вполне определенный однозначный смысл и обозначающих различные действия. Эти элементы соединяются между собой стрелками, указывающими направления переходов от одних действий (операций) к другим. Будем считать, что элемент в виде ромба означает проверку некоторого условия или сравнение, в виде прямоугольника — вычислительные операции любого вида, операции ввода и вывода данных обозначим фигурами в виде профиля бочонка. Внутри фигур (блоков, блок-схем), как правило, указываются те операции, которые

в данном блоке выполняются. Для построения блок-схемы могут использоваться и другие геометрические фигуры.

Наименования величины в программе можно выбрать произвольно. Обычно применяются такие обозначения, которые подсказывают, что данная

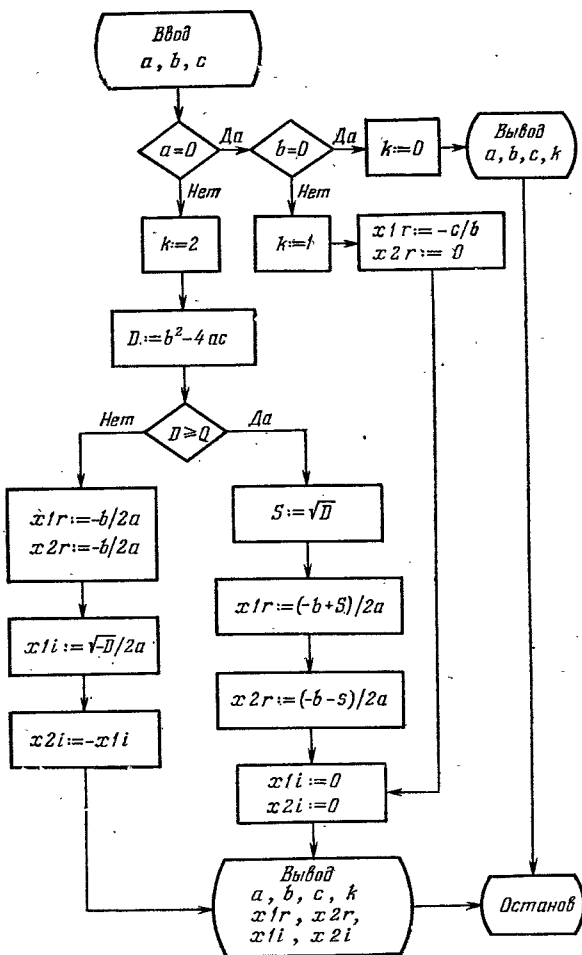


Рис. 1.

переменная в действительности означает. Исходя из этого принципа, наименования величин, являющихся коэффициентами  $a$ ,  $b$ ,  $c$  уравнения, оставим прежними. Для обозначения двух корней введем четыре числа, представляющие собой вещественные ( $x1r$ ,  $x2r$ ) и мнимые ( $x1i$ ,  $x2i$ ) части первого и второго корней соответственно. Если корни вещественные, то  $x1i = x2i = 0$ , если корни совпадают —  $x1r = x2r$ . Подкоренное выражение понадобится

дважды, есть смысл обозначить его одним символом, например  $D$ , и вычислить один раз; можно обозначить также  $\sqrt{D}$  по той же причине, например, символом  $S$ . Еще введем константу  $k$ , значение которой полагается равным нулю, если уравнение вырожденное ( $a = b = c = 0$ ), равным 1, если уравнение линейное; в случае квадратного уравнения  $k = 2$ . Значение  $k$  будет составной частью решения вместе со значениями  $x_{1r}$ ,  $x_{2r}$ ,  $x_{1i}$ ,  $x_{2i}$ .

Для указания конкретных операций, выполняемых в блоке, обычно используются математические символы или записи в виде текста. Употребляются также некоторые специальные символы, например символ  $:=$  означает, что указанное в блоке значение должно быть заменено новым.

Блок-схема программы для рассматриваемой задачи может быть изображена в таком виде, как показано на рис. 1.

Отметим, что из блоков, обозначенных ромбами, всегда выходит больше одной стрелки, т. е. эти блоки вызывают разветвление программы. Случай  $D = 0$  целесообразно рассматривать как случай вещественных корней. В первых, отпадает третья ветвь в условии  $D \cong 0$ , во-вторых, точное равенство нулю дискриминанта маловероятно, поскольку значения  $a$ ,  $b$ ,  $c$  берутся из физических измерений. Ненужная операция вычисления квадратного корня из нуля и двух нулевых значений  $x_{1i}$ ,  $x_{2i}$  в тех редких случаях, когда  $D = 0$ , потребует меньше машинного времени, чем проверка в каждом варианте равно ли  $D$  нулю.

Одинаковые операции для различных ветвей программы могут быть изображены на блок-схеме одним блоком, т. е. только один раз. В таком случае в один блок входит больше одной стрелки. В рассматриваемом примере по две стрелки направлены в блок вычисления нулевых значений мнимых частей корней в случае вещественных корней и в блок вывода в случаях  $k = 1$  и  $k = 2$ .

Оформление алгоритма в виде блок-схемы на этом закончено. Остается записать его в виде программы на языке, понятном вычислительной машине. Программа вычисления корней квадратного уравнения с вещественными коэффициентами на языке алгол будет дана в гл. II, на языке фортран — в гл. III, а на языке ПЛ/1 — в гл. IV.

### Упражнения

1. Составить блок-схему для решения системы уравнений
 
$$a_{11}x_1 + a_{12}x_2 = b_1, \quad a_{21}x_1 + a_{22}x_2 = b_2$$
 с учетом обстоятельства, что система может иметь единственное решение, бесконечное множество решений или вовсе не иметь решения.
2. Дана прямая  $y = ax + b$  и окружность  $x^2 + y^2 = r^2$ . Исследовать возможности наличия общих точек у этих линий и составить блок-схему вычислений.
3. В  $n$ -мерном пространстве даны векторы  $x$  и  $y$  с компонентами  $x_1, x_2, \dots, x_n$  и  $y_1, y_2, \dots, y_n$ . Найти вектор  $z$  вида  $x + ky$ , ортогональный вектору  $y$ . Составить блок-схему для решения этой задачи.

## § 4. Алгоритмические языки

Развитие вычислительной техники обусловило появление языков программирования. Назначение такого языка — в оснащении набора вычислительных формул дополнительной информацией, которая превращает этот

набор в алгоритм. В дальнейшем под языком программирования понимается язык для составления программ, т. е. язык, на котором записывается алгоритм для решения задачи на ЭВМ.

Программирование для ЭВМ первого поколения велось исключительно на *машинном языке*. Машинный язык представляет собой свод правил кодирования в числовом виде определенных действий (в большинстве арифметических). Для всех машин понятна только двоичная система счисления, которая, однако, для сокращения записи программистами заменялась восьмеричной.

Под системой счисления обычно понимается совокупность приемов наименования и обозначения чисел. Обычная система записи чисел представляет собой позиционную десятичную систему счисления в соответствии с тем, что от позиции, занимаемой любой из десяти используемых в этой системе цифр, зависит ее числовое значение. Двоичная система счисления является простейшей, так как в ней используются только две цифры: 0 и 1, а восьмеричная система счисления удобна тем, что основание ее, а именно числовое значение 8, является степенью основания двоичной системы счисления 2. Например, десятичное число 65 в десятичной системе счисления можно представить как

$$6 \times 10^1 + 5 \times 10^0 = 65,$$

в восьмеричной системе как

$$1 \times 8^2 + 0 \times 8^1 + 1 \times 8^0 = 101$$

и в двоичной системе счисления в виде

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + \\ + 0 \times 2^1 + 1 \times 2^0 = 100\,001.$$

Каждое действие, которое должно быть выполнено ЭВМ, на машинном языке задается в виде команды. Команда — это информация, представленная в форме, позволяющей ввести ее в машину, и определяющая действия ЭВМ в течение некоторого отрезка времени. Таким образом, каждая команда определяет, вообще говоря, некоторую элементарную часть процесса обработки информации, называемую машинной операцией. Исходная информация для обработки поставляется, как правило, набором конкретных значений, называемых обычно данными. Исходные данные для выполнения любого действия, в том числе и машинной операции, будем называть *операндами*.

В команде в общем случае должны быть указаны вид действия, местонахождение в машине (адрес) исходной информации, над которой производится машинная операция, адрес результата, а также следующая команда, которая должна быть выполнена после данной. Для арифметических действий (или операций) исходная информация задается, как правило, в виде двух чисел, следовательно, в команде для нее должны быть указаны два адреса. Таким образом, команда должна содержать код операции, задающий вид выполняемой машинной операции, и четыре адреса: два адреса операндов, адрес результата и адрес следующей команды. Как правило, необходимое число адресов в команде меньше четырех.

Широко распространены ЭВМ с трехадресными командами или трехадресные машины. В них не указывается адрес следующей команды, а автоматически выполняется команда из следующей ячейки памяти (с номером, на единицу большим, который и является адресом следующей команды). Например, если принять для операции сложения код 01, то для сложения двух чисел из ячеек с номерами 2051 и 2052 с результатом, помещаемым в ячейку с номером 2345, в трехадресной машине команда будет выглядеть так:

01 2051 2052 2345

В двухадресной машине обычно отсутствует адрес результата, так что результат помещается либо в ячейку одного из операндов, либо на так называемый регистр результата (сумматор). В одноадресных машинах результат всегда получается на сумматоре, так что для выполнения той же операции сложения двух операндов должны быть предусмотрены три команды: записи одного из операндов на сумматор, сложения содержимого сумматора с операндом, находящимся по определенному адресу, и пересылки результата по заданному адресу, чтобы освободить сумматор для следующей операции.

Составление программы на машинном языке носит характер решения сложной комбинаторной задачи, так как одновременно с составлением команд программисту необходимо также распределять память. Распределение памяти машины, т. е. размещение в запоминающих устройствах машины всей информации, относящейся к решению задачи (команд и вспомогательных кодов, исходного числового материала и промежуточных данных, окончательных результатов), неотделимо от составления команд. Команды можно составлять лишь после того, как известны номера ячеек, где будет храниться вся необходимая для этих команд информация. С другой стороны, трудно произвести размещение информации в памяти машины, если неизвестно количество команд программы и количество промежуточных результатов, которые должны одновременно находиться в памяти.

Первым усовершенствованием процесса программирования явилось введение символических адресов, позволившее составление команд и распределение памяти выполнять раздельно. Сущность этого приема заключается в разбиении оперативной памяти машины на массивы, число ячеек в которых заранее не известно, а номера ячеек массива задаются буквенно-числовыми обозначениями типа  $a_i + 1$ ,  $a_i + 2$ , ..., называемыми символическими адресами. Распределение памяти осуществляется приписыванием всем  $a_i$  числовых значений уже после составления программы. Последний процесс является чисто механическим и может быть автоматизирован, т. е. присвоение истинных адресов может быть поручено самой вычислительной машине.

Такое усовершенствование процесса программирования вскоре привело к созданию языков символического программирования, или *автокодов*. Такой язык отличается от машинного языка лишь тем, что вместо числовых значений, выражающих код операции команды и ее адреса, используются символические (буквенные) обозначения. Поэтому в первых автокодах существовало взаимно однозначное соответствие между операциями, записанными на языке символического программирования (или кодирования), и командами в машинном языке, на что указывал символ 1:1, который записывался после

наименования языка. Напрямер, автокод 1:1 — АВТОматическое КОДярование один к одному.

Дальнейшее совершенствование автокодов выражалось в появлении дополнительных средств, устанавливающих по обычным правилам порядок действий в арифметических формулах или обеспечивающих в необходимых условиях разветвление вычислительного процесса, циклическое повторение участков программы и другие операции, вытекающие из условия задачи. Так постепенно автокоды утратили приставку 1:1, а их входные языки стали не чисто машинными, а *машинно-ориентированными*. Машинная ориентированность означает, что в основе этих языков продолжала лежать система команд какой-либо конкретной вычислительной машины.

Первые машинно-ориентированные языки в целом были несовершенны. У одних языков описание последовательности вычислений было оторвано от самих формул, другие имели сложную символику, мало наглядную или слишком специализированную, третьи были приспособлены лишь для решения ограниченного круга задач. Основной же недостаток заключался в привязанности языка к данной машине.

С появлением машин второго поколения возникла потребность создания языков, целиком ориентированных на особенности задач и не зависящих от конкретной машины. Это требование усугублялось еще и тем, что ЭВМ разных марок быстро сменяли одна другую или использовались совместно. Символом второго поколения ЭВМ стали *проблемно-ориентированные* языки программирования. Их развитие все в большей степени определялось спецификой задач, а не особенностями машин. На первый план выступило то общее, что было в различных задачах, а это сближало разные языки, созданные в эпоху господства вычислительных задач. Эти языки принято называть формальными алгоритмическими языками или просто *алгоритмическими языками*.

От формального алгоритмического языка требуется многое. Во-первых, он должен быть наглядным, что может быть достигнуто использованием существующей математической символики и других легко понимаемых изобразительных средств. Во-вторых, — гибким, чтобы любой алгоритм мог быть описан без излишнего усложнения, связанного с недостаточностью изобразительных средств. В-третьих, от языка требуется однозначность — запись любого алгоритма, выполненная с соблюдением всех правил языка, должна не допускать различных толкований, и многоступенчатость — сложный алгоритм может быть описан в виде сочетания более простых алгоритмов. И, наконец, язык должен быть единым — с одной стороны, число изобразительных средств не должно быть слишком большим, и с другой стороны, чтобы один и те же средства можно было применять для выражения одних и тех же или родственных понятий в разных (по их назначению) частях алгоритма.

Такой язык служит: средством мышления — логическое несовершенство предполагаемого метода решения задачи часто выявляется в процессе записи этого метода средствами алгоритмического языка; средством общения между людьми — описание процесса, выполненное одним человеком, должно быть доступно другим; посредником между человеком и машиной — при этом перевод с алгоритмического языка на язык машины выполняется самой машиной с помощью программирующей программы, или *транслятора*.

Одним из первых и наиболее удачных языков такого рода стал фортран, разработанный фирмой IBM. В 1954 г. группа американских специалистов в области программирования во главе с проф. Дж. В. Бэкусом опубликовала первое сообщение о языке. Название языка происходит от словосочетания FORMulae TRANslation — преобразование формул. Язык фортран не только просуществовал до наших дней, но и уверенно удерживает первое место в мире по распространенности. Среди причин такого долголетия можно отметить простую структуру как самого фортрана, так и предназначенных для него трансляторов. Программа на фортране записывается в виде последовательности предложений, или *операторов* (под оператором понимается описание некоторого преобразования информации), и оформляется по определенным правилам. Эти правила накладывают ограничения, в частности, на форму записи и расположения частей оператора в строке бланка для записи операторов. Программа, записанная на фортране, представляет собой один или несколько сегментов (подпрограмм) из операторов. Сегмент, управляющий работой всей программы в целом, называется основной программой.

Фортран был задуман для использования в сфере научных и инженерно-технических вычислений. Однако на этом языке легко описываются задачи с разветвленной логикой (моделирование производственных процессов, решение игровых ситуаций и т. д.), некоторые экономические задачи и особенно задачи редактирования (составления таблиц, сводок, ведомостей и т. д.).

Модификация языка фортран, появившаяся в 1958 г., получила название фортран II и содержала понятия подпрограммы и общих переменных для обеспечения связи между сегментами.

К 1962 г. относится появление языка, известного под названием фортран IV и ставшего наиболее употребительным в настоящее время. К этому же времени относится и начало деятельности комиссии при Американской Ассоциации Стандартов (ASA), которая выработала (к 1966 г.) два стандарта — языки фортран и базисный (основной) фортран (Basic FORTRAN). Эти языки приблизительно соответствуют модификациям IV и II, однако базисный фортран является подмножеством фортрана, в то время как фортран II таковым для фортрана IV не является.

Язык фортран до сих пор продолжает развиваться и совершенствоваться, оказывая влияние на создание и развитие других языков. Например, фортран заложен в основу диалогового языка BASIC (Beginner's Allpurpose Symbolic Instruction Code — многоцелевой язык символических команд для начинающих, Вычислительный центр Дартмутского колледжа, 1966 г.) и его расширения BASIC-PLUS (Бейсик-плюс, фирма Digital Equipment Corporation, 1975 г.), широко распространенных языков во всех системах с режимом разделения времени, превосходных языков для обучения навыкам использования алгоритмических языков в практике программирования. Эти языки реализованы на ряде отечественных машин, в частности на малых ЭВМ СМ-4 [20]. В настоящее время создан новый стандарт — фортран 77, описание которого содержится в [16] (см. также § 9 гл. V).

Вскоре после создания фортрана (1957 г.) появился язык алгол (ALGOrithmic Language — алгоритмический язык), созданный на основе широкого международного сотрудничества. В 1960 г. под редакцией проф. П. Наура в

журнале «Communications of the Association for Computing Machinery» было опубликовано «Сообщение об алгоритмическом языке АЛГОЛ-60», которое легло в основу языка, существующего до сих пор. Число 60 означает год утверждения языка.

Алгол-60 создавался после разработки и практического применения фортрана, поэтому характеризуется как введенным новых конструкций, так и обобщенным понятий, имеющихся в фортране. Например, если в фортране операторы с функциональной точки зрения подразделяются на исполняемые и неисполняемые, то в алголе такого деления нет, а роль неисполняемых операторов фортрана выполняют конструкции, называемые описаниями.

Имеются и другие отличия. Однако общим для фортрана и алгола является то обстоятельство, что в основе обоих языков лежит понятие *выражения*, практически совпадающее с понятием математического выражения, использующего лишь алгебраические операции и элементарные функции. Простейшие объекты, из которых составляются выражения, — это целые и приближенные вещественные числа и логические значения.

Алгол повсеместно признан как весьма удобное средство для публикации алгоритмов и для обучения основам программирования. Уточнения и поправки к языку алгол-60 содержатся в [9] и отражены в данной книге.

И фортран, и алгол-60 до недавнего времени по праву заслуживали название *универсальных* языков, так как обеспечивали программирование основной массы научно-технических задач (преимущественно вычислительных). Но ни один из этих языков, конечно, не позволял описать все без исключения возникавшие задачи. Поэтому примерно в то же время появились алгоритмические языки с другой ориентацией, отвечающие нуждам тех новых направлений науки и техники, которые стали интенсивно развиваться в последующие годы.

Примером могут служить экономические задачи — задачи учета материальных ценностей, выпущенной продукции, личного состава, финансов и т. д. предприятия или отрасли. Для таких задач основными действиями являются операции ввода и вывода при относительно небольшом количестве несложных вычислений, а также последовательная обработка массивов данных. Описание действий такого рода может быть осуществлено на языке кобол (COBOL Business Oriented Language), предложенным фирмой ИВМ в 1959 г. Этот язык имел и сохраняет до сих пор заслуженный успех у пользователей.

Задач обработки символьной информации возникают преимущественно в области научных исследований. Это, например, преобразование формул, решение уравнений (не численное, а в аналитическом виде); анализ и синтез текстов на искусственном или естественном языке (в частности, автоматическое программирование и машинный перевод) и т. п.

Из языков для обработки символьной информации очень популярным, главным образом среди представителей физико-математических наук, является язык лисп, созданный группой исследователей под руководством проф. Дж. Маккарти в 1960 г. в Массачусетском технологическом институте. В этом языке вся находящаяся в обработке информация, в том числе и сама программа, организуется в так называемые списки — последовательности элементов. Элемент может быть первичным (буквенным обозначением или числом)

или в свою очередь списком. Так могут возникать сколь угодно сложные структуры.

Другой язык — снобол — применяется в основном для машинного анализа текстов, написанных на естественных языках. В нем основным понятием является строка — произвольная последовательность букв, цифр и других знаков. Главная операция — это поиск в строке части строки, построенной по заданному образцу, и замена этой части другой строкой. Как образ, так и заменяющие его строки составляются из отдельных элементов простого вида. Исход поиска определяет последовательность дальнейших действий. Язык снобол очень прост для изучения.

Основное достоинство проблемно-ориентированных, машинно-независимых алгоритмических языков в том, что они были построены с максимальным учетом представлений человека если не о существе, то о форме решаемой задачи, с максимальным приближением к той форме, в которой человек привык описывать эти задачи, и с учетом тех логических связей, которые он научился выделять в исследуемых явлениях.

Для алгола, например, характерно приближение к привычной математической символике и лучшее, чем у его предшественников, отражение структуры вычислительной задачи. Фортран же, в отличие от алгола, ближе к машинному языку, чем к языку человека. Для лиспа характерно использование аппарата так называемых рекурсивных описаний, широко применяемого в математической логике, в исследованиях по основаниям математики и т. п.

Обилие алгоритмических языков, появившихся в период второго поколения ЭВМ, с одной стороны, во многом объясняется модой, с другой — невозможностью ни одним из предложенных языков удобно описывать все возникавшие задачи. Третье поколение ЭВМ поставило на повестку дня выработку нового подхода к созданию действительно универсального алгоритмического языка.

Одной из попыток такого рода является создание фирмой IBM алгоритмического языка ПЛ/1 (Programming Language/1 — язык программирования один). Он основан на языках фортран и кобол, ряд изобразительных средств и понятий был почерпнут из алгола и других языков, в частности языков для обработки символьной информации. В первоначальной версии 1964 г. он назывался New Programming Language или NPL — новый язык программирования. Затем последовательно было опубликовано несколько версий языка, которые сильно отличались друг от друга, но постепенно язык стабилизировался, и теперь новые публикации отличаются от предыдущих лишь редакционными поправками, устранением неточностей или усовершенствованием отдельных элементов.

Основными элементами программы, написанной на языке ПЛ/1, являются операторы, с помощью которых описываются как данные, так и операции их обработки. По аналогии с фортраном исходная программа представляет собой совокупность основной программы и подпрограмм, имеющих форму блока. Понятие блока в ПЛ/1 базируется на концепциях блока в языке алгол-60. Таким образом, этот язык построен в целом на базе понятий существующих алгоритмических языков и в их традициях.

Другая попытка связана с дальнейшим развитием алгола. Группа математиков, членов IFIP (Международной федерации по обработке информации) пошла по линии обобщения и углубления основных понятий в области обработки информации и в 1968 г. опубликовала документ с изложением основ нового универсального алгоритмического языка, получившего название алгол-68. В этом языке число основных понятий сведено к разумному минимуму с целью добиться высокой изобразительной силы языка, обеспечив свободу сочетания и взаимодействия этих понятий между собой.

Однако и алгол-68 традиционен, поскольку проявляется стремление обеспечить всех пользователей готовыми средствами для описания их алгоритмов. До сих пор этот подход не мог предотвратить появления все новых специализированных языков. Поэтому в последнее время проявляется тенденция к созданию так называемых *расширяемых* универсальных языков. Основная идея такого направления — не избегать специализированных языков-диалектов, а создать общую основу «программистских диалектов».

Расширяемый язык должен располагать средствами для грамматического разбора текстов любого из расширений, чтобы выяснить, какие тексты являются грамматически правильными для данного диалекта и какой структурой они обладают. Такой язык должен дать будущим его потребителям целый ряд важных возможностей: например, строить семантические модели (т. е. вводить новые термины и описывать выражаемые ими понятия, их взаимосвязь с некоторыми основными, исходными понятиями и с понятиями, введенными ранее) и описывать реализацию расширений — способы их наиболее целесообразного представления с помощью средств, которыми располагают современные ЭВМ. Можно отметить и другие характерные черты расширяемых языков.

Работы по созданию и совершенствованию языков ведутся во многих странах, большое количество исследований выполняется в рамках IFIP.

## § 5. Трансляция и входные языки

Современный алгоритмический язык существенно отличается от языка конкретной машины, поэтому алгоритм, записанный как программа на алгоритмическом языке, не может непосредственно быть воспринятым ЭВМ. Для получения программы, понятной ЭВМ, необходимо перевести запись алгоритма с алгоритмического языка на язык машины. Этот перевод осуществляется самой машиной при помощи специальной программы, написанной в кодах данной машины (или на автокоде для этой же машины) и называемой транслятором.

Алгоритм для решения любой задачи, записанный на алгоритмическом языке (программа), представляет собой некоторый текст на листе бумаги в виде отдельных строк. Как правило, для каждого языка готовятся специальные бланки, на которых и записывается на данном языке исходная программа.

Затем вся информация с бланков набивается на перфокарты или на перфоленгу с помощью, например, устройства подготовки перфокарт (УПП). Существуют различные типы УПП, однако принцип кодировки исходного

текста у всех устройств один и тот же — каждому символу текста соответствует вполне определенный код в восьмеричных цифрах, набиваемый на перфокарты комбинацией отверстий. Как правило, каждой строке текста программы соответствует одна перфокарта.

При помощи устройства ввода перфокарты вводятся в ЭВМ. После ввода исходной программы осуществляется ее трансляция, в результате которой создается программа на машинном языке, т. е. программа, состоящая из команд той машины, на которой будет решаться задача. Эта программа называется *рабочей программой*.

Процесс перевода алгоритма (трансляция) и процесс его выполнения машиной (выполнение рабочей программы) могут сочетаться двумя способами. Первый способ, называемый компиляцией, заключается в том, что процесс выполнения алгоритма машиной осуществляется после того, как процесс перевода полностью завершен. Для компиляции характерно, что осуществляющая ее программа-транслятор во время выполнения рабочей программы уже не нужна и потому не находится в оперативной памяти ЭВМ. Тем самым достигается более экономное использование ячеек оперативной памяти.

Второй способ сочетания процесса перевода и процесса выполнения алгоритма — интерпретация — предполагает, что отдельные операторы (или другие части исходной программы) сразу после трансляции выполняются, после чего та же процедура совершается над другими операторами и т. д. При интерпретации во время выполнения рабочей программы программа-транслятор находится в оперативной памяти ЭВМ, т. е. требуется дополнительная память для нее. Кроме того, процесс решения задачи замедляется, так как между отдельными этапами выполнения рабочей программы управление передается транслятору. В то же время при интерпретации упрощается задача распределения памяти.

Оба эти способа имеют свои достоинства и недостатки. Очевидно, возможно сочетание компиляции с интерпретацией, заключающееся в том, что компилируется программа, в которую включены специальные коды, называемые макрокомандами. Макрокоманда объединяет в рабочей программе определенную группу одиночных машинных команд и интерпретируется при выполнении программы. Другой вариант возможен, если основным является процесс интерпретации, однако получаемые при ней участки программ (модули) сохраняются (если позволяет объем оперативной памяти) и включаются в компилируемую программу вместо соответствующих этим участкам групп кодов, получаемых обычно при повторной интерпретации.

Для отечественных универсальных ЭВМ создано много трансляторов с различных алгоритмических языков. Как правило, каждая марка ЭВМ имеет хотя бы один транслятор. В то же время для машин одного и того же класса создано несколько типов трансляторов с одного и того же языка. Например, существует несколько вариантов трансляторов для ЭВМ БЭСМ-6 с алгола и фортраиа и других языков. Обилие трансляторов объясняется тем, что практически не существует ни одного транслятора, переводящего точно и полностью все элементы формального алгоритмического языка. Такой транслятор с полного языка был бы малоэффективен из-за необходимости учитывать все свойства языка, а также маловероятные ситуации. При

переводе он был бы очень громоздким и неоправданно сложным. Поэтому во многих трансляторах предусматривается, как правило, перевод только с неполного языка.

Различие трансляторов может быть вызвано еще и тем, что не на всех вычислительных машинах могут быть в точности реализованы все символы и обозначения, составляющие алфавит *эталонного* языка — общепринятой стандартной версии языка. Все языки, которые получаются после того, как в эталонном языке произведены некоторые замены, упрощения или, наоборот, расширения, называются конкретными представлениями языка (или *входными* языками). Разумеется, каждый входной язык (и каждая версия языка тоже) имеет свои особенности, которые отражаются на соответствующем трансляторе, отличая его от других трансляторов с этого же алгоритмического языка для данной вычислительной машины. Например, в некоторых трансляторах с алгола разрешается использовать буквы греческого и русского алфавитов, хотя эталонный язык допускает только латинские буквы.

В связи с этим уместно отметить, что перед тем, как использовать данную машину и данный транслятор при решении какой-либо задачи, необходимо ознакомиться с описанием транслятора, его характеристиками и входным языком.

### § 1. Элементарные конструкции языка

Алгольная программа представляет собой последовательность основных символов, которые составляются из элементов алфавита — цифр, букв и специальных знаков.

Цифры — это десять арабских цифр:

1 2 3 4 5 6 7 8 9 0

Буквами в эталонном языке являются 26 строчных и столько же прописных букв латинского алфавита:

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z

В конкретных представлениях языка могут быть ограничения или расширения алфавита. Так, в большинстве отечественных трансляторов в соответствующих входных языках разрешается использование строчных и прописных букв русского алфавита, а в некоторых — и греческого.

Специальными знаками являются следующие символы:

+	-	×	/	↑	÷
<	≤	=	≥	>	≠
∧	∨	∟	≡	⊃	
(	)			'	,
.	10	:	;	,	(пусто)

Последний символ никак не обозначается (пусто), в дальнейшем для большей наглядности иногда вместо пробела будем пользоваться символом —.

**1. Основные символы.** Основные символы языка алгол делятся по своему назначению на четыре группы: цифры, буквы, логические константы и ограничители. Назначение каждого основного символа языка и его использование для записи алгоритмов определяются правилами составления синтаксических конструкций (предложений) языка и будут рассмотрены в дальнейшем. Непосредственно из основных символов образуются элементарные конструкции языка: числа, идентификаторы, переменные, указатели функций и строки, а также выражения, описания и операторы.

*Цифры* служат для образования чисел, идентификаторов и строк.

*Буквы* используются для записи идентификаторов и строк.

*Логические константы* — это два символа в виде слов: **true** (истина) и **false** (ложь), которые используются при построении логических выражений. Любая логическая переменная, или логическое выражение, может принимать только одно из этих двух значений.

Кроме символов в виде слов, в языке используются конструкции, также состоящие из последовательностей букв. В отличие от последних, основные символы-слова (иначе, служебные, или ключевые, слова) в печатных изданиях выделяются полужирным (или жирным) шрифтом, а в рукописном тексте подчеркиваются.

*Ограничители* делятся на три группы: знаки операций, разделители и описатели.

*Знаки операций* — это:

— знаки арифметических операций:  $+$   $-$   $\times$   $/$   $\uparrow$   $\div$ , служащие для построения арифметических выражений,

— знаки отношений:  $<$   $\leq$   $=$   $\geq$   $>$   $\neq$ ,

— знаки логических операций:  $\wedge$   $\vee$   $\uparrow$   $\equiv$   $\supset$ , которые используются для записи логических выражений.

К числу *разделителей* относятся:

— указатели следования, управляющие порядком действия в программе:

**go to** (перейти к) — для записи операторов перехода,

**if** (если),

**then** (то),

**else** (иначе) — для построения выражений, содержащих условие, и условных операторов,

**for** (для),

**step** (шаг),

**until** (до),

**while** (пока),

**do** (выполнить) — для записи операторов цикла;

— конструктивные разделители, расчленяющие запись программы на отдельные элементы (выражения, операторы и др.):

$\cdot$   $_{10}$  — в записи чисел,

**:** — для отделения метки от оператора и в описании массивов,

**;** — для разделения операторов,

**:=** — в операторе присваивания,

**,** — для разделения элементов в списках переменных, параметров и в списках граничных пар,

**—** — не имеет смыслового значения, используется при записи конструкций языка;

— скобки, служащие для выделения отдельных частей программы:

**( )** — в записи выражений и в операторах, в описании процедур,

**[ ]** — в описании массивов, в записи переменных с индексами и указателей переключателя,

**'** — при записи строк,

**begin** (начало), **end** (конец)— в записи частей программы, являющихся составным оператором или блоком.

*Описатели* указывают на роль и свойства той информации, которой предшествуют в программе. Описатели делятся на:

— собственно описатели, используемые для описания соответствующих объектов программы и для спецификации формальных параметров процедур:

**array** (массив),

**Boolean** (логический),

**integer** (целый),

**real** (действительный),

**procedure** (процедура),

**switch** (переключатель);

— описатель собственных величин:

**own** (собственный);

— спецификаторы, служащие для спецификации формальных параметров процедур:

**label** (метка),

**string** (строка),

**value** (значение);

— указатель комментария, используемый для записи примечания в тексте программы:

**comment** (примечание).

Основные символы записываются в строки. Подчеркнутое служебное слово в рукописном тексте рассматривается как один основной символ. У нижнего основания строки пишутся символы , (запятая), . (точка) и  $10$  (основание порядка). Положение у верхнего края строки занимают ' и ' (строчные кавычки). Все остальные символы располагаются посередине строки. Например:

$a + b$ ,  $5.31_{10} - 03$ , 'Algol'

**2. Числа.** Для записи чисел используются цифры, знаки + или —, а также точка и символ  $10$ .

Под *числом* (или числовой константой) понимается запись числа, а не соответствующее ей числовое значение, поэтому одному и тому же числовому значению могут соответствовать разные числа.

Все числа подразделяются на два типа — целые и действительные. Целые числа представляются в машине точно и все операции над ними выполняются точно, в то время как действительные числа в общем случае представляются лишь приближенно и в результате выполнения арифметических действий над ними получаются приближенные значения.

Таким образом, понятие «действительное число» в алгоритмическом языке отлично от этого же понятия в математике, где оно является общим названием для положительных, отрицательных чисел и нуля и противопоставляется понятию «мнимое число». Это отличие выражается как в существе понятия числа, так и в форме его записи. В дальнейшем будет использован термин

«действительное число» всюду, где речь идет о типе числа, и термин «вещественное число», если число понимается в математическом смысле.

*Целое число* записывается в виде последовательности одной или нескольких цифр, перед которой могут стоять символы  $+$  или  $-$ . В первом случае целое число будет положительным и знак  $+$  может быть опущен, во втором — отрицательным. Максимальная длина такой последовательности и соответствующее такой записи максимальное числовое значение зависят от конкретного представления и типа ЭВМ. Так, в записи целого числа на БЭСМ-6 допускается до десяти цифр. Примеры целых чисел:

0 +987 01 -4208

*Правильной дробью* называется последовательность цифр, перед которой стоит точка. Например,

.367 .1 .005 .300

Отличие от общепринятой математической записи здесь в том, что вместо запятой ставится точка и нуль в разряде единиц перед точкой не пишется. Так, правильная дробь .367 в обычной математической записи выглядит как 0,367.

Последовательность из целого числа без знака и правильной дроби называют дробью. Например,

1.314 25.47 0.38 5.005

Целое число без знака, правильная дробь и дробь имеют общее название — десятичное число.

Десятичный порядок записывается в виде разделителя  $_{10}$ , за которым следует целое число, содержащее не более двух цифр со знаком или без него. Например:

$_{10}3$   $_{10}+3$   $_{10}03$   $_{10}-11$

Десятичный порядок представляет собой запись степеней числа  $10$ , т. е. трем первым приведенным числам соответствует обычная запись  $10^3$ , а последняя есть  $10^{-11}$ .

*Действительное число* представляет собой либо десятичное число, либо десятичный порядок, либо строку из десятичного числа и следующего за ним десятичного порядка, или любую из названных конструкций, перед которой стоит знак  $+$  или  $-$ . Примеры записи действительных чисел:

-25  $_{10}-05$   $3.14_{10}+2$   
 $.037$   $-25.037$   $.037_{10}01$

Как следует из вышесказанного, число не может заканчиваться точкой (запись 8. неверна, 8.0 правильная).

Допустимые максимальное и минимальное числовые значения действительного числа зависят от конкретного представления и должны лежать в заданных для конкретной ЭВМ пределах. Для большинства старых отечественных машин эти пределы будут  $10^{-19}$  и  $10^{+19}$ , для новых (в частности, ЕС ЭВМ) —  $10^{-78}$  и  $10^{+75}$ . Однако точность представления слишком

малых и излишком больших значений невелика, так как в мантиссе сохраняется ограниченное количество старших разрядов.

**3. Идентификаторы.** Наряду с основными символами и числовыми константами вводятся понятия простых переменных, массивов, функций, меток, переключателей и процедур. Простые переменные, массивы и функции составляют величины. При написании программы каждый из указанных объектов определенным образом должен быть обозначен. Роль таких обозначений выполняют *идентификаторы*. Идентификатор — это наименование, присвоенное объекту автором программы, представляющее собой произвольной длины последовательность цифр и букв, которая начинается с буквы. Примеры различных идентификаторов:

<i>x</i>	<i>AB</i>	<i>cfirst</i>	<i>Alfa</i>	<i>Det and Sol</i>
<i>xk</i>	<i>BA</i>	<i>x1y2x4</i>	<i>alpha</i>	<i>program</i>

Поскольку пробелы не имеют смыслового значения, а используются лишь для наглядности, то в любых конструкциях языка, исключая строки, пробел между символами не принимается во внимание. Следовательно, в идентификаторе могут содержаться пробелы. Отметим, что, по существу, идентификатор задает адрес объекта в оперативной памяти ЭВМ, поскольку каждому идентификатору отводится вполне определенный участок (ячейка) памяти.

За некоторыми часто встречающимися математическими процедурами в языке закрепляются следующие стандартные идентификаторы:

*abs, iabs, sign, entier, sqrt, sin, cos, arctan, ln, exp, inchar, outchar, length, outstring, outterminator, stop, fault, ininteger, outinteger, inreal, outreal, maxreal, minreal, maxint, epsilon.*

Функции и процедуры, обозначаемые с помощью этих идентификаторов, называются стандартными. Использование стандартных функций и процедур описано в [9] и в приложении I. В конкретных представлениях список стандартных идентификаторов может быть расширен.

Соответствие идентификаторов тем объектам, которые ими обозначаются, устанавливается в каждой конкретной программе по содержащимся в ней описаниям.

**4. Переменные.** *Переменная* — это величина, которой дано наименование и которая на данном этапе вычислений принимает определенное значение. Значением может быть либо число или логическая константа, либо некоторое упорядоченное множество чисел или логических констант.

Переменная, обозначаемая только идентификатором, называется *простой переменной*. Значением ее может быть либо число (целого или действительного типа), либо логическое значение. В соответствии с принимаемыми переменной значениями различают типы переменных. Переменная целого типа, или целая переменная, принимает значения, задаваемые числами целого типа. Действительная переменная принимает значения, определяемые числами действительного типа. Логическая (или булева) переменная принимает только два значения: значения логических констант *true* и *false* (логические единица и нуль). Для указания типа переменной служат описания типа (§ 2).

*Переменная с индексами* записывается как конструкция, состоящая из идентификатора, за которым следует заключенный в квадратные скобки спи-

сок индексов. Список индексов состоит из одного или нескольких арифметических выражений (в частном случае из целых чисел, см. § 9), разделенных запятыми. Эти выражения в данном случае называются индексными выражениями.

Переменная с индексами является аналогом однородного объекта в математике, который обозначается идентификатором с последующими индексами в квадратных скобках, выделяющими данный конкретный объект среди других. Например, компоненты вектора  $x$  могут быть обозначены  $x[1]$ ,  $x[2]$ ,  $x[3]$ , а элемент матрицы  $a$  в строке  $i$  и столбце  $j$  записывается как  $a[i, j]$ .

Индексы принимают только целые значения (положительные, отрицательные и нулевые). Если же значением индексного выражения оказывается число действительного типа, то оно алгебраически округляется до ближайшего целого до того, как будет использоваться как индекс переменной. Например, записи  $A[2.4, 5.8, 6.5]$  и  $A[1.9, 6.4, 7.3]$  задают одну и ту же переменную  $A[2, 6, 7]$ .

Переменные с индексами являются элементами массивов — упорядоченных последовательностей значений одного типа. Вообще, массив — это конечная совокупность величин, обозначенных одним и тем же идентификатором с различными целочисленными индексами. Примером может служить массив из трех элементов  $x[1]$ ,  $x[2]$ ,  $x[3]$ .

Массив, содержащий элементы с одним индексом, называется одномерным, с двумя — двумерным и т. д. Следовательно, массив  $x[1]$ ,  $x[2]$ ,  $x[3]$  одномерный, и, вообще, вектор  $n$ -мерного пространства является одномерным массивом, содержащим  $n$  элементов. Матрица может служить примером двумерного массива.

Информация о том, какие идентификаторы в программе являются идентификаторами массивов, какова размерность каждого массива, сколько элементов в определенном массиве и как изменяются индексы элементов, а также о том, какого типа значения принимают переменные с индексами, приводится в описаниях массивов (§ 2).

**5. Указатели функций.** Под *функцией* понимается алгоритм для вычисления некоторого значения в зависимости от значений заданных выражений и значений идентификаторов других заданных величин. Эти величины называются фактическими параметрами. Алгоритм вычислений для функции задается описанием функции, содержащим набор формальных параметров, которые при обращении к функции должны быть заменены фактическими параметрами. Функция обозначается идентификатором, который в этом случае называется идентификатором процедуры, так как функция является частным случаем процедуры (§ 17).

Указатель функции представляет собой обращение к процедуре, задающей функцию, для вычисления одного значения этой функции. Указатель функции состоит из идентификатора, вслед за которым в круглых скобках выписываются фактические параметры, отделяемые друг от друга запятыми. Например:

$$f(x) \quad F(G(H)) \quad y(k, l) \quad z(2.5)$$

Так, если в программе с помощью описаний задана функция  $f(x)$ , в обычной математической записи представленная выражением  $8x^2 + 5x + 3$ , то указатель этой функции в случае вычисления значения при  $x = 2$  имеет вид  $f(2)$ .

Фактический параметр может быть любым арифметическим выражением (см. § 3). Значение фактического параметра вычисляется непосредственно перед обращением к вычислению значения функции.

Процедура, по которой вычисляется значение функции, может и не содержать параметров. Указатель функции в таком случае состоит лишь из идентификатора функции. Например А.

Использование в программе стандартных функций предусмотрено без предварительного описания. По указателю стандартной функции транслятор автоматически включит в программу вычисляющие значения этой функции команды.

**6. Строки.** Строка является либо последовательностью любых литер, не содержащих 'или', заключенная в строчные кавычки, либо последовательностью строк и литер. В последнем случае вся последовательность также содержится между открывающей и закрывающей кавычками. Под литерами понимаются не только основные символы языка алгол-60. Например, строками будут последовательности:

'abcd'                    '3x, — : = k ≤'  
'determination of the symbol 'string''

Строки используются в качестве фактических параметров в процедурах. Например, строка может содержаться в операторе вывода как фактический параметр и, следовательно, быть напечатана на устройстве вывода.

#### Упражнения

1. Среди приведенных ниже записей чисел определить те, которые являются: а) целыми числами; б) действительными числами; в) равными по величине; г) записями, не допускаемыми правилами алгола:

378	50000	0.100 <sub>10</sub> 03	+500.0 <sub>10</sub> +2.0
100	—. <sub>10</sub> —6	.5 <sub>10</sub> —06	0.03870 <sub>10</sub> +04
805	387.0	+0.1 <sub>2</sub> 10	0.0000805
+2	0.387	.05 <sub>10</sub> 05	.0001 <sub>10</sub> +6

2. Среди приведенных ниже записей выбрать последовательности символов, являющиеся идентификаторами:

massiv	COЮЗ-35	α + β
integer	determination	fortran
exp	NATALI	RKZH
2a	'algol'	for(tran)

3. Указать, какие из приведенных ниже указателей функций не содержат ошибок:

$f(x, y)$	$z(z(s))$	$real(integer)$
$\sin(2, x)$	$f(xy)$	$entier(-15.4)$
$\sin(2x)$	$\cos[x]$	$\log(false)$

4. Указать, какие из приведенных записей можно рассматривать как обозначения переменных:

A3	A[3]	LENA	epsilon
A(3)	L[true]	pi3.14	q[-4.1]

5. Указать, какие из приведенных ниже записей можно рассматривать как строки:

string	'a∨b'	'NAME: ALLA'
string	""a'b'c"	'A-----':
'string'	'L[false]'	longitude

6. Определить, допускаются ли в алголе следующие конструкции. В случае положительного ответа указать класс соответствующих объектов:

array	'a + b'	a[3.6]
.076	algol	a[3,6]
'x := 25'	arctgx	3E + 05
256.	НАТАЛИЯ	'integer'
FORTRAN	language	REAL

## § 2. Описание типа

Информация о свойствах используемых в программе величин задается в *описаниях*, которые указывают, какими идентификаторами обозначаются величины, какого они типа и какова их область действия.

1. **Описание простых переменных.** Описание типа простых переменных состоит из основного символа **integer**, **real** или **Boolean**, после которого следует список идентификаторов, отделенных друг от друга запятыми. Список идентификаторов заканчивается символом ; (точка с запятой). Например:

**integer** i, j, k, N; **real** a, x, y, c7; **Boolean** l;

Описание типа означает, что в его области действия (в частном случае во всей программе) перечисленные в нем идентификаторы обозначают именно простые переменные, принимающие значения только того типа, который определен описанием (целочисленные — в случае описателя **integer**, действительные — в случае описателя **real** и логические — в случае **Boolean**). В программе могут быть использованы только те переменные, которые перечислены в списке при описании. Порядок следования идентификаторов в списке произвольный.

Само по себе описание типа не задает переменным, перечисленным в его списке, никаких значений.

2. **Описание массивов.** Для задания массива служит описание массивов, состоящее в простейшем случае из основного символа **array**, списка идентификаторов и заключенного в квадратные скобки списка граничных пар. Каждое описание должно отделяться от других элементов программы точкой с запятой.

Список идентификаторов указывает, что перечисленные в нем идентификаторы обозначают массивы и только массивы.

Список граничных пар состоит из одной или более граничных пар (разделенных запятыми, если их больше одной) и показывает, в каких пределах должно лежать численное значение каждого индекса. Граничная пара состоит из двух арифметических выражений, разделенных символом ; (двоеточие).

Выражение, стоящее слева от двоеточия, называется нижней границей, справа — верхней границей.

Значение нижней границы не должно превосходить значения верхней границы. Массив, у которого какая-либо верхняя граница меньше нижней, считается пустым. Индексы элементов массива принимают последовательные значения с шагом 1 от нижней границы до верхней. Каждый элемент получает только один набор значений индексов, которые могут быть отрицательными, положительными и нулевыми. Примеры описаний массивов:

```
array x[1 : 3]; array a[0 : 10, -1 : 16];
array A[i × (i - 1) : i × (i + 1)];
```

Список граничных пар является общим для всех идентификаторов в предшествующем ему списке. Таким образом, все идентификаторы списка будут обозначать массивы одинаковой структуры. Например, описание

```
array j, k, m, n[1 : 3, 1 : 3];
```

задает четыре двумерных массива со значениями 1, 2, 3 для каждого из индексов.

Одно описание может вводить в программу сразу несколько массивов. В этом случае список идентификаторов распадается на несколько частей, в конце каждой части ставится свой список граничных пар. Например:

```
array a, b, c[0 : 35], d[-3 : 14], p, q[3 : 13];
```

Элементы одного массива должны быть одного типа: целого, действительного или логического. Описание целого массива начинается символами **integer array**, действительного — символами **real array** или **array**, логического — символами **Boolean array**. Следовательно, во всех предыдущих примерах описывались массивы действительных значений. Описания массивов различных типов могут быть включены в программу в любом порядке. Каждое такое описание рассматривается как отдельное и должно заканчиваться символом ; (точка с запятой).

Пример допустимых описаний:

```
array A, B, C, D[1 : 5]; integer array a[1 : 5], b[0 : 7], c[-3 : 5];
real array E[-8 : 100]; Boolean array N[10 : 20, 10 : 20];
```

Очевидно, первое и третье описания в этом примере могут быть объединены в одно:

```
array A, B, C, D[1 : 5], E[-8 : 100];
```

При трансляции в соответствии с размерностью и размерами описываемого массива в памяти ЭВМ ему отводится определенная область. Элементы многомерных массивов располагаются в памяти последовательно, при этом первым меняется последний индекс (например, двумерный массив располагается в памяти по строкам).

**Упражнения**

1. Указать, какие из приведенных записей можно рассматривать как описания:

*integer one, first, x1, xf1, second;*  
*real a, b, -c, y[10];*  
**Boolean** *sum, sum1, summa, s[1 : 5];*  
*integer array M[1 : 10, 1 : 20, 1 : 10], N[0 : N];*  
*array mas[m : n], sam[-m : m], massa[n : m];*  
*array Boolean LOG[L < M : M];*

2. Описать входящие в некоторую программу идентификаторы: *A, B, C, D, F, K, x, y, z, l, m, k*, если известно, что *A, B, F, x, y* относятся к действительному типу, *K, z* — целому типу, *l, k* — логическому типу, *C, D* — векторы с компонентами *c<sub>1</sub>, c<sub>2</sub>, c<sub>3</sub>* и *d<sub>1</sub>, d<sub>2</sub>, d<sub>3</sub>* соответственно, действительного типа, *m* — массив из 100 элементов логических значений, первый из которых есть *m[-99]*.

3. Определить, допустимо ли в программе, содержащей описание **integer array** *m[0 : 5, -5 : 4]*; использование переменных с индексами: *m[sqrt(16), sin(k)]* и *m[4.8, 4.8]*.

**§ 3. Простые выражения**

Под *выражением* понимается последовательность символов, составленная по определенным правилам, таким, что после подстановки в эту последовательность значений всех идентификаторов и выполнения всех операций, символы которых содержатся в выражении, будет получено некоторое вполне определенное значение. Это значение есть результат выражения.

Выражения в алголе различают трёх видов: арифметические, логические и именующие. Классификация выражений связана с типом значения, вычисляемого с помощью данного выражения. Кроме того, различают выражения простые и выражения, содержащие условие.

Результат арифметического выражения есть числовое значение (действительное или целое), логического выражения — логическое значение (**true** или **false**), и, наконец, результатом именующего выражения является метка оператора.

Простое именующее выражение — это либо метка оператора, либо указатель переключателя. Определение метки оператора будет дано в § 7, а переключатель и указатель переключателя рассмотрены в § 12. В этом параграфе рассматриваются простое арифметическое и простое логическое выражения, которые для краткости именуются арифметическим и логическим выражениями соответственно.

**1. Арифметическое выражение.** Арифметические выражения строятся из чисел, переменных и указателей функции с помощью знаков арифметических операций и круглых скобок. Для обозначения операций употребляются следующие символы:

+	сложение,	/	деление,
-	вычитание,	÷	специальное деление,
×	умножение,	↑	возведение в степень.

Примеры выражений:

$$13.2 \quad r \uparrow 2 - q \times 3 \quad (x - y) \uparrow (3 \times 7 / (a + b))$$

$$al \quad a - b + c - d \quad k / x^2 - x \times 2 / x^2$$

Каждой переменной, которая встречается в выражении, должно быть присвоено значение до использования ее в выражении. Это может быть сделано в программе оператором присваивания (§ 6) или при вводе (§ 5).

Два знака арифметических операций нельзя писать рядом. Так, вместо  $a \times -b$  следует писать

$$a \times (-b) \text{ или } -b \times a.$$

Порядок выполнения арифметических операций в выражениях, не содержащих скобок, определяется правилом рангов: сначала выполняются все возведения в степень, затем все умножения и деления и в последнюю очередь — сложения и вычитания.

Очередность выполнения операций одинакового ранга определяется порядком следования знаков этих операций слева направо. Порядок выполнения арифметических операций может быть изменен с помощью круглых скобок. Значения указателей функций определяются перед выполнением с ними соответствующих операций. Запись  $x/y \times z$  означает  $(x/y) \times z$ , а не  $x/(y \times z)$ . Нельзя опускать знак умножения (т.е. необходимо писать  $x \times y$ , а не  $xy$ ). И, наконец, порядок одноранговых вычислений может оказать влияние на значение выражения. Например, в математике безразлично, в какой последовательности выполняются вычисления в выражении  $a - b + c$  — результат всегда единственный. В случае, когда  $a$  и  $b$  велики, а  $c$  мало, вычисления на ЭВМ в том порядке, как записано выше, дают результат более точный, чем, например, для математически эквивалентного выражения  $a + c - b$ , в котором при первом сложении младшие разряды (с помощью которых и записано число  $c$ ) будут потеряны.

Величины целого и действительного типов могут входить в одни и те же выражения. Операции, обозначаемые символами  $+$ ,  $-$ ,  $\times$ , дают значение выражения целого типа, если оба операнда выражения имеют целый тип, в противном случае тип значения будет действительный. Операция деления, обозначаемая символом  $/$ , всегда приводит к результату действительного типа.

Специальное деление, обозначаемое символом  $\div$ , определено только для операндов целого типа и приводит к результату целого типа, получаемого отбрасыванием дробной части частного (т.е. игнорируется остаток). Например,  $11 \div 6$  есть 1,  $47 \div 48$  есть 0. Последовательность операций в выражениях, содержащих символ  $\div$ , может повлиять на результат. Например,  $5 \times 6 \div 4$  дает 7, в то время как значение выражения  $5 \times (6 \div 4)$  будет 5.

Операция возведения в степень приводит к результатам различных типов в зависимости от величины и типа основания и показателя. Если ввести обозначения:  $i$  — величина целого типа,  $r$  — величина действительного типа,

$a$  — величина произвольного типа, то результат операции возведения в степень определяется следующими правилами:

$$a \uparrow i = \begin{cases} a \times a \times \dots \times a \text{ (} i \text{ раз), если } i > 0; \text{ тип совпадает с типом } a; \\ 1, \text{ если } i = 0 \text{ и } a \neq 0; \text{ тип совпадает с типом } a; \text{ неопределено,} \\ \text{если } i = 0, a = 0; \\ 1/(a \times a \times \dots \times a) \text{ (в знаменателе } i \text{ сомножителей), если } i < 0 \\ \text{и } a \neq 0; \text{ действительный тип;} \\ \text{неопределено, если } i < 0 \text{ и } a = 0. \end{cases}$$

$$a \uparrow r = \begin{cases} \exp(r \times \ln(a)), \text{ если } a > 0, \text{ т. е. } e^{r \ln a}, \text{ действительный тип;} \\ 0.0, \text{ если } a = 0 \text{ и } r > 0; \text{ действительный тип;} \\ \text{неопределено, если } a = 0 \text{ и } r \leq 0; \\ \text{неопределено, если } a < 0. \end{cases}$$

2. **Логическое выражение.** Логическое выражение составляется с помощью знаков пяти логических операций из следующих элементов, называемых первичными логическими выражениями:

- логических констант **true** и **false**;
- переменных логического типа;
- указателей логических функций, т. е. функций, которые, согласно описанию, имеют тип **Boolean**;
- отношений.

Любое логическое выражение, состоящее из перечисленных элементов, может быть превращено в первичное путем заключения его в круглые скобки. Круглые скобки могут, таким образом, использоваться в логических выражениях для обозначения группировки по обычным правилам.

*Отношением* называется выражение, состоящее из двух простых арифметических выражений, связанных одним из шести символов  $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$ ,  $\neq$ , определяющих операции отношения. Таким образом, в общем виде отношение запишется как  $E_1 R E_2$ , где  $E_1$ ,  $E_2$  — простые арифметические выражения, а  $R$  — любая из операций отношения. Отношение может быть истинно или ложно в зависимости от того, выполнено оно или нет для значений выражений  $E_1$  и  $E_2$ , иными словами, отношение представляет собой частный случай логического выражения, принимающего значения либо **true** либо **false**. Например, если переменные  $x$  и  $y$  имеют соответственно целые значения 2 и 3, то отношения

$$x + y > x - y \quad x \neq y - x + 1 \quad 2 < x$$

имеют значения: **true**, **false**, **false**.

Простейшая логическая операция, которая рассматривается в логическом выражении, это операция отрицания. Обозначается она символом  $\neg$  и всегда относится только к одному логическому выражению, т. е. является одноместной операцией. Если первичное логическое выражение  $B$  имеет значение **true**, то выражение  $\neg B$  (читается «не  $B$ ») имеет значение **false**, если  $B$  есть **false**, то  $\neg B$  будет **true**.

Операция, обозначаемая символом  $\wedge$ , называется логическим умножением (конъюнкцией). Логическое выражение  $B \wedge L$  (читается « $B$  и  $L$ ») имеет значение **true** только в том случае, когда оба первичных логических

выражения и  $B$ , и  $L$  имеют значение **true**. Во всех остальных случаях выражение  $B \wedge L$  имеет значение **false**.

Операция логическое сложение (дизъюнкция) обозначается символом  $\vee$ . Логическое выражение  $B \vee L$  (читается « $B$  или  $L$ ») имеет значение **true**, если значение **true** имеет хотя бы одно из первичных логических выражений. В случае, когда и  $B$ , и  $L$  имеют значение **false**, выражение  $B \vee L$  имеет значение **false**.

Логическая операция, обозначаемая символом  $\supset$ , называется импликацией. Логическое выражение  $B \supset L$  (читается « $B$  влечет  $L$ ») имеет значение **true** во всех случаях, кроме одного, когда первичное логическое выражение  $B$  имеет значение **true**, а  $L$  — **false**.

Тождеством называется логическая операция, обозначаемая символом  $\equiv$ . Если первичные логические выражения  $B$  и  $L$  оба имеют значение **true** или **false**, то  $B \equiv L$  (читается « $B$  эквивалентно  $L$ ») имеет значение **true**, в противном случае  $B \equiv L$  имеет значение **false**.

Между двумя первичными логическими выражениями, как правило, пишется лишь один символ логической операции, за исключением случая, когда вторым символом является символ  $\neg$  операции отрицания. Примеры логических выражений:

$$A \vee \neg B \quad A \wedge x < y \quad \neg E \equiv A$$

В этих примерах предполагается, что  $x, y$  — переменные типа **integer** или **real**;  $A, B, E, F$  — типа **Boolean**.

В одном выражении логические операции выполняются последовательно слева направо с соблюдением следующего правила рангов: сначала выполняется операция  $\neg$ , затем —  $\wedge$ , следующей — операция  $\vee$ , за нею  $\supset$  и в последнюю очередь — операция  $\equiv$ . Последовательность выполнения операций может быть изменена, если использовать круглые скобки.

Если в логическом выражении содержатся символы операций отношения, то эти операции выполняются последовательно слева направо до логических операций, так как отношения являются первичными логическими выражениями. Все операции отношения являются одноранговыми. Арифметические операции, если они содержатся в логическом выражении, выполняются еще раньше (с соблюдением своего правила рангов).

### Упражнения

1. Определить порядок выполнения операций в следующих простых выражениях:

$$\sin(a + b) \times b[m, n + 3] \quad 2 + \sin(x) > y \equiv A \\ a \uparrow (b + c) - (a - b) \quad x \leq y \vee x \uparrow 2 = x \times y$$

2. Определить значения следующих выражений:

$$3 \times (i/j) \times 2 + k \quad 3 \times (i + j) \times 2 + k \quad \neg a > b \wedge a < c$$

если переменные  $i, j, k, a, b, c$  относятся к целому типу и их значения соответственно равны 7, 2, -3, 3, 4, 5.

3. Составить арифметические выражения, соответствующие математическим формулам:

$$\begin{array}{ll}
 a + \frac{b}{c+a} & \frac{x}{a} - \frac{1}{b} \ln(a + be^{kx}), \\
 (x+y) \frac{ky}{ly} & \sin(2\pi + x) + \sin(\pi + 2x), \\
 \left(\frac{q_1}{q_2}\right)^{\frac{y}{y-1}} & a^{b^{a+b}}, \\
 \left[2 \sin\left(\arctg \frac{x}{2}\right)\right]^{\sin \frac{x}{2}} & \sqrt{a^2 + b^2} + 4 \sqrt{\frac{a^2}{b^2}}.
 \end{array}$$

4. Определить, соответствуют ли приведенные в правом столбце выражения на алголе стоящим слева математическим выражениям:

$$\begin{array}{ll}
 a^{b^2} & a \uparrow b \uparrow 2 \\
 ab^2 & (a \times b) \uparrow 2 \\
 \frac{ab}{x+y} & a \times b / (x+y) \\
 \left(\frac{a+b+c+\pi}{2d}\right)^2 & (a+b+c+3.1416)/(2 \times d) \uparrow 2
 \end{array}$$

5. Определить, допустимы ли в алголе следующие конструкции и совпадают ли значения выражений:

$$\begin{array}{ll}
 -2 \uparrow 3 \uparrow 2 + 12 \times 8/2 \times 5 & \text{и} \quad -2 \uparrow 3 \uparrow 2 + 12 \times 8 \div 2 \times 5 \\
 (13 - 67/3) \div 3 & \text{и} \quad (13 - 67 \div 3)/3 \\
 (2 \uparrow 5 + 8 \times 5) \div 4 & \text{и} \quad 2 \uparrow 5 \div 4 + 8 \times 5 \div 4
 \end{array}$$

6. Определить эквивалентны ли значения выражений:

$$\begin{array}{ll}
 \text{entier}(abs(x)) & \text{и} \quad abs(\text{entier}(x)) \\
 \text{entier}(y) \div z & \text{и} \quad \text{entier}(y/z)
 \end{array}$$

7. Для каких целочисленных значений  $x$ , заключенных между  $-6$  и  $+5$ , логическое выражение

$$(x+4 \leq 0 \vee x > -4) \wedge abs(x) \geq 1 \wedge (x \leq 3 \vee x-4 > 0)$$

ложно? Можно ли это выражение написать без скобок?

8. Допустимо ли в программе, содержащей описание аггау  $A[0:8, -8:4]$ ; использование переменных с индексами:  $A[7+k, 16 \times k+1]$ , если  $k$  принимает значения на отрезке  $-0.5, +0.5$ ?

## § 4. Организация программы

Программа на алголе, как уже отмечалось в § 1, состоит из последовательности записываемых в строку по определенным правилам основных символов языка. На практике эту последовательность приходится разрывать на части и переносить часть записи с одной строки на другую. Такие переносы не обозначаются никакими дополнительными символами, делаются в любом удобном для программиста месте и не влияют на выполнение программы. Например, выражение

$$x/a[i] + 3.1416 \times f(\cos(x1 + y1), \text{random}[j], b[j] \times c)$$

может быть записано в следующем виде:

$$\begin{aligned} &x/a[i] + 3.1416 \\ &\times f(\cos(x1 + y1), r \\ &andom[j], b[j] \times c) \end{aligned}$$

Программа должна быть *блоком*, который, будь то вся программа или только ее часть, состоит из описаний и операторов. Описания помещаются в начале блока, операторы — после них.

Описания служат для характеристики встречающихся в данном блоке переменных, переключателей и процедур. Описания сами по себе не предписывают каких-либо действий в программе (лишь отводится транслятором место в памяти ЭВМ для описываемых объектов). Операторы, напротив, указывают, какие действия и в какой последовательности должны быть выполнены.

Операторы программы выполняются последовательно в том порядке, в котором они выписаны в программе. В языке существуют операторы, которые позволяют изменить этот порядок.

Операторы бывают следующих видов: присваивания, перехода, процедуры, пустые, условные, циклов, составные и блоки. Первые четыре вида операторов считаются основными, они не содержат внутри себя других операторов. Остальные включают в себя внутренние операторы и предназначены для организации их работы.

Для точного указания места, где кончается оператор, после каждого оператора ставится либо символ ; (точка с запятой), либо символ **end**, либо символ **else**. Использование каждой из таких возможностей определяется синтаксическими правилами языка; наиболее часто употребляется точка с запятой.

Составной оператор образуется размещением нескольких операторов любого вида внутри так называемых операторных скобок **begin** и **end**. Перед основным символом **end** уже нет необходимости ставить точку с запятой. Операторные скобки используются по тому же принципу, что и обычные скобки, поэтому два составных оператора либо вовсе не имеют общих элементов, либо один из них целиком содержится в другом. В начале составного оператора (после символа **begin**) могут быть помещены описания, в этом случае он превращается в блок.

Отдельный участок программы, представляющий собой логически законченную ее часть, которая имеет самостоятельное значение, называется *подпрограммой*. Подпрограмма оформляется по всем правилам построения программы, так что подпрограмма — это независимый блок, обращение к которому осуществляется из другого независимого блока (основной программы или подпрограммы). Подразумевается существование фиктивного блока, охватывающего программу и содержащего описание стандартных функций и задание начальных значений собственным переменным. Собственные переменные рассмотрены в § 15.

В операторы и описания входят выражения, для построения которых используются преимущественно ограничители, являющиеся знаками операций. Операторы и описания строятся из отдельных выражений с помощью группы

ограничителей, называемых разделителями и описателями, изображаемыми служебными словами. Благодаря этому выражения имеют почти обычный в математике вид, а запись операторов напоминает предложения на естественном языке.

### § 5. Ввод и вывод информации

При решении задач на ЭВМ обычно бывает необходимо в ходе выполнения рабочей программы вводить в машину исходные числовые данные. Для этой цели служат операторы ввода, в которых указывается, какие данные должны быть введены в машину и каким переменным присвоены соответствующие значения.

Операторы вывода указывают, значения каких переменных или выражений и в какой последовательности должны быть переданы на выводное устройство.

Операторы ввода и вывода не являются операторами эталонного языка алгол, поэтому для различных трансляторов их запись осуществляется различными способами. Обычно это — некоторые стандартные для данного входного языка идентификаторы с последующим списком оператора — списком идентификаторов (а также выражений или строк в случае оператора вывода) в круглых скобках. В этой главе такими идентификаторами являются слова *ввод* и *вывод*. Таким образом, чтобы осуществить ввод, например, начальных значений переменным, описанным в виде

`integer a, b; real x, y, z; array M [1 : 2, 1 : 2];`

следует написать оператор

`ввод (a, b, x, y, z, M);`

Значения простых переменных (пять чисел) и элементов массива (четыре числа) в указанном порядке должны содержаться во вводном устройстве, например быть набитыми на перфокарты.

Оператор вывода записывается аналогично:

`вывод (a, b, x, y, z, M);`

Если изменить последовательность идентификаторов в списке, то соответственно изменится и порядок вывода. Если в операторе вывода в списке оператора стоит переменная (выражение), то напечатано будет значение этой переменной (выражения). Если же в списке оператора стоит строка, то на печать выводятся элементы этой строки. Например, если в процессе работы программы было вычислено значение переменной  $A$ , равное 5.3 и представляющее собой скорость некоторого объекта в метрах в секунду, то оператор вывода

`вывод ('СКОРОСТЬ _ОБЪЕКТА: _A=', A, '_ _M/C');`

вызовет печатание следующей информации:

**СКОРОСТЬ ОБЪЕКТА: A=5.3 M/C**

## § 6. Оператор присваивания

Оператор *присваивания* является одним из наиболее часто используемых операторов. Он служит для вычисления значения выражения и присписывания (присваивания) этого значения одной или нескольким переменным. В простейшей форме оператор выглядит так:

$$v := E;$$

где  $v$  — переменная,  $E$  — выражение.

Переменная (простая или с индексами), за которой следует символ  $:=$  (знак присваивания), называется левой частью. Одна или несколько левых частей, написанных подряд, называются списком левой части. Оператор присваивания состоит из списка левой части и следующего за этим списком выражения, образующего правую часть. Значение правой части присваивается всем переменным списка левой части.

Символ  $;$  (точка с запятой) указывает конец оператора.

Примеры операторов присваивания:

$x := 8;$  — заменить данное значение  $x$  на значение, равное 8;

$p[l] := (x + y) \uparrow (k - z \uparrow 2)$  — переменной с индексами  $p[l]$  присваивается значение выражения, стоящего справа;

$i := j := k := 0;$  — переменные  $i, j, k$  получают значение, равное 0;

$m := m - 1;$  — значение  $m$  уменьшается на единицу;

$b := \text{false};$  — логической переменной  $b$  присваивается значение «ложь».

Предполагается, что всем переменным, встречающимся в правых частях операторов присваивания, уже присвоены значения некоторыми предыдущими операторами.

Оператор присваивания выполняется по следующим правилам:

— если список левой части содержит переменные с индексами, то все индексные выражения у таких переменных вычисляются слева направо, тем самым выделяются соответствующие им элементы массивов;

— вычисляется значение выражения, стоящего в правой части;

— всем простым переменным списка левой части и выделенным элементам массивов присваивается вычисленное значение правой части.

Например, в операторе присваивания

$$k := a[k + 2] := b[k + 3] := l := k + 1;$$

сначала вычисляется значение индексного выражения для элемента массива  $a$ , т. е. значение  $k + 2$ , затем значение выражения  $k + 3$ , а после этого значение выражения в правой части, т. е. значение  $k + 1$ , и в последнюю очередь все переменные из списка левой части получают это значение ( $k + 1$ ). Так, если начальное значение переменной  $k$  было нулем, то  $a[k + 2]$  будет  $a[2]$ ,  $b[k + 3]$  будет  $b[3]$ ,  $k + 1$  будет 1, и в результате выполнится присваивание

$$k := a[2] := b[3] := l := 1;$$

Допускается случай, когда переменная слева имеет целый тип, а выражение справа — действительный тип, и наоборот. В этом случае значение выражения преобразуется к типу, который имеет переменная в левой части.

Все переменные в списке левой части оператора присваивания должны быть одного типа. Если список левой части имеет логический тип, то выражение справа должно быть только логическим.

### Упражнения

1. Найти ошибки в записи следующих операторов присваивания:

$$\begin{array}{l} 3.14 := a + b; \quad -m := a \uparrow 2 - b \uparrow 2; \\ x + y := z; \quad f(x) := x := x + 1; \end{array}$$

2. Описать в виде операторов присваивания вычислительные операции при решении квадратного уравнения  $ax^2 + bx + c = 0$ .

## § 7. Оператор перехода

Порядок написания операторов, как уже отмечалось, определяет последовательность их выполнения в программе.

Оператор *перехода* служит для того, чтобы в случае необходимости прервать естественную последовательность выполнения операторов и указать, какой из операторов программы должен выполняться следующим. Чтобы такое указание было возможным, перед операторами ставятся метки, к которым и адресуются операторы перехода.

*Метка* представляет собой идентификатор перед оператором, отделенный от него символом : (двоеточие). Помеченный оператор может быть для удобства таким же способом помечен еще раз, но другой меткой. Два оператора в одном блоке не должны иметь совпадающих меток. Метка перед оператором может отсутствовать.

Оператор перехода записывается в виде

**go to M;**

где  $M$  — именуемое выражение. Такой оператор показывает, что следующим должен выполняться оператор, имеющий метку, задаваемую именуемым выражением  $M$ . Например, если в программе содержится оператор  $M1: x := a + b$ ; то оператор перехода **go to M1**; указывает, что следующим должен выполняться оператор с меткой  $M1$  независимо от его местоположения в программе относительно оператора перехода. Затем выполняется оператор, следующий за оператором, на который было передано управление.

**Пример.** Пусть требуется написать программу для вычисления таблицы значений функции  $f = f(x)$ , если заданы значения аргумента  $x$  и параметры  $a, b, c$ :

$$f(x) = \frac{\sqrt{x} \sin x / (ax + e^x)}{\sqrt{a^2 + (2\pi bx - 1/(2\pi cx))^2}}$$

Предположим, что параметры имеют постоянные значения, а значения аргумента  $x$  набиты на перфокартах по одному значению на каждой карте.

Блок-схема программы изображена на рис. 2, и программа может быть записана следующим образом:

```
begin real a, b, c, f, x; ввод (a, b, c); вывод (a, b, c);
      M: ввод (x);
```

```

f := sqrt(x) * sin(x) / (a * x + exp(x)) / sqrt(a ↑ 2 + (2 * 3.1416 * b *
x - 1 / (2 * 3.1416 * c * x)) ↑ 2);
вывод (f, x); go to M

```

end

Программа как блок начинается с символа **begin** и заканчивается символом **end**. В конце программы символ ; не пишется. Не ставится точка с запятой также перед **end**.

Напечатанный сразу после ввода список параметров задачи, с одной стороны, позволяет проверить правильность ввода данных, с другой — служит

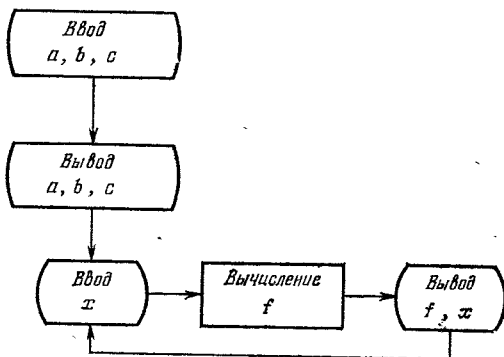


Рис. 2.

заголовком для печатающейся впоследствии таблицы результатов, которая содержит как функцию  $f$ , так и аргумент  $x$ .

В написанной программе оператор перехода, после того как напечатан очередной результат, передает управление на ввод следующей перфокарты, вызывая тем самым повторение вычислений. После прочтения последней перфокарты вычислительная машина будет пытаться ввести очередную карту, так как нет в программе указания об окончании ввода — это недостаток программы. Другой вариант этой программы, свободный от этого недостатка, будет дан ниже.

В программе расположение операторов и других элементов конструкции по строкам, как известно, произвольное. Для большей наглядности каждый программист располагает элементы программы так, чтобы было удобно ее читать. Обычно выделяют операторные скобки, располагая их друг под другом попарно, однородные операторы также располагают друг под другом, метку не отрывают от оператора, начинают строки не с самой левой позиции, выделяют структурные элементы программы (блоки) и т. д.

#### Упражнения

1. Написать фрагмент программы, который производил бы последовательно вычисление значения  $y$  по следующим формулам:

$$\frac{x^3 + x^2 - 1}{\sin x + x - 2}, \quad \frac{x^2 + 1}{\sin^2 x + 1}, \quad \frac{x - 1}{\sin x + \cos x},$$

при заданном значении  $x$ , а для вывода результата использовался бы только один оператор.

2. Не меняя последовательности расположения пяти операторов присваивания, содержащихся в программе, обеспечить их выполнение в следующем порядке: пятый, второй, первый, третий, четвертый.

## § 8. Условие. Условный оператор

Рассмотренный выше оператор перехода позволяет изменить последовательность выполнения операторов безусловно. Поэтому его можно назвать оператором безусловного перехода. Вообще, *безусловным* оператором называется любой основной оператор, оператор цикла и составной оператор.

*Условием* называется конструкции вида **if**  $R$  **then**, где  $R$  — логическое выражение.

*Условный оператор* (или *условный оператор if*) представляет собой последовательность, состоящую из условия и безусловного оператора:

**if**  $R$  **then**  $S$ ;

где  $R$  — любое логическое выражение,  $S$  — произвольный безусловный оператор. Если  $S$  является оператором перехода, то эта конструкция есть *условный оператор перехода*.

Действие этого оператора заключается в следующем: вычисляется значение логического выражения  $R$ ; если значение  $R$  есть **true**, то выполняется оператор  $S$ ; если же значение  $R$  будет **false**, то оператор  $S$  не выполняется и управление передается следующему за ним оператору. Например:

**if**  $x < 2$  **then go to**  $M$ ;

**if**  $x + y > z$  **then begin**  $z := a \times b \uparrow 3$ ; **go to**  $M2$  **end**;

В последнем примере после **then** стоит составной оператор, требующий заключения в операторные скобки **begin** и **end**. Они здесь существенны. Без них действие этого оператора будет правильным до тех пор, пока логическое выражение (в данном случае отношение) имеет значение **true**, но как только оно получит значение **false**, происходит передача управления к оператору перехода, минуя лишь оператор присваивания.

Используя условный оператор, можно составить лучший вариант программы для примера из предыдущего параграфа.

Предположим, что вместо чтения с перфокарты каждого нового значения аргумента  $x$  вводят в начале работы программы с одной перфокарты вместе с параметрами задачи три значения  $x_0$ ,  $xk$ ,  $xh$ , являющиеся соответственно наименьшим и наибольшим значениями аргумента  $x$  и шагом его изменения.

Общая схема вычислений включает получение функции  $f(x_0)$ , затем многократное изменение значения  $x$  каждый раз на величину  $xh$  и окончание процесса, если будет достигнуто значение  $x = xk$ . Блок-схема программы представлена на рис. 3, а программа примет следующий вид:

**begin real**  $a, b, c, f, x, x_0, xk, xh, p2$ ;

*ввод* ( $a, b, c, x_0, xk, xh$ ); *вывод* ( $a, b, c$ );

$x := x_0$ ;  $p2 := 2 \times 3.1416$ ;

$M: f := \text{sqrt}(x) \times \sin(x) / (a \times x + \exp(x)) / \text{sqrt}(a \uparrow 2 + (p2 \times b \times x - 1 / (p2 \times c \times x)) \uparrow 2);$   
 вывод ( $t, x$ );  
 $x := x + xh$ ; if  $x \leq xk$  then go to  $M$   
 end

Оператор останова, который отмечен в блок-схеме, в программе фактически отсутствует, поскольку в случае значения **false**, полученного отношением  $x \leq xk$ , управление переходит на конец программы — последний в

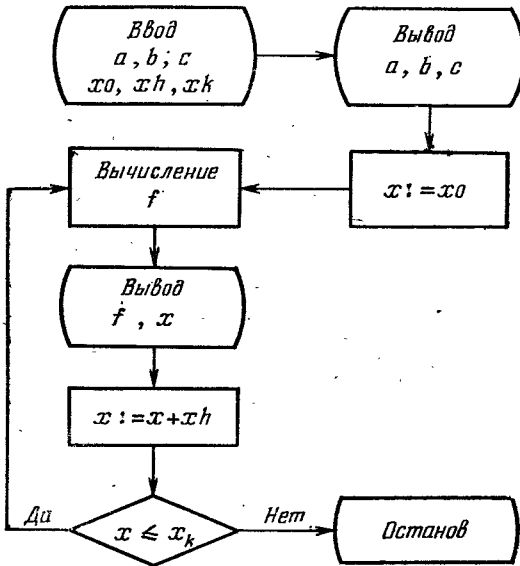


Рис. 3.

программе символ **end**. В конкретных представлениях оператор останова может быть явно указан.

Пример. Рассмотренных операторов достаточно для написания программы вычисления корней квадратного трехчлена в соответствии с блок-схемой рис. 1 § 3 гл. I:

```

begin real a, b, c, x1r, x2r, x1i, x2i, D, S;
  integer k; ввод (a, b, c);
  if a = 0 then begin if b = 0 then
    begin k := 0; вывод (a, b, c, k); go to L end;
    k := 1; x1r := -c/b; x2r := 0; go to M end;
    k := 2; D := b ↑ 2 - 4 × a × c;
  if D < 0 then begin
    x1r := x2r := -b/(2 × a); x1i := sqrt(-D)/(2 × a);
    x2i := -x1i; go to N end;
  S := sqrt(D);

```

$x1r := (-b + S)/(2 \times a); x2r := (-b - S)/(2 \times a);$   
 $M := x1i := x2i := 0;$   
 $N: \text{вывод}(a, b, e, k, x1r, x2r, x1i, x2i);$   
 $L: \text{end}$

Отметим, что после условия  $a = 0$  стоит условный оператор, входящий в составной оператор. Благодаря операторным скобкам оператор после **then** уже является безусловным. Метки  $M$ ,  $N$  и  $L$  введены в программу для того, чтобы в соответствующих местах осуществить безусловную передачу управления.

### Упражнения

1. Вычислить последовательные значения  $F$  как функции переменной  $x$ , меняющейся с шагом 0.01 от значения  $x = 0$  до  $x = 5$ . Напечатать значения  $F$  вместе с соответствующим значением аргумента и порядковым номером вычисленного значения. Функция  $F$  задается формулой

$$F = F_0 + F_1 \frac{x}{1!} + F_2 \frac{x^3}{3!} + F_3 \frac{x^5}{5!},$$

$$F_0 = 1, F_1 = 0.5, F_2 = 0.3, F_3 = 0.1.$$

2. Вычислить  $F_x$  и  $F_y$  по следующим формулам:  $F_x = x + \sin^2 y$ ,  $F_y = y - \cos^2 x$ , но при выполнении условия  $x > y$  учесть, что  $F_x$  вычисляется по второй формуле, а  $F_y$  — по первой.

3. Написать программу для реализации алгоритма, описанного в упражнении 1 § 3 гл. I.

## § 9. Полный условный оператор

Полный условный оператор в простейшем случае записывается в виде **if R then S1 else S2**

где  $R$  — логическое выражение,  $S1$  — безусловный оператор,  $S2$  — произвольный оператор. Например:

**if**  $x > 0$  **then**  $y := \text{sqrt}(x)$  **else**  $y := x \uparrow 2$ ;  $S$ ;  
**if**  $k = l$  **then** **go to**  $M$  **else** **go to**  $N$ ;  $S$ ;

Действие полного условного оператора сводится либо к выполнению безусловного оператора  $S1$ , стоящего за условием, в случае, когда логическое выражение  $R$  имеет значение **true**, либо к выполнению оператора  $S2$ , стоящего после разделителя **else**. После этого выполняется либо оператор, стоящий вслед за полным условным оператором (оператор  $S$ ), либо оператор, указываемый исполненным оператором. Так, в первом из приведенных примеров операторы выполняются в такой последовательности:

при  $x > 0$ :  $y := \text{sqrt}(x)$ ;  $S$ ;  
 при  $x \leq 0$ :  $y := x \uparrow 2$ ;  $S$ ;

а во втором примере:

при  $k = l$ : **go to**  $M$ ;  $M$ : ...;  
 при  $k \neq l$ : **go to**  $N$ ;  $N$ : ...;

Оператор  $S_2$ , указанный в определении полного условного оператора, может быть в свою очередь условным (или полным условным) оператором. В этом случае условные операторы как бы вкладываются друг в друга. Очевидно, возможностей у такого вложенного оператора больше, он может успешно применяться при программировании разветвляющихся процессов и служит для того, чтобы в зависимости от значений входящих в их состав логических выражений выбрать и выполнить один из нескольких содержащихся в нем внутренних операторов.

В соответствии со сказанным полный условный оператор определяется в общем случае как один или несколько условных операторов, отделенных один от другого разделителем `else`, последний оператор в этом списке может не быть условным оператором. Например:

```
if R1 then S1 else S2;
if R1 then S1 else if R2 then S2;
if R1 then S1 else if R2 then S2 else S3;
```

где  $R_1$ ,  $R_2$  — логические выражения,  $S_1$ ,  $S_2$ ,  $S_3$  — безусловные операторы. Цепочку вложенных условных операторов можно продолжать до какой угодно длины.

Выполнение полного условного оператора происходит в таком порядке. Поочередно слева направо проверяются входящие в него условия, т. е. вычисляются значения логических выражений в этих условиях, пока не будет найдено первое выражение, имеющее значение `true`. В этом случае выполняется следующий за этим условием безусловный оператор, а за ним — либо оператор, следующий за всем полным условным оператором, либо оператор, указываемый выполненным оператором. Если же ни одно из условий не удовлетворяется (все логические выражения, входящие в условия, принимают значение `false`), то выполняется последний безусловный оператор (стоящий непосредственно после последнего символа `else`) или же не выполняется ни один из внутренних безусловных операторов (когда после последнего `else` стоит также условный оператор). В последнем случае результат выполнения полного условного оператора эквивалентен работе пустого оператора.

Таким образом, в результате проверки условий может быть выбран и выполнен самое большее один внутренний безусловный оператор, после чего осуществляется переход к оператору, указываемому выполненным оператором, или к оператору, следующему за полным условным оператором. Этот принцип сохраняется и в том случае, если внутренний оператор содержит метку и работает в результате выполнения оператора перехода, ведущего к этой метке.

Блок-схемы на рис. 4 являются иллюстрацией исполнения приведенных выше трех модификаций полного условного оператора. Здесь буквой  $S$  обозначен оператор, стоящий за полным условным оператором, и предполагается, что ни один из безусловных операторов  $S_1$ ,  $S_2$ ,  $S_3$  не является оператором передачи управления.

**Примеры.**

1. Полный условный оператор, реализующий вычисление значения  $y$  по одной из трех формул:

$$y = \begin{cases} \ln x, & \text{если } x > 0, \\ \ln(-x), & \text{если } x < 0, \\ 0, & \text{если } x = 0, \end{cases}$$

может быть записан в виде

if  $x > 0$  then  $y := \ln(x)$  else if  $x < 0$  then  $y := \ln(-x)$  else  $y := 0$ ;

2. Пусть задан следующий полный условный оператор:

if  $x < y \wedge y < z$  then M1:  $a := b + c$  else M2:  $a := b - c$ ; go to M;

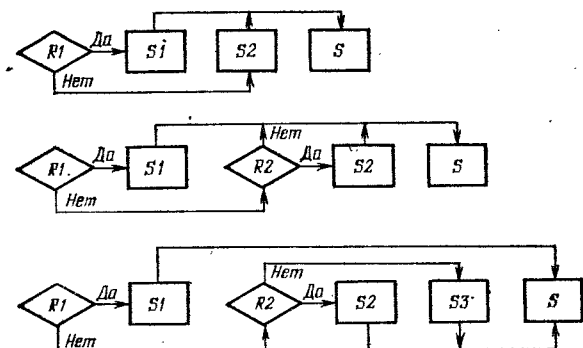


Рис. 4.

Если в программе осуществляется переход к содержащимся в этом полном условном операторе помеченным операторам, то:

после go to M1; будут выполнены операторы  $a := b + c$ ; go to M;

после go to M2; будут выполнены операторы  $a := b - c$ ; go to M;

### Упражнения

1. Переписать программу вычисления корней квадратного трехчлена  $ax^2 + bx + c$ , приведенного в предыдущем параграфе, с использованием полного условного оператора.

2. Вычислить сумму квадратов сумм отрицательных и положительных элементов последовательности  $a_1, a_2, \dots, a_k$ . Результат вывести на печать.

3. Вычислить

$$y = \begin{cases} a_{11}x^2 + a_{12}x + a_{13}, & \text{если } k = 1, \\ a_{21}x^2 + a_{22}x + a_{23}, & \text{если } k = 2, \\ a_{31}x^2 + a_{32}x + a_{33}, & \text{если } k = 3, \end{cases}$$

и вывести на печать значение  $y$  с соответствующими коэффициентами  $a_{ij}$  ( $i, j = 1, 2, 3$ ) и  $k$ .

4. Эквивалентны ли следующие конструкции:

1) if  $x > y$  then  $z := \ln(x - y)$  else go to L;

2) if  $x \leq y$  then go to L else  $z := \ln(x - y)$ ;

3) if  $x > y$  then  $z := \ln(x - y)$ ; go to L;

4) if  $x \leq y$  then go to L;  $z := \ln(x - y)$ ;

## § 10. Условие в выражениях

Простые выражения, рассмотренные в § 3, в общем случае являются составной частью выражений, содержащих условие. В дальнейшем понятие «выражение» будет относиться к выражению, содержащему условие.

*Арифметическим выражением* называется либо простое арифметическое выражение, либо последовательность, состоящая из условия и двух простых арифметических выражений, разделенных символом *else*, т. е. последовательность

$$\text{if } R \text{ then } A1 \text{ else } A2$$

где  $R$  — логическое выражение,  $A1$ ,  $A2$  — простые арифметические выражения. В наиболее общем случае на месте  $A2$  может стоять конструкция, совпадающая с только что определенной. Например:

$$\begin{aligned} &\text{if } a > 0 \text{ then } x \text{ else } y \\ &\text{if } b < 0 \text{ then } -1 \text{ else if } b > 0 \text{ then } 1 \text{ else } 0 \end{aligned}$$

Во втором примере на месте  $A2$  стоит выражение, начинающееся с условия. Таким образом, арифметическое выражение в общем случае может содержать несколько условий; при этом должно выполняться требование, чтобы арифметическое выражение, стоящее между условием и последующим разделителем *else* было простым. Если же возникает необходимость включить условие в это выражение (т. е. в  $A1$ ), то оно должно быть заключено в круглые скобки. Например:

$$\begin{aligned} &\text{if } x = 0 \text{ then (if } y < z \text{ then } A \text{ else } A \times B) \\ &\text{else if } x = \text{entier}(x) \text{ then } C \text{ else } A \times B - C \end{aligned}$$

В выражение, стоящее за *else*, всегда включается максимальное количество символов, т. е. под  $A2$  подразумевается самое длинное среди всех арифметических выражений, которые можно выделить непосредственно справа от разделителя *else*. Таким образом, в последнем примере в случае нецелых значений  $x$  необходимо рассматривать выражение  $A \times B - C$ , а не  $A$  и не  $A \times B$ .

Значение и тип арифметического выражения определяются значением и типом выбираемого после выполнения всех условий простого арифметического выражения.

Отметим особо, что в отношении могут входить только простые арифметические выражения; во все другие конструкции алгола могут входить арифметические выражения, содержащие условие.

При записи и вычислении логических и именующих выражений используются те же принципы.

*Логическим выражением* называется либо простое логическое выражение, либо последовательность, имеющая вид

$$\text{if } R \text{ then } B1 \text{ else } B2$$

где  $B1$  — простое логическое выражение;  $R, B2$  — простые логические выражения или выражения, имеющие структуру в соответствии с данным определением.

Например:

$$\text{if } a > b \text{ then } b > c \text{ else } b \leq c$$

$$\text{if if } x \text{ then } y \text{ else } z \text{ then } x1 \text{ else } y1 \text{ then } z1 \text{ else } d < e$$

Здесь  $a, b, c, d, e$  — переменные типа **integer** или **real**, а  $z, x1, y1, z1$  — переменные типа **Boolean**.

Значение логического выражения, содержащегося в условии, определяет выбор одного из двух выражений — предшествующего символу **else** или следующего за ним. Значение выбранного логического выражения вычисляется и принимается в качестве значения всего логического выражения.

Таким образом, второй пример, если воспользоваться скобками для пояснения его структуры, выглядит так:

$$\text{if (if (if } x \text{ then } y \text{ else } z) \text{ then } x1 \text{ else } y1) \text{ then } z1 \text{ else } (d < e)$$

*Именуемым выражением* называется либо простое именуемое выражение, либо последовательность, состоящая из условия, простого именуемого выражения, разделителя **else** и именуемого выражения, которое в свою очередь может содержать условие, т. е. последовательность

$$\text{if } R \text{ then } M1 \text{ else } M2$$

где  $R$  — логическое выражение,  $M1$  — простое именуемое выражение,  $M2$  — простое именуемое выражение или выражение, имеющее структуру приведенной в определении последовательности.

Например:

$$\text{if } \text{abs}(x - xk) > e \text{ then } M \text{ else } N$$

где  $x, xk, e$  — переменные,  $M, N$  — метки операторов.

Использование выражений, содержащих условие, в операторах присваивания и перехода позволяет сделать запись этих операторов более компактной. Например, вместо громоздкого полного условного оператора, содержащего два оператора присваивания,

$$\text{if } x = y \text{ then } z := 0 \text{ else } z := 1;$$

может быть записан один компактный оператор присваивания

$$z := \text{if } x = y \text{ then } 0 \text{ else } 1;$$

а условному оператору вида

$$\text{if } \text{abs}(x - xk) < 10 - 6 \text{ then go to } M \text{ else go to } N;$$

следует предпочесть оператор перехода в форме

$$\text{go to if } \text{abs}(x - xk) < 10 - 6 \text{ then } M \text{ else } N;$$

Пример. Составить программу вычисления значения  $M$  по формулам

$$M = \begin{cases} \frac{-F(A+B)}{(1+A^2x^2)(1+B^2x^2)} - \frac{ABx^2}{1+B/A} & \text{при } x < 12, \\ \frac{-F(C+D)}{(1+C^2x^2)(1+D^2x^2)} + \frac{CDx^2}{1+C/D} & \text{при } x \geq 12, \end{cases}$$

где аргумент  $x$  изменяется от 2 до 20 с шагом 0.5, а  $A, B, C, D, F$  заданы.

Блок-схему можно построить в виде, показанном на рис. 5, а программу для решения рассматриваемой задачи — в форме

```
begin real A, B, C, D, F, M, x, y, S1, S2, P1, P2, A2, B2, C2, D2;
  ввод (A, B, C, D, F);
  S1 := -F × (A + B); S2 := -F × (C + D);
  P1 := A × B / (1 + B/A); P2 := C × D / (1 + C/D);
  A2 := A ↑ 2; B2 := B ↑ 2; C2 := C ↑ 2; D2 := D ↑ 2; x := 2;
MET: y := x ↑ 2; M := if x < 12 then
  S1 / ((1 + A2 × y) × (1 + B2 × y)) - P1 × y else
  S2 / ((1 + C2 × y) × (1 + D2 × y)) + P2 × y; вывод (M, x);
if x ≥ 20 then go to NET; x := x + 0.5; go to MET;
NET: end
```

В этой программе, кроме идентификаторов, обозначающих параметры задачи, введено несколько дополнительных обозначений. Это обычный прием

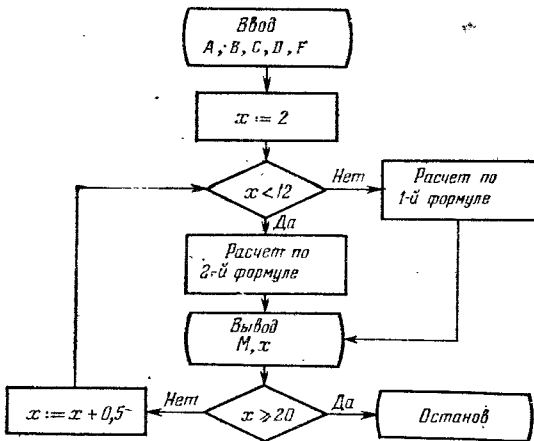


Рис. 5.

программирования, когда выражения, много раз встречающиеся в алгоритме, вычисляются только один раз, а получаемое значение, будучи присвоенное некоторой промежуточной переменной, затем используется в записи элементов программы. Одновременно запись программы получается короче и нагляднее.

В программе содержатся две метки, одна из которых помечает оператор вычисления значения  $M$ , вторая — пустой оператор. Обращение к оператору,

помеченному меткой *NET* выполняется многократно, тем самым организуется цикл, при каждом прохождении которого переменная  $x$  получает новое значение, а оператор с меткой *NET* работает только один раз — в конце программы. Арифметическое выражение, значение которого присваивается  $M$ , записано в наиболее общей форме, т. е. с условием.

#### Упражнения

1. Записать программы упражнений 2 и 3 предыдущего параграфа с использованием выражений общего вида.

2. Написать программу вычисления интеграла  $J = \int_a^b f(x) dx$  по формуле

$$\text{трапеций: } J = \frac{h}{2} [f(a) + 2f(a+h) + 2f(a+2h) + \dots + 2f(b-h) + f(b)] + 2f(b-h) + f(b), \text{ где } a, b, h \text{ заданы; } f(x) = \frac{\sqrt{x} \sin x}{x + e^x}; \quad h = \frac{b-a}{n}.$$

## § 11. Пустой и составной операторы

*Пустым* оператором называется оператор, который не вызывает никаких действий и никакими символами не обозначается. Как и любой другой оператор, пустой оператор может быть помечен. Таким образом, пустой оператор используется для того, чтобы поместить метку в определенное место программы. Часто помеченный пустой оператор ставится в конце программы (перед последним *end*), чтобы в зависимости от выполнения некоторых условий можно было прервать вычисления и выйти на конец программы.

Пример с использованием пустого оператора приведен в § 10.

*Составной* оператор служит для объединения группы операторов, которую по смыслу алгоритма следует рассматривать в программе как единый оператор. Для объединения группы операторов в составной, как уже отмечалось в § 4, их достаточно заключить в операторные скобки *begin* и *end*. Так, конструкция

*begin* S1; S2; ...; Sn *end*;

представляет собой составной оператор. Здесь S1, S2, ..., Sn — произвольные операторы.

Например, если при выполнении условия  $a > b$  необходимо одновременно присвоить переменной  $x$  значение  $a$ , а переменной  $y$  — значение  $b$ , а при  $a \leq b$ , наоборот,  $x$  присвоить  $b$  и  $y$  присвоить  $a$ , то соответствующий полный условный оператор будет содержать два составных оператора, в каждом из которых по два оператора присваивания:

if  $a > b$  then *begin*  $x := a$ ;  $y := b$  *end* else  
*begin*  $x := b$ ;  $y := a$  *end*;

Операторы S1, S2, ..., Sn, входящие в составной, могут быть помечены. Эти метки, естественно, могут содержаться в имеющихся в программе операторах перехода. Если осуществляется передача управления на помеченный оператор, входящий в составной, то выполнение последнего уже начинается

не с первого оператора (т. е. с  $S_1$ ), а с оператора, на который было передано управление.

Заканчивается выполнение составного оператора обычно последним простым оператором  $S_n$ . Однако если среди  $S_1, S_2, \dots, S_n$  есть операторы передачи управления, то выход из составного оператора будет осуществляться по соответствующему оператору перехода. Например, в составной оператор

**begin**  $x := 2$ ;  $M: y := 3$ ; **go to**  $N$ ;  $L: z := 5$  **end**;

если ему предшествует оператор **go to**  $M$ ; вход осуществляется через метку  $M$ , а выход — на метку  $N$ , без выполнения оператора с меткой  $L$ .

### Упражнения

1. Составить программу для вычисления площади треугольника и радиусов описанной и вписанной окружностей, если заданы стороны. Предусмотреть проверку существования треугольника.

2. Составить программу, которая проверяет, можно ли построить параллелограмм из отрезков с длинами  $a, b, c, d$ . Предполагается, что все четыре величины заданы и положительны.

## § 12. Переключатели

При составлении программы для сложных разветвляющихся алгоритмов выбор ветви осуществляется, как правило, после проверки каких-либо условий или в зависимости от значения некоторого арифметического выражения. Каждая ветвь обычно начинается с некоторого помеченного оператора, и выбор ветви означает, таким образом, выбор метки из определенного списка меток. Это может быть выполнено с помощью оператора перехода с именуемым выражением, содержащим условия, или же с помощью соответствующего условного оператора, в котором содержится несколько операторов перехода.

Например, если переменная  $i$  принимает значения 1, 2, 3, 4, 5 и в зависимости от этого значения необходимо передать управление соответственно к операторам, помеченным метками  $M_1, M_2, M_3, M_4, M_5$ , то такую передачу управления можно осуществить оператором перехода с довольно громоздким именуемым выражением

**go to if**  $i = 1$  **then**  $M_1$  **else if**  $i = 2$  **then**  $M_2$  **else if**  $i = 3$  **then**  $M_3$  **else if**  $i = 4$  **then**  $M_4$  **else**  $M_5$ ;

Для получения компактной записи алгоритмов с большим числом разветвлений предусмотрены *переключатели*.

Понятие переключателя содержит описание переключателя и указатель переключателя. Описание переключателя задается в следующей форме:

**switch**  $p := L$ ;

где  $p$  — идентификатор переключателя,  $L$  — список переключателя. Список переключателя состоит из именуемых выражений, отделенных друг от друга запятыми.

Например, все метки, характеризующие различные ветви в предыдущем примере, могут составлять список некоторого переключателя с идентификатором, допустим  $Q$ :

**switch**  $Q := M_1, M_2, M_3, M_4, M_5$ ;

Описание переключателя располагается в начале блока среди других описаний и имеет силу только в пределах этого блока.

Указатель переключателя представляет собой конструкцию вида  $p[i]$ , где  $p$  — идентификатор переключателя,  $i$  — индексное выражение. Индексное выражение может быть любым арифметическим выражением, принимающим одно из значений от 1 до  $n$ , где  $n$  — число именуемых выражений в списке переключателя. Если вычисленное значение индексного выражения оказывается не целым, то оно в указателе переключателя округляется до ближайшего целого.

Указатель переключателя является разновидностью простого именуемого выражения и используется в операторе перехода. Следовательно, значения указателя переключателя — это метки. Они выбираются из списка переключателя, содержащегося в описании того самого переключателя, идентификатор которого задан в указателе переключателя в операторе перехода. При этом каждому элементу списка переключателя ставится в соответствие его порядковый номер. В каждом конкретном случае поиск значения указателя переключателя начинается выбором того элемента списка переключателя, порядковый номер которого совпадает со значением индексного выражения указателя переключателя. Если этим элементом является метка, то она и становится значением указателя переключателя. Так, указатель переключателя  $Q$ , описанного выше, имеет вид  $Q[i]$ , где  $i$  принимает значения 1, 2, 3, 4, 5, а соответствующий оператор перехода для выбора одной из меток  $M_1, M_2, M_3, M_4, M_5$  может быть записан так: `go to Q[i]`;

Еще пример описания переключателя:

```
switch s := M, M5, p[i], N, p[i + 5];
```

где  $p$  — идентификатор некоторого другого переключателя.

Для определения значения указателя этого переключателя прежде всего вычисляется значение его индексного выражения. После этого, как уже отмечалось выше, происходит обращение к описанию переключателя с тем же идентификатором, что и в указателе переключателя, и выбирается в списке переключателя элемент (именуемое выражение), имеющий своим номером (считая слева направо) вычисленное значение индексного выражения. Если выбранное именуемое выражение окажется меткой, то процесс на этом заканчивается (например, если в указателе переключателя  $s[k]$  значение  $k$  будет равно 1, 2 или 4). В противном случае начнется вычисление значения найденного нового указателя переключателя (так будет в нашем примере, если  $k = 3$  или 5). Таким образом, процесс вычисления значения именуемого выражения может состоять в переборе целой цепочки указателей переключателей, но в конце концов должен закончиться выбором некоторой вполне определенной метки.

Если вычисленное числовое значение индексного выражения в указателе переключателя оказывается больше числа элементов соответствующего списка или меньше единицы, то значение указателя переключателя считается неопределенным, и соответствующий оператор перехода не определен.

Пример. Пусть требуется вычислить на некотором участке программы один из первых пяти полиномов Лежандра как функцию индекса  $l$  ( $l = 0, 1,$

2, 3, 4). Полиномы задаются формулами

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = \frac{3}{2}x^2 - \frac{1}{2},$$

$$P_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x, \quad P_4(x) = \frac{35}{8}x^4 - \frac{15}{4}x^2 + \frac{3}{8}.$$

Обозначим вычисляемый полином идентификатором  $P$  и введем переключатель, элементами которого являются метки  $L0, L1, L2, L3, L4$ . Операторы получения значения индекса  $l$  приводить не будем, но в программе укажем их место с помощью текста в угловых скобках; в дальнейшем, написание некоторого текста в угловых скобках будет означать, что этот текст не на алгоритмическом языке. Программа имеет вид

```
begin real P, x; integer l; switch q := L0, L1, L2, L3, L4;
  <операторы получения l, x>; go to q[l + 1];
  L0: P := 1.0; go to M; L1: P := x; go to M;
  L2: P := 1.5 × x ↑ 2 - 0.5; go to M;
  L3: P := 2.5 × x ↑ 3 - 1.5 × x; go to M;
  L4: P := x ↑ 4 × 35/8 - x ↑ 2 × 15/4 + 3/8;
  M: вывод (P, x, l)
```

end

### Упражнения

1. Составить описание переключателя и операторы, которые обеспечивают переход к меткам  $M1, M2, M3$  в следующей периодической последовательности:  $M1, M2, M1, M3, M2, M3, M1, M2, M1, M3, M2, M3, \dots$
2. Определить, какие из меток списка переключателя, заданного описанием переключателя **switch**  $T := A, B, C$ ; являющиеся значениями следующих указателей переключателя:  $T[2], T[i + 1], T[j], T[3]$ , если  $i = 0, 1, 2$ ;  $j = 3, 4$ .

## § 13. Оператор цикла

В большинстве вычислительных алгоритмов отдельные участки вычислений повторяются многократно, при этом каждый раз используются, как правило, новые значения исходных данных. Такие вычислительные процессы принято называть циклическими, а повторяемые участки вычислений — циклами.

Для записи алгоритма повторяющихся вычислений может быть использован условный оператор. В частности, он был использован на примере программы в § 10. Однако, принимая во внимание требования удобства, экономичности и наглядности записи алгоритмов циклических вычислительных процессов вследствие их частого появления, в алгоритмическом языке введен специальный оператор цикла.

Оператор *цикла* определяется в виде следующей конструкции:

```
for i := C do S;
```

где  $i$  — параметр цикла,  $C$  — список цикла,  $S$  — произвольный оператор. Перед оператором цикла, как и перед любым другим оператором, может стоять метка.

Оператор цикла обеспечивает многократное выполнение входящего в его состав оператора  $S$ , при этом простая переменная  $i$  целого или действительного типа после каждого выполнения оператора  $S$  получает, вообще говоря, новое значение. Каждое новое значение параметра цикла задается с помощью очередного элемента списка цикла  $C$ . Элементы списка цикла отделяются друг от друга запятыми, если их больше чем один. Количество элементов в списке цикла произвольное.

Конструкция **for  $i := C$  do** называется заголовком цикла. Элементом списка цикла может быть одна из конструкций:

$A1$      $A1$  step  $A2$  until  $A3$      $A1$  while  $B1$

где  $A1$ ,  $A2$ ,  $A3$  — арифметические выражения,  $B1$  — логическое выражение. В одном списке цикла могут содержаться элементы разных видов в произвольной последовательности.

Элемент первого вида задает параметру цикла лишь одно значение, равное значению арифметического выражения  $A1$ . Вслед за присваиванием параметру цикла этого значения выполняется оператор  $S$ . Например, пусть при значениях  $x$ : 1, 2, -3, 4, 5, 10, -6 требуется произвести вычисления по формуле:  $y = x^3 - 2x^2 + 4$  и результаты напечатать. Соответствующий оператор цикла со списком элементов первого вида будет следующий:

**for  $x := 1, 2, -3, 4, 5, 10, -6$  do begin**  
 $y := x \uparrow 3 - 2 \times x \uparrow 2 + 4$ ; *вывод* ( $y$ ) **end**;

В этом примере несколько значений переменной  $x$  (параметра цикла) заданы перечислением. Переменной  $x$  последовательно присваиваются значения арифметических выражений, перечисленных в списке (в частности, числовых констант, как здесь) и для каждого из этих значений выполняется оператор, стоящий за основным символом **do**. После перебора всех значений переменной  $x$  оператор цикла считается выполненным, и происходит переход к выполнению следующего за ним оператора.

Второй вид элемента списка цикла называется элементом типа арифметической прогрессии. Арифметические выражения  $A1$ ,  $A2$ ,  $A3$ , входящие в заголовок цикла, задающие соответственно начальное значение параметру цикла, шаг изменения параметра и значение, ограничивающее изменение параметра цикла, называются началом, приращением и концом цикла.

Выполнение оператора цикла с элементом этого вида производится следующим образом: приращение  $A2$  и выражение  $A3$  (конец) вычисляются один раз перед началом выполнений цикла  $S$ , параметр цикла  $i$  полагается равным значению выражения  $A1$  (начало), затем значение параметра сопоставляется со значением  $A3$ , если  $(i - A3) \times \text{sign}(A2) \leq 0$ , то выполняется оператор  $S$ , следующему за **do**, в противном случае управление передается оператору, стоящему за всем оператором цикла. После выполнения оператора  $S$  параметр цикла изменяется на величину, равную значению выражения  $A2$ , и снова делается проверка на окончание.

В качестве примера перепишем программу вычисления функции  $M$  из § 10 с использованием оператора цикла:

```

begin real A, B, C, D, F, M, x, y, S1, S2, P1, P2, A2, B2, C2, D2;
  ввод (A, B, C, D, F);
  S1 := -F × (A + B); S2 := -F × (C + D);
  P1 := A × B / (1 + B/A); P2 := C × D / (1 + C/D);
  A2 := A ↑ 2; B2 := B ↑ 2; C2 := C ↑ 2; D2 := D ↑ 2;
  for x := 2 step 0.5 until 20 do
    begin y := x ↑ 2; M := if x < 12 then
      S1 / ((1 + A2 × y) × (1 + B2 × y)) - P1 × y else
      S2 / ((1 + C2 × y) × (1 + D2 × y)) + P2 × y; вывод (M, x)
    end
  end
end

```

Нетрудно видеть, что программа стала проще и нагляднее. Отпала необходимость в метках, операторах передачи управления, условных операторах и операторах подготовки цикла. Заметим, что при значении параметра цикла, равном значению выражения «конец», оператор, стоящий за **do**, выполняется.

Третий вид элемента списка цикла называется элементом типа пересчета или элементом итеративного типа. Выполняется в этом случае оператор цикла следующим образом: параметру цикла присваивается значение арифметического выражения  $A1$  и проверяется истинность логического выражения  $B1$ . Если  $B1$  имеет значение **true**, выполняется оператор  $S$ , стоящий за **do**, если же  $B1$  имеет значение **false**, действие цикла прекращается и выполняется оператор, стоящий за полным оператором цикла.

После выполнения оператора  $S$  происходит новое присвоение параметру  $i$  значения  $A1$ . Хотя элемент списка цикла итеративного типа присваивает параметру цикла значение всегда одного и того же выражения, это не значит, что это значение остается одним и тем же. Оно может меняться либо при выполнении внутреннего оператора (точнее, из-за того, что этот оператор изменяет значения переменных, от которых зависит значение арифметического выражения  $A1$ ), либо потому, что арифметическое выражение явно зависит от параметра цикла. В любом случае ведется пересчет значения параметра цикла в соответствии с выражением  $A1$ .

Например, если предварительно в предыдущем примере включить в программу оператор  $x := 2$ ; то заголовок цикла может быть переписан с использованием конструкции итеративного типа:

```
for x := x + 0.5 while x ≤ 20 do
```

Присвоение начального значения параметру цикла может быть выполнено и внутри заголовка цикла:

```
for x := 2, x + 0.5 while x ≤ 20 do
```

В тех случаях, когда в цикле шаг явно не задан, а для окончания цикла необходимо делать какие-либо оценки, конструкция элемента третьего вида предпочтительнее конструкции типа арифметической прогрессии. Выбор той или иной конструкции элемента списка цикла зависит от конкретной задачи.

Например, при вычислении квадратного корня  $r$  числа  $x$  по итерационной формуле  $r = 0.5(x/r1 + r1)$ , где  $r1$  — некоторое начальное приближенное значение  $r$ , значение  $r$  при каждой итерации пересчитывается, поэтому естественно использовать оператор цикла с параметром цикла  $r$  в виде

**for**  $r := 0.5 \times (x/r1 + r1)$  **while**  $abs(r - r1) > 10^{-7}$  **do**  $r1 := r$ ;

В операторы, управляемые оператором цикла, входить можно только при выполнении оператора цикла, т.е. оператор перехода, написанный вне оператора цикла и приводящий к выполнению управляемого оператора, запрещен. Например, в случае оператора цикла

**M: for**  $i := a$  **step**  $b$  **until**  $c$  **do**  $N: S$ ;

разрешено извне оператора цикла передавать управление на метку  $M$ , но запрещено это делать на оператор с меткой  $N$ .

Оператор цикла выполняется до тех пор, пока не исчерпается список элементов цикла. Если же в составном управляемом операторе есть оператор перехода, то действие цикла прекращается до того, как параметр цикла пробежит все значения, предписываемые списком цикла. Так, при выполнении оператора цикла

**for**  $k := 1$  **step**  $1$  **until**  $n$  **do** **begin**  $c := b + a \times k$ ;  
вывод ( $c$ ); **if**  $a \times k \geq b$  **then** **go to**  $L$  **end**;

если произведение  $a \times k$  достигнет значения  $b$ , то действие этого цикла прекратится независимо от значения параметра  $k$  (не достигнутого  $n$ ).

После выхода из оператора цикла, будь то по оператору перехода или по исчерпанию списка цикла, параметр цикла сохраняет последнее присвоенное ему заголовком цикла значение.

Оператор  $S$ , стоящий за **do**, определяется как оператор самого общего вида, в том числе этот оператор может содержать в свою очередь оператор цикла — случай так называемого цикла в цикле. Глубина вложений циклов может быть произвольной. Если на месте  $S$  стоит несколько операторов, они должны быть заключены в операторные скобки.

Вложенные циклы используются, например, при табулировании функции от двух переменных. Фрагмент программы вычисления функции  $F(x, y) = (x^2 - y^2)/(x + y)^2$ , где  $0 \leq x \leq 1$ ,  $0.1 \leq y \leq 0.5$ , с шагом по  $x$ , равным 0.1, и по  $y$ , равным 0.05, записывается следующим образом:

**for**  $x := 0$  **step**  $0.1$  **until**  $1.0$  **do**  
  **for**  $y := 0.1$  **step**  $0.05$  **until**  $0.5$  **do**  
  **begin**  $F := (x \uparrow 2 - y \uparrow 2)/(x + y) \uparrow 2$ ; **вывод** ( $F, x, y$ ) **end**;

Пр и м е р. Вычислить интеграл  $I(t) = \frac{1}{t^2} \int_a^b \frac{e^{-x} \cos \frac{x}{4} dx}{x^3 (e^{t/x} - 1)}$  как функцию

параметра  $t$  методом Симпсона. Интервал изменения параметра  $t$  и пределы интегрирования  $a$  и  $b$  заданы.

Формула Симпсона имеет вид

$$I = \int_a^b f(x) dx = \frac{h}{3} [f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \dots \\ \dots + 2f(b-2h) + 4f(b-h) + f(b)],$$

где  $h = (b-a)/n$ ,  $n$  — четное количество интервалов.

Суммирование в этой формуле можно выполнить по-разному. Естественно использовать особенности членов ряда. Очевидно, достаточно эффективно суммировать отдельно члены с коэффициентом 4 от членов с коэффициентом 2. Поскольку  $n$  — четное, то число членов с коэффициентом 4 на один больше, чем с коэффициентом 2. Тем не менее можно использовать один оператор цикла, а «лишний» член с коэффициентом 4 добавить после выхода из цикла.

Оператор цикла организуется путем прибавления на каждом шаге суммирования значения  $2h$  к параметру цикла  $x$ , начиная со значения  $a+h$ . Тогда параметр цикла  $x$  будет определять член, имеющий коэффициент 4 и суммирующийся, например, в сумму  $S_4$ , а параметр  $x+h$  — член с коэффициентом 2, который является слагаемым другой суммы, скажем,  $S_2$ . После выхода из цикла надо к значению  $S_4 + S_2$  прибавить члены при  $x=a$ , при  $x=b$  и учетверенный член с аргументом  $x=b-h$  и все умножить на  $h/3$ .

Таким образом, фрагмент программы интегрирования по формуле Симпсона примет следующий вид:

```
S4 := S2 := 0;
for x := a + h step 2 × h until b - 3 × h do
  begin S4 := S4 + f(x); S2 := S2 + f(x + h) end;
I := h/3 × (f(a) + f(b) + 4 × f(b - h) + 4 × S4 + 2 × S2);
```

где  $f(x)$  — произвольное подынтегральное выражение.

Обозначим идентификаторами  $t0$ ,  $th$  и  $tk$  переменные, значения которых совпадают с начальным значением, шагом изменения и конечным значением переменной  $t$ . Теперь интеграл  $I(t)$  вычисляется по следующей программе:

```
begin real t, t0, th, tk, a, b, x, h, S4, S2; integer n;
  ввод (t0, th, tk, a, b, n); h := (b - a)/n;
for t := t0 step th until tk do
  begin S4 := S2 := 0; for x := a + h step 2 × h until b - 3 × h do
    begin S4 := S4 + exp(-x) × cos(x/4)/(x ↑ 3 × (exp(t/x) - 1));
      S2 := S2 + exp(-(x + h)) × cos((x × h)/4)/((x + h) ↑ 3 ×
        (exp(t/(x + h)) - 1))
    end;
  I := 1/t ↑ 2 × h/3 × (exp(-a) × cos(a/4)/(a ↑ 3 × (exp(t/a) - 1))
    + exp(-b) × cos(b/4)/(b ↑ 3 × (exp(t/b) - 1)) + 4 × exp(-(b - h))
    × cos((b - h)/4)/((b - h) ↑ 3 × (exp(t/(b - h)) - 1)) + 4 × S4 + 2
    × S2); вывод (I, t)
  end
end
```

end

В этой программе в цикл, выбирающий исходный параметр задачи (значение  $t$ ), вложен другой цикл, который при каждом значении параметра вычисляет определенный интеграл.

Программа громоздка по написанию. Это обусловлено тем, что подынтегральное выражение выписывается для пяти различных аргументов. В дальнейшем, когда будут рассмотрены процедуры, мы увидим, как можно подынтегральное выражение выписать лишь один раз в теле процедуры, а затем обращаться к нему из различных мест программы.

В операторе цикла широко используются переменные с индексами. При этом индексы, как правило, выступают в роли параметров циклов, а алгоритм вычислений выписывается в виде операторов для одного элемента массива, индексы которого изменяются согласно заголовку цикла. В результате операции выполняются над всеми элементами массива.

Например, произведение матрицы порядка  $n$  на  $n$ -мерный вектор, т. е.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix},$$

где  $c_i = \sum_{j=1}^n a_{ij} b_j$ ,  $i = 1, 2, \dots, n$ , может быть вычислено по программе, содержащей два цикла. Необходимо описать двумерный массив  $a$  и два одномерных массива  $b$  и  $c$  и задать значение  $n$ . При  $n = 10$  программа имеет вид

```
begin integer i, j; array a[1 : 10, 1 : 10], b, c[1 : 10];
  ввод (a, b); for i := 1 step 1 until 10 do
    begin c[i] = 0; for j := 1 step 1 until 10 do
      c[i] := c[i] + a[i, j] × b[j]; вывод (c)
    end
end
```

Для того чтобы написанную программу можно было использовать при другом значении  $n$ , переменная  $n$  должна входить в описание массивов. Как это сделать, будет ясно после рассмотрения блочной структуры алгольной программы.

#### Упражнения

1. Переписать программу для упражнения 2 из § 10, используя оператор цикла.
2. Составить программу для вычисления вещественного корня уравнения  $x - \sin x = 0.35$  с заданной абсолютной погрешностью, равной 0.001.
3. Найти значения положительного кубического корня нечетных чисел от 1 до 99, т. е. составить программу для решения уравнения  $x^3 = a$  при  $a = 1, 3, \dots, 99$ , используя итерационную формулу  $x_n = \frac{1}{3} \frac{a}{x_{n-1}^2} + \frac{2}{3} x_{n-1}$ .

В качестве первого приближения для  $\sqrt[3]{a_i}$  использовать значение  $\sqrt[3]{a_{i-1}}$ . Точность: 3 десятичных знака.

4. Составить программу вычисления  $e^x$  для заданного  $x$ , просуммировав 20 членов ряда  $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$

5. Составить программу вычисления  $e^x$  путем разложения в ряд, приведенный в упражнении 4. Прекратить вычисления после того, как очередной член окажется меньше чем число  $10^{-8}$ , умноженное на частичную сумму, полученную до этого.

6. Составить программу для упражнения 3 из § 2 гл. I.

7. Найти наименьший и наибольший (по абсолютной величине) элемент в массивах: а)  $x[1:50]$ , б)  $y[1:25, 0:75]$ , в)  $z[1:10, 10:20, 20:30]$ .

Примечание. В этом, как и в последующих упражнениях, предполагается составление программы.

8. Напечатать одномерный массив  $y$  значений функции  $y(x) = x^3/(5x^2 + 3)^2$  для аргумента  $x$ , изменяющегося следующим образом: от 0 до 1 — с шагом 0.1, от 1 до 10 — с шагом 1, от 10 до 50 — с шагом 5, от 50 до 100 — с шагом 10. Вместе со значением  $y[i]$  напечатать значение номера  $i$ .

9. В двумерном массиве поменять местами строки с номерами  $i$  и  $j$ .

## § 14. Блочная структура программы

*Блок* определяется как последовательность, состоящая из одного или нескольких описаний, операторов и заключенная в основные символы **begin** и **end**. Среди операторов, входящих в блок, могут находиться другие блоки, так как блок сам является оператором. Блок — это безусловный оператор и, как любой оператор, может быть помечен. От составного оператора, как следует из определения, блок отличается тем, что имеет в своем составе описания, размещаемые в начале перед последовательностью операторов. Если среди операторов блока содержатся блоки, то последние называются вложенными, или внутренними, блоками или подблоками.

Сама программа является блоком, который не содержится внутри другого оператора и не использует операторов, не содержащихся в нем. Если обозначить буквами  $D$  и  $S$  соответственно описания и операторы, то конструкцию программы можно представить в следующем виде:

```
begin D; D; ...; S; S; ...; S;
  A: begin D; ...; S; ...;
    B: begin D; ...; S; ...; S end;
      S; ...; S;
    C: begin D; ...; S; ...; S end;
      S; ...; S;
  end
end
```

В приведенном примере блок  $A$  содержится в самом внешнем блоке, т. е. является по отношению к нему подблоком. В блоке  $A$  в свою очередь содержатся подблоки  $B$  и  $C$ . Подблоки являются независимыми, если внутри операторных скобок одного из них не содержится операторных скобок другого подблока. Очевидно, подблоки  $B$  и  $C$  независимые.

Областью действия описания является блок, в начале которого описание помещается. Таким образом, с помощью описаний указывается, что та или иная величина существует (или определена) только внутри данного блока и в этой области она обозначается данным идентификатором. Вне блока описанная в нем величина не определена, а идентификатор, которым она была

обозначена, может быть использован для наименования любой другой величины. Следовательно, каждый идентификатор, описанный в данном блоке, локализуется в нем, т.е. смысл этого идентификатора приобретаетс при входе в блок и теряется при выходе из него. Идентификаторы, описанные в данном блоке, так и называются — локализованными. В начале блока один и тот же идентификатор может быть описан только один раз.

Описанию подлежат все идентификаторы, за исключением идентификаторов стандартных функций, имеющих постоянный, заранее установленный смысл и тип во всех блоках (и программах), меток, а также формальных параметров в описаниях процедур. Считается, что запись идентификаторов с последующим двоеточием в тексте программы перед оператором является описанием этого идентификатора как метки. Формальные параметры процедур включаются в список спецификаций (§ 17).

Если некоторый блок является подблоком другого, объемлющего его блока, то в подблоке могут быть использованы идентификаторы, описанные не в подблоке, а во внешнем блоке. Такие идентификаторы (соответственно и обозначаемые ими величины) называются глобальными по отношению к блоку, в котором они действуют, но не описаны.

Пр и м е р. Рассмотрим фрагмент программы

```
M: begin real a, b, c; <операторы>;
    M1: begin integer i, j; <операторы>; end M1;
    M2: begin real x, y; <операторы>; end M2; <операторы>;
end M;
```

Выражение, стоящее после символа `end` до первой точки с запятой, машиной не воспринимается — это комментарий, т.е. примечание для программиста (§ 16). В данном примере идентификаторы меток после `end` указывают, где кончается помеченный блок.

В рассматриваемом фрагменте содержатся три блока, помеченные метками `M`, `M1`, `M2`. Переменные `a`, `b`, `c` локализованы в блоке `M`, `i`, `j` — в блоке `M1`, а переменные `x`, `y` локализованы в блоке `M2`. Для блоков `M1` и `M2` переменные `a`, `b`, `c` являются глобальными, так как описаны в блоке, объемлющем как блок `M1`, так и блок `M2`. Переменные `i`, `j` не определены за пределами блока `M1` (например, в `M2` на эти переменные нельзя ссылаться), на переменные `x`, `y` нельзя ссылаться в блоке `M1`. Блоки `M1` и `M2` независимы.

Говоря о величинах в некотором блоке, всегда подразумевают величины, описанные в данном блоке или в объемлющем блоке, но не в подблоке.

Использование блоков облегчает составление программ, особенно больших и сложных. В частности, в независимых подблоках можно использовать одни и те же обозначения для совершенно различных целей. Это позволяет одну и ту же область памяти использовать для нескольких идентификаторов. Так, в предыдущем примере для переменных `i`, `j`, `x`, `y` потребуются в памяти машины только две ячейки, так как одновременно используются только значения двух переменных. Транслятор создан так, чтобы такая возможность могла быть реализована. Нетрудно видеть, что переменные `i` и `x`, `j` и `y` могут быть обозначены совпадающими идентификаторами.

Метки также локализованы в том блоке, в котором они встречаются перед операторами. Отсюда следует, что в разных блоках метки могут по написанию не отличаться друг от друга, и невозможно передать управление на внутренний оператор блока из другого блока. Вход в блок осуществляется только через его начало.

В одном блоке каждый идентификатор обозначает только один объект и поэтому может быть описан (или поставлен перед оператором в качестве метки) только один раз. Но при этом любой такой идентификатор может быть вновь описан (или употреблен в качестве метки) в подблоке данного блока.

Блочная структура делает программу более наглядной и, более того, видно, как можно программу составлять по частям. При написании программы всегда рекомендуется выделять один уровень переменных, глобальных для всей программы, а все остальные переменные описывать в подблоках.

Если в программе одним и тем же идентификатором обозначены разные величины, причем одна из них описана во внешнем блоке, а другая — во внутреннем, то глобальная величина становится недействительной во время выполнения подблока, где описана величина с тем же наименованием. Значение глобальной величины не утрачивается, но во время выполнения подблока данный идентификатор относится к локализованной величине. Следовательно, после выхода из внутреннего блока рассматриваемый идентификатор вновь получает прежний смысл (тот, который он имел во внешнем блоке).

Таким образом, областью действия идентификатора является блок, в начале которого помещено его описание, за исключением тех внутренних блоков (вместе с их подблоками), в которых описаны такие же идентификаторы.

Блочный принцип построения программ позволяет описывать массивы, а соответственно и определять размеры массива и выполнять распределение памяти под них при входе в этот блок. Другими словами, если в блоке описан массив с граничными парами, содержащими арифметические выражения, то такое описание вызывает при каждом входе в блок вычисление значений границ индексов. В таком случае память машины распределяется более экономно, так как при каждом входе в блок задается лишь необходимое минимальное число элементов данного массива.

Переменные в арифметических выражениях для граничных пар могут быть только глобальными для того блока, в котором массив описывается, т. е. эти переменные должны быть описаны во внешнем блоке и получить там числовые значения. Следовательно, самый внешний блок может содержать описания массивов только с постоянными границами. Например, возможно такое начало программы:

```
begin integer n, m;
  <операторы ввода или вычисления n, m>;
begin array x[1 : n, 1 : m];
  <продолжение программы>;
```

или такое:

```
begin integer k, l; array y[1 : 10, 1 : 100];
  <продолжение программы>;
```

Используя блочный принцип построения программ, можно видоизменить программу из § 13 для вычисления произведения матрицы  $n$ -го порядка на  $n$ -мерный вектор так, что она будет универсальной, т.е. пригодной при произвольном значении  $n$ . Для этого достаточно предусмотреть еще один внешний блок, описать в нем переменную  $n$  и присвоить ей здесь же значение:

```
begin integer i, j, n; ввод (n);
  begin array a[1:n, 1:n], b, c[1:n]; ввод (a, b);
  for i := 1 step 1 until n do
    begin c[i] := 0; for j := 1 step 1 until n do
      c[i] := c[i] + a[i, j] × b[j]; вывод (c)
    end
  end
end
```

В этом варианте программы для изменения значения  $n$  достаточно заменить при вводе перфокарту, содержащую числовое значение  $n$ . Остальные операторы программы не зависят от конкретного значения  $n$ .

#### Упражнения

1. Написать в виде блока программу, осуществляющую транспонирование квадратной матрицы  $M$  порядка  $m$ .

2. Представить в виде блока алгоритм итеративного метода для решения уравнения  $y = y(x)$  с помощью формул

$$y_{n+1} = y(x_n), \quad n = 0, 1, 2, \dots;$$

$$x_1 = y_1, \quad x_{n+1} = y_{n+1} - \frac{(y_{n+1} - y_n)(y_{n+1} - x_n)}{y_{n+1} - y_n - x_n + x_{n-1}}, \quad n = 1, 2, \dots$$

В блоке локализовать все промежуточные (используемые только в пределах данного алгоритма) величины.

3. Составить программу для упорядочения последовательности значений  $a_1, a_2, \dots, a_n$  в порядке убывания их модулей.

4. Составить программу нахождения четных значений элементов, расположенных на главной диагонали и выше для квадратной матрицы  $n$ -го порядка с целыми элементами. Предусмотреть ввод всех элементов и вывод найденных значений с указанием номеров соответствующих строки и столбца и общего их количества.

5. Поменять столбцы матрицы размером  $m \times n$ , состоящей из элементов  $x_i$ , так, чтобы элементы заданной строки убывали по модулю. В упорядоченной матрице запомнить исходные номера столбцов. Составить программу для произвольного  $i$  и выдать результаты на печать для  $i$ , равного 1.

6. Написать программу вычисления произведения  $C$  двух произвольных матриц  $A$  и  $B$  с размерами  $m \times l$  и  $l \times n$  соответственно.

## § 15. Собственные величины

После выхода из блока все локализованные величины, как отмечалось выше, теряют свои значения, т.е. при возвращении в блок значения этих величин должны вычисляться заново. Это правило следует из принципа локализации величин в блоке и вызвано необходимостью экономии памяти, а также естественным изменением условий работы при новом обращении к блоку. Тем не менее может возникнуть ситуация, когда после выхода из блока

необходимо сохранить значения некоторых величин. Так, например, в итерационном процессе результат предыдущего приближения служит исходным для вычисления последующего и его необходимо сохранить. Требование же описать такие величины во внешнем блоке программы (т. е. объявить их глобальными) лишает данный блок замкнутости и универсальности при решении многих однотипных задач. Кроме того, описание большого количества величин во внешнем блоке делает программу излишне громоздкой.

Описание величины как *собственной* дает возможность сохранить значение этой величины после выхода из блока, в котором она описана.

Величины объявляются собственными с помощью описателя **own**, который ставится впереди других описателей и описаниях этих величин, например:

```
own array A [1 : 30]; own real a, b, c, d;
```

Обратим внимание, что запись **own array** означает **own real array**.

Переменная с описателем **own** ведет себя так, как если бы она была описана в охватывающем блоке, но с тем отличием, что доступна она только в пределах ее собственной области действия. При первом входе в блок, в котором такая переменная описана, она получает значение нуль (если ее тип целый или действительный) или **false** (если переменная логического типа). При повторном входе в блок собственная величина сохраняет то значение, которое она имела в момент последнего выхода из блока (несмотря на то, что она оставалась неопределенной до этого повторного входа).

Для собственных массивов сохраняются значения всех их элементов, если границы индексов при новом входе в блок остались теми же самыми, что и при предыдущем входе. Границы массива, снабженного описателем **own**, должны быть заданы целыми константами. В случае изменения границ индексов собственного массива при очередном входе в блок сохраняются от предыдущего выполнения блока значения лишь тех элементов массива, индексы которых лежат одновременно внутри старых и новых границ. Те элементы массива, индексы которых выходят за границы, соответствующие предыдущему выполнению блока, оказываются неопределенными, даже если они были определены когда-то при более ранних выполнениях этого блока.

Например, если при трех последовательных входах в блок индексы собственного одномерного массива соответственно задаются граничными парами [1 : 20], [4 : 25] и [-5 : 15], то после второго входа в блок сохраняют значения, оставшиеся от первого выполнения блока, элементы массива с индексами от 4 до 20, а после третьего входа в блок сохраняются (от второго выполнения блока) значения элементов с индексами от 4 до 15. Элементы же с индексами 1, 2, 3, хотя и сохранились до второго входа в блок, к третьему обращению к этому блоку теряют свои значения.

Пример. При заданных значениях  $t$  и  $n$  вычислить все частичные суммы ряда

$$S = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \dots + \frac{t^n}{n!} + \dots$$

и напечатать вместе со значением последнего слагаемого.

Возможен следующий вариант программы:

```
begin integer i, n; real sn, t; ввод (t, n); sn := 1;
  for i := 0 step 1 until n do
    begin own real yn; if i = 0 then
      begin yn := 1; sn := 1; вывод (sn, yn) end;
      yn := yn × t; sn := sn + yn; вывод (sn, yn)
    end
  end
```

Здесь во внутреннем блоке переменная  $yn$ , обозначающая последнее слагаемое в частичной сумме  $sn$  (при  $i = 0$   $yn = 1$ , при  $i = 1$   $yn = t$ , при  $i = 2$   $yn = t^2/2!$  и т. д.), описана как собственная. При первом входе в блок (при  $i = 0$ ) переменной  $yn$  присваивается значение 1. В дальнейшем (при  $i > 1$ ) это присваивание пропускается, и новые значения переменной  $yn$  получаются из предыдущих. Обход присваивания исходного значения при повторных входах во внутренний блок достигается с помощью переменной  $i$ , описанной во внешнем блоке и являющейся глобальной для внутреннего блока.

### Упражнения

1. Написать программу вычисления суммы  $S = \sum_{i=1}^n \frac{1}{i}$ , в которой содер-

жатся два блока: внешний, включающий описания переменных  $i$  и  $n$ , присваивание значения переменной  $n$  и заголовок цикла по  $i$ , и внутренний, полностью определяющий величину  $S$ .

2. Составить программу для упражнения 2 из § 14, используя собственные величины.

## § 16. Комментарии

В ряде случаев программа может оказаться слишком сложной для понимания без дополнительных словесных пояснений, называемых *комментариями*. Такие пояснения могут быть непосредственно включены в программу. Комментарии не влияют на выполнение алгоритма: назначение их исключительно в том, чтобы помочь человеку, читающему программу, лучше разобраться в структуре программы, пояснить смысл и назначение всего алгоритма, его отдельных частей или переменных, входящих в алгоритм, отметить те или иные особенности программы.

В тексте программы перед комментарием ставится разделитель **comment**, и весь текст, заключенный между этим символом и первым встретившемся далее символом ; (точка с запятой), при работе транслятора пропускается. Такой комментарий помещается в программе после разделителя ; или после символа **begin** и может состоять из любых литер.

Часто комментарий пишется после окончания какого-либо блока, т. е. после символа **end**. В этом случае символ **comment** можно не писать, а признаком окончания текста комментария является один из символов: или ; или **end** или **else**. Текст комментария в этом случае может состоять из основных

символов языка, за исключением указанных. Например, возможна такая запись программы с комментариями:

```
begin integer i, j; real x, y;
  comment i — НОМЕР СТРОКИ, j — НОМЕР СТОЛБЦА;
  begin <операторы программы> end OF BLOCK;
  <продолжение программы>
end PROGRAM
```

В качестве комментария можно использовать метку. Так, в записи блока  
*VELOCITY CALCULATION: V; begin <операторы> end;*

первая метка поясняет назначение блока (расчет скорости).

Пояснительный текст может быть включен в программу еще при вычленении параметров процедур (§ 17).

### Упражнения

1. Написать программу нахождения минимума функции  $y = x + e^{1/x}$ ,  $x > 0$ , используя свойство смены знака производной  $y'$  при прохождении через минимум. Расчет выполнить с точностью до  $10^{-6}$ . Употребить комментарий при написании программы.
2. Составить программу транспонирования произвольной матрицы. Снабдить программу пояснительным текстом.
3. Переписать программу из упражнения 2 § 14 с подробными комментариями.

## § 17. Процедуры

В ходе решения различных задач на ЭВМ могут встретиться случаи, когда отдельные участки программы используются многократно, однако исходные значения величин могут быть различными. Такие участки могут быть характерными не только для данной конкретной задачи, но и для целого ряда родственных задач. Кроме того, в каждой отрасли науки или техники существуют свои, специфические для данной отрасли типовые зависимости между исследуемыми величинами. Было бы неразумно каждому программисту заново всякий раз составлять программы в указанных выше случаях, поэтому представляется желательным иметь возможность оформить программу только один раз, а пользоваться по мере надобности, задавая при этом новые исходные данные. В языке алгол организация вычислительного процесса подобным образом реализуется введением объектов программы, называемых процедурами.

Оформление алгоритма процедурой выполняется с помощью описания процедуры. Обращение к процедуре для ее выполнения в программе может быть реализовано с помощью двух конструкций: оператора процедуры и указателя функции.

Следовательно, *процедура* — это алгольная программа, блок, составной или простой оператор, исполняющие роль подпрограммы. Такая подпрограмма может быть подключена к рабочей программе транслятором автоматически после указания ее наименования, как в случае стандартных функций, или же с помощью некоторой другой подпрограммы обращения к библиотеке стан-

дартных программ. В любом случае для обращения к процедуре нет необходимости в выписывании составляющих процедуру операторов. Эти операторы приводятся один раз в описании процедуры.

**1. Описание процедуры.** Описание процедуры состоит из заголовка процедуры и тела процедуры. Заголовок процедуры дает название процедуры, а также названия ее параметров и определенную информацию о них. Тело процедуры является либо оператором, либо составным оператором или блоком. Оно представляет собой описание действий над параметрами процедуры. Заголовок процедуры начинается символом **procedure**, за которым следует идентификатор процедуры со списком параметров, разделенных запятыми и заключенных в круглые скобки, список значений (если он есть) и список спецификаций. Заголовок процедуры отделяется от тела процедуры точкой с запятой. Например, запись

```
procedure p(a, b, c, d, e); V; C; T;
```

представляет собой описание процедуры с наименованием  $p$ , параметрами  $a, b, c, d, e$ , списком значений  $V$ , списком спецификаций  $C$  и телом  $T$ .

Параметры процедуры, содержащиеся в описании процедуры, называются формальными параметрами. Ими обозначаются величины, значения которых являются либо исходными данными при выполнении операторов тела процедуры, либо величины, значения которых представляют собой результат выполнения данной процедуры. Формальные параметры не являются какими-либо конкретными объектами в программе; они указывают лишь те позиции в теле процедуры, которые при ее выполнении займут соответствующие им по смыслу реальные объекты. Поясняющая информация о формальных параметрах указывается спецификациями. Формальные параметры составляют список формальных параметров.

Описание процедуры помещается среди других описаний в начале того блока, в котором эта процедура используется.

**Пример.** Процедура вычисления модуля  $R$   $n$ -мерного вектора  $x(x_1, x_2, \dots, x_n)$  может быть записана так:

```
procedure M(x, n, R); array x; integer n; real R;
begin integer i; real S; S := 0;
for i := 1 step 1 until n do S := S + x[i] ↑ 2; R := sqrt (S)
end;
```

Здесь в первой строке задан список спецификаций (см. п. 3).

Описание процедуры определяет процесс реализации определенного алгоритма. По описанию процедуры создается группа команд на машинном языке, которая будет включена в рабочую программу, но никаких действий по этим командам описание не вызывает и вся группа команд пропускается целиком.

Для стандартных функций, являющихся частным случаем процедур, описания не требуются.

**2. Оператор процедуры.** Исполнение операторов, входящих в тело процедуры, возможно лишь после обращения к процедуре, которое реализуется написанием ее наименования с фактическими параметрами, подставленными

вместо формальных параметров. Указанная конструкция называется оператором процедуры. Количество формальных и фактических параметров в описании процедуры и в операторе процедуры должно быть одинаковым. Обращение записывается, например, так:

$$p(A, B, C, D, E);$$

где  $A, B, C, D, E$  — фактические параметры, соответствующие формальным  $a, b, c, d, e$ .

В отличие от формальных, фактические параметры являются реальными объектами программы. Их значения либо являются исходными для выполнения операторов тела процедуры в данном конкретном обращении, либо должны быть получены при выполнении указанной процедуры.

Каждый формальный параметр представляет собой идентификатор. В списке формальных параметров все идентификаторы должны быть отличны друг от друга и от идентификатора процедуры. Фактическим параметром в зависимости от спецификации может быть либо выражение (в частности, число или переменная), либо строка, либо идентификатор, являющийся в блоке, содержащем обращение к процедуре, идентификатором массива, переключателя или процедуры. Список фактических параметров состоит из выписанных друг за другом фактических параметров, разделенных запятыми.

Например, алгоритм вычисления одного из корней квадратного уравнения  $ax^2 + bx + c = 0$  можно оформить в виде следующего описания процедуры:

```
procedure k(a, b, c, x); real a, b, c, x;
  x := (-b + sqrt(b ↑ 2 - 4 × a × c))/(2 × a);
```

Обращение  $k(1.54, s + t, A - 10, y)$ ; вызовет вычисление корня квадратного уравнения  $1.54y^2 + (s + t)y + (A - 10) = 0$  в следующей последовательности:

- отыскивается описание процедуры  $k$ ;
- между фактическими и формальными параметрами устанавливается взаимно однозначное соответствие путем пересчета их слева направо, и в теле процедуры формальные параметры заменяются на фактические; тело процедуры принимает вид

$$y := (-(s + t) + \text{sqrt}((s + t) \uparrow 2 - 4 \times 1.54 \times (A - 10)))/(2 \times 1.54);$$

оно называется модифицированным телом процедуры;

- выполняется оператор — модифицированное тело процедуры; после него выполняется оператор, стоящий в программе после обращения к процедуре, т. е. сохраняется естественная последовательность исполнения операторов.

Если уравнение имеет вид  $2x^2 + ax + b = 0$ , то вычисление его корня, названного  $z$ , осуществляется обращением  $k(2, a, b, z)$ ; т. е. наименования формальных и фактических параметров могут, в частности, совпадать, однако по смыслу это различные величины.

Процедуру для вычисления обоих корней квадратного уравнения можно оформить как процедуру с телом в виде составного оператора:

```
procedure k2(a, b, c, x1, x2); real a, b, c, x1, x2;
begin real r; r := sqrt(b ↑ 2 - 4 × a × c);
x1 := (-b + r)/(2 × a); x2 := (-b + r)/(2 × a)
end;
```

Оператор процедуры определен только в блоке, содержащем описание данной процедуры. В соответствии с этим идентификаторы величин, встречающихся в списке фактических параметров должны быть описаны либо в этом же блоке, либо в одном из объемлющих данный внешний блоке.

3. Спецификации. Каждый формальный параметр должен иметь спецификацию, которая начинается одним из следующих описателей или их сочетаний:

Boolean	Boolean array	real array
integer	integer array	procedure
real	Boolean procedure	string
array	integer procedure	switch
label	real procedure	

После этих описаний следует список идентификаторов из числа формальных параметров. Каждый формальный параметр может быть включен только в одну из спецификаций.

Спецификация состоит в указании типа и класса объектов, которые могут быть использованы в качестве фактических параметров, заменяющих данные формальные. Как следует из определения, спецификация лишь сообщает информацию транслятору, а не вызывает никаких действий. В частности, в спецификации массива не указываются границы. Примеры использования спецификаций приведены выше.

В некоторых входных языках спецификации могут и отсутствовать. Однако это очень часто приводит к совершенно бесполезной общности и к большой дополнительной работе для транслятора. Например, два формальных параметра  $a$  и  $b$  в теле процедуры могут составлять комбинацию  $a \uparrow b$ . Если транслятору неизвестен тип переменных, то эта операция будет выполняться по формуле  $\exp(a \times \ln(b))$ , в то время как в случае целых переменных она могла бы осуществиться намного экономнее ( $a \times a \times \dots \times a$ ,  $b$  раз).

В зависимости от спецификации формальных параметров фактические параметры процедур должны удовлетворять условиям:

— если формальный параметр специфицируется как переменная целого, действительного или логического типов, то в качестве фактического параметра можно использовать любое арифметическое (соответственно логическое) выражение, за исключением случая, когда такое выражение может появиться в левой части оператора присваивания;

— если формальный параметр специфицируется как массив, то в качестве фактического параметра может стоять только идентификатор массива, тип которого указан в спецификации;

— если формальный параметр специфицируется как метка, то такому формальному параметру должно соответствовать в качестве фактического параметра именуемое выражение; для формальных параметров, являющихся переключателями, строками или процедурами, роль фактического параметра может исполнять лишь идентификатор соответствующего класса.

При несовпадении типов целого и действительного для формального и фактического параметров преобразование типа подразумевается в каждом месте нахождения формального параметра в теле процедуры.

**4. Комментарий в списке параметров.** Некоторую информацию о формальных или фактических параметрах для читателя программы может дать своеобразный комментарий, используемый вместо обычного разделителя , (запятая) в списке параметров. Запятую можно заменить следующей комбинацией: закрывающая скобка, произвольная строка букв, двоеточие, открывающая скобка, т. е.

)СТРОКА БУКВ:(

Пример. Пусть требуется найти наибольший по абсолютной величине элемент в заданной строке двумерного массива  $a$  размером  $n \times n$ .

Входными будут величины, обозначаемые следующими идентификаторами:  $a$  — наименование массива,  $i$  — номер строки,  $n$  — размер массива. Выходных величин две:  $aij$  — величина наибольшего элемента и  $j$  — номер столбца, в котором этот элемент расположён.

Заголовок процедуры можно записать следующим образом:

**procedure** *max*( $a, i, n, aij, j$ ); **array**  $a$ ; **integer**  $i, n, j$ ; **real**  $aij$ ;

Для удобства чтения, используя комментарий в списке параметров, этот же заголовок можно оформить более понятно:

**procedure** *max*( $a$ ) *строка: (i) размер: (n)*

*элемент: (aij) столбец: (j)*;

**array**  $a$ ; **integer**  $i, n, j$ ; **real**  $aij$ ;

Такая запись списка параметров может быть использована также в обращении к процедуре для пояснения смысла вычисляемых величин. Нельзя применять комментарий к первому параметру и, следовательно, к описанию или обращению к процедуре с одним параметром. Использование такой формы записи списка параметров процедуры, как и любого комментария, не сказывается на работе программы, и она применяется исключительно с целью сделать программу более понятной при ее чтении.

Тело процедуры имеет вид

**begin** **integer**  $k$ ;  $aij := abs(a[i, 1])$ ;  $j := 1$ ;

**for**  $k := 2$  **step** 1 **until**  $n$  **do**

**if**  $abs(a[i, k]) > aij$  **then**

**begin**  $aij := abs(a[i, k])$ ;  $j := k$  **end**

**end**;

В этой программе последовательно сравниваются все элементы рассматриваемой строки, и переменная  $aij$  в итоге получает значение наибольшего по абсолютной величине элемента  $a[i, k]$  массива.

**5. Список значений.** Некоторые или даже все формальные параметры процедуры могут быть включены в список значений, который состоит из основного символа *value* и следующего за ним списка идентификаторов. Список значений размещается в заголовке процедуры после списка формальных параметров перед спецификациями и заканчивается разделителем ; Способ задания параметров при обращении к процедуре различен для формальных параметров, включенных в список значений, и параметров, не вошедших в него.

Формальные параметры, которые перечислены в списке значений (т. е. после символа *value*), считаются заданными по значению, а все остальные — по наименованию. Например, в заголовке процедуры

*procedure L(x, y, k); value x, y; real x, y; integer k;*

в список значений включены действительные переменные *x, y*.

Формальные параметры, заданные по наименованию, в теле процедуры заменяются на соответствующие фактические параметры. Формальным параметрам, заданным по значению, перед выполнением тела процедуры присваиваются значения соответствующих фактических параметров. Эти присваивания выполняются в блоке, объемлющем тело процедуры.

**Пример.** Пусть требуется вычислить две суммы:

$$S1 = \sum_{x=1}^{10} x^2, \quad S2 = \sum_{z=3}^{12} \sqrt{a_z + b_z + c}.$$

Процедура, с помощью которой можно провести суммирование в наиболее общем случае, т. е. вычислить сумму  $S = \sum_{i=j}^k f_i$ , где предполагается, что  $f_i$  — выражение, зависящее от индекса суммирования  $i$ , имеет вид

*procedure summa (S, i, j, k, f); real S, f; integer i, j, k;*  
*begin S := 0; for i := j step 1 until k do S := S + f end;*

Суммы  $S1, S2$  могут быть вычислены в результате двух обращений к процедуре *summa*:

*summa (S1, x, 1, 10, x ↑ 2);*  
*summa (S2, z, 3, 12, sqrt(a[z] + b[z] + c));*

вызывающих выполнение двух составных операторов, представляющих собой модифицированные тела процедуры:

*begin S1 := 0; for x := 1 step 1 until 10 do S1 := S1 + x ↑ 2 end;*  
*begin S2 := 0; for z := 3 step 1 until 12 do*  
*S2 := S2 + sqrt(a[z] + b[z] + c) end;*

Таким образом, обращение к процедуре в этом примере передает в тело процедуры правило для вычисления значений переменных, а не сами значения. Это и есть обращение к процедуре по наименованию. В результате операторы однотипных вычислений по одной процедуре, т. е. соответствующие команды

на языке машины для тела процедуры, располагаются в памяти машины только в одном месте.

Предположим далее, что необходимо вычислить еще одну сумму:

$$S3 = \sum_{l=n+1}^{2n(n+1)} l^3.$$

Очевидно, можно написать обращение:

*summa* (*S3*, *l*, *n + 1*,  $2 \times n \times (n + 1)$ ,  $l \uparrow 3$ );

что обеспечивает выполнение составного оператора

```
begin S3 := 0; for l := n + 1 step 1 until 2 × n × (n + 1) do
  S3 := S3 + l ↑ 3 end;
```

Выражение  $2 \times n \times (n + 1)$  при таком обращении к процедуре *summa* будет вычисляться заново после каждого шага суммирования, так как текущее значение параметра цикла сравнивается с концом, задаваемым указанным выражением. Однако это выражение не зависит от текущего значения параметра *l* и потому в повторном его вычислении нет необходимости.

Объявление формального параметра в описании процедуры значением позволяет фактический параметр, соответствующий этому формальному, вычислять только один раз до выполнения оператора процедуры. Перепишем описание процедуры в виде

```
procedure sum(S, i, j, k, f); value j, k; integer i, j, k; real S, f;
begin S := 0; for i := j step 1 until k do S := S + f end;
```

При таком описании процедуры значения выражений, соответствующих формальным параметрам *j* и *k*, будут вычислены один раз до входа в тело процедуры, исходя из заданных значений фактических параметров. Это произойдет как бы в объемлющем тело процедуры дополнительном блоке, из которого в тело процедуры будут переданы только значения фактических параметров, соответствующих формальным параметрам *j* и *k*.

Таким образом, вместо прежнего модифицированного тела процедуры в случае вычисления суммы *S3* при обращении к процедуре *sum* будем иметь

```
j := n + 1; k := 2 × n × (n + 1);
begin S3 := 0; for i := j step 1 until k do S3 := S3 + i ↑ 3 end;
```

где формальные параметры *j*, *k*, включенные в список значений, при модификации тела сохраняют свои обозначения, т. е. в модифицированном теле процедуры эти формальные параметры не заменяются на фактические.

Обращение к параметрам процедуры по значению делается исключительно для повышения эффективности программы, а потому обращаться по значению следует ко всем входным параметрам процедуры, которые действительно принимают лишь одно значение, и это значение целесообразно вычислить предварительно. Бессмысленно говорить об обращении по значению к выходным параметрам, которые получают значения только в результате вычислений в теле процедуры.

**6. Классификация объектов в теле процедуры.** При описании процедуры в ее теле могут использоваться объекты, которые несут различную смысловую нагрузку, однако часто обозначаются совпадающими идентификаторами. Точная классификация объектов в теле процедуры позволяет избежать ошибок при работе с процедурами.

В теле процедуры различают локализованные, формальные и глобальные (прочие) объекты.

Локализованные объекты — это идентификаторы и метки, описанные в теле процедуры или помечающие операторы тела процедуры. К локализованным относятся величины, описанные в теле процедуры; это, как правило, идентификаторы для обозначения промежуточных результатов, неоднократно используемых в процедуре.

Формальными объектами являются объекты тела процедуры, которые обозначаются теми идентификаторами формальных параметров процедуры, которые не используются для обозначения локализованных величин. К формальным объектам, в частности, относятся величины, значения которых являются исходными данными, и, следовательно, варьируются при каждом обращении к процедуре, а также величины, значения которых являются результатами выполнения данной процедуры.

К глобальным относятся все объекты тела процедуры, которые обозначены идентификаторами, не совпадающими с идентификаторами локализованных и формальных объектов. Это величины, которые используются не только в теле процедуры, но и в операторах программы, содержащей данную процедуру; их описание находится в объемлющем процедуру блоке.

Пусть задан следующий фрагмент программы:

```
begin real a, b, c, p; integer i, k;
  ввод (a, b, c); p := 3.14; i := 10;
  begin procedure F(x, y); real x, y;
    begin real y, k;
      y := p/(x + a × k); k := (i + 1) × b + x × y
    end
  end
end;
```

В теле процедуры содержатся идентификаторы  $a, b, p, i, k, x, y$ , которые классифицируются следующим образом:

$k, y$  — описаны в самом теле — это локализованные объекты; при этом не имеет значения, что  $y$  является формальным параметром, а  $k$  описан во внешнем блоке;

$x$  — формальный объект, так как является формальным параметром процедуры и не описан как локализованный;

$a, b, p, i$  — глобальные объекты, так как описаны во внешнем блоке.

Переменная  $c$  не имеет отношения к процедуре и, по определению, является локализованной в блоке, в котором описана.

При модификации тела процедуры для выполнения оператора процедуры транслятор руководствуется следующими правилами:

— формальные объекты, не включенные в список значений, заменяются фактическими параметрами; объекты, подставленные на место формальных, понимаются как объекты, описанные в блоке, где расположен оператор процедуры, т. е. где находится обращение к процедуре;

— наименование подставленного объекта может совпадать с наименованием локализованного объекта тела процедуры; несмотря на это, при трансляции эти объекты считаются различными;

— глобальные объекты понимаются как объекты, описанные в блоке, где расположено описание процедуры; совпадение наименования глобального объекта с наименованием объекта, действующего там, где расположен оператор процедуры, не стирает различия между этими двумя объектами, если они действительно различные.

После этих преобразований модифицированное тело процедуры выполняется так, как если бы оно находилось на месте обращения к процедуре.

В соответствии с этими правилами в предыдущем примере при обращении к процедуре  $F$  формальный параметр  $x$  будет при модификации тела процедуры заменен фактическим. Остальные переменные тела процедуры модификацией не затрагиваются.

### Упражнения

1. Написать процедуру определения наименьшего по абсолютной величине элемента одномерного массива произвольного размера и использовать ее для вычисления минимального элемента среди найденных в случае  $k$  различных одномерных массивов.

2. Написать процедуру определения наибольшего из элементов прямоугольной матрицы.

3. Оформить в виде процедуры упражнение 1 из § 14.

4. Составить процедуру вычисления среднего квадратического для  $k$  значений элементов массива  $a$  по формуле  $m = \sqrt{\sum_{i=1}^k a_i^2/k}$ , и с помощью этой

процедуры вычислить отношение двух средних квадратических для элементов массивов  $b[j:n]$  и  $c[l:n]$ .

5. Составить процедуру вычисления функции Бесселя первого рода целого порядка  $n$ :

$$I_n(x) = \frac{(x/2)^n}{n!} \left[ 1 - \frac{(x/2)^2}{1 \cdot (n+1)} - \frac{(x/2)^4}{1 \cdot 2 \cdot (n+1)(n+2)} - \frac{(x/2)^6}{1 \cdot 2 \cdot 3 \cdot (n+1)(n+2)(n+3)} + \dots \right].$$

Вычисления членов ряда прекратить после того, как очередной вычисленный член станет меньше суммы предыдущих членов, умноженной на  $10^{-6}$ .

## § 18. Процедура-функция

Частным случаем процедуры является процедура-функция. Процедурой-функцией, как правило, оформляется алгоритм, в результате выполнения которого определяется значение только одной величины независимо от числа входных параметров. Оформление алгоритма процедурой-функцией существенно упрощает обращение к процедуре.

Процедура является *процедурой-функцией*, если перед основным символом **procedure** в описании стоит наименование типа (**integer**, **real**, **Boolean**), а значение присваивается идентификатору процедуры внутри ее тела. Например, алгоритм вычисления одного из корней квадратного уравнения из § 17, п. 2 может быть оформлен в виде процедуры-функции:

```
real procedure k(a, b, c); real a, b, c;  
k := (-b + sqrt(b ↑ 2 - 4 × a × c))/(2 × a);
```

Обращение к процедуре-функции, в отличие от процедуры, осуществляется не посредством оператора процедуры, а в виде указателя функции. Указатель функции по внешнему виду совпадает с оператором процедуры, т. е. состоит из идентификатора и списка фактических параметров, заключенного в круглые скобки, но отличается способом использования в программе. Указатель функции включается в арифметическое или логическое выражение в качестве операнда, а оператор процедуры размещается в последовательности операторов программы как самостоятельный оператор. Так, обращение к только что написанной процедуре-функции в случае вычисления корня уравнения  $2x^2 + (h + c)x + 3 = 0$  осуществляется указателем функции  $k(2, h + c, 3)$ , который можно использовать в выражениях как переменную, например:  $y := z + k(2, h + c, 3)$ ;

Таким образом, процедура-функция — это как бы обычная переменная, только описывается она в виде процедуры. Отметим, что обращение к стандартным функциям выполняется также по указателю функций, только стандартные функции не описываются в программе.

Наименование процедуры-функции не локализовано в теле процедуры. Идентификатор ее не должен описываться в теле процедуры и не должен употребляться нигде в теле процедуры, кроме левой части оператора присваивания.

Составим процедуру суммирования в общем виде как процедуру-функцию для суммы  $S = \sum_{i=j}^k f_i$ , где  $f_i$  — элемент одномерного массива  $f$ . Получим

```
real procedure S(i, j, k, f); integer i, j, k; real f;  
begin real S1; S1 := 0; for i := j step 1 until k do  
S1 := S1 + f[i]; S := S1  
end;
```

Использование промежуточной переменной  $S1$  связано с ограничением, накладываемым на использование идентификатора процедуры-функции  $S$  только в левой части оператора присваивания. После описания процедуры-функции  $S$  ее идентификатор может использоваться в выражениях как обычная переменная. Например, оператор присваивания

```
z := exp(S(i, 1, 10, 1/i ↑ 2) + sqrt(S(i, 2, 7, x[i] × y[i])));
```

соответствует математической формуле

$$z = \exp \left[ \sum_{i=1}^{10} \frac{1}{i^2} + \sqrt{\sum_{i=2}^7 x_i y_i} \right].$$

Пример. При вычислении интеграла

$$I(t) = \frac{1}{t^2} \int_a^b \frac{e^{-x} \cos \frac{x}{4} dx}{x^3 (e^{t/x} - 1)}$$

подынтегральную функцию можно представить в виде процедуры-функции, после чего программа из § 13 для вычисления интеграла  $I(t)$  методом Симпсона переписывается следующим образом:

```

begin real t, t0, th, tk, a, b, x, h, S4, S2; integer n;
  real procedure f(x); real x;
    f := exp(-x) × cos(x/4) / (x ↑ 3 × (exp(t/x) - 1));
  ввод (t0, th, tk, a, b, n); h := (b - a) / n;
  for t := t0 step th until tk do begin S4 := S2 := 0;
    for x := a + h step 2 × h until b - 3 × h do
      begin S4 := S4 + f(x); S2 := S2 + f(x + h) end;
    I := 1/t ↑ 2 × h/3 × (f(a) + f(b) + 4 × f(b - h) + 4 × S4 + 2 × S2);
    вывод (I, t) end
end

```

Нетрудно убедиться, что только что написанная программа компактнее и нагляднее предыдущего варианта.

В описании процедуры-функции  $f$  входит формальный параметр  $x$  и глобальный параметр  $t$ , как не являющийся ни локализованным, ни формальным. Через  $x$  обозначен также фактический параметр процедуры, который описан в начале программы.

#### Упражнения

1. Описать как процедуру-функцию алгоритм вычисления дисперсии эле-

ментов одномерного массива  $a$  по формуле  $D = \frac{1}{n} \sum_{i=1}^n a_i^2 - M^2$ , где  $M =$

$= \frac{1}{n} \sum_{i=1}^n a_i$  — математическое ожидание. Модифицировать процедуру так,

чтобы вычисленное значение  $M$  также сохранялось.

2. Составить процедуру-функцию для вычисления периметра произвольного многоугольника с заданными координатами вершин  $x_i, y_i$ .

3. Описать функцию, вычисляющую  $n!!$ .

4. Составить функцию с формальными параметрами  $x$  и  $y$ , принимающую значение true, если треугольник со сторонами  $x_1, x_2, x_3$  подобен треугольнику со сторонами  $y_1, y_2, y_3$  и false в противном случае. В теле процедуры-функции описать процедуру, располагающую значения  $x_i$  и  $y_i$  ( $i = 1, 2, 3$ ) в порядке их возрастания.

## § 19. Процедура без параметров

Описание процедуры с некоторым количеством формальных параметров позволяет к ней обращаться с самыми разнообразными фактическими параметрами. Однако возможна ситуация, когда обращение к процедуре происхо-

дит всегда с одними и теми же фактическими параметрами, которые только меняют свои числовые значения. В таких случаях в описании процедуры могут отсутствовать параметры.

Заголовок процедуры без параметров имеет вид *procedure p*; где *p* — идентификатор, а обращение состоит лишь из указания идентификатора процедуры *p*; , т.е. отсутствуют также фактические параметры. Аналогично процедуре без параметров может быть и процедура-функция без параметров.

В теле процедуры без параметров встречаются лишь локализованные и глобальные объекты, а функции формальных параметров при передаче в тело процедуры значений исходных величин и получении результата выполняют идентификаторы глобальных величин. Поэтому перед обращением к процедуре без параметров глобальным идентификаторам, которые обозначают исходные величины, должны быть присвоены требуемые значения.

Результат выполнения процедуры без параметров присваивается в ее теле соответствующим идентификаторам глобальных величин, которые могут использоваться после обращения к процедуре в последующих операторах. В случае процедуры-функции без параметров указатель ее представляет собой обычный идентификатор и ничем не выделяется в выражении среди идентификаторов простых переменных.

Пример. В ЭВМ вводятся тройки положительных чисел *a*, *b*, *c*, и требуется определить, можно ли из отрезков с длинами соответственно *a*, *b*, *c* составить треугольник. Решение этой задачи, очевидно, будет заключаться в проверке следующего условия: сумма двух любых сторон треугольника должна быть больше третьей стороны. Если однажды написать процедуру проверки этого условия для тройки чисел, то обращение будет иметь всегда один и тот же вид. Необходимо лишь перед каждым обращением к процедуре обеспечить ввод новых значений.

Программу для решения этой задачи в случае 100 троек чисел можно составить в виде

```
begin real a, b, c; Boolean p; integer i, N;
  procedure ТК; p := a < b + c ∧ b < c + a ∧ c < a + b;
  N := 100; for i := 1 step 1 until N do
    begin ввод (a, b, c); ТК; вывод (p) end
end
```

### Упражнения

1. Составить программу ввода матриц третьего порядка

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

и вычисления детерминанта матриц. При вычислении детерминанта использовать процедуру-функцию без параметров.

2. Написать программу ввода квадратных матриц порядка *M* и печатания соответствующих транспонированных матриц. Транспонирование выполнить с помощью процедуры без параметров.

## § 20. Рекурсии в процедурах

В теле процедуры допускается явное обращение к этой же самой процедуре, а также разрешается использовать идентификатор процедуры в качестве фактического параметра при обращении к этой же процедуре. Такое допущение приводит к рекурсиям в процедурах.

*Рекурсивными* процедурами называются такие процедуры, в теле которых есть обращение к самим себе непосредственно или с помощью другой процедуры.

Классическим примером рекурсивной процедуры служит вычисление факториала числа  $n$  ( $n$  — целое). Для вычисления  $n!$  можно написать очень короткую процедуру в виде рекурсивной процедуры-функции:

```
integer procedure f(n); value n; integer n;
  f := if n = 1 then 1 else n × f(n - 1);
```

В теле этой процедуры происходит обращение к той же процедуре, если  $n \neq 1$ . Если  $n = 2$ , то факториал  $f(2)$  есть  $2 \times f(1)$ , т. е.  $2 \times 1! = 2!$ . Чтобы вычислить  $n!$ , к процедуре-функции  $f(n)$  необходимо обратиться  $n$  раз, так как вычисления начинаются с конца, т. е. с умножения на число  $n$  значения  $f(n-1)$ , которое в свою очередь есть произведение аргумента  $n-1$  на функцию  $f((n-1)-1)$ , и т. д.

Очевидно, это не очень выгодный путь вычисления, и поэтому алгоритм для вычисления  $n!$  разумнее оформить циклом

```
integer procedure f(n); value n; integer n;
  begin integer i, F; F := 1;
  for i := 1 step 1 until n do F := F × i; f := F
end;
```

Эта процедура выполняет итеративное вычисление факториала в противоположность рекурсивному, описанному выше.

В таком варианте рабочая программа получится короче, так как вычисление  $n!$  выполняется с начала, т. е. начиная с  $i = 1$ , для каждого  $i$  делается лишь одно умножение, поскольку произведение предыдущих  $i-1$  сомножителей сохраняется.

На основании приведенного примера вычисления  $n!$  нельзя делать общий вывод о невыгодности рекурсивных процедур. Рекурсивные процедуры обладают многими достоинствами и являются мощным средством при создании оптимальных программ в таких задачах, как перевод с языка на язык, в том числе при трансляции, в доказательствах теорем и др.

Смысл использования в программе рекурсивных процедур заключается в том, что в процессе выполнения программы оказывается возможным несколько раз войти в один и тот же блок (в данном случае — в тело процедуры), ни разу не выходя из него. Говоря о повторном входе в один и тот же блок без выхода из него, имеется в виду только повторное описание величин и действий над ними. Значения же локализованных величин при каждом рекурсивном входе оказываются иными.

Уровнем (или глубиной) рекурсий называется число рекурсивных входов. В приведенной процедуре  $f(n)$  глубина рекурсий совпадает с  $n$ .

Использование в операторе процедуры идентификатора этой же процедуры в качестве фактического параметра, если только соответствующий формальный параметр не включен в список значений, вызовет рекурсивное обращение к процедуре. Например, обращение вида  $f(f(x))$ ; где  $f$  — идентификатор процедуры и формальный параметр не включен в список значений, рекурсивно.

Если же формальные параметры входят и список значений, и хотя соответствующие фактические параметры содержат идентификатор этой же процедуры, такое обращение не является рекурсивным. Например, если заголовок процедуры-функции имеет вид

**real procedure**  $F(x, y)$ ; **value**  $x, y$ ; **real**  $x, y$ ;

то обращение

$z := F(F(a, b), F(u, F(a, v)))$ ;

не рекурсивно. Если бы список значений не содержал хотя бы одного из формальных параметров ( $x$  или  $y$ ), то написанное обращение было бы рекурсивным, так как значения всех переменных, включенных в список значений, вычисляются до входа в модифицированное тело процедуры, т. е. в теле процедуры участвуют лишь формальные параметры со значениями, вычисленными ранее (пусть с помощью этой же процедуры-функции).

Рассмотрим на примере вычисления двойной суммы

$$S2 = \sum_{j=1}^{10} \sum_{k=2}^{12} (j^2 + k^2)^{1/2}$$

возможность использования рекурсивного обращения.

Процедура-функция для вычисления суммы  $\sum_{i=j}^k f_i$  незначительно отличается от процедуры, приведенной в § 17, п. 5:

**real procedure**  $s(i, j, k, f)$ ; **value**  $j, k$ ;  
**integer**  $i, j, k$ ; **real**  $f$ ;  
**begin** **real**  $su$ ;  $su := 0$ ;  
**for**  $i := j$  **step** 1 **until**  $k$  **do**  $su := su + f$ ;  $s := su$   
**end**;

Двойная сумма вычисляется в результате обращения

$S2 := s(j, 1, 10, s(k, 2, 12, \text{sqrt}(j \uparrow 2 + k \uparrow 2)))$ ;

которое является рекурсивным.

### Упражнения

1. Составить рекурсивную процедуру-функцию для вычисления квадратного корня  $x$  величины  $a$ , т. е.  $x = \sqrt{a}$  по итеративной формуле  $x_{n+1} = 0.5(x_n + a/x_n)$ . В качестве начального приближения использовать значение  $x_0 = 0.5(1 + a)$ . В результате обеспечить точность  $10^{-5}$ .

2. Написать рекурсивную процедуру для нахождения наибольшего общего делителя двух положительных чисел. Составить иерекурсивный вариант этой же процедуры.

3. Описать в виде рекурсивной процедуры алгоритм  $m$ -кратного дифференцирования полинома степени  $n$ : 
$$L_n(x) = \sum_{i=0}^n a_i x^{n-i}$$

## § 21. Побочный эффект при вычислении функций

Изменение при обращении к процедуре-функции значений переменных, не локализованных в теле процедуры, определяющей функцию, или же присваивание значения формальному параметру, включенному в список значений, называется побочным эффектом. Основным результатом при вычислении процедуры-функции, естественно, является значение этой функции. Например, в процедуре-функции

```
real procedure y(x); real x; y := x := x + 1;
```

не только вычисляется значение  $y$ , но и изменяется значение аргумента. Поэтому значение некоторой переменной  $s$  будет зависеть от того, выполнялся ли оператор  $s := x + y(x)$ ; или оператор  $s := y(x) + x$ ;

Аналогичная ситуация складывается и после выполнения операторов  $R := r + a$ ; и  $R := a + r$ ; где  $r$  — это указатель функции без параметров:

```
real procedure r; begin r := sqrt(x ↑ 2 + y ↑ 2); a := x + y end;
```

Если бы  $r$  был идентификатором простой переменной, операция сложения сохраняла бы коммутативность.

Разумеется, не во всех описаниях процедур-функций наличие побочного эффекта проявляется столь прозрачно, как в приведенных примерах. В общем случае значение выражения, содержащего указатель функции с побочным эффектом, можно правильно определить при знании алгоритмов трансляции. Отметим также, что изменяться могут и значения некоторых переменных, которые появляются в теле процедуры в результате замены по наименованию формальных параметров на фактические.

Если в силу побочных эффектов указателей функций различные порядки исполнения приводят к различным результатам, то программа не определена. Следовательно, во избежание недоразумений, при составлении программы очень важно точно учитывать ту последовательность вычисления значений выражений, которая предусмотрена правилами языка. Правила эти сводятся к следующим.

Значения первичных выражений, входящих в некоторое более сложное выражение, вычисляются слева направо. Например, при вычислении выражения  $x - y \times z \uparrow f(x)$  ни одна арифметическая операция не может быть выполнена до вычисления значения функции  $f(x)$ . Однако первоначальные значения переменных  $x$ ,  $y$ ,  $z$  фиксируются слева направо и используются они, а не измененные вследствие возможного побочного эффекта при вычислении  $f(x)$ .

Значения индексных выражений у переменной с индексами вычисляются одно за другим слева направо. Например, элемент массива  $m[i, j(k), l]$  может и не иметь равные значения первого и третьего индексов, так как первый (самый левый) индекс принимает значение переменной  $i$  до вычисления функции  $j(k)$ , а третий — значение, которое будет иметь переменная  $l$  после вычисления функции.

Значения границ индексов также вычисляются слева направо в той последовательности, в которой они приведены в описании массива.

Значения индексных выражений в списке левой части оператора присваивания находятся тоже слева направо. Например, в операторе  $a[i, j(k)] := b[l] := f(x)$ ; последовательность вычислений такова: первый индекс элемента массива  $a$ , функция  $j(k)$ , индекс элемента массива  $b$ , функция  $f(x)$ .

При выполнении арифметических операций всегда используются значения входящих в выражения операндов. Например, выражение  $f(x) \uparrow 2$  не равносильно выражению  $f(x) \times f(x)$ , так как в первом случае один раз вычисляется значение функции  $f(x)$  и перемножаются два одинаковых множителя, в то время как во втором случае обращение к процедуре  $f(x)$  делается дважды и перемножаются полученные значения, которые могут быть различными.

При обращении к процедуре формальным параметрам, включенным в список значений, значения присваиваются в том порядке, в каком эти параметры встречаются в списке формальных параметров слева направо. Тем самым определяется последовательность вычисления значений соответствующих фактических параметров. Например, имея описание процедуры

```
procedure F(x, y, z); value x, y; real x, y, z;
  <тело процедуры>;
```

после обращения  $F(a, b, c)$ ; соблюдается последовательность операторов:  $x := a$ ;  $y := b$ ; <модифицированное тело>; т.е. значение  $x$  вычисляется раньше, чем значение  $y$ .

### Упражнения

1. Одинаковы ли значения переменных  $z_1, z_2, z_3, z_4$ , получаемые после выполнения операторов:

```
z1 := S(l, 1, 8, x[l]) + y[l];   z3 := T(l, 1, 8, x[l]) + y[l];
z2 := y[l] + S(l, 1, 8, x[l]);   z4 := y[l] + T(l, 1, 8, x[l]);
```

где процедура функция  $S(i, j, k, f)$  описана в § 18, а описание  $T(i, j, k, f)$  имеет вид

```
real procedure T(i, j, k, f); value j, k; real f; integer i, j, k;
  begin real s; s := 0; i := j; M: s := s + f; i := i + 1;
  if i ≤ k then go to M; T := s
  end;
```

2. Определить значение переменной  $y$  после выполнения последовательности операторов:  $x := 2$ ;  $y := f(x) + f(x) + x$ ; если описание  $f$  имеет вид:

- real procedure  $f(z)$ ; real  $z$ ; begin  $z := z \uparrow 2$ ;  $f := z \uparrow 2/4$  end;
- real procedure  $f(z)$ ; real  $z$ ; begin  $f := z \uparrow 2/4$ ;  $z := z \uparrow 2$  end;
- real procedure  $f(z)$ ; value  $z$ ; real  $z$ ; begin  $f := z \uparrow 2/4$ ;  $z := z \uparrow 2$  end;
- real procedure  $f(z)$ ; value  $z$ ; real  $z$ ; begin  $z := z \uparrow 2$ ;  $f := z \uparrow 2/4$  end;
- real procedure  $f(z)$ ; value  $z$ ; real  $z$ ;  $f := z \uparrow 4/4$ ;
- real procedure  $f(z)$ ; real  $z$ ;  $f := z \uparrow 4/4$ ;

## § 22. Примеры программ

1. **Нахождение простых чисел.** Одним из методов нахождения простых чисел является метод Эйлера, заключающийся в делении любого нечетного числа  $k$  на все найденные простые числа, меньшие  $\sqrt{k}$ .

Пусть первые 5 простых чисел найдены, тогда проверять следует лишь все последующие нечетные числа. Если нечетное число  $k$  не делится нацело ни на одно из уже найденных простых чисел, меньших  $\sqrt{k}$ , то оно простое.

Программа для нахождения первых 1000 простых чисел будет иметь вид

```
begin integer i, j, k, l, m, p, d; real n; integer array f[1 : 1000];
  k := 9; p := 0; f[1] := 1; f[2] := 2; f[3] := 3; f[4] := 5; f[5] := 7;
  вывод (f[1], f[2], f[3], f[4], f[5]); i := 5;
M: for j := 1, 2, 3 do begin k := k + 2; if j = 3 then go to M;
  n := sqrt(k); m := 5; for l := 4, l + 1 while m < n do
    begin m := f[l]; d := k ÷ m; if abs(m × d - k) < 10 - 5
      then go to M
    end;
  i := i + 1; f[p + i] := k; вывод (k); if i = 10 then
    begin i := 0; p := p + 10; вывод (p); if p = 1000 then go to N
    end
  end go to M;
N: end
```

В программе для сокращения вычислений исключаются все числа  $k$ , которые делятся на 3 (цикл по  $j$ ). Аргумент стандартной функции  $abs$  целый, поэтому в отношении после знака  $<$  справа может стоять любое положительное число, меньшее единицы. Порядковыми номерами при выводе снабжается каждое десятое печатаемое простое число.

2. **Интегрирование системы дифференциальных уравнений первого порядка.** Дана система уравнений

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_n)$$

с начальными условиями:  $y_i(x_0) = y_{i0}$ ,  $i = 1, 2, \dots, n$ .

Интегрирование по методу Рунге — Кутты выполняется с использованнем следующих формул

$$y_i(x + h) = y_i(x) + \frac{h}{6}y'_{i1} + \frac{h}{3}(y'_{i2} + y'_{i3}) + \frac{h}{6}y'_{i4}$$

где

$$y'_{i1} = f_i(x, y_1, \dots, y_n),$$

$$y'_{i2} = f_i\left(x + \frac{h}{2}, y_1 + \frac{h}{2}y'_{i1}, \dots, y_n + \frac{h}{2}y'_{n1}\right),$$

$$y'_{i3} = f_i\left(x + \frac{h}{2}, y_1 + \frac{h}{2}y'_{i2}, \dots, y_n + \frac{h}{2}y'_{n2}\right),$$

$$y'_{i4} = f_i(x + h, y_1 + hy'_{i3}, \dots, y_n + hy'_{n3}).$$

Удобно ввести дополнительные переменные  $a_0 = 0$ ,  $a_1 = a_2 = a_3 = h/2$ ,  $a_4 = a_5 = h$ , после чего вышеприведенные формулы могут быть переписаны так:

$$y_i(x+h) = y_i(x) + \frac{1}{3} \sum_{j=1}^4 a_{j+1} y'_{ij},$$

где

$$\begin{aligned} y'_{ij} &= f_i(x + a_{i-1}, r_{1j-1}, r_{2j-1}, \dots, r_{nj-1}), \\ r_{i,j-1} &= y_i + a_{j-1} y'_{ij-1}, \quad j = 1, 2, 3, 4. \end{aligned}$$

Процедура интегрирования системы методом Рунге—Кутты (с наименованием *RKS*) содержит следующие формальные параметры:

$x$  — начальное значение независимой переменной,

$y[1:n]$  — одномерный массив начальных значений искомых функций,

$h$  — шаг интегрирования,

$f$  — идентификатор процедуры счета правых частей интегрируемых уравнений,

$$f = f(x, r, n, z),$$

где  $z$  — массив значений переменных  $y'_{ij} = z[i]$  для данных значений  $r[i] = r_{i,j-1}$  элементов массива  $r$  переменных и аргумента  $x + a_{j-1}$ ,

$n$  — количество дифференциальных уравнений в системе,

$xx$  — значение независимой переменной в конце интервала интегрирования,

$yy$  — одномерный массив значений искомых функций.

Описание процедуры имеет следующий вид:

```

procedure RKS(x, y, h, f, n, xx, yy);
integer n; real x, h, xx; array y, yy; procedure f;
begin integer i, j; array a[1:5], r, z[1:n];
a[1] := a[2] := a[5] := h/2; a[3] := a[4] := h; xx := x;
for i := 1 step 1 until n do yy[i] := r[i] := y[i];
for j := 1, 2, 3, 4 do
begin f(xx, r, n, z); xx := x + a[j]; for i := 1 step 1 until n do
begin yy[i] := yy[i] + a[j+1] * z[i]/3;
r[i] := y[i] + a[j] * z[i]
end
end
end;

```

Пусть теперь имеется система уравнений

$$\frac{dy_1}{dx} = 2x - 3y_1 - 4y_2,$$

$$\frac{dy_2}{dx} = x + y_1 + y_2,$$

где  $0 \leq x \leq 1$ ,  $y_1(0) = y_2(0) = 0$  и задан шаг интегрирования  $h = 0.05$ .

Тогда программа интегрирования этой системы может быть получена добавлением к описанию процедуры *RKS* следующих операторов:

```

procedure f(x, y, n, z); real x; integer n; array y, z;
begin z[1] := 2 × x − 3 × y[1] − 4 × y[2]; z[2] := x + y[1] + y[2]
end;
real x, xx; integer k; array y, yy[1 : 2];
x := y[1] := y[2] := 0; for k := 1 step 1 until 20 do
  begin RKS(x, y, 0.05, f, 2, xx, yy);
    x := xx; y[1] := yy[1]; y[2] := yy[2]; вывод (k, x, y)
  end

```

Для того чтобы программа была завершенной, необходимо ее заключить в окаймляющие операторные скобки.

### § 1. Элементы языка

**1. Символы.** При записи программ используются символы трех категорий: *буквы, цифры и специальные знаки.*

К первой категории относятся 26 прописных букв латинского алфавита:

A B C D E F G H I J K L M  
N O P Q R S T U V W X Y Z

а также знак денежной единицы, обозначаемый обычно \$ (иногда иначе). Во входных языках для отечественных трансляторов, как правило, разрешается также использовать в качестве букв прописные буквы русского алфавита.

Вторая категория символов — это десять арабских цифр:

0 1 2 3 4 5 6 7 8 9

Специальными знаками языка являются символы:

( пробел),	( открывающая скобка),
= (равно),	) (закрывающая скобка),
+ (плюс),	, (запятая),
- (минус),	. (точка),
* (звездочка),	' (апостроф),
/ (наклонная черта)	& (коммерческое «и»).

Символ пробел служит указателем пропуска позиции при записи программы, а потому не имеет начертания. Для явного указания пробела обычно используется символ —, который принят и в этой книге. В некоторых источниках этот символ имеет другое графическое изображение, например Ъ.

Набор символов каждого конкретного транслятора может отличаться от приведенного. В частности, в записи комментариев и текстовых констант допускается использование всех литер, которые имеют как перфорирующее, так и печатающее устройства.

С помощью перечисленных символов образуются величины, выражения и операторы, которые составляют *элементы языка*. При построении элементов языка используются конструкции, представляющие собой неделимые символы, хотя и образованные из нескольких первичных символов (букв, цифр и специальных знаков). Такие конструкции называются *основными символами*. К ним относятся, в частности, *ключевые* (служебные) слова, используемые при записи операторов.

Ключевыми словами являются следующие основные символы

ASSIGN (присвоить), BACKSPACE (назад), BLOCK DATA (блок данных), CALL (вызвать), COMMON (общий), COMPLEX (комплексный), CONTINUE (продолжать), DATA (данные), DEFINE FILE (определение файла), DIMENSION (размерность), DO (выполнить), END (конец), END FILE (конец файла), ENTRY (вход), EQUIVALENCE (эквивалентность), EXTERNAL (внешний), FIND (найти), FORMAT (формат), FUNCTION (функция), GO TO (перейти к), IF (если), IMPLICIT (неявный), INTEGER (целый), LOGICAL (логический), NAMELIST (список), PAUSE (останов), PRINT (печать), PUNCH (перфорация), READ (читать), REAL (действительный), RETURN (вернуться), REWIND (возврат), STOP (стоп), SUBROUTINE (подпрограмма), TO (к), WRITE (писать).

В различных версиях языка список ключевых слов может быть расширен (или сокращен).

Основными символами являются также следующие конструкции:

- логические константы: .TRUE. (истина), .FALSE. (ложь);
- символ операции возведения в степень: \*\*;
- знаки операций отношения: .LT. (меньше чем), .LE. (не больше), .GT. (больше чем), .GE. (не меньше), .EQ. (равно), .NE. (не равно);
- знаки логических операций: .NOT. (не), .AND. (и), .OR. (или).

К величинам относятся константы, переменные и метки (номера операторов). Константы могут быть арифметическими (числовыми или комплексными), логическими, шестнадцатеричными и текстовыми, переменные — простыми и с индексами. Переменные с индексами объединяются в массивы. Константы и переменные составляют данные.

**2. Числовые константы.** Для изображения числовых констант (чисел) используются цифры, точка, знаки + и —, а также буквы E и D.

Числа различают двух типов: целые и действительные. Целое число — это любая конечная последовательность цифр, перед которой может стоять знак + или —. Отсутствие знака или знак + означают, что число положительное, знак — указывает, что число отрицательное. Примеры целых чисел:

15 +306 —00 —88 0 1938

Правильной дробью называется целое число без знака, перед которым стоит точка, например:

.1 .3578 .0064 .0 .120338

Правильная дробь относится к числам действительного типа.

Числа действительного типа допускают в записи дробную часть и имеют две формы представления: форму F и форму E.

Числом действительного типа в форме F называется либо целое число, за которым поставлена точка, либо правильная дробь, либо последовательность, состоящая из целого числа и правильной дроби; перед всей конструкцией может стоять знак + или —. Отсутствие знака означает, что число положительное. Например, действительными числами в форме F являются

10. 3.14159 +1.5789 0.317  
 .51 —37.000 —.00561 +.317

Таким образом, наличие точки при написании действительного числа в форме F обязательно. Нетрудно видеть, что форма F отличается от обычной математической записи лишь тем, что вместо десятичной запятой пишется точка.

Форма E представления действительного числа — это аналог записи числа с помощью мантиссы, умноженной на степень десяти (десятичный порядок). Десятичный порядок образуется из символа E и следующего за ним целого числа со знаком или без него, содержащего не более двух десятичных разрядов.

Числом действительного типа в форме E называется либо десятичный порядок, либо конструкция, состоящая из мантиссы в виде действительного числа в форме F или целого числа и последующего десятичного порядка. Например, действительными числами в форме E являются:

E-01	+0.27E+04	27E2
E2	+ .27E4	27.E + 2
2700E	27000E-1	27E + 02

Первые две записи являются порядками и обозначают действительное число 0, все последующие записи обозначают одно и то же действительное число, равное 2700.

Различают числа двух видов: стандартной длины и нестандартной длины. Стандартное машинное слово имеет длину в 4 байта (напомним, что 1 байт = 8 битов), такова же стандартная длина числа (как целого, так и действительного типов). Максимальным значением целого числа стандартной длины является  $2^{31} - 1 = 2147483647$

Целое число нестандартной длины содержит 2 байта, модуль его находится в пределах от 0 до  $2^{15} - 1$ . Например, целыми константами нестандартной длины являются приведенные выше целые числа, а целыми константами стандартной длины будут

40001	-56789	+2537648	998877
-------	--------	----------	--------

Действительное число в форме F имеет стандартную длину, если оно содержит не более семи цифр, а в остальных случаях — нестандартную длину. В первом случае число занимает в памяти машины четыре байта, во втором — восемь байт. Максимальное количество цифр в изображении действительного числа в форме F нестандартной длины равно 16. Например, действительные числа в форме F

123456.	0.38745	-357.482
---------	---------	----------

являются числами стандартной длины, а

12345678.	+3.14159624	-1234567.89
-----------	-------------	-------------

нестандартной длины.

Мантисса действительного числа в форме E, являющегося числом стандартной длины, содержит не более семи цифр. Если в изображении действительного числа в форме E символ E заменить на символ D, то образуется

действительное число нестандартной длины (8 байт). Мантисса такого числа может содержать до 16 цифр. Примеры:

12345678.87654321D03 —.01D—13 2700D

Поскольку действительные числа нестандартной длины позволяют получать значения повышенной точности, они называются действительными числами с двойной точностью. Как следует из вышесказанного, число с двойной точностью записывается либо как действительное число с точкой, содержащее от 8 до 16 цифр, либо как число, содержащее десятичный порядок с символом D вместо E.

Область абсолютных значений чисел действительного типа имеет границы  $\sim 5.4 \times 10^{-79}$  и  $\sim 7.2 \times 10^{+75}$ .

**3. Комплексные константы.** *Комплексная* константа записывается в виде пары двух действительных чисел, окруженных скобками и разделенных запятой. Первое действительное число представляет вещественную часть комплексного числа, второе — мнимую. Пара действительных чисел стандартной длины образует комплексную константу стандартной длины (8 байт), а пара действительных чисел с двойной точностью — комплексную константу нестандартной длины (16 байт).

Например, комплексное число  $12-4.3i$  представляется как (12., -4.3) или (12E, -0.43E+1) или (12., -43E-1) или (.12D2, -43D-1) и т. д. Последняя из приведенных записей является записью комплексного числа  $12-4.3i$  с двойной точностью (нестандартной длины).

**4. Логические константы.** *Логические* константы обозначаются символами .TRUE. и .FALSE., которые являются соответственно логической единицей (истина) и логическим нулем (ложь).

**5. Шестнадцатеричные константы.** *Шестнадцатеричная* константа записывается в виде последовательности, образованной из набора шестнадцатеричных цифр, которой предшествует символ Z.

Шестнадцатеричными цифрами являются символы

0 1 2 3 4 5 6 7 8 9 A B C D E F

Максимальное количество цифр, допустимое в шестнадцатеричной константе, зависит от типа и длины переменной, которой присваивается данное значение, т. е. в памяти машины шестнадцатеричная константа представляется как слово длиной в 1, 2, 4, 8 или 16 байт. Один байт памяти содержит две шестнадцатеричные цифры.

Шестнадцатеричные константы могут быть использованы только как величины, присваиваемые переменным в операторе DATA (§ 17) и в операторах явного описания типа (§ 4).

**6. Текстовые константы.** *Текстовая* константа представляет собой последовательность (*строку*) символов. Длина такой последовательности, которая рассматривается как значение некоторой переменной и подвергается обработке аналогично числовой константе, соответствует количеству байт, занятых этой переменной, т. е. 2, 4 или 8 байт. Возможно задание текстовой константы в виде строки символов, заключаемой в апострофы (строчные кавычки) или же строкой символов, которой предшествует конструкция wN (w — целая

константа, указывающая количество символов в текстовой константе). В последнем случае количество символов в строке не должно превышать 255. Каждый символ (в том числе пробел) занимает 1 байт.

Для того чтобы использовать символ ' при задании текстовой константы, ограниченной апострофами, этот символ следует указать в строке дважды. Например, строки символов

```
'_FORTRAN' PROGRAM_
17H_FORTRAN' PROGRAM_
```

задают одну и ту же текстовую константу, которая при выводе на печать имеет вид: `_FORTRAN' PROGRAM_`.

7. **Метки.** Меткой (номером оператора) является индивидуальное название, присвоенное оператору программистом. Метка образуется как последовательность цифр, которая, если ее рассматривать как целое число без знака, может принимать значения от 1 до 99999. Впереди стоящие нули у меток игнорируются, так что записи 5, 05, 00005 являются одной и той же меткой. В записи меток допускаются пробелы. Например, одна и та же метка 25 может быть записана любым из следующих способов:

```
_25_ _ 2_ _ _5  0_2_5
```

Оператор помечается только одной меткой, по которой осуществляется ссылка на него, два оператора не должны иметь одинаковые метки. Порядок снабжения операторов метками произвольный, в частности, метка у оператора может отсутствовать. Таким образом, номер оператора (метка) не является порядковым номером оператора.

8. **Переменные.** Понятие *переменная* употребляется для обозначения величины, обращение к которой производится через ее наименование и которая может принимать различные значения (а не одно-единственное значение как константа).

Для обозначения переменной служит *идентификатор* — последовательность, состоящая не более чем из шести цифр и букв алфавита, причем первым символом должна быть буква. Пробелы в записи идентификаторов не допускаются. Примеры идентификаторов:

```
A YZ X12 APOLLO CRAY
```

Различают два вида переменных: простая переменная и переменная с индексами.

Простая переменная представляет собой величину, принимающую числовые, логические или текстовые значения, и обозначается идентификатором. Так, идентификаторы, приведенные выше, обозначают простые переменные.

Одним идентификатором может быть обозначена группа величин, называемая *массивом*. Каждая отдельная величина представляет собой элемент массива — переменную с индексами. У одного элемента массива индексов может быть несколько. Максимальное число индексов равно семи, в ряде версий языка оно сокращено до трех. Индексы у переменной с индексами составляют список индексов. Количество индексов в списке определяет размерность массива,

Записывается переменная с индексами при помощи идентификатора, после которого в скобках следует список индексов. Например, компоненты вектора  $x(x_1, x_2, x_3)$  составляют одномерный массив: X(1) X(2) X(3), где X — идентификатор массива, а индексы, являющиеся целыми числами 1, 2, 3, можно рассматривать как номера элементов.

Индексы в списке отделяются друг от друга запятыми. Например, элементом двумерного массива будет запись Y(1, 3). Индексом может служить любое арифметическое выражение (§ 2), не содержащее переменных с индексами и называемое в этом случае индексным выражением (в частности, целая константа или целая переменная).

Значение индексного выражения должно быть целочисленным и больше либо равно 1. Если в результате вычислений значение индексного выражения не есть целое, то отбрасывается дробная часть.

В памяти ЭВМ элементы массива располагаются линейно. При этом первым меняется первый индекс. Например, двумерный массив из четырех элементов в памяти располагается по столбцам

$$M(1, 1) \quad M(2, 1) \quad M(1, 2) \quad M(2, 2)$$

Переменные делятся на четыре типа: целые, действительные, комплексные и логические. Каждая переменная может принимать только значения, соответствующие ее типу, однако значения текстовых констант могут принимать переменные любого типа.

Каждому типу переменной соответствуют две длины — стандартная и нестандартная. Действительные и комплексные переменные нестандартной длины принимают значения констант с двойной точностью.

Для указания типа и длины переменных служат операторы описания типа, которые будут рассмотрены в § 4. В то же время для указания типа целых и действительных переменных стандартной длины существует способ автоматического объявления типа, состоящий в том, что для обозначения переменной целого типа стандартной длины применяются символы: I, J, K, L, M, N, употребляемые в качестве первого символа в идентификаторе переменной. Для обозначения переменной действительного типа стандартной длины используются идентификаторы, первый символ в которых есть любая буква, отличная от перечисленных. Например, идентификаторы

$$J \quad KOP \quad MA \quad LIMAX2 \quad I836 \quad N55$$

обозначают переменные целого типа, а

$$E \quad AV \quad VA \quad X701IK \quad COEFF \quad \$55$$

— переменные действительного типа.

Элементы массива должны быть все одного типа. Первая буква в наименовании массива имеет тот же смысл, что и в случае наименований простых переменных.

**9. Функции.** Функция записывается в виде идентификатора, за которым следует заключенный в скобки список аргументов, разделенных запятыми. Объекты программы, задающие конкретные значения аргументов функции, называются фактическими параметрами. Фактическими параметрами могут

быть константы, переменные, выражения, идентификаторы массивов, функций и подпрограмм. Конструкция, состоящая из идентификатора функции и списка фактических параметров, называется обращением к функции или *указателем* функции. Выполнение операций над аргументами функции и получение значения функции осуществляется путем выполнения соответствующего алгоритма при обращении к функции. Значением функции является константа, тип которой зависит от типа функции. Примеры обращений к функции:

EXP(Y) X(T) Z(K, L) F(X)

### Упражнения

1. Среди приведенных ниже записей чисел определить те, которые являются: а) целыми числами; б) действительными числами; в) числами с двойной точностью; г) равными по величине из перечисленных в пп. а), б), в); д) комплексными числами; е) записями чисел, не допускаемыми правилами формата:

152.16E3	152.16D3	+2
-0.891	387.0	2.
+347.D020	100	.1.
-89.1E-02	.891E	0.0000891D
235412	-891.D-03	0.387E3.0
0.235412 <sub>10</sub> +6	891E-3	387E3
(3.4, 0.51E+2)	(7.1D5, 10)	(6.60.E)

2. Среди приведенных ниже конструкций выбрать последовательности символов, являющиеся идентификаторами. У остальных указать ошибки в записи:

ARRAY	ALPHA	.LOG.
string	5ALPHA	α
END	НАТАША	САЛЮТ-7

3. Объяснить имеющиеся ошибки в написании приведенных ниже переменных с индексами и указать, какие из этих записей являются правильными:

FLIGHT[S]	B(ITL)	ARRAY(1, 1, 3, 4)
X(K(6))	A(-INDEX)	N(N)
Y(2.)	AREA(2, K)	N(FALSE.)

4. Указать, какие из приведенных ниже конструкций можно рассматривать как текстовые константы. В остальных найти ошибки:

4N18XYZ	7H1234567	0003HXXX
'1.2/MN'-'*	5NNNNNN	"ABC"
07H-10+135	.FALSE.'	""""

5. Указать, какие из приведенных конструкций нельзя рассматривать в качестве указателей функций и почему:

A137(M,N)	Q(X, -X)	Y(X, Z)
TIP(2A, B, Z)	R((3, 4.))	ART(X, Y)
\$(MONEY)	C(A.GT.B)	AB(A, B)

## § 2. Выражения

Под *выражением* понимается строка символов языка, удовлетворяющая определенным правилам и служащая для вычисления арифметического или логического значений. Выражение образуется из констант, переменных и

указателей функций, называемых операндами, и знаков операций. Выражение в свою очередь может быть использовано как операнд, если оно заключено в скобки. В простейшем случае выражение состоит только из переменной или константы.

Операции подразделяются на арифметические, логические и операции отношения. В зависимости от типа значения выражения определяются арифметические и логические выражения.

**1. Арифметическое выражение.** *Арифметическое* выражение представляет собой имеющую математический смысл конструкцию из констант, переменных, обращений к функциям, знаков арифметических операций и скобок. Арифметические операции определяются символами

+	(сложение).	*	(умножение).
-	(вычитание).	/	(деление).
		**	(возведение в степень).

При построении выражений два знака арифметических операций не должны стоять рядом. Например, вместо  $A*-B$  следует писать  $A*(-B)$  или  $-B*A$ . Примеры арифметических выражений:

A	K + L	$(X+Y)*Z(5)$	$-(X-Y)/Z(1)$
3.14	$A**B$	$X**2/Y**2$	$+ X*Y$

Последовательность выполнения арифметических операций следующая: сначала — возведения в степень, затем — умножения и деления и в последнюю очередь — сложения и вычитания. Порядок выполнения операций может быть изменен скобками по обычным правилам. Если в качестве операнда в арифметическое выражение входит указатель функции, то значение функции вычисляется в первую очередь.

Операции одного ранга выполняются последовательно слева направо. Исключение составляет операция возведения в степень, которая выполняется справа налево. Так, выражение  $L**M**N$  понимается как  $L**(M**N)$ .

Если операнды, входящие в выражение, все одного типа и одной длины, то вычисляемое значение (результат) имеет тот же тип и ту же длину. Если типы операндов смешанные, то всему выражению приписывается более сложный тип из типов операндов: целые значения преобразуются в действительные, а действительные — в комплексные; при этом результат имеет наибольшую длину из длин всех операндов. Так, результат имеет двойную точность, если в выражение входит хотя бы одна величина с двойной точностью.

Например, результат выражения  $I**J$  будет целый стандартной длины, если переменные  $I, J$  обе имеют стандартную длину, результат выражения  $5/2$  будет целый и равный 2, а выражение  $5./2$  — действительного типа со значением 2.5; результат выражения  $X + 3$  действительный, если  $X$  действительного типа, комплексный, если  $X$  — комплексная величина, двойной точности, если двойную точность имеет  $X$ .

Операция возведения в целую степень выполняется как многократное умножение, а возведение в действительную степень — через логарифмы. Следовательно, комплексную величину можно возводить в целую степень, но нельзя в действительную и двойной точности. Показатель степени не может

быть комплексной величиной. Результат операции  $A**R$  не определен также в случае, когда  $A < 0$ , а  $R$  — действительного типа.

**2. Логическое выражение.** Логическое выражение — это по определенным правилам составленная конструкция из логических констант, логических переменных, отношений, символов логических операций и скобок.

Логические переменные — это переменные, тип которых определяется с помощью соответствующего описательного оператора как логический. Переменные логического типа принимают только два значения: значения логических констант, `.TRUE.` и `.FALSE.`

Операции отношения обозначаются следующими комбинациями букв с окаймляющими с обеих сторон точками:

`.GT.` `.GE.` `.EQ.` `.NE.` `.LE.` `.LT.`

Смысл этих операций может быть выражен математическими символами (соответственно):

$>$   $\geq$   $=$   $\neq$   $\leq$   $<$

Каждый символ операции отношения объединяет два операнда, представляющих собой арифметические выражения произвольного типа и длины, за исключением выражений комплексного типа. В версии фортран-63 это ограничение снято; при этом из арифметического выражения комплексного типа для сравнения используется только вещественная часть. Результат сравнения, т. е. результат операции отношения, является логическим значением. Например:

`2..GT.B (A+B)..LE.AB D(1)..LT.1. A*B.NE.B—C`

Логические операции определены для операндов логического типа, обозначаются символами

`.NOT.` `.AND.` `.OR.`

и имеют соответственно следующий смысл: отрицание, логическое умножение (конъюнкция), логическое сложение (дизъюнкция).

С помощью символов `.AND.` и `.OR.` соединяются два операнда, а операция `.NOT.` относится к одному операнду. Например:

`P.AND.Q .NOT.L R.OR.FALSE.`

Здесь  $L$ ,  $P$ ,  $Q$ ,  $R$  — переменные логического типа.

Результатом логической операции `.NOT.` является логическое значение `.TRUE.` в том случае, когда операнд имеет значение `.FALSE.` и `.FALSE.`, когда операнд имеет значение `.TRUE.`

Результат операции `.AND.` есть логическое значение `.TRUE.` если оба операнда имеют значение `.TRUE.` и `.FALSE.` — во всех других случаях. \*

И, наконец, когда оба операнда в логической операции `.OR.` имеют значение `.FALSE.` результатом этой операции будет значение `.FALSE.` Во всех остальных случаях результат есть логическое значение `.TRUE.`

При построении логического выражения с помощью логических операций два символа логических операций не должны следовать непосредственно друг

за другом, за исключением случая, когда вторым является символ операции отрицания, например:

L1.AND.L2.OR.L3.AND..NOT.L1

где L1, L2, L3 являются логическими выражениями.

Таким образом, в логическом выражении могут содержаться знаки арифметических операций, символы операций отношения и логических операций. Порядок действий определяется скобками, а при отсутствии скобок сначала выполняются арифметические операции, затем — операции отношения и в последнюю очередь — логические операции.

Внутри первого уровня иерархии соблюдается рассмотренное выше правило раигов. Операции отношения являются все одноранговыми. Последовательность выполнения логических операций следующая: отрицание, логическое умножение, логическое сложение. Одноранговые операции выполняются последовательно слева направо. В следующем примере цифрами под символами операций указана очередность выполнения действий

A.GT.D\*\*K.AND..NOT.L.OR.N  
<sub>2 1 4 3 5</sub>

### Упражнения

1. Составить арифметические выражения, соответствующие математическим формулам:

$$a + \frac{b+c}{a}, \quad (x+y)^{\frac{kx}{ly}}, \quad (a^2 + b^2)^{3/2},$$

$$a + \frac{b}{c+a}, \quad \left(\frac{q_1}{q_2}\right)^{\frac{y}{y-1}}, \quad 4\left(\frac{a^2}{b^2}\right)^{1/3},$$

$$x \frac{x^2 - y^2}{x^2 + y^2}, \quad a^{b^{a+b}}, \quad (2\pi + x)(4\pi - y)^3.$$

2. Определить, соответствуют ли приведенные в правом столбце выражения на фортране стоящим слева математическим выражениям:

$a^{b^2}$	A**B**2
$ab^2$	A*B**2
$\frac{ab}{x+y}$	A*B/(X+Y)
$\left(\frac{a+b+c+\pi}{2d}\right)^2$	(A+B+C+3.1416)/(2*D)**2

3. Определить типы и длины значений следующих арифметических выражений:

$$\begin{aligned} &(A**2+N)*B+3.0*N+L \\ &((K-L**3)*K+1)*A+2-4*K \\ &-(X+Y*R)+((Z+Y)/X+K**I+R)**2 \\ &(C/(H*X-R)+R/H)*G+1D07 \end{aligned}$$

и типы, длины и значения следующих арифметических выражений:

$$\begin{aligned} &X+(A*B-X)*3.1+X/A \\ &(N**2-A1*C+4.1)*A1-6.3*C/A1+B1 \end{aligned}$$

если N, L — целые переменные нестандартной длины; K, I — целые переменные стандартной длины; A, B, A1, B1 — действительные переменные стандарт-

ной длины; X, Y, R, Z — действительные переменные нестандартной длины; C, H, G — комплексные переменные стандартной длины.

В двух последних выражениях переменные X, A, B, N, A1, C, B1 имеют соответственно значения:  $-3.76D + 1$ , 0.06, 0.0002E2,  $-7$ , 4.02, (5.1, 0.06), 105.06E-4

4. Для каких целочисленных значений K, заключенных между  $-6$  и  $+5$ , логическое выражение

$$(K+4.LE.0.OR.K.GT.-4).AND.(K.LE.3.OR.K-4.GT.0)$$

имеет значение FALSE? Можно ли это выражение написать без скобок?

5. Являются ли следующие записи конструкциями языка фортран? Если да, то как они называются?

E(K)	L(1, 2, 3)	X**2 + Y**2
XYZ	(A, B)**2	F(L)*D(I, J, K)
Y(Z)	OPERAT	A.AND..NOT.B.OR.C
E(X)	TRUE	.TRUE.

6. Определить значения следующих логических выражений:

a)  $.NOT.A.AND.B.OR.X.GE.Y+1.0$

если X и Y — переменные действительного типа, значения которых соответственно равны 64.02E02 и 6002. A и B — переменные логического типа, значения которых соответственно равны TRUE и FALSE.

б)  $X*Y/(Z-X).NE.X.OR.NOT.(F.AND.A.OR.F).AND..NOT.F.OR.A$

если значения действительных переменных X, Y, Z соответственно равны  $-4.6$ , 3.7,  $-16.2E-1$ , а логических переменных F и A — соответственно TRUE и FALSE.

### § 3. Структура программы

Программа составляется из операторов. *Оператор* — это основной элемент языка, представляющий собой предписание, приводящее к однозначному выполнению определенных действий. Действия ЭВМ могут заключаться в выполнении операций вычисления или передачи управления или же в сообщении информации о свойствах данных и отдельных элементов программы, структуре переменных, размещении массивов или совмещении рабочих полей в памяти. Операторы языка, записанные на бланке программирования (или листе бумаги) по установленным правилам в соответствии с алгоритмом решаемой задачи, в совокупности образуют *программу*. Для выполнения на вычислительной машине программа заносится на перфокарты, а получающийся при этом массив перфокарт, снабженный соответствующими служебными управляющими картами, вводится в машину.

В зависимости от характера действий различают операторы исполняемые и неисполняемые. *Неисполняемый* оператор является указанием транслятору и не переводится транслятором в команды машины. В соответствии с *исполняемыми* операторами программы транслятор создает в кодах машины рабочую программу, по которой затем производятся заданные действия (например, вычисления).

Операторы могут быть подразделены на пять основных видов: описательные операторы, операторы присваивания, операторы управления, операторы ввода и вывода и подпрограммы.

*Подпрограмма* представляет собой последовательность операторов и комментариев, которая по своему смысловому значению рассматривается как

один оператор. Под *комментарием* понимается текст, используемый для пояснения действий, выполняемых в программе.

Записывается оператор с помощью ключевого слова или заголовка оператора, после которого следует тело оператора:

$M \text{---} KC \text{---} T$

где  $M$  — метка,  $KC$  — ключевое слово,  $T$  — тело оператора. Исключение составляют арифметический и логический операторы присваивания и оператор определения оператор-функции, запись которых начинается с идентификатора с последующим символом  $=$  (знак присваивания), после чего следует выражение. Тело оператора представляет собой список, элементы которого и разделители между ними в каждом конкретном случае зависят от назначения оператора.

Последовательность операторов и комментариев, не содержащая операторов BLOCK DATA, FUNCTION, SUBROUTINE, называется *основной программой*.

*Программной единицей* (или сегментом программы) называется основная программа или подпрограмма.

Программа всегда состоит из основной программы и может включать одну или несколько подпрограмм.

Выполнение программы начинается с первого исполняемого оператора основной программы. К программной единице можно обращаться как из основной программы, так и из других подпрограмм. Программная единица, из которой осуществляется обращение к другой подпрограмме или к функции, называется *вызывающей программой*.

## § 4. Операторы описания типа

*Описательные* операторы определяют свойства переменных, массивов и функций и содержат информацию, необходимую для размещения величин в памяти. С помощью описательных операторов можно задавать начальные значения данных. В состав описательных операторов входят операторы описания типа.

**1. Неявное объявление типа.** Определение типа величины (переменной, массива или функции) с помощью первой буквы ее идентификатора является неявным определением типа и называется автоматическим объявлением типа или объявлением типа по предварительному соглашению. Автоматическое объявление типа допустимо только при задании типа целой или действительной величины стандартной длины.

Для описания всех величин, идентификаторы которых начинаются с заданной буквы или с любой буквы из заданной группы подряд идущих в латинском алфавите букв, служит оператор *неявного* описания типа, имеющий вид

IMPLICIT ТИП \*S1(A1), ТИП \*S2(A2), ...

где ТИП — это любой из основных символов COMPLEX, INTEGER, LOGICAL, REAL; \*S1, \*S2, ... — указатели длины (числа байтов); A1, A2, ... — список, элементами которых являются отдельные буквы или конструкция из двух

букв, между которыми стоит символ — (минус); в качестве разделителя между элементами в списке используется символ , (запятая).

Символ COMPLEX используется для описания комплексного типа, INTEGER — для описания целого типа, LOGICAL — для описания логического типа, REAL — для описания действительного типа.

В качестве указателя длины используются символы \*1, \*2, \*4, \*8 или \*16. Указатель длины \*1 применяется для задания нестандартной длины величин логического типа, указатель длины \*2 — для задания нестандартной длины величин целого типа, \*4 — стандартной длины величин логического, целого и действительного типов, \*8 — нестандартной длины величин действительного типа и стандартной длины величин комплексного типа, \*16 — нестандартной длины величин комплексного типа. Если указатель длины опущен, то предполагается задание стандартной длины.

Элемент списка оператора IMPLICIT указывает либо букву, с которой начинается идентификатор величины, либо начальную и конечную буквы последовательности букв, расположенных в алфавитном порядке, каждая из которых может являться первой буквой описываемого идентификатора. Например, два оператора

```
IMPLICIT INTEGER *2(N, P), INTEGER (Q—T)
IMPLICIT REAL (A—C, F—I), REAL *8(D)
```

описывают все величины, идентификаторы которых начинаются с букв N и P, как величины целого типа нестандартной длины (2 байта), величины, имена которых начинаются с букв Q, R, S, T, — как целые величины стандартной длины, величины с идентификаторами, начинающимися с букв A, B, C, F, G, H, I, — как действительные величины и, наконец, величины, у которых первая буква идентификатора есть D, — с двойной точностью.

Оператор неявного описания типа задает тип каждой величины по тому же принципу, что и при автоматическом объявлении типа, однако в качестве начальных букв идентификаторов величин того или иного типа выбираются буквы по усмотрению программиста. Возможно описание величин всех типов как стандартной, так и нестандартной длины. Оператор IMPLICIT, кроме того, отменяет тип и длину, установленные при автоматическом объявлении типа, т. е. приоритет отдается оператору неявного описания типа.

2. **Явное объявление типа.** Оператор *явного* описания типа объявляет тип величин не по первой букве идентификатора, а по всему наименованию. Ключевым словом такого оператора является один из символов:

```
COMPLEX INTEGER LOGICAL REAL
```

после которого следует список идентификаторов. Элементами списка могут быть идентификаторы, являющиеся именами простых переменных, массивов или функций, а также идентификаторы массивов с указанием верхних границ изменения индексов. Верхние границы изменения индексов записываются в виде целых чисел (через запятую) в скобках сразу после идентификатора массива и представляют собой максимальные значения соответствующих индексов. Количество верхних границ указывает размерность массива.

Элементы списка идентификаторов отделяются один от другого запятыми. Примеры операторов явного описания типа, определяющие величины стандартной длины:

```
INTEGER V, X(6), K, VALUE, Y1, Z, Z1
REAL MIV, J, MAXI, FORINT(10, 10)
LOGICAL MIMA, L1, X1, F(6, 6, 6)
COMPLEX C1, C2, C, M1, P(20, 10)
```

Указатель длины слова, отводимого под определяемые переменные, может быть включен в оператор явного описания типа в виде конструкции \*S. В этом случае оператор будет выглядеть так:

```
ТИП *S A1 *S1(K1), A2 *S2(K2), ...
```

где ТИП — это одно из четырех перечисленных выше ключевых слов; \*S — общий указатель длины; A1, A2, ... — идентификаторы величин; \*S, \*S2, ... — указатели длины величин A1, A2, ...; K1, K2, ... — списки верхних границ, если описываемая величина является массивом (списки K1, K2, ... могут и отсутствовать, даже если A1, A2, ... — идентификаторы массивов).

Если указатель длины для конкретной величины отсутствует, то ее длина описывается общим указателем длины \*S. Если же и общий указатель длины отсутствует, то длина всех величин в списке оператора явного описания типа принимается стандартной, и оператор имеет простейший вид. Так, оператор

```
INTEGER *2 B, C(5, 5), Q
```

описывает простые переменные B и Q и двумерный массив из 25 элементов как величины целого типа нестандартной длины, а в операторе

```
REAL *8 A, B, C *4, D, E *4, F *4(2, 2)
```

переменные A, B, D являются переменными с двойной точностью, а C, E и элементы массива F имеют стандартную длину. В операторе явного описания типа одновременно с объявлением типа и длины величины ей может быть присвоено *начальное значение*. В этом случае оператор явного описания типа имеет вид

```
ТИП *S A1 *S1(K1)/X1/, A2 *S2(K2)/X2/, ...
```

где сохранен смысл ранее использованных обозначений, а X1, X2, ... — списки значений.

Начальные значения могут быть присвоены только переменным и массивам. Начальное значение переменной задается константой, начальное значение массива — списком констант, разделяемых запятыми. В списке значений разрешено использование коэффициента кратности (повторителя) в виде конструкции J\*, где J — целая константа. Повторитель указывает, сколько раз должно быть повторено следующее за символом J\* значение.

Количество констант в списке значений должно быть равно длине массива, а сами константы имеют тип, определенный символом ТИП и указателем длины \*S (или \*S1, \*S2, ...). Например, оператор

```
REAL *8 A, B/3./, F *4(2, 2)/5., 2* 3., 6E-01/
```

приводит к тому, что переменным А и В отводятся поля по 8 байт, при этом А не получает значения, а начальное значение В будет равно 3.0; массив F расположится на четырех полях по 4 байта каждое, и элементы получат начальные значения 5.0, 3.0, 3.0, 0.6 соответственно. Здесь  $2^*$  — коэффициент кратности.

В одной программе может содержаться несколько операторов описания типа как неявных, так и явных. Оператор явного описания типа имеет приоритет над автоматическим объявлением типа и над оператором неявного определения типа. Это означает, что оператор явного задания типа отменяет тип и длину, установленные по предварительному соглашению или оператором IMPLICIT. Например, если в программе содержится оператор

```
INTEGER P2
```

то он объявляет переменную P2 стандартной целой величиной, несмотря на наличие, допустим, такого оператора:

```
IMPLICIT INTEGER *2(N, P)
```

который все идентификаторы, начинающиеся с букв N и P, связывает с целыми величинами нестандартной длины.

**3. Оператор задания размеров массивов.** Если оператором явного описания типа определяется массив, однако список верхних границ изменения индексов в нем не указывается, т.е. отсутствует указание размеров массива, то соответствующее задание структуры массива должно содержаться либо в операторе COMMON (§ 16), либо в операторе задания *размеров* массивов, записываемого в виде

```
DIMENSION M1(K1), M2(K2), ...
```

где M1, M2, ... — идентификаторы массивов; K1, K2, ... — списки верхних границ изменения индексов описываемых массивов.

В одной программе оператор DIMENSION может встретиться любое число раз. Примеры операторов задания размеров массивов:

```
DIMENSION Y(10, 30), Z(5, 10, 15)
DIMENSION A(20, 20), B(20, 20), X(100)
```

В первом примере описываются два массива: двумерный Y с 10 строками и 30 столбцами и трехмерный Z (размеры  $5 \times 10 \times 15$ ), во втором — два массива A и B одинаковой структуры и одномерный массив X, содержащий 100 элементов.

Использование в программе переменных с индексами без указания размеров массива недопустимо. Размеры массива должны объявляться лишь однажды одним из трех способов: оператором явного описания типа, оператором DIMENSION или оператором COMMON. Например, описание матрицы действительных значений

$$\begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix}$$

может быть выполнено оператором

```
REAL L(2, 2)
```

или же с помощью двух операторов:

```
REAL L
DIMENSION L(2, 2)
```

В качестве верхних границ используются главным образом целые константы. Список верхних границ может состоять из переменных целого типа только в одном случае — когда идентификатор данного элемента из списка массивов и его размеры появляются в списке формальных параметров функции или подпрограммы (§ 13). В этом случае размеры могут быть определены посредством значений фактических параметров, задаваемых при обращении к этой функции или подпрограмме. Максимальный размер, который может иметь каждый данный массив, определяется соответствующим оператором описания массива в основной программе.

Операторы описания типа, как и оператор задания размеров массивов, являются неисполняемыми и в соответствии со структурой программы должны быть расположены в программной единице до первого исполняемого оператора.

### Упражнения

1. Первыми операторами программы являются:

```
REAL MANT, INDEX, LIP(10)
INTEGER STEP, X10
COMPLEX C1, C2
REAL *8 BLOCK, PRIX, D, F
LOGICAL L
```

Определить тип следующих переменных, встречающихся в этой программе:

A	CONT	D	L	P
ARRAY	C1	F	LIM	PRIX
ALPHA	C	INDEX	LIP(5)	STEP
BLOCK	C2	INTER	M	X10

2. Описать в виде массива матрицу целых чисел:

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \end{pmatrix}.$$

3. В программе содержатся операторы:

```
IMPLICIT INTEGER *2(C, F), REAL *4(L, P)
REAL *8 FA(4)/-0.2D3, 2* -7.6D3, 0.72D-2/
REAL *8 IP *4(2, 3)/1.25, -3.14E2, 4* .6E3/, PA
```

Определить типы и длины величин:

IP, I2, FA, FB, LI, PI, L17, CA, F2, PA, P2, X2, Y,

используемых в программе, а также количество элементов в массивах, встречающихся в этой же программе.

## § 5. Операторы присваивания

Операторы *присваивания* осуществляют приписывание переменным значений. Указанные значения могут быть результатами выражений, т. е. результатами определенных действий над операндами, входящими в выражения, или же значениями меток.

Операторы присваивания различают двух видов: первого вида — арифметический и логический операторы присваивания, и второго вида — оператор присваивания метки.

**1. Операторы присваивания первого вида.** В общем случае оператор присваивания первого вида записывается следующим образом:

$$V = E$$

где  $V$  — переменная (простая или элемент массива),  $E$  — выражение, символ  $=$  называется знаком присваивания.

При выполнении оператора присваивания вначале вычисляется значение выражения, стоящего справа от знака присваивания, а затем оно присваивается величине, записанной слева от символа  $=$ .

Оператор присваивания первого вида называется арифметическим оператором присваивания, если  $E$  — арифметическое выражение, и логическим оператором присваивания, если  $E$  — логическое выражение. Левая и правая части логического оператора присваивания относятся к логическому типу, а арифметического — к одному из арифметических типов (целому, действительному или комплексному).

При выполнении оператора присваивания старое значение переменной  $V$  теряется, а операнды выражения  $E$  сохраняют свои значения. Например:

$$V = X**2 - Y**2$$

$$Z(I) = Z(I+10) + 1.$$

$$L = .FALSE.$$

$$N = N - 1$$

В последнем примере текущее значение переменной  $N$  уменьшается на единицу. Предполагается, что в третьем примере переменная  $L$  имеет логический тип.

Значения всех переменных в правой части оператора присваивания должны быть определены.

Если тип и длина правой части арифметического оператора присваивания отличаются соответственно от типа и длины его левой части, то значение выражения, стоящего справа, автоматически преобразуется к типу величины в левой части. При этом соответственно может измениться значение переменной, так как преобразования в операторе  $V = E$  выполняются по следующим правилам:

— если  $V$  — целого типа, а  $E$  — действительного типа, то  $V$  будет присвоена целая часть  $E$ ;

— если  $V$  — целого типа, а  $E$  — комплексного типа, то значением  $V$  берется целая часть вещественной компоненты  $E$ ;

— если  $V$  — действительного типа, а  $E$  — комплексного типа, то в качестве значения  $V$  берется вещественная компонента  $E$ ;

— если  $V$  — комплексного типа, а  $E$  — целого или действительного типов, то вещественной компонентой  $V$  будет значение  $E$  (преобразованное в действительный тип, если  $E$  было целым), а мнимой компонентой — нуль.

Например, если  $A, B, D, E, F$  имеют действительный тип,  $I, J, K, M$  — целый тип, а  $C$  — комплексный тип и  $A=2.0, B=3.0, J=9, K=-2$ , то операторы

$$\begin{array}{lll} I = B/A & D = J/K & M = (2.0, 3.0) \\ F = B/A & C = K + J & E = (2.0, 3.0) \end{array}$$

дают соответственно результаты: 1, 1.5, —4., (7., 0.), 2, 2.

**2. Оператор присваивания метки.** Оператором присваивания второго вида или оператором присваивания метки называется конструкция

ASSIGN  $n$  TO  $M$

где  $n$  — метка оператора,  $M$  — простая переменная целого типа. При выполнении этого оператора переменная  $M$  принимает значение метки  $n$ . Например, после выполнения оператора

ASSIGN 27 TO  $K$

переменная  $K$  получает значение метки, равное 27, и только после этого она может быть использована в программе, а именно, в присваиваемом операторе перехода (§ 7).

Изменение значения переменной, указанной в операторе ASSIGN, осуществляется новым оператором присваивания метки.

Переменная, получившая значение метки, не может быть использована как переменная, имеющая числовое значение, до тех пор, пока она его не получит, например, в арифметическом операторе присваивания.

### Упражнения

1. Описать в виде операторов присваивания вычислительные операции для нахождения действительных корней  $X_1$  и  $X_2$  квадратного уравнения  $AX^2 + BX + C = 0$ , используя для вычисления квадратного корня прием возведения в степень  $1/2$ .

2. Указать, какие из приведенных ниже операторов присваивания не содержат ошибок:

```
K=C+(A*B+C)-4.76
Z=X-(T**Z-Z)/X+T-R.GT..TRUE.
L=A.LE.B.OR.L
ASSIGN 070 TO F
ASSIGN M TO L
```

3. Составить операторы присваивания переменной  $Z$  значений следующих математических выражений:

$$\begin{array}{ll} x = y, & \frac{x}{x+y} - 2z \neq 2x \wedge x > y, \\ x + y \leq 3.15, & \neg(x \vee y) \wedge x \vee \neg y, \\ x^{y^2}, & x = y \vee z \neq y \wedge x + y > 2z, \\ \frac{x+y}{x^2+y^2} - \frac{z}{x+y}, & \neg(a \vee \neg b) \vee a \wedge b. \end{array}$$

Каков может быть тип переменной  $Z$  в каждом из получаемых операторов присваивания, а также тип остальных переменных в правой части?



Если оператор не помещается в одну строку, то разрешается часть оператора перенести на следующую строку. Перенос можно делать в любом месте строки. Признаком того, что последующая строка является продолжением предыдущей, служит любой отличный от пробела и нуля символ в шестой, помеченной на рис. 6 буквой П, позиции строки. В строке продолжения в позициях с первой по пятую должны содержаться пробелы. Например, можно писать оператор в виде, указанном на рис. 6 в третьей и четвертой заполненных строках. Символом продолжения здесь использована буква Р.

Максимальное количество строк, которое можно занимать под один оператор, равно 20.

Первая позиция бланка на рис. 6 помечена буквой С. Если в какой-либо строке в первой позиции написать символ С, то строка не воспринимается транслятором как оператор. Она, будучи набита на перфокарте, вводится в машину и печатается при выводе программы. Такая строка является пояснительным текстом и называется комментарием. В тексте комментария, начинающегося непосредственно за буквой С (со второй позиции строки), разрешается использовать все символы, которые имеются на устройствах ввода и вывода конкретной машины.

Позиции 2—5 используются под метки операторов, для этой цели разрешается использовать и первую позицию. В начальной строке оператора шестая позиция должна быть либо пустой, либо содержать цифру 0.

Во многих входных языках, если операторы короткие, разрешается в одну строку писать несколько операторов. В этом случае между операторами ставится определенный разделитель, обычно не являющийся символом фортрана и зависящий от транслятора. Так, запись строки, приведенной на рис. 6 последней, допустима для фортрана ЭВМ БЭСМ-6; здесь разделителем служит символ  $\diamond$  (ромбик). В такой строке помеченным может быть только первый оператор. Следовательно, в приведенном примере метка 26 относится к оператору  $C=A*B$ .

## § 7. Операторы перехода

Порядок выполнения операторов в программе определяют операторы *управления*, частным случаем которых являются операторы *перехода*. Выполнение программы начинается всегда с исполняемого оператора, встретившегося в записи программы первым, а затем последовательно выполняются все остальные операторы, пока некоторый оператор управления, например оператор перехода, не нарушит эту последовательность.

Различают три вида операторов перехода: оператор безусловного перехода, вычисляемый оператор GO TO, или оператор перехода по вычислению, присваиваемый оператор GO TO, или оператор перехода по предписанию (по назначению).

**1. Оператор безусловного перехода.** Записывается оператор безусловного перехода в следующем виде:

GO TO N

где  $N$  — метка исполняемого оператора. Этот оператор обеспечивает передачу управления на оператор с меткой  $N$ . Например, наличие в программе оператора

```
GO TO 347
```

означает, что следующим должен выполняться оператор, имеющий метку 347. После этого оператора выполняется оператор, стоящий за ним (а не за оператором перехода), и так последовательно до тех пор, пока не встретится другой оператор управления.

**2. Вычисляемый оператор перехода.** Этот оператор записывается в следующей форме:

```
GO TO (N), M
```

где  $N$  — список меток, содержащий любое количество меток исполняемых операторов, а  $M$  является простой целой переменной. Метки операторов в списке меток разделяются запятыми.

Вычисляемый оператор `GO TO` производит передачу управления оператору с меткой, имеющей в списке меток порядковый номер, равный текущему значению  $M$ . Значение  $M$  определяется оператором присваивания или ввода до выполнения оператора `GO TO` (т. е. вычисляется, отсюда и название — вычисляемый оператор). Например, если в программе содержится оператор

```
GO TO (7, 9, 1020, 346), L
```

то управление будет передано оператору с меткой 7, если  $L = 1$ , оператору с меткой 9, если  $L = 2$ , и т. д.

В списке меток номера операторов могут повторяться. Если к моменту выполнения вычисляемого оператора перехода переменная  $M$  принимает значение, превосходящее количество элементов в списке меток, то данный оператор передаёт управление непосредственно стоящему за ним оператору.

**3. Присваиваемый оператор перехода.** Присваиваемый оператор `GO TO` имеет вид

```
GO TO M, (N)
```

где  $N$  — список меток, состоящий из произвольного количества элементов, а  $M$  — целая переменная без индексов. Метки операторов в списке меток могут повторяться.

К моменту выполнения оператора `GO TO` значение  $M$  должно быть определено предшествующим оператором `ASSIGN` как значение одной из меток операторов, включенных в список меток.

Между операторами `ASSIGN` и присваиваемым `GO TO` может стоять любое количество операторов.

После выполнения оператора `ASSIGN` любой последующий присваиваемый оператор перехода, использующий переменную  $M$ , будет передавать управление на исполняемый оператор с меткой, значение которой присвоено  $M$ . Например, из двух операторов

```
ASSIGN 15 TO K
GO TO K, (4, 13, 16, 15, 100)
```

первый присваивает переменной  $K$  значение метки 15, а второй передает управление оператору с этой меткой.

Вычисляемый оператор перехода удобно использовать в том случае, когда из нескольких формул необходимо выбрать одну определенную формулу для расчета. Оператор перехода по предписанию используется главным образом в случае необходимости организации обращения к одному и тому же фрагменту из разных мест программы. В этом случае фрагмент записывают только один раз, а передачи управления на него организуют с помощью безусловных операторов перехода. Места возврата устанавливаются заранее с помощью операторов присваивания метки. Повторяющийся фрагмент должен оканчиваться присваиваемым оператором GO TO.

Пример. Вычислить значения выражения  $y = x^2 + P_l(x)$ , где  $P_l(x)$  — полином Лежандра;  $l = 0, 1, 2, 3, 4$ ;

$$P_0(x) = 1, P_1(x) = x, P_2(x) = (3x^2 - 1)/2, \\ P_3(x) = (5x^3 - 3x)/2, P_4(x) = (35x^4 - 30x^2 + 3)/8.$$

Фрагмент программы вычисления значения  $y$  может быть оформлен так:

```
6 M=L+1
  GO TO (8, 10, 12, 14, 16), M
8 P=1.
  GO TO 17
10 P=X
  GO TO 17
12 P=(3*X**2-1.)/2
  GO TO 17
14 P=(5.*X**3-3.*X)/2
  GO TO 17
16 P=(35.*X**4-30*X**2+3.)/8
17 Y=X**2+P
```

Здесь предполагается, что значения  $L$  и  $X$  определяются предшествующим данному участком программы, а вывод на печать значений  $Y$  и четырехкратный переход на оператор с меткой 6 — последующим участком.

Практически вычисляемый и присваиваемый варианты оператора GO TO взаимозаменяемы. Расположение меток в присваиваемом операторе GO TO не имеет значения, в то время как в вычисляемом операторе перехода оно является решающим.

### Упражнения

1. Составить последовательность операторов для вычисления значения  $y$ , если

$$y = \begin{cases} ax^2 + bx + c & \text{при } k = 1, \\ dx^2 + ex + f & \text{при } k = 2, \\ gx^2 + hx + i & \text{при } k = 3. \end{cases}$$

2. Указать метку оператора, которому будет передано управление в результате выполнения операторов:

```
60 ASSIGN 5 TO M
  GO TO 6
61 ASSIGN 64 TO M
```

```
6 ASSIGN 50 TO M
  ASSIGN 63 TO M
  GO TO M, (5, 37, 50, 63, 64, 77)
```

3. Написать операторы, осуществляющие вычисление площади одной из геометрических фигур: квадрата, прямоугольника, прямоугольного треугольника и круга, если сторона квадрата равна  $A$ , стороны прямоугольника или катеты треугольника —  $B$  и  $C$ , радиус круга равен  $R$ , а выбор той или иной фигуры осуществляется с помощью переменной  $K$  в присваиваемом операторе `GO TO`.

## § 8. Условные операторы управления

*Условными* принято называть операторы управления, ключевым словом которых является символ `IF`. Различают два вида условных операторов: арифметический условный оператор (арифметический оператор `IF`) и логический условный оператор (логический оператор `IF`).

**1. Арифметический условный оператор.** Арифметический оператор `IF` используется для изменения последовательности выполнения операторов в программе после проверки заданного условия, содержащегося в записи оператора. Оператор имеет вид

```
IF (E) N1, N2, N3
```

где  $E$  — арифметическое выражение целого или действительного типа;  $N1, N2, N3$  — метки операторов.

Действие этого оператора заключается в вычислении значения арифметического выражения  $E$  и последующей передаче управления на один из операторов, помеченных метками  $N1, N2, N3$ . Если вычисленное значение выражения  $E < 0$ , то осуществляется переход к оператору, помеченному меткой  $N1$ ; если  $E = 0$ , то управление будет передано оператору с меткой  $N2$ ; если же  $E > 0$ , то осуществится переход к оператору, имеющему метку  $N3$ .

В языке фортран-63 в арифметическом операторе `IF` допускаются выражения комплексного типа. В этом случае переход осуществляется аналогично в зависимости от значения вещественной части выражения  $E$ .

**Пример.** Пусть по ходу выполнения программы требуется вычислить  $y$  как функцию  $x$ , если значение  $x$  задано, по формулам

$$y = \begin{cases} 0.6x + 0.8 & \text{при } x \leq 3.9, \\ 0.7x + 1.0 & \text{при } x > 3.9. \end{cases}$$

Фрагмент программы, соответствующий вычислению значения  $y$ , может быть оформлен так:

```
IF (X-3.9) 40, 40, 30
40 Y=.6*X+.8
  GO TO 45
30 Y=.7*X+1.
45 <продолжение программы>
```

Метки в арифметическом условном операторе могут совпадать. Если оператор `IF` имеет метку, то она не должна совпадать с  $N1, N2$  или  $N3$ .

С помощью оператора `IF` возможно неоднократное повторение одного оператора или целой группы операторов. Например, фрагмент программы

для вычисления факториала числа  $N$ , т. е. произведения  $1 \cdot 2 \cdot \dots \cdot N$ , можно оформить, используя арифметический оператор IF:

```

NF=1
K=1
1 NF=NF*K
  K=K+1
  IF (N-K) 2, 1, 1
2 <продолжение программы>

```

Здесь результат обозначен идентификатором NF. Группа из трех операторов, первый из которых имеет метку 1, повторяется  $N$  раз.

**2. Логический условный оператор.** Логический оператор IF имеет вид

```
IF (L) S
```

где  $L$  — логическое выражение,  $S$  — любой исполняемый оператор, отличный от оператора цикла (§ 9) и логического условного оператора.

Выполнение логического условного оператора заключается в вычислении значения выражения  $L$  и в передаче управления либо оператору  $S$ , если значение выражения  $L$  есть .TRUE. либо оператору, непосредственно следующему за данным оператором, если значение выражения  $L$  есть .FALSE. Например:

```

IF (A*B.LT.4.) GO TO 11
IF (.NOT.(X.LT.26.5)) R=X*Y-12.4

```

В большинстве случаев арифметический оператор IF может выполнить те же действия, что и логический оператор IF. Иногда применение логического условного оператора более рационально.

**Пример 1.** Два варианта записи

```

IF (A-B) 10,20, 10   и   IF (A.EQ.B) GO TO 20
10 Z=X/Y              Z=X/Y

```

эквивалентны, однако во втором случае возможно написание второго оператора без метки.

**Пример 2.** Программа для вычисления корней квадратного трехчлена  $ax^2 + bx + c$  в соответствии с блок-схемой, приведенной на рис. 1 § 3 гл. I, может быть записана в следующем виде:

```

C   PROGRAM ROOTS
    READ 2, A, B, C
  2  FORMAT (3F7.3, I5)
    IF (A.NE.0.) GO TO 12
    IF (B.NE.0.) GO TO 8
    K=0
    PRINT 2, A, B, C, K
    GO TO 27
  8  K=1
    X1R=-C/B

```

```

X2R=0.
GO TO 23
12 K=2
D=B**2-4*A*C
IF (D) 15, 20, 20
15 X1R=-B/(2*A)
X2R=X1R
X1I=SQRT(-D)/(2*A)
X2I=-X1I
GO TO 25
20 S=SQRT(D)
X1R=(-B+S)/(2*A)
X2R=(-B-S)/(2*A)
23 X1I=0.
X2I=0.
25 PRINT 26, A, B, C, K, X1R, X2R, X1I, X2I
26 FORMAT (3F7.3, I5, 4E12.5)
27 STOP
END

```

Использованные в этом примере операторы STOP и END будут рассмотрены в § 10, операторы READ и PRINT — в § 20, а оператор FORMAT — в § 22.

#### Упражнения

1. Составить последовательность операторов для вычисления значений компонент векторов  $a(a_1, a_2, \dots, a_m)$  и  $b(b_1, b_2, \dots, b_n)$  по формулам

$$a_i = \begin{cases} x_i, & \text{если } x_i \geq 0, \\ -1, & \text{если } x_i < 0, \end{cases} \quad b_j = \begin{cases} x_j, & \text{если } x_j < 0, \\ 1, & \text{если } x_j \geq 0, \end{cases}$$

где  $x(x_1, x_2, \dots, x_k)$  — вектор, причем  $m \leq k$ ,  $n \leq k$ .

2. Составить последовательность операторов для вычисления значений элементов матрицы  $X(x_{ij})$ , где  $x_{ij} = y_{ij}$ , если  $i = j$  и  $x_{ij} = 0$ , если  $i \neq j$ , а матрица  $Y(y_{ij})$  задана ( $i = 1, 2, \dots, n$ ;  $j = 1, 2, \dots, n$ ).

3. Определить, какой из операторов получит управление после выполнения следующих условных операторов:

а) IF (A+X\*B.GT.0.) IF (A+2.) 3, 4, 5

если  $A = -2.0$ ,  $B = 4.25$ ,  $X = 1.5$ ;

б) IF (X+Y) 33, 2, 7

в) IF (A.OR.B.AND.C) IF (X-Y) 37, 21, 72

если  $A = .TRUE.$ ,  $B = C = .FALSE.$ ,  $X = 1.6$ ,  $Y = 1.9$ .

4. Написать программу, которая присваивает каждому элементу массива  $M$  значение, равное порядковому номеру элемента в массиве. Так, значение  $M(1)$  есть 1,  $M(2)$  есть 2 и т. д. Массив содержит 280 элементов.

5. Написать операторы, осуществляющие проверку принадлежности значения  $X$  заданному интервалу  $[A, B]$ . В случае положительного ответа вычислить  $Y = AX^2 + BX + C$ , в противном случае присвоить  $Y$  значение, равное  $X^2$ .

6. Вычислить  $F_x$  и  $F_y$  по формулам:  $F_x = 2xy - y^2$ ,  $F_y = x^2 - 2xy$ , если выполнено условие  $x > y$ . При  $x \leq y$  вычислить  $F_x$  по второй формуле, а  $F_y$  — по первой.

7. Вычислить последовательные значения  $F$  как функции переменной  $x$  по формуле:  $F = F_0 + F_1 \frac{x}{1!} + F_2 \frac{x^3}{3!} + F_3 \frac{x^5}{5!}$ , если  $x$  изменяется с шагом 0.01 от значения  $x = 0$  до  $x = 5$ , а  $F_0 = 1$ ,  $F_1 = 0.5$ ,  $F_2 = 0.3$ ,  $F_3 = 0.1$ .

## § 9. Оператор цикла

При программировании циклических вычислительных алгоритмов могут быть использованы условные операторы. Пример вычисления значения  $N!$ , приведенный в предыдущем параграфе, является иллюстрацией такой возможности.

Для этих же целей служит специальный оператор *цикла* (или оператор DO), являющийся наиболее сложным и мощным из числа операторов управления. Он записывается в виде

DO M I=N1, N2, N

где  $M$  — метка,  $I$  — простая переменная целого типа — параметр или управляющая переменная цикла,  $N1$  — нижняя граница параметра цикла,  $N2$  — верхняя граница,  $N$  — величина шага, с которым параметр  $I$  пробегает значения от нижней границы до верхней.  $N1$ ,  $N2$ ,  $N$  — целые константы или целые переменные без индексов (возможно со знаками).

Конструкция  $I=N1, N2, N$  называется заголовком цикла. В заголовке цикла величину  $N$  можно не указывать, в этом случае автоматически величина шага принимается равной 1

Операторы, следующие за оператором DO, включая тот, который помечен указанной меткой  $M$ , образуют область цикла или область действия оператора DO. Последний оператор области цикла называется концевым оператором цикла или концом цикла. Соответственно начало цикла — это первый оператор после DO.

Оператор цикла вызывает многократное исполнение операторов, составляющих область цикла. При этом первоначально параметр цикла принимает заданное начальное значение  $N1$  и выполняются операторы области цикла. Затем к значению параметра цикла прибавляется шаг  $N$ , и если новое значение параметра цикла не превосходит верхнюю границу  $N2$ , то снова выполняются операторы области цикла, и т. д. При значении  $I$ , равном  $N2$ , оператор цикла выполняется. Если же  $N2$  не кратно  $N$ , то последнее исполнение цикла производится при ближайшем, но не превосходящем  $N2$  значении параметра.

Если  $N > 0$ , то должно быть  $N1 < N2$ , если же  $N < 0$ , то, естественно,  $N1 > N2$ .

Пример 1. Вычисление  $N!$  может быть выполнено в следующем цикле

```
NF=1
DO 2 K=1, N
2 NF=NF*K
```

Пример 2. Фрагмент программы вычисления суммы квадратов 40 чисел выглядит так:

```
DIMENSION X(40)
<ввод(X)>
S=0
DO 7 I=1, 40
7 S=S+X(I)**2
```

В этом примере (как и в примере 1) цикл состоит только из одного оператора, т.е. совпадают начало и конец цикла. Тем не менее в заголовке цикла метку конца цикла необходимо указать.

Параметр цикла может использоваться внутри цикла не только как управляющая переменная. В частности, параметр цикла может выступать как индекс массива. Вообще говоря, параметр цикла может быть использован в любом выражении справа от оператора присваивания, однако он не должен подвергаться изменению, помимо естественного изменения, определяемого заголовком цикла.

Внутри цикла не допускается также изменение значений  $N_1$ ,  $N_2$ ,  $N$ , если эти величины заданы не константами, а переменными.

Пример 3. В цикле

```
DO 8 I=200, 400, 5
X=I
X=X/100
```

8 <конец цикла>

значениям  $I$  от 200 до 400 с шагом 5 ставятся в соответствие значения  $X$  от 2.0 до 4.0 с шагом 0.05. Тем самым организуется цикл, в котором шаг меньше единицы.

После последнего исполнения цикла управление передается оператору, следующему за оператором с меткой  $M$ . Выход из цикла в этом случае называется нормальным, а значение параметра  $I$  не определено.

Выход из цикла при значении параметра, не достигшем верхней границы, называется специальным. Специальный выход из цикла обеспечивается операторами GO TO или IF, передающими управление на оператор, не принадлежащий данному циклу. Текущее значение параметра цикла здесь сохраняется и может быть использовано в последующих операторах.

Первый оператор в области цикла должен быть исполняемым. Последним оператором цикла не может быть оператор GO TO, арифметический оператор IF, другой оператор DO, операторы STOP (§ 10) и RETURN (§ 13).

Концом цикла может быть логический оператор IF (L) S, в состав которого не входят операторы, которым запрещено быть последними в операторе цикла.

Если возникает такая ситуация, что цикл заканчивается одним из запрещенных операторов, в него необходимо добавить оператор продолжения

CONTINUE

Это неисполняемый оператор, он отмечает конец цикла и тем самым обязан иметь метку, указанную в операторе DO. Нахождение оператора CONTINUE в любом другом месте программы не оказывает влияния на последовательность выполнения операторов, т. е. этот оператор пропускается.

Пример 4. Найти в массиве IX первый элемент, значение которого равно 2. Программа может быть оформлена так:

```

DIMENSION IX(100)
<ввод(IX)>
DO 5 I=1, 100
  IF (IX(I)-2) 5, 10, 5
5 CONTINUE
10 <продолжение программы>

```

В этом примере оператор CONTINUE, во-первых, обеспечивает проверку следующего элемента массива, если данный оказался не равным 2, во-вторых, обеспечивает выход из цикла, если  $I = 100$ , а ни один из элементов не равен 2. Если найдется элемент, равный 2, то осуществится специальный выход из цикла, и значение  $I$  будет соответствовать номеру элемента массива.

Разрешается использование циклов в цикле. В этом случае область внутреннего DO не должна выходить за область внешнего DO. Допустим общий конец обоих операторов.

Пример 5. Переписать таблицу T размером  $10 \times 1000$  в таблицу S можно по следующей программе:

```

DIMENSION T(10, 1000), S(10, 1000)
<ввод(T)>
DO 1 I=1, 10
  DO 1 J=1, 1000
1 S(I, J)=T(I, J)

```

Комбинацию двух циклов, удовлетворяющую правилу вхождения циклов, называют вложенными циклами. Понятие вложенных циклов обобщается на случай трех и более циклов; при этом количество вложений в общем случае не ограничивается.

Вложенные циклы должны иметь в качестве параметров различные переменные. Циклы же, выполняемые последовательно, свободны от этого ограничения.

Запрещено передавать управление (операторами GO TO или IF) извне в область действия оператора цикла. Можно передавать управление извне только на оператор DO.

В случае вложенных циклов разрешается передавать управление из области действия внутреннего цикла в область действия внешнего цикла, но запрещается обратное. Разрешается также в ходе выполнения оператора DO передавать управление внешнему по отношению к данному циклу участку программы (например, к подпрограмме) с последующим возвратом в область действия того же оператора цикла.

**Упражнения**

1. Написать программу транспонирования квадратной матрицы размера  $12 \times 12$ .
2. Составить программу, определяющую число отрицательных, нулевых и положительных элементов массива T(38).
3. В двумерном массиве поменять местами строки с номерами I и K.
4. Вычислить одномерный массив Y значений функции  $y(x) = x^3 / (5x^2 + 3)^2$  для аргумента X, изменяющегося следующим образом: от 0 до 1 с шагом 0.1, от 1 до 10 с шагом 1, от 10 до 50 с шагом 5, а от 50 до 100 с шагом 10. Вместе со значением Y(I) фиксировать номер I.
5. Выполнить упражнение 1 из § 8, используя оператор цикла.
6. Составить последовательность операторов для упражнения 2 из § 8, используя оператор цикла.

**§ 10. Операторы останова и окончания**

1. Оператор полного останова. Этот оператор имеет вид

STOP N

где N — целая константа без знака, содержащая не более пяти цифр. Константа в записи оператора может и отсутствовать.

Оператор *полного останова* прекращает выполнение программы и печатает символ STOP и константу. Если в записи оператора отсутствует константа N, то ничего не печатается.

По оператору STOP задача снимается со счета и возобновить работу программы можно только с самого ее начала. Управление передается операционной системе для перехода к следующей программе.

2. Оператор условного останова. Форма записи этого оператора

PAUSE N

где константа N имеет тот же вид и смысл, что и в операторе STOP. Константа может отсутствовать или заменяться комментарием в виде текста, заключенного в апострофы:

PAUSE 'ТЕКСТ'

Оператор *условного останова* аналогичен оператору STOP. Отличие лишь в том, что после условного останова печатается либо символ PAUSE с константой или комментарием (без кавычек), либо просто символ PAUSE, если в записи оператора отсутствуют и константа, и комментарий, и программа переходит в состояние ожидания. Продолжить работу программы, начиная с оператора, следующего за оператором PAUSE, можно вмешательством с пульта машины. Этот оператор служит для контроля работы программы при отладке.

Операторы STOP и PAUSE останавливают только исполнение рабочей программы, но не влияют на работу транслятора.

3. Оператор окончания. Вид оператора

END

Оператор *окончания* отмечает конец программной единицы. Роль END заключается в том, что он указывает транслятору конец текста данной программной единицы. Это неисполняемый оператор. Оператор END должен быть последним оператором в исходной программе.

## § 11. Библиотечные подпрограммы

Под *стандартными функциями*, или библиотечными подпрограммами, подразумеваются программы вычисления элементарных функций (например,  $\sin$ ,  $\cos$  и др.), а также программы для некоторых часто встречающихся алгоритмов.

Стандартные функции подразделяются на два вида: встроенные и внешние, различающиеся реализацией процесса вычисления при обращении к ним. На практике программист не ощущает различий между обоими видами функций.

Записывается стандартная функция в виде указателя функции, т. е. с помощью идентификатора (с соблюдением обычных правил формирования идентификатора) и стоящего за ним аргумента (аргументов) в скобках. Стандартные функции используются в выражениях в качестве операндов как обычные переменные. Не разрешается только употребление идентификаторов стандартных функций в левой части оператора присваивания.

Аргументом стандартной функции может быть произвольное арифметическое выражение. Это выражение, в частности, может в свою очередь содержать библиотечную подпрограмму. Например:

$$Y = \text{COS}(X+Z)$$

Такой оператор находит значения переменных  $X$  и  $Z$ , вычисляет их сумму, затем извлекает из библиотеки подпрограмму с наименованием COS (вычисление косинуса), вычисляет значение косинуса соответствующего аргумента и присваивает полученное значение переменной  $Y$ . После этого выполняется очередной оператор программы.

Для стандартных функций тип результата устанавливается заранее. В случае целых или действительных функций сохранено правило первой буквы в наименовании при определении типа. Для результата комплексного типа и двойной точности в качестве первой буквы в наименовании стандартных функций обычно употребляются буквы C и D соответственно.

Количество стандартных функций для различных версий языка, как правило, различное. Так, в первоначальной версии фортрана IV их число равнялось 66, а в базисном фортране — только 45. В настоящее время число библиотечных подпрограмм (например, для ДЭС ЭВМ) доведено до 88. Список идентификаторов и назначения стандартных функций в каждом конкретном случае следует уточнять по описанию используемого входного языка. Стандартные функции для различных версий фортрана приведены, например, в [10, 15, 16, 18], см. также приложение II.

## § 12. Оператор-функция

В каждой программе могут встречаться выражения, зависящие от параметров и используемые многократно. Обращение к значениям таких выражений может быть осуществлено с помощью указателя функции, если их оформить (описать) в виде *оператор-функции*.

Общий вид оператор-функции:

$$F(A) = E$$

где  $F$  — наименование оператор-функции,  $E$  — выражение, арифметическое или логическое, не содержащее переменных с индексами,  $A$  — список формальных параметров; элементы списка разделяются запятыми.

Список формальных параметров состоит из идентификаторов простых переменных. В списке не допускается повторение одинаковых элементов, однако одни и те же идентификаторы могут использоваться в качестве формальных параметров нескольких оператор-функций. Формальные параметры обычно используются в выражении  $E$ . Например, конструкция

$$R(X, Y) = X**2 + Y**2$$

является определением оператор-функции  $R(X, Y)$ . Сам по себе этот оператор не исполняется, и поэтому все оператор-функции должны быть помещены перед первым исполняемым оператором программной единицы. Согласно правилу первой буквы тип этой оператор-функции действительный. Однако тип может быть задан особо описательными операторами или оператором IMPLICIT. Так, группа операторов

LOGICAL V, P

$$V(A, B, P) = A.LT.B.AND.P$$

задает оператор-функцию  $V(A, B, P)$  логического типа ( $A, B$  — действительные переменные).

Выполнение оператор-функции происходит после включения в выражение указателя оператор-функции, т. е. ее идентификатора со списком фактических параметров. В качестве фактических параметров допускаются выражения. Между формальными и фактическими параметрами должно соблюдаться соответствие в количестве, типах и порядке следования.

В процессе обращения к оператор-функции сначала вычисляются значения всех фактических параметров. Рассчитанные значения присваиваются соответствующим формальным параметрам. Затем вычисляется значение оператор-функции. Оно используется далее в том выражении, из которого происходило обращение к данной оператор-функции. Оператор-функции могут использоваться только в тех программных единицах, где они определены. Например, если в начале программы содержится оператор-функция  $R(X, Y)$ , то внутри программы оператор

$$T = R(A, B) - R(C, D)$$

вызовет вычисление значения оператор-функции  $R$  (дважды) согласно описанию при указанных фактических параметрах ( $A$  и  $B$ ,  $C$  и  $D$  соответственно).

Правая часть оператор-функции (выражение E) может содержать не только формальные параметры, но и другие переменные, а также обращения к стандартным функциям, ранее определенным оператор-функциям и подпрограммам типа FUNCTION (§ 13). В выражение E не должно входить обращение к этой же самой оператор-функции. Так, определение оператор-функции в виде

$$Q(Z) = Z + Q(X)$$

является недопустимым. Однако обращение

$$S = R(A, B) + R(R(C, D), F)$$

является правильной конструкцией. Здесь сначала вычисляется значение выражения  $R(C, D)$ , затем вновь происходит обращение к этой же оператор-функции с новыми фактическими параметрами, второй из которых есть F. Полученное значение является вторым слагаемым при получении значения переменной S. Первое слагаемое вычисляется однократным обращением к оператор-функции R.

Наименования формальных параметров оператор-функции не должны появляться в операторах COMMON, DATA, DIMENSION, EQUIVALENCE и EXTERNAL. Эти операторы будут рассмотрены ниже.

#### Упражнения

1. Оформить упражнение 3 из предыдущего параграфа в виде оператор-функции.
2. Написать оператор-функции для вычисления: 1) площади круга радиуса R; 2) объема шара радиуса R.

### § 13. Подпрограммы-функции

Рассмотренные в предыдущем параграфе оператор-функции при всех своих положительных качествах имеют ограниченное применение вследствие того, что состоят из единственного оператора — оператора присваивания, в котором вычисляется значение лишь одного выражения, обеспечивая получение, таким образом, одного значения. Так, например, нельзя составить оператор-функцию для вычисления факториала и для других достаточно простых алгоритмов.

*Подпрограмма-функция*, или подпрограмма типа FUNCTION, отличается от оператор-функции двумя особенностями:

— для определения подпрограммы-функции в ее описании может быть использовано любое количество операторов; число выходных величин также произвольно;

— отладка и трансляция подпрограммы-функции может осуществляться как совместно с основной программой, так и отдельно от нее.

Таким образом, подпрограмма-функция — это самостоятельная программная единица. Она выполняется после обращения к ней из другой программной единицы. При определении подпрограммы типа FUNCTION используются формальные параметры; информация, необходимая при выполнении подпрограммы-функции, поставляется фактическими параметрами. Возможность ав-

тономной трансляции (т. е. отдельно от других программных единиц) предоставляет значительные удобства при отладке программ.

Описание подпрограммы-функции составляется следующим образом:

```
ТИП FUNCTION F *S(A)
<любое количество операторов>
RETURN
END
```

Первый из написанных здесь операторов называется оператором начальной строки подпрограммы-функции, где F — идентификатор, являющийся наименованием подпрограммы-функции, ТИП — указатель типа, А — список формальных параметров, \*S — указатель длины подпрограммы-функции.

На месте указателя типа могут использоваться символы COMPLEX, INTEGER, LOGICAL, REAL или же он может быть опущен. Если в качестве указателя типа использован один из перечисленных символов, то он определяет тип подпрограммы-функции. Определение типа подпрограммы-функции может быть выполнено и без указателя типа и операторе начальной строки, а по первой букве идентификатора или использованием соответствующего оператора описания типа внутри подпрограммы-функции. Указатель длины в операторе начальной строки также может быть опущен.

Список формальных параметров А состоит из наименований простых переменных, массивов, других подпрограмм-функций или подпрограмм. Элементы списка разделяются запятыми. Список должен содержать хотя бы один элемент.

Формальные параметры служат только для указания типа и длины, количества и порядка записи фактических параметров, которые ставятся им в соответствие при каждом обращении к подпрограмме. Тип формальных параметров может быть задан неявно или установлен операторами описания типа, следующими за оператором начальной строки подпрограммы-функции. Если формальным параметром является идентификатор массива, то он должен быть описан внутри подпрограммы-функции.

Наименования формальных параметров имеют силу только внутри подпрограммы-функции, а за ее пределами эти же наименования могут иметь другой смысл. Список формальных параметров может содержать наименования переменных, значения которых вычисляются вместе со значением F.

После оператора начальной строки, операторов описания типа и операторов задания размеров массивов (если таковые имеются) следуют другие неисполняемые и исполняемые операторы. В неисполняемых операторах не должно упоминаться наименование F подпрограммы-функции, которое входит в оператор начальной строки. Это наименование должно встретиться хотя бы один раз как левая часть оператора присваивания или как элемент списка оператора ввода, либо как фактический параметр в операторе вызова подпрограммы (§ 14), и по крайней мере один из этих операторов должен выполняться.

Внутри подпрограммы-функции не могут находиться операторы, определяющие другие подпрограммы-функции или подпрограммы, а также операторы BLOCK DATA (§ 18) и останова. Операторы определения оператор-

функций, если они имеются, должны располагаться до первого исполняемого оператора. Ни один из элементов списка формальных параметров не должен входить в операторы COMMON, DATA, EQUIVALENCE или EXTERNAL внутри данной подпрограммы-функции.

Обращение к подпрограмме-функции осуществляется использованием ее указателя в качестве операнда в выражениях, содержащихся в другой программной единице. Указателем подпрограммы-функции, точно так же как и в случае стандартных функций и оператор-функций, является конструкция, состоящая из идентификатора подпрограммы-функции и списка фактических параметров (в скобках).

Между формальными и фактическими параметрами должно быть соблюдено соответствие в количестве, типе и порядке следования. При обращении к подпрограмме-функции в общем случае формальные параметры заменяются соответствующими фактическими и выполняются операторы, содержащиеся в определении подпрограммы-функции между оператором начальной строки и оператором *возврата* RETURN. С помощью оператора RETURN осуществляется возврат из подпрограммы в вызывающую программу. Операторов RETURN в подпрограмме-функции может быть несколько.

В подпрограмме-функции должен содержаться только один оператор END, который указывает конец подпрограммы-функции и всегда является последним оператором в определении подпрограммы типа FUNCTION.

При вычислении значений выражений, в которых содержится обращение к подпрограмме-функции, вместо указателя подпрограммы-функции фигурирует значение, которое вычисляется в результате выполнения операторов этой подпрограммы-функции при заданных значениях фактических параметров. Таким образом, операторы, содержащиеся в описании подпрограммы-функции, выполняются во всех случаях, если только в вызывающей программе используется указатель этой подпрограммы-функции.

Пример 1. Вычисление  $N!$  можно оформить в виде следующей подпрограммы-функции:

```
FUNCTION NF(N)
  NF=1
  DO 2 K=1, N
2  NF=NF*K
  RETURN
END
```

Теперь может быть получено значение трехчлена  $(N!)^2 + N! + N$ , если в вызывающей программе написать оператор

$$L = NF(N)**2 + NF(N) + N$$

где L — идентификатор результата. Отметим, что в этом примере формальный и фактический параметры совпадают по написанию, тем не менее это разные величины.

Вычисление L осуществляется в такой последовательности: после знака присваивания по идентификатору NF управление будет передано к подпрограмме-функции с таким же наименованием; формальный параметр в описа-

нии заменяется фактическим и выполняются все операторы, содержащиеся в описании; по оператору RETURN происходит возврат в вызывающую программу в оператор, из которого было произведено обращение к подпрограмме-функции, т.е. в то же место после знака присваивания, только теперь здесь уже будет использовано значение  $NF(N)$ , которое возводится в квадрат; после знака сложения вновь происходит обращение к подпрограмме-функции; затем значение  $NF(N)$ , которое здесь вычислялось заново, прибавляется к уже известному квадрату этого значения; в последнюю очередь к полученной сумме прибавляется значение  $N$ .

Очевидно, вычисление  $L$  будет экономнее, если вместо одного оператора в вызывающей программе написать два:

$$M = NF(N)$$

$$L = M**2 + M + N$$

т.е. благодаря введению дополнительной переменной и дополнительного оператора удалось избавиться от второго обращения к подпрограмме.

Отметим, что значение  $L$  может быть вычислено и в подпрограмме. Например, в следующей подпрограмме-функции одновременно вычисляются значения  $NF$  и  $L$ :

```
FUNCTION NFL(N, L)
  NFL=1
  DO 2 K=1, N
2  NFL=NFL*K
  L=NFL**2+NFL+N
  RETURN
END
```

Если тип подпрограммы-функции задается явно, то идентификатор подпрограммы-функции должен быть описан также в вызывающей программе. Например, если подпрограмма-функция имеет вид

```
REAL FUNCTION INT(X)
  INT=SIN(2./X)-COS(2./X)
  RETURN
END
```

то в вызывающей программе следует указать оператор

```
REAL INT
```

Отметим, что подпрограмма-функция  $INT(X)$  может начинаться также с операторов:

```
FUNCTION INT(X)
  REAL INT
```

или

```
REAL FUNCTION INT *4(X)
```

или

```
FUNCTION INT(X)
REAL *4 INT
```

Среди операторов, содержащихся в определении подпрограммы-функции, не допускается оператор, который непосредственно или косвенно обращался бы к ней самой.

При обращении к подпрограмме-функции в качестве фактического параметра могут быть использованы: переменная, элемент массива, массив, выражение, стандартная функция или подпрограмма-функция, подпрограмма.

Если формальным параметром подпрограммы-функции является идентификатор переменной, то фактическим параметром может быть простая переменная, элемент массива или выражение (в частности, константа или указатель функции). Формальному параметру, являющемуся идентификатором массива, соответствует фактический параметр в виде элемента массива или идентификатора массива. Если формальным параметром подпрограммы-функции является идентификатор подпрограммы-функции, то фактическим параметром может быть идентификатор стандартной функции или идентификатор подпрограммы-функции. Формальному параметру, являющемуся идентификатором подпрограммы, соответствует в качестве фактического параметра идентификатор подпрограммы.

При выполнении подпрограммы-функции замена формальных параметров фактическими осуществляется двумя способами — по наименованию и по значению.

В первом случае перед началом выполнения подпрограммы формальный параметр во всех операторах подпрограммы-функции заменяется на соответствующий фактический параметр и все действия, предусмотренные над формальным параметром, в действительности будут выполняться над соответствующим фактическим параметром. Замена формальных параметров фактическими осуществляется по наименованию, если соответствующий формальный параметр является либо идентификатором массива, либо идентификатором подпрограммы-функции или подпрограммы.

Во втором случае перед началом выполнения подпрограммы-функции значение фактического параметра присваивается соответствующему формальному параметру. После выполнения операторов подпрограммы-функции фактический параметр получает значение соответствующего формального параметра. Замена формальных параметров осуществляется по значению, если формальный параметр является идентификатором простой переменной.

Пример 2. В подпрограмме-функции

```
FUNCTION F(X, Y, Z)
A=X+Y
Z=A
Y=A**2
F=A**2+A+X
RETURN
END
```

формальные параметры являются простыми переменными. Следовательно, в качестве фактических параметров при обращении к F могут быть использованы простые переменные, элементы массива и выражения, т. е. возможны обращения:

$$R1 = R + P * F(S, T, U)$$

$$R2 = R + P * F(S ** 2 + SIN(C), COS(C) * 3.14, U)$$

$$R3 = R + P * F(D(5), D(K), D(7))$$

Во всех этих обращениях операторы подпрограммы-функции F(X, Y, Z) выполняются над формальными параметрами X, Y, Z, получившими значения соответствующих фактических параметров. Первый и второй фактические параметры являются входными, так как их значения используются в вычислениях внутри подпрограммы-функции. Третий параметр получает значения в результате выполнения подпрограммы-функции, и он называется выходным. Значение, которое имел этот параметр до выполнения подпрограммы функции, после выхода из нее будет потеряно. По смыслу подпрограммы-функции на месте третьего фактического параметра не должно стоять выражение.

Что касается третьего обращения к этой подпрограмме-функции, то массив D должен быть определен в вызывающей программной единице и все переменные, входящие в индексное выражение K, должны получить значения там же. Следует учитывать, что значение индексного выражения не может изменяться внутри подпрограммы-функции.

Если в качестве формального и фактического параметров выступают идентификаторы массивов, то формальный параметр должен быть описан внутри подпрограммы-функции, а фактический параметр — в вызывающей программной единице. Таким образом, этими описаниями определяются формальный массив и фактический массив. Формальный массив при обращении заменяется фактическим массивом: первый элемент формального массива заменяется первым элементом фактического массива и т. д. Размерности формального и фактического массивов могут быть различными.

Пример 3. Найти произведение элементов на главной диагонали матрицы. Подпрограмма-функция для этой задачи может быть записана так:

```
FUNCTION DP(A, N)
DIMENSION A(10, 10)
DP = A(1, 1)
DO 5 I = 2, N
5 DP = DP * A(I, I)
RETURN
END
```

Значение фактического параметра, соответствующего формальному параметру N в этой подпрограмме, не должно превосходить 10, следовательно, в операторе DIMENSION в вызывающей программе соответствующие максимальные значения индексов должны удовлетворять этому же условию. Иными словами, данная подпрограмма-функция может быть использована для

обработки двумерных квадратных матриц с числом строк и столбцов не более десяти. Например, в вызывающей программе могут содержаться операторы

```
DIMENSION X(8, 8)
R=DP(X, 8)
```

В массивах, которые описываются в подпрограммах-функциях, т. е. в формальных массивах, допускаются переменные или регулируемые размеры. Это означает, что при описании такого формального массива список верхних границ изменения индексов содержит простые переменные целого типа, которые и называются переменными размерами. Каждый из таких размеров должен входить либо в список формальных параметров подпрограммы-функции, либо в одну из общих областей (§ 16). К моменту обращения к подпрограмме-функции переменные размеры должны получить конкретные значения, которые передаются в первом случае через фактические параметры указателей функций, а во втором — через переменные, входящие в состав общих областей. Здесь рассматривается первая возможность.

Пример 4. Подпрограмму функцию из предыдущего примера можно переписать так:

```
FUNCTION DP(A, N)
  DIMENSION A(N, N)
  DP=A(1, 1)
  DO 5 I=2, N
5 DP=DP*A(I, I)
  RETURN
END
```

Здесь идентификатор массива  $A$  и переменный размер  $N$  являются формальными параметрами. В таком варианте не накладываются ограничения на размеры обрабатываемых матриц, так что из одной вызывающей программы может быть обращение  $R=DP(X, 8)$ , а из другой  $R=DP(Y, 100)$ , если, конечно, в вызывающих программных единицах содержатся соответственно операторы

```
DIMENSION X(8) и DIMENSION Y(100)
```

В случае использования в качестве фактического параметра элемента массива, когда формальным параметром является идентификатор массива, формальный и фактический массивы должны быть описаны обычным образом. Формальный массив заменяется в процессе обращения к подпрограмме-функции фактическим массивом, причем первый элемент формального массива ставится в соответствие элементу массива, заданному фактическим параметром. Это определяет соответствие между остальными элементами массивов. Необходимо, чтобы каждому элементу формального массива соответствовал элемент фактического массива.

Пример 5. Пусть два первых оператора некоторой подпрограммы-функции записаны в виде

```
FUNCTION P(A)
  DIMENSION A(5)
```

а обращение к ней выполняется оператором  $R=P(B(3))$  из программной единицы, в которой описан фактический массив DIMENSION B(7). Тогда в процессе обращения к подпрограмме функции P устанавливается соответствие между элементом массива A(1) и элементом B(3), A(2) и B(4), ..., A(5) и B(7).

Если формальным параметром подпрограммы-функции является идентификатор подпрограммы-функции или подпрограммы, а в качестве соответствующего фактического параметра выступает идентификатор внешней стандартной функции, подпрограммы-функции или подпрограммы, то замена формального параметра производится по наименованию, а идентификатор такого фактического параметра должен быть указан в списке идентификаторов оператора *внешних подпрограмм* в вызывающей программной единице. Этот оператор относится к неисполняемым операторам и имеет вид

EXTERNAL LI

где LI — список идентификаторов (наименований стандартных функций подпрограмм-функций и подпрограмм); элементы списка разделяются запятыми. Например:

EXTERNAL PF, COS, ALPHA, EXP

Оператор внешних подпрограмм указывает, что идентификатор, входящий в список LI, является наименованием фактического параметра, представляющего собой подпрограмму или функцию, а не переменную. Оператор EXTERNAL должен стоять перед первым исполняемым оператором в вызывающей программе.

Отметим, что наименование оператор-функции не должно появляться в списке идентификаторов оператора EXTERNAL. В то же время, если фактическим параметром оператор-функции является наименование другой функции или подпрограммы, то это наименование должно быть упомянуто в операторе внешних подпрограмм в той программной единице, в которой содержится данная оператор-функция.

Пр и м е р 6. Подпрограмма-функция имеет вид

```
FUNCTION H(A, P)
H=P(A)
RETURN
END
```

В программной единице, из которой осуществляется обращение к подпрограмме-функции H, например такое:

$V=D+H(AF, PF)$

должен содержаться оператор EXTERNAL PF, где PF — фактический параметр, являющийся наименованием некоторой определенной подпрограммы-функции. Формальный параметр A при указанном обращении заменяется на фактический AF, а значение H вычисляется как  $PF(AF)$ .

Если формальный параметр подпрограммы-функции является простой переменной, то замена его фактическим параметром может быть выполнена по

наименованию, если формальный параметр заключить с обеих сторон в символы / (наклонная черта). Например, если оператор начальной строки в примере 2 переписать в виде

```
FUNCTION F(X, Y, /Z/)
```

то при обращении к этой подпрограмме-функции оператором

```
R1=R+P*F(S, T, U)
```

при вычислении F будут использованы текущие значения фактических параметров S и T и наименование третьего параметра U.

#### Упражнения

1. Составить подпрограмму-функцию для суммирования элементов трехмерного массива.

2. Составить подпрограмму типа FUNCTION для вычисления дисперсии элементов одномерного массива A:  $D = \frac{1}{N} \sum_{i=1}^N A_i^2 - M^2$ , где  $M = \frac{1}{N} \sum_{i=1}^N A_i$

— математическое ожидание. Значение M также включить в число выходных параметров.

3. Составить подпрограмму-функцию для вычисления по формуле

$$y_n(x, y, h) = y + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4),$$

где  $K_1 = f(x, y)$ ;  $K_2 = f(x + \frac{h}{2}, y + \frac{h}{2} K_1)$ ;  $K_3 = f(x + \frac{h}{2}, y + \frac{h}{2} K_2)$ ;  $K_4 = f(x + h, y + hK_3)$ , а  $f(x, y)$  есть оператор-функция для вычисления значения выражения  $\frac{2y}{x} + x^2 e^x$ .

4. Разработать подпрограмму типа FUNCTION для нахождения корня уравнения четвертой степени. В качестве аргументов подпрограммы включить коэффициенты уравнения, начальное приближение, точность решения, наименование корня и максимально допустимое количество итераций.

## § 14. Подпрограммы

*Подпрограмма* (называемая также подпрограммой типа SUBROUTINE или подпрограммой-процедурой) является самостоятельной автономной программной единицей и имеет большое сходство с подпрограммой-функцией. Однако подпрограмма обладает рядом особых свойств, расширяющих возможности ее использования при программировании. Наиболее важное различие между подпрограммой и подпрограммой-функцией состоит в том, что с идентификатором подпрограммы не связывается никакое значение и обращение к подпрограмме осуществляется специальным оператором обращения, называемым оператором вызова подпрограммы.

Описание подпрограммы составляется следующим образом:

```
SUBROUTINE S(A)
```

```
<любое количество операторов>
```

```
RETURN
```

```
END
```

Первый оператор в описании подпрограммы называется оператором начальной строки или оператором-заголовком подпрограммы. В нем S — наименование (идентификатор) подпрограммы, A — список формальных параметров.

Список формальных параметров состоит из элементов, разделяемых запятыми. В качестве формального параметра может стоять простая переменная, идентификатор массива, подпрограммы-функции или подпрограммы, символ \* (звездочка). Формальный параметр, являющийся простой переменной, может быть заключен с обеих сторон в символы / (наклонная черта). Список формальных параметров (вместе с окаймляющими его скобками) у подпрограммы может отсутствовать.

Тип и длина формальных параметров задаются либо неявно, либо устанавливаются операторами описания типа, следующими за оператором-заголовком подпрограммы. Если формальным параметром является массив, то в подпрограмме должен содержаться описательный оператор, в котором задаются размеры массива. Формальные параметры не могут получать начальные значения, а также содержаться в операторах COMMON, EQUIVALENCE или EXTERNAL.

Среди исполняемых и неисполняемых операторов подпрограммы (между оператором начальной строки и единственным, но обязательным оператором окончания END) не должны находиться операторы BLOCK DATA, FUNCTION, SUBROUTINE, STOP и PAUSE.

Подпрограммы могут содержать внутри себя операторы CALL (см. ниже), вызывающие другие подпрограммы, и любые операторы, выполняющие обращения к подпрограммам типа FUNCTION, библиотечным подпрограммам и описанным в самой подпрограмме оператор-функциям.

Наименование подпрограммы не может появиться внутри подпрограммы в операторе присваивания, в операторах ввода или вывода или же в качестве фактического параметра в операторе вызова подпрограммы. Значения, получаемые в результате работы подпрограммы, присваиваются идентификаторам формальных параметров.

Оператор RETURN обеспечивает возврат из подпрограммы в программную единицу, из которой было произведено обращение к данной подпрограмме. Допускается использование оператора возврата с параметром в виде

```
RETURN I
```

где I — простая переменная целого типа стандартной длины или целая константа без знака. Укажем, что использование оператора RETURN в основной программе равносильно оператору STOP.

Обращение к подпрограмме выполняется из другой программной единицы оператором *вызова подпрограммы*. Этот оператор имеет вид

```
CALL S(AF)
```

где S — идентификатор подпрограммы, AF — список фактических параметров. В случае подпрограммы без параметров оператор вызова подпрограммы также не содержит параметров.

Оператор CALL передает управление подпрограмме, и подпрограмма будет выполняться до тех пор, пока в ней не встретится оператор RETURN.

Если оператор CALL является последним оператором в цикле DO, то после возврата продолжится выполнение цикла.

Подпрограммы могут вызываться как из основной программы, так и из других подпрограмм, но никакая подпрограмма не может обращаться прямо или косвенно к вызвавшей ее программной единице. Например, две подпрограммы не могут обращаться друг к другу. Следовательно, подпрограмма не должна содержать обращения к самой себе ни непосредственно, ни через другие подпрограммы.

Оператор вызова подпрограммы для подпрограмм с параметрами обеспечивает замену формальных параметров подпрограммы фактическими параметрами. Обязательно должно выполняться количественное соответствие между формальными и фактическими параметрами, а также соответствие по порядку следования их в списках A и AF. Формальные и фактические параметры, имеющие тип, т.е. параметры, отличные от наименований подпрограмм и символа \* должны быть согласованы по типу и длине.

Пример 1. Подпрограмма определения наибольшего по абсолютной величине элемента в заданной строке двумерного массива имеет вид

```

SUBROUTINE EM(A, I, XM, J, N)
C   A — ИМЯ МАССИВА, XM — НАИБОЛЬШИЙ ЭЛЕМЕНТ,
C   N — РАЗМЕР МАССИВА,
C   I — НОМЕР СТРОКИ, J — НОМЕР СТОЛБЦА, A, I,
C   N — ВХОДНЫЕ ПАРАМЕТРЫ, XM, J — ВЫХОДНЫЕ ПАРАМЕТРЫ
DIMENSION A(10, 10)
XM=ABS(A(I, 1))
J=1
DO 9 K=2, N
IF(ABS(A(I, K))—XM) 9, 9, 7
7 XM=ABS(A(I, K))
J=K
9 CONTINUE
RETURN
END

```

Нахождение наибольшего по абсолютной величине элемента YB в строке M массива ARRAY(8, 8), описанного, например, в основной программе, можно теперь осуществить, включив в основную программу оператор

```
CALL EM(ARRAY, M, YB, L, 8)
```

где L обозначена переменная, которой присваивается номер столбца. Этот оператор вызовет подпрограмму с наименованием EM, произведет замену формальных параметров фактическими и осуществит передачу управления подпрограмме EM. После выполнения подпрограммы оператором RETURN управление передается оператору, следующему в основной программе за оператором вызова подпрограммы. Полученные значения фактических параметров (выходных данных) можно использовать в основной программе до следующего обращения к данной подпрограмме.

В качестве фактического параметра в операторе вызова подпрограммы могут быть использованы: переменная, элемент массива, массив, выражение (в частности, константа, переменная или указатель функции), стандартная функция или подпрограмма-функция, подпрограмма, конструкция вида &N (N — метка оператора).

Если формальным параметром является переменная, принимающая значение в результате выполнения подпрограммы, то соответствующим ему фактическим параметром может быть только переменная, в остальных случаях фактическим параметром, соответствующим формальному параметру, являющемуся переменной, может быть выражение. Фактическим параметром может быть идентификатор массива или переменная с индексами (элемент массива), если соответствующий формальный параметр есть идентификатор массива.

Если формальным параметром подпрограммы является идентификатор подпрограммы-функции, то фактическим параметром может быть идентификатор стандартной функции или наименование подпрограммы-функции. Формальному параметру, являющемуся идентификатором подпрограммы, соответствует в качестве фактического параметра наименование подпрограммы. Фактический параметр в виде конструкции &N соответствует формальному параметру, обозначаемому символом \*. Если в списке формальных параметров подпрограммы содержится символ \*, то внутри подпрограммы используется оператор возврата с параметром, т. е. оператор RETURN I, где значение I является порядковым номером того символа \* среди всех таких символов (звездочек) в списке формальных параметров данной подпрограммы (считая слева направо), которому соответствует в списке фактических параметров метка оператора возврата в вызывающую программу.

Пример 2. Пусть вызывающая программа содержит операторы

```
10 CALL TAK(A, B, &60, C, &80)
30 Y=A+B
   GO TO 90
60 Y=A+C
   GO TO 90
80 Y=B+C
90 <продолжение программы>
```

а подпрограмма имеет вид

```
SUBROUTINE TAK(X, Y, *, Z, *)
<операторы>
IF (P) 2, 3, 4
2 RETURN
3 RETURN 1
4 RETURN 2
END
```

Очевидно, оператор с меткой 10 вызовет подпрограмму TAK. Если при выполнении оператора IF в подпрограмме окажется, что  $P < 0$ , то осуществится возврат в вызывающую программу на оператор, стоящий за оператором

CALL и имеющий метку 30, т. е. обычным путем. Если  $P = 0$ , то оператор с меткой 3 передает управление тому оператору, метка которого как фактический параметр соответствует первому символу \*, т. е. оператору с меткой 60. Если  $P > 0$ , то возврат осуществляется на оператор с меткой 80.

При выполнении подпрограммы замена формальных параметров фактическими параметрами, как и при выполнении подпрограммы-функции, осуществляется двумя способами — по наименованию и по значению. Замена формальных параметров выполняется по наименованию, если формальный параметр окаймлен с обеих сторон символами /, либо является идентификатором массива, подпрограммы-функции или подпрограммы, и по значению, если формальный параметр является идентификатором переменной и не заключен в символы /, либо является символом \*.

Связь между фактическими и формальными параметрами в подпрограмме устанавливается по тем же правилам, которые были рассмотрены в предыдущем параграфе для подпрограммы-функции, в частности, в формальных массивах допускаются переменные размеры.

Пример 3. Подпрограмма сложения матриц может быть представлена в виде

```
SUBROUTINE MAT(X, Y, Z, M, N)
  DIMENSION X(M, N), Y(M, N), Z(M, N)
  DO 2 I=1, M
  DO 2 J=1, N
  2 Z(I, J)=X(I, J)+Y(I, J)
  RETURN
  END
```

Если теперь основная программа содержит описание массивов

```
DIMENSION A(10, 10), B(10, 10), C(10, 10)
```

то она может вызывать подпрограмму MAT для сложения любых матриц с числом элементов в пределах  $10 \times 10$ , например:

```
CALL MAT(A, B, C, 14, 6)   или   CALL MAT(A, B, C, 8, 12)
```

Идентификаторы фактических параметров, являющихся наименованиями внешних стандартных функций, подпрограмм-функций или подпрограмм, должны быть указаны в операторе EXTERNAL в вызывающей программной единице.

#### Упражнения

1. Составить подпрограмму для задачи определения возможности построения треугольника из отрезков, длины которых заданы значениями переменных A, B, C.

2. Написать подпрограмму типа SUBROUTINE, с помощью которой можно было бы по известным значениям переменных A, B, X и L вычислить

$$R = \cos(2\pi X + A) e^{BX}, \quad T = \left( \frac{A + BX}{2} \right)^{L-1} - \left( \frac{A - BX}{3} \right)^{L+2}$$

$$S = \sqrt{A + BX + X^L}, \quad U = (A^2 + B^2)^{3/2} + 4X \left( \frac{A^2}{B^2} \right)^{L/3}$$

3. Составить программу вычисления определителя матрицы третьего порядка (т. е. матрицы с тремя строками и тремя столбцами).

4. Используя программу из предыдущего упражнения как подпрограмму, написать программу вычисления определителя матрицы четвертого порядка разложением по строке или столбцу.

## § 15. Оператор входа

При обращении к подпрограмме-функции или к подпрограмме стандартным способом, рассмотренным в предыдущих параграфах, управление передается оператору начальной строки соответствующей программной единицы, и выполняются последовательно все операторы, пока не встретится в этой последовательности оператор возврата. В ряде случаев такой стандартный вход в подпрограмму является нежелательным, и возникает необходимость входа в данную программную единицу не с самого ее начала. Для организации дополнительных входов в подпрограмму-функцию или подпрограмму используются операторы *входа*.

Оператор входа записывается в виде

ENTRY FS(A)

где FS — наименование входа (идентификатор), A — список формальных параметров.

Все операторы входа, а их в одной подпрограмме-функции или подпрограмме может быть несколько, располагаются среди операторов данной программной единицы в любом месте, кроме области действия оператора цикла. Перед оператором ENTRY не должна стоять метка. Оператор входа может находиться и в основной программе.

В вызывающей программе обращение к подпрограмме-функции или подпрограмме по наименованию входа выполняется по тем же правилам, что и обращение стандартным способом, т. е. по указателю функции, где идентификатором служит наименование входа, или же оператором вызова подпрограммы, где также на месте имени подпрограммы указывается идентификатор входа.

Если обращение к подпрограмме осуществляется по идентификатору, записанному в операторе ENTRY, то управление передается первому исполняемому оператору, следующему за этим оператором. При последовательном выполнении операторов, среди которых имеется оператор входа, последний, как неисполняемый оператор, пропускается.

Пр и м е р 1. Пусть подпрограмма-функция имеет вид

```

INTEGER FUNCTION F(K, L)
2  F=K-L
   RETURN
   ENTRY F1(K, L)
5  IF (L.LT.K) GO TO 2
3  F=K+L
   RETURN
END

```

Если из основной программы обращение к данной подпрограмме-функции осуществляется, например, так:

$$I = J + F(5 * M, N - 2 * M)$$

то после выполнения оператора с меткой 2 происходит возврат в вызывающую программу.

Если обращение имеет вид

$$I = J + F1(5 * M, N - 2 * M)$$

то вычисление значения функции выполнится по оператору либо с меткой 3, либо с меткой 2 в зависимости от условия в операторе с меткой 5. Второй вариант входа из основной программы в подпрограмму-функцию обеспечивается включением в ее описание оператора ENTRY.

Наименование дополнительного входа в подпрограмме-функции должно быть согласовано с наименованием подпрограммы-функции по типу. Поэтому в рассматриваемом примере, поскольку функция F имеет целый тип, в вызывающей программе должен содержаться оператор описания типа INTEGER, в список которого включены идентификаторы F и F1.

Список параметров в операторе CALL или в обращении к подпрограмме-функции должен быть согласован со списком параметров в соответствующем операторе ENTRY, если обращение выполняется по наименованию входа. В то же время списки формальных параметров в операторе начальной строки и в операторах входа могут быть различными.

**Пример 2.** Если имеется подпрограмма

```

SUBROUTINE PR(X, Y, P)
  IF (Y) 5, 3, 3
8  X = -X
5  Y = X + Y
  ENTRY PT(Y, P)
  P = 2 * Y ** 2
  RETURN
END

```

а в основной программе содержатся операторы

```

X = 8.0
Y = 3.
CALL PR(X, Y, Z)
CALL PT(4., P)

```

то результатом выполнения первого оператора CALL будет значение  $Z = 50.0$ , результатом второго — значение  $P = 32.0$ .

### Упражнения

1. Написать подпрограмму вычисления определителя матриц вплоть до четвертого порядка, предусмотрев исключение выполнения «лишних» операторов для матриц меньшего порядка (второго или третьего).

2. Создать подпрограмму-функцию для упражнения 3 из § 7, используя операторы входа. Написать операторы основной программы, которые осуществляют выбор фигуры и обращение к подпрограмме-функции.

## § 16. Оператор описания общих блоков

В предыдущих параграфах было показано, что передавать информацию из одной программной единицы в другую можно при помощи замены формальных параметров фактическими в подпрограммах-функциях и подпрограммах. Для этих же целей используются общие области памяти или общие блоки. *Общим блоком* называется группа величин, обозначенная идентификаторами простых переменных или идентификаторами массивов (с указанием верхних границ изменения индексов или без этого) и размещаемая на участке оперативной памяти, к которой возможно обращение из нескольких программных единиц. Входящие в общий блок величины, называемые обычно элементами общего блока, разделяются запятыми. Общий блок может быть помечен идентификатором, который окаймляется символом / с обеих сторон. Например, общий блок, содержащий в качестве элементов переменные X, Y, Z и помеченный идентификатором B, обозначается как /B/X, Y, Z.

Для определения общих блоков служит оператор описания общих блоков, записываемый в виде

COMMON P

где в списке P общих блоков должен стоять хотя бы один общий блок.

При описании общего блока идентификатор может отсутствовать. В этом случае соответствующий общий блок является непомеченным. Непомеченный общий блок определяется либо опусканием идентификатора блока и связанных с ним символов / (если он встречается в начале оператора COMMON), либо наличием двух последовательных символов / перед группой идентификаторов непомеченного блока. Примеры использования оператора описания общих блоков:

COMMON A, B, C, D

COMMON //A, B, C, D

COMMON /X/ P, Q /Y/ R, S//T, W, V

В первых двух примерах указан один и тот же непомеченный общий блок, в третьем примере первый блок помечен идентификатором X, второй — идентификатором Y, а третий не помечен. Отметим, что в списке общих блоков не используется в качестве разделителя символ , (запятая).

Если в общем блоке содержится идентификатор массива без индексов, то значения индексов должны определяться оператором DIMENSION или оператором описания типа в той же программной единице. Если в операторе COMMON указан идентификатор массива во списке верхних границ, то отпадает необходимость дополнительного описания этого массива. Так, в последнем примере идентификаторы P, Q, R, S, T, W, V могут относиться как к простым переменным, так и к массивам, а в общих блоках, определяемых оператором

COMMON /B1/ A(100), B(100) /B2/ C(10), D(8, 10)

A, B, C и D являются идентификаторами массивов.

Структура общего блока определяется порядком следования в нем элементов, а длина — количеством и типом идентификаторов, перечисленных в списке данного блока. Например, оператор

```
COMMON /A/ P(4), Q(4), C(2)
```

содержит общий блок длиной 48 байт, если P и Q — действительные, а C — комплексный массивы.

Идентификатор общего блока используется только для опознавания блока в процессе трансляции, так что этим же идентификатором может быть обозначена любая другая величина в программной единице, в частности элемент общего блока. Один и тот же идентификатор общего блока в операторе описания общих блоков может встречаться несколько раз. В этом случае совокупность элементов, записанных после всех совпадающих имен, определяет один общий блок. Например, операторы

```
COMMON X, Y /A/ U, V, W //Z, C(5) /A/ A(6)
COMMON X, Y, Z C(5) /A/ U, V, W, A(6)
```

определяет совпадающие общие блоки: непомеченный длиной 8 элементов и помеченный идентификатором A — 9 элементов.

Оператор COMMON неисполняемый, и он должен находиться среди неисполняемых операторов, предшествующих первому исполняемому оператору в программной единице. В программной единице может содержаться любое количество операторов описания общих областей, однако элемент из одного общего блока не должен появляться в другом общем блоке.

Оператор общих блоков обеспечивает доступ к одной и той же области памяти, соответствующей данному общему блоку, из всех программных единиц, содержащих оператор COMMON. Каждая общая область памяти выделяется именем общего блока независимо от того, какими идентификаторами обозначены элементы этого общего блока в различных программных единицах.

Пример 1. Если некоторая переменная X обозначает один и тот же объект в подпрограмме и в основной программе, то в обеих программных единицах следует написать оператор

```
COMMON X
```

Пример 2. Пусть в основной программе содержатся переменные A, B, C, а в подпрограмме соответствующие объекты обозначены X, Y, Z. В этом случае в основной программе следует написать оператор

```
COMMON /S/ A, B, C
```

а в подпрограмме — оператор

```
COMMON /S/ X, Y, Z
```

Длины одинаково помеченных общих блоков во всех программных единицах должны совпадать, а непомеченные общие блоки, определенные в разных программных единицах, могут иметь различные длины.

Пример 3. Основная программа содержит оператор

```
COMMON A(3), B, C
```

а в подпрограмме используются величины A(3) и B. В этом случае в подпрограмме можно написать оператор

```
COMMON A(3), B
```

и длины непомеченных общих блоков в основной программе и подпрограмме будут различны

Если же общий блок в основной программе является помеченным, например имеет вид

```
COMMON /T/ A(3), B, C
```

или, хотя и является непомеченным, но в подпрограмме используется величина Z, соответствующая величине C в основной программе, а величина B не используется, то длины общих блоков должны совпадать. В этих случаях в подпрограмме должен содержаться один из следующих операторов:

```
COMMON /T/ A(3), B, Z или COMMON A(3), B, Z
```

В первом из них элемент Z в подпрограмме не используется, он вводится, чтобы не нарушить длину общего блока /T/. Во втором операторе элемент B введен для того, чтобы не было нарушено соответствие величин C и Z в непомеченном общем блоке.

Оператор COMMON не допускает описания формальных параметров, для которых соответствие фактическим параметрам устанавливается обращением к подпрограммам-функциям или с помощью оператора CALL. Следовательно, все используемые в подпрограмме переменные должны быть упомянуты либо в списке ее формальных параметров, либо в списке общих блоков, т.е. в операторе COMMON (за исключением переменных, появляющихся в левой части оператора присваивания в подпрограмме).

В ряде случаев, в частности когда подпрограмма вызывается лишь из одной программной единицы, например основной программы, можно переменную перевести из списка формальных параметров в список общих блоков для основной программы и подпрограммы с помощью оператора COMMON.

Пример 4. Подпрограмма переписывания чисел из одного массива в другой может быть оформлена так:

```
SUBROUTINE C(A, B, N)
DIMENSION A(100), B(100)
DO 1 I=1, N
  B(I)=A(I)
RETURN
END
```

Формальный параметр N (N — размер массива,  $N \leq 100$ ) можно включить в список общих блоков, если в основной программе будет оператор COMMON N.

Подпрограмма примет вид

```

SUBROUTINE C(A, B)
DIMENSION A(100), B(100)
COMMON N
DO 1 I=1, N
1 B(I)=A(I)
RETURN
END

```

Если же размеры массивов в основной программе не изменяются, то подпрограмма может вообще не иметь формальных параметров:

```

SUBROUTINE C
COMMON A(100), B(100), N
DO 1 I=1, N
1 B(I)=A(I)
RETURN
END

```

В этом случае в основной программе должен быть оператор COMMON A, B, N, а также заданы размеры массивов A, B и оператор вызова подпрограммы в виде CALL C.

Общие блоки используются не только для передачи информации из одной программной единицы в другую, но и для целей экономии памяти машины, когда вводится общий блок для того, чтобы одну и ту же область памяти сделать доступной нескольким программным единицам, хотя и нет необходимости в обмене информацией между этими программными единицами. Поэтому сопоставляемые переменные в операторах COMMON, принадлежащих различным программным единицам, могут иметь любой тип (целый, действительный, комплексный, логический) и любую длину (стандартную, нестандартную), лишь бы только совпадали длины в байтах для всех одноименных (помеченных) общих блоков. Например, в основной программе могут содержаться операторы

```

COMMON /X/ A(2), B, D(3)
REAL *8 A
COMPLEX D

```

а в подпрограмме —

```

COMMON /X/ P(2), L(6), M, N, C(2)
INTEGER L*2, M, N
COMPLEX C

```

так что область памяти /X/ длиной 44 байта используется обеими программными единицами.

#### Упражнения

1. Основная программа обрабатывает массивы T1(10) и T2(5, 5), действительные переменные A, B, C, D, E, F и целые K, L, M, N; она обращается к трем подпрограммам. В первой из них участвует массив T1, действитель-

ные переменные A, C, E и все целые переменные; во второй — массив T2, все действительные переменные и целые L, M; третья подпрограмма использует оба массива, действительные переменные C, D, E, F и целые K, L, M. Написать операторы COMMON для всех программных единиц. Сколько общих блоков всего потребуется?

2. Написать программу вычисления корней квадратного уравнения  $AX^2 + BX + C = 0$  с использованием подпрограмм нахождения каждого из корней. Идентификаторы корней уравнения оформить как общие переменные подпрограммы и основной программы.

## § 17. Оператор присваивания начальных значений

Перед выполнением программы простым переменным и элементам массива могут быть присвоены начальные значения неисполняемым оператором

DATA A /R/, B /S/, ...

который называется оператором *присваивания начальных значений*.

Здесь A, B, ... — списки, содержащие простые переменные, переменные с индексами (индексы должны быть целыми константами) или наименования массивов; R, S, ... — списки констант (числовых, логических, текстовых или шестнадцатеричных). Любой константе может предшествовать конструкция вида J\*, называемая коэффициентом кратности (повторителем), где J — целая константа, отличная от нуля. Повторитель J\* указывает число переменных из предыдущего списка, которым должно быть присвоено данное значение, т. е. сколько раз должна быть повторена данная константа.

Элементы списка разделяются запятыми. Количество элементов в одном списке переменных должно совпадать с числом элементов в соответствующем списке констант. Тип значения должен совпадать с типом элемента, которому присваивается значение. Значения, задаваемые текстовыми и шестнадцатеричными константами, можно присвоить элементам любого типа.

Пример 1. Оператор

DATA K1 /1/, K2, T(1, 2) /2, 3.0/, X, Y /2\*1.5/, L /.TRUE./,  
\*C, D /'TEXT', 4NTEST/, LB /2, 3, 2\* 1, 2\* 4, 6, 5/

присваивает переменным K1 и K2 значения 1 и 2 соответственно, элементу массива T(1, 2) — значение 3.0, обоим переменным X и Y — значение 1.5, логической переменной L — значение .TRUE., а переменным C и D — значения текстовых констант TEXT и TEST соответственно. Элементы массива LB получают соответственно значения 2, 3, 1, 1, 4, 4, 6, 5. Так как здесь оператор DATA занимает две строки, вторая строка начинается с символа продолжения, в качестве которого выбран символ \*.

Все описательные операторы, определяющие переменные, используемые в операторе DATA, должны предшествовать оператору присваивания начальных значений. Так, в предыдущем примере должен быть описан массив LB(8).

Переменные получают начальные значения с помощью оператора DATA перед выполнением программы независимо от того, в каком месте программы он расположен, однако оператор присваивания начальных значений должен находиться в программе после всех других неисполняемых операторов.

Значения переменных, определенные оператором DATA, могут в процессе выполнения программы изменяться, например, операторам присваивания или ввода, но не оператором DATA. В списке переменных оператора DATA не должны содержаться формальные параметры подпрограмм-функций и подпрограмм, а также элементы из общих блоков.

Пример 2. Операторы

```
DATA A, B, C(5) /2.5, 2* 3.7/
DATA A /2.5/, B /3.7/, C(5) /3.7/
```

являются эквивалентными и присваивают переменным A и B соответственно значения 2.5 и 3.7, значение 3.7 получает также элемент массива C(5).

Пример 3. Если заданы операторы

```
DIMENSION X(5), Y(8)
DATA X, Y /1.1, 2.1, 3.1, 4.1, 5.1, 4* 6.1/
```

то элементы массива X получают начальные значения: 1.1, 2.1, 3.1, 3.1, 3.1, а элементы массива Y — следующие начальные значения: 3.1, 3.1, 4.1, 5.1, 6.1, 6.1, 6.1, 6.1.

#### Упражнения

1. В программе содержатся операторы

```
REAL A, B(10), C, D(3, 5), E
DATA A, B, C, D, E /5* 2.8, 3.2, 3* 1.6, 6* 2.9, 15* 0., 8* 1.0/
```

Определить начальные значения всех описанных действительных величин.

2. Составить операторы, в которых были бы описаны типы величин A, B, C, D, E, F, H, если A, C — целые стандартной длины, B, D, E — действительные нестандартной длины, F, H — комплексные стандартной длины, и всем величинам присвоены посредством оператора DATA начальные значения, в том числе вещественной и мнимой частям комплексных величин, равные единице.

## § 18. Подпрограмма данных

Для присваивания начальных значений элементам из помеченного общего блока используется специальная подпрограмма, называемая *подпрограммой данных*.

Структура подпрограммы такова:

```
BLOCK DATA
. . . . .
END
```

где между операторами BLOCK DATA и END могут стоять операторы описания типа, операторы DIMENSION, COMMON, EQUIVALENCE, IMPLICIT и оператор присваивания начальных значений. Никакие операторы, кроме перечисленных, не должны содержаться в подпрограмме BLOCK DATA. Подпрограмма данных должна содержать хотя бы один оператор COMMON. С помощью одной подпрограммы данных можно присвоить начальные значения нескольким помеченным блокам.

Начальные значения можно присванвать не всем элементам общего блока, а только некоторым. Однако в подпрограмме данных в соответствующем операторе COMMON должны быть перечислены все элементы общего блока — как те, которым присванваются начальные значения внутри подпрограммы данных (операторам описания типа или оператором DATA), так и те элементы, которые не получают начальных значений.

Пример.

```
BLOCK DATA
COMMON /P/ A, M, B, D, C
DIMENSION M(2, 2)
REAL M, D, B
INTEGER A, C /1/
DATA M(1, 2), D /2.3, 4.12/, A /0/
END
```

С помощью этой подпрограммы начальные значения получают только переменные A, C, D, M(1, 2), являющиеся элементами общего блока /P/.

Подпрограмма BLOCK DATA автономно не транслируется, а подключается к той программной единице, в которой указаны содержащиеся в подпрограмме данных общие блоки. Если такой программной единицей является подпрограмма, то переменные из общего блока определены с момента обращения к подпрограмме до выхода из нее по оператору RETURN.

Таким образом, задание начальных значений переменным может быть осуществлено тремя способами: операторам явного описания типа, оператором DATA и подпрограммой BLOCK DATA. Второй способ от первого отличается лишь тем, что в операторе DATA не определяется тип, а третий способ пригоден только для помеченных блоков COMMON. Задание начальных значений элементам непомеченного общего блока не предусмотрено.

#### Упражнения

1. Оформить подпрограмму из упражнения 2 § 14 как отдельную программу, предусмотрев задание начальных значений.
2. Пусть задана следующая подпрограмма данных:

```
BLOCK DATA
INTEGER A(2)
REAL K
LOGICAL L, M
DIMENSION B(3)
COMPLEX C
COMMON /BA/ A, K, L, M /BB/ B, C, D(2)
DATA A /2* 15/, K /7.5/, L, M /FALSE., .TRUE./,
*B /2* 1.5, 2.5/, C /(1.2, 1.2)/, D /5, 1.5/
END
```

Указать, какие начальные значения будут иметь входящие в нее величины.

## § 19. Оператор эквивалентности

В тех случаях, когда в различных частях одной и той же программной единицы используются величины, которые имеют отношение только к одной части этой программной единицы, можно достичь экономии памяти при

программировании, если для запоминания указанных величин использовать одну и ту же область памяти. Выделение одного и того же места в памяти для размещения нескольких величин осуществляется оператором эквивалентности, имеющим вид

### EQUIVALENCE Q

где Q — список групп эквивалентности. Каждая группа эквивалентности представляет заключенный в скобки список, элементы которого являются простыми переменными или переменными с индексами, размещаемыми в одной и той же области памяти. Элементы в обоих списках разделяются запятыми, количество их в группе эквивалентности должно быть не меньше двух.

Пример 1. Оператор

### EQUIVALENCE (K, L, M), (A, B)

указывает, что значения переменных K, L, M помещаются в одной и той же области памяти, а переменных A и B — в другой, т. е. тождественными являются переменные K, L и M, и A и B.

Если в группе эквивалентности стоит переменная с индексами, то ее список индексов может содержать только константы. Одновременно с объявлением эквивалентными элементов массивов, указанных в группе эквивалентности, эквивалентность распространяется на все предшествующие и последующие элементы массивов. Например, оператор

### EQUIVALENCE (X(10), Y(2), Z(3))

устанавливает эквивалентность между следующими тройками переменных с индексами:

X(9)	Y(1)	Z(2)
X(10)	Y(2)	Z(3)
X(11)	Y(3)	Z(4)
.....		

образующими в памяти некоторую область эквивалентности.

Обычно оператор EQUIVALENCE используется в тех случаях, когда два или более массивов могут занимать одно и то же место памяти, при этом структура и длина массивов могут не совпадать.

При установлении эквивалентности необходимо учитывать типы переменных и массивов. При помощи оператора эквивалентности нельзя расчленять массивы на отдельные части, а также устанавливать соответствие между элементами одного и того же массива.

Если переменная, занимающая два слова (по 4 байта каждое), эквивалентна переменной, занимающей одно слово, то последняя будет совмещена с первым словом первой переменной.

Пример 2. Если в программной единице заданы операторы

```
DIMENSION A(2, 4), B(2, 3, 2), C(6)
REAL *8 U, V(5), T
COMPLEX R, S(3)
```

EQUIVALENCE (A(1, 1), B(1, 2, 1), U),  
 \*(S(1), V(1), C(1)), (R, T, J)

то в памяти образуются три области эквивалентности. Первая из них содержит в последовательных ячейках (длиной 4 байта) следующие элементы: A(1, 1), B(1, 2, 1), старшие разряды U; A(2, 1), B(2, 2, 1), младшие разряды U; A(1, 2), B(1, 3, 1); ...; A(2, 4), B(2, 2, 2); вторая: вещественная часть S(1), старшие разряды V(1), C(1); мнимая часть S(1), младшие разряды V(1), C(2); ...; мнимая часть S(3), младшие разряды V(3), C(6); третья: вещественная часть R, старшие разряды T, J; мнимая часть R, младшие разряды T.

Оператор EQUIVALENCE является неисполняемым и должен находиться среди неисполняемых операторов, предшествующих первому исполняемому оператору в программе или подпрограмме.

Не допускается установление эквивалентности между элементами общих блоков (прямо или косвенно). Если две переменные или два элемента массива совмещены по памяти вследствие действия оператора EQUIVALENCE, идентификаторы этих переменных или массивов не могут оба одновременно появляться в операторах COMMON в той же самой программной единице.

Одна и та же переменная может встретиться и в операторе COMMON и в операторе EQUIVALENCE. В этом случае сначала будет этой переменной отведено место в общем блоке, а уже потом установлено соответствие по оператору эквивалентности. Например, операторы

```
DIMENSION D(3)
COMMON A, B, C
EQUIVALENCE (B, D(1))
```

расположат переменные в памяти следующим образом:

```
  A      B      C
      D(1)  D(2)  D(3)
```

тем самым расширяя общий блок размещением новых элементов после последнего элемента блока.

Не разрешается задавать оператор эквивалентности, который потребовал бы выход за начало общего блока, т. е. запрещено расширять общий блок путем размещения новых элементов до первого элемента блока. Так, в приведенном примере на месте D(1) нельзя писать D(3), поскольку это приведет к следующему расположению переменных:

```
      A      B      C
D(1) D(2)  D(3)
```

#### Упражнения

1. Написать оператор, предусматривающий размещение в одном и том же месте памяти 70 первых элементов массива M и 70 последних элементов массива N. Массивы M и N двумерные и состоят каждый из 100 элементов (10 × 10).

2. Определить значения элементов массива X, которые они получают в результате выполнения следующей программы:

```

REAL X(10) /2* -1.8, 2.75, 7* 0.12/,
*Y(6) /16.1, 3* -11.8, -17.1, -7.9/, Z(6)
EQUIVALENCE (X(5), Z(1))
DO 6 K=1, 6
6  Z(K)=Y(K)
STOP
END

```

## § 20. Операторы ввода и вывода

**1. Понятие файла.** Операторы *ввода* и *вывода* предназначены для обмена информацией между оперативной памятью вычислительной машины и внешними запоминающими устройствами. Внешними устройствами являются: устройство ввода перфокарт, магнитофон, магнитный барабан, устройство с дисками, алфавитно-цифровое печатающее устройство (АЦПУ), выходной перфоратор и др. Вводимая или выводимая информация представляется в виде последовательностей записей и носит название *файла*. *Запись* представляет собой последовательность символов.

Каждому файлу присваивается условный номер, который может быть использован в операторах ввода или вывода в виде целого числа без знака или переменной целого типа. В качестве файла, как правило, рассматривают последовательность данных на совокупности перфокарт, на бумажной ленте, на диске, барабане, на магнитной ленте и т. д. Так, совокупность перфокарт (массив перфокарт) представляет собой файл, в котором записью является отдельная перфокарта, содержащая не более 80 символов. Устройство печати рассматривается как файл, в котором под записью понимается одна печатная строка на бумажной ленте. Совокупность определенного количества строк образует страницу. Количество позиций в строке и число строк в странице определяются устройством печати (для АЦПУ-128 запись может иметь максимальную длину 128 символов, в странице 60 строк). Одну перфокарту (80 символов) и одну строку (128 символов) принято называть единицей записи, подчеркивая тем самым, что запись для файлов на перфокартах и бумажной ленте имеет всегда фиксированную длину. Количество реально занятых позиций может быть меньше указанного числа, но не больше.

Различают файлы последовательного доступа и прямого доступа.

Файл последовательного доступа — это файл, в котором определены начальная и конечная записи, а относительно любой другой записи определены понятия «текущая запись», «предыдущая запись» и «последующая запись». Файл последовательного доступа используется в тех случаях, когда записи передаются в файл в порядке их расположения.

Файл прямого доступа — это файл, в котором нумерация записей имеет определяющее значение. Файл прямого доступа используется в тех случаях, когда необходимо выполнить передачу отдельных записей файла в произвольном порядке. Каждый из файлов прямого доступа должен быть описан в программе один раз с помощью оператора описания файлов.

Во время передачи информация может быть преобразована; например, числовые значения преобразованы в двоичную (при вводе) или десятичную (при выводе на АЦПУ) формы. Обмен информацией в ее машинном (двоичном) представлении называется бесформатным обменом. Например, занесение программы или результатов выполнения программы на магнитные ленты, диски, барабаны в тех случаях, когда эта информация будет повторно использована, обычно представляет собой бесформатный обмен. Обмен информацией с преобразованием в соответствии со спецификациями оператора **FORMAT** (§ 22) называется форматным обменом. Примером форматного обмена может служить вывод данных на АЦПУ — здесь информация из двоичного представления преобразуется в десятичную форму, или ввод числового материала с перфокарт, где он набит в десятичном представлении, и при вводе преобразуется в двоичную форму представления.

Операторы ввода и вывода указывают характер обмена информацией (чтение, запись, печать и т. д.), номер файла, с которым происходит обмен (номер файла, как правило, содержит информацию о типе устройства, на котором организован файл, о так называемом логическом номере устройства), и список величин, которые участвуют в обмене. Кроме того, для организации ввода или вывода в определенных случаях необходимо указать тип передаваемых величин и связанную с этим информацию о преобразовании (формат данных и их расположение в файле).

2. **Оператор описания файлов.** Оператор *описания файлов* используется для описания файлов прямого доступа и имеет вид

**DEFINE FILE N(M, H, F, K)**

где наборов символов  $N(M, H, F, K)$  в списке оператора может быть произвольное число (в этом случае один набор от другого отделяется запятой). Каждый набор  $N(M, H, F, K)$  определяет свой файл. Здесь  $N$  — номер файла,  $M$  — целое число без знака, определяющее максимальное число записей этого файла (записей может быть на самом деле меньше  $M$ , но не больше),  $H$  — целое число без знака, определяющее максимальный размер каждой записи. Числовое значение  $H$  указывается либо в байтах, либо в машинных словах; при этом выдерживается соотношение: одно слово равно четырем байтам. Что принимается за единицу измерения, указывается параметром  $F$ , на месте которого может стоять один из трех символов:  $E$ ,  $L$  или  $U$ .

Если  $F$  есть  $E$ , то размеры записей файлов указываются в байтах, и обращение к файлу такого типа осуществляется операторами ввода или вывода только с форматами (см. пп. 7 и 8); в этом случае говорят, что все записи являются форматными. Если  $F$  есть  $U$ , то размеры  $H$  записей файлов измеряются количеством слов, и обращение к файлам выполняется операторами ввода и вывода только без форматов (пп. 5 и 6); записи являются бесформатными. И, наконец, если  $F$  есть  $L$ , значение  $H$  указывается в байтах, а обращение к файлу такого типа осуществляется операторами ввода и вывода как с форматами, так и без форматов (имеет место смесь форматных и бесформатных записей).

Символ  $K$  — простая переменная целого типа. Эта переменная автоматически принимает значение, равное номеру записи, которая непосредственно

следует за последней из обработанных (т. е. введенных или выведенных) записей. После выполнения оператора поиска элемента файла переменная принимает значение, равное номеру найденной записи. Переменная  $K$  — номер записи — не может использоваться в программе в каком-либо другом смысле.

В одной программе может содержаться несколько операторов описания файлов, и все они должны находиться среди неисполняемых операторов до первого исполняемого оператора программы. Например, оператор

DEFINE FILE 9(128, 8, U, J), 14(66, 50, E, I), 5(30, 48, L, N)

определяет три файла с номерами 9, 14 и 5: первый — это 128 бесформатных записей, каждая запись содержит 8 слов, переменная  $J$  будет принимать значения 2, 3, ..., 129 после обработки записей соответственно с номерами 1, 2, ..., 128; второй содержит 66 форматных записей, каждая из которых состоит не более чем из 50 байт, переменная  $I$  будет принимать после обработки первой записи значение 2, после обработки второй записи — значение 3 и т. д. до значения 67; файл, имеющий номер 5, состоит из 30 записей, каждая длиной не более 48 байт, переменная  $N$  будет принимать значения 2, 3, ..., 31 после обработки соответствующих записей с номерами 1, 2, ..., 30. Символ  $U$  в описании первого файла указывает, что для передачи информации должны использоваться операторы ввода или вывода только без форматов. Символ  $E$  в описании файла с номером 14 указывает, что для обработки записей этого файла должны использоваться только операторы ввода или вывода с форматами. Записи файла с номером 5 могут передаваться операторами ввода или вывода с форматами и без форматов, на что указывает символ  $L$  в описании.

**3. Список ввода-вывода.** С операторами ввода и вывода связано понятие *списка ввода-вывода*, который определяет участки оперативной памяти, используемые для ввода или вывода информации. Иными словами, в списке ввода-вывода указываются величины, участвующие в обмене информацией между оперативной памятью и внешними устройствами, и порядок передачи значений этих величин.

Список ввода-вывода имеет вид

$A_1, A_2, \dots, A_N$

где элементами списка могут быть простые переменные и переменные с индексами, наименования массивов и неявные циклы.

Под неявным циклом понимается конструкция вида

$(B_1, B_2, \dots, B_K, I=N_1, N_2, N_3)$

где элементы  $B_1, B_2, \dots, B_K$  — либо переменные (простые или с индексами), либо в свою очередь неявные циклы,  $I=N_1, N_2, N_3$  — заголовок цикла, в котором:  $I$  — целая переменная, используемая в индексных выражениях переменных  $B_1, B_2, \dots$ ;  $N_1, N_2, N_3$  — целые константы или простые переменные целого типа; если  $N_3$  имеет значение, равное +1, то элемент  $N_3$  в записи заголовка цикла может быть опущен. Например, неявный цикл может иметь вид

$((R(I, J, K), I=L_1, L_2, L_3), J=M_1, M_2, M_3), K=N_1, N_2, N_3)$

Эта конструкция осуществляет перечисление элементов массива R в соответствии с порядком выполнения вложенных циклов:

```
DO N K=N1, N2, N3
DO N J=M1, M2, M3
DO N I=L1, L2, L3
```

где N — метка конца цикла.

Примеры списков ввода-вывода:

```
X, Y, K, H(J), N(3, 5)
VA, S, (Z(I), I = 1, 20), E(J + 1)
((A(I, J), I=1, 3), J=1, 5)
((A(I, J), J=1, 5), I=1, 3)
(X(J), Y(J), J=1, 30)
(I, P(I), I=1, 100)
```

Если в качестве элемента списка ввода-вывода указана переменная, то это означает, что необходимо выполнить ввод или вывод значения этой переменной. Если в качестве элемента списка указан идентификатор массива, то это означает, что надо выполнить ввод или вывод всех элементов этого массива в том порядке, в котором эти элементы размещены в массиве. Если в качестве элемента списка ввода-вывода указан неявный цикл, то это означает, что необходимо передать значения всех переменных, указанных в неявном цикле, организовав порядок их следования в соответствии с циклическим изменением параметров циклов.

Элементы списка ввода-вывода обрабатываются последовательно слева направо. В одном списке могут содержаться элементы разных типов.

В первом из приведенных примеров последовательно перечисляются идентификаторы X, Y, K, которые могут быть как наименованиями простых переменных, так и массивов, элемент массива N с индексом J, элемент массива N с индексами 3 и 5.

Во втором примере третьим элементом списка является неявный цикл для элементов массива Z с индексами от 1 до 20.

В третьем примере перечисляются элементы двумерного массива по столбцам (первым меняется первый индекс), а в четвертом примере эти же элементы перечисляются по строкам (первым изменяется второй индекс J).

В пятом примере попарно перечисляются элементы массивов X и Y: X(1), Y(1), X(2), Y(2), ..., а в последнем примере указан элемент списка, который используется обычно в операторе вывода. В неявном цикле здесь включена простая переменная, являющаяся параметром цикла. В этом списке перечисляются попарно параметр цикла и соответствующий ему элемент массива: 1, P(1), 2, P(2), ...

**4. Операторы управления файлами.** Операторы *управления файлами* используются при работе с магнитными лентами и дисками. К этим операторам относятся: оператор подвода файла, оператор возврата, оператор конца файла и оператор поиска записи файла. Первые три из них применяются для файлов на ленте и для файлов последовательного доступа на диске, последний — только для файлов прямого доступа на диске.

Оператор *подвода файла* имеет вид

REWIND N1

где N1 — номер файла. Этот оператор устанавливает файл с номером N1 в начальное состояние, т. е. подготавливает первую запись этого файла к работе с операторами ввода и вывода.

Оператор *возврата* записывается в виде

BACKSPACE N1

где N1 — номер файла. Этот оператор при своем выполнении осуществляет переход от текущей записи файла с номером N1 к его предыдущей записи, т. е. к обработке подводится предыдущая, участвовавшая в работе запись. Если текущей записью при этом была первая запись файла, то оператор возврата не выполняет никакого действия. Например, если магнитная лента уже установлена в начальное положение, ни оператор REWIND, ни оператор BACKSPACE не выполняются.

Оператор конца файла имеет вид

END FILE N1

и записывает признак (метку) конца файла с номером N1. Выполнение этого оператора заключается в преобразовании текущей записи файла в его конечную запись, т. е. файл с номером N1 заканчивается данной текущей записью.

Оператор *поиска записи файла* имеет вид

FIND (N1'R)

где N1 — номер файла, R — номер записи этого файла. В результате выполнения оператора FIND запись файла N1 с номером R подготавливается к работе с операторами ввода и вывода, т. е. устанавливается в положение, когда обработка этой записи осуществляется за минимальное время.

**5. Операторы ввода без формата.** Ввод без формата (или бесформатный ввод) используется тогда, когда записи файла содержат данные, представленные в двоичной форме, т. е. преобразование представления информации не требуется.

Оператор *ввода последовательного доступа без формата* имеет вид

READ (N1) L

где N1 — номер файла, L — список ввода. Этот оператор осуществляет ввод текущей записи файла с номером N1 в оперативную память, определенную списком L. Например, оператор

READ (4) X, Y

вводит из текущей записи файла с номером 4 в область оперативной памяти, определяемой переменными X и Y, два значения.

Если в рассматриваемом операторе отсутствует список ввода, т. е. оператор записан в форме

READ (N1)

то действие оператора заключается в переходе от текущей записи файла к следующей его записи. Эта разновидность оператора ввода последовательного доступа без формата употребляется в том случае, когда считываемая информация не используется, но есть необходимость перехода к новой записи (например, можно перемотать участок магнитной ленты, не снимая с него информацию).

Оператор ввода прямого доступа без формата имеет вид

READ (N1'R) L

где N1 — номер файла, R — номер записи этого файла, L — список ввода. Под R понимается выражение, задающее относительный номер или местоположение требуемой записи (разумеется, в пределах максимального числа записей M, определенного оператором DEFINE FILE).

Выполнение этого оператора сводится к вводу записи с номером R файла N1 в область оперативной памяти, которая определена списком ввода L. Например, оператор

READ (6'7) X, Y

вводит из записи номер 7 файла с номером 6 в область оперативной памяти, определяемую переменными X и Y, два значения.

**6. Операторы вывода без формата.** Промежуточный вывод данных на ленты и диски в тех случаях, когда выведенные данные через некоторое время вновь будут при выполнении программы использованы в оперативной памяти, целесообразно организовать без преобразования формы представления. Для этих целей используются два оператора: оператор *вывода последовательного доступа без формата* и оператор *вывода прямого доступа без формата*.

Первый из них имеет вид

WRITE (N1) L

где N1 — номер файла, L — список вывода. Этот оператор осуществляет выборку данных из оперативной памяти, определенных списком L, и, не изменяя формы представления, образует текущую запись файла, который имеет номер N1. Например, оператор

WRITE (5) A, B

есть оператор последовательного доступа без формата, который передает значения переменных A и B в файл номер 5 без преобразования, формируя очередную запись этого файла.

Оператор вывода прямого доступа без формата имеет вид

WRITE (N1'R) L

где N1 — номер файла, R — номер записи этого файла, L — список вывода. Этот оператор выводит данные, определенные списком L, из оперативной памяти и образует запись номер R файла с номером N1. Например, оператор

WRITE (7'9) A, B

выводит значения переменных А и В в запись номер 9 файла с номером 7 без преобразования.

**7. Операторы ввода с форматом.** Выполнение операторов ввода и вывода с форматом управляется оператором задания формата (§ 22), указывающего вид и структуру передаваемых величин.

Оператор *ввода последовательного доступа с форматом* обычно имеет вид

```
READ (N1, N2) L
```

где N1 — номер файла, N2 — метка оператора FORMAT, L — список ввода.

С помощью этого оператора последовательно вводятся записи файла с номером N1 в те места оперативной памяти, которые определяются элементами списка L. При этом информация преобразуется в соответствии со списком спецификаций оператора FORMAT с меткой N2. Оператор задания формата с меткой N2 указывает также количество вводимых записей и их структуру.

Пример 1. Оператор

```
READ (N1, N2) A, B, C, M
```

с файла, имеющего номер N1, например с перфокарт, вводит четыре числа; значение первого из них присваивается переменной А, второго — В и т. д.

В элементе списка, заданном неявным циклом, на месте параметров циклов (нижней границы, верхней границы и шага) могут стоять целые переменные, значения которых задаются самим оператором READ.

Пример 2. Оператор

```
READ (N1, N2) (X(I), I=K, 200, 4)
```

сначала вводит значение K, затем элемент массива X(K), далее элемент X(K+4) и т. д.

Пример 3. Оператор

```
READ (N1, N2) (X(I), Y(I), I=1, 6)
```

вводит элементы двух массивов X и Y в такой последовательности: X(1), Y(1), X(2), Y(2), X(3), ...

Если же структура файла с номером N1 такова, что в нем расположены сначала элементы одного массива, затем другого, то соответствующий оператор имеет вид

```
READ (N1, N2) (X(I), I=1, 6), Y(I), I=1, 6)
```

Оператор ввода с форматом может быть записан в виде

```
READ (N1, N2, END=N3, ERR=N4) L
```

где N3, N4 — метки операторов; параметры END=N3 и ERR=N4 могут даваться в любом порядке и любой из них (либо оба) может быть опущен; N3 — метка оператора, на который передается управление после окончания ввода, если нет совпадения по количеству переменных в списке L и констант на вводе; N4 — метка оператора, получающего управление, если при вводе будет обнаружена ошибка. Например, в операторе ввода

```
READ (5, 71, ERR=101, END=26) A, B, C, D
```

после окончания ввода данных файла с номером 5 в соответствии с оператором FORMAT, имеющим метку 71, предусмотрена передача управления на оператор с меткой 26. Если при вводе будет зафиксирована ошибка, управление передается на оператор с меткой 101.

Поскольку совокупность перфокарт представляет собой файл, где записью является каждая отдельная перфокарта, при вводе перфокарт номер соответствующего файла есть логический номер устройства ввода с перфокарт и в этом случае оператор ввода может быть записан проще:

READ N2, L

где N2 и L имеют тот же смысл, что и в общем случае. Например, запись

READ 18, (A(I), B(I), C(I), I=1, 10)

представляет собой оператор ввода с перфокарт значений элементов массивов A, B и C в соответствии с оператором задания формата с меткой 18.

Оператор ввода может употребляться и без списка ввода в виде

READ (N1, N2)

где N1 — номер файла, N2 — метка оператора FORMAT. Эта разновидность оператора ввода используется тогда, когда вся считываемая информация идет для пополнения текста, заданного в самом операторе FORMAT с меткой N2.

Оператор *ввода прямого доступа с форматом* имеет вид

READ (N1'R, N2) L

где N1 — номер файла, R — номер записи этого файла, N2 — метка оператора задания формата, L — список ввода. Этот оператор осуществляет чтение данных из одной или нескольких записей файла с номером N1, начиная с записи, имеющей номер R, преобразование их в двоичную форму представления и размещение в областях оперативной памяти, заданных списком ввода L. Информация о количестве считываемых записей и о структуре записей содержится в операторе FORMAT с меткой N2. Например, оператор

READ (9'J, 12) S

обеспечивает чтение значения переменной S из записи J файла с номером 9 согласно формату, заданному в операторе с меткой 12. Значение J к моменту выполнения оператора READ должно быть определено.

**8. Операторы вывода с форматом.** Различают операторы вывода последовательного доступа с форматом и операторы вывода прямого доступа с форматом.

Оператор *вывода последовательного доступа с форматом* в общем случае записывается в виде

WRITE (N1, N2) L

где N1 — номер файла, N2 — метка оператора задания формата, L — список вывода. Этот оператор осуществляет последовательный вывод данных из участков оперативной памяти, определенных списком L. Преобразование

данных и их расположение в файле с номером N1 выполняется в соответствии со спецификациями оператора FORMAT, имеющего метку N2. Например, оператор

```
WRITE (1, 22) I, J, A, B, C, (X(N), N=1, 10)
```

выводит 15 величин: два целых числа, три действительных числа и 10 значений элементов массива X. Если номер файла 1 есть номер файла печати (т. е. логический номер АЦПУ), то указанные данные будут напечатаны (с предварительным преобразованием в десятичную форму представления). Количество записей (строк) и структура строк определяются оператором задания формата с меткой 22.

Оператор

```
WRITE (1, N2) (I, A(I), I=1, 100)
```

выводит элементы одномерного массива A с соответствующими номерами, т. е. последовательность значений: 1, A(1), 2, A(2), 3, A(3), 4, ...

Вывод через печатающее устройство может быть осуществлен также оператором

```
PRINT N2, L
```

Например, оператор печати элементов массива A со своими номерами будет выглядеть так:

```
PRINT N2, (I, A(I), I=1, 100)
```

Для передачи информации на выходной перфоратор служит оператор PUNCH N2, L

Если вся выводимая информация содержится в операторе FORMAT, то вывод осуществляется оператором WRITE без списка вывода

```
WRITE (N1, N2)
```

Смысл обозначений N2, L в операторах PRINT и PUNCH и обозначений N1, N2 в последнем операторе WRITE совпадает с принятым ранее.

Оператор *вывода прямого доступа с форматом* имеет вид

```
WRITE (N1'R, N2) L
```

где, как и выше, N1 — номер файла, R — номер записи, N2 — метка оператора задания формата, L — список вывода.

Этот оператор осуществляет выборку данных из областей оперативной памяти, заданных списком вывода L, преобразование и размещение их в одной или нескольких записях файла N1, начиная с записи с номером R. Информация о количестве записей и о форме представления данных в записях содержится в операторе задания формата с меткой N2.

**Пример 4.** Пусть задан двумерный массив X(10, 30). Требуется записать его на магнитную ленту так, чтобы каждая строка массива образовала отдельную запись.

Для выполнения указанной операции можно воспользоваться операторами

```
DO 1 I=1, 10
1 WRITE(7) (X(I, J), J = 1, 30)
```

где принято, что файл на магнитной ленте имеет номер 7.

Чтобы прочитать записанную информацию, необходимо установить магнитную ленту в начало или же осуществить возврат ленты назад на определенное количество записей. Так, если требуется прочитать значения элементов шестой строки массива и присвоить их элементам некоторого другого массива Y(30), то эти действия могут выполнить операторы

```
DO 3 I=1, 5
3 BACKSPACE 7
READ (7) (Y(I), I=1, 30)
```

где первые два оператора обеспечивают необходимый возврат на пять записей.

Возможен другой путь решения этой задачи. Так, запись строк массива X(10, 30) можно произвести вместе с соответствующими номерами строк:

```
DO 1 I=1, 10
1 WRITE (7) I, (X(I, J), J=1, 30)
REWIND 7
```

Запись теперь представляет собой набор значений, первое из которых есть номер строки, остальные — элементы данной строки. Последний оператор этой группы устанавливает магнитную ленту в начальное положение. Чтение шестой строки матрицы в таком варианте выполняется операторами

```
4 READ (7) I
IF (I.NE.6) GO TO 4
BACKSPACE 7
READ (7) I, (Y(J), J=1, 30)
```

в которых после считывания первого элемента I очередной строки его значение сравнивается с номером разыскиваемой (шестой) строки. Если прочитанное значение не совпадает с номером необходимой строки (т. е. I не равно 6), то нужно перейти к чтению первого элемента следующей записи (строки). Если же значение I совпадает с заданным номером, то необходимо считывать остальные элементы строки, которые составят массив данных Y(30). При этом предварительно необходимо выполнить оператор возврата BACKSPACE 7, так как после считывания *любой* текущей записи или ее части к операции считывания будет подготовлено начало следующей записи (к считывающим головкам подводится начало следующей записи).

#### Упражнения

1. Описать три файла с номерами 3, 13 и 23 емкостью до 62 записей в каждом файле для произвольных записей длиной до 6 слов.
2. Составить список ввода-вывода для передачи значений элементов двух массивов A(20, 40) и B(40, 20) в такой последовательности: первая строка массива A, первый столбец массива B, вторая строка массива A, второй столбец

массива В и т. д. с указанием каждый раз перед обработкой очередной строки или очередного столбца соответствующего номера строки или столбца.

3. Составить программу вывода значений элементов массива М(16) в файл 5 последовательного доступа из пяти записей, так, чтобы эти записи содержали соответственно элементы массива с номерами:

```
1, 3, 5, 7, 9, 11, 13, 15
2, 4, 6, 8, 10, 12, 14, 16
1, 2, 4, 5, 7, 8, 10, 11, 13, 14
3, 6, 9, 12, 15
1, 2, 5, 6, 9, 10, 13, 14
```

4. Первая из 25 записей файла прямого доступа с номером 6 содержит числа:

```
1.8, -3.7, 5.2, 6.7, -1.7, -0.8, 2.0;
```

вторая запись состоит из чисел:

```
4.7, -3.1, 9.2, -8.0, -0.8.
```

Эти числа имеют стандартную длину, относятся к действительному типу и представлены в двоичной форме. Определить значения переменных X, Y, A, B, C, которые они получают в результате выполнения следующей программы:

```
REAL A, B, C, X, Y
DEFINE FILE 6(25, 120, L, K)
READ (6'2) A, B, C, X, Y
READ (6'1) X, Y
STOP
END
```

## § 21. Оператор формирования списков

Операторами ввода и вывода последовательного доступа могут быть обработаны значения без перечисления их имен в списке ввода-вывода, что создает определенные удобства для оперативной смены значений переменных, используемых в программе. В этом случае переменные и массивы должны быть описаны неисполняемым оператором *формирования списков*, имеющим вид

```
NAMELIST P
```

где P — список блоков наименований. Каждый блок наименований в списке имеет вид

```
/X/ A1, A2, ..., AN
```

Здесь X — идентификатор блока; A1, A2, ..., AN — переменные или наименования массивов. Разделителем двух блоков является первый из символов /, которыми окаймляется наименование следующего блока.

Наименование блока может содержаться только в одном операторе NAMELIST и не должно появляться где-либо в программе, кроме как в операторах ввода и вывода, в которых отсутствует список. Идентификатор переменной или массива может принадлежать одному или нескольким блокам.

В операторах READ и WRITE не задается ни список ввода-вывода, ни метка оператора FORMAT. Вместо этого указывается идентификатор блока

наименований оператора NAMELIST, т. е. операторы ввода и вывода в этом случае соответственно имеют вид

```
READ (N1, X)
WRITE (N1, X)
```

где N1 — номер файла, X — идентификатор блока.

Во вводном файле (например, на перфокартах) записываются наименования переменных и их значения в форме

```
A1=K
```

где A1 — простая переменная или элемент массива, K — значение (константа). Форма записи массивов следующая:

```
A2=J1* K1, J2* K2, ...
```

где A2 — идентификатор массива, K1, K2, ... — значения (константы), J1, J2 — целые константы (коэффициенты кратности), указывающие, сколько раз должно быть повторено следующее за коэффициентом кратности значение. Коэффициенты кратности могут и отсутствовать.

Файл начинается с записи &X, где X — идентификатор блока, и заканчивается записью &END; при этом первая позиция (первая колонка на перфокарте) игнорируется, т. е. символ & должен содержаться во второй позиции.

Между этими записями располагаются записи, в которых задаются значения переменных и массивов указанным выше способом. Количество элементов в записях может быть любым, в частности, не обязательно, чтобы все элементы блока наименований были перечислены и получили значения, но все элементы записей должны принадлежать заданному блоку наименований. Между отдельными элементами ставится разделитель , (запятая). Каждая запись может содержать одну или несколько переменных, в записи допускаются пробелы. Порядок следования переменных в записях (на перфокартах) несуществен.

Пример. Пусть в программе содержатся операторы

```
DIMENSION S(5)
NAMELIST /N/ A, S, M
```

а в устройство ввода заложен массив перфокарт, содержащий следующую информацию:

```
&N
M=5, A=3.45, S=1.1, 2* 1.5, 1.9, 2.1
&END
```

Здесь каждая строка соответствует содержимому одной перфокарты. Тогда оператор

```
READ (3, N)
```

осуществит ввод (если 3 — номер устройства ввода перфокарт) с указанных перфокарт семи значений и присвоение этих значений элементам блока /N/. Все данные вводятся в таком виде (формате), какой был задан на вводном

устройстве. При выводе данных сохраняется тот же самый основной формат, что они имели при вводе.

В выводной файл включаются идентификаторы и значения всех переменных и элементов массивов, принадлежащих заданному в операторе WRITE блоку. Порядок следования их при выводе соответствует порядку следования наименований в блоке. Так, если в процессе работы программы значения элементов блока /N/ из предыдущего примера не изменились, то оператор вывода

```
WRITE (1, N)
```

сформирует выводной файл с номером 1 в виде

```
—&N
—A=3.45, S=1.1, 1.5, 1.5, 1.9, 2.1, M=5
—&END
```

### Упражнения

1. Написать программу, обеспечивающую ввод значений элементов матриц  $A(3, 3)$  и  $B(3, 3)$ , вычисление элементов матрицы  $C(3, 3)$ , являющейся суммой матриц  $A$  и  $B$  (элементы матрицы  $C$  получаются по алгоритму  $c_{ij} = a_{ij} + b_{ij}$ ), и вывод значений элементов матрицы  $C$ , если значения элементов матрицы  $A$  набиты на одной перфокарте следующим образом:

A=37., 81., 16., 25., 44., 76., 59., 62., 98.

а значения элементов матрицы  $B$  — на другой перфокарте:

B=77.2, 2\* 66., 99., 4\* 44., 55.

2. На перфокарте набита следующая информация:

X=5, Y=2, Z=4, B=2.5, A=1.5, C=0.5

Составить операторы, с помощью которых на выводном устройстве может быть получена перфокарта, содержащая следующую информацию:

A=1.5, B=2.5, C=0.5, X=5, Y=2, Z=4

Какую информацию следует предварительно ввести в машину?

## § 22. Оператор задания формата

Обычно операторы ввода и вывода используются совместно с оператором *задания формата*, который обеспечивает необходимые преобразования и редактирование при обмене информацией между оперативной памятью и внешними устройствами.

Оператор задания формата имеет вид

```
N2 FORMAT (S)
```

где N2 — метка оператора (эта метка указывается в соответствующем операторе ввода или вывода), S — список спецификаций.

Оператор FORMAT является неисполняемым, он может располагаться в любом месте программы.

1. **Список спецификаций.** *Список спецификаций S* состоит из кодов (спецификаций) оператора FORMAT, разделяемых запятыми или символами / (наклонная черта). Так, список S может иметь вид

```
S1, S2, .../..., N(SM, ...), JSQ, .../...
```

где  $S_1, S_2, \dots$  — коды оператора FORMAT;  $N, J$  — коэффициенты кратности (целые константы без знака), указывающие число повторений следующих за ними повторяемых групп спецификаций. Например,  $3SQ$  означает  $SQ, SQ, SQ$ , а конструкция  $2(SM, SN)$  указывает, что группа спецификаций  $SM, SN$  повторяется дважды, т. е. имеем  $SM, SN, SM, SN$ .

Внутри одной пары скобок, составляющих повторяемую группу спецификаций первого уровня, могут в свою очередь содержаться повторяемые группы спецификаций, которые в этом случае составляют повторяемые группы спецификаций второго уровня. Более высокие уровни повторяемых групп спецификаций не допускаются.

Элементы в списке ввода-вывода оператора, содержащего метку оператора задания формата, преобразуются из внешнего (относительно оперативной памяти) представления во внутреннее или из внутреннего представления во внешнее согласно кодам (спецификациям) в списке  $S$ . Символ / в списке означает начало новой записи при обработке списка ввода-вывода.

В общем случае спецификация может быть представлена в виде

$Cw.d$

где  $C$  — индекс спецификации,  $w$  и  $d$  — целые константы без знака.

Константа  $w$  указывает длину поля, отведенного для данного значения в записи, т. е. количество символов в значении; константа  $d$  — это количество знаков в дробной части числовых значений, т. е. количество десятичных цифр справа от точки внутри поля (за исключением спецификации  $G$ ).

Все спецификации по своему назначению могут быть разбиты на две группы: к первой группе относятся спецификации, предназначенные для описания структуры значений вводимых или выводимых величин; ко второй группе относятся спецификации, предназначенные для управления вводом и выводом или для редактирования. Индексами спецификаций первой группы являются символы

$F, E, D, I, L, G, Z, A,$

которые называются индексами спецификаций преобразования. Индексы спецификаций второй группы — это символы

$, H, X, T,$

которые называются индексами управляющих редакционных спецификаций.

Индекс спецификации обязан содержаться в коде, константы  $w$  и  $d$ , а также точка в некоторых спецификациях отсутствуют.

Установление соответствия между переменными в списке ввода-вывода и спецификациями в операторе FORMAT осуществляется с учетом управляющих редакционных спецификаций согласно их следованию в списке слева направо.

Каждой спецификации преобразования, содержащейся в списке спецификаций, соответствует один элемент в списке ввода-вывода. Исключение составляет элемент комплексного типа, которому в списке спецификаций соответствуют два кода оператора FORMAT с индексами спецификаций  $F, E$  или  $D$ .

Для редакционных управляющих спецификаций не существует соответствующего элемента в списке ввода-вывода. Эти спецификации предназначены

для управления вводом и выводом или для редактирования записей соответствующего файла.

Таким образом, выполнение операторов ввода или вывода вместе с соответствующим оператором FORMAT начинается с перебора спецификаций слева направо с учетом коэффициентов кратности и скобок. Если выбранная очередная спецификация относится к первой группе, то из списка ввода-вывода выбирается соответствующий элемент. Значение этого элемента (переменной) преобразуется согласно спецификации и передается либо в поле записи (при выводе), либо в оперативную память машины (при вводе). Если выбранная очередная спецификация относится ко второй группе, то выполняются необходимые (редакционные) действия без обращения к списку ввода-вывода.

Если список ввода-вывода будет исчерпан раньше, чем список спецификаций, то перебор спецификаций прекратится либо при обнаружении в списке спецификаций элемента первой группы, либо при исчерпании списка спецификаций. После этого выполнение оператора ввода или вывода будет закончено.

Если список спецификаций будет исчерпан раньше, чем список ввода-вывода, то перебор элементов списка спецификаций будет повторен, начиная с первого элемента последней группы спецификаций первого уровня, ннымн словам — с самой правой открывающей скобки первого уровня. Повторный перебор элементов списка спецификаций, не содержащего повторяемых групп спецификаций, начинается с первого элемента.

Переход на повторный перебор списка спецификаций сопровождается переходом к новой (следующей по порядку) записи файла. Переход к новой записи происходит и тогда, когда в процессе перебора элементов списка спецификаций встречается символ / (наклонная черта).

Действие оператора задания формата в последующих примерах рассматривается совместно с операторами ввода или вывода. Поэтому каждая спецификация преобразования будет рассмотрена как при вводе, так и при выводе. Для большей наглядности примеров в дальнейшем предполагается, что ввод осуществляется с перфокарт, а выводится информация на АЦПУ, т. е. номер файла в примерах — это либо логический номер устройства ввода с перфокарт, либо логический номер печатающего устройства.

**2. Спецификации преобразования.** F-спецификация служит для описания формы представления значений переменных действительного типа и имеет вид

NFw.d

где коэффициент кратности N может отсутствовать.

В в о д. Преобразование, указываемое спецификацией F, заносит число, содержащее w символов, на соответствующее поле оперативной памяти. При этом производится перекодировка числа из десятичного представления в двоичное. Например, оператор

N2 FORMAT (5F8.3)

указывает, что с одной перфокарты вводится 5 чисел, каждое из которых занимает на перфокарте поле, состоящее из 8 колонок. Если точка на перфо-

карте не набита или набита перед третьей цифрой, считая справа, то, согласно спецификации F, она будет проставлена после ввода именно перед третьим символом, считая справа. Если точка набита на поле из  $w$  символов в любой другой позиции, то задание точки F-спецификацией независимо от значения  $d$  отменяется, и вводимое число содержит точку в указанной на перфокарте позиции. Например, число 1.37296E5 может быть введено при любом  $d$ , но всегда будет записано в память машины как  $1.37296 \times 10^5$ .

Если в спецификации значение  $d$  отсутствует, то оно принимается равным нулю.

Индекс спецификации F указывает, в какой форме будет записано данное действительное число в памяти машины, и не накладывает никаких ограничений на форму представления этого числа в вводимой записи (на перфокарте), за исключением одного — с поля ввода считывается  $w$  символов. Например, оператор

### N2 FORMAT (F5.3, F6.2, F7.4)

совместно с оператором ввода предполагает ввод с одной перфокарты трех чисел соответственно с полей длиной в 5, 6 и 7 символов. Если нет соответствия между указанным форматом и длиной полей на перфокарте, возможны ошибки при вводе.

С одной перфокарты считывается только такое количество символов, которое указано в коде для данной единицы записи, т. е.  $N \times w$  или  $\Sigma w$  символов ( $N$  — коэффициент кратности). Содержимое остальных колонок не воспринимается. Так, в предпоследнем примере считывается информация с  $5 \times 8 = 40$  колонок, а в последнем примере — с  $5 + 6 + 7 = 18$  колонок.

Если необходимо ввести массив данных, который не помещается на одной перфокарте, оператор FORMAT, написанный для одной записи, используется повторно, пока не будет введен весь массив. Например,

```
READ (3, 2) (X(I), I=1, 100)
2 FORMAT (16F5.1)
```

Здесь с одной перфокарты (если 3 — логический номер устройства ввода перфокарт) вводится 16 чисел, каждое из 5 символов. Всего оператор READ осуществляет ввод содержимого  $100/16 + 1 = 6 + 1 = 7$  перфокарт. При этом с последней перфокарты будет прочитана только информация с  $4 \times 5 = 20$  колонок, т. е. четыре числа, хотя набито может быть и больше.

Примеры ввода при F-спецификации:

— с поля, на котором записано число 763.9325, при спецификации F8.4 запишется результат в виде 763.9325; здесь в поле присутствуют дробная и целая части и точка;

— с поля 25793 при спецификации F5.7 запишется в память машины число .0025793, так как в поле нет дробной части и точки и введенное число преобразуется к виду  $25793 \times 10^{-7}$ ;

— с поля .43625 при спецификации F6.d ( $d$  — любое) запишется .43625, так как в поле содержится точка, и поэтому значение  $d$  не оказывает влияния; целой части в поле нет;

— с поля +245.15E—03 при спецификации F11.2 запишется результат .24515; здесь положение точки определяется заданием ее десятичным порядком на перфокарте.

Вывод. Поле, на которое выводится число, занимает  $w$  позиций в записи (например, в строке). Значение величины заносится в запись в виде десятичного числа со знаком с фиксированной точкой и  $d$  дробными и максимум  $w - d - 2$  целыми разрядами.

Если число положительное, то знак  $+$  не печатается. Если длина поля недостаточна для помещения туда выводимого числа, то оно заполняется символами  $*$  (звездочка). Если поле содержит больше позиций, чем требуется для записи числа, то избыточные позиции слева заполняются пробелами.

Пример 1. Операторы

```
PRINT 5, X, Y, Z
5 FORMAT (F8.5)
```

осуществляют печать трех строк. В каждой строке содержится одно число: в первой — значение  $X$ , во второй —  $Y$ , в третьей —  $Z$ . Каждое число занимает поле из восьми позиций, из них одна позиция отводится под точку, одна — под знак, пять — под дробные десятичные разряды.

Пример 2. Операторы

```
WRITE (1, 1) (X(I), I=1, 1000)
1 FORMAT (5F9.5)
```

выводят на печать (если 1 — номер АЦПУ) 200 строк, в каждой по 5 значений элементов массива  $X$ .

Максимальное число выводимых в запись символов ( $N \times w$  или  $\sum w$ ) не должно превышать допустимое число символов записи для данного файла.

Пример 3. Пусть значение переменной  $A$  равно +12.475; операторы

```
PRINT 2, A
2 FORMAT (F8.3)
```

вызовут печатание результата в виде — —12.475, в котором слева содержится два пробела, а знак  $+$  не печатается.

Пример 4. Значение переменной  $A$  равно —12.475; в результате действия операторов

```
PRINT 5, A
5 FORMAT (F6.3)
```

будет напечатано \*\*\*\*\* (шесть звездочек), так как на поле длиной в шесть символов отсутствует место для знака минус.

Е-спецификация, как и F-спецификация, служит для ввода и вывода действительных значений. Код ее имеет вид

```
NEw.d
```

Ввод. Сохраняются все правила, перечисленные при рассмотрении F-спецификации. Число, записанное на поле (на перфокарте), не обязательно содержать

все части, важно лишь, чтобы занимаемое числом поле не превосходило значения  $w$  в коде E.

Примеры ввода при E-спецификации:

— с поля  $+126.43E-03$  при спецификации E11.2 в память запишется значение .12643; здесь в поле присутствуют все элементы записи числа;

— с поля  $-12.437629E+1$  при спецификации E13.6 запишется значение, равное  $-124.37629$ ;

— с поля  $3698E+04$  при спецификации E9.10 запишется значение .003698; здесь в поле нет дробной части, поэтому введенное число будет  $3698 \times 10^{+4} \times 10^{-10}$  ( $d$  равно 10);

— с поля  $127.256$  при спецификации E7.3 введется значение 127.256, хотя в поле и нет десятичного порядка;

— с поля  $-.0002736E5$  при спецификации E11.7 введется значение, равное  $-27.36$ ; здесь целая часть в поле состоит только из знака минус;

— если поле содержит только пробелы, то при любой спецификации Ew.d введется значение 0;

— с поля  $1E1$  при спецификации E3.0 ( $d$  равно 0) введется значение 10. ( $1 \times 10^{+1} \times 10^{-0}$ );

— с поля, содержащего только десятичный порядок, независимо от его значения и значения  $d$  записывается всегда нуль; так, поле  $E+07$  при любой спецификации интерпретируется как 0.

Вывод. Поле в записи состоит из  $w$  позиций. Выводимое число представляется в виде

$$\pm 0.\underbrace{aa\dots a}_d E \pm aa$$

$d$  позиций

где знак плюс обычно заменяется пробелом, разряды мантиссы после точки содержат  $d$  позиций; буквой  $a$  обозначены позиции, занимаемые цифрами.

Длина поля должна быть достаточной, чтобы в нем уместились значащие цифры, знаки, точка и десятичный порядок, поэтому  $w$  должно быть больше или равно  $d+7$ . При недостаточной длине поле заполняется звездочками. Если размер поля больше количества символов в выводимом числе, то левые избыточные позиции поля заполняются пробелами. Например, если выводятся два элемента массива  $X$  со значениями  $-16.245$  и  $1052.61$  соответственно с помощью операторов:

```
WRITE (1, 1) (X(I), I=1, 2)
1 FORMAT (F10.4)
```

то будут напечатаны две строки:

```
— — —16.2450
— 1 0 52.6100
```

За исключением значащих цифр в младших разрядах после точки печатаются нули. Если бы был задан код F9.4, на месте второго числа (во второй строке) напечатались бы звездочки, так как отсутствует позиция для знака, хотя он и не печатается.

Если организовать вывод операторами

```
WRITE (1, 3) (X(I), I=1, 2)
3 FORMAT (E13.5)
```

то будет напечатано:

```
— —0.16245E+02
— —0.10526E+04
```

Хотя длины поля достаточно для вывода указанных значений (перед первым числом остается один, перед вторым — два пробела), во втором числе потерялся один младший разряд, так как значение  $d$  равно в коде 5, а число значащих цифр равно 6.

*D-спецификация* определяет преобразование, которое применяется в случае значений с двойной точностью. Код ее имеет вид

```
NDw.d
```

Правила использования *D-спецификации* полностью совпадают с правилами, рассмотренными для *E-спецификации*, за исключением того, что при выводе на поле вывода символ *E* заменяется символом *D*. Элементы списка ввода-вывода соответствующего оператора ввода или вывода должны иметь двойную точность.

*I-спецификация* служит для описания формы представления значений переменных целого типа и имеет вид

```
NIw
```

Эта спецификация используется только для ввода и вывода целых десятичных чисел, поэтому соответствующий элемент списка ввода-вывода должен быть целой величиной.

В *во*д. В поле могут присутствовать цифры и пробелы, а также знаки  $+$  и  $-$ , которые должны предшествовать первой цифре. Например, операторы

```
READ 5, K, L, M
5 FORMAT (I6, 2I10)
```

считывают с перфокарты содержимое трех полей: одного — длиной в 6 колонок и двух следующих — по 10 колонок в каждом.

В *во*д. Целое число занимает поле из  $w$  позиций, причем одна позиция всегда отводится для знака. Если в поле вывода позиций больше, чем требуется для записи числа, избыточные позиции заполняются пробелами слева; если длина поля недостаточна для записи, все поле заполняется символами  $*$ . Например, если в результате вычислений получены значения  $I, J, M$ , равные соответственно  $+17356, -376, +2180$ , то после вывода операторами

```
PRINT 8, I, J, M
8 FORMAT (I10, I5, I5)
```

будет напечатано:

```
— — — — — 1 7 3 5 6 — — 376 — 2180
↑           ↑   ↑   ↑
1           10  15  20 позиции
```

*L-спецификация* используется при вводе и выводе логических значений. Она записывается в виде

NLw

В в о д. С перфокарты вводится содержимое поля из w символов. Если на этом поле первый, отличный от пробела символ есть буква T, то в область оперативной памяти, определяемой соответствующим элементом логического типа списка ввода, заносится значение .TRUE. если же первый символ есть буква F или все w позиций заняты пробелами, в память заносится логическое значение .FALSE. Таким образом, в поле логического значения при вводе должно присутствовать последовательность символов, в которой в качестве первого символа, отличного от пробела, записана буква T или F (или все символы на поле — пробелы).

В ы в о д. Поле вывода имеет длину w символов, где в последней позиции печатается символ T или F в зависимости от того, имеет ли соответствующий элемент (логического типа) списка вывода значение соответственно .TRUE. или .FALSE. Например, в результате выполнения операторов

```
WRITE (1, 4) A, B, C, D, E
4 FORMAT (5L3)
```

будет напечатана строка

— T — T — F — T — F

если логические переменные A, B и D имели значение .TRUE. а C, E — значение .FALSE.

*G-спецификация* используется при передаче целых, действительных, комплексных и логических значений. Эта спецификация записывается в виде

NGw.d

Поле, занимаемое значением, состоит из w позиций.

В в о д. Для целых и логических значений G-спецификация эквивалентна соответственно I- и L-спецификациям, т.е. вместо Iw и Lw может стоять в списке спецификаций конструкция Gw. При вводе действительных значений спецификация Gw.d эквивалентна коду Fw.d. Смысл константы d тот же, что и выше. В случае целых и логических значений наличие d в G-спецификации игнорируется.

В ы в о д. Если десятичный порядок p преобразуемого действительного числа, представленного в форме E с нормализованной мантиссой, меньше нуля или больше d, то значение выводится в форме E. Если же  $0 \leq p \leq d$ , то число выводится в форме F с p цифрами в целой части и d—p цифрами в дробной части числа; при этом четыре правые позиции заполняются пробелами. Таким образом, при выводе константа d указывает количество значащих цифр в поле.

Например, используя спецификацию G10.3, исходное число  $0.555 \times 10^{-1}$  будет выведено в виде —0.555E—01, число  $0.555 \times 10^0$  имеет вид —0.555 — — —, число  $0.555 \times 10^3$  — вид — — —555. — — —, а число  $0.555 \times 10^4$  будет представлено как —0.555E + 04

Пример 5. Предположим, что имеются операторы

```
WRITE (1, 3) J, X, Y, D, C, L
3 FORMAT(G4, 2G9.2, G10.7, 2G5.2, G2)
```

а целая переменная J получила значение 126, действительные переменные X и Y имеют соответственно значения 236.71 и 11.86, D — переменная с двойной точностью со значением 3.1415972, C — комплексное число (1.2, 7.3), L — логическая переменная со значением .FALSE.

Результатом действия этих операторов будет печать строки

```
  126  0.23E+03  11.  3.141597  1.2  7.3  .F
```

Z-спецификация служит для задания шестнадцатеричных чисел, код ее имеет вид

NZw

В списке оператора ввода или вывода должны быть переменные целого или действительного типов или с двойной точностью. Считается, что значениями этих величин могут быть последовательности шестнадцатеричных цифр. Количество l шестнадцатеричных цифр в значении переменной определяется типом переменной из расчета по две цифры на один байт.

В в о д. Если  $w > l$ , то вводимое значение усекается слева, т. е. теряются левые (старшие) разряды. При  $w < l$  справа добавляется соответствующее количество  $(w - l)$  шестнадцатеричных нулей.

В в о д. При  $w > l$  в поле вывода слева добавляются пробелы. В случае  $w < l$  усекаются левые цифры.

A-спецификация используется для задания текстовых значений элементам списка ввода-вывода. Ее код имеет вид

NAw

В в о д. Все w символов с поля ввода, в том числе пробелы, если они есть, вводятся в оперативную память. Эти символы образуют текстовую константу, значение которой получает соответствующая переменная в списке ввода.

Количество символов l, используемых при образовании текстовой константы, зависит от того, какова разрядность процессора. Один символ занимает 1 байт памяти. Следовательно, для ЭВМ БЭСМ-6 максимальное значение l равно 6, для ЕС 1050 — 8. Обозначим это число буквой г. Если  $w \geq g$ , то при вводе по спецификации A в память заносится г символов, расположенных в правой части поля. Если  $w < g$ , то в памяти образуется значение переменной, состоящее из г введенных символов, за которыми (справа) расположены  $g - w$  пробелов. Например, с помощью операторов

```
READ (3, 2) X, Y, Z
2 FORMAT (A4, A8, A12)
```

переменные X, Y, Z, если вводятся следующие последовательности символов

```
TYPE PROGRAM FORTRAN TEXT
```

получают соответственно значения (при  $g = 8$ )

```
TYPE_____ PROGRAM_ RAN_ TEXT
```

Вывод. Если  $w > g$ , то в записи будут содержаться  $w-g$  пробелов и  $g$  символов. В противном случае (при  $w \leq g$ ) выводится  $w$  символов, расположенных в левой части слова в памяти.

Предполагая, что был осуществлен приведенный выше ввод текстовых значений переменных X, Y, Z, операторы

```
PRINT 3, X, Y, Z
3 FORMAT (A5, A10, A12)
```

выведут на печать следующее:

```
____TIP____PROGRAM____RAN_ TEXT
   X         Y         Z
```

Присвоение переменным и элементам массивов текстовых значений может быть выполнено, таким образом, операторами ввода с использованием в соответствующем операторе FORMAT A-спецификаций.

В языке фортран IV не предусмотрены специальные операторы для обработки текстовой информации, но переменные или элементы массива, которым присвоены текстовые значения, могут быть объединены в логическое выражение знаками операций отношения .EQ. и .NE., а также выступать в качестве правых частей операторов присваивания. Например, при соблюдении условий предыдущего примера операторы

```
IF (X.EQ.Y) GO TO 13
GO TO 27           и   IF (X.NE.Y) GO TO 27
```

осуществят одну и ту же передачу управления на оператор с меткой 27, а после выполнения оператора  $Z=Y$  переменная Z получит значение текстовой константы PROGRAM.

В одном операторе FORMAT список спецификаций может содержать коды различных видов. Например, возможен такой оператор:

```
N2 FORMAT (A5, F4.2, E8.1, 2I7, D19.12, L2)
```

Единственное требование, которое при этом должно выполняться, заключается в том, что каждый код оператора FORMAT обязан соответствовать типу «своей» переменной в списке оператора ввода или вывода.

Примеры использования оператора FORMAT с оператором ввода.

Пр и м е р 6. Оператор

```
N2 FORMAT (5F10.2/4F9.2)
```

указывает, что вводится перфокарта с пятью числами на ней (каждое поле занимает 10 колонок) в форме F10.2, после которой вводится другая перфокарта, с которой считываются четыре числа в форме F9.2. Если есть еще третья перфокарта, то информация с нее будет считана в форме F10.2 и т. д.

Пр и м е р 7. Оператор

```
N2 FORMAT (I5/(4F5.3))
```

указывает, что с первой перфокарты вводится одно число в форме I5, а со всех остальных, сколько бы перфокарт ни было, считается по четыре числа в форме F5.3

Таким образом, появление символа / в списке спецификаций означает переход к другой записи. Символ / может появляться не только между элементами в списке спецификаций, но и в конце списка. Допустимо написание подряд нескольких символов /. Два символа // означают при вводе пропуск одной перфокарты, а при выводе — одну строку пробелов и т. д.

Пример 8. Оператор

N2 FORMAT (2(F3.2, I6))

равносложен оператору

N2 FORMAT (F3.2, I6, F3.2, I6)

который вводит с одной перфокарты четыре числа.

В последнем примере реализована возможность образования повторяемой группы спецификаций. Повторяемые группы спецификаций, как уже отмечалось, могут быть образованы глубиной до двух уровней. Например:

N2 FORMAT (I5, (I4, F4.1, 4(E8.3, F6.2)))  
                   0      1                  2                  210

Скобки, помеченные нулем, соответствуют нулевому уровню, единицей — первому уровню, двойкой — второму уровню. Если в списке ввода-вывода содержится достаточно много элементов, то перебор спецификаций в списке приведенного оператора FORMAT будет повторен с открывающей скобки, помеченной единицей, т. е. со спецификацией I4.

Использование операторов FORMAT из примеров 6, 7 и 8 с оператором вывода соответственно указывает:

— печать двух строк, в первой из которых располагаются пять чисел в форме F10.2, во второй — четыре числа в форме F9.2;

— печать в первой строке числа в форме I5, а в остальных — по четыре числа в форме F5.3;

— печать строки из четырех чисел.

**3. Масштабный коэффициент.** Спецификациям F, E и D может предшествовать *масштабный коэффициент* в форме nP, где n — целая константа со знаком. Масштабный коэффициент употребляется со спецификацией F при вводе и выводе и со спецификациями E и D — только при выводе.

Использование масштабного коэффициента nP с F-спецификацией означает:

— при вводе вводимое число умножается на  $10^{-n}$ ; например, если число 123.456 вводится при спецификации 2PF7.3, то в память заносится значение  $123.456 \times 10^{-2}$ , т. е. 1.23456;

— при выводе значение из оперативной памяти умножается на  $10^n$  и результат выдается согласно спецификации Fw.d; например, если в памяти хранилось число 1.23456, то при спецификации —1PF9.4 выводится значение — — — 0.1234, а при спецификации 3PF10.4 результатом вывода будет значение — — — 1234.5600



В в о д. Из вводимой записи считываются символы до тех пор, пока в тексте не встретится апостроф или не будет заполнено все поле в операторе FORMAT новым текстом. Первоначально, т.е. при включении в программу оператора FORMAT с 'спецификацией, достаточно поле между апострофами заполнить пробелами. Например, ввод заголовка FORTRAN PROGRAM, набитого на перфокарте, в оператор FORMAT может быть осуществлен следующим образом:

```
READ 1
1 FORMAT ('_____')
```

В дальнейшем оператор с меткой 1 в программе имеет вид

```
1 FORMAT ('FORTRAN PROGRAM')
```

В ы в о д. В запись переносится содержимое списка спецификаций между апострофами. Результатом действия приведенных в начале п. 4 операторов будет печать строки

```
—X= — — —12.345AB°C
```

если переменная X имела значение —12.345.

Размер выводимого на печать текста не должен превышать длину строки. Если отдельный апостроф случайно оказался в выводимой информации, он ограничит поле вывода.

*N-спецификация* имеет вид

```
wN
```

(целая константа w пишется перед индексом N) и, так же как 'спецификация, используется при вводе или выводе текстовой информации, содержащейся в операторе FORMAT, однако в коде wN указывается количество символов текста, тогда как в апостроф-спецификации это не требовалось.

В в о д. В поле оператора FORMAT из w позиций, следующих за индексом N, заносится весь текст, содержащийся в одной записи. Например, если на перфокарте набито

```
TEST AND TEXT
```

то операторы

```
READ 8
8 FORMAT (14N_____)
```

преобразуют оператор с меткой 8 к виду

```
8 FORMAT (14NTEST AND TEXT_)
```

В ы в о д. Информация в количестве w символов, которая содержится в операторе FORMAT после индекса спецификации N, выводится на печатающее устройство. Так, использование приведенного оператора FORMAT с меткой 8, например, с оператором

```
PRINT 8
```

вызовет печать строки

```
TEST _ AND _ TEXT _
```

Операторы

```
PRINT 4, Q
```

```
4 FORMAT (6HINDEX=, F5.2)
```

выведут на печать строку

```
INDEX= _ 2.35
```

если переменная Q имела значение 2.35

При использовании H-спецификации список ввода-вывода отсутствует. В последнем примере элемент Q списка относится к F-спецификации. Константа перед индексом H ставится и в случае  $w = 1$

X-спецификация записывается в виде

```
wX
```

(константа w пишется перед индексом спецификации) и используется при вводе для пропуска w символов, а при выводе для включения в запись w пробелов. Например, оператор FORMAT в виде

```
11 FORMAT (10X, 110, 20X, 4I10)
```

совместно с оператором ввода

```
READ 11, I, J, K, L, M
```

указывает, что значение переменной I считывается с поля, занимающего позиции с 11 до 20, а значения J, K, L, M — начиная с позиции 41, т.е. три поля (по 10 колонок) на перфокарте не воспринимаются, а в результате выполнения операторов

```
PRINT 9, X, Y
```

```
9 FORMAT (F10.2, 4X, F8.3)
```

будут напечатаны значения переменных X и Y с интервалом между ними, равным четырем пробелам. Очевидно, оператор с меткой 9 эквивалентен оператору

```
9 FORMAT (F10.2, ' _ _ _ _ ', F8.3)
```

T-спецификация используется для задания позиции в записи, начиная с которой производится ввод или вывод, если есть необходимость считывания информации (при вводе) или занесения результата (при выводе) не с начала записи. Эта спецификация имеет вид

```
Tw
```

В одном операторе FORMAT T-спецификация может употребляться несколько раз в сочетании с любыми другими спецификациями. При этом нет необходимости в последовательном указании кодов в операторе FORMAT с возрастающими значениями констант w.

В в о д. Константа *w* равна номеру первой позиции в поле записи. Так, операторы

```
READ 1, S
1 FORMAT (T21, F10.2)
```

осуществляют ввод для переменной *S* значения, расположенного на перфокарте, начиная с колонки 21.

В ы в о д. Значение константы *w* совпадает с номером первой позиции поля, на которое переписывается информация, если только внешним носителем информации не является печатающее устройство. В последнем случае номер первой занятой позиции в напечатанной строке равен  $w-1$ . Например, при выводе

```
PRINT 2, X
2 FORMAT (T33, F8.2, T31, 'X=', T61, 'PRIME')
```

если полученное значение *X* равно, скажем, 135.41, будет напечатана строка

```
X=---135.41|-----|PRIME
  ↑  ↑                ↑
  30 32              60 позиции
```

Не следует допускать перекрытия полей в записи несколькими спецификациями. Так, ошибочным будет оператор

```
N2 FORMAT (T12, I15, T22, I6)
```

который при выводе должен отвести под число в форме I15 позиции с 11 по 25, а под число в форме I6 — позиции с 21 по 26.

Управляющие редакционные спецификации могут быть использованы в списке кодов оператора FORMAT совместно со спецификациями преобразования в произвольной последовательности.

**5. Управление печатью.** Если информация выводится на печатающее устройство, то первый символ записи не печатается, а интерпретируется как *управляющий* и определяет величину интервала перед печатью строки. Как правило, этот управляющий символ задается H-спецификацией или апостроф-спецификацией, хотя, вообще говоря, допустим любой другой способ задания.

Управляющими являются символы

```
— 0 + 1
```

которые соответственно вызывают: перевод одной строки перед печатью, перевод двух строк, печать без перевода строк, прогон бумаги до первой строки следующей страницы. Например, операторы

```
PRINT 5, X
5 FORMAT (3X, F8.3)
```

вызовут печатание на следующей строке значения переменной *X*, начиная с третьей позиции строки. Переход к следующей строке обеспечивается тем, что первым символом печатаемой строки является пробел.

**Упражнения**

1. Пятая запись файла с номером 3 имеет вид:

```
0.16E-010.21E+030.61E-010.22E+030.80E+05
0.12E+030.18E-040.39E+030.71E+010.93E+03
0.57E-010.21E+02
```

Спрелделить значения элементов массива E, которые они получают в результате выполнения программы:

```
REAL E(10) /-19.13, 2.01, -2.86, 3.12, 3* 10., 3* 4:12/
DEFINE FILE 3(5, 800, L, L)
3 FORMAT (3E8.2)
READ (3'5, 3) (E(I), I=3, 10, 3)
STOP
END
```

2. Указать, какой вид будет иметь восьмая запись файла 18 в результате выполнения программы

```
LOGICAL P, Q
DEFINE FILE 18(20, 100, E, L)
6 FORMAT (2G11.3, G9.1)
K=18
P=.TRUE.
Q=.NOT.P
WRITE (18'8, 6) K, P, Q
STOP
END
```

3. Определить последовательность символов, которая будет помещена в двадцатую запись файла с номером 13 в результате выполнения программы

```
DEFINE FILE 13(3, 150, E, K)
7 FORMAT (3A4)
READ (13'5, 7) A, B, C
WRITE (13'20, 7) A, B, C
STOP
END
```

если пятая запись файла с номером 13 имеет вид: OTS—END—FINAL

4. Составить программу умножения векторов A(100) и B(100) по алгоритму:  $C(I) = A(I) * B(I)$ ,  $I = 1, 2, \dots, 100$ , где C(100)—массив результата. Результат напечатать в виде таблицы  $10 \times 10$  значений элементов массива C, представленных по формату E12.5. Таблицу надписать:  $C = A * B$

## § 23. Классификация операторов и наименований

Рассмотренные выше операторы делятся на пять типов (см. также § 3): описательные операторы, операторы присваивания, операторы управления, операторы ввода и вывода, подпрограммы.

Перечислим здесь операторы каждого из типов.

Операторы COMMON, COMPLEX, DATA, DEFINE FILE, DIMENSION, ENTRY, EQUIVALENCE, EXTERNAL, FORMAT, IMPLICIT, INTEGER, LOGICAL, NAMELIST, REAL и оператор определения оператор-функции относятся к первому типу; операторы ASSIGN, арифметический и логический операторы присваивания — ко второму типу; операторы BACKSPACE, CALL, CONTINUE, DO, END, END FILE, FIND, GO TO, IF, PAUSE, RETURN,

REWIND, STOP являются операторами управления; операторы PRINT, PUNCH, READ, WRITE относятся к четвертому типу; операторы BLOCK DATA, FUNCTION, SUBROUTINE — к пятому.

Наименования переменных, массивов и оператор-функций, а также метки операторов и формальные параметры локализованы в той программной единице, в которой они появляются. Это значит, что всякое обращение к ним разрешается только из данного сегмента программы.

Наименования подпрограмм-функций и подпрограмм, наименования всех входов, а также наименования общих блоков являются общими (нелокализованными) для всех сегментов данной программы, и к ним можно обращаться из любой программной единицы в соответствии с установленными правилами (по идентификатору подпрограммы-функции, имени входа или идентификатору стандартной функции, с помощью операторов CALL или COMMON).

Порядок расположения операторов в программной единице следующий:

- оператор определения подпрограммы BLOCK DATA, FUNCTION или SUBROUTINE (используется только в подпрограмме);

- оператор IMPLICIT;

- описательные операторы, отличные от оператора IMPLICIT;

- определения оператор-функций;

- исполняемые операторы;

- оператор END.

Операторы DATA, FORMAT, ENTRY, RETURN могут находиться в любом месте программной единицы. Оператор DATA должен следовать после оператора IMPLICIT за описательными операторами, объявляющими величины, используемые в операторе DATA. Оператор ENTRY не должен находиться в цикле.

Оператор DEFINE FILE должен логически предшествовать операторам ввода и вывода прямого доступа, а оператор NAMELIST — операторам ввода и вывода, которые используют величины, объявленные в операторе NAMELIST.

## § 24. Примеры программ

1. Вычисление интеграла методом Симпсона. Вычислить интеграл  $Q(p) =$

$$= \int_a^b \frac{p^v x e^x dx}{(p + e^x)^{v+1}} \quad \text{как функцию параметра } p, \text{ интервал по которому задан}$$

задано также значение  $v$ . Формула Симпсона имеет вид

$$\int_a^b f(x) dx = \frac{h}{3} [f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \dots + 2f(b-2h) + 4f(b-h) + f(b)],$$

где  $h = (b-a)/n$ ,  $n$  — четное количество интервалов,  $f(x)$  — подынтегральное выражение.

Вычисление подынтегрального выражения может быть осуществлено с помощью оператор-функции

$$F(X) = X * \text{EXP}(X) / (P + \text{EXP}(X)) ** (\text{NU} + 1)$$

где  $X$  — формальный параметр.

Значения оператор-функции, которые умножаются на множитель 4, получаются обращением к  $F$  с фактическими параметрами, равными  $A + H$ ,  $(A + H) + 2H$ , ..., а те значения подынтегрального выражения, которые умножаются на множитель 2, получаются обращением к  $F$  с фактическими параметрами  $(A + H) + H$ ,  $((A + H) + H) + 2H$ , ... Суммирование указанных двух групп значений подынтегральной функции выполняется в цикле с помощью оператора  $IF$ .

Вычисление интеграла  $Q$  в интервале значений параметра  $P$  от  $P_0$  до  $P_K$  осуществляется во внешнем цикле.

Программа имеет вид

```

C      SIMPSON
      REAL NU
      F(X) = X*EXP(X)/(P+EXP(X))**(NU+1)
5      FORMAT (5F11.2, 2I5)
      READ 5, P0, PH, PK, A, B, NU, N
      H = (B-A)/N
      P = P0
10     S4 = 0
         S2 = 0
         I = 1
         X = A + H
15     S4 = S4 + F(X)
         S2 = S2 + F(X + H)
         IF (I - (N - 3)) 20, 25, 25
20     I = I + 2
         X = X + 2 * H
         GO TO 15
25     Q = P ** NU + H / 3 * (F(A) + F(B) +
         * 4 * F(B - H) + 4 * S4 + 2 * S2)
         PRINT 30, Q, P
30     FORMAT (2E15.5)
         IF (P - PK) 35, 40, 40
35     P = P + PH
         GO TO 10
40     STOP
      END

```

**2. Решение системы обыкновенных дифференциальных уравнений.** Требуется написать программу для решения уравнения второго порядка  $y'' - 8y' + x^2y = e^x$  с начальными условиями  $y(0) = y'(0) = 0$  методом Эйлера на отрезке  $[0, 1]$  с шагом  $h = 0.1$

Записав уравнение в виде системы

$$y_1' = y_2'$$

$$y_2' = 8y_2 - x^2y_1 + e^x,$$

воспользуемся подпрограммой для одного шага интегрирования по методу Эйлера с выбранным шагом  $H$  в виде

```

SUBROUTINE E(F, Y, Y0, X, X0, N, H)
  DIMENSION F(N), Y(N), Y0(N)
  DO 5 I=1, N
    Y(I) = Y0(I) + F(I)*H
  5 X = X0 + H
  RETURN
END

```

В этой подпрограмме в число формальных параметров включены следующие идентификаторы:  $Y0$  — массива начальных данных,  $Y$  — массива результатов интегрирования,  $F$  — массива значений правых частей,  $X0$  — начального значения аргумента  $X$ ,  $N$  — числа уравнений в системе,  $H$  — шага интегрирования.

Подпрограмма вычисления правых частей для уравнений системы будет выглядеть так:

```

SUBROUTINE P
  DIMENSION F(2), Y(2), Y0(2)
  COMMON F, Y, Y0
  F(1) = Y0(2)
  F(2) = 8*Y0(2) - X0**2*Y0(1) + EXP(X0)
  RETURN
END

```

И, наконец, еще одна программная единица — основная программа — представляется в виде

```

C      EULER
      COMMON F(2), Y(2), Y0(2)
      X=0
      DO 10 I=1, 2
10    Y(I)=0
      PRINT 15, X, Y(1), Y(2)
15    FORMAT (3E15.5)
      DO 25 K=1, 10
      X0=X
      DO 20 J=1, 2
20    Y0(J)=Y(J)
      CALL P
      CALL E(F, X0, Y0, X, Y, 2, 0.1)
      PRINT 15, X, Y(1), Y(2)
25    CONTINUE

```

```
STOP  
END
```

Окончательная программа составляется из выписанных сегментов, например, в такой последовательности:

```
EULER  
SUBROUTINE P  
SUBROUTINE E
```

#### Упражнения

1. Написать подпрограмму для нахождения корня произвольного алгебраического уравнения  $f(x) = 0$  методом деления отрезка пополам, если известно, что корень существует, единственный и содержится в интервале  $[x_1, x_2]$ .

2. Найти на отрезке  $[0.5, 2.0]$  корень уравнения  $(4 + x^2)(e^x - e^{-x}) = 18$ , используя подпрограмму из предыдущего упражнения.

## § 1. Данные

Все величины, из которых могут быть образованы выражения и над которыми могут быть совершены определенные действия, задаваемые операторами, называются *данными*. В языке ПЛ/1 используются следующие виды данных: константы, переменные, массивы, структуры. Массивы бывают внутренние, подразделяемые на *n*-мерные массивы и массивы структур, и внешние массивы, или файлы.

Для записи элементов языка — данных, выражений и операторов — используется 60 символов, составляющих алфавит языка:

— 29 букв: прописные буквы от А до Z, \$ — знак денежной единицы, @ — коммерческий знак «at», # — знак номера;

— 10 цифр от 0 до 9;

— 21 специальный символ:

+	(плюс),	:	(двоеточие),
-	(минус),	<	(меньше),
*	(звездочка),	>	(больше),
/	(наклонная черта),		(«ИЛИ»),
=	(равно),	&	(«И»),
(	(открывающая скобка),	]	(«НЕ»),
)	(закрывающая скобка),	%	(процент),
,	(запятая),	—	(символ разбивки (черта под строкой)),
.	(точка),	?	(вопрос),
'	(апостроф),		(пробел (пусто)).
;	(точка с запятой),		

Место пробела в тексте иногда обозначают как —

Кроме указанного 60-символьного алфавита, разрешается пользоваться его 48-символьным подмножеством, в котором отсутствуют символы @ и # первой группы, в третьей группе присутствует только 11 символов: плюс, минус, звездочка, наклонная черта, равно, открывающая скобка, закрывающая скобка, запятая, точка, апостроф, пробел, а вместо отсутствующих символов

; : < > | & ] %

приняты следующие замены

.. .. LT GT OR AND NOT //

При этом:

— перед двоеточием должен стоять пробел, если предыдущий символ есть точка, т. е. .—.;

— если впереди или после символов //, заменяющих знак %, стоит символ \*, то он отделяется от // пробелом, т. е. \*—//—\*;

— символ „ обозначает точку с запятой лишь в тех случаях, когда за ним не стоит цифра и если он не стоит внутри примечания или строки символов.

Для конструкций языка, обозначаемых двумя символами, написанными подряд, в случае 48-символьного алфавита вместо символов <=, >=, ] =, ] <, ] >, ||, —> введены обозначения (соответственно) LE, GE, NE, NL, NG, CAT, PT. Если такому символу предшествуют или за ним непосредственно следуют буква или цифра, то между ними должен стоять пробел, например A—NE—1.2.

**1. Константы.** Константы различают *арифметические* и *строковые*. Числа действительные и комплексные, двоичные и десятичные, с плавающей и фиксированной точкой, с различной точностью составляют арифметические константы. Точность числа с фиксированной точкой определяется заданием количества цифр в числе и указанием количества дробных разрядов, а точность числа с плавающей точкой — заданием количества значащих цифр в числе. К строковым константам относятся символьные строки (константы типа строки символов) и битовые строки (константы типа строки битов).

Последовательность десятичных цифр, среди которых содержится точка, с предшествующими знаками + или — или без знака, образует *действительную десятичную константу* (десятичное число) с фиксированной точкой. Например:

+123.45 —67.890 0.9

Действительная десятичная константа с фиксированной точкой, при записи которой отсутствует точка, называется *действительной целой десятичной константой*. Например:

35 —67 +289

Действительная десятичная константа с плавающей точкой образуется как последовательность, состоящая из действительной десятичной константы с фиксированной точкой (мантиссы), символа E (основания порядка) и действительной целой десятичной константы, содержащей не более двух цифр (порядка). Так, запись

21.5E+3 35E—02 .035E5

изображают числа  $21.5 \times 10^3$ ,  $35 \times 10^{-2}$ ,  $0.035 \times 10^5$  соответственно.

Если в изображении десятичного числа с плавающей точкой содержится шесть или меньше значащих цифр, то в памяти машины число хранится с обычной точностью и занимает полное слово (4 байта). Если количество значащих цифр больше шести, то под данное число отводится в памяти 8 байт (двойное слово) и число представляется с повышенной (двойной) точностью.

*Действительная двоичная константа* (двоичное число) с фиксированной точкой записывается при помощи последовательности цифр 0 и 1, в которой может содержаться точка, а предшествовать один из знаков + или —, и

обязательного элемента — символа В — в конце последовательности. Например:

$$1101.011В - .0010111В + 100011В$$

Последнее число записано без точки, образуя тем самым действительную целую двоичную константу.

Действительная двоичная константа с плавающей точкой записывается, например, так:

$$1.11011ЕЗВ - .00101Е-4В + 1101Е-02В$$

т. е. обязательным элементом является символ В после порядка. Порядок — это действительная целая десятичная константа, являющаяся показателем степени при основании 2. Следовательно, 1.11011ЕЗВ — это  $1.11011 \times 2^3$ , т. е. 1110.11.

Под двоичное число с плавающей точкой в памяти отводится полное слово, если число содержит 21 или меньшее количество значащих цифр, в противном случае оно занимает двойное слово.

*Мнимая константа* — это любая действительная константа, за которой следует символ I. Например:

2.16I — мнимая десятичная константа с фиксированной точкой;  
 101VI — мнимая двоичная константа с фиксированной точкой;  
 21.6E-1I — мнимая десятичная константа с плавающей точкой.

*Комплексная константа* записывается как сумма двух величин — действительной и мнимой констант. Например:

$$21.6+3.16I \quad 2.35E+0I \quad -1.75E-01I$$

Комплексная константа хранится в памяти как пара действительных чисел.

*Символьная строка* записывается в программе в виде последовательности символов алфавита, заключенной в апострофы (кавычки). В апострофы могут быть заключены любые символы, в том числе символ ', который следует записывать дважды. Так, символьные строки

'ТЕКСТ' '//5+\$500.' 'IT'S'

представляют собой такие последовательности символов:

ТЕКСТ //5+ \$500. IT'S

*Битовая строка* образуется из цифр 0 и 1, заключаемых в апострофы; после правого символа ' пишется буква В. Например, битовым строкам

'1101'В '0010'В '11111'В

соответствуют последовательности

1101 0010 11111

Строка может быть повторена несколько раз, если перед открывающей кавычкой написать в скобках коэффициент повторения в виде целого десятичного числа без знака. Так, строкам

(2)'ABC' (6)'1'В (0)'TEXT'

соответствуют последовательности

АВСАВС 111111

В последнем примере приведена пустая строка (коэффициент повторения равен нулю), которая представляет собой строку без символов.

**2. Переменные, массивы, структуры.** Переменные по своему назначению делятся на арифметические и управляющие. Различают два вида переменных: простые (скалярные) переменные и переменные с индексами. Для обозначения простой переменной служит идентификатор. Переменная с индексами записывается как идентификатор с последующим списком индексов. Переменные с индексами объединяются в массивы. Структура образуется как упорядоченная совокупность данных. Элементами структуры могут быть простые переменные, массивы и другие структуры.

Арифметические переменные делятся на переменные действительного и комплексного типов и строковые (типа строки символов), а управляющие переменные — на переменные типа метки, указателя, ветви, события, области, ячейки. Ниже из управляющих переменных будут рассмотрены только первые два типа.

### Упражнения

1. Преобразовать записи, составленные из символов 60-символьного алфавита, в записи, составленные из символов 48-символьного алфавита.

A    B    C	M=K   =L;	X &   Y
P->Q	30*% (S>=S1)	25%   =S
X & Y	X   > Y   < Z	(1.:10.;:9:10)
2.:3.14:5	27<72	X     Y   Z

2. Среди приведенных ниже записей констант определить те, которые являются: а) десятичными числами; б) двоичными числами; в) действительными числами; г) комплексными числами; д) константами с плавающей точкой; е) символьными строками; ж) битовыми строками; з) записями, не допускаемыми правилами языка:

123.45B	40E0B	'0100'B1
37.73	f01E1B	'SIN'
-347.	'#5138B'	(2)'TAN'
.12E-16	10011B	11-111
11E-01	101E11B	11-11E-111
55D-02	.01B	10B+101E1B1
'11E+01'	2733B	(11)'101'B
(7)'10'B	1B11+1	55 <sub>10</sub> -02

3. Представить в виде десятичных чисел с плавающей точкой так, чтобы мантисса содержала три цифры, первая из которых не нуль:

100	$12 \times 10^{-3}$	$2 \times 2^3$
-12,3	0,0003	1/3
2,75	-666	0,0088
-1/8	370	$-10^{-10}$

## § 2. Идентификаторы

Для обозначения переменной служит *идентификатор* — последовательность не более чем из 31 символа. Первым символом идентификатора должна быть латинская буква или символы @, # или \$. Кроме указанных символов, в

записи идентификаторов разрешается использовать цифры, а также символ разбивки, который может быть использован в любом месте идентификатора, за исключением его начала и конца. Например,

A X5 NAME—OF— INDEX \$1000 #77

Идентификаторами обозначаются также метки операторов, формальные параметры подпрограмм, указатели входов в подпрограммы, метки подпрограмм, имена файлов.

Ряд идентификаторов в виде последовательностей букв, являющихся словами английского происхождения, используется для обозначения объектов языка, называемых операторами, описателями, дополнениями, ситуациями прерывания, встроенными функциями. Это служебные или ключевые слова, которые, однако, могут быть употреблены в любом другом смысле, например в качестве имен переменных. В таком случае истинный смысл данного идентификатора определяется транслятором контекстуально. При использовании 48-символьного алфавита сочетания букв, которыми заменяются специальные символы, т. е. составные символы LT, GT и др., а также ключевые слова могут быть использованы только в этом специальном смысле.

В качестве ключевых слов используются следующие идентификаторы (в скобках указываются допускаемые сокращения, дается также перевод на русский язык):

при записи операторов:

ALLOCATE — разместить, BEGIN — начало, CALL — вызвать, CLOSE — закрыть, DECLARE (DCL) — объявить, DELETE — удалить, DISPLAY — показывать, DO — выполнить, END — конец, ENTRY — вход, FORMAT — формат, FREE — освободить, GET — выбрать, GO TO — перейти к, IF — если, LOCATE — разместить, ON — на, OPEN — открыть, PROCEDURE (PROC) — процедура, PUT — послать, READ — читать, RETURN — вернуть, REVERT — отменить, REWRITE — переписать, SIGNAL — сигнал, STOP — стоп, WRITE — записать;

для обозначения описателей:

ALIGNED — выровненный, AUTOMATIC — автоматический, BACKWARDS — обратный, BASED — базированный, BINARY (BIN) — двоичный, BIT — битовый, BUFFERED — с буфером, CHARACTER (CHAR) — символьный, COMPLEX (CPLX) — комплексный, CONSECUTIVE — последовательный, CONTROLLED (CTL) — управляемый, DECIMAL (DEC) — десятичный, DEFINED (DEF) — определенный по, DIRECT — прямой, ENTRY — входной, ENVIRONMENT (ENV) — оборудованный, EXCLUSIVE — с защитой, EXTERNAL (EXT) — внешний, FILE — файл, FIXED — фиксированный, FLOAT — плавающий, INDEXED — индексный, INITIAL (INIT) — начальный, INPUT — для ввода, INTERNAL (INT) — внутренний, KEYED — с признаком, LABEL — метка, LIKE — подобный, LINESIZE — размер строки, OUTPUT — для вывода, PACKED — упакованный, PAGESIZE — размер страницы, PICTURE (PIC) — шаблон, POINTER (PTR) — указатель, POSITION — позиция, PRINT — для печати, REAL — действительный, RECORD — запись, REGIONAL — региональный, RECURSIVE — рекурсивный, RETURNS — возвраты, SEQUENTIAL — последовательный, STATIC — статический, STREAM — поток,

UNBUFFERED — без буфера, UPDATE — обновить, VARYING (VAR) — переменный;

при указании дополнений:

BY — шаг, BY NAME — по имени, COLUMN — столбец, COPY — копия, DATA — данные, EDIT — редактирование, ELSE — иначе, FROM — из, IGNORE — игнорировать, INTO — в, KEY — ключ, KEYFROM — ключ из, KEYTO — ключ в, LINE — строка, LIST — список, MAIN — главный, OPTIONS — выбор, PAGE — страница, REPLY — ответ, SET — установить, SKIP — пропустить, SNAP — состояние, STRING — строка, SYSTEM — системный, THEN — то, TO — до, WHILE — пока;

для задания ситуаций прерывания:

CHECK — проверка, CONDITION — состояние, CONVERSION (CONV) — преобразование, ENDFILE — конец файла, ENDSPAGE — конец страницы, ERROR — ошибка, FINISH — окончание, FIXEDOVERFLOW (FOFL) — переполнение, NAME — наименование, NO — префикс ситуации прерывания, OVERFLOW (OFL) — переполнение, RECORD — запись, SIZE — размер, STRINGRANGE (STRG) — диапазон строки, SUBSCRIPTRANGE (SUBRG) — диапазон индекса, TRANSMIT — передача, UNDEFINEDFILE (UNDF) — неопределенный файл, UNDERFLOW (UFL) — исчезновение порядка, ZERODIVIDE (ZDIV) — деление на нуль.

При написании в программе два идентификатора не должны непосредственно примыкать друг к другу. Они должны отделяться каким-нибудь знаком операции, примечанием, одним из символов-разделителей из набора:

( ) , ' . ; = :

или пробелом. Сказанное относится также к константам и шаблонам (см. § 4, п. 2).

Что касается символа пробел, то он широко используется при построении конструкций языка. Пробелы в произвольном количестве могут стоять по обе стороны от знаков операций и символов-разделителей и допускаются внутри символьных строк. Однако идентификаторы, составные знаки операций, шаблоны, константы (кроме символьных строк) не должны содержать пробелов.

Идентификаторы используются также для обозначения так называемых встроенных функций — автоматически включаемых в программу подпрограмм, реализующих часто встречающиеся математические процедуры. Идентификаторы встроенных функций могут в программе использоваться для обозначения других объектов.

### Упражнения

1. Среди приведенных ниже последовательностей символов выбрать идентификаторы. У остальных указать ошибки в записи.

DELTA	A50R17S	X1100LOGICAL
2PI	2.A	\$10F5
AREA	NAME—JOE	IT'S
REAL	#1982	A—B—C—D
@123	EC=1060	COEFFICIENT—
ЗАДАНИЕ	TY—156	DET OF INIT
FLOAT	AND	3NNNN
55ALPHA	'VARYING'	V(P)

2. Составить из символов I и 1 все возможные идентификаторы, состоящие из трех символов.

3. Сколько идентификаторов, состоящих из четырех символов, можно составить из: а) символов A, B, C, D; б) символов A, B, C, 1; в) символов A, B, 2, 1; г) символов A, 3, 2, 1; д) символов 4, 3, 2, 1?

### § 3. Операторы

Под оператором понимается предписание, приводящее к вполне определенному функционированию ЭВМ (к выполнению определенных действий). Все операторы могут быть разделены на два класса: *неисполняемые* и *исполняемые*.

К классу неисполняемых относятся операторы, используемые в программе для указания свойств величин, для описания формы представления данных на внешних носителях информации, для указания о распределении памяти и т. д. Это оператор объявления данных, оператор размещения данных, оператор освобождения памяти, оператор форматов, оператор конца. При выполнении программы, если управление передается одному из этих операторов, он пропускается.

Класс исполняемых операторов составляют операторы, используемые в программе для указания конкретных действий и порядка выполнения этих действий. Это оператор присваивания, оператор перехода, условный оператор, оператор цикла, оператор блока, оператор процедуры, оператор останова, оператор вызова процедуры, оператор входа в процедуру, оператор возврата, оператор ожидания, оператор задания режима обработки прерываний, оператор отмены реакции на прерывание, операторы ввода и вывода, пустой оператор.

Различают следующие основные логические группы операторов: объявления данных, присваивания, управления, структуры программы, отладки, распределения памяти, ввода и вывода.

В общем случае оператор записывается в форме

(C): M: KC — XSL;

где C — список имен ситуаций прерывания, M — список меток, KC — ключевое слово оператора; XSL — тело оператора, представляющее собой список, элементами которого могут быть идентификаторы, обозначающие переменные, массивы, структуры, описатели, дополнения, элементы формата, другие операторы; символ-разделитель точка с запятой указывает конец оператора.

Основным (обязательным) элементом оператора (за исключением оператора присваивания и пустого оператора) является ключевое слово, обязательным элементом является также символ ; (точка с запятой) в конце оператора, остальные элементы могут частично или полностью отсутствовать. В дальнейшем в каждом конкретном случае структура тела оператора будет подробно рассмотрена. Там же будет пояснен смысл элементов с различными наименованиями, которые входят в список тела оператора. В частности, если оператор содержит в списке тела оператора другой оператор, он называется составным. Соответственно простой оператор — это оператор, не содержащий в себе других операторов.

Простым оператором является пустой оператор, который никаких действий не выполняет и тело которого никак не обозначается. Следовательно, пустой оператор имеет вид

M: ;

где M — список меток — может отсутствовать. Однако если метка имеется, можно, передавая управление на пустой оператор, переходить на конец некоторого составного оператора или блока, пропуская какие-то другие операторы.

При записи операторов на бланке допускается так называемый свободный формат. Из 80 позиций строки для записи операторов разрешается использовать позиции 2 — 72, соответствующие колонкам 2 — 72 на перфокарте. Первая позиция не должна заполняться, а позиции 73 — 80, хотя и могут быть заполнены информацией, транслятором игнорируются. Для удобства чтения между элементами одного оператора и между соседними операторами разрешается оставлять пробелы. В одной строке может быть записано несколько операторов; один оператор может занимать несколько строк, при этом перенос разрешается в любом удобном месте.

#### Упражнение

Определить по формальным признакам, являются ли операторами следующие конструкции:

```
DECLARE A COMPLEX FIXED;
SIZE: PROCEDURE;
(NOSIZE): E=F*N;
K: L: M: N: STOP;
57: END;
SYSTEM: ON SYSTEM;
```

## § 4. Описание данных

Для описания свойств идентификаторов, обозначающих переменные, массивы и структуры, применяются три способа: *явное описание*, *контекстуальное описание* и *неявное описание* (по умолчанию).

Явное описание осуществляется оператором объявления данных (с ключевым словом DECLARE), а для данных, организованных в файлы, кроме того, еще оператором открытия файлов (с ключевым словом OPEN).

Контекстуальное описание осуществляется без явного указания каких-то свойств идентификаторов, а по имеющимся другим его свойствам или же на основе характера использования идентификатора в программе.

Неявное описание представляет собой присвоение определенных свойств идентификатору по предварительному соглашению (по умолчанию).

Оператор объявления данных имеет вид

```
DECLARE XL;
```

где элементы списка XL есть идентификаторы и описатели.

1. Описание простых арифметических переменных. Для описания арифметических данных используются описатели

— для задания основания системы счисления DECIMAL и BINARY, указывающие соответственно, что значение величины представляется либо в десятичной, либо в двоичной системе счисления;

— для задания формы представления FLOAT и FIXED, указывающие соответственно, что значение величины представляется либо в форме с плавающей точкой, либо с фиксированной точкой;

— для задания типа REAL и COMPLEX, указывающие, что величина является либо действительной, либо комплексной;

— для задания точности (разрядности) (w,d) и (w) соответственно для данных с фиксированной точкой и данных с плавающей точкой. Здесь w — целое десятичное число без знака, представляющее собой общее (максимальное) количество цифр, допустимое для значения величины (значащие цифры в мантиссе в случае числа с плавающей точкой); d — десятичное целое число со знаком или без знака, при  $d > 0$  — это количество дробных разрядов в десятичном или двоичном значении переменной (если значение d не задано, предполагается, что d равно нулю), при  $d = 0$  точка располагается в конце числа; если  $d < 0$ , то соответствующее значение состоит из  $w + |d|$  разрядов, причем младшие  $|d|$  разрядов есть нули.

При описании комплексных данных предполагается, что вещественная и мнимая части имеют одинаковые описатели (как пара действительных величин).

Некоторые из перечисленных описателей (или все) могут отсутствовать. В этом случае по умолчанию предполагается, что описываемым данным присваивается: из описателей основания системы счисления — DECIMAL, из описателей формы представления — FLOAT, из описателей типа — REAL, а точность, которая, вообще говоря, зависит от типа ЭВМ, обычно задается как (5, 0) или (6) для десятичных значений и как (15, 0) или (21) в случае двоичных значений.

Описатели в операторе объявления данных указываются после идентификатора, и могут следовать в произвольном порядке, однако непосредственно за идентификатором не может быть указан описатель разрядности.

Примеры описания простых переменных для арифметических данных:

```
DECLARE X REAL DECIMAL FIXED (5, 3);
DECLARE C REAL BINARY FLOAT;
DECLARE F DECIMAL FIXED (3,-2);
```

Здесь с помощью трех операторов описаны переменные, принимающие значения: X — действительное, десятичное с фиксированной точкой, общее количество разрядов — 5, дробных — 3; C — действительное, двоичное с плавающей точкой и 15 (по умолчанию) разрядами в мантиссе; F — действительное (по умолчанию), десятичное с фиксированной точкой, содержащее пять разрядов, из них младшие два состоят только из нулей (например, значением F может быть число 12300, но не 12340 или 12345).

В одном операторе может быть описано любое количество переменных. Так, приведенные операторы можно объединить в один; в этом случае между описаниями пишется разделитель запятая:

```
DECLARE X REAL DECIMAL FIXED (5,3),
      C REAL BINARY FLOAT,
      F DECIMAL FIXED (3,-2);
```

Разрешается использовать в программе переменные, идентификаторы которых не указаны в операторе объявления данных. В этом случае, если первый символ идентификатора есть одна из букв I, J, K, L, M или N, то соответствующей переменной приписываются по умолчанию описатели BINARY FIXED REAL (15, 0), во всех остальных случаях — описатели DECIMAL FLOAT REAL (6).

Для описания строковых переменных используются описатели CHARACTER и BIT. Первый из них является описателем символьной строки, второй — битовой строки. Длина строки указывается целым числом в скобках (p) в байтах для описателя CHARACTER и битах — для описателя BIT. Например, оператором

```
DECLARE L CHARACTER(7), R BIT (8);
```

описываются символьная строка L, размещаемая на поле длиной 7 байт, и битовая строка R, занимающая поле длиной 8 битов, (т. е. 1 байт), например

```
'ABCDEFH' '01010101'
```

Для строковой переменной может быть задан описатель VARYING, который указывает, что строка имеет переменную длину; если он опущен, то считается, что строка имеет постоянную длину. Например, значениями переменной Y, объявленной оператором

```
DECLARE Y CHARACTER(5) VARYING;
```

могут быть строки

```
'ABCDE' 'ABCD' 'ABC' 'AB' 'A' "
```

Описатель длины (p) должен следовать за описателем BIT или CHARACTER, опущен быть не может. Значение p — это длина в случае строк с постоянной длиной и максимальная длина для строк с переменной длиной. В качестве p допускается скалярное выражение, принимающее целое значение или символ \*, если длина может быть взята из предшествующего размещения данных.

Максимально допустимая разрядность в случае описателей: DECIMAL FIXED равна (15, d), DECIMAL FLOAT — (16), BINARY FIXED — (31, d), а BINARY FLOAT — (53); для описателя длины p максимальное значение равно 32767.

Несколько переменных, которые описываются одинаково, могут быть объединены в один список, заключаемый в скобки, за которым описатели пишутся только один раз. Например, вместо

```
DECLARE A BIT (10), B BIT (10), C BIT (10);
```

можно написать

```
DECLARE (A, B, C) BIT (10);
```

**2. Шаблоны.** Строки символов и десятичные данные в форме с фиксированной или плавающей точкой могут быть описаны *шаблоном*.

Записывается шаблон с помощью описателя PICTURE, после которого следует шаблон, заключаемый в кавычки. Таким образом, общая форма оператора будет следующая:

DECLARE X PICTURE 'шаблон';

где X — идентификатор, 'шаблон' — строка определенных символов.

В случае комплексных данных указывается описатель COMPLEX и один шаблон, общий для вещественной и мнимой частей описываемой величины. Применение шаблонов для двоичных данных не предусмотрено.

Описание с помощью шаблона используется в случае необходимости внутреннего представления данных в памяти в виде строк, в ряде случаев, удобном для организации ввода или вывода, в частности для редактирования подлежащих печати выходных данных, когда необходимо вставить знак, подвить незначащие нули и т. д.

В шаблоне используются символы, на месте которых в описываемом шаблоне значении подразумеваются вполне определенные символы алфавита. Список символов шаблона для десятичных данных с указанием, что эти символы означают, следующий:

9 — любая десятичная цифра;

V — подразумеваемая точка;

S — место знака числа; если число больше нуля, обозначает +, если оно меньше нуля — знак минус;

+ — место знака числа; если число больше нуля, обозначает +, если меньше нуля — пробел;

— — место знака числа; если число меньше нуля, обозначает минус, если число больше нуля или равно нулю — пробел;

E — место буквы E, указывающей начало показателя степени числа с плавающей точкой;

K — подразумеваемое начало показателя степени числа с плавающей точкой;

T — цифра с дополнительной пробивкой для знака числа на перфокарте над этой цифрой (в 12-й строке для знака + и в 11-й строке для знака минус);

I — цифра с дополнительной пробивкой для знака числа над этой цифрой, если число на поле больше нуля или равно нулю;

R — цифра с дополнительной пробивкой для знака числа над этой цифрой, если число на поле меньше нуля;

CR — два символа, которые будут содержаться в соответствующих позициях, если число больше нуля или равно нулю;

DB — два символа, которые будут содержаться в соответствующих позициях, если число меньше нуля;

F — наличие в шаблоне масштабного множителя, который указывается в скобках как положительное или отрицательное целое число; не разрешается в случае полей с плавающей точкой;

Z — цифровая позиция, которая заменяется пробелом, если в ней содержится незначащий нуль (имеются в виду старшие разряды числа);

\* — аналогичен символу Z, только незначащие нули заменяются звездочками;

Y — аналогичен символу Z, только все незначащие нули заменяются пробелами;

\$ — позиция, которая содержит символ \$; она может быть либо крайней левой, либо крайней правой;

, — позиция, в которую вносится символ запятой; если наличие символов Z или \* вызвало подавление нулей и цифровые символы слева от запятой отсутствуют, то запятая заменяется пробелом в случае символа Z и звездочкой в случае символа \*;

. — аналогична запятой, только вместо запятой используется точка;

/ — аналогична запятой, только используется наклонная черта;

V — пробел, который вводится в указанную позицию.

В шаблоне можно использовать только один символ знака из числа: S, +, -, T, I, R, CR, DB, за исключением числа с плавающей точкой, состоящего из двух частей — мантиссы и порядка, которые могут иметь собственные знаки. Символ знака должен занимать крайнее левое или крайнее правое положение в соответствующем поле. Для повторения символа в шаблоне перед ним в скобках записывается коэффициент повторения.

Примеры шаблонов с содержимым поля памяти и соответствующим числом, представляемым шаблоном:

'999V99'	12345	123.45
'(3)9V99'	12345	123.45
'S99V9'	-123	-12.3
'+99V9'	+123	12.3
'-99V9'	-123	-12.3
'V99ES99'	12E+03	.12×10 <sup>3</sup>
'V99KS99'	12+03	.12×10 <sup>3</sup>
'99F(-2)'	12	.12

Символы Z, \*, Y, \$, ., /, V и , шаблона используются в основном при редактировании.

Символы \*Z и \$, S, +, -, если они появляются в шаблоне один раз, означают действия, описанные выше, и являются статическими символами. Появление их более одного раза в шаблоне превращает их в плавающие символы; это означает, что денежный знак, плюс, минус или пробел должны быть введены перед старшей значащей цифрой в поле или в крайнюю правую позицию плавающего символа в строке, т. е. слева будут пробелы, а знаки \$, +, - займут один символ в позиции, соответствующей самому правому пробелу.

Примеры результатов вывода числа с различными шаблонами:

12.34	'\$999V.99'	\$012.34
12.34	'\$ZZ9V.99'	\$—12.34
12.34	'\$**9V.99'	\$*12.34
12.34	'\$\$\$9V.99'	\$12.34

Для описания строк символов в шаблоне используются те же символы, что и для числовых шаблонов, но в шаблон для строки должен входить по крайней мере один из символов — А или Х, где А указывает любую букву или пробел, а Х — любой символ алфавита языка. Кроме того, в шаблоне символьных строк цифра 9 означает любую цифру от 0 до 9, а также пробел. Так, в операторе

```
DECLARE ТЕКСТ PICTURE 'XXXXX',
      TEXT PICTURE '(4)A',
      TEST PICTURE '99AX';
```

описываются строковые переменные: ТЕКСТ — принимающая значение строковой константы, состоящей из пяти любых символов, TEXT — из четырех букв или пробелов, TEST — из двух цифр или пробелов, одной буквы или пробела и одного произвольного символа.

3. Описание массивов. Переменная с индексами записывается с помощью идентификатора, за которым в скобках указываются индексы в виде выражений. Количество индексов определяет размерность массива. Максимально допустимое число индексов равно 32, однако в ряде трансляторов, где используется подмножество ПЛ1, оно сокращено до трех. Примеры записи переменных с индексами:

```
M(7,1) A(1) XY(1,2,5,7,1,1,8)
```

При описании массива в операторе DECLARE сразу после идентификатора указываются границы изменения индексов (описатель измерения). В остальном описание массива совпадает с описанием простой переменной; следовательно, все элементы массива имеют одинаковый тип и одни и те же описатели.

Границы изменения индексов задаются в виде граничной пары, состоящей из нижней границы, двоеточия и верхней границы. Предполагается, что индекс принимает значения от нижней границы до верхней с шагом +1. Границы изменения индексов — целые десятичные константы (возможно, отрицательные и нуль) или скалярные арифметические выражения. Например, оператор

```
DECLARE F (0 : 2) DECIMAL FIXED (6, 0),
      E (-3.5, 1:8), A (1:2, 1:2);
```

описывает три массива: одномерный F с элементами (переменными с индексами) F(0), F(1), F(2); двумерный E(-3.1), E(-3,2); ..., E(-2,1), ...; двумерный A(1,1), A(1,2); A(2,1), A(2,2).

В памяти элементы массива записываются с последовательным упорядочением индексов, начиная с первого (иными словами, первым изменяется последний индекс).

Если нижняя граница изменения индекса равна +1, то в описателе измерения можно указывать лишь верхнюю границу. Следовательно, последний массив из предыдущего примера можно описать так:

```
DECLARE A(2,2);
```

Допускается объявление массивов с переменными (пока неизвестными) верхними границами изменения индексов, если массив является параметром процедуры. В этом случае в операторе DECLARE вместо всех границ следует писать символы \*. Например:

```
DECLARE M(**,*) BIT (4);
```

Максимальное значение, которое может быть использовано в качестве индекса, равно 32767.

Указание в выражениях или операторах идентификатора массива означает, что обращение делается ко всем элементам массива. Указание идентификатора с индексами означает обращение к конкретному элементу. Обращение к части массива осуществляется как обращение к сечению массива. Обращение к сечению массива производится при помощи имени массива, за которым следует список индексов, по крайней мере один из которых есть \*. Так, если в  $i$ -й позиции в списке индексов стоит \*, то сечение включает все элементы массива, получаемые при изменении  $i$ -го индекса между его границами. Размерность сечения равна количеству звездочек в списке индексов. Например:

STN			
NAME		FAC	COURSE
F	N		

Рис. 7.

$A(3,*)$  означает обращение к третьей строке массива  $A$ ,

$B(**,2)$  — к двумерному сечению (второй плоскости массива  $B$ ),

$A(**,*)$  — обращение ко всему массиву, т. е. эквивалентно указанию идентификатора массива  $A$ .

**4. Описание структур.** Описание структуры производится оператором DECLARE, в котором указываются идентификатор (имя) структуры, идентификаторы групп данных, входящих в структуру (имена подструктур), и идентификаторы первичных составляющих (имена элементов структуры). Для указания иерархии соподчинения идентификаторов перед каждым из них записывается номер уровня — десятичное целое число без знака, меньшее 256. Самая внешняя структура называется старшей и номер ее равен 1; содержащиеся в ней структуры являются младшими. Младшие структуры должны всегда иметь номер уровня, численно больший номера уровня той структуры, в которой они содержатся. Младшим структурам присваиваются не обязательно последовательные номера.

**Пример.** Требуется дать сведения о каждом из студентов-спортсменов, представляющих различные факультеты и курсы учебного заведения, т. е. необходимо указать номер факультета, на котором учится студент, номер курса, имя и фамилию студента. Организация такой структуры с идентификатором (именем) STN схематически изображена на рис. 7, где NAME — идентификатор студента, состоящий из фамилии F и имени N; FAC и COURSE — соответственно имена подструктур (факультета и курса).

Соответствующий оператор DECLARE примет вид

```
DECLARE 1 STN,
        2 NAME,
        3F CHARACTER (16),
```

3N CHARACTER (16),  
 2 FAC DECIMAL FIXED (3, 0),  
 2 COURSE DECIMAL FIXED (2, 0);

В этом примере старшая структура с именем STN состоит из подструктур NAME, FAC и COURSE, а подструктура с именем NAME в свою очередь содержит подструктуры F и N. Запись оператора DECLARE «в столбик» наглядна, но не обязательна; повторяющиеся номера уровней и описатели можно вынести за скобки. Например, приведенный оператор можно записать так:

DECLARE 1 STN,  
 2 (NAME, 3(F,N) CHARACTER (16),  
 FAC DECIMAL FIXED (3,0),  
 COURSE DECIMAL FIXED (2,0));

Однотипные структуры из предыдущего примера, содержащие сведения о каждом из студентов рассматриваемой группы, могут быть объединены в более сложные конструкции — массивы структур. Например, если студентов-спортсменов всего 40, то сведения о всей группе могут быть описаны оператором, совпадающим с только что приведенным, за исключением первой строки, принимающей вид

DECLARE 1 STUDENT(40),

где идентификатор STUDENT — имя массива структур из 40 элементов.

Для обращения к элементам структуры используется составное или уточненное имя, которое представляет собой последовательность идентификаторов, записываемых один за другим в порядке увеличения номеров уровней структуры. Идентификаторы последовательности отделяются точкой, вокруг точки разрешается оставлять пробелы. Составное имя не обязательно содержит имена всех структур, но должно включать достаточно имен для того, чтобы избежать неоднозначности. Например, если имеем описание

DECLARE 1 A, 2 B, 3(D, E), 2 C, 3 D CHARACTER (4);

то составное имя A.B.D означает обращение к элементу D подструктуры B, а составное имя A.C.D — обращение к элементу D подструктуры C. Очевидно, однозначными будут также обращения B.D и C.D

Для обращения к структуре в целом указывается ее идентификатор. Обращение к подструктуре означает, что выполняется обращение ко всем ее составляющим.

Если структура является компонентой массива структур, то обращение записывается в форме переменной с индексами, образуя составное имя с индексами. Составное имя с индексами должно иметь по крайней мере один индекс. Так, для структуры S, определяемой оператором

DECLARE 1 S(5,10), 2 T(7), 3(R(6),Q);

составные имена S(1,3).T(5).R(2), S(1).T(3,5).R(2), S(1).T(3).R(5,2) и т. д. относятся все к одному и тому же элементу. Использование составного имени

ные контакты 1 и 2 установлены на концах гетинаксовых штанг 5 и 6. Неподвижные контакты 3 и 4 выполнены в виде сопел. При размыкании выключателя подвижные контакты 1 и 2 под действием пневматического привода отходят от неподвижных контактов 3 и 4. При этом на каждый полюс образуются две последовательно включенные электрические дуги. При испытании такой выключатель показал отключающую мощность 1 250 Мва; гашение дуги осуществилось в течение одного полупериода.

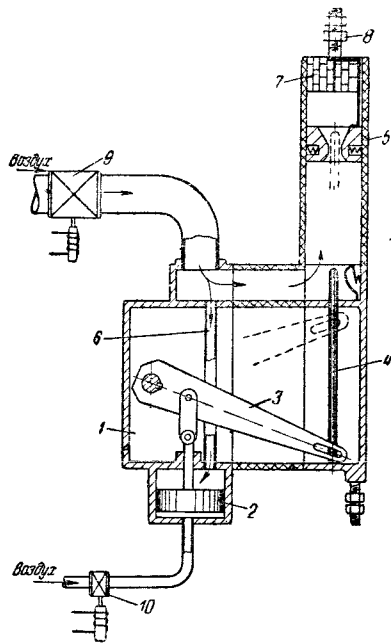


Рис. 8-11. Выключатель с гашением дуги сжатым воздухом с одним разрывом цепи тока на полюс фирмы АЭГ на 10 ква, 600 а (1933 г.)

На коробчатом основании 1 в нижней части помещен цилиндр с поршнем 2, приводимый в движение сжатым воздухом. В процессе включения открывается кран 10, поршень 2 перемещается в крайнее верхнее положение и своим штоком поворачивает рычаг 3, связанный с подвижным контактом 4. Неподвижный контакт 5 установлен сверху гасительной камеры, выполненной из изолирующего материала и снабженной глушителем 7. При отключении через клапан 9 по трубопроводу подается сжатый воздух в гасительную камеру и одновременно по трубке 6

Приведенные выше выключатели следует рассматривать как опытные конструкции, построенные фирмой АЭГ по разработкам Руппеля.

На аналогичный выключатель с двойным разрывом цепи тока Руппель получил в Германии патент № 580699 по заявке от 4 июля 1929 г.

В 1933—1934 гг. фирма АЭГ разработала первую серию выключателей с гашением дуги сжатым воздухом с одним разрывом цепи тока на полюс для напряжений от 3 до 220 кв (рис. 8-11).

сжатый воздух поступает под верхнюю крышку цилиндра пневматического привода. Поршень 2 опускается вниз и между подвижным 4 и неподвижным 5 контактами образуется электрическая дуга, которая гасится струей сжатого воздуха, выходящего через сопло неподвижного контакта 5.

На рис. 8-12 показаны внешний вид и разрез одного из серии выключателей для напряжений 60—220 кВ с мощ-

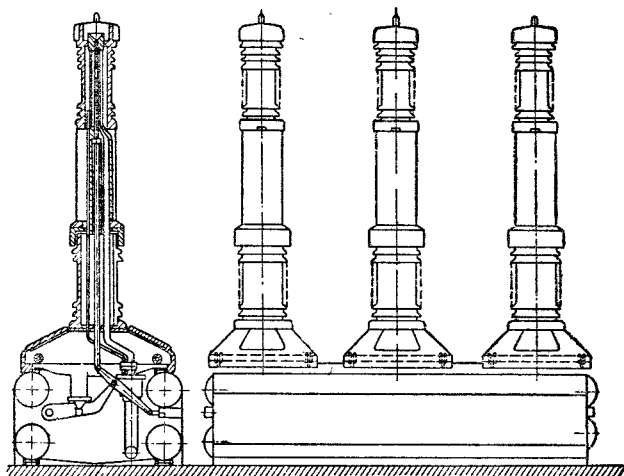


Рис. 8-12. Внешний вид и разрез колонкового выключателя с гашением дуги сжатым воздухом фирмы АЭГ с одним разрывом цепи тока на полюс для напряжений 60—200 кВ (1933 г.)

ностью отключения от 1 000 до 2 500 Мва. Все три фазы выключателя смонтированы на общей раме. Пневматический привод размещен между боковыми стенками рамы, образованными четырьмя резервуарами сжатого воздуха.

Характерной особенностью привода этой серии выключателей было наличие трех пневматических цилиндров, поршни которых через шток и систему рычагов осуществляли перемещение подвижных контактов выключателей.

В 1928 г. фирма АЭГ предложила новую конструкцию привода для перемещения контактов выключателя путем установки поршня, непосредственно укрепленного на нерабочем конце подвижного контакта. На эту конструкцию привода подвижного контакта фирма АЭГ получила

в 1930 г. германский патент [Л. 8-11]. Изобретателем этой конструкции был Риттмейер, как то указано в германском патенте.

Подвижной контакт 1 (рис. 8-13) связан с поршнем 2. При подаче сжатого воздуха по трубе 3 над поршнем 2 создается давление воздуха и поршень 2 вместе с подвижным контактом 1 передвигается вниз.

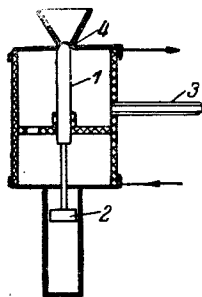


Рис. 8-13. Новая конструкция привода подвижного контакта в выключателях фирмы АЭГ, предложенная К. Риттмейером (1923 г.)

Между контактами 1 и 4 образуется дуга, которая быстро гасится сильной струей воздуха, протекающего через сопло, установленное на неподвижном контакте 4.

Здесь следует отметить, что идея установки на конце подвижного контакта поршня для передвижения подвижного контакта принадлежит Руппелю. На такую конструкцию в 1928 г. Руппелю был выдан германский патент № 543132. В предложенной Руппелем конструкции пружины у поршня ускоряли ход подвижного контакта, а в выключателе фирмы АЭГ они возвращают подвижной контакт в исходное положение по окончании процесса отключения.

Идея непосредственной связи поршня с подвижным контактом выключателя нашла широкое применение в новой серии выключателей фирмы АЭГ со свободной струей воздуха.

## б) Выключатели со свободной струей воздуха

При поставке выключателей серии 1932—1933 гг. появились нарекания со стороны заказчиков на большие размеры выключателей, высота которых при 220 кв достигала 7 м. В этой серии выключателей гасительная камера выполнялась из изолирующего материала, и ее размеры по длине сильно увеличивались с ростом рабочего напряжения. Это объяснялось тем, что на одном конце гасительной камеры размещался неподвижный контакт, а через дно камеры проходил подвижной контакт с длинной штангой, связанной с рычажным механизмом пневматического привода. Чтобы уменьшить габаритные размеры выключателя и сделать его более компактным и дешевым, фирма АЭГ в 1935 г. разработала новую конструкцию выключателя со свободной струей воздуха и получила на нее в 1939 г. гер-

манский патент № 683016. В этом типе выключателя нашла применение идея непосредственной связи подпружиненного поршня с подвижным контактом.

Подвижные контакты *1* (рис. 8-14) помещены в гасительные головки и в момент отключения втягиваются внутрь грибообразных изоляторов *2*. Контакты *1* передвигаются под давлением сжатого воздуха на подпружиненные поршни *4*, укрепленные на концах подвижных контактов

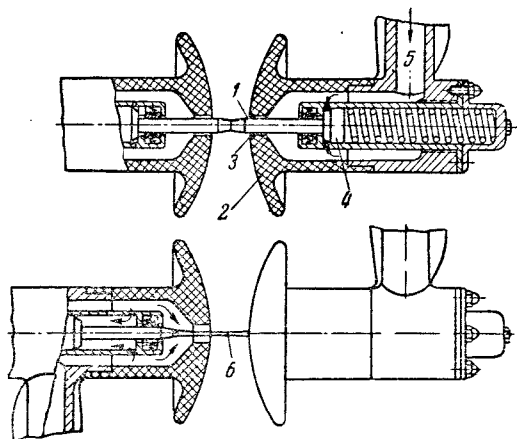


Рис. 8-14. Гасительная головка выключателя фирмы АЭГ со свободной струей воздуха (1935 г.)

тов. Последние проходят через изоляторы *2* с зазором, так что при подаче сжатого воздуха в процессе отключения выключателя свободная струя воздуха устремляется через кольцевую щель *3* к месту расхождения контактов *1*, где образуется электрическая дуга *6*. При дальнейшем движении контактов дуга затягивается внутрь гасительных камер.

В момент прохождения контакта *1* через отверстие в головке грибообразного изолятора *2* оба конца дуги интенсивно обдуваются струей сжатого воздуха, поступающего по трубе *5*, и дуга *6* гасится в течение одного-двух полу-периодов.

К дальнейшему улучшению выключателя со свободной струей воздуха служило предложение поместить обе гасительные головки на концах пустотелых рычагов, установленных на вершину опорных изоляторов. После окончания

процесса гашения дуги опорные изоляторы поворачивались на угол около  $80^\circ$  и тем создавался достаточный воздушный промежуток между обеими гасительными головками, что давало возможность не ставить отъединителя.

На конструкцию выключателя в 1936 г. (рис. 8-15) была подана заявка и получен германский патент № 703743.

На головках опорных изоляторов укреплены пустотелые рычаги, несущие на свободных концах гасительные голов-

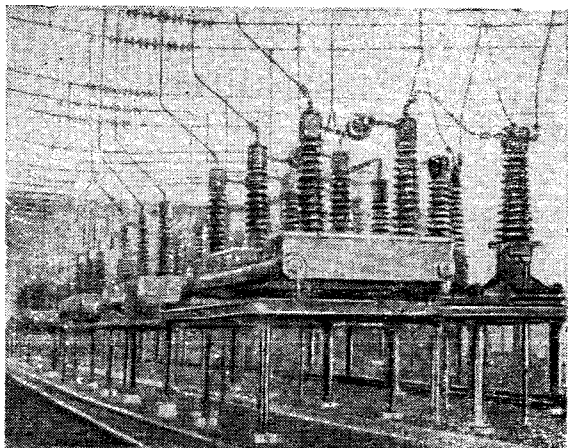


Рис. 8-15. Трехполюсный выключатель фирмы АЭГ с гашением дуги сжатым воздухом на 220 кв (1940 г.)

ки. Изоляторы поворачиваются, и гасительные головки расходятся одна от другой в горизонтальной плоскости. Этого типа выключатели снабжались устройством для автоматического повторного включения. Недостатком этой конструкции выключателя являлось то обстоятельство, что в процессе гашения дуги пары металла обгорающих контактов вдувались в зону дуги и тем затрудняли деионизацию дугового пространства.

#### в) Выключатели фирмы АЭГ выпуска 1952—1953 гг.

В журнале «ЕТЗ» за 1953 г. [Л. 8-12] помещена статья И. Бирманса, где он приходит к выводу, что современные выключатели с дутьем воздуха под давлением являются наиболее дешевым устройством, имеют наивысшую скорость

отключения и особо пригодны, когда по условиям надежности и бесперебойности подачи энергии необходимо осуществлять быстрые повторные включения. Кроме того, выключатели с гашением дуги сжатым воздухом наиболее безопасны в пожарном отношении. Одновременно он указывает, что требующиеся мощности отключения настолько быстро растут, что уже теперь в США требуются выключатели с мощностью отключения 15 000 *Mva*,

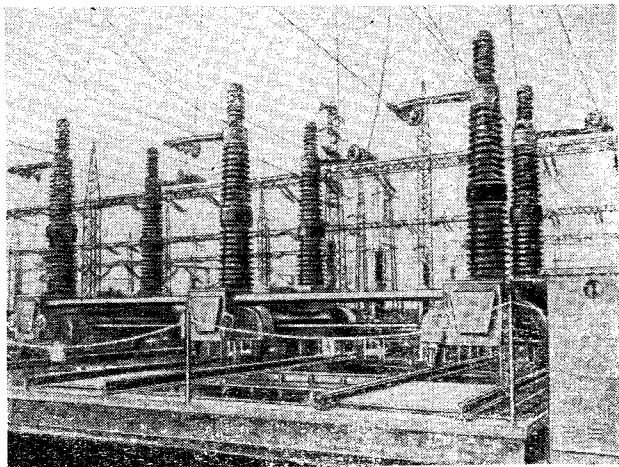


Рис. 8-16. Трехполюсный выключатель фирмы АЭГ с гашением дуги сжатым воздухом на 300 *кв*, 4 000 *Mva* на Рейнско-Вестфальской электростанции (1952 г.)

что соответствует отключаемому току 50 *ка* при рабочем напряжении 220 *кв*. Такие требования в отношении мощности отключения предъявлять к выключателю, по мнению Бирманса, нерационально. Является более целесообразным делить мощные системы на части или устанавливать ограничители токов короткого замыкания. Наиболее мощные выключатели с гашением дуги сжатым воздухом в 1952 г. установлены фирмой АЭГ на Рейнско-Вестфальской электрической станции, где была введена первая в Германии линия электропередачи между Браунвейлером и Рейнау 300 *кв* протяженностью 225 *км*. Для этой установки фирма АЭГ изготовила выключатели с гашением дуги сжатым воздухом с мощностью отключения 4 000 *Mva* (рис. 8-16).

### г) Выключатели фирмы АЭГ выпуска 1954 г.

Дальнейший ход развития энергетических систем не стал на путь, предложенный Бирмансом в части ограничения мощности коротких замыканий в системах.

Чтобы не отстать от других конкурирующих фирм, фирме АЭГ пришлось пересмотреть свои взгляды на этот вопрос и разработать новую конструкцию выключателя с гашением дуги сжатым воздухом на мощности отключения до 12 000 Мва. Это новое направление в работах фир-

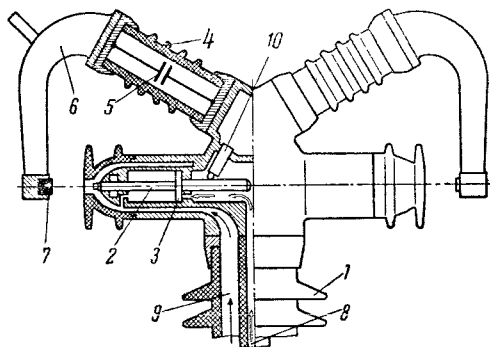


Рис. 8-17. Разрез гасительного устройства выключателя фирмы АЭГ с многократным разрывом цепи тока (1954 г.)

мы АЭГ освещено в статьях А. Хохрайнера и И. Бирманса — директоров «Института высоких напряжений фирмы АЭГ», помещенных в 1955 г. в журнале «ETZ» № 8 и 20. Из этих статей можно усмотреть, что фирма АЭГ стала на путь создания выключателя с многократным разрывом цепи тока, выполненного из ряда последовательно включенных гасительных камер, установленных на отдельных колоннах опорных изоляторов.

В отличие от ранее применявшихся грибообразных гасительных камер со свободной струей воздуха, у которых подвижные контакты разводились под действием давления воздуха на подпружиненный поршень, в новой конструкции гасительной камеры выключателя отсутствует подпружиненный поршень. Он заменен поршнем, движение которого осуществляется путем подвода сжатого воздуха по отдельным воздухопроводам к одной или другой стороне поршня.

На рис. 8-17 показан разрез двойной гасительной камеры этого выключателя в отключенном положении. На опорном изоляторе 1 установлена гасительная камера с двумя подвижными контактами 2, снабженными поршнями 3.

Наверху гасительной камеры помещены в фарфоровых крышках 4 делители напряжения, выполненные в виде конденсаторов 5. На концах фарфоровых крышек укреплены держатели 6 с неподвижными контактами 7.

При включении сжатый воздух подается по трубе 8, выполненной из изолирующего материала и установленной по центру опорного изолятора 1.

В процессе отключения сжатый воздух поступает по кольцевому пространству 9, образованному телом опорного изолятора 1 и трубой 8.

В отключенном положении защелка 10 защемляет подвижный контакт и тем предотвращает возможность произвольного выдвигания.

Ход каждого подвижного контакта около 200 мм, что обеспечивает достаточный воздушный промежуток для предотвращения перекрытия в выключенном состоянии под действием соответствующего нормам испытательного напряжения.

Наличие реактивного типа делителей напряжения и увеличенный ход подвижных контактов позволили отказаться от установки отделителя.

Первый трехполюсный выключатель этого типа был установлен фирмой АЭГ на подстанции Вейль-Добке Рейнско-Вестфальской электрической станции.

Выключатель имеет восемь разрывов цепи тока. Последовательно включенные гасительные камеры смонтированы на четырех колоннах опорных изоляторов. Отключаемая мощность 8 000 Мва.

Путем соответствующего набора гасительных камер фирма АЭГ разработала единую серию выключателей в следующем исполнении:

Т а б л и ц а 8-2

Напряжение, кв	110	220	380
Число разрывов на полюс . . . . .	4	8	12
Отключаемая мощность, Мва . . . . .	4 000	8 000	12 000

На рис. 8-18 показан один полюс выключателя на 380 кв, 12 000 Мва.

У выключателя этого типа подвижной и неподвижный контакты подвергаются непосредственному воздействию атмосферного воздуха, что будет неблагоприятно отражаться на их состоянии — на них может образоваться отложение льда. При отключенном положении выключателя подвижной контакт втягивается внутрь гасительной головки, и через сопло гасительной головки может попадать влага внутрь выключателя, что при понижении внешней температуры воздуха создаст ледяную пробку и сделает невозможным включение. Выключатель фирмы АЭГ в новом конструктивном исполнении вряд ли окажется пригодным для работы в наших климатических условиях.

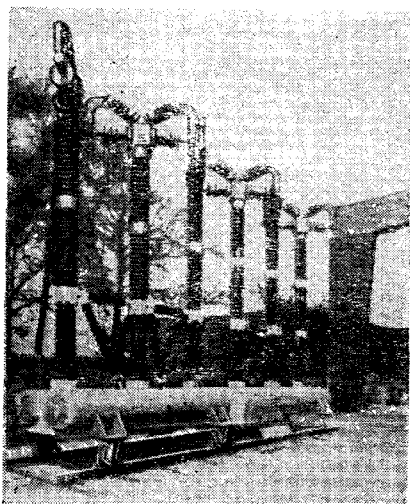


Рис. 8-18. Один полюс выключателя фирмы АЭГ с многократным разрывом цепи тока на 380 кв, 12 000 Мва (1955 г.)

В заключение обзора развития выключателей с гашением дуги сжатым воздухом фирмы АЭГ следует обратить особое внимание на успешное применение фирмой выключателей с грибообразными гасительными головками и со свободной струей воздуха. Как видно из рис. 8-4 и соответствующего к нему описания, конструкция выключателя со свободной струей воздуха была разработана Джексоном в 1905 г. В выключателе Джексона электрическая дуга образовывалась при расхождении контактов вне гасительной камеры. Струи сжатого воздуха подавались по осевому каналу, находящемуся в теле подвижного контакта выключателя.

Фирма АЭГ в своем выключателе с грибообразными гасительными головками подает сжатый воздух через кольцевую щель, образованную телом подвижного контакта и стенками сопла. Дуга гасится в момент прохождения конца подвижного контакта через отверстие сопла, т. е. в зоне наиболее интенсивного дутья.

Нельзя, конечно, утверждать, что конструкция выключателей фирмы АЭГ со свободной струей воздуха повторяет предложение Джексона, но известная преимущество в конструкции между обоими типами выключателей существует. Это можно видеть в следующих конструктивных особенностях:

а) Образование электрической дуги вне гасительной камеры с одновременным аксиальным обдуванием дуги струей сжатого воздуха:

б) Применение пневматического устройства для перемещения подвижного контакта. При этом подвижной контакт непосредственно связан со штоком поршня пневматического устройства.

в) Самостоятельное управление пневматическим устройством для передвижения подвижного контакта, что позволило осуществить выключатель с многократным разрывом цепи тока без отъединителя.

Приведенное сопоставление конструкций выключателей 1905 и 1955 гг. показывает, что старые идеи вновь становятся жизнеспособными для создания новых современных выключателей.

## **8-6. ВЫКЛЮЧАТЕЛИ ФИРМЫ БРАУН-БОВЕРИ**

Эта фирма строит выключатели для напряжения 400 кв с мощностью отключения 12 000 Мва.

### **Первые конструкции выключателей Браун-Бовери**

#### **1. Выключатель конструкции 1922 г.**

В фирменном журнале Браун-Бовери помещена статья «К истории выключателей переменного тока Браун-Бовери» [Л. 8-13].

В статье приведена фотография первого выключателя, построенного фирмой в 1922 г. (рис. 8-19).

Выключатель однополюсный с одним разрывом цепи тока. На крышке цилиндрического бака 1 установлен ввод 2, на конце металлического стержня которого укреплен торцового типа неподвижный контакт 3. Перегородка 4 снабжена в центре отверстием, через которое проходит трубчатый контакт 5, укрепленный на подпружиненном поршне 6. Во включенном положении выключателя контакт 5 плотно прижат к неподвижному контакту 3 натягом

пружины 7. При отключении по трубке 10 из резервуара подается сжатый воздух и одновременно освобождается защелка, которая удерживает контакт 5 во включенном положении. Под давлением сжатого воздуха контакт 5 отходит от неподвижного контакта 3 и между ними образуется электрическая дуга. При расхождении контактов к месту образования дуги устремляется струя сжатого воздуха, поступающая в трубчатый контакт 5 через

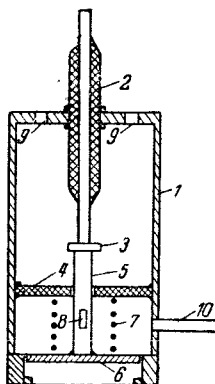


Рис. 8-19. Гасительная камера первого выключателя фирмы Браун-Бовери с гашением дуги сжатым воздухом (1922 г.)

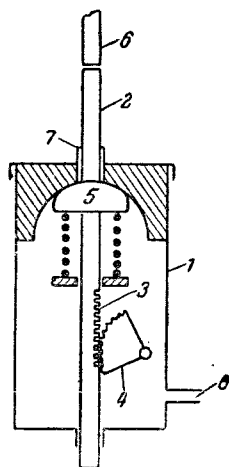


Рис. 8-20. Выключатель с гашением дуги сжатым воздухом фирмы Браун-Бовери со свободной струей и механическим передвижением контактов (1932 г.)

окно 8, прорезанное в его стенке. Ионизированные газы удаляются в атмосферу через отверстия 9. Давление воздуха составляет 0,6 ат.

Испытания показали, что выключатель отключал ток 1 400 а при 8 000 в. Время гашения дуги составляло один полупериод.

Характерной особенностью конструкции выключателя было то, что сам неподвижный контакт является одновременно тарелкой клапана, закрывающего выход сжатого воздуха из нижнего резервуара. Эта попытка фирмы Браун-Бовери сконструировать выключатель с гашением дуги сжатым воздухом показывает, что стремление применить в качестве дугогасящей среды воздух взамен масла имела место в Европе еще в начале двадцатых годов.

## 2. Выключатели конструкции 1930—1931 гг.

### а) Выключатель со свободной струей и механическим передвижением контактов

В 1932 г. фирма Браун-Бовери получила германский патент на выключатель с гашением дуги сжатым воздухом [Л. 8-14].

На рис. 8-20 показана схематически гасительная камера этого выключателя.

В баке 1, наполненном сжатым воздухом, помещается стержень подвижного контакта 2 с рейкой 3, с которой сцеплен сектор шестерни 4, передвигающей контакт 2. На стержне контакта 2 установлен подпружиненный клапан 5, закрывающий во включенном положении выключателя выход сжатого воздуха из бака 1. При отключении поворачивается сектор 4 и контакт 2 отходит от неподвижного контакта 6. Одновременно клапан 5 открывает кольцевую щель 7 между телом стержня контакта 2 и отверстием в крышке бака 1. Сжатый воздух из резервуара подается по трубке 8. Создается продольное дутье свободной струей воздуха, которое охлаждает дугу и деионизирует дуговое пространство. Эти выключатели не поступили в производство.

### б) Выключатель с гашением дуги в зоне повышенного давления воздуха

В 1931 г. фирма Браун-Бовери сделала заявку и в том же году получила германский патент на выключатель с гашением дуги в зоне повышенного давления воздуха [Л. 8-15].

В баке 1 (рис. 8-21) встроена перегородка 2, являющаяся неподвижным контактом и имеющая отверстие 3 для выхода сжатого воздуха из правой полости бака 1 в левую. Подвижной контакт 4 сплошной и одновременно является воздушным клапаном, открывающим выход сжатого воздуха из правой части бака 1 к месту образования дуги между расходящимися контактами.

Сжатый воздух в гасительную камеру подается из резервуара по трубке 5 и имеет давление 15 ат. При

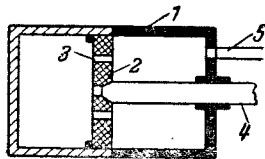


Рис. 8-21. Выключатель фирмы Браун-Бовери с гашением дуги в зоне повышенного давления воздуха (1931 г.) №

отключении выключателя сжатый воздух дросселируется через отверстие 3 до давления 10 ат, которое поддерживается в левой части бака 1 во время процесса отключения.

Такое конструктивное оформление выключателя было предложено фирмой для повышения электрической прочности воздуха в зоне образования дуги. По мнению фирмы, оно давало возможность создавать лучшие условия гашения дуги при значительно более высоких напряжениях отключаемого тока. Мысль поместить оба контакта выключателя в зону повышенного давления воздуха позднее была использована фирмой ДЖИИ при создании выключателей с гашением дуги сжатым воздухом.

в) Выключатели с одним разрывом цепи тока на полюс и пристроенным отъединителем

В 1935 г. фирма Браун-Бовери стала изготавливать выключатели для внутренней установки (рис. 8-22 и 8-23) и разработала серию выключателей на напряжение  $6 \div 50$  кв с мощностями отключения в пределах  $100 \div 600$  Мва [Л. 8-16].

Выключатель с одним разрывом цепи тока на фазу снабжен пристроенным отъединителем 1. При отключении сжатый воздух подается по трубке 9 в гасительную камеру и поршень 10 вместе с подвижным контактом 7 перемещается вниз. Образовавшаяся дуга между неподвижным 6 и подвижным 7 контактами деионизируется струей воздуха и в течение одного полупериода гаснет. Одновременно с подачей сжатого воздуха в гасительную камеру сжатый воздух подается в цилиндр 11 и тяга 3 рычагом 4 поворачивает вал 5, на котором насажен нож отъединителя 1. Отключение отъединителя производится после погасания дуги между контактами 6 и 7.

При включении сжатый воздух подается в цилиндр 2 и передвижением тяги 3 включается нож отъединителя. В отключенном положении выключателя в гасительной камере отсутствует давление, а потому контакты 6 и 7 оказываются замкнутыми под действием натянутой пружины 8.

В 1936 г. фирма Браун-Бовери разработала серию выключателей для внутренней установки напряжением до 64 кв, мощностью отключения 800 Мва.

Широкий выпуск на рынок выключателей с гашением дуги сжатым воздухом, снабженных пристроенными отъ-

единителями, является заслугой фирмы Браун-Бовери. Однако здесь следует отметить, что идея установки отъединителя у выключателя с гашением дуги сжатым воздухом была предложена Руппелем и на такие выключатели им были получены два германских патента. По первому

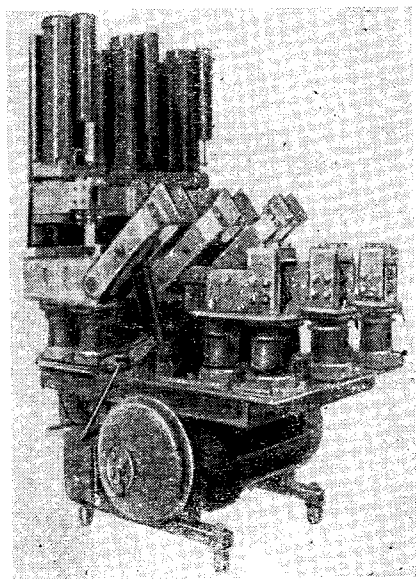


Рис. 8-22. Выключатель фирмы Браун-Бовери с гашением дуги сжатым воздухом при одном разрыве цепи тока на полюс и пристроенным отъединителем на 10 кв (1935 г.)

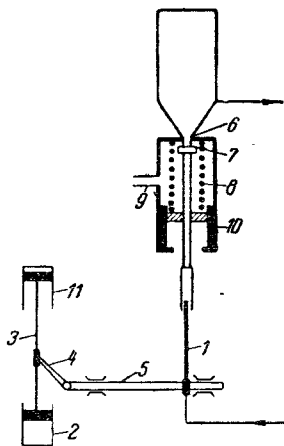


Рис. 8-23. Кинематическая схема и разрез гасительной камеры выключателя фирмы Браун-Бовери с гашением дуги сжатым воздухом и пристроенным отъединителем (1935 г.)

стержневым подвижным контактом отъединителя. В процессе отключения вначале размыкались контакты выключателя, а после гашения дуги при дальнейшем движении подвижного контакта освобождался подпружиненный отъединитель, который откидывался в сторону, создавая этим необходимый воздушный промежуток.

По второму германскому патенту Руппеля № 540152, выданному по заявке от 29 сентября 1929 г., предусматривалась установка у трехфазного выключателя с гашением дуги сжатым воздухом трех отъединителей ножевого типа. Поворот ножей происходил в горизонтальной плоскости.

патенту № 556808 по заявке от 23 августа 1928 г. подвижной контакт выключателя жестко заблокирован со

Предложение Руппеля устанавливать отъединители нашло впервые широкое применение не на родине, а в Швейцарии у выключателей фирмы Браун-Бовери, а затем получило распространение во всем мире.

### 3. Выключатели колонкового типа с одним и двумя разрывами цепи тока

В 1937 г. фирма Браун-Бовери стала строить выключатели колонкового типа с одним и двумя разрывами цепи тока для наружной установки в сетях с напряжением 64, 87 и 150 кВ, мощностью отключения 600 ÷ 1 800 Мва.

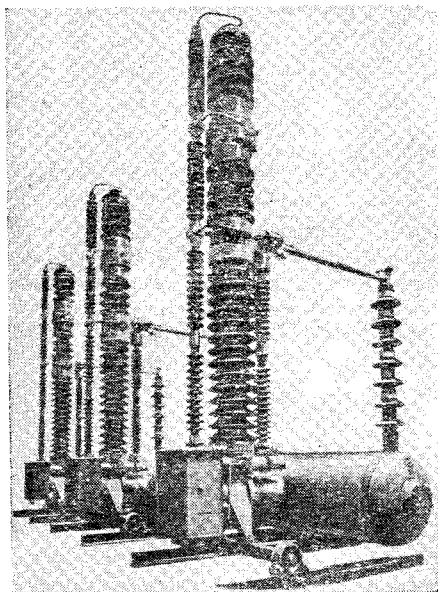


Рис. 8-24. Выключатель колонкового типа фирмы Браун-Бовери с гашением дуги сжатым воздухом при двух разрывах цепи тока и автоматическим повторным включением на 150 кВ, 1 800 Мва с пристроенным отъединителем (1937 г.)

Выключатели снабжались пристроенными отъединителями и были приспособлены для автоматического повторного включения (рис. 8-24, 8-25).

Гасительная камера имеет два разрыва цепи тока на фазу. Около главной колонны / выключателя, через кото-

рую подается дутье в гасительную камеру, установлен дополнительный воздухопровод 2. Он служит для подачи сжатого воздуха в полость 3, где расположены поршни 4, укрепленные на нерабочих концах подвижных контактов 5.

При отключении подается сжатый воздух из резервуара 6 в гасительные камеры 7. После первого отключения выключателя и гашения дуги срабатывает управляющее устройство для быстрого повторного включения. В этом случае по дополнительному воздухопроводу 2 подается сжатый воздух в полость 3 и поршни 4 передвинут и вновь замкнут контакты выключателей 5 и 8. Если короткое замыкание в сети не исчезло, то вновь срабатывает защитное реле и прекращается впуск сжатого воздуха по дополнительному воздухопроводу 2, и контакты 5 и 8 вторично разомкнутся. Вслед за этим произойдет поднятие ножа 9 отъединителя.

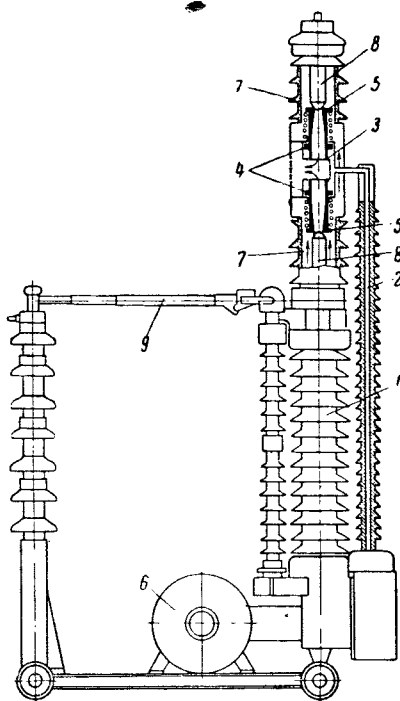


Рис. 8-25. Разрез выключателя на 150 кв, 1800 Мва фирмы Браун-Бовери с гашением дуги сжатым воздухом с автоматическим повторным включением и пристроенным отъединителем (1937 г.)

В 1939 г. был построен выключатель на 220 кв, гасительные камеры которого имели четыре разрыва цепи тока на фазу (рис. 8-26). Выключатель не имел устройства для автоматического повторного включения. Он был рассчитан на рабочий ток 630 а и имел мощность отключения 2500 Мва. По своей конструкции он представлял собой спаренный выключатель на 110 кв и имел четыре гасительные камеры, установленные попарно на двух колонках с общим отъединителем, находящимся между колонками выключателя [Л. 8-17]. В отличие от ранее описанных вы-

ключателей он имел делители напряжения для выравнивания восстанавливающегося напряжения между отдельными местами разрыва цепи тока. При наличии четырех мест разрыва цепи тока на каждое место разрыва приходилось

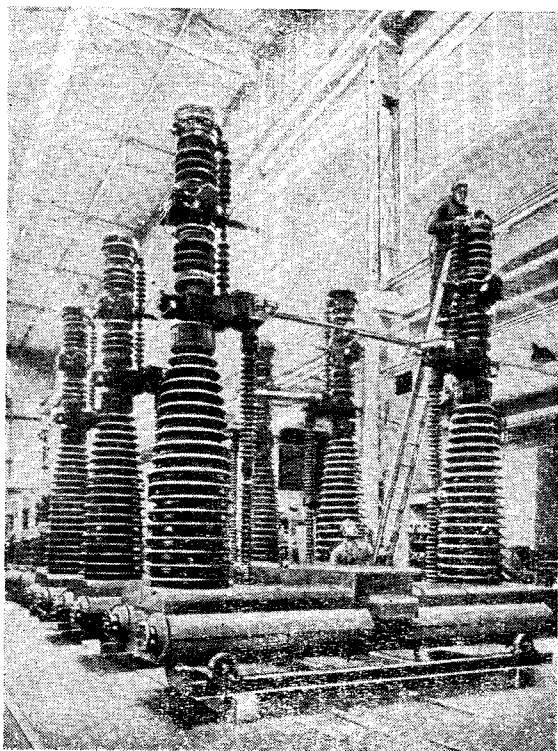


Рис. 8-26. Сдвоенный выключатель на 220 кВ, 2500 Мва, фирмы Браун-Бовери с четырьмя разрывами цепи тока с гашением дуги сжатым воздухом и внешним отъединителем (1939 г.)

восстанавливающееся напряжение свыше 50 кВ, а при наблюдающихся в процессе отключения перенапряжениях оно еще могло значительно возрастать. Изготовленные сдвоенные выключатели на 220 кВ оказались весьма тяжелыми и дорогими.

Перед фирмой встал вопрос о создании новой, более совершенной конструкции выключателя. Взамен выключа-

телей с небольшим числом мест разрыва цепи тока фирма стала строить выключатели с гашением дуги сжатым воздухом и многократным разрывом цепи тока.

#### 4. Колонковые выключатели с многократным разрывом цепи тока

К проектированию колонкового выключателя с многократным разрывом цепи тока было приступлено в 1937 г. Характерной особенностью нового выключателя является увеличение числа мест разрыва цепи тока, примерно в 2 раза по сравнению с аналогичными по напряжению выключателями серии 1937 г.

Кроме того, новая серия выключателей должна была комплектоваться для всех напряжений из одних и тех же основных элементов и узлов.

В связи с большим числом мест разрывов в гасительной камере остро вставал вопрос об устройствах для выравнивания распределения восстанавливающегося напряжения между отдельными местами разрыва. Решение вопроса было найдено фирмой в постановке активных и реактивных делителей восстанавливающегося напряжения. На эту конструкцию выключателя с двойным делителем восстанавливающегося напряжения фирма в 1940 г. сделала заявку и получила в Германии патент [Л. 8-18], о чем было подробно изложено в гл. 6 этой книги.

По внешнему виду такой выключатель имел вид колонны, почему и получил свое название — колонковый (рис. 8-27) [Л. 8-19].

Гасительная камера состоит из четырех отдельных частей 1, в которые вмонтированы отключающие элементы, каждый с одним разрывом цепи тока.

Число отключающих элементов выбирается в зависимости от требующейся мощности отключения и в соответствии с величиной рабочего напряжения. В каждом отключающем элементе имеется неподвижный контакт 2, выполненный в виде гнезда клапана с соплом, и сплошной подпружиненный подвижной контакт 3, закрывающий отверстие неподвижного контакта. Сжатый воздух из резервуара подводится к отдельным местам разрыва цепи тока по каналу 4. Образовавшиеся газы при отключении выключателя отводятся в атмосферу через отверстия 5.

При отключении выключателя сплошные подпружиненные контакты 3 отжимаются давлением воздуха от неподвижных контактов. Образовавшиеся дуги в местах расхождения контактов охлаждаются струями воздуха, и

умолчанию в соответствии с первой буквой идентификатора, обозначающего точку входа в процедуру-функцию. В этом случае данный оператор DECLARE выполняет лишь одну функцию — объявляет в вызывающем блоке M как имя процедуры. Такой оператор

```
DECLARE M ENTRY;
```

обязателен, если процедура-функция M не имеет параметров.

Пр и м е р. Следующий фрагмент программы представляет собой процедуру-функцию для вычисления  $N! = 1 \cdot 2 \cdot \dots \cdot N$ :

```
NF: PROCEDURE (N) DECIMAL FIXED;
  DECLARE N FIXED (10);
  IF N <= 1 THEN RETURN (1);
  F = 1; DO I = 2 TO N; F = F * I; END;
  RETURN (F); END NF;
```

Здесь формальный параметр N описан явно оператором объявления данных. Описатели DECIMAL FIXED в операторе PROCEDURE характеризуют вычисляемое значение, которое задается в двух операторах возврата RETURN. Эти же описатели подразумеваются у идентификатора NF, обозначающего точку входа (метку процедуры-функции).

Отметим, что первая строка в приведенном примере допускает другую форму записи:

```
NF: PROCEDURE (N) RETURNS (DECIMAL FIXED);
```

где описатели, указанные в скобках, характеризуют вычисляемое значение. В вызывающем блоке следует написать оператор

```
DECLARE NF ENTRY RETURNS (DECIMAL FIXED);
```

а само обращение может выглядеть, например, так:

```
MN = NF(10) + 10;
```

где константа 10, указанная в скобках — фактический параметр. В этом операторе вычисляется значение переменной MN как  $10! + 10$ .

Если в списке фактических параметров указано имя входа в процедуру со списком своих фактических параметров или же указано в скобках имя функции без параметров, то считается, что такое имя входа представляет функцию и в список фактических параметров соответствующей вызываемой процедуры передается вычисленное значение. Например, операторы

```
CALL P(F(A,B),N);
CALL P((E),N);
```

где F и E — функции, описание которых имеет вид

```
F: PROCEDURE (A,B); ...;
E: PROCEDURE; ...;
```

передадут в вызываемую процедуру значения функций F и E соответственно.

Если в списке фактических параметров имеется имя входа, которое не заключено в скобки, то в вызываемую процедуру передается это имя входа. Например, в фрагменте программы

```
DECLARE E ENTRY; ...; CALL P(E,N);
```

в процедуру P будет передано имя входа E.

4. Процедура-подпрограмма. При обращении к функции в вызывающий блок передается (возвращается), как правило, одно вычисленное значение. Если же процедурный блок используется для вычисления набора значений, его следует оформить как *процедуру-подпрограмму*.

В операторе PROCEDURE для подпрограммы, как и для функции, указаны формальные параметры, но в нем нет описателей, поскольку в операторе возврата RETURN не указывается выражение, значение которого возвращается в вызывающую программу. Для возврата в вызывающую программу можно также использовать оператор конца END, замыкающий данную процедуру. Действие этого оператора полностью эквивалентно выполнению оператора возврата RETURN

Значения, которые вычисляются в подпрограмме и возвращаются в вызывающий блок, соответствуют формальным параметрам. Части формальных параметров, как и для функции, соответствуют фактические входные данные.

При обращении к подпрограмме оператором CALL указываются на месте входных данных идентификаторы, определенные в вызывающем блоке. Идентификатор точки входа в операторе CALL считается описанным в вызывающей программе контекстуально, т. е. появление идентификатора в операторе CALL есть его описание.

Описатели формальных параметров и описатели фактических параметров процедуры могут быть согласованы описанием фактических параметров в вызывающей процедуре оператором объявления данных с теми же описателями, что и у формальных параметров вызываемой подпрограммы. В то же время такое согласование описателей может быть выполнено указанием в вызывающей процедуре оператора

```
DECLARE M ENTRY (LFP);
```

где M — имя вызываемой процедуры, LFP — список требуемых описателей соответствующих формальных параметров, ENTRY — описатель имени входа. Описатели каждого формального параметра отделяются в списке LFP от описателей другого формального параметра запятой, при этом в случае уже имеющей место согласованности описателей в соответствующей позиции списка LFP описатели опускаются, однако все разделяющие запятые должны быть сохранены.

Пример. Процедура-подпрограмма, в которой вычисляются значения  $F=N!$  и  $L=(N!)^2+N!+N$ , имеет вид

```
NFL: PROCEDURE (N,F,L);
  DECLARE (N,F,L,I) DECIMAL FIXED;
  IF N=0 THEN DO; F=1; L=2; RETURN; END;
  IF N=1 THEN DO; F=1; L=3; RETURN; END;
```

```
F=1; DO I=2 TO N; F=F*I; END;
L=F**2+F+N; RETURN; END NEL;
```

Из вызывающего блока, начинающегося оператором

```
P: PROCEDURE;
```

в котором содержится, например, описание

```
DECLARE (FF,LF) DECIMAL FIXED;
```

или же оператор

```
DECLARE NFL ENTRY (,DECIMAL FIXED,DECIMAL FIXED);
```

можно обратиться к данной процедуре оператором

```
CALL NFL(10,FF,LF);
```

в результате которого идентификатор FF получит значение 10!, а LF — значение  $(10!)^2 + 10! + 10$

В этом примере в списке описателей после слова ENTRY первый элемент может быть опущен, так как первый фактический параметр задается десятичным числом в форме с фиксированной точкой, т. е. имеет место согласованность описателей формального и фактического параметров.

Если процедуру, к которой производится обращение, разместить непосредственно в вызывающем блоке, то к ней можно обратиться, не задавая фактических параметров. Так, в приведенном примере, если процедура вычисления F и L содержится в том блоке, из которого она вызывается, можно оператор

```
DECLARE (N,F,L,I) DECIMAL FIXED;
```

перенести в вызывающий блок, одновременно разместив там операторы

```
N=10; FF=F; LF=L;
```

т. е. процедурный блок P примет вид

```
P: PROCEDURE; ...;
  DECLARE (FF,LF,N,F,L,I) DECIMAL FIXED; ...;
NFL: PROCEDURE;
  IF N=0 THEN DO; F=1; L=2; RETURN; END;
  IF N=1 THEN DO; F=1; L=3; RETURN; END;
  F=1; DO I=2 TO N; F=F*I; END;
  L=F**2+F+N; RETURN; END NFL;
  N=10; CALL NFL;
  FF=F; LF=L; ...; END P;
```

Здесь процедура вычисления значений  $F=N!$  и  $L=(N!)^2 + N! + N$  не содержит параметров и обозначена меткой NFL.

Любая подпрограмма может вызывать другие подпрограммы, в том числе и саму себя, т. е. может быть рекурсивной. Последнее означает, что при выполнении процедурного блока может встретиться обращение к имени входа

этого же процедурного блока. В этом случае у оператора PROCEDURE указывается описатель RECURSIVE, который относится ко всем входам этой процедуры. Например, рекурсивный вариант процедуры вычисления  $N!$  может быть составлен так:

```
NFR: PROCEDURE (N) DECIMAL FIXED RECURSIVE;
      DECLARE N FIXED (10);
      DECLARE NFR RETURNS (DECIMAL FIXED);
      IF N <= 1 THEN RETURN (1);
      F=NFR(N-1)*N;
RETURN (F); END NFR;
```

**5. Оператор входа в процедуру.** Оператор *входа в процедуру* указывает дополнительный вход в процедурный блок и имеет вид

```
M: ENTRY (FP);
```

где M и FP имеют тот же смысл, что и в операторе PROCEDURE.

Оператор входа в процедуру задает точку входа, в которую передается управление при вызове процедуры. Поэтому оператор ENTRY, как и оператор PROCEDURE, должен иметь по крайней мере одну метку. Эти метки называются именами точек входа. При любом обращении к процедуре задается какое-либо имя точки входа и тем самым указывается, с какого места следует выполнять данную процедуру. Когда процедура вызывается по имени, заданному в операторе ENTRY, обработка начинается с первого исполняемого оператора, следующего после оператора ENTRY. Оператор входа при обычном последовательном выполнении операторов процедуры во внимание не принимается.

Оператор входа в процедуру не может быть внутренним по отношению ни к какому блоку, содержащемуся в этой процедуре; он не может также содержаться внутри группы, для которой в операторе задан цикл. При вызове процедуры список фактических параметров должен быть согласован со списком формальных параметров для соответствующей точки входа. Списки формальных параметров, заданные в различных точках входа процедуры, могут не совпадать друг с другом.

*Пример.* Рассмотрим процедурный блок

```
P1: PROCEDURE(A,B,C); ...; B1: BEGIN; ...; END B1;
E1: ENTRY (D,E); ...; P2: PROCEDURE, ...; END P2;
E2: ENTRY; ...; END P1;
```

Обращение к этой процедуре с помощью оператора вызова процедуры возможно в трех вариантах:

```
CALL P1(X,Y,Z); CALL E1(T,P); CALL E2;
```

где X, Y, Z, T, P — фактические параметры.

Оператор вызова процедуры может быть использован в описателе INITIAL при присваивании начальных значений идентификаторам. Например, в операторе

```
DECLARE D(16) INITIAL CALL DATA (A,B,C);
```

присваивание начальных значений элементам массива D производится путем обращения к подпрограмме DATA. Здесь A, B, C — фактические параметры. Массив D определяется в DATA заранее либо он передается как параметр. После обработки вызванной процедуры управление передается на оператор INITIAL.

Операторы BEGIN, PROCEDURE, ENTRY, END являются операторами структуры программы. К ним можно отнести также оператор DO, одновременно выполняющий функции управления.

#### Упражнения

1. Указать, каковы будут значения элементов массива R после выполнения следующей программы. HBOUND(A, 1) — встроенная функция, результатом которой является целое число, равное текущему максимальному значению индекса одномерного массива A.

```
A: PROCEDURE OPTIONS (MAIN);
  DECLARE R(4) FIXED(2);
  DO J=1 TO 4; R(J)=B(J)+C(J); END;
  CALL D(R,07);
B: PROCEDURE (K); K=K*2;
C: ENTRY (K); RETURN(K-2); END B;
D: PROCEDURE (A,L); DECLARE (L,A(*) ) FIXED (2);
  DO J=1 TO HBOUND (A,1); IF A(J)>L THEN
  RETURN; A(J)=A(J)+1;
END A;
```

2. Составить описание процедуры-функции для упорядочивания последовательности N чисел в порядке их возрастания.

3. Оформить в виде процедуры-функции алгоритм умножения многочлена степени n на многочлен степени m.

4. Составить процедуру-функцию для вычисления значения  $(k+l)/(k+l!)$  (k, l — натуральные числа).

5. Составить подпрограмму, которая вычисляет координаты точки пересечения прямых, заданных уравнениями

$$\begin{aligned} a_1x + b_1y &= c_1, \\ a_2x + b_2y &= c_2. \end{aligned}$$

Если прямые параллельны или совпадают, то подпрограмма должна передать управление оператору с меткой NO.

6. Составить процедуру-функцию, которая вычисляет сумму квадратов n ее аргументов, где  $1 \leq n \leq 5$ . Для каждого значения n предусмотреть отдельную точку входа в процедуру.

## § 11. Встроенные функции

Наиболее часто встречающиеся математические процедуры представлены в виде составной части языка и получили название *встроенных функций*. К числу таких функций относятся математические функции, арифметические функции, функции для обработки строк, функции для обработки массивов, функции специального назначения.

Записывается встроенная функция с помощью идентификатора и стоящего за ним аргумента (аргументов) в скобках. Идентификаторы встроенных функций заранее predeterminedены, как и форма представления и свойства

аргументов и получаемых значений. Используются встроенные функции (в зависимости от назначения функций) в выражениях в качестве операндов как обычные переменные.

К математическим функциям относятся тригонометрические и гиперболические функции, функции извлечения квадратного корня, возведения в степень и логарифмические функции. Аргументы математических функций могут быть скалярными выражениями и массивами и должны быть представлены в форме с плавающей точкой. В противном случае перед вызовом функции они преобразуются в эту форму. Если аргументом является массив, то результатом обращения к функции будет также массив с размерностью и границами, как у аргумента; функция выполняется над каждым элементом массива. Значение, вычисляемое математической функцией, есть число с плавающей точкой, с основанием и разрядностью аргумента.

Арифметические функции имеют аргументы с описателями `FLOAT` или `FIXED` и `DECIMAL` или `BINARY`. В случае отсутствия этих описателей преобразование аргумента производится автоматически. Аргумент может быть скалярным выражением или массивом. Если аргументом является массив, результат выполнения функции есть также массив с описателями аргумента. Если не указаны описатели результата, они будут такими же, как и у аргумента.

Аргументы функций для обработки строк представляют собой строки символов, или массивы, состоящие из строк символов. Те аргументы, которые не являются строками символов, перед вызовом функции преобразуются в строку битов, или в символьную строку.

Функции для обработки массивов в качестве аргументов содержат массивы. Результат выполнения функции является скалярным значением, поэтому обращение к любой функции для обработки массивов рассматривается как скалярное выражение.

К функциям специального назначения относятся функции, которые не объединены общим назначением и не связаны с другими классами встроенных функций. В их число входят функции-ситуации, не содержащие аргументов и принимающие значения, характеризующие прерывания; эти функции могут употребляться только в операторах, входящих в оператор `ON`, или блоках, вызываемых оператором `ON`.

Некоторые из встроенных функций могут быть использованы слева от знака присваивания в операторе присваивания, в качестве параметра цикла в операторе цикла или же как элемент списка данных в операторе ввода, вследствие чего их называют псевдопеременными. Например, псевдопеременной является встроенная функция `COMPLEX(X,Y)`, где  $X, Y$  — идентификаторы, возможно, с различными описателями. Так, в результате выполнения оператора `COMPLEX(X,Y)=E`; вещественная часть выражения  $E$  присваивается переменной  $X$ , мнимая —  $Y$ . Если  $E$  есть  $5.1+7.5I$ ,  $X$  будет равно  $5.1$ ,  $Y$  —  $7.5$ . Теперь выполнение оператора присваивания `C=COMPLEX(X,Y)`; где  $C$  — комплексная переменная, приведет к тому, что  $C$  получит значение  $5.1+7.5I$ ; здесь конструкция `COMPLEX(X,Y)` выступает как арифметическая функция, формирующая комплексное число.

Список встроенных функций зависит от версии языка, транслятора, и может пополняться новыми функциями. Встроенные функции перечисляются в описании каждого транслятора; их список с различной степенью полноты содержится, например, в [6, 8, 10], см. также приложение III.

## § 12. Обработка прерываний

1. Оператор задания режима обработки прерываний. В различных исключительных случаях выполнение программы может прерываться. Это происходит, если оператор, записанный в программе, невыполним. Для управления выполнением программы в таких случаях, иначе, при возникновении ситуации прерывания, особенно в процессе отладки программы, используется оператор задания режима обработки прерываний, записываемый в двух разновидностях:

```
ON C SNAP S;  
ON C SYSTEM;
```

где ON — ключевое слово оператора, C — ключевое слово, обозначающее ситуацию, при возникновении которой управление передается оператору S, указанному после дополнения SNAP и определяющему режим обработки ситуации прерывания. На месте оператора S может стоять непоименованный обычный блок (но не процедурный блок) или одиночный оператор (но не группа), за исключением операторов DECLARE, END, DO, FORMAT, RETURN.

В записи оператора ON дополнение SNAP может быть опущено, однако если оно указано, печатается определенная информация о состоянии программы, позволяющая оценить ситуацию прерывания и полезная при отладке. Выдача этой информации происходит до обработки ситуации прерывания, т. е. до выполнения оператора S.

Оператор ON с дополнением SYSTEM указывает, что имеет место стандартная реакция операционной системы на данную ситуацию, или стандартный режим обработки прерывания. В такой форме оператор ON используется обычно тогда, когда есть необходимость отменить действие предыдущего оператора задания режима обработки прерываний и по умолчанию передать управление операционной системе.

Реакция на данную ситуацию прерывания, заданная оператором ON, сохраняет силу в пределах того блока, в который входит этот оператор ON. Режим обработки ситуации прерывания распространяется также на все внутренние блоки и все вызванные из данного блока процедурные блоки.

Выполнение оператора ON заключается в том, что для соответствующей ситуации прерывания устанавливается указанный режим обработки. Выполнение следующего оператора ON для такой же ситуации прерывания задает для нее другой режим обработки. При этом, если новый оператор ON выполняется в блоке, вызванном данным блоком, режим обработки, заданный предыдущим оператором ON, временно отменяется и восстанавливается лишь при завершении работы блока, содержащего новый оператор ON, или же в результате выполнения оператора REVERT (см. примеры в пп. 3 и 8 этого параграфа).

Предусмотрены следующие ситуации прерывания: реакции системы, вычислительные ввода и вывода, контроля программы, обработки списков, ситуация, определяемая программистом. Для ситуаций контроля программы и вычислительных ситуаций имя ситуации как средство ее включения задается в программе в виде приставки к операторам (см. п. 6 этого параграфа); остальные ситуации включаются по умолчанию.

**2. Ситуации реакции системы.** Ситуациями реакции системы являются ERROR и FINISH; последняя возникает, когда программа заканчивается и управление будет передано операционной системе, ситуация ERROR — когда программа заканчивается необычным образом, например из-за ошибки в программе. Если для данной ситуации предусмотрена реакция на прерывание, то после выполнения этой реакции возникает ситуация FINISH.

**3. Вычислительные ситуации.** К вычислительным ситуациям относятся ситуации со следующими именами: CONVERSION, FIXEDOVERFLOW, OVERFLOW, UNDERFLOW, ZERODIVIDE.

Ситуация CONVERSION возникает при преобразовании строки символов, содержащей символы, отличные от 0 и 1, в строку битов при наличии недопустимых символов. Результат не определен. Стандартная реакция системы — печать сообщения об ошибке (сигнализация) и возникновение (вызов) ситуации ERROR.

Ситуация FIXEDOVERFLOW возникает при выполнении арифметических операций над числами с фиксированной точкой, если результат операции выходит за пределы числового поля (15 разрядов для описателя DECIMAL и 31 бита в случае описателя BINARY). Результат ускается слева. Стандартная реакция системы — сигнализация и продолжение счета.

Ситуация OVERFLOW возникает при выполнении арифметических операций над числами с плавающей точкой, если порядок результата превышает максимально допустимый (75 при основании 10 или 252 при основании 2). Результат не определен. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация UNDERFLOW возникает при выполнении арифметических операций над числами с плавающей точкой, если порядок результата меньше допустимого минимума (—79 при основании 10 или —260 при основании 2), и не возникает при вычитании равных чисел. Результат равен нулю. Стандартная реакция системы — сигнализация, продолжение счета.

Ситуация ZERODIVIDE возникает при попытке деления на нуль. Результат не определен. Стандартная реакция системы — сигнализация, ситуация ERROR.

Пр и м е р. При выполнении программы

```
P: PROCEDURE OPTIONS (MAIN); ...;
  ON OVERFLOW GO TO M; ...;
  CALL P1;
P1: PROCEDURE; ...;
  ON OVERFLOW GO TO M1; ...;
END P1; ...; END P;
```

все переполнения, происходящие в процедуре P до появления оператора ON, вызывают стандартную реакцию системы. Переполнение, происшедшее после выполнения оператора ON, приведет к передаче управления оператору процедуры P с меткой M. Это положение сохраняется до выполнения оператора ON, вызванного оператором CALL P1; процедурного блока P1. С этого момента и до тех пор, пока не закончится выполнение процедуры P1, переполнение будет приводить к передаче управления оператору с меткой M1, который может находиться как в процедуре P1, так и в процедуре P. После выполнения процедуры P1 действие оператора

ON OVERFLOW GO TO M;

восстанавливается до окончания выполнения программы.

4. **Ситуации ввода и вывода.** К *ситуациям ввода и вывода* относятся ситуации со следующими именами: ENDFILE, ENDPAGE, KEY, NAME, RECORD, TRANSMIT, UNDEFINEDFILE. Ситуации ввода и вывода всегда относятся к конкретному файлу, имя которого F указывается в скобках после соответствующей ситуации прерывания, например NAME (F).

Ситуация ENDFILE возникает при попытке считать из указанного файла что-либо, находящееся за ограничителем этого файла, т.е. после того, как прочитана последняя запись. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация ENDPAGE возникает при попытке печати за последней строкой на странице. Стандартная реакция системы — начинается новая страница.

Ситуация KEY возникает, если заданный признак (ключ) не обнаружен (при вводе) или задан уже существующий признак (при выводе). Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация NAME возникает, если при вводе появляется нераспознаваемый идентификатор или имя, отсутствующее в списке данных. Стандартная реакция системы — сигнализация, игнорируется присваивание для отсутствующего имени.

Ситуация RECORD возникает, если фактическая длина записи не соответствует длине, указанной в объявлении файла. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация TRANSMIT возникает при наличии ошибки в операторах ввода или вывода, т.е. при передаче данных. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация UNDEFINEDFILE возникает при попытке открыть файл, который не определен, или когда совокупность описателей файла противоречива. Стандартная реакция системы — сигнализация, ситуация ERROR.

5. **Ситуации контроля программы и обработки списков.** К *ситуациям контроля программы* относятся ситуации с именами: CHECK, SIZE, STRING-RANGE, SUBSCRIPTRANGE.

Ситуация CHECK (L), где L — список элементов (каждый элемент списка может быть идентификатором скалярной величины, массива, или структуры, константой типа метки, или меткой входа), возникает каждый раз, когда выполняется оператор, метка которого содержится в списке L, или изменяется переменная из списка элементов, т.е. ситуация заключается как бы

в проверке элементов списка L. Стандартная реакция системы — печатаются элемент и (в случае переменной) его новое значение.

Ситуация SIZE возникает, если значение, полученное при вводе или в результате выполнения оператора присваивания, не помещается в отведенном поле. Результат не определен. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация STRINGRANGE возникает, если при обращении к встроенной функции с идентификатором SUBSTR допущена ошибка в длине списка элементов. Стандартная реакция системы — сигнализация, длина списка преобразуется в допустимые границы, программа продолжает выполняться.

Ситуация SUBSCRIPTRANGE возникает, когда вычисленное значение индекса находится вне указанных для него границ. Результат не определен. Стандартная реакция системы — сигнализация, ситуация ERROR.

6. Использование ситуаций прерывания в программе. Автоматически устанавливается режим игнорирования ситуаций SIZE, SUBSCRIPTRANGE, CHECK, однако при необходимости реакция системы на каждую из этих ситуаций может быть включена. Реакция на ситуации UNDERFLOW, OVERFLOW, ZERODIVIDE, CONVERSION, FIXEDOVERFLOW обычно включена, но может выключаться программистом. Реакция на все остальные ситуации включена всегда и не подвластна программисту.

Включение реакции на ситуацию осуществляется указанием имени ситуации в списке имен ситуаций (см. § 3) перед меткой оператора. Указанное включение относится только к данному оператору (или процедуре). Например:

(SIZE): M: A = (B-3)\*4;

Выключение реакции на ситуацию осуществляется добавлением к имени ситуации приставки NO и записью полученного ключевого слова в списке имен ситуаций перед оператором. Например, запись вида

(NOUNDERFLOW,SIZE): P: Y=3.14-X;

обеспечивает реакцию на ситуацию SIZE и отсутствие реакции на ситуацию UNDERFLOW.

Действие приставки может быть отменено, если написать имя ситуации непосредственно перед данным оператором. Например, в следующем фрагменте программы:

(NOOVERFLOW): P: PROCEDURE; ...;

(OVERFLOW): S: Y=2.3\*(X+1); ...; END;

для всей процедуры реакция на ситуацию OVERFLOW отменяется, за исключением одного оператора с меткой S.

Область действия ситуаций прерывания, указанной перед оператором присваивания, распространяется только на данный оператор, но не на одну из процедур, которые этот оператор вызывает, а в случае условного оператора — на выражение, следующее за ключевым словом IF. Отметим, что перед конструкциями, начинающимися с дополнений THEN и ELSE, могут быть свои собственные имена ситуаций. Если ситуация прерывания указана перед оператором DO, то область ее действия распространяется на выражения в заго-

ловке цикла, но не на операторы группы DO. В случае блоков PROCEDURE и BEGIN ситуация прерывания оказывает действие на все операторы вплоть до оператора END, включая все вложенные блоки. На вызываемые процедуры, лежащие вне блока, влияние ситуации прерывания, указанной перед данным блоком, не распространяется.

7. Оператор имитации ситуации прерывания. Оператор, записываемый в виде

SIGNAL C;

где C — имя ситуации прерывания, вызывает немедленное возбуждение указанной в нем ситуации и управление передается оператору ON, включающему в свой состав данную ситуацию C и выполненному к моменту появления оператора SIGNAL. Этот оператор позволяет упростить процесс отладки сложной программы, так как создается эффект действительного возникновения указанной ситуации и порядок выполнения операторов изменится. После выполнения действия по прерыванию управление возвращается к оператору, следующему сразу за оператором SIGNAL.

Если к моменту выполнения оператора SIGNAL указанная в нем ситуация не включена (не был выполнен оператор ON, или такого оператора вообще нет в программе, или реакция на данную ситуацию выключена), прерывания не происходит. Например, если некоторый фрагмент программы содержит операторы

M: PROCEDURE; P: ON ENDFILE (F) Y,Z=0; ...;

S: SIGNAL ENDFILE (F); Q: X=1; ...;

T: SIGNAL SIZE; ...; END M;

то оператор SIGNAL с меткой S вызовет прерывание так же, как если бы действительно произошла попытка считывания чего-либо, находящегося за ограничителем файла. Управление будет передано оператору, указанному в ON с меткой P, после чего выполнится оператор с меткой Q. Оператор SIGNAL с меткой T прерывания не вызывает.

8. Оператор отмены реакции на прерывание. Оператор, имеющий вид

REVERT C;

где C — имя ситуации, отменяет любой режим обработки прерывания для ситуации C, заданный в текущем блоке, т.е. отменяется действие оператора ON с той же ситуацией C в пределах блока, в котором содержатся как оператор ON, так и оператор REVERT с одной и той же ситуацией C.

После выполнения оператора отмены реакции на прерывание устанавливается тот режим обработки, который действовал в момент входа в данный блок. Если же в данный момент для какой-либо ситуации не задан никакой режим обработки прерывания, то при возникновении этой ситуации будет применяться стандартный режим обработки прерывания. Например, в процедуре

PR: PROCEDURE; ...;

REVERT ZERODIVIDE; ...;

R: ON ZERODIVIDE GO TO P; ...; END PR;

оператор с меткой R воспринимается как пустой оператор, и в случае деления на нуль возникнет стандартная реакция системы ERROR.

**9. Оператор вывода на дисплей.** Для выдачи на пульт машины определенной информации, например директив для программиста, сообщений о выполнении отдельных блоков программы и др., а также для прерывания выполнения программы используется оператор *вывода на дисплей*, имеющий вид

DISPLAY (E) REPLY (V);

где E — скалярное выражение, V — строковая переменная, REPLY — дополнение. Конструкция REPLY (V) может отсутствовать. В этом случае оператор DISPLAY выдает на пульт сообщение, представляющее собой строку символов, полученную в результате вычисления значения выражения E и преобразования его, если это необходимо, в строку символов. Максимальная длина этой строки зависит от применяемой аппаратуры. После выполнения оператора DISPLAY управление передается следующему оператору. Например, в результате выполнения оператора

DISPLAY ('END — BLOCK' || W);

где переменная W получила значение 13, на пульт будет выдано сообщение END BLOCK 13.

Если в операторе содержится конструкция REPLY (V), то переменной V присваивается значение E и выдается в качестве сообщения, после чего выполнение программы прерывается и состояние ожидания будет продолжаться до тех пор, пока оператор-программист у пульта не ответит. Ответ оператора-программиста в виде строки символов становится значением переменной V и может использоваться в программе. Переменная V не должна иметь длину больше допустимой для данного устройства: (в случае пишущей машинки 72 байт).

Оператор DISPLAY используется обычно в режиме диалога человек — машина.

**10. Ситуация, определяемая программистом.** Ситуация, определяемая программистом, имеет вид CONDITION (X), где X — идентификатор, задаваемый программистом; CONDITION — имя ситуации. По умолчанию предполагается, что X — внешнее имя, следовательно, во всей программе ситуации с именем CONDITION с одинаковыми идентификаторами представляют одну и ту же ситуацию. Стандартная реакция системы при обработке прерываний из-за ситуаций CONDITION — сигнализация (печать соответствующего сообщения) и продолжение выполнения программы.

Ситуации CONDITION (X) возбуждаются только оператором имитации прерываний в виде

SIGNAL CONDITION (X);

Например, оператор

ON CONDITION (CONTROL) GO TO ROL;

обеспечивает в программе после выполнения оператора

SIGNAL CONDITION (CONTROL);

включение реакции на ситуацию CONTROL и выполнение оператора GO TO ROL;

Операторы, рассмотренные в этом параграфе, относятся к операторам отладки.

#### Упражнения

1. Указать, какие из приведенных операторов содержат ошибки:

```
ON ZERODIVIDE SYSTEM;
ON ERROR RETURN;
ON SIZE;
ON SIZE BEGIN; I=1; J=10; END;
(CHECK (X)): X=Y;
M: ON SIZE GO TO M;
ON ERROR SYSTEM SNAP;
ON SIZE ON FIXEDOVERFLOW SYSTEM;
(NOCHECK (K,L)): DO K=1 TO 7; L=K; END;
ON ERROR SNAP;
```

2. Определить, какие значения получит переменная Z в конце выполнения программы

```
P: PROCEDURE OPTIONS (MAIN);
ON SIZE Z=-1; ON CONDITION (Y1) Z=Z+1;
ON CONDITION (Y2) Z=Z+1; ON ERROR Z=Z+1;
ON FINISH Z=Z+1; Z=0;
SIGNAL CONDITION (Y1); SIGNAL SIZE;
IF Z<0 THEN M: SIGNAL FINISH;
SIGNAL CONDITION (Y2); SIGNAL ERROR;
END;
```

## § 13. Распределение памяти

Под *распределением памяти* подразумевается выделение участков памяти для хранения программы и обрабатываемых данных. Для переменных, которые описаны в блоке, транслятором автоматически выполняется распределение памяти. В то же время программист может активно вмешиваться в процесс распределения памяти, указывая при объявлении данных в операторе DECLARE соответствующие описатели.

1. **Описатели области действия.** *Описателями области действия* являются описатели EXTERNAL и INTERNAL, служащие для указания областей алгоритма, в которых объявляемые имена доступны.

По умолчанию все имена переменных имеют описатель INTERNAL, и считаются описанными как внутренние, а имена процедур и входов — описатель EXTERNAL, и считаются описанными как внешние имена. Последний из этих описателей уже рассматривался в § 10.

Имя, объявленное как внутреннее, считается описанным и может быть использовано только в блоке, в котором оно определено, а также в обычных блоках и процедурах, являющихся внутренними для блока с описанием имени. Внешнее имя может быть использовано в любом месте программы.

2. **Описатели выравнивания и упаковки.** Данные в памяти могут размещаться либо последовательно, без каких-либо промежутков, либо таким образом, чтобы начало данных приходилось на границу, соответствующую этому типу данных. Первый случай размещения данных обеспечивается описателем

UNALIGNED, и такие данные называются невыровненными или упакованными. Во втором случае размещение обеспечивается описателем ALIGNED, и такие данные называются выровненными или неупакованными.

Использование описателя UNALIGNED позволяет экономить память, но замедляется доступ к данным. Доступ к данным с описателем ALIGNED ускоряется, но данные в памяти могут быть размещены с разрывом (образуются неиспользуемые участки памяти). По умолчанию для строковых переменных принимается описатель UNALIGNED, а для остальных типов данных — описатель ALIGNED.

3. Описатели классов памяти. Распределение памяти может быть статическим (до выполнения программы) или динамическим (во время выполнения программы). Для управления размещением данных в памяти предусмотрены четыре различных класса памяти: статическая память, автоматическая, управляемая и базированная. Объявление класса памяти осуществляется соответственно описателями STATIC, AUTOMATIC, CONTROLLED, BASED (P), где P — указатель, которыми программист может воспользоваться в операторе DECLARE. Три последних описателя характеризуют динамическое распределение памяти. В случае структур описатели классов памяти могут быть заданы лишь для старшей структуры.

Память для переменной с описателем STATIC распределяется перед выполнением программы и не освобождается до конца работы программы. Значения таких переменных можно использовать и при повторном входе в блок, в котором имеется их описание. Описатель STATIC употребляется совместно с описателями EXTERNAL или INTERNAL. Для переменных с описателем EXTERNAL, если класс памяти не указан, по умолчанию подразумевается описатель STATIC.

Описатель AUTOMATIC означает, что данные размещаются в динамической памяти, распределение которой производится при каждом входе в блок. После выхода из блока память освобождается и значения переменных, описанных в этом блоке, теряются. Исключение составляет только рекурсивное обращение. Для переменных с описателем INTERNAL, если класс памяти не указан, по умолчанию подразумевается описатель AUTOMATIC.

Размещение в памяти переменных, имеющих описатели CONTROLLED или BASED, т. е. управляемых и базированных переменных, выполняется специальными операторами. Память, отведенная под управляемые или базированные переменные, освобождается при выполнении соответствующих операторов.

4. Размещение управляемых данных. Для отведения памяти под управляемые переменные используется оператор размещения данных вида

ALLOCATE X L;

где X — идентификатор, L — описатели управляемой переменной, массива или структуры. В последнем случае перед идентификатором указывается номер уровня. Список описателей L может включать в себя только описатели изменения и описатели BIT, CHARACTER с описателями разрядности, а также описатели начальных значений. При отсутствии описателей действует принцип умолчания. Память, отведенная оператором ALLOCATE, будет занята и после

выхода из блока, хотя обращение к переменной не разрешается вне блока.

Освобождение памяти производится оператором

```
FREE X;
```

где  $X$  — один или несколько идентификаторов, память под которые была отведена оператором `ALLOCATE`. Оператор `FREE` вызывает освобождение памяти, последний раз отведенной под переменные  $X$ . Оператор освобождения памяти и соответствующий ему оператор размещения данных должны принадлежать одной и той же процедуре.

Описатель `CONTROLLED` используется при динамическом распределении памяти, например для размещения в памяти массивов, у которых диапазоны изменения индексов не определены в момент входа в блок, а вычисляются в процессе выполнения блока. Следующий фрагмент программы иллюстрирует сказанное:

```
BEGIN: DECLARE Z(M,N) CONTROLLED FIXED; ...;
      M=10; N=20; ALLOCATE Z; ...;
      L1: FREE Z; ...; M,N=30; ALLOCATE Z; ...;
      L2: FREE Z; END;
```

Здесь при описании массива  $Z$  границы заданы переменными и при входе в блок распределение памяти для данного массива не производится. После того как  $M$  и  $N$  получили значения, оператором `ALLOCATE Z`; резервируется память для  $Z$ . После освобождения памяти, отведенной для  $Z$ , оператором с меткой `L1` массив  $Z$  может быть использован в блоке с новыми границами, если написать новый оператор `ALLOCATE Z`; Вторично память освобождается оператором с меткой `L2`.

Определить, выделялась ли память для управляемой переменной, можно с помощью встроенной функции `ALLOCATION (X)`. Если память для  $X$  выделялась, то значение, вычисляемое этой функцией, равно '1'B, если не выделялась, то оно равно '0'B. Например, можно написать следующие операторы:

```
DECLARE T(M,N) CONTROLLED; ...; IF ALLOCATION (T)
      THEN ALLOCATE T(I,J) INITIAL ((I*J)0);
```

5. Размещение базированных данных. Оператор *размещения для базированных данных* имеет вид

```
ALLOCATE X SET (P);
```

где  $X$  — имя базированной переменной массива или структуры,  $P$  — скалярная переменная типа указателя (возможно элемент массива или структуры); переменная  $P$  не обязана совпадать с переменной типа указателя, связанной с переменной  $X$  и описанной в операторе `DECLARE`. Этот оператор резервирует ранее свободные участки памяти, причем размер каждого участка равен размеру соответствующей переменной  $X$ . Положение (адрес) начала участка определяется либо значением указателя  $P$ , либо, в случае отсутствия конструкции `SET (P)`, — значением указателя, заданного в описателе `BASED` переменной  $X$ . Допускается смешанная форма оператора `ALLOCATE`, когда

впережку в одном операторе размещаются как базированные, так и управляемые переменные.

Освобождение памяти, отведенной под базированные переменные с помощью оператора размещения, выполняется оператором освобождения памяти вида

```
FREE P—> X;
```

где знак  $\rightarrow$  обозначает освобождение памяти, отведенной для переменной X с данным указателем P (конструкция  $P\rightarrow$  может быть опущена). Начало освобождаемого участка задается указателем переменной X, размер участка определяется размером переменной X. Освобождаемый участок должен совпадать с ранее занятым участком. Например, в операторе

```
DECLARE T(100) BASED (P);
```

описан одномерный массив T, который размещается в базированной памяти. Переменная P задана описателем POINTER. Размещение массива T в памяти осуществится после выполнении оператора

```
ALLOCATE T SET (P);
```

который присваивает указателю P начальный адрес размещения элементов массива T. Освобождение памяти производится оператором

```
FREE P—> T;
```

#### Упражнения

1. Найти ошибки в следующих операторах объявления данных:

```
DECLARE A(16) STATIC FIXED AUTOMATIC;
DECLARE B(10) AUTOMATIC EXTERNAL;
DECLARE C CHARACTER (N) STATIC;
DECLARE D, 5 E STATIC, 5 F STATIC;
DECLARE H(L,M) EXTERNAL;
```

2. Определить, какую длину будут иметь переменные X, Y, Z в конце выполнения процедуры

```
C: PROCEDURE;
  DECLARE (X,Y,Z) CONTROLLED CHARACTER (9);
  ALLOCATE X,Y,Z CHARACTER (4);
  ALLOCATE X,Y CHARACTER (6), Z;
  FREE X,Z;
  ALLOCATE X,Y CHARACTER (*), Z CHARACTER (*);
END C;
```

3. Какие из операторов объявления данных недопустимы и почему?

```
DECLARE A BASED (P), P BASED (S), S POINTER;
DECLARE I B BASED (P), 2 C, 2 D BIT (I), P POINTER;
DECLARE C (L) BASED (P), P POINTER;
DECLARE (D BASED (S), S POINTER) EXTERNAL;
DECLARE E BASED (P,S), 1 P, 2 (S,R) POINTER;
```

4. Написать программу для вычисления значения каждого очередного члена ряда  $a_1 + a_2 + \dots + a_n + \dots$ , если  $a_1 = a_2 = 1$ , а при  $n \geq 3$ :  $a_n = a_{n-2} + a_{n-1}$ . Для заданного n сформировать массив значений всех n членов.

## § 14. Описатели файла

Данные, размещенные на внешних носителях информации, составляют комплекты (или наборы) данных. Из них образуются *файлы*, состоящие из отдельных записей. *Запись* — это основная единица обмена информацией между внешними носителями и памятью. Запись содержит значения одной или целого набора переменных. Обмен между внешним носителем и памятью производится обычно через специальную область памяти — *буфер*. При объявлении файла указываются различные описатели файла, от которых будет зависеть обработка файла при вводе и выводе. Описатели одного и того же файла в нескольких внешних процедурах не должны противоречить друг другу.

К числу описателей файла относятся ключевые слова: FILE, RECORD, STREAM, INPUT, OUTPUT, UPDATE, SEQUENTIAL, DIRECT, BUFFERED, UNBUFFERED, PRINT, BACKWARDS, EXCLUSIVE, KEYED, ENVIRONMENT, CONSECUTIVE, INDEXED, REGIONAL, LINESIZE, PAGESIZE.

Описатель FILE определяет данный идентификатор как имя файла. Этот описатель используется со всеми другими перечисленными выше описателями и может быть опущен, если употребляется хотя бы один из этих описателей; наличие описателя FILE в таком случае определяется контекстуально.

**1. Описатели способа обработки.** Описатели способа обработки RECORD и STREAM определяют метод использования данных в файле.

Описатель RECORD указывает, что данный файл следует рассматривать как последовательность записей и определяет передачу комплекта данных, состоящего из отдельных записей. Передача идет без преобразования либо прямо в память или из памяти, либо на буфер.

Описатель STREAM указывает, что данный файл рассматривается как последовательность символов и определяет передачу комплекта данных как непрерывный поток символов. В этом случае записи, которые находятся в буфере, разделяются на составляющие элементы. Они редактируются и преобразуются из внешнего представления во внутреннее.

Файл с описателем RECORD может использоваться только в операторах OPEN, CLOSE, READ, WRITE, REWRITE, LOCATE, DELETE, UNBLOCK. Файл с описателем STREAM может использоваться только в операторах OPEN, CLOSE, PUT, GET и не употребляется с описателями RECORD, UPDATE, DIRECT, SEQUENTIAL, BUFFERED, UNBUFFERED, BACKWARDS, EXCLUSIVE, KEYED.

**2. Описатели функции файла.** Описателями функции файла являются INPUT, OUTPUT, UPDATE. Эти описатели указывают назначение файлов. Первый указывает, что данные из файла будут передаваться в память (в программу), второй — что данные будут передаваться из памяти на внешний носитель (в файл). Описатель UPDATE указывает, что файл может использоваться как для ввода, так и для вывода данных.

Файл с описателем INPUT не может иметь описателей EXCLUSIVE и PRINT, а файл с описателем OUTPUT — описателей EXCLUSIVE и BACKWARDS. Файл с описателем UPDATE не может употребляться с описателями BACKWARDS, STREAM и PRINT.

**3. Описатели доступа.** Описатели DIRECT и SEQUENTIAL указывают возможность соответственно прямого и последовательного доступа к файлу с описателем RECORD.

Последовательный доступ заключается в том, что устройство ввода просматривает данные в порядке их следования. Обмен между внешним носителем и памятью, как правило, начинается с первой и заканчивается последней записью. Последовательный доступ осуществляется для файлов на перфокартах и магнитных лентах. Например, объявление файла с описателями UPDATE и SEQUENTIAL определяет способ обновления данных, т. е. следующий процесс: ввод записи в память, выполнение над элементами записи операций и вывод записи обратно на то же место, которое она занимала на внешнем носителе. Обращение к данным в таких файлах должно осуществляться последовательно операторами READ и затем REWRITE.

Прямой доступ заключается в том, что к любой части внешнего носителя, в которой размещены данные, можно обратиться непосредственно. Для этого физическим записям файла присваивается ключ, который указывает положение записи, следовательно, файлы с описателем DIRECT должны также иметь описатель KEYED.

**4. Описатели буферизации.** Описатели буферизации, обозначаемые ключевыми словами BUFFERED и UNBUFFERED, применимы только для файла с описателями SEQUENTIAL и RECORD и определяют, необходимо или нет пользоваться промежуточной памятью (буфером) во время обмена данными между памятью и внешними носителями.

**5. Дополнительные описатели.** Описатель PRINT из этой группы описателей файла указывает, что окончательно данные должны быть расположены на печатной странице. Файл с описателем PRINT подразумевает описатели OUTPUT и STREAM, описатель PRINT не может быть указан для файла с описателем RECORD.

Описатель BACKWARDS указывает, что к файлу с описателями SEQUENTIAL и INPUT обращения происходят в обратном порядке, т. е. от последнего элемента файла к первому.

Описатель EXCLUSIVE указывает, что файл с описателями DIRECT и UPDATE будет использован таким образом, чтобы запрещалось чтение, исключение или перепись записи в одной ветви, в то время как в другой ветви эта запись находится в одном из перечисленных состояний. Здесь речь идет о параллельном (асинхронном) выполнении отдельных частей программы, что в этой книге не рассматривается.

Описатель KEYED (K) указывает, что каждая запись в файле имеет связанный с ней признак, т. е. свой ключ. После описателя в скобках пишется десятичная целая константа — (K), которая определяет длину признака в символах. Файл с описателем KEYED может иметь описатели STREAM и PRINT. Описатель KEYED должен быть указан для каждого файла, содержащего ключ, даже если записи читаются последовательно.

Описатель ENVIRONMENT (D), где D — список дополнений, характеризует особенности физического размещения данных на внешнем носителе. Состав и правила задания компонент в списке дополнений существенно зависят от используемого оборудования ЭВМ, ее программного обеспечения и

способа организации файла. В частности, в списке дополнений могут быть указаны форматы записей, устанавливающие размеры записей и наборов записей (блоков), подлежащих передаче.

По способу организации файлы подразделяются на стандартные последовательные, индексно-последовательные и региональные (с прямым доступом). В описателе ENVIRONMENT в списке дополнений способ организации задается одним из описателей: CONSECUTIVE, INDEXED, REGIONAL.

Последовательный (CONSECUTIVE) способ организации файла характеризуется тем, что блоки данных запоминаются на внешнем носителе в следующих друг за другом областях. Записи на перфокартах — типичный пример последовательной организации файла. Файлы на магнитных лентах и файлы, выведенные на печать, также являются последовательно организованными.

Региональная (REGIONAL) организация файла заключается в том, что память на внешнем носителе подразделяется на области. В каждой из них может размещаться одна или несколько записей (записи в блоки не объединяются). Перед записью указываются ключи, которые обеспечивают прямой доступ к записи файла. Региональная организация файлов возможна только на дисках.

Последовательная и региональная организация файлов обеспечивают соответственно последовательный и прямой доступ к записям. При индексно-последовательной (INDEXED) организации файла записи образуют логически упорядоченную последовательность в соответствии с заданным признаком (ключом), значения которого возрастают от записи к записи. Расположение записей файла описывается с помощью индексов. Доступ возможен как прямой, так и последовательный. Индексно-последовательная организация возможна для файлов, размещаемых на дисках. Если файл не является стандартным последовательным, он должен иметь ключи, т. е. либо в операторе DECLARE, либо в операторе OPEN необходимо указать описатель KEYED (K).

Если ни один из трех способов организации файла не указан, то по умолчанию предполагается способ CONSECUTIVE. Остальные дополнительные описатели никогда не выбираются по умолчанию, а должны всегда задаваться явно либо в операторе объявления файла, либо в операторе открытия файла. Единственным исключением является описатель PRINT, присваиваемый по умолчанию стандартному выводному файлу SYSPRINT. Все другие рассмотренные выше описатели, называемые взаимно исключаящими, могут быть выбраны по умолчанию, а именно: из описателей способа обработки приписывается описатель STREAM, из описателей функции — INPUT, из описателей доступа — SEQUENTIAL, из описателей буферизации — BUFFERED, а из описателей области действия — всегда описатель EXTERNAL.

**6. Описатели при печати.** Имеются два описателя, которые применяются только для файлов с описателем PRINT. Это описатели LINESIZE (E) и PAGESIZE (E), где E — выражение. Первый из них указывает длину печатаемой строки (включая символ управления). По умолчанию длина строки равна 120 печатным знакам. Описатель PAGESIZE указывает число строк, приходящихся на одну страницу. По умолчанию число строк на странице равно 60.

**Упражнения**

1. Составить оператор описания файла F, предназначенного для обработки данных как потока символов, если известно, что файл будет использоваться: а) для вывода, но не на печать; б) и для ввода и для вывода; в) для вывода на печать; г) для ввода и имя файла не должно быть внешним;
2. Указать ошибки при описании файлов в операторах:

```

DECLARE A FILE REAL;
DECLARE B FILE STREAM INPUT RECORD;
DECLARE C INPUT STREAM PRINT;
DECLARE FILE OUTPUT INPUT;
DECLARE D INTERNAL RECORD PRINT;

```

**§ 15. Подготовка файлов к вводу-выводу**

Прежде чем приступить к чтению или записи какого-либо файла, он должен быть открыт. Открытие файла означает, что некоторое имя ставится в соответствие конкретному файлу. Любой файл можно открыть двумя способами: явно — путем использования оператора открытия файлов — и неявно.

Оператор открытия файлов имеет вид

```
OPEN FILE (F) FL;
```

где F — имя файла, FL — совокупность описателей. В операторе OPEN конструкций вида FILE (F) FL может быть несколько. Разделителем в этом случае выступает символ запятая. Для файла, который уже открыт, оператор открытия файлов игнорируется.

Файл открывается неявным образом, если выполняется хотя бы один из операторов ввода или вывода, относящийся к файлу, который не был открыт. Открытие файла неявным образом не позволяет программисту вводить какие-либо описатели во время открытия.

После обработки последней записи файла он должен быть закрыт. Для этого служит оператор

```
CLOSE FILE (F);
```

где конструкций вида FILE (F), записываемых через запятую, может содержаться произвольное число. Оператор CLOSE для неоткрывшегося файла игнорируется.

После закрытия файл может быть открыт снова. Однако если файл закрывается, то теряются те описатели, которые были указаны в операторе OPEN, остаются лишь описатели, записанные в операторе DECLARE. Поэтому можно закрыть файл, использованный для вывода, и затем открыть его повторно, но уже как файл для ввода. Например, в процедуре

```

PRINT: PROCEDURE (P);
      DECLARE P(*) FIXED F FILE PRINT; ...;
S: OPEN FILE (F) LINESIZE (49); ...;
      CLOSE FILE (F); ...;
END PRINT;

```

файл F используется для вывода. Если данные печатаются до выполнения оператора с меткой S или после оператора CLOSE, то печатная строка будет

иметь стандартную длину, в противном случае оператор OPEN с меткой S обеспечит длину строки в 49 символов.

Для удобства организации ввода и вывода информации, кроме файлов, которые объявляются в программе и имена которых выбираются произвольно, существуют два файла — SYSIN и SYSPRINT, — называемые соответственно *стандартным вводным* и *стандартным выводным файлами*. Эти файлы объявлять не надо; по умолчанию считается, что описателями файла SYSIN являются STREAM и INPUT, а файл SYSPRINT имеет описатели STREAM и PRINT.

Стандартный вводной файл, как правило, используется для ввода информации с перфокарт. С каждой перфокарты может быть введено до 80 символов. Стандартный выводной файл практически всегда используется для печати. При этом как стандартная длина строки, так и стандартное число строк на странице могут быть изменены оператором OPEN.

#### Упражнение

Составить оператор открытия файла FL, предназначенного для передачи потока данных, если файл явно не описан и известно, что файл будет использоваться для: а) вывода на печать со стандартными размерами строк и страниц; б) вывода, но не на печать; в) ввода; г) вывода на печать со строкой из 50 символов и страницей из 50 строк; д) вывода на печать со стандартной по размеру строкой и страницей из 29 строк; е) вывода на печать со строкой длиной 60 символов и стандартной по размеру страницей.

## § 16. Ввод и вывод данных

Под *вводом* и *выводом* понимается передача данных из внешних носителей информации в память ЭВМ и соответственно из памяти на внешние носители. Ввод и вывод осуществляются операторами, которые в качестве составной части имеют список ввода-вывода.

Список ввода-вывода определяет участки памяти, используемые для ввода или вывода, и имеет вид (A), где A — список элементов, разделяемых запятыми. В качестве элемента списка допускаются переменные любого типа, имена массивов, структур и подструктур, отдельные элементы массивов и структур, псевдопеременные, индексированные элементы, а также выражения (в списке вывода). Конкретный вид списка ввода-вывода зависит от способа обработки информации.

Если в качестве элемента списка ввода-вывода указана переменная, то это означает, что необходимо выполнить ввод (вывод) значения этой переменной. Если в качестве элемента списка ввода-вывода указано имя массива или старшей структуры, то это означает, что надо выполнить ввод (вывод) всех элементов этого массива или структуры в том порядке, в котором эти элементы размещены в массиве или в соответствии со строением структуры и с порядком размещения элементов массивов, входящих в эту структуру.

Индексированный элемент имеет вид:

```
(( (A(I,J,K) DO I=I1 TO I2 BY I3) DO J=J1 TO
J2 BY J3) DO K=K1 TO K2 BY K3)
```

и представляет собой сокращенную запись заголовков вложенных циклов для переменных с индексами. Если в качестве элемента списка ввода-вывода указан индексированный элемент, то это означает, что необходимо ввести (вывести) значения величин, указанных в списке индексированного элемента, организовав порядок их следования в соответствии с изменением параметров циклов, число которых не обязательно равно трем, как в данном примере. Использование индексированных элементов в списках ввода-вывода иногда позволяет избежать сложных групп и операторов цикла, в других дает возможность организовать единственный простой способ передачи данных в удобной форме. Например, индексированный элемент

```
((X(I,J) DO I=2 TO 4, 7) DO J=1 TO 5 BY 2)
```

представляет собой совокупность элементов массива  $X$  в такой последовательности:  $X(2,1)$ ,  $X(3,1)$ ,  $X(4,1)$ ,  $X(7,1)$ ,  $X(2,3)$ ,  $X(3,3)$ ,  $X(4,3)$ ,  $X(7,3)$ ,  $X(2,5)$ ,  $X(3,5)$ ,  $X(4,5)$ ,  $X(7,5)$ .

В соответствии с тем, что файлы, подлежащие обработке, могут иметь описателем либо `STREAM`, либо `RECORD`, существует два способа передачи данных: ввод-вывод, ориентированный на поток, и ввод-вывод, ориентированный на запись.

Первый способ передачи данных предполагает, что входной или выходной поток состоит из непрерывной последовательности символов. Объем данных, передаваемых за одно выполнение оператора ввода или вывода, определяется списком ввода-вывода, элементам которого присваиваются данные из этого потока или из значений, формирующих выходной поток данных. Преобразования данных из символьной формы представления во внутреннюю, определяемую характеристиками элемента списка ввода-вывода, или, наоборот, из внутренней формы в символьную осуществляются по необходимости.

Возможны следующие способы передачи потока данных: передача, управляемая списком, передача, управляемая данными, передача, управляемая редактированием. Кроме обмена информацией между памятью и внешними носителями, возможны внутренние пересылки данных. Операторы ввода, ориентированные на поток данных, начинаются с ключевого слова `GET`, операторы вывода — с ключевого слова `PUT`.

Способ передачи данных, ориентированный на запись, предполагает, что каждый файл рассматривается как последовательность записей. При вводе запись целиком и без преобразования присваивается некоторой переменной (массиву, структуре), при выводе значение переменной (массива, структуры) целиком и без преобразования образует некоторую запись. В общем случае запись представляет собой конструкцию вида  $KR$ , где  $K$  — последовательность символов, называемая ключом (признаком),  $K$  может и отсутствовать,  $R$  — данное. В качестве  $R$  может быть действительное число с фиксированной или плавающей точкой, комплексное число (пара действительных чисел, разделенных запятой), строка символов или строка битов.

Различают операторы ввода-вывода записей последовательного доступа и операторы ввода-вывода записей прямого доступа. При использовании операторов ввода-вывода записей последовательного доступа записи могут быть

переданы с использованием буфера или без него. Операторы ввода, ориентированные на записи, начинаются с ключевых слов READ, DELETE, операторы вывода — с ключевых слов WRITE, LOCATE, REWRITE.

### Упражнения

1. Составить список ввода-вывода для обработки элементов двумерного массива  $A = \{a_{ij}\}$  ( $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , где  $m$ ,  $n$  — заданы) в следующем порядке:

- а)  $a_{11}, a_{21}, \dots, a_{m1}, a_{12}, a_{22}, \dots, a_{m2}, \dots, a_{1n}, a_{2n}, \dots, a_{mn}$ ;
- б) в порядке, обратном по отношению к случаю а);
- в)  $a_{11}, a_{31}, a_{51}, \dots, a_{13}, a_{33}, a_{53}, \dots, a_{22}, a_{42}, \dots, a_{24}, a_{44}, \dots$

2. Составить список ввода-вывода для обработки элементов обеих главных диагоналей матрицы  $A$  (см. упражнение 1) в прямом и обратном порядке.

3. Составить список ввода-вывода для обработки элементов одномерных массивов  $B = \{b_i\}$  и  $C = \{c_i\}$  ( $1 \leq i \leq n$ , где  $n$  задано) в следующем порядке:

- а)  $b_1, c_1, b_2, c_2, \dots, b_n, c_n$ ;
- б)  $b_1, c_n, b_2, c_{n-1}, \dots, b_n, c_1$ ;
- в)  $b_n, b_{n-1}, \dots, b_2, b_1, c_1, c_2, \dots, c_{n-1}, c_n$ .

## § 17. Операторы ввода-вывода потока

1. Ввод-вывод, управляемый списком. Ввод данных обеспечивается оператором

GET FILE (F) LIST (A);

где F — имя файла, A — список ввода, LIST — дополнение, специфицирующее список. Данные читаются символ за символом с вводного устройства, рассматриваются как строки символов и присваиваются переменным из списка ввода с соответствующими преобразованиями. Данные на вводимом устройстве разделяются запятой или пробелом. Границы карт игнорируются, и вводимые данные обрабатываются как непрерывный поток.

Вывод данных осуществляется оператором

PUT FILE (F) LIST (A);

где A — список вывода, остальные обозначения такие же, как и в операторе ввода. Границы карт или строк игнорируются; значению каждой переменной отводится столько места, сколько нужно; в качестве разделителя используется пробел.

Конструкция FILE (F) в операторах GET и PUT может быть опущена. В этом случае при вводе и выводе используются стандартные файлы (системные устройства) с именами SYSIN и SYSPRINT соответственно.

Пример. Операторы

DECLARE A FIXED (5,2), B FIXED (3,2), C FIXED (5,1);  
GET LIST (A,B,C);

обеспечат ввод перфокарт с системного устройства с отперфорированными на них, вапример, следующими данными:

```
1.3 — 3.45 — 6.7,
80 — — 9.87 — 6.54 — 3 — — 2 —
1
```

Если оператор GET проработал один раз, переменные A, B, C получают соответственно значения 001.30, 3.45, 0006.7; после второго выполнения — 080.00, —9.87, 0006.5; после третьего — 003.00, —2.00, 0001.0.

К оператору GET может быть добавлено дополнение COPY, которое обеспечивает печать вводимых данных на SYSPRINT. Например, оператор

```
GET LIST (A) COPY;
```

выполняет чтение и преобразование переменной A во внутреннюю форму, а также выдает на печать значение переменной A в том виде, в каком оно было воспринято на устройстве ввода.

2. Ввод-вывод, управляемый данными. Передача, управляемая данными, позволяет читать или записывать данные, представляемые в виде допустимых констант с именами передаваемых значений. Операторы ввода и вывода, управляемые данными, соответственно имеют вид

```
GET FILE (F) DATA (A);
PUT FILE (F) DATA (A);
```

где F — имя файла, DATA — дополнение, специфицирующее данные, A — список ввода или вывода; при вводе A — список идентификаторов простых переменных или массивов, при выводе A не может содержать выражений; как при вводе, так и при выводе список A не может содержать формальных параметров, а также имен базированных и повторно определяемых переменных. Конструкция FILE (F) может быть опущена, тогда предполагается использование SYSIN для оператора GET и SYSPRINT для оператора PUT. Список ввода-вывода в операторах GET и PUT может отсутствовать.

При вводе значения могут быть присвоены только переменным, идентификаторы которых указаны в списке ввода, однако не обязательно всем перечисленным там переменным. На вводимом устройстве данные представляются в виде элементов V=C, где V — переменная, C — константа, которые друг от друга отделяются запятой или пробелом. В конце списка ставится точка с запятой. Порядок следования элементов в списке на вводимом устройстве может отличаться от порядка переменных и списке идентификаторов оператора GET. Например, оператор

```
GET DATA (Q,X,Y,Z);
```

может прочитать исходные данные, набитые на перфокарте в виде

```
Y=12, X=34 — Z=56;
```

в результате чего значения будут присвоены переменным X, Y, Z, а идентификатор Q не получит значения.

Если в операторе ввода отсутствует список ввода, то будут введены все элементы, указанные на вводном устройстве во входной последовательности. Так, если на вводном устройстве имеются указанные выше входные данные, то они могут быть введены оператором

```
GET DATA;
```

который эквивалентен оператору

```
GET DATA (Y,X,Z);
```

Во входных данных могут использоваться переменные с индексами. Например, если на перфокарте набито

```
M(4,3)=8, M(1,2)=3, M(9,9)=5;
```

где M — идентификатор массива, описанного оператором

```
DECLARE M(10,10) FIXED;
```

то после выполнения любого из операторов

```
GET DATA (M); или GET DATA;
```

будут введены только значения элементов M(4,3), M(1,2), M(9,9). Такой способ используется, когда необходимо изменить лишь несколько значений в большом массиве. К оператору GET может быть добавлено слово COPY, которое обеспечивает выдачу входных данных на SYSPRINT.

Оператор PUT обеспечивает вывод значения каждой переменной как элемента в виде V=C. Элементы разделяются пробелом, в конце списка ставится точка с запятой. Так, если значения переменных X, Y, Z из предыдущего примера не изменились, оператор

```
PUT DATA (X,Y,Z);
```

осуществит их вывод на устройство SYSPRINT в виде следующей печатной строки:

```
X=34 Y=12 Z=56;
```

Вид выводимых значений определяется соответствующими описателями, константы в виде строк символов заключаются в кавычки, а после строки битов ставится буква B. Если в операторе вывода отсутствует список, то подразумевается такой список вывода, который содержит имена всех переменных и известен в той точке программы, где расположен данный оператор PUT.

Отметим, что ввод-вывод, управляемый данными, требует значительного объема оперативной памяти в рабочей программе.

**3. Ввод-вывод, управляемый редактированием.** Передача данных, при которой можно детально задавать форму поля данных в потоке (формат), осуществляется операторами ввода-вывода, управляемыми редактированием. Операторы ввода и вывода, управляемые редактированием, соответственно имеют вид

```
GET FILE (F) EDIT (A) (B);
```

```
PUT FILE (F) EDIT (A) (B);
```

где F — имя файла, A — список ввода или вывода, B — список форматов, EDIT — дополнение, специфицирующее редактирование. Если в записи операторов опущена конструкция FILE (F), то предполагается SYSIN для ввода и SYSPRINT для вывода. В операторе GET после списка форматов можно указать дополнение COPY, что вызовет печать входных данных на SYSPRINT. Список форматов состоит из элементов, являющихся спецификациями формата. Порядок размещения переменных в списке идентификаторов соответствует тому порядку, в котором располагаются вводимые значения.

Спецификации форматов подразделяются на спецификации формата данных и спецификации управления (управляющие форматы). Спецификации формата данных задают форму полей данных в потоке. Управляющие форматы определяют при выводе операции над страницами, строками и обеспечивают разрядку текста, а при вводе используются для пропуска части символов. Обычно соблюдается поэлементное соответствие между переменными списка ввода-вывода и спецификациями форматов списка форматов. Однако такое соответствие не является обязательным, более того, оно невозможно, если число элементов списка ввода-вывода неизвестно. В таком случае передача данных осуществляется по правилам: а) если список ввода-вывода исчерпан, передача данных прекращается; б) если список форматов исчерпан, то спецификации повторяются от начала списка форматов до тех пор, пока не будут переданы все элементы списка ввода-вывода.

**4. Спецификации формата данных.** Спецификации формата данных описывают форму представления данных в потоке данных. Имеется шесть типов спецификаций формата: спецификация формата десятичного числа с фиксированной точкой, спецификация формата десятичного числа с плавающей точкой, спецификация формата комплексного числа, спецификация формата с шаблоном, спецификация формата строк символов, спецификация формата строк битов.

В общем случае спецификация формата данных может быть представлена в виде

$$D(w,d,s),$$

где D — индекс спецификации, w, d, s — выражения, принимающие целые значения, причем обычно предполагается, что  $d \leq s \leq w$ . Выражение w задает длину поля данных в символах, используемых при внешнем представлении (включая знаки, точки, пробелы и буквы E, B, которые используются при записи констант). Значение d определяет количество позиций после точки, а s указывает количество значащих цифр (спецификация формата десятичного числа с плавающей точкой) или является масштабным множителем (спецификация формата десятичного числа с фиксированной точкой). В записи спецификации формата индекс спецификации D является обязательным элементом, а в ряде спецификаций некоторые из компонентов w, d, s (или все) могут отсутствовать.

В качестве индекса спецификации формата данных используются следующие символы: F — для элемента списка ввода-вывода с фиксированной точкой, E — для элемента с плавающей точкой, C — для комплексного элемента, P — для элемента с шаблоном, A — для строк символов, B — для строк битов.

Спецификации формата данных с фиксированной точкой имеют вид  $F(w)$ ,  $F(w,d)$  и  $F(w,d,s)$ .

В случае спецификации  $F(w)$  входное поле содержит  $w$  символов, соответствующих десятичному числу с фиксированной точкой. Если точка на перфокарте не набита, то предполагается, что число целое. Перед отрицательным числом должен стоять знак минус. При выводе данные воспринимаются как целые числа, при этом точка не изображается (не перфорировуется и не печатается).

Для спецификации  $F(w,d)$ , если при вводе данных на входном поле точка не содержится, по умолчанию считается, что она стоит перед последней (считая справа) из  $d$  десятичных цифр в  $w$ -символьном поле. Если же точка отперфорирована, то заданное значение  $d$  во внимание не принимается. При выводе данных точка печатается или перфорировуется всегда. В этом случае для записи цифр после точки отводится  $d$  десятичных разрядов, а для записи цифр до точки  $w-d-1$  разрядов, если число положительное, или же  $w-d-2$  разрядов, если число отрицательное.

В случае спецификации  $F(w,d,s)$  при вводе данных считываемая величина перед преобразованием умножается на  $10^s$ . При выводе число, хранимое в оперативной памяти, умножается на  $10^{-s}$ . В этой спецификации  $s$  — масштабный множитель и на него ограничение  $d \leq s \leq w$  не распространяется.

Пример. В результате выполнения операторов

$A=1.234$ ;  $B=-5673$ ;  $C=90$ ;  
 PUT EDIT (A,B,C) (F(6,3), F(8,3,3), F(5));

будет напечатана строка

$\underbrace{1.234}_A \quad \underbrace{-5.673}_B \quad \underbrace{90}_C$

Спецификации формата данных с плавающей точкой имеют вид

$E(w,d)$  и  $E(w,d,s)$ .

В случае спецификации  $E(w,d)$  предполагается, что при вводе внешний носитель содержит число с плавающей точкой, расположенное в любом месте поля, имеющего длину  $w$  символов. Допустимы различные формы записи чисел в поле, однако требуется, чтобы перед показателем степени присутствовал по крайней мере один из символов  $E$ ,  $+$ ,  $-$ . Если точка не отперфорирована, количество разрядов после точки определяется форматом. При выводе показатель степени всегда записывается с помощью четырех символов в форме  $E \pm aa$ , где  $a$  — десятичная цифра, и число печатается с  $d$  дробными десятичными разрядами. Число располагается справа в  $w$ -символьном поле. Если число положительное, то оставлять место для знака (слева) не требуется.

В случае спецификации  $E(w,d,s)$  при вводе данных значение  $s$  во внимание не принимается. При выводе для записи числового значения выделяется  $s-d$  позиций перед точкой и  $d$  позиций после точки. Знак минус предшествует самой левой цифре отрицательных чисел. Так, если в предыдущем примере написать оператор

PUT EDIT (A,B,C) (E(10,4), E(10,2), E(10,0,2));

то будет напечатана следующая строка:

$$\underbrace{1.2340E + 00}_{A} \underbrace{-5.67E + 03}_{B} \underbrace{-90.E + 00}_{C}$$

Отметим, что спецификация  $E(w,d,d+1)$  эквивалентна  $E(w,d)$ .

Спецификация формата комплексного числа имеет вид  $C(D_1, D_2)$ , где  $D_1$  и  $D_2$  — любая разновидность спецификаций F, E или P. Используется эта спецификация для комплексных элементов списка ввода-вывода. Спецификация  $D_1$  характеризует вещественную часть,  $D_2$  — мнимую часть комплексной величины;  $D_2$  может быть опущена, тогда обе части комплексного числа специфицируются одинаково.

Числовые данные могут быть описаны при помощи числового шаблона с использованием спецификации формата с шаблоном в виде P 'шаблон'. При вводе спецификация шаблоном описывает форму данных на внешнем носителе, а также способ их числовой интерпретации. Из вводимого потока считывается определяемое шаблоном количество символов, и полученная строка символов присваивается соответствующей переменной. Если прочитанные символы не удовлетворяют указанному шаблону, возникает ситуация CONVERSION. При выводе значение элемента списка вывода преобразуется перед передачей к виду, задаваемому шаблоном. Двоичные числовые поля будут иметь после передачи символическое представление.

**Пример.** Ввод из потока в качестве значений переменных A, B, C трех наборов символов 123, ABC1 и 1.2 по спецификациям P'9V99', P'AAХ9' и P'999' соответственно приведет к присвоению переменным A и B значений '123' и 'ABC1', при этом полученное числовое значение в первом случае равно 1.23; переменной C не будет присвоено никакое значение, а возникает ситуация CONVERSION. При выводе числового значения 12.3 по шаблону P'9V99' получаем выводимую строку в виде 230.

Спецификации формата строк символов имеют вид A(w) и A. В случае спецификации A(w) при вводе значение строки символов длины w присваивается символьной переменной. Если переменная содержит больше чем w символов, то она справа дополняется соответствующим числом пробелов. Если переменная содержит меньше чем w символов, справа производится усечение вводимого поля. При выводе переменная располагается слева в поле, правая часть дополняется пробелами. Выводимая переменная должна иметь тип строки символов.

Спецификация формата A применяется только при выводе. Здесь значение w берется равным длине переменной, имеющей тип строки символов; при этом в списке идентификаторов может непосредственно содержаться строка символов, а не идентификатор переменной типа строки символов.

Спецификации формата строк битов имеют вид B(w) и B. В случае спецификации B(w) при вводе строка символов длиной w считывается и преобразуется во внутреннюю строку символов из нулей и единиц; при выводе переменная располагается слева в выводимом поле. Правая часть поля дополняется пробелами, если длина полученной строки меньше w.

Спецификация формата В используется только при выводе. Здесь значение  $w$  предполагается равным длине выводимой строки битов. Например, после выполнения операторов

```
DECLARE X CHARACTER (6), Y BIT (4);
X='CONST='; Y='1001'B;
PUT EDIT(X,Y,'_ BINAR') (A(6),B,A);
```

будет выведена следующая печатная строка:

```
CONST=1001 BINAR
```

Отметим, что если в приведенных спецификациях  $w \leq 0$ , то при вводе соответствующие элементы списка ввода и списка форматов пропускаются, если только элемент в первом списке не является строкой (в этом случае значение будет равно пустой строке). При выводе элемент списка форматов пропускается, если  $w \leq 0$ .

Если в списке форматов несколько спецификаций имеют одинаковую структуру, то возможно применение коэффициента повторения в виде целой константы в скобках перед спецификацией. Так, список форматов (F(8,3),F(8,3),F(8,3)) может быть переписан как ((3)F(8,3)).

**5. Управляющие форматы.** Управляющие форматы используются в тот момент, когда они встречаются при поиске очередной спецификации формата данных. Самой спецификации управления в списке ввода-вывода не соответствует никакой элемент. После того как список ввода-вывода исчерпан, оставшиеся управляющие форматы соответствующего списка форматов игнорируются. Спецификации управления подразделяются на формат для задания интервалов и форматы для печати.

Формат для задания интервалов имеет вид X(w). При вводе формат X означает, что очередные w символов потока данных должны быть пропущены. При выводе в поток данных добавляется w пробелов. Ситуации  $w < 0$  эквивалентна  $w = 0$ .

Спецификации управления для печати используются только в случае файлов с описателями STREAM и PRINT. Каждый такой файл состоит из страниц, а страница — из строк, расположенных друг под другом с некоторым интервалом. Первая строка на любой странице имеет номер один. Индексами форматов для печати являются дополнения: PAGE, SKIP, LINE, COLUMN.

Спецификация PAGE означает, что необходимо поток продвинуть до новой страницы (происходит прогон бумаги до первой строки следующей страницы).

Спецификация SKIP(w) означает, что необходимо страницу продвинуть на w строк, т. е. пропускается w-1 строк до печати следующей строки. Если при этом происходит выход за пределы страницы, то действие эквивалентно формату PAGE. Если  $w = 1$ , то в формате это число может быть опущено.

Спецификация LINE(w) означает продвижение страницы к строке с порядковым номером w на этой странице. Совместное использование двух спецификаций в виде PAGE LINE(w) означает переход к строке с порядковым номером w на следующей странице.

Спецификация COLUMN ( $w$ ) указывает, что некоторое количество пробелов должно быть вставлено в поток с тем, чтобы очередной символ имел порядковый номер  $w$  в текущей строке. Если в текущей строке уже записано не менее  $w$  символов, то эта строка считается заполненной; начинается новая строка, в начало которой вставляется  $w-1$  пробелов, так что эта новая строка начинается с символа, имеющего порядковый номер  $w$ .

Во всех спецификациях управления для печати, если  $w < 1$ , принимается  $w = 1$ .

Пример. При выполнении оператора

```
PUT EDIT ((Y(I,K) DO K=1 TO 7), B(I)
DO I=1 TO 5) (SKIP,(7)E(14,6),E(24,6));
```

осуществится переход к следующей строке, после чего будет напечатано пять строк, каждая из которых содержит восемь величин.

Во всех случаях выдачи файлов на печать место первого символа в каждой строке не занимает выводимыми данными, а этот символ используется для управления движением бумаги перед печатью. Управляющими символами могут быть: 1 (указывает начало новой страницы), — (предписывает перевод строки), 0 (перевод двух строк), — (перевод трех строк), + (означает блокировку перевода). Не следует предполагать, что в начале выполнения программы бумага находится в начале страницы.

**6. Оператор форматов.** Иногда весь формат или часть формата используется в нескольких операторах ввода или вывода, управляемых редактированием. Можно избежать повторной записи спецификаций в списке форматов, если там предусмотреть косвенный (удаленный) формат вида  $R(M)$ , где  $R$  — индекс спецификации косвенного формата,  $M$  — константа типа метки или переменная типа метки, принимающая значение метки оператора форматов, записываемого в виде

$M: \text{FORMAT } (B);$

Здесь список форматов  $B$  состоит из спецификаций, которые заменяются в операторах GET или PUT косвенным форматом. В этом списке могут присутствовать другие косвенные форматы, но при этом форматы не должны ни непосредственно, ни через другие косвенные форматы ссылаться на первоначальный оператор форматов (т. е. рекурсии здесь не допускаются).

Косвенный формат и соответствующий ему оператор FORMAT должны быть внутренними для одного и того же блока. Если оператор GET (или PUT) является оператором некоторой реакции на прерывание, то он не может содержать косвенного формата. Количество спецификаций косвенных форматов в одном операторе GET или PUT произвольное, как и число обычных спецификаций форматов в формате  $R$ .

Оператор FORMAT относится к числу неисполняемых операторов. Это означает, что если управление будет передано оператору форматов, то он пропускается и управление получает следующий за ним оператор.

Пример использования косвенных форматов:

```
PUT EDIT (A,B,C,D,E) (R(KOC),F(5),R(KC));
KOC: FORMAT (F(6,3),F(7,3,3));
KC: FORMAT (F(6),F(10));
GET EDIT (X, Y) (R(KC));
P: FORMAT (SKIP(2),(2)F(6,3),F(5),SKIP(3));
PUT EDIT (K,L,M) (R(P));
```

В последнем из выписанных здесь операторов используется косвенный формат с меткой P, в соответствии с которым сначала пропускается одна строка, затем печатаются значения элементов K, L, M списка вывода. Поскольку список вывода исчерпан, элемент SKIP(3) списка форматов игнорируется.

**7. Операторы внутренней пересылки данных.** В операторах GET и PUT вместо употребившейся выше конструкции FILE (F) можно использовать конструкцию вида STRING (T), где STRING — дополнение, T — переменная типа строки символов. После такой замены вместо вводимой извне или выводимой в файл информации используется содержащаяся в памяти строка символов с заданным именем, т. е. рассматриваемые операторы осуществляют внутреннюю пересылку данных. Оператор GET распределяет данные по отдельным переменным, оператор PUT объединяет данные в одну переменную.

Конструкция STRING (T) может служить для обозначения строк символов со всеми тремя рассмотренными способами ввода и вывода, однако чаще всего она используется в операторах ввода-вывода, управляемых редактированием.

Пример. Для объединения в одну карточку (одну строку символов) выходных данных о статье в журнале может быть использован следующий оператор:

```
PUT STRING (CARD) EDIT (NAME,TITLE,
VOLUM, PAGE) (A(50),A(20),(2)F(5));
```

где CARD — наименование переменной, которая содержит все данные о статье, NAME — название статьи, TITLE — название журнала, VOLUM — номер тома, PAGE — номер страницы. Переменная CARD должна иметь описатель CHARACTER и длину по крайней мере 80.

Если переменная CARD, заданная в операторе DECLARE с описателем CHARACTER (80), имеет значение строки символов, представляющей собой выходные характеристики некоторой статьи, как приведено выше, то для того, чтобы разместить эти характеристики в различных участках памяти и обработать их, следует выполнить оператор

```
GET STRING (CARD) EDIT (NAME, TITLE,
VOLUM, PAGE) (A(50),A(20),(2)F(5));
```

#### Упражнения

1. Указать, какие из следующих операторов ввода и вывода содержат ошибки:

```
PUT FILE(F) SKIP(1); PUT FILE (F) SKIP COPY;
PUT PAGE FILE(F) SKIP (2);
```

GET COPY SKIP FILE (F) LIST (X);  
GET SKIP 5 FILE (F); GET FILE (F);

2. Определить, что будет напечатано после выполнения операторов:

- a) DECLARE F OUTPUT;  
OPEN FILE (F) LINESIZE (20);  
PUT FILE (F) LIST (1,2,3,4,5,6,7,8) SKIP;
- б) DECLARE E PRINT, I FIXED (2);  
OPEN FILE (E) LINESIZE (15);  
PUT LIST ('BEGIN') FILE (E) SKIP;  
DO I=2 TO -2 BY -1; PUT LIST (I||'END');  
FILE (E) SKIP (1); END;
- в) PUT EDIT ('XYZ','101'B) (A(2), B(2));
- г) PUT EDIT ('ABC','111'B) (A,B);
- д) DO I=1 TO 6; PUT EDIT ('A') (X(I),A); END;
- е) PUT EDIT (A,B,C) (P'\*/99',P'Z/99');

(в случае е) значения переменных A,B,C равны соответственно 123, 45 и 6).

3. Составить программу, которая вводит значения элементов определителя  $D = ||d_{ij}||$  ( $i, j = 1, 2, 3$ ), вычисляет значение определителя и печатает это значение.

4. Составить такой оператор ввода, управляемого редактированием, после выполнения которого переменным A, B и C будут присвоены значения, соответственно равные  $1.23E+01$ ,  $4.56E-01$  и  $1.00E+02$ , при условии, что входной поток имеет вид:

- a) +123E+00 — — — 45.6E-02 — 100E+01 — ;  
б) 12300 — — — +4.56E-11000E+02;  
в) 123E-0145.6E-2100 — — — ;  
г) — 12.3E+00 — — 0.456 — — 100E+02 — — .

## § 18. Операторы ввода-вывода записей

При вводе или выводе записей обрабатываемая информация понимается как дискретные записи, формат которых совпадает с форматом внутренних данных для языка. Назначение ввода заключается в том, чтобы скопировать запись, передав ее с устройства ввода в оперативную память. Назначение вывода — копирование записи с передачей ее из оперативной памяти на устройство вывода. Преобразование при этом не предусматривается.

В качестве переменных, участвующих в операторах ввода и вывода записей, допускаются простые переменные, массивы и старшие структуры. Не разрешены подструктуры и отдельные элементы массивов и структур, формальные параметры процедур. Переменные могут быть также базированными, иметь описатели VARYING или EXTERNAL. Ввод-вывод записей осуществляется в машинном коде, поэтому он удобен для файлов, хранимых на лентах и дисках. Это более быстрая операция, чем ввод-вывод потока.

1. Операторы ввода-вывода последовательного доступа. Для ввода записей с последовательно и индексно-последовательно организованных файлов с последовательным доступом как с описателем INPUT, так и с описателем UPDATE (в режиме обновления, или исправления) используются операторы

```
READ FILE (F) INTO X;
READ FILE (F) INTO X KEYTO (T);
READ FILE (F) IGNORE (E);
READ FILE (F) INTO X KEY (ET);
```

где F — имя файла, с которого осуществляется ввод, X — переменная, массив или структура, T — переменная типа строки символов, E, ET — скалярные выражения, INTO, KEYTO и IGNORE — дополнения.

Первый из перечисленных операторов передает данные из файла в рабочий участок памяти, идентификатор которого указан после дополнения INTO. Записи читаются последовательно одна за другой с начала файла. Оператор READ точно в такой же форме используется при последовательном чтении без ключей при вводе из файлов с описателем INDEXED.

Для файлов с описателем KEYED в операторе READ может быть указано дополнение KEYTO (T), где переменной T присваивается значение ключа, хранящегося в файле перед записью. Второй из приведенных операторов отвечает этому случаю.

Третий из приведенных операторов обеспечивает в случае положительного значения E (дробная часть значения отбрасывается) пропуск соответствующего количества (E) записей из указанного файла. Иначе говоря, следующий оператор ввода при обращении к этому же файлу получит доступ к (E+1)-й записи. Пустые записи в счет не идут. При отрицательных значениях E оператор игнорируется. Операторы

```
READ FILE (F) IGNORE (I);
READ FILE (F);
```

эквивалентны, т. е. в обоих вариантах пропускается одна запись.

Четвертый (последний) из перечисленных операторов позволяет последовательное чтение записей установить с любого места в файле с описателем KEYED (начиная с заданного ключа). При выполнении этого оператора запись из файла F с ключом, заданным выражением ET в дополнении KEY (значение ET преобразуется в строку символов), вводится в память, заданную в виде переменной после дополнения INTO. Начало последовательного ввода для следующих операторов READ устанавливается на запись, непосредственно следующую за записью, прочитанной данным оператором.

Для вывода записей в последовательно организованный файл с последовательным доступом с описателем OUTPUT используется оператор

```
WRITE FILE (F) FROM (X);
```

где дополнение FROM (X) указывает, что значение переменной X необходимо записать в файл с именем F. Каждое выполнение этого оператора выводит очередную подготовленную запись из переменной X.

В случае индексно-последовательных файлов вывод осуществляется оператором

```
WRITE FILE (F) FROM (X) KEYFROM (ET);
```

Каждое выполнение этого оператора выводит очередную запись из переменной X, указанной в дополнении FROM, причем ключ записи задается в дополнении KEYFROM выражением ET, преобразуемым в строку символов. Записи располагаются последовательно с начала файла F.

Исправления записей файлов с описателем UPDATE для любых файлов с последовательным доступом выполняются оператором вывода в виде

REWRITE FILE (F) FROM (X);

При выполнении этого оператора на место прочитанной перед этим записи засылается содержимое переменной X, т.е. последняя прочитанная запись исправляется (обновляется).

Такой же вид имеет оператор вывода из буфера, выполняемый в режиме обновления и используемый для замены существующей записи. Конструкция FROM (X) здесь может быть опущена; в файл заносится последняя прочитанная в буфер запись.

Для файлов, допускающих пустые записи подобно оператору REWRITE, можно использовать оператор исключения записи

DELETE FILE (F);

который последнюю прочитанную запись в файле превращает в пустую. Этот оператор применим только в том случае, если файл имеет индексно-последовательную организацию.

При обработке записей в буфере используется оператор

READ FILE (F) SET (P) KEYTO (T);

где P — переменная типа указателя, T — переменная типа строки символов. Действие этого оператора заключается в том, что очередная запись из файла F переносится в буфер, а указатель P, заданный дополнением SET, принимает значение, указывающее на начало данных этой записи. Это значение сохраняется до тех пор, пока в другом операторе READ следующее дополнение SET не устанавливает тот же самый указатель. Для файлов с описателем KEYED использование дополнения KEYTO (T) аналогично описанному выше.

Оператор размещения в буфере имеет вид

LOCATE X FILE (F) FROM (X);

где X — базированная переменная (массив, структура), обеспечивающая совмещение данных X с областью, определяемой указателем P в файле F. Выполнение оператора LOCATE не сопровождается выводом; вывод данных производится немедленно перед следующим оператором LOCATE или WRITE, в котором встречается обращение к тому же самому файлу.

Оператор размещения в буфере для файлов с описателем KEYED имеет вид

LOCATE X FILE (F) SET (P) KEYFROM (ET);

где смысл указателя P и выражения ET описан выше.

**2. Операторы ввода-вывода прямого доступа.** Особенностью обработки файлов с прямым доступом является то, что программист сам определяет структуру размещения записей (на дисках) и до выполнения программы с помощью специальных директив (не являющихся элементами языка ПЛ/1) обеспечивает подготовку необходимого количества участков для хранения записей.

Для ввода записей с индексно-последовательно организованных файлов с прямым доступом как с описателем INPUT, так и с описателем UPDATE используются операторы

```
READ FILE (F) INTO (X) KEY (ET);
READ FILE (F) SET (P) KEY (ET);
```

где KEY (ET) — дополнение, в котором ET — скалярное выражение. При выполнении этих операторов в конструкции KEY (ET) вычисляется значение выражения ET, которое, будучи представленным целым числом без знака и преобразованным в строку символов, определяет, какая запись вводится. Если запись с указанным ключом в файле не окажется, возникает ситуация KEY.

Второй из приведенных операторов используется в случае файлов с описателем BUFFERED для размещения данных в буфере.

Для вывода записей используется оператор

```
WRITE FILE (F) FROM (X) KEYFROM (ET);
```

где дополнение KEYFROM (ET) указывает, что целое десятичное число без знака, являющееся значением выражения ET, преобразуется в строку символов и представляется в виде ключа записи, который переносится в файл. Длина K строки символов, изображающей ключ, задается описателем файла KEYED (K) в операторе DECLARE. При выполнении этого оператора запись выводится из переменной, указанной в дополнении FROM, и помещается в файл в соответствующем (согласованном с требованием возрастающей ключей) месте. Если в файле уже окажется запись с таким же ключом, то возникнет ситуация KEY. Ситуация KEY возникает также и в том случае, если для выводимой записи не хватает в файле места.

Оператор, исключаящий из файла F запись с ключом ET, прочитанную из переменной X, имеет вид

```
DELETE FILE (F) FROM (X) KEYFROM (ET);
```

где конструкция FROM (X) может отсутствовать. Оператор исключения записи применим только к файлам с описателем UPDATE. Если запись, подлежащая исключению, в файле не окажется, возникает ситуация KEY.

Для замены (обновления) содержания существующей в файле записи используется оператор

```
REWRITE FILE (F) FROM (X) KEY (ET);
```

Перед выполнением этого оператора данная запись должна быть прочитана оператором READ. Поэтому в выполняющихся по порядку операторах READ и REWRITE в дополнении KEY должно быть указано одно и то же значение ключа. В дополнении FROM указывается переменная, содержащая возвращаемую запись.

Операторы ввода и вывода для файлов с региональной организацией имеют точно такой же вид, как и только что рассмотренные операторы для файлов с описателем INDEXED. Оператор WRITE обеспечивает создание файла в режиме прямого доступа, чем достигается рациональное использование

внешней памяти. Оператор READ обеспечивает доступ к любой заданной записи в файле, не прибегая к обработке других записей.

### Упражнения

1. Указать, в каких операторах содержатся ошибки:

```

READ FILE (F) INTO (X); DECLARE X(10,10) FIXED;
READ FILE (F) INTO (X) IGNORE (5);
READ FILE (F) INTO (X(*,3));
WRITE FROM (X); DECLARE X(10);
WRITE FILE (F) FROM (X); DECLARE X(5) CHARACTER (7).
WRITE FILE (F) FROM (X); DECLARE (Y DEFINED X,
X) CHARACTER (80);
REWRITE FILE (F);
REWRITE FILE (F) FROM (X(16));
REWRITE FILE (F) FROM (X); DECLARE X
CHARACTER (20) VARYING;
LOCATE X FILE (F); DECLARE X BASED (P);
LOCATE X SET (P); DECLARE X BASED (Q);
DECLARE F UPDATE; LOCATE A FILE (F);

```

2. Определить, из каких записей будет состоять набор данных, сопоставленный файлу F, после выполнения программы

```

T: PROCEDURE OPTIONS (MAIN);
  DECLARE F ENVIRONMENT (F(10)),
  A PICTURE 'S(9)9' BASED (P);
  DO I=1 TO 3; LOCATE A FILE (F);
  A=I; END; CLOSE FILE (F);
END T;

```

3. Определить, какие значения будут иметь переменные X, Y и Z после выполнения процедуры и указать, в каких случаях будет возникать ситуация RECORD:

```

P: PROCEDURE;
  DECLARE F ENVIRONMENT (F(10)),
  X CHARACTER (5), (Y,Z VARYING) CHARACTER (15);
  ON RECORD (F); OPEN FILE (F) OUTPUT;
  X='BEGIN'; WRITE FILE (F) FROM (X);
  Y=(2)'OUTPUT'; WRITE FILE (F) FROM (Y);
  Z=SUBSTR (Y,2,10); WRITE FILE (F) FROM (Z);
  CLOSE FILE (F); OPEN FILE (F);
  READ FILE (F) INTO (X); READ FILE (F) INTO (Y);
  READ FILE (F) INTO (Z); END P;

```

## § 19. Пример программы

Программа для вычисления определенного интеграла методом Симпсона базируется на формуле

$$\int_a^b f(x) dx = \frac{h}{3} [f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \dots + 2f(b-2h) + 4f(b-h) + f(b)],$$

где  $h = (b-a)/n$ ,  $n$  — четное количество интервалов,  $f(x)$  — подынтегральное выражение.

Пусть задан интеграл

$$Q(p) = \int_a^b \frac{p^v x e^x dx}{(p + e^x)^{v+1}}$$

как функция параметра  $p$ . Заданы интервал изменения  $p$  и константа  $v$ . Вычисление подынтегрального выражения может быть осуществлено с помощью процедуры-функции вида

```
FX: PROCEDURE (X);
  F=X*EXP(X)/(P+EXP(X))**(V+1);
RETURN (F); END;
```

где FX — метка, X — формальный параметр, F — идентификатор результата. Значения этой функции, которые при вычислении интеграла  $Q(p)$  умножаются на множитель 4, получаются обращением к ней с фактическими параметрами, равными  $A+H$ ,  $(A+H)+2H$ , ..., а те значения подынтегрального выражения, которые умножаются на множитель 2, получаются обращением к процедуре-функции с фактическими параметрами  $(A+H)+H$ ,  $((A+H)+H)+2H$ , ...

Суммирование указанных двух групп значений подынтегральной функции может быть выполнено в цикле. Во внешнем цикле вычисляется интеграл  $Q$  в интервале значений параметра  $P$  от  $P_0$  до  $P_K$ .

Программа имеет вид

```
SIMPSON: PROCEDURE OPTIONS (MAIN);
  DECLARE (A,B) DECIMAL FIXED (6,3),
  (V,P) DECIMAL FLOAT (2) EXTERNAL,
  (Q,X,PO,PH,PK) DECIMAL FLOAT (8),
  N DECIMAL FIXED (5,0);
  GET EDIT (PO,PH,PK,A,B,V,N)
  ((3)E(12,5),(3)F(6,3),F(5)); H=(B-A)/N;
  DO P=PO TO PK BY PH; DO; S2, S4=0;
  DO X=A+H TO B-3*H BY 2*H; DO;
  S2=S2+FX(X+H); S4=S4+FX(X); END;
  END; END;
  Q=P**V*H/3*(FX(A)+FX(B)+
  4*FX(B-H)+4*S4+2*S2);
  PUT EDIT (Q,P) ((2)E(13,5)); END;
STOP; END SIMPSON;
FX: PROCEDURE (X);
  F=X*EXP(X)/(P+EXP(X))**(V+1);
  RETURN (F);
END FX;
```

В этой программе переменные  $V$ ,  $P$  описаны как внешние имена, поэтому их значения доступны при вычислении  $F$ , когда выполняется обращение к процедуре-функции с меткой FX.

# Глава V. КРАТКИЕ СВЕДЕНИЯ О НЕКОТОРЫХ ВХОДНЫХ ЯЗЫКАХ

## § 1. Структура и функции операционной системы

*Математическим* (программным) обеспечением или системой математического обеспечения ЭВМ называют комплекс программных средств, предназначенных для повышения эффективности использования машины, облегчения ее эксплуатации и снижения трудоемкости подготовительной работы при решении задач на машине. Система математического обеспечения по своей структуре обычно может быть разделена на две части — *общее* математическое обеспечение и *специальное* математическое обеспечение. Общее математическое обеспечение представляет собой комплекс программ, обеспечивающих применение ЭВМ как некоторой универсальной системы обработки информации, в то время как специальное математическое обеспечение образует программы, которые добавляются к общему математическому обеспечению и решают задачу применения ЭВМ как некоторой специализированной системы обработки информации.

Операционная система, набор пакетов прикладных программ, комплекс программ технического обслуживания и система документации являются составными частями общего математического обеспечения. Здесь будут даны только самые общие сведения о структуре и функциях лишь одной составной части математического обеспечения — операционной системы, которые необходимы при рассмотрении в последующих параграфах этой главы различных входных языков и вопросов подготовки задания для ЭВМ с использованием трансляторов с этих языков. Отметим лишь, что прикладные программы, входящие в общее математическое обеспечение, позволяют расширить возможности операционной системы для специальных комплексов технических средств и способов их применения, а также составляют комплексы программ для решения типовых задач.

В простейшем варианте математическое обеспечение ЭВМ сводится к программным средствам, представляющим собой комплекс из транслятора с одного из алгоритмических языков, библиотеки стандартных программ, программ отладки и, возможно, транслятора с автокода. В этом случае обычно говорят не об операционной системе ЭВМ, а о *системе программирования* на данном языке. Например, это система БЭСМ-алгол, мониторная система «Дубна» и др.

Под *операционной системой* или монитором понимают набор программ, которые организуют непрерывную работу машины в различных режимах без вмешательства оператора. Режим работы ЭВМ — это способ организации решения задачи (задач). Основными являются следующие режимы: однопро-

граммный, режим пакетной обработки, режим разделения времени, режим запрос-ответ, комбинированные режимы. Все эти режимы, кроме первого, относятся к мультипрограммным (многопрограммным) режимам. В однопрограммном режиме работы ЭВМ все устройства машины заняты выполнением только одной программы. В режиме пакетной обработки информации одновременно решается несколько задач, для которых вся необходимая информация (программы, исходные данные) вводится в память машины заранее, до начала решения задач, а в ходе решения вмешательство абонента (пользователя) с пульта не допускается. В режиме разделения времени некоторое число независимых абонентов с помощью периферийных устройств ввода и вывода имеет в процессе решения своих задач непосредственный постоянный и одновременный доступ к ЭВМ. Режим запрос-ответ — это способ организации решения задач, при котором их программы постоянно хранятся в запоминающем устройстве машины, а запросы на их выполнение и необходимые исходные данные поступают извне от абонентов, имеющих прямой доступ к машине. Комбинированный режим работы ЭВМ может быть организован, например, сочетанием режима запрос-ответ или режима разделения времени, с режимом пакетной обработки, или однопрограммным режимом.

В дальнейшем, при рассмотрении входных языков в различных операционных системах, будут использованы понятия и определения, общие для всех операционных систем. Поэтому остановимся на них здесь подробнее.

Основная независимая единица работы ЭВМ, формируемая извне программистом (абонентом), называется *заданием*. Каждое задание обычно описывается с помощью определенных управляющих операторов операционной системы. Описание задания определяет, например, имя задания, начало и конец задания, название программы, которая должна быть выполнена. Задания могут быть представлены либо в виде одиночного задания, либо в виде пакета, образующего входной поток заданий. Задания не зависят друг от друга и могут выполняться одновременно.

По усмотрению программиста задание может быть разбито на *шаги* задания. Шаги задания для одного задания выполняются последовательно, причем выполнение очередного шага зависит от того, насколько успешно реализованы предыдущие шаги. Например, возможны шаги задания: трансляция, редактирование, выполнение программы. Простейшее задание состоит из одного шага.

Совокупность процедур, включающая прием заданий, их контроль, подготовку необходимых программ к выполнению, запуск этих программ и автоматический переход к новому, очередному, заданию, представляет собой *управление* заданиями. Осуществляется управление заданиями управляющей программой операционной системы.

Работа, которая должна быть выполнена программой, указанной в шаге задания, представляет собой *задачу*. Задача может решаться в однопрограммном или мультипрограммном режимах. В зависимости от этого операционная система выделяет для решения задачи необходимые *ресурсы* — время процессора, определенную область оперативной и внешней памяти, необходимые внешние устройства. В однопрограммном режиме все ресурсы находятся в распоряжении одной задачи, в мультипрограммном режиме их распределяет операционная система.

Часть области оперативной памяти, предоставляемой рабочим программам абонента, называют *разделом*. Размеры разделов и их число не являются строго фиксированными и обычно имеют определенные названия.

Совокупность единиц информации образует *запись* (логическую запись), а совокупность записей, объединенных по некоторому общему признаку, образует *файл*. Файлы данных размещаются во внешней памяти. Форматы записей, организация их в физические блоки, способы хранения и поиска отдельных записей определяются свойствами операционной системы. Обработка записей, их поиск, хранение и передача осуществляются программами операционной системы.

Емкость стандартного блока внешней памяти измеряется в *томах*. Например, том — это емкость одной магнитной ленты или одного пакета дисков. Том, содержащий несколько файлов информации, называется многофайловым томом, а файл, располагающийся в нескольких томах, является многотомным файлом. Характеристики файла (или тома), позволяющие операционной системе его опознать, составляют *метку* файла (тома).

В операционной системе обычно устанавливается стандартный набор символических имен, принятых для обозначении логических устройств, которые программист использует в своей исходной программе во всех случаях, когда необходимо обратиться к внешнему устройству. Следовательно, в программе указывается логическое устройство, а не конкретный физический адрес требуемого внешнего устройства. Такой метод, обеспечивающий независимость программ операционной системы и исходных программ абонентов от конкретных физических адресов внешних устройств, принято называть *методом логических устройств*. Логические устройства, которые используются операционной системой для собственных надобностей, называются *системными* логическими устройствами. Для них символические имена зафиксированы постоянно, и каждому из них ставится в соответствие конкретный тип физических внешних устройств.

Программа, составленная на одном из алгоритмических языков, допускаемых данной операционной системой (*исходная программа* или *исходный модуль*), транслируется в так называемый *объектный модуль*. Объектный модуль — это некоторый промежуточный вид записи программы. В объектном модуле специфика исходного языка программирования теряется, однако это еще не рабочая (машинная) программа. Для выполнения на машине объектный модуль должен пройти этап редактирования, после которого получается *абсолютный модуль* (программная фаза). Абсолютный модуль представляется на машинном языке и не подлежит дроблению при вызове его в оперативную память для выполнения. Все модули, как правило, хранятся в соответствующей библиотеке — библиотеке исходных модулей, библиотеке объектных модулей или библиотеке абсолютных модулей. В библиотеке абсолютных модулей хранятся в виде программных фаз также программы самой операционной системы (*системные программы*), поэтому библиотека абсолютных модулей в операционной системе является обязательной (две другие библиотеки могут отсутствовать). В операционной системе могут существовать как системные, так и личные (принадлежащие конкретным абонентам) библиотеки объектных и исходных модулей.

Поскольку перечень и объем операций, выполняемых операционной системой, различен для различных типов операционных систем, то по структуре они могут отличаться друг от друга, однако основные компоненты операционных систем, как правило, одни и те же. Это — управляющая программа и обрабатывающие программы.

Основные функции операционной системы реализуются с помощью *управляющей программы*. Именно она обеспечивает автоматическое выполнение как обрабатывающих программ самой операционной системы, так и программ абонентов в различных режимах работы в рамках одного задания или в пределах различных заданий. Управляющая программа операционной системы выполняет следующие три группы действий: управление данными, управление заданиями и управление задачами.

Управление данными осуществляется комплексом программ (в дисковой операционной системе ЕС ЭВМ — ДОС ЕС — программой, называемой УПРАВЛЕНИЕ ДАННЫМИ), предназначенных для управления процессами организации, идентификации, хранения и выборки всех данных, обрабатываемых в системе.

Комплекс программ, предназначенных для управления процессом прохождения через вычислительную машину непрерывного потока заданий, осуществляет управление заданиями. Например, в ДОС ЕС программа УПРАВЛЕНИЕ ЗАДАНИЯМИ обеспечивает подготовку операционной системы для выполнения пакета заданий.

Управление задачами — это комплекс программ, предназначенных для управления процессом выполнения задач при различных режимах работы, в том числе для управления процессом одновременного выполнения нескольких задач по обработке данных. Этот комплекс программ выполняет функции распределения оперативной памяти, загрузки программ в оперативную память, управление выполнением задач, переключения управления от одной задачи к другой и др. В число программ этой группы входят, например, программы ЗАГРУЗЧИК и СУПЕРВИЗОР. Первая из них (называемая в ДОС ЕС ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА) обеспечивает подготовку операционной системы к работе и загрузку необходимых программ в оперативную память. Среди них будет также программа СУПЕРВИЗОР (или ДИСПЕТЧЕР), осуществляющая управление всем вычислительным процессом на машине.

В состав обрабатывающих программ входят трансляторы, системные обслуживающие (сервисные) программы, а также обрабатывающие программы, написанные абонентами. Обрабатывающие программы предназначены для сокращения объема работы и времени, затрачиваемых на написание, подготовку и выполнение программ абонентов.

Используя языки программирования данной операционной системы, программист получает доступ к ее средствам. Результатом работы трансляторов являются объектные модули. Объектные модули объединяются сервисной программой РЕДАКТОР для получения программ, готовых к загрузке в оперативную память. РЕДАКТОР позволяет вносить изменения в программу без повторной трансляции всей программы, а также разделять программу на части (сегменты), если в этом есть необходимость (например, вследствие недостаточной емкости оперативной памяти).

Среди сервисных программ могут содержаться программа БИБЛИОТЕКАРЬ, которая выполняет функции корректировки и обслуживания библиотек операционной системы, программа СОРТИРОВКА, осуществляющая сортировку или объединение файлов, а также программы УТИЛИТЫ — набор системных программ, обеспечивающих выполнение часто повторяющихся вспомогательных операций (передачу информации из одного запоминающего устройства или устройства ввода (вывода) в другое, обновление программ и данных в библиотеке системы и др.). Имеются также различные *отладочные* программы, назначение которых состоит в оказании помощи программисту при отладке программ. Эти отладочные программы задают так называемый *отладочный режим* выполнения программы.

В целом операционная система ЭВМ представляет собой программно-аппаратный комплекс средств организации и управления процессом функционирования машины. В аппаратную часть операционной системы входят переключатели режимов работы машины и операционной системы, схемы выработки сигналов прерываний работы ЭВМ, регистры прерываний и др. В общем случае чем больше удельный вес аппаратной части, тем сложнее ЭВМ по своей конструкции, но операционная система работает быстрее и наоборот. Программная часть операционной системы, представляющая собой иерархический набор управляющих программ, определяет стратегию и тактику прохождения задач через ЭВМ. Анализ характеристик задач и имеющихся ресурсов определяет стратегию управления, в то время как тактика в основном определяется текущим состоянием машины, количеством задач в машине, временем на их обслуживание и т. д.

Предметом рассмотрения в последующих параграфах этой главы являются главным образом входные языки для различных операционных систем и классов ЭВМ. При этом основное внимание уделяется отличиям (как ограничениям, так и расширениям) входного языка от стандарта соответствующего алгоритмического языка. Вопросы, связанные с оформлением задания для конкретной операционной системы, рассматриваются в минимальном объеме. Поэтому программисту, прежде чем приступить к решению своей задачи на конкретной ЭВМ, необходимо ознакомиться с такими вопросами, как:

- наличие в операционных системах трансляторов с данного языка программирования;
- особенности входного языка выбранного транслятора;
- какие управляющие операторы необходимы для формирования задания;
- правила формирования задания.

В этой главе дается краткая характеристика лишь некоторых входных языков и трансляторов для отечественных вычислительных машин, в настоящее время находящихся в эксплуатации.

## § 2. Подмножества языка алгол-60

В рамках эталонного представления языка определены два подмножества алгола-60: подмножество уровня 1 и подмножество уровня 2. Полный язык, изложенный в гл. II, имеет уровень 0.

Подмножество уровня 1 определяется как уровень 0 со следующими дополнительными ограничениями:

— задан только алфавит из 26 букв, рассматриваемый как алфавит малых букв эталонного языка;

— не включен оператор **own**;

— требуется эквивалентность типов между соответствующими формальными и фактическими параметрами;

— если формальный параметр специфицирован как переменная, массив или функция целого или действительного типов, то в качестве соответствующего фактического параметра можно использовать лишь арифметическое выражение, массив и функцию соответственно целого и действительного типов;

— различие в идентификаторах должно содержаться до тринадцатого символа.

Подмножество уровня 2 содержит перечисленные ограничения уровня 1 и следующие добавочные ограничения:

— список левой части в операторе присваивания может состоять лишь из одной левой части;

— именуемые выражения в списке переключателя могут быть только метками;

— недопустимы ни прямые, ни косвенные рекурсивные вызовы процедур;

— если формальный параметр задан по наименованию, то соответствующий фактический параметр может быть только идентификатором или строкой;

— исключены спецификации **switch**, **procedure**, **Boolean procedure**, **integer procedure**, **real procedure**;

— идентификаторы должны различаться по первым шести символам.

### § 3. Система БЭСМ-алгол

Система БЭСМ-алгол представляет собой основанную на алгоритмическом языке алгол-60 систему программирования для вычислительной машины БЭСМ-6. Система включает в себя транслятор с полного языка алгол-60, библиотеку алгоритмов общего пользования, архив индивидуальных библиотек алгоритмов, транслятор с автокода и библиотеку программ, записанных на языке автокод. В системе предусмотрена удобная для пользователя выдача информации об ошибках в программе, что значительно облегчает отыскание ошибок и отладку программ. В этом параграфе будет рассмотрена только та часть системы, которая связана с транслятором с языка алгол. Транслятор выявляет все синтаксические и многие семантические ошибки. При печати информации об ошибке указывается номер строки и печатается строка, содержащая ошибку.

**1. Входной язык.** Входной язык отличается от языка алгол-60 рядом уточнений и несущественных ограничений. Ограничения носят в основном количественный характер и обусловлены параметрами и особенностями ЭВМ БЭСМ-6 и выбранными методами трансляции. Особенности входного языка являются:

— алфавит входного языка содержит в качестве букв только заглавные буквы латвиского и русского алфавитов; в строках разрешается использовать любые символы устройства подготовки перфокарт, кроме символа  $\diamond$

— диапазон действительных чисел — от  $10^{-19}$  до  $10^{+19}$ ; в мантиссе, как и в целых числах, допускается до 10 разрядов, этим определяется точность их представления;

— длина идентификатора — не более 90 символов;

— рабочая программа, административная система (служебные программы, обеспечивающие нормальное функционирование рабочей программы, стандартных процедур и функций) и динамически активные массивы должны уместиться в оперативной памяти;

— глубина рекурсий, т. е. число последовательно вложенных друг в друга описаний процедур, не должна превышать 9;

— оператор перехода при неопределенном указателе переключателя не эквивалентен пустому оператору, если при вычислении именованного выражения в результате побочного эффекта производится присваивание значений какому-либо переменным;

— все собственные величины трактуются статически, т. е. величины, описание которых начинается описателем *own* и содержится в теле какой-либо процедуры, считаются одними и теми же при всех обращениях к этой процедуре;

— идентификаторы стандартных функций и процедур нельзя использовать в качестве фактических параметров;

— без предварительного описания могут использоваться следующие идентификаторы стандартных функций и процедур:

<i>ABS</i>	<i>COS</i>	<i>LN</i>	<i>OUTPUT</i>	<i>SQRT</i>
<i>ARCSIN</i>	<i>ENTIER</i>	<i>MARG</i>	<i>READ</i>	<i>TAN</i>
<i>ARCTAN</i>	<i>EXP</i>	<i>MAX</i>	<i>SIGN</i>	<i>TG</i>
<i>ARCTG</i>	<i>INPUT</i>	<i>MIN</i>	<i>SIN</i>	<i>WRITE</i>

— спецификации обязательны лишь для формальных параметров, заданных по значению;

— при нормальном выходе из цикла, т. е. при выходе в связи с тем, что список цикла исчерпан, значение параметра цикла равно последнему значению, при котором цикл исполнялся, плюс шаг; последнее, присвоенное ему заголовком цикла значение параметра цикла сохраняет при специальном (т. е. по оператору перехода) выходе из цикла.

2. Стандартные процедуры ввода и вывода. Оператор, записываемый в виде

*INPUT(F);*

где *F* — список фактических параметров (объектов ввода), представляющий собой последовательность разделенных запятыми идентификаторов массивов, простых переменных и переменных с индексами, осуществляет ввод чисел, логических значений и текста.

Каждому фактическому параметру должна соответствовать одна группа (или набор) данных. Простым переменным и идентификаторам массивов со-

ответствуют группы числовых или логических данных. Переменным с индексами соответствуют группы текстовых данных. Количество данных в группе не должно превосходить размеры объектов ввода, которые определяются описаниями.

Группа числовых данных состоит из элементов числового ввода, разделенных запятыми. Заканчивается группа символом ; (точка с запятой). Элементом числового ввода может быть число или число с комментарием перед ним.

Комментарий начинается с буквы, может содержать любой символ языка, кроме : (двоеточие), = (равно) и := (знак присваивания), которые используются для указания конца комментария. При вводе комментариев отбрасывается.

Примеры различной записи одной и той же группы числовых данных:

12, 15, 76, 105;

A=12, B : 15, C := 76, D=105;

КОЭФФИЦИЕНТ: 12, ПАРАМЕТР :=15, C : 76, 105;

Допускается ввод чисел, набитых для машины БЭСМ-6 (с признаком Ч), или для машины БЭСМ-3 (в нормализованном виде). В этом случае запятые, отделяющие одно число от другого отсутствуют, а в конце группы ставится либо символ ; (для БЭСМ-6), либо особый разделитель (для БЭСМ-3).

Группа логических данных представляет собой последовательность логических значений true и false разделенных запятыми. Группа заканчивается символом ; и, как в случае числовых данных, логические значения могут быть снабжены комментарием.

Группа текстовых данных есть строка. При вводе текста символы строки (включая окаймляющие кавычки) заменяются целыми значениями в соответствии с принятой кодировкой и присваиваются последовательно элементам массива, начиная с элемента, указанного в объекте ввода. Например, если группа текстовых данных имеет вид

'ПАРАМЕТР К ='

а в программе содержится оператор

INPUT(X[5]);

то целые значения, соответствующие символам указанной строки, будут помещены в массив X, начиная с элемента X[5], т.е. в элемент X[5] поместится символ ', в элемент X[6] — символ П и т. д.

Вывод чисел, логических значений и текста на печатающее устройство и соответствующее редактирование осуществляется оператором

OUTPUT(F);

где список фактических параметров F представляет собой одну или несколько разделенных запятыми групп, каждая из которых задает либо вывод числовых значений, либо вывод логических значений, либо вывод текста, либо размещение.

Группа вывода числовых значений имеет вид

$NF, A$

где  $NF$  — числовой формат,  $A$  — список объектов вывода (элементы списка разделяются запятыми). Объектом вывода может быть либо арифметическое выражение, либо идентификатор массива.

Числовой формат определяет вид, в котором выводятся значения арифметических выражений и значения всех элементов массивов, указанных в объектах вывода. Вывод осуществляется в том порядке, в каком объекты вывода следуют в списке  $A$ .

Числовой формат представляет собой строку, начинающуюся с символа  $E$  для вывода чисел в экспоненциальной форме или начинающуюся с символов  $Y$  либо  $Z$  для вывода чисел в виде десятичной дроби соответственно с печатью или без печати незначащих нулей в старших разрядах.

В задании формата, кроме первого символа ( $E$ ,  $Y$  или  $Z$ ), могут быть использованы следующие символы:

+ в соответствующей позиции печатается знак плюс, если число положительное, и знак минус, если число отрицательное;

— в соответствующей позиции печатается знак минус, если число отрицательное;

$B$  означает пробел, в соответствующей позиции ничего не печатается;

$D$  означает деситичную цифру, в соответствующей позиции печатается цифра, если только эта цифра не является незначащим нулем в формате  $Z$ ;

. означает точку, которая печатается;

$10$  означает порядок числа, печатается символ  $10$

Размер поля, занимаемого печатаемым числом, равняется количеству вышеуказанных символов, использованных при задании формата. Так, длина поля для формата ' $E-D.DDDDB_{10}+DDBB$ ' составляет 14 позиций.

Для удобства записи несколько подряд стоящих символов  $D$  или  $B$  могут быть заменены конструкцией, содержащей коэффициент кратности в виде целого числа без знака. Так, приведенный формат можно записать в следующей форме: ' $E-D.ADB_{10}+2D2B$ '.

При выводе значение соответствующим образом округляется.

Если при выводе в формате  $Y$  и  $Z$  число не умещается в задаваемый формат, то печатается символ \* и затем нули. Например, при выводе числа 1234.5678 в формате ' $Z+2D.2D$ ' будет напечатано

\*0.00

Если в формате знак не указан, то печатается абсолютное значение. В форматах  $E$  и  $Y$  знак печатается в позиции, определяемой форматом. Формат  $Z$  указывает, что незначащие нули в старших разрядах не печатаются (подавляются), и в этом случае знак печатается либо на месте последнего подавляемого нуля, либо на своем месте, если подавления нулей не было. Так, при выводе числа  $-12.3456$  в формате ' $Z+4D.4D$ ' будет напечатано

$-12.3456$



В первом случае результатом будет вывод второго параметра без кавычек, во втором — вывод элементов (символов) строки, закодированных в виде целых чисел и размещенных в массиве  $A$ , начиная с элемента  $A[i]$ . Например, оператор

`OUTPUT('T', 'ПАРАМЕТР X=', 'Z—D.3D', X);`

обеспечивает вывод, если в результате выполнения получено значение переменной  $X$ , равное 3.46, в виде

`ПАРАМЕТР X = 3.460`

Если в процессе вывода окажется заполненной строка (108 позиций) или страница (60 строк), то вывод продолжается с начала следующей строки (страницы).

Внутри строк, используемых в группах вывода текста, не разрешается употреблять символы ' и ', а символ : (двоеточие) используется только в сочетании с символами  $B$ , /,  $\times$ , 1, 2, 3, при этом соответствующие комбинации означают

- :  $B$  вывод пробела (пропуск)
- :/ переход к началу следующей строки
- : $\times$  переход к началу следующей страницы
- :1 вывод символа ' '
- :2 вывод символа ' '
- :3 вывод символа :

Группа размещения состоит из одного или двух фактических параметров, т. е. представляется в двух формах

`RF RF, E`

где  $RF$  — формат размещения,  $E$  — арифметическое выражение.

Формат размещения представляет собой строку, состоящую из любой последовательности символов  $B$ , /,  $\times$  и целых чисел без знака, предшествующих любому из этих символов. Смысл символов  $B$ , /,  $\times$  был указан при рассмотрении группы вывода текста, а целые константы в формате размещения служат коэффициентами кратности.

Арифметическое выражение в группе размещений определяет число повторений всей совокупности действий, задаваемых форматом размещения. Например, если заданы форматы:

`'7B' '3/5B' '2X', 1 '5B', 3`

то их выполнение сводится соответственно к следующему:

- вывести семь пробелов;
- трижды перейти на новую строку, затем отступить на 5 позиций;
- дважды перейти на новую страницу;
- вывести пять пробелов и выполнить это три раза, т. е. всего отступить на 15 позиций,

**3. Процедура оформления страницы.** При выводе информации на АЦПУ с помощью оператора *OUTPUT* принят стандартный постраничный способ размещения печатаемой информации. Страницы отделяются друг от друга строкой, состоящей из 128 прочерков. В правом нижнем углу каждой страницы печатается ее номер.

Вид страницы характеризуется следующими шестью величинами, которыми являются:

- размер левого поля (в символах),
- длина печатаемой строки (в символах),
- размер правого поля (в символах),
- размер верхнего поля (в строках),
- число печатаемых на одной странице строк,
- размер нижнего поля (в строках).

Для стандартной страницы эти размеры соответственно равны: 10, 108, 10, 5, 60, 5.

Оператор

*MARG(F)*

где *F* — список из шести или семи фактических параметров, позволяет изменить вид страницы, т. е. изменить размещение выводимой информации на странице. Первые шесть фактических параметров указывают соответственно рассмотренные выше шесть размеров, характеризующих страницу. Седьмой фактический параметр (если он имеется) задает начальное значение номера страницы. В случае отсутствия седьмого параметра нумерация страниц начинается с единицы. Например, оператор

*MARG (0, 128, 0, 1, 100, 2, 5);*

размещает выводимую информацию на страницах, начиная со страницы номер 5, которые содержат по 100 полных (128 символов) строк с верхним и нижним полями соответственно в одну и две строки.

Если пятый фактический параметр, задающий число печатаемых на одной странице строк, равен нулю, то при выводе не будет происходить автоматического разбиения на страницы. Переход на новую страницу в этом случае задается явно с помощью символа  $\times$  в группе размещения в операторе *OUTPUT* или с помощью пары символов  $:\times$  при выводе текста.

Если шестой фактический параметр, задающий размер нижнего поля, на котором, в частности, печатается номер страницы, равен нулю, то не будут печататься номер страницы и строка, разделяющая страницы друг от друга (строка из 128 прочерков).

При заполнении страницы всегда имеет силу тот оператор *MARG*, который выполняется последним перед тем как начала заполняться текущая страница. Изменение вида страницы во время ее заполнения не предусмотрено.

Размеры страницы ограничены сверху. Так, число строк на одной странице не должно превышать 150, а размеры верхнего и нижнего полей не должны превышать 40 для каждого из них.

**4. Процедуры обмена.** Обмен информацией между оперативной памятью и внешней памятью машины осуществляется стандартными процедурами

*READ* и *WRITE*. Первая из этих процедур считывает коды с магнитного барабана, ленты или диска в оперативную память, а вторая — записывает коды из оперативной памяти на перечисленные внешние устройства.

Операторы обращения к этим процедурам имеют вид

*READ (F)*; и *WRITE (F)*;

где *F* — список фактических параметров, состоящий из пяти элементов, разделенных запятыми.

Первый из фактических параметров задает начальный адрес области оперативной памяти, участвующей в обмене. Этот параметр должен быть идентификатором массива или простой переменной, его описание определяет объем обрабатываемой информации. Остальные четыре параметра определяют начальный адрес информации на внешних носителях. Эти параметры представляются арифметическими выражениями.

Второй параметр задает номер направления. При обмене с барабанами номерами направлений являются 1 и 2, при обмене с магнитной лентой и дисками — номера 3, 4, 5, 6.

Третий параметр является математическим номером барабана, ленты или диска и может принимать значения от 0 до 7.

Номер тракта (от 0 до 31) в случае обмена с барабаном или номер зоны (от 0 до 511) при обмене с магнитной лентой или дисками определяется четвертым параметром.

Последний, пятый, параметр задает номер слова в тракте или зоне. Одна зона магнитной ленты и один тракт магнитного барабана содержат по 1024 слова и имеют номера от 0 до 1023. Например, оператор

*READ (X, 2, 5, 27, 0)*

осуществляет считывание информации с барабана номер 5 второго направления с тракта номер 27, начиная с нулевого слова, и размещение этой информации в оперативной памяти в массиве *X*. Оператор

*WRITE (Y, 5, 7, 421, 10)*

записывает содержимое оперативной памяти, соответствующее элементам массива *Y*, на магнитную ленту номер 7 пятого направления в зоне номер 421, начиная с десятого слова.

**5. Архив.** Имеется возможность организовать общую и индивидуальные библиотеки алгоритмов, составляющие архив системы. В эти библиотеки помещаются описания процедур, записанные на алголе (либо частично или полностью на автокоде). Для архива выделяется ряд трактов системных (т. е. недоступных программисту) устройств (барабана, ленты или дисков).

В общую библиотеку обычно помещаются алгоритмы, используемые более или менее широким кругом программистов. Комплектуется общая библиотека обслуживающим транслятор персоналом. Программист имеет доступ ко всем алгоритмам общей библиотеки и ко всем алгоритмам тех индивидуальных библиотек, пользование которыми не запрещено их авторами.

При работе с архивом программист может выполнять следующие операции:

- завести в архиве индивидуальную библиотеку;
- поместить алгоритмы в индивидуальную библиотеку;
- изъять алгоритм из индивидуальной библиотеки;
- запретить пользование индивидуальной библиотекой;
- запретить изменения в индивидуальной библиотеке;
- разрешить работу с индивидуальной библиотекой;
- изменить срок существования индивидуальной библиотеки;
- ликвидировать индивидуальную библиотеку;
- отпечатать каталог общей библиотеки;
- отпечатать каталог индивидуальной библиотеки;
- отпечатать текст алгоритма из общей или индивидуальной библиотек;
- отпечатать список алгоритмов, ссылающихся на данный алгоритм из индивидуальной библиотеки.

Для того чтобы можно было пользоваться алгоритмами из архива, перед алгольной программой следует поместить один или более списков наименований алгоритмов в виде

**БИБ:** *IA*; или **БИБ** *IB: IA*;

где *IA* — список наименований алгоритмов, *IB* — наименование библиотеки.

Первый вариант обращения к архиву применяется в тех случаях, когда алгоритмы, используемые в программе, выбираются из общей библиотеки, второй вариант — если алгоритмы выбираются из индивидуальной библиотеки с наименованием *IB*.

Каждое наименование алгоритма — это идентификатор соответствующей процедуры, текст которой находится в библиотеке. Элементы списка наименований разделяются запятыми. Списки наименований друг от друга отделяются символом ; (точка с запятой).

Символ **БИБ** со списком наименований алгоритмов перед алгольной программой означает включение описаний всех перечисленных в списке алгоритмов в дополнительный внешний блок, содержащий в себе алгольную программу. В списках наименований алгоритмов должны содержаться идентификаторы всех библиотечных алгоритмов, явно встречающихся в программе, но не описанных в ней, за исключением стандартных функций и процедур, которые к числу библиотечных не относятся.

## § 4. Алгол-60 ОС ЕС ЭВМ

Транслятор АЛГОЛ-60(F), включенный в состав операционной системы ОС ЕС ЭВМ, используется для преобразования исходных программ, записанных на языке алгол-60, в объектные программы. Транслятор анализирует исходную программу на алголе и генерирует объектную программу, которая может быть обработана РЕДАКТОРОМ СВЯЗЕЙ и в дальнейшем выполнена. Кроме того, при обнаружении в исходной программе ошибок транслятор выдает соответствующие сообщения.

Минимальный объем оперативной памяти, необходимый для работы транслятора АЛГОЛ-60(F), составляет 64К байт.



значащих десятичных цифр от 7 до 8 (короткий формат) или от 16 до 17 (длинный формат).

В списке индексов у переменной с индексами может быть до 16 индексов. В сомнительных случаях арифметическому выражению приписывается действительный тип.

Различают нормальный и специальный выходы из цикла. В первом случае, т.е. при выходе из цикла вследствие исчерпания списка цикла, значение параметра цикла считается неопределенным, во втором, т.е. при выходе из цикла по оператору перехода, параметр цикла сохраняет последнее значение, при котором цикл выполняется.

Существует стандартная функция *LENGTH* (*S*), аргументом *S* которой является строка, а результатом — величина целого типа, а именно, число символов в строке, заключенной в самые внешние строчные кавычки. Например, операторы

$$I := \text{LENGTH} ('AB \_ CD'),$$

$$K := \text{LENGTH} ('END'),$$

присваивают переменным *I*, *K* одинаковое целое значение 5.

Оператор перехода, в именуемое выражение которого входит неопределенный указатель переключателя, не эквивалентен пустому оператору, а является оператором с ошибкой. Требуется эквивалентность типов между соответствующими формальными и фактическими параметрами, вызываемыми по наименованию. Понятие собственной величины не реализовано.

В программе могут содержаться обращения к ранее оттранслированной процедуре определяемой как процедура, написанная на языке алгол, трансляция которой проводится независимо от конкретной программы и к которой могут обращаться несколько программ или процедур на алголе.

Ранее оттранслированная процедура определяется так же, как и процедура, описанная в конкретной программе. Это описание начинается одним из описателей 'PROCEDURE' или ТИП 'PROCEDURE', за которым следуют заголовок процедуры и тело процедуры. Тело процедуры может быть оператором, составным оператором или блоком. Описание такой процедуры не должно заключаться в операторные скобки. Каждая ранее оттранслированная процедура, к которой обращается программа, должна иметь единственный идентификатор процедуры. Идентификаторы, которые содержатся в теле такой процедуры, могут быть либо идентификаторами формальных параметров, задаваемых в заголовке процедуры, либо идентификаторами, локализованными в теле процедуры, либо идентификаторами стандартных процедур.

В программе, в которой содержится обращение к ранее оттранслированной процедуре, должна быть номинально описана эта процедура. Номинальное описание состоит из описателя 'PROCEDURE' или ТИП 'PROCEDURE', заголовка ранее оттранслированной процедуры и ограничителя 'CODE', обозначающего тело процедуры. Например:

$$'REAL' \text{ 'PROCEDURE' } RPR(X, Y).. 'REAL' \ X, Y, 'CODE'$$

Номинальное описание делает возможным любые обращения к ранее оттранслированной процедуре внутри блока или процедуры, в которых

встречается это номинальное описание. К ранее оттранслированной процедуре можно обращаться точно так же, как к обычной процедуре или указателю функции.

**2. Наборы данных при вводе и выводе.** Данные на внешнем устройстве создаются в соответствии с форматами, описанными в процедурах ввода и вывода, и объединяются в файлы. Файл понимается как поименованный набор логически связанных данных, который абонент может рассматривать как непрерывную строку.

В исходной программе каждый файл однозначно идентифицируется целым числом — номером файла. В качестве номеров файлов разрешается использовать числа от 0 до 15. Файлы с номерами 0 и 1 зарезервированы за системными, соответственно входным и выходным файлами *SYSIN* и *SYSPRINT*.

Непрерывная строка данных, составляющая файл, делится на записи. В записи содержится информация, обрабатываемая программой как одно целое. Длина записи зависит от предполагаемого ее использования. Так, для перфорирующего устройства запись содержит 80 символов, для АЦПУ — 128 символов. В том случае, когда запись используется для передачи данных из одной программы в другую, ее длина определяется максимальным числом элементов данных. Это число элементов в записи является фиксированным, называется длиной записи и обозначается обычно символом *P*. Значение *P* не должно превышать 32760.

Учитывая физические границы внешнего носителя, величина *P* может задаваться стандартной процедурой *SYSACT*, второй параметр которой (параметр *FUNCTION*) равен 6, перед тем как создать файл. Файл может быть разделен на секции (участки), если задать фиксированное число *Q* записей для одной секции. Для этого используется та же стандартная процедура *SYSACT* с параметром *FUNCTION*, равным 8. Однако для файлов, разбитых на секции, любая попытка возврата назад в исходной программе не определена, поэтому такие файлы могут использоваться только в процедурах для вывода данных.

Каждый файл может находиться только в одном из трех состояний: открыт, закрыт и исчерпан. На все то время, пока файл открыт, он логически связан с исходной программой, т. е. данные могут считываться или записываться в него. Файл не может быть связан с программой все то время, пока файл закрыт, т. е. ни передача, ни считывание данных невозможны при закрытом файле. Если файл исчерпан, любой оператор процедуры для считывания данных из этого файла не определен.

Открытие файла производится либо после обращения к процедуре *SYSACT* с параметром *FUNCTION*, равным 12, либо, если он еще не открыт, автоматически при выполнении любой процедуры, запрашивающей обмен информацией с этим файлом. Закрывается файл либо при обращении к процедуре *SYSACT* с параметром *FUNCTION*, равным 12, либо автоматически при окончании работы исходной программы, если он был открыт. Файл, который был закрыт, может быть открыт снова в этой же программе или в других программах. \*

При выполнении процедуры ввода, передающей последние данные из файла в программу, происходит исчерпание файла. Он снова становится от-

крытым при возврате назад или выполнении какой-нибудь процедуры ввода.

**3. Стандартные процедуры.** Для работы с файлами в трансляторе предусмотрены стандартные процедуры, которые не требуют описаний в программе. Они считаются описанными в блоке, объемлющем всю программу, и за ними закреплены вполне определенные идентификаторы. Если какой-нибудь из этих идентификаторов будет описан в программе в ином смысле, то соответствующая стандартная процедура станет недоступной.

Стандартные процедуры делятся на четыре класса и обозначаются следующими идентификаторами:

— процедуры ввода — *INSYMBOL*, *INREAL*, *ININTEGER*, *INBOOLEAN*, *INARRAY*, *INTARRAY*, *INBARRAY*;

— процедуры вывода — *OUTSYMBOL*, *OUTINTEGER*, *OUTREAL*, *OUTBOOLEAN*, *OUTSTRING*, *OUTARRAY*, *OUTTARRAY*, *OUTBARRAY*;

— управляющая процедура — *SYSACT*;

— процедуры для работы с рабочими файлами — *PUT*, *GET*.

Процедуры ввода и вывода соответственно передают данные из файла или в файл. Файл задается номером, который является в операторе процедуры первым фактическим параметром, соответствующим формальному параметру, называемому *DATA SET NUMBER*. Эти процедуры обрабатывают файлы последовательно.

Для каждого открытого файла создается указатель записи *S* и указатель знака *R*. При открытии файла указатель записи устанавливается на первую запись файла ( $S=1$ ), а указатель знака — на первый символ в этой записи ( $R=1$ ). При каждой передаче в файл или из него любого символа значение *R* увеличивается на единицу до тех пор, пока это значение не будет равно длине записи *P*. Если  $R=P$  и символ уже передан, *S* увеличивается на 1, а *R* устанавливается равным единице. Эти два указателя однозначно определяют адрес символа внутри файла. Каждая процедура ввода или вывода передает один или несколько символов, начиная с символа, адрес которого задают значения указателей *S* и *R*.

Если файл был разделен на секции по *Q* записей каждая, то после того, как заполнится некоторая секция, значение указателя записей *S* устанавливается равным 1. Таким образом, в этом случае указатель записей *S* считает записи только внутри секции и его значение не превосходит заданного значения *Q*.

Конец записи рассматривается как ограничитель передаваемых данных. Кроме того, последовательность из *K* или более пробелов, встречающаяся в файле, также представляет собой ограничитель передаваемых данных; число пробелов, меньше чем *K*, игнорируется. Обычно для каждого файла  $K=2$ , однако это значение может быть переопределено с помощью процедуры *SYSACT* с параметром *FUNCTION*, равным 10.

Процедуры *INSYMBOL* и *OUTSYMBOL* содержат по три параметра: *DATA SET NUMBER*, *STRING*, *DESTINATION* и *DATA SET NUMBER*, *STRING*, *SOURCE* соответственно. Первый фактический параметр каждой из этих процедур задает файл, указывая его номер, второй является строкой,

представляющей соответствие между символами внешнего носителя информации и программы, третий фактический параметр является переменной или выражением.

В каждой процедуре соответствие между символами заданного файла и значением выражения устанавливается отображением один в один последовательности символов, заданных в строке между самыми внешними кавычками для строк и взятых слева направо по порядку, на последовательность положительных целых чисел 1, 2, 3, ... Используя это соответствие, процедура *INSYMBOL* присваивает третьему фактическому параметру значение, соответствующее текущему символу в заданном файле (определяемому текущими значениями указателей записи и знака). Если символ, соответствующий этой позиции, отсутствует в строке, задаваемой в качестве второго фактического параметра, то принимается значение, равное нулю. Процедура *OUTSYMBOL* передает основной символ, соответствующий значению третьего фактического параметра, по адресу текущей позиции в заданный файл (определяемый указателями записи и знака). В этом случае значение нуль соответствует пробелу, отрицательные значения или значения большие, чем длина заданной строки, приводят к неопределенным результатам. Каждая из рассматриваемых процедур устанавливает указатели записи и знака на следующую позицию внутри файла.

Процедуры *INREAL* и *OUTREAL* содержат по два параметра: *DATA SET NUMBER*, *DESTINATION* и *DATA SET NUMBER*, *SOURCE* соответственно. Эти процедуры осуществляют обмен значениями действительного типа между файлом, заданным первым фактическим параметром, и переменной в программе, заданной как второй фактический параметр. В более общем случае вторым параметром в процедуре *OUTREAL* может быть выражение.

При выполнении процедуры *INREAL* просматривается файл с заданным номером, пока не будет найден первый символ, который является символом некоторого числа. Первый недопустимый символ рассматривается как ограничитель числа. Ограничителем является также либо *K* или более пробелов, либо конец записи. Число, найденное таким образом, преобразуется в действительный тип, и его значение присваивается второму фактическому параметру. После этого ограничитель числа в файле пропускается и значениями указателей записи и знака после выполнения процедуры *INREAL* будет адрес символа, следующего за ограничителем.

При выполнении процедуры *OUTREAL* значение второго фактического параметра преобразуется в число действительного типа во внешнем стандартном формате, т. е. занимает на внешнем носителе поле длиной 13 (короткий формат) или 22 (длинный формат) позиций. Поле содержит последовательно знак числа, первую значащую цифру, точку, 6 или 15 цифр, разделитель ', знак порядка, две десятичные цифры, образующие порядок.

Процедуры *ININTEGER* и *OUTINTEGER* содержат каждая по два параметра, и правила выполнения их аналогичны правилам для процедур *INREAL* и *OUTREAL*, за тем исключением, что указанные процедуры используются для передачи целых значений. В соответствии с этим внешний стандартный формат занимает поле из 11 последовательных символов, один из которых есть знак числа.

Процедуры *INBOOLEAN* и *OUTBOOLEAN* используются для передачи логических значений. Число параметров этих процедур равно двум, и они имеют тот же смысл, что и параметры процедур *INREAL* и *OUTREAL*. Действие процедуры *INBOOLEAN* совпадает с действием процедуры *INREAL*, за тем исключением, что процедура *INBOOLEAN* просматривает указанный файл, и если будет найдена запись, являющаяся логическим значением, то это значение будет присвоено второму фактическому параметру. Процедура *OUTBOOLEAN* записывает значение второго фактического параметра на внешнем носителе во внешнем стандартном формате, занимающем поле из семи последовательных символов в виде 'TRUE\_' или 'FALSE'.

Процедура *OUTSTRING* содержит два параметра *DATA SET NUMBER* и *STRING* и передает символы, заключенные в самые внешние кавычки для строк на месте второго фактического параметра, в файл, задаваемый первым фактическим параметром, начиная с позиции, адрес которой определяется текущими значениями указателей записи и знака.

Процедуры *INARRAY*, *OUTARRAY*, *INTARRAY*, *OUTTARRAY*, *INBARRAY*, *OUTBARRAY* содержат по два параметра и служат для обмена информацией между файлом, номер которого указывается первым фактическим параметром, и массивом программы, задаваемым как второй фактический параметр. Первые две из перечисленных процедур используются в случае действительных массивов, следующие две — для целых массивов и последние две процедуры — в случае массивов логических значений соответственно для ввода и вывода. Возможная многомерная структура массива в исходной программе не отражается на соответствующих данных в файле, где данные располагаются в линейной последовательности.

Управляющая процедура *SYSACT* предоставляет возможность оказывать определенное влияние на процессы ввода и вывода. С ее помощью можно получить некоторые системные параметры, чтобы влиять на характеристики файла и в особенности изменять последовательный порядок передачи данных из файла. Такими параметрами, характеризующими файл, являются: указатель знака *R*, указатель записи *S*, длина записи *P*, число записей в секции *Q*, число пробелов, служащих ограничителем *K*, состояние файла *C* (*C* = 1 означает «открыт», *C* = 0 означает «закрыт», *C* = -1 означает «исчерпан»).

Процедура *SYSACT* содержит три параметра: *DATA SET NUMBER*, *FUNCTION*, *QUANTITY*. Первый параметр, называемый *DATA SET NUMBER*, задает, как и выше, свой файл. Второй параметр *FUNCTION* принимает значения целых чисел от 1 до 15 и в зависимости от заданного значения определяет специальное действие оператора процедуры *SYSACT*. Третий параметр *QUANTITY* определяет переменную в программе (или в особых случаях выражение), которая устанавливает значение одного из системных параметров *R*, *S*, *P*, *Q*, *K* и *C* или который присваивает значение соответствующего системного параметра.

Так, при значении параметра *FUNCTION*, равном 1, действие процедуры *SYSACT* заключается в присвоении фактическому параметру *QUANTITY*, являющемуся переменной, значения указателя знака *R*, т.е. номера текущего символа записи. В этом случае оператор процедуры *SYSACT* определен, если заданный файл открыт или исчерпан.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 2, определен только в том случае, если заданный файл открыт, а параметр *QUANTITY* имеет положительное целое значение, не превосходящее длину записи *P* для заданного файла. Выполняется присваивание указателю знака *R* значения *QUANTITY*.

Если задано значение параметра *FUNCTION*, равное 3, то оператор процедуры *SYSACT* определен в случае открытого или исчерпанного файла. Фактическому параметру *QUANTITY*, являющемуся переменной, присваивается значение указателя записи *S*, т. е. номера текущей записи.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 4, присваивает параметру *S* значение *QUANTITY* и определен в том случае, если заданный файл открыт или исчерпан, не разделен на секции и номер его не равен 0 или 1.

Если значение параметра *FUNCTION* равно 5, то параметру *QUANTITY*, который должен быть переменной, присваивается оператором процедуры *SYSACT* значение длины записи *P* для заданного файла. Предполагается, что файл открыт или исчерпан.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 6, определен только в случае, если указанный файл закрыт и в нем еще нет данных, т. е. файл будет создан операторами процедуры ввода позднее. Действие оператора заключается в определении длины записи *P*, которая задается значением *QUANTITY*.

Если значение параметра *FUNCTION* равно 7, то оператор процедуры *SYSACT* определяет, будет ли заданный файл разделен на секции и присваивает переменной *QUANTITY* значение числа записей, приходящихся на одну секцию.

Если в операторе процедуры *SYSACT* задано значение параметра *FUNCTION*, равное 8, то этот оператор определяет, что файл должен быть разбит на секции, в каждой из которых содержится число записей, заданное положительным целым значением параметра *QUANTITY*. Предполагается, что заданный файл закрыт и в нем еще нет данных.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 9, определен только в том случае, если третий фактический параметр *QUANTITY* является переменной. Этот оператор присваивает переменной *QUANTITY* значение параметра *K*, т. е. определяет число пробелов, которое следует для заданного файла рассматривать как ограничитель данных.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 10, задает параметр *K*, т. е. параметру *K* присваивается значение *QUANTITY*, и с этого момента указанное в *QUANTITY* число пробелов является ограничителем для файла.

Если в операторе процедуры *SYSACT* параметр *QUANTITY* равен 11, то этот оператор присваивает переменной *QUANTITY* значение 1 в случае, когда файл открыт, 0, когда он закрыт, и -1, когда он исчерпан, т. е. оператор процедуры определяет состояние данного файла.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 12, определен только в том случае, если значение третьего фактического параметра *QUANTITY* равно 0 или 1. В первом варианте файл закрывается, если он

не был закрыт, во втором — открывается, если он был закрыт, т. е. этот оператор присваивает значения системному параметру *C*.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 13, определен только в том случае, если данный файл открыт или исчерпан, но не разделен на секции, а параметр *QUANTITY* является переменной. Этот оператор выполняет два действия, внешнее и внутреннее. Внешнее действие совпадает с действием оператора *SYSACT* с параметром *FUNCTION*, равным 3, т. е. переменной *QUANTITY* присваивается текущее значение указателя записи *S*. Внутреннее действие заключается в помещении значения *S* во внутренний индекс, который может быть использован для текущей записи оператором процедуры с параметром *FUNCTION*, равным 4.

Оператор процедуры *SYSACT* с параметром *FUNCTION*, равным 14, определен только в том случае, если файл открыт и значение параметра *QUANTITY* больше нуля. Оставшиеся в текущей записи символы и следующие *QUANTITY*—1 записей пропускаются или заполняются пробелами в зависимости от того, была ли последняя операция, выполненная для этого файла, вводом или выводом.

Действие оператора процедуры *SYSACT* с параметром *FUNCTION*, равным 15, заключается в следующем: перейти к очередной секции в файле, присвоить указателю записи значение *QUANTITY*, установить указатель знака в положение 1 и заполнить все пропущенные позиции пробелами.

Процедуры *PUT* и *GET* дают возможность временно запоминать данные на внешнем носителе без преобразования их во внешний формат и считывать их при текущем выполнении программы. Эти данные задаются описательными процедурами. Процедуры *PUT* и *GET* содержат по два параметра: *N* и *L*.

Процедура *PUT* запоминает на внешнем носителе список значений, задаваемый описательной процедурой, являющейся вторым фактическим параметром, и использует значение первого фактического параметра в качестве идентификационного номера. Если ранее под этим же идентификационным номером хранилась какая-либо информация, то она уничтожается.

Процедура *GET* выбирает список значений, записанный ранее с помощью процедуры *PUT*, используя значение первого фактического параметра в качестве идентификационного номера. Эти значения присваиваются переменным, задаваемым описательной процедурой, идентификатор которой является вторым фактическим параметром.

Описание процедуры, используемой как описательная процедура, должно иметь один формальный параметр, который в свою очередь сам должен быть процедурой с одним параметром. Передаваемые данные должны фигурировать в теле описательной процедуры в качестве параметров процедуры, задаваемой как параметр описательной процедуры.

Пример типичной описательной процедуры:

```
'PROCEDURE'L(E), 'PROCEDURE'E,
'FOR' I.=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN' E(A(I/I)), E(B(I/I)) 'END',
```

При выполнении оператора

```
PUT (N, L),
```

будут переданы значения:  $A(1/1)$ ,  $B(1/1)$ ,  $A(1/2)$ ,  $B(1/2)$ , ... в указанном порядке.

Использование процедур *SYSACT*, *PUT* и *GET* позволяет программисту с повышенной гибкостью и эффективностью управлять потоками данных.

## § 5. Фортран IV ОС ЕС ЭВМ

В операционную систему ОС ЕС ЭВМ входит система программирования на фортране, которая состоит из языка программирования фортран IV, библиотеки программ и трансляторов. Язык фортран IV ОС ЕС ЭВМ включает в себя в виде правильного подмножества язык ASN-фортран, подмножеством которого является в свою очередь язык Basic FORTRAN.

В систему программирования на фортране входят трансляторы: отладочный транслятор ФОРТРАН(G) и оптимизирующий транслятор ФОРТРАН(H). Минимальный объем оперативной памяти, необходимый для работы первого из этих трансляторов, составляет 128К, второго — 256К байт. Транслятор ФОРТРАН(G) имеет высокую скорость трансляции, а транслятор ФОРТРАН(H) — более качественные объектные программы.

Язык обоих трансляторов содержит, с одной стороны, общие элементы расширения языка фортран IV, ставшие уже традиционными для современных реализаций и полностью приводимые в описании языка фортран IV в данной книге, с другой — средства совместимости с ранними версиями языка, в частности с фортраном II. Для совместимости с языком фортран II в фортране IV ОС ЕС ЭВМ используется ряд системных программ библиотеки фортрана IV, вызов которых осуществляется оператором CALL.

Транслятор ФОРТРАН(H) допускает указание повышенной точности во всех описательных операторах, в которых разрешено использование указателя числа байт при описании величин. Повышенная точность задается конструкциями

`REAL *16` и `COMPLEX *32`

т. е. действительные величины объявляются длиной 16 байт, а комплексные — 32 байта. Ввод и вывод данных с повышенной точностью производятся по спецификации Q.

В языке фортран ОС ЕС ЭВМ допускается переменный формат, описанный в п. 4 § 6 этой главы.

Большой набор системных программ операционной системы позволяет эффективно контролировать выполнение программы на всех этапах. В частности, возможен опрос машинных индикаторов, отражающих состояние ЭВМ и устанавливающихся при возникновении ситуаций, которые приводят к так называемому программному сбою машины (например, деление на ноль, переполнение или исчезновение порядка и др.). Возможна также установка программных индикаторов, используемых обычно абонентами в качестве переключателей.

Каждый индикатор может быть в состоянии «включен» или «выключен» (в зависимости от содержимого этого индикатора). Если индикатор содержит

нуль, то он находится в состоянии «выключен». Если содержимое индикатора отлично от нуля, то он находится в состоянии «включен».

Для установки индикаторов используется программа

SLITE (K)

где K — выражение целого типа. С помощью этой программы может быть включен один из четырех индикаторов (если K принимает значения 1, 2, 3 или 4) или же все четыре индикатора выключаются (при  $K = 0$ ).

Проверка и запоминание индикаторов выполняются программой

SLITET (K, J)

где K принимает значения 1, 2, 3, 4 и указывает, какой индикатор проверяется, J — переменная целого типа. В результате выполнения этой программы переменная J получает значение 1 или 2 в зависимости от того, включен или выключен индикатор.

Распечатка данных из оперативной памяти (так называемый сброс памяти) осуществляется программой

DUMP (B1, E1, K1, B2, E2, K2, ...)

PDUMP (B1, E1, K1, B2, E2, K2, ...)

из которых первая прекращает выполнение программы после распечатки, а вторая обеспечивает продолжение выполнения программы после распечатки.

Здесь BN и EN задают начало и конец N-й области оперативной памяти, подлежащей сбросу по формату, задаваемому целой константой KN; BN, EN — идентификаторы переменных как с индексами, так и без них,  $N = 1, 2, \dots$

Установлено следующее соответствие между константами KN и форматами:

0 — шестнадцатеричный	6 — REAL *8
1 — LOGICAL *1	7 — COMPLEX *8
2 — LOGICAL *4	8 — COMPLEX *16
3 — INTEGER *2	9 — текстовый
4 — INTEGER *4	10 — REAL *16
5 — REAL *4	11 — COMPLEX *32

Системная программа

DVCHK (K)

выполняет включение индикатора контроля деления и фиксирует состояние деления на нуль, устанавливая K равным 2, если деление на нуль имеет место, и равным 1 в противном случае.

Программа для проверки переполнения и исчезновения порядка записывается в виде

OVERFL (K)

Если произошло переполнение порядка, т. е. превышение значения, допустимого для порядка, устанавливается K, равное 1. Если будет зафиксировано

состояние исчезновения порядка, т. е. значение порядка стало меньше допустимого значения, К становится равным 3. В обоих этих случаях индикатор включается. В отсутствие перечисленных состояний К устанавливается равным 2. После обращения к программе индикатор выключается.

Выполнение программы прекращается после обращения к системной программе

### EXIT

в записи которой нет параметров.

При использовании транслятора ФОРТРАН(G) в программу могут быть включены операторы отладки DEBUG, AT, TRACE ON, TRACE OFF.

Оператор DEBUG устанавливает режим отладочных операций, относящийся к одной программной единице. Этот оператор предшествует операторам, составляющим вместе с исходной программой пакет отладки, и имеет вид

DEBUG R1, R2, ...

где R1, R2, ... — параметры, определяющие режим отладки. Записываются параметры R1, R2, ... в любом порядке и представляются в одной из следующих форм:

```
UNIT (N1)      INIT (V1, V2, ...)
TRACE          SUBCHK (A1, A2, ...)
SUBTRACE
```

Режим отладки, задаваемый параметром UNIT (N1), где N1 — номер файла, указывает, что все результаты отладки выводятся в файл с номером N1. Если этот режим не указан, результаты отладки выводятся на системное устройство печати. Файл, в который выводятся результаты отладки, называется файлом отладки.

Режим отладки, называемый режимом трассирования меток, задается параметром TRACE и используется для прокрутки программы, т. е. вывода в файл отладки меток выполняемых операторов участка программы, заданного в пакете отладки. Если режим TRACE не указан, прокрутка не выполняется.

Параметр, записанный в виде SUBTRACE, осуществляет заказ на трассирование имени данной программной единицы.

Режим, задаваемый параметром INIT (V1, V2, ...), где V1, V2, ... — наименования переменных или массивов, действует на всю программную единицу и осуществляет вывод в файл отладки значения любого элемента из списка V1, V2, ... каждый раз, когда этот элемент получает значение.

Режим проверки правильного использования индексов указывается параметром SUBCHK (A1, A2, ...), где A1, A2, ... — наименования массивов. Если индекс какого-либо элемента любого из массивов A1, A2, ... превышает допустимое для данного массива значение, в файл отладки выдается сообщение с указанием соответствующего элемента.

Оператор

AT M

где  $M$  — метка оператора, определяет начало того участка программной единицы, где должны выполняться операторы отладки.

Операторы

TRACE ON и TRACE OFF

осуществляют соответственно включение и отключение трассирования меток (прокрутку программы), если в операторе DEBUG был указан режим TRACE.

Оптимизирующий транслятор ФОРТРАН(Н) предоставляет дополнительные гибкие средства организации ввода и вывода с помощью операторов READ и WRITE, называемых операторами асинхронного ввода и вывода, и оператора WAIT — оператора асинхронного ожидания.

Асинхронный ввод выполняется оператором

READ (N1, I=K) A

где  $N1$  — номер файла,  $I$  — целая переменная,  $K$  — выражение целого типа, значение которого однозначно идентифицирует данный оператор,  $A$  — список наименований массивов.

Асинхронный вывод осуществляется оператором

WRITE (N1, I=K) A

где смысл обозначений  $N1$ ,  $I$ ,  $K$ ,  $A$  тот же, что и в предыдущем операторе. Асинхронное ожидание вызывается оператором

WAIT (N1, I=K, COND=J, NUM=N) A

где  $N1$  — номер файла,  $I$  — целая переменная,  $K$  — выражение целого типа, идентифицирующее своим значением один из операторов асинхронного ввода или вывода (не обязательно принадлежащий той же программной единице, что и оператор WAIT). По оператору WAIT производится приостановка выполнения программы вплоть до момента завершения ввода или вывода соответствующими операторами READ или WRITE. Оператор WAIT всегда относится к тому оператору READ или WRITE, который еще не завершен другим оператором WAIT, имеет тот же идентифицирующий номер и относится к тому же самому файлу.

В операторе WAIT параметр  $J$  — идентификатор целой переменной, значение которой после выполнения оператора устанавливается равным 1 при успешном завершении оператора ввода или вывода, к которому данный оператор WAIT относится, равным 2 при возникновении состояния ошибки при передаче информации и равным 3 при обращении к записи END FILE;  $N$  — идентификатор целой переменной, значение которой после завершения действия оператора WAIT становится равным числу байт переданной информации.

Материал этого параграфа соответствует версии 4.1 операционной системы ОС ЕС ЭВМ.

## § 6. Фортран для ЭВМ БЭСМ-6

Версия языка фортран, транслятор с которого установлен на машине БЭСМ-6, разработана на основе входного языка Церн-фортран для ЭВМ CDC. Транслятор с этой версии является составной частью системы

программирования «Дубна» (мониторной системы «Дубна»), поэтому входной язык называется обычно дубненской версией фортрана (фортран-Дубна).

**1. Отличия входного языка от фортрана IV.** Отличия, которые имеет дубненская версия языка фортран от языка фортран IV, вызваны главным образом особенностями ЭВМ БЭСМ-6. Так, целая константа может занимать полную ячейку памяти в виде 48-разрядного ненормализованного двоичного числа, откуда следует, что целая константа может состоять не более чем из 12 десятичных цифр, т. е. абсолютное значение должно находиться в диапазоне от 0 до  $2^{40} - 1$ .

При записи действительных констант в форме E, а также чисел с двойной точностью наличие порядка после символов E или D обязательно (даже если порядок равен нулю). Например, число 1.2345 можно записать так:

1.2345E+00 1.2345E0 1.2345D0

Обязательна хотя бы одна цифра в целой или дробной части мантииссы, которая может иметь до 12 (форма представления E) или 24 (форма D) десятичных цифр. Значение действительной константы лежит в интервале  $10^{-19}$ ,  $10^{+19}$ .

Не рассматриваются комплексные константы с двойной точностью и шестнадцатеричные константы. В то же время вводятся восьмеричные константы, которые представляются в виде последовательности восьмеричных цифр, заканчивающейся символом В. Восьмеричная константа может содержать до 16 цифр. Допускаются отрицательные константы, цифрам которых предшествует знак минус. Например:

1234567В -376В +77300В

Для восьмеричных констант арифметические действия не определены, поэтому эти константы обычно используются лишь в операторе DATA для получения нестандартного содержимого ячейки оперативной памяти.

Для ввода и вывода восьмеричных величин используется спецификация

Ow

где O — индекс спецификации, w — целая константа, указывающая длину поля.

При вводе прочитывается содержимое w позиций (колонок на перфокарте). Пробелы в поле ввода игнорируются, символ В не указывается.

При выводе по O-спецификации выводятся w младших восьмеричных цифр, если  $w < 16$ . Если  $w \geq 16$ , то выводятся 16 восьмеричных цифр — содержимое полной ячейки памяти, а оставшиеся левые позиции поля занимаются пробелами. Символ В на поле вывода не указывается.

Текстовая константа может содержать не более 120 символов.

Алфавит транслятора расширен за счет заглавных букв русского алфавита и символа  $\diamond$

Максимальное целое число, используемое в качестве метки, равно 32767.

На одной перфокарте можно располагать несколько операторов. В этом случае они отделяются друг от друга символом  $\diamond$ . После символа  $\diamond$  нельзя

записывать оператор с меткой. Нельзя символ  $\diamond$  ставить непосредственно после оператора FORMAT.

Символ  $\$$  не является буквой и может использоваться вместо символа  $\diamond$  для указания конца оператора.

Идентификатор, кроме букв и цифр, может содержать любое число пробелов. При этом пробелы транслятором игнорируются, а остаются лишь значащие символы. Таким образом, идентификаторы, состоящие из одних и тех же значащих символов (например, SIGMA, S I G M A, S I G M A) считаются одинаковыми. Если идентификатор состоит более чем из шести значащих символов, то такой идентификатор при трансляции преобразуется к виду, содержащему лишь первые шесть значащих символов, выдается соответствующая информация (предупредительная диагностика) и трансляция продолжается.

В операторах описания типа не должен содержаться указатель длины. В соответствии с этим из величин нестандартной длины рассматриваются только действительные величины с двойной точностью, описываемые оператором DOUBLE PRECISION. Операторы описания типа не используются для присваивания начальных значений. Отсутствует оператор IMPLICIT.

Максимальная размерность массивов равна трем. В качестве индекса может быть использовано лишь целое выражение вида  $K, J, J \pm K, K * J, K * J \pm K_1$ , где  $K, K_1$  — целые константы,  $J$  — целая переменная. Если элемент массива указан без индексов, то считается, что указан первый элемент массива. Это правило соблюдается для всех операторов, кроме операторов ввода и вывода.

Все арифметические операции одного ранга выполняются последовательно слева направо (включая возведение в степень). Так выражение  $A ** B ** C$  понимается как  $(A ** B) ** C$ . Поскольку комплексные величины с двойной точностью не определяются, в одном арифметическом выражении не могут участвовать величины комплексные и с двойной точностью.

Первым в основной программе должен быть оператор  
PROGRAM NAME

где NAME — идентификатор (наименование) программы.

Библиотека стандартных подпрограмм содержит 55 функций.

Наименование входа в оператор ENTRY должно быть согласовано с наименованием подпрограммы-функции (если оператор ENTRY используется в подпрограмме-функции) по типу, а также по списку формальных параметров. Максимальное число дополнительных входов в подпрограмму, т. е. включенных в подпрограмму операторов ENTRY, не должно превосходить 19.

Количество формальных параметров в описаниях оператор-функций, подпрограмм-функций и подпрограмм не должно быть больше 63.

Отсутствуют формальные параметры в виде символа \*, а также формальные параметры, заключенные в символы /, т. е. вызываемые по наименованию. Не определяется оператор RETURN I.

В одной программной единице идентификатор общего блока может встречаться не более одного раза. Длина общего блока, уже загруженного в память, например в основной программе, не может быть увеличена подпрограммами.

Допускается отличная от фортрана IV форма записи оператора DATA и переменный формат. Не рассматриваются спецификации преобразования Z и G и редакционная управляющая спецификация T.

В операторах ввода и вывода используются логические номера внешних устройств. Расширен список управляющих символов. Отсутствуют операторы

```
READ (N1, N2, END=N3, ERR=N4) L
NAMELIST P
```

В то же время дополнительно введены операторы внутренней передачи информации ENCODE и DECODE.

**2. Видоизмененный оператор DATA.** Оператор присваивания начальных значений записывается следующим образом:

```
DATA (A=R), (B=S), ...
```

где A, B, ... — простые переменные, переменные с индексами или идентификаторы массивов, R, S, ... — константы или последовательности констант, разделенных запятыми.

Если в списке констант имеются повторения, то запись списка можно сократить, заключив повторяющиеся константы в скобки и поставив перед открывающей скобкой целую константу, равную числу повторений (коэффициент кратности). Например, если в программе содержатся операторы

```
DIMENSION X(2, 3), Y(5)
DATA (A = 3.5), (X(1,3) = 2.7), (Y = 1.4,, 3(1.5), 1.6)
```

то начальные значения получат: простая переменная A, элемент массива X с индексами 1 и 3, все элементы массива Y.

Тип переменных в операторе DATA определяется типом присваиваемых им констант, а не оператором описания типа. Например, хотя оператор

```
DIMENSION Z(4)
```

описывает действительный массив, после выполнения оператора

```
DATA (Z=1.5, 2, 2(.TRUE.))
```

только первый элемент массива будет иметь действительное значение; элемент Z(2) получит целое значение, а Z(3) и Z(4) — логические значения.

Оператором DATA можно оставлять часть массива незаполненной, однако в списке оператора DATA не может быть указано больше констант, чем требуется для заполнения массива.

В операторе DATA допускается запись переменных в виде неявного цикла. Например:

```
DIMENSION A(5)
DATA (A(I), I=1, 4)=2(1,-1.))
```

В этом примере элементы A(1) и A(3) получают значения, равные +1.0, а элементы A(2) и A(4) — значения, равные -1.0. Элемент A(5) не получает начального значения.

Наряду с описанной формой записи оператора DATA возможна запись этого оператора в обычной форме, принятой в языке фортран IV.

**3. Ввод и вывод.** Каждому устройству, с которым производится обмен информацией, задаваемой операторами ввода и вывода, присвоен определенный номер, называемый логическим номером. На БЭСМ-6 магнитофон (накопителям на магнитных дисках) присвоены логические номера с 1 по 15, магнитному барабану — 16, вводному перфоратору — 50, печатающему устройству — 51, выводному перфоратору — 52.

Логический номер означает, что реальное устройство, имеющее данный логический номер, может быть любым из однотипных устройств (например, магнитофонов). Соответствие логических номеров реальным устройствам устанавливается операционной системой, распределяющей эти устройства между параллельно решаемыми задачами. Одна программа может использовать не более одного АЦПУ, не более одного выводного перфоратора, не более 15 магнитофонов и не более одного магнитного барабана. Использование накопителей на магнитных дисках вместо магнитофонов организуется путем запроса их в управляющих операторах пакета задачи. Логический номер устройства, с которым производится обмен, указывается в операторах ввода и вывода.

Управляющими символами при печати являются

0 + 1 4 5 6 S

которые соответственно вызывают:

- перевод двух строк перед печатью;
- печать без перевода строк;
- прогон до первой строки следующей страницы, представляющей один блок из 66 строк;
- прогон до первой строки следующего блока, где на странице два блока по 32 строки;
- прогон до первой строки следующего блока, где на странице три блока по 21 строке;
- прогон до первой строки следующего блока, где на странице четыре блока по 15 строк;
- отмену проверки переполнения страницы.

Любой символ в первой позиции строки, отличный от приведенных, вызывает пропуск одной строки перед печатью. Управляющие символы в первой позиции строки не печатаются.

Поиск признака конца предыдущего файла с целью формирования очередного файла осуществляется в прямом направлении управляющей программой SCHEOF, обращение к которой имеет вид

CALL SCHEOF (N1)

где N1 — целая константа или переменная, равная логическому номеру магнитной ленты (диска). По окончании поиска лента (диск) устанавливается в начале следующего файла, если таковой имеется, т. е. вызовом программы SCHEOF можно перемотать ленту (провернуть диск) вперед на один файл.

Вместо операторов ввода и вывода можно использовать комплекс управляющих программ, позволяющих работать с внешними накопителями информации на БЭСМ-6 как устройствами прямого доступа.

Единицей записи на БЭСМ-6 является: одна полная строка бумажной ленты на печатающем устройстве (128 позиций), одна перфокарта (80 колонок), одна физическая запись (144 символа), один или несколько физических записей (рекордов), составляющих одну логическую единицу записи, массив ячеек памяти, содержащий определенное число символов, указанное в операторах ENCODE и DECODE.

Одна логическая единица записи образуется при работе одного бесформатного оператора вывода. В общем случае каждый массив информации, записанный с помощью одного оператора вывода, образует одну или несколько логических единиц записи. Логическая единица записи подразделяется на массивы длиной по 24 машинных слова (физические записи) длиной 48 двоичных разрядов. При образовании физической записи к ней приформировываются два служебных слова. В каждой зоне магнитной ленты (диска) или в каждый тракт магнитного барабана помещается по 39 таких записей (состоящих из 26 машинных слов). При этом емкость одной ленты составляет 512 зон, а емкость одного магнитного барабана — 32 тракта.

**4. Переменный формат.** В процессе выполнения программы допускается формирование списка спецификаций оператора FORMAT, т.е. может быть использован так называемый *переменный формат*. В этом случае список спецификаций вместе с окаймляющими его скобками заносится в виде текстовой константы в заранее зарезервированный массив или присваивается простой переменной. В операторе ввода или вывода вместо метки оператора FORMAT указывается наименование переменной или массива переменных, в котором хранится список спецификаций.

Элементы списка спецификаций или весь список вместе с окаймляющими скобками как текстовые константы могут быть присвоены переменным или элементам массивов либо с помощью оператора DATA, либо вводом по спецификации A. Например, если в программе содержатся операторы

```
DIMENSION A(6), K(6)
FORMAT(F10.3/2(F8.2, F5.1), F7.0)
READ 25, A
```

то значения элементов массива будут введены, согласно списку спецификаций оператора FORMAT с меткой 25. Использование части списка, например, стоящей после символа /, возможно, если написать новый оператор FORMAT, списком спецификаций которого будет указанная часть предыдущего списка. Однако удобнее занести полный список рассматриваемого оператора FORMAT в определенное место памяти, например в массив K, и в каждом из двух случаев обращаться к началу соответствующего списка спецификаций. Для этого достаточно вместо оператора FORMAT в программу включить следующий оператор:

```
DATA (K(1)=(F10.3/), (K(2)=2(F8.2, F5.1), (K(3)=F7.0))
```

Теперь оператор

```
READ K, A
```

указывает, что значения элементов массива *A* будут введены в соответствии со списком спецификаций, который в виде символов составляет список значений элементов массива *K*.

Для того чтобы использовать весь список спецификаций, в операторе *READ* указывается идентификатор массива или первый элемент массива. Если же в операторе *READ* указать, например, элемент *K(2)*, то преобразование вводимых значений будет выполняться по списку спецификаций, начиная со значения элемента *K(2)*, т. е. используется список (2(F8.2, F5.1), F7.0).

**5. Операторы внутренней передачи информации.** Операторы внутренней передачи информации, или операторы *ENCODE* и *DECODE*, аналогичны операторам *WRITE* и *READ* с той разницей, что в передаче информации с помощью операторов *ENCODE* и *DECODE* внешние устройства не участвуют. Происходит передача информации с одного участка оперативной памяти на другой с преобразованием передаваемой информации согласно спецификациям оператора *FORMAT*.

Оператор *ENCODE* преобразует информацию, заданную в форме, определяемой списком спецификаций оператора *FORMAT*, и последовательность текстовых символов. Оператор *DECODE* производит обратное преобразование, т. е. перекодирует информацию, заданную в виде последовательности текстовых символов, в ту форму ее представления, которая определяется соответствующей спецификацией оператора *FORMAT*.

Оператор *ENCODE* записывается в виде

```
ENCODE (N, N2, M) L
```

где *N* — количество символов в записи (т. е. длина логической единицы записи), это целая константа или простая переменная целого типа, *N2* — метка оператора *FORMAT*, в соответствии с которым производится преобразование информации, *M* — наименование массива или переменной, куда передается информация, получаемая в текстовом виде, *L* — список переменных, имеющий структуру списка ввода-вывода.

Оператор *DECODE* имеет вид

```
DECODE (N, N2, M) L
```

и производит перекодировку текстовой информации, заданной на участке оперативной памяти, определяемом идентификатором *M*, в соответствии со списком спецификаций оператора *FORMAT*, имеющего метку *N2*, с последующим запоминанием этой информации на участке оперативной памяти, указанном списком *L*.

Таким образом, оператор *ENCODE* преобразует информацию из двоичных слов, размещая ее по шесть символов в ячейку. Если значение параметра *N* не кратно шести, то единица записи заданной длины *N* заканчивается в середине ячейки (машинного слова), и оставшееся свободное место в последней ячейке заполняется пробелами. В операторе *DECODE*, производящем обратное преобразование, избыточные символы, т. е. символы, заполняющие последнюю

ячейку в единице записи длиной N, пропускаются. Например, если переменная L имеет значение 251, то с помощью операторов

```
ENCODE (3, 7, I) L
7 FORMAT (I3)
```

произойдет перекодировка этой величины в текстовую константу вида 3H251, значение которой получает переменная I. Оператор

```
DECODE (3, 7, I) L
```

произведет перекодировку текстовой константы 3H251 в целую константу 251 и обратное присваивание этого значения переменной L.

**6. Формирование пакета задач.** Трансляторы со всех входных языков имеют своим выходным языком так называемый язык загрузки. Управляющая программа МОНИТОР организует процесс обработки всех задач, вызывая соответствующие системные программы для выполнения необходимых действий. Определение необходимых действий и порядок их выполнения основаны на анализе информации, содержащейся в управляющих операторах. Карты, на которых набиты управляющие операторы (управляющие карты), вместе с картами самой задачи (подпрограммами и данными) составляют пакет задачи.

В управляющих картах содержится как обязательная информация о задаче, так и необязательная информация. К обязательной информации относится заказ на необходимые ресурсы, которые, в частности, включают в себя объем оперативной памяти, отводимой для задачи, лимит машинного времени, требуемый для решения задачи, внешние устройства, которые использует задача, библиотеки программ, к которым обращается задача, а также информация о режиме работы системы, указывающая, какие трансляторы и в каком порядке должны быть вызваны для трансляции отдельных подпрограмм задачи, какие дополнительные действия требуются от системы в процессе трансляции (печать текстов подпрограмм, выдача подпрограмм на перфокарты и т. д.), какие данные использует задача — текстовые или двоичные, требуется ли задаче выход на счет, надо ли печатать (в случае выхода на счет) таблицу загрузок, содержащую адреса подпрограмм и общих областей памяти, какие дополнительные операции до выхода задачи на счет надо выполнить (редактирование, запись в личные библиотеки и др.).

Без заказа задаче выделяются устройства ввода и вывода, магнитные барабаны, а также нестандартные внешние устройства (например, графопостроители). Без заказа каждая задача получает доступ к библиотеке стандартных подпрограмм. Предусмотрено в случае отсутствия заказа стандартное выделение ресурсов в объеме 24К ячеек оперативной памяти и 5 минут машинного времени. В заказе можно указать объем оперативной памяти не более 32К. В процессе счета задача не может затребовать ресурсы сверх указанных.

В случае отсутствия информации о режиме работы системы устанавливается стандартный режим, который содержит режим трансляции с языка фортран, выдает абоненту тексты подпрограмм на их входных языках, снаб-

женные некоторой дополнительной информацией, не предусматривает выход на счет, выдает таблицы загрузки, рассматривает все данные как текстовые.

Обязательная информация содержится в трех управляющих картах и одной дополнительной (не управляющей) карте.

Управляющая карта содержит символ \* в первой колонке, после которого указывается фиксированный текст, определяющий тип управляющей карты; далее следует некоторая дополнительная информация, зависящая от типа управляющей карты.

Пакет задачи начинается с управляющей карты вида

\*NAME — ТЕКСТ

где ТЕКСТ — это произвольный набор символов, содержащий хотя бы один символ, отличный от пробела; обычно здесь помещается имя (фамилия) абонента.

Следующей обязательной управляющей картой в пакете является карта вида

\*PASS — ШИФР

где ШИФР — это специальное наименование (шифр, пароль), присвоенное данному абоненту и открывающее ему доступ к машине. Шифр, состоящий не более чем из шести букв и цифр и начинающийся с буквы, заранее вносится в каталог абонентов данной машины, который хранится в мониторной системе.

После этих управляющих карт, если в этом есть необходимость, следуют карты заказа ресурсов, за ними расположены карты самой программы, т. е. карты, на которые набиты все программные единицы.

Последней управляющей картой пакета должна быть карта

\*END — FILE

после которой ставится еще одна дополнительная карта, обозначающая признак конца ввода пакета (так называемый диспетчерский конец).

Скорость трансляции пакета — около 10—20 операторов в секунду. Качество оттранслированной программы зависит от состава операторов исходной программы. Трансляция арифметических выражений, не содержащих переменных с индексами, производится практически оптимально.

Трансляция программ с фортрана осуществляется в два этапа. На первом этапе производится трансляция на автокод, затем трансляция с автокода, в результате которой выдается окончательный результат трансляции в виде программного модуля на языке загрузки.

Диагностика ошибок производится на всех этапах обработки пакета задачи, а именно: при декодировке, при трансляции с фортрана на автокод, при трансляции с автокода на язык загрузки, при загрузке подпрограмм в память и при счете. Транслятором выдается информация, достаточно точно определяющая характер каждой ошибки и позволяющая легко их обнаружить и исправлять.

## § 7. Фортран ДОС ЕС

В дисковой операционной системе ДОС ЕС фортран представлен двумя версиями — базисным (основным) фортраном ДОС и фортраном IV ДОС. Система программирования для каждого из этих языков включает язык программирования, транслятор и библиотеку программ. Варианты языка фортран, реализованные в ДОС ЕС, полностью согласуются с соответствующими стандартами указанных версий. По сравнению со стандартами языка ДОС ЕС содержат ряд дополнений, связанных с особенностями ЕС ЭВМ и с расширением возможностей языков.

Язык базисный фортран ДОС отличается простотой и отсутствием сложных конструкций. Он легок при изучении и удобен для программирования. Вместе с тем этот язык предоставляет программисту больше возможностей по использованию средств ввода и вывода. Фортран IV ДОС полностью включает базисный фортран ДОС. К дополнительным возможностям фортрана IV ДОС следует отнести наличие операторов отладки и более широкие возможности ввода и вывода данных.

Трансляторы с базисного фортрана ДОС и фортрана IV ДОС переводят программу на исходном языке в объектный модуль, который представляет собой программный модуль в промежуточном формате, общем для всех трансляторов операционной системы. Во время трансляции выдается диагностическая информация, облегчающая поиск ошибок в исходной программе. Объектный модуль может содержать символические адреса, не определенные во время трансляции, а также некоторую другую информацию, поэтому до непосредственного выполнения на машине объектный модуль проходит обработку программой РЕДАКТОР.

Объединение объектных модулей в программные фазы (абсолютные модули) происходит независимо от того, когда и с какого языка программирования транслировался тот или иной модуль. Фаза может собираться из независимо оттранслированных частей и подпрограмм, хранящихся в библиотеках. В соответствующих библиотеках могут храниться все программы на любой стадии подготовки (исходный модуль, объектный модуль, абсолютный модуль). Библиотека абсолютных модулей обязательна для функционирования операционной системы ДОС ЕС. Библиотеки базисного фортрана ДОС и фортрана IV ДОС являются частью библиотеки объектных модулей и включают программы вычисления различных математических функций, организации ввода и вывода и программы выполнения ряда служебных (системных) действий.

Трансляторы с базисного фортрана ДОС и фортрана IV ДОС в качестве рабочих файлов могут использовать диски и магнитные ленты. Оба транслятора могут быть отредактированы для выполнения в режиме пакетной обработки в любой части оперативной памяти (в любом разделе в мультипрограммном режиме). Для работы транслятора с базисного фортрана ДОС требуется не менее 10К байт оперативной памяти, а для выполнения трансляции с фортрана IV ДОС — 40К байт оперативной памяти.

1. Отличия фортрана IV ДОС от стандарта языка фортран. Фортран IV ДОС включает в себя ряд элементов, отсутствующих в более ранних вер-

ниях языка фортран, в том числе и в стандарте языка фортран. Изложение языка фортран IV в гл. III велось с учетом всех дополнительных возможностей, так как эти дополнения уже стали неотъемлемой частью языка, называемого фортраном IV. Не рассматривались лишь средства отладки и управляющие операторы, т. е. служебные программы фортрана IV, которые могут существенно зависеть от операционной системы. Переменный формат описан в этой главе в § 6.

Перечислим элементы, которые содержатся в языке фортран IV ДОС, но отсутствуют в стандарте (1966 г.) языка фортран. К ним относятся:

- знаки & и ' среди символов языка;
- шестнадцатеричные константы;
- текстовые константы, заключенные в апострофы;
- размерность массива, превосходящая 3;
- выражения смешанного типа;
- многократное возведение в степень без скобок, указывающих порядок вычислений;
- произвольная форма индексного выражения;
- присвоение начальных значений в операторах явного описания типа;
- оператор IMPLICIT;
- указание длины переменных и массивов в операторах явного описания типа и длины функции в операторе FUNCTION;
- наименование массива в операторе DATA;
- оператор PAUSE 'ТЕКСТ';
- стандартные встроенные функции: DINT, DMOD, DFLOAT, IFIX, DCMPLEX, DCONJG;
- стандартные внешние функции: CDEXP, CDLOG, CDSIN, CDCOS, DTANH, CDSQRT, CDABS, ARSIN, DARSIN, ARCOS, DARCOS, SINH, DSINH, COSH, DCOSH, ERF, DERF, ERFC, DERFC, GAMMA, DGAMMA, ALGAMA, DLGAMA, COTAN, DCOTAN, TAN, DTAN, CABS, CDABS;
- формальные параметры, заключенные в символы /;
- текстовая константа как фактический параметр при обращении к подпрограмме-функции;
- оператор RETURN I;
- оператор ENTRY;
- переменные размеры массивов, передаваемые через общие области COMMON;
- оператор READ N2, L;
- оператор PUNCH N2, L;
- оператор PRINT N2, L;
- оператор READ с параметрами ERR и END;
- операторы ввода и вывода прямого доступа;
- оператор NAMELIST;
- спецификации T, Z, G;
- операторы отладки;
- служебные программы библиотеки.

**2. Средства отладки.** Средства отладки представляют собой набор операторов языка, посредством которых можно проследить последовательность

выполнения операторов программы, вывести на печать значения переменных и массивов, проверить правильность использования индексов и обращений к подпрограммам. Операторы отладки являются составной частью входного языка фортран IV ДОС, но отсутствуют в базисном фортране ДОС.

К операторам отладки относятся неисполняемые операторы DEBUG и AT и исполняемые операторы TRACE ON, TRACE OFF, DISPLAY. Отладочные действия, задаваемые операторами отладки, выполняются для одной программной единицы. Для этого в конце исходной программы (перед оператором END) должен быть помещен оператор DEBUG, за которым могут следовать пакеты отладки. Каждый пакет отладки состоит из оператора начала пакета AT, одного или более исполняемых отладочных операторов, а также, если необходимо, из операторов исходной программы.

Операторы DEBUG, AT, TRACE ON и TRACE OFF рассматривались в § 5 этой главы. Оператор DISPLAY имеет вид

```
DISPLAY L
```

где L — список идентификаторов простых переменных или массивов. Элементы в списке разделяются запятыми.

Оператор DISPLAY предназначен для вывода в файл отладки в формате NAMELIST значений переменных и массивов, указанных в списке. Значения элементов списка выводятся перед выполнением оператора, метка которого указана в операторе начала пакета отладки AT. Таким образом, оператор DISPLAY выполняет те же действия, что и группа операторов

```
NAMELIST /X/ L  
WRITE (N1, X)
```

где X — наименование списка L, N1 — номер файла.

Если оператор DISPLAY используется в подпрограмме типа SUBROUTINE, то в списке оператора DISPLAY не должны употребляться формальные параметры этой подпрограммы. Оператор DISPLAY не должен появляться в логическом операторе IF.

**3. Служебные программы.** Служебные программы, содержащиеся в библиотеке базисного фортрана ДОС и фортрана IV ДОС, используются для выполнения некоторых вспомогательных действий, например таких, как завершение выполнения программы, вывод на печать содержимого участков памяти и т. д. Каждая служебная программа вызывается указанием ее наименования в операторе CALL. Наименованиями служебных программ являются следующие идентификаторы: SLITE, SLITET, DUMP, PDUMP, DVCHK, OVERFL, EXIT.

Описание служебных программ дано в § 5. Отметим лишь, что параметрами KN при обращении к программам DUMP и PDUMP в базисном фортране ДОС могут служить константы 0, 4, 5, 6, а в фортране IV ДОС — все перечисленные в § 5 константы, кроме констант 10 и 11.

**4. Ввод и вывод.** В фортране ДОС ЕС обрабатываются файлы на перфокартах, магнитных лентах, файлы последовательного и прямого доступа на дисках, файлы печати и файлы пишущей машинки. Файлы на магнитных лен-

тах могут быть с метками или без меток. Файлы на дисках всегда должны иметь метки. Не обрабатываются многотомные файлы и файлы, разделенные на участки (многоучастковые файлы), а также многофайловые тома магнитных лент.

В исходной программе обращение к устройствам ввода и вывода производится через номера файлов. В управляющих операторах программы УПРАВЛЕНИЕ ЗАДАНИЯМИ для описания файлов на дисках и на магнитной ленте с метками используются однозначно определенные в ДОС ЕС имена файлов. Между номерами файлов, логическими устройствами и именами файлов установлено определенное соответствие, которое не может изменяться программистом. Программист может не заботиться о соответствии логических и физических устройств, ему следует лишь указывать соответствующий номер файла.

Некоторые номера файлов могут быть использованы только с определенными типами устройств. Так, номер 1 относится к вводу, 2, 3, 15 — к выводу. С номерами 1, 2 связываются устройство ввода с перфокарт, накопитель на магнитной ленте и накопитель на дисках для размещения последовательного файла; с номером 3 — печатающее устройство, накопитель на магнитной ленте и накопитель на дисках для размещения файла последовательного доступа; с номером 15 связывается вывод на печатающее устройство или пишущую машинку. Номера файлов 4—14 могут быть использованы как при вводе, так и при выводе с любыми устройствами (в том числе с дисками с прямым доступом), кроме пишущей машинки.

Операторы управления файлами BACKSPACE, END FILE и REWIND могут использоваться только для файлов на магнитных лентах и последовательных файлов на дисках.

По аналогии с понятиями логического и физического устройства различают логическую и физическую записи. Логическая запись (запись фортрана) — это совокупность данных, связанных между собой с точки зрения программной обработки. Физическая запись — это блок данных, размер которого определяется используемым физическим устройством. В зависимости от вида записи (форматная или бесформатная) и метода доступа к данным (прямой или последовательный) логическая запись может состоять из одной или нескольких физических записей. Размер физической записи в фортране устанавливается в байтах в зависимости от номера файла и от типа физического устройства, назначенного файлу.

Для перфокарточных устройств ввода и вывода всегда (независимо от номера файла) устанавливается размер физической записи 80 байт. По 80 байт отводится под физическую запись для накопителя на магнитной ленте и диске, если файл имеет номер 1 и по 81 байту, если номер файла — 2. Из них первый символ является управляющим. Для файла с номером 3 размер физической записи для всех допустимых устройств равен 121 байту, из них первый символ является управляющим. Максимальный размер физической записи в базисном фортране для файла с номером 15 равен 260 (пишущая машинка) или количеству позиций в строке печати, но не больше 260 для устройства печати.

В случае файлов с номерами 4—14 размер физической записи для печатающего устройства равен количеству позиций в строке печати, но не больше 260, для накопителя на магнитной ленте от 18 (от 15 в базисном фортране) до 260, для накопителя на магнитных дисках с последовательным доступом — 260, для накопителя на дисках с прямым доступом в соответствии с заданием в операторе DEFINE FILE, но не больше 3625 (1726 в базисном фортране ДОС) байт для устройств EC-5052, EC-5055, EC-5056 и 7294 (1726) байт для устройства EC-5061.

**5. Отличия версий фортрана ДОС ЕС.** Как уже отмечалось выше, язык фортран IV ДОС полностью включает в себя средства языка базисный фортран ДОС. Следующие элементы языка фортран IV ДОС отсутствуют в базисном фортране ДОС:

- комплексные, логические и шестнадцатеричные константы;
- использование текстовых констант где-либо, кроме оператора FORMAT;
- типы COMPLEX и LOGICAL в операторах явного описания типа и в операторе FUNCTION;
- размерность массива, превосходящая 3;
- представление индекса элемента массива в виде произвольного арифметического выражения;
- массивы с переменными размерами;
- оператор IMPLICIT;
- указание длины переменных и массивов в операторах явного описания типа и в операторе FUNCTION;
- присвоение начальных значений данным в операторах явного описания типа;
- оператор DATA;
- оператор BLOCK DATA;
- операции отношения;
- логические операции;
- логические выражения;
- логический оператор присваивания;
- логический оператор IF;
- присваиваемый оператор GO TO;
- оператор ASSIGN;
- оператор PAUSE 'ТЕКСТ';
- текстовая константа как фактический параметр при обращении к подпрограммам типа FUNCTION и SUBROUTINE;
- формальные параметры, заключенные и символы /;
- формальный параметр в виде символа \*;
- метка оператора в качестве фактического параметра;
- оператор RETURN I;
- оператор ENTRY;
- именованные общие блоки;
- оператор NAMELIST;
- оператор READ N2, L;
- оператор PRINT N2, L;

- оператор PUNCH N2, L;
- оператор READ с параметрами ERR и END;
- спецификации L, G, Z в операторе FORMAT;
- использование форматов в массиве (переменного формата);
- вложенность групп спецификаций в операторе FORMAT, превосходящая 1;
- операторы отладки;
- расширение форматов вывода в служебных программах DUMP и PDUMP.

В базисном фортране ДОС, кроме перечисленных отличий от фортрана IV ДОС, число стандартных функций (встроенных и внешних) равно 45. Оператор DEFINE FILE должен находиться среди описательных операторов.

**6. Количественные ограничения.** Программы, которые обрабатываются трансляторами с языка фортран в операционной системе ДОС ЕС, должны быть составлены с учетом ограничений, налагаемых на элементы исходной программы. Для транслятора с базисного фортрана ДОС эти ограничения сводятся к следующим.

Максимальное значение, равное 65532 байтам, установлено для размеров рабочей программы, общей области, определенной оператором COMMON, и области данных вне общей области (т.е. области данных, включающих переменные, не входящие в общую область, константы и рабочие поля, построенные транслятором).

Вложенность операторов цикла ограничена 25 операторами DO, а вложенность ссылок к подпрограмме-функции не должна превосходить 15.

Не должно превышать 511 общее число каждого из следующих объектов программы:

- уникальных целых констант;
- уникальных действительных констант;
- уникальных констант с двойной точностью;
- переменных;
- массивов;
- наименований в операторах REAL;
- наименований в операторах INTEGER;
- наименований в операторах DOUBLE PRECISION;
- переменных и массивов в общей области;
- наименований в списках оператора эквивалентности вместе с количеством списков оператора EQUIVALENCE;
- определенных оператор-функций;
- уникальных наименований подпрограмм (имен точек входа в подпрограммы);
- формальных параметров в подпрограмме или оператор-функции;
- параметров во всех подпрограммах и оператор-функциях;
- помеченных операторов (включая одну дополнительную метку на каждый оператор DO или неявный цикл в списке ввода-вывода).

Данные этого параграфа соответствуют версии 2.0 операционной системы ДОС ЕС ЭВМ.

## § 8. Фортран-Дубна и фортран ДОС ЕС

Языки программирования фортран для ЭВМ БЭСМ-6 и фортран IV ДОС ЕС ЭВМ являются довольно близкими версиями языка фортран, тем не менее при переводе программ с одной машины на другую необходимо принимать во внимание имеющиеся различия. В связи с тем, что ЭВМ серии ЕС получают все большее распространение и, следовательно, число их абонентов растет, представляется целесообразным перечислить здесь основные ограничения, налагаемые на язык фортран правилами версии фортран IV ДОС ЕС (которая в дальнейшем называется для краткости фортран ЕС) по сравнению с правилами версии языка фортран для ЭВМ БЭСМ-6 мониторной системы «Дубна» (называемой в дальнейшем фортран-Дубна). Сюда включены также основные различия в интерпретации одних и тех же конструкций языка фортран-Дубна трансляторами мониторной системы «Дубна» и ДОС ЕС и приведены краткие рекомендации по использованию возможностей языка фортран-Дубна для достижения максимальной совместимости с языком фортран ЕС.

Нарушение каждого из указываемых ниже правил при использовании языка фортран ЕС может привести к различным последствиям. В целях удобства как использования материала этого параграфа, так и изложения правил в тексте каждого пункта правил, в квадратных скобках указывается одно (или несколько) условное обозначение в виде символов: Т, С, Р, П, И, \*. Каждый из этих символов означает соответственно следующее:

Т — нарушение данного правила будет обнаружено транслятором ДОС ЕС, т. е. при трансляции и будет выдана диагностика ошибки;

С — нарушение данного правила может быть обнаружено в процессе счета на ЕС ЭВМ;

Р — нарушение данного правила может привести к получению неверных результатов при счете на ЕС ЭВМ;

П — данная особенность представления числовой информации на ЕС ЭВМ может привести к значительному увеличению погрешности в результатах по сравнению с БЭСМ-6;

И — при переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ (и обратно) необходимо внести изменения в программу;

\* — ограничение на язык для краткости сформулировано более жестко по сравнению с правилами версии фортран ЕС.

Если у одного пункта правил указано несколько символов, то это означает, что в данном случае возможен любой из исходов, определяемых каждым из этих условных обозначений.

**1. Основные ограничения языка фортран ЕС.** Правилами языка фортран ЕС запрещается:

— использовать символ \$ (или  $\diamond$ ) в качестве разделителя между операторами в пределах одной строки (перфокарты); на ЕС ЭВМ символ \$ является буквой [Т];

— использовать оператор PROGRAM для указания начала основной программы [Т];

— использовать операторы ENCODE и DECODE [Т];

— использовать восьмеричные константы; вместо них можно использовать шестнадцатеричные константы [T];

— указывать целые константы, превосходящие по модулю  $2^{31-1} \approx 2 \times 10^9$  [T]; на БЭСМ-6 аналогичный предел равен  $2^{40-1} \approx 1 \times 10^{12}$ ;

— указывать константы с двойной точностью, превосходящие по модулю  $16^{63} \approx 7 \times 10^{75}$  [T]; на БЭСМ-6 такой предел равен приблизительно  $1 \times 10^{1000}$ ;

— использовать текстовые константы, состоящие более чем из четырех символов [TC\*];

— использовать идентификаторы, содержащие буквы русского алфавита или состоящие более чем из шести символов [T];

— указывать текстовые константы в операторах присваивания и в логическом операторе IF [T];

— использовать элементы массивов без индексов или с меньшим числом индексов, чем указано в описании соответствующих массивов; исключения допускаются только в операторах EQUIVALENCE и DATA, а также в фактических параметрах при вызове подпрограмм [TCP];

— использовать одинаковые метки у исполняемых операторов и у операторов FORMAT в пределах одной программной единицы [T.]; на БЭСМ-6 в одной и той же подпрограмме допускаются, например, операторы

```
7 A=3.14
```

```
7 FORMAT (F10.3)
```

— применять оператор DATA для передачи данных в элементы непомеченного общего блока, а также в элементы помеченных блоков COMMON, расположенных вне подпрограммы BLOCK DATA [T];

— располагать оператор DATA ранее описательных операторов, содержащих те переменные и массивы, которым присваиваются начальные значения с помощью данного оператора DATA [T]; рекомендуется располагать оператор DATA последним среди других неисполняемых операторов данной программной единицы;

— записывать оператор DATA в форме, использующей скобки в качестве ограничителей списка констант; нельзя, например, написать DATA (X=5.), но можно написать DATA X/5./ [T];

— допускать несовпадение по типу в операторе DATA соответствующих элементов в списке переменных и в списке констант; нельзя, например, написать DATA X/5/, а можно написать только DATA X/5./ [T];

— использовать переменные и массивы типов REAL или INTEGER вместе с величинами любого из других типов в одном блоке COMMON или в одной группе эквивалентности в операторе EQUIVALENCE; например, такая совокупность операторов:

```
COMPLEX C(2)
```

```
COMMON /A/ B, C
```

где B — простая переменная типа REAL\*4, недопустима [T\*];

— использовать помеченные блоки COMMON разной длины в различных подпрограммах, описывающих одни и те же общие блоки [T]; на БЭСМ-6

нарушение этого правила иногда остается без последствий и не приводит к неправильным результатам;

- указывать массивы в качестве фактических параметров, если соответствующие им формальные параметры являются простыми переменными [СИ];

- использовать при вводе по спецификациям I, F, E, D пробелы в поле ввода, за исключением самых левых позиций в каждом поле; на БЭСМ-6 такие пробелы при вводе игнорируются, а на ЕС ЭВМ интерпретируются как нули [СР];

- указывать в операторах FORMAT скобки с глубиной вложения более двух уровней [Т]; на БЭСМ-6 допускается глубина вложения скобок до трех уровней;

- использовать спецификацию преобразования  $Ow$  для восьмеричных чисел: для преобразования шестнадцатеричных чисел используется спецификация  $Zw$  [Т];

- располагать в операторе FORMAT в первой позиции каждой строки какие-либо символы, кроме управляющих, т. е. символов 0, 1 и + [С], смысл которых тот же, что и на БЭСМ-6; остальные символы, используемые на БЭСМ-6 в качестве управляющих, на ЕС ЭВМ не являются управляющими;

- использовать библиотечную подпрограмму  $ASIN(X)$  для вычисления значения арксинуса действительного аргумента [И]; взамен используется стандартная функция  $ARSIN(X)$ .

**2. Основные рекомендации.** При переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ в целях достижения максимальной совместимости с языком фортран ЕС, а также во избежание получения неверных результатов не рекомендуется:

- использовать вычисляемый оператор перехода

GO TO (N1, N2, ..., NK), M

в случае, когда условие  $1 \leq M \leq K$  не будет выполнено; на ЕС ЭВМ произойдет передача управления следующему исполняемому оператору [СР], а на БЭСМ-6 будет неправильная передача управления;

- использовать оператор ENTRY во всех случаях; кроме тех, когда соответствующая подпрограмма является подпрограммой типа SUBROUTINE без параметров; правила языка фортран ЕС требуют, чтобы у каждого оператора ENTRY был указан список его формальных параметров, если таковые имеются [РСИ]; на БЭСМ-6 список формальных параметров у оператора ENTRY не указывается, а предполагается, что список параметров у оператора ENTRY тот же, что и у заголовка подпрограммы, т. е. у операторов SUBROUTINE или FUNCTION;

- записывать выражение вида  $A**B**C$  без скобок, указывающих точный порядок выполнения операций возведения в степень; на ЕС ЭВМ запись  $A**B**C$  понимается как  $A**(B**C)$ , а на БЭСМ-6 — как  $(A**B)**C$  [СР];

- использовать «пустые» циклы в операторах DO, т. е. циклы вида

DO M I=N1, N2, N

при  $N1 > N2$  ( $N > 0$ ); если это условие выполнено, то на ЕС ЭВМ цикл будет выполнен один раз, на БЭСМ-6 — ни разу [СР];

— использовать константы для обозначения логических номеров устройств ввода и вывода [СИ]; вместо констант рекомендуется использовать простые переменные целого типа;

— указывать в операторе DATA текстовые константы, содержащие более четырех символов [СИ].

Перечисленные здесь отличия языка фортран ЕС от языка фортран-Дубна необходимо учитывать не только при переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ, но и при обратном переходе.

**3. Особенности представления числовой информации.** При переходе с БЭСМ-6 на ЕС ЭВМ рекомендуется учитывать следующие особенности представления числовой информации в ЕС ЭВМ:

— точность машинного представления величин действительного типа при использовании машинных слов стандартной длины (4 байта) составляет приблизительно 7 десятичных цифр против 12 десятичных цифр на ЭВМ БЭСМ-6; при переходе на машинные слова двойной длины (8 байт) эта точность увеличивается до 17 десятичных цифр, что, однако, меньше точности машинного представления величин типа DOUBLE PRECISION на БЭСМ-6 (24 десятичные цифры) [СПИ];

— представление действительных величин в ЕС ЭВМ в виде машинных слов двойной длины увеличивает только точность их машинного представления, но не расширяет, в отличие от БЭСМ-6, диапазон чисел, представимых в машине [СП];

— в ЕС ЭВМ арифметические операции над величинами, представимыми машинными словами двойной длины, реализованы, в отличие от БЭСМ-6, не программно, а аппаратно, поэтому при переходе на двойную точность время счета возрастает всего в 1,3—1,5 раза, а не в 8—10 раз, как на БЭСМ-6 [С];

При переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ с учетом возможного использования величин с двойной точностью (длиной 8 байт) взамен действительных рекомендуется:

— не описывать оператором REAL на БЭСМ-6 переменные и массивы действительного типа, которые, возможно, потребуется перевести на двойную точность [С];

— описывать оператором REAL переменные и массивы действительного типа, которые при переходе с БЭСМ-6 на ЕС ЭВМ не предполагается переводить на двойную точность.

При переходе на двойную точность с соблюдением правил, указанных в п. 3, достаточно в исходной программе, написанной на языке фортран-Дубна, добавить оператор языка фортран ЕС

```
IMPLICIT REAL *8 (A—Z)
```

в каждой подпрограмме, где производится переопределение действительного типа на тип с двойной точностью.

Точное соблюдение правил, приведенных в пп. 1 и 2, обеспечивает практически полную совместимость исходных программ на уровне языка фортран. В то же время ввиду отличий в способе машинного представления числовой информации в БЭСМ-6 и ЕС ЭВМ, это не обеспечивает, вообще говоря,

достаточно точного совпадения результатов счета, полученных на этих машинах. Дело в том, что, как это следует, в частности, из сказанного в п. 3, стандартное представление действительных величин в машинах серии ЕС ЭВМ в виде машинных слов длиной 4 байта, из которых для представления мантиссы отводится 3 байта, довольно часто не обеспечивает удовлетворительной точности результатов. В таких случаях и возникает необходимость представления действительных величин в ЕС ЭВМ с двойной точностью, т.е. в виде машинных слов, состоящих из 8 байт, из которых под мантиссу отводится 7 байт. Это требует, с одной стороны, введения дополнительных ограничений при составлении программ, совместимых для БЭСМ-6 и ЕС ЭВМ, и, с другой стороны, введения дополнительных описательных операторов языка фортран ЕС при переходе с БЭСМ-6 на ЕС ЭВМ. Эти требования будут удовлетворены, если соблюдать рекомендации, перечисленные в п. 3.

### § 9. Стандарт фортран 77

Первоначально язык фортран разрабатывался для численных приложений, и стандарт 1966 г. отражал запросы соответствующего класса пользователей. Программа на фортране, разработанная для некоторой вычислительной системы, если она удовлетворяла стандарту, могла без всяких изменений использоваться и на другой вычислительной системе. Однако сфера применения фортрана расширялась, охватывая не только область сугубо численных приложений, но и такие, например, области, как обработка текстов и работа с файлами. Происходила модификация языка, в него включались более мощные средства обработки данных, появлялись все новые и новые версии (диалекты) фортрана. Отличия диалектов от стандарта касаются главным образом расширения возможностей языка и включения дополнительных элементов. В результате увеличилась область потенциальных приложений фортрана, но за счет ухудшения переносимости написанных на фортране программ. Возникла необходимость создания нового стандарта.

Проект нового стандарта был опубликован в 1976 г., а в 1978 г. была принята окончательная версия нового стандарта для фортрана, получившего название фортран 77. Новый стандарт не исключает использования старых фортранных программ, он лишь расширяет возможности языка при вводе и выводе при описании данных, описании подпрограмм и конструкций, в которых ранее допускались только значения целого типа, а также включает ряд изменений разнообразного характера, в том числе синтаксические и семантические усовершенствования.

**1. Расширения по сравнению с предыдущим стандартом.** В алфавит добавлен символ : (двоеточие). Отсутствуют зарезервированные слова, поэтому идентификатор может по написанию полностью совпадать с основным символом (например, ключевым словом); смысл последовательности литер определяется по контексту.

Введены переменные текстового типа, принимающие в качестве значений текстовые константы в  $\cdot$ -представлении. Текстовая константа в  $\cdot$ -представлении (холлеритовская константа) может встречаться только в операторе DATA,

в списке параметров оператора CALL и в операторе FORMAT. Текстовый тип описывается оператором

```
CHARACTER *S X *S1, Y *S2, ...
```

где S, S1, ... — описатели длины, X, Y, ... — идентификаторы простых переменных или описатели массива, S относится ко всем элементам списка X, Y, ..., у которых нет собственного описателя длины, а S1, S2, ... — только к предшествующему элементу X, Y, ... . Описатель длины может быть целой константой без знака, целым выражением из констант в скобках, имеющим положительное значение, или звездочкой в скобках. Если описатель длины отсутствует, длина принимается равной единице. Описатель длины \* означает, что длина должна быть определена иным образом (например, при вызове подпрограммы). Так, в операторах

```
CHARACTER *4 L, K(10) *8, C *(*) , M
```

```
CHARACTER A, B
```

```
CHARACTER *(*) E, P*1, D
```

переменные L, M имеют длину 4 символа, A, B, P — длину 1, элементы массива K — длину 8, а для переменных C, E, D длина не определена. Длина текстовой переменной может быть вычислена с помощью стандартной функции LEN. Так, значение LEN (M) есть 4

Может быть образована текстовая подцепочка (подстрока) — последовательность расположенных подряд литер, входящих в состав некоторого элемента данных текстового типа. Подстрока имеет также текстовый тип. Значение подстроки образуется указанием после идентификатора текстовой переменной в скобках с разделителем : (двоеточие) порядковых номеров первого и последнего символа из тех литер, которые должны быть включены в подстроку. Например, если текстовая переменная Z имеет значение 'FORTRAN', то величина Z(1:3) является подстрокой со значением 'FOR', а Z(4:6) имеет значение 'TRA'. Аналогично образуется подстрока в случае переменной с индексами. Так, значением T(I,J)(K:L) является подстрока от K-го до L-го символа элемента T(I,J) массива текстовых значений.

Для текстовых величин определена операция, называемая конкатенацией (сцеплением) и обозначаемая как //. Используется эта операция при построении текстовых выражений — выражений, значением которых являются константы текстового типа. Например, если A и B являются текстовыми переменными со значениями '+' и '123' соответственно, то выражение A//B//'X' имеет значение '+123X'. Операция сцепления выполняется после арифметических операций перед операциями отношения.

Операндами в операции отношения могут быть текстовые выражения. В таком случае сравниваются двоичные коды текстовых операндов, а результатом операции является «истина» или «ложь» и зависит результат от упорядоченности кодов литер по возрастанию. Если текстовые операнды имеют разную длину, более короткий операнд при сравнении дополняется справа пробелами до длины более длинного.

Определен оператор присваивания для текстовых переменных. При этом слева от знака присваивания может стоять текстовая переменная, элемент

текстового массива или наименование подстроки, справа — текстовое выражение. При вычислении текстового выражения и присваивании его значения текстовой переменной никакая из изменяющихся литерных позиций в переменной не должна использоваться при вычислении выражения. Если длина выражения меньше длины переменной, значение его дополняется справа пробелами до длины переменной. Если длина выражения больше длины переменной, значение выражения при присваивании укорачивается справа. Присваивание подстроке определяет только те литерные позиции, которым делается присваивание. Например, после выполнения операторов

```
CHARACTER P *3, R *6, S *7
P='ABC'
R=P//DEF
S(3:6)=R(2:5)
```

переменная R имеет значение 'ABCDEF', а подстрока S(3:6) — значение 'BCDE'

Определены логические операции .EQV. — эквивалентность и .NEQV. — неэквивалентность. Результатом выражения L1.EQV.L2, где L1, L2 — логические величины, будет значение .TRUE. если L1 и L2 имеют одинаковые значения и .FALSE. — в противном случае. Результат выражения L1.NEQV.L2 имеет противоположное по сравнению с выражением L1.EQV.L2 значение.

Основная программа начинается с оператора PROGRAM A, где A — идентификатор (имя основной программы).

Определен оператор неявного описания типа по первой букве с ключевым словом IMPLICIT.

Допускается размерность массивов до семи. Нижняя граница изменения индекса у переменной с индексами не обязательно равна 1, и в качестве индекса массива допускаются отрицательные и нулевые целые константы, переменные и выражения, а диапазон изменения индекса при описании массива можно задавать граничной парой вида I:J, где I, J — соответственно нижний и верхний пределы изменения индекса. Например, оператор

```
INTEGER A(0:99), B(-100:0, -10:10, 101)
```

описывает одномерный массив A, индекс которого изменяется от 0 до 99 и трехмерный массив B, первый индекс которого изменяется от -100 до 0, второй — от -10 до 10, третий — от 1 до 101

Введен оператор SAVE, действие которого заключается в том, что перечисленные в его списке переменные, массивы или общие блоки не окажутся неопределенными после выполнения в подпрограмме операторов RETURN или END. Например:

```
SAVE /Q/, X,M
```

где Q — имя общего блока, X — переменная, M — идентификатор массива. Таким образом значения переменных и массивов от одного обращения к подпрограмме до следующего могут быть сохранены, если они заданы:

— в операторе SAVE;

— в непомеченном общем блоке, который, по существу, связан с основной программой;

— в помеченном общем блоке, который описан и в подпрограмме и в вызывающей программной единице.

В операторе DATA разрешено использование идентификаторов массивов, текстовых переменных, подстрок и неявных циклов. Введено описание символических констант оператором

```
PARAMETER (P1=K1, P2=K2, ...)
```

где P1, P2, ... — идентификаторы (параметры), K1, K2, ... — выражения из констант. Каждая пара величин (P1 и K1, P2 и K2 и т. д.) должна быть согласована по типу. Текстовые параметры с длиной, отличной от 1, нуждаются в описателе длины в операторе IMPLICIT или CHARACTER. Например:

```
LOGICAL L
```

```
CHARACTER N *(*), M *3
```

```
PARAMETER (L=.TRUE., K=105, M='CDC', N='P1')
```

Параметр и константа не являются взаимозаменяемыми. Параметр можно использовать только там, где это явно разрешено, например в качестве операнда в выражении, или в операторе DATA. Нельзя параметр использовать, например, для формирования комплексной константы и для обозначения текстовой константы в операторе FORMAT.

Целую переменную, которой присвоена метка оператором присваивания метки, можно использовать не только в присваиваемом операторе перехода, но и для задания формата в операторах ввода и вывода.

Допускаются пустые строки, которые обрабатываются как комментарии, т.е. игнорируются. В качестве символа для указания комментария, кроме буквы C, можно использовать символ \*; при этом строка комментария может содержать произвольную последовательность литер в позициях со второй по 72.

В вычисляемом операторе перехода GO TO (N), M на месте M может стоять любое целое выражение. Если значение выражения больше количества элементов списка меток или если это значение не положительно, выполнение программы продолжается с оператора, который следует за вычисляемым оператором GO TO. В присваиваемом операторе перехода GO TO M, (N) список меток может отсутствовать.

Условные операторы управления дополнены конструкцией IF — THEN — ELSE, позволяющей задавать условное выполнение групп операторов. Рассматриваемая конструкция имеет следующую структуру:

```
IF (L) THEN
```

```
<группа IF>
```

```
ELSE IF (L1) THEN
```

```
<группа ELSE IF>
```

```
.....
```

```
<другие группы ELSE IF>
```

```
.....
```

```
ELSE
<группа ELSE>
END IF
```

в которой могут быть опущены группа IF, группа ELSE или группа ELSE IF, представляющие собой один или несколько операторов. В этой записи L, L1 — логические выражения, IF (L) THEN, ELSE IF (L1) THEN, ELSE и END IF — операторы. Для каждого оператора IF должен существовать свой оператор END IF.

Входящие в данную конструкцию группы операторов, как следует из ее структуры, могут быть вложенными, причем вложение должно быть полным, т. е. если одна группа IF содержится внутри другой группы IF, то все операторы первой из них (вложенной) должны находиться между операторами IF (L) THEN и END IF второй (внешней) группы IF. Для каждого из входящих в IF — THEN — ELSE операторов определяется уровень оператора  $l$  как разность  $m - n$ , где  $m$  — число операторов IF (L) THEN от начала конструкции до  $l$  включительно, а  $n$  — число операторов END IF в конструкции до  $l$ , но не включая  $l$ .

В операторе IF (L) THEN (или ELSE IF (L) THEN) вычисляется значение логического выражения L и, если оно истинно, выполняется ближайшая группа IF (или соответственно группа ELSE IF) и управление передается на оператор END IF того же уровня, что и у оператора IF (или соответственно ELSE IF), если, конечно, внутри группы IF (или группы ELSE IF) нет операторов передачи управления. Если L — ложно, выполнение операторов продолжается со следующего оператора ELSE IF, ELSE или END IF с тем же уровнем, что и у оператора IF (или ELSE IF). Управление не может быть передано извне в группу IF (или группу ELSE IF).

Оператор ELSE обеспечивает выполнение группы ELSE вплоть до оператора END IF. Передача управления внутрь группы ELSE извне запрещена. Оператор END IF отмечает место в программе и не выполняет никаких действий. На оператор END IF можно передать управление (при наличии метки), а на операторы ELSE IF и ELSE нельзя. Операторы IF, ELSE IF, ELSE, END IF не могут выступать в качестве последнего оператора в области цикла.

Пример конструкции IF — THEN — ELSE:

```
IF (L) THEN
    I=I1
    J=J1
ELSE IF (M) THEN
    A=A1
    B=B1
ELSE
    X=X1
    Y=Y1
END IF
```

В операторе цикла компоненты заголовка цикла: параметр цикла I, нижняя N1 и верхняя N2 границы и шаг N изменения параметра могут быть

выражениями целого или действительного типа или с двойной точностью. Сказанное относится и к неявным циклам. Если типы компонент заголовка цикла не согласованы, выполняются преобразования к типу параметра цикла. Выполнение цикла начинается с проверки числа повторений цикла, определяемого как значение  $\text{MAX}(\text{INT}((N2-N1+N)/N), 0)$ , и если оно равно нулю, цикл не выполняется ни разу. Можно изменить значения компонентов  $N1$ ,  $N2$ ,  $N$  во время выполнения цикла, однако это не повлияет на количество повторений цикла, так как оно вычисляется прежде, чем цикл выполнится в первый раз. При выходе из цикла параметр цикла сохраняет последнее присвоенное ему значение.

В записи операторов STOP и PAUSE после ключевого слова может быть указана целая константа не более чем из пяти цифр или текстовая константа.

Стандартные функции подразделяются на встроенные и внешние. При обращении к встроенной функции генерируется последовательность команд, непосредственно реализующая эту функцию, обращение к внешней функции есть обращение к библиотечной подпрограмме. Все стандартные функции, соответствующие одной и той же математической функции (процедуре), имеют общее имя, называемое универсальным именем, которым можно пользоваться независимо от типа аргумента (аргументов) функции. Обращение к соответствующей специфической функции (с конкретным типом аргументов и значения) обеспечивается автоматически. Например, ко всем специфическим функциям ABS(X), IABS(K), DABS(D), CABS(C) можно обратиться по универсальному имени ABS. Общее число стандартных функций — 81.

Если при обращении к внешней процедуре (т. е. к подпрограмме-функции или к подпрограмме) фактическим параметром является стандартная функция, то следует использовать специфическое имя функции, описав его в операторе INTRINSIC в той программной единице, где находится обращение к внешней процедуре. Например:

```
INTRINSIC SIN
CALL FS(A, M, N, SIN)
```

Как и в случае оператора EXTERNAL, оператор INTRINSIC не позволяет ввести переменную вместо функции, заданной в качестве фактического параметра в вызывающей программной единице. Но если же в операторе EXTERNAL встретилось имя стандартной функции, то оно в данной программной единице не будет рассматриваться как имя стандартной функции. Никакое имя в пределах одной программной единицы не может встречаться одновременно в операторах INTRINSIC и EXTERNAL. Наименования встроенных функций преобразования типов (INT, IFIX, IDINT, FLOAT, SNGL, REAL, DBLE, CMPLX, ICHAR, CHAR), а также функций для вычисления наибольшего и наименьшего значения (MAX, MAX0, AMAX1, DMAX1, AMAX0, MAX1, MIN, MIN0, AMIN1, DMIN1, AMIN0, MIN1) нельзя упоминать в операторе INTRINSIC.

Допускаются описания оператор-функций и подпрограмм-функций без параметров, но с сохранением скобок после идентификатора функции, например:

```
CHARACTER *4 FUNCTION C( )
```

В качестве формального параметра в подпрограмме допускается символ \*, которому соответствует фактический параметр в виде \*M, где M — метка оператора в вызывающей программе. Оператор возврата в этом случае имеет вид RETURN I, где I — целое выражение, и обеспечивает несколько точек выхода из подпрограммы. Для подпрограммы без параметров допустимы две формы записи: SUBROUTINE S и SUBROUTINE S( ). Может быть указано несколько точек входа во внешние процедуры оператором ENTRY.

В операторе DATA не требуется совпадения типа переменной и типа соответствующего ей значения, преобразование будет происходить так, как если бы переменная и значение участвовали в операторе присваивания. Начальные значения могут быть присвоены не всем элементам массива, а лишь части его. Для этого в списке переменных указывается конструкция вида M(I1) : M(I2), где M — идентификатор массива, I1, I2 — индексы. Такая конструкция означает, что значения должны быть присвоены всем элементам массива M, начиная с элемента с индексом I1 и заканчивая элементом с индексом I2.

Допускается описание нескольких подпрограмм данных. В этом случае после оператора BLOCK DATA указывается имя, например:

```
BLOCK DATA LIST1
```

Имя подпрограммы данных может быть указано в списке оператора EXTERNAL.

В операторе эквивалентности разрешено использование идентификаторов массивов и идентификаторов текстовых переменных, текстовых массивов, элементов текстовых массивов, имен подстрок. Если в операторе EQUIVALENCE указано имя массива, соответствие устанавливается точно так же, как если бы был задан его первый элемент. Текстовые величины можно совместить при помощи эквивалентности только с другими текстовыми величинами. Совмещаются первые байты памяти, занятые объектами, входящими в список эквивалентности оператора EQUIVALENCE, что приводит также к совмещению других текстовых элементов. Например, операторы

```
CHARACTER A *8, B(3) *2, C *3
EQUIVALENCE (A(2:7), B(1)), (B(2), C)
```

расположат переменные в памяти следующим образом:

```
Байты :      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
Переменная A:  ──────────── A ────────────|
Массив B:      ──B(1)─┬─B(2)─┬─B(3)─┬─|
Переменная C:      ─── C ───|
```

Не разрешается для определения эквивалентности задавать *n*-мерный массив как одномерный. В стандарте фортрана 1966 г. это допускалось.

Ввод и вывод информации определен как для файлов последовательного, так и прямого доступа. Определена также внутренняя передача данных (из одной области оперативной памяти в другую) как передача внутренних фай-

лов-областей памяти, описанных как текстовая переменная, текстовый массив или подстрока. Передача данных с внешних носителей — передача внешних файлов осуществляется через канал, который задается в виде целого положительного или нулевого значения.

Имеется две формы оператора ввода READ; вывод осуществляется операторами WRITE и PRINT:

```

READ (N) L      WRITE (N) L
READ N2, L     PRINT N2, L

```

где N — список управляющей информации, L — список ввода-вывода, N2 — формат (элемент списка N). Элементами N являются: UNIT=N1 — канал, FMT=N2 — формат, REC=R — номер записи, IOSTAT=IS — код ответа, END=N3 — возврат по концу файла, ERR=N4 — возврат по ошибке. Первые два элемента называются позиционными параметрами, их место в списке строго определено, остальные параметры являются ключевыми и могут быть расположены в произвольном порядке, их присутствие в списке N обязательно. Не обязательно также указывать конструкции UNIT= и FMT=; N1 может быть целым неотрицательным выражением (номер внешнего канала), \* (внешний системный канал, предварительно подсоединенный для ввода-вывода), текстовой величиной (идентификатор внутреннего файла); N2 — это либо метка оператора FORMAT, либо целая переменная, которой присвоено значение метки оператора FORMAT, либо имя текстового массива, либо текстовое выражение определенного вида, либо символ \*, означающая свободный формат (свободный формат подразумевается и в том случае, если отсутствует элемент FMT=N2); R — целое положительное выражение, указывающее номер записи при вводе-выводе с прямым доступом (отсутствие элемента REC=R подразумевает последовательный доступ); IS — целая переменная или элемент целого массива, получает значение в конце операции ввода-вывода: 0 — в случае отсутствия ошибки, положительное значение, если произошла ошибка, отрицательное значение, если встретилась запись конца файла; N3, N4 — метки операторов, куда передается управление в случаях встречи конца файла или ошибки соответственно.

Операторы READ N2, L и PRINT N2, L реализуют возможность обмена данными со стандартными системными устройствами без явного указания номера канала.

В списке ввода-вывода могут содержаться текстовые величины. Свободный формат означает, что данные вводятся так, как они представлены на внешнем носителе. Значения данных на внешнем носителе отделяются друг от друга запятой или пробелами и должны иметь вид, допустимый для ввода в языке фортран. Можно использовать коэффициент повторения в виде K\*, где K — целая константа без знака (не нуль). При выводе элементов списка вывода записываются в одну или несколько последовательных записей в некотором подходящем формате, который зависит от реализации. Выводимые значения разделяются запятыми или пробелами.

Установление связи между каналом и файлом (подсоединение) выполняется оператором открытия файла OPEN. Канал может быть отсоединен оператором закрытия файла CLOSE.

Оператор открытия файла имеет вид

**OPEN (ON)**

где ON — список параметров, элементами которого являются: UNIT=N1 — канал, IOSTAT=IS — код ответа, ERR=N4 — возврат по ошибке, FILE=F — файл, STATUS=S — диспозиция, ACCESS=A — доступ, FORM=FM — форматность, RECL=RL — длина записи, BLANK=B — пробел. Параметр UNIT=N1 — обязательный, конструкция UNIT= может быть опущена, остальные параметры могут отсутствовать. Смысл обозначений N1, IS, N4 приведен выше, F — текстовое выражение, задающее имя файла, который следует подсоединить, S — текстовое выражение с одним из значений OLD (старый), NEW (новый), SCRATCH (временный), UNKNOWN (неизвестный), характеризующий файл, A — значения DIRECT (прямой) или SEQUENTIAL (последовательный), по умолчанию принимается SEQUENTIAL, FM — значения FORMATTED (форматный) или UNFORMATTED (бесформатный), по умолчанию принимается FORMATTED для файлов последовательного доступа и UNFORMATTED — для файлов прямого доступа, RL — целое положительное выражение, обозначающее длину записи в байтах для форматного файла и в единицах, зависящих от транслятора, для бесформатного файла, имеет смысл лишь для файлов прямого доступа, B — текстовые значения либо NULL (пусто), либо ZERO (нуль), по умолчанию выбирается NULL, значение NULL означает, что при вводе числовых значений пробелы надо игнорировать, значение ZERO — все пробелы, кроме начальных, нужно воспринимать как нули.

Оператор закрытия файла имеет вид

**CLOSE (CN)**

где CN — список параметров, элементами которого являются: UNIT=N1 — канал, IOSTAT=IS — код ответа, ERR=N4 — возврат по ошибке, STATUS=S — диспозиция. Параметр UNIT=N1 обязательный, конструкция UNIT= может быть опущена, остальные параметры могут отсутствовать. Смысл обозначений N1, IS, N4 приведен выше, S — текстовое значение с одним из выражений KEEP (сохранить), DELETE (уничтожить). По умолчанию для файлов, которые были открыты как SCRATCH, принимается значение DELETE, для всех остальных — KEEP.

Для получения информации о свойствах и текущем состоянии файла или канала служит оператор

**INQUIRE (IN)**

где IN — список параметров, в который входит либо параметр UNIT=N1 (конструкция UNIT= может быть опущена), если оператор относится к каналу с номером N1, либо параметр FILE=F, если необходимо получить информацию о файле с именем F. Остальные параметры следующие: IOSTAT=IS — код ответа, ERR=N4 — возврат по ошибке, EXIST=EX — существует?, OPENED=OP — открыт?, NUMBER=NU — номер канала, NAMED=NA — с именем?, NAME=NE — имя, ACCESS=A — доступ, SEQUENTIAL=SE — последовательный?, DIRECT=DI — прямой?, FORM=FM — форматность, FORMATTED=FO — форматный?, UNFORMATTED=UF — бесформатный?,

**RECL=RL** — длина записи, **NEXTREC=NR** — следующая запись, **BLANK=B** — пробел. В зависимости от того, какими свойствами обладает файл или канал, входящие в параметры выражения (справа от символа **=**), получают соответствующие значения. Каждый из параметров может входить в оператор **INQUIRE** не более одного раза.

Операторы управления файлами **REWIND**, **BACKSPACE**, **END FILE** могут содержать в качестве списка либо номер канала **N1**, либо конструкцию (**UNIT=N1**, **IOSTAT=IS**, **ERR=N4**), где смысл входящих параметров был пояснен выше, в частности, **N1** может быть целым выражением (конструкция **UNIT=** может быть опущена).

В список спецификаций оператора **FORMAT** введены дополнительные спецификации, управляющие табуляцией (**T**, **TL**, **TR**), выводом знака (**S**, **SP**, **SS**) и вводом пробелов (**BN**, **BZ**).

**T**-спецификация имеет вид **Tw**, где **w** — целая ненулевая константа без знака, и указывает, что (при выводе) следующий символ будет выведен в позицию номер **w** записи или (при вводе) следующий символ будет взят на позиции номер **w** записи. **TL**- и **TR**-спецификации имеют вид **TLw** и **TRw** и указывают, что текущую позицию во внешней записи данных следует переместить на **w** позиций назад (т. е. влево) или вперед (т. е. вправо) соответственно.

Спецификации **S**, **SP**, **SS** управляют выводом знака плюс для чисел и влияют на действие спецификаций преобразования **I**, **F**, **E**, **D** и **G**. Действие спецификации **S**, **SP** или **SS** продолжается до тех пор, пока не встретится другая спецификация управления выводом знака. При вводе эти спецификации игнорируются. **S**-спецификация указывает, что следует вернуться к стандартному режиму проставления знака плюс, **SP** — спецификация обеспечивает вывод знака плюс везде, где он возможен, а **SS** — спецификация указывает, что везде, где возможно, знак плюс следует опускать.

Спецификации управления вводом пробелов указывают, как следует при вводе чисел интерпретировать пробелы, кроме старших пробелов. Режим ввода пробелов устанавливается оператором **OPEN** или по умолчанию. Спецификации **BN** и **BZ**, если они встретятся в операторе **FORMAT**, изменяют предыдущий режим. **BN**-спецификация указывает, что пробелы следует игнорировать, а остальные символы прижимать к правому краю поля, как если бы все опущенные пробелы располагались с левого края поля. Поле, состоящее из одних пробелов, соответствует нулевому значению. **BZ** — спецификация обеспечивает интерпретацию пробелов нулями.

Допускается вариант спецификации **I** вида **Iw.d**. При вводе спецификация **Iw.d** действует так же, как **Iw**. При выводе отличие от **Iw** в том, что выводится не менее **d** цифр — при этом могут появиться незначащие нули в начале числа. Если выводимое значение равно нулю и **d = 0**, то поле вывода будет состоять целиком из пробелов.

Имеется вариант **E**-спецификации вида **Ew.dEe**. При вводе редактирование для этой спецификации выполняется так же, как и для **Ew.d**. При выводе отличие в том, что порядок числа будет содержать **e** цифр. Вариант **G**-спецификации вида **Gw.dEe** при вводе эквивалентен **Gw.d**. При выводе **Gw.dEe**

эквивалентна Fw.d, если  $0.1 \leq K < 10^4$ , и Ew.dEe, если  $K < 0.1$  или  $K \geq 10^4$ . Здесь  $K$  — абсолютная величина элемента данных.

В качестве самостоятельной спецификации редактирования или в качестве разделителя в списке форматов можно использовать символ **‡** (двоеточие). Если двоеточие встретилось при выводе, когда в списке вывода не осталось ни одного элемента, то завершается выполнение операции вывода. Если же двоеточие встретилось при вводе или при выполнении операции вывода в списке вывода еще остались элементы, двоеточие игнорируется.

**2. Ограничения по сравнению с фортраном IV.** Некоторые элементы и конструкции языка фортран IV не вошли в новый стандарт фортран 77. К ним относятся:

- символ **&**; символ **§** относится к специальным знакам;
- указатель длины **\*S** в операторах REAL, INTEGER, LOGICAL, COMPLEX, IMPLICIT, FUNCTION; для указания двойной точности действительных величин используется оператор DOUBLE PRECISION;
- присваивание начальных значений при описании данных в операторах REAL, INTEGER, LOGICAL, COMPLEX;
- стандартные функции COTAN(X), DCOTAN(D), HFIX(X), REAL(C), CLOG10(C), ERF(X), DERF(D), ERFC(X), DERFC(D), GAMMA(X), DGAMMA(D), ALGAMA(X), DLGAMA(D), DCMLPX(D1, D2), DCONJG(CD), CDSIN(CD), CDCOS(CD), CDSQRT(CD), CDEXP(CD), CDLOG(CD), CDLG10(CD), CDABS(CD);
- операторы DEFINE FILE, NAMELIST, FIND, PUNCH; различен способ записи операторов ввода-вывода файлов с прямым доступом;
- Z-спецификация формата; в списке форматов после X- и H-спецификаций требуется разделитель.

**3. Отличия фортрана IV.** Следующие элементы языка фортран 77 отсутствуют в фортране IV:

- символ алфавита **:** (двоеточие);
- оператор CHARACTER;
- понятие подстроки;
- операция конкатенации **//**;
- оператор PARAMETER;
- оператор присваивания для текстовых переменных и выражений;
- логические операции .EQV. и .NEQV. ;
- оператор PROGRAM;
- отрицательный и нулевой индексы у элементов массивов; граничная пара при определении границ индексов;
- оператор SAVE;
- переменная из оператора ASSIGN для указания оператора FORMAT;
- пустая строка и символ **\*** в первой позиции строки для указания комментария;
- целое выражение **M** в вычисляемом операторе передачи управления GO TO (N), M;
- отсутствие списка (N) в присваиваемом операторе GO TO;
- конструкция IF — THEN — ELSE;

- выражения целого и действительного типов и с двойной точностью в качестве компонент заголовка цикла;
- выполнение цикла 0 раз, если  $N1 > N2$ ,  $N > 0$  (или  $N1 < N2$ ,  $N < 0$ );
- сохранение параметром цикла последнего присвоенного ему значения при любом выходе из цикла;
- стандартные функции ICHAR(T), CHAR(K), DINT(D), ANINT(X), DNINT(D), NINT(X), IDNINT(D), DDIM(D1, D2), DPROD(X1, X2), LEN(T), INDEX(T1, T2), LGE(T1, T2), LGT(T1, T2), LLE(T1, T2);
- универсальные имена;
- оператор INTRINSIC;
- оператор-функции и подпрограммы-функции без параметров; начальная строка подпрограммы-функции без параметров имеет вид ТИП FUNCTION F ( );
- оператор возврата RETURN I, где I — целое выражение;
- отсутствие требования совпадения типа переменной и типа соответствующего ей значения в операторе DATA; возможность присвоения начальных значений части массива;
- наличие имени у подпрограммы данных;
- имя массива и текстовая величина в группе эквивалентности оператора EQUIVALENCE;
- внутренняя передача данных;
- операторы OPEN, CLOSE, INQUIRE; различен способ записи операторов ввода-вывода файлов с прямым доступом;
- целые выражения в операторах REWIND, BACKSPACE, ENDFILE;
- спецификации формата TL, TR, S, SP, SS, BN, BZ, атрибуты .d в спецификации I и Ee в E- и G-спецификациях;
- двоеточие в качестве разделителя в списке форматов.

## § 10. Подмножество ПЛ/1 ОС ЕС ЭВМ

Версия языка ПЛ/1 для операционной системы ОС ЕС (ПЛ/1 ОС ЕС) представляет собой подмножество полного языка ПЛ/1. Транслятор с этой версии содержит оптимизирующий режим и создает рабочие программы, немного уступающие по своим качествам программам, составленным на языке ассемблера. В обычно реализуемое подмножество не входит целый ряд элементов полного языка ПЛ/1, описание которого с незначительными сокращениями приведено в гл. IV. Ниже будут перечислены основные из не включенных в подмножество ПЛ/1 ОС ЕС элементов языка. К ним относятся:

- данные комплексного типа, выражения над комплексными данными и встроенные комплексные функции;
- выражения над массивами и структурами;
- возможность нахождения оператора DECLARE во внутренних процедурах и блоках;
- возможность нахождения в операторе ON не только единичного оператора, но и целого блока;
- дополнительные точки входа в процедуру, задаваемые оператором ENTRY;

- рекурсивное обращение к процедурам;
- описатель управления памятью **STATIC**;
- описатель управления памятью **CONTROLLED**;
- переменные типа области;
- ввод и вывод, управляемые данными;
- обработка записей в буфере операторами **READ** с дополнением **INTO** и оператором **WRITE** с дополнением **FROM**;
- обработка записей в буфере файла с помощью наложения на запись соответствующей базированной переменной с использованием оператора **READ** с дополнением **SET** и оператора **LOCATE**;
- файлы с описателями **REGIONAL (2)** и **REGIONAL (3)**;
- телеобработка файлов, которые объявляются с описателем **TRANSIENT** и являются доступными с абонентских пунктов при использовании **TSCAM** (общего телекоммуникационного метода доступа);
- параллельное выполнение нескольких задач, а также использование описателей **EVENT**, **TASK**, **PRIORITY** и оператора **WAIT**, позволяющих организовать параллельное выполнение процедур-подпрограмм и синхронизацию выполнения операций ввода-вывода и ряда других операций;
- использование средств препроцессора, позволяющее вносить изменения в текст программы непосредственно перед компиляцией, например возможность копировать хранящиеся в библиотеке фрагменты программы (объявления данных, тексты внутренних процедур и т. п.), изменять и добавлять части отдельных операторов.

Перечисленные элементы не являются запрещенными для транслятора с языка ПЛ/1, ограничения имеют лишь некоторые реализации подмножества ПЛ/1 ОС ЕС. Транслятор имеет определенные количественные ограничения, среди которых выделим следующие:

- размер одного оператора, кроме **DECLARE**, ограничен 3500 символами, что эквивалентно 50 перфокартам;
- максимальное число переменных в исходной программе зависит от полного размера словаря программы, который ограничен приблизительно 65 000 байт; это эквивалентно ограничению приблизительно 1200 переменными для пользователя;
- количество параметров процедур должно быть не более 64;
- максимальная глубина уровня при объявлении структур — 63;
- размер блока должен быть не более 32760 байт;
- число блоков **PROCEDURE**, **BEGIN**, групп **DO** и операторов **ON** в сумме должно быть не более 255;
- в любой точке компиляции не должно быть больше 50 уровней гнездования; степень гнездования в данной точке определяется как число операторов **PROCEDURE**, **BEGIN** или **DO** без соответствующих операторов **END**, плюс число текущеактивных составных операторов **IF**, плюс число текущеактивных левых скобок, плюс число размерностей в каждом выражении над массивами, плюс максимальное число измерений в каждом активном выражении над структурами;
- максимальное число элементов, допускаемых в списке оператора ввода, управляемого данными, — 320;

— выдаваемое оператором DISPLAY сообщение имеет длину 72 символа, ответ — 127 символов;

— максимальный размер записи не должен превышать 32760 байт.

При трансляции исходной программы могут быть выданы четыре типа диагностических сообщений: предупреждение, ошибка, серьезная ошибка, ошибка окончания, которые записываются вслед за листингом исходной программы и всеми другими листингами.

## § 11. ПЛ/1 ДОС ЕС

Реализованный в операционной системе ДОС ЕС язык ПЛ/1 представляет собой подмножество полного языка ПЛ/1, в котором отсутствуют такие элементы полного языка как синхронное выполнение ветвей программы, предтрансляторная обработка исходного текста, передача потоком, управляемая данными, рекурсивные обращения к процедурам, возможность обработки комплексных данных, присваивания для структур с дополнением BY NAME и ряд других особенностей.

Отсутствует региональная организация файла REGIONAL(2). В то же время на магнитной ленте и магнитных дисках допускаются многофайловые тома и многотомные файлы. Описатель ENVIRONMENT(D) должен присутствовать в каждом объявлении файла. В списке дополнений D этого описателя может быть указано большое количество режимов, описывающих те свойства и признаки файлов, которые связаны с особенностями ДОС ЕС.

На уровне входного языка в исходной программе можно выполнить ряд отладочных операций, среди которых укажем печать значений переменных, печать изменяющихся значений и печать переходов.

Печать значений переменных выполняется с помощью подпрограммы DYNDUMP, для вызова которой используется оператор

```
CALL DYNDUMP(FA);
```

где FA — список аргументов. Аргумент может быть скалярным выражением, именем массива или структуры. Значения, выдаваемые на печать, представляют собой шестнадцатеричную форму внутреннего представления указанных аргументов.

Печать изменяющихся значений выполняется после обращения к подпрограмме IJKEXHC оператором

```
CALL IJKEXHC (FA);
```

где элемент списка аргументов FA может быть скалярной переменной, именем массива или структуры, а также строковой или арифметической константой.

Имена аргументов в описателе AUTOMATIC печатаются всегда при первом выполнении оператора CALL IJKEXHC; в блоке. Идентификаторы с описателем STATIC печатаются только тогда, когда оператор печати изменяющихся значений выполняется в первый раз, если данный блок является внутренним. Если блок определения имен аргументов внешний, они печатаются каждый раз при выполнении этого оператора. При последующих выполнениях оператора CALL IJKEXHC; имена и соответствующие значения печатаются

только в том случае, если со времени последнего выполнения этого оператора значения изменились.

Оператор печати переходов обеспечивает выдачу на печать информации о каждом выполнении оператора GO TO в данном блоке. Для организации (включения) печати переходов используется оператор

CALL IJKTRON;

а для выключения ее — оператор

CALL IJKTROF;

Язык ПЛ/1 и транслятор ДЭС ЕС ЭВМ накладывают на объекты исходной программы количественные ограничения, которые необходимо учитывать при разработке алгоритма и написании программы. Количественные ограничения, присущие языку, были отмечены в гл. IV, перечислим здесь основные ограничения, связанные с транслятором (там, где не оговорено, имеется в виду максимальное значение):

- длина строки символов — 255;
- длина строки битов — 64;
- минимальная длина строк — 1;
- размерность массива — 3;
- размер массива, байт — 32767;
- количество массивов во внешней процедуре — 32;
- уровень вложенности описателей при объявлении наименований — 8;
- глубина вложенности структур — 8;
- уровень вложенности условных операторов IF — 100;
- уровень вложенности групп DO — 12;
- уровень вложенности блоков — 3;
- уровень вложенности повторяющихся спецификаций в списках данных операторов PUT и GET — 12;
- уровень вложенности списков элементов формата — 5;
- уровень вложенности списка элементов формата, в котором допустимо использование косвенного формата — 2;
- количество блоков внешней процедуры — 63;
- размер блока, байт — 32К;
- количество знаков в выводимом оператором DISPLAY сообщении — 80;
- количество символов в сообщении-ответе в операторе DISPLAY — 255;
- количество фактических параметров (аргументов) в обращении к функции или процедуре — 12;
- длина ключа для файла с описателями INDEXED или REGIONAL (3), символов — 255;
- минимальная длина ключа для файла INDEXED, символов — 1;
- минимальная длина ключа для файла REGIONAL (3), символов — 9;
- единственно допустимая длина исходного ключа для файла с описателем REGIONAL (1), символов — 8;
- значение исходного ключа для файла REGIONAL (1) — 16777215.

Для работы транслятора ПЛ/1 в операционной системе ДОС ЕС требуется 12К байт основной памяти, один процессор, один селекторный и один мультиплексный каналы. При этом обеспечивается возможность многопрограммной работы машины. Мультипрограммирование, организуемое ДОС ЕС, выполняется таким образом, что при генерации системы для конкретной машины основная память делится на фиксированное число частей — разделов. Таких разделов может быть один, два или три. В каждом разделе одновременно может выполняться только одна программа. Каждому из разделов присвоено свое наименование и обозначение: фоновый раздел — BG, первый раздел переднего плана — F1, второй раздел переднего плана — F2. Каждому разделу отводится не только участок основной памяти, но и внешние устройства ввода-вывода.

Транслятор ПЛ/1 может быть выполнен только в фоновом разделе, протранслированная и отредактированная программа — в любом из разделов.

Ошибки, которые вызваны несоблюдением синтаксических правил языка, обнаруживаются транслятором. В этом случае для каждой обнаруженной ошибки печатается сообщение. Сообщение содержит степень грубости ошибки, номер оператора, десятичный код ошибки и текст ошибки.

## Приложение I. СТАНДАРТНЫЕ ФУНКЦИИ И ПРОЦЕДУРЫ АЛГОЛА-60

### 1. Простые функции:

- 1)  $abs(E)$  — вычисление абсолютного значения  $E$ ,  $E$  — действительное (если  $E \geq 0$ , то  $abs = E$ , при  $E < 0$   $abs = -E$ );
- 2)  $iabs(E)$  — вычисление абсолютного значения  $E$ ,  $E$  — целое;
- 3)  $sign(E)$  — функция знака выражения  $E$ , при  $E < 0$   $sign = -1$ , при  $E = 0$   $sign = 0$ , при  $E > 0$   $sign = +1$ ;
- 4)  $entier(E)$  — наибольшее целое число, не превышающее  $E$ , т. е.  $E - 1 < entier \leq E$ .

### 2. Математические функции:

- 1)  $sqrt(E)$  — вычисление квадратного корня,  $\sqrt{E}$ ,  $E \geq 0$ ;
- 2)  $sin(E)$  — вычисление синуса,  $\sin E$ ,  $E$  в радианах;
- 3)  $cos(E)$  — вычисление косинуса,  $\cos E$ ,  $E$  в радианах;
- 4)  $arctan(E)$  — вычисление главного значения арктангенса  $E$ , результат в радианах, т. е.  $-\pi/2 \leq \arctan \leq \pi/2$ ;
- 5)  $ln(E)$  — вычисление натурального логарифма  $E$ ,  $E > 0$ ;
- 6)  $exp(E)$  — вычисление показательной функции,  $e^E$ .

### 3. Процедуры прекращения работы:

- 1)  $stop$  — останов, выполнение программы прекращается;
- 2)  $fault(s, r)$  — останов, процедура вызывается, когда действие программы не определено, в следующих случаях: деление на нуль; неопределенная операция при возведении в степень; извлечение квадратного корня от отрицательного аргумента; вычисление логарифма от отрицательного или нулевого аргумента; переполнение при вычислении экспоненциальной функции; недопустимый параметр в процедуре  $outchar$ ; недопустимая литера в процедурах  $ininteger$  или  $inreal$ .

### 4. Процедуры ввода-вывода:

- 1)  $inchar(c, s, i)$  — параметр  $i$  получает значение, соответствующее первому вхождению в строку  $s$  текущей литеры канала  $c$ ; если этой литеры нет в  $s$ , то  $i$  получает значение нуля; указатель канала переходит на следующую литеру;  $c, i$  — целые,  $s$  — строка;
- 2)  $outchar(c, s, i)$  — литера в строке  $s$ , соответствующая значению  $i$ , передается в канал  $c$ ;  $c, i$  — целые,  $s$  — строка;
- 3)  $length(s)$  — подсчитывается число литер в строке  $s$ , заключенной между самыми внешними строковыми кавычками;
- 4)  $outstring(c, s)$  — литеры строки  $s$  передаются в канал  $c$ ;
- 5)  $outterminator(c)$  — выводит в канал  $c$  разделитель чисел; используется после вывода очередного числа;
- 6)  $ininteger(c, i)$  — параметр  $i$  получает значение целого числа, прочитанного с канала  $c$ ;
- 7)  $outinteger(c, i)$  — в канал  $c$  передаются литеры, представляющие значение  $i$ , а вслед за ними — разделитель;
- 8)  $inreal(c, r)$  — действительное число  $r$  получает значение числа, прочитанного с канала  $c$ ;
- 9)  $outreal(c, r)$  — в канал  $c$  передаются литеры, представляющие значение  $r$ , а вслед за ними разделитель.

5. Запросы к обстановке:

- 1) *maxreal*;
- 2) *minreal*;
- 3) *maxint*;

эти функции суть соответственно наибольшее допустимое положительное действительное число, наименьшее допустимое положительное действительное число и наибольшее допустимое положительное целое число, такие, что значение любого законного выражения будет правильно вычислено при условии, что значение каждого из операндов, а также их математически верный результат лежит внутри интервала ( $-maxreal, minreal$ ) или ( $minreal, maxreal$ ) или есть нуль, если это выражение действительного типа, и лежит внутри интервала ( $-maxint, maxint$ ), если целого типа;

4) *epsilon* — наименьшее положительное действительное число, такое, что результат вычисления  $1.0 + epsilon$  больше, чем 1.0, а результат вычисления  $1.0 - epsilon$  меньше, чем 1.0.

Приложение II. БИБЛИОТЕЧНЫЕ ПОДПРОГРАММЫ  
ФОРТРАНА IV

- 1) SIN(X) — вычисление синуса,  $\sin x$ ;
- 2) COS(X) — вычисление косинуса,  $\cos x$ ;
- 3) TAN(X) — вычисление тангенса,  $\operatorname{tg} x$ ;
- 4) COTAN(X) — вычисление котангенса,  $\operatorname{ctg} x$ ;
- 5) SINH(X) — вычисление гиперболического синуса,  $\operatorname{sh} x$ ;
- 6) COSH(X) — вычисление гиперболического косинуса,  $\operatorname{ch} x$ ;
- 7) TANH(X) — вычисление гиперболического тангенса,  $\operatorname{th} x$ ;
- 8) ARSIN(X) — вычисление арксинуса,  $\operatorname{Arcsin} x$ ;
- 9) ARCOS(X) — вычисление арккосинуса,  $\operatorname{Arccos} x$ ;
- 10) ATAN(X) — вычисление арктангенса,  $\operatorname{Arctg} x$ ;
- 11) ATAN2(X1, X2) — вычисление арктангенса частного,  $\operatorname{Arctg}(x_1/x_2)$ ;
- 12) SQRT(X) — вычисление квадратного корня,  $\sqrt{x}$ ;
- 13) EXP(X) — вычисление показательной функции,  $e^x$ ;
- 14) ALOG(X) — вычисление натурального логарифма,  $\ln x$ ;
- 15) ALOG10(X) — вычисление десятичного логарифма,  $\lg x$ .

Результат действия этих подпрограмм — действительного типа, аргументы — тоже действительного типа. Аргументы подпрограмм 1 — 7 должны быть выражены в радианах, у подпрограммы 12 аргумент неотрицательный. У каждой из подпрограмм 14 и 15 аргумент только положительный. Подпрограммы 8 — 11 вычисляют главное значение соответствующих функций.

- 16) DSIN(D) ;
- 17) DCOS(D) ;
- 18) DTAN(D) ;
- 19) DCOTAN(D) ;
- 20) DSINH(D) ;
- 21) DCOSH(D) ;
- 22) DTANH(D) ;
- 23) DARSIN(D) ;
- 24) DARCOS(D) ;
- 25) DATAN(D) ;
- 26) DATAN2(D1, D2) ;
- 27) DSQRT(D) ;
- 28) DEXP(D) ;
- 29) DLOG(D) ;
- 30) DLOG10(D) .

Подпрограммы 16 — 30 выполняют действия, совпадающие с действиями функций 1 — 15 соответственно, только результат получается с двойной точностью. Двойную точность имеют также аргументы D, D1, D2.

- 31) ABS(X) ;
- 32) DABS(D) ;
- 33) IABS(K) .

Подпрограммы 31 — 33 вычисляют абсолютную величину аргумента, который имеет действительный тип, двойную точность и целый тип соответственно.

- 34) AMIN1(XZ) ;
- 35) MIN1(XZ) ;
- 36) AMIN0(IZ) ;
- 37) MIN0(IZ) .

Подпрограммы 34 — 37 определяют наименьшее значение в списке аргументов XZ или IZ; аргументов в списке может быть два или больше. В списке аргументов элементы разделяются запятыми. Аргументами могут быть константы, простые переменные и переменные с индексами. В одном списке все аргументы должны иметь один и тот же тип и перечисляться. Так, нахождение наименьшего среди четырех элементов массива X выполняется стандартной функцией

MIN1(X(1), X(2), X(3), X(4))

Результат подпрограмм 34 и 36 — действительного типа, 35 и 37 — целого типа. Аргументы в 34 и 35 имеют тип действительный, а в 36 и 37 — целый. Таким образом, подпрограммы 35 и 36 осуществляют, кроме нахождения минимального значения, еще и преобразование типа.

- 38) DMIN1(DZ) .

Стандартная функция 38 аналогична подпрограмме 34, только в списке аргументов DZ содержатся величины с двойной точностью.

- 39) AMAX1(XZ) ;
- 40) MAX1(XZ) ;
- 41) AMAX0(IZ) ;
- 42) MAX0(IZ) ;
- 43) DMAX1(DZ) .

Подпрограммы 39 — 43 определяют наибольшее значение в списках аргументов XZ, IZ или DZ по правилам, полностью совпадающим с правилами действия подпрограмм 34 — 38 соответственно.

- 44) FLOAT(K) ;
- 45) DFLOAT(K) ;
- 46) IFIX(X) ;
- 47) HFIX(X) ;
- 48) DBLE(X) ;
- 49) SNGL(D) .

Аргумент подпрограмм 44, 45 — целого типа, 46 — 48 — действительного, 49 — с двойной точностью. Эти подпрограммы преобразуют тип своих аргументов. Тип результата у функции 44 и 49 — действительный, 45 и 48 — двойной точности, 46 — целый, 47 — целый, нестандартной длины. Таким образом, при выполнении стандартных функций 46 и 47 игнорируется дробная часть, а при выполнении подпрограммы 49 — младшие разряды. Например, оператор

Y = FLOAT(K)

присваивает переменной Y то же числовое значение, которое имеет K, однако тип уже будет действительный (ср. с оператором Y = K).

- 50) AINT(X) ;
- 51) INT(X) ;
- 52) IDINT(D) .

Подпрограммы 50 — 52 выделяют целую часть аргумента. Результат 50 имеет действительный тип, 51, 52 — целый, аргумент X — действительный D — с двойной точностью, у результата сохраняется знак аргумента.

Например, после выполнения оператора

$$I = \text{INT}(-7.5)$$

переменная I получает значение, равное -7

53) AMOD(X1, X2) ;

54) MOD(K1, K2) ;

55) DMOD(D1, D2) .

Стандартные функции 53 — 55 содержат всегда по два аргумента и осуществляют деление первого аргумента по модулю второго аргумента, т. е. результат этих подпрограмм — остаток от деления. Он имеет действительный тип для функции 53, целый — для 54, двойной точности — для подпрограммы 55.

Например, оператор

$$Y = \text{AMOD}(23., 4)$$

дает в результате действительное число 3. а оператор

$$\text{IF}(\text{MOD}(N, 2)) 1, 2, 1$$

передает управление к оператору с меткой 1 в случае нечетных значений N и оператору с меткой 2 при четных N (результат функции MOD равен 0 при четных N и 1 при нечетных N).

56) SIGN(X1, X2) ;

57) ISIGN(K1, K2) ;

58) DSIGN(D1, D2) .

Подпрограммы 56 — 58 осуществляют присвоение знака второго аргумента абсолютному значению первого аргумента. Тип результата функции 56 — действительный, 57 — целый, 58 — с двойной точностью; таков же тип аргументов.

59) DIM(X1, X2) ;

60) IDIM(K1, K2) .

Стандартные функции 59 и 60 называются подпрограммами уменьшения аргумента или вычисления положительной разности. Каждая из них определяет меньшее значение одного из двух своих аргументов, и первый аргумент уменьшается на это значение. Аргументы и результат функции 59 — действительные, 60 — целые.

Например, результат оператора

$$I = \text{IDIM}(3, 8)$$

есть 0, а операторов

$$I = \text{IDIM}(8, 3) \text{ и } I = \text{IDIM}(-3, -8)$$

— целое значение 5.

61) ERF(X) ;

62) DERF(D) ;

63) ERFC(X) ;

64) DERFC(D) .

В подпрограммах 61 и 63 действительный тип имеют как аргумент, так и результат, а в 62 и 64 — двойную точность. Эти подпрограммы вычисляют

функции ошибок, причем 61 и 62 по формуле  $\frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2} dy$ , а 63 и 64 — по

формуле  $\frac{2}{\sqrt{\pi}} \int_x^\infty e^{-y^2} dy$ .

65) GAMMA(X) ;

66) DGAMMA(D) ;

67) ALGAMA(X) ;

68) DLGAMA(D) .

Подпрограммы 65 и 66 вычисляют гамма-функцию (тип аргумента и результата у стандартной функции 65 — действительный, у 66 — двойной точности). Значение натурального логарифма от функции 65 и 66 вычисляется соответственно подпрограммами 67 и 68. Результат 67 — действительный, 68 — двойной точности.

69) CSIN(C)	$\sin(a + bi)$ ;
70) CCOS(C)	$\cos(a + bi)$ ;
71) CSQRT(C)	$\sqrt{a + bi}$ ;
72) CEXP(C)	$\exp(a + bi)$ ;
73) CLOG(C)	$\ln(a + bi)$ ;
74) CLOG10(C)	$\lg(a + bi)$ ;
75) CABS(C)	$\sqrt{a^2 + b^2}$ .

В подпрограммах 69 — 75 предполагается, что  $C$  — выражение комплексное, т. е. в обычной математической записи имеет вид  $a + bi$ . Результат этих функций имеет комплексный тип, за исключением функции 75, имеющей действительный тип.

76) CDSIN(CD) ;
77) CDCOS(CD) ;
78) CDSQRT(CD) ;
79) CDEXP(CD) ;
80) CDLOG(CD) ;
81) CDLG10(CD) ;
82) CDABS(CD) .

Подпрограммы 76 — 82 аналогичны соответствующим функциям 69 — 75, однако в них как аргумент CD, так и результат являются комплексными с двойной точностью, т. е. типа COMPLEX \*16, за исключением стандартной функции 82, результат которой действительный нестандартной длины.

83) REAL(C) ;
84) AIMAG(C) .

Подпрограммы 83 и 84 выделяют вещественную и мнимую части комплексного аргумента  $C$  соответственно. Результат здесь имеет действительный тип.

85) CMPLX(A, B) ;
86) DCMPLX(D1, D2) .

Подпрограмма 85 имеет два действительных аргумента. Результат этой стандартной функции — комплексное число, которое образуется из аргументов  $A$  и  $B$ , т. е.  $A + Bi$ . Функция 86 образует из двух аргументов с двойной точностью величину типа COMPLEX \*16.

87) CONJG(C) ;
88) DCONJG(CD) .

Эти функции имеют комплексные как аргумент, так и результат. Они дают сопряженную с аргументом величину, т. е.  $A - Bi$ , если  $C$  равно  $A + Bi$ . Функция 87 имеет комплексного типа стандартной длины аргумент и результат, 88 — типа COMPLEX \*16.

Из перечисленных стандартных функций функции 31 — 33, 44 — 60 и 83 — 88 в фортране IV являются встроенными, остальные — внешними.

### Приложение III. ВСТРОЕННЫЕ ФУНКЦИИ ПЛ/1

1. Математические функции:

- 1) ATAN(X) ; а)  $X$  действительный, результат  $\text{Arctg } X$ , в радианах;
- б)  $X$  — комплексный, результат комплексный;
- 2) ATAN(X, Y) ;  $X, Y$  действительные, результат: при  $Y > 0$   $\text{ATAN}(X/Y)$ , при  $X > Y$ ,  $Y = 0$   $\pi/2$ , при  $X \geq 0$ ,  $Y < 0$   $\pi + \text{ATAN}(X/Y)$ , при  $X < 0$ ,  $Y = 0$   $-\pi/2$ , при  $X < 0$ ,  $Y < 0$   $-\pi + \text{ATAN}(X/Y)$ , в радианах, возникает ситуация ERROR, если  $X = Y = 0$ ;
- 3) ATAND(X) ;  $X$  действительный, результат  $\text{Arctg } X$ , в градусах;

4) ATAND(X, Y) ; X, Y действительные, результат  $(180/\pi) * \text{ATAN}(X, Y)$ , в градусах, возникает ситуация ERROR, если  $X = Y = 0$ ;

5) ATANH(X) ; а) X действительный, результат  $\text{Arctg} X$ , возникает ситуация ERROR, если  $|X| \geq 1$ ; б) X комплексный, результат  $(\ln((1+X)/(1-X)))/2$ , возникает ситуация ERROR, если  $|X| = 1$ ;

6) COS(X) ; а) X действительный, в радианах, результат  $\cos X$ ; б) X комплексный, результат комплексный;

7) COSD(X) ; X действительный, в градусах, результат  $\cos X$ ;

8) COSH(X) ; а) X действительный, результат  $\text{ch} X$ ; б) X комплексный, результат комплексный;

9) ERF(X) ; X действительный, вычисляется интеграл  $\frac{2}{\pi} \int_0^x e^{-t^2} dt$ ;

10) ERFC(X) ; X действительный, вычисляется функция  $1 - \text{ERF}(X)$ ;

11) EXP(X) ; результат  $e^X$ ;

12) LOG(X) ; а) X действительный, результат  $\ln X$ ; б) X комплексный, результат комплексный;

13) LOG10(X) ; X действительный, вычисляется десятичный логарифм  $\ln X$ ;

14) LOG2(X) ; X действительный, вычисляется логарифм при основании 2, т. е.  $\log_2 X$ ;

Для функций 12) — 14) возникает ситуация ERROR, если  $X \leq 0$ .

15) SIN(X) ; а) X действительный, в радианах, вычисляется  $\sin X$ ; б) X комплексный, результат комплексный;

16) SIND(X) ; X действительный, в градусах, вычисляется  $\sin X$ ;

17) SINH(X) ; а) X действительный, результат  $\text{sh} X$ ; б) X комплексный, результат комплексный;

18) SQRT(X) ; а) X — действительный, вычисляется  $\sqrt{X}$ , возникает ситуация ERROR, если  $X < 0$ ; б) X комплексный, результат комплексный;

19) TAN(X) ; X — в радианах, результат  $\text{tg} X$ ;

20) TAND(X) ; X — в градусах, результат  $\text{tg} X$ ;

21) TANH(X) ; результат  $\text{th} X$ .

2. Арифметические функции (здесь приняты обозначения: w — общее количество разрядов в представлении числа, d — количество разрядов после точки; w, d — целые десятичные числа; w — положительное; d — может быть и отрицательным):

1) ABS(X) ; результат  $|X|$ ;

2) ADD(X, Y, w, d) ; вычисляется сумма  $X + Y$  с точностью (w) в форме с плавающей точкой, (w, d) — с фиксированной точкой;

3) BINARY(X, w, d) ; X преобразуется в двоичное число с точностью (w) в случае с плавающей точкой, (w, d) — в случае с фиксированной точкой, если w, d опущены, результату присваивается точность аргумента; допускается сокращение BIN;

4) CEIL(X) ; X действительный, результат — наименьшее целое число, которое больше или равно X;

5) COMPLEX(X, Y) ; X, Y действительные, формируется комплексное число  $X + Yi$ , допускается сокращение CPLX;

6) CONJ(X) ; X комплексный, результат — число, сопряженное X;

7) DECIMAL(X, w, d) ; X преобразуется в десятичное число по аналогии с функцией BINARY, допускается сокращение DEC;

8) DIVIDE(X, Y, w, d) ; вычисляется  $X/Y$  с разрядностью (w, d), если X, Y — с плавающей точкой, d опускается;

9) FIXED(X, w, d) ; X преобразуется в число с фиксированной точкой, если w, d опущены, используется принцип умолчания;

10) FLOAT(X, w) ; X — действительный, X преобразуется в число с плавающей точкой, если w опущено, используется принцип умолчания;

- 11) FLOOR(X) ; X действительный, определяется наибольшее целое число, которое меньше или равно X;
- 12) IMAG(X) ; X комплексный, результат — мнимая часть X;
- 13) MAX(XZ) ; XZ — список действительных аргументов, результат — значение наибольшего из аргументов;
- 14) MIN(XZ) ; XZ — список действительных аргументов, вычисляется значение наименьшего из аргументов;
- 15) MOD(X, Y) ; X, Y действительные, результат — положительный остаток от деления X/Y;
- 16) MULTIPLY(X, Y, w, d) ; результат — произведение  $X * Y$  с разрядностью (w, d), если X, Y — с плавающей точкой, d опускается;
- 17) PRECISION(X, w, d) ; X преобразуется в число с разрядностью (w, d) в случае фиксированной точки и (w) в случае плавающей точки (здесь d опускается); допускается сокращение PREC;
- 18) REAL(X) ; X комплексный, результат — вещественная часть X;
- 19) ROUND(X, N) ; N — целая десятичная константа, X — арифметическая переменная или строка цифровых знаков, если аргумент X задан с фиксированной точкой, то X округляется до N-го разряда справа от точки, если X — с плавающей точкой, то использование функции не имеет смысла;
- 20) SIGN(X) ; X действительный, результат есть -1, 0, +1 соответственно при  $X < 0$ ,  $X = 0$ ,  $X > 0$ ;
- 21) TRUNC(X) ; X действительный, соответствует функции CEIL(X) при  $X < 0$  и FLOOR(X) при  $X \geq 0$ .
3. Функции для обработки строк:
- 1) BIT(X, N) ; X — строка или выражение, N — десятичное целое число без знака, X преобразуется в строку битов длины N, если N не задано, длина результата определяется X;
- 2) BOOL(X, Y, Z) ; X, Y — переменные типа строки битов, Z — битовая строка длины 4 бита, результат — строка битов с длиной, равной наибольшей из длин строк X и Y, биты выбираются в зависимости от значений битов в Z;
- 3) CHAR(X, N) ; X — строка или выражение, N — десятичное целое число без знака, X преобразуется в символьную строку длины N, если N не задано, длина результата определяется X;
- 4) HIGH(N) ; N — десятичное целое число без знака, результат — символьная строка длины N, каждый байт которой есть шестнадцатеричное число FF;
- 5) INDEX(X, Y) ; X, Y — строки или выражения, результат — целое двоичное число без знака, указывающее номер позиции в строке X (считая слева), с которой строка Y содержится в X, или нуль, если Y не входит в состав X;
- 6) LENGTH(X) ; X — строка или выражение, результат — двоичное целое число, представляющее текущую длину X;
- 7) LOW(N) ; N — целое десятичное число без знака, результат — строка символов длины N, каждый байт представляет собой шестнадцатеричное число 00;
- 8) REPEAT(X, N) ; X — строка или выражение, N — целое десятичное число, результат — строка, полученная сцеплением строки X N раз, если  $N \leq 0$  — строка X;
- 9) STRING(X) ; X — имя переменной, массива или структуры, осуществляется сцепление всех элементов в X;
- 10) SUBSTR(X, M, N) ; X — строка или выражение, M — скалярное выражение, которое может быть преобразовано в целое число (M может быть массивом, если X — массив), N — целое десятичное число без знака, результат — часть строки X, которая начинается с позиции M и имеет длину N;
- 11) TRANSLATE(X, Y, Z) ; X, Y, Z — строки символов или битов, результат — строка, полученная из X заменой тех ее элементов, которые содержатся в Z, соответствующими элементами из Y;

12) UNSPEC(X) ; X не может быть переменной типа метки или массивом (может быть переменной типа указателя), X представляется в памяти в форме строки битов;

13) VERIFY(X, Y) ; X, Y — строки символов или битов, результат — целое двоичное число, равное 0, если каждый элемент X входит также в Y или же X есть пустая строка, равное 1, если X — не пустая, а Y — пустая строка; в других случаях результат равен номеру первого слева элемента из X, который не входит в Y.

#### 4. Функции для обработки массивов:

1) ALL(X) ; X — массив из строк битов, результат — строка битов, где *i*-й бит равен 1, если в X имеется *i*-й бит каждого элемента и он равен 1, в противном случае *i*-й бит равен 0;

2) ANY(X) ; X — массив из строк битов, результат — строка битов, где *i*-й бит равен 1, если хоть в каком-нибудь из элементов X имеется *i*-й бит и он равен 1, в противном случае *i*-й бит равен 0;

3) DIM(X, N) ; N — двоичное целое число, результат — двоичное целое число, задающее текущую N-ю размерность массива X;

4) HBOUND(X, N) ; N — двоичное целое число, результат — двоичное целое число, равное текущей верхней границе массива X по N-му измерению;

5) LBOUND(X, N) ; N — двоичное целое число, результат — двоичное целое число, равное текущей нижней границе массива X по N-му измерению;

6) POLY(X, Y) ; X, Y — одномерные массивы, результат — многочлен от X и Y;

7) PROD(X) ; произведение всех элементов массива X;

8) SUM(X) ; сумма всех элементов массива X.

#### 5. Функции специального назначения:

1) ADDR(X) — имя переменной, результат — скалярное значение указателя, которое указывает адрес переменной X в памяти;

2) ALLOCATION(X) ; X — имя небазированной переменной с управляемым способом размещения, результат — '1'B, если память выделена для X, определенного в данном блоке, в противном случае — '0'B;

3) COUNT(F) ; F — имя файла с описателем STREAM, результат — двоичное целое число с фиксированной точкой, задающее количество элементов, переданных в последней операции ввода или вывода с файлом F;

4) DATAFIELD ; результат — строка символов, содержащая поле данных, вызвавших последнее прерывание по ситуации NAME;

5) DATE ; результат — строка символов вида 'YYMMDD', где YY — текущий год, MM — текущий месяц, DD — текущий день;

6) EVENT(X) ; X — имя переменной типа события, результат — строка битов '1'B или '0'B в зависимости от текущего состояния события с именем X;

7) LINENO(F) ; F — имя файла с описателем PRINT, результат — двоичное целое число с фиксированной точкой, являющееся текущим номером печатной строки;

8) NULL определяет нулевое значение указателя;

9) ONCHAR ; результат — строка длины 1, содержащая символ, который вызвал последнее прерывание по ситуации CONVERSION;

10) ONCODE ; результат — двоичное целое число, определяющее своим значением последнее прерывание (например, константа 0 соответствует ситуации FINISH);

11) ONCOUNT ; результат — двоичное значение, равное числу прерываний, включая последнее;

12) ONFILE ; результат — строка символов, содержащая имя файла, для которого были выполнены последние ввод или вывод или преобразование, вызвавшие прерывание;

13) ONKEY ; результат — строка символов, содержащая ключ записи, вызвавшей последнее прерывание;

14) ONLOC ; результат — строка символов, содержащая имя точки входа в процедуру, которая вызвала прерывание;

15) ONSOURCE ; результат — строка символов, включающая содержимое поля, обрабатываемого при возникновении последнего прерывания по ситуации CONVERSION;

16) PRIORITY(X) ; X — имя переменной типа ветви, результат — ветвь с именем X получает приоритет по отношению к ветви, в которой выполняется данная функция;

17) STRING(X) ; X — упакованная структура, содержащая только символьные строки и (или) строки из цифровых знаков, результат — строка, полученная путем сцепления всех элементов структуры X;

18) TIME ; символьная строка из девяти элементов, указывающая текущее время в виде 'HHMMSS.TTT', где HH — часы, MM — минуты, SS — секунды, TTT — миллисекунды.

#### 6. Псевдопеременные:

1) COMPLEX(A, B) ; A, B — идентификаторы, возможно, с различными описателями, результат — действительная часть выражения присваивается A, мнимая — B (например, в результате выполнения оператора  $\text{COMPLEX}(X, Y) = 5.1 + 7.5I$ ; X будет равно 5.1, Y — 7.5;

2) IMAG(C) ; C — комплексная переменная, результат — мнимая часть выражения присваивается мнимой части переменной C, вещественная часть этой переменной остается без изменения;

3) ONCHAR ; результат — символ, являющийся значением этой функции после выполнения оператора присваивания, заменяет символ, при обработке которого возникла ситуация CONVERSION (например, оператор  $\text{ON CONVERSION ONCHAR} = 'I'$ ; обеспечивает замену непробуемого символа единицей);

4) ONSOURCE ; результат — заменяется символьная строка (поле), при обработке которой возникла ситуация CONVERSION;

5) REAL(C) ; C — комплексная переменная, результат — вещественная часть выражения присваивается вещественной части переменной C, мнимая часть этой переменной остается без изменения;

6) SUBSTR(X, M, N) ; X, M, N имеют тот же смысл, что и в п. 3, только X не может быть массивом или выражением, результат — значение выражения, преобразованного в строку символов, присваивается подстроке, определенной псевдопеременной SUBSTR (например, если X есть 'ABCDE', а Y есть 'XYZPQ', то оператор  $\text{SUBSTR}(X, 2, 3) = \text{SUBSTR}(Y, 3, 3)$ ; заменяет значение X на 'AZPQE');

7) UNSPEC(X) ; X имеет тот же смысл, что и в п. 3, только X не может быть выражением, результат — выражение преобразуется в строку битов и присваивается переменной X без преобразования в тип X (например, оператор  $\text{UNSPEC}(V) = F$ ; преобразует F в строку битов и присваивает это значение V).

## ЛИТЕРАТУРА

### Общая

1. Алгоритмический язык АЛГОЛ-60: Пересмотренное сообщение. (Под ред. А. П. Ершова, С. С. Лаврова, М. Р. Шура-Бура).— М.: Мир, 1965.
2. American Standards Association, FORTRAN vs. Basic FORTRAN. — Communications of the ASM, 7, N 10, October 1964, с. 591—625.
3. Универсальный язык программирования PL/1 (Под ред. В. М. Курочкина).— М.: Мир, 1968.
4. Королев Л. Н. Структуры ЭВМ и их математическое обеспечение. — М.: Наука, 1978.
5. Крилицкий Н. А., Миронов Г. А., Фролов Г. Д. Программирование и алгоритмические языки. — М.: Наука, 1979 [Справочная математическая библиотека].
6. Джермейн К. Программирование на IBM/360. — М.: Мир, 1978.
7. Турский В. Методология программирования. — М.: Мир, 1981.

### К главе II

8. Лавров С. С. Универсальный язык программирования (АЛГОЛ-60).— М.: Наука, 1972.
9. Алгоритмический язык АЛГОЛ-60. Модифицированное сообщение. — М.: Мир, 1982.

### К главе III

10. Грунд Ф. Программирование на языке ФОРТРАН IV. — М.: Мир, 1976.
11. Хьюз Ч., Пфлигер Ч., Роуз Л. Методы программирования: курс на основе ФОРТРАНА. — М.: Мир, 1981.

### К главе IV

12. Скотт Р., Сондак Н. PL/1 для программистов. — М.: Статистика, 1977.
13. Аугустон М. И., Балодис Р. П., Барздинь Я. М., Икауниекс Э. А., Калининш А. А. Программирование на PL/1 ОС ЕС. — М.: Статистика, 1979.

### К главе V

14. Демин В. Ф., Добролюбов Л. В., Степанов В. А. Системы программирования на алголе. — М.: Наука, 1977.
15. Салтыков А. И., Макаренко Г. И. Программирование на языке ФОРТРАН. — М.: Наука, 1976.
16. Катцан Г. Язык ФОРТРАН-77. — М.: Мир, 1982.
17. Системы математического обеспечения ЕС ЭВМ. (Под ред. А. М. Ларионова). — М.: Статистика, 1974.
18. Языки программирования ДОС ЕС ЭВМ: Краткий справочник. — М.: Статистика, 1977.
19. Лебедев В. Н., Соколов А. П. Введение в систему программирования ОС ЕС. — М.: Статистика, 1978.
20. Программирование на языке Бейсик-плюс для СМ-4/В. П. Семик, Б. Р. Монцбонч, Д. П. Непочатых и др. — М.: Финансы и статистика, 1982.

## ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- Автокод 22  
Алгол 24, 27, 30, 252, 253  
Алгоритм 17  
Аналоговая вычислительная машина 7
- Базисный фортран 270, 282  
Байт 10  
Библиотечная подпрограмма 120  
Бит 10  
Бланк фортрана 109  
Блок 44, 66, 201, 208  
— вложенный 66  
— именованный 156  
Блок-схема 16, 18  
Быстродействие ЭВМ 8
- Величина 34, 92  
— собственная 70  
Выражение 25, 39, 97, 99, 197  
— арифметическое 39, 54, 98  
— именованное 39, 55  
— индексное 35, 96  
— логическое 39, 41, 54, 99
- Глубина рекурсий 85  
Граница изменения индексов 37, 103  
Граничная пара 37  
Группа 201, 205
- Данные 92, 178  
Десятичный порядок 33, 93  
Дробь 33  
— правильная 33, 92
- Заголовок оператора 102  
— процедуры 73  
— цикла 61, 116  
Задание 249  
Задача 249  
Замена формального параметра 74, 77, 126, 130, 134  
Запись 146, 250, 264  
Запоминающее устройство 8, 13  
Значение начальное 104
- Идентификатор 34, 95, 181  
— глобальный 67, 79  
— локализованный 67, 79  
— процедуры 35  
Индекс спецификации 159  
Интерпретация 28
- Ключевое слово 91, 102, 182  
Кобол 25  
Команда 8, 21  
Комментарий 71, 76, 102, 110, 200  
Компиляция 28  
Константа 32, 92, 179  
— арифметическая 92, 179  
— действительная 33, 92, 179  
— комплексная 94, 180  
— логическая 31, 94  
— строковая 19  
— текстовая 94  
— целая 33, 92
- Константа числовая 32  
— шестнадцатеричная 94  
Коэффициент кратности 104, 141, 157, 159  
— масштабный 168
- Лисп 25
- Макрокоманда 28  
Массив 35, 95, 105, 181  
— с переменными границами 68, 128  
Масштабный коэффициент 168  
Математическое обеспечение 9, 248  
Машинная операция 21  
Машинное слово 8, 11  
Метка оператора 47, 95, 195  
Модуль 250
- Именованное 173, 174  
Начальное значение 104, 195
- Область действия идентификатора 68  
— цикла 116  
Обращение к функции 35, 81, 97, 124, 126  
Общий блок 137  
Объявление типа 96  
— — автоматическое 96, 102  
— — неявное 102  
— — явное 103  
Операнд 21, 98  
Оператор 24, 30, 44, 91, 101, 173, 184  
— безусловного перехода 49, 110, 203  
— безусловный 49  
— ввода 45, 101, 146, 150, 152, 174, 231, 233, 242  
— внешних подпрограмм 129, 173  
— внутренней передачи 241, 279  
— возврата 124, 173  
— — с параметром 131  
— ввода 135, 214  
— вывода 45, 101, 146, 151, 153, 174, 231, 233, 242  
— вызова подпрограммы 131, 173, 210  
— задания размеров массивов 105  
— — формата 158, 173  
— исполняемый 101, 184  
— конца файла 173  
— неисполняемый 101, 184  
— объявления данных 185  
— окончания 119, 173  
— описания общих блоков 173  
— — файлов 147, 173  
— описательный 102, 173  
— останова 119, 174, 208  
— отладки 272  
— перехода 44, 47, 110, 173, 203  
— поиска записи файла 150  
— присваивания 44, 46, 107, 173, 202  
— — метки 108, 173  
— — начальных значений 141, 173  
— продолжения 117, 173  
— процедуры 74  
— пустой 44, 57, 185  
— составной 44, 57  
— управления 101, 110, 174, 203

Оператор управления файлами 149, 173  
 — условный 49, 51, 113, 204

— — полный 51  
 — формата 173, 240  
 — формирования списков 156, 173  
 — цикла 60, 116, 173, 206

— эквивалентности 144, 173  
 Оператор-функция 121, 174  
 Операционная система 12, 248  
 Операция 21, 39, 98, 197  
 — арифметическая 39, 98, 197  
 — логическая 41, 99, 197  
 — машинная 21  
 — отношения 41, 99, 197

Описание 37, 44, 185  
 — массива 37, 190  
 — переключателя 58  
 — процедуры 73  
 — структуры 191  
 — типа 37  
 — функции 81

Описатель 32, 224, 227  
 Основная программа 102  
 Основной символ языка 30, 91  
 Отладка программы 16, 252  
 Отношение 41, 99

Память 8  
 — внешняя 13  
 — оперативная 8, 9

Параметр фактический 35, 74, 96, 126, 212  
 — формальный 35, 73, 123, 209  
 — цикла 60, 116, 207

Переключатель 58  
 Переменная 34, 92, 95, 181  
 — действительная 34, 96, 186  
 — комплексная 96, 186  
 — логическая 34, 96  
 — простая 34, 95, 185  
 — с индексами 34, 95, 190  
 — целая 34, 96

ПЛ/1 26, 178  
 Побочный эффект 86  
 Повышенная точность 270  
 Подблок 66  
 — независимый 66

Подпрограмма 24, 44, 72, 101, 130  
 — библиотечная 120  
 — данных 142, 173

Подпрограмма-процедура 130, 174, 212  
 Подпрограмма-функция 122  
 Поколение ЭВМ 9

Прерывания 217  
 Программа 16, 17, 43, 101  
 — вызывающая 102  
 — обрабатывающая 251  
 — основная 24, 102  
 — отладочная 252  
 — рабочая 28  
 — сервисная 251  
 — системная 252  
 — служебная 284  
 — управляющая 251

Программирование 8  
 Программная единица 102  
 Программное обеспечение 9, 248  
 Процедура 72, 201, 212  
 — без параметров 82  
 Процедура-функция 80, 210  
 Процессор 8

Рабочая программа 28  
 Раздел памяти 250

Размерность массива 35, 95, 103, 190  
 Распределение памяти 22, 223  
 Режим отладочный 252  
 Рекурсивная процедура 84  
 Рекурсивное обращение к процедуре 85  
 Ресурсы задачи 249

Сегмент программы 24, 102  
 Система команд 8  
 — математического обеспечения 248  
 — операционная 12, 248  
 — программирования 248  
 —числения 21

Слово ключевое 31, 91, 102  
 — машинное 8, 11  
 — служебное 31, 102  
 Снобол 26

Спецификации 75  
 — преобразования 159, 160  
 — управляющие редакционные 159, 169  
 — формата 158, 236

Список ввода-вывода 148, 233  
 — граничных пар 37, 294  
 — значений 77  
 — индексов 35, 96  
 — левой части 46  
 — переключателя 58  
 — спецификаций 158  
 — фактических параметров 74, 96, 124, 131, 210  
 — формальных параметров 73, 123, 131, 209

Стандартные подпрограммы 181  
 — процедуры 34  
 — функции 34, 120, 215  
 Строка 36, 180  
 Структура 181

Тело процедуры 73  
 Транслятор 23, 27

Указатель длины 102  
 — переключателя 59  
 — функции 35, 81, 97  
 Управление печатью 172, 239  
 Управляющие символы 172, 239  
 Условие 49

Файл 146, 193, 250, 264  
 Формат 158, 236  
 — переменный 278  
 Фортран 24, 91, 292  
 Функция 35, 96, 210

Цикл 60, 116, 173, 206  
 — вложенный 63, 118, 207  
 — невязный 148  
 Цифровая вычислительная машина 7

Число 32, 33, 92, 93, 94, 179

Шаблон 188  
 Шаг задания 249

Электронная вычислительная машина 8  
 Элемент массива 35, 95, 103, 181

Язык алгоритмический 23  
 — входной 29  
 — машинный 21  
 — программирования 20  
 — эталонный 29  
 Ячейка памяти 8

*Аарне Антонович Пярнпуу*  
ПРОГРАММИРОВАНИЕ  
НА АЛГОРИТМИЧЕСКИХ ЯЗЫКАХ

Редактор *Л. Г. Силкова.*  
Техн. редактор *С. Я. Шкляр.*  
Корректоры: *О. А. Сигал* и *Т. С. Вайсберг*

ИБ № 12303

Сдано в набор 06.01.83. Подписано к печати 28.10.83. Т-19276.  
Формат 60×90<sup>1/16</sup>. Бумага тип. № 3. Литературная гарнитура. Высокая печать. Условн. печ. л. 20. Усл. кр.-отт. 20, 125.  
Уч.-изд. л. 25,07. Тираж 78 000. экз. Зак. № 501. Цена 1 р.

Издательство «Наука»  
Главная редакция физико-математической литературы  
117071, Москва, В-71, Ленинский проспект, 15

Ленинградская типография № 2 головное предприятие  
ордена Трудового Красного Знамени Ленинградского  
объединения «Техническая книга» им. Евгения Соколовой  
Союзполиграфпрома при Государственном комитете СССР  
по делам издательств, полиграфии и книжной торговли.  
198052, г. Ленинград, Л-25, Измайловский проспект, 29.