



М
МММ
МММММММ
МММ
М
М
МММ
М

А. И. САЛТЫКОВ
Г. Л. СЕМАШКО

ПРОГРАММИРОВАНИЕ ДЛЯ ВСЕХ

А. И. САЛТЫКОВ
Г. Л. СЕМАШКО

ПРОГРАММИРОВАНИЕ ДЛЯ ВСЕХ

Под редакцией
В. П. ШИРИКОВА

ИЗДАНИЕ ВТОРОЕ,
ПЕРЕРЕБОТАННОЕ



МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1986

ББК 22.18
С16
УДК 519.6

Салтыков А. И., Семашко Г. А. Программирование для всех.— М.: Наука. Главная редакция физико-математической литературы, 1986.— 176 с.

В книге излагаются основы программирования на ЭВМ. Рассматриваются вопросы, связанные с представлением решения математических задач в виде алгоритмов и программ. Основные приемы программирования иллюстрируются на языке фортран. Это позволяет читателям приобрести навыки программирования, достаточные для практической работы на ЭВМ (например, на БЭСМ-6 или ЕС ЭВМ).

Книга рассчитана на широкий круг читателей и может быть использована как учебное пособие для учащихся старших классов средней школы, техникумов и студентов вузов.

1-е издание вышло в 1980 г.

Альберт Иванович Салтыков, Галина Львовна Семашко

ПРОГРАММИРОВАНИЕ ДЛЯ ВСЕХ

Редактор *Н. И. Воронина*

Художественный редактор *Т. Н. Кольченко*

Технический редактор *В. Н. Кондакова*

Корректор *Н. Б. Румянцева*

ИБ № 12836



Scan AAW

Подписано к печати 16.07.86. Т-16724. Формат 84×108¹/₃₂. Бумага тип. № 3. Гарнитура литературная. Печать высокая. Усл. печ. л. 9,24. Усл. кр.-отт. 9,45. Уч.-изд. л. 9,69. Цена 30 коп.

Ордена Трудового Красного Знамени издательство «Наука». Главная редакция физико-математической литературы. 117071 Москва В-71, Ленинский проспект, 15

Отпечатано дополнительным тиражом 335 000 экз. с матриц Первой Образцовой типографии имени А. А. Жданова в типографии «Известий». Москва, Пушкинская пл., 5. Зак. 2634.

С $\frac{1702070000-126}{053(02)-86}$ 41-86

© Издательство «Наука»,
Главная редакция
физико-математической
литературы, 1980;
с изменениями, 1986

ОГЛАВЛЕНИЕ

Предисловие ко второму изданию	5
Предисловие к первому изданию	7
Г л а в а I. Для начинающих	9
1.1. ЭВМ в нашей жизни	9
1.2. Зачем машине программа?	10
1.3. Структура ЭВМ	13
1.4. Путь от постановки задачи до ее решения на ЭВМ	15
1.5. Некоторые сведения о чтении и записи информации на ЭВМ	19
1.6. Всегда ли $2 \times 2 = 4$?	19
1.7. От машинных команд — к алгоритмическим языкам	21
1.8. Какие языки «знает» ЭВМ?	22
1.9. Элементы языка фортран	23
1.9.1. Константы языка фортран	24
1.9.2. Переменные величины	25
1.9.3. Арифметическое выражение	26
1.9.4. Операторы языка фортран	28
1.9.5. Массивы переменных	51
1.10. Первый выход на машину. Задания для БЭСМ-6 и ЕС ЭВМ	52
Г л а в а II. Для тех, кто решился идти дальше	61
2.1. Дополнительные сведения о языке фортран	61
2.1.1. Вычисления с повышенной точностью	61
2.1.2. Вычисления с комплексными числами	63
2.1.3. Использование величин разных типов в одном арифметическом выражении	63
2.1.4. Преобразование типа величин в результате выполнения оператора присваивания	65
2.1.5. Операторы описания типа	66
2.1.6. Оператор DATA	66
2.1.7. Вычисляемый оператор GO TO	68
2.1.8. Оператор IF логический	70
2.1.9. Двумерные массивы	72
2.1.10. Вывод двумерного массива на печать	73
2.1.11. Подпрограмма	74
2.1.12. Подпрограмма-функция	78

2.1.13. Оператор COMMON	82
2.1.14. Модульная структура программ	84
2.2. Как работает транслятор?	87
2.3. Некоторые поучительные примеры	92
2.4. Работа с магнитными лентами и дисками	95
2.5. Начинающему пользователю терминала	99
2.6. Работа за терминалом ЭВМ БЭСМ-6 в операционной системе Дубна (вариант Мультитайп)	104
Г л а в а III. Полезные сведения и практические советы . .	123
3.1. Диалекты языка фортран	123
3.2. Версии фортрана для ЕС ЭВМ и БЭСМ-6	124
3.3. О переводе программ с БЭСМ-6 на ЕС ЭВМ	125
3.4. Перевод программ с ЕС ЭВМ на БЭСМ-6	129
3.5. Случаи разной трактовки фортранной программы трансляторами на БЭСМ-6 и ЕС ЭВМ	132
3.6. Краткие сведения о языке фортран 77	134
Г л а в а IV. Наши первые программы	140
4.1. ЭВМ рисует картинки	140
4.2. Программы печатают таблицы функций	144
4.3. Работа с массивами	146
4.4. Решение квадратного уравнения	147
4.5. Простейший лабиринт	148
4.6. Решение уравнения $f(x)=0$ методом половинного де- ления	149
4.7. Некоторые рекомендации начинающему програм- мисту	150
Заключение	159
Ответ и решения	161
Приложение I. Таблица отличий версий фортрана	163
Приложение II. Стандартные функции	172
Список литературы	176

ПРЕДИСЛОВИЕ КО ВТОРОМУ ИЗДАНИЮ

Активными читателями первого издания книги, вышедшего в 1980 г., оказались пользователи ЭВМ, а также студенты младших курсов. В средних школах г. Дубны, где вот уже более 20 лет ведется преподавание основ программирования, книга стала базовым учебным пособием.

Сейчас, когда с нового учебного года по всей стране вводится компьютерный всеобуч, книга может оказаться полезной в его практическом осуществлении.

Отметим еще одно событие, совпавшее с выпуском второго издания: 20 лет назад, весной 1965 г., на страницах центральной прессы впервые появилось слово «БЭСМ-6». Созданная коллективом Института точной механики и вычислительной техники (ИТМиВТ) под руководством выдающегося ученого и конструктора академика Сергея Алексеевича Лебедева, БЭСМ-6 составила целую эпоху в развитии отечественной вычислительной техники. И сейчас, спустя два десятилетия, она успешно трудится во многих вычислительных центрах страны и за ее пределами (в ГДР, Индии).

Во втором издании учтены замечания и пожелания читателей, а также внесены изменения, связанные с модернизацией вычислительной техники и математического обеспечения ЭВМ. Кроме того, добавлен материал об экранном редакторе для БЭСМ-6.

Язык фортран, основы которого излагаются на страницах книги, несмотря на почтенный возраст все еще остается одним из самых распространенных языков программирования для решения вычислительных задач науки и техники. Будучи достаточно простым для изучения, он может оказаться полезным и для последующего освоения более современных языков программирования (паскаль, ада и др.). Отметим, что свойство модульности,

впервые реализованное в фортране, в том или ином виде перекочевало во многие языки программирования, созданные позднее. Для удобства читателей параграфы, посвященные описанию языка фортран, сопровождаются развернутым перечнем пунктов.

Авторы выражают благодарность А. А. Корнейчуку, Г. Л. Мазному, М. Б. Попову и Вик. С. Штаркману за полезные консультации. Мы благодарны А. А. Аникеичу и А. С. Хорозовой, обратившим наше внимание на отдельные недочеты и опечатки, допущенные в первом издании.

Мы будем признательны читателям, которые пришлют свои пожелания, замечания и предложения по новому изданию книги.

А. И. Салтыков, Г. Л. Семашко

ПРЕДИСЛОВИЕ К ПЕРВОМУ ИЗДАНИЮ

В книге освещаются вопросы, связанные с программированием на электронных вычислительных машинах (ЭВМ). Материал книги излагается применительно к большому ЭВМ, имеющему наибольшее распространение в нашей стране: БЭСМ-6 и ЕС ЭВМ, с некоторым акцентом на БЭСМ-6.

Книга состоит из четырех глав. Глава I рассчитана на читателя, впервые приступающего к программированию на ЭВМ. Здесь даются начальные представления о программировании, о структуре ЭВМ, об алгоритмических языках. Далее читатель знакомится с элементами языка фортран — одного из наиболее распространенных и простых алгоритмических языков. В конце главы изложены вопросы, полезные читателям, собирающимся практически работать на одной из ЭВМ типа БЭСМ-6 или ЕС ЭВМ. Таким образом, круг вопросов, освещенных в гл. I, позволяет не только составлять простейшие программы, но и получать результаты их работы на ЭВМ.

Поскольку материал этой главы рассчитан на начинающего читателя, мы были вынуждены в ряде случаев поступиться строгостью изложения. Строгое описание вводимых понятий труднодоступно для начинающего читателя. Поэтому мы приняли следующую методику изложения. На первом этапе даются формулировки, с одной стороны, понятные начинающему читателю и, с другой стороны, достаточно правильные с учетом их последующего уточнения. При дальнейшем изложении материала сообщаются дополнительные сведения, позволяющие читателю сформировать более полное представление о данном понятии.

В гл. II изложен ряд вопросов, предназначенных для читателя, обладающего знаниями в объеме первой главы. Здесь приводятся дополнительные сведения о фортране,

необходимые для составления достаточно сложных программ. Материал по языку фортран подобран так, что все сказанное в равной степени справедливо как для ЕС ЭВМ, так и для БЭСМ-6, за исключением тех случаев, которые оговорены особо. Отметим, что язык фортран в книге изложен не в полном объеме. Читатель, желающий получить сведения о всех возможностях фортрана, может обратиться к одной из книг: [7, 15] (БЭСМ-6), [3, 17] (ЕС ЭВМ).

Программисту, работающему на ЭВМ, могут быть полезны приведенные здесь сведения о работе транслятора, а также поучительные примеры, содержащие практические рекомендации по реализации некоторых вычислительных алгоритмов. Кроме того, в этой главе изложены некоторые вопросы, адресованные начинающим пользователям ЭВМ: о работе с магнитными лентами и дисками, о работе с терминалами в операционной системе «Дубна» на БЭСМ-6.

Материал гл. III предназначен для пользователей, которым приходится переходить на работу с ЭВМ другого типа: с БЭСМ-6 на ЕС ЭВМ или с ЕС ЭВМ на БЭСМ-6. Здесь же сообщается информация о версиях и стандартах языка фортран, полезная для общего развития.

В гл. IV приведены примеры простых программ.

Некоторым читателям может показаться, что в книге часто рассматриваются слишком «элементарные» вопросы технического характера. Наш опыт показывает, что когда пользователь приступает к новому виду работы на ЭВМ, то у него возникает множество как раз таких вопросов. Например, человек собирается впервые запустить программу на ЭВМ. Что он должен для этого сделать? Или, скажем, в программе предусмотрена работа с внешней памятью. Как это практически осуществить? В этих элементарных сведениях нуждаются многие пользователи. На поиски такого рода информации начинающие программисты тратят много времени и сил. Мы старались им помочь конкретными рекомендациями.

Мы благодарны А. В. Ракитскому, прочитавшему рукопись и сделавшему ряд полезных замечаний. Нам помогли консультации О. Н. Ломидзе и Н. Е. Мазепы.

Просим читателей присылать свои замечания и пожелания.

Дубна, 1980 г.

Авторы

Г л а в а I

ДЛЯ НАЧИНАЮЩИХ

1.1. ЭВМ в нашей жизни

С древнейших времен люди создавали всевозможные механизмы и машины для облегчения своего труда.

Однако до недавнего времени машины облегчали лишь физический труд человека. Деятельность, связанная с умственным трудом, обходилась без серьезного участия машин.

Создание быстродействующих электронных вычислительных машин (ЭВМ) справедливо считается одним из выдающихся научно-технических достижений человечества.

Современные ЭВМ способны выполнить за секунду миллионы арифметических операций (сложений, вычитаний и т. д.), в то время как опытный вычислитель тратит на выполнение одного арифметического действия в среднем около полминуты. Таким образом, с появлением ЭВМ за короткий промежуток времени (около 35 лет) скорость вычислений возросла примерно в 100 миллионов раз.

Приведем пример, ярко иллюстрирующий преимущества ЭВМ [16].

Английский математик XIX века Шенкс потратил более 20 лет своей жизни на вычисление числа π с точностью 707 верных значащих цифр. Этот результат получил славу рекорда вычислений XIX века. Однако впоследствии было обнаружено, что Шенкс ошибся в 520-м знаке, и поэтому все последующие значащие цифры были вычислены неверно.

На ЭВМ число π вычислено с точностью 500 тысяч знаков. На это потребовалось всего лишь несколько часов работы машины.

Такие качества ЭВМ, как быстроедействие и возможность получения гарантированно правильного результата, позволили решать принципиально новые задачи. Например, до появления ЭВМ задача управления с земли ракетой вообще не могла быть решена. Объем вычислений настолько велик, что координаты только одной точки траектории ракеты при ручном счете были бы получены уже после ее падения.

Электронные вычислительные машины прочно вошли в нашу жизнь. Многие читатели сами пользовались услугами ЭВМ, отправляясь поездом из Москвы в дальний путь. Билеты, приобретаемые на московских вокзалах, выдаются прямо с телетайпа, управляемого ЭВМ.

ЭВМ вычисляют элементы орбиты искусственного спутника Земли и управляют производственным процессом на заводе, помогают судьям на спортивных соревнованиях и ведают распределением мест на самолеты. Уже в 1973 г. сотрудники американского журнала «Computers and Automation» насчитали 2500 областей применения ЭВМ.

Для решения разнообразных задач, выдвигаемых наукой и народным хозяйством, создаются ЭВМ разных типов. Одни из них имеют специальное назначение (например, для управления космическим кораблем). Такие ЭВМ называются специализированными. Другие, называемые универсальными, могут применяться в различных областях науки и техники. Все, что говорится в этой книге, относится только к универсальным ЭВМ, которые мы будем называть просто ЭВМ, или машинами.

1.2. Зачем машине программа?

Во многих восточных сказках рассказывается о могущественных джиннах, способных творить чудеса. Джинн покорно служит каждому, кто знает соответствующие заклинания и умеет правильно их произносить. Но если человек забыл или спутал заклинание, то джинн отказывается служить ему.

Современные электронные вычислительные машины ведут себя в некотором смысле подобно джиннам. «Заклинаниями» для ЭВМ служат программы, составленные людьми (программистами). Машины аккуратно выполняют все действия, предписанные программами. Сами программы записываются в строгом соответствии с опреде-

ленными правилами. Малейшее нарушение этих правил может привести к тому, что машина откажется выполнять требуемые действия. Только неукоснительно выполняя все формальные правила написания программ, можно заставить ЭВМ верно служить.

Подобно тому, как джинны с одинаковой готовностью исполняли и добрую и злую волю своего повелителя, так и ЭВМ педантично следуют указаниям любой программы, в которой нет нарушений формальных правил. «Умной» машина будет лишь тогда, когда «умна» введенная в нее программа: программист выбрал удачный метод решения задачи и рационально реализовал его на ЭВМ.

Следует подчеркнуть, что машина не может заметить ошибки в алгоритме [11] вашей программы (например, вместо сложения запрограммировано вычитание).

Программист должен продумать и записать однозначную последовательность действий, ведущую к решению задачи. Дело в том, что конструкция машины не позволяет сразу произвести достаточно сложное действие, например, решить квадратное уравнение.

ЭВМ устроена таким образом, что умеет выполнять лишь арифметические и некоторые другие простые операции. Человек задает программу работы машины. В программе, состоящей из отдельных команд, указывается, какие действия, в каком порядке и над какими числами следует выполнить.

Например, для вычисления на ЭВМ площади круга радиуса $R=15$ надо дать машине следующие команды.

1. Умножить 15 на 15.
2. Умножить предыдущий результат на 3,14159.
4. Отпечатать результат.
4. Закончить работу («стоп»).

При решении этой задачи машина выполнит две операции умножения и исполнит команду печати. По команде «стоп» она переключит свое «внимание» на новую программу.

Рассмотрим, какие действия следует произвести машине для решения линейного уравнения $ax=b$. Вспомним, как решаются такие уравнения. Если $a \neq 0$, то имеется единственное решение $x=b/a$; если $a=0$ и $b=0$, то любое значение x является решением; если $a=0$, а $b \neq 0$, уравнение не имеет решения.

Теперь запишем программу вычислений в виде следующих команд.

1. Проверить, равняется ли a нулю. Если да — перейти к п. 5, если нет — продолжить вычисления.

2. Разделить b на a .

3. Отпечатать результат.

4. Закончить работу («стоп»).

5. Проверить, равняется ли b нулю.

Если да, перейти к п. 8, если нет — продолжить вычисления.

6. Отпечатать текст: «Уравнение не имеет решения».

7. Закончить работу («стоп»).

8. Отпечатать текст: « x — любое число».

9. Закончить работу («стоп»).

В зависимости от значений a и b машина будет выполнять либо команды 1, 2, 3, 4 ($a \neq 0$), либо 1, 5, 6, 7 ($a = 0, b \neq 0$), либо 1, 5, 8, 9 ($a = 0, b = 0$).

В приведенном примере программой предусмотрен случай $a = 0$. Если исследование на нуль a и b не проводить, то программа окажется гораздо короче.

1. Разделить b на a .

2. Отпечатать результат.

3. Закончить работу («стоп»).

Но в том случае, когда окажется, что $a = 0$, машина прекратит выполнение программы при попытке разделить на нуль. Программист не будет знать, какой из двух случаев имел место: задача не имеет решений либо имеет бесконечное множество решений.

Во всех случаях, когда машина не может выполнить какое-либо действие (разделить на нуль, извлечь корень из отрицательного числа, перемножить два слишком больших числа и т. п.), она прекращает выполнение программы. Такие ситуации, как правило, являются следствием допущенной в программе ошибки, которую надлежит найти и исправить.

Итак, решая какую-либо задачу, ЭВМ выполняет программу, в которой точно описан весь ход вычислений.

Разнообразие решаемых машиной задач обусловлено тем, что посредством программ можно задавать различные последовательности операций и разные исходные данные.

Вопросы, связанные с методикой составления программ для ЭВМ, потребовали создания целой научно-прикладной дисциплины, называемой программированием.

Как правило, можно составить несколько разных программ, предназначенных для одной и той же цели. Эти

программы будут отличаться друг от друга длиной и скоростью выполнения на машине. Для написания «красивой» (т. е. оптимальной по параметрам) программы надо хорошо знать программирование, математику, методы вычислений, проявлять изобретательность и стремиться к созданию совершенной программы.

Прежде чем стать настоящим музыкантом, надо долго и упорно овладевать техникой исполнения, играя скучные этюды и упражнения. Так же и программист должен начать с терпеливого изучения и запоминания основных правил программирования, и только после этого он может познать радость творчества.

1.3. Структура ЭВМ

Современная большая ЭВМ — это сложный агрегат, состоящий из множества электронных, электромеханических и механических устройств [10]. Для их размещения требуется площадь в несколько десятков и даже сотен квадратных метров. Так, для размещения ЭВМ БЭСМ-6 необходим зал площадью 200—250 м².

Развитие электроники в 70-х годах привело к уменьшению размеров электронных блоков ЭВМ. Появились мини-ЭВМ, умещающиеся в небольшой комнате, и микро-ЭВМ размером не более спичечной коробки. Однако размеры электромеханических и механических устройств (магнитофонов, печатающих устройств и т. п.), которыми оснащаются в основном большие ЭВМ, не проявляют явной тенденции к уменьшению.

В настоящее время к ЭВМ подсоединяется все большее количество вспомогательных устройств (графопостроители, дисплеи), которые по своим размерам могут существенно превосходить собственно ЭВМ. Поэтому несмотря на уменьшение размеров электронных блоков современных ЭВМ, для размещения ЭВМ (вместе со вспомогательным оборудованием) по-прежнему требуются значительные площади.

В ЭВМ можно выделить наиболее важные устройства (блоки), каждое из которых осуществляет определенные, только ему присущие функции. Этих блоков 5: память, устройство управления, арифметическое устройство, устройства ввода и вывода.

П а м я т ь ЭВМ является хранилищем всей информации, с которой работает машина. Эта информация включает в себя программу, исходные данные и резуль-

таты, получаемые в процессе счета. Память состоит из ячеек, в каждую из которых можно поместить число либо команду программы. Объем памяти составляет от нескольких тысяч до миллионов ячеек, в зависимости от типа ЭВМ. Единицей измерения объема памяти ЭВМ является К. Один К (или двоичная тысяча) равен 1024. Например, память БЭСМ-6 содержит 32768 ячеек, что составляет 32К.

Многие ЭВМ, в частности машины серии ЕС ЭВМ, имеют иную единицу измерения объема памяти — *байт*. Каждый байт состоит из 8 двоичных разрядов (битов).

Байты удобны при работе с символьной информацией, когда каждый символ (например, буква) кодируется восьмиразрядным двоичным числом. При выполнении арифметических операций над числами необходимы более крупные информационные единицы (иначе точность вычислений будет мала). Поэтому для проведения вычислений объединяют несколько соседних байтов в одну ячейку. На машинах серии ЕС такая ячейка состоит из 4-х или 8-ми (реже из 2-х) байтов.

В дальнейшем мы будем называть *ячейкой* (или *машинным словом*) участок памяти, отведенный для хранения одного числа.

Та память, о которой до сих пор говорилось, называется *оперативной памятью*. Любое число, хранящееся в оперативной памяти, может быть использовано для выполнения операций. Кроме оперативной памяти, у ЭВМ есть так называемая *внешняя память* (магнитные ленты, диски, барабаны). Информация, хранящаяся во внешней памяти, не может быть непосредственно использована, ее надо предварительно перенести из внешней памяти в оперативную.

Внешняя память превосходит по объему оперативную память в десятки и сотни раз. Однако скорость обмена информацией между внешней и оперативной памятью в десятки раз меньше скорости выполнения машинных операций.

Устройство управления организует весь процесс работы ЭВМ в соответствии с заданной программой. Оно производит расшифровку каждой команды и подает сигналы другим устройствам ЭВМ, участвующим в выполнении этой команды.

Машинные операции (сложение, умножение и т. д.) выполняются в арифметическом устройстве ЭВМ. Сюда из памяти поступают те числа, над ко-

торыми должна быть выполнена операция. Полученный результат либо остается в арифметическом устройстве, либо сразу помещается в память.

Устройство ввода читает информацию, необходимую для работы машины, т. е. программу и исходные данные. Эта информация должна быть подготовлена для восприятия машиной.

Один из способов ввода — ввод с перфокарт. *Перфокарта* — это лист плотной бумаги стандартных размеров. Посредством специальных устройств — перфораторов — в карте пробиваются прямоугольные отверстия. Каждому вводимому символу (цифре, букве и т. д.) на карте соответствует свое расположение таких пробивок. Пробитые карты закладываются в устройство ввода, в котором при чтении перфокарт комбинации пробивок преобразуются в электрические сигналы.

После того как программа введена в ЭВМ, она может быть выполнена. Результаты работы машины по команде программы поступают на устройство вывода, где происходит преобразование электрических сигналов в числа, графики и т. д. Наиболее распространенным способом вывода является печать результатов на широкой бумажной ленте.

Многие ЭВМ оснащены графопостроителями, позволяющими выдавать с машины результаты в виде графиков.

Все большее распространение получают дисплеи — устройства, имеющие телевизионный экран и клавиатуру. Дисплеи служат как для вывода, так и для ввода информации. Установленные на некотором от ЭВМ расстоянии и соединенные с ней кабельной линией, такие устройства называются терминалами.

1.4. Путь от постановки задачи до ее решения на ЭВМ

Проследим путь от постановки задачи до ее решения на примере нахождения корней квадратного уравнения.

П о с т а н о в к а з а д а ч и. Решить квадратное уравнение

$$1,2381x^2 - 3,2651x + 8,121 = 0.$$

В ы б о р а л г о р и т м а. Прежде всего надо выбрать метод решения задачи на ЭВМ, затем записать спо-

соб решения (алгоритм) в виде последовательных шагов (схемы решения).

1. Ввести в память числа 1,2381; —3,2651; 8,121.
2. Вычислить дискриминант

$$D = (-3,2651)^2 - 4 \cdot 1,2381 \cdot 8,121.$$

3. Проверить условие $D \geq 0$. Если $D \geq 0$, то перейти к п. 4, если $D < 0$, то — к п. 7.

4. Вычислить корни уравнения по формулам

$$x_1 = \frac{-(-3,2651) + \sqrt{D}}{2 \cdot 1,2381}, \quad x_2 = \frac{-(-3,2651) - \sqrt{D}}{2 \cdot 1,2381}.$$

5. Выдать значения корней на печать.

6. Окончить работу.

7. Отпечатать текст: «Уравнение не имеет действительных корней».

8. Окончить работу.

С о с т а в л е н и е п р о г р а м м ы. Итак, выбор алгоритма и составление схемы — первый этап программирования.

Для того чтобы машина могла реализовать выбранный алгоритм, нужно представить каждый шаг схемы в виде последовательности отдельных действий. Сделаем это, например, для п. 2.

1. Умножить — 3,2651 на —3,2651 (вычислить b^2).
2. Занести произведение в память.
3. Умножить 4 на 1,2381 (вычислить $4a$).
4. Результат умножить на 8,121 ($4ac$).
5. Вычесть из b^2 произведение $4ac$ (получить D).
6. Занести D в память ЭВМ.

Каждое такое действие представляется в виде машинной команды. *Команда* — это числовой код, в котором содержится информация о том, какую операцию и над какими числами должна выполнить машина.

Для каждого типа ЭВМ существует своя система кодовых обозначений машинных команд. В этих обозначениях и должны быть представлены действия, какие следует выполнить машине при решении поставленной задачи. Программу, состоящую из машинных команд, называют *машинной программой* или *программой в кодах*.

П р и м е р. На ЭВМ БЭСМ-4 вычисление произведения $b \cdot b$ может быть записано в виде машинной команды 05 0003 0003 0004. Здесь 05 обозначает операцию умножения; 0003 — номер ячейки, в которой хранится число b ,

0004 — номер ячейки, куда будет послано произведение $b \cdot b$.

На БЭСМ-6 эта задача решается тремя машинными командами:

```
00 010 0003
00 017 0003
00 000 0004
```

Здесь 010 0003 — считывание числа из ячейки 0003; 017 0003 — умножение на число, содержащееся в ячейке 0003; последняя команда — засылка результата в ячейку 0004.

Как видно из приведенного примера, машинная программа представляет собой набор числовых кодов, записанных в соответствии с системой команд машины. Команды, реализующие вычисления по одной и той же формуле, по-разному выглядят для ЭВМ разного типа. Поэтому для решения одной и той же задачи на разного типа машинах надо составлять разные программы.

Кодирование программы. Программа пробивается на устройстве подготовки данных (УПД) или набирается на клавиатуре дисплея и сразу попадает во внешнюю память машины.

Отладка программы. Как правило, при первом запуске программы получается неверный результат, так как и в алгоритме, и при составлении схемы и, особенно, при кодировании и пробивке могут быть допущены ошибки. Поэтому следующим этапом работы является отладка — выявление и исправление ошибок. Этот этап часто занимает больше времени, чем все предыдущие.

Остановимся подробнее на этом этапе. Ошибки бывают следующих видов.

А) *Ошибки, обнаруживаемые транслятором.*

Сообщение о наличии и характере таких ошибок (диагностику) выдает программа-транслятор (см. п. 1.7).

Пример. Если в программе встретится формула $A = (B + C * (D + F))$, транслятор выдаст сообщение «НЕ-ЗАКРЫТЫЕ СКОБКИ». Нарушение правил языка ведет к появлению ошибок при трансляции.

Б) *Ошибки при счете.* На подобного рода ошибки программисту указывает система.

Пример. $A = 0$
 $B = C / A$

Попытка разделить на нуль приведет к диагностике «ДЕЛЕНИЕ НА НУЛЬ» и к прекращению счета данной задачи.

В) *Ошибки в алгоритме либо в начальных данных.* Такие ошибки ЭВМ обнаружить не может, поэтому она выполнит программу и выдаст ошибочный результат. Чтобы обнаружить подобные ошибки, необходимо составить тест. Тестирование представляет собой выполнение программы при таких начальных данных, для которых ответ известен заранее.

П р и м е р. Пусть составлена программа решения квадратного уравнения

$$ax^2 + bx + c = 0.$$

Чтобы проверить правильность ее работы, можно взять $a=0,1$; $b=0,2$; $c=0,1$, т. е. решить уравнение

$$0,1x^2 + 0,2x + 0,1 = 0.$$

Если программа выдаст $x_1=-1$, $x_2=-1$, то по ней можно считать. Если результат будет иной, надо искать ошибку.

О методах отладки см. также п. 2.1.14.

Практика показывает, что процесс отладки — обычно самый длительный среди всех остальных этапов решения задачи. Он занимает больше всего времени программиста, а часто — и времени ЭВМ. Поэтому ускорение процесса отладки ведет к повышению эффективности работы программиста и экономии машинного времени.

Одним из самых простых приемов ускорения отладки является *тщательность проверки* на каждом из предыдущих этапов решения задачи. Существуют и системные средства: на каждой ЭВМ есть специальные *программы отладки*.

К сожалению, неопытные программисты часто спешат выдать программу-полуфабрикат за готовую продукцию, т. е. торопятся объявить свою программу уже отлаженной. Это ведет к появлению неверных расчетов при работе такой программы.

Если эта программа учебная, то пострадает лишь сам учащийся. Гораздо опасней, если по недоотлаженной программе идет производственный счет. В результате могут появиться (и появляются) неверные технические и научные расчеты.

Каждый программист должен ясно понимать свою ответственность и самым тщательным образом отлаживать программы.

Счет по отлаженной программе — последний этап решения задачи на ЭВМ.

1.5. Некоторые сведения о чтении и записи информации на ЭВМ

Пусть вычисление по формуле $c = a \cdot b$ записано в виде команды 05 0007 0003 0004. Выполняя эту команду, машина извлекает (считывает) числа из ячеек с номерами 0007 и 0003, пересылает их в арифметическое устройство, там перемножает и результат засылает в ячейку 0004. Таким образом, в процессе выполнения команды происходит извлечение информации из памяти и засылка ее в память.

Возникает вопрос, а что происходит с содержимым ячейки после извлечения из нее информации? Обращаем внимание читателя на следующее свойство машинной памяти: *на ЭВМ всех типов содержимое ячеек после считывания информации не меняется.*

Операция вывода чисел из машины для печати или перфорации также не меняет состояния ячеек, содержащих эти числа.

При записи (засылке) числа в память старое содержимое соответствующей ячейки заменяется новым. Например, если до выполнения команды 05 0007 0003 0004 в ячейке 0004 было какое-либо число, то после ее выполнения старое содержимое этой ячейки сотрется и заменится новым — произведением чисел, находящихся в ячейках 0007 и 0003.

1.6. Всегда ли $2 \times 2 = 4$?

Важной особенностью ЭВМ является то, что она может работать не с любыми числами. Для каждого типа ЭВМ существует верхний и нижний предел допустимых значений чисел (по модулю). Если в результате выполнения операции получается число, превосходящее максимально допустимое, то дальнейшее выполнение программы обычно прекращается. Если же результат оказывается меньше минимально допустимого, то он заменяется нулем (иногда называемым машинным нулем). Напри-

мер, на БЭСМ-6 диапазон изменения чисел (по модулю) — от 10^{-19} до 10^{19} , а на ЕС ЭВМ — от 10^{-78} до 10^{76} .

Другой особенностью ЭВМ является то, что она выполняет действия, вообще говоря, не точно, а приближенно. Точность результата арифметических операций зависит от типа ЭВМ. Так, на БЭСМ-6 эта точность равна 12 значащим цифрам, а на ЕС ЭВМ — 7. Вычисляя на машине $a = (2/3 \cdot 3) \cdot 2$, мы не вправе рассчитывать на получение точного результата 4, так как при вычислении дроби $2/3$ будет получено приближенное значение (с недостатком либо с избытком, в зависимости от типа ЭВМ). Поэтому после умножения этого значения на 3 точный результат 2 может не получиться. Таким образом, при вычислении на ЭВМ равенство $2 \times 2 = 4$ не всегда справедливо.

Однако не следует думать, что равенство всегда несправедливо. Так, $(5-3) \cdot (10/5)$ дает в результате точно 4. Укажем некоторые условия, необходимые для получения точного результата.

Пусть M — максимальное по модулю целое число, допустимое для ЭВМ данного типа (например, для БЭСМ-6 $M = 2^{40} - 1 \approx 10^{12}$). Если a и b — целые числа ($|a| \leq M$ и $|b| \leq M$), то $a+b$, $a-b$ и $a \cdot b$ будут вычислены точно при условии, что результат по модулю не превосходит M . Для получения точного значения частного a/b необходимо еще, чтобы a делилось на b нацело.

При выполнении операций над дробными числами результат обычно получается приближенным.

Возможна и обратная картина, когда два неравных результата оказываются равными при вычислении их на ЭВМ. Например, вычисляя на ЕС ЭВМ значения выражений $x + x^3/2$ и $x + x^3$ при $x = 10^{-5}$, мы получим одинаковые числа. Это объясняется тем, что разность между этими выражениями, равная $x^3/2$, будет меньше каждого из них примерно в 10^{-10} раз. А поскольку точность вычислений на ЕС ЭВМ составляет всего 7 значащих цифр, то эта разность слишком мала, чтобы быть «замеченной» машиной.

Такие нарушения соотношений «равно» — «не равно» могут произойти при работе с числами достаточно большими либо малыми по абсолютной величине. В подобных случаях полезно преобразовать формулы так, чтобы уменьшить возможность получения машинного нуля либо слишком большого числа. Некоторые рекомендации даны нами в гл. II, § 2.3.

1.7. От машинных команд — к алгоритмическим языкам

В первые годы развития программирования человек записывал алгоритмы решения задач в виде последовательности машинных команд. Такой способ имел ряд недостатков: низкой была производительность труда программиста, большие усилия тратились на трудоемкую и утомительную работу — кодирование алгоритма числами, часто допускались ошибки, поиск и устранение которых отнимали много времени. Кроме того, программа решения какой-либо задачи, составленная для одной машины, совершенно не годилась для ЭВМ другого типа.

Важным событием, позволившим сделать большой шаг вперед в области программирования, явилось создание *алгоритмических языков* (или *языков программирования*). Описание алгоритмов на таких языках допускает использование некоторых слов, букв, знаков +, — и т. д. При этом не требуется вручную переводить программу в машинные команды — процесс кодировки осуществляется автоматически самой ЭВМ при помощи программы-транслятора (от английского слова translation — перевод, преобразование).

Транслятор — это специальная программа, которая производит преобразование записи алгоритма с алгоритмического языка, как правило, в последовательность машинных команд. Разработка транслятора требует больших затрат. Обычно для создания транслятора необходима работа коллектива квалифицированных программистов в течение нескольких лет. Объем транслятора обычно составляет тысячи (иногда десятки тысяч) машинных команд. Однако затраты на разработку транслятора затем окупаются, так как благодаря ему значительно увеличивается производительность труда программистов.

Итак, если для ЭВМ данного типа создан транслятор, преобразующий запись алгоритма с языка программирования в машинную программу, то процесс составления программы сводится к записи алгоритма в специальной символической форме, в соответствии с правилами данного языка.

Первые трансляторы, созданные в середине 50-х годов, требовали подробного описания всех производимых действий. Например, программа вычисления дискрими-

нанта квадратного уравнения $ax^2+bx+c=0$ выглядела примерно так:

1. Ввести a, b, c .
2. $r_1=4 \cdot a$.
3. $r_2=r_1 \cdot c$.
4. $r_3=b \cdot b$.
5. $D=r_3-r_2$.

Здесь в символическом виде записаны команды, которые должна выполнить машина. Каждая такая «символическая» команда транслируется в одну машинную команду. Подобные языки получили название автокодов. Эти языки применяются и теперь при написании программ, требующих учета специфики данной ЭВМ.

В дальнейшем были созданы трансляторы, допускавшие запись в виде сложных формул, т. е. без разбиения их на отдельные арифметические действия. Программа предыдущей задачи уже могла быть такой:

1. Ввести a, b, c .
2. $D=b \cdot b-4 \cdot a \cdot c$.

Один из трансляторов, который преобразовывал формулы в машинные команды, был создан в 1956 г. в США. Этот транслятор «понимал» формулы, записанные в форме, очень сходной с обычной математической записью. Соответствующий язык программирования получил название FORTRAN — от английских слов FORmula TRANslation. В дальнейшем мы будем пользоваться русским написанием — *фортран*.

Программа на языке фортран состоит уже не из машинных команд, а из более крупных частей, называемых операторами. *Оператор* — это либо математическая формула, написанная в соответствии с правилами алгоритмического языка, либо обозначение какого-нибудь действия машины, например вывода чисел на печать, либо указание транслятору (например, отвести 100 ячеек памяти). Ниже мы более подробно познакомим читателя с фортраном как с одним из наиболее простых и распространенных универсальных алгоритмических языков. Трансляторами с фортрана снабжены ЭВМ почти всех типов.

1.8. Какие языки «знает» ЭВМ?

После появления языка фортран было предложено еще немало языков программирования. Среди них наибольшее распространение получили *алгол*, *кобол*, *PL/1* и *паскаль* [6].

Для чего понадобилось такое множество языков? Ведь фортран — универсальный язык, пригодный для работы на всех типах ЭВМ.

Действительно, на таком языке, как фортран, можно составлять программы для решения самых разнообразных задач. Однако некоторые задачи будут решаться по этим программам недостаточно эффективно. Первоначально на ЭВМ в основном решались задачи вычислительного характера, где основная часть работы машины сводилась к расчетам по формулам. Для составления программ решения задач такого типа и рекомендуются языки фортран или алгол.

Задачи экономического характера имеют уже иную специфику. Например, при расчете на ЭВМ заработной платы важно не только начислить зарплату, но и получить с машины готовую ведомость. Для решения подобного рода проблем языки фортран и алгол не очень удобны. Особенности экономических задач были учтены при создании языка кобол.

Вообще для каждого класса задач обычно создается один или несколько языков программирования. Предпринимаются попытки создания таких языков, которые содержали бы средства для эффективного решения самых разных задач (PL/I, алгол-68).

Для ЭВМ каждого типа разрабатываются трансляторы с нескольких наиболее распространенных языков программирования. Это делается для того, чтобы каждый, желающий воспользоваться услугами данной машины, мог программировать на том языке, который более удобен для решения поставленной задачи.

1.9. Элементы языка фортран

Как и любой язык, фортран имеет свой алфавит, т. е. набор символов, из которых составляются слова и предложения. Алфавит фортрана составляют:

- *заглавные буквы английского алфавита от A до Z;*
- *арабские цифры от 0 до 9;*
- *знаки + — * / . , = () '.*

Наряду с указанными символами в программе можно использовать пробелы, которые обычно не влияют на ее содержание и служат для достижения большей наглядности. В тех случаях, когда надо указать точное расположение пробелов, мы будем использовать символ `—`.

В фортране для БЭСМ-6 допускаются также заглавные буквы русского алфавита и знак \$ (или \diamond). В фортране ЕС ЭВМ также используется знак \$ (обозначаемый как \square), который имеет смысл буквы и, кроме того, знак &. Русские буквы и знаки \$ и & мы использовать не будем.

1.9.1. Константы языка фортран. Константы фортрана являются аналогами чисел, используемых при записи формул. Каждая константа, указанная в программе, заносится транслятором в свою ячейку и находится там в течение работы данной программы. Константы могут быть нескольких *типов*: вещественные, целые и др.

Вещественные константы — это десятичные дроби и, в частности, целые числа, записанные в виде десятичных дробей. Каждая десятичная дробь содержит, как известно, запятую, отделяющую ее целую часть от дробной. В фортране вместо запятой используется точка, так что числу 3,14 будет соответствовать константа 3.14. Таким образом, каждая вещественная константа такого вида содержит точку. Если целая часть константы равна нулю, то она может быть опущена. Например, числу 0,18 можно поставить в соответствие константу .18 или 0.18. Аналогично нулевая дробная часть может быть опущена; например, 35. означает то же самое, что и 35.0.

При записи чисел часто используют степень десяти, например $2,5 \cdot 10^9$. В фортране предусмотрена запись соответствующих констант с использованием буквы E в качестве указателя основания 10. При этом точка (знак умножения) опускается, а показатель степени записывается справа от буквы E. Так число $2,5 \cdot 10^9$ будет представлено на фортране константой 2.5E9.

Пример. Число 5,5 можно записать как 5.5; 0.55E1; 55.E—1 и т. д.; 0,002 — как 0.002; .002; 2.E—3 и т. д.

Знак константы (+ или —) указывается слева от нее, причем знак + может быть опущен. После буквы E указывается знак показателя степени, причем знак + также может быть опущен.

Целые константы имеют вид обычных целых чисел. Например, целая константа 15 соответствует целому числу 15 (без точки). Перед целой константой может быть поставлен знак (+ или —), причем знак + может быть опущен.

Одно и то же целое число (например, 5) может быть записано и как вещественная константа (5.) и как целая

(5). При этом в машине они будут представлены поразному.

Множество вещественных и целых констант фортрана ограничено. Это связано с тем, что в ячейку памяти нельзя поместить любое число. Слишком большое (по модулю) число может не «уместиться» в ячейку, а слишком малое будет представлено нулем.

На БЭСМ-6 вещественные константы заключены (по модулю) между числами $2^{-65} \approx 10^{-19}$ и $2^{63} \approx 10^{19}$, а значения целых констант (по модулю) не должны превосходить $2^{40} - 1 \approx 10^{12}$. На ЕС ЭВМ диапазон изменения вещественных констант (по модулю) от $16^{-65} \approx 10^{-78}$ до $16^{63} \approx 10^{76}$, а целых — от 0 до $2^{31} - 1 \approx 2 \cdot 10^9$.

У п р а ж н е н и е 1. Записать на фортране вещественные константы, соответствующие числам 78,05; 0,000 000 2; —3800 000; 18,905; -10^{-10} .

1.9.2. Переменные величины. Величина, значение которой может меняться в процессе работы программы, называется *переменной*. Каждой переменной величине транслятор отводит ячейку памяти или две (см. 2.1). В этой ячейке будет находиться текущее значение переменной. При вычислении нового значения переменной производится изменение содержимого ячейки памяти, в которой находилось прежнее значение этой переменной. Это новое значение переменной становится текущим до получения следующего ее значения.

Каждая переменная должна быть обозначена. В научной литературе принято обозначать переменные величины одной буквой, например v , a , t , α , M , a_{11} , b . При этом используются буквы латинские, греческие (иногда русские и готические), большие и малые. Кроме того, буквы могут снабжаться различными значками (штрихи, тильды, звездочки, черточки и т. п.) и индексами. В фортране используется всего 26 букв, и поэтому для обозначения переменных применяют сочетания букв и цифр. Такие сочетания напоминают имена в обычном смысле и называются символическими именами, или просто *именами*. Вместо термина «имя» используют также термин «наименование», или «идентификатор».

Итак, каждая переменная должна иметь свой идентификатор, или имя. *Идентификатор всегда начинается с буквы, состоит только из букв и цифр и содержит не более 6 символов.* Например A, B1, GAMMA, DUBNA,

MOSCOW являются идентификаторами, а 2C, A+B, BESM-6, X**2 — не являются.

Переменные, как и константы, могут быть нескольких типов: вещественные, целые и др. В фортране принято следующее соглашение: тип переменной определяется первой буквой ее имени. Если имя переменной начинается с одной из букв I, J, K, L, M, N, то переменная имеет тип «целый». Если имя переменной начинается с любой другой буквы, то переменная имеет тип «вещественный». Другие способы описания типа переменной будут рассмотрены ниже (см. п. 2.1.5).

Пример. Переменные I1, KARL12, L3410, MASTER, J1NR — целого типа, а переменные A11, SKAM20, DUBNA, TETA, PSI — вещественного типа.

Упражнение 2. Какие из приведенных комбинаций символов могут быть именами переменных? Целых переменных? DVI; 7A3C; C3A7; L+B; LB; KORT; I7C; C71; C/1; β 3.

1.9.3. Арифметическое выражение. Итак, мы познакомились с константами и переменными фортрана. Теперь перейдем к записи действий над ними.

В фортране используются следующие знаки арифметических операций:

- + сложение,
- вычитание,
- * умножение,
- / деление,
- ** возведение в степень.

Аналогом математической формулы на языке фортран является *арифметическое выражение*. Оно может содержать числовые константы, переменные, функции (например SIN(X)), знаки арифметических операций и круглые скобки.

Пример. Математической формуле $b^2 - 4ac$ соответствует арифметическое выражение

$B**2 - 4.*A*C$.

Порядок действий при вычислении значения арифметического выражения совпадает с принятым в математике, но операции одинакового старшинства выполняются строго в порядке написания (слева направо).

Отличия арифметического выражения от формулы состоят в следующем.

1. *Арифметическое выражение записывается линейно, т. е. в строку.*

Например, формуле $\frac{x}{b}$ соответствует выражение X/V , а формуле B^2 — выражение $B**2$.

2. *Порядок действий однозначно определен в записи арифметического выражения*, в то время как производить вычисления по одной и той же математической формуле можно разными способами.

Например, существует около двадцати способов вычисления значения $\frac{ab}{cd}$. Вот некоторые из них:

а) вычислить $a \cdot b$, затем $c \cdot d$ и разделить первый результат на второй;

б) вычислить a/c , затем b/d и первый результат умножить на второй;

в) вычислить b/d , разделить на c и умножить на a .

Но арифметическому выражению $A*B/(C*D)$ соответствует только способ а). Если по каким-либо соображениям предпочтительнее способ в), то вид арифметического выражения будет иной: $V/D/C*A$.

3. *Из всех видов скобок используются только круглые*. Например, формула $a\{b+c[d+e(f+g)]\}$ записывается в виде $A*(B+C*(D+E*(F+G)))$.

4. *Запрещено ставить подряд два знака арифметических операций*. Например, недопустима запись $3*-2$. В подобных случаях надо использовать скобки: $3*(-2)$.

5. *Нельзя опускать знак умножения между сомножителями*. Так, если формула $2a$ будет записана как $2A$, то это сочетание истолкуется транслятором как имя переменной, начинающееся с цифры, что недопустимо. В таком случае транслятор выдаст сообщение об ошибке. Хуже обстоит дело, если имя первого сомножителя начинается с буквы (например, ba , записанное как BA). Такое имя допускается правилами фортрана, и транслятор не заметит ошибки.

6. *Результат любой арифметической операции над целыми константами или целыми переменными имеет тип «целый»*. Остановимся подробно на операции деления, так как остальные операции над целыми числами дают в результате всегда целое число.

В фортране принято следующее правило получения результата: при делении целых констант либо переменных друг на друга в частном отбрасываются все знаки после точки и сама точка.

Так, $4/5$ (частное 0.8) дает в результате 0
 $3999/1000$ (частное 3.999) дает 3
— $9/2$ (частное —4.5) дает —4

Приведем пример, указывающий на одну из возможностей внесения в программу ошибки при записи арифметического выражения.

Пр и м е р. Вычислить значение $15 \cdot 3/4$.

Запрограммируем это выражение двумя способами:
а) $15 \cdot 3/4$ и б) $15/4 \cdot 3$.

В первом случае сначала выполнится умножение, а затем деление $45/4$. В частном 11.25 отбросятся десятичные знаки и результат будет 11.

Во втором — сначала вычислится $15/4$ (частное 3.75, результат 3), а затем $3 \cdot 3$. Окончательный результат получится 9. Итак, одно и то же выражение, запрограммированное разными способами, может дать разные результаты.

Подчеркнем, что дробный результат при делении целых констант и целых переменных друг на друга не округляется. Для положительного частного он совпадает с математической функцией $f(x)=[x]$, где $[x]$ — целая часть x ; для отрицательного — нет.

Так, значение выражения $15/4$ на ЭВМ есть 3 и $[15/4]=3$, но значение $-15/4$ есть —3, а $[-15/4]=-4$.

Результат вычисления арифметического выражения имеет тип «целый» тогда и только тогда, когда в это выражение входят *только целые константы и целые переменные*.

Например, результат вычисления $(3 \cdot IR/8 \cdot 9)/100$ имеет тип «целый», а $(3 \cdot IR/8 \cdot 9)/100 + 4$ — тип «вещественный», так как в это выражение входит вещественная константа 4.

1.9.4. Операторы языка фортран. Программа решения какой-либо задачи на ЭВМ представляет собой последовательность конкретных указаний машине: ввести данные, вычислить значение выражения, отпечатать результат и т. д. Каждое отдельное указание фортранной программы называется *оператором*. Оператор, как правило, представляет собой английское слово, сопровождаемое дополнительной информацией в виде переменных и констант. Английское слово указывает, какое действие следует произвести (READ — прочитать, PRINT — отпечатать, RETURN — вернуться и т. д.).

В фортране используется около двадцати английских слов. Если в этих словах программистом будут допущены ошибки, то транслятор, сличающий слова с эталонами, выдаст на печать сообщение об ошибке (диагностике).

Заголовки программы. Первым оператором каждой программы должен быть оператор

PROGRAM name

где name — имя программы. Оно выбирается произвольно, но с учетом правил написания имени на фортране: должно содержать только буквы и цифры, начинаться с буквы и состоять не более чем из 6 символов. Удобно дать программе имя, связанное с названием решаемой задачи. Например, если программируется составление какой-либо таблицы, то можно назвать программу TAB; если вычисляется $\sin x$, то программа может быть названа SINUS и т. д. В программах для ЕС ЭВМ оператор PROGRAM отсутствует.

Арифметический оператор присваивания имеет вид

$X = A$

где X — имя переменной, A — арифметическое выражение.

В ЭВМ оператор присваивания выполняется так: вычисляется значение арифметического выражения и результат заносится в ячейку, соответствующую переменной X . Знак «=», совпадающий по написанию со знаком равенства, означает засылку числового значения в ячейку памяти.

Все переменные, входящие в арифметическое выражение A , должны быть определены к моменту выполнения данного оператора.

Пример. $B = 2. - 3.5$

$X5 = B + 2. * 5$

$CAP = 3. / B + X5$

Здесь значения арифметических выражений имеют тип «вещественный» и присваиваются вещественным переменным. Однако тип переменной может и не совпадать с типом арифметического выражения (например, $X = 5 - 3$). Как быть в этом случае? В фортране действует правило: *в ячейку засылается число в виде, соответствующем типу той переменной, которая стоит в левой части оператора присваивания.*

Например, после выполнения оператора $X=5-3$ в ячейке X будет число 2. (т. е. «вещественное» 2). Если же выполнен оператор $K=5./7.$, то результат должен быть преобразован к типу «целый», т. е. у результата отбрасывается дробная часть. Таким образом, после выполнения оператора $K=5./7.$ в ячейку K занесется «целый» нуль.

Начинающих программистов часто приводят в замешательство операторы вида $A=A+1$. Однако ничего удивительного в этом операторе нет: к числу, находящемуся в ячейке A , прибавляется 1., и результат заносится в ту же самую ячейку.

У п р а ж н е н и е 3. 1. Записать в виде операторов присваивания следующие формулы:

$$\text{а) } x = \frac{b^2 + c^2}{d}, \quad \text{б) } E = \frac{35x + y}{a^3}, \quad \text{в) } A = \frac{3c}{25d}.$$

2. Каким формулам соответствуют следующие арифметические выражения: а) $A/B*(C+D)$; б) $A*(C+D)/B$; в) $A/B/(C+D)$; г) $A/B*C+D$.

У п р а ж н е н и е 4. Какие значения будут присвоены переменным K и A ?

а) $K=1./2.$; б) $K=3./2.+1.2$; в) $K=3/2+1/2$; г) $A=1*7/8$; д) $A=9/8*7$.

О п е р а т о р б е з у с л о в н о г о п е р е х о д а
GO TO. В программе операторы фортрана записываются один под другим, каждый оператор с новой строки. Они выполняются машиной в порядке написания, пока не встретится оператор, указывающий на какой-либо иной порядок их выполнения. Одним из таких операторов является

GO TO п

где p — номер, приписанный слева тому оператору, к выполнению которого должна перейти ЭВМ. Этот номер p называется *меткой оператора*.

Значение p должно лежать в пределах от 1 до 32767 для БЭСМ-6 и от 1 до 99999 для ЕС ЭВМ. Программист может выбирать метки по своему желанию, но надо следить за тем, чтобы в пределах одной программы не было одинаковых меток. Удобно составлять программу так, чтобы от начала к концу программы метки (номера) монотонно возрастали.

Пример. Припишем оператору $A=15.3$ метку 10.
 $10 \text{---} A=15.3$

Если надо, «перескочив» несколько операторов программы, выполнить именно этот, то в программе запишем оператор **GO TO 10**.

Переход к выполнению оператора с меткой n часто называют *передачей управления оператору с меткой n* .

Пример. Пусть задана последовательность операторов

```
B=15.3
D=8.6
GO TO 123
100 B=-1.2
    D=2.2
123 A=B+D
```

После выполнения первых трех операторов управление передается оператору с меткой 123, поэтому в ячейке A будет число $23.9=15.3+8.6$.

Упражнение 5. Написать такой оператор перед всеми операторами предыдущего примера, чтобы в ячейке A был получен результат 1.

Операторы END и PRINT. Последним оператором каждой программы должен быть оператор **END**. Пользуясь оператором **PROGRAM**, операторами присваивания и **END**, уже можно составлять программы. Например, запишем программу вычисления дискриминанта квадратного уравнения $1,32x^2+2,75x-3,45=0$.

```
PROGRAM DISKR
A=1.32
B=2.75
C=-3.45
D=B**2-4.*A*C
END
```

Такая программа не представляет интереса, так как результат D , вычисленный машиной, программисту сообщен не будет.

Для вывода информации из ЭВМ на печать служит оператор

```
PRINT n, D
```

Здесь D — переменная, значение которой следует напе-

чатать, а *p* — метка оператора **FORMAT**, в сочетании с которым работает **PRINT**.

В операторе **PRINT** указывается, значения каких переменных следует напечатать, а в операторе **FORMAT** содержится информация о том, как расположить эти числа на бумаге, сколько знаков сохранить в числе, каким текстом сопроводить выводимые числа. Например, можно записать оператор **FORMAT** так, что вычисленное значение дискриминанта *D* отпечатается так:

ДИСКРИМИНАНТ УРАВНЕНИЯ $1.32 * X^{**}2 +$
 $+ 2.75 * X - 3.45 = 0$
ЕСТЬ $D = 25.78$

Программист заранее должен продумать, в каком виде он хочет получить результат. Полезно знать, что страница широкоформатной печати может содержать до 66 строк по 144 символа в каждой (лучше использовать не более 120).

Каждому месту символа на бумаге соответствует своя позиция печатающего устройства. Поэтому говорят, что текст, состоящий из пяти символов, занимает 5 позиций, трехзначная целая константа занимает 3 позиции, а трехзначная вещественная — 4, так как точка, содержащаяся в записи вещественного числа (283., 5.14, и т. д.), является равноправным символом. Не рекомендуется занимать при печати первую позицию, так как могут возникнуть незапланированные режимы работы печатающего устройства.

О п е р а т о р **FORMAT**. Общий вид:

p **FORMAT** (список спецификаций)

Здесь *p* — метка, а в скобках содержится информация о том, в каком виде печатать и как размещать на бумаге выдаваемые результаты.

Если результат имеет тип «целый», то используют спецификацию *I**m*. Здесь *I* — признак типа «целый», а *m* — число позиций на бумаге, отводимых для печати результата (включая позицию для знака минус, если он есть). При этом младшая цифра результата всегда занимает самую правую из *m* позиций. Если для результата требуется меньше позиций, чем *m*, то левые позиции сверх необходимых будут пробелами. Например, если значение выводимой переменной равно —150, то в операторе **FORMAT** достаточно указать спецификацию *I*4. Если же взять спецификацию *I*7, то на бумаге слева от —150 бу-

дет 3 пробела. Если же указать меньше позиций, чем требуется, то отпечатаются только младшие цифры результата.

Если результат имеет тип «вещественный», то используется спецификация Fm.p. Здесь F — признак типа «вещественный», m — общее число позиций на бумаге, отведенных для печати результата, а p — число позиций, отведенных под дробную часть.

Например, для печати результата —15,347 можно использовать спецификацию F7.3. Здесь 7 — общее число позиций, предназначенных для размещения результата (включая позиции под знак минус и точку), 3 — число позиций для дробной части результата. Если отпечатать —15,347 по спецификации F10.3, то 3 позиции слева будут пустыми. Если указать m меньше, чем требуется для размещения данного результата, то на печать в самой левой из m позиций будет выведен символ *.

Например, выводя число 3,14 по спецификации F3.2, получим на печати *14.

Значения разных вещественных переменных можно печатать по одинаковым спецификациям, выбирая соответствующие m и p.

У п р а ж н е н и е 6. Записать спецификации вывода чисел 9,8; —2,128; 103,14; 15; —728; —25.5.

При выводе по спецификациям Im и Fm.p надо знать примерный порядок выводимых чисел, чтобы правильно задать m и p. Если же оценить порядок результата трудно, то можно взять для целой переменной спецификацию I14 (БЭСМ-6) либо I11 (ЕС ЭВМ), а для вещественной воспользоваться спецификацией Em.p. Здесь m — общее число позиций, отводимых под весь результат, а p — под его дробную часть. По этой спецификации выводится p либо p+1 значащих цифр (в зависимости от типа ЭВМ). Значение m берут таким, чтобы $m \geq p+7$.

П р и м е р. Число —1258,687 при выводе по спецификации E12.3 будет выглядеть на БЭСМ-6 как —1.259+03, на ЕС ЭВМ как —0.126E—04.

Как видно из примера, на БЭСМ-6 слева от точки выводится одна значащая цифра, а порядок занимает 3 позиции. На ЕС ЭВМ все p значащих цифр располагаются справа от точки и порядок занимает 4 позиции.

Если необходимо отпечатать значения нескольких переменных, то их имена перечисляются в операторе

PRINT (список переменных)

Например, 4 переменных A, B, C, D выводятся на печать оператором

```
PRINT—5, A, B, C, D.
```

В операторе FORMAT, соответствующем данному PRINT, указываются 4 спецификации вывода (по одной для каждой переменной). Например,

```
PRINT 5, A, B, C, D  
5 FORMAT (E8.3, F12.7, F5.1, F9.4)
```

Часто для вывода нескольких переменных используют одну и ту же спецификацию. Например, переменные A, B, C, D можно выводить по спецификации F12.7:

```
PRINT 5, A, B, C, D  
5 FORMAT (F12.7, F12.7, F12.7, F12.7)
```

Здесь F12.7 повторена 4 раза. Чтобы не выписывать многократно одну и ту же спецификацию, перед ней можно записать коэффициент повторения:

```
PRINT 5, A, B, C, D  
5 FORMAT (4F12.7)
```

Если несколько раз повторяется целая группа спецификаций, то коэффициент повторения пишется перед скобками, в которых заключается повторяемая группа. Например, вместо

```
10 FORMAT (F5.3, F2.1, F5.3, F2.1, F5.3, F2.1)
```

можно записать

```
10 FORMAT (3(F5.3, F2.1))
```

У п р а ж н е н и е 7. Записать операторы для вывода переменных A, B и C, содержащих числа 73.25, —37235, 0,01272.

Кроме числовых спецификаций Im, Fm.n и Em.n существуют редакционные спецификации mX, mH, используемые для управления расположением чисел на бумаге и снабжения их комментариями.

Спецификация mX служит для пропуска m позиций, т. е. на бумаге будет m пробелов. Например, пусть A=—15., B=0.45. Эти переменные можно вывести с помощью операторов

```
PRINT 10, A, B  
10 FORMAT (5X, F4.0, 5X, F4.2)
```

На бумаге числа расположатся так:

_____15. _____0.45

Спецификация mH позволяет перенести на бумагу m символов, следующих за буквой H. Тот же результат получится, если эти m символов взять в апострофы.

П р и м е р. Операторы

```
PRINT 10, A, B
10 FORMAT (5X, 2HA=, F4.0, 5X, 2HB=, F4.2)
```

либо

```
PRINT 10, A, B
10 FORMAT (5X,'A=', F4.0, 5X, 'B=', F4.2)
```

отпечатают числа A и B в таком виде:

_____ A=—15. _____ B=0.45

Список спецификаций сыграл следующую роль: 5X — пропущено 5 позиций; 2HA= — на бумагу перенесены два символа после буквы H, т. е. A=; F4.0 — отпечаталось значение A; 5X — пропущено 5 позиций; 2HB= — на бумагу перенесены два символа B=; F4.2 — отпечатано значение B.

Пользуясь только редакционными спецификациями, можно на печать выдавать рисунки (см. примеры в гл. IV).

П р и м е р. Написать программу для вычисления произведения чисел $a=1,5873956$ и $b=3,5$.

```
PROGRAM N2A
A=1.5873956
B=3.5
C=A*B
PRINT 2,C
2 FORMAT (10X, 4HA*B=, F10.8)
END
```

На бумаге с 11-й позиции отпечатается следующее:
A*B=5.55588460

У п р а ж н е н и е 8. 1. Написать программу вычисления площади треугольника по основанию $a=15,7328$ и высоте $h=0,031289$.

2. Написать программу вычисления суммы первых десяти членов геометрической прогрессии с $b_1=1,135619$ и $q=0,999999$.

В фортране имеется несколько символов, управляющих работой устройства печати в том случае, если эти

символы выводятся на печать первыми в какой-либо строке. Подобных управляющих спецификаций в фортране БЭСМ-6 несколько:

- 1N0 — пропуск одной строки;
- 1N1. — переход на новую страницу;
- 1N4 — переход на следующую половину страницы;
- 1N5 — переход на следующую треть страницы;
- 1N6 — переход на следующую четверть страницы;
- 1N+ — печать без перехода на новую строку, т. е. наложение символов новой строки на символы старой;
- 1NS — блокировка автоматического перехода на новую страницу во время печати текста.

При печати происходит автоматический перевод страницы после определенного числа строк. При этом между страницами получается зазор. Если на печать выводится график либо рисунок, то часто этот зазор искажает картину. В таких случаях в начале каждой строки выводят символ S.

Подчеркнем, что символы 0, 1, 4, 5, 6, +, S управляют работой печатающего устройства только тогда, когда они являются первыми символами в строке. В этом случае на печать сами они не выводятся.

Программист должен помнить, что *вышеупомянутые символы всегда управляют печатью, если они выводятся первыми в строке*. Поэтому, если с первой позиции выводится число, начинающееся с 0, 1, 4, 5, 6 либо со знака +, то устройство печати обязательно среагирует на выводимый символ (например, пропустит часть страницы).

В таких случаях неопытный программист утверждает, что «печать сломалась».

Чтобы гарантировать себя от подобных неприятностей, новую строку всегда следует начинать с пробела, т. е. при выводе чисел в списке оператора FORMAT на первом месте должна стоять спецификация nX.

В фортране ЕС всего 3 символа, управляющих работой печати,— это 0, 1 и +. Они означают соответственно пропуск одной строки, переход на новую страницу и печать без перехода на новую строку.

Перфорирование карт. Написанная программа может быть пробита на перфокартах для ввода в ЭВМ. На перфокарте имеется 80 колонок (позиций). Операторы фортрана пробиваются в колонках с 7-й по 72-ю. Колонки 1—5 отведены для метки оператора. В ко-

лонках 73—80 можно размещать любой комментарий, например номер перфокарты. Если оператор фортрана не умещается на одной перфокарте, то его продолжение пробивают на следующей карте, у которой в 6-й позиции ставят признак продолжения — любой символ, кроме нуля. Карт продолжения может быть несколько.

Итак, назначение позиций перфокарты при пробивке фортранной программы следующее:

Позиция	Назначение
1—5	Метки операторов
6	Признак продолжения (например, A или *)
7—72	Оператор фортрана (или его продолжение)
73—80	Комментарий (например, номер перфокарты)

О п е р а т о р READ. Для ввода чисел, используемых в качестве исходных данных, служит оператор READ, который записывается в виде

READ n, A, B, . . . , X

Здесь n — метка оператора FORMAT, который сообщает, как пробиты на перфокартах вводимые числа; A, B, , X — имена переменных, которым присваиваются вводимые значения.

В операторе FORMAT, работающем совместно с READ, используются те же спецификации (Im, Em.n, Fm.n, nX, nH), что и при работе с оператором PRINT.

Пусть, например, коэффициенты квадратного уравнения $Ax^2 + Bx + C = 0$ для программы вычисления дискриминанта пробиты на одной перфокарте так: A — в первых четырех колонках, B — в следующих четырех (с 5-й по 8-ю), C — в следующих пяти (с 9-й по 13-ю). Значения A, B и C соответственно равны 1.32, 2.75 и —.345. Тогда этим числам будут соответствовать спецификации ввода F4.2, F4.2 и F5.3. Для ввода этих чисел служат операторы

READ 5, A, B, C
5 FORMAT (F4.2, F4.2, F5.3)

Если числа A, B и C разместить на перфокарте иначе, то изменятся и спецификации в операторе FORMAT. Например, можно пробить числа так, чтобы они были

отделены друг от друга пробелом: 1.32—2.75— .345—, Пустую позицию можно отнести к предыдущему числу и рассматривать ее как еще одну цифру после точки (равную нулю). Таким образом, оператор FORMAT запишется в виде

```
5 FORMAT (F5.3, F5.3, F5.3)
```

Как и при печати, повторяющиеся спецификации можно не выписывать многократно, а использовать коэффициент повторения:

```
5 FORMAT (3F5.3)
```

Запишем теперь новый вариант программы DISKR1; пусть значения A, B и C пробиты на перфокарте по спецификации F5.3

```
PROGRAM DISKRL  
  READ 2, A, B, C  
2  FORMAT (3F5.3)  
   D=B**2—4.*A*C  
  PRINT 3, D  
3  FORMAT (3X, F5.2)  
  END
```

В рассмотренном примере все вводимые числа пробиты на одной перфокарте. Если же каждое из вводимых чисел будет пробито на отдельной перфокарте, то спецификации этих чисел в операторе FORMAT будут отделены друг от друга не запятой, а косой чертой:

```
5 FORMAT (F5.3/F5.3/F5.3)
```

или

```
5 FORMAT (3(F5.3/))
```

Рассмотрим случай, когда в списке оператора READ больше переменных, чем числовых спецификаций (Im, Fm.n, Em.n) в операторе FORMAT. Например,

```
  READ 2, A, B, C  
2  FORMAT (F5.2, F4.2)
```

В этом случае предполагается, что два числа пробиты на одной карте по спецификациям F5.2 и F4.2, а третье — на следующей карте по спецификации F5.2: список спецификаций оператора FORMAT просматривается заново, начиная с последней открывающей скобки. В примере

```
  READ 3, A, B, C, D, E, F  
3  FORMAT (F5.2, 2(F3.2, F7.3))
```

пять чисел должны быть пробиты на одной карте по спецификациям F5.2, F3.2, F7.3, F3.2, F7.3, а шестое число — на следующей карте по спецификации F3.2. Для того чтобы ввести числа A , B и C , пробитые на отдельных перфокартах по одной и той же спецификации F5.3, достаточно записать

```
READ 10, A, B, C
10 FORMAT (F5.3)
```

При пробивке чисел на перфокартах можно занимать все 80 позиций. В операторе FORMAT надо предусмотреть переход на новую карту всякий раз, как только будут исчерпаны 80 позиций очередной карты. Например, оператор

```
25 FORMAT (5F15.6, I8)
```

требует занять на одной карте более 80 позиций ($5 \cdot 15 + 8 = 83$) и, следовательно, является ошибочным.

**Э л е м е н т а р н ы е м а т е м а т и ч е с к и е
ф у н к ц и и.** Для вычисления таких функций, как $\sin x$, $\ln x$, $|x|$, \sqrt{x} и т. п., имеются заранее составленные стандартные программы. Для того чтобы воспользоваться такой программой, достаточно указать имя соответствующей функции и в скобках записать значение аргумента в виде произвольного арифметического выражения, например $\text{SIN}(3.14/6)$ либо $\text{COS}(X)$.

Приведем таблицу имен наиболее употребительных функций (см. также Приложение 2):

Функция	Запись на фортране	Функция	Запись на фортране
$\sin x$	SIN (X)	\sqrt{x}	SQRT (X)
$\cos x$	COS (X)	$ x $	ABS (X)
$\ln x$	ALOG (X)	e^x	EXP (X)
$\lg x$	ALOG 10 (X)		

Для тригонометрических функций угол должен быть задан в радианах.

П р и м е р. Запрограммируем на фортране формулу $y = ab\sqrt{a} + \ln \sin(t + x)$:

$$Y = A*B*SQRT(A) + ALOG(SIN(T + X))$$

Пример. Составить программу для вычисления длины окружности L , если задана S — площадь соответствующего круга. Пусть $S < 400$ и это число пробито на перфокарте с двумя знаками после точки.

Запишем формулу, выражающую L через S :

$$L = 2\pi R = 2\pi \sqrt{\frac{S}{\pi}} = 2\sqrt{\pi S}.$$

Составляем программу.

```

PROGRAM L
  READ 10, S
10 FORMAT (F6.2)
  A=2.*SQRT (3.14*S)
  PRINT 20, A
20 FORMAT (10X, 17НДЛИНА ОКРУЖНОСТИ=,
  *F5.2)
  END
  
```

Для ввода S взята спецификация F6.2, так как это число может занимать шесть позиций, две из которых — после точки. После вычисления длины окружности A машина отпечатает результат. Первые две спецификации вывода редакционные: 10X — для отступления на десять позиций вправо, а 17Н . . . выводит на печать комментарий «длина окружности =». Затем будет напечатано значение A по спецификации F5.2, взятой по следующим соображениям: если $S < 400$, то $\sqrt{S} < 20$, $A = 2\sqrt{\pi} \cdot \sqrt{S} < 80$, т. е. целая часть A содержит не более 2-х цифр. Если порядок результата прикинуть не так просто, то следует использовать спецификацию E_m.n.

В приведенном примере длина окружности была обозначена через A , а не общепринятым L , так как имя L на фортране обозначает целую переменную, и поэтому после вычисления $L = 2\sqrt{\pi S}$ десятичные знаки были бы отброшены.

У п р а ж н е н и е 9. 1. Составить программу вычисления корней квадратного уравнения, если коэффициенты a , b и c пробиты на одной перфокарте, лежат в диапазоне $[-10, 10]$ и заданы с двумя десятичными знаками после точки. Числа a , b и c предполагаются заданными так, что $b^2 - 4ac \geq 0$.

2. Составить программу вычисления площади и периметра треугольника по двум сторонам a , b и углу α между ними. Значения

a , b и α (в радианах) пробиты на одной перфокарте с 4 знаками после точки, причем a и b не превышают 100.

3. Записать на фортране формулы:

$$a) y = \sqrt{ab} \sin a \lg b + |\sin a| \sqrt{|\sin^2 a - \cos^2 b|},$$

$$б) y = \frac{a}{2\sqrt{b}\sqrt{a}}.$$

О п е р а т о р CONTINUE. Этот оператор является носителем метки и никакой другой функции не выполняет. Его общий вид

m CONTINUE

где m — метка.

П р и м е р. Пусть надо составить программу так, чтобы выход из нее (окончание работы) осуществлялся после операторов, расположенных в разных местах программы. Тогда в конце программы можно записать оператор 5 CONTINUE, а после каждого из упомянутых трех операторов поставить GO TO 5.

PROGRAM C

GO TO 5

GO TO 5

GO TO 5

5 CONTINUE
END

Особенно удобно использовать оператор CONTINUE в процессе написания программы, когда указывается переход на участок программы, еще не написанный программистом. Пусть, например, в программе указан переход GO TO 17, но мы еще не записали операторы, соответствующие дальнейшему ходу решения. Написав 17 CONTINUE, мы, во-первых, можем получить в окончательном виде начало программы (т. е. указать в GO TO конкретную метку) и, во-вторых, будем гарантированы, что метка 17 не будет пропущена в программе.

Оператор CONTINUE удобен и тем, что позволяет при необходимости совместить функции нескольких операторов. Пусть, например, в программе написаны операторы GO TO 17 и GO TO 36, а впоследствии стало ясно, что операторы с метками 17 и 36 должны иметь одно и то

же назначение. Тогда можно расположить подряд два оператора CONTINUE

17 CONTINUE

36 CONTINUE

...

Другие случаи использования этого оператора будут представлены в последующих программах.

О п е р а т о р у с л о в н о г о п е р е х о д а IF. При решении квадратного уравнения $ax^2+bx+c=0$ могут представиться два случая: дискриминант $D=b^2-4ac \geq 0$ либо $b^2-4ac < 0$. Прежде чем извлекать корень из дискриминанта, необходимо исследовать его знак, и в зависимости от знака поступить одним из двух

способов: либо вычислить корни $x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$,

либо объявить, что данное уравнение вещественных корней не имеет.

В языке фортран имеется оператор условного перехода IF, который осуществляет выбор дальнейшего действия в зависимости от некоторого условия. Общий вид этого оператора

IF(E) n, m, k

Здесь n, m, k — метки каких-либо операторов программы, E — арифметическое выражение (в частности, имя переменной).

Оператор IF(E) n, m, k работает так: вычисляется значение выражения E и производится переход на оператор с одной из меток n, m и k по следующему правилу:

- 1) если $E < 0$, то — на метку n,
- 2) если $E = 0$, то — на метку m,
- 3) если $E > 0$, то — на метку k.

Например, оператор

IF(X**2-5.)15, 18, 31

производит переход на оператор с меткой 15, если $X**2-5. < 0$, на оператор с меткой 18, если $X**2-5. = 0$, и на оператор с меткой 31, если $X**2-5. > 0$.

В зависимости от смысла условия две метки, указанные в операторе IF, могут совпадать. Например, при решении квадратного уравнения случаи $D > 0$ и $D = 0$ можно объединить, так что оператор IF будет записан

В виде

IF (B**2—4.*A*C) m, k, k

Если же нас интересует только случай равных корней, т. е. $D=0$, то соответствующий оператор IF будет записан так:

IF (B**2—4.*A*C) m, n, m

Пример. Программа решения квадратного уравнения для произвольного значения дискриминанта. Коэффициенты уравнения пробиты на одной карте по спецификации F9.6.

```
PROGRAM SQUEQU
READ 1, A, B, C
1 FORMAT (3F9.6)
D=B**2—4.*A*C
IF (D)2, 3, 3
3 X1=(—B+SQRT (D))/(2.*A)
  X2=(—B—SQRT (D))/(2.*A)
  PRINT 4, X1, X2
4 FORMAT (5X, 3HX1=, F12.6/5X, 3HX2=, F12.6)
  GO TO 6
2 PRINT 5
5 FORMAT (5X, 10HКОРНЕЙ—НЕТ)
6 CONTINUE
END
```

В случае $D \geq 0$ будут отпечатаны значения корней в таком виде:

```
_____ X1=KKKKK.nnnnnn
_____ X2=KKKKK.nnnnnn
```

Здесь K обозначает позиции, отведенные для размещения целой части, считая и знак, n — позиции для дробной части. В случае $D < 0$ будет отпечатан текст

КОРНЕЙ — НЕТ

У п р а ж н е н и е 10. Записать на фортране алгоритм вычисления функции $y=|a|$, не используя функции ABS(X).

Ц и к л ы. В некоторых задачах бывает необходимо выполнить несколько раз одну и ту же последовательность действий.

Пример. Вычислить площади пяти треугольников по трем сторонам, значения которых пробиты по

спецификации F6.3 на пяти соответствующих перфокартах.

В программе для решения такой задачи потребуется пять раз повторить выполнение группы операторов:

```
PROGRAM S5
READ 2, A, B, C
P = (A + B + C)/2.
S = SQRT(P*(P-A)*(P-B)*(P-C))
PRINT 3, S
```

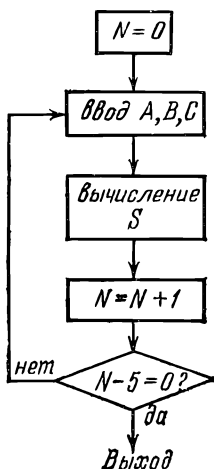
} группа операторов повторяется 5 раз

```
3 FORMAT (5X, 2HS=, F12.3)
2 FORMAT (3F6.3)
```

```
...
READ 2, A, B, C
P = (A + B + C)/2.
S = SQRT (P*(P-A)*(P-B)*(P-C))
PRINT 3, S
END
```

Чтобы упростить написание этой программы, используют следующий алгоритм. Какой-либо переменной N , называемой счетчиком, первоначально присваивают значение 0. После каждого выполнения повторяемой группы операторов к N добавляют 1 и проверяют условие $N=5$. Как только это условие выполнится, процесс заканчивают.

Схема решения задачи (блок-схема):



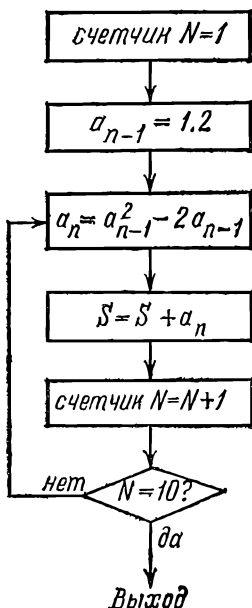
Запись этого алгоритма на фортране:

PROGRAM N6	
N = 0	Очистка счетчика
30 READ 2, A, B, C	Ввод чисел
2 FORMAT (3F6.3)	
P = (A + B + C)/2.	Вычисление полупериметра
S = SQRT(P*(P - A)*(P - B) * (P - C))	Формула Герона
PRINT 3, S	Печать значения S
3 FORMAT (5X, 2HS=, F12.3)	
N = N + 1	Прибавление к счетчику 1
IF (N - 5)30, 4, 4	Проверка: N = 5?
4 CONTINUE	Если N = 5 — конец задачи
END	

Участки программы, выполняемые многократно, называются *циклами*. В примере участок программы от оператора с меткой 30 до IF (N - 5)30, 4, 4 является циклом.

Пример. Вычислить сумму первых десяти членов последовательности $a_n = a_{n-1}^2 - 2a_{n-1}$; $a_1 = 1,2$.

Блок-схема программы:



В счетчик засылается не 0, а 1, так как вычисления в цикле начинаются с a_2 .

Программа на фортране:

PROGRAM N7	Занесение 1 в счетчик
N = 1	
S = 1.2	Начальное значение S
AN = 1.2	$a_1 = 1,2$
1 AN = AN**2 - AN*2	Рекуррентная формула вычисления a_n
S = S + AN	Прибавление очередного a_n к S
N = N + 1	Прибавление к счетчику 1
IF(N-10)1, 2, 2	Проба: N = 10?
2 CONTINUE	Если N = 10,
PRINT 3, S	печатать значения S
3 FORMAT(5X, 2HS=, F12.3)	
END	

Циклом является участок от оператора с меткой 1 до IF(N-10)1, 2, 2.

Пример. Вычислить $K=11!$

Программа на фортране:

PROGRAM FACT	
K = 1	Начальное значение факториала
N = 0	Очистка счетчика
1 N = N + 1	Увеличение счетчика на 1
K = K*N	Вычисление K!
IF(N-11)1, 2, 2	Проба: N = 11?
2 CONTINUE	Если N = 11,
PRINT 3, K	печатать значения K!
3 FORMAT(7X, 2HK=, I10)	
END	

О п е р а т о р ц и к л а DO. Циклы могут быть построены не только с помощью оператора IF. В фортране существует специальный оператор цикла DO. Общий вид оператора

DO n I=m, j, k

где n — метка последнего оператора цикла; I — простая целая переменная — параметр (счетчик) цикла; m —

начальное значение параметра цикла I ; j — верхняя граница I : выход из цикла происходит при $I > j$; k — шаг по значениям параметра цикла при повторении исполнения.

Здесь m , j и k могут быть целыми константами без знака или простыми целыми переменными, принимающими положительные значения.

Повторяется группа операторов, первый из которых следует за DO, а последний имеет метку n .

П р и м е р.

```
DO 5 I=2, 11, 3
A=1.
```

5 CONTINUE

Будут повторены операторы от $A=1.$ до 5 CONTINUE четыре раза: для $I=2$, $I=5$, $I=8$, $I=11$. Если шаг по I равен 1, то оператор DO можно записать в виде

```
DO n I=m, j
```

(k опускается вместе с предшествующей запятой).

Используя оператор цикла DO, можно упростить написание программы FACT:

```
PROGRAM FACT1
K=1
DO 1 L=2, 11 }
K=K*L          ]цикл, повторяемый 10 раз
1 CONTINUE
PRINT 3, K
3 FORMAT (7X, 2HK=, 110)
END
```

П р и м е р. Найти 20-й член последовательности

$$a_n = a_{n-1}^2 + 0,01; \quad a_1 = 1.$$

Два варианта программы на фортране:

<pre>PROGRAMN9 A=1. DO 1 K=2, 20 A=A**2+0.01 1 CONTINUE PRINT 2, A 2 FORMAT (10X, 2HA=, F6.2) END</pre>	<pre>PROGRAM N9A A=1. DO 1 K=2, 20 1 A=A**2+0.01 PRINT 2, A 2 FORMAT (10X, *2HA=, F6.2) END</pre>
---	---

Параметр цикла — целая переменная, т. е. ее имя должно начинаться с одной из букв I, J, K, L, M, N и содержать не более шести букв и цифр. Если в операторе DO n I=m, j, k величина j задана константой, то она не может превосходить определенного значения для каждого типа ЭВМ. Так, для БЭСМ-6 $j \leq 2^{15} - 1$.

Можно использовать циклы в цикле. В этом случае говорят о внешнем и внутреннем цикле, целиком расположенном внутри внешнего. Во внутреннем цикле нельзя использовать параметр внешнего цикла. Например:

```
DO 3 N3=5, 10
```

```
  DO 2 N2=1, 20
```

```
    DO 1 N1=3, 100
```

```
1  CONTINUE
```

```
2  CONTINUE
```

```
3  CONTINUE
```

Внешний цикл по параметру N3 оканчивается оператором 3 CONTINUE. Этот цикл содержит в себе цикл по N2, оканчивающийся оператором 2 CONTINUE. Самый внутренний цикл по параметру N1 оканчивается оператором 1 CONTINUE.

Работает этот фрагмент программы так:

1) выполняются все операторы до 1 CONTINUE, затем полностью заканчивает работу внутренний цикл (N1 меняется от 3 до 100);

2) выполняются операторы до 2 CONTINUE, т. е. 20 раз срабатывает цикл по N2, включающий в себя 98-кратное срабатывание цикла по N1;

3) выполняются операторы до 3 CONTINUE, т. е. 6 раз работает внешний цикл. При этом цикл по N2 повторится $6 \cdot 20$ раз, а цикл по N1 — $6 \cdot 20 \cdot 98$ раз.

Пример. Найти число «счастливых» билетов с номерами от 000000 до 999999 включительно. Билет считается «счастливым», если сумма левых трех цифр номера равна сумме правых трех цифр. Решать задачу будем перебором всех возможных номеров, т. е. варьируя каждую цифру числа от 0 до 9.

Пусть номер записан в виде IJKLMN. Программа решения этой задачи выглядит так:

PROGRAM HAPPY

NC = 0

DO 1 I = 1, 10

DO 1 J = 1, 10

DO 1 K = 1, 10

DO 1 L = 1, 10

DO 1 M = 1, 10

DO 1 N = 1, 10

IF (I+J+K-(L+M+N)) 1, 2, 1

2 NC = NC + 1

1 CONTINUE

PRINT 3, NC

3 FORMAT (10X,

*18НСЧАСТЛИВЫХ_БИЛЕТОВ, I6)

END

Заметим, что значения параметров (I, J, K, L, M, N) меняются от 1 до 10, в то время как цифры соответствующих разрядов числа должны меняться от 0 до 9. Но такой сдвиг не сказывается на результате: лишние 3 единицы суммы $I+J+K$ уничтожаются тремя единицами вычитаемого $(L+M+N)$. В приведенном примере использовано шесть вложенных циклов. Фортран допускает достаточно большое число таких вложений (на БЭСМ-6, например, до 50).

При написании циклов надо придерживаться следующих правил:

1. Входить в цикл можно только через оператор DO.
2. Заканчивать цикл можно любым оператором, кроме IF, GO TO, RETURN, DO, STOP.

3. Внутри данного цикла DO не может оканчиваться внешний по отношению к нему цикл DO.

4. Можно использовать до 50 циклов, вложенных один в другой.

NC — число

«счастливых»

билетов; перебор всех комбинаций в 6-разрядном числе;

I — сотни тысяч

J — десятки тысяч

K — тысячи

L — сотни

M — десятки

N — единицы

Проба: билет «счастливый»?

Если да, то число NC увеличивается на 1

Печать числа

«счастливых»

билетов

5. Последние операторы вложенных циклов могут совпадать.

Пр и м е р.

```
DO 5 I=1, 7, 2
DO 5 K=5, 10, 3
DO 5 L=3, 5
```

5 CONTINUE

6. Значение параметра цикла (начальное и конечное) и шаг нельзя менять внутри цикла.

7. Значение параметра цикла нельзя использовать после завершения цикла.

Выход из цикла может осуществляться по какому-либо условию.

Пр и м е р. Вычислить сумму ряда $S=1-1/2!+1/3!-1/4!+\dots$ с точностью 0,01. Это значит, что процесс надо оборвать на слагаемом, абсолютная величина которого $<0,01$.

```
PROGRAM N11
```

```
S=1.
```

Первоначальные значения S и A

```
A=1.
```

```
EPS=0.01
```

Задается точность вычислений

```
DO 2 I=2, 10
```

Конечное значение параметра (10) заведомо больше, чем требуется

```
A=-A/I
```

При первом прохождении цикла $A=-1/2$, при втором $A=+1/(2\cdot3)$ и т. д.

```
IF(ABS(A)-EPS)
```

Если $|A| \geq EPS$, то процесс не закончен;

```
*1, 2, 2
```

```
2 S=S+A
```

вычисляется сумма ряда; этот оператор—последний оператор цикла: следующим (если $I \leq 10$) выполняется оператор $A=-A/I$

```
1 CONTINUE
```

Выход на этот оператор происходит при условии $|A| < EPS$ либо $I \geq 10$

```
PRINT 3, S
```

Вывод на печать суммы ряда

```
3 FORMAT (10X, 2HS=, F6.3)
```

```
END
```

У п р а ж н е н и е 11.

1. Найти 15-й член последовательности $a_k = \sqrt{a_{k-1}} + 1$, если $a_1=1$.

2. Найти сумму двадцати членов последовательности $a_n = \frac{a_{n-1}^2}{a_{n-1} + 3}$, если $a_1 = 2$.

1.9.5. Массивы переменных. Часто бывает удобно объединять несколько переменных в один массив. Массиву дается имя, например В. Каждая переменная, входящая в этот массив (элемент массива), носит то же имя В, а ее порядковый номер в массиве указывается в скобках и называется индексом. Такие переменные называются переменными с индексом в отличие от простых переменных, не имеющих индекса, т. е. не являющихся элементами массива.

Пусть В — массив, объединяющий 5 переменных, числовые значения которых 1.6, 1.1, 2.2, 1.8, 3.05. Тогда переменная В(3) имеет числовое значение 2.2; В(5) — значение 3.05. Подобные массивы, элементы которых определяются одним индексом, называются одномерными. В этой главе мы будем рассматривать только одномерные массивы.

О п е р а т о р DIMENSION. Чтобы под массив было отведено транслятором определенное число ячеек, после оператора PROGRAM необходимо записать оператор

DIMENSION В(М)

где В — имя массива, М — число элементов в нем. Здесь М — целая константа без знака.

В приведенном примере массив В содержит 5 элементов, поэтому для описания В используют оператор DIMENSION В(5).

Для вывода на печать или для ввода с перфокарт всего массива достаточно указать его имя в операторах PRINT или READ. Например, для вывода массива А, состоящего из 1000 элементов, достаточно записать

PRINT 16, А

Если надо вывести один N-й элемент массива, то надо указать А(N) в операторе PRINT. Например, для вывода 5-го элемента массива А оператор печати выглядит так:

PRINT 16, А(5)

Следует обратить внимание, что запись А(10) в операторе DIMENSION А(10) означает массив, состоящий из 10 элементов, а в остальных изученных нами операторах — это один 10-й элемент массива.

Оператор DIMENSION присутствует в программе только для того, чтобы сообщить транслятору, сколько ячеек отвести под массив, т. е. как распределить память машины. В транслированной программе нет машинных команд, соответствующих этому оператору. Такие операторы фортрана называются *невыполняемыми* или *декларативными*.

В одном операторе DIMENSION можно задать информацию о нескольких массивах.

DIMENSION A(50), B(100), C(30)

Пример. На картах пробито 1000 чисел по 10 на одной карте. Вычислить их среднее арифметическое, если спецификация, по которой пробиты все числа, F7.2.

PROGRAM MIDL	
DIMENSION A(1000)	Запрашивается веществен-
	ный массив A в 1000 ячеек
READ 1, A	Вводятся числа, пробитые
1 FORMAT (10F7.2)	на перфокартах (по 10
S=0.	чисел на одной карте)
DO 2 I=1, 1000	
2 S=A(I)/1000.+S	Вычисляется среднее ариф-
PRINT 3, S	метическое 1000 чисел
3 FORMAT (10X,	
*8НСРЕДНЕЕ□	
*10НАРИФМЕТИЧЕ	
*5НСКОЕ=, F7.2)	
END	

1.10. Первый выход на машину. Задания для БЭСМ-6 и ЕС ЭВМ

Если вы, уважаемый читатель, внимательно прочли всю главу, терпеливо выполнили упражнения, то можете приступить к составлению несложных программ. Даже в том случае, если у вас нет возможности запустить их на ЭВМ, вы получите необходимые навыки, программируя различные алгоритмы. Как решение шахматных этюдов совершенствует мастерство шахматиста, так и упражнение в составлении простых программ является необходимым этапом в овладении искусством программирования.

Упражнения по программированию на фортране можно найти в книгах [3, 5, 9, 12, 14, 15].

Если у вас есть возможность выйти на машину, обязательно воспользуйтесь ею. Для ввода в ЭВМ вашу

программу, например, можно пробить на перфокартах. Прежде чем программа начнет работать, она должна быть введена машиной в оперативную память и преобразована в машинные коды.

На большинстве современных ЭВМ работа осуществляется под управлением специального комплекса программ, называемого *операционной системой*. Машины каждого типа могут быть оснащены несколькими операционными системами. Например, для машин серии ЕС ЭВМ разработаны операционные системы ДОС ЕС и ОС ЕС. Система ДОС ЕС эксплуатируется в основном на «младших» моделях (ЕС-1020, ЕС-1030), а ОС ЕС — на «старших» моделях (начиная с ЕС-1040).

Для того чтобы запустить вашу программу на ЭВМ, надо составить так называемое задание. К перфокартам с программой нужно добавить карты, содержащие информацию о вашей задаче.

Например, операционная система должна «знать» максимальное количество машинного времени, которое необходимо отвести для решения задачи. Надо указать также, на каком языке программирования написана программа. Карты с подобной информацией называются управляющими. Для каждой операционной системы имеется свой набор управляющих карт. Мы рассмотрим некоторые из управляющих карт для операционной системы «Дубна» на БЭСМ-6 [13] и для систем ДОС ЕС и ОС ЕС [17].

1.10.1. Пример задания для БЭСМ-6 с операционной системой Дубна (пакет задачи).

```
*NAME PETROV
*PASS: PET125
*TIME: 00.05
      PROGRAM TAB
      DIMENSION A (10)
      ...
      END
*EXECUTE
*ENDFILE
Диспетчерский конец
```

*Управляющие карты пробиваются с первой позиции перфокарты и начинаются с символа *. В первой карте после «*NAME» пробивается фамилия программиста, во второй после «*PASS:» — его шифр на ЭВМ, под которым решаются все задачи данного пользователя (программи-*

ста). В третьей карте после «*TIME:» указывается время в часах и минутах, заказанное для решения задачи. В этой карте точка разделяет часы и минуты, например, *TIME:01.05 означает 1 час 5 минут.

После трех управляющих карт ставят карты фортранной программы (от PROGRAM до END). После фортранной программы должны стоять две управляющие карты: «*EXECUTE» и «*END — FILE».

Последней картой пакета является диспетчерский конец: карта со всеми пробивками по 1-й и 41-й колонкам. При вводе с терминала эта карта не нужна.

Если в программе есть оператор READ, то вводимые этим оператором карты помещают между управляющими картами «*EXECUTE» и «*END — FILE».

Для пробивки перфокарт используются устройства различных типов, отличающиеся друг от друга способом кодировки символов на перфокартах. Операционная система Дубна допускает ввод перфокарт с различной кодировкой. Даже в пределах пакета одной задачи могут быть карты, пробитые на устройствах различного типа.

Операционная система Дубна настроена на несколько стандартных кодировок, не требующих дополнительной о себе информации. Такими кодировками являются: УПП, УПДК, УПП М-220, а также одна из кодировок КПК-12 или CDC. Но если используется нестандартная кодировка (CDC или КПК-12, в зависимости от конкретной БЭСМ-6), то перед первой картой с такой кодировкой надо поставить карту, указывающую тип кодировки. Такими картами являются 7/9IBM для кодировки КПК-12 и 7/9CDC для кодировки CDC. Здесь 7/9 означает пробивку в 7-й и 9-й строках первой колонки. Это соответствует нажатию на клавиши с цифрами 7 и 9 в режиме MP (Multy Punch) на устройствах типа ЕС-9080, ЕС-9015, ICL-72 и т. п. Например, карта 7/9CDC настраивает операционную систему на восприятие последующих в кодировке CDC (наряду с кодировками УПП, УПДК и УПП М-220). Если в том же пакете последуют карты, пробитые в кодировке КПК-12, то перед первой необходимо поставить карту 7/9IBM.

1.10.2. Задание для ЕС ЭВМ с операционной системой ДОС ЕС.

```
//_JOB_PETROV
//_OPTION_LINK
//_EXEC_FFORTAN
```

```

        DIMENSION A(10) ] фортранная
        ...                программа
        END

/*
//__EXEC__LNKEDT
//__EXEC                данные для оператора READ
/*                      помещаются после //__EXEC
/&

```

В первой карте этого пакета после `//__JOB__` указывается фамилия программиста. Вторая карта задает режим работы ДООС ЕС, обеспечивающий запись результатов трансляции на магнитный диск и редактирование после трансляции. Третья карта означает вызов транслятора. Карта `«/*»` определяет конец файла; две последующие карты — редактирование и выполнение программы. Карта `«/&»` — конец пакета.

1.10.3. Примерное задание в ОС ЕС.

```

// задание JOB шифр, фамилия, MSGLEVEL = (2, 0),
//__CLASS__ = A
//__EXEC__FORTGCLG
//FORT.SYSPUNCH DD DUMMY
//FORT.SYSIN DD__ *
        DIMENSION A(10) ] фортранная
        ...                программа
        END

/*
//GO.SYSIN__DD__ *
/*
//

```

Данные для оператора READ ставятся после карты `//GO.SYSIN__DD__*`

Итак, вы написали и пробили фортранную программу и дополнили ее управляющими картами. Ваш пакет задачи готов к вводу в ЭВМ.

1.10.4. Примеры.

Пример 1. Пусть на ЭВМ БЭСМ-6 пользователь Петров, имеющий шифр PET123, должен вычислить площадь треугольника по основанию a и высоте h для $a=15,151$ и $h=12,087$.

Пакет задачи будет выглядеть так:

```

*NAME PETROV
*PASS:PET123
*TIME:00.05

```



```

PROGRAM T
  READ 1, A, H
  1 FORMAT (2F6.3)
  S = A * H * 0.5
  PRINT 2, S
  2 FORMAT (3X, 'S=', F8.3)
END
*EXECUTE
15.15112.087
*END FILE
Диспетчерский конец

```

Пример 2. Пусть задачу предыдущего примера надо решить на ЕС ЭВМ. Назовем задание TEST1. Пакет задания будет следующим:

```

//TEST1 JOB PET123, PETROV, MSGLEVEL=(1,1),
// GLASS = A
// EXEC FORTGCLG
//FORT.SYSPUNCH DD DUMMY
//FORT.SYSIN DD *
  READ 1, A, H
  1 FORMAT(2F6.3)
  S = A * H * 0.5
  PRINT 2,S
  2 FORMAT (3X, 'S=', F8.3)
  END
/*
//GO.SYSIN DD *
15.15112.087
/*
//

```

Если в программе встречаются операторы ввода/вывода

```

READ (m, <метка>), <список>
WRITE (m, <метка>), <список>

```

то среди управляющих карт должны быть соответствующие им карты вида

```
//FTmF001 DD . . .
```

Здесь m — двузначный (05, 11, 07. . .) номер устройства, имеющий то же значение, что и в операторе ввода/вывода.

Пример 3. Для чтения с магнитной ленты, имеющей имя АААААА, использован оператор

```
READ (09, 125), L
```

Ему должны соответствовать управляющие карты:

```
//GO.FT09F001 DD UNIT=TAPE,  
// DISP=OLD, LABEL=(⟨номер файла⟩, SL)  
// VOL=SER=АААААА  
// DSN=⟨имя файла⟩
```

По умолчанию приняты следующие стандартные значения *m*:

```
m=05 при чтении с перфокарт,  
m=06 при выводе на печать,  
m=07 при выводе на перфокарты.
```

Пример 4. Для ввода данных с перфокарт оператором

```
READ (05, 137), L
```

следует использовать управляющую карту

```
//GO.FT05F001 DD *
```

Пример 5. Для вывода на печать оператором

```
WRITE (06, 712), L
```

надо среди управляющих карт иметь такую:

```
//GO.FT06F001 DD SYSOUT=A
```

Пример 6. Для вывода на перфокарты оператором

```
WRITE (07, 52)L
```

нужна соответствующая управляющая карта

```
//GO.FT07F001 DD SYSOUT=B
```

1.10.5. Листинг ЭВМ БЭСМ-6 (фортран-Дубна). Результаты работы ЭВМ выдаются на печать. Кроме результата ЭВМ сообщает еще дополнительную полезную информацию. Рассмотрим подробнее, что печатается на каждой странице листинга.

Первая страница. В первой строке листинга ЭВМ сообщает дату и время запуска задачи, ниже — предприятие и тип ЭВМ, номер задачи в машине (например «ШИФР 2»), обозначение *версии* (варианта) операционной системы.

В ЭВМ БЭСМ-6 одновременно могут считаться несколько задач (*мультипрограммный режим*). Каждая из этих задач имеет свой счетный шифр. Когда кончается задача, шифр освобождается и вводится следующая задача.

Далее печатается крупными буквами фамилия программиста, указанная в карте *NAME.

Под ней — управляющие карты, расположенные перед программой.

Если на 1-х строках 1-й страницы имеется сообщение, то карта, на которую указывает одна из этих строк, пробита с ошибкой и ее надо пробить заново.

В т о р а я с т р а н и ц а. Повторяется дата и время запуска задачи, затем сообщается версия транслятора, (дата внесения последнего изменения в транслятор).

Далее печатается текст программы.

П р и м е р.

```
PROGRAM T
1  A=8*1.25
2  B=A**2
3  PRINT 7, B
4  7 FORMAT (2X, 2NB=, F9.3)
5  END
```

Если появляется текст какого-либо сообщения (диагностика) между операторами, то *надо искать ошибку в операторе, стоящем перед диагностикой.*

Например,

...
 $A=B*(C*(D+E))$

НЕЗАКРЫТЫЕ СКОБКИ

$D=F*K+A$

...

Диагностика «НЕЗАКРЫТЫЕ СКОБКИ» относится к оператору

$A=B*(C*(D+E))$

Если была хотя бы одна фатальная (недопустимая) ошибка при трансляции или при вводе (на первой странице листинга), то после текста программы машина выдает «БЫЛИ ОШИБКИ ПРИ ВВОДЕ ИЛИ ТРАНСЛЯЦИИ».

Если ошибок не было, то печатается «ВЫЗЫВАЕМЫЕ ФУНКЦИИ И ПОДПРОГРАММЫ» и перечисляются все функции и подпрограммы, которые встречались в программе.

Пр и м е р.

```
Y=SIN(X)+A*SQRT(ABS(B))
```

...

ЭВМ на листинге напечатает:

```
ВЫЗЫВАЕМЫЕ ФУНКЦИИ И ПОДПРОГРАММЫ
      SIN      SQRT      ABS
```

Далее выдается таблица восьмеричных относительных адресов. Остановимся на ней более подробно.

Каждый оператор фортрана, реализующий алгоритм решения задачи, превращается в одну или несколько машинных команд. Машинные команды размещаются в ячейках памяти. Относительный адрес оператора указывает, в какой по счету ячейке от начала программы находится данная команда этого оператора.

Пр и м е р. Пусть оператор № 1 соответствует четырем машинным командам, а оператор № 2 соответствует семи командам. Тогда относительный адрес оператора № 2 будет 0004, а относительный адрес следующего оператора № 3 будет 0011.

В первой строке таблицы относительных адресов помещены номера операторов от 1 до 20. Далее печатаются относительные адреса соответствующих операторов.

Пр и м е р. Таблица относительных адресов

	1	2	3	4	5	6...20
000:	0001	0005	0012			

Если операторов больше двадцати, то относительные адреса располагаются в последующих строках таблицы.

Пр и м е р.

	1	2	3	4	20
000:	0001	0003	0012	0020...	0117
001:	0125	0131	0132	0137...	0254
002:	0261	0263			

В каждой строке относительных адресов первым печатается номер этой строки, начиная с нулевого. По этой таблице можно определить оператор, если известен относительный адрес машинной команды этого оператора.

Пример. Найти номер оператора по приведенной таблице, если относительный адрес команды 136. Из таблицы видно, что эта команда принадлежит интервалу от 132 до 137, т. е. оператору № 23.

Ниже таблицы печатается управляющая карта «*EXECUTE», а затем таблица всех работавших при счете подпрограмм, в том числе и тех, что хранятся в памяти машины постоянно.

Эта таблица выдается в следующем виде: слева колонка с именами подпрограмм, справа колонка с соответствующими адресами первой команды данной подпрограммы. При этом PROGRAM всегда располагается с адреса 1000. По этой таблице можно определить, какой подпрограмме соответствует данный адрес.

Затем, если машина не нашла в программе ошибок, она решает предложенную задачу и печатает ответ. В заключение печатается последняя информация о задаче: сколько времени и когда она считалась.

Пример.

```
АДРЕС РАУ *И15* *И14* *И13* *И12* *И11* *И10*
00430 006 63401 00000 00427 01103 53052 02511
      *И9** *И7** *И6** *И5** *И4** *И3** *И2**
      34207 00000 07716 10270 00000 27251 23165
СВ=00.00.25 КВ=00.01.45 АВ=12.36.40
```

```
ДАТА=19/11/85
КОНЕЦ ЗАДАЧИ ПОТОК Ш-04 ВЫ/ВВ=6500
ОТДЕЛ=999—ШКОЛ
```

Здесь СВ=00 часов 00 минут 25 секунд — время, затраченное только на счет задачи (*счетное время*); КВ=00 часов 01 минута 45 секунд — время на счет, ввод программы, печать результата (*коммерческое время*); АВ=12 часов 36 минут 40 секунд — это *астрономическое время* запуска задачи на ЭВМ; ВЫ/ВВ=6500 — номер задачи в очереди на вывод; ОТДЕЛ — определяется по информации в карте «*PASS».

Читатель из этой информации может узнать, сколько времени считалась задача и сработала ли программа до конца. Если — да, то выдается сообщение КОНЕЦ ЗАДАЧИ. Если не хватило времени, указанного в *TIME, то выдается диагностика ИСЧЕРПАН ЛИМИТ ВРЕМЕНИ. Такое окончание задачи часто свидетельствует о заиклиивании программы. Остальная приведенная в примере информация может быть использована системным программистом.

Г л а в а II

ДЛЯ ТЕХ, КТО РЕШИЛСЯ ИДТИ ДАЛЬШЕ

Изучив первую главу, читатель получил необходимые сведения для составления простых программ. В этой главе мы продолжим рассказ о программировании для тех, кто хочет решать достаточно сложные задачи.

Кроме того, мы дадим некоторое представление о работе транслятора и познакомим читателя с поучительными примерами программ.

2.1. Дополнительные сведения о языке фортран

2.1.1. Вычисления с повышенной точностью. Для каждого числа, как мы знаем, обычно отводится одна ячейка памяти. Максимальная точность машинного представления чисел определяется конструктивными особенностями ЭВМ данного типа. Напомним, что на БЭСМ-6 эта точность равна 12 десятичным знакам, а на ЕС ЭВМ — 7.

В ряде случаев требуется произвести вычисления с бóльшей точностью. Например, при работе на ЕС ЭВМ мы хотим получить результат с 9-ю верными знаками. В подобных случаях прибегают к вычислениям с *двойной точностью*, т. е. под каждое число отводят не одну, а две последовательные ячейки памяти.

На ЕС ЭВМ таким образом увеличивают точность вычислений до 17 верных значащих цифр, а на БЭСМ-6 — до 24-х. В языке фортран такому представлению чисел соответствует тип `DOUBLE PRECISION` — с *двойной точностью*.

Константы с двойной точностью записываются в виде $K.nD \pm L$ или $K.D \pm L$. Здесь K — целая часть константы, n — дробная часть, D — признак двойной точности,

$\pm L$ — показатель степени при основании 10. Целая часть может отсутствовать, а знак + у показателя степени опущен.

Пример. Число $e=2,718\ 281\ 828\ 459\ 045$ можно записать в виде $2.718\ 281\ 828\ 459\ 045\ D0$ или $0.271\ 828\ 182\ 845\ 9045\ D1$ или $.002\ 718\ 281\ 828\ 459\ 045\ D3$.

Переменные величины, имеющие тип «с двойной точностью», описываются оператором DOUBLE PRECISION.

Примеры.

DOUBLE PRECISION A, XT7, PI, NK(8)
DOUBLE PRECISION BPRIM(5), ENTRY

В каждом операторе DOUBLE PRECISION можно указать несколько переменных или массивов, имена которых надо разделить запятыми.

Если нужно описать массив с двойной точностью, то можно указать в операторе DOUBLE PRECISION имя этого массива вместе с размерностью. В этом случае оператор DIMENSION не нужен.

Можно, однако, указать в операторе DOUBLE PRECISION только имя массива (без размерности). В этом случае для полного описания массива необходим еще оператор DIMENSION, в котором указывается имя массива и его размерность.

Например, массив X (15) типа «с двойной точностью» может быть описан любым из трех способов:

- а) DOUBLE PRECISION X (15)
- б) DOUBLE PRECISION X
DIMENSION X (15)
- в) DIMENSION X (15)
DOUBLE PRECISION X

Оператор DOUBLE PRECISION является декларативным оператором. Он должен предшествовать любому выполняемому оператору и может располагаться в произвольном порядке по отношению к другим декларативным операторам (в частности, DOUBLE PRECISION и операторам DIMENSION).

Если в программе используется константа, переменная или массив типа DOUBLE PRECISION, то это означает, во-первых, что для размещения каждой из величин этого типа будет отведено по две ячейки памяти, и, во-вторых, что арифметические операции над ними будут выполняться по иным правилам, чем, скажем, для величин вещественного типа.

2.1.2. Вычисления с комплексными числами. При решении ряда прикладных задач возникает необходимость вычислений с комплексными числами.

Каждое комплексное число $x+iy$ занимает в памяти машины две последовательные ячейки, в первой из которых находится действительная часть x , а во второй — мнимая часть y этого числа.

Комплексные числа записываются на фортране иначе, чем это принято в математике. Комплексному числу $x+iy$ на фортране соответствует запись в виде комплексной константы (x, y) . Здесь x и y должны быть *вещественными* константами. Например, комплексное число $2,3-5i$ может быть представлено в виде $(2.3,-5.)$ или $(0.23E1, -5.00)$, но не $(2.3E1, -5)$.

Переменная или массив комплексного типа описываются оператором COMPLEX.

П р и м е р ы.

```
COMPLEX ART, VEGA(6), INTEGR  
COMPLEX DRUM, VECTOR  
DIMENSION VECTOR(18)
```

Здесь комплексный массив VEGA описан в операторе COMPLEX вместе с размерностью, а комплексный массив VECTOR описан в операторе COMPLEX без указания размерности, а в операторе DIMENSION указаны его имя и размерность.

Оператор COMPLEX, как и оператор DOUBLE PRECISION, является декларативным оператором и предназначен лишь для того, чтобы сообщить транслятору о том, что данная переменная или массив имеют комплексный тип. Соответственно этому указанию о типе переменной транслятор отводит для размещения каждого такого числа по две ячейки памяти и формирует машинные команды, осуществляющие арифметические операции по правилам, установленным для комплексных чисел.

П р и м е р.

```
COMPLEX RT, MAN  
RT=(-2.5, 7.16)  
MAN=RT*(12.3, -6.)+(1.7, 2.6)**2
```

2.1.3. Использование величин разных типов в одном арифметическом выражении. Мы уже видели, что в одном арифметическом выражении можно использовать величины вещественного и целого типов. Введя два новых

типа величин (DOUBLE PRECISION и COMPLEX), мы должны определить и правила использования их в одном арифметическом выражении.

Введем следующие обозначения:

R — величина вещественного типа (REAL),

I — величина целого типа (INTEGER),

C — величина комплексного типа (COMPLEX),

DP — величина типа с двойной точностью (DOUBLE PRECISION),

— — недопустимая комбинация типов.

Приведенные ниже таблицы показывают (на пересечении соответствующей строки и столбца), какого типа результат получается при выполнении арифметических операций над величинами любого из четырех типов R, I, C и DP. В таблицах A даны по вертикали, B — по горизонтали.

Тип результата для $A+B$, $A-B$, $A*B$, A/B .

БЭСМ-6

B A	B			
	R	I	C	DP
R	R	R	C	DP
I	R	I	C	DP
C	C	C	C	—
DP	DP	DP	—	DP

ЕС ЭВМ

B A	B			
	R	I	C	DP
R	R	R	C	DP
I	R	I	C	DP
C	C	C	C	C
DP	DP	DP	C	DP

Тип результата для $A**B$
БЭСМ-6 и ЕС ЭВМ

B A	B			
	R	I	C	DP
R	R	R	—	DP
I	R	I	—	DP
C	—	C	—	—
DP	DP	DP	—	DP

Из приведенных таблиц видно, что если арифметическая операция связывает две величины одинакового типа, то результат будет того же типа (за исключением недопустимых комбинаций типов). Если же операция выполняется над двумя величинами разных типов, то тип

результата соответствует более «старшему» из типов величин, участвующих в операции. При этом порядок старшинства типов (от старшего к младшему) таков: C, DP, R, I.

Из последней таблицы следуют два правила:

1. *Нельзя возводить в комплексную степень.*
2. *Комплексные величины можно возводить только в целую степень.*

П р и м е р. Рассмотрим пример арифметического выражения, содержащего операции над величинами разных типов:

$$X + 2.5D0 - 5*(K + 2)**0.28$$

В процессе вычисления этого выражения будут получаться такие промежуточные результаты:

- 1) $D1 = X + 2.5 D0$ (тип DP),
- 2) $I1 = K + 2$ (тип I),
- 3) $R1 = I1**0.28$ (тип R),
- 4) $R2 = 5*R1$ (тип R),
- 5) $D2 = D1 - R2$ (тип DP).

Результат будет иметь тип DP, т. е. самый старший из типов, участвующих в данном выражении.

2.1.4. Преобразование типа величин в результате выполнения оператора присваивания. Оператор присваивания содержит слева от знака = имя переменной (простой или с индексом), а справа от этого знака — арифметическое выражение. Тип переменной может не совпадать с типом арифметического выражения. Например, в операторе $K = X + 5$. переменная K имеет целый тип, а выражение $X + 5$. — вещественный.

При выполнении оператора присваивания результат всегда преобразуется к типу переменной, стоящей слева от знака =. В рассмотренном примере результат, полученный после вычисления выражения $X + 5$., будет сначала преобразован в число целого типа и затем присвоен переменной K. Возможны, однако, недопустимые комбинации типов. На БЭСМ-6 таких комбинаций две: $C = DP$ и $DP = C$. На ЕС ЭВМ все комбинации типов являются допустимыми.

П р и м е р ы.

$$A = X + (1.5, -6.)*5 \text{ (результат типа R)}$$

$$N = 7./3. \text{ (результат типа I, равен 2)}$$

$$T = 1.3 D1 + (3., 7.) \text{ (недопустимая комбинация типов для БЭСМ-6; для ЕС ЭВМ результат типа R)}$$

2.1.5. Операторы описания типа. До сих пор при определении типа величин мы пользовались следующим соглашением: если имя переменной или массива начинается с одной из букв I, J, K, L, M, N, то переменная (массив) имеет тип «целый», в противном случае «вещественный». Это соглашение не распространяется, естественно, на переменные и массивы, описанные операторами DOUBLE PRECISION и COMPLEX.

В фортране допускается и другой способ описания вещественного и целого типа. Можно описать имя соответствующей переменной (или массива) в операторах REAL или INTEGER. Например, операторы

```
INTEGER T, PSI(25), IVAN  
REAL ANT, M50(50), J, TER
```

устанавливают для переменных T и IVAN и массива PSI тип целый (INTEGER), а для переменных ANT, J и TER и массива M50 — тип вещественный (REAL).

Если имя переменной или массива не указано ни в одном из операторов REAL, INTEGER, COMPLEX или DOUBLE PRECISION, то этой переменной (массиву) будет приписан вещественный или целый тип согласно правилу о первой букве имени.

Операторы REAL, INTEGER, COMPLEX и DOUBLE PRECISION называются *операторами описания типа*. Эти операторы являются декларативными и располагаются в начале программы до первого выполняемого оператора. Порядок расположения этих операторов по отношению к другим декларативным операторам произволен.

Кроме рассмотренных четырех типов, язык фортран допускает и другие типы величин, которые мы рассматривать не будем. Отметим только, что язык фортран ЕС ЭВМ допускает большее количество типов по сравнению с языком фортран для БЭСМ-6.

2.1.6. Оператор DATA. При составлении программ может возникнуть необходимость занести в определенные ячейки памяти нужные числа до начала работы программы.

Пусть, например, мы вычисляем значение многочлена $P = a_0 + a_1x + a_2x^2 + a_3x^3$, где коэффициенты a_0, a_1, a_2 и a_3 — некоторые заданные числа. Можно, конечно, присвоить соответствующим переменным заданные значения, написав, например, 4 оператора присваивания. Однако каждый такой оператор, во-первых, займет некоторое

количество ячеек памяти и, во-вторых, потребует определенное количество машинного времени для его выполнения. В этом случае удобнее занести нужные значения констант в соответствующие ячейки памяти до начала работы программы. Такое занесение производится оператором DATA. Пусть, например, мы хотим присвоить коэффициентам многочлена P значения 1.38, —2.5E4, 18.3 и 3.6E2. Мы можем обозначить эти коэффициенты именами A0, A1, A2 и A3 и написать такой оператор:

```
DATA A0, A1, A2, A3/1.38,—2.5E4, 18.3, 3.6E2/
```

Здесь сначала перечисляются через запятую имена переменных, куда надо занести нужные числа, а затем в косых черточках (/.../) перечисляются константы, которые подлежат занесению.

Можно объединить коэффициенты A0, A1, A2 и A3 в один массив и затем написать оператор DATA, производящий занесение в этот массив значений четырех констант:

```
DIMENSION A(4)  
DATA A/1.38, —2.5E4, 18.3, 3.6E2/
```

В операторе DATA можно после закрывающей косой черты / поставить запятую и затем указать новый список переменных и соответствующий ему список констант. Например,

```
DATA A0/1.38/, A1/—2.5E4/, A3/18.3/, A4/3.6E2/
```

Если среди заносимых констант имеются повторения, то можно перед константой указать коэффициент повторения и знак *. Например, оператор

```
DATA X1, Y1, Z1/1., 1., 1./
```

допускает более компактную запись

```
DATA X1, Y1, Z1 /3*1./
```

Оператор DATA, в отличие от оператора присваивания, является *невыполняемым* оператором. Занесение констант в ячейки памяти, указанные в операторе DATA, производится системой математического обеспечения до начала работы программы.

Отличие оператора DATA от оператора присваивания состоит также в том, что занесение констант производится *без преобразования* их к нужному типу. Поэтому для большинства трансляторов требуется, чтобы тип кон-

станты соответствовал типу переменной, которой присваивается значение этой константы.

Например, для транслятора на ЕС ЭВМ запись DATA RNP/5/ является ошибочной, так как константа 5 является целой, а переменная RNP — вещественной. Транслятором БЭСМ-6 указанная запись не будет признана ошибочной, однако переменной RNP будет присвоено значение целой константы 5 без преобразования ее к вещественному типу. Такое несоответствие типа может привести к неприятностям при счете. Например, попытка разделить на RNP будет расценена как деление на нуль.

Поскольку оператор DATA является невыполняемым, то он должен быть расположен среди других невыполняемых операторов программы и обязательно предшествовать первому выполняемому оператору. Транслятор на ЕС ЭВМ требует, чтобы оператор DATA был расположен *после* декларативных операторов, описывающих переменные и массивы, указанные в данном операторе DATA. Например, последовательность операторов

```
DATA X1 /5., 3./  
DIMENSION X1 (2)
```

при трансляции на ЕС ЭВМ будет признана ошибочной. Необходимо сначала указать оператор DIMENSION, а затем оператор DATA.

На БЭСМ-6 допускается *любой* порядок расположения оператора DATA относительно других невыполняемых операторов.

У п р а ж н е н и е 12.1. Занести в массив C, состоящий из 8 элементов, следующие числа: 15.2, 17.3, 17.3, 17.3, 17.3, 17.3, 1.5, 3.2.

2. Какого типа результат получится в следующих случаях:

```
DOUBLE PRECISION A, C  
COMPLEX B, D
```

а) $F = A * C + 15$

б) $F = B ** 2 - B + 1.5$

в) $F = A ** 2 - C + D$

2.1.7. Вычисляемый оператор GO TO. Мы уже познакомились с оператором перехода GO TO m. Обобщением этого оператора является вычисляемый GO TO, который записывается в виде

```
GO TO (m1, m2, . . . , mk), L
```

Здесь m_1, m_2, \dots, m_k — метки операторов, на которые можно осуществлять переход, L — простая целая переменная, значение которой определяет, куда именно производится переход. Если к моменту выполнения вычисляемого GO TO значение L равно 1, то произойдет переход на оператор с меткой m_1 ; если $L=2$, то на оператор с меткой m_2 и т. д. Значение L определяет, таким образом, порядковый номер метки, на которую производится переход с помощью вычисляемого оператора GO TO.

П р и м е р.

LABEL=3

GO TO (42, 7, 5, 28), LABEL

Здесь управление будет передано оператору с меткой 5, так как переменной LABEL было присвоено значение 3, и поэтому она указывает на третью метку списка.

Для выполнения вычисляемого GO TO необходимо, чтобы значение переменной L удовлетворяло условию $1 \leq L \leq K$, где K — число меток в списке. Если же это условие не выполняется, то значению L не будет соответствовать метка. В этом случае выполнение оператора происходит по-разному в зависимости от транслятора.

На ЕС ЭВМ перед выполнением вычисляемого GO TO производится проверка условия $1 \leq L \leq K$, и если это условие не выполняется, то управление передается оператору, следующему за вычисляемым GO TO. На БЭСМ-6 проверка указанного условия не производится, и в случае его нарушения происходит передача управления, не предусмотренная в программе (потеря управления). Поэтому при работе на БЭСМ-6 желательно перед выполнением вычисляемого GO TO запрограммировать проверку выполнения условия $1 \leq L \leq K$.

Рассмотрим пример использования вычисляемого оператора GO TO в программе. Пусть имеется группа операторов, которая используется в нескольких местах программы. Для того чтобы не переписывать эту группу столько раз, сколько она встречается в программе, можно поступить так. Первый оператор группы снабжается меткой. Последним оператором группы пишется вычисляемый оператор GO TO, в котором должно быть столько меток, сколько возможных выходов предусмотрено из этой группы операторов.

```

L = 1
GO TO 7
15 CONTINUE

```

```

...
L = 3
GO TO 7
83 CONTINUE

```

```

...
L = 2
GO TO 7
42 CONTINUE

```

```

7 CONTINUE
GO TO (15, 83, 42), L

```

} группа операторов,
к которой происходит
обращение

Перед тем как использовать группу операторов

```

7 CONTINUE

```

```

GO TO (15, 83, 42), L

```

переменной L присваивается значение 1, либо 2, либо 3 в зависимости от порядкового номера метки (15, 83, 42), на которую будет передано управление по окончании работы группы операторов.

2.1.8. Оператор IF логический. Нам уже знаком оператор вида IF (A) m, n, k, который в зависимости от значения арифметического выражения A производит переход на одну из трех меток m, n, k. Этот оператор называют также IF *арифметический*.

Другой разновидностью оператора IF является IF *логический*. Общий вид этого оператора

```

IF (L) S

```

где L — логическое выражение, S — оператор.

Логическое выражение L может принимать одно из двух значений — «истина» или «ложь». В случае, когда значение L — «истина», после оператора IF логический выполняется оператор S . В противном случае (т. е. когда значение L — «ложь») оператор S не выполняется, а производится передача управления оператору, следующему за IF логическим. В качестве оператора S может быть взят любой оператор, кроме DO и IF логического.

Рассмотрим простейшие логические выражения — выражения отношения, которые наиболее часто используются на практике. Общий вид этих выражений:

A1.mn.A2. Здесь A1 и A2 — арифметические выражения, .mn.— знак операции отношения. Допускается 6 операций отношения: .EQ.— равно, .NE.— не равно, .GT.— больше, .GE.— больше или равно, .LT.— меньше, .LE.— меньше или равно.

Например, $A > 5.3$ записывается как A.GT.5.3; $B < C$ как B.LT.C, $k^2/5 \leq 25i$ как K**2/5..LE.25*I.

П р и м е р.

IF (K.NE.0) C=M/K*D

X=P+Q

...

Если справедливо условие $K \neq 0$, то после оператора IF будет выполнен оператор $C=M/K*D$, а затем оператор $X=P+Q$. Если же $K=0$, то после оператора IF сразу же выполняется оператор $X=P+Q$.

П р и м е р. Пусть извлекается корень из величины C, которая может оказаться отрицательной за счет погрешности вычислений. Разумно застраховать себя от подобных неприятностей следующим образом:

IF(C.LT.0.) C=0.

B=SQRT(C)

В случае $C < 0$ выполняется оператор присваивания $C=0$, который не работает в случае $C \geq 0$.

У п р а ж н е н и е 13. В случае $x > 3$ вычислить $A = -B + +B\sqrt{x-3}$ В остальных случаях положить $A = -B$.

П р и м е р. Поменять местами значения переменных A и B, если $A > B$; в противном случае оставить все без изменения.

IF (A.LE.B) GO TO 1

C=A

A=B

B=C

1 CONTINUE

Первый оператор проверяет условие $A \leq B$, и если оно выполняется (т. е. выражение A.LE.B истинно), то передает управление оператору с меткой 1. Если же $A > B$, то оператор GO TO 1 не выполняется и производится перестановка переменных A и B. Эта перестановка выполняется посредством трех операторов присваивания. Сначала надо переслать значение переменной A в ячейку C и только затем можно заносить в A значение B. Если не будет указан оператор $C=A$, то оператор

$A=B$ сотрет прежнее значение переменной A . После выполнения оператора $A=B$ в ячейку B пересылается значение переменной A , сохраненное в ячейке C .

2.1.9. Двумерные массивы. Мы уже знакомы с одномерными массивами. Каждый элемент такого массива имеет один индекс, например $A(I)$. В фортране допускаются также двумерные и трехмерные массивы. Аналогом двумерного массива является множество мест в кино-театре. Каждое место определяется двумя координатами: «ряд» и «место». Координатами, определяющими положение каждого элемента двумерного массива, являются индексы — номер строки и номер столбца.

Пример. Пусть задан двумерный массив C :

0.1	5.3	6.8	10.15
13.1	8.6	3.5	15.2
7.5	3.8	8.4	0.8

Этот массив содержит 3 строки по 4 элемента в каждой. Элемент массива обозначается именем C и двумя индексами (номер строки и номер столбца), которые указываются в скобках и разделяются запятыми. Например, 8.6 есть элемент $C(2, 2)$, 15.2 — элемент $C(2, 4)$, 3.8 — элемент $C(3, 2)$.

Каждый двумерный массив описывается оператором вида `DIMENSION A (m, n)`. Здесь A — имя массива, m — число строк, n — число столбцов. Значения m и n задаются в виде целых констант без знака. Например, рассмотренный выше массив C описывается оператором `DIMENSION C (3, 4)`.

Пример. Возвести в квадрат все элементы 3-й строки массива A , имеющего 5 строк и 7 столбцов.

```
DIMENSION A (5, 7)
DO 2 J=1,7
  A (3, J)=A (3, J)**2
2 CONTINUE
```

Здесь переменная J , определяющая номер столбца у элемента массива A , пробегает все значения от 1 до 7, а номер строки остается постоянным, равным 3.

Пример. Возвести в квадрат все элементы массива A предыдущего примера.

```
DO 2 I=1, 5
DO 2 J=1, 7
2 A(I, J)=A(I, J)**2
```

Здесь использованы два цикла, оканчивающиеся одним и тем же оператором. Внешний цикл производит перебор номеров строк от 1 до 5. Во внутреннем цикле для каждого значения номера строки перебираются все номера столбцов от 1 до 7.

Пример. Занести во все элементы главной диагонали массива $X(10, 10)$ число 3. Главная диагональ идет от элемента $X(1, 1)$ к элементу $X(10, 10)$, проходя через элементы $X(2, 2)$, $X(3, 3)$ и т. д.

```
DIMENSION X(10, 10)
DO 5 K=1, 10
  5 X(K, K)=3.
```

Здесь элементы главной диагонали имеют индексы (K, K) , $K=1, \dots, 10$.

2.1.10. Вывод двумерного массива на печать. Начнем с примера. Пусть массив A состоит из 3-х строк по 4 элемента в каждой:

3.	2.1	8.	4.3
0.2	0.3	0.8	0.4
0.05	0.001	0.03	0.15

В памяти машины эти числа хранятся по столбцам, т. е. в такой последовательности: 3., 0.2, 0.05, 2.1, 0.3, 0.001, 8., 0.8, 0.03, 4.3, 0.4, 0.15. Для вывода всех элементов этого массива в той последовательности, в какой они хранятся в памяти машины, достаточно написать оператор `PRINT 42,A`, где 42 — метка оператора `FORMAT`, A — имя массива. Например, операторы

```
PRINT 42, A
42 FORMAT (10X, 12F6.3)
```

отпечатают все 12 чисел в одну строку.

Для вывода элементов массива в иной последовательности можно использовать неявный цикл. Например, если мы хотим вывести эти элементы по строкам, то можно написать оператор

```
PRINT 42, ((A(I, J), J=1, 4), I=1, 3).
```

Этот оператор работает следующим образом. Сначала I принимает значение 1 и перебираются все элементы первой строки с номерами столбцов 1, 2, 3, 4 ($J=1, 4$). Затем I принимает значение 2 и перебор значений J повторяется. Последним значением I будет 3, при котором также будет произведен перебор всех значений J

от 1 до 4. Если задать оператор FORMAT так, чтобы в каждой строке было отпечатано 4 числа, то мы получим на бумаге все элементы массива в их «естественном» расположении, т. е. в виде двумерной таблицы, состоящей из 3-х строк по 4 элемента в каждой.

Операторы

```
PRINT 42, ((A (I, J), J=1, 4), I=1, 3)
42 FORMAT (5X, 4F6.3)
```

и позволяют получить такое расположение элементов массива A при печати.

В неявном цикле можно не перебирать все значения строк и все значения столбцов. Например, если мы хотим вывести на печать только 1-ю и 3-ю строки массива A, то соответствующий PRINT можно записать так:

```
PRINT 42, ((A (I, J), J=1, 4), I=1, 3, 2).
```

Здесь перебор строк идет с шагом 2 (если шаг равен 1, то его можно не указывать).

В общем случае оператор PRINT, содержащий неявный цикл, можно записать так:

```
PRINT p, ((A (I, J), J=m1, n1, k1), I=m2, n2, k2).
```

Здесь p — метка оператора FORMAT, m1 и m2 — начальные значения индексов J и I, n1 и n2 — конечные значения этих индексов, k1 и k2 — значения шагов по индексам J и I. Значения величин m1, n1, k1, m2, n2, k2 должны быть строго положительны. Эти величины могут быть заданы в виде целых констант или целых простых переменных.

Неявный цикл можно использовать и в операторе READ. Например, операторы

```
DIMENSION R (8, 6)
READ 5, ((R (K, L), L=2, 6, 2), K=1, 8, 5)
5 FORMAT (6F8.2)
```

производят ввод 6 чисел, пробитых на одной перфокарте, и занесение этих чисел в элементы R (1, 2), R (1, 4), R (1, 6), R (6, 2), R (6, 4), R (6, 6).

2.1.11. Подпрограмма. При решении разных задач часто возникает необходимость проводить вычисление по одним и тем же алгоритмам, например, вычислять корень уравнения $f(x)=0$ или находить максимальный элемент в двумерном массиве. Нельзя ли не программировать такие алгоритмы каждый раз заново, а сделать это однажды и использовать уже готовые программы?

В языке фортран предусмотрена возможность объединения любой последовательности операторов в отдельную подпрограмму. Для приведения ее в действие необходимо написать соответствующие операторы, подключающие подпрограмму к основной программе. В дальнейшем мы уточним понятия «программа» и «подпрограмма».

Подпрограмма имеет заголовок вида

SUBROUTINE S или SUBROUTINE S (X, Y, Z, ...).

Здесь S — имя подпрограммы, а X, Y, Z, ... — формальные параметры. Рассмотрим их подробнее.

Формальные параметры — это наименования переменных и массивов, через которые передается информация из основной программы в подпрограмму (исходные данные) или из подпрограммы в основную программу (результаты).

Пусть, например, мы составили подпрограмму решения квадратного уравнения $ax^2+bx+c=0$:

SUBROUTINE SQ (A, B, C, X1, X2)

Она содержит 5 формальных параметров, причем первые три отведены для коэффициентов a , b и c , а последние два отведены для корней уравнения x_1 и x_2 . Определение «формальные» подчеркивает, что подпрограмма решает задачу в общем виде.

Иллюстрацией принципа работы подпрограммы может служить автомат по продаже билетов на пригородные поезда. Этот автомат настроен на работу с различными наборами монет (формальные параметры) и выдает билеты до разных станций назначения (результат). Для того чтобы автомат сработал, надо опустить монеты конкретного достоинства (фактические параметры).

Для того чтобы запустить подпрограмму в работу, необходимо к ней обратиться (или ее вызвать). Вызов подпрограммы (или обращение к подпрограмме) производится оператором CALL. В этом операторе указываются имя подпрограммы и в скобках — фактические параметры (если у подпрограммы имеются формальные параметры). Например, для вызова подпрограммы SQ (A, B, C, X1, X2) мы должны в операторе CALL указать значения пяти фактических параметров, соответствующих A, B, C, X1 и X2. Если, допустим, мы хотим решить уравнение $5x^2-12x+3=0$ и обозначили его корни через T и C, то обращение к подпрограмме SQ будет выглядеть так:

CALL SQ (5., -12., 3., T, C)

Нет необходимости задавать, значения коэффициентов уравнения в виде констант.

Если, например, значение старшего коэффициента уравнения содержится в переменной P, а остальных коэффициентов — соответственно в Q и R, то обращение к SQ может быть таким:

CALL SQ (P, Q, R, T, C)

Программу, в которой указан оператор CALL, называют *вызывающей программой*.

При выполнении оператора CALL производятся следующие действия. Информация о фактических параметрах (если они имеются) засылается в n подряд идущих ячеек памяти, где n — число параметров. Затем управление передается на начало вызываемой подпрограммы.

Подпрограмма получает через вышеупомянутые ячейки информацию о фактических параметрах и устанавливает взаимнооднозначное соответствие между формальными и фактическими параметрами. Затем производится выполнение подпрограммы.

После того как подпрограмма выполнится, управление будет передано в вызывающую программу на оператор, следующий за оператором CALL. На ЕС ЭВМ предусмотрена также возможность передачи управления на оператор, метка которого указана в качестве одного из фактических параметров (см. [17]).

Составим подпрограмму SQ (A, B, C, X1, X2) для решения квадратного уравнения с коэффициентами A, B, C, которая в качестве результатов выдает значения вещественных корней X1 и X2. Будем предполагать, что при заданных значениях коэффициентов уравнение имеет вещественные корни.

SUBROUTINE SQ (A, B, C, X1, X2)

D=SQRT (B**2-4.*A*C)

X1=(-B+D)/(2.*A)

X2=(-B-D)/(2.*A)

RETURN

END

В подпрограмме SQ встретился незнакомый нам оператор RETURN. Этот оператор производит возврат (т.е. передачу управления) в вызывающую программу. Он не тождествен оператору END, который дает «знать» транслятору, что больше нет операторов, относящихся к данной подпрограмме. В подпрограмме может быть несколько

ко операторов RETURN, а оператор END всегда только один.

Составленная нами подпрограмма SQ позволяет решать квадратные уравнения, имеющие неотрицательный дискриминант. А что произойдет, если в вызывающей программе будут заданы такие значения коэффициентов, что дискриминант получится отрицательным (например CALL SQ (3., 1., 7.5, RT, PSI). В этом случае произойдет *аварийный останов (авост)*, т. е. прекращение работы программы при попытке извлечения корня из отрицательного числа.

Аналогичная ситуация может возникнуть в случае, когда какой-либо из фактических параметров будет указан в виде целой (а не вещественной) константы или переменной. Например, при обращении

CALL SQ (3, 1., 7.5, RT, PSI)

где вместо вещественной константы 3. указана целая константа 3, подпрограмма SQ может не сработать. То же самое может произойти и в случае, когда фактических параметров указано больше или меньше, чем следует.

Таким образом, подпрограмма требует правильного обращения, иначе она может либо выдать неправильный результат, либо ее работа будет прекращена.

Аналогично ведет себя упоминавшийся выше автомат по продаже билетов. Попытка использовать «плохие» монеты (гнутые, иного достоинства) может привести либо к тому, что билет не будет выдан, либо к порче автомата.

П р и м е р. Рассмотрим, как можно воспользоваться подпрограммой SQ для нахождения суммы кубов корней уравнений

$$3x^2 + 8x - 0,1 = 0 \quad \text{и} \quad -6y^2 + 12y + 0,3 = 0.$$

PROGRAM RESULT

CALL SQ (3., 8., -0.1, X1, X2) Для решения 1-го уравнения

CALL SQ (-6., 12., 0.3, Y1, Y2) Для решения 2-го уравнения

S = X1**3 + X2**3 + Y1**3 + Y2**3

PRINT 5, S

5 FORMAT (6X, 6HCYMMMA=, E12.3)

END

После выполнения двух операторов CALL переменные X1, X2, Y1, Y2 будут содержать значения корней данных уравнений.

2.1.12. Подпрограмма-функция. Другим способом оформления алгоритма на фортране является подпрограмма-функция. Заголовок ее имеет вид

```
FUNCTION F (X, Y, Z, . . .)
```

У подпрограммы-функции, в отличие от SUBROUTINE, обязательно должен быть хотя бы один формальный параметр. Другое, более важное отличие FUNCTION от SUBROUTINE состоит в том, что эти подпрограммы по-разному вызываются из основной программы. Для того чтобы вызвать подпрограмму-функцию, надо указать ее имя вместе со списком фактических параметров в любом арифметическом выражении. Пусть, например, подпрограмма-функция AMAX (X, Y) вычисляет значение максимального из параметров X и Y. Тогда оператор $R = \text{AMAX} (T, P * Q)$ означает присваивание переменной R максимального из значений двух величин T и $P * Q$.

Отличительной особенностью FUNCTION является также то, что результатом ее работы является одно число, которое присваивается имени этой FUNCTION. Например, упоминавшаяся подпрограмма-функция AMAX может быть оформлена так:

```
FUNCTION AMAX (X, Y)
  AMAX = X
  IF (X.LT.Y) AMAX = Y
  RETURN
END
```

Имя подпрограммы-функции определяет тип выдаваемого ею результата. Поэтому, например, стандартная подпрограмма-функция, вычисляющая натуральный логарифм, называется ALOG (X), а не LOG (X). В последнем случае имя начинается с буквы L и определяет величину типа «целый».

Подпрограмма-функция и подпрограмма, несмотря на их различие, имеют много общего. Фактическими параметрами у той и другой могут быть арифметические выражения. Тип и порядок следования фактических параметров должен строго соответствовать типу и порядку следования формальных параметров.

Если среди формальных параметров имеется массив, то он должен быть описан оператором DIMENSION. До сих пор мы имели дело с массивами, размерности которых задавались в виде целых констант, например DIMENSION

X (50). В подпрограммах и подпрограммах-функциях в качестве формальных параметров можно указывать массивы, размерности которых задаются в виде целых переменных, например DIMENSION R(N). В этом случае имена R и N должны быть указаны в списке формальных параметров.

Проиллюстрируем сказанное примером. Составим подпрограмму-функцию, вычисляющую максимальное значение элемента массива A, если A состоит из N элементов.

```
FUNCTION RMAX (A, N)
  DIMENSION A (N)
  RM=A (1)
  DO 1 I=2, N
1 IF (A (I).GT.RM) RM=A (I)
  RMAX=RM
  RETURN
END
```

Здесь в качестве A может быть задан любой массив, содержащий не менее двух элементов. На выходе (в результате) получается максимальное значение среди элементов заданного массива. Обратим внимание на последний оператор цикла DO—IF логический. Такое окончание цикла допускается. При этом в случае истинности выражения, указанного в скобках, производится выполнение оператора RM=A (I), а в случае ложности — продолжение цикла.

Следует особо отметить, что подпрограмма и подпрограмма-функция получают информацию через формальные параметры и больше ничего о вызывающей программе не «знают» (см. также 2.1.13). Подобно этому и автомат, выдающий билеты, реагирует лишь на опущенные в него монеты и ничего не «знает» о тех, что лежат у вас в кармане.

В виде подпрограмм-функций оформляется вычисление элементарных функций: $\sin x$, $\ln x$, e^x и др. Поскольку значения таких функций часто используются в программах, соответствующие подпрограммы-функции оформлены в виде библиотеки стандартных подпрограмм-функций. Эти функции вычисляются с точностью, близкой к максимально возможной для данной ЭВМ и по возможности быстро.

Приведем данные о наиболее распространенных стандартных (или библиотечных) подпрограммах-функциях на ЕС ЭВМ и БЭСМ-6 (см. также Приложение II).

Назначение	Обращение	Тип результата
<p>Экспонента</p> e^X e^{CX} e^{DX}	$EXP(X)$ $CEXP(CX)$ $DEXP(DX)$	<p>Вещественный Комплексный С двойной точностью</p>
<p>Натуральный логарифм</p> $\ln X$ $\ln CX$ $\ln DX$	$ALOG(X)$ $CLOG(CX)$ $DLOG(DX)$	<p>Вещественный Комплексный С двойной точностью</p>
<p>Десятичный логарифм</p> $\lg X$ $\lg DX$ $\lg CX$	$ALOG10(X)$ $DLOG10(DX)$ $CLOG10(CX)$ нет на БЭСМ-6, есть на ЕС ЭВМ	<p>Вещественный С двойной точностью Комплексный</p>
<p>Арктангенс</p> $\arctg X$ $\arctg DX$ $\arctg CX$	$ATAN(X)$ $DATAN(DX)$ $CATAN(CX)$ нет на БЭСМ-6, есть на ЕС ЭВМ	<p>Вещественный С двойной точностью Комплексный</p>
<p>Корень квадратный</p> \sqrt{X} \sqrt{CX} \sqrt{DX}	$SQRT(X)$ $CSQRT(CX)$ $DSQRT(DX)$	<p>Вещественный Комплексный С двойной точностью</p>
<p>Синус угла (в радианах)</p> $\sin X$ $\sin CX$ $\sin DX$	$SIN(X)$ $CSIN(CX)$ $DSIN(DX)$	<p>Вещественный Комплексный С двойной точностью</p>
<p>Косинус угла (в радианах)</p> $\cos X$ $\cos CX$ $\cos DX$	$COS(X)$ $CCOS(CX)$ $DCOS(DX)$	<p>Вещественный Комплексный С двойной точностью</p>
<p>Модуль числа</p> $ X $ $ IX $ $ CX $ $ DX $	$ABS(X)$ $IABS(IX)$ $CABS(CX)$ $DABS(DX)$	<p>Вещественный Целый Вещественный С двойной точностью</p>

Назначение	Обращение	Тип результата
Целая часть числа [X] [X] [DX] [DX]	AIN(T)(X) INT(X) IDINT(DX) DINT(DX)	Вещественный Целый Целый С двойной точностью
Остаток от деления X/Y IX/IY	AMOD(X, Y) MOD(IX, IY)	Вещественный Целый
Выбор максимального элемента среди X1, X2, X3, ... XN IX1, IX2, IX3, IXN IX1, IX2, IX3, IXN X1, X2, X3, ... XN DX1, DX2, DX3, DXN	AMAX1(X1, X2, ...) AMAX0(IX1, IX2, ...) MAX0(IX1, IX2, ...) MAX1(X1, X2, ...) DMAX1(DX1, DX2, ...)	Вещественный Вещественный Целый Целый С двойной точностью
Выбор минимального элемента среди X1, X2, X3, ... XN IX1, IX2, IX3, IXN IX1, IX2, IX3, IXN X1, X2, X3, ... XN DX1, DX2, DX3, DXN	AMIN1(X1, X2, ...) AMIN0(IX1, IX2, ...) MIN0(IX1, IX2, ...) MIN1(X1, X2, ...) DMIN1(DX1, DX2, ...)	Вещественный Вещественный Целый Целый С двойной точностью
Выделение главной части аргумента с двойной точностью	SNGL(DX)	Вещественный
Выделение действительной части комплексной величины CX	REAL(CX)	Вещественный
Выделение мнимой части комплексного числа CX	AIMAG(CX)	Вещественный
Приведение к типу «с двойной точностью»	DBLE(X)	С двойной точностью
Объединение X1 и X2 в комплексную величину $X1 + iX2$	CMPLX(X1, X2)	Комплексный
Получение комплексного числа, сопряженного данному CX	CONJG(CX)	Комплексная
Тангенс угла (в радианах) $\text{tg}X$	TAN(X)	Вещественный

В таблице использованы следующие обозначения:
X — вещественная переменная,
CX — комплексная переменная,
DX — с двойной точностью,
IX — целая.

По первой букве имени этих подпрограмм можно определить тип результата: если первая буква — одна из I, J, K, L, M, N, то тип «целый», если буква C, то «комплексный», если D, то «с двойной точностью», во всех остальных случаях тип результата «вещественный».

2.1.13. Оператор COMMON. Программа, вызывающая какую-либо подпрограмму или подпрограмму-функцию, обменивается с ней информацией через параметры. В фортране имеется и другой способ передачи информации — через общие массивы ячеек, называемые *COMMON-блоками* или *общими блоками*.

Для описания таких блоков служит невыполняемый оператор COMMON. Этот оператор имеет вид

COMMON / A / B, C, D, . . . , X

Здесь A — имя общего блока, B, C, D, . . . , X — переменные, входящие в этот блок.

Пусть в PROGRAM MIST указан оператор

COMMON / LXTP / R (15), C, M28A

где C и M28A — простые переменные. Это означает, что в памяти выделен блок из 17 ячеек, который может использовать любая подпрограмма (или подпрограмма-функция).

Допустим, мы хотим, чтобы SUBROUTINE IVAR получила доступ к этому COMMON-блоку. Тогда в подпрограмме надо описать блок с именем LXTP длиной 17 ячеек.

Имя блока должно быть одно и то же в вызывающей и вызываемой программах. Имена переменных и массивов, входящих в блок, выбираются по усмотрению программиста и могут не совпадать с именами, выбранными в PROGRAM. Пусть в SUBROUTINE IVAR указан оператор

COMMON / LXTP / A (8), B (8), NT

Первая ячейка блока названа в MIST R (1), а в IVAR — A (1). Таким образом, R (1) и A (1) — это обозначения одной и той же ячейки в разных программах.

Если подпрограмма использует COMMON-блок, то она может не иметь параметров.

Пр и м е р. Проиллюстрируем это на примере подпрограммы решения квадратного уравнения, которая была уже составлена в варианте, имеющем 5 формальных параметров. В новом варианте вместо параметров будет использоваться COMMON-блок.

```
SUBROUTINE SQCOM
COMMON / AB / A, B, C, X1, X2
D=SQRT (B**2-4.*A*C)
X1=(-B+D)/(2.*A)
X2=(-B-D)/(2.*A)
RETURN
END
```

Для того чтобы воспользоваться такой подпрограммой, необходимо в вызывающей программе описать блок AB. До обращения к SQCOM в первые 3 ячейки COMMON-блока должны быть занесены значения коэффициентов уравнения. После обращения к SQCOM в 4-й и 5-й ячейках COMMON-блока будут находиться значения корней уравнения.

Вообще один и тот же алгоритм можно оформить в виде различных подпрограмм, отличающихся друг от друга числом формальных параметров.

При обращении к подпрограмме число фактических параметров должно быть в точности равно числу формальных. Если даже какой-либо фактический параметр не существен для решения конкретной задачи, он должен быть указан при обращении к подпрограмме (или подпрограмме-функции). COMMON-блоки допускают в этом смысле большую свободу в использовании.

Если информация, содержащаяся в COMMON-блоке, не нужна, то в программе можно не описывать соответствующий COMMON-блок. Поэтому при составлении подпрограммы обычно придерживаются такого принципа. Информация, необходимая для работы подпрограммы, а также основные результаты оформляются в виде формальных параметров. Иногда представляют интерес и промежуточные значения каких-либо переменных подпрограммы. Такие результаты выдаются через COMMON-блоки.

Для решения задачи на ЭВМ мы можем составить либо только одну PROGRAM (головную программу), либо PROGRAM и несколько SUBROUTINE и FUNCTI-

ON. В любом случае PROGRAM должна быть обязательно (и притом только одна), а количество FUNCTION и SUBROUTINE может быть произвольным. Программа (PROGRAM) и подпрограммы (FUNCTION, SUBROUTINE), входящие в пакет какой-либо задачи, называются *модулями программы*.

Обратим внимание еще на одно обстоятельство, связанное с именами переменных (или массивов), используемых в подпрограммах. *Если имя не является ни формальным параметром, ни фактическим, ни элементом COMMON-блока, то к соответствующей ячейке (ячейкам) памяти не имеют доступа другие модули программы.* Например, в подпрограмме SQCOM таким именем является D. Это имя является локальным, т. е. «личной собственностью» данной подпрограммы. Если есть необходимость, то можно сделать величину D доступной для других модулей программы. Например, можно описать переменную D в некотором COMMON-блоке. Однако такая возможность должна быть заложена при написании подпрограммы SQCOM.

2.1.14. Модульная структура программ. До сих пор мы говорили об отдельных подпрограммах (модулях) фортранной программы. Теперь рассмотрим программу в целом. Работа начинается с главной программы (головного модуля), которая имеет заголовок PROGRAM (на ЕС ЭВМ этот модуль не имеет заголовка). Любой другой модуль (FUNCTION или SUBROUTINE) может быть вызван из PROGRAM. Однако никакой модуль не может вызвать головную программу.

П р и м е р.

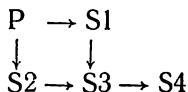
```
PROGRAM P
A=15.5
B=SIN(A)
PRINT 2, B
2 FORMAT (3X, 'B = ', F8.6)
END
```

Здесь работают два модуля: P и SIN(X). Из P вызывается SIN(X). Схематически:

$P \rightarrow \text{SIN}(X)$.

Вызов модулей может быть многоступенчатым, т. е., если P, S1, S2, S3, S4 — это имена модулей, то может

быть, например, вызов следующего вида



т. е. программа P вызывает и модуль $S1$ и модуль $S2$. Каждый из этих модулей вызывает модуль $S3$, который, в свою очередь, обращается к $S4$.

Но недопустима рекурсия при вызове модулей, т. е. последующие модули в схеме вызовов не могут обращаться к предыдущим. Например, не может быть такой схемы:

$$P \rightarrow S1 \rightarrow S2 \rightarrow S3 \rightarrow S1$$

Модульная структура программ позволяет отлаживать не всю программу сразу, а каждый модуль в отдельности.

П р и м е р. Пусть наша программа состоит из трех модулей P , $S1(T)$ и $F(X)$.

```
PROGRAM P
  CALL S1(T)
END
SUBROUTINE S1(T)
  A=F(X)
  RETURN
END
FUNCTION F(X)
  RETURN
END
```

Схема вызовов модулей следующая:

$$P \rightarrow S1 \rightarrow F$$

Как отлаживать эту программу, если модули P , $S1$ и F достаточно сложные? Существует два метода отладки: *нисходящий* и *восходящий*.

1. Нисходящим методом отладки пользуются при создании больших программных комплексов. Сначала отлаживают **PROGRAM**. При этом вместо всех остальных

модулей вставляют «заглушки», т. е. оставляют только заголовки, пишут необходимые операторы присваивания, затем RETURN и END.

```
Например,  
SUBROUTINE S1(T)  
T=0.5  
RETURN  
END  
FUNCTION F(X)  
F=2.  
RETURN  
END
```

Теперь можно отлаживать PROGRAM, заранее зная результаты работы остальных модулей.

После того как отлажена PROGRAM, приступают к отладке S1(T). Убирают «заглушку» S1 и вставляют в программу истинный модуль S1, а вместо F оставляют «заглушку».

```
FUNCTION F(X)  
F=2.  
RETURN  
END
```

После того как будет отлажен модуль S1, вместо «заглушки» вставляют исходный текст FUNCTION F(X) и отлаживают этот, самый нижний уровень.

При этом приходится отлаживать на каждом этапе один единственный модуль (остальные либо уже отлажены, либо заменены «заглушками»). Кроме того, часто в процессе отладки выявляются дополнительные требования к остальным модулям. Например, часто вообще модули нижнего уровня пишут только после того, как отладили предыдущие.

2. В восходящем методе отладки начинают с самого нижнего уровня. В нашем примере — это отладка FUNCTION F(X). Для этого пишут максимально простую PROGRAM, обращающуюся к F(X) и печатающую результат.

После того как программист убедится в правильности работы F(X), он начинает отлаживать S1. Для этого он пишет новую PROGRAM, вызывающую только S1 и печатающую ее результат, но уже в S1 работает отлаженная F. Таким образом, по отлаженному нижнему уровню отлаживается верхний уровень.

Модульная структура программы позволяет также поручать составление сложной программы нескольким лицам. Каждый из участников составляет и отлаживает один или несколько модулей, оформленных в виде FUNCTION или SUBROUTINE.

Наиболее часто используемые модули объединяют в *библиотеки подпрограмм*. Библиотечные модули заранее отлажены и постоянно присутствуют в памяти ЭВМ. Если в программе есть обращение к каким-либо библиотечным модулям, то они во время загрузки программы в память ЭВМ «пристраиваются» к тем модулям, которые написал сам программист. Таким образом, во время работы программы в одной области памяти находятся все модули, используемые в этой программе.

Библиотеки программ насчитывают несколько сотен модулей. Так, библиотека программ общего назначения, используемая в ОИЯИ, содержит около 400 модулей.

2.2. Как работает транслятор?

Программа, написанная на фортране и введенная в машину, должна быть преобразована транслятором в последовательность машинных команд.

Рассмотрим в общих чертах, как происходит процесс работы транслятора. Знакомство с этим процессом может оказаться полезным как при составлении программ, так и при их отладке.

Рассмотрим процесс работы транслятора на конкретном примере. Пусть имеется такая программа:

```
PROGRAM SUMMA
N=100000
S=0.
DO 1 K=1, N
1 S=S+1./K
PRINT 2, S
2 FORMAT (1X, 2HS=, E12.6)
END
```

Первый выполняемый оператор ($N=100000$) содержит переменную N и константу 100 000. В одну ячейку транслятор занесет значение константы, другую ячейку отведет для переменной N , запомнит номера этих ячеек и использует при составлении машинной команды либо нескольких команд (в зависимости от типа ЭВМ), реализующих засылку 100 000 в ячейку N .

Оператор $S=0$. транслируется аналогично предыдущему. Если бы вместо $S=0$. было написано $S=0$ (т. е. указан «целый» нуль), то для выполнения такого оператора потребовалось бы больше машинных команд, чем для первого. Дело в том, что типы переменной и константы в случае $S=0$ не совпадают. Поэтому перед выполнением операции засылки нуля машина должна преобразовать «целый» нуль в «вещественный». Чтобы не удлинять машинную программу, следует следить за соответствием типов констант и переменных.

В операторе $DO\ 1\ K=1, N$ указана метка последнего оператора цикла. Этот оператор появится в программе позже, а пока транслятор заносит метку 1 в особый список. При трансляции последующих операторов будет проверяться, не совпадает ли метка оператора с меткой 1 из списка. В случае совпадения оператор считается последним в цикле. Может случиться, что в программе не окажется оператора с меткой 1. Тогда, дойдя до END , транслятор выдаст диагностику об отсутствии последнего оператора цикла DO . В рассматриваемом операторе $DO\ 1\ K=1, N$ указана константа 1 и переменные K и N . Константу 1 транслятор засылает в ячейку памяти, для K отводит свою ячейку, а для N ячейка была уже отведена ранее.

Следующий оператор $1\ S=S+1./K$ вычисляет значение S и одновременно является последним оператором цикла. В этом операторе встретилась новая константа 1. (иного типа, чем 1), которую транслятор засылает в отведенную ячейку. Переменные S и K уже встречались, и в машинных командах можно использовать соответствующие номера ячеек. После трансляции оператора $1\ S=S+1./K$ будут формироваться команды, проверяющие условие окончания цикла.

Оператор $PRINT$, который указан после цикла, будет транслироваться несколько иначе, чем предыдущие. Дело в том, что для вывода каждого числа на печать надо выполнить довольно много машинных команд. Надо преобразовать каждое выводимое число из машинного представления в десятичное в соответствии со спецификацией формата. Необходимо также указать команды, управляющие работой печатающего устройства. Поэтому для вывода чисел на печать используется заранее составленная и находящаяся в машине программа. Транслятор же формирует только команды обращения к указанной программе. Эти команды определяют, из каких ячеек

памяти надо взять информацию для вывода на печать. В них указывается также, где находится оператор **FORMAT**, соответствующий данному оператору **PRINT**.

Оператор **FORMAT**, в отличие от предыдущих, не является выполняемым оператором. Поэтому ему не будут соответствовать никакие машинные команды. Однако определенный участок памяти для него будет отведен. Символы, из которых состоит оператор **FORMAT** (т. е. все, что начинается с открывающей скобки), будут преобразованы транслятором в машинное представление и размещены в одной или нескольких ячейках памяти. Например, на БЭСМ-6 в каждой ячейке будет размещено по 6 символов, так что содержимое данного оператора **FORMAT** (15 символов) займет 3 ячейки памяти. На ЕС ЭВМ под каждый символ будет отведен один байт памяти, т. е. всего 15 байт.

Все операторы **FORMAT** транслятор выносит в конец программы, располагая их вместе с константами и ячейками, отведенными под переменные и массивы.

Оператор **END** определяет конец программы. Встретив этот оператор, транслятор производит оформление результата трансляции.

2.2.1. Ошибки, обнаруживаемые транслятором. В программе, предъявленной для трансляции, могут быть ошибки. Те из них, которые связаны с нарушением правил языка фортран, будут обнаружены транслятором.

В каждом трансляторе предусмотрена выдача сообщений (диагностики) об ошибках, допущенных в программе. Диагностика может выдаваться либо сразу после распечатки текста каждого ошибочного оператора, либо по окончании трансляции всей программы для всех ошибочных операторов сразу. В последнем случае каждый диагностический текст снабжается номером строки программы, к которой он относится.

Содержание текста диагностики может либо непосредственно указывать на характер ошибки (например, «незакрытые скобки»), либо определять только номер ошибки, по которому в специальном справочном пособии можно найти соответствующий текст.

На БЭСМ-6 диагностика печатается сразу после ошибочно выполняемого оператора в виде текста, определяющего характер ошибки. Такой способ очень удобен для пользователя, поскольку вся информация об ошибках находится под рукой. Если ошибки допущены в невыполняемых операторах (**COMMON**, **DIMENSION**,

DATA и т. д.), то диагностика выдается после последнего невыполняемого оператора. Рассмотрим некоторые особенности, связанные с выдачей диагностики.

Многие неопытные программисты иногда предъявляют претензии к транслятору по поводу несоответствия текста диагностики характеру допущенной ошибки. Такое несоответствие объясняется тем, что транслятор реагирует не на причину ошибки, а на ее последствия. Пусть, например, мы пропустили знак $+$ в записи оператора $A=B+1.5E3$. В результате получился оператор присваивания $A=B1.5E3$, в правой части которого указано выражение, содержащее недопустимый символ «десятичная точка». В диагностике и будет сказано, что незаконно поставлена десятичная точка.

Необходимо учитывать также, что многие операторы программы взаимосвязаны. Например, если в левой части оператора присваивания указана переменная с индексами, то в программе должен быть описан соответствующий массив. Если описание массива отсутствует или сделано с ошибкой, то использование переменной с индексами становится незаконным и будет сопровождаться диагностикой. Такая диагностика называется *наведенной*.

Обычно транслятор выдает диагностику по первой ошибке, обнаруженной в операторе. При этом ошибочный оператор исключается из программы, что может вызвать наведенную диагностику по другим операторам.

Рассмотрим теперь несколько примеров.

```
SUBROUTINE FC
  DIMENSION A (20)
  A (1)=5.3
  DO 5 I=2, 20
    5 A(I)=A (I-1)**2+1.
  PRINT 23, A
  23 FORMAT (10(1X, F10.5))
  END
```

Здесь ошибка допущена при описании массива A : в операторе DIMENSION вместо буквы S указана буква T. Поэтому оператор DIMENSION A (20) будет признан неопознанным. В результате описание массива A будет как бы отсутствовать. Это повлечет за собой выдачу диагностики по оператору $A(1)=5.3$ и оператору счетки 5. Таким образом, в результате единственной ошибки появится сообщение транслятора о трех ошибках.

Иногда в результате допущенной ошибки может измениться смысл оператора. В этом случае транслятор уже не сможет ничего заметить, так как нарушений правил фортрана не происходит. Вот пример такого рода ошибки.

```
PROGRAM SEQUEN
A=1.
DO 7 I=1.20
7 A=A+1./A**2
PRINT 36, A
36 FORMAT (1X, E12.6)
END
```

Результат, выданный программой, был равен 2. Это свидетельствует о том, что цикл был выполнен, судя по всему, один раз, а не 20 раз, как задумано автором программы. Ошибка допущена в операторе DO, где вместо запятой указана точка. Почему же транслятор не заметил этой ошибки? А потому, что теперь вместо оператора DO получился оператор присваивания, в левой части которого указана переменная DO 7 I, а в правой части — константа 1.20. Заметим, что транслятор не принимает во внимание пробелы, указанные в записи оператора.

Коснемся теперь других ошибок, которые транслятор обнаружить не может. Пусть, например;

```
PROGRAM DIMEN
DIMENSION A (100)
DO 3 I=1, 100
3 A(I+1)=0.
PRINT 16, A
16 FORMAT (2X, 20F5.0)
END
```

Здесь при $I=100$ произойдет занесение нуля в элемент A (101). Но массив A содержит только 100 элементов. В результате будет «испорчена» ячейка памяти, следующая за той, которая соответствует элементу A (100). Последствия этой ошибки могут быть самыми разными — все зависит от того, какую роль в программе играет «испорченная» ячейка.

Может ли транслятор обнаружить такую ошибку? Как правило, не может, так как значение индекса определяется в процессе счета. Однако в некоторых трансляторах предусмотрен специальный режим работы, когда в транслированную программу вставляются дополни-

тельные команды. Они проверяют (в процессе счета), находится ли значение индекса в допустимых пределах. При появлении недопустимого значения индекса выдается диагностика (во время счета). Однако такая проверка может значительно замедлить работу программы, поэтому указанный режим трансляции обычно используют лишь при отладке программ.

Особенностью языка фортран является автономность каждого модуля программы (PROGRAM, SUBROUTINE и FUNCTION). Каждый из них транслируется независимо от других. Поэтому ошибки, связанные со взаимодействием модулей не могут быть обнаружены транслятором. Вот пример такой ошибки.

```
PROGRAM DUBNA
C=3.14
CALL LCT (C, 2.)
A=SQRT (2.)
PRINT 12, A
12 FORMAT (2X, F12.8)
END
SUBROUTINE LCT (A, X)
X=0.
IF (A.GT.0.) X=-1./A
RETURN
END
```

Подпрограмма LCT выдает в качестве результата либо нуль (если $A \leq 0$), либо $-1./A$ (если $A > 0$). Этот результат заносится в ячейку памяти, соответствующую формальному параметру X. В PROGRAM в качестве ячейки для результата указана та, в которой находится константа 2. Поэтому после выполнения оператора CALL в этой ячейке будет уже не 2., а $-1./3.14$. Поэтому вместо извлечения корня из числа 2. мы будем извлекать его из отрицательного числа, что приведет к прекращению счета.

В заключение заметим, что устранение ошибок в программе достигается тщательной отладкой. При этом важно, чтобы при отладке были проверены все варианты работы программы.

2.3. Некоторые поучительные примеры

Здесь будут рассмотрены примеры, которые на первый взгляд могут показаться простыми. Однако простота их обманчива. Начинающий программист обычно не

улавливает скрытых трудностей, заключенных в подобных задачах. В результате самое простое и очевидное решение может оказаться неудовлетворительным

З а д а ч а 1. Вычислить a_{10000} , если $a_1=1$ и

$$a_{n+1} = a_n + \frac{1}{a_n^2 + 1}, \quad n \geq 1.$$

Обычно начинающие программисты заказывают массив длины 10000, каждый элемент которого предназначен для хранения соответствующего члена последовательности. Получается примерно такое решение задачи:

```
PROGRAM A 10000
DIMENSION A (10000)
A (1)=1.
DO 1 I=2, 10000
1 A (I)=A (I-1) + 1./ (A (I-1)**2+1.)
PRINT 10, A (10000)
10 FORMAT (1X, 7HA10000=, E12.6)
END
```

Решение, конечно, верное. Однако здесь слишком нерационально используется память ЭВМ. В самом деле, нам нужно получить в качестве результата всего одно число. Это число вычисляется рекуррентно, причем на каждом шаге производится вычисление очередного члена последовательности через предыдущий. Все остальные члены последовательности, которые вычислялись на предыдущих шагах, уже не нужны. Отсюда следует, что нет необходимости хранить все члены последовательности. Достаточно иметь лишь тот член последовательности, который необходим на данном шаге вычисления.

Таким образом, вместо массива достаточно иметь одну переменную, содержащую значение текущего члена последовательности. Вот так выглядит новое, более рациональное решение.

```
PROGRAM ANEW
A=1.
DO 2 I=2, 10000
2 A=A+1./ (A**2+1.)
PRINT 20, A
20 FORMAT (1X, 2HA=, E12.6)
END
```

З а д а ч а 2. Вычислить $\sqrt{x^2 + y^2}$.

На первый взгляд, все решается просто:

```

PROGRAM SQRTXY
  READ 3, X, Y
  3 FORMAT (2E10.3)
  SQ=SQRT (X**2+Y**2)
  PRINT 6, SQ
  6 FORMAT (2X, E10.3)
  END

```

Однако такая программа может вычислить SQ не для всех допустимых значений X и Y. Например, при $|X| \geq 10^{10}$ машина БЭСМ-6 прекратит выполнение программы, так как $(10^{10})^2 = 10^{20}$, что превышает максимально допустимое число 10^{19} (переполнение). На ЕС ЭВМ переполнение произойдет при $|X| > 10^{38}$. Однако $\sqrt{x^2 + y^2} \leq \sqrt{2} \max(|x|, |y|)$, т. е. корень может быть вычислен и при гораздо больших значениях $|x|$ и $|y|$.

Преобразуем выражение $\sqrt{x^2 + y^2}$ следующим образом:

если $|x| > |y|$, то $\sqrt{x^2 + y^2} = |x| \sqrt{1 + \left(\frac{y}{x}\right)^2}$;

если $|x| \leq |y|$ и $|y| \neq 0$, то $\sqrt{x^2 + y^2} = |y| \sqrt{1 + \left(\frac{x}{y}\right)^2}$;

если $|x| \leq |y|$ и $|y| = 0$, то $\sqrt{x^2 + y^2} = 0$.

Такое решение позволяет получить результат как при больших, так и при малых значениях x и y . Например, при $|x| < 10^{-10}$ и $|y| < 10^{-10}$ старое решение выдало бы на БЭСМ-6 «машинный» нуль, новое же выдаст верный результат.

```

PROGRAM SQNEW
  READ 13, X, Y
  13 FORMAT (2E10.3)
  ABSX = ABS(X)
  ABSY = ABS(Y)
  IF (ABSX - ABSY) 2, 2, 1
  1 SQ = ABSX * SQRT(1. + (Y/X)**2)
  GO TO 10
  2 IF (ABSY) 11, 12, 11
  11 SQ = ABSY * SQRT(1. + (X/Y)**2)

```

Вводим новые
переменные
ABSX и ABSY

```

GO TO 10
12 SQ = 0.
10 PRINT 15, SQ
15 FORMAT (1X, 3HSQ=, E10.3)
END

```

$$\sqrt{x^2 + y^2} = 0$$

Новое решение, конечно, значительно сложнее. Однако оно позволяет получать результат практически при всех допустимых значениях аргумента.

Задача 3. Вычислить меньший по модулю корень уравнения $x^2 + 2000x + 1 = 0$.

Для решения этой задачи можно, конечно, воспользоваться формулой

$$x = -1000 + \sqrt{1000^2 - 1}.$$

Однако здесь скрыта вот какая неприятность. Значение

$$\sqrt{1000^2 - 1} \approx 999,9995,$$

т. е. будет весьма близко к 1000. Разность $1000 - 999,9995$, равная 0,0005, таким образом, будет иметь точность на 6 знаков меньше, чем исходные числа.

На ЕС ЭВМ, для которой точность вычисления составляет 7 верных знаков, в разности останется не более одного верного знака. На БЭСМ-6, которая позволяет получать 12 знаков, у результата все же останется 6 верных знаков. Однако желательно получить точность, соответствующую возможностям машины.

Выход из создавшегося положения состоит в том, чтобы, преобразовав «неудачную» формулу, исключить вычитание двух близких величин. Умножив и разделив x на число $1000 + \sqrt{1000^2 - 1}$, получим

$$x = \frac{-1}{1000 + \sqrt{1000^2 - 1}}.$$

В этом случае потери точности уже не будет.

Приведенные примеры специально подобраны таким образом, чтобы обратить внимание начинающего программиста на некоторые «скрытые» трудности, встречающиеся при фактической реализации программ.

2.4. Работа с магнитными лентами и дисками

Оперативная память часто бывает недостаточной для размещения всех данных и программ задачи. Тогда прибегают к использованию внешней памяти: магнитных дисков, лент, барабанов.

Каков объем внешней памяти ЭВМ? Одна из реальных ЭВМ БЭСМ-6 имеет такой набор устройств внешней памяти:

- 10 магнитных барабанов по 32К слов каждый;
- 8 магнитных дисков ЕС 5052 по 7,25 Мбайт;
- 8 магнитных дисков ЕС 5061 по 29 Мбайт;
- 16 магнитофонов БЭСМ-6 (до 700К слов на каждой ленте);
- 16 магнитофонов ЕС 5012 (до 1500К слов на ленте).

На одной из реальных ЭВМ ЕС-1040 имеется:

- 8 магнитных дисков ЕС 5052 по 7,25 Мбайт;
- 8 магнитных дисков ЕС 5061 по 29 Мбайт;
- 6 магнитофонов ЕС 5012 (до 20 Мбайт на ленте).

Магнитофоны ЕС 5012 допускают запись информации с различной плотностью. Для работы с внешней памятью пакет задачи должен содержать соответствующие управляющие карты, из которых операционная система узнает, какие ленты и диски нужны для данной программы.

2.4.1. Как пользоваться магнитными лентами и дисками на БЭСМ-6? Программист может использовать внешнюю память в двух различных режимах. Первый — это режим, при котором информация, записанная данной программой, хранится только во время работы этой программы. По окончании работы память отдается в распоряжение следующей работающей программе. Если такой режим программиста устраивает то ему не нужно иметь своих лент и дисков. В распоряжении операционной системы есть «рабочие» ленты либо «рабочие» файлы (участки) на диске, которые могут быть использованы любой работающей в данный момент программой. При этом операционная система следит за тем, чтобы разные задачи не портили информацию друг друга до окончания счета.

Для заказа «рабочей» ленты служит управляющая карта

*TAPE : n — SCRATCH, Fm, W

где n — произвольный номер от 1 до 4095, SCRATCH — имя «рабочей» ленты (*всегда только такое*), m — номер, используемый в программе в ператорах ввода-вывода (см п 2.4.2).

В процессе обработки управляющей карты операционная система ставит в соответствие номеру m то реальное устройство, на котором установлена «рабочая» лента.

Например, карта заказа «рабочей» ленты 1 может иметь вид:

*TAPE : 4 — SCRATCH, F1, W

Если операционная система предоставляет «рабочий» файл на диске, то карта заказа выглядит несколько иначе:

*FILE : SCRATCH, Fm, W, p

Новым здесь является только параметр p; для работы вашей программы операционная система должна отвести на диске p зон

(объемом по 1К слов). Заметим, что число p задается в восьмеричной системе счисления.

Приведем пример карты заказа «рабочего» файла на диске:
*FILE : SCRATCH, F1, W, 300

Вторым режимом работы с внешней памятью является такой, при котором информация, записанная задачей, сохраняется для дальнейшего использования. В таких случаях программист должен иметь свою «личную» ленту либо «личный» файл на диске.

Для того чтобы начать работать с новой лентой, ее надо дать операторам ЭВМ для разметки. В сопроводительной инструкции необходимо указать имя, которое вы выбрали для своей ленты. Имя должно состоять не более, чем из шести произвольных букв и цифр, например ЛЕНТА1.

Операторы ЭВМ нанесут на бобину ленты номер и разметят саму ленту. ЭВМ в процессе разметки запишет на определенных участках имя и номер бобины вашей ленты. Разные бобины имеют разные номера и по этим номерам операторы ЭВМ различают магнитные ленты.

В пакете задачи «личные» ленты описываются управляющими картами:

*TAPE : $p - a, Fm, S$

где p — номер бобины, a — имя, указанное при разметке, m — имеет прежний смысл, S — имеет значение R либо W (от английских слов read и write); R позволяет только читать информацию с ленты, но писать туда запрещает, W разрешает и читать, и писать информацию на ленту.

Имя ленты является своеобразным паролем: без согласия владельца никто не может воспользоваться его лентой.

Для работы с «личным» файлом на диске надо знать имя диска, на котором вам выделен файл, и «фамилию» владельца диска. «Фамилией» может служить любая комбинация не более чем из шести букв и цифр. Диск следует описать управляющей картой

*DISC : $p - c, q$

где p — номер дискового пакета (аналог номера бобины магнитной ленты), c — имя диска, q — «фамилия» владельца.

После этой карты ставится карта заказа файла на данном диске:

*FILE: a, Fm, S

где a — имя того файла, который вы можете использовать, m — номер, используемый в операторах ввода-вывода, S — режим R либо W .

Например, управляющие карты

*DISC : 7 — MOSKWA, MAZNY

*FILE : TASK, F2, W

ваказывают файл с именем TASK на диске с именем MOSKWA владельца MAZNY; файл имеет номер 2, допускается и чтение, и запись информации в файл TASK. На дисковом пакете написан номер 7.

Одна задача может пользоваться несколькими внешними запоминающими устройствами. Каждому из них в управляющих картах соответствует свой номер. Поэтому, выбирая эти номера произвольно, надо следить, чтобы параметры m были все различными в картах заказа запоминающих устройств.

2.4.2. Как из фортранной программы обратиться к ленте или диску? Программа имеет право использовать до 15 внешних запоминающих устройств на лентах и дисках. С точки зрения фортранной программы файлы на дисках и ленты совершенно идентичны.

Оператор WRITE (m)S позволяет записать информацию из оперативной памяти на ленту или диск. Здесь m — номер от 1 до 15, соответствующий номеру устройства, заказанному в управляющей карте *TAPE либо *FILE, S — список выводимых данных.

Например, если мы хотим записать информацию из ячеек A, B и C в файл TASK, то оператор WRITE должен выглядеть так:

WRITE (2) A, B, C

Здесь $m=2$, так как в управляющей карте

*FILE : TASK, F2, W

выбрано устройство номер 2 (F2).

После выполнения этого оператора содержимое ячеек A, B и C запишется на диск и создастся одна «запись». При втором обращении к оператору WRITE (m) с тем же номером m на диске (ленте) создается вторая «запись» и т. д.

Оператор READ (m) A, B, C читает содержимое одной «записи» устройства с номером m и засылает в ячейки A, B и C оперативной памяти. Повторное выполнение оператора READ (m) с тем же номером m читает информацию из следующей записи и т. д. При этом считывающая головка устанавливается за концом последней прочитанной записи. Чтобы прочитать первую запись, надо вернуть головку к началу этой записи.

Оператор REWIND m осуществляет возврат головки устройства с номером m к началу первой записи.

Если надо вернуться назад только на одну запись, следует использовать оператор BACKSPACE m ; если на несколько записей, то приходится выполнить столько же раз оператор BACKSPACE m .

Например, на устройство с номером 5 произведено три записи, после чего надо содержимое второй записи занести в ячейки A, B, D:

BACKSPACE 5

BACKSPACE 5

READ (5) A, B, D

Описанные операторы имеют один и тот же вид для ЕС ЭВМ и БЭСМ-6, но в пакете задачи ЕС ЭВМ необходимо иметь соответствующие управляющие карты (см. п. 1.10.4).

2.5. Начинаящему пользователю терминала

Первые вычислительные машины были полностью во владении программиста. Он получал всю машину в свое распоряжение и, сидя за пультом, следил за прохождением задачи. При поиске ошибки программист часто использовал одноктактный режим, т. е. машина останавливалась после выполнения каждой команды, и по лампочкам пульта можно было видеть, какие числа участвуют в операции и какой получается результат.

Довольно скоро стала очевидна низкая эффективность такого использования ЭВМ: машина простаивала, пока программист обдумывал свой следующий шаг.

Как только встал вопрос о повышении эффективности использования машин, программист у пульта машины был заменен оператором ЭВМ.

В настоящее время на многих вычислительных машинах работа происходит следующим образом. Операторы ЭВМ записывают на магнитную ленту ввода (либо на файл диска) пакеты задач, отданные программистами на счет. Пакеты поочередно обрабатываются на ЭВМ, а результаты записываются на магнитную ленту (файл диска) вывода. Затем информация выдается на печать (либо на перфорацию).

Эти усовершенствования повысили эффективность использования самой машины, но труд программиста стал менее интенсивным и потерял часть своей увлекательности. Если раньше процесс отладки программы состоял практически только из поиска ошибки и ее устранения, то теперь — это в основном процесс ожидания выдачи с машины. Найдя ошибку в программе и внося изменения, программист получает результаты часто только на следующий день: Такая недопустимо низкая производительность труда программиста послужила причиной того, что в его руках оказались устройства, создающие иллюзию работы за пультом ЭВМ — *выносные пульта* или *терминалы*. Они подключаются к машине и устанавливаются как в непосредственной близости от ЭВМ, так и на значительных расстояниях от нее.

Работа пользователя за терминалом организована по-разному на разных типах ЭВМ и в разных операционных системах. Но почти все выносные пульта позволяют ввести информацию в ЭВМ, прочитать информацию с магнитных лент (дисков), создать пакет задачи, запустить задачу на счет.

Первые терминалы разрабатывались в расчете на использование пишущих машинок. В настоящее время в качестве базы выносных пультов служат в основном различные типы алфавитно-цифровых дисплеев, представляющих собой телевизионный экран с клавиатурой, на которой можно набирать информацию.

В нашей стране одним из таких распространенных дисплеев является Видеотон 340. Это устройство удобно в эксплуатации и надежно в работе.

2.5.1. О дисплее Видеотон 340 (ЕС 7168). Этот дисплей имеет следующие данные:

- полезная площадь экрана — 200×140 мм²,
- количество строк — 16,
- количество символов в строке — 80,
- емкость буферной памяти — 1280 символов,
- формат изображения — точечная матрица размерностью 5×7 в системе телевизионной развертки,
- набор знаков: 26 латинских букв, 10 цифр, 28 специальных знаков, 31 русская буква (кроме ё и ъ).

Возможности редактирования:

- стирание изображения с экрана,
- установка метки в 1-ю позицию 1-й строки,
- перевод строки,
- перемещение метки в 4-х направлениях,
- замена, вставка, стирание символа,
- замена, вставка, стирание строки,
- подчеркивание,
- подъем кадра на одну строку вверх при заполнении последней строки.

Режимы работы:

- OFF LINE (автономный),
- ON LINE (посимвольный обмен с машиной),
- SEND (передача массивов информации в машину),
- PRINT (печать с экрана на АЦПУ дисплея).

Этот дисплей выпускается в двух основных вариантах: вариант с латинским алфавитом и вариант с русским и латинским алфавитами. Остановимся на описании второго.

Все знаки нанесены на клавиши. Дисплей имеет собственную (буферную) память, содержимое которой высвечивается на экране. В памяти дисплея может накапливаться и храниться информация во время работы программиста.

При нажатии клавиш вырабатывается соответствующий код, который либо попадает в память дисплея, либо управляет его работой, либо передается сразу в ЭВМ.

Кодировка символов дисплея Видеотон 340

A	101	A	141	Ю	140	.	056
B	102	Б	142	0	060	/	057
C	103	Ц	143	1	061	→	010
D	104	Д	144	2	062	←	020
E	105	Е	145	3	063	↑	031
F	106	Ф	146	4	064	↓	032
G	107	Г	147	5	065	IC	034
H	110	Х	150	6	066	DC	035
I	111	И	151	7	067	DL	036
J	112	Й	152	8	070	ERASE	037
K	113	К	153	9	071	TAB	011
L	114	Л	154	└	040	LINE FEED	012
M	115	М	155	:	072	IL	013
N	116	Н	156	;	073	HOME	014
O	117	О	157	<	074	RETURN	015
P	120	П	160	=	075	PRINT	027
Q	121	Я	161	>	076		
R	122	Р	162	?	077		
S	123	С	163	!	041		
T	124	Т	164	»	042		
U	125	У	165	‡	043		
V	126	Ж	166	\$	044		
W	127	В	167	%	045		
X	130	Ь	170	&	046		
Y	131	Ы	171	'	047		
Z	132	З	172	(050		
[133	Ш	173)	051		
\	134	Э	174	*	052		
]	135	Щ	175	+	053		
^	136	Ч	176	,	054		
_	137	DEL	177	—	055		

Кодировка букв и цифр совпадает с кодировкой ISO.

2.5.2. Клавиши установления режима работы. Дисплей может работать в режиме OFF LINE — автономном режиме, при котором набираемая на клавиатуре информация попадает в память дисплея. При этом обмен информацией с ЭВМ блокируется.

Нажатие клавиши «ON LINE» блокирует передачу с клавиатуры в память дисплея и разрешает обмен информацией с ЭВМ.

Клавиша «SEND» служит для того, чтобы набранную в режиме OFF LINE информацию, находящуюся в памяти дисплея (на экране), передать в ЭВМ. В этом случае связь с машиной односторонняя: разрешена передача информации из дисплея в ЭВМ, но блокирован прием.

Нажатием клавиши «PRINT» можно сбросить содержимое экрана на АЦПУ дисплея (алфавитно-печатающее устройство типа Видеотон 342), если оно подключено к данному дисплею.

Изобразим схематически разрешенные направления передачи информации стрелкой. Тогда различные режимы работы дисплея выглядят следующим образом:

OFF LINE: клавиатура → память дисплея.

ON LINE: клавиатура → ЭВМ, ЭВМ → память дисплея.

SEND: память дисплея → ЭВМ.

PRINT: память дисплея → АЦПУ дисплея.

Клавиша «OFF LINE» обладает наивысшим приоритетом. При нажатии на нее дисплей из любого другого режима переходит в режим OFF LINE.

В режимы ON LINE и SEND дисплей может быть переведен только из режима OFF LINE; в режим PRINT — как из OFF LINE, так и из ON LINE. Печать с экрана можно осуществить при подаче соответствующего кода (см таблицу кодировки) из ЭВМ. После окончания печати дисплей возвращается в тот режим, из какого он вошел в режим PRINT.

На алфавитно-цифровых клавишах изображены два символа: *верхний* и *нижний*. При нажатии любой клавиши генерируется нижний символ, если клавиша «LAT» нажата, и верхний символ, если клавиша «LAT» отжата.

Например, нажимая только клавишу «Г/Г», получаем символ Г; а «утопив» клавишу «LAT», получаем Г.

Для редактирования набранного на экране текста служат *клавиши редактирования*:

«ERASE» — сбрасывает содержимое экрана, очищает буферную память, устанавливает метку (курсор) в первую позицию первой строки;

«→» — перемещает метку на одну позицию вправо (из последней позиции строки — в первую позицию следующей строки, а из последней позиции последней строки — в первую позицию первой строки);

«←» — смещает метку на одну позицию влево (из первой позиции строки — в последнюю позицию предыдущей строки; из первой позиции первой строки — в последнюю позицию последней строки);

«↑» — переводит метку на строку вверх, и из первой строки — в последнюю;

«↓» — переводит метку вниз на одну строку, а из последней — в первую;

«LINE FEED» — переводит метку в первую позицию следующей строки, а из последней строки — в первую позицию первой строки;

«НОМЕ» — устанавливает метку из любого положения в первую позицию первой строки;

«ТАВ» — устанавливает метку в первую позицию следующей строки, либо за символ], если он присутствует в данной строке;

«ІС» — вставляется пробел на месте символа, указываемого меткой; вся строка, начиная с этого символа, сдвигается на одну позицию вправо (символ из последней позиции строки пропадает);

«DC» — символ, указываемый меткой, пропадает, а оставшаяся справа от него часть строки сдвигается на позицию влево (в последней позиции строки появляется пробел);

«ІL» — на месте строки, указанной меткой, образуется строка пробелов, а все строки, начиная с данной, сдвигаются на одну вниз (последняя строка экрана теряется);

«DL» — строка, указываемая меткой, пропадает, а строки, расположенные ниже, поднимаются на одну вверх (последняя строка экрана заполняется пробелами);

С помощью этих клавиш можно редактировать набранный текст. Покажем на примере, как это сделать.

Пусть на экране набран в режиме OFF LINE текст:

МОСКВСКАЯ ОБЛ,
УЛИЦА МОРЬКОГО,
ДОММ 6, КВ. 32

Очевидно, этот текст является адресом, в котором допущен ряд ошибок. Для исправления их надо сделать следующее.

1. Вставить после К букву О в первой строке.
2. После первой строки вставить название населенного пункта (например г. Жуковский).
3. Букву М заменить на Г в слове МОРЬКОГО.
4. Стереть вторую букву М в слове ДОММ.

Отредактируем текст.

1. С помощью клавиш «↑», «→», «↓», «←» установим метку под букву В, нажмем клавишу «ІС», затем «О».

2. Переведем метку на строку вниз клавишей «↓», нажмем «ІL» — образуется строка пробелов, на которой наберем

Г. ЖУКОВСКИЙ,

3. Подведем метку под букву М и нажмем клавишу «Г».

4. Чтобы стереть лишнюю букву М, достаточно подвести под нее метку и нажать клавишу «DC».

Чтобы стереть экран и начать набор нового текста, достаточно нажать клавишу «ERASE».

«ETX» — нажатием этой клавиши заканчивают текст, набранный для передачи в режиме SEND; иногда используют как «LINE FEED».

«MP ON» — в описываемых случаях должна быть отжата.

«ROLL» — нажатие этой клавиши позволяет писать информацию таким образом, что новая строка оказывается ниже предыдущей. Если на экране заполнены все строки, то при попытке записать новую информацию все строки поднимаются на одну вверх, первая пропадает и последняя освобождается для приема.

«REPT» — при нажатии этой клавиши одновременно с какой-либо другой, код последней выдается с частотой 10 гц.

«UL» — если нажата эта клавиша, то все набираемые символы появляются на экране с подчеркиванием.

«CTRL BLINK» — служит для отображения символов, хранящихся в памяти, но не изображаемых на экране. Например, LINE FEED отображается как буква J, HOME отображается как L; эти символы (J и L) вспыхивают на экране с частотой 5 гц.

2.6. Работа за терминалом ЭВМ БЭСМ-6 в операционной системе Дубна (вариант Мультитайп)

В качестве терминалов в операционной системе Дубна можно использовать дисплеи Видеотон 340, телетайпы и пишущие машинки Консул. Опишем подробно работу с дисплеем Видеотон 340.

Программист ведет работу с терминала сеансами в *диалоговом режиме*. На действия пользователя за терминалом ЭВМ реагирует сообщениями, выдаваемыми на экран. Пользователь, в свою очередь, с клавиатуры набирает приказы, исполняемые машиной.

Н а ч а л о с е а н с а.

1. Видеотон 340 следует включить в сеть и нажать клавишу «POWER». После того как на экране появится метка, надо нажать «ON LINE». Терминал готов к работе.

2. Нажать клавишу «пробел». Система выдаст на экран:

МУЛЬТИТАЙП / ДАТА ВЕРСИИ/
PASS:

3. В ответ следует набрать то же ключевое слово (шифр), какое вы используете в управляющей карте *PASS, и нажать клавишу исполнения «LF». ЭВМ засекретит ваш шифр и на экран выдаст звездочки *****. Если вы промедлите с набором очередного символа (более 5 секунд), то ЭВМ прервет сеанс и выдаст на экран

КОНЕЦ.

Если шифр набран неверно, то на экране появится сообщение:
НЕТ ТАКОГО!
КОНЕЦ.

Сеанс надо начинать заново.

4. Если ключевое слово набрано правильно (список ключей хранится в памяти ЭВМ), то на экране появится

БУФЕР:

В ответ следует набрать информацию о том буфере, на котором вы собираетесь работать. В качестве буфера может быть либо магнитная лента, либо файл на диске.

Информация о ленте набирается в том же виде, в каком ее заказывают в карте *TAPE: номер бобины, затем символ / и имя ленты. Например, лента на бобине 135, имя ее ABC155. Тогда на экране высвечивается

БУФЕР: 135/ABC155

Каждый приказ должен оканчиваться нажатием клавиши «LINE FEED».

Если в качестве буфера используется диск, то следует набирать после БУФЕР: обязательно пробел, затем номер пакета диска, символ /, имя диска, фамилию владельца файла, имя файла, W.

Например, пакет диска 153, имя диска MOSCOW, фамилия владельца PETROV, имя файла DUBNA1.

БУФЕР: 153/MOSCOW, PETROV, DUBNA1, W («LINE FEED»)

После нажатия клавиши «LINE FEED» приказ выполняется, но может появиться диагностика:

НЕТ ФАЙЛА — возможно, неправильно набрана информация о буфере;

ЗАНЯТ ФАЙЛ — диск либо лента, которую вы указываете, занята (с ней работают);

МЛ НЕТ — либо ваша лента еще не установлена операторами ЭВМ, либо ошибка в наборе.

2.6.1. Приказы, набираемые на клавиатуре.

1. *Приказ ВВОД N, M*, где N и M — восьмеричные номера. По этому приказу информация, набираемая далее (со следующей строки экрана), попадает в зону номер N вашего буфера и может занимать участок до M-й зоны включительно.

Например, ВВОД 1,4 — информация может занимать от 1-й до 4-й зоны буфера. Если номер M не указан, т. е. приказ имеет вид ВВОД N, то информация может занимать весь буфер, начиная с N-й зоны.

Если вы начали сеанс и машина сама выдала на экран ВВОД — наберите N, M и нажмите «LINE FEED». Метка экрана перейдет на новую строку, на которой появится трехзначный восьмеричный номер 001 — номер строки информации.

В случае ошибок при наборе этого приказа может появиться диагностика:

ОШ. КОНЦА
ОШ. НАБОРА
ОШ. БУФЕРА

Надо проверить правильность указываемых номеров зон (N и M) и снова набрать ВВОД \square N, M.

2 Приказ ВВОД + N, M продолжает запись информации на участок, начиная с N-й зоны (куда часть информации была уже записана ранее). Работает как приказ ВВОД N, M с той разницей, что нумерация строк идет не от 001, а от номера последней ранее записанной строки.

Набирая текст для записи в буфер, оканчивайте каждую неполную строку нажатием клавиши «LINE FEED». В ответ метка будет переходить на новую строку и устанавливаться после выданного машиной восьмеричного номера очередной строки. Каждая строка экрана соответствует одной перфокарте.

Помните, что при этом управляющие карты надо набирать сразу после выданных на экран номера строки и пробела, т. е. с 5-й позиции строки, а операторы фортрана — с 11-й позиции (5+6). Например:

055 $\square\square\square\square\square\square\square$ END

056 \square *EXECUTE

057 \square *END FILE

Если вы набираете на клавиатуре управляющие карты, то ЭВМ выдает «подсказки» — начало следующей управляющей карты. Например, после *NAME на экране появляется *PASS и т. д.

Не забывайте нажимать «LINE FEED» после набора каждой карты. Если вы обнаружили ошибку в набранном тексте, то можете поступить следующим образом. Если строка, в которой допущена ошибка, еще не закончена нажатием «LINE FEED», то нажмите клавишу «DL». При этом ЭВМ «забудет» набранную строку (но оставит ее на экране дисплея). Если вы нажмете клавишу «→», то ЭВМ «вспомнит» один символ и высветит его на экране. Многократное нажатие клавиши «→» восстанавливает символ за символом «забытой» строки. Так можно воспроизвести все правильные символы, а дойдя до ошибки, исправить ее нажатием соответствующей клавиши.

Если ошибку в строке текста вы обнаружили после нажатия «LINE FEED», то исправить ее можно способом, описанным ниже.

3. Для завершения ввода информации следует в очередной новой строке экрана, не набирая ни одного символа, нажать «LF» (ввод пустой строки).

4. Приказ ЛИСТ \square N позволяет прочитать информацию, начиная с N-й зоны буфера. Эта информация выдается на экран полностью, сколько бы зон она ни занимала.

В ответ на приказ ЛИСТ N машина начинает выдавать на экран информацию из зоны N. Если на клавиатуре нажата клавиша «ROLL» и отжата «MP ON», то по заполнении экрана (16 строк) вся информация сдвигается на строку вверх и нижняя строка освобождается для приема следующей строки информации. Если положение клавиш «ROLL» и «MP ON» иное, то строки принимаемой информации всегда заполняют экран сверху вниз, т. е. после 16-й строки экрана заполняется первая.

Если вы хотите прекратить дальнейшую выдачу на экран, можете нажать клавишу «LF» — вывод прекратится. Так же можно поступить, если вы не успели прочитать или проанализировать выданную на экран порцию информации. Продолжить вывод можно нажатием той же клавиши.

5. Если вы хотите видеть строки с нумерацией, то можете воспользоваться приказом ЛИСТ + N. Этот приказ выполняется аналогично приказу ЛИСТ □ N, но строки выдаются с теми номерами, какие им были приписаны при вводе.

6. Приказ KBYI позволяет получить информацию о задачах, находящихся в файле вывода.

В ответ ЭВМ выдает следующее:

⟨состояние⟩ ⟨номер ВЫВ⟩ ⟨номер ВВ⟩ ⟨имя⟩

Здесь ⟨состояние⟩ — символ, указывающий на состояние задачи:

* — задача посчитана и результаты выданы,

/ — задача уже поступила в файл вывода,

□ — задача выводит на АЦПУ;

⟨номер ВЫВ⟩ и ⟨номер ВВ⟩ — порядковые номера в выходной и входной очередях,

⟨имя⟩ — содержимое карты *NAME.

7. Приказ РЕЗ ⟨номер ВЫВ⟩ позволяет просмотреть выдачу вашей задачи.

Приказ РЕЗ + ⟨номер ВЫВ⟩ дополняет выдачу нумерацией строк.

8. Если вы хотите ускорить просмотр информации на дисплее, то можете пропустить часть материала, применив приказ

ДО □ ⟨образ⟩

Здесь ⟨образ⟩ состоит не более, чем из шести символов, соответствующих первым символам той строки, с которой начинается выдача на экран. Пробелы игнорируются. Например, в зоне № 5 буфера имеется пакет задачи:

*NAME PETROV

*PASS:P23456

*TIME:00.05

PROGRAM T

...

PR = H**2

...

END

*EXECUTE

*END□FILE

Если перед приказом ЛИСТ 5 набрать

ДО *, то начнется выдача с карты *NAME,

если ДО *Т,	»	»	»	*TIME,
если ДО PR,	»	»	»	PROGRAM Т,
если ДО PR = ,	»	»	»	PR = H**2,
если ДО *EN	»	»	»	*END□FILE,

и т. д.

Приказ ДО можно набирать до приказа ЛИСТ. Тогда сначала ищется строка, первые символы которой совпали с (образом), а затем с этой строки выводится информация на экран.

При желании вывод можно прерывать несколько раз (нажатием клавиши «LF») и применять приказы ДО с соответствующими (образами).

Приказ ДО без (образа) производит поиск по предыдущему (образу). Если нет строки, первые символы которой совпадают с (образом), то выдается диагностика НЕТ ОБРАЗА.

Вывод на экран можно прервать нажатием клавиши «LF», продолжить повторным нажатием «LF».

9. Приказ ДОН □N позволяет пропустить первые N—1 строк информации и начать просмотр с N-й строки.

П р и м е ч а н и е. Приказы ДО и ДОН не могут вернуть по тексту назад.

10. Если в зоне номер N имеется пакет задачи, то эту задачу можно запустить на счет *приказом ПУСК□N*. В ответ появится сообщение «СЧЕТ», если задача пошла на счет.

11. Приказом TNP□N задачу можно запустить на счет. При запуске задачи приказом TNP□N результат выдается на экран терминала, а не на АЦПУ.

П р и м е ч а н и е. Если машина задачу уже считает, буфером пользоваться нельзя, так как он находится в распоряжении задачи. После окончания счета можно продолжить сеанс. При этом машина попросит набрать буфер заново.

12. Приказом КАК можно узнать, что происходит с задачей в машине. В ответ ЭВМ сообщает

а) Ш (номер): {t комм.} — {t счет.}; {сообщ} {имя}. Здесь {номер} — номер шифра (канала), на котором считается задача, {t комм.} — {t счет.} — коммерческое и счетное время, {сообщ} — сообщение системы о причинах прекращения счета (например, НЕГОТ МФ и т. д.),

{имя} — информация с карты *NAME.

Если пользователем запущено несколько задач, то выдается информация обо всех этих задачах.

б) НЕТ ЗАДАЧИ — если счет уже окончен либо задача не попала в очередь.

Результаты счета выдаются на АЦПУ так же, как если бы задача была введена с перфокарт.

13. Приказ АЦПУ $\square M, K$ позволяет записать на буфер все, что выдает задача на АЦПУ. Здесь $M \div K$ — восьмеричные номера зон буфера, куда машина направит выдачу вашей задачи. В этом случае листинг вы не получите, но набрав ЛИСТ M, можете увидеть его содержимое на экране. Приказ АЦПУ M, K действует до следующего приказа АЦПУ N, L либо до конца сеанса. Перейти на работу с выдачей результатов на листинг можно приказом АЦПУ без указания N и L.

Пр и м е р. Пакет задачи набран в зоне номер 5, результаты надо поместить в 7-ю зону. Наберем следующие приказы:

АЦПУ 7
ПУСК 5
КАК

Пусть машина ответила

НЕТ ЗАДАЧИ

Чтобы увидеть результат, надо набрать:

ЛИСТ 7

Длинные строки АЦПУ (более 80 символов) выводятся в две строки дисплея. Вторая часть строки АЦПУ помечается на экране признаком продолжения =.

Для сокращения вывода разумно пользоваться приказом ДО {образ}. Например,

ДО *EXESU
ЛИСТ 7

Иначе на экран начнет выдаваться содержимое всего листинга, включая вашу фамилию (крупными буквами).

14. Ускорить просмотр текстовой информации можно с помощью приказа ФОР {список спецификаций}. Этот приказ управляет форматом выдаваемых строк текста. Спецификации в списке перечисляются через запятую. Существует три типа спецификаций.

S — любая группа пробелов заменяется одним пробелом в части строки, расположенной левее последнего отличного от пробела символа; такая «сжатая» строка справа дополняется до 80 символов пробелами. Например, строка в 80 символов

$\square\square\square\square\square A\square\square\square B C\square\square\square\square\square D...$

после сжатия по спецификации S будет иметь вид:

```
┐A┐BC┐Д...
```

Xп — не выводится на экран п соответствующих символов каждой строки. Если п не указано, то не выводится вся оставшаяся часть строки.

Ап — выдаются неизменными п соответствующих символов каждой строки. Если п отсутствует, спецификация действует до конца строки.

Пр и м е р. Пусть в зоне 15 находится пакет задачи:

```
*NAME PETROV
```

```
*PASS:P23456
```

```
*TIME:00.05
```

```
PROGRAM T
```

```
...
```

```
END
```

```
*EXECUTE
```

```
*END FILE
```

а) посмотрим первые 5 символов пакета, сжав группы пробелов:

```
ФОР S A5, X
```

```
ЛИСТ 15
```

На экране увидим:

```
*NAME
```

```
*PASS
```

```
*TIME
```

```
┐PROG
```

```
...
```

```
┐END
```

```
*EXEC
```

```
*END
```

б) выведем только с 6-го по 11-й символы каждой строки, оставив их неизменными:

```
ФОР X5, A6, X
```

```
ЛИСТ 15
```

Выдаваемый текст будет таким:

```
PETROV
```

```
:P2345
```

```
:00.05
```

```
┐PROGR
```

```
...
```

```
END
```

```
UTE
```

```
FILE
```

Если в списке спецификаций допущена ошибка, на экране появится только правильная часть списка. Пользователь может дополнить ее.

Отказаться от форматного вывода можно приказом ФОР

15. Приказом. РN можно выдать на перфоратор ЭВМ текстовую информацию из зоны с номером N. Например, выполнив приказ. Р15, получим на перфокартах пакет задачи предыдущего примера.

16. Приказ ТЕЛ *(произвольный текст)* передает произвольный текст на пульт оператора ЭВМ.

Например, ТЕЛ ПОСТАВЬТЕ, ПОЖАЛУЙСТА, ЛЕНТУ 135.

17. Приказом SEND□N, M можно ввести в ЭВМ не один символ, а целый массив. Здесь N и M — восьмеричные номера зон, ограничивающих участок буфера (ленты, диска), отведенный для вводимой информации.

Информацию, подлежащую передаче в ЭВМ, можно набрать на клавиатуре либо предварительно прочитать из некоторого участка буфера приказом ЛИСТ□K.

В случае набора текста на клавиатуре дисплея надо произвести следующие действия.

а) Дать машине приказ SEND□N, M.

б) Нажать клавиши «OFF LINE» (перевод в автономный режим) и, если нужно, «ERASE» (стирание экрана, перевод метки в первую позицию первой строки).

в) Набрать информацию.

г) В случае обнаружения ошибки воспользоваться клавишами «IC» (insert character), «DC» (delete character), «IL» (insert line), «DL» (delete line), перемещая метку нажатием клавиш «↑», «→», «↓», «←».

д) После проверки и исправления текста установить метку за последним символом и нажать клавишу «ETX». При этом в память дисплея заносится признак конца передаваемой информации. На экране этот символ не высвечивается.

е) Подвести метку под первую строку и нажать клавишу «SEND»; при этом в ЭВМ попадет вся информация от строки, указанной меткой до символа «ETX».

После окончания передачи дисплей остается в приказе SEND и можно передать продолжение предыдущего текста, выполняя все действия с пункта б).

ж) После передачи всего пакета набрать приказ %%%. Если дисплей после этого не войдет в режим диалога, нажмите клавиши «OFF LINE», «ERASE» и три раза клавишу «%».

Если вы хотите передать информацию (или часть ее), из зоны с номером K в зону с номером N, то выполните следующие действия.

1) Приказом ЛИСТ□К считайте с зоны К требуемую информацию на экран (можете нажатием клавиши «пробел» приостановить вывод). Учтите, что одна строка экрана при этом должна оставаться свободной.

2) Дайте приказ SEND□N, M (с первой позиции строки).

3) Нажмите клавишу «OFF LINE». Отредактируйте информацию (если это требуется). В конце текста проставьте символ ETX.

4) Подведите метку экрана под первую строку передаваемого текста и нажмите клавишу «SEND».

5) Наберите приказ % % % (см. ж)).

6. Для передачи продолжения текста из зоны К надо выполнить все действия, начиная с п. 1).

18. Приказом SEND+N, M можно продолжить в последующих сеансах запись информации в массив, расположенный с зоны N. Диагностика по поводу SEND имеет тот же смысл, что и по поводу ВВОД. После записи информации в буфер ее уже нельзя исправлять в режиме OFF LINE. Способы редактирования, используемые в этом случае, описаны ниже.

19. Заканчивать сеанс следует *приказом СТОП*. Если такой приказ не будет последним в сеансе, то информация на данном буфере может испортиться при работе следующего пользователя.

20. Задачу, запущенную на счет с терминала, можно снять *приказом ОТКАЗ*.

21. Если в фортранной программе, запущенной на счет, используется подпрограмма — функция IFPULT [13], то информация для нее передается *приказом ПУЛ□И*, где И — информация (восьмеричная константа), набираемая в том же виде, как для пакетной обработки.

Пример. Пусть программист, имеющий PASS PET456, хочет использовать в качестве буфера магнитную ленту с именем ABC123, намотанную на 135-ю бобину. Цель данного сеанса — ввести в 5-ю зону ленты информацию: 13.567, 0.125, 315.9 — и запустить на счет пакет, находящийся в 11-й зоне ленты.

Чтобы различать информацию, выдаваемую машиной и информацию, набираемую программистом, последнюю будем выделять курсивом. Название не алфавитно-цифровых клавиш, нажимаемых программистом в процессе работы, возьмем в скобки.

□

МУЛЬТИТАЙП /20/02/85/

PASS: PET456 (LINE FEED)

БУФЕР: 135/ABC123, W (LINE FEED)

SEND□5, 10 (LINE FEED) (OFF LINE) (ERASE)

13. 567 (LINE FEED)

0.125 (LINE FEED)

315.9 (LINE FEED)

(ETX) (↑) (↑) (↑) (SEND)
%%% (LINE FEED)
ЛИСТ 5 (LINE FEED)
13.567
0.125
315.9
ПУСК 11 (LINE FEED)
СЧЕТ
КАК (LINE FEED)
НЕТ ЗАДАЧИ
СТОП (LINE FEED)
КОНЕЦ

22. Если задачу запустить на счет *приказом* `TERLIN` (*N* — номер зоны, содержащей пакет), то можно организовать диалог между задачей и терминалом.

Для обмена информацией с терминалом служат следующие операторы фортранной программы:

`CALL TEROUT (A, n)` — выдача строки информации в коде ISO из задачи на терминал;

`CALL TERIN (A, N)` — прием строки информации с терминала.

Здесь *A* — имя массива, в котором находится передаваемый (принятый) текст; *n* — число ячеек, занимаемых этим текстом; *N* — переменная, значение которой после приема текста будет равно числу принятых машинных слов.

При обращении к `TEROUT (A, n)` число *n* должно задаваться в программе; при обращении к `TERIN (A, N)` должна указываться переменная *N*.

Появление на экране знака «!» означает готовность ЭВМ принять очередную информацию для `TERIN`. Эта информация набирается сразу после знака «!».

Примером программы, работающей в режиме диалога с терминалом, является информационно-справочная система для библиотеки стандартных программ БЭСМ-6. Если вы хотите воспользоваться этой системой, то введите в *N*-ю зону буфера следующий пакет задачи:

```
*NAME ...  
*PASS ...  
*TIME ...  
*DISC ...  
*FILE ...  
*LIBRARY : 2  
*MAIN INFLIB  
*EXECUTE  
INST.  
*END FILE
```

Здесь в картах *DISC и *FILE указывается диск и файл, на котором записана информационно-справочная система. Например, для БЭСМ-6 ОИЯИ эти карты выглядят так:

*DISC : 667 — SYSTEM, BESM6

*FILE: ISS, 41, R

После того как пакет набран, его следует запустить на счет приказом ТЕР N. Инструкция к дальнейшей работе будет выдана на экран дисплея. Появление знака «!» означает готовность системы к приему ваших приказов.

23. Приказ ОТЛ□N позволяет вести отладку программы, записанной на N-й зоне буфера. Приведем некоторые приказы отладки:

ЧИЗ A, B — засылка в переменную A программы B десятичного числа, набранного в следующий (за ЧИЗ A, B) строке по спецификации E,

ДЕС A, B — выдача на экран значения переменной A,

ОЧТ A, B — останов по чтению переменной A,

ОЗА A, B — останов по записи в переменную A,

ИДИ A, B — передача управления на метку A,

ИДИ — продолжение выполнения программы, прерванной отладкой,

ОСА A, B — останов по метке A.

Здесь B — имя отлаживаемой программы (подпрограммы), а A — идентификатор либо метка в этой программе.

При остановках ОСА, ОЧТ и ОЗА на экран выдается сообщение, в котором указаны адрес останова, команда, на которой произошел останов.

Если A — имя COMMON-блока или подпрограммы, то B должно быть символом *. Если A — переменная, используемая в программе P, то B — имя этой программы.

B — всегда уточнение (например, имя программы). Если B — число N+1, то оно трактуется как смещение A (N-й элемент A).

Например, прочитать содержимое 28-го элемента COMMON-блока C можно приказом

ДЕС□*C*, 29

Если B опущено, то сохраняется последнее значение B предыдущих приказов.

З а м е ч а н и я.

1. Название COMMON-блоков дополняется слева и справа звездочками

2. Метка оператора снабжается звездочкой слева (метка 5 переходит в *5).

3. Метка оператора FORMAT дополняется слева косой чертой (метка 6 переходит в /6).

4. Числа, заносимые по приказу ЧИЗ, должны быть вида $\pm m.n \pm E_r$.

5. Если предполагается отлаживать задачу с терминала и обращаться при этом к локальным (не только COMMON) переменным, то среди управляющих карт должна присутствовать карта
*CALL▯DEBUGER

Пр и м е р. Рассмотрим сеанс отладки программы с терминала. Пусть в зоне 5 буфера имеется пакет:

*NAME...

*PASS...

*TIME...

*CALL▯DEBUGER

PROGRAM T

A=1.5

1 DO 2 I=1, 10

A=A/I

2 B=A+B

3 CONTINUE

END

*EXECUTE

*END▯FILE

В процессе отладки предполагается:

1) выдать содержимое В после окончания цикла;

2) занести в ячейку А число 2,15;

3) передать управление оператору с меткой 1;

4) выдать содержимое В после окончания цикла.

На экране все действия отобразятся таким образом:

...

ОТЛ 5

СЧЕТ

ОСТАН. ПО НК 01000 ...

ОСА *3, Т

ADR. NNNNN *3 ...

NNNNN — адрес начала оператора с меткой 3

ИДИ

ОСТАН ПО НК NNNNN

ДЕС▯В

$\pm m.n \pm E_r$ (десятичное число — содержимое В)

ЧИЗ А

1+2.15E+0

ОСА *3

ADR. NNNNN *3

ИДИ▯*1

ОСТАН. ПО НК NNNNN

ДЕС▢В

±m.n±Er (десятичное число — содержимое В)

ИДИ

...

Пример. Ваша задача была ранее «выброшена» из машины с диагностикой КОНТР. КОМ. (контроль команды). Это может произойти, если на место программы попали числа. Адрес NNNNN, выданный на листинге вместе с такой диагностикой, соответствует той ячейке памяти, в которой оказалось число вместо команды программы. Очевидно надо определить, какой оператор программы заносит число в ячейку NNNNN. Для этого наберем приказы:

ОТЛ▢М

ОЗА▢NNNNN

На экране появится

ОСТАН NK XXXXX

Здесь XXXXX — адрес команды, «портящей» ячейку NNNNN. По листингу предыдущего запуска можно определить, какому оператору принадлежит адрес XXXXX (см. п. 1.10.5).

24 Если задача, считаемая в ЭВМ, введена не с терминала и на работу с терминалом не рассчитана, то *приказом ТЕР* можно перевести эту задачу в режим отладки с терминала.

Приказ ТЕР позволяет легко находить такие ошибки, поиск которых обычно отнимает довольно много времени и у пользователя, и у ЭВМ. Приведем примеры использования этого приказа.

Пример. У вас есть листинг предыдущего запуска задачи, из которого следует, что задача не может выйти из цикла. В данное время ваша задача считается на ЭВМ. Выполнив приказы

ТЕР

ШАГ

ЭВМ выдаст на экран

ОСТАН NK NNNNN

где NNNNN — адрес команды программы, выполняемой в момент срабатывания приказа ШАГ. Эта команда расположена, вероятнее всего, внутри неправильно запрограммированного цикла. Узнав этот адрес (NNNNN) и сопоставив его с информацией, выданной на листинг при загрузке программы (адрес начала программы и относительные адреса всех операторов), можно определить, какому оператору соответствует адрес NNNNN. Повторяя приказ ШАГ, можно достаточно точно локализовать ошибку.

25 Если задача вышла из режима диалога, то для его восстановления надо выдать *приказ ТЕР*.

26. Если надо освободить терминал, находящийся в режиме диалога с задачей, то следует выдать *приказ РЕТ*.

2.6.2. Редактирование с терминала текста, записанного на буфер. Помимо общих средств редактирования текстов [13] система имеет специальные средства, используемые только при работе с терминала.

Как было сказано выше, все строки текста при вводе в буфер терминала нумеруются системой. Каждой строке слева присваивается трехзначный восьмеричный номер (001, 002, 003 и т. д.). Эти номера можно увидеть, если прочитать зону N приказом ЛИСТ+N.

Существуют следующие приказы редактирования.

1) .=M, где M — номер зоны, куда будет записана отредактированная информация.

2) .+K — вставить после строки с номером K информацию, набираемую на клавиатуре. (Номера строк — восьмеричные числа.)

3) .—K, L — исключить из текста строки от K-ой до L-й включительно и вставить информацию, набираемую на клавиатуре. Если исключить только K-ю строку, то L можно опустить (.—K).

4) .RN, где N — номер зоны, начиная с которой расположен редактируемый материал. Этот приказ должен предшествовать приказам типа 2) и 3). Он может отсутствовать только в том случае, когда приказы редактирования находятся в конце редактируемой информации.

5) Приказ . . означает конец приказов редактирования.

Для запуска задачи на счет надо выполнить приказ

ПУСК K

где K — номер зоны, содержащей файл приказов редактирования.

Счет начинается с редактирования текста. Если редактируемый текст является пакетом задачи и расположен в той же зоне K, то сразу после конца редактирования начнется счет задачи.

6) Если в 5) запуск задачи на счет не нужен, то перед приказом ПУСК K надо выполнить приказ .S.

7) С помощью приказов

.LN

.S

ПУСК N

можно выдать на АЦПУ машины содержимое N-й зоны буфера с нумерацией строк. Это полезно для предварительного обдумывания редактирования текста.

Приведем пример редактирования текста. Пусть в зоне номер 2 буфера находится почтовый адрес:

001 МОСКОВСКАЯ ОБЛ

002 МОСКОВСКАЯ ОБЛ

003 УЛ. ГОРЬКОГО

004 ИВАНОВУ И. И.

Для исправления следует произвести следующие действия.

1) Стереть 2-ю строку и вместо нее записать название населенного пункта, например г. Дубна.

2) После 3-й строки записать номер дома и квартиры, например д. 6, кв. 32

Отредактированный текст запишем в 15-ю зону буфера. На экране дисплея эта работа отобразится так:

```
ВВОД+2 (LINE FEED)
005 .—002,002 (LINE FEED)
006 г. ДУБНА (LINE FEED)
007 .—003 (LINE FEED)
010 д. 6, кв. 32 (LINE FEED)
011 (LINE FEED)
012.=15 (LINE FEED)
ПУСК 2 (LINE FEED)
```

В результате в зону номер 15 запишется информация:

```
001 МОСКОВСКАЯ ОБЛ.
002 г. ДУБНА
003 УЛ. ГОРЬКОГО
004 Д. 6, КВ. 32
005 ИВАНОВУ И. И.
```

Если после редактирования запуск на счет не нужен, то последние строки экрана будут такими:

```
...
012.=15(LINE FEED)
.S (LINE FEED)
(LINE FEED)
ПУСК 2
```

Экранный редактор. Если в какой-либо текст, находящийся на ленте либо диске, надо внести много исправлений, то это удобно сделать с помощью *экранного редактора*. Дадим краткое описание его работы.

1. Приказ РЕД переводит сеанс в режим экранного редактора. При этом система выдает на экран следующее:

```
КОНЕЦ
ЭКРАННЫЙ РЕДАКТОР ВЕРСИЯ {номер}
#
```

Символ # — признак готовности к приему приказов редактирования.

2. Приказ # FILE: {ИД}, {описание}

Здесь {ИД} — произвольный идентификатор (не более 6 символов). {описание} — буква Т для магнитной ленты, D — для

диска, затем информация о файле в том виде, как для приказа БУФЕР.

Пример. $\#$ FILE: A, D662/OS, BOSS, FILE, W
 $\#$ FILE: B, T362/МОЯ, W

Если файл находится на ленте ЕС ЭВМ с высокой плотностью записи, то он описывается так:

Н (НБОБ) / ИМЯ МЛ.

Пример. $\#$ FILE : C, H420 / ДУБНА, R

Если файл рабочий (SCRATCH), то — S(N), W, где N — число требуемых зон.

Пример. $\#$ FILE, D, S(100), W

По приказу FILE система находит на внешней памяти описанный файл и приписывает ему имя, заданное в (ИД).

3. Приказ: $\#$ EDIT: (ИД), N

По этому приказу файл с именем ИД, описанный ранее в приказе FILE, переписывается из внешней памяти в N-ю зону внутреннего буфера редактора. Значение N можно выбирать произвольно от 1 до 100.

После того как исполнится приказ EDIT, 14 строк файла появятся на экране — так называемое «окно» редактирования.

Для того чтобы в «окне» появился определенный участок файла, используют следующие приказы.

4. Приказ $\#$ + продвигает «окно» до конца файла (но не более, чем на 4096 строк).

5. Приказ $\#$ — продвигает «окно» файла в начало файла (не более, чем на 4096 строк).

6. Приказ $\#$ +NN продвигает «окно» по файлу вниз на NN строк.

7. Приказ $\#$ —NN продвигает «окно» по файлу вверх на NN строк.

8. Приказ $\#$ (ограничитель) (образ) (ограничитель) продвигает «окно» вниз по файлу до строки, содержащей (образ).

Пример. /.LT./ продвинет «окно» до строки, содержащей фрагмент .LT., но приказ $\#$.LT. в качестве фрагмента воспримет только LT, а точку истолкует как ограничитель. В качестве ограничителя можно использовать любой символ, не содержащийся внутри (образа).

9. Приказ $\#$ D— уничтожает часть файла от текущей строки (на которую указывает курсор) до начала файла.

10. Приказ $\#$ D+ уничтожает часть файла от текущей строки до конца.

11. Приказ $\#$ D—NN уничтожает NN строк от текущей строки вверх.

12. Приказ $\#$ D+NN уничтожает NN строк от текущей строки вниз.

13. Приказ $\# D$ —〈ограничитель〉 〈образ〉 〈ограничитель〉 уничтожает часть файла от текущей строки вверх по 〈образ〉 включительно.

14. Приказ $\# D+$ 〈ограничитель〉 〈образ〉 〈ограничитель〉 уничтожает часть файла от текущей строки вниз по 〈образ〉 включительно.

15. Приказ $\# \langle \text{RETURN} \rangle$. Нажатие клавиши «RETURN» позволяет перейти непосредственно к операциям редактирования «окна» с помощью клавиш дисплея: DC, IC, «↑», «←», «↓», «→», «DL», «IL» (см. 2.6.1).

После того как файл будет полностью отредактирован, следует нажать клавишу «RETURN». После этого редактор будет готов к приему следующих приказов.

16. Приказ $\# \text{SAVE: } \langle \text{ИД1} \rangle$, N переписывает файл из N-й зоны внутреннего буфера редактора во внешнюю память — файл $\langle \text{ИД1} \rangle$, описанный ранее соответствующим приказом $\# \text{FILE}$.

17. Приказ $\# \text{END}$ заканчивает работу с редактором.

2.6.3. Диагностика, выдаваемая при работе с терминала.

1. НЕТ ТАКОГО либо ОШ. КЛЮЧ — набран ключ (PASS), отсутствующий в списке ключей на ЭВМ. Сеанс надо начать заново.

2. КЛЮЧ ЗАНЯТ — набран ключ, который в данный момент уже используется. Следует либо перейти на другой, либо подождать, когда ключ освободится.

3. ОШ. НАБОР — неверно набран приказ. Например, номер зоны превышает максимально допустимый на данном буфере либо номер зоны содержит цифры 8 и 9 (не является восьмеричным). Приказ следует набрать заново.

4. ОШ. БУФЕРА — неверно описаны границы участка буфера. Например, номер начальной зоны больше номера конечной. В ответ на запрос машины БУФЕР: следует заказать буфер заново.

5. ПОВТОРНО — повторно набран буфер, начните сеанс заново.

6. МЛ ЗАНЯТА — данный буфер уже используется кем-либо (возможно вами, если задача запущена на счет). Надо подождать, когда буфер освободится, и заказать его заново.

7. НЕТ МЛ — ваша лента или диск не установлены операторами ЭВМ либо вы ошиблись при наборе. Можно позвонить операторам и попросить установить ленту.

8. ШИФР ЗАНЯТ — набран приказ START N, S, а шифр S занят. Можно послать сообщение операторам с просьбой освободить шифр, если пользователь имеет на это право (приказ ТЕЛ ОСВОБОДИТЕ, ПОЖАЛУЙСТА, ШИФР S).

9. ЗАПР. ЗАПИСЬ — в качестве буфера указана лента или диск, на которых запрещена запись (режим только чтения). Позвоните операторам ЭВМ.

10. ПЛОХ. ОБМЕН — выдается либо в случае плохой работы аппаратуры, либо при попытке читать информацию, записанную с другой плотностью. В последнем случае надо сначала выполнить приказ ПЛО 2, а затем ЛИСТ N.

Следующая диагностика выдается при редактировании текста. Здесь термин «карта» является синонимом термина «строка».

11. НЕТ ОБРАЗА — не найдена строка, первые символы которой (отличные от пробелов), совпадали бы с \langle образом \rangle в приказе ДО \langle образ \rangle . Выдайте приказ ДО с правильным \langle образом \rangle .

12. ПОВТОРНАЯ РЕДАКЦИЯ В КАРТАХ M И N — в строках M и N заказана редакция на одно и то же место текста.

13. ОШИБОЧНЫЙ НОМЕР КАРТЫ — редактируется строка с ошибочным номером (возможно, расположенная ниже строки приказа на ее редактирование).

14. ИСПОРЧЕН ТЕКСТОВЫЙ ФАЙЛ — файл состоит не из символьной информации (возможно, не только из символьной).

15. ОШИБОЧНО СФОРМИРОВАН ФАЙЛ — в файле имеется признак конца редактирования (.), но нет приказов редактирования.

16. ПЛОХОЙ ФОРМАТ УКАЗАТЕЛЯ — указатель диапазона зон в приказе АЦПУ_LN, M состоит не только из восьмеричных цифр (есть 8 или 9). Набрать приказ заново.

17. ПЕРЕСЕЧЕНИЕ ДИАПАЗОНОВ — в заказах на исключение строк (.—N, M) пересекаются номера. Например,

.—003, 010

.—007, 011

18. КАРТА ОТСУТСТВУЕТ — не найден номер карты (строки), указанный в задании на редактирование

19. ВРЕМЯ — слишком долго набираете ключ либо более 2-х минут не работаете на терминале. Если характер вашей работы требует длительного обдумывания (более 2-х минут между какими-нибудь двумя действиями на терминале), то следует воспользоваться приказом ЖДУ. В таком режиме система вас не «выбросит» при сколь угодно долгом бездействии.

20. ОШ. КОНЦА — неправильно указаны границы файла буфера.

21. БУФЕР — после пуска задачи на счет система «забыла» буфер. В ответ на эту диагностику закажите буфер заново.

22. НЕ СПЕШИ — набран очередной приказ отладки в момент, когда еще не выполнен предыдущий приказ. Через некоторое время наберите приказ заново.

23. ЧУЖ. ЛИСТ — делаете попытку во время отладки выдать содержимое ячейки, не принадлежащей вашей программе. Такое действие не допускается операционной системой.

24. Сообщения ФАЙЛ ЗАНЯТ, НЕТ ПАКЕТА, НЕТ ДОСТУПА К ФАЙЛУ соответствуют пп. 6, 7, 9, но выдаются по поводу буфера на диске.

25 НЕТ ФАЙЛА — при заказе буфера набрано ошибочное имя.

В зависимости от версии операционной системы, работающей на конкретной БЭСМ-6, могут быть отклонения от описанного выше набора приказов. Набор отсутствующего приказа повлечет за собой диагностику ОШ. НАБОР.

Глава III

ПОЛЕЗНЫЕ СВЕДЕНИЯ И ПРАКТИЧЕСКИЕ СОВЕТЫ *)

3.1. Диалекты языка фортран

Фортран считается машинно-независимым языком. Это должно означать, что программа, написанная на фортране, может работать на любой ЭВМ, имеющей транслятор с этого языка.

Однако на практике дело обстоит несколько иначе. Правила языка фортран для различных трансляторов могут несколько отличаться друг от друга. Таким образом, язык фортран существует в виде конкретных разновидностей или, как говорят, *версий*.

Аналогичная ситуация имеет место и для других языков программирования. В этом смысле языки программирования ведут себя так же, как и обычные разговорные языки, имеющие диалекты, характерные для каждой местности.

Появление различных версий фортрана обусловлено несколькими причинами. Во-первых, язык фортран непрерывно развивается. При этом происходит обогащение его новыми элементами и одновременное отмирание тех его элементов, которые себя не оправдали. Во-вторых, при разработке транслятора с языка фортран приходится учитывать специфику конкретного типа ЭВМ.

На так называемых малых ЭВМ, имеющих сравнительно небольшой объем памяти и небогатый набор машинных операций, приходится ограничиться использованием лишь части возможностей, предоставляемых языком фортран.

Наконец, во многих случаях разработчики трансляторов видоизменяли те или иные правила языка из со-

*) Эта глава, в отличие от предыдущих, адресована достаточно подготовленному читателю. Здесь используются без объяснения термины и понятия языка фортран, описанные, например в [3, 7, 15].

ображений удобства их реализации в каждом конкретном трансляторе.

Наличие большого числа версий языка затрудняет обмен программами между ЭВМ разных типов. Поэтому были предприняты попытки стандартизации языка фортран с целью достижения большей совместимости программ на ЭВМ разных типов.

В 1966 г. в США был принят стандарт языка фортран, который узаконил две версии фортрана: одну для больших ЭВМ и другую для малых. Версия фортрана для малых ЭВМ получила название *основной фортран* (Basic Fortran) [1] в отличие от версии для больших ЭВМ, называемой *просто фортран*.

Стандарт языка фортран принят и в СССР [20]. В основу его положен американский стандарт 1966 г.

В 1978 г. был принят новый стандарт языка фортран, известный под названием фортран 77 [8]. Краткое описание новых возможностей языка фортран, предусмотренных этим стандартом, будет дано ниже.

Среди версий языка фортран, используемых в нашей стране, наибольшее распространение получили фортран-Дубна (для БЭСМ-6) и фортран IV (для ЭВМ серии ЕС). Эти версии, несмотря на их различие, имеют много общего. Мы старались так построить изложение сведений о фортране, чтобы читатель мог с одинаковым успехом применить их как на БЭСМ-6, так и на ЕС ЭВМ.

3.2. Версии фортрана для ЕС ЭВМ и БЭСМ-6

Рассмотрим основные особенности двух версий языка фортран: фортран-Дубна и фортран IV. Мы будем предполагать, что читатель знаком с каждой из этих версий фортрана. Фортран-Дубна подробно описан в [7, 15]. Литература по фортрану IV еще более обширна: [3, 17] и ряд других. Однако для пользователей ЕС ЭВМ наиболее подходит последняя из указанных книг.

Версия фортран-Дубна используется в трансляторе, эксплуатируемом на БЭСМ-6 с 1969 г. Сейчас можно использовать, кроме основного, еще два оптимизирующих варианта транслятора, а также трансляторы фортран-ГДР и FOREX [2]. Эти варианты почти не отличаются от основного по языку, но позволяют получить более высокое качество транслированной программы.

На ЕС ЭВМ, где используется фортран IV, также имеется несколько вариантов транслятора, отличающихся

друг от друга уровнем оптимизации и некоторыми сервисными возможностями.

В основу версии фортран-Дубна положена версия ЦЕРН-фортран, разработанная в конце 50-х годов. Фортран IV появился позже — в 1962 г. Параллельно с фортраном IV разрабатывался стандарт языка фортран, принятый в США в 1966 г. Фортран IV имеет своей основой этот стандарт, однако во многом его расширяет.

Фортран IV располагает большими возможностями по сравнению с фортраном-Дубна, в котором отсутствуют операторы ввода-вывода прямого доступа, оператор NAMELIST, а также операторы RETURN i, IMPLICIT и операторы описания типа REAL*8 и ему подобные. При этом операторы ввода-вывода прямого доступа и оператор NAMELIST не имеют эквивалентных им операторов на фортране-Дубна. Остальные операторы, имеющиеся в фортране IV, но отсутствующие в фортране-Дубна, допускают эквивалентную замену посредством некоторых других операторов.

В фортране-Дубна есть, однако, операторы ENCODE и DECODE, которых нет в фортране IV.

Особенностью фортрана IV по сравнению с фортраном-Дубна является большая стандартизация в записи программы. Поэтому при переходе с «более узкой» версии фортран-Дубна на «более широкую» версию фортран IV возникает довольно много неприятностей, обусловленных более строгими правилами языка фортран IV по сравнению с фортраном-Дубна (подробнее см. [4]).

3.3. О переводе программ с БЭСМ-6 на ЕС ЭВМ

Мы рассмотрим те особенности языка фортран IV, которые обычно приводят к диагностике ошибок при трансляции на ЕС ЭВМ программ, написанных для БЭСМ-6.

3.3.1. Общая организация программы. На фортране-Дубна допускается запись нескольких операторов в пределах одной перфокарты. Эти операторы разделяются знаком \$ (или \diamond). На фортране IV такая возможность отсутствует, причем знак \$ имеет смысл буквы. Поэтому несколько операторов фортрана-Дубна, разделенных знаком \$, при трансляции на ЕС ЭВМ будут восприняты как один «неправильный» оператор. Например, операторы $A=1$. $\$B=X+5$. будут восприняты как один оператор присваивания, в правой части которого указано

выражение 1. $\$B=X+5$. Такое выражение является ошибочным, о чем будет выдана диагностика.

На фортране-Дубна допускается бо́льшая свобода в использовании меток у операторов. Например, можно указывать метки у декларативных операторов. Однако и та же метка может быть указана у выполняемого оператора и у оператора FORMAT, например

5___A=3. и 5___FORMAT(F10.3)

На фортране IV эти возможности отсутствуют, и их использование приводит к ошибке при трансляции.

Фортран-Дубна предусматривает для головной программы заголовок PROGRAM. В фортране IV такой заголовок запрещен, а головная программа не имеет заголовка.

3.3.2. Константы и их использование. В фортране IV по сравнению с фортраном-Дубна имеются определенные ограничения на константы и их использование в операторах.

В частности, не допускаются восьмеричные константы. Взамен можно использовать шестнадцатеричные константы. Однако последние допускаются только в операторах DATA и в операторах описания типа, в то время как восьмеричные константы могут использоваться в операторах фортран-Дубна наравне с константами других типов.

Диапазон изменения целых констант на фортране IV от 0 до $2^{31}-1$ (по модулю), в то время как на фортране-Дубна — от 0 до $2^{40}-1$. Более узкий диапазон имеют и константы с двойной точностью (от $5 \cdot 10^{-79}$ до $7 \cdot 10^{+75}$). На фортране-Дубна эти константы имеют диапазон от 10^{-1232} до 10^{1232} .

Фортран IV допускает использование текстовых констант только в операторах описания типа и операторах DATA, а также в качестве фактических параметров при обращении к подпрограммам-функциям и подпрограммам. На фортране-Дубна текстовые константы используются и в других операторах наравне с константами иных типов (например, в операторах присваивания).

3.3.3. Запись идентификаторов. Правила фортрана IV не допускают использования русских букв в идентификаторах. Не допускаются также идентификаторы, содержащие более 6 символов.

В фортране-Дубна русские буквы используются в идентификаторах наравне с латинскими. Идентифика-

торы, содержащие более 6 символов, транслятор «укорачивает» до первых 6 символов и выдает предупредительную диагностику, не препятствующую выходу на счет.

3.3.4. Использование переменных с индексами. Фортран IV предусматривает более строгие правила записи переменных с индексами, чем фортран-Дубна. У переменной, являющейся элементом массива, должно быть указано ровно столько индексов, сколько их имеется в описании соответствующего массива. Исключение допускается только в операторе EQUIVALENCE, где возможно использование элементов многомерных массивов как переменных с одним индексом.

На фортране-Дубна можно использовать элементы массивов как переменные с меньшим числом индексов, чем указано в описании соответствующих массивов. А первые элементы массивов можно указывать и без индексов. Например, если указан оператор DIMENSION A(10, 5), то на фортране-Дубна любой из операторов A(1, 1)=0., A(1)=0. и A=0. будет правильным. На фортране IV будет правильным лишь оператор A(1, 1)=0.

3.3.5. Использование оператора DATA. Фортран-Дубна допускает значительно бóльшую свободу в использовании оператора DATA по сравнению с фортраном IV, а также две формы записи этого оператора. Например, наряду с записью DATA X/5./, допустимой правилами фортрана IV, в фортране-Дубна возможна также запись DATA (X=5.).

Перечислим ограничения на использование оператора DATA в фортране IV по сравнению с фортраном-Дубна.

1. Нельзя рассылать данные посредством оператора DATA в элементы неименованного COMMON-блока.

2. В элементы именованных COMMON-блоков рассылка данных посредством оператора DATA возможна лишь внутри BLOCK DATA.

3. Оператор DATA должен располагаться в подпрограмме после декларативных операторов, описывающих переменные и массивы, куда производится рассылка данных. Например, операторы

```
DIMENSION A (5)  
DATA A (2) /10./
```

можно указывать только в таком порядке (но не в обратном).

4. Тип рассылаемых констант должен совпадать с типом переменных, куда они рассылаются. Например, оператор DATA R/1/ не допускается. На фортране-Дубна указанный оператор является допустимым, однако преобразование целой константы 1 к вещественному типу не производится.

3.3.6. Расположение элементов в списке оператора COMMON и длина COMMON-блока. Правила фортрана IV налагают ограничения на порядок расположения элементов в списке оператора COMMON. Необходимо, в частности, выполнение условия: каждому элементу типа COMPLEX или DOUBLE PRECISION должно предшествовать четное число элементов типа REAL, INTEGER и LOGICAL. Указанное ограничение связано с тем, что машины типа ЕС ЭВМ имеют байтовую структуру памяти. Выполнение этого условия обеспечивается, например, когда в списке оператора COMMON сначала будут указаны все элементы типа COMPLEX и DOUBLE PRECISION, а затем все элементы остальных типов.

Длина именованного COMMON-блока, описанного в нескольких подпрограммах, должна быть одинаковой во всех этих подпрограммах. В фортране-Дубна можно, например, описать именованный COMMON-блок в головной подпрограмме на максимальную длину, а в остальных подпрограммах указать меньшую или равную ей.

3.3.7. Расположение элементов в операторе EQUIVALENCE. В операторе EQUIVALENCE так же, как и в операторе COMMON, имеют место ограничения на расположение элементов. Например, операторы

```
DOUBLE PRECISION A (10)
REAL B (15)
EQUIVALENCE (A, B (2))
```

задают недопустимую эквивалентность. Здесь элементу A (1) типа DOUBLE PRECISION предшествует один элемент типа REAL (B (1)), а необходимо, чтобы число таких элементов было четно.

3.3.8. Оператор FORMAT. В операторе FORMAT допускается глубина вложения скобок, равная 2, не считая внешних. В фортране-Дубна допустимая глубина вложения скобок равна 3. Например, оператор.

```
6 FORMAT (2 (1X, 3 (I3, 2 (F10.2/5X, E8.1))))
```

можно использовать в фортране-Дубна, но нельзя в фортране IV.

Первая позиция каждой строки не выводится на печать. Это касается любых символов. В фортране-Дубна указанное ограничение распространяется только на управляющие символы.

В фортране IV отсутствует спецификация формата O, предназначенная для ввода и вывода восьмеричных величин. Аналогичные функции выполняет спецификация Z, которая служит для ввода и вывода шестнадцатеричных величин.

3.3.9. Отсутствие операторов ENCODE и DECODE. В фортране IV отсутствуют операторы ENCODE и DECODE. Эти операторы позволяют, например, организовать формирование текстов при выводе информации на печать. Отсутствие этих операторов в фортране IV создает значительные трудности при переводе программ с фортрана-Дубна на фортран IV.

3.4. Перевод программ с ЕС ЭВМ на БЭСМ-6

Язык фортран-Дубна, как уже говорилось, располагает меньшими возможностями по сравнению с языком фортран IV. Однако на БЭСМ-6 имеются библиотечные подпрограммы, реализующие многое из того, что невозможно описать средствами языка фортран-Дубна.

3.4.1. Операторы языка фортран IV, отсутствующие в фортране-Дубна. В фортране-Дубна отсутствуют следующие операторы языка фортран IV: DEFINE FILE, FIND, NAMELIST, IMPLICIT, REAL*4, REAL*8, INTEGER*4, INTEGER*2, LOGICAL*4, LOGICAL*1, COMPLEX*8, COMPLEX*16.

Операторы DEFINE FILE и FIND используются в совокупности с операторами ввода-вывода прямого доступа. Отсутствие в фортране-Дубна этих операторов в определенной мере восполняется программами для работы с внешними устройствами как устройствами прямого доступа.

Оператор NAMELIST предназначен для организации бесформатного ввода с перфокарт. На БЭСМ-6 аналогичные функции выполняются программой M256, входящей в библиотеку программ ОИЯИ.

Оператор IMPLICIT, определяющий тип величин в зависимости от первой буквы их идентификаторов, на фортране-Дубна может быть заменен операторами описания типа.

Операторы REAL*4 и REAL*8 заменяются соответственно на REAL и DOUBLE PRECISION. Во многих случаях оператор REAL*8 удобнее заменить оператором REAL. Это обусловлено тем, что на БЭСМ-6 величины типа REAL представляются с большей точностью, чем на ЕС ЭВМ (12 знаков вместо 7).

Операторы INTEGER*4 и INTEGER*2 заменяются оператором INTEGER, а операторы LOGICAL*4 и LOGICAL*1 оператором LOGICAL.

Оператору COMPLEX*8 соответствует оператор COMPLEX. Для оператора COMPLEX*16 на фортране-Дубна нет эквивалента, однако возможна замена его оператором COMPLEX. В этом случае при работе на БЭСМ-6 будет получаться меньшая точность вычислений (12 знаков против 17).

3.4.2. Ограничение на значение констант и их типы. В фортране-Дубна отсутствуют шестнадцатеричные константы, которые могут быть заменены восьмеричными.

Диапазон изменения констант типа REAL в фортране-Дубна более узкий, чем в фортране IV (от 10^{-19} до 10^{19} по модулю). Вещественные константы, выходящие за указанный диапазон, могут быть заменены равными им константами с двойной точностью. Например, константу 1.6E25 можно заменить на 1.6D25. Аналогичное ограничение имеет место для действительных и мнимых частей комплексных констант. Однако в этом случае замена константами с двойной точностью не проходит.

3.4.3. Ограничение на число размерностей массива. В фортране-Дубна допускаются массивы с числом размерностей не более 3. В фортране IV можно указывать до 7 размерностей.

3.4.4. Ограничение на использование в одном арифметическом выражении величин разных типов. В фортране IV разрешается в одном арифметическом выражении указывать величины типа DOUBLE PRECISION (или REAL*8) и COMPLEX (соответственно COMPLEX*8 и COMPLEX*16). В фортране-Дубна такое смешение типов запрещено. Выход может состоять в том, что все величины типа DOUBLE PRECISION (или REAL*8) заменяются величинами типа REAL.

3.4.5. Ограничения на спецификации формата. В фортране-Дубна отсутствуют спецификации формата G, Z и T. Необходимо заменить все спецификации Gw, d на Ew, d, а спецификации Zw— на Ow. Для спецификации Tw в фортране-Дубна нет подходящей замены.

3.4.6. Ограничения на фактические параметры при вызове подпрограмм. В фортране IV при вызове подпрограмм (SUBROUTINE) в качестве фактического параметра допускается номер метки оператора, на который может быть произведен возврат. Например, допускается обращение CALL SUB (X, &15, 1.E6). В этом случае подпрограмма SUB должна иметь заголовок вида SUBROUTINE SUB (R, *, C). Возврат на оператор с меткой 15 в этом случае производится оператором RETURN 1 в подпрограмме SUB.

В фортране-Дубна такой способ обращения запрещен. При переходе с фортрана IV необходимо изъять из списка формальных параметров все *, а из списка фактических параметров те, которые соответствуют изъятым «*». Все операторы RETURN i придется заменить на RETURN. Однако при этом необходимо обеспечить правильную работу программы.

Можно предложить следующий способ. Пусть имеется обращение

```
CALL LCT (A, &20, &7, —2.3, &50)
```

и соответственно подпрограмма

```
SUBROUTINE LCT (X, *,*, T, *)
```

внутри которой могут быть операторы RETURN, RETURN 1, RETURN 2 и RETURN 3. Первый из этих операторов определяет возврат на оператор, следующий за оператором CALL, а последующие — на операторы с метками 20, 7 и 50 соответственно.

При переходе на фортран-Дубна из списка формальных параметров подпрограммы LCT изымаются все «*» и добавляется один параметр целого типа, играющий роль переключателя. Назовем этот параметр IFLAG. Итак, новый заголовок у LCT будет такой:

```
SUBROUTINE LCT (X, T, IFLAG).
```

Дальнейшее преобразование LCT будет состоять в следующем. Вместо оператора RETURN ставятся два оператора IFLAG=0 и RETURN, а вместо оператора RETURN i — операторы IFLAG=i и RETURN. Таким образом, на выходе из подпрограммы LCT значение параметра IFLAG будет равно 0, если выход был через оператор RETURN, и будет равно i, если выход был через оператор RETURN i. Это позволяет произвести в вызывающей подпрограмме переход к выполнению

нужного оператора после выполнения оператора CALL. Для этого достаточно написать, например, так:

```
CALL LCT (A, -2.3, IFLAG)  
IF (IFLAG.NE.0) GO TO (20, 7, 50), IFLAG
```

В этом случае при IFLAG=0 произойдет переход к следующему оператору, а при IFLAG=1, 2, 3 — на один из операторов с меткой 20, 7, 50 соответственно.

Отметим еще одну особенность записи формальных параметров, допускаемую правилами языка фортран IV.

Если формальный параметр (в SUBROUTINE или FUNCTION) является простой переменной, то он может быть заключен в косые черточки (/). Например, FUNCTION ABC(/X/, /Y/, Z). В этом случае в ячейку, отведенную для формального параметра, заносится адрес соответствующего фактического параметра (вызов фактического параметра по адресу).

Если формальный параметр, являющийся простой переменной, в косые черточки не заключен, то в ячейку, отведенную для формального параметра, заносится само значение соответствующего параметра (вызов фактического параметра по значению).

В фортране-Дубна на БЭСМ-6 вызов фактического параметра всегда производится только по адресу (но косые черточки не ставятся).

3.4.7. Ограничения на оператор ввода-вывода. В фортране-Дубна допускаются лишь операторы ввода-вывода последовательного доступа, а также лишь форматный оператор ввода с перфокарт. В операторах ввода-вывода последовательного доступа не допускаются указатели вида ERROR= и END=.

3.5. Случаи разной трактовки фортранной программы трансляторами на БЭСМ-6 и ЕС ЭВМ

Различия между версиями фортран-Дубна и фортран IV обусловлены также и тем, что трансляторы на БЭСМ-6 и ЕС ЭВМ по-разному воспринимают некоторые элементы языка фортран. Это означает, что одна и та же программа может по-разному трактоваться на БЭСМ-6 и на ЕС ЭВМ. При этом никакой диагностики, естественно, не будет выдано. Именно эти различия доставляют наибольшие неприятности при переводе программ с БЭСМ-6 на ЕС ЭВМ и обратно. К счастью, этих различий не очень много.

3.5.1. Выражение $A**B**C$ на БЭСМ-6 понимается как $(A**B)**C$, а на ЕС ЭВМ — как $A**(B**C)$. Поэтому необходимо путем расстановки скобок указывать порядок выполнения операций.

3.5.2. Вычисляемый $GO\ TO$, т. е. оператор вида $GO\ TO\ (m_1, m_2, \dots, m_n), K$

может по-разному сработать на БЭСМ-6 и ЕС ЭВМ.

Различие обнаруживается лишь при значениях переменной K , выходящих за пределы диапазона от 1 до n . В этом случае на ЕС ЭВМ происходит переход на оператор, следующий за данным $GO\ TO$. На БЭСМ-6 в этом случае произойдет переход на команду, не предусмотренную в программе. Такой непредусмотренный переход иногда называют потерей управления. В случае потери управления правильная работа программы не гарантируется.

3.5.3. Оператор $DO\ m\ i=i_1, i_2, i_3$ или $DO\ m\ i=i_1, i_2$ при $i_2 < i_1$ на ЕС ЭВМ вызывает однократное выполнение соответствующего цикла, а на БЭСМ-6 этот цикл не будет выполнен ни разу.

Указанное различие обусловлено разным алгоритмом реализации оператора DO . На БЭСМ-6 в транслированной программе сначала производится проверка выполнения условия $i_1 \leq i_2$ и выход из цикла при нарушении этого неравенства. На ЕС ЭВМ сначала производится выполнение цикла и только затем проверка условия $i_1 \leq i_2$. При этом к i_1 уже прибавлено значение шага (i_3 или 1).

3.5.4. Оператор $ENTRY$ по-разному оформляется на фортране IV и фортране-Дубна. Если, например, имеется вход $R215$ в подпрограмму с заголовком $SUBROUTINE\ SGL\ (A, B, LIST)$, то на фортране-Дубна он обозначается оператором $ENTRY\ R215$. При этом считается, что $R215$ имеет те же параметры, что и главный вход SGL . Поэтому для обращения к входу $R215$ надо написать оператор $CALL$ с тремя фактическими параметрами.

В фортране IV принят иной способ записи оператора $ENTRY$. У каждого оператора $ENTRY$ указывается свой список формальных параметров, который может не совпадать со списком параметров у главного входа. Поэтому запись $ENTRY\ R215$ на фортране IV будет означать, что у данного входа параметры отсутствуют и, следовательно, обращение к нему должно быть $CALL\ R215$.

Указанное различие приводит к тому, что подпрограмма, имеющая параметры и дополнительные входы, будет неправильно работать при переходе на другую ЭВМ. Выход может состоять в том, что все операторы ENTRY изымаются и соответствующие им части программы выделяются в независимые подпрограммы.

3.5.5. При вводе по спецификации I пустые колонки (позиции перфокарты) в поле ввода по-разному воспринимаются на ЕС ЭВМ и БЭСМ-6. На ЕС ЭВМ эти пробелы понимаются как нули, а на БЭСМ-6 игнорируются. Например, если вводится число по спецификации I4 и на карте пробито 25□□, то на ЕС ЭВМ это число введется как 2500, а на БЭСМ-6 — как 25. Для исключения указанного эффекта необходимо вводимые числа располагать в правых колонках поля ввода.

3.5.6. Представление данных в БЭСМ-6 и ЕС ЭВМ существенно отличается друг от друга. Это может привести к несовместимости программ либо отразиться на результатах счета.

Различие в диапазоне представимых чисел может привести к тому, что программа, нормально работающая на одной машине, будет давать переполнение или машинные нули на другой.

Различие в точности машинного представления чисел может привести к существенно различным результатам счета. БЭСМ-6 имеет большую точность машинного представления чисел (12 десятичных знаков с обычной точностью и 24 — с двойной), чем ЕС ЭВМ (7 десятичных знаков с обычной точностью и 17 знаков с двойной). Поэтому при переходе с БЭСМ-6 на ЕС ЭВМ довольно часто возникает необходимость введения двойной точности.

Различие в представлении текстовых данных состоит в том, что в одно машинное слово (ячейку) БЭСМ-6 упаковывается 6 символов, а на ЕС ЭВМ — 1, 2, 4 или 8 символов в зависимости от типа данных. Это различие может сказаться при вводе или выводе текстовых величин по спецификации формата Aw. Для БЭСМ-6 типично w=6, а для ЕС ЭВМ — 1, 2, 4 или 8.

3.6. Краткие сведения о языке фортран 77

В 1978 г. в США был принят новый стандарт языка фортран [8]. Его разработка была закончена в 1977 г., поэтому он получил название фортран 77.

При разработке нового стандарта был учтен опыт использования языка фортран на ЭВМ различных типов за последние 10 лет. Новый стандарт существенно расширил возможности языка по сравнению с прежним стандартом 1966 г. При этом в нем сохранены практически все возможности старого стандарта.

В фортране 77 вводятся данные нового типа CHARACTER, или текстовые. Хотя текстовые данные использовались в языке фортран и прежде, они не имели удобного способа представления. При работе с текстовыми данными в фортране использовались машинные слова. Однако в зависимости от типа ЭВМ в машинном слове помещается разное количество символов. Это приводило к несовместимости программ для ЭВМ разных типов.

Константы типа CHARACTER представляют собой строки символов, ограниченные апострофами, например 'DUBNA', 'DON'T'. В последней константе двойной апостроф определяет одинарный апостроф, т. е. эта константа состоит из символов DON'T.

Каждая текстовая переменная (или элемент массива) имеет фиксированную длину, описанную в соответствующем декларативном операторе. Если длина не указана, то она полагается равной 1. Например, операторы

```
CHARACTER*6 VOLGA  
CHARACTER UNIT(20)
```

описывают текстовую переменную VOLGA длины 6 и текстовый массив UNIT, состоящий из 20 элементов длины 1.

Переменные и элементы массивов можно использовать в операторе присваивания. Например,

```
VOLGA='VOLGA'  
UNIT(5)='T16'
```

При выполнении операции присваивания слишком длинные строки усекаются справа, а слишком короткие дополняются пробелами справа. В указанных примерах константа 'VOLGA' дополнится одним пробелом справа для получения шести символов, а константа 'T16' будет усечена до первого символа, так как переменная UNIT(5) имеет длину 1.

Для текстовых данных вводится операция сцепления, обозначаемая двумя знаками //. Например,

```
VOLGA='RT'//UNIT(8)
```


Здесь переменная VOLGA длины 6 будет включать в себя символы RT, затем переменную UNIT (8), состоящую из одного символа, и еще 3 пробела.

Можно выделить часть строки, указав номера начального и конечного символов. Эти номера разделяются двоеточием и заключаются в круглые скобки. Например,

UNIT (10)=VOLGA (5 : 5)

VOLGA (3 : 6)='ABCD'

В первом примере переменная UNIT (10) получает значение 5-го символа переменной VOLGA. В последнем примере у переменной VOLGA первый и второй символы останутся прежними, а остальные 4 символа будут заменены на ABCD.

Строки символов можно сравнивать между собой посредством операций отношения. При этом предполагается, что символы определенным образом упорядочены (например, в соответствии с кодом ISO) и что символ «пробел» предшествует любому иному символу.

Например, выражение 'FOUR'.LT.'FOURTEEN' истинно, а отношение 'L3'.GT.'2L' ложно.

Вводятся ограничения на использование текстовых данных в операторах COMMON. Если COMMON-блок содержит текстовые величины, то он не может содержать величин иных типов.

Для преобразования отдельного символа в целую величину и обратно вводятся стандартные функции ICHAR и CHAR. Функция INDEX (A, B) определяет, является ли строка A частью строки B и выдает в качестве результата либо начальную позицию совпавших строк, либо нуль. Например,

INDEX ('DUBNA', 'NA')=4

INDEX ('DUBNA', 'VOLGA')=0

В фортране 77 расширены возможности работы с индексами. Допускаются, в частности, нулевое и отрицательное значения индекса. Максимальное число индексов равно 7. При описании массивов можно указывать значения нижней и верхней границы индекса, разделяя их двоеточием. Если нижняя граница не указана, то она полагается равной 1.

П р и м е р ы.

DIMENSION A (5, —1 : 12, 8, 3 : 5)

CHARACTER CH (0 : 5, 8)*4

В описаниях массивов в FUNCTION и SUBROUTINE размерности могут быть указаны в виде арифметических выражений, содержащих переменные целого типа. Эти переменные должны быть указаны либо в COMMON-блоке, либо как формальные параметры. Индекс элемента массива может иметь вид арифметического выражения целого типа.

Важным расширением языка фортран является возможность использования в операторе DO выражений любого типа в качестве начального значения, конечного значения и шага переменной цикла. Кроме того, после метки можно ставить запятую. Например,

```
DO 5 K=6, -1, -2  
DO 18, T=R+5, X+SQRT(1.5+Y)
```

В первом примере цикл будет выполняться при $K = 6, 4, 2, 0$, т. е. до тех пор, пока значение K будет больше или равно -1 . Во втором примере переменная цикла T имеет тип REAL и в качестве ее начального и конечного значений заданы выражения вещественного типа.

Число повторений цикла

```
DO m V=V1, V2, V3
```

равно

```
MAX (INT ((n2-n1+n3)/n3), 0),
```

где n_1, n_2, n_3 — значения V_1, V_2, V_3 , приведенные к типу переменной V .

По окончании выполнения цикла переменная цикла принимает то значение, которое она имела бы при следующем повторении цикла. Например, после выполнения цикла

```
DO 25 N=1, 10, 2  
25 M=N
```

значение M будет равно последнему значению N , при котором происходило выполнение цикла, т. е. 9. Значение же N будет равно 11.

Схема выполнения цикла DO соответствует тому, как это реализовано в фортране-Дубна; сначала проверяется возможность выполнения очередного повторения цикла и только затем производится его выполнение, если это возможно.

Важным новшеством, вводимым в язык фортран, является конструкция IF...THEN...ELSE. Эта конст-

рукция отвечает идеям структурного программирования, получившим распространение в последние годы [18]. Рассмотрим работу IF...THEN...ELSE на следующем примере:

```
LOGICAL FUNCTION PRIME(N)
IMPLICIT INTEGER (A—Z)
IF (N.LE.1) THEN
PRIME=.FALSE.
ELSE IF (N.EQ.2) THEN
PRIME=.TRUE.
ELSE IF (MOD(N, 2).EQ.0) THEN
PRIME=.FALSE.
ELSE
DO 18 DIVISR=3, INT(SQRT(REAL(N))), 2
IF (MOD(N, DIVISR).EQ.0) THEN
PRIME=.FALSE.
RETURN
END IF
18 CONTINUE
PRIME=.TRUE.
END IF
END
```

Логическая функция PRIME выдает в качестве результата .TRUE., если ее аргумент N является простым числом. В противном случае значение PRIME равно .FALSE.

Как видно из примера, оператор IF...THEN определяет начало блока. Внутри блока могут быть операторы ELSE IF и ELSE, а также другие операторы IF...THEN. Окончанием блока, начинающегося очередным оператором IF...THEN, является оператор END IF. В указанном примере имеются два вложенных блока. Внешний блок начинается оператором

```
IF (N.LE.1) THEN
внутренний — оператором
IF (MOD (N, DIVISR).EQ.0) THEN
```

Окончанием внутреннего блока служит первый из операторов END IF, расположенный в области действия оператора DO. Внешний блок оканчивается оператором END IF, расположенным перед оператором END.

Данный пример показывает удобство конструкции IF...THEN...ELSE при программировании сложных логических выражений.

В фортране 77 введены унифицированные стандартные функции, которые могут принимать значения разных типов в зависимости от типа аргумента. Например, функция `SQRT (X)` принимает значение вещественного или комплексного типа либо типа с двойной точностью соответственно типу аргумента. Таким образом, `SQRT (2.)` определяет квадратный корень для вещественного значения аргумента, а `SQRT (2.D0)` — для аргумента с двойной точностью (аналогично `DSQRT(2.D0)`). Указанное новшество значительно упрощает перевод программ с ЭВМ одного типа на другой.

НАШИ ПЕРВЫЕ ПРОГРАММЫ

Мы приведем несколько программ, с которых можно начинать работу на ЭВМ.

4.1. ЭВМ рисует картинки

Для выдачи рисунка с ЭВМ сначала его выполняют какими-либо символами на бумаге в клетку, затем рассчитывают, в какие позиции должны попасть эти символы. С помощью редакционных спецификаций nX и mH оператора FORMAT осуществляют задуманный рисунок и оператором PRINT выдают на печать.

Программа 1. Рисунок «Ослик»

```

PROGRAM PAL
PRINT 10
10 FORMAT (10X, '*_*/10X, '***'/9X, '*****'/
C8X, '*****'/8X, '**_*****'/13X, 15('*')/
C14X, 13('*'), 1X, '**'/14X, 13('*'), 1X, '**'/
C13X, 13('*'), 2X, '**'/12X,
C'*_**_**_**_**_**_**_**_*/11X, '**', 5X, '**',
C5X, '*_**_*/
C12X, '**', 5X, '**', 4X, '*_**_**_*/13X,
C'*_**_**_**_**_**_**_**_*/14X,
C'*_**_**_**_**_**_**_**_')
END

```

В следующей программе использована подпрограмма BIGTIT, печатающая крупными символами произвольный текст (не более 18 символов). Обращение

CALL BIGTIT (H<ТЕКСТ>, M, 6H*****)

Для печати текста его разбивают на группы по 6 символов в каждой. После каждых 6Н пишется соответствующая группа. М — рабочий массив, его следует описать как DIMENSION M(144). После последних 6Н записываются символы, из которых будут составлены крупные буквы. Эта подпрограмма описана в [13].

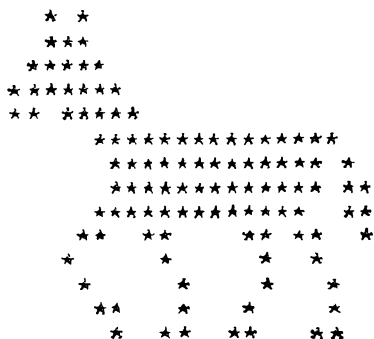


Рис. 1

Программа 2. Рисунок «MOSCOW OLYMPIC GAMES XXII».

```

PROGRAM GAMES
DIMENSION M(144)
DO 10 I=1, 5
PRINT 20
20 FORMAT (1H1)
CALL BIGTIT (6HMOSCOW, 6H□□OLYM,
C6HPIC□□□, M, 6HMMMMMM)
CALL BIGTIT (6HGAMES□, 6H□XXII□,
C6H□□□□□□□, M, 6HMMMMMM)
PRINT 30
30 FORMAT (36X, 10(1HM)/8X, 5(1HM), 21X,
C14(1HM)/
C5X, 12(1HM), 15X, 18(1HM)/3X, 15(1HM),
C13X, 19(1HM)/2X, 18(1HM), 10X, 19(1HM)/
C2X, 18(1HM), 5X, 24(1HM)/1X, 18(1HM), 2X,
C27(1HM)/2X, 45(1HM)/2X, 44(1HM)/3X,
C45(1HM)/5X, 44(1HM)/8X, 24(1HM), 3X,
C15(1HM)/7X, 12(1HM), 3X, 9(1HM), 1X, 2(1HM),
C2X, 15(1HM)/7X, 11(1HM), 1X, 2(1HM), 1X,
C4(1HM), 3X, 23(1HM), 3X, 1HM/
C6X, 15(1HM), 5X, 2(1HM), 7X, 17(1HM),
C3X, 4(1HM), 1X)/6X, 14(1HM), 12X,
C1HM, 4X, 15(1HM), 2X, 8(1HM)/7X, 12(1HM),
C4X, 9HM —————, 5X, 15(1HM), 1X,
C9(1HM)/7X, 13(1HM), 17X, 14(1HM), 1X, 9(1HM)/
C8X, 13(1HM), 15X, 14(1HM), 1X, 9(1HM)/, 8X,
C15(1HM), 10X, 16(1HM), 1X, 10(1HM)/

```

C9X, 39(1HM), 1X, 10(1HM)/11X, 34(1HM), 2X,
 C11(1HM)/13X, 30(1HM), 1X, 13(1HM)/15X,
 C24(1HM), 2X, 14(1HM)/13X, 8(1HM), 1X, 22(1HM),
 C1X, 9(1HM)/12X, 10(1HM), 1X, 22(1HM), 1X,
 C6(1HM)/



Рис. 2

Ç11X, 12(1HM), 1X, 22(1HM), 1X, 4(1HM)/
 Ç11X, 13(1HM), 1X, 21(1HM), 2X, 1HM/
 Ç11X, 13(1HM), 1X, 21(1HM)/12X, 13(1HM),
 Ç1X, 21(1HM)/12X, 13(1HM), 1X, 21(1HM)/
 Ç12X, 1HM, 1X, 10(1HM), 1X, 22(1HM)/

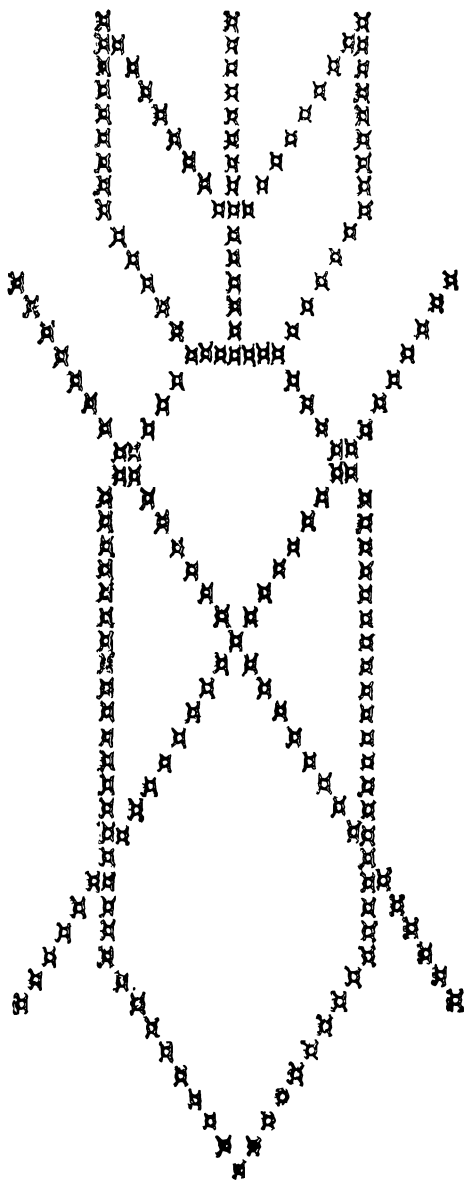


Рис. 3


```

C12X, 12(1HM), 1X, 22(1HM)/12X, 1HM,
C2X, 9(1HM), 6X, 5(1HM), 1X, 4(1HM), 1X,
C3(1HM), 1X, 2(1HM)/11X, 1HM, 3X, 8(1HM),
C1X, 4(1HM), 5X, 1HM, 4X, 1HM, 3X, 1HM,
C3X, 1HM/10X, 3(1HM), 1X, 8(1HM), 6X,
C6(1HM), 3X, 2HMM, 3X, 1HM/9X, 4(1HM),
C1X, 7(1HM), 1X, 5(1HM), 12X, 2HMM, 4X,
C1HM/8X, 4(1HM), 1X, 4(1HM, 1X), 15(1HM),
C3X, 1HM, 4X, 1HM/7X, 10(1HM), 1X, 1HM,
C1X, 16(1HM), 1X, 3HMMM, 2X, 1HM/6X,
C14(1HM),
C1X, 21(1HM)/5X, 17(1HM), 1X, 17(1HM)/6X,
C18(1HM), 1X, 12(1HM), 1X, 4(1HM)/7X, 17(1HM),
C1X, 10(1HM), 1X, 6(1HM)/9X,
C16(1HM), 1X, 5(1HM),
C1X, 9(1HM)/11X, 14(1HM), 1X, 14(1HM)/12X,
C13(1HM), 1X, 13(1HM)/11X, 15(1HM), 1X,
C11(1HM)/
C11X, 17(1HM), 1X, 12(1HM)/14X,
C15(1HM), 5X, 5(1HM)/19X, 8(1HM))
10 CONTINUE
END

```

Аналогично получен и рисунок «Нет — нейтронной бомбе» (рис. 3).

4.2. Программы печатают таблицы функций

Программа 3. Таблица косинусов.

```

PROGRAM COSIN
DIMENSION X(64), Y(64)
X(1)=0
DO 4 N=1,63
Y(N)=COS(X(N))
X(N+1)=X(N)+0.1
4 CONTINUE
PRINT 8, (X(L), Y(L), L=1, 63)
8 FORMAT (2(4HCOS(, F5.3, 2H)=,F6.3, 3X))
END

```

Результаты работы программы:

COS(0.000) =	1.000	COS(0.100) =	0.995
COS(0.200) =	0.980	COS(0.300) =	0.955
COS(0.400) =	0.921	COS(0.500) =	0.878
COS(0.600) =	0.825	COS(0.700) =	0.765

$\text{COS}(0.800) = 0.697$	$\text{COS}(0.900) = 0.622$
$\text{COS}(1.000) = 0.540$	$\text{COS}(1.100) = 0.454$
$\text{COS}(1.200) = 0.362$	$\text{COS}(1.300) = 0.267$
$\text{COS}(1.400) = 0.170$	$\text{COS}(1.500) = 0.071$
$\text{COS}(1.600) = -0.029$	$\text{COS}(1.700) = -0.129$
$\text{COS}(1.800) = -0.227$	$\text{COS}(1.900) = -0.324$
$\text{COS}(2.000) = -0.416$	$\text{COS}(2.100) = -0.505$
$\text{COS}(2.200) = -0.589$	$\text{COS}(2.300) = -0.666$
$\text{COS}(2.400) = -0.738$	$\text{COS}(2.500) = -0.801$
$\text{COS}(2.600) = -0.857$	$\text{COS}(2.700) = -0.904$
$\text{COS}(2.800) = -0.942$	$\text{COS}(2.900) = -0.971$
$\text{COS}(3.000) = -0.990$	$\text{COS}(3.100) = -0.999$
$\text{COS}(3.200) = -0.998$	$\text{COS}(3.300) = -0.987$

. . .

Программа 4. Таблица синусов, косинусов, тангенсов.

```

PROGRAM TAB
DIMENSION X(64), Y(63), Z(63), A(63)
X(1) = 0.
DO 4 N = 1, 63
  Y(N) = SIN(X(N))
  Z(N) = COS(X(N))
  A(N) = TAN(X(N))
  X(N + 1) = X(N) + 0.1
4 CONTINUE
PRINT 5, (X(L), Y(L), Z(L), A(L), L = 1, 63)
5 FORMAT (10X, 1HX, 6X, 5HSIN X, 8X,
C5HCOS X, 8X,
C5HTAN X/63(10X, F3.1, 4X, F6.3, 7X, F6.3,
C6X, F7.3/))
END

```

Результаты работы программы:

X	SIN X	COS X	TAN X
0.0	0.000	1.000	0.000
0.1	0.100	0.995	0.100
0.2	0.199	0.980	0.203
0.3	0.296	0.955	0.309
0.4	0.389	0.921	0.423
0.5	0.479	0.878	0.546
0.6	0.565	0.825	0.664
0.7	0.644	0.765	0.842
0.8	0.717	0.697	1.030
0.9	0.783	0.622	1.260

1.0	0.841	0.540	1.557
1.1	0.891	0.454	1.965
1.2	0.932	0.362	2.572
1.3	0.964	0.267	3.602
1.4	0.985	0.170	5.798
1.5	0.997	0.071	14.101
1.6	1.000	—0.029	—34.233
1.7	0.992	—0.129	—7.697
1.8	0.974	—0.227	—4.286
1.9	0.946	—0.323	—2.927
2.0	0.909	—0.416	—2.185

. . .

4.3. Работа с массивами

Программа 5. Выдача из массива M чисел, делящихся на 3.

```

PROGRAM MA
DIMENSION M(10)
READ 9, M
9 FORMAT (10I2)
DO 8 I=1, 10
  IF (M(I)/3*3—M(I)) 8, 2, 8
2 PRINT 7, M(I)
7 FORMAT (5X, I2)
8 CONTINUE
END

```

Результат работы программы:

```

3
6
9

```

Эта программа вводит с перфокарт десять целых чисел и печатает те из них, которые делятся нацело на 3. Проверка делимости основана на следующем: $M(I)/3*3$ есть $M(I)$ только для тех $M(I)$, которые делятся на 3; для остальных результат $M(I)/3$ — целая часть частного $M(I) : 3$, и поэтому $M(I)/3*3$ будет меньше $M(I)$.

Программа 6. Упорядочение массива из 100 чисел по возрастанию. Эта программа использует алгоритм, иногда называемый методом «пузырька»: упорядочивается каждая пара соседних чисел так, чтобы справа стояло большее. В результате максимальное

число оказывается в последней ячейке массива, при следующем просмотре пар максимальное из оставшихся чисел оказывается на предпоследнем месте и т. д. Процесс заканчивается, когда при просмотре массива никакие два числа не надо менять местами. Подробно этот алгоритм описан в [6].

```
PROGRAM S
DIMENSION C(100)
READ 11, C
11 FORMAT (10F6.3)
CALL SORT (100, C)
PRINT 21, C
21 FORMAT (10(2X, F6.3))
END
SUBROUTINE SORT (N, A)
DIMENSION A(N)
M=N-1
10 PR=0
DO 20 I=1, M
IF (A(I)-A(I+1)) 20, 20, 40
20 CONTINUE
M=M-1
IF (PR) 10, 30, 10
30 RETURN
40 C=A(I)
A(I)=A(I+1)
A(I+1)=C
PR=1
GO TO 20
END
```

Для запуска этой программы надо пробить 100 произвольных чисел по спецификации F6.3 (десять чисел на одной карте) и вставить эти карты после *EXECUTE на БЭСМ-6 и после //EXEC на ЕС ЭВМ.

4.4. Решение квадратного уравнения

Программа 7. Решение квадратного уравнения с полным исследованием.

```
PROGRAM SQRTEQ
READ 1, A, B, C
1 FORMAT (3F6.3)
IF (A) 3, 2, 3
```

```

3 IF (B**2—4*A*C) 4, 5, 5
5 X1=(—B+SQRT(B**2—4*A*C))/(2*A)
  X2=(—B—SQRT(B**2—4*A*C))/(2*A)
  PRINT 6, A, B, C, X1, X2
6 FORMAT (10X, 2HA=, F6.3, 2X, 2HB=, F6.3,
  C2X, 2HC=, F6.3/10X, 3HX1=, F7.3, 2X,
  C3HX2=, F7.3)
  GO TO 100
2 IF (B) 8, 7, 8
8 X=—C/B
  PRINT 9, A, B, C, X
9 FORMAT (4F6.3)
  GO TO 100
7 IF (C)4, 11, 4
4 PRINT 10, A, B, C
10 FORMAT (10X, 2HA=, F6.3, 2X, 2HB=, F6.3,
  C2X, 2HC=, F6.3/
  C10X, 10HКОРНЕЙ_НЕТ)
  GO TO 100
11 PRINT 12, A, B, C
12 FORMAT (10X, 2HA=, F6.3, 2X, 2HB=, F6.3,
  C2X, 2HC=, F6.3/
  C10X, 7HX—ЛЮБОЕ)
100 CONTINUE
  END

```

Результат работы программы:

```

A=23.333_ _ _ B=15.028_ _ _ C=—2.051
X1=0.116_ _ _ X2=—0.760

```

Для этой программы коэффициенты A, B, C пробиваются по спецификации F6.3 на одной перфокарте.

4.5. Простейший лабиринт

Программа 8 «Лабиринт». Эта программа решает такую задачу: двумерный массив LAB(10, 10), состоящий из нулей и единиц, представляет собой лабиринт без тупиков и колец. Стена изображается единицами, а коридор — нулями. Машина должна «пройти» от входа до выхода из лабиринта по коридору, заменяя нули на восьмерки.

```

PROGRAM MOD
DIMENSION LAB (10, 10)
READ 1, LAB

```

```

1 FORMAT (10I1)
  I=1
  J=2
2 LAB (I, J)=8
  IF (LAB (I, J+1)) 3, 4, 3
4 J=J+1
  IF (J—11) 2, 6, 6
3 IF (LAB (I+1, J)) 5, 7, 5
5 IF (LAB (I, J—1)) 11, 8, 11
7 I=I+1
  IF (J—11) 2, 6, 6
8 J=J—1
  IF (J) 6, 6, 2
11 IF (LAB (I—1, J)) 6, 12, 6
12 I=I—1
  IF (I) 6, 6, 2
6 PRINT 13, ((LAB (I, J), J=1, 10), I=1, 10)
13 FORMAT (2X, 10I1)
  END

```

Результат работы этой программы:

```

1811111111
1811111111
1811888811
1811811811
1811881881
1811181181
1818881181
1818111181
1888111188
1111111111

```

4.6. Решение уравнения $f(x)=0$ методом половинного деления

Этот метод позволяет найти приближенно нуль функции, заданной на отрезке $[A, B]$ таким образом, что $f(A) \times f(B) \leq 0$. На этом отрезке $f(x)$ непрерывна и имеет единственный нуль.

Программа 9. Вычисление корня уравнения $x^3 - x^2 - 2 = 0$ на отрезке $[1, 2]$ с точностью $\epsilon = 10^{-2}$.

```

PROGRAM MPD
A = 1.
B = 2.

```

EPS = 1.E-2	Начальные данные
IF (F(A)*F(B)) 1, 2, 3	Исследование знака
1 C = (A + B)/2	Деление отрезка пополам
IF (F(A)*F(C)) 4, 7, 6	
4 B = C	Отбрасывание правой точки
5 IF (ABS(A - B) - EPS)	Выход по точности
C7, 1, 1	
7 T = F(C)	
PRINT 9, C, T	
9 FORMAT(20X, 2HC=, F4.2, 5X, 5HF(C)=, F5.3)	
GO TO 100	
2 IF F(A)) 10, 11, 10	
11 C = A	Если $f(A) * f(B) = 0$
GO TO 7	
10 C = B	
GO TO 7	
3 PRINT 12	
12 FORMAT (20X, 10HКОРНЕЙ НЕТ)	
GO TO 100	
6 A = C	Отбрасывание левой точки
GO TO 5	
100 CONTINUE	
END	
FUNCTION F(X)	
F = X**3 - X**2 - 2.	
RETURN	
END	

Результат счета:

C=1.70 F(C)=0.000

4.7. Некоторые рекомендации начинающему программисту

Приведенные в этой главе программы не претендуют на оптимальность. Они написаны для БЭСМ-6 школьниками 10-х классов на втором году обучения программированию на уроках труда.

Мы поместили их из тех соображений, что начинающему читателю более понятны именно те программы,

которые составлены людьми, делающими первые шаги в программировании.

Выше мы уже писали, что каждая задача может быть решена с помощью разных программ, отличающихся и по времени выполнения и по числу операторов. Это может зависеть как от квалификации программиста, так и от назначения программы и параметров ЭВМ.

Естественно, что каждый программист стремится к созданию оптимальной программы. Но понятие оптимальности не определяется однозначно. Вообще говоря, программа, являющаяся и более короткой и более быстрой, является более оптимальной. Но часто эти два параметра находятся в обратной зависимости: более короткая программа работает дольше более длинной. Какой же из этих двух программ отдать предпочтение? Это зависит от конкретных условий.

Для ЭВМ с малой памятью оптимальными считаются короткие программы. Если же на машине решается задача управления полетом ракеты, то первостепенным является требование максимальной скорости работы. Такие программы имеют сложную структуру и требуют много времени на разработку и отладку.

Многие программы управления реальными процессами не содержат циклов: команды пишутся столько раз, сколько они выполняются. Этим самым снижаются «накладные расходы», связанные с организацией циклов. В таких условиях подобная громоздкая программа и является оптимальной.

Если перед программистом стоит задача составить в короткий срок программу, занимающую менее часа счетного времени и предназначенную для разового выполнения, то оптимальной будет программа, написанная быстро и требующая минимальное число выходов на машину при отладке.

Если вы составили и отладили такую программу, а потом придумали более короткий способ решения, имеет ли смысл менять «старую» программу? Очевидно, нет, так как переделка и отладка отнимут время и у вас, и у машины, а в итоге выигрыша, вероятнее всего, не будет.

Другое дело, если программа предназначена для многократного использования либо требует для своего выполнения много машинного времени. В этом случае надо тщательно продумать алгоритм и на любой стадии разработки программы вносить изменения, улучшающие ее параметры.

Особые требования предъявляются к стандартным подпрограммам, вычисляющим на ЭВМ элементарные математические функции

$$\sin x, \cos x, \operatorname{tg} x, \sqrt{x} \text{ и т. д.}$$

Эти функции используются при решении большинства задач. Они срабатывают миллионы раз. Поэтому неоптимальность таких программ ведет к потере многих часов машинного времени.

Сейчас мы приведем один из простейших алгоритмов вычисления $\sin x$.

Вычисление функции $\sin x$ разложением в ряд. В курсе математического анализа доказывается, что если угол задан в радианах, то справедливо равенство

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots,$$

т. е.

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

— сумма сходящегося бесконечного ряда.

Обозначим через a_n члены этого ряда:

$$a_0 = x, \quad a_1 = -\frac{x^3}{3!}, \quad a_2 = \frac{x^5}{5!}, \quad a_3 = -\frac{x^7}{7!} \text{ и т. д.}$$

Сумма сходящегося бесконечного ряда с некоторой погрешностью может быть заменена суммой конечного ряда.

Пусть мы хотим найти значение $\sin x$ с некоторой точностью ε .

Так как $\lim_{n \rightarrow \infty} \frac{x^{2n+1}}{(2n+1)!} = 0$, то найдется такой номер N , что для всех $n > N$ выполнится соотношение $|a_n| < \varepsilon$.

Известно, что сумму такого бесконечного знакопеременного ряда $\sum_{n=0}^{\infty} a_n$ можно заменить суммой конечного ряда $\sum_{n=0}^N a_n$ с погрешностью, не превышающей первого отброшенного члена a_{N+1} , т. е.

$$\left| \sum_{n=0}^{\infty} a_n - \sum_{n=0}^N a_n \right| \leq |a_{N+1}| < \varepsilon.$$

Вычислив очередной член ряда, проверяют условие $|a_n| < \varepsilon$. Если оно не выполнено, то a_n прибавляют к сумме предыдущих членов и вычисляют a_{n+1} . Если $|a_n| < \varepsilon$,

то полученная сумма $\sum_{n=0}^N a_n$ дает значение $\sin x$ с точностью ε .

Приведем подпрограмму-функцию, осуществляющую этот алгоритм. Ее формальными параметрами служит аргумент X и точность EPS .

```

FUNCTION SINUS (X, EPS)
  S=0.
  A=X
  B=3.
  X2=X**2
3 S=S+A
  A=-A*X**2/(B*(B-1))
  B=B+2
  IF (ABS(A)-EPS) 2, 3, 3
2 SINUS=S
  RETURN
END

```

Поясним работу этой программы. В ячейке S накапливается сумма $\sum_{n=0}^N a_n$. Первоначальное значение ее — нуль.

Ячейка A отведена под переменную a_n ($a_0=x$); B содержит $2n+1$.

Оператор с меткой 3 прибавляет значение очередного a_n к сумме предыдущих членов.

Оператор $A=-A*X**2/(B*(B-1))$ вычисляет следующий член. При этом используется рекуррентное соотношение

$$a_n = a_{n-1} \left(-\frac{x^2}{2n(2n+1)} \right), \quad n > 0,$$

т. е. последующий член отличается от предыдущего тем, что он

- 1) противоположен по знаку,
- 2) в x^2 раз больше,
- 3) в $2n(2n+1)$ раз меньше.

Действительно,

$$\begin{aligned}
 a_0 &= x, \quad a_1 = a_0 \left(-\frac{x^2}{2 \cdot 3} \right) = -\frac{x^3}{3!}, \\
 a_1 &= -\frac{x^3}{3!}, \quad a_2 = a_1 \left(-\frac{x^2}{4 \cdot 5} \right) = \frac{x^5}{5!}, \\
 a_2 &= \frac{x^5}{5!}, \quad a_3 = a_2 \left(-\frac{x^2}{6 \cdot 7} \right) = -\frac{x^7}{7!} \text{ и т. д.}
 \end{aligned}$$

Для вычисления следующего a_n к В прибавляется 2, а оператор

IF (ABS(A)—EPS) 2, 3, 3

проверяет условие $|a_n| < \epsilon$. В случае $|a_n| \geq \epsilon$ управление передается оператору с меткой 3 и продолжается вычисление членов ряда.

Если $|a_n| < \epsilon$, то имени подпрограммы присваивается значение вычисленной суммы и происходит возврат в вызывающую программу, которая получит значение $\sin x$ с точностью ϵ .

Пример. Составить таблицу синусов для аргумента x , меняющегося от 0 до 9,9 с шагом $h=0,1$.

Таблицу будем заполнять так. Первый вертикальный столбец отведем для целой части аргумента x , первую строку — для дробной. Тогда значения функций будем располагать на пересечении строки и столбца, соответствующих целой и дробной части аргумента. Например, значение $\sin 3.1$ занесем в четвертую строку и второй столбец таблицы (так как первая строка и первый столбец соответствуют нулевым значениям) и т. д.

В качестве подпрограммы-функции, вычисляющей $\sin x$, возьмем составленную нами подпрограмму SINUS.

```
PROGRAM STAB
DIMENSION TAB(11, 11)
DO 1 K=2,11
  TAB (K, 1)=K-2
1 TAB(1, K)=(K-2)*0.1
  TAB(1, 1)=0
  DO 2 K=2, 11
    DO 2 L=2, 11
2 TAB(K,L)=SINUS(TAB(K, 1)
  C+TAB(1, L), 1.E-6)
  PRINT 3, ((TAB(I, J), J=1, 11),
  CI=1, 11)
3 FORMAT (1H1, 45X
  C16H ТАБЛИЦА СИНУСОВ//
  C(2X, 11F10.6))
END
```

Программа работает так. Сначала заносятся в первый столбец массива В (элементы В(К, 1)) целые значения аргумента 0; 1; 2; 3; ...; 9, а в первую строку (элементы В(1, К)) — дробные 0, 0; 0,1; 0,2; 0,3; ...; 0,9. После этого присваивается нуль элементу TAB(1, 1) и начинается

заполнение таблицы значениями синуса. В качестве первого фактического параметра служит арифметическое выражение $TAB(K, 1) + TAB(1, L)$, которое вычисляет значение аргумента X : $TAB(K, 1)$ содержит целую часть, $TAB(1, L)$ — дробную. Вторым фактическим параметром содержит точность вычислений.

В элемент $TAB(K, L)$ засылается значение синуса от аргумента, целая часть которого соответствует строке K , а дробная — столбцу L .

После заполнения всей таблицы она выводится на печать по строкам, содержащим 11 элементов.

Приведем результаты работы программы.

0.000 000	0.000 000	0.100 000	0.200 000...
0.000 000	0.000 000	0.099 833	0.198 669
1.000 000	0.841 471	0.891 207	0.932 039
2.000 000	0.909 297	0.863 209	0.808 496
3.000 000	0.141 120	0.041 581	—0.058 374
4.000 000	—0.756 802	—0.818 277	—0.871 576
5.000 000	—0.958 924	—0.925 815	—0.883 455
6.000 000	—0.279 416	—0.182 163	—0.083 089
7.000 000	0.656 987	0.728 969	0.793 668
8.000 000	0.989 358	0.969 890	0.940 731
9.000 000	0.412 118	0.319 098	0.222 89 ...

Остановимся подробнее на первой спецификации оператора `FORMAT — 1H1`. Она предписывает вывести на печать в качестве первого символа строки единицу. Такое указание воспринимается печатающим устройством как приказ печатать последующую информацию с новой страницы листинга.

Если в приведенном нами алгоритме вычисления $\sin x$ не использовать рекуррентного соотношения, а считать по формуле

$$a_n = (-1)^n \frac{x^{2n+1}}{(2n+1)!},$$

то сразу ухудшаются параметры программы.

Во-первых, вычисляться $\sin x$ будет гораздо дольше, так как придется организовать цикл для нахождения $(2n+1)!$.

Во-вторых, для достаточно больших x величины x^{2n+1} либо $(2n+1)!$ (если вычислять отдельно числитель и знаменатель) могут переполнить разрядную сетку машины. Но даже вычисления по рекуррентной формуле не спасают программу от существенного недостатка. Дело

в том, что алгоритм на ЭВМ работает не для любых значений аргумента. Для достаточно больших x члены ряда сначала растут по абсолютной величине, а затем уже начинают уменьшаться. Например, при $|x| > \sqrt[3]{6}$ второй член разложения a_1 больше первого члена (по модулю).

Рассмотрим поведение членов ряда при x порядка 10^3 . Тогда примерно до $n=500$ абсолютные величины a_n возрастают с ростом n . Уже $|a_4| > 10^{19}$, что приведет к аварийному останову (авосту) ЭВМ БЭСМ-6 при попытке вычислить a_4 .

Но если даже x таково, что аварийный останов не произошел, нельзя быть уверенным, что результат выдан правильный.

Например, пусть $x=10$. Нетрудно посчитать, что $|a_5|$ разложения в ряд будет порядка 10^4 . На ЕС ЭВМ эта величина вычисляется с семью верными знаками, т. е. с тремя верными знаками после запятой. А так как в результате суммирования членов ряда старшие (верные) разряды пропадают, то значение $\sin x$ выдается всего с тремя верными знаками.

Для больших x число верных знаков еще меньше. Может случиться, что погрешность вычислений будет одного порядка с результатом либо даже превзойдет его.

В [14] приведен конкретный пример вычисления на ЭВМ синуса по аналогичной программе. Вычислялся $\sin\left(\frac{\pi}{6} + 2\pi n\right)$ для $n=0, 1, 2, 3, \dots, 8$ на машине, позволяющей работать с восемью верными знаками. Результаты представлены в табл. 1.

Т а б л и ц а 1

30°	0.523 598 78	0.499 999 99
390°	6.806 784 15	0.499 999 93
750°	13.089 969 52	0.500 135 07
1110°	19.373 154 88	0.516 584 90
1470°	25.656 340 12	24.254 018
1830°	31.939 525 60	14 380.237
2190°	38.222 711 09	25 902 480.
2550°	44.505 896 09	—130 402 508.
2910°	50.789 081 57	—83 272 283.

В первой колонке приведены углы в градусах, во второй колонке — в радианах, в третьей — значения синуса, полученные с точностью $\epsilon=10^{-8}$.

Для каждого аргумента точное значение синуса равно 0,5.

Из приведенной таблицы следует, что уже для $x = 1470^\circ$ погрешность вычислений во много раз превысила искомую величину.

Это произошло потому, что для угла в 1470° ($x = 25.656\ 340\ 12$ радиан) наибольший член разложения синуса в ряд приближенно равен 5 503 768 000, а вычисления велись с восемью верными знаками. Следовательно, этот член получен с абсолютной погрешностью 50 (погрешность округления) и значение синуса лежит в интервале

$$24.254\ 018 \pm 50.$$

Если подобные вычисления провести на ЕС ЭВМ, позволяющей получить всего 7 верных знаков, то погрешности будут еще больше.

ЭВМ БЭСМ-6 дает 12 верных знаков, но уже для $x = 2550^\circ$ происходит авост, так как максимальный по абсолютной величине член превышает 10^{19} .

В табл. 2 приведены результаты работы на ЭВМ БЭСМ-6 подпрограммы SINUS, описанной выше.

Т а б л и ц а 2

30°	0.523 598 78	0.500 000 00
390°	6.806 784 15	0.500 000 00
750°	13.089 969 52	0.500 000 00
1110°	19.373 154 88	0.500 013 57
1470°	25.656 340 12	0.519 137 63
1830°	31.939 525 60	5.080 034 50
2190°	38.222 711 09	6468.179 740 77

Вычисления с двойной точностью несколько улучшают картину, но принципиальной роли не играют, так как для достаточно больших аргументов погрешность недопустимо велика. Разбирая так подробно программную реализацию и вычисление на ЭВМ синуса по формуле

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n(2n+1)},$$

мы хотели показать читателю, что вопрос об оптимальности программы неразрывно связан с вопросом о точности вычислений.

Каждая программа должна обеспечивать необходимую точность при различных значениях переменных.

Отлаживая программу, надо не только проверять, правильно ли реализован алгоритм, но и оценивать погрешности, получаемые на ЭВМ при экстремальных значениях переменных.

Программист должен хорошо представлять, в каких границах могут изменяться переменные, и учитывать это при выборе алгоритма.

Например, формула интегрирования по Симпсону

$$I = \frac{h}{3} (y_0 + 4y_1 + 2y_2 + 4y_3 + \dots + 4y_{n-1} + y_n)$$

должна быть представлена как

$$I = \frac{hy_0}{3} + \frac{4}{3} hy_1 + \frac{2}{3} hy_2 + \frac{4}{3} y_3 + \dots + \frac{hy_n}{3}$$

для достаточно больших значений функции, так как в противном случае может произойти переполнение (авост).

Наоборот, при малых значениях функции лучше сначала просуммировать y_i , а потом уже умножить на малую величину h , чтобы избежать получения машинного нуля.

К сожалению, нельзя дать рекомендации на все случаи. Некоторые советы начинающему программисту содержатся в [14]. Но надо всегда помнить об авостах, машинных нулях и о том, что машина может дать лишь ограниченное число верных знаков.

ЗАКЛЮЧЕНИЕ

В этой небольшой книге мы коснулись лишь немногих вопросов, связанных с обширной областью человеческой деятельности — программированием на ЭВМ.

Теперь, когда настала пора расставаться с читателем, мы попробуем еще раз окинуть взором тот предмет, о котором шла речь на страницах этой книги. Нам хотелось бы, в частности, схематично рассмотреть некоторые вопросы программирования, которые не удалось здесь затронуть.

В программировании можно выделить несколько самостоятельных разделов. Такими разделами являются, по нашему мнению, системное, прикладное и теоретическое программирование. Сознвая условность такого разделения, мы все же расскажем вкратце о каждом из этих разделов.

Системное программирование охватывает круг вопросов, связанных с разработкой программ, входящих в состав математического (или программного) обеспечения ЭВМ. Это такие программы, без которых невозможна, вообще говоря, нормальная эксплуатация ЭВМ.

Среди программ, составляющих систему математического обеспечения современной ЭВМ, особую роль играют те, которые работают совместно с аппаратурой машины. Эту часть математического обеспечения обычно называют операционной системой (ОС). Операционная система является, так сказать, «программным приложением» к ЭВМ и поставляется заводом-изготовителем вместе с комплектом машины. Отметим, что довольно часто в состав ОС включают и некоторые обслуживающие программы, а также трансляторы с языков программирования.

Современные ОС представляют собой огромные по размерам комплексы программ, содержащие до несколь-

ких миллионов команд. Программы, входящие в состав ОС, должны быть особенно надежны в эксплуатации и работать достаточно быстро. Поэтому в разработке ОС принимают участие наиболее квалифицированные программисты, которых обычно называют системными программистами.

Все остальные (не системные) программы принято называть прикладными. Для выполнения таких программ и существуют, собственно, ЭВМ. Вопросы, относящиеся к разработке прикладных программ, являются предметом *прикладного программирования*.

Среди прикладных программ значительную часть составляют вычислительные (называемые также научными или инженерными). Вопросам, связанным с разработкой таких программ, посвящена большая часть этой книги.

Прикладные программы часто объединяют в библиотеки, формируемые, например, по тематическому признаку. Такие библиотеки имеются практически на каждой современной ЭВМ.

Хотя программирование и является областью практической деятельности, для успешного его развития необходимы и теоретические исследования. Сюда относится, например, разработка теории языков программирования, методов трансляции и оптимизации программ. Эти и ряд других теоретических вопросов программирования составляют предмет *теоретического программирования*.

Отметим, что рассмотренные разделы программирования выделились в процессе его развития, которое еще далеко не закончено.

Мы надеемся, что нам хотя бы частично удалось достичь двух целей: ознакомить читателей с некоторыми аспектами программирования и помочь им в практическом освоении этого предмета.

ОТВЕТЫ И РЕШЕНИЯ

1. 78.05; 0.0000002 либо $2.E-7$; -3800000 , либо $-3.8E6$; 18.905; $-1.E-10$

2. Наименованиями переменных могут быть следующие комбинации: C3A7, LB, KOPT, I7C, C71. Из них целые LB, KOPT, I7C

3. 1. а) $X = (B^{**}2 + C^{**}2)/D$

б) $E = (35.*X + Y)/A^{**}3$

в) $A = 3.*C/(25.*D)$

2. а) $\frac{A}{B}(C+D)$, б) $\frac{A(C+D)}{B}$, в) $\frac{A}{B(C+D)}$, г) $\frac{AC}{B} + D$

4. а) 0; б) 2; в) 1; г) 0; д) 7.

5. GO TO 100

6. F3.1, F6.3, F6.2, I2, I4, F5.1

7. PRINT 2, A, IB, C

2 FORMAT (F5.2, I6, F7.5)

8. 1) PROGRAM SQ

A = 15.7328

H = 0.031289

S = 0.5 * A * H

PRINT 1, S

1 FORMAT (10X, F10.7)

END

2) PROGRAM GEOM

B = 1.135619

Q = 0.999999

S = B * (1. - Q ** 10) / (1. - Q)

PRINT 1, S

1 FORMAT (10X, F12.6)

END

9. 1) PROGRAM ROOT

READ 10, A, B, C

10 FORMAT (3F7.2)

X1 = (-B + SQRT(B**2 - 4*A*C)) / (2*A)

X2 = (-B - SQRT(B**2 - 4*A*C)) / (2*A)

PRINT 20, X1, X2

20 FORMAT (10X, 6HКОРНИ=, E12.5, 3H_И_, E12.5)

END

на бумаге с 11-й позиции будет напечатано: КОРНИ=
(число X1) И (число X2)

- 2) PROGRAM SP
 READ 15, A, B, ALFA
 15 FORMAT (3F8.4)
 $S = A * B * 0.5 * \sin(ALFA)$
 $C = \sqrt{A^2 + B^2 - 2 * A * B * \cos(ALFA)}$
 $P = A + B + C$
 PRINT 21, S, P
 21 FORMAT (20X, 10ПЛОЩАДЬ S=, E12.5,
 *11ПЕРИМЕТР P=, E12.5)
 END
- на бумаге с 21-й позиции будет напечатано:
 ПЛОЩАДЬ (число) ПЕРИМЕТР (число)
- 3) а) $Y = \sqrt{A * B} * \sin(A) * \log_{10}(B) +$
 $+ \text{ABS}(\sin(A) * \sqrt{\text{ABS}(\sin(A))^2 - \cos(B)^2})$
 в) $Y = A / (2 * \sqrt{B * \sqrt{A}})$
10. IF (A) 1, 2, 2
 1 Y = -A
 GO TO 3
 2 Y = A
 3 CONTINUE
11. 1) PROGRAM AK
 A = 1.
 DO 10 I = 2, 15
 10 A = $\sqrt{A} + 1$
 PRINT 30, A
 30 FORMAT (10X, 2HA=, E12.5)
 END
- 2) PROGRAM SUM
 A = 2
 S = 2
 DO 5 I = 2, 20
 $A = A^2 / (A + 3)$
 5 S = S + A
 PRINT 10, S
 10 FORMAT (10X, 2HS=, E12.5)
 END
12. 1. DIMENSION C(8)
 DATA C/15.2, 5*17.3, 1.5, 3.2/
 2. а) недопустимое для БЭСМ-6 смешение типов;
 б) комплексный;
 в) недопустимое для БЭСМ-6 смешение типов,
13. A = -B
 IF (X.GT.3) A = -B + B * $\sqrt{X - 3}$

Приложение I

ТАБЛИЦА ОТЛИЧИЙ ВЕРСИЙ ФОРТРАНА

В таблице приведены те возможности языка фортран, реализации которых имеют отличия в разных версиях «старого» и «нового» фортрана. В основе «старого» лежит стандарт ANSI 1966 г., поэтому для краткости этот фортран в таблице назван «фортран 66». В основе «нового» фортрана лежит стандарт ANSI 1977 г.; этот язык в таблице назван «фортран 77». Версия Ф77ЕС совместима с версией VS FORTRAN, являющейся расширением стандарта ANSI 1977 г. Версия СПФ77 соответствует транслятору, разработанному в Институте НИИ ЭВМ БССР (Минск).

Знаком + отмечены те возможности, которые допускает данный транслятор. Например, фортран-Дубна допускает русские буквы в именах, но это не означает, что нельзя использовать латинские буквы. Знаками ~ + отмечены те возможности, в использовании которых есть отличия по сравнению с другими версиями.

Возможности языка, реализуемые транслятором	Фортран 66				Фортран 77			
	ГОСТ 25056-78	БЭСМ-6		ЕС	Стандарт ANSI	БЭСМ-6	ЕС	
		ДУБНА	ГДР	IV		FOREX	Ф77ЕС	СПФ77
1. Строки, имена, метки, алфавит								
1. Максимальное число строк продолжения	19	19	19	19	19	∞	19	19
2. Русские буквы в именах	—	+	+	—	—	+	—	—
3. Использование символа \$ (⌘) в качестве буквы	—	—	—	+	—	—	+	+
4. Использование символа \$ в качестве разделителя операторов	—	+	+	—	—	+	—	—
5. Использование символа & для альтернативного выхода	—	—	—	+	—	—	—	—
6. Свободный формат оператора	—	—	—	—	—	—	+	—
7. Одинаковые метки у выполняемого оператора и оператора FORMAT	—	+	—	—	—	—	—	—

Возможности языка, реализуемые транслятором	Фортран 66				Фортран 77			
	ГОСТ 25056-78	БЭСМ-6		ЕС	Стандарт ANSI	БЭСМ-6	ЕС	
		ДУБНА	ГДР	IV		FOREX	Ф77ЕС	СПФ77
8. Метки у невыполняемых операторов (не считая FORMAT)	—	+	—	—	+	—	+	+
9. Комментарий может начинаться не только с символа C (в первой позиции)	—	—	—	—	+	—	+	+
10. Пустая карта трактуется как комментарий	—	—	—	—	+	—	+	+
2. Типы данных								
11. Типы:								
REAL*4, REAL*8	—	—	—	+	—	+	+	+
INTEGER*2, INTEGER*4	—	—	—	+	—	—	+	+
COMPLEX*8, COMPLEX*16	—	—	—	+	—	—	+	+
LOGICAL*1, LOGICAL*4	—	—	—	+	—	—	+	+
REAL*16, COMPLEX*32	—	—	—	—	—	—	+	+
CHARACTER	—	—	—	—	+	+	+	+
BIT	—	—	—	—	—	+	—	—
12. Операторы:								
RECORD	—	—	—	—	—	+	—	—
BASE	—	—	—	—	—	+	—	—
IMPLICIT	—	—	—	+	+	+	+	+
13. Засылка данных посредством операторов описания типа	—	—	—	+	—	—	+	—

14. Константа четверной точности вида $\pm m.nQ \pm p$	—	—	—	—	—	—	+	+
3. Константы								
15. Восьмеричные константы	—	+	—	—	—	+	—	—
16. Шестнадцатеричные константы в операторе DATA	—	—	—	+	—	~+	+	—
17. Символьные константы в форме с H в апострофах	+	+	+	+	—	—	—	—
	—	+	+	+	+	+	+	+
18. Комплексная константа, состоящая из компонент типа DOUBLE PRECISION	—	—	—	+	—	—	+	+
4. Массивы и элементы массивов								
19. Максимальная размерность массивов	3	3	3	7	7	8	7	7
20. Задание элемента массива с числом индексов, меньшим размерности массива (не считая EQUIVALENCE)	—	+	+	—	—	—	—	—
21. Задание в декларативных операторах диапазона изменения индекса	—	—	—	—	+	+	+	+
22. Задание в операторе EQUIVALENCE многомерного массива как одномерного	+	+	+	+	—	—	—	+
23. Нулевые и отрицательные индексы в DIMENSION	—	—	—	—	+	+	+	+
24. Целые индексные выражения в декларативных операторах	—	—	—	—	+	~+	+	+
5. Операции и выражения								
25. Смещение операндов типа двойной точности и комплексного в операциях +, —, *, /	—	—	—	+	—	+	+	+

Возможности языка, реализуемые транслятором	Фортран 66				Фортран 77			
	ГОСТ 25056-78	БЭСМ-6		ЕС	Стандарт ANSI	БЭСМ-6	ЕС	
		ДУБНА	ГДР	IV		FOREX	Ф77ЕС	СПФ77
26. Порядок вычисления выражения a**b**c: слева направо справа налево		+	+	-	-	-	-	-
27. Возведение комплексного числа в комплексную степень	-	-	-	-	+	-	+	+
28. Операция «логическое тождество». EQN.	-	-	-	-	+	-	+	+
29. Операция «исключающее ИЛИ» .EOR.	-	-	-	-	-	+	-	-
30. Текстовые константы как целые в выражениях	-	+	+	-	-	-	-	-
31. В операциях .EQ. и .NE. операнды: комплексные символьные	- -	- -	- -	- -	+	+	+	+
6. Операторы присваивания								
32. Символьная переменная в левой части оператора присваивания	-	-	-	-	+	+	+	+
33. Нет ограничений на численные *) типы левой и правой частей оператора присваивания	-	-	-	+	+	-	+	+

7. Управляющие операторы

34. Оператор GO TO по предписанию без списка меток	—	+	+	—	+	+	+	+
35. В вычисляемом GO TO (n_1, n_2, \dots, n_k) m — выражение целого типа	—	—	—	—	+	+	+	+
m — вне диапазона $[1; K]$ — GO TO игнорируется	—	—	—	+	+	и REAL +	+	+
36. Переменная цикла типа REAL и DOUBLE PRECISION	—	—	—	—	+	+	+	+
37. Начальное, конечное значения и шаг переменной цикла могут: быть выражениями принимать нулевые (кроме шага) и отрицательные значения значения переопределены внутри цикла	— — —	— — —	— — —	— — —	— — +	— — +	— — +	— — +
38. Проверка на конец цикла до выполнения цикла после	— +	— +	— +	— +	— +	— +	— +	— +
39. Структурный оператор IF DO	— — —	— — —	— — +	— — —	— — —	— — —	— — —	— — —
40. Условный логический оператор вида IF (L) m, n	—	+	+	—	+	+	+	+
41. Оператор END в качестве RETURN (SUBROUTINE, FUNCTION без RETURN)	—	—	—	+	+	+	+	+
42. Оператор RETURN n	—	—	—	+	+	+	+	+
43. Оператор RETURN в головном модуле в качестве STOP	—	+	+	+	—	+	+	+

*) Численными называют типы: INTEGER, REAL, COMPLEX, DOUBLE PRECISION

Возможности языка, реализуемые транслятором	Фортран 66				Фортран 77			
	ГОСТ 25056-78	БЭСМ-6		ЕС	Стандарт ANSI	БЭСМ-6	ЕС	
		ДУБНА	ГДР	IV		FOREX	Ф77ЕС	СПФ77
44. Оператор STOP m, где m — текст либо пятизначная целая константа	—	+	+	—	+	+	+	+
8. Декларативные операторы, функции, подпрограммы								
45. Оператор PARAMETER	—	—	—	—	+	+	+	+
46. Появление имени общего блока более одного раза в одном модуле (в частности, в одном операторе COMMON)	+	—	—	+	+	+	+	+
47. Элемент многомерного массива в операторе EQUIVALENCE	+	—	—	+	+	+	+	+
48. Оператор DATA со скобками	—	+	+	—	—	—	—	—
49. Засылка начальных данных оператором DATA в переменные помеченного блока	—	+	+	—	—	—	—	—
непомеченного блока (вне BLOCK DATA)	—	+	+	—	—	+	—	—
50. Неявный цикл в операторе DATA	—	+	+	—	+	+	+	+
51. Комбинация в операторе DATA: любых числовых типов для переменной и константы	—	+	+	—	+	+	+	+
символьной константы и переменной типа: REAL, INTEGER, COMPLEX, LOGICAL	—	+	+	+	—	—	—	— *)
52. Оператор NAMELIST	—	—	—	+	—	—	+	— *)

53. Указание длины в операторе FUNCTION	—	—	—	+	—	+	+	+
54. Элемент массива в функции-операторе (внутренней функции)	—	+	+	—	—	+	—	—
55. Функция типа CHARACTER	—	—	—	—	+	+	+	+
56. FUNCTION без параметров	—	—	—	—	+	+	+	+
57. В операторе EQUIVALENCE имя массива без индексов	—	+	+	—	+	—	+	+
58. Целые индексные выражения в декларативных операторах	—	—	—	—	+	+) **)	+	+
59. Оператор BLOCK DATA с именем	—	—	—	—	+	+	+	+
60. Оператор PROGRAM	—	+	+	—	+	+	+	+
61. Головной модуль без заголовка	+	—	—	+	+	+	+	+
62. Вызов параметров по значению (by value)	—	+	+	+	—	+	—	—
63. Формальные параметры, заключенные в косые черты	—	—	—	+	—	+	—	—
64. Фактический параметр-метка	—	—	—	+	+	+	+	+
9. Порядок операторов в модуле								
65. Расположение EQUIVALENCE произвольное среди декларативных операторов	+	+	+	+	+	—	+	+
66. Расположение DATA произвольное среди декларативных операторов	—	+	+	+	—	—	—	—
произвольное среди выполняемых операторов	+	—	+	+	+	—	+	+

*) Будет реализован начиная с версии 3.0 СПФ77

**) Для транслятора FOREX допустимы только целые константные выражения

Возможности языка, реализуемые транслятором	Фортран 66				Фортран 77			
	ГОСТ 25056-78	БЭСМ-6		ЕС	Стандарт ANSI	БЭСМ-6	ЕС	
		ДУВНА	ГДР	IV		FOREX	Ф77ЕС	СПФ77
10. Операторы ввода-вывода								
67. Константы в списке вывода	—	+	—	—	+	+	+	+
68. Выражения в списке вывода	—	—	—	—	+	+	+	+
69. Пробелы при вводе: игнорируются	—	+	+	—				
воспринимаются как нули	+	—	—	+				
программно устанавливается режим а) либо б)	—	—	—	—	+	+	+	+
70. Вывод символа (не управляющего) в первую позицию	—	+	+	—	—	+	—	—
71. Оператор ввода-вывода прямого доступа	—	—	—	~+	+	— *)	+	+
72. Спецификации ERR и END в опера- торах ввода	—	—	—	+	+	—	+	+
73. Операторы OPEN и CLOSE	—	—	—	—	+	— *)	+	+
74. Операторы ENCODE и DECODE	—	+	+	—	—	+	—	—
75. Символьная переменная или константа в качестве спецификации формата в операторе ввода-вывода (переменный формат)	—	+	+	—	+	+	+	+
76. Операторы асинхронного ввода-вывода: READ, WRITE, WAIT	—	—	—	—	—	—	+	+

Приложение II

СТАНДАРТНЫЕ ФУНКЦИИ

В приведенной ниже таблице указаны имена стандартных функций для наиболее распространенных версий (диалектов) языка фортран, имеющихся на машинах БЭСМ-6 и ЕС ЭВМ, а также для тех стандартов, на которых базируются рассматриваемые версии фортрана. Для версии СПФ77 список стандартных функций соответствует стандарту ANSI.

В левой колонке таблицы указаны имена функций. В скобках после имени функции указаны ее параметры (аргументы). Тип параметра определяется по первой букве его наименования: I—целый, R—вещественный, D—двойной точности, C—комплексный, Q—четверной точности, H—целый половинной длины (2 байта), L—логический, T—текстовый (CHARACTER), B—битовый, A—адресный. Начальные буквы CD означают, что параметр типа комплексный двойной точности, CQ—комплексный четверной точности. Если наименование параметра начинается с символа *, то данный параметр может быть любого численного типа (т. е. кроме L, T, B и A).

Если функция, указанная в левой колонке, имеется в данной версии фортрана, то в соответствующей колонке указан тип этой функции (I—целый и т. п.). После обозначения типа в скобках может быть указана буква U (например I(U)). Это означает, что имя данной функции является универсальным и допускает параметры любого численного типа, предусмотренного в данной версии. Обозначение *(U) соответствует универсальным именам функции, для которых тип результата тот же, что и тип параметра (параметров). Прочерк (—) означает, что соответствующей функции нет в данной версии языка фортран.

Стандартная функция	Фортран 66			Фортран 77		
	ГОСТ 12056-78	БЭСМ-6 (Дубна, ГДР)	ЕС ЭВМ	СПФ77	БЭСМ-6 FOREX	Ф77ЕС
INT(RX)	I	I	I	I(U)	I(U)	I(U)
IFIX(RX)	I	I	I	I	I	I
IDINT(DX)	I	I	I	I	I	I
HFIX(RX)	—	—	H	—	—	H
IQINT(QX)	—	—	—	—	—	I
REAL(CX)	R	R	R	R(U)	R(U)	R(U)
DREAL(CX)	—	—	—	—	—	D
QREAL(CX)	—	—	—	—	—	Q
FLOAT(IX)	R	R	R	R	R	R
DFLOAT(IX)	—	—	D	—	—	D
QFLOAT(IX)	—	—	—	—	—	Q
SNGL(DX)	R	R	R	R	R	R
SNGLQ(QX)	—	—	—	—	—	R
DBLE(RX)	D	D	D	D(U)	D(U)	D(U)
DBLEQ(QX)	—	—	—	—	—	D
CMPLX(RX, RY)	C	C	C	C(U)	C(U)	C(U)
DCMPLX(DX, DY)	—	—	CD	—	—	CD
QCMPLX(QX, QY)	—	—	—	—	—	Q
QEXT(RX)	—	—	—	—	—	Q
QEXTD(DX)	—	—	—	—	—	Q
ICHAR(TX)	—	—	—	I	I	I

Стандартная функция	Фортран 66			Фортран 77		
	ГОСТ 12056-78	ВЭСМ-6 (Дубна, ГДР)	ЕС ЭВМ	Стандарт ANSI	ВЭСМ-6 FOREX	ЕС ЭВМ
CHAR(IX)	—	—	—	T	T	T
AINTRX)	R	R	R	*(U)	*(U)	*(U)
DINT (DX)	—	—	—	D	D	D
QINT(QX)	—	—	—	—	—	Q
ANINT(RX)	—	—	—	*(U)	*(U)	*(U)
DNINT(DX)	—	—	—	D	D	D
NINT(RX)	—	—	—	*(U)	*(U)	*(U)
IDNINT,DX)	—	—	—	D	D	D
IABS(IX)	I	I	I	I	I	I
ABSRX)	R	R	R	*(U)	*(U)	*(U)
DABSDX)	D	D	D	D	D	D
CABSCX)	R	R	R	R	R	R
CDABS(CDX)	—	—	D	—	—	D
QABSQX)	—	—	—	—	—	Q
CQABSCQ)	—	—	—	—	—	Q
MOD(IX,IY)	I	I	I	*(U)	*(U)	*(U)
AMOD(RX,RY)	R	R	R	R	R	R
DMOD(DX,DY)	D	D	D	D	D	D
QMOD(QX, QY)	—	—	—	—	—	Q
SIGN(RX,RY)	R	R	R	*(U)	*(U)	*(U)
ISIGN(IX,IY)	I	I	I	I	I	I
DSIGN(DX,DY)	D	D	D	D	D	D
QSIGN(QX,QY)	—	—	—	—	—	Q
DIM(RX, RY)	R	R	R	*(U)	*(U)	*(U)
IDIM(IX, IY)	I	I	I	I	I	I
DDIM(DX,DY)	—	—	—	D	D	D
QDIM(QX,QY)	—	—	—	—	—	Q
DPROD(RX,RY)	—	—	—	D	D	D
MAX(X1,...,XN)	—	—	—	*(U)	*(U)	*(U)
MAXO(IX1,...,IXN)	I	I	I	I	I	I
AMAX1(RX1,...,RXN)	R	R	R	R	R	R
DMAX1(DX1,...,DXN)	D	D	D	D	D	D
QMAX1(QX1,...,QXN)	—	—	—	—	—	Q
AMAXO(IX1,...,IXN)	R	R	R	R	R	R
DMAX1(RX1,...,RXN)	I	I	I	I	I	I
MIN(X1,...,XN)	—	—	—	*(U)	*(U)	*(U)
MINO(IX1,...,IXN)	I	I	I	I	I	I
AMINI(RX1,...,RXN)	R	R	R	R	R	R
DMINI(DX1,...,DXN)	D	D	D	D	D	D
AMINO(IX1,...,IXN)	R	R	R	R	R	R
MINI(RX1,...,RXN)	I	I	I	I	I	I
LEN(TX)	—	—	—	I	I	I
INDEX(TX)	—	—	—	I	I	I
AIMAG(CX)	R	R	R	R	R	R
CONJG(CX)	C	C	C	C	C	C
DCONJC(CDX)	—	—	CD	—	—	CD

Стандартная функция	Фортран 66			Фортран 77		
	ГОСТ 12056-78	БЭСМ-6 (Дубна, ГДР)	ЕС ЭВМ	Стандарт ANSI	БЭСМ-6 FOREX	ЕС ЭВМ
QCONJG(CQ X)	—	—	—	—	—	CQ
SQRT(R X)	R	R	R	*(U)	*(U)	*(U)
DSQRT(D X)	D	D	D	D	D	D
CSQRT(CX)	C	C	C	C	C	C
CDSQRT(CD X)	—	—	CD	—	—	CD
QCQRT(Q X)	—	—	—	—	—	Q
CQSQRT(CQ X)	—	—	—	—	—	CQ
EXP(R X)	R	R	R	*(U)	*(U)	*(U)
DEXP(D X)	D	D	D	D	D	D
CEXP(CX)	C	C	C	C	C	C
CDEXP(CD X)	—	—	CD	—	—	CD
QEXP(Q X)	—	—	—	—	—	Q
CQEXP(CQ X)	—	—	—	—	—	CQ
LOG(X)	—	—	—	*(U)	*(U)	*(U)
ALOG(R X)	R	R	R	R	R	R
DLOG(D X)	D	D	D	D	D	D
CLOG(CX)	C	C	C	C	C	C
CDLOG(CD X)	—	—	CD	—	—	CD
QLOG(Q X)	—	—	—	—	—	Q
CQLOG(CQ X)	—	—	—	—	—	CQ
LOGL0(X)	—	—	—	*(U)	*(U)	*(U)
ALOG10(R X)	R	R	R	R	R	R
DLOG10(D X)	D	D	D	D	D	D
QLOG10(Q X)	—	—	—	—	—	Q
SIN(R X)	R	R	R	*(U)	*(U)	*(U)
DSIN(D X)	D	D	D	D	D	D
CSIN(CX)	C	C	C	C	C	C
CDSIN(CD X)	—	—	CD	—	—	CD
QSIN(Q X)	—	—	—	—	—	Q
CQSIN(CQ X)	—	—	—	—	—	CQ
COS(R X)	R	R	R	*(U)	*(U)	*(U)
DCOS(D X)	D	D	D	D	D	D
CCOS(CX)	C	C	C	C	C	C
CDCOS(CD X)	—	—	CD	—	—	CD
QCOS(Q X)	—	—	—	—	—	Q
CQCOS(CQ X)	—	—	—	—	—	CQ
TAN(R X)	—	R	R	*(U)	*(U)	*(U)
DTAN(D X)	—	—	D	D	D	D
QTAN(Q X)	—	—	—	—	—	Q
COTAN(R X)	—	—	R	—	—	R
DCOTAN(D X)	—	—	D	—	—	D
QCOTAN(Q X)	—	—	—	—	—	Q
ASIN(R X)	—	R	—	*(U)	*(U)	*(U)
ARSIN(R X)	—	—	R	—	R	R
DASIN(D X)	—	D	—	D	D	D
DARSIN(D X)	—	—	D	—	—	D

Стандартная функция	Фортран 66			Фортран 77		
	ГОСТ 12056-78	БЭСМ-6 (Дубна, ГДР)	ЕС ЭВМ	Стандарт ANSI	БЭСМ-6 FOREX	ЕС ЭВМ
QARSIN(QX)	—	—	—	—	—	Q
ACOS(RX)	—	R	—	*(U)	*(U)	*(U)
ARCOS(RX)	—	—	R	—	R	R
DACOS(DX)	—	D	—	D	D	D
DARCOS(DX)	—	—	D	—	—	D
QARCOS(QX)	—	—	—	—	—	Q
ATAN(RX)	R	R	R	*(U)	*(U)	*(U)
DATAN(DX)	D	D	D	D	D	D
QATAN(QX)	—	—	—	—	—	Q
ATAN2(RX,R)	R	R	R	*(U)	*(U)	*(U)
DATAN2(DX,DY)	D	D	D	D	D	D
QATAN2(QX,QY)	—	—	—	—	—	Q
SINH(RX)	—	—	R	*(U)	—	*(U)
DSINH(DX)	—	—	D	D	—	D
QSINH(QX)	—	—	—	—	—	Q
COSH(RX)	—	—	R	*(U)	—	*(U)
DCOSH(DX)	—	—	D	D	—	D
QCOSH(QX)	—	—	—	—	—	Q
TANH(RX)	R	R	R	*(U)	—	*(U)
DTANG(DX)	—	—	D	D	—	D
QTQNH(QX)	—	—	—	—	—	Q
ERF(RX)	—	—	R	—	—	R
DERF(DX)	—	—	D	—	—	D
QERF(QX)	—	—	—	—	—	Q
ERFC(RX)	—	—	R	—	—	R
DERFC(DX)	—	—	D	—	—	D
QERFC(QX)	—	—	—	—	—	Q
GAMMA(RX)	—	—	R	—	—	R
DGAMMA(DX)	—	—	D	—	—	D
ALGAMA(RX)	—	—	R	—	—	R
DLGAMA(DX)	—	—	D	—	—	D
LGE(TX,TY)	—	—	—	L	—	L
LGT(TX,TY)	—	—	—	L	—	L
LLE(TX,TY)	—	—	—	L	—	L
LLT(TX,TY)	—	—	—	L	—	L
IBIT(BX)	—	—	—	—	I	—
BBIT(BX)	—	—	—	—	A	—
LBIT(BX)	—	—	—	—	L	—
BIT(IX,IY)	—	—	—	—	B	—
BITB(AC,IY)	—	—	—	—	B	—
BITL(LX,IY)	—	—	—	—	B	—
IANY(TX)	—	—	—	—	I	—
NOTANY(TX)	—	—	—	—	I	—
NUNITS(BX)	—	—	—	—	I	—
ADDR(X)	—	—	—	—	A	—
RELOC(AX,IY)	—	—	—	—	A	—
ISIZE(X)	—	—	—	—	I	—

СПИСОК ЛИТЕРАТУРЫ

1. Базисный фортран/Под ред. Н. П. Брусенцова.— М.: Изд-во МГУ. 1982.
2. Б а я к о в с к и й Ю. М., В ь ю к о в а Н. И., Г а л а т е н к о В. А. и др. Расширенный фортран — форекс. Руководство для пользователя.— М.: Ин-т. прикл. мат. им. М. В. Келдыша АН СССР, 1983.
3. Б у х т и я р о в А. М., М а л и к о в Ю. П., Ф р о л о в Г. Д. Практикум по программированию на фортране / Под ред. Н. А. Криницкого.— М.: Наука, 1979.
4. Г о р е л и к А. М., У ш к о в а В. Л., Ш у р а — Б у р а М. Р. Мобильность программ на фортране.— М.: Финансы и статистика, 1984.
5. Д р е й ф у с М., Г а н г л о ф К. Практика программирования на фортране.— М.: Мир, 1978.
6. Й е н с е н К., В и р т Н. Паскаль.— М.: Финансы и статистика, 1982.
7. К а р п о в В. Я. Алгоритмический язык фортран / Под ред. Н. Н. Говоруна.— М.: Наука, 1976.
8. К а т ц а н Г. Язык фортран 77.— М.: Мир, 1982.
9. К е р н и г а н Б., П л о д ж е р Ф. Элементы стиля программирования.— М.: Радио и связь, 1984.
10. К о р о л е в Л. Н. Структуры ЭВМ и их математическое обеспечение.— М.: Наука, 1978.
11. К р и н и ц к и й Н. А. Алгоритмы вокруг нас.— М.: Наука, 1984.
12. Л а м у а т ь е Ж.— Л. Упражнения по программированию на фортране IV.— М.: Мир, 1978.
13. М а з н ы й Г. Л. Программирование на БЭСМ-6 в системе «Дубна» / Под ред. Н. Н. Говоруна.— М.: Наука, 1978.
14. М а к — К р а к е н Д., Д о р н У. Численные методы и программирование на фортране.— М.: Мир, 1977.
15. С а л т ы к о в А. И., М а к а р е н к о Г. И. Программирование на языке фортран для БЭСМ-6 / Под ред. Н. Н. Говоруна.— М.: Наука, 1984.
16. Т и х о н о в А. Н., К о с т о м а р о в Д. П. Рассказы о прикладной математике.— М.: Наука, 1979.
17. Фортран ЕС ЭВМ.— М.: Статистика, 1978.
18. Х ь ю з Д., М и ч т о м Д. Структурный подход к программированию. М.: Мир, 1980.
19. Ш е п е л е н к о В. И. Фортран БЭСМ-6,— Новосибирск: Наука, 1993.
20. Языки программирования Фортран и Базисный фортран. ГОСТ 23056—78, ГОСТ 23057—78.— М.: Издательство стандартов, 1982.

30 коп.

