БИБЛИОТЕЧКА ПРОГРАММИСТА

Ю. А. ПЕРВИН

Основы Фортрана

БИБЛИОТЕЧКА
ПРОГРАММИСТА

Ю. А. ПЕРВИН

ОСНОВЫ ФОРТРАНА

ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1972

518
П 26
УДК 519.95

Основы Фортрана. Первин Ю. А., Главная редакция физико-математической литературы изд-ва «Наука», 1972.

Книга знакомит читателя с самым распространенным в настоящее время языком программирования для вычислительных машин. Она может быть использована в качестве учебника начинающими программистами, ибо не предполагает предварительных знаний вычислительных машин, программирования и других языков. В равной мере книга будет полезной и опытным программистам, у которых возникла необходимость изучения нового для них языка Фортран.

Учитывая известную основную направленность Фортрана на инженерно-технические и научные вычислительные задачи, автор пытается показать эффективные возможности Фортрана также при решении экономических и информационно-логических задач, задач обработки геометрической информации и редактирования результатов.

Большое количество упражнений сопровождает изложение материала и заключает каждую главу. Кроме того, книга содержит примеры реальных программ, написанных на Фортране.

Иллюстративный материал книги составляют 13 таблиц и 37 рисунков.

2-2-4
57—72

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Появление настоящей книги обусловлено резко возросшим за последнее время в нашей стране интересом к алгоритмическому языку Фортран. Язык этот достаточно давно известен и широко распространен за рубежом, и ознакомление с зарубежным опытом в области программирования, вычислительной математики, прикладных вопросов кибернетики становится затруднительным без знания основ Фортрана. Последние модели отечественных машин получают трансляторы с Фортрана, и это в еще большей степени определяет необходимость появления пособия, знакомящего широкие круги программистов с основами языка.

Основные цели, которые читатель может поставить, приступая к чтению этой книги, — научиться читать и самому писать программы средней сложности; освоить развитый алгоритмический язык, с тем чтобы повысить производительность программирования по сравнению с использованием автокодов или, тем более, машинных языков; научиться использовать программы других вычислительных центров, составленные для других типов машин. Стремлением автора, соответственно, было желание помочь читателю справиться с этими целями. Описание Фортрана построено так, что не предполагает у читателя предварительных знаний, выходящих за пределы элементарных алгебраических понятий.

Таким образом, книга может служить первым учебником по программированию и будет потому полезной студентам инженерных и технических вузов, естественных факультетов университетов, где ведется преподавание программирования, ориентированное на решение научных и инженерно-технических задач, а также на-

чинающим программистам производственных вычислительных центров. Фортран имеет богатые возможности редактирования данных, полезных и в экономических приложениях. В книге есть упражнения на неклассические для Фортрана сюжеты: экономические задачи, обработка информационных массивов, редактирование результатов.

Книга может быть использована и опытными программистами, имеющими необходимость в освоении нового для них языка Фортран. Для этого круга читателей книга может показаться излишне подробной, и автор полагает, что квалификация таких читателей поможет им самостоятельно отобрать в книге интересный для них материал.

Книга рассчитана на лиц, использующих язык, а отнюдь не на разработчиков трансляторов с Фортрана. Это дало возможность в значительной степени сократить и упростить изложение за счет отказа от многих деталей языка, особенностей, связанных с конкретными операционными системами и машинами. Это позволяет, с одной стороны, считать, что изложенного в книге материала вполне достаточно, чтобы научиться программированию на Фортране, а с другой стороны, определить название книги не более как «Основы Фортрана».

В книге речь идет о наиболее распространенной в настоящее время версии языка, известной под названием Фортран IV. Это не означает, однако, претензий автора на описание всех деталей Фортрана IV. Читатель, которому предстоит писать на Фортране программы для какой-либо конкретной машины, всегда сможет легко познакомиться с особенностями машины и ее фортрановского транслятора, если он усвоил изложенные здесь основы языка.

Автор изучал Фортран в ходе решения конкретных производственных задач в вычислительном центре Electricité de France (Париж) и поэтому, выпуская эту книгу, не может не выразить благодарности своим французским коллегам, которые помогли ему многочисленными советами, материалами, примерами, возможностью машинных экспериментов.

Эта книга не могла бы увидеть свет без неоценимой помощи В. Н. Линьковой-Островенок.

Автор

ГЛАВА I

ВВОДНАЯ

§ 1. Общие замечания

Основное содержание книги составляют пять глав (II—VI). Каждая из этих глав заканчивается контрольными упражнениями, которыми читатель может проверить степень усвоения материала, прибегая к разделу «Ответы к упражнениям», помещенному в конце книги.

Кроме того, изложение материала иногда прерывается упражнениями, контролирующими усвоение более мелких разделов, также имеющими ответы.

Фортран стал сейчас одним из самых распространенных языков общения с машиной — почти все современные вычислительные машины «понимают» Фортран. Это говорит о том, что язык расположен вне специфики отдельных машин. И все же полностью игнорировать некоторые особенности машины при описании языка не удается. В первую очередь это относится к инструкциям ввода-вывода. Вот почему первый параграф главы III посвящен описанию носителей информации на конкретных машинах. Впрочем, этот параграф может быть при первом чтении опущен без ущерба для понимания грамматики Фортрана.

В главе VII разобраны три примера составления полных программ. Интересующийся читатель всегда сможет найти достаточное количество разнообразных программ конкретных задач в появляющихся в последнее время статьях, посвященных описанию и использованию Фортрана.

Вне основных задач книги стоит настоящая, вводная глава, дающая историческую справку о Фортране и его месте в системах программного обеспечения ЭВМ, а также общие сведения об основных этапах решения задачи на машине. Программисту, уже работавшему на машине,

нет нужды в этих известных сведениях. Однако, параграф о некоторых условностях орфографии Фортрана, связанных с его базой — английским языком, не следует опускать: он, по существу, является частью описания языка.

Рождение Фортрана относят к 1954 г. В ноябре этого года группа американских специалистов в области программирования, возглавляемая проф. Дж. В. Бэкусом, подготовила доклад, который сейчас считается первым сообщением, описывающим этот язык. Фортран довольно быстро завоевал мир программирования. Причина такой популярности Фортрана не случайна. Прежде всего, Фортран очень близок, с одной стороны, к языку обычной алгебры, с другой стороны, весь словарный запас его заимствован из естественного человеческого языка — английского. Он легок при изучении, программировании и отладке.

Язык был задуман для использования в сфере научных и инженерно-технических вычислений. Происхождение слова Фортран объясняет это как нельзя лучше: FORTRAN есть сокращение двух слов FORmula TRANslation — «преобразование формул». Но возможности у языка оказались гораздо более широкими: на Фортране легко описываются задачи с разветвленной логикой (моделирование интеллектуальной деятельности человека, решения игровых ситуаций, моделирование производственных процессов и т. д.), задачи редактирования результатов (составления форм ведомостей, построение графиков кривых по результатам расчетов), различного рода экономические задачи.

В 1958 г. была создана новая версия Фортрана со значительным расширением языка, которая названа Фортран II [1]). В Фортране II было введено понятие подпрограммы: были определены инструкции SUBROUTINE, CALL, RETURN и инструкция FUNCTION. Была добавлена и инструкция COMMON, обеспечивающая связи независимых подпрограмм и основной программы. Впервые конец программы стал отмечаться инструкцией END вместо существовавшего до этого указателя конца массива.

[1]) Существует русский перевод одного из описаний Фортрана II: статья «Фортран» в сборнике переводов «Автоматизация программирования» под ред. А. П. Ершова, Физматгиз, 1961.

В 1961 г. появился Фортран III. В язык были включены инструкции обработки сложных логических выражений и буквенно-символьной информации. Ряд дополнений к языку был сделан при разработке трансляторов с Фортрана III — тогда была обеспечена в Фортране арифметика с двойной точностью и арифметика комплексных чисел.

1962 г. — год рождения Фортрана IV. В Фортране IV появились инструкции декларации типа величин, определяющие способ задания переменных. Были допущены логические выражения в качестве аргумента инструкций условной передачи управления, введены инструкции организации информационных массивов. Из языка были исключены инструкции, зависящие от конкретной машины.

Фортран IV — не последняя версия Фортрана. Язык продолжает развиваться. Существенно развиты инструкции ввода и вывода, увеличена гибкость логических операций. Одно из наиболее интересных последних направлений в развитии языка, это так называемый разговорный Фортран, в котором учтены возможности диалогового режима «человек — машина».

Фортран оказал влияние на создание и развитие многих других языков. В частности, завоевывающий сейчас популярность диалоговый язык BASIC наследует многие черты Фортрана, получивший признание универсальный язык программирования PL-1 многое заимствовал из Фортрана. Одна из первых версий PL-1 носила даже название Фортран VI.

§ 2. Основные этапы решения задачи на машине

Электронная вычислительная машина (ЭВМ) может работать, лишь руководствуясь указаниями, полученными от человека в любой заранее согласованной форме: нажатие определенных кнопок на пульте машины, ввод по определенной системе кодирования перфорированных карт или чтение импульсов с магнитной ленты.

Перечень всех указаний машине для решения некоторой задачи называется *программой* этой задачи. Программа управляет машиной. Набор указаний, непосредственно воспринимаемых машиной, обычно весьма

ограничен. Например, машина не может выполнить указание

«Вычислить площадь круга радиуса 4». (1.1)

Как правило, машины могут в той или иной форме воспринимать указания для выполнения конкретных элементарных действий типа сложения двух заданных чисел, сравнения их и т. д. В частности, можно представить следующую совокупность указаний машине:

«Умножить число 4 само на себя»,
«Умножить результат предыдущего действия на 3,1416»,
«Записать результат предыдущего действия»,
«Закончить работу». (1.2)

Нетрудно видеть, что перечисленные четыре элементарных указания обеспечивают выполнение одного более сложного указания о вычислении площади круга радиуса 4. *Программирование*, т. е. составление программ, состоит в написании перечня элементарных, понятных машине указаний для решения поставленной задачи.

Каждая вычислительная машина воспринимает обычно только ей понятный набор указаний. Указания, воспринимаемые машиной «Минск-22», непонятны машине БЭСМ-6, и наоборот.

Большинство машин способно воспринимать указания лишь в закодированной форме. Слово «умножить», например, может быть зашифровано символом $\times$, символом *, цифровым кодом 05 и т. д. Каждая машина имеет свой набор кодов.

Для того чтобы написать программу для заданной вычислительной машины, необходимо знать набор воспринимаемых машиной указаний и способы кодирования указаний, принятые для этой машины. Эти формальные сведения — набор указаний и способ кодирования — представляют, таким образом, средство общения человека с машиной и потому заслуживают названия *языка*. Это *машинный язык*, на нем программист — специалист в области программирования — может давать машине задание любой степени сложности.

Задание, или программа, записывается в виде последовательности указаний машине сначала на бумаге или специальных формулярах. Затем с помощью перфори-

рующего устройства эта запись наносится на перфокарты, где система отверстий может обозначать тот или иной символ, воспринимаемый машиной. (В настоящее время появились устройства, позволяющие заменить перфорацию карт перенесением информации на магнитную ленту, а также устройства, считывающие в машину машинописный и стилизованный рукописный текст или графическую информацию.) Проперфорированные карты с помощью читающего устройства вводятся в машину, и информация размещается в ее запоминающем устройстве. Далее машина, управляемая введенной в нее программой, автоматически проводит решение, печатает результаты (если есть такое указание) и останавливается (если есть такое указание).

Этап решения задачи, описанный здесь коротким абзацем, как правило, бывает достаточно трудоемким и в организационном смысле разделяется на два этапа: отладка программы и собственно решение, или счет задачи. *Отладка* — это проверка на машине написанной программы с помощью специально заготовленных тестов — проверочных наборов данных. Практика показывает, что этот этап неизбежен: в программах, особенно больших по объему, трудно избежать ошибки. Ошибки любого происхождения — логическая ошибка в формулировке задачи, описка при написании программы, неверная перфорация и т. д. — должны быть обнаружены до того, как приступить к счету задачи, к ее решению на ЭВМ. Для некоторых сложных по структуре, но быстро решаемых задач процесс отладки может потребовать даже бо́льшего машинного времени, чем собственно решение: ошибка программы может быть обнаружена лишь после многократных проверок программы на машине.

Общая схема решения задачи на машине приведена на рис. 1, где пунктирная стрелка означает возврат программы с машины после отладки для исправления ее программистом в случае необходимости, а двойная стрелка — выдачу готового результата.

Во всей этой подготовительной и непосредственно вычислительной работе этап программирования является самым трудоемким. Это этап работы программиста, на котором он лишен помощи всех вспомогательных устройств, включаемых в работу на последующих этапах.

Кроме того, здесь необходимы специальные знания конкретного машинного языка. На этом этапе возникает еще одна значительная трудность: программы решения даже самых распространенных задач, составленные для одной машины, не годятся для решения тех же задач на машинах других моделей.

Этот трудоемкий этап существенно упрощается при использовании *алгоритмических языков*. Алгоритмический язык дает метод описания задач средствами, отличными от указаний машинного языка. Если эти средства

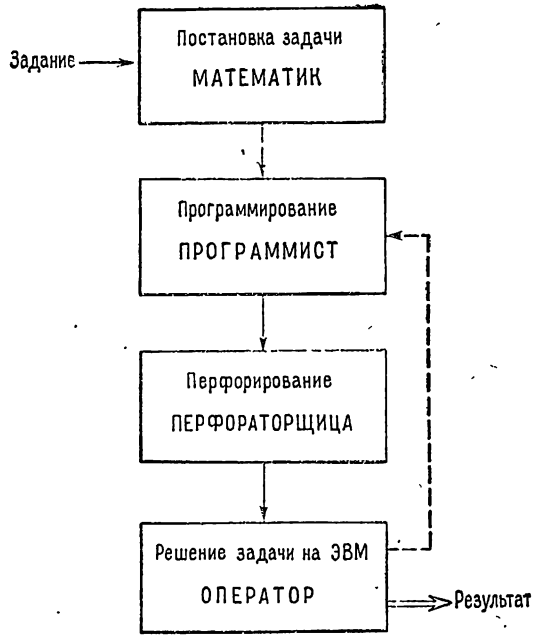

Рис. 1.

более характеризуют задачу, чем машину, на которой она решается, язык называется *проблемно-ориентированным*. С помощью проблемно-ориентированного языка можно относительно просто (по сравнению с программированием в машинных кодах) записать последовательность указаний машине. Замечательно, что программа, написанная на проблемно-ориентированном языке, годна для использования, вообще говоря, на любой вычисли-

тельной машине. Впрочем, оговорка необходима. Дело в том, что для использования на машине программы, написанной на проблемно-ориентированном алгоритмическом языке, необходима предварительная работа переводчика с алгоритмического языка на машинный. Работа эта трудоемкая, но легко формализуемая, а потому может быть поручена самой машине. Переводчик — это достаточно сложная программа, которая способна воспринимать символы, слова и фразы алгоритмического языка и, исходя из возможностей конкретной машины, переводить их на машинный язык. Такая программа-переводчик называется *транслятором* [1]).

Таким образом, машина может воспринимать программу на алгоритмическом языке, если она оснащена программой-транслятором с этого алгоритмического языка. В функции транслятора входит не только перевод — *трансляция* — с одного языка на другой, но и указание некоторых формальных ошибок, допущенных программистом при написании программы. Например, транслятор может сообщать о том, что программист забыл написать закрывающую скобку, если написана открывающая, что в программе используются числа, выходящие из допустимого машиной диапазона, и т. д. В зависимости от запрограммированных в трансляторе возможностей он способен анализировать и более тонкие ошибки и отклонения от правил языка.

Исходными данными для транслятора является программа, написанная на алгоритмическом языке (*программа-источник* с точки зрения транслятора). Результат транслятора — это *рабочая программа*, составленная в машинном языке конкретной машины.

Одним из алгоритмических языков программирования является язык Фортран, описанию которого посвящается настоящая книга.

Как и всякий другой алгоритмический язык, Фортран имеет свой алфавит, словарь и грамматику. Фортран является живым языком потому, что существующие сегодня вычислительные машины имеют фортрановские трансляторы. Некоторые машины располагают даже несколькими версиями трансляторов с Фортрана.

---

[1]) В зарубежной литературе трансляторы чаще именуют *компиляторами*.

Например, транслятор 1 может очень долго транслировать (составлять) рабочую программу по заданному описанию на Фортране, но зато обеспечивает быстрое решение рабочей программы на машине. Транслятор 2, напротив, имеет короткий период трансляции, но составленная им рабочая программа может оказаться не самой оптимальной версией с точки зрения расхода машинного времени.

Транслятор (или трансляторы) с Фортрана являются составной частью программного обеспечения машины, ее *системы программирования*. Система программирования может включать в себя трансляторы с разных языков, библиотеку часто используемых программ и управляющие программы, которые контролируют последовательное или параллельное прохождение задач через машину. Таким образом, программисту для решения задачи бывает недостаточно описать ее на Фортране. Он должен еще уметь общаться с системой программирования, например указать ей, с какой версией фортрановского транслятора он желает работать.

Однако, каким бы ни был фортрановский транслятор, программа, написанная на Фортране, не видит и не знает этих особенностей, как содержание книги, написанной на английском языке, не зависит (вообще говоря) ни от переводчика, взявшегося за ее перевод, ни от издательства, выпускающего ее в свет.

Оставив в стороне все, что связано с конкретными особенностями, — машины, трансляторы, системы программирования, настоящая книга описывает то, что свойственно самому языку Фортран, его грамматику.

§ 3. Некоторые условности орфографии Фортрана

Фортран построен на базе английского языка. Поэтому все выражения Фортрана, слова языка, названия переменных составлены исключительно из букв латинского алфавита. При практической работе с Фортраном следует учитывать этот факт. Например, переменную, выражающую количество изделий на складе, нельзя назвать ЗАПАС — алфавит Фортрана не допускает букв З и П. По всей видимости, лучше воспользоваться для этой цели английским словом STOCK (ЗАПАС), чем коверкать до неузнаваемости русское

слово ZAPAS. Впрочем, у программиста, не имеющего даже элементарных знаний английского, всегда есть хороший выход — обозначить эту величину одной буквой, например S.

В Фортране используются, однако, некоторые английские слова, употребления которых избежать невозможно. Таковы, например, названия инструкций READ, WRITE, IF и т. д. О русском значении этих слов можно при желании справиться в таблице 1 на стр. 20. Примеры, в которых необходим перевод на русский язык каких-либо терминов, в книге немногочисленны.

Автор был свидетелем интересного педагогического эксперимента: молодой педагог-математик, энтузиаст своего дела, с успехом рассказывал о Фортране своим ученикам — малышам 10-летнего возраста. Ребята хорошо понимали простые и четкие правила языка и сами писали несложные программы. Автор обратился к учителю с вопросом: «Не должны ли Вы предполагать предварительных знаний английского у Ваших ребят?» «Совсем нет, — ответил тот, — GO TO, CONTINUE, IF — это для них не английские слова, это слова Фортрана!» Этот эпизод рассказан в вводной главе, чтоб снабдить дополнительной порцией оптимизма читателей, приступающих к освоению Фортрана с английским словарным запасом, состоящим из слов good-bye и all right.

Читатель будет много раз иметь возможность убедиться, как близок Фортран к языку обычной алгебры при написании формул. Поэтому здесь сделаем замечание о самом значительном отличии: в обычных алгебраических формулах чаще всего употребляются для обозначения переменных и констант малые, строчные латинские буквы, тогда как в Фортране строчные буквы не разрешены — в алфавите Фортрана используются только большие (прописные) буквы. Поэтому алгебраическая формула

$$z = ax + by - d$$

в Фортране имела бы вид

```
Z = A*X+B*Y—D.
```

И еще одна орфографическая условность: перенося на перфокарты программу, написанную на бумаге программистом, перфораторщица должна уметь различать

букву О и цифру нуль — на перфокарте и внутри машины эти два символа одинакового начертания должны различаться. С этой целью буква О, в отличие от нуля, перечеркивается наклонной чертой Ø. Так, операция безусловного перехода к инструкции номер 100 на программном бланке будет записана:

GØ TØ 100

Здесь сначала следуют две буквы О в словах GØ TØ, поэтому они перечеркнуты, потом две цифры 0, они не перечеркнуты. Это соглашение действительно в дальнейшем на протяжении всей книги.

Читателю будет непривычна одна деталь в изображении чисел в Фортране — *десятичная точка*. Число π в Фортране запишется 3.141593, но не 3,141593. Точка, разделяющая целую и дробную части числа, всюду в книге именуется десятичной точкой, в отличие от точки в любом другом ее употреблении, например как символа конца фразы.

ГЛАВА II

ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА

Настоящая глава знакомит читателя с формами используемых в Фортране чисел, правилами образования простейших слов и фраз языка, а также способами их записи.

§ 1. Алфавит Фортрана

Слова и выражения Фортрана составляются из элементов трех категорий: букв, цифр и специальных символов.

Как было отмечено в предыдущей главе, Фортран использует 26 заглавных букв латинского алфавита.

$$\left.\begin{array}{l}\text{A, B, C, D, E, F, G, H, I, J, K, L, M,}\\ \text{N, O, P, Q, R, S, T, U, V, W, X, Y, Z.}\end{array}\right\} \quad (2.1)$$

В § 5 настоящей главы отмечается возможность расширения списка (2.1) буквами русского алфавита в трансляторах отечественных машин.

Фортран использует десять арабских цифр

0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Специальные символы Фортрана таковы:

\+ плюс
— минус
* звездочка
/ наклонная черта
= равно
(открывающая, левая скобка
) закрывающая, правая скобка
, запятая
. точка

Отдельно следует сказать о символе, не имеющем начертания — *пробеле*. Пробел может употребляться для разделения символов, слов или выражений.

В большинстве трансляторов, в частности во всех американских, используется еще один специальный символ — $.

Первые четыре символа из приведенного выше списка являются знаками арифметических операций и подробно рассмотрены в этой главе в следующем параграфе.

Символ = используется в Фортране в несвойственном обычному математическому языку смысле. Он не констатирует равенство двух величин, расположенных слева и справа от него. Этот символ является знаком операции *присваивания*: по этой операции значение величины или выражения, записанных справа от символа =, присваивается величине, записанной слева от него.

Так,

$$A = B$$

в Фортране означает, что величина A получает значение величины B;

$$A = B - C$$

свидетельствует о присваивании величине A значения разности величин B и C;

$$A = A + 1. \qquad (2.2)$$

говорит о том, что величина A в результате операции присваивания получает старое значение A (значение A до начала операции присваивания), увеличенное на единицу.

Выражение (2.2) отнюдь не уравнение, не имеющее корней, и знак равенства означает здесь *получает значение* или *замещается на*, но не «равно» в математическом понимании символа. Детально выражение (2.2) должно быть прочитано так: *сложить значение переменной* A *с единицей, и старое значение переменной* A *заменить новым, только что полученным.*

Скобки используются в Фортране для группирования аргументов и индексов либо для указания последовательности выполнения арифметических или логических операций.

Запятая — это разделитель в списках аргументов или индексов.

У *точки* есть два значения — разделитель элементов логических выражений (см. § 4 гл. III) и точка десятичная.

Грамматические правила, описывающие возможности и ограничения использования специальных символов, вводятся в книге по мере необходимости.

Слова в Фортране — это образования из букв и цифр алфавита. Слова, используемые в Фортране, можно разделить на две категории — ключевые слова, или слова языка, и наименования.

Ключевые слова имеют строго фиксированное начертание и значение. Программист использует ключевые слова только в том значении, в котором они описываются в грамматике Фортрана. Ему не дано право изобретать новые ключевые слова, не определенные в языке. Так, слово IF всегда в программах Фортрана может быть употреблено только в значении ЕСЛИ, слово SIN всегда означает наименование стандартной подпрограммы синуса. В ключевых словах допускаются пробелы: два слова GØ TØ, разделенные пробелом, представляют ключевое слово, которое употребляется в программе лишь в значении ПЕРЕЙТИ К.

Ключевое слово, по существу, является элементом алфавита Фортрана.

Ниже (таблица 1) приводится перечень основных ключевых слов языка, в котором они перечислены в алфавитном порядке.

Кроме слов таблицы 1, будут введены еще две небольшие группы слов языка — слова, являющиеся знаками операторов отношений (гл. III, § 4, таблица 2) и логических операторов (гл. III, § 4, таблица 3), и слова, являющиеся названиями стандартных библиотечных программ (гл. V, § 1, таблицы 9 и 11) [1]).

---

[1]) Приводимые в этой книге слова Фортрана IV не исчерпывают всего перечня ключевых слов этого языка. Такие ключевые слова, как BACKSPACE, REWIND, PRINT, PUNCH, DATA, END FILE и другие, используются в программах Фортрана средней и повышенной сложности для описания особенностей работы с внешними устройствами или особенностей организации данных. Они немного добавляют к пониманию логики языка и поэтому опущены здесь, в описании основ Фортрана. Читателю предстоит познакомиться с ними в руководствах по пользованию конкретными трансляторами.

Таблица 1

На Фортране	По-русски
ASSIGN TØ	ПРИСВОИТЬ
CALL	ВЫЗВАТЬ
CØMMØN	ОБЩИЙ
CØMPLEX	КОМПЛЕКСНЫЙ
CØNTINUE	ПРОДОЛЖАТЬ
DIMENSIØN	РАЗМЕРНОСТЬ
DØ	ВЫПОЛНИТЬ
DØUBLE PRECISIØN	ДВОЙНАЯ ТОЧНОСТЬ
END	КОНЕЦ
EQUIVALENCE	ЭКВИВАЛЕНТНОСТЬ
FØRMAT	ФОРМАТ
FUNCTIØN	ФУНКЦИЯ
GØ TØ	ПЕРЕЙТИ К
IF	ЕСЛИ
INTEGER	ЦЕЛЫЙ
LØGICAL	ЛОГИЧЕСКИЙ
PAUSE	ОСТАНОВ
PRØGRAM	ПРОГРАММА
READ	ЧИТАТЬ
REAL	ДЕЙСТВИТЕЛЬНЫЙ
RETURN	ВЕРНУТЬСЯ
STØP	СТОП
SUBRØUTINE	ПОДПРОГРАММА
WRITE	ПИСАТЬ

Среди главных типов ключевых слов выделим:

слова, указывающие предписания для выполнения действий; таковы, например, DØ (ВЫПОЛНИТЬ), CALL (ВЫЗВАТЬ), READ (ЧИТАТЬ);

слова, которые являются описаниями величин, как, например, INTEGER (ЦЕЛЫЙ), LØGICAL (ЛОГИЧЕСКИЙ), REAL (ДЕЙСТВИТЕЛЬНЫЙ);

слова, определяющие структурные подразделения программ, например SUBRØUTINE (ПОДПРОГРАММА), FUNCTIØN (ФУНКЦИЯ);

слова, описывающие группы величин и их отношения, таковы, например, CØMMØN (ОБЩИЙ), DIMENSIØN (РАЗМЕРНОСТЬ).

Слова, являющиеся комбинациями букв и цифр алфавита, не совпадающие с ключевыми словами, используются в фортрановской программе для присваивания наименований объектам различной природы — величинам, группам величин, индексам, частям программы, программе в целом.

Правила формирования наименований одинаковы для всех этих объектов.

Наименованиями могут быть любые наборы букв и цифр, включающие от одного до шести символов. Обязательное требование: первым элементом должна быть буква. В слове, являющемся наименованием, никакие символы, кроме букв и цифр, не разрешены. В число символов наименования не может входить и пробел.

Вот несколько примеров допустимых наименований:

L	A2	ALPHA
SALARY	I24	B315

и недопустимых:

4SLØN — здесь первый символ — цифра. Можно было бы написать SLØN4;

FØRTRAN — это наименование слишком длинно, число символов превосходит 6;

TØT-SR — дефис, разделяющий две части наименования, делает его недопустимым: дефис не является ни буквой, ни цифрой;

ØLD F — пробел внутри наименования не разрешен, приемлемым было бы ØLDF.

Правила, разрешающие формировать наименования, достаточно свободны и позволяют вложить в наименование содержательный смысл. Например, вместо малоразличительной формулы $I = S + F - T$ для вычисления величины дохода можно, дав наименования величинам

INCØME (ДОХОД), FEE (ГОНОРАР),
SALARY (ЗАРПЛАТА) TAX (НАЛОГ),

написать на Фортране

```
INCØME = SALARY + FEE — TAX
```

§ 2. Арифметические инструкции

Процесс подготовки задачи к решению ее на вычислительной машине состоит в том, что программист разделяет решение задачи на отдельные элементарные этапы и формально описывает каждый этап в виде указаний,

предписаний машине. Такие предписания называются *инструкциями*.

Например, выражение

$$S = X + Y + Z \qquad (2.3)$$

является инструкцией в Фортране. Оно указывает элементарную последовательность действий: сложить величины, имеющие обозначения X, Y и Z, и величине, называемой S, присвоить значение суммы.

Программа в Фортране — это последовательность инструкций. По их возможностям и правилам написания различают несколько типов инструкций. Один из наиболее употребляемых типов инструкций — *арифметические инструкции*. Они указывают машине, какие и в каком порядке арифметические операции следует выполнять. Арифметической инструкцией является и инструкция (2.3). Основной частью арифметической инструкции является *арифметическое выражение*, определяемое как комбинация констант и переменных, соединенных знаками операций и скобками. Оно расположено справа от знака присваивания =. Частным случаем арифметической инструкции является *инструкция присваивания*, в ней арифметическое выражение имеет тривиальный вид единственной величины, константы или переменной.

Подчиняясь арифметической инструкции, машина может выполнять четыре основных действия арифметики: сложение, вычитание, умножение и деление. Сложение и вычитание указываются в инструкциях Фортрана привычным образом, в соответствии с правилами алгебраической орфографии с помощью символов + и —. Так, инструкция (2.3) — это инструкция, по которой складываются три числа. По инструкции

$$A = B + C - D$$

следует сначала сложить величины B и C, а затем из суммы вычесть величину D.

Умножение имеет в Фортране несколько отличающееся от обычного обозначение. Знаком умножения является звездочка *. Выражение X*Y предписывает умножить X на Y. Ни одна из следующих записей

$$X \times Y, \text{ или } X \cdot Y, \text{ или } XY$$

не эквивалентна выражению X*Y. Записи X × Y и X · Y вообще не имеют смысла в Фортране, а XY в лучшем случае могла бы обозначать некоторую величину, отличную и от X и от Y, но не выражать умножения X на Y.

Знаком деления в Фортране выбрана наклонная черта /. Например, выражение X/Y означает предписание разделить величину X на величину Y.

Упражнение 2.1. Записать на Фортране следующие алгебраические выражения:

$$1)\ ax+by-cz; \qquad 2)\ \frac{a}{x}+\frac{c}{z}.$$

К числу основных арифметических операций, которые могут быть выполнены по арифметическим инструкциям Фортрана, относится операция возведения в степень. Знаком этой операции служат две подряд стоящие звездочки **. Например, возведение в куб величины A можно записать выражением A**3.

Выбор наклонной черты в качестве знака деления и двойной звездочки как знака возведения в степень обеспечивает запись на Фортране единой строкой таких выражений, которые записываются в алгебре в виде двух (и более)-этажных дробей с показателями степеней, вынесенными выше уровня строки. Например, алгебраическое выражение

$$\frac{a^3}{b^2}$$

на Фортране записывается в одну строчку

A**3/B**2

Как увидим ниже, на Фортране существуют приемы записи подстрочных индексов в одной строке с индексируемой величиной. Таким образом, все отклонения от записи в одну строку, свойственные математическим формулам, в Фортране устранены. Линейное (одной строкой) представление математических выражений существенно упрощает их запись на любых носителях информации и их автоматическое считывание, не уменьшая при этом привычной содержательности обозначений.

Величины, участвующие в арифметической операции, называются ее *операндами*. Все арифметические операции

имеют по два операнда: в операции сложения операндами являются слагаемые, в операции вычитания — уменьшаемое и вычитаемое, в операции умножения — множители, в операции деления — делимое и делитель, в операции возведения в степень — основание степени и показатель степени.

Порядок выполнения операций в сложном выражении очень важен. Транслятор просматривает фортрановское выражение всегда слева направо. Поэтому в выражении

$$X + Y - Z$$

сначала будут сложены X и Y, а затем сумма их будет уменьшена на Z.

В выражении

$$x + y + z$$

математику было бы безразлично, какое из сложений выполнять первым. Машина же по инструкции

$$S = X + Y + Z$$

обязательно сначала сложит X и Y, а затем к сумме прибавит Z.

При ручном вычислении значения выражения $ax + by$ математик сначала умножает a на x, затем умножает b на y и только затем складывает произведения ax и by: порядок выполнения операций — их *иерархия*, по которой сначала выполняется умножение, а затем сложение, — известен из начальной школы. Фортран полностью заимствует эту иерархию. Сначала выполняются все операции более высокого порядка — умножение и деление, затем действия более низкого порядка — сложение и вычитание. Среди операций одного порядка первой выполняется та, которая в выражении указана левее.

Высшим порядком в иерархии операций обладает возведение в степень. В выражении

```
X+Y*Z**2
```

первой будет выполнена операция возведения в степень Z**2, затем умножения Y на предыдущий результат и, наконец, сложения X с последним результатом.

В соответствии с указанными выше правилами иерархии операций арифметическая инструкция

$$R = A + B/C + D$$

соответствует формуле

$$r = a + \frac{b}{c} + d,$$

но отнюдь не

$$r = \frac{a+b}{c+d}.$$

Фортран заимствует из обычной алгебраической символики скобки, которые могут определять порядок выполнения операций. Так, по формуле

$$y = (a + b)(c + d)$$

сначала выполняются сложения внутри скобок и лишь затем перемножаются суммы. На Фортране и запись, и порядок выполнения операций аналогичны:

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

Как в алгебре, в Фортране каждой открывающей скобке соответствует закрывающая. Запись предыдущей инструкции в виде

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

неверна: отсутствует одна из открывающих скобок.

Чтобы указать желаемый порядок выполнения операций в Фортране, программист должен будет указать иногда скобки там, где обычная алгебраическая нотация обошлась бы без них.

Так, в формуле

$$y = \frac{a+b}{x} \tag{2.4}$$

порядок ясен без скобок. На Фортране эту формулу нельзя переписать в виде

$$Y = A + B/X \tag{2.5}$$

Это ошибка: выражение (2.5) соответствует формуле

$$y = a + \frac{b}{x},$$

которая, очевидно, даст иной результат, чем (2.4).

Вообще говоря, скобки необходимы лишь там, где желаемый порядок операций не соответствует правилам иерархии операций. Но тем не менее скобки могут быть употреблены и там, где они не являются необходимыми. Для этого достаточно иметь такое основание, как наглядность записи. Например, скобки в выражении (A*B) + (C*D) могли бы быть опущены.

Выражение, заключенное в скобки, может вычисляться не одной, а несколькими операциями. Все операции внутри одной пары скобок подчинены тем же правилам иерархии операций.

Выражение, заключенное в скобки, может в свою очередь содержать скобки. В таком случае вычисление выражения начинается с выполнения операций в самых внутренних скобках.

Вот несколько примеров алгебраических формул и соответствующих им фортрановских выражений, все скобки которых необходимы:

$(a+b+c)(x+y+z)$	(A + B + C) * (X + Y + Z)
$\frac{1}{i+j+k}$	1/(I + J + K)
t^{k+2}	T**(K + 2)
$(a+b)(x+y)^{n-1}$	(A + B)*(X + Y)**(N — 1)

Скобки в выражениях Фортрана обязательны всегда для разделения двух рядом стоящих знаков операций. Например, $\frac{x}{-y}$ не может быть записано как X/—Y: здесь рядом стоят знаки двух операций — деления и вычитания. Допустимая запись —

X/(— Y)

Вариант записи —X/Y допустим и без скобок.

Упражнение 2.2. Написать выражения Фортрана, соответствующие алгебраическим выражениям

1) $\frac{a}{bc}$; 2) $\frac{ab}{c}$.

В выражения Фортрана включать только те скобки, которые не могут быть опущены.

§ 3. Величины и формы их представления

В арифметических инструкциях буквенные символы представляют величины, участвующие в операциях. Эти величины могут быть константами или переменными.

Можно определить величину, указав число, которое ей соответствует. Определяемая так величина называется *константой*. Константа имеет постоянное и известное в математическом выражении значение. Так, в выражении, которое вычисляет длину окружности

$$D = 2.0*3.14159*R \qquad (2.6)$$

величины 2.0 и 3.14159 являются константами.

Величина R в инструкции (2.6) является переменной. *Переменная* в Фортране — это величина, которая может принимать, вообще говоря, различные значения в ходе исполнения программы. Переменная получает или меняет свое значение при исполнении только тех инструкций, где эта переменная записана в левой части инструкции. Так как значения переменной могут быть различными в разные моменты времени, изображение переменной не может определяться ее значением. Буквы, употребляемые в арифметических выражениях и инструкциях Фортрана в предыдущем параграфе, служат изображениями переменных.

Инструкция (2.6) может вычислить длину D окружности для каждого из изменяющихся по мере исполнения программы значения переменной R с неизменными значениями констант 2.0 и 3.141592. Буква R является символическим представителем числа, участвующего в выражении инструкции (2.6) в качестве третьего сомножителя. Подразумевается, что до момента исполнения инструкции (2.6) значение величины R было тем или иным способом определено.

При составлении программы для каждой неизвестной или изменяющейся в программе величины выбираются наименования, и эти наименования участвуют в дальнейшем в фортрановских программах вместо чисел, которые они представляют.

Во всех вышеприведенных примерах наименования переменных были буквами. В действительности наименования переменных могут быть образованы по общим правилам формирования наименований в Фортране.

Так, переменные могли бы быть названы DELTA, STØCK, SHØP7.

Фортран допускает рассмотрение *действительных* и *целых* чисел, различая их как по форме записи, так и по способам обработки. Термин *действительное число* в Фортране отличается от принятого в математике понятия действительного числа. В Фортране *действительный* — это только название способа записи чисел. На самом деле машина может оперировать лишь с рациональными числами, имеющими конечное количество знаков в дробной части числа. Действительное число π можно записать в виде рациональных 3.14 или 3.141592. Любое из этих приближенных представлений числа π называется в Фортране действительным числом уже потому, что в обоих выделены целая и дробная части.

Целые числа дробной части не имеют, и десятичная точка поэтому отсутствует в их изображениях.

Действительное число может иметь, в частности, целое значение, но для того чтобы сохранять название действительного числа — форму представления, оно должно иметь десятичную точку и в том случае, когда в ней нет необходимости при немашинной обработке. Так, число 10. или 10.0 есть действительное число «десять», 10 — это целое число «десять».

Если дробная часть действительного числа отсутствует, то нуль в дробной части не обязателен, оба изображения действительного числа, имеющего целое значение, 10. и 10.0 равноправны. Если действительное число является правильной дробью, то нуль целой части можно по желанию писать или опускать, но десятичная точка в обоих случаях обязательна. Так, обе записи 0.25 и .25 одинаково верны.

Форма представления числа определяется программистом. Задавая форму представления чисел, он указывает тем самым форму констант и переменных, которые могут принимать эти значения. Таким образом, константы и переменные также могут быть действительными и целыми.

Форма представления переменной определяется первой буквой ее наименования. Если наименование переменной начинается с букв I, J, K, L, M, N — это целая переменная, во всех остальных случаях переменная будет действительной.

Переменные

IKØN, J, LØS, MIR, NUMBER

целые, а переменные

ALPHA, DELTA, P, SALARY, X, Y

действительные.

Форму представления действительных чисел с обязательной десятичной точкой в Фортране называют *формой* F, а форму представления целых чисел — *формой* I.

Давая переменным содержательные наименования, следует помнить, что первая буква их наименования определяет форму представления переменной. Так, в инструкции, вычисляющей площадь прямоугольника, естественно было бы назвать переменные

AREA	(ПЛОЩАДЬ)
LENGTH	(ДЛИНА)
WIDTH	(ШИРИНА)

Однако если все три переменные должны быть действительными, то для второй из них надо выбрать другое наименование. Действительно, LENGTH по определению является наименованием целой переменной. Добавление любой буквы (кроме I, J, K, L, M, N) впереди слова LENGTH превращает его в наименование действительной переменной. Правда, длина его тогда становится равной семи символам, и для превращения его в допустимое наименование надо опустить одну, например последнюю, букву. Переменные ALENGT или XLENGT действительные. Инструкцию вычисления площади можно записать

AREA = ALENGT * WIDTH

Вообще говоря, переменные и константы одного выражения должны иметь одну и ту же форму представления. Тогда и о самом выражении говорят как о действительном или целом. Если выражение смешанное, т. е. содержит как действительные, так и целые переменные и константы, то его арифметические операции не могут выполняться до преобразования всех величин к одной форме. Поэтому в фортрановском арифметическом

выражении, как правило, не рекомендуется смешивать две формы представления величин. Ниже будут обсуждены исключения из этого правила.

Выражение

$$A + 1$$

смешанное, ибо переменная A действительная, а единица — это целая константа. Следует писать

$$A + 1,$$

тогда выражение становится действительным, поскольку обе его компоненты на этот раз действительные: единица с точкой — действительная константа.

Одно из исключений, разрешающих употребление смешанных выражений, — использование показателей степени. Целые показатели степени могут быть в равной степени как целыми, так и действительными величинами. Так, независимо от формы выражения в нем могут фигурировать

A**2, I**5.0, LIN**3., X**N

Показатель степени в свою очередь может быть константой, переменной и выражением. Формула $y = X^{2k+i}$ на Фортране будет записана

Y = X**(2*K+I)

Поскольку обе формы представления показателей одинаковы и не оказывают влияния на форму выражения, всюду ниже мы будем употреблять целые показатели там, где это возможно. Например, записывая квадратный корень из числа в показательной форме, избежать действительного показателя невозможно. Формулу $y = \sqrt{x} = x^{0.5}$ можно записать в форме фортрановской инструкции, лишь используя действительный показатель 0.5:

Y = X**0.5

Упражнение 2.3. Написать на Фортране следующие формулы, не смешивая форм переменных в одном выражении:

1) $f = \frac{m_1 \cdot m_2}{r^2}$; 2) $V = \frac{4}{3}\pi R^3$;

3) $y = ax^2 + bx + c$; 4) $f = ma$.

Мы рассмотрели два класса чисел, представимых в Фортране, — целые и действительные. Действительные числа в свою очередь имеют две формы представления — *с фиксированной запятой* (*форма* F) и *с плавающей запятой* (*форма* E).

Приведенные выше примеры действительных констант — это действительные числа с фиксированной запятой. Название этой формы представления действительного числа связано с тем, что десятичная точка определена, зафиксирована в самой записи числа.

Для записи очень больших или, наоборот, очень малых чисел часто используют действительные числа в форме с плавающей запятой. В этой форме действительное число состоит из двух множителей: *мантиссы* и числа 10, которое возведено в целую степень, называемую *порядком* изображаемого действительного числа. Так, число 2500000. можно записать, например, как 2.5×10^6. В Фортране эта форма изображается так: сначала мантисса, затем буква E и, наконец, величина порядка. Действительная константа 2500000. на Фортране может иметь вид 2.5E6.

Такое же обозначение используется для очень малых чисел. Так, константу 0.0000025 можно изобразить в форме E:

$$2.5E - 6.$$

Порядок, как видно, может быть положительным и отрицательным, но обязан всегда быть целым. Величина порядка может иметь не более двух десятичных знаков. В том случае, когда величина порядка меньше 10, можно впереди величины порядка писать 0. Для положительных порядков знак + может быть указан или опущен. Таким образом, четыре ниже перечисленных записи

$$2.5E6,\ 2.5E06,\ 2.5E+6,\ 2.5E+06$$

равноправны и будут восприняты фортрановским транслятором как действительное число 2500000.; это число может быть записано в форме E многими способами:

$$2.5E6 = 0.25E7 = 25.E5 = 2500.E3 = 0.0025E9 = \ldots \tag{2.7}$$

Программист вправе выбирать любой из этих способов.

Один из способов записи чисел в форме E выделен особо: мантисса в этой записи имеет вид правильной дроби, у которой первая цифра после десятичной точки значащая. Такая мантисса называется *нормализованной*, и само действительное число, записанное в форме E с нормализованной мантиссой, называется *нормализованным числом*. В строчке (2.7) второй элемент представляет нормализованное число.

Форма E представления действительных чисел названа формой *с плавающей запятой*, поскольку положение запятой (десятичной точки) числа определяется величиной порядка — десятичная точка «плавает» в числе вместе с изменением величины порядка, как это наглядно показывает строчка (2.7).

Вот еще два примера использования формы E, которые убеждают в ее компактности:

$$12\,000\,000\,000. = 0.12E+11$$
$$-0.000\,000\,025 = -0.25E-07$$

Другое важное преимущество изображения действительных чисел в форме с плавающей запятой будет отмечено в следующем параграфе.

Упражнение 2.4. В приведенных ниже примерах объяснить ошибки в написании действительных чисел:

1) 142.61 E+3.
2) 2.0E
3) 1.000.000
4) 345,208
5) 5.1E+012
6) 25E—08

Форма представления констант и переменных в выражении определяет форму выражения. Выражение (IND—KØN) * J — целое. Если результат действительного выражения всегда может быть действительным значением, то для целых выражений подобное утверждение ошибочно: сложение, вычитание и умножение целых величин приводят к целому результату, но результатом деления целых чисел может быть дробь. Тем не менее, в целых выражениях деление не запрещается. Результатом деления в целом выражении считают целую часть частного — результат не округляется. При I = 17 и и J = 23 два выражения I/8 и J/8 дадут один и тот же результат, равный 2, хотя первое частное равно 2.125, а второе — 2.875.

Следует внимательно относиться к этой условности деления в целых выражениях. Например, выражения

X**(1/2) и X**0.5

дают разный результат, ибо в первом из них показатель равен нулю, как целая часть правильной дроби. Действительные выражения

(A*B)/C и (A/C)*B

приводят к одному результату. Для целых выражений

(I*K)/J и (I/J)*K

это не так: пусть

I = 5, J = 2, K = 3,

тогда значение первого выражения есть (5*3)/2 = 7, а второго (5/2)*3 = 6.

Итак, *арифметическая инструкция* Фортрана формально может быть определена как запись, состоящая из наименования переменной, за которой сначала следует знак равенства, а затем арифметическое выражение в любой форме. По арифметической инструкции машина сначала вычисляет значение арифметического выражения, затем присваивает это значение переменной, стоящей слева от знака равенства.

Если переменная A = 15.5, то переменная B после выполнения инструкции

B=A + 1.

получит значение 16.5.

Выше говорилось о том, что существуют различные способы определения значений переменных. Арифметическая инструкция — один из них. Так, инструкция

A = —25.41

указывает, что после ее выполнения переменная A будет иметь значение —25.41 независимо от того, какое значение она имела до этого.

Упражнение 2.5. Определить значение переменной X после выполнения инструкций

```
A = 3.0
B = 1.0
Y = A**2 — B**2
X = (Y — A)*(— 2.)
```

Два этапа арифметической операции — вычисление арифметического выражения и присвоение вычисленного значения переменной левой части инструкции — независимы: выражение справа от знака равенства вычисляется в соответствии с формой входящих в него переменных и независимо от формы представления переменной, стоящей слева. Арифметическое выражение в правой части инструкции и переменная в левой могут иметь разные формы представления.

В допустимой Фортраном инструкции

$$L = A*X - B*Y$$

сначала вычисляется действительное выражение в правой части инструкции, и на втором этапе полученное действительное значение преобразуется в целое число, чтобы быть присвоенным целой переменной L. Преобразование состоит в отбрасывании дробной части значения арифметического выражения.

В равной степени допустима инструкция

$$X = I - J*K$$

Здесь сначала вычисляется целое выражение, а затем величина результата — целое число — после преобразования в действительную форму присваивается действительной переменной.

Начальное значение единицы может быть присвоено переменной A с равным успехом двумя инструкциями

$$A = 1. \text{ и } A = 1$$

Правда, во втором случае машине потребуется дополнительное время на преобразование единицы в форму действительного числа. Поэтому при программировании рекомендуется избегать инструкций типа A = I, если это не вызывается необходимостью.

Упражнение 2.6. Вычислить результаты двух арифметических инструкций

$$X = 5./2. \text{ и } K = 5/2$$

Независимость правой и левой частей арифметической инструкции приводит к тому, что переменная левой части может присутствовать в правой. Так, инструкция

$$I = I + 1 \qquad (2.8)$$

допустима и, более того, весьма употребительна в Фортране.

Инструкции типа (2.8) часто употребляются, когда надо считать, сколько раз выполняется предшествующая ей инструкция или группа инструкций программы. Переменную I в таком случае называют *счетчиком*. К счетчику автоматически прибавляется единица, как только выполнена отмеченная инструкция или группа инструкций.

Наименования переменных использовались выше для обозначения отдельных величин. В Фортране можно такие же наименования давать группам величин — *массивам* чисел. Если одно наименование относится к массиву чисел, нужно иметь способ находить в массиве любой его элемент. Для этой цели используются индексы.

Пусть массив из десяти чисел

21, 27, 18, 34, 95, 64, 70, 22, 20, 46

назван LIST, и в этом массиве необходимо выделить для использования в вычислениях его пятый элемент. Обычная математическая символика позволила бы указать пятый элемент в виде $LIST_5$. Цифра, написанная внизу справа от названия массива, называется *индексом* пятого элемента, она указывает место разыскиваемого элемента в массиве. Выше говорилось о необходимости линейной записи фортрановских выражений, поэтому Фортран заимствует из математики символику индексов с некоторым изменением: индекс 5 записывается в виде целой константы 5 в скобках непосредственно после наименования массива.

Таким образом, LIST(5) служит наименованием величины 95 из массива LIST.

Индекс может быть целой константой, целой переменной или целым выражением: LIST(4) означает четвертый элемент массива LIST; LIST(I) означает его I-й элемент, где I — целая переменная, значение которой предварительно определяется; LIST(I + 2) — элемент, номер которого вычисляется целым выражением I + 2.

Индекс элемента массива — это его номер в упорядоченном списке элементов массива. Он служит для

ссылки на фиксированный элемент массива в выражениях и инструкциях. Так, выражение

LIST (3)*KØL

вычисляется как произведение третьего элемента массива LIST на переменную KØL.

Наименование массива можно рассматривать как наименование переменной, значениями которой служат элементы массива и только они. LIST (I) — это целая переменная, принимающая значения

21, 27, 18, 34, 64, 70, 22, 20, 46.

Такую переменную называют *индексированной переменной.*

Таким образом, массив чисел может всегда быть поставлен в соответствие с некоторой индексированной переменной, которая может быть целой, как в предыдущем номере, или действительной. Например, массив X пяти действительных чисел

1.5, 2.1, 3.2, 8.0 и 7.3

порождает индексированную действительную переменную X (I), где I принимает соответственно целые значения 1, 2, 3, 4, 5. Наименование массива поэтому определяется формой представления его элементов: для массивов, состоящих из целых чисел, выбирается наименование, начинающееся с букв I, J, K, L, M, N, как для целых переменных. Остальные буквы могут быть начальными в наименованиях массивов, состоящих из действительных чисел.

Индексированные переменные могут быть как действительными, так и целыми, но индексы всегда целые. Более того, являясь номерами элементов в упорядоченных списках, индексы не могут быть отрицательными или принимать нулевое значение. В Фортране не могут появиться индексированная переменная X (T) или элемент массива X (—3).

И еще одно ограничение в использовании индексов — сам индекс не может быть индексированной величиной. Выражение X (I (3)) в Фортране не имеет смысла.

Количество элементов массива заранее определено. Тем самым определены границы изменения индекса. В приводимом выше примере массива LIST индекс I

не может ни превосходить 10, ни быть равным нулю. Индекс, как об этом уже говорилось, может вычисляться целым арифметическим выражением. Однако, вид выражений, вычисляющих индекс, не может быть произвольным. Допускаются лишь следующие виды целых выражений для вычисления индекса:

1) переменная ± константа (но не наоборот!), например:

A (I + 1) или A (I — 1)

но индексированная переменная A (5—I) недопустима;

2) положительная константа, умноженная на переменную, например:

A (2 * I)

но не A (I*2)

3) положительная константа, умноженная на переменную ± константа, например:

A (2*I + 1) или A (2*I — 1)

Другой порядок операций при формировании значения индекса недопустим.

Упражнение 2.7. Объяснить ошибки в написании приведенных ниже индексированных переменных:

1) DLINA (— 2*J)	2) ALPHA (0*K)
3) FI (SLØN)	4) A (— IND)
5) DELTA (J — 2.)	6) X (2 + I)
7) X (J + K)	8) KL (H + 5)

Индексированная переменная с одним индексом описывает массив одного измерения, или *линейный* массив. Это измерение — его длина, или количество элементов массива. В математической практике часто используются *двухразмерные* массивы. Таковы, например, матрицы. Одно измерение матрицы — число столбцов, второе — число элементов в столбце, или число строк матрицы.

Фортран располагает возможностями для описания матриц или других двухразмерных групп чисел. Эти возможности реализуются двойным индексированием переменных. Переменная, являющаяся элементом такого массива, имеет вслед за своим наименованием заключенную в скобки пару индексов. Индексы разделены запятой.

Массив с двумя индексами рассматривается как группа линейных массивов: первый индекс элемента указывает его позицию внутри линейного массива, а второй — номер линейного массива в группе.

Линейные массивы, составляющие двухразмерный массив, называют *подмассивами* этого двухразмерного массива.

Так, прямоугольная матрица A размером 10×20 есть 20 подмассивов по десяти чисел, и, например, A (6,5) есть шестой элемент в пятом подмассиве. Комбинация двух индексов однозначно определяет положение элемента в массиве. Любой из пары индексов может быть целой константой, целой переменной или целым выражением, построенным в соответствии с рассмотренными выше правилами формирования индексных выражений.

Переменные с двумя индексами

X(I, J), BLØCK(2*J — 3, 3*J + 2),
LIST (N + 1, M + 1)

описывают двухразмерные массивы;

переменные

CØLØN(K, 4), NØM(3*K+5, 8)

выделяют фиксированные подмассивы;

переменные

A (3, INDEX), LIGN (4, 2*J)

представляют выделенные «строчки» массива;

переменные

VAL (3,7), Y (1,1)

суть элементы двухразмерных массивов.

Двухразмерный массив легко представим в виде обычного, линейного массива.

Элементы прямоугольной матрицы, состоящей из двух строк и трех столбцов, записаны в запоминающем устройстве машины в одну строку в таком порядке, как на рис. 2.

Матрицы упорядочиваются по возрастанию второго индекса — номера столбца, а внутри группы элементов одного столбца — по возрастанию первого индекса, который соответствует номеру строки. В линейном массиве,

построенном из элементов двухразмерного массива по принципу, изображенному на рис. 2, первый индекс растет быстрее второго.

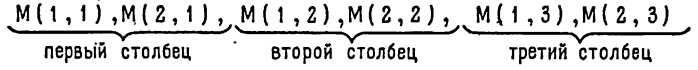

Рис. 2.

В некоторых из своих версий Фортран допускает запись и обработку трехразмерных массивов. Переменная, описывающая текущий элемент такого массива, должна иметь, очевидно, три индекса.

Фортран разрешает представление комплексных чисел в виде упорядоченной пары двух действительных чисел, окруженных скобками и разделенных запятой. Комплексная константа $2.5 + 4.2i$ в Фортране представляется как (2.5, 4.2). Комплексные переменные должны иметь наименования, подчиняющиеся общим правилам формирования наименований переменных в Фортране. Фортран допускает все действия арифметики комплексных чисел. Если, например, известно, что комплексные переменные C1, C2 и C3 имеют соответственно значения

$$1.5 + 2.0\ i,\ 0. + 2.5\ i \text{ и } 2.—3.5\ i,$$

то инструкция

```
C = C1 + C2 — C3**2
```

присвоит величине C комплексное значение (9.75, 18.5). Действительно,

$$C3**2 = (— 8.25,\ — 14),$$

и потому вещественной частью результата будет

$$1.5 + 0. —(—8.25) = 9.75,$$

а мнимой

$$2.0 + 2.5 — (—14) = 18.5.$$

В этой книге арифметика комплексных чисел не рассматривается. Читатель сможет удовлетворить свою любознательность, знакомясь с описанием возможностей конкретных фортрановских трансляторов, допускающих представление комплексных чисел.

Определение формы представления переменной с помощью первой буквы ее наименования называется *неявным определением типа* переменной. Столь простое определение типа переменной ограничивает иногда возможности содержательного наименования переменных. Так, для того чтобы использовать переменную с наименованием MASSE в арифметических выражениях для обозначения действительного значения физической величины — массы, необходимо это наименование видоизменить, например: AMASSE или XMASSE.

Для разрешения трудностей такого рода в Фортране можно пользоваться инструкциями декларации типа величин. *Инструкция декларации типа* определяет форму представления величины независимо от первой буквы ее наименования. В отличие от арифметических инструкций, инструкции декларации типа величины не исполняются вычислительной машиной. Они вставляются в программу для того, чтобы сообщить дополнительные сведения об описываемых величинах.

Существует пять инструкций декларации типа величины INTEGER (ЦЕЛЫЙ), REAL (ДЕЙСТВИТЕЛЬНЫЙ), CØMPLEX (КОМПЛЕКСНЫЙ), DØUBLE PRECISIØN [1]) (ДВОЙНАЯ ТОЧНОСТЬ) и LØGICAL [2]) (ЛОГИЧЕСКИЙ). Инструкция декларации типа состоит из одного из этих ключевых слов и списка переменных, разделенных запятыми. По инструкции декларации типа все величины, указанные в списке, принимают форму, предписываемую инструкцией.

Например, инструкция

INTEGER VALUE, X, Z15, PØINT

преобразует величины VALUE, X, Z15, PØINT в форму целых чисел, и всюду в дальнейшем в программе эти величины будут использоваться как целые. Наоборот, инструкция

REAL MINI, MAXI

определяет величины MINI и MAXI как действительные, невзирая на первые буквы их наименований.

---

[1]) Понятие двойной точности определено в § 4 настоящей главы.

[2]) Логические величины рассматриваются в § 4 главы III.

Определение формы представления переменной с помощью инструкции декларации типа величины называется *явным определением типа* переменной.

Инструкции декларации типа величины, если они имеются в программе, должны быть самыми первыми ее инструкциями.

Упражнение 2.8. Пусть первыми инструкциями программы являются

```
REAL    MANTIS, LIP
INTEGER BLØCK,  STEP
```

Определить форму представления следующих переменных, участвующих в этой программе:

```
ALPHA   LEV   MANTIS  SLØN   TRIØL
BLØCK   LIP   NUMBER  STEP
```

§ 4. Диапазон и точность чисел

Подобно счетам, имеющим фиксированное количество проволочек с костяшками, подобно арифмометру, имеющему фиксированное количество разрядов в регистре и сумматоре, электронная вычислительная машина может записывать и обрабатывать числа, заключенные в некоторый интервал. Для каждой машины границы этого интервала различны. На машинах серии IBM-360 наибольшее допустимое действительное число равняется примерно 10^{+75}, а наименьшее положительное число, разрешенное в машине — это 10^{-75}. Для БЭСМ-6 и Минск-22 эти числа равны соответственно 10^{+19} и 10^{-19}.

Наибольшее целое число, представимое в IBM-360, это $2^{+31}-1$, т. е. 217448347, в БЭСМ-6 — $(+2^{39}-1) = 549755813887$, в «Минск-22» — $(+2^{36}-1) = 6871947635$.

Границы диапазона известны транслятору, который может указать ошибку в программе, если при трансляции обнаружит число, выходящее из диапазона представимых в данной машине чисел.

Однако, границы диапазона представимых чисел являются характеристиками машины и не определяют количества значащих цифр в изображении числа. Это количество есть характеристика языка. Большая часть фортрановских программ оперирует с числами, имеющими семь значащих цифр. Так, ближайшими к числам π и e являются разрешенные в Фортране числа 3.141593

и 2.718282 независимо от машины, на которой эти числа обрабатываются. В силу этого ограничения действительные числа, имеющие более семи цифр в целой части числа, не могут быть изображены в форме F. Число 987654321.01 можно изобразить в форме E, но не точнее, чем 0.9876543E + 09 = 98754300.0, так как и в форме E действительные числа могут содержать не более семи значащих цифр. Число —123.456789 в форме F теряет две свои последние значащие цифры и округляется до —123.4568.

В большинстве практических задач точность, достигаемая семью значащими цифрами, оказывается достаточной. Тем не менее нередко встречаются задачи, требующие повышенной точности. Для изображения действительных чисел повышенной точности существует еще одна форма представления чисел — форма D. В отличие от формы E (представления с обычной точностью), форма D называется представлением с *двойной точностью*. Форма D есть представление действительного числа в форме с плавающей запятой, когда мантисса может иметь до 16 [1]) значащих цифр. Признаком числа с удвоенной точностью служит либо буква D перед величиной порядка, либо предшествующая числу декларация типа величины DØUBLE PRECISIØN.

Действительная константа 45647740891509.1334 в форме D может иметь вид 4564774089150913.D—02, а число *e* можно было бы в этой форме записать 2.718281828459045D0. Все правила использования чисел в форме D в фортрановских инструкциях те же самые, что и для чисел с плавающей запятой с обычной точностью (форма E).

§ 5. Бланк Фортрана

Длина инструкции в Фортране зависит от сложности арифметического выражения. Она ограничивается форматом *бланка*, на котором должны быть записаны инструкции Фортрана. Впрочем, использование бланков для написания фортрановских программ — требова-

[1]) Числа 7 и 16 — максимально допустимые количества значащих цифр в представлении действительных чисел для обычной и двойной точности — могут различаться в различных трансляторах. Эти числа указываются в описании каждого конкретного транслятора.

ние не обязательное. Тем не менее рекомендуется программы Фортрана писать на специальных бланках. Это существенно облегчает чтение программы и ее перфорирование.

Бланк Фортрана изображен на рис. 3. Для инструкции, как правило, отводится одна строка бланка. В строке пронумеровано 72 позиции, из них 66 — с 7-й по 72-ю включительно — отведены для записи инструкции. В одной позиции строки может быть указан только один символ. Изображение инструкции

$$X = A + B - C$$

на бланке приведено на рис. 4. Пробелы между символами в Фортране значения не имеют, и та же инструкция может быть записана на бланке по-разному (рис. 5). Пробелы включаются в число символов инструкции. Длина инструкции, изображенной на рис. 4, равна 7 символам, а три инструкции рис. 5 имеют соответственно длину 13, 9 и 15 символов.

Пробелы не допускаются между цифрами одного числа и между символами одного наименования.

На бланке Фортрана можно записывать инструкции, длина которых превосходит 66 символов — длину отводимой для инструкции строки бланка. В этом случае разрешен перенос неразместившейся в первой строке части выражения на следующую строку. Правила переноса несколько отличаются от принятых в математике: последний символ первой строки (даже если он является знаком операции) не повторяется в следующей строке. Возможен, например, такой перенос алгебраической формулы:

$$y = (ax^2 + bx + c) - 2b^2c^2 + \\ + cx^2 + ax + b.$$

Соответствующая ей арифметическая инструкция может быть перенесена таким образом:

```
Y = (A *X**2+B*X+C)—2.*B**2*C**2
*  + C*X**2+A*X+B                    (2.9)
```

Но если при переносе длинной алгебраической формулы рекомендуемыми местами разрыва строк являются знаки операций сложения или вычитания, то в Фортране перенос может быть сделан в любом месте строки.

ФОРТРАН

Задача

Программист | Дата | Страница | Идентификация (73 … 80)

С | Номер инструкции (1 … 5) | Продолж. (6) | ИНСТРУКЦИИ ФОРТРАНА (7 … 72)

Рис. 3.

С | Номер инструкции (1 … 5) | Продолж. (6) | (7 … 20)

$X_1 = A_1 + B_1 - C_1$

Рис. 4.

С | Номер инструкции (1 … 5) | Продолж. (6) | (7 … 20)

$X_1 = A_1 + B_1 - C_1$

$X_1 = A_1 + B_1 - C_1$

$X_1 = A_1 + B_1 - C_1$

Рис. 5.

Например, инструкция (2.9) могла бы быть перенесена еще и таким образом:

```
      Y = A*X**2 + B*X + C - 2.*B**2*
     *    C**2 + C*X**2 + A*X + B
```

Признаком того, что вторая строка является продолжением первой, служит любой, кроме нуля и пробела, символ в шестой позиции второй строки. На рис. 6 показан перенос длинной арифметической инструкции на бланке Фортрана. Здесь символом продолжения выбрана звездочка.

Инструкция может быть столь длинной, что потребует переноса в третью, четвертую и т. д. строки. В Фортране разрешены инструкции, которые могут быть размещены не более чем в 19 строках. Каждая из строк-продолжений обязательно должна иметь ненулевой символ в шестой позиции, которую иногда называют *позицией продолжения*. Ограничение длины инструкции $19 \times 66 = 1254$ символами позволяет записать практически любые инструкции, тем более, что одна длинная инструкция всегда может быть записана как две более короткие. Например, инструкция рис. 6 может быть переписана введением новых промежуточных переменных (рис. 7).

Длинная строка почти всегда проигрывает в наглядности, поэтому бывает целесообразным делать перенос в следующую строку, не заполнив все 66 символов строки.

Ту же инструкцию рис. 6 можно переписать, и не вводя новых переменных (рис. 8). Здесь отдельной строкой выписаны слагаемые с четвертыми степенями переменных, отдельно — слагаемые с выделенными множителями третьей степени и, наконец, три отдельные строки, в каждой из которых по одному слагаемому. Позиции продолжения здесь заполнены буквой A.

Бланк Фортрана предназначен главным образом для удобства перфорирования программы. Одна строка бланка соответствует одной 80-колонной перфокарте (рисунок перфокарты описан в § 1 гл. IV), при этом каждая позиция бланка соответствует одной колонке перфокарты.

Последние восемь позиций строки бланка (или, что то же самое, последние восемь колонок перфокарты) предназначены для наименования программы. Наименование программы, во-первых, одинаково для всех

строк бланка одной программы и, во-вторых, не является частью инструкции. Поэтому последние восемь позиций общие для всех строк вынесены в шапку бланка выше программного поля (рис. 3). Значение первых пяти позиций бланка объясняется в § 1 главы III.

На бланке выделена пунктиром первая колонка. Буква С, стоящая в начале строки, в первой ее колонке, указывает, что вся строка занята *комментарием*. Комментарий не воспринимается транслятором, а поэтому может содержать любые символы Фортрана, любой текст, поясняющий программу и предназначенный для восприятия этих пояснений человеком: текст комментария перфорируется на картах, занимает определенное место в памяти, выводится на печать вместе с текстом программы, но никак не влияет на программу. Этот текст можно разместить в любом месте программы. Комментарий может объяснить назначение программы или любого ее участка, детально описывать переменные, рассказывать о форме предполагаемых результатов. Однако, несмотря на полную свободу, с которой программист выражает комментариями свои мысли при написании программы, рекомендуется тем не менее писать комментарии по существу задачи. Длина одного комментария не должна превосходить длины 72-символьной строки.

Примеры комментариев показаны на рис. 9. Рисунок изображает фрагмент программы, вычисляющей корни квадратного уравнения. Комментарии этой программы переводятся на русский язык так:

```
C ВЫЧИСЛЕНИЕ КОРНЕЙ КВАДРАТНОГО
  УРАВНЕНИЯ
C С КОЭФФИЦИЕНТАМИ A, B, C
C ВЫЧИСЛЕНИЕ ДИСКРИМИНАНТА
C ВЫЧИСЛЕНИЕ КОРНЕЙ
C СЛУЧАЙ ОДНОГО ДВОЙНОГО ДЕЙСТВИ-
  ТЕЛЬНОГО КОРНЯ
C СЛУЧАЙ ДВУХ РАЗЛИЧНЫХ ДЕЙСТВИ-
  ТЕЛЬНЫХ КОРНЕЙ
C СЛУЧАЙ ДВУХ КОМПЛЕКСНЫХ КОРНЕЙ
```

Последний комментарий «*****», очевидно, не требует перевода на русский язык. Впрочем, на месте этого комментария, который просто отмечает конец программы,

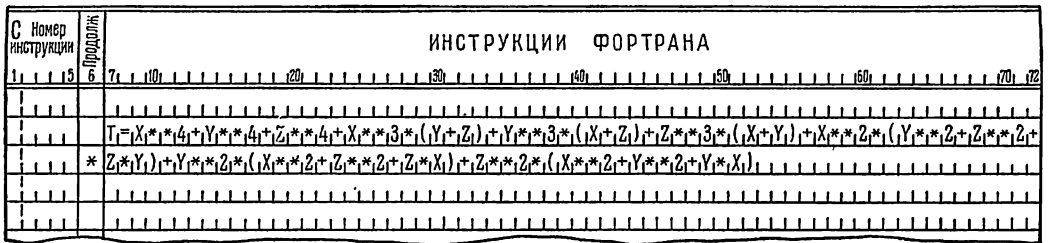

Рис. 6.

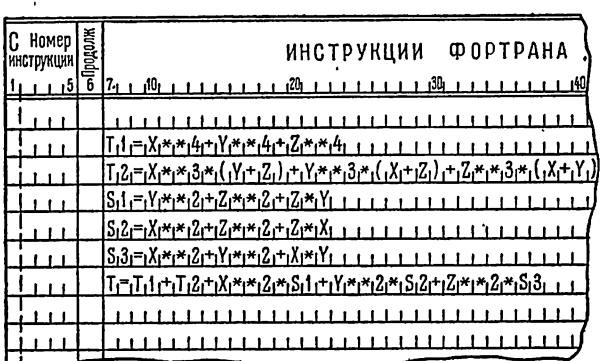

Рис. 7.

Рис. 8.

можно было бы указать любой другой текст или вообще ничего не указывать.

Этот пример показывает, насколько полезными могут быть комментарии ... для программистов, владеющих английским языком.

Вновь разрабатываемые трансляторы с Фортрана для советских вычислительных машин могут предусматривать

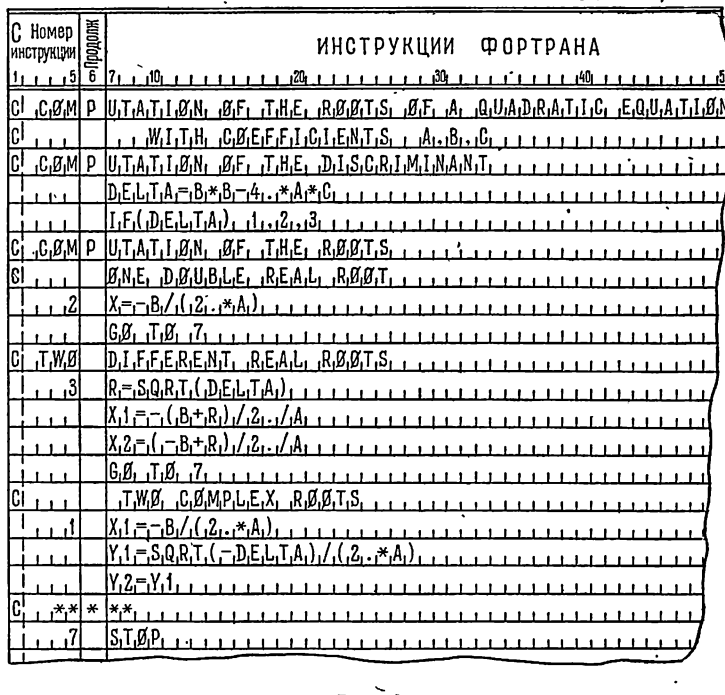

Рис. 9.

комментарии на русском языке, если печатающее устройство имеет русский алфавит (кроме латинского).

Контрольные упражнения к главе II

2.1К. Вычислить значение X как результат исполнения каждой из приведенных ниже инструкций, при условии, что

A = 4.0 B = 1.0 C = 2.0 D = 5.0

1) X = A/D + C*B
2) X = A + C — 3.0
3) X = C**2 — B**2
4) X = A + B — C*2
5) X = X + D — C

полагая, что до выполнения инструкции 5) X имел значение, определенное в инструкции 4)

6) X = A*B*C
7) X = A*D/B + C
8) X = — B
9) X = 2.*A — 3.*C**2
10) X = C*B/C

2.2К. Написать инструкции для вычисления действительных корней X1 и X2 квадратного уравнения $AX^2 + BX + C$, используя для вычисления квадратного корня прием возведения в степень 0.5.

2.3К. Объяснить ошибки в перечисленных ниже арифметических инструкциях

1) NUMBER = VALUE — QUANTITY
2) X = Y**2 — 2,0*Y
3) A = X*(T + 2P)
4) X = (A**2) + B**2)*A + B
5) 2.718 = E
6) T + 1. = (X*Y)/(X**2 + Y**2)

2.4К. Формула потенциальной энергии в консервативной системе (из механики) имеет вид

$$U=\frac{1}{2}(k_{11}q_1^2+2k_{12}q_1q_2+k_{22}q_2^2),$$

где

$$k_{11} = k_1 + k,$$
$$k_{22} = k_2 + k,$$
$$k_{12} = -k.$$

Написать программу вычисления величины U, не смешивая форм представления переменных в арифметическом выражении.

2.5К. Каждую из трех нижеследующих арифметических инструкций, состоящих из нескольких арифметических операторов, записать в виде последовательности инструкций, состоящих из одного оператора по образцу

$$Y = A*B**(J + 2) \rightarrow \begin{cases} I = J + 2 \\ X = B**I \\ Y = A*X \end{cases}$$

1) Y = A*X**2 + B*X + C
2) Y = (A + B)**(K*(J — 1))
3) Y = A/B/C + D/A

ГЛАВА III

ИНСТРУКЦИИ УПРАВЛЕНИЯ

В предыдущей главе мы познакомились с инструкциями, из которых могут составляться простейшие программы или, точнее, элементарные части программ Фортрана.

Реальные программы представляют собой обычно нечто более сложное, чем простая последовательность инструкций. Кроме элементарных последовательностей инструкций, в реальных программах допускаются разветвления, которые позволяют машине принять решение о дальнейшем пути вычислений в зависимости от значения ранее вычисленной величины.

Для этой цели Фортран имеет набор инструкций передач управления, которые позволяют конструировать разветвленные программы.

В этой главе детально обсуждаются важнейшие из фортрановских инструкций передач управления, а также инструкции останова и окончания.

§ 1. Условная и безусловная передачи управления

Обычный порядок выполнения машиной последовательности инструкций — это порядок, в котором они записаны. Естественно назвать такой порядок *последовательным*. Из двух подряд записанных инструкций

```
1 X = A*B — 2.*C
2 Y = Y + X
```

первой будет выполнена полностью инструкция номер 1, а затем инструкция номер 2.

В этой главе вводятся инструкции передач управления, которые позволяют программисту менять последовательный порядок выполнения инструкций. Инструк-

ция *передачи управления* по определению является инструкцией, заставляющей машину выполнять не ту инструкцию, которая непосредственно следует за только что выполненной, а некоторую другую, определяемую инструкцией передачи управления.

Правила написания арифметических инструкций, как было видно в предыдущей главе, допускают большую свободу выражений: в известной степени произвольными могут быть и длина инструкций, и набор операций, и типы переменных. Правила написания инструкций передач управления в большинстве своем более строги: существенное значение могут иметь расположение, тип и количество символов, пунктуация и другие компоненты инструкции.

Чтобы с помощью инструкции передачи управления заставить машину выполнять инструкцию, не следующую непосредственно за только что выполненной, необходимо отмечать инструкции некоторыми указателями. В Фортране такими указателями являются *номера инструкций*.

В Фортрановской программе каждой инструкции можно присвоить номер. Программист может выбирать эти номера произвольно. Размещаются они слева от инструкций. В примере

```
25 X = A*B — 2.*C
 2 Y = Y + X
```

первой выполненной инструкцией будет инструкция номер 25, следом за ней исполняется инструкция номер 2.

Правила нумерации инструкций просты:

— инструкции можно присвоить только один номер;

— порядок, в котором номера присваиваются инструкциям, совершенно произволен, существенным и очевидным требованием является лишь то, что никакие две инструкции в одной программе не могут иметь один и тот же номер.

Еще один пример:

```
   1 X = 3.14159
   5 C = 1.18
1020 A = 2. + C*2.
   3 Y = (A — X)**2 + (A + X)**2
```

(3.1)

Машина выполнит эти инструкции в том порядке, в котором они записаны:

1, 5, 1020, 3.

Хотя принципиально каждой инструкции программы может быть присвоен номер, это не является обязательным правилом: номер может быть и опущен. На практике номерами снабжаются только те инструкции, на которые ссылаются инструкции передач управления. Номера инструкций — это целые положительные числа, выбираемые программистом произвольно. Большинство

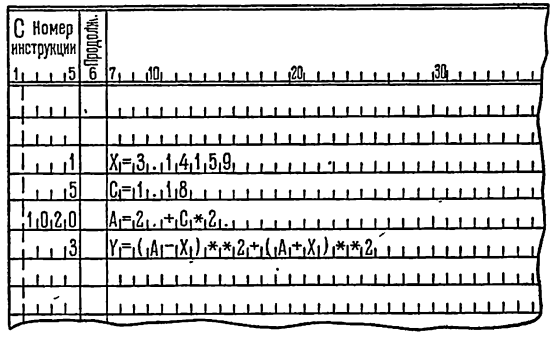

Рис. 10.

трансляторов с Фортрана вводят условность, ограничивающую произвол в выборе номеров пятью десятичными цифрами. В этом случае номером инструкции может быть любое целое число от 1 до 99999.

На фортрановском бланке для пяти десятичных цифр номера инструкции используются пять первых позиций строки бланка. Рис. 10 изображает группу инструкций (3.1), записанную на бланке.

Номера инструкций играют роль указателей, на которые могут ссылаться инструкции передач управления. Инструкция передачи управления указывает машине, какая инструкция должна следовать за только что выполненной; указание это осуществляется с помощью номеров, присвоенных инструкциям.

Если из нескольких возможных продолжений последовательности инструкций следует выбрать одно, инструкция передачи управления делает этот выбор в за-

висимости от элементарных условий. Таким элементарным условием может быть проверка: является ли проверяемое число положительным, отрицательным или нулем. В этом параграфе рассказано, как элементарные условия могут быть использованы для изменения порядка исполнения инструкций.

Одна из наиболее используемых инструкций передач управления — инструкция IF. Эта инструкция обязательно включает ключевое слово IF, за которым следует сначала пара скобок, содержащая арифметическое выражение, а затем три номера инструкций, разделяемых запятыми. С точки зрения этого формального определения инструкция

IF (A*B — X) 5, 10, 15

является допустимой.

По инструкции IF вычислительная машина должна выполнить такую последовательность операций:

1. Прежде всего вычислить значение арифметического выражения, содержащегося в скобках инструкции IF.

2. а) Если вычисленное значение отрицательно, то следующей выполнять инструкцию, номер которой указан первым среди трех номеров, названных в инструкции IF;

б) если вычисленное значение равно нулю, выполнить инструкцию, номер которой указан вторым;

в) если вычисленное значение положительно, выполнить инструкцию, номер которой указан третьим.

Так, если при исполнении инструкции

IF (A*B — X) 5, 10, 15

окажется, что $A*B-X<0$., то следующей будет исполнена инструкция номер 5;

при $A*B-X=0$. следующей исполняется инструкция номер 10;

в случае $A*B-X>0$. выполняется инструкция номер 15.

Запомнить, каким значениям арифметического выражения соответствуют три номера инструкций, можно с помощью мнемонического правила: эти три номера расположены в инструкции IF в том же порядке, в каком соответствующие области — область отрицательных

чисел, нуль и область положительных чисел — расположены на числовой оси.

Инструкция IF реализует в программе так называемый *условный переход*.

Следует еще раз напомнить, что форма инструкции IF, являющаяся ее синтаксическим определением, должна строго соблюдаться. Например, запятые между номерами инструкций обязательны: они четко определяют, где кончается один номер инструкции и где начинается следующий. Но запятая не допустима ни перед первым номером, ни после последнего. Содержащееся в скобках арифметическое выражение может иметь любую допустимую конструкцию и длину, в частности оно может в свою очередь содержать скобки. С другой стороны, в скобках инструкции IF может стоять переменная как тривиальный частный случай арифметического выражения.

Хотя в инструкции IF обязательно указывают три номера инструкции, тем не менее в случае необходимости IF способна осуществить условный переход и по двум направлениям: для этого надо, чтобы два номера из трех были одинаковыми.

Например, инструкция

IF (A — B) 20, 20, 10

обязывает выполнять следующей инструкцию номер 20 (или, применяя чаще употребляемое программистами выражение, *передает управление* инструкции 20) в случае $A \leqslant B$ и инструкцию номер 10 при $A > B$.

Аналогично инструкция

IF (A — B) 20, 10, 20

передает управление инструкции 20 при $A \neq B$ и инструкции 10 в случае равенства A и B.

Инструкция IF, как всякая другая инструкция Фортрана, может иметь номер, стоящий, как всегда, слева от нее и связывающий ее с другими инструкциями программы. Правда, на номер инструкции IF, очевидно, накладывается еще одно дополнительное условие — этот номер не может совпадать ни с одним из трех номеров инструкций, составляющих эту инструкцию IF. В самом деле, инструкция

17 IF (A**2 — B**2) 15, 16, 17

привела бы при $A^2 > B^2$ к бесконечному повторению — «зацикливанию» — инструкции номер 17.

Одно из наиболее частых употреблений инструкции IF — выяснить: вычисляемая величина меньше, равна или больше некоторой другой заданной величины. Для этого содержащееся в скобках арифметическое выражение записывают в виде разности вычисляемой и заданной величин, которая принимала бы соответственно отрицательное, нулевое и положительное значения.

Инструкция

```
IF (A — 60.) 1, 2, 3
```

сравнивает с константой 60. величину A. Разумеется, инструкции, сравнивающие две переменные, как, например,

```
IF (X — Y) 1, 2, 3
```

столь же допустимы и в равной мере употребительны.

Итак, резюме первого знакомства с инструкцией IF: эта инструкция заставляет машину вычислить арифметическое выражение, как это сделала бы соответствующая арифметическая инструкция, а затем выбрать, какой из частей программы следует передать управление в зависимости от значения вычисленного выражения.

Упражнение 3.1. Рассмотрим три переменные X, Y и Z, которые считаются предварительно вычисленными. Написать последовательность инструкций IF, по которой управление будет передано инструкции номер 60, если все три величины равны константе 10.0. Если хотя бы одно из трех чисел X, Y или Z отличается от 10.0, передать управление инструкции номер 70.

Важно заметить, что величина арифметического выражения, вычисленная инструкцией IF, не сохраняется, она не присваивается никакой из переменных.

Например, если есть необходимость сохранить для дальнейших вычислений величину X**2—Y**2, которая служила бы для определения условного перехода, то вместо одной инструкции

```
IF (X**2 — Y**2) 10, 20, 30          (3.2)
```

следует писать две:

```
A = X**2 — Y**2
IF (X**2 — Y**2) 10, 20, 30          (3.3)
```

или, короче,

```
A = X**2 — Y**2
IF (A) 10, 20, 30
```

Две инструкции (3.3) обеспечивают тот же результат, что одна инструкция (3.2), но во втором случае величина A будет сохранена для дальнейших вычислений.

Таким образом, прежде чем записывать сложное арифметическое выражение в скобках инструкции IF, следует оценить, не понадобится ли величина этого выражения в дальнейшем. Такая оценка помогает избежать ненужных потерь времени машинного счета и ручного перфорирования.

Упражнение 3.2. Допустим, что три величины X, Y, Z имеют различные значения.
Если X:

а) больше Y и больше Z, перейти к инструкции 1;
б) меньше Y и больше Z, перейти к инструкции 2;
в) больше Y и меньше Z, перейти к инструкции 3;
г) меньше Y и меньше Z, перейти к инструкции 4.

Написать последовательность инструкций IF, реализующую все эти предписания (так как заранее предположено, что X, Y и Z различны, второй номер в списках инструкций IF значения не имеет).

Познакомившись с определением и свойствами инструкции IF, рассмотрим теперь примеры ее использования в фортрановских программах.

Пусть для правильных дробей $|X| < 1$. необходимо вычислить приближенно натуральный логарифм выражения 1.0 — X для положительных X и выражения 1.0 + X для отрицательных X и результат присвоить переменной ALN.

Для этой цели можно воспользоваться формулой разложения функции ln (1—x) в степенной ряд в интервале (—1, +1):

$$\ln (1 - x) \approx -(x + x^2/2 + x^3/3 + x^4/4). \quad (3.4)$$

Для отрицательных x можно использовать ту же формулу, заменив x на $-x$.

В такой программе, прежде чем приступить к вычислению по формуле (3.4), следует проверить знак величины X:

```
      IF (X) 1, 2, 2
    1 X = — X
    2 ALN =
     *  — (X + X**2/2. + X**3/3. + X**4/4.)
```
(3.5)

После выполнения инструкции 1 знак величины X будет изменен, и машина перейдет к инструкции 2, чтоб вычислить натуральный логарифм (1.—X) для положительного числа X. Если же X было положительным, то инструкция IF обеспечивает непосредственный переход к вычислению логарифма (1.—X) (инструкция 2).

Общей чертой всех программ для вычислительных машин (не только фортрановских программ) является возможность неоднократного выполнения одной инструкции или последовательности инструкций внутри одной программы. Такое многократное исполнение инструкций называют *циклом*. Для управления циклом может быть использована инструкция IF. Вычисление факториала числа N

$$N! = N \cdot (N - 1) \cdot (N - 2) \ldots 2 \cdot 1$$

может служить одним из простейших примеров цикла:

```
      IFACT = 1
      K = N
    1 IFACT = IFACT*K
      K = K — 1
      IF(K) 2, 2, 1
    2 продолжение программы
```
(3.6)

В этом примере результат N! присваивается переменной IFACT. Две первые инструкции этой программы предшествуют циклу и устанавливают исходные состояния переменных IFACT и K.

Инструкция IFACT = 1 подготавливает переменную IFACT для первого умножения.

Инструкция K = N необходима, если в ходе вычисления N! желательно сохранить величину N. Инструкции

```
1 IFACT = IFACT*K
  K = K — 1
  IF (K) 2, 2, 1
```

составляют собственно цикл. Первая цифра определения индекса в инструкции IF здесь значения не имеет, поскольку K никогда не сможет стать отрицательным. Инструкция IF обязывает машину повторить эту группу инструкций N раз — до тех пор, пока K не станет равным нулю: инструкция

$$K = K - 1$$

уменьшает переменную K от начального состояния N на единицу при каждом повторении цикла. Инструкция

$$IFACT = IFACT * K$$

последовательно накапливает в переменной IFACT произведения, дающие в результате N-кратного повторения цикла IFACT = N!

Этот пример показывает, как понятие цикла помогает существенно экономить место в памяти машины и время на программирование (запись трех инструкций вместо N последовательно записанных инструкций умножения), а также делает программы универсальными (вышеописанный цикл вычисляет факториал для любого N).

В предыдущем примере инструкция IF, следя за переменной K, обеспечивала выход из цикла, как только K = 0.

Инструкция IF может в такой же степени управлять циклом, заканчивая его, как только переменный счетчик повторений цикла достигает заранее заданного фиксированного значения. Примером такого использования IF может служить приводимая ниже программа суммирования арифметической прогрессии.

Для вычисления суммы арифметической прогрессии необходимо знать первый член прогрессии (пусть он

называется A1), разность прогрессии (назовем ее D) и число ее членов (названное N)

```
      S = A1
      I = 1
1     S = S + D
      I = I + 1
      IF (I — N) 1, 2, 2
2     продолжение программы
```
(3.7)

Первые две инструкции и здесь являются подготовительными к циклу: первая из них присваивает сумме (переменная S) значение первого члена прогрессии, вторая — счетчику (переменная I) начальное значение 1. Сам цикл составляют последующие три инструкции, начинающиеся с инструкции номер 1. При каждом прохождении цикла сумма S увеличивается на величину D, а счетчик I на единицу. Как только переменная достигает значения константы N, цикл заканчивается передачей управления инструкции 2.

Если N = 10, то цикл повторится 9 раз, это объясняется тем, что к началу цикла I = 1, при первом прохождении цикла I = 2, а при девятом, следовательно, I = 10. Если по каким-либо соображениям целесообразно иметь величину I совпадающей с числом прохождения цикла, необходимо подготовительной инструкцией присвоить величине I значение на единицу меньшее: I = 0.

В рассмотренной программе участвуют величины как действительные (S, D, A), так и целые (I, N). Поскольку константа, фиксирующая число повторений цикла, была названа, тем самым был определен выбор названия для переменной-счетчика I.

Следует заметить, что программа носит учебный характер. На практике вычисление арифметической прогрессии может быть выполнено одной арифметической инструкцией

```
S = (2.*A1 + D*(AN — 1.))/2.*AN
```

где величины S, A1 и D имеют тот же смысл, что и в примере (3.7), а AN — число членов прогрессии, которое здесь должно быть действительным.

Упражнение 3.3. Написать программу, которая присваивает каждому элементу массива (названного LIST) значение, равное порядковому номеру элемента в массиве. Так, значением LIST(1) должна быть единица, LIST(2) = 2, ..., LIST(N) = N. Массив содержит 1200 элементов.

Выше обсуждался пример использования инструкции для реализации разветвления по двум направлениям.

Для этого, как было показано, необходимо, чтобы в списке трех номеров инструкций два были одинаковыми. Вообще говоря, можно представить инструкцию со списком из трех одинаковых номеров. Легко видеть, что инструкция

IF (A — B) 10, 10, 10

передает управление инструкции 10, какой бы ни была величина арифметического выражения A—B. Такое использование инструкции IF не лишено смысла, так как иногда при программировании возникает ситуация, требующая обязательной, безусловной передачи управления инструкции, расположенной не непосредственно после только что выполненной. Однако, для этих целей IF обычно не используют, ибо такая передача управления может быть выполнена специальной, но более простой инструкцией GØ TØ.

Инструкция GØ TØ состоит из двух английских слов, за которыми следует единственный номер инструкции. По этой инструкции вычислительная машина будет обязана в качестве следующей инструкции выполнять ту, номер которой указан в инструкции GØ TØ. Например, инструкция GØ TØ 25 приказывает машине приступить к исполнению инструкции 25.

Инструкция GØ TØ осуществляет так называемый *безусловный переход*, который используется для обязательных передач управления другим частям программы.

Пусть инструкция IF должна в зависимости от знака арифметического выражения A—B передать управление одному из трех участков программы, а после исполнения каждого из этих участков необходимо продолжить вычисления, начиная с инструкции номер 5. Тогда, используя инструкцию GØ TØ, можно записать решение этой

задачи:

```
      IF (A — B) 1, 2, 3
1 . . .
  инструкция или группа инструкций
  GØ TØ 5
2 . . .
  инструкция или группа инструкций
  GØ TØ 5
3 . . .
  инструкция или группа инструкций
  GØ TØ 5
5 продолжение программы
```

§ 2. Циклы

В предыдущем параграфе было рассмотрено использование инструкций IF для управления циклом с помощью счетчика — переменной, увеличивающейся при каждом прохождении цикла. Для этого потребовались три инструкции: одна подготовительная, устанавливающая начальное значение счетчика и две инструкции цикла — инструкция изменения счетчика и инструкция IF, управляющая циклом.

Фортран располагает мощным средством программирования циклов, при котором указанные три инструкции могут быть заменены одной. Это инструкция DØ. Она имеет такую форму: слово DØ, за ним номер инструкции, вслед за номером инструкции — определение индекса:

```
DØ 2 J = 1,25                    (3.8)
```

Здесь 2 — это номер инструкции, а

J = 1,25 — определение индекса.

Инструкция DØ заставляет машину многократно выполнять все инструкции, начиная с инструкции, непосредственно следующей за DØ, до инструкции, номер которой указан в инструкции DØ, включительно. Определение индекса (например, J = 1,25) говорит, что

индекс (переменная, указанная в определении) при первом прохождении цикла равен первой из двух целых величин (1) и

цикл должен повторяться до тех пор, пока значение индекса не станет равным второй величине (25).

Определением индекса называется выражение, составленное всегда из неиндексированной целой переменной, за которой следует знак присваивания и две (или три) целых величины, разделенных запятой. Неиндексированная целая переменная, стоящая слева от знака присваивания в определении индекса, называется *индексом цикла.*

Орфография инструкции DØ строга. Строка (3.8) — это пример синтаксически правильно построенной инструкции DØ, но инструкция

$$\text{DØ 2 A} = 1, 25$$

уже недопустима, хотя от (3.8) ее отличает замена буквы J на букву A — индекс цикла не может быть действительным. Точно так же некорректна инструкция

$$\text{DØ 2, J} = 1, 25$$

— поставлена лишняя, недопустимая запятая между номером инструкции и определением индекса.

В таком определении подразумевается, что каждое прохождение цикла увеличивает индекс на единицу.

Инструкция DØ, таким образом, управляет циклом, границы которого определены номером инструкции (от инструкции DØ до инструкции с указанным в DØ номером включительно), а количество прохождений — определением индекса.

Например, инструкция DØ в (3.9) определяет цикл как группу инструкций с номерами 17, 13 и 2. Она предполагает прохождение цикла (повторение инструкций 17, 13, 2) 25 раз (число прохождений равно разности между второй и первой константами в определении индекса, увеличенной на единицу):

$$\left.\begin{array}{l} \text{DØ 2 J} = 1{,}25 \\ \text{17 K(J)} = \text{J}*2 \\ \text{13 L(J)} = \text{J}*3 \\ \text{2 M(J)} = \text{L(J)} - \text{K(J)} \end{array}\right\} \qquad (3.9)$$

Цикл, определяемый инструкцией

$$\text{DØ 20 I} = 5, 95$$

должен повториться $95 - 5 + 1 = 91$ раз.

Выполнив при первом прохождении цикла, определенного инструкцией

```
DØ 20 I = 5,95
```

все инструкции до инструкции 20 включительно и увеличив на 1 индекс I (I = 6), машина возвращается для выполнения второго прохождения цикла к инструкции, непосредственно следующей за инструкцией DØ. Второе прохождение заканчивается также выполнением инструкции 20, в этот момент I = 7, затем вновь выполняется инструкция, непосредственно следующая за инструкцией DØ, и т. д. В конце 91-го прохождения цикла I = 95, этим обеспечивается выход из цикла — переход к следующей инструкции после инструкции 20.

Величины, стоящие справа от знака равенства в определении индекса инструкции DØ, (одна из них или обе) могут быть не только целыми константами, но и целыми переменными. Следовательно, следующие инструкции:

```
DØ  5 I = 2,30
DØ 15 I ND = K, 50
DØ 21 J = LIM, KØN
```

являются формально допустимыми.

Использование переменных в определении индекса инструкции DØ делает возможным формирование числа повторений цикла: как начальное, так и конечное значения индекса могут быть вычислены до начала цикла и присвоены целым константам в определении индекса.

Итак, величины, указанные в определении индекса, могут быть либо константами, либо переменными, но на их месте недопустимы никакие арифметические выражения. Конечно, величина может быть рассмотрена как тривиальный частный случай арифметического выражения. Здесь речь идет о недопустимости нетривиальных выражений. Так, инструкция

```
DØ 2 IND = 1, K + 3
```

недопустима.

Из двух величин в определении индекса первая (*нижняя граница индекса*) должна быть меньше второй (*верхней границы индекса*), так как при повторении цикла индекс всегда возрастает. Если по какой-либо причине

это правило не соблюдено, инструкции, составляющие цикл, выполняются только один раз, после чего управление передается следующим после цикла инструкциям программы.

Фортран допускает употребление третьей целой величины в определении индекса. Ее используют в том случае, когда при каждом прохождении цикла необходимо увеличивать индекс не на единицу, а на некоторую другую постоянную величину, называемую *шагом*.

Если, например, необходимо в инструкции DØ указать, что индекс IND пробегает все нечетные значения от 1 до 99, это может быть записано

$$\text{DØ 10 IND} = 1, 99, 2 \qquad (3.10)$$

Последовательные значения индекса IND при повторениях цикла будут 1, 3, 5, ..., 99 — третья величина в определении индекса IND (3.10) обеспечивает прибавление 2 к переменной при каждом прохождении цикла.

Если в инструкции (3.10) верхняя граница индекса будет равна 100, то возникает ситуация, при которой индекс, пробегая последовательность нечетных чисел, никогда не достигнет верхней границы. В такой ситуации последним значением индекса, которое он принимает при исполнении цикла, будет ближайшее к верхней границе, но не превосходящее ее значение. В инструкции

$$\text{DØ 5 I} = 1, 100, 2 \qquad (3.11)$$

последним значением индекса в цикле будет 99. После последнего выполнения группы инструкций цикла (при I = 99) управление будет передано следующей после цикла инструкции.

Число повторений цикла можно определить и в том случае, когда в определении индекса указаны три величины; только на этот раз формула будет выглядеть несколько сложнее. Если k — число повторений цикла, l_1 — нижняя граница индекса, l_2 — верхняя граница индекса, p — шаг, то

$$k = \left[\frac{l_2 - l_1}{p}\right] + 1 \qquad (3.12)$$

В формуле (3.12) квадратная скобка означает целую часть числа ([X] — целая часть X). В соответствии с

формулой (3.12) для циклов (3.10) и (3.11) получаем одно и то же число повторений цикла. Действительно,

для (3.10)

$$k=\left[\frac{99-1}{2}\right]+1=50,$$

и для (3.11)

$$k=\left[\frac{100-1}{2}\right]+1=50.$$

Все три величины в определении индекса могут быть как константами, так и переменными, при этом третья величина— шаг — может быть опущена. Отсутствие шага в инструкции DØ автоматически определяет шаг равным единице.

В предыдущем параграфе были построены примеры циклов, управляемых инструкцией IF. В отличие от них, цикл, управляемый инструкцией DØ, называется *циклом* DØ.

Рассмотрение примеров использования инструкции DØ начнем с простой программы суммирования 100 элементов массива A (I) (I = 1, 100). Если переменная, которой будет присвоено значение суммы, названа SUM, то программа может быть записана так:

```
      SUM = 0.0
      DØ 1 I = 1, 100
    1 SUM = SUM + A (I)
```

(3.13)

Первая инструкция этой программы — подготовительная к циклу, она присваивает 0.0 величине SUM, которая будет накапливать сумму элементов массива. Сам цикл состоит из единственной инструкции с номером 1, которая выполняется 100 раз. При каждом ее исполнении индекс I увеличивается на единицу, обеспечивая прибавление к величине SUM новых элементов списка A: A (1), A (2), ..., A (100). Когда цикл заканчивается, величина SUM равняется сумме всех элементов массива.

Инструкцию DØ, определяющую и начинающую его, назовем *началом цикла*, а инструкцию, номер которой указан в DØ,— его *концом*. Ниже многократно будут использоваться выражения *внутри цикла*, *вне цикла*. Они определяются вместе с определением

понятия *внутренняя инструкция цикла* DØ. Внутренними инструкциями цикла DØ являются все инструкции, последовательно исполняемые между началом и концом цикла. При этом конец цикла включается в число внутренних инструкций, а начало — нет. Так, в цикле (3.9) инструкции с номерами 17, 13 и 2 внутренние, а инструкция с номером 2 — еще и конец цикла.

Все инструкции, кроме внутренних инструкций некоторого цикла и его начала, называют *внешними* по отношению к этому циклу.

Если сравнить цикл DØ (3.13) и цикл (3.7) предыдущего параграфа, можно заметить, что величина I в обоих случаях играет роль счетчика. Но в цикле (3.13) у этой величины есть и вторая роль — роль индекса массива A (I). Вообще индекс цикла DØ может участвовать в инструкциях цикла как слагаемое, множитель, показатель, индекс массива и т. д. При его использовании внутри цикла, надо помнить, что он является целой переменной, хотя превращение его в действительную переменную с помощью инструкций типа XI = I вполне допустимо.

Использование индекса в преобразованиях внутри цикла открывает широкие возможности при программировании. Вот одна из них. Выше отмечалось, что индекс цикла может лишь монотонно увеличиваться при повторениях цикла, однако несложный пример

```
DØ 5 I = 1, N
J = N + 1 — I
```

показывает, как благодаря введению искусственной переменной можно реализовать цикл с монотонно уменьшающимся параметром: достаточно во всех внутренних инструкциях цикла, следующих за инструкцией

```
J = N + 1 — I
```

использовать в качестве переменного параметра не индекс цикла I, а определяемую им величину J. Действительно, когда I пробегает последовательность значений

1, 2, 3, ... N—2, N—1, N,

J становится равным соответственно

N, N—1, N—2, ... 3, 2, 1.

Единственное ограничение, которое накладывается на использование индекса цикла DØ: индекс не должен появляться в левой части арифметического выражения. Иными словами, допускается лишь автоматическое, предусмотренное инструкцией DØ изменение индекса на один шаг при каждом повторении цикла; никакие инструкции, искусственно изменяющие значение индекса внутри цикла, не разрешаются.

Еще один простой цикл осуществляет «очистку массива» — присвоение нулевого значения всем элементам массива B (J):

```
    DØ 5 J = 1, 1000
5 B (J) = 0.
```

Здесь подготовительных инструкций нет, в цикле исполняется одна инструкция

```
5 B (J) = 0.
```

которая при каждом прохождении цикла изменяется за счет изменения индекса J:

```
B (1) = 0.
B (2) = 0.
. . .
```

Цикл заканчивается, когда инструкция B (J) = 0. будет выполнена последний раз:

```
B (1000) = 0.
```

Внутри цикла DØ может быть использована любая инструкция. Для последней же исполняемой инструкции цикла (ее номер указан в инструкции DØ) существует ограничение: цикл не может кончаться инструкциями передачи управления IF и GØ TØ. Если же необходимость обязывает закончить цикл инструкцией передачи управления, в этом случае после такой инструкции записывается еще одна — CØNTINUE (ПРОДОЛЖАТЬ). Эта неисполняемая инструкция служит только для отметки конца цикла и обязательно имеет номер, который указывается в инструкции DØ. Без DØ инструкция CØNTINUE никогда не употребляется.

Пусть задача состоит в том, чтобы в массиве X(I) найти первый элемент, равный десяти. Необходимо, во-первых, в случае неравенства

$$X(I) \neq 10.$$

обеспечить переход к проверке следующего элемента X (I + 1) и, во-вторых, если все элементы массива оказались не равными десяти, обеспечить выход из цикла.

Вот решение поставленной задачи:

```
      DØ 5 I = 1, 200
      IF (X(I) — 10.) 5, 10, 5
    5 CØNTINUE
```
(3.14)

Инструкция CØNTINUE в цикле (3.14) необходима. Сама по себе она не оказывает действия на элементы массива и не использует их, но каждое ее «исполнение» фиксирует прохождение цикла, т. е. обеспечивает прибавление шага к индексу, возврат к следующей после DØ инструкции в случае, если индекс не достиг верхней границы, и выход из цикла при достижении ее.

Инструкция DØ может быть использована в программе определения максимальной или минимальной величины в массиве. Пусть X(I) — массив 50 положительных элементов. Поставим задачу: найти в этом массиве наибольшее число и присвоить это значение переменной AMX. Перед циклом подготовительная операция

$$AMX = 0.0$$

дает переменной AMX заведомо меньшее значение, чем минимум из элементов массива X(I).

```
      AMX = 0.0
      DØ 1 I = 1, 50
      IF (AMX — X(I)) 2, 2, 1
    2 AMX = X(I)
    1 CØNTINUE
```
(3.15)

В этом примере видно, что без инструкции CØNTINUE инструкция IF не могла бы обеспечить повторение цикла в случае

$$AMX > X(I)$$

передавая управление инструкции, следующей за инструкцией 2, т. е. инструкции вне цикла. В примере же (3.15) даже в случае AMX > X(I) происходит возврат в начало цикла (инструкция IF), чтобы проверить следующий элемент X(I + 1), который может оказаться больше AMX.

В этом примере индекс I упомянут дважды. Изменение индекса при достижении последней инструкции цикла (здесь — инструкции CØNTINUE) происходит независимо от числа его употреблений, в каждой из инструкций цикла при следующем прохождении будет фигурировать измененная величина.

Важно еще раз подчеркнуть, что CØNTINUE используется в тех случаях, когда исполнение цикла заканчивается или может закончиться инструкциями передачи управления. В примере (3.15) последней написанной инструкцией цикла является инструкция

```
2 AMX = X(I)
```

однако возможна ситуация (при AMX > X(I)), когда последней исполняемой в цикле инструкцией будет условная передача управления

```
IF (AMX — X(I)) 2, 2, 1
```

это и вызывает необходимость окончания цикла (3.15) инструкцией CØNTINUE.

Упражнение 3.4. Написать программу с циклом DØ определения минимального по абсолютной величине значения в массиве X (I) из тысячи элементов. Найденное значение присвоить величине AMN. Первое число в массиве можно считать положительным. Элементы массива должны быть сохранены.

Вернемся снова к циклу (3.14). Он интересен тем, что инструкция IF, расположенная внутри цикла, может при X(I) = 10. передать управление инструкции 10 вне цикла. Индекс I при этом может и не достигнуть 200 — своей верхней границы.

Значение индекса при таком выходе из цикла сохраняется и может быть использовано последующими инструкциями программы. В нашем примере индекс I после выхода из цикла на инструкцию 10 будет иметь конкретное содержательное значение — это номер первого в массиве X(I) элемента, равного десяти.

Выход из цикла после исполнения последней его инструкции при значении индекса, равном верхней границе (или отстоящем от верхней границы меньше чем на шаг), называется *нормальным*.

Выход из цикла при значении индекса, не достигшем верхней границы, будем называть *специальным*.

Таким образом, выход из цикла (3.14) после инструкции будет нормальным, а выход на инструкцию 10 — специальным. В цикле (3.15) возможен только нормальный выход.

Упражнение 3.5. Использовать цикл для того, чтобы переписать элементы массива X(I), I = 1, 100 в массив Y(J), J = 1, 100 так, чтобы порядок элементов в массиве Y (J) был противоположен исходному, т. е.

$$X(1) = Y(100),\quad X(2) = Y(99),\ \ldots,\quad X(100) = Y(1).$$

Цикл DØ, вообще говоря, не является независимым участком программы. Однако, связь инструкций цикла DØ с инструкциями вне цикла не произвольна, она ограничена рамками четких правил.

1. Запрещена любая передача управления, условная или безусловная (IF или GØ TØ), извне цикла инструкциям, расположенным внутри него. Извне можно передавать управление лишь на начало цикла, т. е. на саму инструкцию DØ, но ни в коем случае ни на одну из внутренних инструкций. Внутренним инструкциям цикла DØ может быть передано управление лишь инструкциями IF и GØ TØ, расположенными внутри того же цикла.

Недопустимой в соответствии с этим правилом является, например, группа инструкций

```
      DØ 1 I = 2, 100
    1 X(I) = Y(I)
      IF (X(I) — 1.0) 1, 2, 1
```

поскольку третья из этих инструкций, внешняя по отношению к циклу DØ, предусматривает возможность передачи управления инструкции, расположенной внутри цикла.

2. Цикл DØ может содержать внутри один или несколько других циклов DØ. При этом не допускается «перекрытий» циклов: включаемый цикл — внутрен-

ний — должен находиться внутри включающего внешнего. Вот пример допустимого включения двух циклов:

```
      DØ 2 I = 1, 10
      DØ 1 J = 1, 100
      T1(I, J) = T2(I, J)
    1 CØNTINUE
    2 CØNTINUE
```
(3.16)

Эта программа переписывает величины из таблицы T2 (I, J) в таблицу T1 (I, J). Обе таблицы имеют одинаковую размерность 10 × 100.

Однако, программа

```
      DØ 2 I = 1, 10
      DØ 1 J = 1, 100
      T1 (I, J) = T2 (I, J)
    2 CØNTINUE
    1 CØNTINUE
```
(3.17)

уже недопустима, так как здесь цикл

```
DØ 1 J = 1, 100
```

начинающийся внутри цикла 2, кончается после него.

Программа (3.16) будет проще, если оба цикла будут иметь общее окончание (что не противоречит правилу вхождения циклов):

```
      DØ 1 I = 1, 10
      DØ 1 J = 1, 100
      T1(I, J) = T2(I, J)
    1 CØNTINUE
```
(3.18)

Более того, так как последняя исполняемая инструкция обоих циклов (3.18) не является передачей управления и тем самым нет необходимости в инструкции CØNTINUE, то программа (3.16) может быть записана:

```
      DØ 1 I = 1, 10
      DØ 1 J = 1, 100
    1 T1 (I, J) = T2 (I, J)
```
(3.19)

Как видно из рассмотренных примеров, инструкция CØNTINUE может заканчивать цикл и в том случае, когда последняя его исполняемая инструкция отличается от IF или GØ TØ.

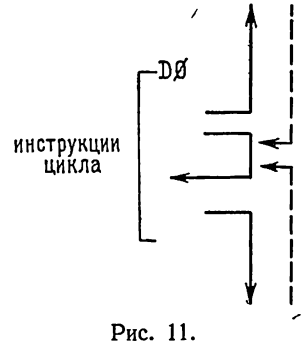

Рис. 11.

Комбинацию двух циклов, удовлетворяющих правилу вхождения циклов, будем называть *вложенными циклами*. Понятие вложенных циклов легко обобщается на случай трех и более циклов DØ. Максимально допустимое число вложенных циклов зависит от возможностей, заложенных в конкретных трансляторах. Трансляторы IBM-360, например, допускают не более 25 вложенных циклов. Транслятор советской машины БЭСМ-6 не имеет ограничений на число допустимых вложенных циклов.

На рис. 11 схематически изображены некоторые возможные (сплошные линии) и недопустимые (пунктирные линии) передачи управления. Стрелки показывают направления условных или безусловных переходов.

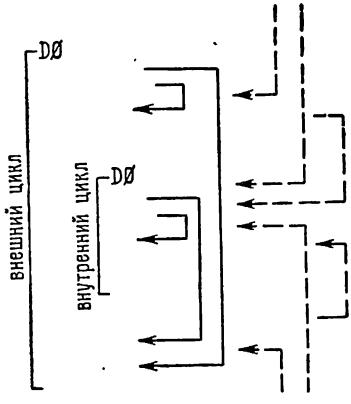

Рис. 12.

Рис. 12 — это схема допустимых и недопустимых передач управления в случае двух циклов — внутреннего и внешнего.

Если k — это число повторений внешнего цикла, а l — внутреннего (k и l определены по формуле (3.12) для соответствующих циклов), то общее число m повторений инструкций внутреннего цикла в программе из двух вложенных циклов будет равно

$$m = k \cdot l$$

Действительно, каждое прохождение внешнего цикла соответствует исполнению всех внутренних инструкций, в том числе и полному исполнению инструкции DØ внутреннего цикла: каждое прохождение внешнего цикла — это l исполнений инструкций внутреннего.

Рассмотрим пример программы с двумя вложенными циклами. Эта программа отыскивает в двух массивах A(I) и B(J) первую пару совпадающих элементов (если такая пара существует), фиксируя номера элементов этой пары в каждом из массивов:

M1 — номер элемента в массиве A (I),
M2 — номер элемента в массиве B (J).

```
   DØ 1 I=1, 100
   DØ 1 J=1, 100
   IF (A(I) — B(J)) 1, 2, 1
1  CØNTINUE
   GØ TØ 3
2  M1 = I
   M2 = J
3  продолжение программы
```

(3.20)

Работа программы начинается с присвоения индексу I значения 1 (начало внешнего цикла), затем присваивается значение 1 индексу J (начало внутреннего цикла). При первом прохождении инструкция IF сравнивает A(1) и B(1). Внутренний цикл присваивает индексу J значение 2 (I по-прежнему равно 1) и сравнивает на этот раз A(1) и B(2). Так продолжается до J = 100. В этот момент индекс I становится равным 2, а внутренний цикл повторяется, так что индекс J снова пробегает все значения от 1 до 100.

При A (1) = B(J) инструкция IF обеспечивает специальный выход на инструкцию 2, две инструкции M1 = I и M2 = J фиксируют номера элементов. Если пара равных элементов не обнаружена, нормальный выход происходит после «исполнения» инструкции CØNTINUE при одновременном выполнении равенств

$$I = 100, \quad J = 100.$$

Упражнение 3.6. Составить с помощью цикла DØ программу, считающую число положительных элементов массива X(I), I = 1, 500.

Два вложенных цикла должны иметь в качестве индексов различные переменные. Циклы же, выполняемые последовательно, свободны от этого ограничения.

Многочисленные программы сортировок всегда используют вложенные циклы. Рассмотрим один из простых вариантов сортировки, расставляющей элементы массива T(I) в порядке возрастания.

В программе сравниваются два соседних элемента массива и, если необходимый порядок (возрастание) в паре нарушен, переставляются местами. Двух вложенных циклов оказывается достаточно, чтобы осуществить полное упорядочение. При первом прохождении внутреннего цикла T(1) сравнивается (и переставляется в случае необходимости) с T(2), при втором — T(2) с T(3), далее T(3) с T(4) и т. д.

Пусть N — число элементов массива. После N сравнений (и, возможно, перестановок), т. е. после N повторений внутреннего цикла, наибольшее число оказывается на нужном месте, в конце цикла. Этим заканчивается первое прохождение внешнего цикла, для второго надо будет проделать уже не N сравнений (и, возможно, перестановок), а только N — 1: последний элемент списка не учитывается. В организованной так программе текущее значение индекса внешнего цикла служит верхней границей индекса внутреннего цикла.

Для N = 200 программа будет иметь вид

```
      N = 200
      DØ 2 I = 1, N
      K = N + 1 — I
      DØ 2 J = 1, K
      IF (T (J) — T (J + 1)) 2, 2, 1
    1 R = T(J)
      T (J) = T(J + 1)
      T (J + 1) = R
    2 CØNTINUE
```

(3.21)

Инструкция IF в программе (3.21) проверяет порядок в паре элементов массива. Если этот порядок совпадает с требуемым (T (J) $\leqslant$ T (J + 1)), то передается управление на последнюю инструкцию внутреннего цикла и тем самым на проверку следующей пары. Если же элементы пары расположены в обратном порядке, действуют три инструкции, начиная с инструкции 1, меняющие местами элементы пары.

В инструкции 1 в (3.21) употреблена переменная R. Это промежуточная величина, которая необходима как временное хранилище значения T(J) при перемене мест. Она обеспечивает сохранение обеих величин T(J) и T(J + 1). Использование промежуточных величин при взаимных пересылках — стандартный прием программирования.

Упражнение 3.7. Написать программу транспонирования квадратной матрицы T (I, J) размерности N × N.

§ 3. Сложные передачи управления

Инструкция IF, рассмотренная в первом параграфе этой главы, могла реализовать разветвления программы максимально по трем направлениям. В случае необходимости сделать более чем три разветвления часто используют инструкцию, называемую *вычисляемой* GØ TØ.

Эта инструкция состоит из слов GØ TØ, за которыми следует пара скобок, содержащая любое желаемое число номеров инструкций, разделенных запятыми; замыкает вычисляемую GØ TØ целая неиндексированная переменная, которая отделена от предшествующих скобок запятой (см. определение инструкции GØ TØ на стр. 60).

Формально допустимый пример вычисляемой GØ TØ:

GØ TØ (5, 4, 3, 2, 1), K (3.22)

Инструкции же

GØ TØ (6, 10, 14, 18) LØM
GØ TØ (302, 205, 102, 17), X

недопустимы: первая из них не имеет запятой между скобкой и переменной, во второй указана не целая, а действительная переменная.

По инструкции «вычисляемая GØ TØ» управление передается инструкции, место которой в списке номеров

равно значению целой неиндексированной переменной в вычисляемой GØ TØ. Например, если в инструкции (3.22) K = 2, то управление передается инструкции номер 4 (этот номер упомянут вторым в списке номеров), при K = 5 — инструкции номер 1.

Итак, значение переменной в вычисляемой GØ TØ не является номером следующей исполняемой инструкции (как в инструкции GØ TØ). Это значение служит скорее индексом списка элементов, стоящих в скобках вычисляемой GØ TØ.

Длина списка номеров инструкций не ограничивается. Более того, один и тот же номер инструкции может появляться в списке несколько раз. Например, допустима инструкция GØ TØ (18, 20, 22, 18, 19, 20), IND На переменную в вычисляемой GØ TØ накладывается ограничение — она не должна превосходить числа элементов списка. В инструкции (3.22) величина K никогда не должна превышать 5.

Упражнение 3.8. Написать первые две инструкции цикла DØ, исполняемого 10 раз, которые бы определили, что при первом прохождении цикла исполняется часть его, начинающаяся с инструкции номер 1, при втором — часть, начинающаяся с инструкции номер 2, и т. д. до номера десятого, когда должна исполняться часть цикла, начинающаяся с инструкции номер 10.

Еще одна инструкция, допускающая переход в одном из нескольких направлений, — *присваиваемая* GØ TØ. Эта инструкция встречается в программах обязательно вместе с инструкцией ASSIGN.

Инструкция ASSIGN состоит из двух ключевых слов ASSIGN и TØ и двух величин. Одна из них — номер инструкции — располагается после слова ASSIGN, другая — целая неиндексированная переменная — после слова TØ. Инструкция ASSIGN может выглядеть, например, так:

```
ASSIGN 20 TØ J
```

Инструкция «присваиваемая GØ TØ» состоит из двух слов GØ и TØ, за которыми сначала следует целая неиндексированная переменная (упомянутая в инструкции ASSIGN после слова TØ), обязательная запятая и, наконец, скобки, содержащие несколько номеров инструкций, разделенных запятыми. Один из этих номе-

ров должен совпадать с номером инструкции, указанным непосредственно после слова ASSIGN. Пример допустимой пары инструкций (ASSIGN, «присваиваемая GØ TØ»)

```
ASSIGN 20 TØ J
GØ TØ J, (5, 10, 15, 20, 25)
```

Общая форма этой пары инструкций такова:

$$\left.\begin{array}{l}\text{ASSIGN } n \text{ TØ } m\\ \text{GØ TØ } m, (n_1, n_2, \ldots, n_k)\end{array}\right\} \qquad (3.23)$$

где n — номер инструкции; m — целая неиндексированная переменная, которой присваивается один из номеров инструкций $n_1, n_2, \ldots, n_k$.

Между ASSIGN и «присваиваемой GØ TØ» может быть расположено любое число фортрановских инструкций.

Итак, n по определению может принимать лишь значения из набора номеров инструкций $n_1, n_2, \ldots, n_k$.

В «присваиваемой GØ TØ», так же как и в «вычисляемой GØ TØ», этот набор может содержать любое число инструкций и даже иметь повторяющиеся номера.

В трех следующих парах инструкций ASSIGN и «присваиваемая GØ TØ» допущены ошибки:

```
ASSIGN 17 TØ ALEF
GØ TØ ALEF, (1, 17, 20, 25)                    (3.24)
ASSIGN 21 TØ ILEF
GØ TØ ILEF, (2, 15, 18, 24, 27)                (3.25)
ASSIGN 5 TØ LIMA
GØ TØ LIMA (5, 20, 29)                         (3.26)
```

В паре инструкций (3.24) переменная ALEF действительная; в паре (3.25) номер инструкции, упомянутый в ASSIGN, не содержится в списке инструкций соответствующей «присваиваемой GØ TØ»; в (3.26) пропущена запятая между переменной и скобками.

Если в инструкции

```
GØ TØ NØRD, (10, 20, 30)                       (3.27)
```

переменная NØRD получила предварительно значение 10, то инструкция (3.27) передает управление инструкции 10. Если NØRD = 20, следующей будет исполняться

инструкция 20, при NØRD = 30 управление передается инструкции 30.

Значение переменной NØRD к моменту исполнения «присваиваемой GØ TØ» определяется последней выполненной инструкцией ASSIGN. При желании изменить значение переменной NØRD или восстановить ее первоначальное значение необходимо использовать инструкцию ASSIGN.

В приводимом ниже примере фрагмента программы

```
    ASSIGN 31 TØ IKØN
  6 GØ TØ IKØN, (128, 134, 205, 230, 31)
128 A = A + B + C
 31 B = B + C*D
    ASSIGN 205 TØ IKØN
    GØ TØ 6
205 C = C + D*E
230 C = C**2
```

присваиваемая GØ TØ IKØN, (128, 154, 205, 230, 31) при первом своем исполнении передает управление инструкции 31, так как предшествующая инструкция ASSIGN присвоила переменной IKØN значение 31.

Тотчас после инструкции 31 следует ASSIGN, присваивающая переменной IKØN новое значение 205, и «присваиваемая GØ TØ», которой передано управление инструкцией GØ TØ 6, теперь осуществляет переход к инструкции 205.

Упражнение 3.9. В программу должны быть включены инструкции, вычисляющие площадь одной из геометрических фигур: прямоугольника,прямоугольного треугольника, круга. Инструкция, вычисляющая площадь прямоугольника, имеет номер 10, треугольника — 20, круга — 30. Выбор той или иной инструкции следует осуществить с помощью переменной KLIN в «присваиваемой GØ TØ».

Написать программу, осуществляющую выбор необходимой инструкции, предположив, что инструкции ASSIGN выполнены ранее.

Пусть A, B — стороны прямоугольника или катеты прямоугольного треугольника, R — радиус круга, S — площадь фигуры.

§ 4. Логические условные передачи управления

В первой главе были подробно изучены арифметические выражения, в § 1 этой главы рассмотрено их применение в инструкциях условной передачи управления.

Большинство трансляторов Фортрана позволяет использовать инструкции условной передачи управления, анализирующие не только арифметические, но и логические выражения.

Для описания логических выражений потребуются определения логических категорий: констант, переменных, операторов, выражений.

Существуют два значения для *логических констант*: .TRUE. (ИСТИНА) и .FALSE. (ЛОЖЬ). При написании этих констант в Фортране точки, окаймляющие ключевые слова TRUE и FALSE, обязательны. Этим константам условно приписывают числовые значения соответственно 1 и 0. Логическая константа .TRUE. указывает, что логическая переменная или выражение, приравненные этой константе, истинны. Константа .FALSE., напротив, означает ложность переменной или выражения, которые приравнены константе.

Элементарное *логическое выражение* может состоять из единственной логической константы или единственной логической (может быть индексированной) переменной, принимающих логическое значение (т. е. либо .TRUE., либо .FALSE.).

Использование *операторов отношений* и *логических операторов* позволяет образовывать более сложные логические выражения.

В Фортране существует шесть операторов отношения. Их определения ясны из приводимой ниже таблицы 2.

Пары букв, обозначающие операторы отношения, обязательно окаймляются точками. *Логическое выражение*, формулируемое оператором отношения, состоит из двух арифметических выражений, разделенных одним из шести операторов отношения.

Например, допустимо логическое выражение

$$(A**2 - C).\ LT.\ B**3 \qquad (3.28)$$

которое принимает значение .TRUE. (единицы) тогда, когда

$$(A**2 - C) < B**3$$

и значение .FALSE. (нуль) в случае

$$(A**2 - C) \geqslant B**3$$

В арифметических выражениях справа и слева от оператора отношения допустимы как действительные, так и целые переменные и константы. Однако, для обоих арифметических выражений (т. е. для обеих частей — правой и левой — логического выражения) тип переменных и констант должен быть одинаков.

Таблица 2

Оператор отношения	Название оператора	Алгебраическое обозначение	Примеры
.GT.	БОЛЬШЕ	$>$	A .GT.B
.GE.	БОЛЬШЕ ИЛИ РАВНО	$\geqslant$	2. .GE. B
.LT.	МЕНЬШЕ	$<$	A .LT. 1.
.LE.	МЕНЬШЕ ИЛИ РАВНО	$\leqslant$	A .LE. B
.EQ.	РАВНО	$=$	2 .EQ. I
.NE.	НЕ РАВНО	$\neq$	J .NE. I

Логические выражения

```
2. .GE. I
IND .EQ. A
X + Y .NE 17.5
```

недопустимы: в первом из них слева стоит действительная константа, а справа — целая переменная; во втором — две переменные разных типов; в третьем — не хватает точки справа от оператора отношения. В допустимом выражении 2..GE.B после двойки стоят две точки. Среди них нет лишней: первая из точек служит признаком действительного числа, вторая — левой окаймляющей точкой оператора отношения .GE.

В логических выражениях операторы отношения выражают некоторое условие, которое может быть истинным или ложным в зависимости от значений двух арифметических выражений. Второй путь формирования логических выражений использует *логические операторы*, которые имеют в качестве операндов логические переменные или логические константы. Таблица 3 — это таблица определения логических операторов.

Таблица 3

Логический оператор	Обозначение в логике	Название оператора	Определение	Примеры
.NØT.	¬ или ⁻	НЕ	.TRUE., когда операнд имеет значение .FALSE., и .FALSE., когда операнд имеет значение .TRUE.	.NØT. A
.AND.	∧ или &	И	.TRUE., когда оба операнда имеют значение .TRUE. и .FALSE. во всех других случаях	X .AND. Y
.ØR.	∨ или ∪	ИЛИ	.FALSE., когда оба операнда имеют значение .FALSE. и .TRUE. во всех других случаях	A .ØR. B

Буквы, образующие логические операторы, также окаймляются точками. В операторах .AND. и .ØR. обязательно присутствуют два операнда — слева и справа от знака оператора. В операторе .NØT. — только один операнд, располагаемый справа от .NØT.

Рассмотрим произвольный четырехугольник, вершины которого последовательно поименованы A, B, C, D. Тогда пары отрезков

(AB, CD) и (BC, AD)

являются парами противоположных сторон, которые назовем соответственно первой и второй парами. Пусть логическая переменная X (I) характеризует параллельность противоположных сторон четырехугольника: X (I) принимает значение .TRUE., когда противоположные стороны четырехугольника параллельны, и значение

.FALSE. в противоположном случае. Нетрудно видеть, почему логическая переменная X(I) обозначена как индексированная переменная: индекс может принимать два значения 1 и 2, соответственно для первой и второй пар противоположных сторон.

X(1) = .TRUE. при AB ∥ CD,
X(2) = .TRUE. при BC ∥ AD.

Логическое выражение

X(1).AND. X(2) (3.29)

принимает значение .TRUE. для параллелограмма.

Логическое выражение

X(1) .ØR. X(2) (3.30)

принимает значение .TRUE. для трапеции.

Логическое выражение

.NØT. X(1) (3.31)

истинно лишь тогда, когда X(1) = .FALSE., т. е. в случае непараллельности сторон AB и CD.

Соединенные логическими операторами .AND., .ØR. или .NØT., элементарные логические выражения типа (3.29), (3.30) также являются логическими выражениями. При составлении сложных логических выражений два логических оператора не могут следовать непосредственно друг за другом, за исключением случая, когда вторым оператором является оператор .NØT.

Примеры логических выражений, которые допустимы

LIN .AND. .NØT. LØN
L1 .AND. L2 .ØR. L3 .AND. L5

и недопустимы —

L.AND..ØR.W (два логических оператора следуют подряд);

.NØT. (.ØR.L1) (логический оператор .ØR. не имеет левого операнда).

Сложные логические выражения могут составляться из логических операторов таблицы 3 и операторов отношения таблицы 2 одновременно. Введем логическую переменную Y(J), характеризующую равенство противоположных сторон рассмотренного четырехугольника,

Переменная Y(J) принимает значение .TRUE., когда противоположные стороны четырехугольника равны, и значение .FALSE. в противном случае. Так же как определенная выше переменная X(I), величина Y(J) является логической индексированной величиной:

$$J = 1, 2$$

Тогда выражение

(Y(1).AND.Y(2)).AND. (AB.EQ. BC)

равно .TRUE. для ромба, у которого обе пары противоположных сторон равны (левый операнд — выражение Y(1).AND.Y(2)) и, кроме того, равны между собой (правый операнд — выражение AB.EQ.BC).

Вот другие примеры допустимых логических выражений:

.NØT. ((A — B).GT. (A*B)) (3.32)

((A — B) ** 2. EQ. 0.). AND. (.NØT. (A.LT. B)) (3.33)

Порядок действий арифметических и логических операторов при вычислении логических выражений определяется скобками.

Скобки могут быть опущены лишь в тех случаях, когда последовательность действий подчинена иерархии операторов, приведенной в таблице 4. В первом ее столбце указаны уровни иерархии. Во втором столбце перечислены операторы отношений, логические операторы и арифметические операции, которые здесь ради общности также названы операторами.

Таблица 4

Порядок выполнения	Операторы
1	2
1	Возведение в степень (**)
2	Умножение (*) и деление (/)
3	Сложение (+) и вычитание (—)
4	.LT. (<), .LE. (≤), .EQ. (=), .NE. (≠), .GT. (>), .GE. (≥)
5	.NØT. (¬)
6	.AND. (∨)
7	.ØR. (∧)

Операторы одного уровня иерархии (одного порядка выполнения) исполняются слева направо.

Например, логическое выражение

A.GT.D**B.AND..NØT.L. ØR.N

вычисляется в следующем порядке:

1) D**B, пусть W — результат возведения в степень;

2) A.GT.W, пусть X — результат логического выражения с оператором отношения .GT.;

3) .NØT.L, пусть Y — результат логического выражения с логическим оператором .NØT.;

4) X.AND.Y, пусть Z — результат логического выражения с логическим оператором .AND.;

5) Z.ØR.N — последнее действие: логический оператор .ØR. имеет наименьший приоритет.

Скобки в логических выражениях часто могут быть рекомендованы, даже если они не прибавляют никакой дополнительной информации к порядку вычислений, задаваемому иерархией операторов: так же как и в арифметических выражениях, скобки помогают сделать логическое выражение более наглядным.

Упражнение 3.10. Освободить логические выражения (3.32) и (3.33) от лишних скобок и вычислить значения этих выражений при A = 2.5 и B = 2.3.

Простейшими логическими инструкциями в Фортране являются инструкции *присваивания логических значений* или значений логических выражений.

Например, инструкция

G = .TRUE.

присваивает переменной G логическое значение .TRUE..

Однако, в основном логические значения, величины, операторы и выражения используются в логических условных передачах управления.

Логическая условная передача управления состоит из слова IF, за которым сначала следует пара скобок, содержащая логическое выражение (ср. с IF в § 1 этой главы), а затем любая фортрановская инструкция, кроме инструкций IF и DØ.

«Арифметическая» инструкция IF может быть представлена в такой общей форме:

IF (*a*) n_1, n_2, n_3,

где a — некоторое арифметическое выражение, а n_1, n_2, n_3 — номера инструкций, присваивающих a какое-либо значение.

Общая форма логической условной передачи такова:

IF (b) S,

где b — некоторое логическое выражение, а S — любая (кроме DØ и IF) фортрановская инструкция.

По логической условной передаче управления (будем кратко называть ее *логической* IF) сначала проверяется истинность логического выражения b, расположенного в скобках инструкции. Если значение логического выражения b есть .TRUE., то следующей выполняется инструкция S. Если логическое выражение b ложно (имеет значение .FALSE.), то инструкция S пропускается, и исполняется инструкция, непосредственно следующая за логической IF.

Следующие логические IF формально допустимы:

```
IF ((A*B).LT. 2.5) GØ TØ 15
IF (.NØT. (L1.ØR. L2)) S = S+1
IF (A.NE. B) X = (A—B)**2+(A—B)**2
```

В большинстве случаев «арифметическая» инструкция может выполнять те же функции, что и логическая IF.

Пусть, например, необходимо перейти к инструкции 20 при равенстве величин A и B и к следующей инструкции в случае неравенства. Этот условный переход может быть реализован «арифметической» IF.

```
    IF (A—B) 10, 20, 10
10 продолжение программы
```

Для этого надо лишь присвоить необходимый номер следующей инструкции программы.

Этот же условный переход осуществляется и логической IF

IF (A.EQ.B) GØ TØ 20

Здесь необходимость в нумерации следующей инструкции, вообще говоря, отпадает.

Однако, в некоторых случаях применение логической IF более рационально. Если, например, известно, что в арифметическом выражении

```
Y = X/((A**2+B ** 2)—(C**3+B*C+B**3))
* + A*B*C
```

величина X принимает лишь целые значения 0. и 1., то целесообразно перед вычислением Y проверить сначала значение X. Вычисление Y с помощью логической IF может быть записано двумя инструкциями:

```
      Y = A*B*C
      IF (X.EQ.1.) Y = Y+1./((A**2+B**2)
     *—(C**3+B*C+B**3))
```

тогда как при использовании «арифметической» IF их потребуется три:

```
      Y = A*B*C
      IF (X—1.) 2, 1, 2
    1Y =Y+1./((A**2 + B**2)—(C**3+B*C+
    *+B**3))
    2 продолжение программы
```

Упражнение 3.11. Переписать программу, составленную в упражнении 3.4 (стр. 69), заменив инструкции условной передачи IF на логические IF.

Это упражнение покажет читателю, в какой мере использование логических IF может сделать программы более компактными и элегантными.

§ 5. Инструкции останова и окончания

Инструкции останова и окончания являются в полном смысле слова операциями управления: они изменяют состояние машины, заставляя ее останавливать исполнение программы. Одна из инструкций останова — инструкция, состоящая из ключевого слова STØP, за которым при желании может быть указана целая константа. По этой инструкции машина останавливается так, что работа по программе может быть возобновлена лишь с самого ее начала. Этот так называемый *полный останов*. Одновременно с остановом машины на ее выводном устройстве печатается константа инструкции STØP, если она указана. Эта константа может содержать от одной до пяти десятичных цифр. Печать константы используется как дополнительная сигнализация об останове.

Инструкция останова PAUSE (*условный останов*), так же как и STØP, может сопровождаться целой кон-

стантой. По инструкции PAUSE машина тоже останавливается, печатая слово PAUSE и константу, если она указана. Инструкция PAUSE отличается от STØP тем, что после PAUSE работа программы может быть возобновлена нажатием кнопки «ПУСК» на пульте машины. Первой исполняемой при этом инструкцией будет инструкция, написанная непосредственно вслед за PAUSE. Инструкция PAUSE может быть использована как эффективное средство отладки или контроля программ с пульта. Для этого в необходимых местах программы вставляются инструкции PAUSE; константы при разных инструкциях PAUSE могут символизировать ту или иную предусмотренную в программе ситуацию; программист при работе за пультом узнает об этих ситуациях по константам, сопровождающим слово PAUSE, и напечатанным на выводном устройстве машины.

Например, в программе вычисления корней квадратного уравнения

$$Ax^2 + Bx + C = 0$$

предварительно вычисляется его дискриминант

$$B^2 - 4AC.$$

Если $B^2 - 4AC < 0$, то действительных корней не существует. Машина останавливается по инструкции PAUSE, печатая, например, константу 1111. Программа после этого может быть продолжена вычислением действительной части корней

```
    D = B**2 - 4.*A*C
    IF (D) 5, 10, 10
 5  PAUSE 1111
    D = 0.
10  X1 = B/(2.*A) + D**0.5
    X2 = B/(2.*A) - D**0.5
```

Упражнение 3.12. Исходя из массива X(I), I = 1,2,... ..., N, получить массив Y(I), состоящий из обратных величин списка X(I), т. е. Y(I) = 1./X(I) (это программа обращения диагональной матрицы порядка N). Если окажется, что деление на X(I) невозможно (X(I) = 0.), остановить машину инструкцией PAUSE 55, предусмотрев продолжение счета. Окончание вычислений отметить инструкцией PAUSE 44.

В некоторых конкретных реализациях Фортрана, в некоторых трансляторах, кроме печати цифровых констант, инструкция PAUSE имеет замечательную возможность печатать любой текст, написанный непосредственно после PAUSE и заключенный между двумя апострофами (').

Например, инструкция

PAUSE 'MØCKBA'

останавливая машину, заставляет ее печатать слово MØCKBA, а инструкция

PAUSE 'END ØF PRØGRAM'

печатает слова

END ØF PRØGRAM

Текст, печатаемый инструкцией PAUSE, может содержать любые буквы, цифры и символы из алфавита Фортрана, за исключением символа апострофа. Это ограничение связано с тем, что символ апострофа выбран в качестве граничных указателей текста, предназначенного для печати. Инструкция PAUSE 'MØC'KBA' отпечатает слово MØC, стоящее между двумя ближайшими апострофами.

В отличие от всех остальных фортрановских инструкций, которые могут присутствовать или отсутствовать в программе, инструкция END обязательна в каждой программе. Более того, место этой инструкции точно определено — инструкция END всегда является последней инструкцией программы.

Инструкция END изображается единственным словом — ключевым словом END, она не оказывает никакого влияния на работу машины — это неисполняемая операция (ср. с CØNTINUE). Роль инструкции END состоит в том, чтобы указать транслятору конец текста программы. Практически полнота программы обеспечивается тем, что в конце каждой колоды перфокарт, представляющей полную программу, добавляют перфокарту, на которой отперфорирована инструкция END.

Контрольные упражнения к главе III

3.1К. Среди десяти перечисленных ниже инструкций указать неправильные и объяснить ошибки:

```
1) IF (FØRTRAN) 2, 3, 5
2) GØ TØ (2, 3, 5) K
3) DØ 10 INDEX = 15, 13
4) IF (JØS — 3) 4, 5, 6
5) IF (NIL) 2, 3, 4, 5
6) DØ 2 A = 1, 100, 2
7) GØ TØ MØRE, (15, 14, 13, 12)
8) DØ 20 J = 1, 50, 2, 3
9) IF (X ** 2 — Y ** 2), 2, 2, 4
10) IF (L1.ØR.L2) X = 2.
```

3.2К. Ниже перечислены десять инструкций управления.

Для вычисления указанных в инструкциях выражений задаются значения участвующих в них переменных:

```
A = — 17.5
B = + 11.3
C = 6.2
I = 2
```

Для каждой из инструкций управления надо указать номер следующей исполняемой инструкции. В том случае, когда в вопросе задано несколько инструкций, указать передачу управления после первой из них:

```
1)    IF (A + B — C) 17, 18, 19
2)    GØ TØ 921
3)    IF (B) 19, 35, 72
4)    PAUSE 44
   34 GØ TØ 2000
   44 X = X + Y
5)    IF (B — C) 172, 2, 95
6)    GØ TØ (4, 12, 15, 4, 14, 14, 20), I
7)    GØ TØ (1, 2, 3, 4), I
8)    DØ 5 IND = 2, 100, 2
   21 X = Y * Z
9)    IF (I — 3) 10, 20, 30
10)   IF (I ** I) 1, 3, 5
```

3.3К. Составить таблицу e^x по приближенной формуле

$$e^x \approx 1 + x + \frac{x^2}{4} + \frac{x^3}{6} + \frac{x^4}{24}$$

для всех целых значений x от 1. до 100., разместив результаты в таблице TAB.

3.4К. Сосчитать, сколько раз будет выполнена каждая инструкция нижеприводимой программы к моменту останова по инструкции STØP.

```
    5 X = 0.
   10 DØ 500 I = 1, 100
   50 X = X + 1.0
   55 IF (X — 20.0) 101, 101, 100
  100 IF (X — 80.0) 102, 102, 103
  101 A (I) = 3.0 * X ** 2 + 4.0 * X
   75 GØ TØ 500
  102 A (I) = 5.0 * X ** 2 + 6.0 * X
   76 GØ TØ 500
  103 A (I) = 7.0 * X ** 2 + 8.0 * X
  500 CØNTINUE
 1000 STØP
```

3.5К. На окружности с центром в точке (XC, YC) задана дуга с координатами начальной и конечной точек — (XN, YN) и (XK, YK) соответственно.

Написать программу, присваивающую величине MØN номер четверти окружности, в которой находится начальная точка, а величине MØK — номер четверти конечной точки.

Если крайняя точка заданной дуги ((XN, YN) или (XK, YK)) совпадает с точкой, лежащей на границе четвертей, то соответствующей искомой величине (MØN или MØK) присваивается номер четверти, в которой находится ближайшая окрестность крайней точки, принадлежащая дуге. (Движение по дуге — против часовой стрелки).

ГЛАВА IV

ИНСТРУКЦИИ ВВОДА И ВЫВОДА

Из инструкций, описанных в двух предыдущих главах, уже могут быть составлены реальные программы решения достаточно сложных задач. Машина сможет работать по этой программе, выполняя арифметические инструкции или инструкции управления. Это основная работа машины. Однако, не менее важны ее функции связи с внешним миром, машина должна уметь читать данные и печатать результаты так же, как и выполнять вычисления.

Инструкциям ввода и вывода языка Фортран посвящена настоящая глава. Первый ее параграф, не включающий описания инструкций Фортрана, может быть, тем не менее, полезен читателю, интересующемуся возможностями подготовки внешних носителей информации на различных машинах.

§ 1. Рисунок внешних носителей информации

Данные для вычислений вводятся в машину с помощью разных носителей информации. Наиболее распространенный из них — это *перфокарта*, довольно часто для этой цели используется бумажная перфорированная лента. Магнитные ленты начали использоваться в качестве внешних запоминающих устройств и носителей вводимой информации почти одновременно с появлением машин. Сейчас к числу устройств, хранящих данные и вводящих информацию в память машины, присоединены магнитные диски. В некоторых специальных приложениях используются устройства, читающие машинописный или даже рукописный стилизованный текст, и устройства, воспринимающие графическую информацию со специальных экранов. Современные вычислительные машины используют стандартную 80-колонную

перфокарту размером 82 × 187 *мм*. Нанесенные на перфокарту прямоугольные отверстия пропускают электрические импульсы на читающие щетки, с которых эти импульсы поступают в машину. Систему отверстий на перфокарте называют ее *рисунком*. Рисунок перфокарты определяется перфорирующим оборудованием, приданным машине, и правилами перфорирования. Если перфокарты всех существующих вычислительных машин стандартны [1]), то рисунок перфокарты на разных машинах различен.

В настоящем параграфе рассмотрены рисунки носителей информации трех типов машин: американских машин серии IBM-360 и советских машин БЭСМ-6 и «Минск-22».

На рис. 13 представлен вид стандартной 80-колонной перфокарты. Каждая перфокарта несет информацию в двенадцати основных строчках. На рис. 13 видно, что только десять строк из двенадцати отмечены отпечатанными цифрами от 0 до 9 (две строчки с номерами колонок — ниже строки нулей и ниже строки девяток — являются вспомогательными и не входят в число основных двенадцати).

Две основные строчки, не отпечатанные на карте, это строчка 12 — самая верхняя строчка карты — и строчка 11, расположенная между строкой 12 и строкой нулей. Строчку 11 называют иногда *строкой* X.

1. Перфокарта IBM-360. В стандартной перфокарте используется 80 колонок, в каждой из которых может быть отперфорировано одно, два или три отверстия. Каждая колонка изображает один знак: букву, цифру или специальный символ. Разрешено отсутствие перфорации в любой из колонок или группе колонок.

Строчки перфокарты разделяются на две категории. Строчки 12, 11, или X, и строка нулей называются *внетекстовыми*, строки 0, 1, 2, ... 9 — это строчки *цифровые*.

---

[1]) В 1969 г. фирма IBM выпустила вычислительную машину IBM-3, предназначенную для малых и средних автономных промышленных предприятий, с нестандартной перфокартой. На втрое меньшей, чем у стандартных перфокарт, поверхности карты IBM-3 предусмотрено место для перфорации 96 и печати 128 букв, цифр и специальных знаков. Отверстия перфорирования круглые.

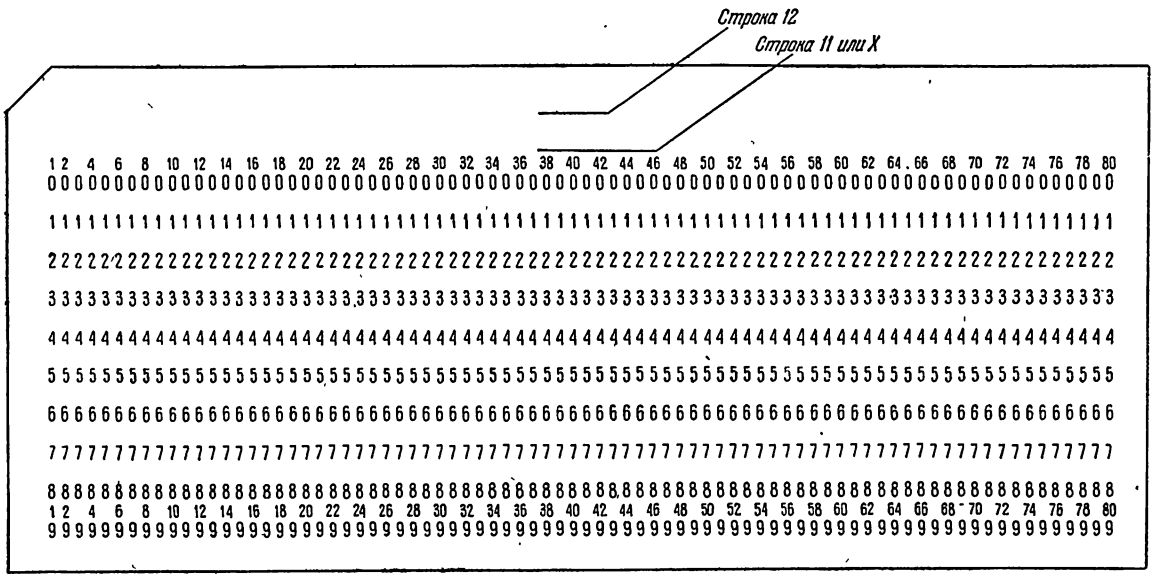
Строка 12
Строка 11 или X
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80

Рис. 13.

Рис. 14 показывает разделение строк на две категории. Строка нулей принадлежит одновременно обеим категориям.

Каждой цифре от 0 до 9 соответствует одна перфорационная пробивка — одно отверстие в перфокарте в соответствующей строке: например, на рис. 15 цифра 8 пробита в девятой колонке.

Каждой букве латинского алфавита от A до Z соответствуют две перфорации в одной колонке, одна из

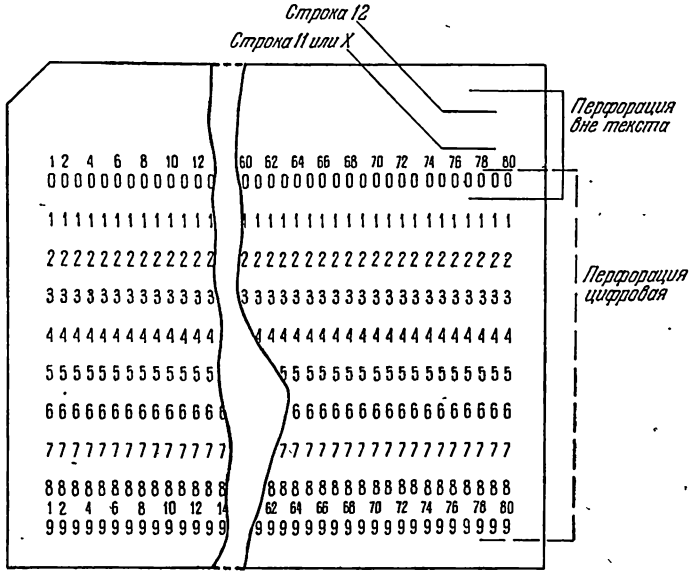

Рис. 14.

этих перфораций находится во внетекстовой строке, другая — в цифровой.

Часть специальных символов изображается одним перфорационным отверстием, другая часть — двумя, третья часть — тремя. Рис. 15 дает изображение букв, цифр и специальных символов.

Одна или несколько подряд расположенных колонок, представляющих одно число, слово или единое обозначение, составляют *зону перфокарты*. Число колонок, отводимое под каждую зону, соответствует самому длинному числу, слову, обозначению, записываемому в эту

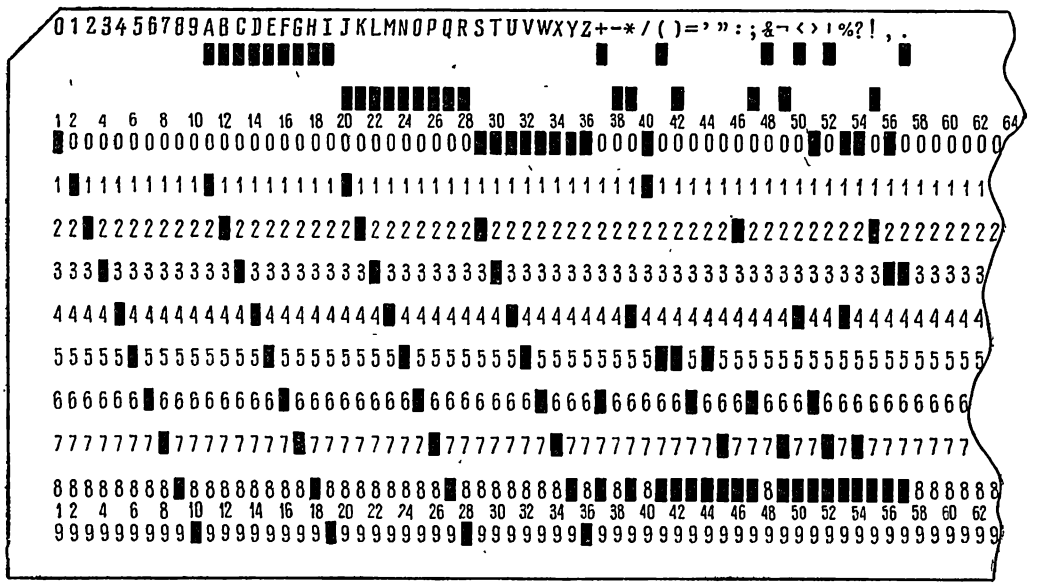
0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z + - * / ( ) = ' " : ; & ¬ < > ! % ? ! , .
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62

Рис. 15.

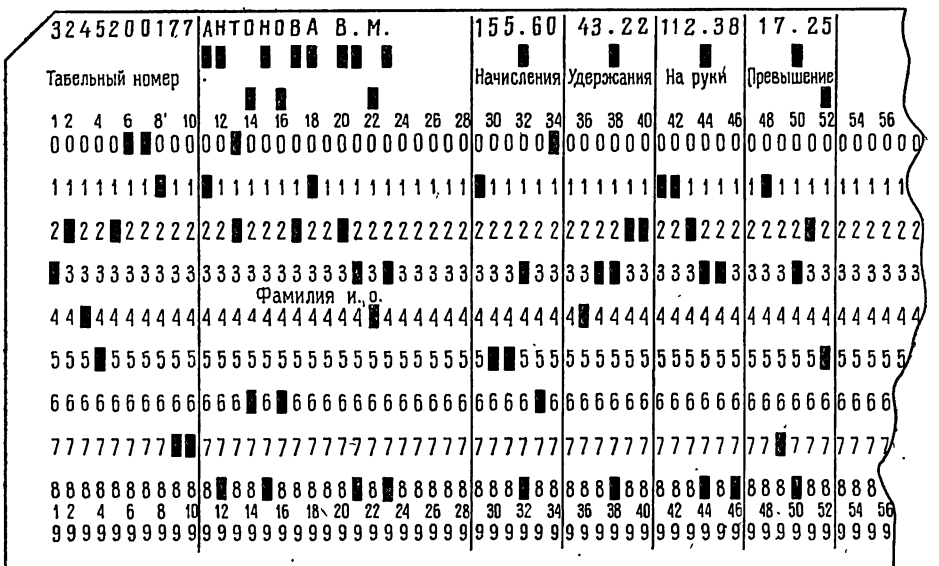
3245200177
АНТОНОВА В.М.
155.60
43.22
112.38
17.25
Табельный номер
Начисления
Удержания
На руки
Превышение
Фамилия и., о.

Рис. 16.

зону. Карта, на которую нанесена перфорация по зонам, изображена на рис. 16.

Отрицательное число на карте отмечается перфорацией знака «минус» (строка X) в крайней правой колонке зоны (над младшим разрядом зоны). Например, на рис. 16 отмечено отрицательное превышение текущей зарплаты над зарплатой прошлого месяца — «—17.25».

Вторая возможность записывать отрицательные числа на карте — предусматривать крайнюю левую колонку зоны для знака числа, записанного в зоне.

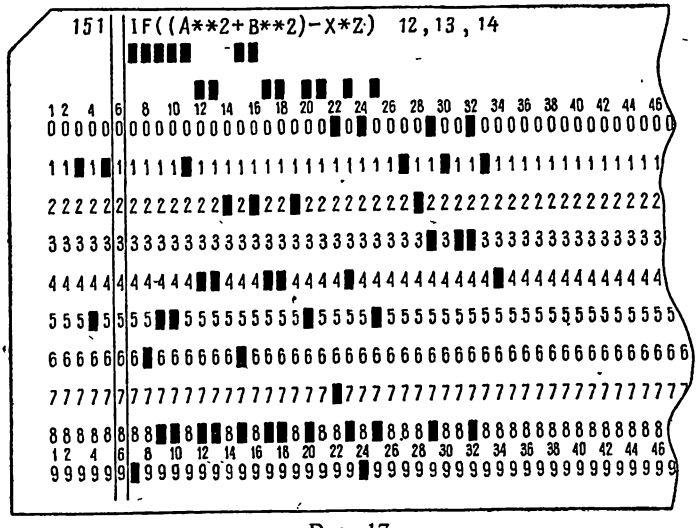

Рис. 17.

Одновременно с перфорированием карты перфорирующее устройство печатает над каждой колонкой выше двенадцатой строки символ, соответствующий перфорации колонки. Например, карта, на которую нанесена фортрановская инструкция

151 IF ((A**2 + B**2) — X*Z) 12, 13, 14

имеет вид, представленный рис. 17.

Одна перфокарта содержит всегда не более одной инструкции Фортрана.

2. Перфокарта БЭСМ-6. Рисунок перфокарты определяется не только техническими параметрами

вычислительной машины, но также и ее *операционной системой* — совокупностью программ, управляющих работой всех устройств машины и, в частности, устройств ввода и вывода. Для машины БЭСМ-6 созданы несколько операционных систем. В этом параграфе рассмотрен рисунок перфокарты, соответствующий одной из них.

Информация на перфокарте БЭСМ-6 записана не поколонно (как на IBM-360), а построчно. Каждый алфавитно-цифровой символ занимает восемь двоичных разрядов: старший разряд — контрольный, он служит для контроля четности кодов, остальные семь разрядов — содержательные. Семиразрядные двоичные коды символов алфавита Фортрана, используемые при пробивке перфокарт, приведены в таблице 5.

Таблица 5

Символ	Двоичный код	Символ	Двоичный код
0	0000000	N	1000101
1	0000001	Ø	0101110
2	0000010	P	0110000
3	0000011	Q	1000110
4	0000100	R	1000111
5	0000101	S	1001000
6	0000110	T	0110010
7	0000111	U	1001001
8	0001000	V	1001010
9	0001001	W	1001011
A	0100000	X	0110101
B	0100010	Y	0110011
C	0110001	Z	1001100
D	0111111	+	0001010
E	0100101	—	0001011
F	1000000	*	0011001
G	1000001	/	0001100
H	0101101	.	0001110
I	1000010	,	0011011
J	1000011	(	0010010
K	0101010	)	0010011
L	1000100	=	0010101
M	0101100		

Первые восемь колонок карты не используются для перфорации символов, одна из этих колонок несет контрольный разряд, остальные семь не содержат никакой

информации. Для перфорации символов отводятся 72 последних колонки карты. Одна строка 80-колонной стандартной перфокарты содержит 9 символов.

Воспринимаемая машиной информация содержится в восьми средних строках перфокарты — две самые верхние строчки, строки 12 и 11 и две самые нижние строчки, строки 8 и 9 не воспринимаются читающим устройством.

Таким образом, на одной перфокарте может быть помещено $9 \times 8 = 72$ алфавитно-цифровых символа. Этого достаточно, чтоб разместить на одной перфокарте символы одной строки фортрановского бланка. Так же как и на IBM-360, одна перфокарта БЭСМ-6 соответствует, вообще говоря, одной инструкции Фортрана. Однако, разряды бланка (с 73-го по 80-й), на которых записывается вспомогательная информация типа номеров строк, наименования программы и т. д., на перфокарте БЭСМ-6 не могут быть проперфорированы.

Рис. 18 изображает перфокарту БЭСМ-6 с перфорацией символов Фортрана в том порядке, в каком они приведены в таблице 5. Для наглядности пунктирно нанесены границы групп разрядов, изображающих символы.

На рис. 19 показан вид инструкции Фортрана

```
A = (X**2 + Y**2) — Z
```

нанесенной на перфокарту БЭСМ-6.

3. Перфолента «Минск-22». При обработке буквенно-символьной информации машина «Минск-22» использует в качестве основного внешнего носителя информации перфорированную бумажную ленту. «Минск-22» допускает, впрочем, и работу с перфокартами.

Вдоль всей длины перфоленты нанесена синхронизирующая дорожка, каждое отверстие которой отмечает строку на перфоленте. В строке перфоленты умещается один символ международного кода М-2. В нем символ изображается пятью двоичными разрядами. При этом 0 или 1 двоичного разряда соответствуют отсутствию или наличию круглого перфорационного отверстия в строке перфоленты. Таблицы 6а и 6б дают представление о коде М-2. Почти каждый код таблицы 6б характеризует не один, а пару символов. Одинаково перфорируются, например, буква Р (русская) и R (латинская), буквы Ы (русская) и Y (латинская).

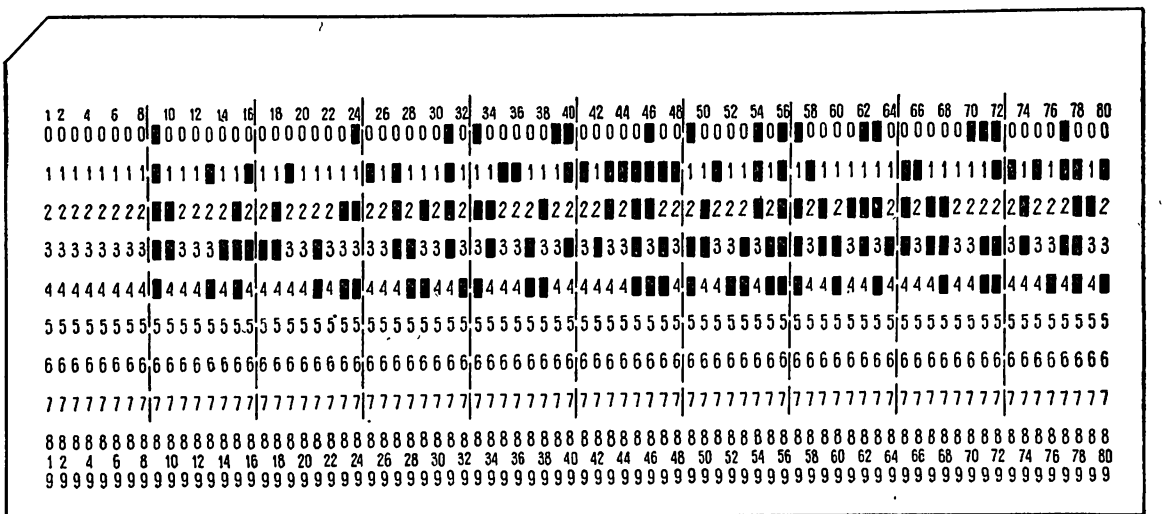

Рис. 18.

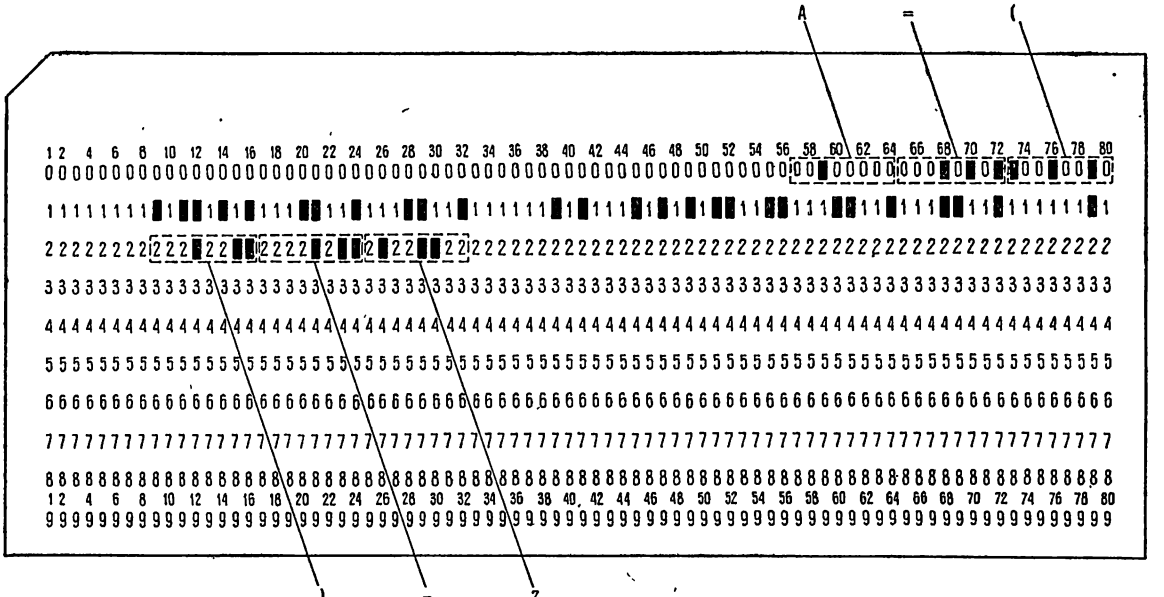
A
=
(
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80
)
-
Z

Рис. 19.

Таблица 6а

Символ	Двоичный код символа	Символ	Двоичный код символа
0	01101	+	10001
1	11101	—	11000
2	11001	/	10111
3	10000	,	00110
4	01010	.	00111
5	00001	(	11110
6	10101	)	01001
7	11100	=	01111
8	01100	*	10010
9	00011	'	10100

Таблица 6б

Русский символ	Латинский символ	Двоичный код	Русский символ	Латинский символ	Двоичный код
А	A	111000	П	P	101101
Б	B	110011	Р	R	101110
В	W	111001	С	S	110100
Г	G	101011	Т	T	100001
Д	D	110010	У	U	111100
Е	E	110000	Ф	F	110110
Ж	V	101111	Х	H	100101
З	Z	110001	Ц	C	101110
И	I	101100	Ш		001011
Й	J	111010	Щ		000101
К	K	111110	Ы	Y	110101
Л	L	101001	Ь	X	110111
М	M	100111	Э		010110
Н	N	100110	Ю		011010
О	O	100011	Я	Q	111101

Пяти позиций в строке перфоленты было бы не достаточно, чтоб закодировать 64 символа кода М-2 без использования регистров трех специальных кодов — «цифра», «буква русская», «буква латинская». Специальный код «цифра» обязывает читать все последующие строки перфоленты как коды таблицы 6а до тех пор, пока последовательность считываемых кодов не будет прервана специальными кодами «буква русская» или «буква латинская». Внутри машины каждому символу — букве, цифре, специальному символу — отведено поле не из

пяти, а из шести двоичных разрядов. Наличие кодов «буква» впереди последовательности строк обязывает машину во время ввода каждого пятиразрядного кода

Рис. 20.

с перфоленты добавлять в поле символа в памяти машины впереди строчки из пяти разрядов единицу старшего шестого двоичного разряда. Так, пробитая на перфоленте строчка рис. 20 будет воспринята как символ «=», если она находится в последовательности, возглавляемой кодом «цифра», как символ «Ж», если впереди стоит код «русская буква», и как символ «V», если впереди стоит код «латинская буква».

Рис. 21 изображает пробитую на перфоленте инструкцию

GØ TØ 217

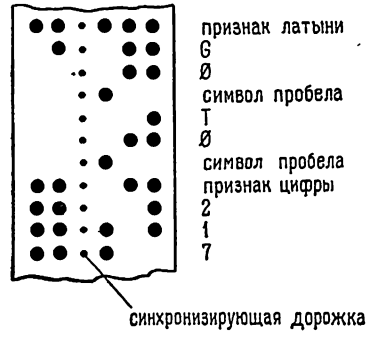

Рис. 21.

В некоторых трансляторах используются не все возможности кода М-2. Так, транслятор ИФВЭ (см. «Литературные указания» в конце книги) с Фортрана для машины «Минск-22» допускает только латинский алфавит.

§ 2. Инструкция ввода

В фортрановской программе определены лишь наименования переменных. Одна из возможностей присвоить этим переменным конкретные значения — использование арифметических инструкций типа

$$A = 3.14159.$$

Каждый раз в таком случае при исполнении программы переменным приписывается одно и то же определенное программой значение.

Кроме того, конкретные значения исходных данных — переменных — могут быть введены в машину с внешних носителей с помощью *инструкций ввода*. В этом случае один раз написанная программа может быть многократно исполнена, каждый раз с вновь вводимыми значениями данных. Инструкции ввода обеспечивают, таким образом, бóльшую гибкость использования программ.

В программе

```
A = 2.5
B = 3.2
C = 1.0
Y = A*X**2 + B*X+C
```

(4.1)

величины A, B и C определены как константы, их значения определяются приведенными в программе арифметическими инструкциями. В любом месте программы, где переменные A, B, C будут употреблены, сколько бы раз ни выполнялась программа, они будут иметь указанные конкретные значения.

Три арифметические инструкции программы (4.1) могут быть заменены одной инструкцией

```
READ(2, 1) A, B, C
```

Это пример инструкции ввода на Фортране. Она указывает, что величины A, B, C должны быть прочитаны с некоторого вводного устройства, например устройства, читающего перфокарты.

Рассматриваемая инструкция ввода формируется из слова READ, скобок и списка переменных. Содержимое скобок — две цéлые положúтельные константы, из которых первая — символический номер устройства ввода, а вторая — номер инструкции. Эти константы, так же как и элементы списка переменных, разделены запятыми. В инструкции

```
READ(2, 1) A, B, C
```

число 2 характеризует некоторое устройство ввода. Это число — код, символический номер устройства: он не связан непосредственно с каким-либо конкретным устройством. Для машины это число говорит, что переменные, перечисляемые указанной инструкцией READ, будут поступать в машину с устройства, которому присвоен

номер 2. Этот номер далее будет называться *номером канала*. В инструкции READ на месте номера канала может быть указана фиксированная переменная. В таком случае одна из предшествующих инструкций должна присваивать этой переменной конкретное значение:

```
NIL = 4
READ (NIL, 2) X, Y
```

Здесь инструкция READ обеспечивает ввод величин X и Y с канала номер 4.

Вообще говоря, один номер канала в разных ситуациях может соответствовать разным конкретным устройствам. Однозначное сопоставление номера канала с устройством осуществляется непосредственно перед исполнением программы: машина получает информацию, указывающую, что канал номер 2, например, соответствует магнитной ленте лентопротяжного механизма номер 4, а канал номер 3 — читающему (перфокарты) устройству номер 1. Такая, казалось бы, искусственная привязка номеров устройств к номерам каналов дает программисту бóльшие возможности гибкого использования внешних устройств. Вместо того чтобы зависеть от конкретных вводных и выводных устройств, программист легко может выбрать те из них, которые находятся в рабочем состоянии и свободны к моменту исполнения программы.

Список переменных в инструкции READ может иметь любую длину. Например, по инструкции

```
      READ (3, 2) A, B, C, D, E, F, G, H, Ø, P,
     *      Q, R, S, T, U, V, W, X, Y, Z
```

будут введены с канала номер 3 двадцать значений и присвоены соответственно перечисленным двадцати переменным.

Переменные в списке инструкции READ могут быть как действительными, так и целыми. Более того, в одной инструкции могут быть названы переменные обоих типов. Так, инструкция

```
READ(4, 1) A, I, B, J, C, K
```

является допустимой.

Переменные в списке инструкции READ могут быть индексированными. Индексы, если они используются, могут быть константами или переменными, конкретные

значения которых определены к моменту исполнения инструкции READ. Так, инструкция

```
READ(1,2) A, X(3), B, L(J)
```

прочитает с вводного устройства числа и присвоит их значения переменной A, третьему элементу массива X, переменной B и J-му элементу массива L (J должно быть определено к моменту исполнения инструкции READ).

Упражнение 4.1. Написать инструкцию, которая прочитает с перфокарт восемь чисел и присвоит их значения переменным A, B, C, D, E, F, G, H. Устройство ввода с перфокарт связано с каналом номер 6, а номер инструкции, о котором идет речь в инструкции ввода, равен 1.

Инструкция, номер которой упоминается в скобках инструкции READ, очень существенна при чтении данных. Эта инструкция называется FØRMAT. Она описывает форму числа и его положение на носителе информации. Почти каждая инструкция ввода использует инструкцию FØRMAT: если инструкция ввода указывает, *что* надо вводить, то инструкция FØRMAT указывает, *как* вводить данные. Этой инструкцией задается формат данных. Когда носителем служит перфокарта, инструкция FØRMAT указывает, какие колонки карт должны быть прочитаны и сколько чисел записано на карте.

Для конкретности изложения всюду ниже в качестве носителя информации будем иметь в виду 80-колонную перфокарту. Там, где будет важно подчеркнуть своеобразие записи на другом носителе, это будет отмечено особо.

С помощью инструкции READ можно вводить массивы величин. Инструкция READ выполняется так, что каждое отдельное прочитанное вводным устройством значение присваивается очередной указанной в списке переменной. Тогда, например, целое число, прочитанное в начале списка, может быть далее использовано в том же списке в качестве индекса, как в примере

```
READ(2,1) I,(A(I))                    (4.2)
```

Инструкция (4.2) читает с вводного устройства два числа. Первое из них — целое число I — после ввода становится индексом для второго.

В инструкции (4.2) можно отметить пару скобок, окаймляющую A (I). По правилам Фортрана такая пара скобок должна выделять индексированные величины, если индекс указан в списке величин инструкции READ. Если есть необходимость прочитать два индекса I и J и две зависящие от этих индексов величины X (I) и Y (J), то это можно сделать с помощью инструкции

```
READ(1, 3) I, J, (X(I), Y(J))
```

Если в списке величин инструкции READ есть индексированная переменная, то присутствие индекса в том же списке совершенно не обязательно: он может быть определен любой ранее исполняемой инструкцией. В ниже приведенном примере инструкция DØ определяет индекс величины в инструкции READ

```
    DØ 5 I = 1,8
5   READ(1, 2) X(I)
```
(4.3)

В цикле DØ (4.3) инструкция READ исполняется 8 раз, каждый раз при новом значении индекса I. Тем самым осуществляется присвоение элементам списка X (I) восьми величин, прочитанных инструкцией READ. В примере (4.3) величина X (I) указана без «лишних» скобок — индекс определен вне инструкции READ.

Упражнение 4.2. Записать инструкции, реализующие чтение пятидесяти перфокарт, на каждой из которых записаны три величины. Первую из этих величин надо записывать в массив X (I), вторую — в Y (I), третью — в Z (I). Предполагается, что необходимая инструкция FØRMAT имеет номер 25, а читающему устройству соответствует канал номер 4.

Использование инструкции READ — один из способов ввода списков, таблиц, массивов данных. Более широко при составлении фортрановских программ для этого используется прием, называемый *автоматической индексацией*.

При автоматической индексации в списке переменных инструкции READ указывается определение индекса, такое же как в инструкции DØ. Пример использования автоматической индексации:

```
READ (5, 3) (X(I), I = 1, 10)
```
(4.4)

Термином «автоматическая индексация» называют и собственно конструкцию — открывающая скобка,

индексированная переменная, определение индекса, закрывающая скобка. В примере (4.4) автоматическая индексация — это выражение вида

```
(X(I), I = 1,10)
```

Инструкция (4.4) эквивалентна

```
      READ (5, 3) X(1), X(2), X(3), X(4), X(5),
     *            X(6), X(7), X(8), X(9), X(10)
```

Несомненно, запись (4.4) более привлекательна. Автоматическая индексация позволяет совместить возможности двух инструкций — READ и DØ — в одной. Все возможности инструкции DØ при этом сохраняются: в определении индекса можно указывать третью величину — шаг, позволяющий считывать каждое второе, каждое третье и т. д. значение; можно в качестве нижней, верхней границы индекса и шага использовать не константы, а целые переменные.

В примере

```
READ (12, 2) L, (X(I), I = L, 100)
```

инструкция ввода будет заполнять таблицу X(I) не с первого элемента, а с элемента номер L, где L — первое прочитанное инструкцией READ значение.

Все правила, касающиеся расстановки символов в определении индекса, сохраняются и в инструкции READ с автоматической индексацией. Более того, эти правила дополняются необходимостью простановки запятой между индексированной переменной и определением индекса, а также окаймления скобками индексированной переменной вместе с определением индекса.

Инструкция

```
READ (7, 2) A(I) I = 1,15
```

записана неверно. Действительно, во-первых, отсутствуют скобки, охватывающие индексированную переменную A (I) вместе с определением индекса I = 1,15 и, во-вторых, нет запятой между индексированной переменной и определением индекса. Правильно эта инструкция должна была бы быть записана

```
READ (7, 2) (A (I), I = 1, 15)
```

Упражнение 4.3. Написать инструкцию, позволяющую ввести L чисел с канала номер 3 в таблицу X (J). Соответствующая инструкция FØRMAT имеет номер 10.

Автоматическая индексация может быть одним из элементов списка переменных инструкции READ. Допустимая инструкция

READ (4, 2) A, B, (C(I), I = 1,4), D, E

прочитает восемь значений, присвоив их соответственно величинам

A, B, C(1), C(2), C(3), C(4), D, E

Возможны две, три и более автоматических индексаций в списке переменных одной инструкции READ. При этом каждая автоматическая индексация выделяется своей парой скобок. Использование нескольких автоматических индексаций имеет смысл в том случае, когда в каждой из них употреблен различный индекс. Дело в том, что по нескольким переменным, зависящим от одного индекса, автоматическая индексация может быть записана однократно, одной парой скобок. Так, инструкция

READ (2, 4) (X(I), Y(I), I = 1, 10) (4.5)

читает с канала номер 2 двадцать значений, присваивая их соответственно переменным X(1), Y(1), X(2), Y(2),... ..., X (10), Y (10).

Однако если с канала номер 2 поступают сначала все элементы массива X, а затем все элементы массива Y, то их чтение следует организовать инструкцией

READ (2, 4) (X(I), I = 1, 10), (Y(I), I = 1,10) (4.6)

Пример инструкций (4.5) и (4.6) наглядно показывает две возможности ввода массивов данных. Выбор той или иной возможности может зависеть от того, как организована последовательность на входе.

Иногда возникает потребность читать индексированные переменные автоматической индексации вместе с постоянными величинами. Например, перечень табельных номеров рабочих и их зарплат может сопровождаться постоянной для всех элементов перечня величиной — номером цеха.

Если назвать табельные номера рабочих N (I), их зарплаты S (I), а номер цеха K, то для ввода сведений о 200 рабочих цеха с номером K можно воспользоваться инструкцией

READ (5, 1) (N(I), S(I), K, I = 1, 200)

Инструкция READ может обрабатывать (читать, переносить с одного носителя информации на другой) массив двух или трех измерений. Автоматическая индексация содержит в этом случае два или три определения индекса.

Для того чтобы прочитать с перфокарт (канал номер 2) все элементы матрицы A, состоящей из 20 столбцов и 10 строк, надо использовать инструкцию

READ (2, 1) ((A(I, J), I = 1,10), J = 1,20)) (4.7)

Все восемь скобок и шесть запятых этой инструкции обязательны. Инструкция (4.7) разместит прочитанные элементы в таком порядке:

A(1, 1), A(2, 1), A(3, 1), ... , A(10, 1),
A(1, 2),A (2, 2), ... , A(10, 2), ... , A(1, 20),
A(2, 20), ... , A(10, 20)

По существу инструкция READ повторяет для каждого значения внешнего индекса J обычную автоматическую индексацию по внутреннему индексу I

A(I, J), I = 1, 10

Форма автоматической индексации для двухразмерных массивов может быть использована и при нескольких двухразмерных переменных. Например, инструкция

READ (3, 4) ((A(I, J), B(I, J), I = 1, 10), J = 20)

читает два числа для любой комбинации индексов. В равной мере одна инструкция ввода может содержать несколько автоматических индексаций, обычных или с двумя индексами. Например, инструкция

READ (1, 2) L, K, ((X(I), I = 1, L),
* (B(I, J), I = 1, L), J = 1, K)

читает сначала две простые переменные, затем линейный массив X (I) и, наконец, матрицу B (I, J).

Инструкция READ имеет одинаковый вид для любого носителя информации — перфоленты, магнитной ленты,

магнитных дисков. Данные могут быть расположены на лентах и дисках таким же образом, как они записаны на картах. Термин «представление карты», используемый в этой главе, относится к числам, записанным на лентах или дисках в такой же форме, как они были бы записаны на картах.

Указать, с каким носителем информации, с каким устройством ввода работает та или иная инструкция READ, можно лишь зная номер канала и выполненную до начала работы программы привязку номеров каналов и номеров вводных устройств.

Упражнение 4.4. 100 чисел записаны на магнитной ленте. Написать инструкцию, которая переписала бы все их на нечетные места массива Y(J) в памяти машины. Магнитной ленте, содержащей исходный массив, соответствует канал номер 5; соответствующая инструкция FØRMAT имеет номер 10.

§ 3. Инструкция FØRMAT

В предыдущем параграфе многократно упоминалась инструкция FØRMAT, которая пока еще не определена строго. Это неисполняемая инструкция, она лишь снабжает инструкции ввода и вывода информацией о форме вводимых данных и выводимых результатов.

Каждая инструкция FØRMAT должна быть упомянута в связи по крайней мере с одной инструкцией ввода или инструкцией вывода. Поэтому инструкция FØRMAT обязательно имеет номер инструкции.

Инструкция FØRMAT может указать количество чисел, записанных на одной перфокарте, форму их представления, количество значащих цифр числа и знаков после запятой.

Например, если необходимо прочитать пять записанных на одной перфокарте восьмизначных чисел с запятой, отделяющей в них трехзначную целую часть, то вся эта информация может быть указана единственной инструкцией

```
3 FØRMAT(5F8.5)
```

Инструкция FØRMAT состоит из ключевого слова FØRMAT и пары скобок, содержащих информацию о рисунке карты (или ее представления). Информация о рисунке карты — буквенно-цифровой код. Буквенная

часть кода указывает форму представления числа. Для действительных чисел с фиксированной запятой используется буква F. Эту букву сопровождают два целых числа, разделенных точкой. Они указывают количество колонок, отводимых в карте для изображения числа, и положение десятичной точки.

В инструкции

3 FØRMAT (F8.5) (4.8)

указано:

F — вводимое число представлено в форме с фиксированной запятой;

8 — восемь первых колонок перфокарты отведены под изображение числа;

5 — пять из восьми колонок отведены под дробную часть числа (три заняты целою частью);

3 — номер инструкции FØRMAT.

Восьмерка в примере (4.8) говорит о том, что в зоне перфокарты для записи числа предусмотрено восемь колонок. Поэтому первое из чисел, следующих за буквой F, называют *длиной зоны*. Если в инструкции FØRMAT указана длина зоны, то количество символов записываемого в нее числа не должно превосходить длину зоны.

Второе число после F в инструкции FØRMAT — указатель положения десятичной точки — необходим в инструкции FØRMAT. Однако, этот указатель учитывается вычислительной машиной лишь в том случае, когда десятичная точка числа не перфорирована на карте, т. е. не указана в записи числа, и ее следует предполагать при вводе на месте, определяемом инструкцией FØRMAT. Десятичная точка, нанесенная на перфокарту, имеет преимущество перед указателем положения десятичной точки в инструкции FØRMAT: при наличии точки на карте машина учитывает лишь длину зоны, десятичная же точка отмечается там, где это указывает перфорация.

На рис. 22 изображена перфокарта, на которой проперфорировано восьмизначное число 24536889.

При желании видеть это число с пятью знаками после запятой можно при вводе его в машину с помощью инструкции READ воспользоваться инструкцией

3 FØRMAT (F8.5)

Десятичная точка на карте не перфорирована, поэтому ввод будет осуществлен в соответствии с информацией инструкции FØRMAT — пять знаков в дробной части.

В рассматриваемом примере видно, что число занимает самое левое положение на карте. Инструкция FØRMAT всегда описывает карту, начиная с первой колонки слева направо.

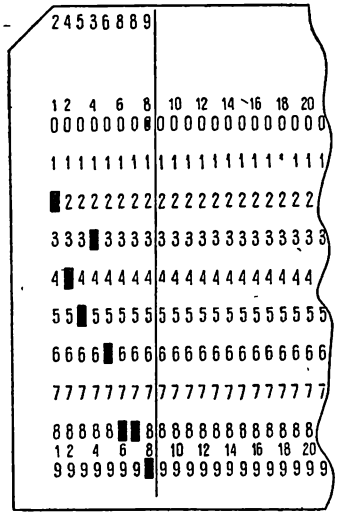

Рис. 22.

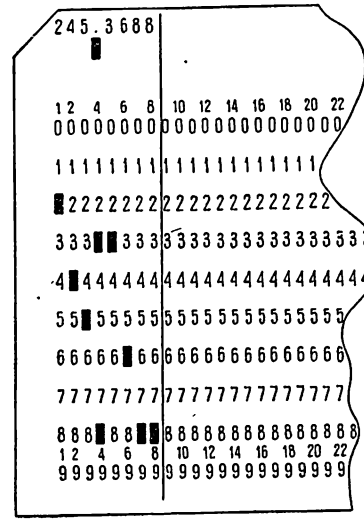

Рис. 23.

Если десятичная точка проперфорирована, то длина зоны включает и колонку десятичной точки.

На рис. 23 изображено число 245.3688. Оно записано в зоне той же длины, что и на рис. 22, однако, на сей раз десятичная точка проперфорирована на карте. В этом случае инструкции

```
3 FØRMAT(F8.4)
3 FØRMAT(F8.1)
3 FØRMAT(F8.3)
```

дадут одинаковый результат — в машину будет введено число 245.3688 в соответствии с перфорацией на карте независимо от величины указателя положения десятичной точки.

Очень важно не путать форму чисел, проперфорированных на карте, и соответствующих величин, записанных в инструкциях программы.

Константа Фортрана должна обязательно иметь десятичную точку, если эта константа представлена в форме с фиксированной запятой. Число в форме F, читаемое машиной, может быть записано на карте и без десятичной точки — в этом случае положение десятичной точки определяется инструкцией FØRMAT.

Упражнение 4.5. 1) Написать инструкцию FØRMAT, позволяющую прочитать при вводе число в форме с фиксированной запятой в первых шести колонках перфокарты. В дробной части числа — две цифры. Инструкция FØRMAT имеет номер 1;

2) то же задание для числа из 10 цифр с 3 знаками после запятой;

3) то же задание для числа из 5 цифр. Дробная часть отсутствует.

Инструкция FØRMAT, используемая, как указывалось выше, вместе с инструкцией READ, полностью описывает перечень и форму вводимых величин.

Если канал номер 3 привязан к читающему (карты) устройству, то инструкции

```
    2 FØRMAT (F5.3)
      READ (3, 2) X
```

читают с перфокарты пятизначное число с фиксированной запятой и тремя знаками в дробной части и присваивают его переменной X.

Содержимое скобок инструкции FØRMAT полностью описывает перфокарту: если в инструкции указан только один код F5.3, это значит, что на карте проперфорировано только одно число. Но на карте может быть размещено не одно, а несколько чисел. Например, инструкция

```
    1 FØRMAT (F6.2, F7.3, F5.4)
```

говорит о том, что на перфокарте записаны три действительных числа в форме с фиксированной запятой — одно в колонках с первой по шестую, второе — в колонках с седьмой по тринадцатую, и третье — в колонках с четырнадцатой по восемнадцатую. Единственное ограничение на количество записываемых на одной карте чисел — сумма длин их зон не должна превосходить

восьмидесяти (числа колонок на перфокарте). Коды внутри скобок инструкции FØRMAT разделяются запятыми.

Если несколько кодов в скобках инструкции FØRMAT одинаковы, они могут быть записаны один раз с целым коэффициентом перед буквой F — числом повторений. Например, инструкция

```
1 FØRMAT (3F5.2)
```

эквивалентна инструкции

```
1 FØRMAT (F5.2, F5.2, F5.2)
```

Инструкция ввода READ определяет, сколько всего величин должно быть прочитано, тогда как инструкции FØRMAT определяет, сколько значений содержит одна перфокарта. Таким образом, сочетание этих двух инструкций может определить количество читаемых перфокарт или представлений карты.

Пусть канал номер 3 поставлен в соответствие лентопротяжному механизму. Тогда инструкция

```
READ (2, 3) (T(I), I = 1, 90)          (4.9)
```

должна будет прочитать с ленты 90 чисел. Очевидно, столько чисел не может быть размещено на одной перфокарте или представлении карты.

Используемая вместе с (4.9) инструкция FØRMAT описывает рисунок одной карты (представления карты), например, так:

```
2 FØRMAT (16F5.1)                      (4.10)
```

Инструкция (4.10) предусматривает расположение в одном представлении карты шестнадцати пятизначных чисел с фиксированной запятой. Это значит, что при чтении массива T(I) будет использовано [90/16] + 1 = 6 представлений карт. Пять из этих представлений будут иметь по 16 чисел, в последнем будет размещено лишь десять: предписания инструкции (4.10) повторяются от одного представления карт к другому до тех пор, пока массив T(I), указанный в инструкции READ, не будет полностью исчерпан.

Упражнение 4.6. Написать инструкцию FØRMAT, представляющую записанные на одной перфокарте числа, которые вводятся инструкцией READ (2,2) A, B, C, D. Каждое из чисел, записанных в форме F, требует 10 колонок и имеет 4 знака после запятой.

Если инструкция FØRMAT определяет меньшее количество колонок, чем их записано на перфокарте, то остальные, не описанные инструкцией FØRMAT колонки, не могут быть прочитаны машиной. Например, инструкция

```
1 FØRMAT (3F20.5)
```

описывает первые шестьдесят колонок перфокарты. Перфорация в последних двадцати колонках карты не имеет значения, поскольку они не учитываются машиной.

Упражнение 4.7. Написать инструкции ввода READ и FØRMAT, записывающие в таблицу R(I) значения платежей в чеках. Количество платежей предварительно определено программой. Максимальная сумма, указанная в чеке, — 1000 рублей. Необходимо ввести значения платежей в машину с точностью до копеек. Разместить на одной перфокарте максимально возможное число платежей.

Действительные числа, вводимые в машину, могут быть записаны и в показательной форме — форме с плавающей запятой. На рис. 24 изображена перфорация чисел 180000. = 1.8*10**5=1.8E5 (в колонках с первой по пятую) и —180000. = —1.8*10**5 = —1.8E.5 (в колонках с шестой по одиннадцатую).

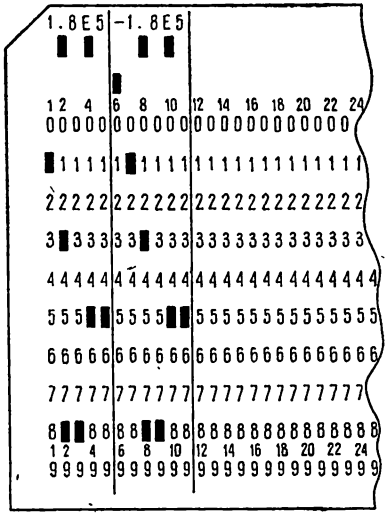

Рис. 24.

Когда инструкция FØRMAT описывает числа в форме с плавающей запятой (в форме E), она имеет тот же вид, что был определен ранее: сначала следует буква E, затем два числа, разделенные точкой, из которых первое является длиной зоны, а второе указывает количество знаков после запятой. Число 1845.61 в форме E имеет вид 184561E — 02, тогда длина занимаемой им на карте зоны равна 10. Для его описания необходима инструкция

```
1 FØRMAT (E10.2)
```

В этом примере число было проперфорировано на карте без десятичной точки. Как и в случае чисел в форме F, десятичная точка, пробитая на перфокарте, имеет приоритет перед записью инструкции.

Если на перфокарте проперфорировано число 12345.6E—2 (вместе с десятичной точкой), а инструкция FØRMAT определяет его как E10.2, то при вводе в машину оно будет прочитано как 123.456.

Упражнение 4.8. Как будут прочитаны машиной с помощью инструкции

```
1 FØRMAT (E10.5)
```

числа, проперфорированные на картах в виде

1) + 34567.0E2
2) + 34.567E—3
3) — 345.6789E4
4) 3456.700E0
5) + 345.678E+3
6) 345678.E—01

Инструкция FØRMAT допускает употребление нескольких, в том числе одинаковых кодов E:

```
1 FØRMAT (4E6.2)
```

Числа в формах E и F могут одновременно присутствовать на перфокарте или в представлении карты. Например, инструкция

```
1 FØRMAT (F4.2, 5F6.1, 3E8.0, E10.2)
```

позволяет прочитать на входе сначала одно число в форме F в первых четырех колонках, затем пять чисел в форме F в колонках с пятой по тридцать четвертую, потом три числа в форме E в колонках с тридцать пятой по пятьдесят восьмую, и, наконец, еще одно число E, записанное в колонках с пятьдесят девятой по шестьдесят восьмую включительно.

Буква F в кодах инструкции FØRMAT означает действительное число в форме с фиксированной запятой, буква E — действительное число в форме с плавающей запятой. Для кодирования целых чисел применяется буква I.

Инструкция FØRMAT, описывающая целое число, содержит в скобках букву I, за которой следует целое число — длина зоны. Инструкция

```
1 FØRMAT (I10)
```

указывает, что целое число (число в форме I) записано в первых десяти колонках перфокарты.

Как и в случае чисел в формах F и E, допустима запись нескольких чисел в форме I (в том числе одинаковой длины) на одной карте, а также запись на одной карте чисел во всех трех формах — F, E, I.

В третьей главе при рассмотрении арифметических инструкций мы видели, что наличие в левой и правой частях одного выражения переменных и констант в разных формах не приводит к неприятностям — значение правой части арифметического выражения преобразуется автоматически в форму переменной, стоящей в левой части. В инструкциях ввода несовпадение формы переменной инструкции READ с ее кодом в инструкции FØRMAT означает ошибку — программист должен быть внимателен при составлении инструкций ввода.

Упражнение 4.9. Для каждого рабочего определен его табельный номер N (целое шестизначное число) и его месячная заработная плата Z (действительное число). Зарплаты указаны в рублях с точностью до копеек ($Z < 1000$ рублей). Одному рабочему соответствует одна перфокарта с табельным номером N и зарплатой Z. Рабочие распределены по трем участкам — с этим связаны три индекса у индексированных переменных $N_\bullet$ и Z.

На слесарном участке S — 102 рабочих:

$$N(IS),\ Z(IS),\ IS = 1, 102$$

на прессовом P — 126 рабочих:

$$N(IP),\ Z(IP),\ IP = 1, 126$$

на токарном T — 95 рабочих:

$$N(IT),\ Z(IT),\ IT = 1, 95$$

Написать инструкции ввода всех перфокарт (номер канала 2), сгруппированных по участкам S, P, T. Зарплаты Z (IS) и Z (IP) записаны в форме F, зарплаты Z (IT) — в форме E.

Всюду выше инструкция FØRMAT описывала каждый раз рисунок одной перфокарты. Однако, возможности инструкции могут быть расширены.

Символ / используется в инструкциях FØRMAT, чтоб определить группы кодов, описывающих рисунки различных карт. Например, инструкция

```
1 FØRMAT (5F10.4/4F12.3)                    (4.11)
```

определяет два различных рисунка карт: следуя этой инструкции, первая карта будет прочитана по коду

5F10.4, следом за ней вторая по коду 4F12.3. Инструкции FØRMAT с символом / используются тогда, когда последовательно расположенные карты имеют разные рисунки. В нашем случае, если группа данных на входе состоит более чем из девяти чисел, третья карта снова будет читаться по коду 5F10.4 и т. д. Если по инструкции (4.11) надо прочитать 200 карт, то 100 из них будут прочитаны по коду 5F10.4 и 100 — по коду 4F12.3.

У знака / есть еще одна интересная возможность: можно с помощью одной инструкции FØRMAT прочитать одну, первую, карту по одному коду и целую группу последующих — по другому. Для этого второй код должен быть окаймлен дополнительными скобками.

Например, инструкция

```
1 FØRMAT (I5/(4F10.5))
```

прочитает на первой карте одно пятизначное число в форме I, на второй карте будут прочитаны четыре числа в форме F. Если карт достаточно много, то на третьей карте будут прочитаны еще четыре числа в форме F, на четвертой — снова четыре числа в форме F и т. д. Первый рисунок карты использован лишь для одной карты, второй — для многих, пока не исчерпан список всех читаемых инструкцией READ переменных.

Несколько усложненный пример инструкции FØRMAT:

```
1 FØRMAT(2F6.3/I20/(5E10.2))                    (4.12)
```

По этой инструкции сначала читается карта по коду 2F6.3, затем карта по коду I20 и, наконец, серия карт по коду 5E10.2. Можно легко сформулировать общее правило повторения кодов инструкции FØRMAT при чтении длинных и сложных списков: машина просматривает все коды до конца инструкции FØRMAT, при этом выполняется по одному разу ввод карты по каждому из кодов инструкции FØRMAT; затем следуют повторения вводов по коду, содержащемуся в последней паре скобок инструкции FØRMAT.

Это правило не сложно, так как использование скобок внутри скобок инструкции FØRMAT разрешено только один раз, при этом код, записанный во внутренних скобках, может быть лишь самым правым кодом инструкции FØRMAT.

Не расходится с этим правилом и еще один способ использования внутренних скобок, иллюстрируемый примером

```
1 FØRMAT (3 (F4.2, I5))
```

который означает допустимую сокращенную запись инструкции

```
1 FØRMAT (F4.2, I5, F4.2, I5, F4.2, I5)
```

Упражнение 4.10. Первые 200 элементов массива T (I) должны быть введены с перфокарт (канал номер 2). На каждой карте записано 10 чисел в форме E. Каждое из чисел записано в восьми колонках и имеет два знака после запятой. Последующие 800 элементов массива T (I) вводятся с магнитной ленты (канал номер 5) с использованием той же инструкции FØRMAT. Написать инструкции, осуществляющие оба эти ввода.

§ 4. Инструкция вывода чисел

Вычислительная машина существует и работает в окружающей ее среде, по заданиям работающих на ней программистов. Связь с этой внешней средой в равной степени с рассмотренными в предыдущих параграфах инструкциями ввода обеспечивается инструкциями вывода. Инструкции вывода позволяют отпечатать полученные в ходе исполнения программы результаты в виде, удобном для непосредственного практического использования потребителем, передать эти результаты другой машине или переписать их на любой внешний носитель информации.

Количество и разнообразие типов выводных устройств варьируется от модели к модели, от одной конкретной машины к другой, однако, как правило, число устройств вывода на каждой машине превосходит число вводных устройств.

Прежде всего, возможен вывод на все те носители информации, которые были рассмотрены в двух предыдущих параграфах: перфорация результатов на карты, на перфорированную бумажную ленту, запись результатов на магнитную ленту, барабаны или диски для последующего их использования на другой или этой же машине.

Однако, следует выделить особо печатающие устройства, которые воспроизводят результаты на бумаге, так

как в этом случае результаты могут быть непосредственно без дополнительной расшифровки прочитаны человеком. Это либо быстропечатающие устройства с выводом на бумажную ленту, либо алфавитно-цифровые печатающие устройства, либо, наконец, пишущие машинки, соединенные с вычислительной машиной [1]).

Инструкции Фортрана дают возможность использовать любое из выводных устройств, имеющихся в оборудовании машины. Аналогично номерам входных каналов в инструкциях ввода, в инструкциях вывода указаны номера выходных каналов, которым перед исполнением программ могут быть присвоены конкретные номера выводных устройств. Для определенности в приводимых ниже примерах будем предполагать, что алфавитно-цифровое печатающее устройство привязано, например, к каналу номер 1, а магнитная лента — к каналу номер 4.

Универсальной *инструкцией вывода* является инструкция WRITE. Она состоит из ключевого слова WRITE, скобок, содержащих номер канала и номер инструкции FØRMAT, и, наконец, списка выводимых переменных и констант, который формируется в основном так же, как аналогичный список инструкции READ.

Инструкция

WRITE (1, 3) A

выводит на печатающее устройство (канал номер 1) по формату, определенному инструкцией FØRMAT (инструкция номер 3) значение величины A.

Аналогично инструкции READ на входе, инструкция WRITE нуждается в инструкции FØRMAT для описания форм выводимых результатов: сама по себе инструкция WRITE указывает лишь перечень величин в том порядке, в котором они должны быть выведены из машины на внешний носитель информации. Правила формирования перечня выводимых величин полностью аналогичны уже известным правилам формирования списка инструкции READ: те же правила пунктуации — расстановки запятых и скобок, то же допущение любой формы

[1]) Сейчас уже используются разнообразные специальные выводные устройства: самописцы, телевизионные экраны — дисплеи, микрофильмирование, устройства выдачи справочной информации голосом и т. д.

переменных и автоматической индексации. Например, инструкция

```
WRITE (1, 2) I, J, K, A, (X(L), L = 1, 10)
```

печатает 14 величин: три числа в форме I, одно действительное число (его форма — F или E — может быть определена лишь в инструкции 2) и 10 действительных чисел массива X (L).

Инструкция FØRMAT используется вместе с инструкцией WRITE по тем же правилам, как она использовалась с инструкцией READ.

В двух предыдущих параграфах был сделан акцент на использование перфокарт как основного носителя вводимой информации. Точно так же в настоящем параграфе главное внимание будет уделено печатающим устройствам. Описания инструкций вывода на печать могут быть заменены соответствующими описаниями вывода на магнитную ленту или перфорацию карт простой заменой номера канала в инструкции WRITE.

На печатающее устройство результаты выводятся по строкам. Структуру строки, ее «рисунок» определяет инструкция FØRMAT.

Пара инструкций

```
5 FØRMAT (F6.5)
  WRITE (1, 5) X, Y, Z
```

печатает три строки, которые содержат соответственно значения X, Y, Z с шестью знаками, из которых пять — после запятой.

Количество печатаемых величин, определяемое инструкцией WRITE, и количество величин в одной строке, определяемое инструкцией FØRMAT, позволяет определить количество печатаемых строк. Например, по инструкциям

```
12 FØRMAT (5F7.5)
   WRITE (1, 12) (X(I), I = 1, 1000)
```

будут отпечатаны 200 строк, в каждой из которых будут размещены по 5 значений массива X (I).

Максимальное число символов в строке зависит от типа печатающего устройства, которым оснащена вычислительная машина. Различные типы отечественных и иностранных машин имеют алфавитно-цифровые уст-

ройства с максимальной длиной строки в 120, 128, 132 или 144 символа. Возможности печатающего устройства конкретной машины должны быть учтены при составлении инструкций вывода на печать в фортрановских программах: сумма длин зон всех величин, предусмотренных для печати в одной строке, не должна превышать максимально допустимого конкретным устройством числа символов в строке [1]).

При работе с числами в формах F и E на входе не было необходимости перфорировать десятичную точку на перфокарте — она могла быть учтена в инструкции FØRMAT. На печатающем устройстве десятичная точка печатается всегда, ее положение определяют коды F- и E инструкции FRØMAT.

Вычислительная машина печатает столько символов, сколько предписано инструкцией FØRMAT. Если же необходимо проанализировать результат и отпечатать лишь значащие цифры, это может быть сделано лишь специально написанной программой. Если известно, что полученный результат содержит два верных знака после запятой, то нет никакого смысла использовать для печати инструкцию FØRMAT (F7.5), печатающую пять знаков после запятой.

Печатаемое число размещается в строчке смещенным вправо в зоне, для него предназначенной: младший разряд числа — крайний правый разряд зоны. Это дает возможность печатать в строке несколько чисел с пробелами.

Например, 12 чисел списка A (I)

A(1) = —16.245 A(2) = 349.271 A(3) = 1052.61
A(4) = 252. A(5) = —17. A(6) = 8765.432
A(7) = 0.00181 A(8) = —0.1534621 A(9) = 2221.1183
A(10) = 4500. A(11) = 202.471 A(12) = —2.0909

будут отпечатаны инструкциями

```
10 FØRMAT (4F9.4)
   WRITE (1, 10) (A (I), I = 1, 12)
```
(4.14)

[1]) Максимальная длина строки печатающих устройств БЭСМ-6 и «Минск-22» — 128 символов. На различных моделях IBM-360 существуют устройства со 120, 132 и 144 символами в строке. Точно так же существуют ограничения на максимальную длину пишущих машинок, подключенных к ЭВМ.

так, как показано на рис. 25. Пробелы между числами отмечены условно на рисунке символами «b». В действительности на месте символов «b» ничего нет.

На этом рисунке видно, что десятичная точка занимает одно и то же положение в изображении напечатанных чисел независимо от их записи в памяти: это определено инструкцией FØRMAT в (4.14). Число A (1) отпечатано с четвертым нулем после запятой, хотя в списке (4.13) оно записано лишь с тремя знаками после запятой:

```
10 FØRMAT (4F9.4)
```

требует обязательно четырех знаков после запятой. С другой стороны, пятая, шестая и седьмая цифры

```
   A(1)       A(2)       A(3)       A(4)
b-16.2450|b349.2710|1052.6100|b252.0000
   A(5)    |  A(6)    |  A(7)    |  A(8)
b-17.0000|8765.4320|bbb0.0018|bb-0.1534
   A(9)    |  A(10)   |  A(11)   |  A(12)
2221 1163|4500.0000|b202.4710|bb-2.0909
```

Рис. 25.

после запятой в числе A (8) не отпечатаны — инструкция FØRMAT их не разрешает. Число A (1) сдвинуто максимально вправо, но даже с учетом знака, десятичной точки и дополнительного нуля оно представляет набор восьми символов, тогда как инструкция FØRMAT их требует девять. Этот символ заменен пробелом слева.

Неприятная ситуация возникает для числа A (6): знак числа расположен в позиции, не предусмотренной инструкцией FØRMAT, и число напечатано поэтому в искаженном виде. Числа A (9) и A (10) напечатаны верно, однако отсутствие разделяющего их пробела делает неразличимой границу между ними. Этого можно было бы избежать, увеличив длину зоны в инструкции FØRMAT в (4.14), например, до 11.

Упражнение 4.11. В ходе выполнения программы вычислены координаты X, Y, Z 150 точек внутри сферы радиусом 100 и с центром в начале координат. У вычисленных величин три верных знака после запятой.

Написать инструкции вывода таблицы координат точек так, чтобы координаты одной точки были записаны в одной строке, а одноименные координаты были отпечатаны в одном столбце. Предусмотреть не менее двух пробелов между столбцами координат.

При печати действительные числа могут быть указаны в инструкции FØRMAT как в форме F, так и в форме E. Из двух целых чисел кода E, разделенных точкой, первое указывает количество выводимых символов, а второе — число знаков в нормализованной мантиссе. Так, инструкция

1 FØRMAT (E12.5) (4.15)

употребляемая в паре с инструкцией WRITE, обусловит вывод 12 символов, из которых 5 будут принадлежать нормализованной мантиссе.

При написании кода E и определении первой числовой его составляющей — количества печатаемых символов — следует учитывать, что, кроме знаков мантиссы, следует резервировать места для следующих символов:

1) нуля в разряде единиц мантиссы;

2) десятичной точки мантиссы;

3) символа E;

4) одного символа перед нулем в целой части мантиссы; этот символ сохраняется для знака числа и может заполняться пробелом для положительных чисел;

5) одного символа непосредственно после E; этот символ сохраняется для знака порядка и может заполняться пробелом для чисел, превосходящих 1.

Кроме того, следует учитывать, что величина порядка обязательно печатается как двузначное число: порядки, меньшие 10, записываются в виде двух цифр, первая из которых — нуль.

Таким образом, чтобы сформировать первую составляющую кода E, нужно к желаемому количеству значащих цифр прибавить не менее чем 7, чтоб не потерять старшие значащие символы.

Так же, как и при изображении числа в форме F, число в форме E смещается вправо в предусмотренной для него зоне. Например, число, записанное в машине как 987.65 и отредактированное инструкцией (4.15), будет иметь вид

b0.98765Eb03

(b — по-прежнему условное обозначение пробела).

Мантисса числа, отпечатанного в форме E, всегда нормализована. Это соответственно определяет величину порядка числа. Второе число в коде E (в нашем примере 5) определяет количество знаков после запятой в изображении числа, т. е. в его мантиссе. С инструкцией FØRMAT (E12.3) наше число имело бы вид

bb̄b0.987Eb03

Несмотря на некоторую сложность формирования кода E, он достаточно широко используется в инструкциях печати. Преимущество кода E становится явным, когда необходимо отпечатать числа очень широкого или

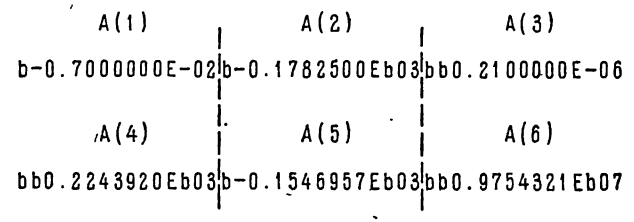

Рис. 26.

заранее неоцениваемого диапазона. Например, необходимо отпечатать список A (I) величин:

A (1) = —. 007
A (2) = — 178.25
A (3) = . 00000021
A (4) = 224.392
A (5) = — 154695.7
A (6) = 9754321.

С помощью инструкций

```
2 FØRMAT (3E15.7)
  WRITE (1, 2) (A(I), I = 1,6)
```

этот список будет отпечатан в виде, изображенном на рис. 26.

Число A (1) отпечатано с нулями, сопровождающими значащую семерку. Нули в конце мантиссы появляются всегда, когда указываемая в коде E длина мантиссы больше количества значащих цифр числа.

Все сказанное выше об инструкции вывода на печатающее устройство в равной степени может быть использовано при работе с любым другим выводным устройством: достаточно заменить номер канала.

Символ /, использованный ранее в инструкции FØRMAT, сопровождающей инструкцию READ, с тем же эффектом может быть использован и инструкциями вывода. С помощью этого символа одна инструкция может описать рисунок двух и более последовательно печатаемых строк. Точно так же, как и в инструкциях ввода, символ / в сочетании с парой внутренних скобок позволяет написать инструкцию FØRMAT, с помощью которой печатается сначала одна или несколько различных строк однократно, а затем серия строк одинакового рисунка. Например, по инструкциям

```
1 FØRMAT (F6.2/3I20/(4E12.5))
  WRITE (1,1) A, I, J, K, (X (L), L = 1,80)
```

печатается в первой строке величина A по коду F6.2, во второй строке три величины I, J, K по коду I20 и, начиная с третьей строки, печатается массив X (L) — двадцать строк, каждая из которых содержит четыре числа в коде E12.5.

В инструкции вывода символ / имеет и новые возможности: с его помощью можно планировать промежутки между строками при печати. Например, если по инструкции

```
1 FØRMAT (10F10.2/12F9.3)
```

вторая строка печатается непосредственно после первой, то по инструкции

```
1 FØRMAT (10F10.2//12F9.3)
```

вторая строка отделена от первой удвоенным промежутком, по инструкции

```
1 FØRMAT (10F10.2///12F9.3)
```

— утроенным и т. д.

Таблица 7 показывает, как выглядят массивы, отпечатанные с обычным промежутком, удвоенным, утроенным и учетверенным.

Наклонные черты (символы /) могут появляться в любом количестве в инструкции FØRMAT не только между

Таблица 7

Вид массива при печати		
Без черты или с одной чертой (исходный массив)	С двумя чертами (удвоенный промежуток)	С тремя чертами (утроенный промежуток)
1293.72	1293.72	1293.72
1740.05		
2087.41	1740.05	
2232.91		1740.05
2901.64	2087.41	
3007.45		
3238.58	2232.91	2087.41
...	...	...

кодами, но в равной степени в начале перечня кодов непосредственно после открывающей скобки или в конце его непосредственно перед закрывающей скобкой.

Каждое появление черты / в начале или в конце инструкции FØRMAT означает печатание строки пробелов.

У инструкций вывода есть еще одна возможность, не имеющая аналога с инструкциями ввода: печатая автоматические индексации, можно каждый элемент списка автоматической индексации изображать с соответствующим значением индекса. Инструкция

WRITE (1, 3) (I, A (I), I = 1, 100)

печатает 1 — первое значение индекса I и значение A (1), затем 2 — второе значение индекса и A (2), потом число 3 и A (3) и т. д.

§ 5. Вывод и ввод буквенно-символьной информации

Гибкость фортрановских инструкций ввода и вывода дополняется удобствами ввода и вывода (и, в частности, печатания) буквенных символов. Эти возможности Фортрана, в известной степени, способствовали его широкой распространенности и сделали весьма удобным для экономических приложений. Для того чтобы отпечатать любой текст, достаточно буквально написать этот текст, окаймленный апострофами, внутри скобок инструкции

FØRMAT. Каждый раз, когда инструкция FØRMAT, содержащая буквенную информацию, используется инструкцией вывода, информация, содержащаяся между апострофами, печатается или записывается на какой-либо внешний носитель информации в том виде, в каком она была записана в инструкцию FØRMAT при составлении программы. Например, инструкции

```
2 FØRMAT ('FØRTRAN')
  WRITE (1, 2)
```
(4.16)

печатают слово FØRTRAN (без апострофов).

Таким образом, написав между апострофами в скобках инструкции FØRMAT текст заголовка таблицы и ее столбцов, можно напечатать эту таблицу с заголовками. Аналогично можно предусмотреть инструкции FØRMAT, содержащие названия отдельных величин, всевозможных поясняющих результаты надписей и подписей, фраз оформления — дата, фамилия автора и т. д.

Информация, заключенная между апострофами, не обязательно буквенная — она может содержать любые разрешенные в Фортране символы: буквы, цифры и специальные символы: + — / = ( ) . , * а также пробел.

Набор символов, заключенный между апострофами, не изменяемый программой и используемый в инструкциях вывода, называется *литералом*. Единственным запрещенным символом внутри литерала является символ апострофа [1]).

В противоположность числовым данным, литерал не имеет соответствующей переменной в списке инструкции WRITE. По инструкции WRITE вычислительная машина печатает литерал из инструкции FØRMAT, номер которой указан в скобках инструкции WRITE. Пример (4.16) показывает этот необычный, без списка переменных, вид инструкции WRITE.

Литералы могут использоваться в инструкциях FØRMAT вместе с числами в любой из форм F, E или I. Например, инструкция

```
1 FØRMAT ('bTØTALbISb', F10.3)
```

[1]) Литералами могут быть русские буквы, слова и фразы, если в число символов печатающего устройства включены буквы русского алфавита и это учтено транслятором.

допустима. Если она используется вместе с инструкцией

```
WRITE (1,1) X
```

то вычислительная машина отпечатает строку, в которой десять позиций займет буквенная информация — литерал TØTAL IS, включая пробелы, и еще десять — переменная X, число в форме F. Литерал может находиться в строке после чисел, впереди них и посредине. Ограничение на длину литерала (точнее, на длину печатаемой строки, включающей литерал) — она не должна превосходить максимальную длину строки используемого печатающего устройства. Литерал может использоваться и в инструкциях FØRMAT, изображающих рисунок нескольких строк. Инструкция

```
1 FØRMAT ('PRØDUCTIØN PRØGRAMME'
  * /// (I10, F11.3))
```

позволяет отпечатать сначала заголовок таблицы PRØDUCTIØN PRØGRAMME (ПРОИЗВОДСТВЕННАЯ ПРОГРАММА), а потом в два столбца таблицу производственной программы.

Если инструкция FØRMAT определяет рисунок нескольких строк, то литералы могут быть указаны в каждой из них.

Выше литералы рассматривались лишь в инструкциях вывода. Однако, их можно использовать и при вводе буквенной и символьной информации в машину. Вводимый материал может быть комментарием или названием вводимого массива. В таком случае в скобках инструкции FØRMAT между апострофами может быть записан сам литерал. Можно, однако, указать вместо текста литерала любой набор символов такой же длины, в том числе набор пробелов. При чтении вводимой информации машина воспримет как литерал столько символов с внешнего носителя (например, с перфокарт), сколько их предусмотрено в инструкции FØRMAT. Для ввода литерала 'FØRTRAN' с перфокарт можно предусмотреть инструкцию

```
1 FØRMAT ('bbbbbbb')
```

с семью пробелами между апострофами при условии, что на картах проперфорированы семь букв — FØRTRAN.

Итак, литералы могут быть прочитаны машиной и могут быть выведены на внешние устройства. Но литералы — буквенно-символьные константы, они не могут быть изменены в ходе выполнения программы.

В некоторых случаях возникает необходимость изменения буквенно-символьной информации: включения, исключения или замены символов. С этой целью используется еще один код в инструкции FØRMAT. Это код A. Код A состоит из буквы A, за которой следует целое число. Это число — назовем его m — указывает, сколько символов должно быть прочитано или, наоборот, записано на внешние носители информации по коду A.

Код A подобно остальным известным кодам F, E и I применяется к переменным. Поэтому, если в инструкции FØRMAT содержится код A, в соответствующих инструкциях READ или WRITE должны быть указаны преобразуемые переменные. Так же, как и остальные коды, код A может употребляться в одной инструкции неоднократно; несколько одинаковых кодов могут быть употреблены один раз с целым коэффициентом, равным числу повторений; можно использовать код A со всеми другими кодами в одной инструкции FØRMAT.

Если целое число m в коде A равно числу n символов в соответствующей переменной списка инструкции READ (или WRITE), то эта переменная будет прочитана (или соответственно выведена на внешний носитель) без изменений. Но даже в этом случае инструкция FØRMAT с кодом A дает возможность при вводе или выводе формировать новые буквенно-символьные значения.

Пусть, например, величина A имеет буквенно-символьное значение FØRT, величина B имеет значение RAN. Тогда инструкции

```
1 FØRMAT (A4, A3)
  WRITE (1, 1) A, B
```

напечатают слово FØRTRAN.

Если $m < n$, то на носитель будут записаны лишь m левых символов соответствующей переменной. Если, например, A и B имеют значения, указанные в предыдущем примере, то инструкции напечатают слово FØRRAN.

Наконец, в случае $m > n$ на внешний носитель записываются все n символов переменной и, кроме того, $m - n$ пробелов слева. Для тех же A и B, что и выше, инструкции

```
1 FØRMAT (2A4)
  WRITE (1, 1) A, B
```

напечатают слово FØRT RAN (с пробелом посредине).

Если код A используется при вводе буквенно-символьных величин, то в память машины вводятся все n символов вводимой переменной при $m \geqslant n$ (m — целое число кода A, а n — длина набора символов вводимой переменной). Если же $m < n$, то m символов вводимой переменной будут прочитаны с перфокарт (магнитной ленты, перфоленты и т. д.), а вместо остальных $n - m$ символов вводятся пробелы справа.

В приведенных выше примерах код A применялся к переменным, составленным из букв, однако точно так же он используется при обработке переменных, состоящих из любых других символов, в том числе из цифр.

Упражнение 4.12. Предприятие выпускает изделия BØC-101, BØC-102, BØC-103 и BØC-104. Отчет об итогах работы предприятия должна завершить строка, в которой перечислены попарно наименования изделий и количество выпущенных изделий. Пары разделены двумя пробелами и запятой, а наименование отделено от количества минимально одним пробелом. Максимальное количество каждого изделия не превосходит 10 000.

Написать инструкции печати последней строки отчета, допустив, что программисту известны значения переменных:

```
B(1) =' —101'        A = 'BØC'
B(2) =' —102'        X = 'bb,' (два пробела и запятая)
B(3) =' —103'        Y = 'b' (один пробел)
B(4) =' —104'
```

(Знак «—» в этом примере следует читать как дефис.)

Количества выпущенных изделий BØC-101, BØC-102, BØC-103, BØC-104 составляют значения переменных C (1), C (2), C (3), C (4) соответственно.

§ 6. Редактирование результатов

Некоторые из приведенных ранее примеров (а также упражнений) убеждают в необходимости предусматривать пробелы в строке: они нужны при редактировании результатов. Выше были изложены некоторые приемы

изображения необходимых пробелов. В частности, можно включать пробелы под знаки апострофов слева или справа от литералов, указывать первую составляющую кодов F и E бóльшей, чем это необходимо для изображения выводимых чисел, указывать целое число, сопровождающее код I, заведомо бóльшим, чем количество символов числа в форме I, писать короткие буквенно-символьные переменные и максимальное целое число в коде A и т. д.

Все приемы такого рода, однако, весьма искусственны и ограничены в применениях. Фортран предусматривает более мощные средства редактирования результатов. Одним из таких средств является код X инструкции FØRMAT. Этот код состоит из буквы X и предшествующего ей целого числа. Это целое число означает количество пробелов, которые включены в строку, изображаемую инструкцией FØRMAT. Например, инструкция

```
1 FØRMAT (F10.2, 6X, F8.3)
```

предполагает, что два числа в форме F будут отпечатаны с шестью пробелами между ними. Шесть пробелов нашего примера могли бы быть реализованы включением в инструкцию FØRMAT литерала, состоящего из шести пробелов — 'bbbbbb':

```
1 FØRMAT (F10.2, 'bbbbbb', F8.3)
```

Однако преимущество кода X здесь несомненно.

Следует отметить небольшую деталь использования кода X: если необходимо указать всего лишь один пробел, то писать следует не X, а 1X. Так, один пробел между литералом и числом в коде I20 обеспечивается инструкцией

```
1 FØRMAT ('TØTAL', 1X, I20)
```

Код X может быть использован и с инструкцией READ при вводе данных. В этом случае целое число при коде X указывает, сколько колонок перфокарты не должно восприниматься при чтении. Соответствующие колонки будут прочитаны как пробелы. Например, если при чтении карты, разделенной на 8 зон, по 10 чисел в форме I, необходимо исключить первую, третью и

четвертую зоны, то можно вместе с инструкцией READ использовать инструкцию

```
3 FØRMAT (10X, I10, 20X, 4I10)
```

Исполняя инструкции вывода, вычислительная машина никогда не печатает первый знак строки: на печать выдаются все символы строки, начиная со второго. Первый символ строки является управляющим сигналом для печатающего устройства, он определяет количество пропускаемых строк перед печатью следующей строки. Таблица 8 определяет значение первого разряда печатаемой строки.

Таблица 8

Значение первого разряда строки	Действие печатающего устройства
b (пробел)	Обычный промежуток между строками
0	Двойной промежуток между строками
1	Переход на следующую страницу [1]
+	Отсутствие промежутка между строками Печать в той же самой строчке

[1]) У печатающих устройств современных ЭВМ предусматривается возможность печатать выводимые результаты фиксированными порциями — страницами. Страница может содержать, например, 60 строк (число строк на странице зависит от модели машины печатающего устройства). Обычно переход с одной страницы на другую совершается после полного заполнения страницы — печатания 60 строк. Однако с помощью управляющих сигналов рабочей программы или операционной системы можно перейти на следующую страницу, не заполнив до конца предыдущую.

Например, инструкции

```
5 FØRMAT ('1')
  WRITE (1,5)
```

обеспечивают переход на новую страницу, а инструкции

```
15 FØRMAT (4X, F10.2)
   WRITE (1,5) A
```

печатают на следующей строке десятиразрядную величину A, начиная с четвертой позиции строки. Переход к следующей строке обеспечивается благодаря тому,

что первый символ печатаемой строки является пробелом: инструкция

```
15 FØRMAT (4X, F10.2)
```

говорит, что пробелами являются первые четыре символа строки.

Возможности редактирования результатов с помощью первого символа строки весьма широки, но их эффективное использование требует внимательности программиста при формировании инструкций FØRMAT.

Вот несколько примеров неаккуратного отношения к первому разряду строки.

1. Пусть необходимо отпечатать на новой странице четыре строчки, на каждой из которых будет по одному десятиразрядному числу — A, B, C, D. Числам должно предшествовать слово TØTAL.

Инструкции

```
 2 FØRMAT ('1TØTAL', 2X, F10.3)
   WRITE (1,2) A, B, C, D
```

не дадут должного результата. Число A будет действительно отпечатано на новой странице благодаря единице в первой позиции инструкции FØRMAT. Но та же инструкция повторяется для каждой величины списка A, B, C, D, и каждая из них, следовательно, будет напечатана на своей странице.

Вот одно из возможных исправлений:

```
 2 FØRMAT ('1TØTAL', 2X, F10.3)
   WRITE (1,2) A
 3 FØRMAT (1X, 'TØTAL', 2X, F10.3)
   WRITE (1,3) B, C, D
```

В инструкции 3 единица не является первым знаком. Это часть кода X — 1X, который указывает, что первый символ строки является пробелом.

2. Инструкции

```
 4 FØRMAT (I2)
   WRITE (1,4) LIN
```

казалось бы, допускают реализацию задуманной идеи — отпечатать одно двузначное целое число. Но, допустим, что LIN = 13. В этом случае единица разряда десятков

переменной LIN будет первым символом строки. В силу этого описанная инструкцией 4 строка, начиная со второй позиции, будет отпечатана на следующей странице: напечатано будет 3 вместо 13.

Одно из возможных исправлений:

```
4 FØRMAT (I3)
```

(если заранее известно, что число LIN не более чем двухразрядное).

3. Пусть массив чисел X (I) должен быть напечатан с заголовком MATRIX.

Инструкции

```
3 FØRMAT ('MATRIX'//(5F12.5))
  WRITE (1,3) (X(I), I = 1, 100)
```

верно отпечатают числа массива, если они достаточно малы, чтоб не занимать двенадцать позиций при печати. Тем не менее инструкция FØRMAT неверна: первый символ строки не печатается, и поэтому заголовок массива будет отпечатан в виде ATRIX.

Одно из возможных исправлений:

```
3 FØRMAT ('bMATRIX'//(5F12.5))
```

Выше всюду подчеркивалась идентичность инструкции FØRMAT для всех внешних носителей. Расхождение имеет место в передаче на внешние носители первого символа строки, описанной инструкцией FØRMAT: если при выводе на печатающее устройство первый символ не изображается на бумаге и фактически исчезает, то при выводе на все остальные внешние носители информации, в частности, магнитные ленты, барабаны и диски, этот символ сохраняется.

Например, инструкция

```
3 FØRMAT ('1TØTAL')
```

при выводе на печатающее устройство отпечатает на бумаге TØTAL, а при выводе на ленту запишет 1TØTAL.

Во всех рассматривавшихся выше примерах использования инструкции FØRMAT предполагалось, что рисунок печатаемой строки формируется, начиная с первой позиции. Заголовки таблиц, подписи, а часто и выводимые результаты более наглядны, если они распо-

ложены не в начале строки. Код T [1]) дает возможность легко планировать рисунок строки. Код T употребляется непосредственно перед кодом, который должен быть размещен в желаемой позиции строки и отделен от него запятой. Код T состоит из буквы T, за которой следует целое число. Это число равно номеру желаемой позиции строки плюс 1. Например, если литерал 'MATRIX' должен быть размещен, начиная с позиции 58 печатаемой строки, то инструкция, формирующая эту строку, может иметь вид

```
1 FØRMAT  (T59, 'MATRIX')
```

Литерал 'MATRIX' будет напечатан в позициях с 58 по 63. Код T может употребляться несколько раз в одной

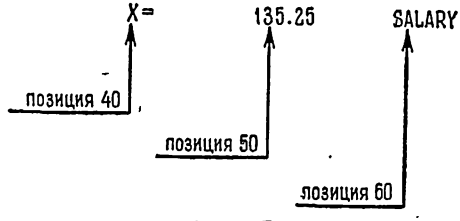

Рис. 27.

инструкции и сочетаться в ней с любыми другими кодами. Соответствие порядка кодов в инструкции FØRMAT и порядка тех же кодов в соответствующей печатаемой строке при этом не обязательно.

Например, с помощью инструкции

```
1 FØRMAT (T51, F6.2, T41, 'X =', T61, 'SALARY')
```

может быть напечатана строка, показанная на рис. 27.

В частности, если в инструкции FØRMAT присутствует код T1, то первый символ последующего кода определяет, как обычно, промежуток между строками. Так, строка, определяемая инструкцией

```
1 FØRMAT (T110, 'BLACK', T1, '0WHITE')
```

будет напечатана с двойным промежутком от предшествовавшей строки и будет содержать слово WHITE (но не 0WHITE!) в начале ее и слово BLACK в конце.

[1]) Возможности использования когда T несколько отличаются в различных трансляторах. Здесь рассказывается об одной из конкретных реализаций.

Во многих случаях код T может выполнять те же функции, что и код X. Например, указание о расположении литерала 'TØTAL', начиная с позиции 10, может быть выражено в равной степени двумя инструкциями

```
1 FØRMAT (10X, 'TØTAL') или
1 FØRMAT (T11, 'TØTAL')
```

Преимущество кода T можно усмотреть в том, что, используя его, можно просто указать номер желаемой позиции для размещения нужного кода вместо предварительного, требующего внимания, вычисления количества пробелов для использования кода X.

Впрочем, с применением кода T возникают свои трудности: следует избегать перекрытия зон, расставленных кодами T. Например, инструкция

```
5 FØRMAT (T31, I16, T41, I6)
```

приведет к нежелаемому результату — с одной стороны в позициях с 40 по 46 должно быть расположено число, определяемое кодами T41, I6, с другой стороны, эти же позиции заняты последними шестью разрядами числа, определяемого кодами T31, I16.

Пусть необходимо отпечатать таблицу, состоящую из двух столбцов X и Y, со следующими значениями:

X	Y
1.245,	5.647,
1.382,	5.505,
1.671,	5.472,
1.906,	5.366,
1.990,	5.293,
2.003,	5.097,
2.246,	5.012,
2.750,	4.863,
2.905,	4.613,
3.074.	4.588.

Заголовок печатаемой таблицы состоит из слова CURVE и целого двузначного номера. Чтоб предусмотреть пробел между словом CURVE и номером, можно записать целый номер кодом I3. Рис. 28 дает представление о том, как информация в инструкциях FØRMAT и

WRITE преобразуется в рисунок и текст печати на печатающем устройстве со 120 символами в строке.

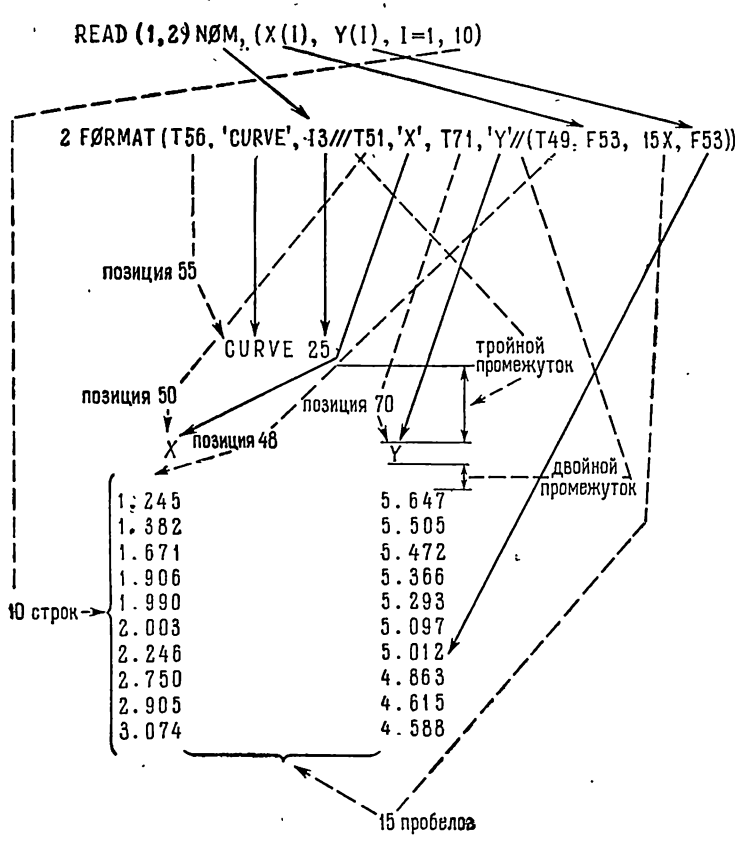

Рис. 28.

Результатом исполнения двух инструкций

```
READ (1,2) NØM, (X(I), Y(I), I = 1,10)
2 FØRMAT (T56, 'CURVE', I3///T51, 'X', T71, 'Y'//
* (T49, F53, 15X, F53))
```

является таблица координат кривой CURVE 25 в таком виде, как она изображена на рис. 28.

Упражнение 4.13. Написать инструкцию FØRMAT, предусматривающую следующий рисунок печатаемой строки:

'SALARY' — начиная с позиции 10;
число в форме F — зарплата (не более 999 рублей с точностью до копеек), начиная с позиции 20;
'PRIZE' — начиная с позиции 35;
число в форме F — сумма премии (того же формата, что и зарплата), начиная с позиции 45;
'TAX' — начиная с позиции 60;
число в форме F — сумма налога, начиная с позиции 70.

Дать два решения — одно с использованием кода T, другое — с кодом X.

Код T может быть использован для записи выводимых результатов на все другие внешние носители информации. Единственным отличием кода T в этом случае от использования его при печати является вычисление целого числа кода: оно равно непосредственно номеру позиции, начиная с которой записывается код, редактируемый кодом T — прибавление 1 не нужно. Так, для записи литерала 'DATA', начиная с 30 позиции, следует выбрать инструкцию

```
1 FØRMAT (T31, 'DATA')
```

для печати и инструкцию

```
2 FØRMAT (T30, 'DATA')
```

для записи на магнитную ленту.

Код T можно использовать и при вводе данных, он указывает в этом случае номер позиции, с которой должно начинаться чтение перфокарты или другого носителя информации. Инструкции

```
1 FØRMAT (T21, F10.2)
  READ (2,1) SUM
```

заставят машину прочитать число SUM, расположенное на перфокарте в колонках с 21 по 31. Первые двадцать колонок перфокарты не будут прочитаны, даже если они содержат какую-либо информацию.

В заключение главы следует сказать несколько слов о месте инструкции FØRMAT в программе. В противоположность всем остальным инструкциям, FØRMAT может быть расположена в совершенно произвольном месте программы (но не после END). Обычаи программистов рекомендуют размещение всех инструкций

FØRMAT, встречающихся в программе в самом начале ее или, наоборот, в самом конце. Если инструкции FØRMAT располагаются в начале программы, то им могут предшествовать лишь инструкции декларации типа величин (INTEGER, REAL, CØMPLEX или DØUBLE PRECISIØN). Распространен также обычай записи инструкций FØRMAT непосредственно рядом с соответствующей инструкцией READ или WRITE. Этот способ столь же практичен, если каждой инструкции READ и WRITE соответствует лишь одна инструкция FØRMAT.

Контрольное упражнение к главе IV

4.1К. Ввести с магнитной ленты (канал номер 3) 1000 представлений карт, на каждом из которых записаны следующие данные: наименование изделия — переменная ART0 — это код из десяти букв и цифр в позициях с 1 по 10; количество деталей на складе — переменная K0 — шестиразрядное целое число в позициях с 26 по 31;

Ввести с 200 накладных сдачи — перфокарт (по каналу номер 1). Содержание накладной сдачи:

наименование изделия — переменная ART1 в позициях с 1 по 10; количество поступивших на склад деталей — переменная K1 — шестиразрядное целое число в позициях с 11 по 16.

Полагаем, что каждое изделие ART1 совпадает по наименованию с одним и только одним изделием ART0, оба массива — изделий ART0 и ART1 — одинаково упорядочены (например, в алфавитном порядке). После ввода всех накладных сдачи сложить (по каждому изделию, отмеченному в накладных) количество поступивших на склад изделий с имеющимся на складе количеством этих изделий.

Затем ввести 150 экспедиционных карт — перфокарт (с канала номер 2). Содержание экспедиционных карт:

наименование изделия — переменная ART2 в позициях с 1 по 10; количество отправленных со склада изделий — переменная K2 — шестиразрядное целое число в позициях с 11 по 16.

Массив ART2 упорядочен так же, как и два предыдущих. После ввода ART2 надо по каждому изделию, отмеченному в экспедиционных картах, вычесть отправленное со склада количество изделий из имеющегося количества этих изделий на складе. Если при вычитании будет получена нулевая разность, приостановить решение задачи (по инструкции PAUSE), печатая слово SØS и наименование соответствующего изделия (на канал номер 7).

В заключение ввести с канала номер 2 перфокарту с датой. Дата — три двухразрядных целых числа, отделенных точками друг от друга.

Элементы, составляющие дату:

DAY (день) — две цифры и точка,
MØNTH (месяц) — две цифры и точка,
YEAR (год) — две цифры.

Отпечатать на печатающем устройстве со 128 символами в строке (канал номер 7):

1. Таблицу наличного запаса изделий на складе. Заголовок состоит из слова STØCK и даты. Между словом STØCK и датой — один пробел. Заголовок расположен в середине строки, между заголовком и таблицей — двойной пробел. Строка таблицы состоит из наименования изделия ARTO и количества изделий KO на складе с 10 пробелами посредине. Строка размещена симметрично относительно середины — позиции 60.

2. Учитывая, что до печати таблицы в ходе исполнения программы был возможен вывод на то же самое устройство промежуточных результатов — наименований изделий с нулевым запасом, рекомендуется печатать таблицу с новой страницы.

После печати таблицы остановить машину.

ГЛАВА V

ПОДПРОГРАММЫ

Одно из самых мощных средств программирования — использование подпрограмм. Вместо того, чтобы переписывать многократно повторяемые части программы, программист может оформить эти части в виде самостоятельных независимых программ. Включение одних программ в другие, использование при решении задачи ранее написанных программ — все эти возможности программист может использовать в Фортране.

Настоящая глава есть описание этих возможностей. В Фортране различаются четыре типа подпрограмм:

1) библиотечные подпрограммы,
2) арифметические подпрограммы,
3) подпрограммы типа FUNCTIØN,
4) подпрограммы типа SUBRØUTINE.

§ 1. Библиотечные подпрограммы

Во многих математических формулах широко употребляются элементарные функции, такие, как квадратный корень, экспонента, логарифм, тригонометрические функции. Программы вычисления таких функций несложны (составление упрощенной программы e^x было предложено читателю в качестве самостоятельного упражнения — упражнение 3 из контрольных к главе III). Тем не менее, если вычисление одной и той же функции встречается несколько раз, переписывание повторяющихся даже несложных частей программы нецелесообразно.

Например, при программировании формулы цилиндрической волны (из оптики)

$$E_x = C\,2\sqrt{2}\,A\,\frac{\cos(\pi q \sin\varphi)}{\cos\varphi - \cos\varphi_l} \qquad (5.1)$$

было бы весьма неразумно трижды писать программу вычисления косинуса для каждого из трех различных аргументов.

Программы элементарных функций написаны один раз, включены в Фортран как составная часть языка и могут быть использованы программистом в любой программе и неоднократно. Такие программы называют *подпрограммами*, ибо, являясь самостоятельными программами, они работают, как правило, в составе включающей их программы. *Включающая программа* управляет подпрограммами, указывая порядок их выполнения и подготавливая для них *аргументы*. Аргументы подпрограмм являются их входными данными.

Множество фортрановских программ элементарных функций называют *библиотекой*, а сами программы элементарных функций — *библиотечными подпрограммами*.

Каждая библиотечная подпрограмма имеет свое наименование. Для того чтобы воспользоваться нужной подпрограммой из библиотеки, достаточно указать ее наименование, так же, как указываются наименования переменных. При желании вычислить, например, синус числа X, достаточно написать инструкцию

$$Y = SIN(X) \qquad (5.2)$$

SIN — это наименование библиотечной подпрограммы синуса. Аргумент подпрограммы по требованиям Фортрана должен быть записан в скобках непосредственно вслед за наименованием библиотечной подпрограммы. Инструкция (5.2) выбирает из библиотеки подпрограмму по ее наименованию, выполняет ее с указанным в инструкции аргументом и, наконец, присваивает вычисленное значение переменной, стоящей слева от знака равенства. Инструкция, использующая библиотечные подпрограммы, исполняется как любая арифметическая инструкция, за исключением предварительного этапа выборки подпрограммы из библиотеки.

Вычислительная машина не может перейти к последующей инструкции, не закончив предыдущую, в частности, не закончив выполнение подпрогаммы. Наименование библиотечной подпрограммы может быть использовано в инструкции так же, как наименование переменной: можно писать инструкции, использующие разные

подпрограммы и одну подпрограмму несколько раз, выполняя действия с результатами подпрограмм как с переменными. Инструкция

```
Y = 2*SIN (ALPHA + BETA)*CØS (ALPHA —
*BETA)/(CØS (ALPHA)**2*CØS (BETA)**2)          (5.3)
```

один раз использует подпрограмму SIN и три раза подпрограмму CØS, каждый раз с разными аргументами [1]). Эта инструкция показывает, что библиотечные подпрограммы могут участвовать во всех арифметических выражениях на правах переменных.

Написание функций библиотечных подпрограмм несколько отличается от их привычной записи в математике. Прежде всего, их наименования пишутся заглавными буквами. Обычно аргумент входит под знак элементарной функции без скобок — $\sin x$, $\cos x$, ..., тогда как в Фортране скобки, окружающие аргумент, обязательны — SIN(X), CØS(X)... Необычным является и способ записи степеней элементарных функций. В математике показатель степени пишут над строкой между знаком функции и аргументом — $f^2(x)$ или, в частности, $\cos^2 x$. В Фортране показатель степени записывается после скобок, содержащих аргумент — CØS(X)**2. Это естественно, поскольку CØS(X) участвует в выражении как переменная. Другие особенности фортрановской записи функций элементарных подпрограмм будут указаны ниже.

Аргумент библиотечной подпрограммы может быть константой, например SQRT(3.14159) [2]), переменной или арифметическим выражением, как в примере (5.3). Аргументом подпрограммы может быть и другая библиотечная подпрограмма. В выражении

```
CØS (SQRT (DELTA))
```

---

[1]) Символ / в начале второй строки выражения (5.3) означает перенос. Он размещается в позиции продолжения.

[2]) SQRT — это наименование подпрограммы вычисления квадратного корня. Выше (в гл. II) была показана возможность вычисления квадратного корня возведением аргумента в степень 0.5. На практике предпочитают для этого пользоваться библиотечной подпрограммой SQRT.

сначала по подпрограмме SQRT вычисляется квадратный корень величины DELTA, затем этот результат становится аргументом другой библиотечной подпрограммы CØS.

Библиотечные подпрограммы на правах переменных могут участвовать в любых, не только арифметических инструкциях. Так, инструкция условной передачи управления

```
IF (CØS (ØMEGA*T + FI)) 1, 2, 3
```

в зависимости от знака косинуса передает управление инструкциям 1, 2 или 3.

Упражнение 5.1. Написать инструкцию вычисления E_x по формуле (5.1) (стр. 143), приняв $2\sqrt{2} = 2.8284$ и $\pi = 3.1416$.

Некоторые из библиотечных подпрограмм Фортрана приведены в таблице 9. Подпрограммы, приведенные

Таблица 9

Подпрограмма	Наименование	Примечание	Математическое обозначение	Подпрограмма с двойной точностью
Синус тригонометрический	SIN (X)	Аргумент в радианах	$\sin x$	DSIN (X)
Косинус тригонометрический	CØS (X)	Аргумент в радианах	$\cos x$	DCØS (X)
Тангенс гиперболический	TANH (X)	Аргумент в радианах	th	
Арктангенс	ATAN (X)		Arc tg x	DATAN (X)
Квадратный корень	SQRT (X)	Аргумент неотрицательный	$\sqrt{x}$	DSQRT (X)
Экспонента	EXP (X)		e^x	DEXP (X)
Натуральный логарифм	ALØG (X)	Аргумент положительный	$\ln x$	DLØG (X)
Десятичный логарифм	ALØGIO (X)	Аргумент положительный	$\log x$	DLØG10 (X)

в таблице 9, имеют одно общее свойство: аргументы всех подпрограмм задаются в форме действительных чисел, результаты всех этих подпрограмм также действительные. Выражение CØS (KØL) не разрешено в Фортране;

у библиотечной подпрограммы CØS не может быть целых аргументов.

Наименования библиотечных подпрограмм являются ключевыми словами Фортрана, хотя они построены по тем же правилам, что и наименования переменных: во-первых, все наименования подпрограмм состоят только из букв и цифр, во-вторых, все они начинаются с буквы, в-третьих, ни одно из этих наименований не состоит более чем из шести символов. Наконец, главное: первая буква наименований подпрограмм соответствует тому, что результаты подпрограмм — действительные числа. Действительно, они отличаются от букв I, J, K, L, M, N, с которых начинаются обычно названия целых переменных. Для того чтобы удовлетворить этому правилу и сохранить близость наименований библиотечных подпрограмм математическим обозначениям соответствующих им элементарных функций, пришлось подпрограмму натурального логарифма назвать ALØG (X) (а не LØG (X)!), а подпрограмму десятичного логарифма ALØG10 (X) (но не LØG10 (X)!).

В последнем столбце таблицы 9 указаны наименования подпрограмм с двойной точностью. Двойную точность имеют как аргументы, так и результаты подпрограммы.

Библиотека подпрограмм не ограничивается подпрограммами, перечисленными в таблице 9. Все другие библиотечные подпрограммы могут иметь не только действительные, но и целые аргументы, каждая из них может быть использована в двух вариантах в зависимости от формы представления результата — целой или действительной.

Одна из этих библиотечных подпрограмм формирует из заданного аргумента его *абсолютное значение*, т. е. приписывает аргументу знак +, каким бы ни был аргумент — положительным или отрицательным. Ее наименование есть ABS в том случае, когда необходимо получить абсолютное значение в действительной форме, и IABS, когда нужен результат в форме целого числа.

Важно отметить, что подпрограммы ABS и IABS, так же как и все другие библиотечные подпрограммы, используют величину аргумента, но не воздействуют на него. Инструкция

$$X = ABS\ (A) \qquad (5.4)$$

присваивает результат подпрограммы переменной X. Переменная A остается той же. И если аргумент A до выполнения инструкции (5.4) был равен —21.45, то после ее исполнения X = +21.56, но, по-прежнему A = = —21.45.

Подпрограмма ABS может быть с успехом использована всегда, когда другие библиотечные подпрограммы требуют положительного аргумента. Например, подпрограммы ALØG, ALØG10, SQRT могут дать результат только в случае положительных аргументов, а выражение SQRT(ABS(A)) гарантирует вычисление квадратного корня даже при отрицательных A. С использованием подпрограммы ABS программа примера (3.4) из третьей главы могла бы быть записана двумя инструкциями:

```
Y = ABS (X)
ALN = — (Y + Y**2/2. + Y**3/3. + Y**4/4.)
```

или (хотя и менее элегантно) даже одной инструкцией

```
ALN = — (ABS (X) + ABS (X)**2/2. +
*ABS (X)**3/3. + ABS (X)**4/4.)
```

Подпрограммы ABS и IABS, вычисляя абсолютные значения, не могут преобразовывать форму представления аргументов. Поэтому для получения абсолютной величины целого числа используется подпрограмма IABS, и только она; наоборот, для действительных аргументов разрешено использовать подпрограмму ABS. Например, инструкция

```
INDEX = IABS (VALUE)
```

недопустима.

В программах весьма часто встречается необходимость определения наименьшего значения среди некоторого множества чисел. Это определение может быть выполнено библиотечной программой *минимума*. Множество чисел, среди которых разыскивается наименьшее, есть множество аргументов этой подпрограммы. В зависимости от формы представления аргументов и результата она может иметь четыре разных наименования.

Таблица 10 предусматривает все возможные комбинации форм представления аргументов и результатов при нахождении минимума, но при этом все аргументы,

Таблица 10

Библиотечная подпрограмма	Форма результата	Форма аргумента	Пример
AMIN1	Действительная	Действительная	AMIN1 (X, Y)
MIN1	Целая	Действительная	MIN1 (З., T, U, V)
AMIN0	Действительная	Целая	AMIN0 (IND, LIN)
MIN0	Целая	Целая	MIN0 (MI, MU, MØ)

сколько бы их ни было, должны иметь одну и ту же форму представления.

Число аргументов у подпрограмм AMIN1, MIN1, AMIN0 и MIN0 не ограничивается. Однако, все аргументы должны быть перечислены индивидуально. В случае необходимости найти наименьшее из десяти чисел массива X(I) можно воспользоваться инструкцией

```
AMIN1 (X(1), X(2), X(3), X(4), X(5),
*X(6), X(7), X(8), X(9), X(10))
```

но автоматическая индексация при перечислении аргументов

```
AMIN1 (X(I), I = 1, 10)
```

уже недопустима.

В любой из четырех своих модификаций библиотечная подпрограмма минимума находит наименьший из аргументов, а в случае подпрограмм AMIN0 и MIN1, кроме того, результат преобразуется в форму, отличную от формы представления аргументов.

Две инструкции

```
X = AMIN0 (KLIN, LINK)
```

и

```
X = MIN0 (KLIN, LINK)
```

дают один и тот же результат. В первом случае библиотечная подпрограмма AMIN0 сама преобразует наименьшее из целых чисел KLIN и LINK в действительную форму. Во второй инструкции тот же результат получается в форме целого числа, а преобразование в действительную форму совершается арифметической инструкцией присвоения результата действительной переменной X.

Упражнение 5.2. Аспиранту устанавливается стипендия S в 100 рублей, если его заработная плата Z до поступления в аспирантуру превышала 100 рублей. Стипендия устанавливается равной заработной плате, если последняя не превосходила 100 рублей.

Написать с использованием подпрограммы минимума инструкцию, определяющую величину S по вышеприведенному положению. Величины S, Z и 100 действительные.

Совершенно аналогично подпрограммам минимума определяются четыре библиотечные подпрограммы определения наибольшего из аргументов. Правила формирования и использования подпрограмм AMAX1, MAX1, AMAX0, MAX0 те же, что для соответствующих подпрограмм минимума. Если A = 1.0, B = —3.0, C = 2.0, то инструкция

```
Y=AMIN1(A,B)+AMAX1(ABS(A),ABS(B),ABS(C))
```

присваивает величине Y значение 0.. (Из двух точек, стоящих после нуля, первая есть признак числа в действительной форме, а вторая — конец фразы.)

Библиотечные подпрограммы FLØAT и IFIX служат для *преобразования формы* единственного их аргумента. FLØAT может иметь аргументом только целые величины и преобразует их в действительные; напротив, аргумент подпрограммы IFIX есть число в действительной форме, а результат — то же число, преобразованное в целую форму. В соответствии с этим определением выражения FLØAT(A) или IFIX(J) не имеют смысла (если предварительно инструкциями декларации типа величин число A не было определено целым, а J — действительным).

Две инструкции

```
X = NØS и X = FLØAT (NØS)
```

дают один и тот же результат — переменную NØS в действительной форме.

Библиотечная подпрограмма AINT имеет аргументом число в действительной форме и дает действительный результат. Эта подпрограмма выделяет *целую часть аргумента*, представляя ее в действительной форме. При X = 5.6 инструкция

```
Y = AINT(3.*X/2.)
```

дает результат Y = 8..

Библиотечная подпрограмма INT имеет действительный аргумент. Так же как и подпрограмма AINT, она выделяет целую часть аргумента, с той лишь разницей, что результат остается в целой форме.

В примере

```
N = INT(3.*X/2.)
```

при том же X = 5.6 результатом будет N = 8 (число в целой форме).

Допустим, что числа надо распределить по классам так, чтобы числа первого десятка принадлежали классу 1, второго десятка — классу 2 и т. д. Тогда целый номер класса K действительного числа X можно определить инструкцией

```
K = INT(X/10.)+1
```

Действительно, для числа X = 31.4159 из четвертого десятка K определяется так: деление X/10. дает действительное число 3.14159, подпрограмма INT (3.14159) находит целое 3 и, наконец, последняя операция сложения определяет номер класса 4.

Библиотечные подпрограммы AMØD и MØD имеют два аргумента. У подпрограммы AMØD оба аргумента действительные, у подпрограммы MØD — целые. Результаты подпрограммы в соответствии с неявным определением их формы (по наименованиям подпрограмм) получаются в действительной форме для AMØD и в целой для MØD.

Подпрограммы AMØD и MØD осуществляют деление первого аргумента по модулю второго аргумента. Например, результат подпрограммы AMØD (23.,4.) — деления 23. по модулю 4., т. е. остатка при делении 23./4.,— есть число 3.. Инструкция K = MØD(I,J) при I = 7, J = 3 определяет K = 1 (целое).

Подпрограммы деления по модулю могут быть полезны, например, при определении четности или нечетности числа. Действительно, MØD(N,2) равно 0 для четных N и 1 для нечетных. Такая проверка на четность может осуществляться арифметической условной передачей управления. Инструкция

```
IF (MØD(N,2)) 1, 2, 1
```

передает управление инструкции 2 при четных N и инструкции 1 — при нечетных.

Библиотечные подпрограммы SIGN и ISIGN называются подпрограммами *присвоения знака*. Они имеют два аргумента. Результатом подпрограмм служит абсолютное значение первого аргумента, взятое со знаком второго. В подпрограмме SIGN оба аргумента имеют действительную форму, в подпрограмме ISIGN — оба целые. Например, если L = +5, M = —7, то после выполнения инструкции

IKØN = ISIGN (L, M)

получим IKØN = —5, а при L = —5, M = 7 результатом будет IKØN = 5.

Библиотечные подпрограммы DIM и IDIM требуют двух аргументов. Оба они должны иметь действительную форму в подпрограмме DIM и целую — в подпрограмме IDIM. По подпрограммам DIM и IDIM определяется наименьшее значение из двух аргументов, и первый аргумент уменьшается на это значение. Следовательно, результат подпрограммы DIM (и IDIM) положителен или нуль.

Выражение IDIM (6,2) равно 4, DIM (2.,—5.) дает 7., DIM (3.5, 4.2) равно 0..

В таблице 11 сведены все описанные выше функции, кроме перечисленных ранее в таблице 9.

Часть из этих подпрограмм имеют варианты, используемые для обработки аргументов с двойной точностью и выдающие результат с двойной точностью. Например, подпрограммы определения максимума и минимума — DMAX1 и DMIN1, подпрограмма присвоения знака DSIGN и подпрограмма определения абсолютного значения — DABS — имеют и аргументы и результаты в форме с двойной точностью. Подпрограмма выделения целой части аргумента IDINT предполагает, что аргумент представлен с двойной точностью, результат этой подпрограммы — целое число.

Не имеет аналогий с подпрограммами таблицы 11 библиотечная подпрограмма DBLE приведения действительного аргумента к форме с двойной точностью.

Возможность представления в Фортране комплексных чисел привела к необходимости иметь в составе библиотеки подпрограммы, оперирующие с комплексными числами в целом и с их частями — вещественными и мнимыми.

Таблица 11

Группы подпрограмм	Библиотечная подпрограмма	Форма аргумента	Число аргументов	Форма результата
Абсолютное значение	ABS	Действительный	1	Действительный
	IABS	Целый	1	Целый
Максимум	AMAX1	Действительный	Не ограничено	Действительный
	MAX1	Действительный	Не ограничено	Целый
	AMAX0	Целый	Не ограничено	Действительный
	MAX0	Целый	Не ограничено	Целый
Минимум	AMIN1	Действительный	Не ограничено	Действительный
	MIN1	Действительный	Не ограничено	Целый
	AMIN0	Целый	Не ограничено	Действительный
	MIN0	Целый	Не ограничено	Целый
Преобразования формы	FLØAT	Целый	1	Действительный
	IFIX	Действительный	1	Целый
	AINT	Действительный	1	Действительный
	INT	Действительный	1	Целый
Деление по модулю (остаток)	AMØD	Действительный	2	Действительный
	MØD	Целый	2	Целый
Присвоение знака	SIGN	Действительный	2	Действительный
	ISIGN	Целый	2	Целый
«Уменьшение» аргумента	DIM	Действительный	2	Действительный
	IDIM	Целый	2	Целый

Библиотечные подпрограммы функцией с комплексными аргументами перечислены в таблице 12. Комплексный аргумент (в первом столбце таблицы 12) имеет вид $a + bi$.

Таблица 12

Подпрограмма	Наименование	Форма аргумента	Форма результата
Модуль аргумента $\sqrt{a^2 + b^2}$	CABS (C)	Комплексный	Комплексный
Экспонента e^{a+bi}	CEXP (C)	Комплексный	Комплексный
Натуральный логарифм $\ln (a + bi)$	CLØG (C)	Комплексный	Комплексный
Синус тригонометрический $\sin (a + bi)$	CSIN (C)	Комплексный	Комплексный
Косинус тригонометрический $\cos (a + bi)$	CCØS (C)	Комплексный	Комплексный
Корень квадратный $\sqrt{a + bi}$	CSQRT (C)	Комплексный	Комплексный
Выделение вещественной части комплексного числа	REAL (C)	Комплексный	Вещественный
Выделение мнимой части комплексного числа	AIMAG (C)	Комплексный	Вещественный
Объединение двух вещественных аргументов в комплексную величину	CMPLX (A, B)	Два действительных аргумента	Комплексный
Получение сопряженной величины $a - bi$	CØNT (C)	Комплексный	Комплексный

§ 2. Арифметические подпрограммы

В библиотеку Фортрана включены подпрограммы, наиболее часто встречающиеся в большинстве практических программ. Специфика задачи часто требует использования подпрограмм, не включенных в библиотеку, характерных только для данной конкретной задачи. Подпрограммы, исполняющие арифметические инструкции и не включенные в библиотеку Фортрана, будем называть *арифметическими подпрограммами*.

Арифметическая подпрограмма подобно арифметической инструкции вычисляет одно единственное арифметическое выражение. В отличие от арифметической инструкции, которая оперирует лишь с указанными в этой

инструкции переменными, арифметическая подпрограмма может при каждом своем выполнении работать с иными аргументами.

Пусть в программе необходимо несколько раз вычислить квадрат расстояния точки от начала координат, т. е. сумму квадратов трех ее координат. Эта величина (назовем ее DIS) есть функция трех переменных X, Y и Z:

$$\text{DIS (X,Y,Z)} = \text{X**2} + \text{Y**2} + \text{Z**2} \qquad (5.5)$$

Пример (5.5) — это пример арифметической подпрограммы. Она определяется как единая инструкция, включающая наименование подпрограммы с ее аргументами, слева от знака равенства и арифметическое выражение от этих аргументов справа от него. Каждый раз, когда в программе встречается наименование арифметической подпрограммы, выполняется эта арифметическая инструкция. Это значит, что, написав один раз определение арифметической подпрограммы (5.5), можно в других частях программы просто упоминать ее наименование на правах переменной. Так, если в некотором месте программы потребуется определить отношение квадратов расстояний от начала координат двух точек, из которых одна имеет координаты A, B, C, а другая — D, E, F, то инструкцию, вычисляющую это отношение, можно записать

$$\text{Y} = \text{DIS(A,B,C)/DIS (D,E,F)}$$

В этой инструкции дважды используется арифметическая подпрограмма DIS. Первый раз по формуле (5.5) вычисляется величина DIS с аргументами A, B, C: в формуле (5.5) заменяются X на A, Y — на B, Z — на C; затем вычисляется величина DIS с другими аргументами D, E, F : X заменяется на D, Y — на E, Z — на F. Арифметическая подпрограмма (5.5) избавляет тем самым от необходимости переписывать дважды одну и ту же последовательность операций. Важно то, что строка (5.5) служит по существу определением функции подпрограммы и сама по себе не является исполняемой инструкцией.

Это определение подпрограммы появляется в программе лишь один раз, но операции, ее составляющие, исполняются всякий раз, когда наименование подпрограммы вместе с аргументами появляется в выражении. Обращение к подпрограмме и возврат в основную программу осуществляются инструкцией, содержащей

наименование подпрограммы в выражении справа от знака равенства.

Выше основным отличием арифметических подпрограмм от библиотечных была названа их специфика, их особенности, характерные лишь для конкретной включающей программы. Отличие это фиксируется и формально: имена арифметических подпрограмм — это наименования, формируемые программистом, тогда как именами библиотечных подпрограмм служат ключевые слова. Наконец, разница между арифметическими и библиотечными подпрограммами состоит в том, что место библиотечных подпрограмм зафиксировано транслятором и не зависит от программиста, место же арифметической подпрограммы, вообще говоря, определяется программистом.

Аргументы в инструкции, определяющей арифметическую подпрограмму — это *искусственные переменные*. Они лишь показывают, что́ должно быть проделано с теми реальными переменными, которые будут указаны как фактические аргументы этой подпрограммы. Инструкции, определяющие арифметическую подпрограмму, не могут содержать индексированных переменных. Кроме искусственных переменных — искусственных аргументов, заменяемых в момент обращения к подпрограмме,— инструкции определения подпрограммы могут содержать и другие переменные. В инструкции

```
TIGR (X) = A*X**2                    (5.6)
```

арифметическое выражение содержит единственный аргумент. Переменная A не является аргументом, она не заменяется при обращении к подпрограммам никакими другими величинами — при выполнении подпрограммы TIGR(X) используется реальное значение переменной A. Если в другом месте программы встретится инструкция

```
Z = B+TIGR(Y)                        (5.7)
```

то Z будет вычислено по формуле (5.6) при X = Y и том значении A, которое эта переменная имела перед исполнением инструкции (5.7).

В выражения арифметических подпрограмм могут быть включены библиотечные подпрограммы. Например, арифметическая подпрограмма вычисления расстояния

точки с координатами X и Y от начала координат может быть записана

```
DISTAN (X, Y) = SQRT (X**2+Y**2)
```

Инструкции, определяющие арифметические подпрограммы, записываются в начале программы перед исполняемыми инструкциями. Предшествовать этим инструкциям могут лишь инструкции декларации типа величин и, при желании, инструкции FØRMAT.

Арифметические подпрограммы создаются программистом для нужд его программы, вместе с подпрограммой программист создает и ее наименование. Формирование наименования арифметической подпрограммы подчиняется общим правилам образования наименований в Фортране; при этом форма представления результата задается неявным определением подпрограммы — первой буквой ее наименования. Инструкция, определяющая арифметическую подпрограмму вычисления выражения

$$-\frac{b+\sqrt{b^2-4ac}}{2a},$$

может быть записана

```
RADIC (A, B, C)
*= (— B + SQRT (B**2 — 4.*A*C))/(2.*A)
```

Упражнение 5.3. Площадь равнобочной трапеции вычисляется по формуле

$$S = (k - l\cos\alpha)\, l \sin\alpha,$$

где l — боковая сторона трапеции, k — ее основание, α — угол при основании.

Написать инструкцию, определяющую арифметическую подпрограмму вычисления площади равнобочной трапеции. И аргументы, и результат подпрограммы должны быть действительными.

§ 3. Подпрограммы типа FUNCTIØN

Пусть необходимо написать программу для вычисления числа сочетаний (соединений из n элементов по m, различающихся составом элементов) по формуле

$$C_n^m = \frac{n!}{m!\,(n-m)!}. \qquad (5.8)$$

Эта формула требует троекратного вычисления факториалов. Поэтому было бы естественно воспользоваться программой вычисления факториала (например, программой (3.6) из главы III) как подпрограммой. Однако, подпрограммы факториала нет в библиотеке Фортрана. Эта подпрограмма не может быть арифметической, так как всякая арифметическая подпрограмма по определению состоит из одной-единственной арифметической инструкции, тогда как программа вычисления факториала, как видно в примере (3.6), требует нескольких инструкций, в том числе неарифметических.

В Фортране существует специальный тип подпрограмм, называемых *подпрограммами типа* FUNCTIØN, допускающих любое необходимое число инструкций. Подпрограмма типа FUNCTIØN является сама по себе полной программой, она имеет даже свою собственную инструкцию END, которая всегда отмечает конец текста программы.

Подпрограмма типа FUNCTIØN полностью независима и отделена от программы, которая ее использует. Такая подпрограмма представляет список инструкций, исполняемых всякий раз, когда основная программа упоминает наименование, выбранное для подпрограммы типа FUNCTIØN. Подпрограмма типа FUNCTIØN отличается от обычной программы своей первой инструкцией. Эта инструкция формируется из ключевого слова FUNCTIØN, наименования подпрограммы и заключенного в скобки списка аргументов. Наименование подпрограммы типа FUNCTIØN подчиняется всем правилам формирования наименований в Фортране. Аргументы подпрограммы типа FUNCTIØN искусственные, так же как и в арифметических подпрограммах. Первой инструкцией подпрограммы вычисления факториала могла бы быть строка

```
FUNCTIØN IFACT(IARG)
```

при условии, что подпрограмма названа IFACT.

Еще одно отличие подпрограммы типа FUNCTIØN от обычной программы состоит в том, что последней ее исполняемой инструкцией является инструкция возврата в основную программу. Эта инструкция состоит из единственного ключевого слова RETURN. Она означает окон-

чание выполнения подпрограммы и необходимость возврата в основную программу. Место инструкции RETURN в подпрограмме может быть произвольным, благодаря использованию передач управления, но, как правило, ее записывают непосредственно перед инструкцией END, замыкающей перечень инструкций подпрограммы типа FUNCTIØN.

Каждая подпрограмма типа FUNCTIØN имеет еще одну обязательную инструкцию. Это арифметическая инструкция присвоения результата подпрограммы переменной, наименование которой совпадает с наименованием подпрограммы (без скобок и списка аргументов). Эта инструкция также может располагаться произвольно, однако чаще всего она оказывается непосредственно перед инструкцией RETURN.

Видно, что выбор наименования подпрограммы типа FUNCTIØN определяет наименование ее результата, и наоборот. Таким образом, если мы хотим использовать программу (3.6) в качестве подпрограммы для вычисления факториала, ее следует назвать IFACT.

После этих замечаний подпрограмма типа FUNCTIØN вычисления факториала будет иметь вид

```
      FUNCTIØN IFACT (IARG)
      IFACT = 1
    1 IFACT = IFACT*K
      K = K — 1
      IF (K) 1, 2, 1
    2 RETURN
      END
```

(5.9)

Отличие (5.9) от (3.6) незначительно: добавлены лишь строки FUNCTIØN, RETURN и END. Обязательная для подпрограммы типа FUNCTIØN инструкция RETURN здесь написана непосредственно перед END, но обязательная инструкция присвоения (инструкция 1) стоит в середине подпрограммы.

Обращение к подпрограмме типа FUNCTIØN из основной программы производится по ее наименованию, как и в случае библиотечных или арифметических подпрограмм. Формула (5.8) из начала настоящего параграфа

могла бы быть запрограммирована одной инструкцией (после того как написана подпрограмма (5.9)):

C = IFACT (N)/(IFACT (M) * IFACT (N — M))

где M — наименование переменной *m*, N — наименование переменной *n*.

Каждый из трех в этой инструкции вызовов подпрограммы IFACT влечет за собой замещение искусственного аргумента IARG подпрограммы соответствующими каждому вызову фактическими аргументами M, N—M и N.

Наименование подпрограммы неявно определяет форму представления результата. В примере (5.9) результат — целое число. Для результата подпрограммы типа FUNCTIØN можно также использовать явное определение формы представления, записывая перед словом FUNCTIØN в первой инструкции подпрограммы слово INTEGER (целый) или REAL (действительный). В этом случае результат принимает указанную форму независимо от наименования подпрограммы. Если, например, нужно получить в действительной форме результаты подпрограммы типа FUNCTIØN с наименованием IRIS и искусственными аргументами X, Y, Z, то первой инструкцией ее следует написать

REAL FUNCTIØN IRIS (X, Y, Z)

В примере (5.9) у подпрограммы IFACT одна-единственная выходная величина, единственный результат — IFACT. Однако, в тех случаях, когда подпрограмма типа FUNCTIØN имеет несколько аргументов, любой из ее аргументов может быть выходной величиной. Легко себе представить, что каждый раз одновременно с факториалом числа IARG необходимо вычислить некоторую функцию от факториала, например трехчлен

IT = IFACT (IARG)**2 + IFACT (IARG) + 1. (5.10)

Ничто не мешает, конечно, выполнить инструкцию (5.10) в основной программе. Однако, если вычисления величин IFACT и IT выполняются много раз, тогда основная программа должна будет повторить инструкцию (5.10) столько же раз. Этого легко избежать, включая инструкцию (5.10) в подпрограмму (5.9), которая в таком

случае примет вид

```
      FUNCTIØN IFACT(IARG, IT)
      IFACT = 1
      K = IARG
    1 IFACT = IFACT*K
      K = K — 1
      IF (K) 1, 2, 1
    2 IT = IFACT**2 + IFACT + 1
      RETURN
      END
```

(5.11)

Если в основной программе надо умножить J! на трехчлен $(J!)^2+J!+1$, можно, используя подпрограмму (5.11), это сделать инструкцией

```
L = IFACT (J, J3)*J3
```

— после того, как подпрограмма вычислила свои выходные величины, их можно использовать в дальнейшем.

В программе типа FUNCTIØN обязательно должна присутствовать еще одна инструкция в том случае, когда в подпрограмме используются индексированные переменные или массивы чисел. Речь идет об инструкции DIMENSIØN. Это неисполняемая инструкция, она сообщает программе сведения о длине (количестве элементов) обрабатываемых подпрограммой массивов. В этом смысле инструкция DIMENSIØN сходна с инструкцией FØRMAT. Если подпрограмма обрабатывает массив TAB из 100 чисел, в ней обязательная инструкция

```
DIMENSIØN TAB(100)
```

За словом DIMENSIØN указывается сначала наименование массива (индексированной переменной), а затем в скобках целая константа, обозначающая длину массива. Целая константа в скобках может быть равной количеству чисел массива или превосходить его, но ни в коем случае не быть меньше. Необходимость в написании целой константы, превышающей длину массива, появляется тогда, когда длина массива меняется от одного исполнения подпрограммы к другому. Тогда должна быть указана максимальная из возможных длин.

Если в подпрограмме используется несколько массивов, то наименование и длина каждого из них должны быть упомянуты инструкцией DIMENSIØN. Так, если в подпрограмме используются массивы X из 200 чисел, Y из 100 чисел и Z из 300 чисел, то в подпрограмме будут написаны три инструкции DIMENSIØN:

```
DIMENSIØN X(200)
DIMENSIØN Y(100)
DIMENSIØN Z(300)
```

которые, впрочем, эквивалентны одной

```
DIMENSIØN X(200), Y(100), Z(300)
```

Инструкция DIMENSIØN описывает длину всех индексированных переменных, упоминаемых в подпрограмме, независимо от того, являются ли они искусственными аргументами подпрограммы или фактическими переменными, используемыми подпрограммой. Кроме того, длина индексированных переменных, используемых в качестве аргументов подпрограмм типа FUNCTIØN, указывается обязательно в инструкции DIMENSIØN, принадлежащей основной программе.

Каждая программа, содержащая индексированные переменные, обязательно содержит записываемые в начале программы инструкции DIMENSIØN. Употребление индексированных переменных в программе без указаний их длины инструкциями DIMENSIØN недопустимо. Каждой индексированной переменной, участвующей в подпрограмме, соответствует две инструкции DIMENSIØN: одна — из подпрограммы для искусственной переменной, другая — из включающей программы для фактической.

Инструкции DIMENSIØN должны предшествовать исполняемым инструкциям, содержащим наименования массивов, которые они описывают. Поэтому в подпрограммах их размещают следом за наименованием подпрограммы, а в основной программе — раньше всех исполняемых инструкций (раньше инструкций FØRMAT, но вслед за инструкциями декларации типа величин).

В качестве примера подпрограммы типа FUNCTIØN, использующей инструкцию DIMENSIØN, рассмотрим программу вычисления скалярного произведения двух векторов. Оба вектора, являющиеся аргументами под-

программы,— это массивы одинаковой длины. Длина массива (число N) — это размерность вектора.

Скалярное произведение определяется формулой

$$S=(x_1,\ x_2,\ \ldots,\ x_N)\times(y_1,\ y_2,\ \ldots,\ y_n)=\sum_{i=1}^{N}x_iy_i. \quad (5.12)$$

Формула (5.12) показывает, что программа вычисления скалярного произведения строится одинаково для векторов любой фиксированной размерности. Поэтому, вместо того чтобы писать отдельно подпрограмму для двухразмерных векторов, отдельно — для трехразмерных и т. д., целесообразно сделать программу для векторов любой размерности, задавая размерность N как аргумент подпрограммы

```
      FUNCTIØN SCAL (VECT1, VECT2, N)
      DIMENSIØN VECT1(700), VECT2(700)
      SCAL = 0.
      DØ 1 I = 1, N
    1 SCAL = SCAL + VECT1(I) * VECT2(I)
      RETURN
      END
```

(5.13)

Подпрограмма названа SCAL. В соответствии с этим переменная, являющаяся результатом, имеет то же наименование (см. инструкцию 1). Аргументы подпрограммы — массивы VECT1, VECT2 и целая константа N, которая определяет размерность обоих векторов. В инструкции DIMENSIØN указано ограничение для размерности векторов — 700. Следовательно, неравенство

$$N \leqslant 700$$

должно выполняться обязательно. Это означает, что подпрограмма (5.13) предусматривает вычисление скалярных произведений векторов, размерность которых не превосходит 700. Тот факт, что наименования переменных VECT1 и VECT2 упомянуты в инструкции DIMENSIØN, дает основание считать их индексированными переменными.

Обязательные инструкции подпрограммы расположены в наиболее употребительном порядке: сначала

инструкции определения подпрограммы, затем инструкция DIMENSIØN, после этого следует серия исполняемых инструкций подпрограммы, наконец, арифметическая инструкция присвоения результата переменной SCAL (здесь она является последней инструкцией цикла DØ), инструкции RETURN и END.

Инструкцию RETURN, заканчивающую исполнение подпрограмм типа FUNCTIØN, следует отнести к инструкциям передачи управления. Поэтому она не может (подобно инструкциям IF и GØ TØ) быть последней инструкцией цикла DØ. Тогда, когда подпрограмма типа FUNCTIØN кончается циклом DØ, инструкция RETURN может быть записана лишь после инструкции CØNTINUE (ср. гл. III, § 2).

В главе III читателю было предложено упражнение 3.6, в котором следовало написать группу инструкций, вычисляющих количество положительных элементов массива X, состоящего из 500 элементов. В ответах к упражнениям главы III приведено одно из решений

```
      N = 0
      DØ 1 I = 1, 500
      IF (X(I)) 1, 1, 2
    2 N = N + 1
    1 CØNTINUE
```

Эта группа инструкций может быть преобразована в подпрограмму следующим образом:

```
      FUNCTIØN N(X)
      DIMENSIØN X(500)
      N = 0
      DØ 1 I = 1, 500
      IF (X(I)) 1, 1, 2
    2 N = N + 1
    1 CØNTINUE
      RETURN
      END
```

(5.14)

В отличие от арифметических подпрограмм, подпрограммы типа FUNCTIØN могут иметь в качестве аргу-

ментов массивы чисел. Но эти аргументы, как и раньше, искусственные: при каждом обращении к подпрограмме массивы-аргументы заменяются фактическими массивами, указанными в инструкциях основной программы. Например, сложить количество положительных элементов в массивах SLØN и TIGR можно инструкцией

$$Y = N(SLØN) + N(TIGR) \qquad (5.15)$$

основной программы, дважды вызывающей подпрограмму (5.14) с наименованием N.

Если в инструкции определения подпрограммы упомянуто несколько искусственных аргументов, то в каждой инструкции основной программы, упоминающей наименование подпрограммы, фактические аргументы перечисляются в том же количестве, порядке и форме представления. Если, например, в основной программе нужно вычислить скалярное произведение векторов VINT1 и VINT2 размерности M с помощью подпрограммы (5.13), то инструкция основной программы

$$Y = SCAL\ (VINT1,\ VINT2,\ M)$$

решает эту задачу, а инструкция

$$Y = SCAL\ (VINT1,\ M,\ VINT2) \qquad (5.16)$$

неверна: в определении подпрограммы написано сначала две индексированные переменные, а затем целая, в инструкции же (5.16) целая переменная стоит между двумя индексированными.

Так же, как для линейных массивов инструкция DIMENSIØN отмечает их длину (или максимально возможную длину), при описании двух- и трехразмерных массивов инструкция DIMENSIØN должна указывать число значений для каждого индекса. Например, инструкция

DIMENSIØN TAB (10, 20)

говорит о том, что TAB — это массив, предусматривающий хранение 2000 чисел, разделенных на 20 групп по десять в каждой.

В основной программе и внутри включаемой в нее подпрограммы могут использоваться одинаковые наименования для различных переменных, если одна из них

употребляется только в основной программе, а другая — только в подпрограмме. Внутри подпрограммы могут быть повторены номера инструкций, употребляемые в основной программе. Функциональные отношения между основной программой и подпрограммой могут быть установлены лишь тогда, когда переменные основной программы указываются на месте искусственных аргументов подпрограммы.

Упражнение 5.4. Написать полностью подпрограмму суммирования массива целых чисел. Подпрограмма должна допускать любую длину массива, не превосходящую 1000.

§ 4. Подпрограммы типа SUBRØUTINE

Подпрограммы типа SUBRØUTINE во многих отношениях похожи на подпрограммы типа FUNCTIØN: в обеих присутствуют обязательные инструкции RETURN и END, для обеих одинаковы правила использования искусственных аргументов, которые могут быть как переменными, так и массивами. Оба типа подпрограммы имеют одинаково формируемые наименования. Так же, как и подпрограммы типа FUNCTIØN, подпрограмма типа SUBRØUTINE полностью независимы и отделены от основной программы, номера их инструкций и их переменные никак не связаны ни с другими подпрограммами (за исключением переменных, являющихся искусственными аргументами), ни с основной программой. Но если подпрограмма типа FUNCTIØN имеет результатом одну или несколько отдельных величин, то подпрограмма типа SUBRØUTINE может иметь в качестве результата массивы чисел.

Подпрограммы типа FUNCTIØN вызываются в основную программу упоминанием наименования подпрограммы. Для вызова подпрограмм типа SUBRØUTINE необходима специальная инструкция CALL. Эта инструкция состоит из ключевого слова CALL, наименования подпрограммы и заключенного в скобки списка ее аргументов.

Первой инструкцией подпрограммы типа SUBRØUTINE, инструкцией ее определения, обязательно является строка, состоящая из ключевого слова SUBRØUTINE, наименования подпрограммы и списка ее аргументов. Последняя исполняемая инструкция подпро-

граммы типа SUBRØUTINE — инструкция RETURN, последняя инструкция текста подпрограммы — инструкция END.

Примером простейшей подпрограммы типа SUBRØUTINE может служить подпрограмма копирования, переписывающая группы чисел из одного массива в другой:

```
      SUBRØUTINE CØPIE (A, B, N)
      DIMENSIØN A(1000), B(1000)
      DØ 1 I = 1, N
    1 B (I) = A (I)
      RETURN
      END
```

(5.17)

Подпрограмма (5.17) рассчитана на работу с массивами, длина которых не превосходит 1000.

Обращение к этой подпрограмме из основной программы осуществляется инструкцией

```
CALL CØPIE (ØLD, NEW, K)
```

по которой искусственная индексированная переменная A подпрограммы (5.17) заменяется фактической индексированной переменной ØLD, искусственный аргумент — массив B — фактическим наименованием NEW, искусственная переменная N — фактической переменной K. Результатом подпрограммы является массив NEW, он указан одним из аргументов в инструкции CALL обращения к подпрограмме типа SUBRØUTINE.

Упражнение 5.5. Программу сортировки (3.21), стр. 74, оформить в виде подпрограммы типа SUBRØUTINE, включив число N элементов сортируемого массива в список аргументов и предусмотрев сортировку массивов до 4000 элементов.

Пусть подпрограмма сортировки имеет своей первой инструкцией

```
SUBRØUTINE SØRT(T, N)
```

Описываемая ниже программа — это пример полной программы, использующей подпрограмму сортировки. Она читает массив ALPHA из 1000 чисел, записанных на перфокартах, сортирует их и затем печатает

отсортированный массив:

```
      DIMENSIØN ALPHA (1000)
    1 FØRMAT (5F10.3)
    2 FØRMAT (10F11.3)
      READ (1,1) (ALPHA(I), I = 1, 1000)
      N = 1000
      CALL SØRT (ALPHA, N)
      WRITE (3, 2) (ALPHA(I), I = 1, 1000)
      STØP
      END
```

(5.18)

Необходимым условием правильного включения подпрограммы SØRT в программу (5.18) является требование, чтобы целая константа в инструкции DIMENSIØN подпрограммы была не меньше 1000.

Если массив ALPHA не превосходит по длине максимально допустимый подпрограммой сортировки массив, то программу (5.18) можно переписать, вводя количество элементов массива с перфокарт. В этом случае в инструкции DIMENSIØN основной программы указывается та же константа, что и в инструкции DIMENSIØN подпрограммы (например, 4000):

```
      DIMENSIØN ALPHA (4000)
    1 FØRMAT (I4/(10F3.1))
    2 FØRMAT (1X. F3.1)
      READ (1,1) N, (ALPHA(I), I = 1, N)
      CALL SØRT (ALPHA, N)
      WRITE (3, 2) (ALPHA(I), I = 1, N)
      STØP
      END
```

Упражнение 5.6. Написать полную программу, которая вводит с перфокарт три вектора — три массива действительных чисел (10 чисел на карте, код чисел — F5.1):

$$A(I),\ I=1,100$$
$$B(I),\ I=1,100$$
$$C(I),\ I=1,100$$

и две действительные величины ALPHA и BETA, затем выполняет с ними операцию по формуле

$$Y=\alpha\ (A+B)+\beta\ (A-B)+C$$

и, наконец, печатает результирующий вектор.

Написать и использовать в программе подпрограммы типа SUBRØUTINE сложения векторов, вычитания векторов и умножения на скаляр, каждая из которых предусматривает обработку векторов размерностью до 1000.

(Написанная здесь программа не будет все же действующей реальной программой из-за отсутствия инструкции CØMMØN, вводимой в следующей главе.)

Подпрограммы типа SUBRØUTINE могут содержать внутри себя инструкции CALL, вызывающие другие подпрограммы типа SUBRØUTINE и любые другие инструкции, в которых может содержаться обращение к подпрограммам типа FUNCTIØN, арифметическим или библиотечным подпрограммам.

Однако, подпрограмма не может содержать обращение к самой себе ни непосредственно, ни через серию других подпрограмм. Две подпрограммы не могут обращаться друг к другу.

Этому же ограничению подчиняются и подпрограммы типа FUNCTIØN.

Контрольные упражнения к главе V

5.1К. Написать единственную инструкцию, определяющую наибольшее из чисел I1, I2, I3, I4, I5, затем наибольшее из чисел J1, J2, J3, и наименьшее из этих двух чисел присвоить действительной переменной PØINT.

5.2К. Написать инструкцию условной передачи управления, передающую управление инструкции номер 1, когда абсолютное значение минимума из трех чисел ALPHA, BETA и GAMMA меньше чем 3.141593/2.

5.3К. Тангенс учетверенного угла (назовем эту величину TG4) вычисляется по формуле

$$\operatorname{tg} 4\alpha = \frac{\sin 4\alpha}{\cos^4\alpha - \frac{3}{2}\sin^2 2\alpha + \sin^4\alpha}.$$

Написать инструкцию определения арифметической подпрограммы TG4 (ALPHA).

5.4К. Массив PLAN содержит 175 чисел, выражающих запланированное количество 175 видов товарных изделий, массив ØUTPUT содержит 175 чисел, выражающих выпущенное количество каждого изделия. Необходимо вычислить процент выполнения плана по каждому изделию и найти CØEF — средний процент выполнения плана предприятия как среднее арифметическое из процентов выполнения плана по каждому изделию.

Написать полностью подпрограмму типа FUNCTIØN, выполняющую необходимые вычисления.

5.5К. Написать полностью подпрограмму типа SUBRØUTINE, которая в матрице MATR1 (I, J), имеющей 20 строк и 30 столбцов, представляет строки в обратном порядке: первая строка становится двадцатой, вторая — девятнадцатой и т. д. Результирующую матрицу назвать MATR2 (I, J), а подпрограмму — INVER.

ГЛАВА VI

ИНСТРУКЦИИ ОРГАНИЗАЦИИ ПРОГРАММ

Инструкции, описываемые в этой небольшой главе, используются при распределении места в памяти машины между массивами исходных данных и промежуточных результатов. Они, по существу, связывают между собой отдельные подпрограммы и подпрограммы с основной программой. Поэтому их называют *инструкциями организации программы.* Здесь же суммируются рекомендации по размещению в программе инструкций разных типов и отдельных частей программы.

В первых двух параграфах этой главы описаны только основные употребления, вообще говоря, достаточно сложных инструкций EQUIVALENCE и CØMMØN. Детальное описание этих инструкций следует читать в руководствах по конкретным трансляторам.

§ 1. Инструкция EQUIVALENCE

Вычислительные машины обладают ограниченными запоминающими устройствами. Поэтому одной из главных забот программиста при программировании более или менее сложных задач является проблема экономии места в памяти машины. Программисты называют эту проблему *распределением памяти.*

Распределяя память, надо учитывать, в частности, место для исходных числовых величин и промежуточных результатов: давая наименование той или иной переменной, программист должен помнить о необходимости разместить в памяти машины эту величину.

Один из способов экономии места в памяти — использование одного и того же места для разных величин, участвующих в программе не одновременно, на разных этапах ее решения. Инструкция EQUIVALENCE позволяет реализовать этот способ, привязывая различные

переменные к одним и тем же позициям памяти. Эта инструкция формируется из ключевого слова EQUIVALENCE, за которым в скобках указан список переменных, привязанных к одному и тому же месту в памяти. Переменные списка разделяются запятыми. Инструкция EQUIVALENCE относится к числу неисполняемых инструкций, она лишь сообщает транслятору сведения, необходимые для составления рабочей программы.

Инструкция

EQUIVALENCE (ØLD, ACT, NEW)

говорит о том, что переменной ACT не выделяется в памяти машины отдельного места, эта величина разместится на том же месте, где была расположена величина ØLD. То же самое можно сказать и о величине NEW. Все три переменные привязаны к одному и тому же месту. Они имеют, тем не менее, разные наименования, так как на трех независимых последовательных этапах одной программы в наименование переменной, место которой фиксировано, может вкладываться различное содержание.

Одна инструкция EQUIVALENCE может выполнить одновременно несколько присвоений мест в памяти. По инструкции

EQUIVALENCE (X2, Y2, Z2), (ALPHA, MU)

три переменные — X2, Y2 и Z2 — привязываются к одному месту в памяти, а две переменные — ALPHA и MU — к другому. Запятая между скобками обязательна. Как показывает последний пример, переменные разной формы представления вполне могут оказаться привязанными к одному месту в памяти. В числе переменных внутри скобок может быть индексированная переменная при условии, что индексом является целая константа. Инструкция

EQUIVALENCE (X(15), Y(J))

недопустима, так как в качестве индекса второй переменной в скобках указана не целая константа, а целая переменная.

Если с помощью инструкции EQUIVALENCE установлена эквивалентность места для двух элементов массива (двух индексированных переменных), то эта

эквивалентность автоматически распространяется на все другие, последующие и предыдущие элементы массивов. Инструкция

EQUIVALENCE (X(15), Y(1), Z (2))

означает, что величины X(15), Y(1) и Z(2) должны быть размещены на одном месте в памяти, но, кроме того, эта же инструкция означает эквивалентность мест:

X(14) и Z(1)
X(16), Y(2) и Z(3)
X(17), Y(3) и Z(4) и т. д.

§ 2. Инструкция CØMMØN

Описанная в предыдущем параграфе инструкция EQUIVALENCE привязывает к одному месту памяти различные переменные одной и той же программы. Можно предусматривать размещение на одном и том же месте переменных, принадлежащих подпрограмме, и переменных включающей программы. Эти функции выполняет инструкция CØMMØN.

Инструкция CØMMØN формируется из ключевого слова CØMMØN и списка наименований переменных, разделенных запятыми. (Список не должен быть заключен в скобки, как в инструкции EQUIVALENCE.) Например, допустима инструкция

GØMMØN ALPHA, BETA, INDEX, MAT

Инструкция CØMMØN записывается в программе по меньшей мере дважды. Если она употреблена один раз в подпрограмме, то еще одно ее употребление обязательно во включающей основной программе или другой подпрограмме той же программы. Если при этом основная программа содержит инструкцию CØMMØN TEST, а подпрограмма — инструкцию CØMMØN TØST, то переменная TEST основной программы и переменная TØST подпрограммы будут размещены на одном и том же месте.

Пара инструкций CØMMØN, из которых одна принадлежит основной программе, а вторая — подпрограмме, могут выполнить одновременно несколько присвоений. Инструкции CØMMØN A, B, J и CØMMØN X, Y, K

указывают, что переменная A должна быть размещена там же, где переменная X, B — на том же месте, где Y, J — там же, где K.

Порядок, в котором перечислены переменные в одной из инструкций CØMMØN (например, принадлежащей основной программе), определяет порядок списка переменных в соответствующей второй инструкции. Если в предыдущем примере переменная B должна располагаться на том же месте в памяти, что и переменная Y, то обе эти переменные указываются на соответствующих, в нашем примере вторых, местах списков переменных в своих инструкциях CØMMØN.

Если основная программа содержит инструкцию

```
CØMMØN AST, BIS, LIST
```

а в одной из включаемых программ необходимо совместить лишь переменную MAT с переменной LIST основной программы (не заботясь о месте, занимаемом переменными AST и BIS), то и в этом случае переменная MAT в инструкции CØMMØN подпрограммы должна стоять на соответствующем, в нашем примере третьем, месте. Для этого предыдущие позиции в списке инструкции CØMMØN должны быть заполнены наименованиями искусственных переменных. Так, инструкция

```
CØMMØN X1, X2, MAT
```

может реализовать необходимое совмещение переменных LIST и MAT.

В списке переменных инструкции CØMMØN могут быть и индексированные переменные (в списке они указаны лишь наименованием без сопровождающих индексов), так что, не зная контекста программы, нельзя определить, идет ли в предыдущем примере речь о целых переменных LIST и MAT или о массивах LIST и MAT целых чисел. Массивы, размещаемые на одно место двумя инструкциями CØMMØN, должны иметь одинаковую размерность.

Экономя место в памяти, инструкции CØMMØN оказываются особенно эффективными, когда в списках переменных участвуют массивы больших размерностей. Если основная программа включает несколько подпрограмм, то общие места в памяти могут быть предусмотрены инструкциями CØMMØN в каждой из них.

При описании подпрограмм говорилось о том, что подпрограммы независимы ни от основной программы, ни между собой. Каждая из подпрограмм может иметь свою нумерацию инструкций (номера инструкций в подпрограмме и основной программе могут совпадать), свои наименования переменных, которые могут быть одинаковыми в программе и подпрограммах.

Следовательно, если использованием одинаковых наименований для переменных в программе и подпрограмме стремятся подчеркнуть, что речь идет об одних и тех же величинах, то для этого необходимы инструкции CØMMØN, имеющие одинаковые списки для основной программы и для подпрограммы.

Если, например, переменную PAS подпрограммы необходимо использовать под тем же наименованием в основной программе, необходимо и в основную программу, и в подпрограмму включить инструкцию

```
CØMMØN PAS
```

Точно так же, если надо подчеркнуть, что переменная PAS основной программы имеет то же значение, что переменная VIST подпрограммы, это следует отметить включением инструкции

```
CØMMØN PAS
```

в основную программу и инструкции

```
CØMMØN VIST
```

в подпрограмму. При таком написании программы и подпрограммы слова PAS и VIST будут просто разными наименованиями одной и той же величины.

Инструкция CØMMØN устанавливает соответствие между переменными в основной программе и подпрограмме. Однако инструкция CØMMØN не допускает описания искусственных аргументов, поскольку присваивание необходимых значений искусственным аргументам подпрограмм осуществляется инструкциями вызова подпрограмм (упоминанием наименования подпрограммы типа FUNCTIØN или инструкцией CALL для подпрограмм типа SUBRØUTINE). Таким образом, все используемые в подпрограмме переменные (за исключением переменных, указанных в левой части инструкций присваивания в подпрограмме) должны быть упомянуты

либо в списке аргументов подпрограммы, либо в списке ее параметров — в списке инструкции CØMMØN. Если подпрограмма используется только в одной основной программе (или в группе программ с едиными наименованиями для семантически одинаковых переменных), можно перевести переменную из списка аргументов в список параметров, назвав ее общей для программы и подпрограммы с помощью инструкции CØMMØN.

Например, в подпрограмме (5.17) величину N — размерность копируемого массива — можно считать не аргументом ее, а параметром, определяемым в основной программе. В таком случае подпрограмма будет иметь вид:

```
      SUBRØUTINE CØPIE(A, B)
      DIMENSIØN A(1000), B(1000)
      CØMMØN N
      DØ 1 I = 1, N                                    (6.1)
    1 B(I) = A(I)
      RETURN
      END
```

а основная программа должна иметь инструкцию CØMMØN N.

Если основная программа и подпрограмма (6.1) работают всегда с одинаковыми массивами, то число аргументов подпрограммы (6.1) может быть еще сокращено. Тогда (6.1) переписывается в подпрограмму без аргументов

```
      SUBRØUTINE CØPIE
      DIMENSIØN A(1000), B(1000)
      CØMMØN A, B, N
      DØ 1 I = 1, N
    1 B(I) = A(I)
      RETURN                                           (6.2)
      END
```

В основной программе в этом случае обязательна инструкция CØMMØN A, B, N, обращение к подпрограмме осуществляется с помощью инструкции CALL CØPIE.

Для подпрограмм типа FUNCTIØN сокращение числа аргументов нельзя производить беспредельно: в списке ее аргументов должен быть по крайней мере один элемент.

§ 3. Организация программы

Программа решения задачи формируется из основной программы и включаемых в нее подпрограмм. Как было показано в главе V, подпрограммы типа FUNCTIØN и SUBRØUTINE являются независимыми частями программы, которые могут быть отдельно от основной программы написаны, отдельно от нее отлажены и независимо от нее использованы как части других программ.

Независимые и самостоятельные части программы называют *сегментами* программы, а организация программы путем составления ее из отдельных сегментов называется *сегментацией*. Сегментация программ позволяет компоновать сложные программы из готовых сегментов, ранее написанных или использовавшихся в других программах. Так открываются возможности создания библиотеки программ в любом вычислительном центре, использующем Фортран в качестве языка описания задач.

Первым сегментом программы всегда является *основная программа*. В начале этого сегмента пишут слово PRØGRAM, которое может сопровождаться наименованием программы. В остальном порядок сегментов в сегментированной программе, вообще говоря, произволен. Тем не менее могут быть предложены рекомендации по сегментации программы, основанные на программистских традициях. За сегментом PRØGRAM размещают сегменты подпрограмм типа SUBRØUTINE, а вслед за ними подпрограммы типа FUNCTIØN.

Все сегменты организованы в основном одинаково. Перечисляемые ниже рекомендации по порядку инструкций внутри сегментов резюмируют упоминаемые в разных главах книги правила размещения инструкций.

В каждом сегменте устанавливается такой порядок инструкций:

инструкции декларации типа величин,
инструкции DIMENSIØN,
инструкции CØMMØN,

инструкции EQUIVALENCE,
инструкции FØRMAT [1]),
инструкции определения арифметических подпрограмм,
исполняемые инструкции программы,
END.

Для сегментов, являющихся подпрограммами типа SUBRØUTINE, впереди этого перечня добавляется наименование подпрограммы с предшествующим словом SUBRØUTINE и последующим списком аргументов, в подпрограммах типа FUNCTIØN — инструкция, состоящая из слова FUNCTIØN наименования подпрограммы и списка аргументов, в сегменте основной программы слово PRØGRAM и наименование программы.

Последней исполняемой инструкцией в сегменте основной программы является STØP, в сегментах подпрограмм — RETURN.

Контрольные упражнения к главе VI

6.1К. Написать инструкцию, предусматривающую размещение на том же месте памяти 20 первых элементов массива LIST и 20 последних элементов массива FILE, имеющего 30 элементов.

6.2К. Допустим, что основная программа обрабатывает массивы TAB1 и TAB2, действительные переменные A, B, C, D, E, F и целые переменные KØL, LIM, NUL, MER. Она включает три подпрограммы. В первой из них участвует массив TAB1, действительные переменные A, C, E и все целые переменные; во второй — массив TAB2, все действительные переменные, а из целых — LIM и MER. Третья подпрограмма использует оба массива, действительные переменные C, D, E, F и целые — KØL, LIM, NUL. Написать инструкцию CØMMØN для всех программ. Сколько инструкций CØMMØN необходимо?

---

[1]) Если место всех остальных инструкций внутри сегмента строго определено или весьма рекомендовано, для инструкций FØRMAT могут быть использованы и другие рекомендации (см. гл. IV). Инструкции FØRMAT обязательно снабжены номерами.

ГЛАВА VII

ПРИМЕРЫ ФОРТРАНОВСКИХ ПРОГРАММ

Эта завершающая глава содержит три примера полных программ на Фортране. Первая из них — небольшая вычислительная задача на классический сюжет — программа численного интегрирования по способу трапеций.

Вторая программа сложнее. Она, хотя и несколько упрощена в учебных целях, но даже в таком виде пригодна для определения критического пути на реальных сетевых графиках с не очень большим числом событий.

Наконец, третья программа представляет фрагмент большого алгоритма составления плана промышленного раскроя материала. Этот фрагмент — программа определения пересечения двух плоских контуров — имеет самостоятельное значение и может быть использован в других алгоритмах обработки геометрической информации.

§ 1. Программа вычисления площади замкнутой фигуры

Численное интегрирование — простой и удобный метод вычисления площадей плоских фигур. Смысл его состоит в том, что определенный интеграл

$$\int_a^b f(x)\,dx,$$

измеряющий площадь, ограниченную вертикалями a и b, осью абсцисс и кривой $f(x)$ (рис. 29), разбивается на сумму интегралов по частичным равным интервалам:

$$\int_a^b f(x)\,dx = \sum_{i=1}^{N} \int_{x_i}^{x_{i+1}} f(x)\,dx,$$

где $x_0 = a$, $x_N = b$, а N достаточно велико, чтоб вычисление интеграла

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \tag{7.1}$$

свести к простым арифметическим операциям.

Один из наиболее известных и простых способов численного интегрирования — способ трапеций. В этом способе фигуры, площади которых, описывают элементарные интегралы (7.1), заменяются трапециями (рис. 29). Тогда весь процесс вычислений сводится к трем основным процедурам:

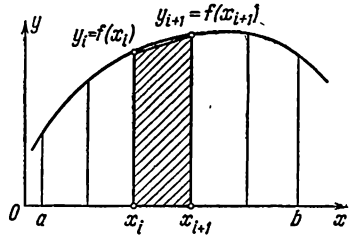

Рис. 29.

1) вычисление значений функций $f(x)$ в точках x_i,

2) вычисление площадей элементарных трапеций,

3) суммирование площадей элементарных трапеций.

Площадь элементарной трапеции вычисляется по формуле

$$S_i = \frac{1}{2}(x_{i+1} - x_i)(f(x_i) + f(x_{i+1})),$$

и тогда площадь S фигуры выражается приближенно

$$S = \int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{i=1}^{N}(x_{i+1} - x_i)(f(x_i) + f(x_{i+1})),$$

или, приняв равными элементарные интервалы оси абсцисс — высоты элементарных трапеций ($x_{i+1} - x_i \equiv h$),

$$S = \int_a^b f(x)\,dx \approx \frac{h}{2}\sum_{i=0}^{N}(f(x_i) + f(x_{i+1})) =$$

$$= \frac{h}{2}(f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)). \tag{7.2}$$

Итак, составление программы для вычисления площади фигуры сводится к программированию простой формулы (7.2) с использованием подпрограммы функции $f(x)$.

Если необходимо вычислить площадь фигуры, ограниченной непрерывным замкнутым контуром, можно определить ее как алгебраическую сумму нескольких площадей, задаваемых определенными интегралами (рис. 30). При этом следует помнить, что площадь фигуры, расположенной ниже оси абсцисс, считается отрицательной.

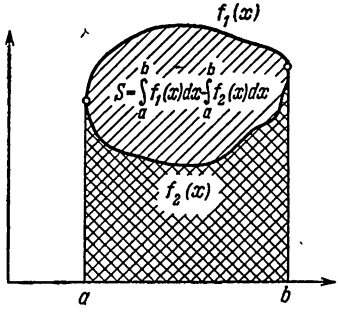

Рис. 30.

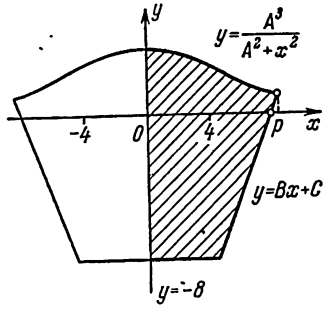

Рис. 31.

Пусть надо запрограммировать вычисление площади фигуры (рис. 31), ограниченной кривой

$$f_1(x)=\frac{A^3}{A^2+x^2}, \qquad A>0, \tag{7.3}$$

и ломаной, которая может быть задана

$$f_2(x)=\left\{\begin{array}{ll} -Bx-C & \text{при } x<-4, \\ -8. & \text{при } -4\leqslant x\leqslant +4, \\ +Bx-C & \text{при } x>4, \\ B>0,\ C>0. & \end{array}\right\} \tag{7.4}$$

Симметрия фигуры позволяет вычислить лишь площадь ее одной, например, правой половины, которая выразится

$$S_1=\int_0^p f_1(x)\,dx+\int_0^p f_2(x)\,dx,$$

где p — абсцисса точки пересечения $f_1(x)$ и $f_2(x)$.

Вычисления p достаточно трудоемки — при определенных выше $f_1(x)$ и $f_2(x)$потребуется построение подпрограммы нахождения и анализа корней кубического

уравнения. Можно избежать вычисления p, суммируя площади элементарных трапеций до тех пор, пока величина $f_1(x) - f_2(x)$ не станет отрицательной величиной. (Здесь учитывается специфика класса кривых (7.3) и (7.4).) При этом, возможно, будет некоторая потеря точности, определяемая величиной h. Вычисление h представляет собой самостоятельную задачу, мы полагаем ее решенной до исполнения нашей программы и потому вправе полагать, что h — заданная константа, обеспечивающая необходимую точность вычислений.

Значения функции $f_1(x)$ могут быть вычислены арифметической подпрограммой

```
F1(X) = A**3/(A**2 + X**2),
```

а для вычисления значений $f_2(x)$ надо написать подпрограмму типа FUNCTIØN:

```
      FUNCTIØN F2(X)
      CØMMØN B, C
      IF (4. + X) 1, 1, 2
    1 F2 = — B*X—C
      GØ TØ 5
    2 IF (4. — X) 3, 3, 4
    3 F2 = — 8.
      GØ TØ 5
    4 F2 = B*X—C
    5 RETURN
      END
```

Программа начинает свою работу вводом (с канала 2) параметров и констант задачи. Вот первые инструкции программы:

```
      PRØGRAM TRAP
      CØMMØN B, C
    1 FØRMAT (4E12.5)
    2 FØRMAT (F7.2)
      F1 (X) = A**3/(A**2+X**2)
      READ (2, 1) A, B, C, H
```

Инструкция номер 2 предусматривает формат результата.

Главная часть программы, вычисляющей одновременно площади обеих составляющих фигур (рис. 34), имеет вид

```
      X = 0.
      S1 = A
      S2 = —8.
    3 X = X + H
      IF (F1(X) — F2(X)) 7, 4, 4
    4 S1 = S1 + 2.*F1(X)
      IF(F2(X)) 5, 5, 6
    5 S2 = S2 + 2.*F2(X)
      GØ TØ 3
    6 S2 = S2 — 2.*F2(X)
      GØ TØ 3
    7 S1 = S1 — F1(X — H)
      S2 = S2 — F2(X — H)
      S = (H/2.)*(S1 — S2)
      WRITE (3, 2) S
      STØP
      END
```

Здесь инструкция

```
      IF (F1(X) — F2(X)) 7, 4, 4
```

по существу фиксирует окончание вычислений.

Инструкция

```
      IF (F2(X)) 5, 5, 6
```

отмечает переход кривой $f_2(x)$ из отрицательной области в положительную — необходимо сменить знак операции, что и выполняется инструкцией 6.

Инструкция 7 и следующая за ней включены в программу с тем, чтоб компенсировать ненужное удвоение последнего слагаемого в формуле (7.2).

§ 2. Определение критического пути на сетевом графике

Одна из часто встречающихся в производственной практике задач — определение критического пути на сетевом графике. Сетевое планирование основывается на представлении сложного многоэтапного комплекса работ

в виде сети, на которой вершинами отмечаются события — окончания отдельных этапов, а соединяющими их стрелками — работы, которые необходимо выполнить для перехода от одного этапа к другому.

Рассмотрим, например, одну из возможных организаций работ по программированию и решению некоторой задачи на ЭВМ.

Сетевой график этой организации приведен на рис. 32. Началом работ — событием *1* — назовем получение задачи математиком. Событие *2* — распределение заданий.

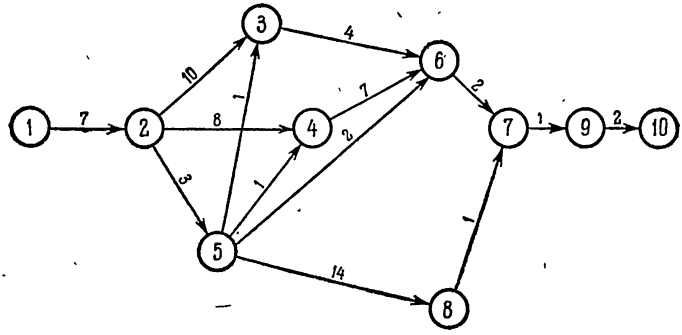

Рис. 32.

Работа (*1*, *2*) состоит в том, что математик составляет общую схему решения задачи, выделив в ней подпрограммы, не зависящие от основной программы, и сформулировав требования к форме входных данных и результатов.

Далее параллельно — разными программистами или группами программистов — могут выполняться три работы: работа (*2*, *3*) — программирование основной программы, которая приводит к событию *3*: «Основная программа готова к отладке», работа (*2*, *4*) — программирование подпрограмм, приводящих к событию «Подпрограммы готовы к отладке» и работа (*2*,*5*) — подготовка данных для отладки программы и подпрограммы, которая заканчивается событием *5*: «подготовлены тесты для отладки программы и подпрограммы».

События *3* и *4* не наступят, однако, пока не будут завершены соответственно две работы — работа (*5*,*3*),

состоящая в передаче программисту из группы подготовки данных отладочного теста для программы и его согласования, а также работа (*5,4*) — передача теста для подпрограмм.

Работа (*3,6*) состоит в отладке основной программы, работа (*4,6*) — это отладка подпрограммы, а работа (*5,6*) — подготовка теста для комплексной отладки — стыковки. Работы (*3,6*), (*4,6*) и (*5,6*) заканчиваются общим событием *6*:«программа подготовлена к комплексной отладке». Стыковка, .или комплексная отладка,— это работа (*6,7*); которая приводит к событию *7*: «программа готова к производственному счету». Событию *7* предшествует еще две работы:· работа (*5,8*) — подготовка исходных данных для производственного счета, оканчивающаяся событием *8*: «данные для решения готовы», и работа (*8,7*) — передача и согласование исходных данных. Работа (*7,9*) — решение задачи на ЭВМ — приводит к событию *9*: «окончание решения» и, наконец, последняя из работ — оформление результатов — к конечному событию *10*: «завершение работ».

Цифры, указанные на стрелках рис. 35, означают длительности в днях соответствующих работ. Например, цифра *4* на стрелке, соединяющей вершины *3* и *6*, означает, что на отладку основной программы планируется 4 дня.

Первый этап алгоритма планирования совокупности работ состоит в том, что каждой вершине сети ставится в соответствие величина t_i — ожидаемое время наступления события i. Если предположить, что ожидаемые времена всех предшествующих событий уже вычислены, то ожидаемое время наступления события i определится как максимум из сумм $t_j + t_{ji}$ по всем работам, приводящим к событию i, где t_{ji} — длительность работы (j, i); отмеченная на стрелке, ведущей от события j к событию i, а t_j — ожидаемое время наступления события j.

На нашей схеме t_2 определяется однозначно: $t_1 = 0$ — начало работ можно условно привязать к нулевому моменту времени и

$$t_2 = t_1 + t_{1.2} = 0 + 7 = 7.$$

Для вычисления t_3 необходимо сначала определить t_5, ибо наступление события *3* одним из условий имеет выполнение работы (*5,3*), начинающейся после события *5*.

К событию 5 приводит лишь одна работа (2,5), поэтому t_5 определяется однозначно

$$t_5 = t_2 + t_{2,5} = 7 + 3 = 10.$$

Но для вычисления t_3 уже необходимо сравнивать величины

$$t_2 + t_{2,3} \text{ и } t_5 + t_{5,3},$$

соответствующие двум работам, без выполнения которых не наступит событие 3:

$$t_3 = \max(t_2 + t_{2,3},\ t_5 + t_{5,3}) = \max(7 + 10,\ 10 + 1) = 17.$$

Такое определение ожидаемого времени вполне естественно, ибо событие наступает лишь тогда, когда завершается наиболее трудоемкая из предшествующих ему

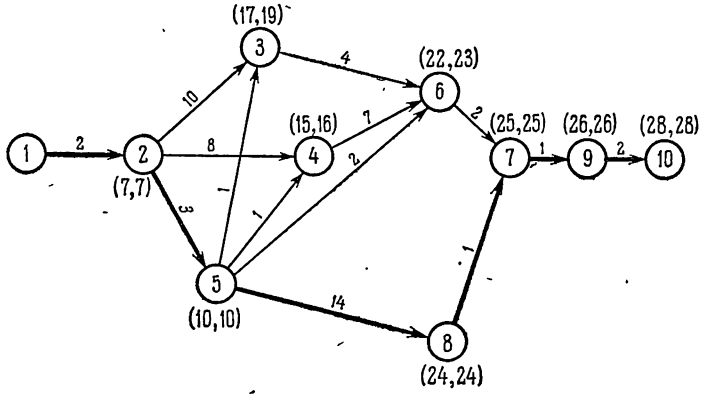

Рис. 33.

работ. Продолжая наши вычисления, придем к тому, что ожидаемое время конечного события — 28 дней со дня начала работ.

Первые числа в паре при вершинах на рис. 33 — величины t_i.

Второй этап алгоритма — вычисление предельных времен t_i^* наступлений событий i, превышение которых приводит к нарушению общего срока завершения всего комплекса работ. Если предположить, что для всех событий, следующих за событием i, предельные времена

определены, то t_i^* определится как минимум по всем работам, начинающимся с события i, из разностей

$$t_j^* - t_{ij}.$$

Для событий *10*, *9*, *7*, *8*, *6*, *4*, каждое из которых является началом лишь единственной работы, величины t_j^* определяются однозначно (по определению t_j^* вычисления необходимо вести с конца сети, начиная с завершающего события):

$$\begin{aligned} t_{10}^* &= 28 \\ t_9^* &= t_{10}^* - t_{9,10} = 28 - 2 = 26 \\ t_7^* &= t_9^* - t_{7,9} = 26 - 1 = 25 \\ t_8^* &= t_7^* - t_{8,7} = 25 - 1 = 24 \\ t_6^* &= t_7^* - t_{6,7} = 25 - 2 = 23 \\ t_4^* &= t_6^* - t_{4,6} = 23 - 7 = 16 \end{aligned}$$

Но для события *5* уже предстоит выбор минимальной из величин

$$t_4^* - t_{4,6},\ t_6^* - t_{5,6} \text{ и } t_8^* - t_{5,8}$$

по трем работам, начинающимся после наступления события *5*. Действительно, событие *5* может наступить несколько позднее ожидаемого времени t_5; это не сорвет сроки наступления последнего события t_{10} и, следовательно, всего комплекса работ, пока задержка наступления события не превзойдет наиболее близкую из дат:

$$t_4^* - t_{5,4},\ t_6^* - t_{5,6},\ t_8^* - t_{5,8}.$$

Итак,

$$\begin{aligned} t_5^* &= \min\,(t_4^* - t_{5,4},\ t_6^* - t_{5,6},\ t_8^* - t_{5,8}) = \\ &= \min\,(16 - 1,\ 23 - 2,\ 24 - 14) = 10 \end{aligned}$$

есть предельное время t_5 — критическое время, превышение которого недопустимо. Продолжая вычисления дальше, можно вычислить t_i^* для остальных вершин сети. Вторые числа в паре при вершинах сети на рис. 33 — величины t_i^*.

Вообще говоря, t_i и t_i^* могут различаться; в этом случае $t_i^* > t_i$, и неотрицательные разности $t_i^* - t_i$ называют резервом времени события i. Так, событие *6* на рис. 33 имеет ожидаемое время $t_i = 22$, а предельное $t_i^* = 23$. Резерв времени события *6* равен одному дню.

Те вершины, для которых $t_i^* - t_i = 0$, называются *критическими*, а цепь отрезков, соединяющих критические вершины, называется *критическим путем*. На правильно построенной сети критический путь всегда существует, иногда он может оказаться не единственным (это утверждение доказывается в соответствующих руководствах, обосновывающих математическую теорию сетевого планирования). Критический путь определяет узкое место всего комплекса работ. В нашем примере после вычисления сначала t_i, а затем t_i^* критический путь определяется как цепь событий

$$1—2—5—8—7—9—10. \qquad (7.5)$$

На рис. 33 критический путь (7.5) выделен жирной линией. В нашем примере узким местом явилась подготовка данных. Это естественно, когда подготавливаются к решению большие задачи обработки данных: перфорация и контроль больших информационных массивов для этого класса задач часто определяют общие сроки решения задачи.

Картина изменится, если работа (*5,8*) будет разделена между двумя группами, работающими параллельно. Это приведет к появлению двух событий: *11* — «окончание перфорирования данных в первой группе» и *12* — «окончание перфорирования во второй группе». Они завершают работы (*5,11*) — перфорирование в первой группе и (*5,12*) — перфорирование во второй группе — и начинают работы (*11,8*) — передача отперфорированного в первой группе материала руководителю службы подготовки данных и (*12,8*) — передача результатов второй группы руководителю службы. Длительность подготовки данных сокращается почти вдвое, и критическим путем становится

$$1—2—4—6—7—9—10.$$

Схема, соответствующая этой ситуации, приведена на рис. 34. Читателю, имеющему вкус к вычислительной работе, представляется возможность убедиться в справедливости этого утверждения.

На практике приходится сталкиваться с сетевыми графиками строительства крупных объектов, отладки

сложных систем. Эти графики имеют сотни и тысячи событий. Определение критических путей для таких графиков вручную становится невозможным.

Предлагаемый алгоритм предполагает обработку сетей, имеющих до 250 вершин и до 1000 работ. К ограничениям сети относятся такие условия: одним событием могут начинаться или могут заканчиваться не более десяти работ. Вершина, или событие, это вектор, состоящий из четырех элементов: номер события i, ожидаемое

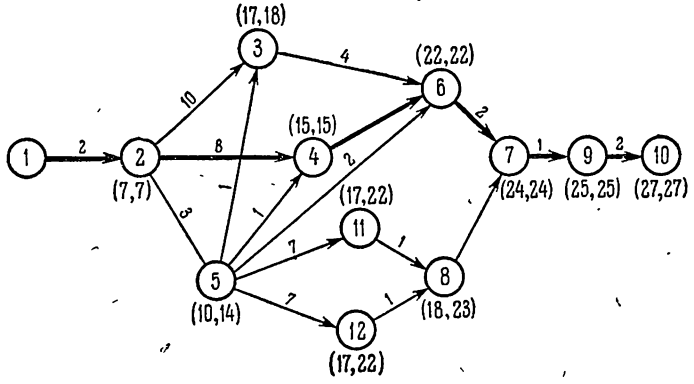

Рис. 34.

время t_i наступления события i, предельное время t_i^* наступления события i и признак p_i [1]). Таким образом, массив SØB событий является двухразмерным: индекс K является номером события в массиве SØB, а индекс L — номером элемента в группе из четырех величин, характеризующих событие. При K = 10 и L = 2 величина SØB (10,2) есть ожидаемое время наступления события, номер которого в массиве SØB равен 10.

Работа — вектор, состоящий из трех действительных величин: номера i события, начинающего работу, номера j

[1]) Все элементы одного вектора должны иметь одну форму представления. Поэтому все они рассматриваются здесь как действительные, хотя номер события и признак никогда не имеют в программе дробной части. Можно было бы принять в качестве единой формы представления целую, если в качестве единицы измерения времени выбрать достаточно короткий интервал времени и величины t_{ij} записывать целыми: в алгоритме не используется оператор деления, и появление новых не целых чисел было бы исключено.

события, завершающего работу, и длительности работы t_{ij}. Массив RAB также двухразмерный: индекс I является номером работы в массиве RAB, а индекс J — номером элемента в группе из трех величин, характеризующих работу. При I = 17, J = 3 величина RAB(17,3) есть длительность работы, номер которой в массиве RAB равен 17.

Размеры основных массивов задачи могут быть описаны инструкцией

```
DIMENSIØN RAB (1000, 3), SØB (250, 4)
```

Нет необходимости полностью перфорировать вектор события на карте в ходе подготовки данных, ибо величины t_i, t_i^* и p_i формируются в процессе работы программы. Однако, ввод номеров событий необходим: он избавляет от ненужных ограничений в нумерации событий.

В числе ограничений остаются лишь естественное условие — не присваивать разным событиям одинаковые номера, и тот факт, что номер — это трехзначное целое число в действительной форме. На карте оно перфорируется по коду F4.0 — код содержит четыре колонки [1]), три из которых содержат цифры и одна — десятичную точку. На одну карту можно записать 20 номеров событий.

Нетрудно принять и следующее ограничение: самым первым перфорируется номер начального события, последним — конечного события сети. NS — количество рассматриваемых событий.

Вектор работы перфорируется на одной карте: две группы колонок по четыре цифры с номером событий и группа из пяти колонок для длительности работ (одна из них — для десятичной точки и одна — для цифры после запятой). NR — количество рассматриваемых работ, т. е. количество вводимых перфокарт массива RAB.

Эти замечания определяют инструкции FØRMAT для ввода данных:

```
для числа событий    101 FØRMAT(I3)
для номеров событий  102 FØRMAT(20F4.0)          } (7.6)
для работ            103 FØRMAT(I3/(2F4.0,F5.1))
```

[1]) Для определенности выбран рисунок перфокарты, описанный в п. 1 §1 гл.IV.

На выходе желательно получить критический путь в виде таблицы: ее строка соответствует критической вершине, а в трех столбцах указаны соответственно номер i вершины, ожидаемое время t_i и предельное время t_i^*.

Порядок вершины в печатаемом критическом пути важен: он начинается с начального события и кончается последним, завершающим. Можно предложить такой рисунок печатаемой строки

104 FØRMAT (T41, F4.0, T61, F6.1, T81, F6.1) (7.7)

Ввод с перфокарт номеров событий обеспечивает лишь заполнение первого элемента вектора события. Чтобы отличить не заполненные величинами t_i и t_i^* места в массиве от нулей (например, $t_1 = 0$ и $t_1^* = 0$ по определению), весь массив SØB перед чтением с перфокарт номеров событий заполняется отрицательными константами, например (—1.).

Первый этап алгоритма — заполнение вторых элементов вектора события, или вычисление t_i. На этом этапе потребуется подпрограмма типа SUBRØUTINE выделения из массива RAB всех тех работ, которые завершаются событием SØB(K,1). Эта подпрограмма ENDR, одним из аргументов которой является номер события, завершающего разыскиваемые работы, записывает все выделенные работы в промежуточный массив RABT для временного хранения. Массив RABT содержит выделенные из массива RAB работы с некоторой дополнительной информацией — ожидаемым временем наступления начального события работы. Если величина NRT есть количество работ в массиве RABT, то индексы двухразмерного массива RABT(MT, NT) изменяются в пределах: MT — от 1 до NRT, NT — от 1 до 4 (номер начального события, номер конечного события, длительность работы, ожидаемое время наступления начального события).

Если для всех работ массива RABT определены ожидаемые времена событий, начинающих эти работы, то можно вычислить ожидаемое время для события K, которым завершаются все работы массива RABT. Если хотя бы одно из начальных событий не определено, следует перейти к построению массива для следующего

события K + 1 и пытаться найти ожидаемое время для него. В силу структуры сети такой перебор обязательно приводит к событию, для которого в массиве RABT все начала работ имеют уже вычисленное ожидаемое время [1]).

Подпрограмма ENDR

```
      SUBRØUTINE ENDR (KØN)
      DIMENSIØN RAB (1000, 3), RABT (10, 4),
     *  SØB (250, 4)
      CØMMØN RAB, SØB, RABT, RABT1,
     *  NR, NS, NRT
      MT = 1
      NRT = 0
      DØ 4 I = 1, NR
      IF (RAB(I, 2) .NE. SØB (K,. 1)) GØ TØ 4
      DØ 1 JT = 1,3
    1 RABT (MT, JT) = RAB (I, JT)
      DØ 2 KT = 1, NS
      IF (RAB (I, 1) — SØB (KT, 1)) 2, 3, 2
    2 CØNTINUE
    3 IF (SØB (KT, 3) .EQ. (— 1.)) GØ TØ 5
      RABT (MT, 4) = SØB (KT, 3)
      NRT = NRT + 1
      MT = MT + 1
    4 CØNTINUE
    5 RETURN
      END
```

---

[1]) Существует алгоритм предварительного упорядочения вершин по уровням, гарантирующий переход от события K к событию K + 1 так, что для каждого последующего события все ему предшествующие уже имеют вычисленные ожидаемые времена. Изложение этого алгоритма было бы весьма громоздким в масштабах этой книги и поэтому опущено в ущерб быстродействию описываемого здесь алгоритма определения критического пути. Описание алгоритма предварительного упорядочения вершин сети по уровням можно найти, например, в книге А. Кофмана и Г. Дебазея «Сетевые методы планирования и их применение» (изд-во «Прогресс», М., 1968).

Ожидаемое время вычисляется по формуле

$$t_i = \max_{j \in U_-} (t_j - t_{ji}), \qquad (7.8)$$

где U_- — множество событий, непосредственно предшествующих событию i. Величины t_{ji} являются длительностями работ массива RABT, а t_j — ожидаемыми временами их начальных событий.

Для вычислений по формуле (7.8) необходима подпрограмма типа FUNCTIØN.

Подпрограмма ØJID вычисления ожидаемого времени:

```
      FUNCTIØN ØJID (R, N)
      DIMENSIØN R (10, 4)
      ØJID = R (1,3) + R (1,4)
      IF (NRT .EQ. 1) GØ TØ 1
      DØ2 MT = 2, N
      IF (ØJID — (R (MT, 3) + R (MT, 4))) 2, 2, 3
    3 ØJID = R (MT, 3) + R (MT, 4)
    2 CØNTINUE
    1 RETURN
      END
```

При всех обращениях к подпрограмме ØJID из основной программы искусственные аргументы — массив R и целая переменная N — заменяются на массив RABT и целую переменную NRT.

Однако, прежде чем вычислить ожидаемое время события, программа должна проверить наличие массива RABT для этого события: он существует, если NRT = 0. Если массив RABT для события не может быть построен, то в признак этого события записывается 0. После построения массива RABT для некоторого события следует проверить все предшествующие события, для которых массив RABT нельзя было построить раньше. Этот этап

алгоритма описывается следующим фрагментом основной программы:

```
      DØ 4 K = 2, NS
      K1 = K — 1
      DØ 1 KT = 2, K1
      IF (SØB (KT, 4).NE. 0.) GØ TØ 1
      CALL ENDR (KT),
      IF (NRT. EQ. 0) GØ TØ 1
      SØB (KT, 4) = — 1.
      SØB (KT, 2) = ØJID (RABT, NRT)
    1 CØNTINUE
      CALL ENDR (K),
      IF (NRT) 2, 3, 2
    2 SØB (K, 2) = ØJID (RABT, NRT)
      GØ TØ 4
    3 SØB (K, 4) = 0.
    4 CØNTINUE
```

(7.9)

На втором этапе надо вычислить третьи элементы в векторах событий — предельные времена t_i^*. Для этого строятся подпрограммы, аналогичные подпрограммам первого этапа — подпрограмма типа SUBRØUTINE, формирующая для каждого события массив RABT1 работ, начинающихся с этого события, и подпрограмма типа FUNCTIØN определения минимума по формуле

$$t_i^* = \min_{j \in U_+} (t_i^* - t_{ij}), \qquad (7.10)$$

где U_+ — множество событий, непосредственно следующих за событием i.

Массив RABT1 аналогичен массиву RABT с той лишь разницей, что места ожидаемых времен начальных событий заняты здесь предельными временами конечных событий соответствующих работ.

Подпрограмма BEGR формирования таблицы RABT1 будет более эффективной, если просмотр массивов RAB и SØB будет выполняться в обратном порядке, начиная

с завершающего события:

```
      SUBRØUTINE BEGR (KØL)
      DIMENSIØN RAB (1000, 3), RABT1 (10, 4),
     *   SØB (250, 4)
      CØMMØN RAB, SØB, RABT, RABT1,
     *   NR, NS, NRT
      MT = 1
      NRT = 0
      DØ 4 I = 1, NR
      MU = NR + 1 — I
      IF (RAB (MU, 1). NE. SØB (KØL, 1) GØ TØ 4
      DØ 1 JT = 1, 3
    1 RABT1 (MT, JT) = RAB (MU, JT)
      DØ 2 KT = 1, NS
      KTU = NS + 1 — KT
      IF (RAB (MU, 2) — SØB (KTU, 1)) 2, 3, 2
    2 CØNTINUE
    3 IF (SØB (KTU, 3). EQ. (— 1.)) GØ TØ 5
      RABT 1 (MT, 4) = SØB (KTU, 3)
      NRT = NRT + 1
      MT = MT + 1
    4 CØNTINUE
    5 RETURN
      END
```

Подпрограмма PRED вычисления предельного времени имеет вид

```
      FUNCTIØN PRED (R, N)
      DIMENSIØN R (10,4)
      PRED = R (1, 3) — R (1,4)
      IF (N.EQ. 1) GØ TØ 1
      DØ 2 MT = 2, N
      IF (PRED — (R (MT, 3) — R (MT, 4))) 3, 3, 2
    2 PRED = R (MT, 3) — R (MT, 4)
    3 CØNTINUE
    1 RETURN
      END
```

Второй этап алгоритма реализуется приводимой ниже частью основной программы:

```
      DØ 14 K = 2, NS
      K1 = NS — K + 1
      DØ 11 KT = 2, K1
      KT1 = NS — K1 + 1
      IF (SØB (KT1, 4) .NE. 0) GØ TØ 11
      CALL BEGR (KT1)
      IF (NRT .EQ. 0) GØ TØ 11
      SØB (KT1, 4) = — 1.
      SØB (KT1, 3) = PRED (RABT1, NRT)
   11 CØNTINUE
      CALL BEGR (K1)
      IF (NRT) 12, 13, 12
   12 SØB (K1, 3) = PRED (RABT1, NRT)
      GØ TØ 14
   13 SØB (K1, 4) = 0.
   14 CØNTINUE
```

(7.11)

Теперь остается лишь выделить критический путь, последовательно отыскивая вершины, для которых $t_i = t_i^*$. Для этого сначала строится промежуточный массив SØBT(I,J) событий, для которых $t_i = t_i^*$ или, в терминах нашей программы, SØB (I,2) = SØB(I,3).

```
      KJ = 1
      DØ 22 K = 1, NS
      IF (SØB (K, 2) .NE. SØB (K, 3)) GØ TØ 22
      DØ 21 J = 1, 4
   21 SØBT (KJ, J) = SØB (K, J)
      KJ = KJ + 1
   22 CØNTINUE
```

(7.12)

Необходимо, чтобы в начале программы была декларирована размерность массива SØBT:

```
      DIMENSIØN SØBT (250, 3)
```

(7.13)

Завершающая часть программы включает инструкции печати критического пути. Здесь используется еще один

промежуточный массив X из трех элементов.

```
      DØ 31 J = 1, 3
   31 X(J) = SØB (1, J)
   32 WRITE (2, 104) (X (J), J = 1, 3)
      DØ 33 I = 1, NR
      IF (RAB (I, 1) .EQ. X (1)) GØ TØ 34
   33 CØNTINUE
      DØ 35 K = 1, NS
      IF (SØB (K, 1) .EQ. RAB (I, 2)) GØ TØ 36
   35 CØNTINUE
   36 IF (SØB (K, 1) — SØB (NS, 1)) 32, 37, 32
   37 WRITE (2, 104) (SØB (NS, J), J = 1, 3)
      STØP
      END
```

(7.14)

Скомпонуем теперь первый сегмент программы — основную программу. После PRØGRAM TRACK следует инструкция DIMENSIØN, имеющая тот же вид, что и инструкция DIMENSIØN подпрограмм ENDR и BEGR. Кроме того, необходимо упомянуть размерности вводимых на последнем этапе промежуточных массивов SØBT (это делается инструкцией (7.13) и X (что можно сделать инструкцией DIMENSIØN X(3)). За инструкциями DIMENSIØN должна следовать инструкция CØMMØN, которая в основной программе будет иметь тот же вид, что и в других сегментах программы. Форматы ввода и вывода определяются инструкциями, имеющими номера 101, 102, 103, 104 ((7.6) и (7.7)). Наконец, следует собственно программа, начинающаяся инструкциями ввода:

```
      READ (1, 101) NS
      DØ 100 K = 1, NS
      DØ 100 L = 1, 4
  100 SØB (K, L) = — 1.
      READ (1, 102) ((SØB (K, L), K = 1, NS)
      READ (1, 103) NR, ((RAB (I, J), J = 1, 3),
     * I = 1, NR)
```

и участки основной программы (7.9), (7.11), (7.12) и (7.14).

После сегмента основной программы размещают сегменты двух подпрограмм типа SUBRØUTINE: ENDR и BEGR и сегменты двух подпрограмм типа FUNCTIØN: ØJID и PRED.

§ 3. Определение пересечения плоских контуров

Во многих задачах обработки геометрической информации очень важно бывает ответить на вопрос: пересекаются ли две определенным образом закодированные плоские фигуры? Рисунок, понятный человеку, всегда дает ясный ответ на этот вопрос, машина же, лишенная зрительного восприятия, должна для этого проделать серию сложных вычислений и сравнений.

Описываемая ниже фортрановская программа TEST определения факта пересечения двух плоских контуров была написана для сложной производственной задачи — составления плана раскроя листового материала на детали произвольной конфигурации. Рассматриваются детали, контуры которых можно сформировать из элементов, являющихся отрезками прямых и дугами окружностей.

Дуга окружности, являющаяся элементом плоского контура, не должна превосходить $\pi/2$. Эта условность легко снимается вспомогательной программой, которая произвольную дугу окружности разбивает на элементы, не превосходящие $\pi/2$ и располагающиеся в пределах одной четверти окружности. Здесь эта программа не рассматривается.

Описываемая программа разрешает рассмотрение контуров, состоящих не более чем из 64 элементов. Встречающиеся в производственной практике технологов-штамповиков детали вполне удовлетворяют ограничениям на вид и количество элементов контура. Впрочем, ничто не мешает рассматривать контуры, состоящие из бо́льшего числа элементов, после простейшего изменения констант в программе.

Каждый элемент задается типом элемента (1.— для отрезка, 2.— для дуги) и координатами начальной (XH, YH) и конечной (XK, YK) точек. Для дуги должны быть указаны, кроме того, координаты (XC, YC) центра дуги, ее радиус R и код выпуклости [1]) (1.— для выпуклых

[1]) Дуга считается выпуклой, если при движении от начальной ее точки к конечной центр дуги остается справа.

дуг, 2.— для вогнутых). Таким образом, элемент контура кодируется девятью величинами, из которых четыре последних для отрезка не имеют значения. Например, плоский контур рис. 35 кодируется табли-

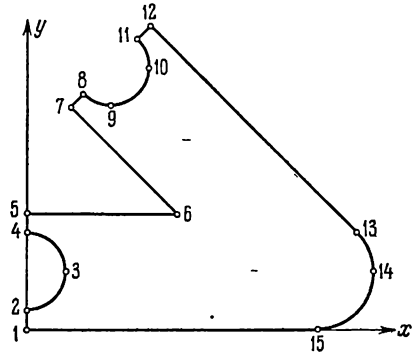

Рис. 35.

цей 13. Номер строчки таблицы 13 соответствует номеру точки контура на рис. 35.

Таблица 13

№	Код элемента	XH	YH	XK	YK	XC	YC	R	Код выпуклости
	1	2	3	4	5	6	7	8	9
1	1.	0.	0.	0.	0.5				
2	2.	0.	0.5	1.	1.5	0.	1.5	1.	2.
3	2.	1.	1.5	0.	2.5	0.	1.5	1.	2.
4	1.	0.	2.5	0.	3.				
5	1.	0.	3.	4.	3.				
6	1.	4.	3.	1.17	5.83				
7	1.	1.17	5.83	1.52	6.18				
8	2.	1.52	6.18	2.23	5.89	2.23	6.89	1.	2.
9	2.	2.23	5.89	3.23	6.89	2.23	6.89	1.	2.
10	2.	3.23	6.89	2.94	7.6	2.23	6.89	1.	2.
11	1.	2.94	7.6	3.29	7.95				
12	1.	3.29	7.95	8.67	4.06				
13	2.	8.67	4.06	9.11	3.0	7.61	3.0	1.5	1.
14	2.	9.11	3.0	7.61	0.	7.61	3.0	1.5	1.
15	1.	7.61	0.	0.	0.				

Определение факта пересечения контуров, описанных двумя кодировочными таблицами TAB1 и TAB2, может

быть сведено к определению наличия пересечения одного из элементов контура 1 с одним из элементов контура 2.

Для этого необходимо иметь подпрограммы, проверяющие наличие пересечения двух отрезков, двух дуг, отрезка из TAB1 с дугой из TAB2 и отрезка из TAB2 с дугой из TAB1. Кроме того, нужна компонующая их основная программа.

Учитывая тот факт, что каждая из перечисленных подпрограмм исполняется в ходе решения обсуждаемой задачи лишь один раз, можно каждую из них представить фрагментом основной программы.

Для размещения координат пересекаемых элементов выделены массивы E1(I), I = 1, 9 и E2(I), I = 1,9 так что

XH1 = E1(2)	XH2 = E2(2)
YH1 = E1(3)	YH2 = E2(3)
XK1 = E1(4)	XK2 = E2(4)
YK1 = E1(5)	YK2 = E2(5)
XC1 = E1(6)	XC2 = E2(6)
YC1 = E1(7)	YC2 = E2(7)
R1 = E1(8)	R2 = E2(8)

В каждом из фрагментов программы определяется величина TEST : TEST = 1. тогда, когда элементы двух контуров и, следовательно, сами контуры пересекаются, и TEST = 0. в противоположном случае.

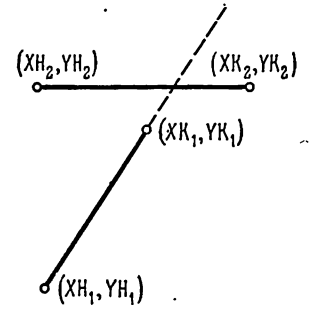

Рис. 36.

1. Фрагмент, проверяющий пересечение двух отрезков, в массивы E1 и E2 записывает координаты начальных и конечных точек отрезков из TAB1 и TAB2 соответственно.

Идею этого фрагмента программы поясняет рис. 36. TEST=0., когда выполняется одно из двух условий — либо обе точки отрезка E1 находятся по одну сторону прямой, на которой лежит отрезок E2, либо обе точки отрезка E2 находятся по одну сторону прямой, на которой находится отрезок E1.

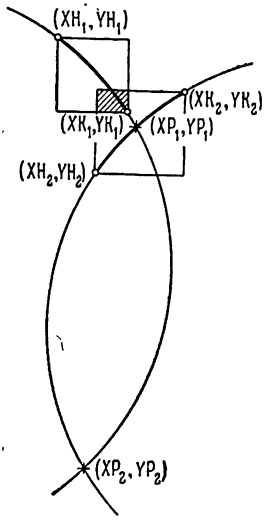

Рис. 37.

2. Фрагмент, определяющий пересечение двух дуг, предусматривает, что массивы E1 и E2 содержат строчки из TAB1 и TAB2, соответствующие двум рассматриваемым дугам. Рис. 37 объясняет этот фрагмент. Если центры окружностей, несущих дуги E1 и E2, находятся на расстоянии, не превышающем сумму радиусов этих окружностей, то определяются точки (XP1, YP1) и (XP2, YP2) пересечения окружностей. Программа проверяет: принадлежит ли хоть одна из этих точек обеим дугам E1 и E2 одновременно. Эта проверка сводится к выяснению факта: попала ли хоть одна из точек (XP1, YP1) или (XP2, YP2) в общую часть прямоугольников, построенных на хордах рассматриваемых дуг как на диагоналях.

3. Фрагмент программы, определяющий наличие пересечения дуги (массив E1) с отрезком (массив E2), проверяет: принадлежит ли хотя бы одна из двух точек пересечения отрезку, а затем в случае отрицательного ответа — принадлежит ли хотя бы одна из точек пересечения дуге. Принадлежность определяется по попаданию точки пересечения в прямоугольники, построенные на отрезке и на хорде дуги как на диагоналях. Такая проверка для дуги, действительно, эквивалентна установлению принадлежности точки дуге в силу принятого ограничения — дуги не превосходят $\pi/2$ и все точки дуги расположены в одной четверти окружности.

Ниже записаны перечисленные фрагменты программы, их заголовки вынесены в комментарии (условно на русском языке).

```
C     ПЕРЕСЕЧЕНИЕ ДВУХ ОТРЕЗКОВ
      1 IF(ABS(E2(4) — E2(2)) — 0.001) 11, 11, 2
      2 R3 = (E2(5) — E2(3))/(E2(4) — E2(2))
        R4 = R3 * (— E2(2)) + E2(3)
        R7 = (R3 * E1(2) + R4 + E1(3)) *
     *         (R3 * E1(4) + R4 + E1(5))
        IF(ABS(E1(4) — E1(2)) — 0.001) 9, 9, 14
     14 R5 = (E1(5) — E1(3))/(E1(4) — E1(2))
        R6 = R5 * (— E1(2)) + E1(3)
        IF(R5 — R3) 9, 15, 9
     15 IF(R6 — R4) 8, 18, 8
     18 IF(((E1(2) .GT. E2(2)) .AND. (E1(4) .GT. E2(2)) .AND.
     *     (E1(2) .GT. E2(4)) .AND. (E1(4) .GT. E2(4))) .ØR.
     *    ((E2(2) .GT. E1(2)) .AND. (E2(2) .GT. E1(4)) .AND.
          (E2(4) .GT. E1(4)) .AND. (E2(4) .GT. E1(4))))
     *     GØ TØ 8
      3 TEST = 1.
        GØ TØ 603
      4 IF(ABS(E1(4) — E1(2)) — 0.001) 6, 6, 5
      5 R3 = (E1(5) — E1(3))/(E1(4) — E1(2))
        R4 = E1(3) — E1(2) * R3
        R7 = (R3 * E2(2) + R4 + E2(3)) *
     *         (R3 * E2(4) + R4 + E2(5))
        IF(ABS(E2(4) — E2(2)) — 0.001) 7, 7, 12
     12 R5 = (E2(5) — E2(3))/(E2(4) — E2(2))
        R6 = R5 * (— E2(2)) + EL2(3)
        IF(R5 — R3) 7, 15, 7
      7 IF(R7) 3, 3, 8
     17 R7 = (E1(2) — E2(2)) * (E1(4) — E2(2))
      9 IF(R7) 4, 4, 8
      6 R7 = (E2(2) — E1(2)) * (E2(4) — E1(2))
        GØ TØ 7
     11 IF(ABS(E1(4) — E1(2)) — 0.001) 13, 13, 17
     13 IF(E1(4) — E2(4)) 8, 16, 8
```

```
 16 IF(.NØT. (((E1(3) .GT. E2(3)) .AND. (E1(5) .GT. E2(3))
  *    .AND. (E1(3) .GT. E2(5)) .AND. (E1(5) .GT. E2(5)))
  *    .ØR. ((E2(3) .GT. E1(3)) .AND. (E1(3) .GT. E1(5))
  *    .AND. (E2(5) .GT. E1(3)) .AND. (E2(5) .GT. E1(5)))))
  *    GØ TØ 3
  8 TEST = 0.
    GØ TØ 603
```

С ТОЧКИ ПЕРЕСЕЧЕНИЯ ДВУХ ДУГ

```
100 IF(E1(6) — E2(6)) 104, 103, 104
103 YP1 = (E1(8) ** 2 — E2(8) ** 2 + E2(7) ** 2 — E1(7) ** 2)/
  *         (2. * (E2(7) — E1(7)))
    YP2 = YP1
    XP1 = E1(6) + SQRT(E1(8) ** 2 — (YP1 — E1(7)) ** 2)
    XP2 = E1(6) — SQRT(E1(8) ** 2 — (YP1 — E1(7)) ** 2)
    GØ TØ 101
104 FI = ATAN ((E2 (7) — E1 (7))/(E2 (6) — E1 (6)))
    IF (E2 (7) — E1 (7)) 105, 105, 106
105 R7 = — 1.
    GØ TØ 107
106 R7 = 1.
107 R8 = SQRT ((E1 (6) — E2 (6)) ** 2 + (E1 (7) — E2 (7)) ** 2)
    RX = (E1 (8) ** 2 ) + R8 ** 2 — E2 (8) ** 2)/(2 * R8) * R7
    RY1 = SQRT (E1 (8) ** 2 — RX ** 2) * R7
    RY2 = — RY1
    XP1 = E1 (6) + RX * CØS (FI) — RY1 * SIN (FI)
    YP1 = E1 (7) + RX * SIN (FI) + RY1 * CØS (FI)
    XP2= E1 (6) + RX * CØS (FI) — RY2 * SIN (FI)
    YP2 = E1 (7) + RX * SIN (FI) + RY2 * CØS (FI)
101 GØ TØ 202
```

С ПЕРЕСЕЧЕНИЕ ДВУХ ДУГ

```
200 R7 = SQRT ((E1 (6) — E2 (6)) ** 2 + (E1 (7) — E2 (7)) ** 2)
    IF (R7 .EQ. 0.) GØ TØ 215
    IF (R7 — (E1 (8) + E2 (8))) 100, 100, 201
201 TEST = 0.
    GØ TØ 603
202 IF (.NØT. ((((XP1 — E1 (2)) * (XP1 — E1 (4))) .LE. 0.)
  *    .AND. (((XP1 — E2 (2)) * (XP1 — E2 (4))) .LE. 0.)
  *    .AND. (((YP1 — E1 (3)) * I(YP1 — E1(5))) .LE. 0.)
  *    .AND. (((YP1 — E2(3)) * (YP1 — E2(5))) .LE. 0.)
  *    GØ TØ 205
```

```
209 TEST = 1.
    GØ TØ 603
205 IF ((((XP2 — E1(2)) * (XP2 — E1(4))) .LE. 0.) .AND.
   *     (((XP2 — E2(2)) * (XP2 — E2(4))) .LE. 0.) .AND.
   *     (((YP2 — E1(3)) * (YP2 — E1(5))) .LE. 0.) .AND.
   *     (((YP2 — E2(3)) * (YP2 — E2 (5))) .LE. 0.))
   * GØ TØ 209
     GØ TØ 201
215 IF (E1(8) — E2(8)) 201, 216, 201
216      IF (((E1(2) .GT. E2(2)) .AND. (E1(4) .GT. E2(2))
   *  .AND. (E1(2) .GT. E2(4)) .AND. (E1(4) .GT. E2(4)))
   *  .ØR. ((E2(2) .GT. E1(2)) .AND. (E2(2) .GT. E1(4))
   *  .AND. (E2(4) .GT. E1(2)) .AND. (E2(4) .GT. E1(4))))
   *  GØ TØ 201
       GØ TØ 209
```

С ТОЧКИ ПЕРЕСЕЧЕНИЯ ДУГИ И ОТРЕЗКА

```
300 IF (ABS (E1(4) — E2(4)) — 0.001) 301, 301, 302
301 R7 = SQRT (E2(7) ** 2 — (E1(2) — E2(6)) ** 2 —
   *              E2(7) ** 2 + E2(8) ** 2)
    YP1 = E2(7) + R7
    YP2 = E2(7) — R7
    XP1 = E1(2)
    XP2 = XP1
    GØ TØ 404
302 R7 = (E1(5) — E1(3))/(E1(4) — E1(2))
    R8 = — R7 * E1(2) + E1(3)
    A = 1. + R7 ** 2
    B = 2. * (R7 * (R8 — E2 (7)) — E2(6))
    C = E2(6) ** 2 + (R8 — E2(7)) ** 2 — E2(8) ** 2
    YP1 = (SQRT (B ** 2—4. * A * C) — B)/(2. * A)
    YP2 = (SQRT (B ** 2 — 4. * A * C) + B)/(— 2. * A)
    XP1 = (YP1 — R8)/R7
    XP2 = (YP2 — R8)/R7
    GØ TØ 404
```

```
C ПЕРЕСЕЧЕНИЕ ДУГИ И ОТРЕЗКА
  400 IF (ABS (E2(5) — E2(3)) — 0. 001) 401, 401, 404
  401 IF (E2(8) — ABS (E2(7) — E2(3))) 300, 300, 403
  404 IF (((((XP1 .LE. E1(4)) .AND. (XP1 .GE. E1(2)))
     *  .AND. ((YP1 .LE. E1(5)) .AND. (YP1 .GE. E1(3))))
     *  .ØR. (((XP2. LE. E1(4)) .AND. (XP2 .GE. E1(2)))
     *  .AND. ((YP2 .LE. E1(5)) .AND. (YP2 .GE. E1(3)))))
     *  .ØR. ((((XP1 .LE. E2(4)) .AND. (XP1 .GE. E2(2)))
     *  .AND. ((YP1 .LE. E2(5)) .AND. (YP1 .GE. E2(4))))
     *  .ØR. (((XP2 .LE. E2(4)) .AND. (XP2 .GE. E2(2)))
     *  .AND. ((YP2 .LE. E2(5)) .AND. (YP2 .GE. E2(3))))))
     *  GØ TØ 405
  403 TEST = 0.
      GØ TØ 603
  405 TEST = 1.
      GØ TØ 603
```

4. Наконец, программа, компонующая перечисленные фрагменты, фиксирует строчку таблицы TAB1, перебирает все строки таблицы TAB2 и разыскивает пересечения элементов контура. Если пересечение не обнаружено, выбирается следующая строка таблицы TAB1 и т. д. Программа выдает ответ TEST = 1., как только обнаружено первое пересечение элементов. (В том конкретном применении, где была использована эта программа — составление плана раскроя в условиях массового и многосерийного производства,— контуры в таблицах TAB1 и TAB2 были конгруэнтны, и потому случай расположения одного контура целиком внутри другого исключен.) Обе эти таблицы имеют одинаковое количество строк — N. Строка таблицы, содержащая 9 элементов, вписывается в одну перфокарту. Результат программы — печать литералов 'INTERSECTIØN' (ЕСТЬ ПЕРЕСЕЧЕНИЕ) и 'NØ INTERSECTIØN' (НЕТ ПЕРЕСЕЧЕНИЯ).

```
      PRØGRAM TEST
      DIMENSIØN TAB1 (64,9), TAB2(64,9),
     * E1(9), E2(9), P(9)
  501 FØRMAT (1X, I2)
  502 FØRMAT (1X, F3.1, 7F9.3, F3.1)
  503 FØRMAT (T21, 'INTERSECTIØN')
  504 FØRMAT (T21, 'NØ INTERSECTIØN')
      READ (5, 501) N
      READ (5, 502) ((TAB1 (I, J), J = 1, 9), I = 1, N),
     *              ((TAB2 (I, J), J = 1, 9), I = 1, N)
      DØ 608 I1 = 1, N
      DØ 608 I2 = 1, N
      DØ 601 J = 1, 9
      E1 (J) = TAB1 (I1, J)
  601 E2 (J) = TAB2 (I2, J)
      IF (E1(1) — E2(1)) 604, 602, 604
  602 IF (E1(1) — 1.) 200, 1, 200
  603 IF (TEST) 609, 608, 609
  604 IF (E1(1) — 1.) 605, 400, 605
  605 DØ 606 J = 1, 9
      P (J) = E2 (J)
      E2 (J) = E1 (J)
  606 E1 (J) = P (J)
      GØ TØ 400
  608 CØNTINUE
      WRITE (3, 504)
      GØ TØ 610
  609 WRITE (3,503)
  610 STØP
      END
```

ЛИТЕРАТУРНЫЕ УКАЗАНИЯ

К настоящему времени на русском языке есть три перевода описания Фортрана. Первый из них — небольшая, но содержательная статья И. С. Пайля «Общее описание языка Фортран» в выпуске № 1 сборника «Современное программирование» (изд-во «Советское радио», Москва, 1966). Второй — большая книга Д. Мак-Кракена и У. Дорна «Численные методы и программирование на Фортране» (изд-во «Мир», Москва, 1969). Изложение языка в этой книге привязано к достаточно четко очерченной области применений. Наконец, недавно переведена книга Ф. П. Фишера и Д. Ф. Суиндла «Системы программирования» (изд-во «Статистика», Москва, 1971), одна глава которой называется «Фортран».

Однако, библиография по Фортрану на английском языке сегодня уже практически необъятна. Чтобы не выписывать здесь длинный перечень статей, отчетов и книг, отошлем читателя к не переведенной еще на русский язык фундаментальной монографии Ж. Саммета «Языки программирования» (Jean E. Sammet. «Programming Languages: History and Fundamentals» Prentice-Hall. Series in Automatic Computation, 1969). Этот весьма полный анализ современных алгоритмических языков содержит, в частности, главу, посвященную Фортрану, которая заканчивается библиографическим списком из 40 названий.

Читателю, познакомившемуся с основами языка и имеющему необходимость углубить свои знания в обстановке конкретной машины, конкретного транслятора и конкретной системы программирования, надо рекомендовать соответствующие руководства. Таким углубляющим конкретным материалом могла бы служить, напри-

мер, появившаяся в 1970 г. книга «Фортран и мониторная система» (изд-во «Статистика», Москва, 1970). Однако следует предупредить читателя, что авторы этой книги — С. Н. Соколов, Л. В. Беляевская, П. А. Калинченко, И. С. Лукашина, В. А. Макаров, А. С. Марков, И. В. Попова — не предназначают ее для знакомства с языком, они пишут: «...эта книга — не популярное пособие для изучения Фортрана, а руководство по практическому использованию Системы автоматизации программирования, разработанной в Институте физики высоких энергий (ИФВЭ) для вычислительной машины «Минск-22)» имеющей Фортран в качестве входного языка».

Заканчивая книгу о Фортране, автор выражает надежду, что «Основы Фортрана» смогут подготовить читателя к углубленному знакомству с книгами, перечисленными здесь в этих кратких литературных указаниях.

ОТВЕТЫ К УПРАЖНЕНИЯМ

2.1. 1) A * X + B * Y — C * Z 2) A/X + C/Z

2.2. 1) A/(B * C) 2) A * B/C

2.3. 1) F = AM1 * AM2/R ** 2
2) V = 4./3. * 3.14159 * R ** 3
3) Y = A * X ** 2 + B * X + C 4) F = XM * A

2.4. 1) Показатель после буквы E должен быть целым; 2) показатель в форме E должен быть указан обязательно; 3) разделение групп разрядов точками не разрешается; 4) десятичная запятая недопустима; 5) показатель может иметь не более двух цифр; 6) мантисса всегда действительная.

2.5. X = — 10.

2.6. 2.5 и 2

2.7. 1) Отрицательная константа не разрешена при формировании значения индекса; 2) индекс не может быть нулем; 3) SLØN — не целая переменная; 4) индекс не может быть отрицательным; 5) выражение J — 2. не является целым: константа 2. действительная; 6) порядок «константа + переменная» недопустим; 7) две переменные в одном выражении для вычисления индекса недопустимы; 8) H — не целая переменная.

2.8. ALPHA, LIP, MANTIS, SLØN, TRIØL — действительные; BLØCK, LEV, NUMBER, STEP — целые.

2.1К. 1) 2.8; 2) 3.; 3) 3.; 4) 1.; 5) 4.; 6) 8.; 7) 22.; 8) —1.; 9) —4.; 10) 1..

2.2К. X1 = (A + (B ** 2 — 4. * A * C) ** 0.5)/(2. * A)
X2 = (A — (B ** 2 — 4. * A * C) ** 0.5)/(2. * A)

2.3К. 1) Наименование переменной QUANTITY содержит более шести символов; 2) вместо десятичной точки поставлена запятая; 3) наименование переменной не может начинаться с цифры; 4) не хватает одной открывающей скобки; 5) константа в левой части арифметической инструкции недопустима; 6) арифметическое выражение недопустимо слева от знака присваивания в арифметической инструкции.

2.4К.
```
AK11 = AK1 + AK
AK22 = AK2 + AK
AK12 = — AK
U = (1./2.) * (AK11 + Q1 ** 2 + 2. * AK12 * Q1 * Q2 +
* AK 22 * Q ** 2)
```

2.5К. 1)
```
Z = X ** 2
T = A * Z
U = B * X
V = T + U
Y = V + C
```
2)
```
I = J — 1
N = K * I
C = A + B
Y = C ** N
```
3)
```
X = A/B
Z = X/C
T = D/A
Y = Z + T
```

3.1.
```
    IF (X — 10.) 70, 1, 70
1   IF (Y — 10.) 70, 2, 70
2   IF (Z — 10.) 70, 60, 70
```

3.2.
```
    IF (X — Y) 6, 6, 5
5   IF (X — Z) 3, 1, 1
6   IF (X — Z) 4, 2, 2
```

3.3.
```
    I = 0
1   I = I + 1
    LIST (I) = I
    IF (I — 1200) 1, 2, 2
2   продолжение программы
```

3.4.
```
    AMN = X (1)
    DØ 4 I = 2, 1000
    IF (X (I)) 1, 2, 2
1   X (I) = — X (I)
2   IF (X (I) — AMN) 3, 4, 4
3   AMN = X (I)
4   CØNTINUE
```

3.5.
```
    DØ 1 I = 1, 100
    J = 101 — I
1   Y (J) = X (I)
```

3.6.
```
    N = 0
    DØ 1 I = 1, 500
    IF (X (I)) 1, 1, 2
2   N = N + 1
1   CØNTINUE
```

3.7.
```
    DØ 1 I = 1, N
    DØ 1 J = 1, N
1   S (J, I) = T (I, J)
```

3.8.
```
    DØ 1 I = 1, 10
    GØ TØ (1, 2, 3, 4, 5, 6, 7, 8, 9, 10), I
```

3.9.
```
    GØ TØ KLIN (10, 20, 30)
10  S = A * B
    GØ TØ 40
20  S = A * B/2.
    GØ TØ 40
30  S = 3.14159 * R ** 2
    GØ TØ 40
    . . . . . .
40  продолжение программы
```

3.10.
```
    .NØT. A — B .GT. A * B = .TRUE.
    (A — B) ** 2 .EQ. 0. .AND. .NØT. A .LT. B = .FALSE.
```

3.11.
```
      AMN = X (1)
      DØ 4 I = 2, 1000
      IF (X (I) .LT. 0.) X (I) = — X (I)
      IF (X (I) .LT. AMN) AMN = X (I)
    4 CØNTINUE
```

3.12.
```
      DØ 1 I = 1, N
      IF (X (I)) 3, 2, 3
    2 PAUSE 55
      GØ TØ 4
    3 Y (I) = 1./X (I)
    4 CØNTINUE
      PAUSE 44
```

3.1К. 1) Переменная имеет слишком длинное наименование; 2) отсутствует запятая между скобкой и переменной; 3) нижняя граница индекса превосходит верхнюю; 5) слишком много номеров инструкций; 6) на месте индекса указана плавающая запятая; 8) слишком много констант в правой части определения индекса; 9) лишняя запятая после закрывающей скобки; 10) отсутствует точка справа от логического оператора.

3.2К. 1) 18, 2) 921, 3) 72, 4) 34, 5) 95, 6) 12, 7) 2, 8) 21, 9) 10, 10) 5.

3.3К.
```
      X = 0.
      DØ 1 I = 1, 100
      X = X + 1.
    1 TAB (I) = 1. + X + X ** 2/2. + X ** 3/6. + X ** 4/24.
```

Номер инструкции	5	10	50	55	100	101	75	102	76	103	500	1000
Количество повторений	1	1	100	100	80	20	20	60	60	20	100	1

3.5К.
```
      IF (XN — XC) 101, 100, 102
  100 IF (YN — YC) 3, 3, 1
  101 IF (YN — YC) 3, 2, 2
  102 IF (YN — YC) 4, 4, 1
    1 MØN = 1
      GØ TØ 5
    2 MØN = 2
      GØ TØ 5
    3 MØN = 3
      GØ TØ 5
    4 MØN = 4
    5 IF (XK — XC) 201, 200, 202
  200 IF (YK — YC) 14, 14, 12
  201 IF (YK — YC) 13, 13, 12
  202 IF (YK — YC) 14, 11, 11
   11 MØK = 1
      GØ TØ 15
   12 MØK = 2
      GØ TØ 15
```

```
      13 MØK = 3
         GØ TØ 15
      14 MØK = 4
      15 продолжение программы
```

4.1.
```
         READ (6, 1) A, B, C, D, E, F, G, H
```

4.2.
```
         DØ 1 I = 1, 50
       1 READ (4, 25) X (I), Y (I), Z (I)
```

4.3.
```
         READ (3, 10) L, (X (J), J = 1, L)
```

4.4.
```
         READ (5, 10) (Y (J), J = 1, 200, 2)
```

4.5.
```
       1 FØRMAT (F6.2)
       1 FØRMAT (F10.3)
       1 FØRMAT (F5.0)
```

4.6.
```
       2 FØRMAT (5F10.4)
```

4.7.
```
         READ (2, 1) (R(I), I = i, N)
       1 FØRMAT (13F 6.2)
```

4.8.
```
       1) 3456700
       2) 0.034567
       3) — 3456789
       4) 3456.7
       5) 345678.
       6) 34567.8
```

4.9.
```
       1 FØRMAT (I6, F5.2)
       2 FØRMAT (I6, E7.2)
         READ (2, 1) (N (IS), Z (IS), IS = 1, 102), (N (IP),
        * Z (IP), IP = 1, 126)
         READ (2, 2) (N (IT), Z (IT), IT = 1, 95)
```

4.10.
```
       1 FØRMAT (10E8.2)
         READ (2, 1) (T (I), I = 1, 200)
         READ (5, 1) (T (I), I = 201, 1000)
```

4.11.
```
       1 FØRMAT (3F8.3)
         WRITE (1, 1) (X (I), Y (I), Z (I), I = 1, 150)
```

4.12.
```
      10 FØRMAT (4 (A3, A4, A1, I5, A3))
         WRITE (1,10) (A, B (I), Y, C (I), X, I = 1, 4)
```

4.13.
```
       1 FØRMAT (10X, 'SALARY' 4X, F6.2, 9X, 'PRIZE';
        * 5X, F6.2, 'TAX' F6.2)
       2 FØRMAT (T11, 'SALARY', T21, F6.2, T36, 'PRIZE',
        * T46, F6.2, T61, 'TAX', T71, F6.2)
```

4.1К.
```
       1 FØRMAT (2A4, A2, T27, I6)
       2 FØRMAT (2A4, A2, I6)
       3 FØRMAT ('0', I6)
       4 FØRMAT ('1', T54, 'STØCKb', 2A3, A2/(T49, 2A4,
        * A2, 10X, I6))
       5 FØRMAT (2A3, A2)
         READ (3, 1) (ART0 (I), K0 (I), I = 1, 1000)
         READ (1, 2) (ART1 (J), K1 (J), J = 1, 200)
         J = 1
```

```
      DØ 8 I = 1, 1000
      IF (ART0 (I) .NE. ART1 (J)) GØ TØ 8
      K0 (I) = K0 (I) + K1 (J)
      J = J + 1
    8 CØNTINUE
      READ (2, 2) (ART2 (J), K2 (J), J = 1, 150)
      J = 1
      DØ 6 I = 1, 1000
      IF (ART0 (I) .NE. ART2 (J)) GØ TØ 6
      K0 (I) = K0 (I) — K2 (J)
      J = J + 1
      IF (K0 (I)) 6, 7, 6
    7 PAUSE 'SØS'
      WRITE (7, 3) ART0 (I)
    6 CØNTINUE
      READ (2, 5) DAY, MØNTH, YEAR
      WRITE (7, 4) DAY, MØNTH, YEAR, (ART0 (I),
     * K (I), I = 1, 1000)
      STØP
      END
```

5.1.
```
      EX = C * 2.8284 * A * CØS (3.1416 * Ø * SIN (FI))/
     * (CØS (FI) — CØS (FIL))
```

5.2.
```
      S = AMIN1 (Z, 100.)
```

5.3.
```
      STRAP (AL, AK, ALPHA) = (AK
     * — AL * CØS (ALPHA)) * AL * SIN (ALPHA)
```

5.4.
```
      FUNCTIØN ISUM (LIST, L)
      DIMENSIØN LIST (1000)
      ISUM = 0
      DØ 1 I = 1, L
    1 ISUM = ISUM + LIST (I)
      RETURN
      END
```

5.5.
```
      SUBRØUTINE SØRT (T, N)
      DIMENSIØN T (4000)
      DØ 2 I = 1, N
      K = N + 1 — I
      DØ 2 J = 1, K
      IF (T (J) — T (J + 1)) 2, 2, 1
    1 R = T (J)
      T (J) = T (J + 1)
      T (J + 1) = R
    2 CØNTINUE
      RETURN
      END
```

5.6.
```
      DIMENSIØN A (100), B (100), C (100), R (100),
     * T (100), S (100)
    1 FØRMAT (1X, 10F5.1)
```

```
2 FØRMAT (2F5.1)
  READ (5, 1) (A (I), I = 1, 100), (B (I), I = 1, 100),
     (C (I), I = 1, 100)
  READ (5, 2) ALPHA, BETA
  CALL VSV (A, B, R, 100)
  CALL CKV (ALPHA, R, T, 100)
  CALL VDV (A, B, R, 100)
  CALL CKV (BETA, R, S, 100)
  CALL VSV (T, S, R, 100)
  CALL VSV (R, C, R, 100)
  WRITE (6, 1) (R (I), I = 1, 100)
  STØP
  END
  SUBRØUTINE VSV (X, Y, Z, K)
  DIMENSIØN X (1000), Y (1000), Z (1000)
  DØ 1 I = 1, K
1 Z (I) = X (I) + Y (I)
  RETURN
  END
  SUBRØUTINE VDV (X, Y, Z, K)
  DIMENSIØN X (1000), Y (1000), Z (1000)
  DØ 1 I = 1, K
1 Z (I) = X (I) — Y (I)
  RETURN
  END
  SUBRØUTINE CKV (FI, X, Z, K)
  DIMENSIØN X (1000), Z (1000)
  DØ 1 I = 1, K
1 Z (I) = FI * X(I)
  RETURN
  END
```

5.1K.
```
  PØINT = AMIN0 (MAX0 (I1, I2, I3, I4, I5),
 *  MAX0 (J1, J2, J3))
```

5.2K.
```
  IF (ABS (AMIN1 (ALPHA, BETA, GAMMA) .LT.
 *  3.141593/2.) GØ TØ 1
```

5.3K.
```
  TG4 (ALPHA) = SIN (4. * ALPHA)/(CØS (ALPHA)
 *** 4—3./2. * SIN (2. * ALPHA) ** 2 + SIN (ALPHA)
 *  ** 4) .
```

5.4K.
```
  FUNCTIØN CØEF (PLAN, ØUTPUT)
  DIMENSIØN PLAN (175), ØUTPUT (175), R (175)
  DØ 1 I = 1, 175
1 R (I) = ØUTPUT (I)/PLAN (I)
  S = 0.
  DØ 2 I = 1, 175
2 S = S + R (I)
  CØEF = (S/175.) * 100.
  RETURN
  END
```

```
5.5K.     SUBRØUTINE INVER (MATR1, MATR2)
          DIMENSIØN MATR1 (20, 30), MATR2 (20, 30)
          DØ 1 I = 1, 20
          K = 21 — I
          DØ 1 J = 1, 30
        1 MATR2 (K, J) = MATR1 (I, J)
          RETURN
          END
6.1K.     EQUIVALENCE (LIST (1), FILE (11))
6.2K.     CØMMØN TAB1, TAB2, A, B, C, D, E, F, KØL,
         * LIM, NUL, MER
```

Необходимо иметь по одной инструкции в каждой подпрограмме и в основной программе.

Юрий Абрамович Первин
ОСНОВЫ ФОРТРАНА
(Серия: «Библиотечка программиста»)
М., 1972 г., 216 стр. 38 рис.
Редактор *Г. Я. Пирогова*
Техн. редактор *К. Ф. Брудно*
Корректоры *О. А. Сигал, В. П. Сорокина*

Сдано в набор 25/XI 1971 г. Подписано к печати 30/III 1972 г. Бумага 84×108$^{1}/_{32}$, тип. № 2. Физ. печ. л. 6,75. Условн. печ. л. 11,34. Уч.-изд. л. 11,18. Тираж 50 000 экз. Т-06117. Цена книги 77 коп. Заказ № 64.

Издательство «Наука»
Главная редакция
физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15

Ордена Трудового Красного Знамени Ленинградская типография № 1 «Печатный Двор» имени А. М. Горького Главполиграфпрома Комитета по печати при Совете Министров СССР, г. Ленинград, Гатчинская ул., 26.

ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ

117071 Москва, В-71, Ленинский проспект, 15

БИБЛИОТЕЧКА ПРОГРАММИСТА

А. Л. Брудно. Программирование в содержательных обозначениях.

А. Л. Брудно. АЛГОЛ.

Ж. Бертэн, М. Риту, Ж. Ружие. Работа ЭВМ с разделением времени.

С. С. Лавров, Л. И. Гончарова. Автоматическая обработка данных. Хранение информации в памяти ЭВМ.

Ю. А. Первин. Основы Фортрана.

С. С. Лавров. Универсальный язык программирования (АЛГОЛ 60).

Цена 77 коп