БИБЛИОТЕЧКА ПРОГРАММИСТА

Транслятор альфа-6 в системе Дубна

БИБЛИОТЕЧКА
ПРОГРАММИСТА

ТРАНСЛЯТОР АЛЬФА-6 В СИСТЕМЕ ДУБНА

Под редакцией
А. П. ЕРШОВА

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1979

22.18
Т 72
УДК 519.6

Транслятор альфа-6 в системе Дубна. Серия «Библиотечка программиста». Аникеева И. Н., Буда А. О., Васючкова Т. С., Кожухина С. К., Козловский С. Э., Шелехов В. И./Под ред. А. П. Ершова.— М.: Наука. Главная редакция физико-математической литературы, 1979.

Книга является руководством к пользованию транслятором с языка альфа-6 (расширения алгоритмического языка алгол-60), включенным в стандартный набор трансляторов мониторной системы Дубна для ЭВМ БЭСМ-6. В руководстве описываются все языковые и программные средства, предоставляемые транслятором в рамках мониторной системы Дубна, и даются инструкции для их использования.

Т $\frac{20204-179}{053(02)-79}$ 65-79. 1702070000

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Система Альфа-6 представляет собой оптимизирующий транслятор, работающий на ЭВМ БЭСМ-6 в рамках мониторной системы Дубна. Входной язык системы Альфа-6 (в дальнейшем упоминаемый как язык альфа-6) является расширением алгоритмического языка алгол-60 и по силе своих выразительных средств занимает промежуточное место между языком алгол-60 и языками программирования следующего поколения, такими как алгол-68 и PL/1. Основной областью применения языка альфа-6 являются научные и инженерные расчеты, хотя он может быть использован и в других областях, связанных с применением ЭВМ.

Отличительная особенность системы Альфа-6 — высокое качество получаемых программ, что позволяет рекомендовать систему в первую очередь тем математикам, задачи которых по необходимости имеют предельные характеристики. Оттранслированная системой Альфа-6 программа представляет собой модуль загрузки мониторной системы Дубна (МС Дубна). Это позволяет математику записывать свою программу частично на языке альфа-6, частично на языках фортран, алгол, мадлен, бемш с последующей комплексацией оттранслированных программ на уровне языка загрузки средствами мониторной системы Дубна.

С точки зрения пользователя мониторной системы Дубна работа с транслятором Альфа-6 производится аналогично работе с другими трансляторами, включенными в стандартный набор математического обеспечения МС Дубна, такими как Фортран-транслятор или Алгол-транслятор. В то же время система Альфа-6 имеет в своем составе независимые от МС Дубна вспомогательные программные средства: текстовый и поперфокартный редакторы, отладчик, библиотеку стандартных процедур, библиотеку для хранения программ и данных. Эти средства расширяют возможности использования транслятора Альфа-6 и предоставляют дополнительные по отношению к МС

Дубна удобства для трансляции, отладки, модификации и сопровождения программ, написанных на языке альфа-6.

Настоящая книга является Руководством к пользованию системой Альфа-6 и состоит из четырех разделов и приложений.

Первый раздел (главы 1—3) служит для первоначального знакомства с языком альфа-6, с составом и возможностями системы и наиболее распространенными режимами ее использования. Овладев материалом раздела 1, математик в состоянии записать алгоритм задачи на языке альфа-6, собрать пакеты для трансляции и выполнения программы, ее редактирования и работы с библиотеками Альфа-6.

Второй раздел (главы 4—12) содержит полную информацию обо всех работах, производимых с использованием системы Альфа-6, таких как: трансляция программы, подготовка исходных данных, задание печатей и перфорации программы или данных, загрузка оттранслированной программы, комплексация нескольких программ, выполнение программ, редактирование, отладочные изменения, библиотечные работы. Раздел содержит также инструкции по комплектации пакетов, информацию о совместимости отдельных работ, о структуре библиотек и информационных массивов, с которыми работает пользователь системы, примеры эксплуатации системы в нестандартных ситуациях.

Третий раздел (главы 13—16) содержит формальное описание языков, используемых системой. Приводятся синтаксис и семантика языка управления системой, входного языка альфа-6, языка, на котором записываются исходные данные, а также формальные правила комплектации системного пакета.

Четвертый раздел (главы 17—22) является списком сообщений, которые могут быть выданы в процессе работы системы Альфа-6. Тексты сообщений, разбитые на группы и упорядоченные в алфавитном порядке, подробно комментируются, а также приводятся рекомендации по обнаружению и устранению ошибок, вызвавших сообщение.

Приложения, включенные в Руководство, содержат справочный материал о кодировках, правилах подготовки перфокарт, написании основных и служебных слов, режимах работы, правилах комплектации пакетов, ограничениях, стандартном наборе библиотечных процедур и другую информацию, необходимую пользователю во время эксплуатации системы.

В Руководство включен также терминологический словарь, в котором дается толкование наиболее часто используемых в Руководстве понятий.

Начинающему пользователю системы рекомендуется ознакомиться прежде всего с материалом раздела I: изобразительными средствами входного языка, возможностями системы, примерами комплектации пакетов для пуска задачи. Этой информации, вместе со справочным материалом, содержащимся в приложениях, обычно бывает достаточно, чтобы пользоваться системой в наиболее часто встречающихся ситуациях.

Раздел II адресуется тем пользователям, которые уже имеют некоторый опыт работы с системой и хотят более детально познакомиться с ней. При этом необходимо сначала прочитать гл. 4, описывающую функционирование системы в целом, а остальные главы раздела читать выборочно, в зависимости от ситуации, вызвавшей затруднение.

Предполагается, что раздел III будет использоваться в основном как справочный материал, а не как учебник по языкам системы. Пользователь может обратиться к формальным правилам тогда, когда возникают сомнения в синтаксической или семантической правильности употребленной им конструкции языка.

Во введении к разделу III дается формализм для синтаксического описания языков.

К материалу раздела IV следует обращаться тогда, когда непонятны некоторые выдаваемые системой сообщения. Однако, чтобы понять структуру раздела IV и иметь представление о содержащейся в нем информации, рекомендуется сразу же после раздела I прочитать гл. 17.

Приложения и терминологический словарь используются как справочный материал.

В руководстве принята следующая система ссылок:

1) п. 2.10 — глава 2, пункт 10.

2) [4] — внешняя ссылка (возможно с указанием страницы) в данном случае на сборник статей «Альфа-система автоматизации программирования»./Под ред. А. П. Ершова. — Новосибирск: Наука, 1967.

3) В терминологическом словаре указывается страница руководства, где впервые встретился данный термин.

В создании системы Альфа-6 принимали участие: И. И. Аникеева, С. Ф. Богданова, А. О. Буда, Т. С. Васючкова, А. А. Граповский, В. А. Грибачевская, А. В. Ерофеев, А. П. Ершов, С. К. Кожухина, Г. И. Кожухин, С. Э. Козловский, П. А. Ким, И. В. Поттосин, А. Е. Хоперсков, В. И. Шелехов, Т. С. Янчук. Научный руководитель проекта — А. П. Ершов.

Предполагается, что читатель настоящего Руководства знаком с основными принципами работы и правилами эксплуатации МС Дубна по руководствам [19], [13], [11].

Авторы и редактор будут благодарны читателям, если они сообщат о любых замечаниях, неточностях или трудностях в использовании этого справочного руководства по адресу: 630090, Новосибирск, ВЦ СО АН СССР, Отдел программирования.

По этому же адресу можно направить просьбу о консультациях по поводу установки и эксплуатации системы Альфа-6.

РАЗДЕЛ I

НЕФОРМАЛЬНОЕ ВВЕДЕНИЕ

ГЛАВА 1

СОСТАВ И ВОЗМОЖНОСТИ СИСТЕМЫ

Система Альфа-6 включена в стандартный набор трансляторов МС Дубна. Это означает, что всякий вызов системы Альфа-6 оформляется как отдельная задача МС Дубна [13]. После окончания работы системы Альфа-6 управление и результирующие информационные массивы (модуль загрузки и данные) передаются в МС Дубна для дальнейшей обработки (загрузка, комплексация, выполнение программы). Информационные карты для запуска системы Альфа-6 являются составной частью пакета для запуска МС Дубна (см. гл. 3).

Ниже перечисляются компоненты системы Альфа-6 и кратко описывается их назначение.

Язык системных команд служит для задания входной информации, определяющей функционирование системы. Вид информации конкретизируется в разделе IV. Синтаксис и семантика языка описаны в гл. 14.

Язык альфа-6 представляет собой расширение языка алгол-60 конструкциями языков альфа [8], алгамс [2] и содержит подмножество автокода бемш [6]. Записанный на языке альфа-6 алгоритм задачи называется альфа-программой.

Язык данных служит для описания входных данных задачи (числовых и логических значений, строк, констант), которые используются при счете. Записанная на этом языке информация называется альфа-данными.

Транслятор осуществляет перевод альфа-программы с языка альфа-6 в систему команд БЭСМ-6, оформляемую как модуль загрузки МС Дубна. При этом производится ряд оптимизирующих преобразований и контролируется синтаксическая и семантическая правильность программ. Транслятор осуществляет также перевод альфа-данных в машинное представление, которое воспринимается операторами ввода при выполнении программы.

Поперфокартный редактор позволяет осуществлять замену перфокарт в альфа-программе и альфа-данных.

Текстовый редактор позволяет осуществлять замену произвольных текстов или идентификаторов в альфа-программе и альфа-данных.

Отладчик позволяет осуществлять отладочные изменения в альфа-программе в процессе ее трансляции. С помощью отладочных операторов можно получить дополнительную информацию о выполнении программы.

Программа библиотечных работ предназначена для создания библиотек Альфа-6 и работы с ними. Она дает возможность хранить альфа-программы и альфа-данные.

Библиотека стандартных процедур содержит набор написанных на языке альфа-6 процедур, обращения к которым могут быть использованы в альфа-программе. Состав библиотеки устанавливается и регулируется организацией, эксплуатирующей систему.

Препроцессор осуществляет первичные обработку и контроль входной информации, а также видоизменяет программу и данные, подготовленные для систем Алгибр [5] и БЭСМ-алгол [7], что дает возможность использовать такие программы и данные в системе Альфа-6.

Руководство к пользованию — имеется в виду данное Руководство.

Основные технические характеристики системы Альфа-6.

Объем системы (тыс. команд БЭСМ-6)	150
Число просмотров программы при трансляции	11
Средняя скорость генерации команд БЭСМ-6 при трансляции (команд/сек)	70
Стандартные ресурсы:	
число листов ОЗУ	32
число трактов МБ	64
число МЛ (МД)	2
число вводных устройств	1
число устройств вывода на печать	1
число устройств вывода на перфорацию	1

Устройства для подготовки программы и данных — УПП (БЭСМ-6), КУ-3, ЕС, УПП (БЭСМ-4), М-220.

Максимальный объем программы или данных, вводимых с перфокарт (пфк) 1 000 *)

*) Условно предполагаем, что одна перфокарта содержит 40 символов УПП.

Максимальное число символов в данных, хранящихся в архиве 180 000

Максимальное число листов ОЗУ, выделяемое оттранслированной программе 30

ГЛАВА 2

НЕАЛГОЛОВСКИЕ КОНСТРУКЦИИ ЯЗЫКА АЛЬФА-6

Язык альфа-6 является дальнейшим развитием языка альфа [1, 8] и разработан для реализации на ЭВМ БЭСМ-6.

Основными добавлениями по отношению к языку альфа являются: стандартные операторы языка алгамс, возможности записи команд БЭСМ-6 с мнемоническими обозначениями автокода бемш и средства комплексации раздельно транслируемых подпрограмм в рамках МС Дубна.

Строгое описание языка альфа-6 с использованием металингвистических формул приводится в гл. 15.

В настоящей главе описываются расширения языка альфа-6 по отношению к алголу-60 [3]. Описание дается неформально, главным образом на примерах, однако сообщаемые сведения достаточно полны для того, чтобы дать возможность читателю пользоваться новыми конструкциями языка. При написании главы использовался материал руководства [1] (п. 2.1—2.13). В текстах программ используются русские варианты служебных слов (см. Приложение 3).

2.1. Комплексные величины и действия над ними

В языке альфа-6 допускаются переменные величины и выражения, принимающие комплексные значения. Комплексное значение z рассматривается как упорядоченная пара вещественных значений (x, y), причем x и y соответственно равны действительной и мнимой частям значения z.

Для описания комплексных переменных служит описатель типа *комплексный*, который употребляется так же, как и остальные описатели типа в алголе-60.

Все арифметические операции и большинство функций (*sqrt*, *ln*, *exp*, *sin*, *cos*, *tan*, *sh*, *ch*, *th*, *arctan*, *arth*, *arcsin*, *arccos*, *arsh*, *arch*) естественным образом переносятся на комплексные значения аргумента. Поскольку *sqrt*, *ln* и т. д. являются для комплексных аргументов многозначными функциями, их вычисление понимается в смысле получения главного зна-

чения. Для комплексных арифметических выражений определены также отношения $=$ и $\neq$.

Специально для комплексных величин в языке альфа-6 введены следующие стандартные функции комплексного аргумента:

Re(E) — вещественная часть *E*;

Im(*E*) — мнимая часть *E*;

Con(*E*) — число комплексно-сопряженное значению *E*;

Arg(*E*) — аргумент *E* (главное значение);

mod(*E*) — модуль *E*.

mod(*E*) определен также для вещественного (и целого) аргумента, совпадая в этом случае с *abs*.

В языке альфа-6 нет функции преобразования целых и вещественных величин в комплексные, и поэтому при записи оператора присваивания

$$z := E$$

типы переменной z и выражения E должны совпадать, если одно из них является комплексным. Однако при выполнении арифметических действий допустимы всевозможные комбинации типов операндов (целый, вещественный, комплексный), например, (z — комплексная величина)

$z + 1$

$5.13 \times z$

$1/z$

и т. д.

Следует отметить, что для бинарных операций, если хотя бы один из операндов является комплексным, то результат выполнения операции всегда считается комплексным.

Необходимо помнить, что в языке альфа-6 комплексная переменная не может быть параметром цикла.

В некоторых случаях при вычислениях с комплексными величинами приходится осуществлять переход из комплексной области значений в вещественную и наоборот. Этот переход состоит в том, что, с одной стороны, вещественные значения должны в какой-то момент объединяться в пары и рассматриваться как комплексные значения или, с другой стороны, оперируя с комплексными значениями, нужно уметь в какой-то момент оперировать раздельно над их действительными и мнимыми частями. Этим целям в языке альфа-6 служит описание действительной и мнимой частей, имеющее вид

$$z = x + iy,$$

где z — комплексная переменная, а x и y — идентификаторы. Наличие такого описания в программе вводит явные обозна-

чения — x и y — для действительной и мнимой частей комплексной переменной z. Служебное слово i символизирует мнимую единицу, знак $+$ в описании не несет операционной роли, а лишь только придает описанию вид стандартного математического обозначения. Следует отметить, что описание $z = x + iy$ не заменяет описания типа переменной z, которое должно быть дано отдельно. Однако для x и y описание $z = x + iy$ играет роль описания типа, и поэтому вещественные переменные x и y описывать не надо.

Примеры. а) Если имеется описание $z = x + iy$, то для того, чтобы задать z с действительной частью, равной нулю, и с мнимой, равной 5, достаточно записать

$$x := 0; \quad y := 5.$$

б) Операторы $t := Re(z)$ и $t := x$ эквивалентны по результату их выполнения.

в) В математике запись $z = x + iy$ может пониматься и так, что z является результатом умножения y на мнимую единицу с последующим сложением с x. Описание

$$z = x + iy$$

в языке альфа-6 лишено такого операционного смысла и, скажем, запись оператора присваивания

$$z := 0 + i5$$

является в языке альфа-6 недопустимой.

г) Присваивание переменной z значения $0 + i5$ можно записать в следующем виде:

$$i = a + ib;$$

$$b := 1; \quad a := 0; \quad z := 0 + i \times 5.$$

Надо, однако, понимать разницу в реализации этой программы и программы примера а). В случае а) происходит просто засылка нужных величин в ячейки, хранящие действительную и мнимую части переменной z. В последнем же варианте, кроме действий $a := 0$ и $b := 1$ будет происходить умножение i на 5, прибавление нуля к действительной и мнимой частям $i \times 5$ и лишь затем пересылка действительной и мнимой частей аргумента в ячейки для хранения переменной z.

Описание действительной и мнимой частей может использоваться и в случае, когда соответствующая комплексная переменная является массивом. Пусть, например, переменная z имеет описание

комплексный массив $z[1:n, 0:2, -1:3]$.

В этом случае описание действительной и мнимой частей компонент массива z должно иметь вид

$$z[i, j, k] = x[i, j, k] + iy[i, j, k].$$

В качестве обозначений буквенных индексов здесь могут быть взяты любые идентификаторы, безотносительно к тому, используются ли они где-нибудь еще в программе или нет, поскольку их роль здесь только в том, чтобы наглядно подчеркнуть, что x и y тоже должны иметь индексы. Граничные пары у вещественных массивов x и y берутся те же, что и у массива z.

2.2. Многомерные значения и переменные

2.2.1. Предварительные замечания. Пусть $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$, $B = (b_1, \ldots, b_n)$ — векторы n-го порядка и $A = ||aij||$ — матрица порядка $n \times n$, связанные соотношением:

$$X = Y + A^{-1} \times B.$$

Алгоритм получения X на алголе-60 можно записать в виде следующего блока:

```
начало массив d, x, y, b[1:n], a[1:n, 1:n];
                целые i, j, m;
```

Обращение матрицы:

Исходная подготовка матрицы:

```
для i:=1 шаг 1 до n цикл a[i, i]:=a[i, i]—1;
```

Основной цикл обращения матрицы:

```
для m: = 1 шаг 1 до n цикл
начало
```

Перенос строки матрицы с номером m в дополнительный вектор и замена ее на соответствующую строку единичной матрицы:

```
для j: = 1 шаг 1 до n цикл
начало d[j]:= a[m, j]; a[m, j]: = 0 конец; a[m,
m]: = 1;
```

Пересчет матрицы:

```
для i:=1 шаг 1 до n цикл
для j:=1 шаг 1 до n цикл
   a[i, j]:=a[i, j]—a[i, m]×d[j]/(1+d[m])
конец;
для i:=1 шаг 1 до n цикл
начало x[i]:=y[i]; для j:=1 шаг 1 до n цикл
                x[i]:=x[i]+a[i, j]×b[j];
конец
конец блока.
```

На языке альфа-6 получение X можно записать в виде:

начало вещественный B, X, Y — **вектор** n;
вещественный A — **матрица** $n \times n$;
$X := Y + A \uparrow (-1) \times B$
конец.

Сопоставление этих двух блоков демонстрирует одну из основных особенностей языка альфа-6, состоящую в том, что в нем идентификаторы простых переменных могут обозначать, в отличие от алгола-60, не только скалярные значения, но и многомерные значения, или массивы (вектора, матрицы и т. д.), а действия над ними указываются в виде операций над массивами в целом, а не в виде операций над их скалярными компонентами, как в алголе-60.

В связи с рассмотрением многомерных значений и переменных введем несколько понятий. В языке альфа-6 характеристикой некоторого нескалярного значения является его тип (целый, вещественный, комплексный, логический) и структура. Наличие структуры у значения показывает, что данное значение образуется целой совокупностью (множеством) чисел, определенным образом упорядоченных. Структура характеризуется прежде всего размерностью, равной некоторому положительному целому числу. Если значение имеет размерность n, то это значит, что числа, образующие данное значение, расположены в узлах целочисленной решетки, заполняющей в n-мерном координатном пространстве прямоугольный параллелепипед со сторонами длины $l_1, l_2, \ldots, l_n$. Стороны этого параллелепипеда параллельны координатным осям, которые называются *измерениями* данного значения. Измерения упорядочены, т. е. есть 1-е, 2-е и т. д. измерения. 1-е измерение называется *старшим*, а n-е измерение — *младшим*.

Длина l_i стороны параллелепипеда, параллельной i-му измерению, называется длиной или *порядком* значения по данному измерению. Если n-мерное значение имеет порядки по измерениям $l_1, \ldots, l_n$, то это значит, что значение образуется совокупностью $l_1 \times \ldots \times l_n$ чисел. Числа, образующие значение, называются *компонентами* значения.

Одномерное значение называется *вектором*, двумерное — *матрицей*. Значение любой положительной размерности n называется (n-мерным) *массивом*. Переменные, значениями которых являются массивы, сами тоже называются векторами, матрицами или массивами в зависимости от их размерности.

Заметим, что понятие структуры применимо и к массивам алгола-60. В алголе-60 можно описать некоторый массив, компоненты которого изображаются переменными с индексами.

Число индексов указывает размерность массива, их порядок задает упорядоченность измерений. Порядок по i-му измерению равен E_2-E_1+1 в том случае, когда граничная пара в i-м индексе имеет вид $E_1:E_2$.

Как уже отмечалось, основным отличием алгола-60 от языка альфа-6 является то, что в алголе-60 массив в целом может использоваться только в качестве фактического параметра процедуры, а в языке альфа-6 оперировать с массивами можно также и в выражениях, задавая их в виде значений многомерных переменных или вычисляя их с помощью операций над заданными многомерными значениями.

Многомерные переменные вводятся в языке альфа-6 тремя способами:

а) с помощью описания многомерной переменной;

б) с помощью пустых позиций индексов;

в) с помощью геометрических операций формирования и компоновки.

2.2.2. Описание многомерной переменной. Пусть необходимо записать, что переменная A является n-мерным массивом с порядками по измерениям l_1, l_2, ..., l_n. Тогда описание переменной A составляется следующим образом. Сначала пишется описание типа по правилам алгола-60, а затем ставится знак «—» (понимаемый в данном случае как тире), за ним пишется служебное слово **массив**, за которым выписываются порядки по измерениям, разделяемые знаками $\times$.

Пример.

вещественный A — **массив** $2 \times 3 \times 5$;
логический K, i — **массив** n;
целый dim — **массив** $m \times (2 \times n)$;
вещественный массив M, $N[1:n]$ — **массив** $m \times 10$;

Как видно, в качестве порядков по измерениям могут быть не только целые без знака, а любые выражения. Эти выражения должны подчиняться тем же правилам, что и выражения, образующие граничные пары в описаниях массивов алгола-60, т. е. не содержать переменных, локализованных в том блоке, в начале которого помещается данное описание. Кроме того, если выражение, определяющее порядок по измерению, содержит в себе знаки операций, оно должно быть взято в скобки (более точно, это выражение должно быть всегда первичным выражением в смысле алгола-60).

Следует сделать особое замечание в отношении последнего примера. Это описание гласит, что компоненты векторов n-го порядка M и N сами являются матрицами порядка $m \times 10$.

Употребление слова «компоненты» в этой фразе противоречит исходному представлению о том, что компоненты массива — это те скалярные значения, совокупность которых образует массив. В то же время допустимо трактовать иногда, например, некоторый трехмерный массив как матрицу векторов или же как вектор матриц. Для того чтобы привести терминологию в соответствие с указанным словоупотреблением, введем дополнительно следующие понятия. Пусть дан n-мерный массив M. Назовем n *абсолютной размерностью* массива M, а его скалярные компоненты — *абсолютными компонентами*. Разобьем теперь n на любые два слагаемых p и q и будем рассматривать в M его q-мерные подмассивы. Тогда можно сказать, что массив M имеет размерность p по отношению к его q-мерным компонентам. Размерность и компоненты такого рода будем называть *относительной размерностью и относительными компонентами.*

В соответствии с высказанным, мы можем говорить, что массивы M и N являются векторами порядка n только по отношению к своим двумерным компонентам порядка $m \times 10$. Абсолютная размерность массива M и N равна трем, причем старшими, *внешними измерениями* являются измерения, указанные с помощью индексов, а младшими, *внутренними измерениями* являются измерения, указанные в описании структуры. Иначе говоря, упомянутое описание для M и N эквивалентно описанию

вещественный массив $M, N[1:n, 1:m, 1:10]$,

в том смысле, что идентификаторы M и N в обоих случаях связываются с массивом значений одной и той же структуры.

Наконец, осталось заметить, что ради большей наглядности в описаниях многомерной переменной можно использовать вместо служебного слова **массив** служебные слова **матрица** или **вектор.**

Например:

логический k, l — **вектор** n;
вещественный массив $M, N[1:n]$ — **матрица** $m \times 10$.

Описание многомерной переменной позволяет вводить некоторый идентификатор в качестве имени целого массива. Однако наряду с использованием массива в целом может возникнуть потребность также в оперировании с отдельными компонентами этого массива. Например, по некоторой совокупности формул рассчитываются компоненты матрицы, а затем эта матрица обращается, или вычисляется ее определитель.

Этим целям в языке альфа-6 служит описание компонент массива, весьма напоминающее описание действительной и мнимой частей комплексной переменной. Пусть A — переменная размерности n. Тогда описание вида

$$A = \| a[i_1, i_2, \ldots, i_n] \|,$$

где $i_1, i_2, \ldots, i_n$ — любые идентификаторы, вводит явное обозначение $a[i_1, i_2, \ldots, i_n]$ для скалярных компонент массива A. Идентификация той или иной компоненты осуществляется надлежащим подбором индексных выражений в позициях, причем считается, что если A имеет порядок l по i-му измерению, то i-я позиция индекса $a[i_1, i_2, \ldots, i_n]$ имеет граничную пару $1{:}l$. Таким образом, это описание играет также роль описания типа и граничных пар переменной a (тип такой же как у переменной A).

Если переменная A сама имеет m индексов, то описание компонент массива в этом случае имеет вид

$$A[k_1, \ldots, k_m] : = \| a[k_1, \ldots, k_m, i_1, \ldots, i_n] \|.$$

Граничные пары у переменной a по первым m индексам берутся от соответствующих граничных пар переменной A.

Так же, как и в случае комплексных переменных, идентификаторы $k_1, \ldots, k_m, i_1, \ldots, i_n$ могут выбираться совершенно произвольно, так как их роль состоит только в указании числа требуемых индексов.

2.2.3. Пустые позиции индексов. Пусть в программе описана переменная с двумя индексами

массив $a[n{:}N, m{:}M]$.

Массив в целом представляет собой матрицу, а переменная $a[i, j]$ — отдельную компоненту этой матрицы. В языке альфа-6 допустимо использование переменных вида $a[i,]$, $a[,j]$, $a[,]$, которые изображают соответственно вектор, образованный i-й строкой матрицы a, вектор — j-й столбец матрицы a и саму матрицу a в целом. Другими словами, оставление какой-то позиции индекса пустой означает автоматическое повышение размерности переменной с индексами на единицу, причем порядок по этой дополнительной размерности равен количеству различных значений, пробегаемых индексом в этой позиции от его нижней границы до верхней. Если пустыми оставлено r позиций индексов, то считается, что появившиеся r дополнительных измерений упорядочиваются слева направо, т. е. са-

мая левая пустая индексная позиция соответствует старшему измерению.

Если переменная с индексом обладает некоторой внутренней размерностью, т. е. ее описание имеет, например, вид

вещественный $a[1{:}n,\ 1{:}n]$ — **вектор** *10*,

то у переменной $a[,]$ размерности по пустым позициям индекса будут старшими по отношению к внутренней размерности, указанной в описании переменной a.

Следует отметить, что в программе на языке альфа-6, содержащей описание типа

массив $a[n{:}N,\ m{:}M]$,

записи $a[,]$ и a, не являются синонимами, т. е. если где-то a употребляется в качестве фактического параметра в операторе $F(a)$, то его нельзя писать в виде $F(a[,])$ и, наоборот, выражение $a[,]\uparrow(-1)$ нельзя писать в виде $a\uparrow(-1)$.

2.3. Операции над многомерными массивами

Все арифметические и логические операции, а также стандартные функции, определенные в алголе-60 и языке альфа-6 над скалярными значениями, переносятся на многомерные. При этом операции бывают покомпонентными и непокомпонентными.

2.3.1. Покомпонентные операции. Все унарные покомпонентные операции определены для любых массивов, бинарные операции определены либо для массивов одинаковой структуры, либо, в специально оговариваемых случаях, для комбинаций скаляр — любой массив. В качестве результата всегда получается массив той же структуры, что и у операнда (операндов), являющегося массивом.

При этих условиях выполнение покомпонентной операции над массивом (массивами) сводится к выполнению этой операции над его (их) соответствующими абсолютными компонентами и взятия полученного результата в качестве соответствующей абсолютной компоненты результирующего массива.

Покомпонентными операциями в языке альфа-6 являются следующие:

1. Все стандартные функции, кроме *mod*, *det*, *shift*, *max*, *min* (см. ниже).

2. Арифметические операции: $+$, $-$, $/$, $\div$, $\uparrow$ (для нецелого показателя).

3. Логические операции: $\vee$, $\&$, $\neg$, $\supset$, $\equiv$.

4. Отношения: $=$, $\leqslant$, $<$, $>$, $\geqslant$, $\neq$. Для всех отношений, кроме $\neq$, отношение равно **истина**, если оно выполнено для всех пар компонент операндов, и **ложь** — в противном случае. Отношение $\neq$ равно **истина**, если оно выполняется хотя бы для одной пары соответствующих компонент операндов, и **ложь**, если все соответствующие компоненты равны друг другу. Допустимо отношение, одним из операндов которого является нуль (*0*, *0.0* или *.0*), например, $A[,] = 0$.

5. Дополнительные к алголу-60 операции $\circ$ (покомпонентное умножение массивов) и $\oplus$ (логическое сложение по *mod 2*: **ложь**$\oplus$**ложь**$=$**ложь**, **истина**$\oplus$**истина**$=$**ложь**, **истина**$\oplus$**ложь**$=$**истина**, **ложь**$\oplus$**истина**$=$**истина**). При этом по старшинству операций покомпонентное умножение $\circ$ относится к уровню умножения $\times$ и деления /, а сложение по *mod 2* — к уровню операции эквивалентности $\equiv$.

2.3.2. Непокомпонентные операции. К непокомпонентным операциям относятся: *mod* (модуль), *det* (определитель), *shift* (сдвиг), $\times$ (умножение), $\uparrow$(целая степень), *max* (максимум), *min* (минимум).

Модуль определен для целых, вещественных и комплексных массивов любой структуры и дает в качестве результата значение корня квадратного из суммы квадратов модулей компонент типа **вещественный**.

Определитель вычисляется для квадратных целых, вещественных и комплексных матриц и дает в качестве результата вещественное (для целых и вещественных матриц) или комплексное значение детерминанта матрицы.

Сдвиг *shift* (L, n) определен только для логического вектора L и целого числа n. В качестве результата получается вектор той же длины m, что и L, причем если k-я компонента вектора L равна l_k ($1 \leqslant k \leqslant m$), то k-й компонентой вектора shift(L, n) будет l_{k+n}, если $1 \leqslant k+n \leqslant m$, и **ложь** в остальных случаях.

Умножение определено для случаев в табл. 2.1.

Тип результата определяется по тем же правилам, что и при перемножении скаляров.

Целая степень массивов определяется для квадратных матриц и понимается в этом случае как матричное умножение. Отрицательная степень понимается как положительная степень обратной матрицы.

Максимум (или минимум) для многомерного значения в качестве результата дает скалярное значение, равное максимальному (или минимальному) значению из всей совокупности скалярных компонент многомерного аргумента.

Таблица 2.1

Комбинация структуры операндов A и B	Как понимается операция $A \times B$
A и B — скаляры	обычным образом
A и B — векторы одного порядка	скалярное произведение векторов
A — вектор порядка n B — матрица порядка $n \times m$	произведение вектора на матрицу, результат — вектор порядка m
A — матрица порядка $n \times m$ B — вектор порядка m	произведение матрицы на вектор, результат — вектор порядка n
A — матрица порядка $n \times m$ B — матрица порядка $m \times k$	произведение матриц, результат — матрица порядка $n \times k$
A — массив B — скаляр	покомпонентное произведение массива на скаляр
A — скаляр B — массив	

2.4. Геометрические операции

При использовании операций с многомерными значениями иногда возникает необходимость в использовании не всех элементов массива, а какой-то его части, например, если имеется вектор $A[1:10]$ и нужно использовать вектор, состоящий из элементов $A[5]$, $A[6]$, $A[7]$. Более того, иногда бывает нужно создать массив, который состоит из других массивов или скалярных переменных.

В языке альфа-6 эта возможность реализуется применением геометрических операций. Существует два способа образования многомерных переменных и значений: формирование и компоновка.

2.4.1. Формирование. Операция формирования позволяет строить массив из скалярных переменных или из массивов. Самым простым примером формирования является образование вектора из скалярных компонент. Запись вида

$$| x, y, a, b, c |$$

означает, что переменные x, y, a, b, c рассматриваются как

последовательные компоненты нового вектора 5-го порядка. Образование вектора путем явного выписывания всех его компонент и взятие их в «прямые скобки» называется операцией формирования вектора.

Допустим, нужно переменным x, y, a, b, c присвоить соответственно значения переменных $x1$, $y1$, $a1$, $b1$, $c1$. На алголе-60 мы написали бы несколько операторов присваивания:

$$x := x1;\quad y := y1;\quad a := a1;\quad b := b1;\quad c := c1;$$

Средствами языка альфа-6 эти же действия можно записать следующим образом:

$$|x,\ y,\ a,\ b,\ c| := |x1,\ y1,\ a1,\ b1,\ c1|$$

Заметим, что в операции формирования массива могут участвовать только переменные одного и того же типа и структуры, т. е. с одинаковой размерностью и длинами по соответствующим измерениям (рис. 2.1).

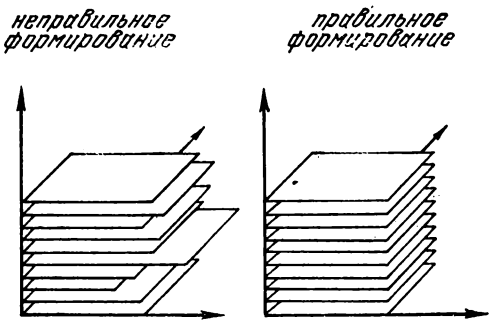


Рис. 2.1. Неправильное и правильное формирование.

Результатом операции формирования является относительный вектор с длиной, равной числу переменных в списке операции формирования. Новое измерение становится старшим по отношению к измерениям переменных из списка формирования. Операция формирования может применяться многократно, например, формирование матрицы 3-го порядка

$$\begin{array}{c} \overset{2}{\longrightarrow} \\ \overset{1}{\downarrow} \end{array} \left\| \begin{array}{ccc} x & t & k \\ y & u & l \\ z & v & m \end{array} \right\|$$

(цифрами указаны номера измерений) может быть выполнено следующим способом:

$$\| x, t, k |, | y, u, l |, | z, v, m \|$$

Заметим, что формирование вида

$$\| x, y, z |, | t, u, v |, | k, l, m \|$$

конструирует следующую матрицу

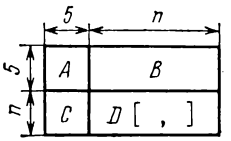
$$\begin{Vmatrix} x & y & z \\ t & u & v \\ k & l & m \end{Vmatrix}$$

2.4.2. Компоновка. Операция компоновки позволяет несколько массивов одинаковой размерности срастить по одному из измерений в более крупный массив той же размерности.

Рассмотрим пример. Пусть у нас имеются описания:

вещественный A **— матрица** 5×5;
вещественный B **— матрица** $5\times n$;
вещественный C **— матрица** $n\times5$;
массив $D\,[1{:}n,\ 1{:}n]$.

Массивы A, B, C, D нужно скомпоновать в матрицу порядка $(n+5)\times(n+5)$ по следующей схеме:

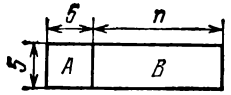


При этом, как обычно, будем считать индекс строк у матриц первым измерением, а индекс столбцов — вторым.

Для получения результирующей матрицы сращиваются сначала матрицы A и B вдоль второго измерения, при этом выписывается следующая операция компоновки:

$$|[2]\ A,\ B\,|.$$

Результатом этой компоновки будет матрица вида



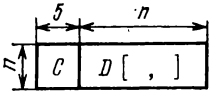
В *списке компоновки* через запятые выписываются сращиваемые массивы в требуемом порядке. Цифра 2 в квадрат-

ных скобках (*указатель измерения*) указывает, что сращивание массивов производится вдоль их второго измерения.

Аналогично сращиваются матрицы C и D

$$|[2]\,C,\ D[,]|\,,$$

давая в качестве результата матрицу



Требуемый результат получится в том случае, если срастить промежуточные матрицы $|[2]\,A,\ B|$ и $|[2]\,C,\ D[,]|$ по первому измерению:

$$|[1]\,|[2]\,A,\ B|,\ |[2]\,C,\ D[,]|\,|\,.$$

Заметим, что в данном случае результирующую матрицу можно было бы получить и другим способом:

$$|[2]\,|[1]\,A,\ C|,\ |[1]\,B,\ D[,]|\,.$$

При компоновке многомерных массивов необходимо, чтобы сращиваемые вдоль некоторого измерения массивы имели по остальным измерениям одинаковые порядки, а также были бы переменными одного и того же типа (рис. 2.2).

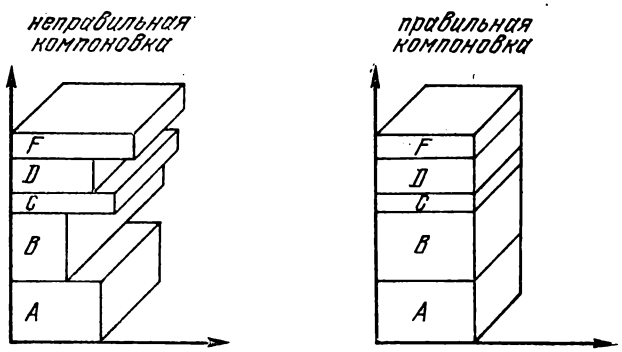


Рис. 2.2. Неправильная и правильная компоновки.

2.4.3. Составные переменные. Многомерные переменные, которые вводятся с помощью операций формирования и компоновки, называются *составными переменными*. Они могут использоваться всюду в программе так же, как и все остальные переменные.

При построении составных переменных операции формирования и компоновки могут произвольным образом комбинироваться.

В качестве примера рассмотрим конструирование окаймленной матрицы следующего вида:

$$\left\|\begin{array}{lll} C[1] \ldots. & C[n] & D \\ a[1,1] \ldots & a[1, n] & a[1, n+1] \\ \ldots\ldots\ldots & \ldots\ldots & \ldots\ldots\ldots \\ \ldots\ldots\ldots & \ldots\ldots & \ldots\ldots\ldots \\ a[n, 1] & a[n, n] & a[n, n+1] \end{array}\right\|$$

причем, для исходных перемепных имеются следующие описания:

массив $C[1{:}n]$, $a[1{:}n, 1{:}n+1]$; **вещественный** D.

Окаймленная матрица в терминах геометрических операций будет такой

$$|[1]|[2]C[], |D|, a[,]|.$$

Особенностью этой составной переменной является искусственное повышение размерности переменных. Например, запись $|D|$ превращает скаляр D в вектор длины 1.

Это необходимо для того, чтобы можно было применить операцию компоновки для сращивания векторов $C[\ ]$ и $|D|$:

$$|[1]C[], |D||.$$

Заметим, что указатель измерения «1» в этой операции дает избыточную информацию, т. к. у векторов всего одно измерение, указатель измерения необходим для того, чтобы синтаксически различить компоновку и формирование. Полученный вектор снова заключается в прямые скобки $||[1]C[], |D||$ — это для того, чтобы он мог трактоваться как матрица порядка $1\times(n+1)$.

Необходимо помнить, что при искусственном повышении размерности с помощью операции формирования, дополнительное измерение становится у результирующего массива старшим. Это значит, что если мы, например, хотим из трех скаляров x, y, z построить матрицу порядка 1×3, то ее можно изобразить в виде

$$|x, y, z|$$

или

$$||[2]|x|, |y|, |z||.$$

Если же мы хотим построить матрицу порядка 3×1, то ее можно изобразить только в виде

$$||[1]|x|, |y|, |z||.$$

Отметим, что составная переменная может быть образована только из переменных одного и того же типа (целого, вещественного, логического или комплексного).

В заключение еще раз подчеркнем разницу между операциями формирования и компоновки. На (рис. 2.3) показано

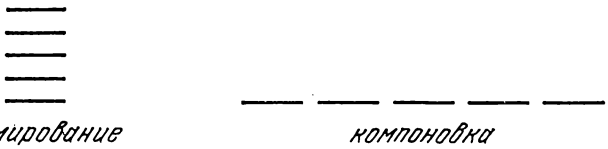


Рис. 2.3. Формирование и компоновка.

применение этих операций к одним и тем же объектам — пяти векторам одинаковой длины.

2.4.4. Геометрические операции над выражениями. В предыдущих пунктах объектами геометрических операций были только переменные, а сами геометрические операции применялись только для конструирования составных переменных. Любая переменная, как в алголе-60, так и в языке альфа-6 играет роль либо источника некоторого значения (переменная-аргумент), либо получателя значения (переменная-результат).

При использовании составной переменной в качестве аргумента роль геометрической операции сводится к тому, чтобы, взяв значения исходных переменных-компонент составной переменной, упорядочить их определенным образом (характер упорядочивания задается геометрической операцией), и полученный массив значений выдать в качестве значения составной переменной. Это употребление иллюстрируется на рис. 2.4.

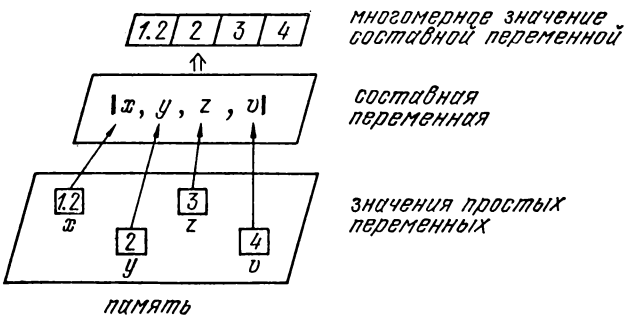


Рис. 2.4. Составная переменная-аргумент геометрической операции.

При использовании составной переменной в качестве результата роль геометрических операций состоит в разнесении компонент некоторого значения по местам хранения значений переменных-компонент составной переменной. Этот в некотором смысле обратный процесс иллюстрируется на рис. 2.5.

При такой трактовке очевидно, что компонентами геометрических операций могут быть не только переменные, но и выражения (как источник значений). В этом случае роль геометрических операций сводится к конструированию массива значений из значений выражений-компонент геометрических операций. Выражения, строящиеся с помощью геометрических операций, называются *составными выражениями*. Например, в языке альфа-6 можно записать следующие операторы присваивания:

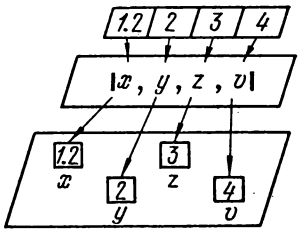


Рис. 2.5. Составная переменная-результат геометрической операции.

$$x := |\, a,\ 1,\ 0\,| \times |\, 1/2,\ a + 2,\ 3\,|;$$

$$|\, x,\ y,\ z\,| := |\, a,\ b,\ c\,|;$$

$$A\,[,] := \|\, 1,\ 2\,|,\ |\, 5,\ 3\,\| \times \|\, x,\ 0\,|,\ |\, x + 1,\ 1/x\,\|.$$

Компоненты составного выражения должны быть одного типа, либо отличаться тем, что некоторые компоненты имеют тип **целый**, другие — **вещественный**. В последнем случае, если есть хотя бы одна вещественная компонента, составное выражение будет иметь тип вещественный. При записи операторов присваивания структура выражения правой части должна совпадать со структурой переменной левой части за возможным исключением, когда правая часть оператора есть нуль (*0*, *0.0* или *.0*).

2.5. Верхний индекс

В соответствии с той геометрической интерпретацией, которая была дана многомерным переменным, можно сказать, что переменная с одним индексом задает некоторую пространственную последовательность. Это имеет тот смысл, что компоненты вектора рассматриваются как совокупность чисел, упорядоченная вдоль некоторой координатной оси. Особенностью пространственной последовательности является равнодоступность всех ее компонент. В самом деле, при входе в

блок, в котором описан некоторый вектор, считается, что любой компоненте вектора, соответствующей значению индекса в пределах граничной пары, может быть присвоено некоторое значение и любая компонента вектора, которой уже было присвоено значение, может быть использована. Трактовка переменных с индексом как пространственных последовательностей в ряде задач имеет смысл не только из-за интерпретации, принятой в языке альфа-6, но и по существу, когда переменная с индексом обозначает значения некоторой физической величины в узлах пространственной сетки.

В ряде математических задач используются также и *временные* последовательности величин. Это значит, прежде всего, что в таких задачах имеется некоторая система отсчета, связанная со временем, или с какой-либо иной величиной, характеризующейся монотонным и равномерным изменением (например, номер итерации, последовательность этапов решения задачи и т. п.). Значения некоторой переменной образуют временную последовательность, если каждому моменту времени соответствует одно значение этой переменной. Нас будут интересовать такие временные последовательности, когда очередное значение переменной вычисляется по некоторой явной формуле, содержащей некоторое число непосредственно предшествующих членов рассматриваемой временной последовательности. Такие последовательности будут называться *рекуррентными*. Формулы, которые связывают очередной член последовательности с предыдущим, называются *рекуррентными соотношениями*. Количество предшествующих членов последовательности, необходимое для вычисления очередного члена, называется *длиной рекурсии*. Естественно, что для того, чтобы точно указать, как зависит очередной член последовательности от предыдущих, члены временной последовательности нумеруются. Нумерация определяется заданием *начального номера* последовательности. Наконец, для любой рекуррентной последовательности необходимо задавать *начальные значения* последовательности для того, чтобы можно было применить рекуррентное соотношение. Очевидно, что количество начальных значений должно быть равно длине рекурсии. Обычно при записи рекуррентных соотношений текущий или начальный номер последовательности изображают в виде верхнего индекса, берущегося в круглые или квадратные скобки (этим, кстати, и объясняется название данного параграфа).

Примеры. 1) Вычисление чисел Фибоначчи $f^{(1)}=0$, $f^{(2)}=1$; $f^{(i+2)}=f^{(i+1)}+f^{(i)}$ $(i=1, \ldots, n)$. Здесь длина рекурсии равна двум, начальный номер равен единице.

2) Решение уравнения $X = A \times X + B$ методом простой итерации $X^{[0]} = 0$, $X^{[i+1]} = A \times X^{[i]} + B$ $(i = 0, 1, \ldots)$. Длина рекурсии $= 1$, начальный номер $= 0$.

3) Явная разностная схема решения одномерного уравнения теплопроводности $u_t = u_{xx}$; $u_n^{(j+1)} = u_n^{(j)} + \sigma\left(u_{n+1}^{(j)} - 2u_n^{(j)} + u_{n-1}^{(j)}\right)$. Рекурсия происходит по j, длина рекурсии $= 1$, начальный номер $= 0$.

В языке альфа-6 имеются средства, позволяющие использовать в программах переменные с верхним индексом для вычисления рекуррентных последовательностей. Для того, чтобы ввести верхний индекс у некоторой переменной x, для нее нужно указать начальный номер последовательности, длину рекурсии и метку некоторого оператора цикла (*управляющего цикла*), в теле которого могут помещаться рекуррентные соотношения и при повторении которого производится отсчет времени. Эта информация указывается специальным *описанием верхнего индекса*, имеющим вид

$x\uparrow[N{:}L{:}M]$,

где N — начальный номер, L — длина рекурсии, M — метка управляющего цикла. Описание верхнего индекса ставится в описание типа данной переменной после идентификатора переменной.

Пример.

вещественный $\Phi\uparrow[K{:}P{:}M]$;
массив A, $B\uparrow[0{:}K{:}M]$ $[1{:}10,\ 1{:}10]$;
целый $X\uparrow[0{:}1$: итерация$]$ — **вектор** 5.

Начальный номер и длина рекурсии могут быть любыми выражениями, подчиняющимися тем же правилам, что и границы индексов в описании массивов (т. е. эти выражения не должны содержать переменных, описанных в том же блоке, в начале которого стоит данное описание).

Метка должна помечать оператор цикла, который стоит в том же блоке, в котором находится описание данной переменной с верхним индексом.

Правила записи верхнего индекса в программе очевидны из следующих примеров (слева содержательная символика, справа — символика языка альфа-6):

$B^{[0]}$	$B\uparrow[0]$,
$X_{ij}^{(l+2)}$	$X\uparrow[l+2][i, j]$.

При использовании верхних индексов важно знать некоторые правила размещения в памяти ЭВМ рекуррентной после-

довательности при выполнении программы. Типичное использование величины с верхним индексом состоит, прежде всего, в задании начальных членов последовательности, многократного применения рекуррентных соотношений для получения нужного члена и, наконец, использовании последнего или нескольких последних членов. Поскольку для очередного члена последовательности нужно не более L (где L — длина рекурсии) предыдущих членов, вычисление последовательности можно представить как «протаскивание» ее членов через $(L+1)$ *циркуляционных ячеек*.

$$
\begin{array}{ll}
t=1, & X^{(1)}X^{(2)}X^{(3)}X^{(4)}X^{(5)} \\
t=2, & X^{(2)}X^{(3)}X^{(4)}X^{(5)}X^{(6)} \\
\ldots\ldots\ldots & \qquad L=4 \\
t=5, & X^{(5)}X^{(6)}X^{(7)}X^{(8)}X^{(9)}
\end{array}
$$

В начальный момент времени ($t=0$) до начала выполнения управляющего цикла в циркуляционных ячейках (более точно, в L первых) размещаются начальные члены последовательности.

Очередное значение, вычисленное по L предыдущим, становится в $(L+1)$-ю циркуляционную ячейку. После очередного выполнения тела управляющего цикла время t увеличивается на единицу, члены последовательности сдвигаются вдоль циркуляционных ячеек на один шаг, освобождая $(L+1)$-ю ячейку для следующего члена. При такой трактовке для того, чтобы узнать, в какую циркуляционную ячейку перед выполнением или в процессе выполнения управляющего цикла попадает член последовательности со значением верхнего индекса, равным E, нужно просто вычесть t из E.

По окончании выполнения управляющего цикла в циркуляционных ячейках остаются последние L членов последовательности (самый последний хранится как в L-й, так и в $(L+1)$-й ячейках). Если в этот момент нужно использовать только последний член последовательности, достаточно упомянуть идентификатор переменной с пустой позицией верхнего индекса. Если же желательно использовать и некоторые предыдущие значения из L последних, то их нужно упоминать по общему правилу $(E-t)$, считая, что время t по выходе из управляющего цикла становится равным нулю.

Более точно манипулирование с переменной с верхним индексом может быть изображено следующими алгольными операторами (на примере вычисления чисел Фибоначчи $fk = fn2 = f^{(n+2)}$ и $fn1 = f^{(n+1)}$).

Язык альфа-6 с верхним индексом:	Язык алгол-60 с циркуляционными ячейками:
начало	**начало**
вещественный $f\uparrow[1:2:M]$;	**массив** $f[1:1+2]$;
	целый t, $k1$; $t:=0$;
$f\uparrow[1]:=0$;	$f[1-t]:=0$;
$f\uparrow[2]:=1$;	$f[2-t]:=1$;
M: **для** $k:=1$, ..., n **цикл**	M: **для** $k:=1$ **шаг** 1 **до** n **цикл**
	начало
$f\uparrow[k+2]:=f\uparrow[k+1]+f\uparrow[k]$;	$f[k+2-t]:=f[k+1-t]+f[k-t]$;
	$t:=t+1$;
	для $k1:=1$ **шаг** 1 **до** 2 **цикл** $f[k1]:=f[k1+1]$
	конец; $t:=0$;
$fn2:=f\uparrow[2]$;	$fn2:=f[2-t]$;
$fn1:=f\uparrow[1]$;	$fn1:=f[1-t]$;
$fk:=f[\,]$;	$fk:=f[3-t]$;
конец	**конец**

2.6. Цепочки неравенств

В языке альфа-6 допускается запись отношений равенства и неравенства между арифметическими выражениями в виде цепочек отношений:

$$a<x<b$$

$$A[\,]=B=T[\,]=0$$

Знаки отношений в цепочке должны быть одной направленности, т. е. либо из группы $<$, $\leqslant$, $=$, либо из группы $>$, $\geqslant$, $=$, поэтому запись вида $a<b>d$ недопустима. Со знаком $\neq$ цепочка не образуется.

По способу вычисления выражение вида

$$a<x\leqslant b=d$$

эквивалентно выражению

$$(a<x)\wedge(x\leqslant b)\wedge(b=d).$$

2.7. Операторы цикла без параметра

В языке альфа-6 допустимы операторы цикла, имеющие вид

E раз цикл $\{S\}$ (I)

пока L **цикл** $\{S\}$ (II)

где E — любое первичное скалярное выражение, L — логическое скалярное выражение и $\{S\}$ — тело цикла. Фигурные скобки в языке альфа-6 являются синонимами скобок **начало и конец.**

Пусть n — текущее значение выражения E, вычисляемое по правилам вычисления индексных выражений, которое E имеет непосредственно перед выполнением цикла типа (I). Если $n \leqslant 0$, то выполнение оператора цикла не происходит и сразу выполняется преемник этого оператора. Если $n > 0$, то тело цикла выполняется n раз подряд, если только выход по метке из тела цикла не оборвет его выполнения раньше.

Выполнение оператора типа (II) вполне эквивалентно по выполнению следующей алгольной программе:

M: **если** L **то** $\{S$; **на** $M\}$.

Выполнение этих циклов, как видно, сводится только к повторному выполнению цикла и не сопровождается пересчетом какого бы то ни было параметра. Надо подчеркнуть, что если в цикле типа (I) E вычисляется только один раз перед началом работы цикла, то в (II) L вычисляется при каждом новом повторении тела цикла S, и естественно, что при выполнении S должны перевычисляться какие-то аргументы выражения L.

Примеры.

200 **раз цикл** $X := A \times X + B$
пока $abs(x-y) > _{10}-3$ **цикл**
начало $y := x$;
$x := k \times (x \times \sin(x)) + b \times exp(x/2)$;
конец

2.8. Оператор останова

В алголе-60 единственным нормальным условием окончания выполнения программы является выход в соответствии с правилами перехода от оператора к оператору за самую внешнюю закрывающую скобку программы. Во входном языке альфа-6 можно прекратить выполнение программы в любой ее точке, поставив в надлежащем месте оператор останова, имеющий вид **стоп.**

Пример.

если $\det(A[,]) < _{10} - 8$ **то стоп иначе** на продолжение счета.

2.9. Описание начального значения

Описание начального значения является средством задания значений переменным перед их использованием в блоке. Описание начального значения целесообразно использовать для

собственных переменных, так как их значения сохраняются при повторном входе в блок и возникает необходимость задавать значение при первом. Описание начального значения имеет вид:

$$x = C,$$

где C — выражение, текущее значение которого берется в качестве начального значения переменной x. В вычисляющее выражение C и в индексные выражения переменной x (если они есть) не могут входить переменные, локализованные в том же блоке, в который входит данное описание начального значения. Левой частью описания начального значения может быть простая переменная, переменная с индексами, как с непустыми, так и с пустыми.

Примеры.

nu := *3.1415926536*;
$a[] = |1/6, 2, 7|$;
$x[i] = E - \delta \times i$.

Для несобственной переменной описание начального значения эквивалентно помещению оператора присваивания в начале блока.

Из вышесказанного, в частности, следует, что описание начального значения подчиняется тем же требованиям согласования типов и структур, как и для операторов присваивания. Особенностью описания начального значения собственной переменной является то, что оно действует при выполнении программы только один раз — при первом входе в блок, в котором находится данное описание.

2.10. Функции-выражения

Рассмотрим случай, когда тело процедуры-функции состоит из одного оператора присваивания, например,

целая процедура $F(n, m)$ **значение** n, m; **целый** n, m;
$F := a[n,] \times a[, m]$.

Язык альфа-6 позволяет записать такие процедуры более просто в виде *описания функции-выражения*:

целая функция $F(n, m) = a[n,] \times a[,m]$

В общем случае описание функции-выражения имеет вид

⟨тип⟩ **функция** $f(p_1, \ldots, p_n) = E$

или

⟨тип⟩ **функция** $f = E$,

где ⟨тип⟩ означает указатель типа функции: **целый**, **вещественный**, **комплексный** или **логический**. Указатель типа **веществен-**

ный в описании функции можно опускать $p_1, \ldots, p_n$ — формальные параметры, не требующие спецификации, т. е. как бы вызываемые именем. Идентификаторы $p_1, \ldots, p_n$ локализуются в пределах данного описания и поэтому вне описания могут использоваться для обозначения других величин. E — любое выражение; в нем могут содержаться как формальные параметры $p_1, \ldots, p_n$, так и другие переменные.

Следует отметить, что описания функций-выражений не расширяют алгоритмических возможностей алгола, но служат только для того, чтобы приблизить запись функции к обычной математической символике.

Значение функции-выражения при всех обращениях к ней должно иметь одну и ту же размерность.

Если значение функции-выражения является скалярным, то соответствующий указатель функции (обращение к функции) может употребляться так же как в алголе-60. Если же значение является вектором, матрицей или вообще многомерным массивом, то в этом случае указатель функции не может употребляться в качестве составной части более сложного выражения (т. е. указатель функции может быть только правой частью оператора присваивания). Структура функции при этом должна совпадать со структурой переменных левых частей соответствующих операторов присваивания.

Например, **функция** $f(x, y)=x+y$; **вещественный** *x1*, *y1*, *z* — **вектор** *5*. Тогда можно написать

$z:=f(x1, y1)$; $z:=-z\times 2$.

Однако, нельзя написать

$z:=f(x1, y1)\times 2$.

2.11. Спецификация-результат

В языке альфа-6 в дополнение к имеющемуся в алголе-60 разделению формальных параметров на параметры, вызываемые по значению, и на параметры, вызываемые по имени (по написанию), добавлены *параметры-результаты*. Параметры-результаты специфицируются в заголовке процедуры в виде списка результатов вида

результат *x*, *y*, *z*,

где *x*, *y*, *z* — идентификаторы специфицируемых формальных параметров.

Список результатов помещается в спецификации вслед за списком значений и перед остальными спецификациями.

Формальный параметр-результат рассматривается как величина, локализованная в теле процедуры. В случае нормального окончания выполнения тела процедуры, инициированного выполнением некоторого оператора этой процедуры, текущее значение формального параметра-результата присваивается соответствующему фактическому параметру. (Под нормальным окончанием подразумевается выход из тела процедуры «через» закрывающую скобку тела процедуры, в противоположность возможному выходу из тела по нелокализованной метке.)

Параметр-результат — это понятие, аналогичное параметру, вызываемому значением. Связь и тех и других с фактическими параметрами осуществляется с помощью операторов присваивания значения, которые (для вызова по значению) снабжают процедуру значениями аргументов и (для параметров-результатов) снабжают фактические параметры значениями результатов выполнения процедуры.

Пример.

На языке альфа-6:

```
начало вещественный a;
процедура p(x);
    результат x;
    вещественный x;
    начало целый k;
    x:=0;
    для k:=1 шаг 1 до 3 цикл
    x:=(k+x)×a;
    конец;
a:=-1; p(a)
конец
```

На языке алгол-60:

```
начало вещественный a;
процедура p(x);
    вещественный x;
    начало целый k;
    вещественный x1;
    x1:=0;
    для k:=1 шаг 1
    до 3 цикл
    x1:=(k+x1)×a;
    x:=x1;
    конец;
a:=-1; p(a)
конец
```

После обращения к процедуре p значение a равно -2. Заметим, что при подстановке x именем (т. е. отсутствует спецификация **результат** x) a равнялось бы нулю.

2.12. Перечисление

Одним из распространенных в математике обозначений являются перечисления занумерованных последовательностей с употреблением многоточия. В качестве примеров можно привести следующие общепонятные обозначения:

$1, \ldots, n$

$X_1 = \ldots = X_{20} = 0$

$B_1 < \ldots < B_t$

$A_{ij}, \ldots, A_{ij+m}$

и т. п. Все они имеют общую структуру вида

$$E_1, \ldots, E_2,$$

или

$$A[E_1]\omega \ldots \omega A[E_2].$$

Последнее перечисление символизирует список выражений $A[i]$, где i — обозначение одной или нескольких позиций индекса, разделенных некоторым ограничителем ω, причем индекс i во всех выделенных позициях пробегает последовательные возрастающие значения, начиная со значения E_1 и кончая значением E_2.

Конструкция вида $E_1, \ldots, E_2$ сокращенно указывает на перечисление значений, начиная с E_1 и кончая E_2. Длиной перечисления называется текущее значение выражения (E_2-E_1+1), оно равно числу выражений $A[i]$ в списке. В языке альфа-6 допустимы перечисления разнообразных типов в зависимости от того, с каким ограничителем ω они используются.

2.12.1. Перечисление в заголовке оператора цикла. В языке альфа-6 допускается элемент списка цикла вида

$$E_1, \ldots, E_2$$

где E_1 и E_2 — арифметические выражения. Этот элемент эквивалентен элементу списка цикла вида

E_1 **шаг** *1* **до** E_2.

Примеры.

для $i:=1, \ldots, n$ **цикл**
для $i:=1, \ldots, j-1, j+1, \ldots, k$ **цикл**
для $i:=2, 4, 6, \ldots, 2\times n$ **цикл**

В последнем примере последовательность значений параметра не представляет собой последовательности четных чисел. Все перечисления, употребляемые в языке альфа-6, предполагают изменение индекса перечисления с шагом единица.

2.12.2. Перечисления в списках геометрических операций. Пусть $X[i]$ — условное обозначение некоторой переменной (может быть, и составной), в которой выделены одна или несколько индексных позиций, обозначенных здесь буквой i. Тогда перечисление вида

$$X[E_1], \ldots, X[E_2],$$

где E_1 и E_2 — арифметические индексные выражения, обозначают список переменных, состоящий из (E_2-E_1+1) элементов и получающийся при подстановке вместо i значений E_1, E_1+1 и т. д. до E_2 включительно.

Примеры:

1) $|\,B[10], \ldots, B[10 + 2 \times v]\,|$

обозначает вектор длины $(2 \times v + 1)$, образованный компонентами $B[10]$, $B[11]$ и так далее до $B[10 + 2 \times v]$

2) $|\,[1]\,A[, 2,], \ldots, A[, 7,]\,|$

обозначает матрицу вида

$$\left\|\begin{array}{l} A[, 2,] \\ A[, 3,] \\ \cdot\ \cdot\ \cdot \\ A[, 7,] \end{array}\right\|$$

3) Запись

$$\|\,a[1, 1], \ldots, a[1, n]\,|, \ldots, |\,a[n, 1], \ldots, a[n, n]\,\|$$

обозначает матрицу вида

$$\left\|\begin{array}{l} a_{1,1} \cdot\ \cdot\ \cdot\ a_{1,n} \\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ a_{n,1} \cdot\ \cdot\ \cdot\ a_{n,n} \end{array}\right\|$$

Имеет смысл проследить, как строилась эта составная переменная. Сначала брался общий элемент двумерного массива $a[i, j]$. Перечислением по j получалось обозначение i-й строки матрицы

$$|\,a[i, 1], \cdots, a[i, n]\,|\,.$$

Затем в этой составной переменной выделялись две индексные позиции, в которых стоит i, и перечислением по этому индексу от 1 до n получалась результирующая матрица.

4) Запись

$$mod\,(\,|\,a[1, 1], \ldots, a[n, n]\,|\,)$$

обозначает вычисление нормы вектора, образованного диагональными элементами матрицы.

2.12.3. Перечисления в цепочках неравенств. Язык альфа-6 позволяет записывать цепочки отношений в виде

$$B[1] < \ldots < B[10];$$

$$a[1, 1] \leqslant \ldots \leqslant a[n, n];$$

(заметим, что последнее перечисление содержит только диагональные элементы матрицы);

$$k > L[1] \geqslant \ldots \geqslant L[10] > P.$$

Смысл этих конструкций не требует особых пояснений.

2.13. Составные метки

В языке альфа-6 существует средство, позволяющее осуществлять переход на внутренний оператор некоторого блока извне этого блока. Таким средством являются составные метки. Синтаксически составная метка состоит из последовательности обычных меток, разделенных символом «точка». Последней меткой, входящей в составную метку, является метка того внутреннего оператора *S*, на который осуществляется переход. Остальные метки являются метками блоков, охватывающих оператор *S*. Правила локализации, которые должны выполняться для составной метки, состоят в следующем:

а) первая метка должна попадать под обычное правило локализации меток;

б) если метка *N* непосредственно следует в составной метке за меткой *M*, то метка *N* должна быть локализована в блоке, помеченном меткой *M*.

Пример.

```
начало целый x;
  M1: начало целый x; x = 2;
      N1: начало целый y; y = 1; ... P1: на M2.P2; ...
          конец блока N1;
      конец блока M1;
      M2: начало целый x; ... P2: ...;
          на M1.N1.P1; ...
          конец блока M2;
конец
```

При переходе по составной метке во всех упомянутых блоках выполняются все действия, которые совершаются при входе в блок, т. е. отводится место для динамических массивов и задаются начальные значения (если они есть). Так, в приведенном примере при переходе по метке *M1. N1. P1* производится начальное присваивание переменным *x* и *y*.

2.14. Указатели каналов ввода/вывода

Под каналами в языке альфа-6 понимаются специальным образом организованные участки внешней памяти, в которые записывается информация, выданная операторами *output* и *marg* для последующей распечатки на АЦПУ, либо помещаются данные, предназначенные для ввода операторами *input* при выполнении программы. Каждая задача может использовать несколько каналов.

Канал 0 интерпретируется как участок барабанной памяти, остальные каналы 1, 2, ..., 14 — как ленточные (или дисковые), при использовании которых в паспорте системного пакета необходимо заказывать соответствующие бобины (приложение 9). Математический номер ленты (диска) определяется по номеру канала i и равен $50+i$ (в восьмеричной системе счисления).

Каналы имеют указатели. Информация в каналах хранится порциями переменной длины (*записями*) и подразделяется на информацию, используемую операторами *input*, и информацию, записанную операторами *output* и *marg*. Каждый канал имеет 3 указателя: точку текущего чтения IN, точку текущей записи OUT и точку конца чтения, которая также совпадает с точкой начала записей НО. Механика заполнения канала и считывания из него следующая. Перед выполнением программы в каналы помещаются оттранслированные данные, при этом указатели НО и OUT устанавливаются на конец записей данных, а указатель IN на начало канала. Затем, во время выполнения программы, при каждом выполнении операторов *input*, *output* и *marg*, указатели IN и OUT увеличивают свои значения, указывая на текущие точки чтения и записи соответственно. Точка НО остается на месте. Если значение указателя IN становится больше значения НО, то выполнение оператора *input* прекращается, выдается сообщение об ошибке и управление передается сбойной программе.

Распечатка на АЦПУ информации, выданной операторами *output* и *marg* (от указателей НО до OUT), осуществляется после окончания задачи по каждому каналу отдельно. При переполнении одного из каналов во время выполнения программы осуществляется *освобождение* канала (т. е. выдача на АЦПУ с последующей установкой указателя OUT равным значению НО). С помощью набора стандартных процедур (15.4.15) можно изменять значения указателей каналов или запоминать их. Имеется также возможность использовать содержимое ленточных каналов, полученное при выполнении альфа-программ или при трансляции альфа-данных, для ввода или распечатки в других альфа-программах, в том числе выполняемых в разных задачах.

Пример управления каналами ввода/вывода.

Требуется стократно повторить ввод одной и той же группы чисел во время выполнения программы. Кроме того, необходимо записать результаты выполнения программы на ленту, с тем чтобы использовать их многократно для ввода в другой программе, выполняемой независимо от первой. Тогда

первую программу схематически можно представить следующим образом.

```
◇ ◇ ◇  ПРОГРАММА
   ПРОГ1:
      начало массив C [0:2047];
      . . . . . .
            вещественный a, b, k, t; целый i;
      . . . . . .
      для i: = 1, ..., 100 цикл начало
      . . . . . .
            input (0, a); ...
            input (0, b, k); ... input (0, t);
      . . . . . .
          изм (0, 1, 0, 0) конец цикла по i;
      output (1, 'e', C);
      конец
◇ ◇ ◇ ЗАДАНИЕ
◇ ◇ ◇ ДАННЫЕ
   ПРОГ1:
      канал 0*
      a=3.14; b=2.8; t=5.12;
◇ ◇ ◇ КНЦ
```

В программе *ПРОГ1* все данные из фрагмента «данные» (после карты ◇ ◇ ◇ ДАННЫЕ) вводятся операторами *input* в теле цикла по *i*. В конце тела цикла оператор *изм*(*0*, *1*, *0*, *0*) устанавливает указатель IN на начало канала, так что при повторном выполнении тела цикла для ввода будут использоваться те же значения, что и при первом.

Выведенный на ленту с математическим номером и направлением 51 (канал 1) массив *C* будет распечатан на АЦПУ после окончания задачи. Кроме того, этот массив можно ввести оператором *input* как, например, это делается ниже.

```
◇ ◇ ◇ ПРОГРАММА
   ПРОГ2:
      начало массив C [0:2047];
      coxp (1); input (1, C);
      . . . . . .
      конец
```

2.15. Оператор ввода

Оператор ввода позволяет присваивать (вводить) переменным программы некоторые значения, которые готовятся отдельно от программы. Последовательность этих значений (дан-

ные) состоит из чисел, логических значений, строк (в смысле алгола-60), а также двоичных кодов (БЭСМ-шкал), команд, констант и чисел БЭСМ-6 (см. гл. 16). В данных выделяются группы значений (или группы данных), которые отделяются друг от друга символом «;». Группа данных может содержать как одно значение, так и несколько однотипных значений (т. е. либо числа, либо логические значения и т. д.). Последние предназначаются для ввода в массивы. Значения в одной группе данных разделяются запятыми.

Оператор ввода имеет вид:

input (*k*, *S1*, *S2*, ..., *Sn*)

где *k* — (выражение типа целый) номер канала, по которому осуществляется выборка данных (2.14),

Si — объект ввода (*i* = 1, ..., *n*): идентификатор массива, скалярная переменная, переменная с индексами.

Если объект ввода — массив типа целый, вещественный или логический, то ему соответствует группа данных, состоящая из нескольких значений (числовых или логических, в зависимости от типа массива), причем количество этих значений не должно превышать числа элементов массива, задаваемого описанием массива. Если их меньше, то ввод осуществляется в последовательные элементы, начиная с первого. Оставшиеся незаполненными элементы массива будут иметь те значения, которые они имели до ввода.

При вводе комплексного массива каждому элементу массива соответствует пара идущих подряд чисел из одной группы данных, первое число этой пары для действительной части, второе для мнимой.

Скалярной переменной типа целый, вещественный или логический должна соответствовать группа данных, состоящая из одного числового или логического значения. Комплексной скалярной переменной соответствует группа данных из двух числовых значений, первое для действительной, второе — для мнимой частей комплексной переменной. Объекту «переменная с индексами типа целый» может соответствовать строка в данных. В этом случае при вводе последовательным элементам массива, начиная с номера, указанного индексом, присваиваются целые значения, соответствующие последовательным символам строки. Кодировка строк дается в приложении 7. Некоторые символы строки могут представляться несколькими кодами, поэтому программисту следует быть осторожным в указании начального индекса при вводе текста.

Пример.

Пусть *a*, *b*, *c* — скалярные переменные типа вещественный, *ЛОГ* — логический вектор длины 3, *M* — вещественный вектор длины 5, *K* — целый вектор длины 3.

Данные для ввода имеют вид:

канал *0***1*; *2*; *3.14*; **истина**, **ложь**, **ложь**;
0.1, *0.2*, *0.3*; '*a*';

Тогда в результате работы операторов

input(*0*, *a*, *b*); *input*(*0*, *c*, *ЛОГ*, *M*, *K*[*1*]);

переменные примут следующие значения: *a*=*1*, *b*=*2*, *c*=*3.14*,

ЛОГ [*1*] = **истина**, *ЛОГ* [*2*] = **ложь**, *ЛОГ* [*3*] = **ложь**,
M [*1*] = *0.1*, *M* [*2*] = *0.2*, *M* [*3*] = *0.3*,
M [*4*] и *M* [*5*] останутся не заданными,
K[*1*] = *26* — код открывающей кавычки,
K[*2*] = *32* — код буквы *A*,
K[*3*] = *27* — код закрывающей кавычки.

Указанный в примере порядок ввода данных может быть изменен при помощи процедур управления указателями канала. Более того, можно задать также ввод значений выведенных ранее с помощью операторов *output*, как это делается в примере п. 2.14.

2.16. Операторы разметки

Вся информация, выводимая с помощью операторов *output* размещается на АЦПУ в постраничном виде. *Страницей* называется прямоугольная область листа АЦПУ (см. рис. 2.6),

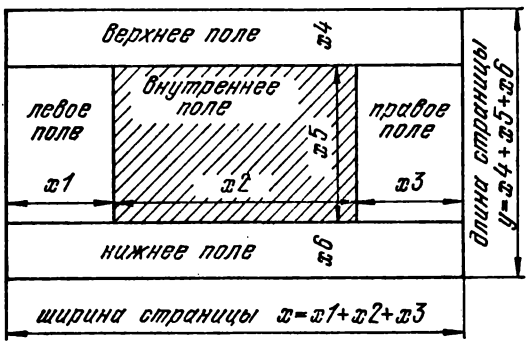


Рис. 2.6. Разметка страницы.

состоящая из поля, занятого выводимой информацией (заштрихована на рисунке) и свободных полей.

Размеры и расположение этих полей характеризуются следующими шестью величинами, которые называются *разметкой страницы*:

— размер левого поля (т. е. число позиций),
— длина печатаемой строки,
— размер правого поля,
— размер верхнего поля,
— число строк внутреннего поля,
— размер нижнего поля.

При выводе на АЦПУ страницы нумеруются, а информация на них распечатывается построчно, с переносом на следующую строку.

Стандартная разметка задает следующий вид страницы:

— размер левого поля = 10 позиций,
— длина печатаемой строки = 108 позиций,
— размер правого поля = 10 позиций,
— размер верхнего поля = 4 строки,
— число строк внутреннего поля = 60,
— размер нижнего поля = 3 строки.

В языке альфа-6 имеется три типа стандартных операторов, которые связаны с разметкой страницы:

установка новой разметки (оператор *marg*),
считывание текущей разметки (оператор *lmarg*),
установка шаблона страниц (операторы *откшаб* и *закшаб*).

2.16.1. Оператор *marg*.

marg (*k*, *X1*, *X2*, *X3*, *X4*, *X5*, *X6*, *N*),

где *k* — (выражение типа целый) номер канала вывода,

X1, ..., *X6* — выражения типа целый, задающие разметку страницы, т. е. размеры полей в том порядке, как они выписаны выше,

N — выражение типа целый, задающее начальный номер страницы при нумерации Если этот аргумент отсутствует, то начальный номер страницы равен 1.

Расположение страниц на листе АЦПУ зависит как от размеров страницы, т. е. ширины x и длины y страницы, так и от размеров листа АЦПУ.

Лист АЦПУ — это двумерное поле со стандартной шириной в 128 позиций, неограниченное по длине. Если $128/x<2$ (где x — ширина страницы), то страницы располагаются последовательно одна за другой (т. е. сверху вниз). Если же $128/x\geqslant 2$, то при одной и той же разметке для всех выводимых страниц

они будут располагаться слева направо сверху вниз (см. рис. 2.7).

Например, операторы

marg (*0*, *0*, *18*, *1*, *0*, *10*, *8*); *output* (*0*, '*e*', *A*);

выведут значения массива *A* столбцами по 10 чисел в каждом и по 6 столбцов на одном развороте листа АЦПУ (рис. 2.7).

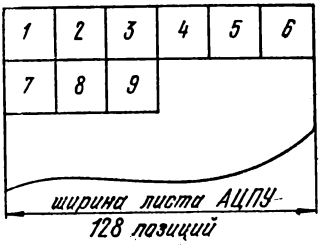


Рис. 2.7. Расположение одинаковых страниц OUTPUT на листе АЦПУ.

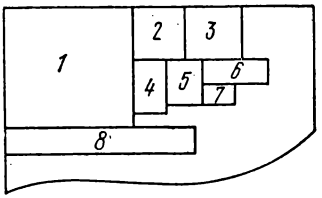


Рис. 2.8. Расположение страниц OUTPUT на листе АЦПУ.

В случае, когда размеры страниц неодинаковые, то дополнительно к правилу «слева направо — сверху вниз», необходимо также учитывать правило упаковки страниц, которое состоит в следующем (см. рис. 2.8):

а) длина любой страницы, размещенной справа, меньше, либо равна длине страницы, размещенной слева,

б) страница располагается так, что ее верхняя и левая границы примыкают к левому краю листа АЦПУ или к предыдущим страницам,

в) если перенумеровать все выводимые страницы, то страница с бóльшим номером не может располагаться выше страницы с меньшим номером.

С помощью процедуры *режк* (п. 15.4.15) можно уменьшить ширину листа АЦПУ, доступную для размещения страниц.

2.16.2. Оператор считывания текущей разметки.

lmarg (*k*, *X1*, *X2*, *X3*, *X4*, *X5*, *X6*),

где *k* — выражение типа целый, задающее номер канала,

X1, ..., *X6* — переменные типа целый, которым присваиваются значения разметки в том порядке, как они выписаны в начале п. 2.16.

2.16.3. Шаблоны страниц. В стандартном режиме на странице распечатывается строка, состоящая из символов, «подчеркивание», и заполняющая строку верхнего поля, на второй строке верхнего поля печатается номер страницы. Если размер

верхнего поля меньше либо равен 2, то номер страницы не печатается, а если этот размер равен нулю или единице, то не печатается строка «подчеркиваний».

Эту распечатку на странице (подчеркивание и номер страницы) можно интерпретировать как шаблон, который «отпечатывается» в том месте листа АЦПУ, где должна быть расположена новая страница, перед тем как на ней будет размещена информация. В языке альфа-6 имеется возможность задавать нестандартные шаблоны (например, сетку для размещения таблицы на странице). Шаблон готовится на отдельном поле (поле шаблона). Размеры поля шаблона равны размерам текущей страницы, т. е. x и y (см. рис. 2.6). Формирование поля шаблона осуществляется следующей последовательностью операторов:

откшаб (k); последовательность операторов языка, содержащая операторы *output*; *закшаб* (k);

где k — выражение типа целый, задающее номер канала, для которого задается шаблон. Операторы *output* должны иметь тот же номер канала.

При помощи оператора *откшаб* (k) происходит заполнение поля шаблона пробелами и настройка процедуры *output* на заполнение поля шаблона. При этом, если до выполнения оператора *откшаб* (k) была установлена разметка ($x1$, ..., $x6$), то после его выполнения устанавливается разметка (0, $x1+x2+x3$, 0, 0, $x4+x5+x6$, 0), т. е. для операторов *output* делается доступным все поле шаблона. Затем при помощи операторов *output* осуществляется необходимое заполнение шаблона, после чего оператор *закшаб* (k) завершает формирование шаблона и восстанавливает разметку ($x1$, ..., $x6$). В последовательности операторов, расположенных в программе между операторами *откшаб* (k) и *закшаб* (k), могут встречаться любые операторы языка альфа-6, кроме оператора *marg* и оператора *output*, задающего переход на новую страницу.

Отмена шаблона осуществляется оператором *marg*, после выполнения которого устанавливается стандартный шаблон.

Пример. Задать вывод табл. П.1. Следующая программа сначала печатает заголовок таблицы, затем устанавливает разметку страницы для размещения колонок с кодами символов и их образов, задает соответствующий шаблон и осуществляет распечатку символов и их кодов.

начало целый массив $a[1:4]$; **целый** k;
задание строки для начального символа:
$a[1]=26$; $a[2]=15$; $a[3]=0$; $a[4]=27$;

```
печать заголовка:
  marg (0, 0, 60, 0, 0, 2, 0);
   output (0, 't' , ' приложение 2. кодировка основ-
   ных символов на УПП (БЭСМ-6):/', '14в', 't',
   '(ГОСТ (10859-64), АЦПУ-128-2):/');
установка разметки:
    marg (0, 0, 12, 0, 4, 40, 2);
задание шаблона:
    откшаб (0);
    output (0, 't', '—', 12, 't', '| восьм | симв ||_ код —|
   — — — — |', 't', '—', 12);
    для k:=1,..., 20 цикл
    output (0, 't', '|— — — — — — | — — — — |', 't', '—', 12);
   закшаб (0);
выдача таблицы: для k: = 0, ..., 90 цикл начало a[3]: = k;
если k = 26 то начало a [2]: = 37; a [3]: = 1 конец;
если k = 27 то a[3] = 2;
если k=28 то a[2]:=15;
     output (0, 'ob3ab', k, '2b', 't', a [1], '/') конец;
конец
```

2.17. Оператор вывода

Распечатка результатов счета в постраничном виде осуществляется с помощью операторов *output*. В п. 2.16 описано размещение страниц на листе АЦПУ и способы задания разметки страниц. В этом параграфе мы укажем, как проводится размещение информации на странице.

Оператор вывода имеет следующий вид.

output (*k*, *E1*, ..., *Ep*),

где *k* — выражение типа целый, задающее номер канала, а *Ei* — (*i*=1, ..., *p*) — *элемент вывода* любого из следующих шести видов:

1) элемент числового вывода, который служит для вывода целых, вещественных или комплексных значений;

2) элемент логического вывода для вывода логических значений;

3) элемент восьмеричного вывода для вывода значений во внутреннем (восьмеричном) представлении в машине;

4) элемент текстового вывода для вывода строк или текстов, закодированных целыми числами;

5) элемент размещения для указания позиции на странице, начиная с которой размещается информация, выдаваемая следующими элементами вывода операторов *output*;

6) элемент вывода номера текущей страницы.

Элементы вывода вида 1), 2) или 3) состоят из следующей последовательности аргументов:

$$Ф, X1, \ldots, Xm,$$

где $Ф$ — *формат* вывода: строка (в смысле алгола-60), формальный параметр — строка или целая переменная с индексами. $Ф$ задает вид, в котором выводятся значения Xi. Если $Ф$ — переменная с индексами, то формат является строкой, закодированной целыми числами (см. табл. П7.1), и состоит из значений последовательных элементов массива, первый элемент массива соответствует открывающей кавычке формата.

Xi — ($i=1, \ldots, m$) переменная, идентификатор массива или выражение (допускаются многомерные переменные и выражения, принимающие многомерные значения).

Элемент вывода вида 4) состоит из последовательности двух или трех аргументов:

$$Ф, T \text{ или } Ф, T, n,$$

где $Ф$ — формат вывода,

T — выводимая строка, формальный параметр-строка или скалярная переменная с индексами типа целый. Если T — переменная с индексами, то выводимая строка закодирована целыми числами (так же как и для формата — переменной с индексами), а T является элементом, соответствующим открывающей кавычке закодированной строки.

n — скалярное выражение типа целый (но не переменная с индексами), задающее *кратность элемента* вывода, т. е. указывающее сколько раз необходимо вывести строку T.

Xi и T называются *объектами вывода*.

Элемент вывода вида 5) или 6) состоит из одного аргумента $Ф$, который называется форматом размещения, или из двух — формата размещения и кратности элемента размещения n.

2.17.1. Элемент числового вывода. Числовой формат бывает трех типов:

а) экспоненциальный формат, употребляемый для выдачи чисел в экспоненциальном виде, т. е. с указанием мантиссы и порядка числа;

б) дробный формат без подавления нулей, употребляемый для выдачи чисел в дробном виде с указанием целой и дробной частей;

в) дробный формат с подавлением нулей; отличается от предыдущего формата тем, что нули, предшествующие значащим цифрам числа заменяются при распечатке пробелами.

Экспотенциальный формат имеет следующий вид:

'*e* ⟨тело экспотенциального формата⟩'

Примеры. По формату '*e—.3d*$_{10}$+*2d*' числа 12.3 и —10.1 будут выведены в следующем виде:

.123$_{10}$+02 —.101$_{10}$+02.

По формату '*e* — *2d.d*$_{10}$ + *2d*' эти числа будут выведены в виде:

12.3$_{10}$ + 00 — 10.1$_{10}$ + 00.

Число перед буквой *d* в формате называется повторителем. Конструкция ⟨повторитель⟩ *d* в формате эквивалентна конструкции, состоящей из соответствующее число раз повторенной буквы *d*, и называется *D*-частью. Для обязательного вывода знака мантиссы или порядка употребляется символ «+».

При выводе нескольких значений (например, массивов) иногда возникает необходимость разделять напечатанные числа пробелами. Например, если выводится массив *A*, элементы которого отделяются друг от друга двумя пробелами, то можно использовать формат:

'*e—2d.d*$_{10}$+*2d2в*'.

Буква *в* вместе с числом, стоящим перед ней в качестве повторителя, называется *B*-частью и задает пропуск нужного количества позиций. *B*-части могут стоять в любых местах строки формата после буквы *e*. Например, при использовании формата '*eв*+*в.dв2d*$_{10}$*вdвd3в*' число 12.3 будет распечатано в виде:

␣+␣1␣23$_{10}$ ␣0␣2␣␣␣.

В экспоненциальном формате могут отсутствовать следующие элементы: точка, знаки порядка и мантиссы, *D*-части перед точкой или перед символом «$_{10}$».

Если формат не охватывает всех значащих цифр мантиссы или порядка, то они не печатаются. Например, оператор

output (*0*, '*ed*$_{10}$*d*', *12.3*)

напечатает число 12.3 в виде

1$_{10}$1.

Можно использовать стандартный формат '*e*', который эквивалентен формату '*e* — *10d*$_{10}$ + *2d2в*'.

Дробный формат без подавления нулей имеет вид:

'*y* ⟨тело дробного формата⟩'.

Тело дробного формата строится аналогично экспоненциальному формату, но не указывается символ «$_{10}$» и порядок *D*-части после точки указывают число символов дробной части числа, а *D*-часть перед точкой — число символов в целой части числа. Если точка отсутствует, то выводится только целая часть. Например, число 12.3 может быть выведено в виде 12.3 с помощью формата '*y2d.d*'. Пусть массив *A* имеет значение: (12.3, 137.82, 0.1), тогда с помощью оператора

output (*0*, '*y2d.2d2в*', *A*)

значение этого массива будет выведено в виде:

12.30 ␣ ␣ 37.82 ␣ ␣ 00.10 ␣ ␣

Дробный формат с подавлением незначащих нулей имеет вид:

'*z* ⟨тело дробного формата⟩'

Этот формат аналогичен предыдущему за исключением того, что старшие незначащие нули целой части числа при распечатке заменяются пробелами. Если целая часть равна нулю, то один нуль сохраняется.

Пусть массив *A* имеет значение (12.3, 137.82, 0, 100), оператор

output (*0*, '*z3dв*', *A*)

выведет значение массива в виде:

␣ 12 ␣ ␣ 137 ␣ ␣ ␣ 0 ␣ 100 ␣

Также как и для экспоненциального формата, для дробных форматов существуют стандартные сокращения в написании:

'*y*' эквивалентно формату '*y6d.6d4в*',
'*z*' эквивалентно формату '*z6d.6d4в*',
'*yi*' эквивалентно формату '*y6dв6d4в*',
'*zi*' эквивалентно формату '*z6dв6d4в*'.

Последние два формата выводят целые части чисел.

При выводе комплексного значения с помощью описанных выше форматов распечатываются два числа, разделенных символом ◇, действительная часть и мнимая. Оба числа распечатываются на одной строке.

Например, пусть действительная часть значения комплексного массива *B* равна (11.3, 12.4, 6.7), мнимая — (0, 1.1, 3.6). Тогда оператор

output (*0*, '*y2d.d2в*', *B*)

выведет значение *B* в следующем виде:

11.3 ␣ ␣ ◇00.0 ␣ ␣ 12.4 ␣ ␣ ◇01.1 ␣ ␣ 06.7 ␣ ␣ ◇03.6 ␣ ␣

2.17.2. Элемент логического вывода. Логический формат имеет вид:

'*l* ⟨тело логического формата⟩'

Тело логического формата содержит только *B*-часть (см. предыдущий пункт) и *F*-части. *F*-часть — это буква *f* с повторителем, которая служит для вывода логического значения в виде последовательности букв TRUE или FALSE.

B-часть может встречаться в любом месте формата после буквы *l*.

Если в формате общее число букв с учетом повторителей больше длины слова TRUE или FALSE, соответствующего выводимому значению, то справа от распечатки этого слова оставляется нужное количество пустых позиций. Например, пусть массив *Л* имеет значение (**истина, ложь, ложь**). Тогда оператор

output (*0*, '*lfв*', *Л*)

выведет *Л* в виде

T_ F_F

Оператор *output* (*0*, '*lвfвfвfвf*', *Л*) выведет значение *Л* в виде:

_T_R_U_E_ _ _F_A_L_S_E_F_A_L_S_E

Стандартный логический формат '*l*' служит для вывода логических значений в виде нулей и единиц. Так массив *Л* в формате '*l*' выведется в виде:

100.

2.17.3. Элемент восьмеричного вывода. Формат восьмеричного вывода имеет вид:

'*o* ⟨тело восьмеричного формата⟩'

Тело восьмеричного формата состоит из *B*-частей и *A*-частей. *A*-часть — это буква *a* с повторителем. Общее количество букв *a* в формате с учетом повторителей, указывает необходимое количество младших восьмеричных цифр (до 16-ти цифр) машинного представления, которое нужно выдать по этому формату. *B*-часть, как и в предыдущих форматах, может стоять везде после буквы *o* и играет ту же роль. Если формат задает вывод более чем 16 восьмеричных цифр машинного представления числа, то недостающие цифры в распечатке дополняются нулями слева.

Пример. Пусть массив *A* типа целый имеет значение (1, 2, 12), тогда оператор

output (*0*, '*o2a2в*', *A*)

выведет эти значения в следующем виде:

01 ␣ ␣ 02 ␣ ␣ 14 ␣

Стандартный формат '*o*' эквивалентен формату '*o16ав*'.

2.17.4. Элемент текстового вывода. Текстовый формат имеет вид:

'*t* ⟨тело текстового формата⟩'

Тело текстового формата состоит из *B*-частей и *A*-частей (см. предыдущие пункты). *A*-части задают вывод символов строки, *B*-части — выдачу пробелов.

Пример. Оператор

output (*0*, '*t 3ава*', 'пример␣строки')

выдает на печать текст в следующем виде

ПРИ␣М␣ЕР␣␣С␣ТРО␣К␣И␣␣.

Общее количество букв *a* в формате с учетом повторителей назовем *интервалом печати*.

Если число выводимых символов строки не кратно числу букв *a* в формате с учетом повторителей, то при выводе строка дополняется пробелами справа.

Если при печати строки символов количество оставшихся позиций до конца строки на странице окажется меньше интервала печати, то оставшаяся часть символов строки будет печататься на следующей строке страницы. Например, если задана разметка страницы с длиной строки равной 7, то текст строки 'пример' в формате '*t3ава*' будет распечатан в две строки:

ПРИ␣М
ЕР␣␣␣.

В выводимых строках двоеточие используется для изменения смысла следующего за ним символа: комбинация *:в* задает пробел при выводе; :/ указывает переход к началу следующей строки; :× — переход к началу следующей страницы; *:1* — задает вывод символа ' ; *:2* — вывод символа ' ; *:3* — вывод символа : ; *:4* — указывает, что код следующего за данной комбинацией символа является повторителем при выводе предыдущего перед комбинацией символа; *:5* — указывает переход на позицию (от начала строки в странице), номер которой закодирован в следующем символе; *:6* — задает вывод символа % ; *:7* — вывод символа ◇ ; *:8* — вывод символа ! ;

:*ю* — указывает переход на предыдущую позицию в строке; :*р* — переход на предыдущую строку с сохранением позиции в строке; :*n* — переход на следующую строку с сохранением позиции в строке.

Пример выдачи текста.

```
output (0, 'ta', 'результаты счета :/');
output (0, 'ta', '*', 16);
output (0, 'tав', ':/тексты', 'тва2вавававава', 'и числа');
output (0, 'ta', ':/все: ю:ю:ю:n — :43:p ␣задания');
```

эти операторы выведут тексты следующим образом:

```
РЕЗУЛЬТАТЫ ␣ СЧЕТА
**********
␣ Т ␣ Е ␣ К ␣ С ␣ Т ␣ Ы ␣ ␣ И ␣ ␣ Ч ␣ И ␣ С ␣ Л ␣ А
ВСЕ ␣ ЗАДАНИЯ
```

2.17.5. Элемент размещения. Формат размещения имеет следующий вид:

$$'П_1E_1 \ldots П_mE_m',$$

где $П_i$ — ($i=1, \ldots, m$) целое число без знака — повторитель компоненты E_i,

E_i — символ (или последовательность символов) — *компонента формата*, некоторые из них перечисляются ниже.

в — указывает переход на следующую позицию в строке,

ю — переход на предыдущую позицию в строке,

р — переход на предыдущую строку с сохранением позиции в строке,

n — переход на следующую строку с сохранением позиции в строке,

$<$ — переход на первую позицию в строке,

$>$ — переход на последнюю позицию в строке,

$\wedge$ — переход на первую строку с сохранением позиции в строке,

$\vee$ — переход на последнюю строку с сохранением позиции в строке,

/ — переход к началу следующей строки,

$\times$ — переход к началу следующей страницы.

Пример 1. Процедура *P* печатает строки символов, выровненные по правому полю страницы.

```
процедура P(s, l); output (0, '>', 'ю', l, 't', s, '/');
```

Следующие операторы

```
P ('строка', 6); P ('длинная ␣ строка', 14);
```

выведут строки текста в виде:

```
␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ СТРОКА
ДЛИННАЯ ␣ СТРОКА
```

Компонента формата *з* позволяет запоминать текущую позицию вывода на странице. После вывода некоторой информации на данной странице можно возвратиться к запомненной позиции, указав в формате размещения компоненту ↑; дальнейший вывод информации будет осуществляться с этой позиции. Процедура *output* может «запомнить» несколько позиций вывода на странице, при этом возврат к запомненным позициям осуществляется в порядке, обратном запоминанию. Если для компоненты ↑ отсутствует парная ей компонента *з*, то возврат осуществляется к началу страницы.

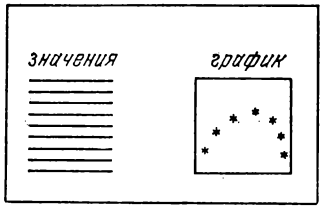


Рис. 2.9. Пример использования возможностей оператора *output*.

Пример 2. Вывести значения функции $P(x)$ и ее график, в котором значения функции отмечены символами * (см. рис. 2.9). Это осуществляется следующей программой.

начало целый *x*;
установка разметки страницы:
marg (*0*, *0*, *61*, *2*, *3*, *52*, *1*);
вывод надписей:
output (*0*, '*t*', 'значение␣функции␣$P(x)$', '<*15в*', '*t*', 'график␣$P(x)$', '*2/з*');
вывод значений *P*:
для $x := 1, \ldots, 51$ **цикл** *output* (*0*, '*e*—.*3*d_{10}+*d*', $P(x)$, '/');
запоминание начальной позиции поля графика:
output (*0*, '↑*10вз*');
вывод графика: **примечание** с нормирующей функцией норм см. (3.16) пример 2;
для $x := -25, \ldots, 25$ **цикл**
output (*0*, '↑*з*', '/', *entier* (*25* — *норм* ($P(x)$)), '*в*', *entier* ($x + 25$), '*t*', '*');
конец программы.

Более сложные компоненты формата размещения, позволяющие защищать участки внутреннего поля страницы путем изменения его размеров, а также перераспределять уже вы-

веденный на страницу текст, описаны в п. 15.4.10, формат размещения.

2.17.6. Элемент вывода номера страницы. Элемент вывода номера страницы состоит из строки, имеющей вид

'*s* ⟨числовой формат⟩'.

Числовой формат указывает формат, в котором выводится номер текущей страницы. Этот формат может быть экспоненциальным или дробным с или без подавления нулей.

Например, оператор

output (*0*, '*szdd.s*')

выведет номер 31-й страницы в следующем виде:

31.␣ .

2.18. Процедура *text*

В языке альфа-6 имеется возможность присваивать последовательным элементам массива целые значения, соответствующие символам строки. Это осуществляется с помощью процедуры *text*, обращение к которой имеет вид:

text (⟨строка⟩, ⟨переменная с индексами⟩).

Переменная с индексами указывает элемент массива, начиная с которого выполняется присваивание целых значений, соответствующих символам строки. Соответствие между символами строки и целыми числами дается в Приложении 7.

Пример. Пусть массив *B* задан в программе описанием

целый массив *B* [*1:11*];

Тогда в результате выполнения оператора

text ('* *бета* * [*i*]', *B* [*1*]);

массив *B* будет иметь следующее значение:

B[*1*] := *26* код открывающей кавычки,
B[*2*] = *25* код *,
B[*3*] := *33* код *б*,
B[*4*] = *37* код *е*,
B[*5*] = *50* код *т*,
B[*6*] = *32* код *а*,
B[*7*] := *25* код *,
B[*8*] = *23* код [,
B[*9*] := *66* код *i*,
B[*10*] = *24* код],
B[*11*] = *27* код закрывающей кавычки.

2.19. Метка *PART*

Программы, требующие для своего выполнения больших объемов оперативной памяти, рекомендуется сегментировать, то есть разбивать на части, которые загружаются динамически, во время выполнения программы. Сегментация может быть осуществлена либо разбиением программы на отдельно транслируемые подпрограммы с последующей их комплексацией (см. п. 2.22), либо расстановкой меток *PART* перед некоторыми блоками программы.

Если блок помечен меткой, идентификатор которой начинается с заглавных букв *PART*, то такая метка называется внешней меткой, или идентификатором части, или меткой *PART*. Блок, помеченный внешней меткой, называется внешним блоком. При трансляции этот блок оформляется в виде отдельного модуля МС Дубна и записывается во временную библиотеку. В основной программе на месте внешнего блока вставляются команды динамического вызова (т. е. загрузки и выполнения) этого модуля.

Сегментация программы с помощью меток *PART* рекомендуется в том случае, когда метками *PART* помечаются несколько не вложенных друг в друга блоков.

2.20. Внешние массивы и операторы обмена

Иногда возникает необходимость в хранении большого количества промежуточных значений, для которых не хватает оперативной памяти машины. В этом случае обычно используют внешнюю память (барабаны, ленты, диски).

В языке альфа-6 имеется возможность задавать эту внешнюю память, а также осуществлять пересылку значений из внешней памяти в оперативную и наоборот.

Внешняя память задается описанием *внешнего массива*, которое отличается от описания обычного массива (внутреннего массива) только тем, что идентификатор внешнего массива начинается с заглавных букв *EX*. Например,

массив *EXA* [*1:2*, *1:30000*]

Непосредственное использование элементов внешнего массива в вычислениях не допускается, можно лишь задавать пересылку значений из внешнего массива во внутренний и обратно. Эта пересылка (обмен) задается с помощью оператора обмена, который имеет один из следующих видов:

copy (*A*, *EXA* [*i*, *j*]) — для записи значений массива во внешний массив *EXA*,

или

copy (*EXA* [*i*, *j*], *A*) — для считывания значений из внешнего массива *EXA* и пересылки их во внутренний массив *A*,

где *i*, *j* — выражения типа целый, задающие начальный элемент внешнего массива, начиная с которого осуществляется пересылка значений.

Типы массивов *A* и *EXA* должны совпадать. Пересылка осуществляется циклически с пересчетом индексов от последнего к первому до тех пор, пока не исчерпаются все элементы внутреннего массива. Например, если массив *A* задан описанием **массив** A [1:2, 1:2], то первый оператор обмена эквивалентен следующим пересылкам:

EXA [*i*, *j*]: = *A* [1, 1]; *EXA* [*i*, *j* + 1]: = *A* [1, 2];
EXA [*i*, *j* + 2]: = *A* [2, 1]; *EXA* [*i*, *j* + 3]: = *A* [2, 2].

Внешний массив является собственным, в том смысле, что при повторном входе в блок, где он описан, старые значения сохраняются. В стандартном режиме для внешних массивов используется барабанная память, которая не сохраняется при повторных запусках задачи. Однако с помощью системной команды * область можно задать в качестве внешней памяти участок на ленте или диске, и сохранить эти значения для последующих запусков задачи.

Имеется возможность осуществлять обмен внутреннего массива непосредственно с участком ленты, диска или магнитным барабаном.

Оператор, осуществляющий такой обмен имеет вид:

copy (*ЧЗ*, *A*, *H*, *M*, *зона*, *сдвиг*),

где *ЧЗ* — признак чтения, если он равен 1, или записи, если нулю, *ЧЗ* может быть выражением типа целый;

A — идентификатор внутреннего массива;

H — номер направления ($0 < H \leqslant 7$);

M — номер устройства ($0 < M \leqslant 7$). *H*, *M* — цифры, образующие математический номер и направление внешнего носителя, если $H \leqslant 2$, то это барабан, иначе лента или диск. При $H \geqslant 3$ ленту (диск) необходимо заказать в паспорте системного пакета (см. Приложение 9). Запрещается использовать следующие комбинации значений *HM*: *01 — 14*, *20 — 33*, (см. п. 8.2);

зона — номер зоны на ленте (диске) или тракта на барабане ($0 \leqslant$ *зона* < 512 для ленты, $0 \leqslant$ *зона* < 995 для диска),

сдвиг — номер ячейки относительно начала указанной зоны (тракта), начиная с которой осуществляется обмен. Сдвиг может превышать длину зоны (тракта), т. е. 1024 ячейки.

Например, оператор *copy* (*0*, *A*, *5*, *2*, *10*, *150*) запишет значения массива *A* на ленту с математическим номером 52 (заказанную в паспорте), начиная с ячейки с номером 150, зоны с номером 10.

Не рекомендуется осуществлять обмен операторами *copy* с лентами (дисками), используемыми для расположения каналов ввода/вывода и каналов *read*, *write* МС Дубна (см. п. 2.14, [11], [13]).

2.21. Оператор *бемш*

Оператор *бемш* позволяет использовать в альфа-программе команды машины БЭСМ-6 [17]. Оператор начинается именем *бемш*, вслед за которым в круглых скобках перечисляются через точку с запятой команды и константы. Любая команда и константа может быть помечена меткой. В качестве метки разрешается использовать только идентификатор.

Каждая команда оператора *бемш* занимает (как правило) половину ячейки и помещается вслед за предыдущей. Если команда помечена меткой, то она всегда располагается в левой половине ячейки.

Следует заметить, что до выполнения первой команды оператора *бемш* устанавливается нулевой режим выполнения команд в АУ (т. е. без блокировки округления).

Команда оператора *бемш* имеет следующую структуру:

коп, *оп* или *коп*, *оп*, *ир*,

где — *коп* — код операции; *оп* — операнд; *ир* — восьмеричное число, указывающее номер индекс-регистра БЭСМ-6. Если в командах используются индекс-регистры, то их следует указать в системной команде * регистры для того, чтобы транслятор не мог их использовать в других местах рабочей программы.

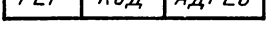


Рис. 2.10. Структура команды БЭСМ-6.

Команду БЭСМ-6 можно представить в виде рис. 2.10, где *рег* — поле индекс-регистра (4 разряда); *код* — код команды (8 или 5 разрядов); *адрес* — адресная часть команды (12 или 15 разрядов).

В качестве кода операции можно использовать либо числовое значение кода машины БЭСМ-6, либо мнемоническое обозначение кода, принятое в автокоде бемш (см. приложение 13).

В качестве операнда может быть идентификатор, восьмеричное число (которое для большинства команд понимается как восьмеричный адрес), литеральная константа или идентификатор плюс (или минус) восьмеричное число.

Идентификатор переменной, массива или метки, являющийся операндом команды, понимается, соответственно, как адрес переменной, адрес первого элемента масива или адрес оператора, помеченного этой меткой.

Идентификатор формальпого параметра (*ФРП*), подставляемого значением, понимается как адрес *ФРП*.

Идентификатор *ФРП*, подставляемый именем, понимается как адрес фактического параметра. При подстановке именем запрещается использовать формальный параметр, которому соответствует фактический параметр — идентификатор процедуры (функции). Формальный параметр, которому соответствуют фактические параметры — переменные различных типов или структур, также не может использоваться в качестве операнда команды *бемш*.

Литеральная константа — это символическая запись значения константы, используемая в качестве значения операнда команды оператора *бемш*. Литеральная константа начинается со знака равенства, за которым следует изображение константы. Изображение константы содержит одну или несколько компонент.

Компоненты константы имеют один из следующих видов:

в 'последовательность восьмеричных цифр'

или

в '— последовательность восьмеричных цифр'

или

т 'последовательность не более шести символов УПП, не содержащая кавычек'.

Перед каждой компонентой константы может быть указан сдвиг (буква *м* и десятичное число).

Компонента константы при переводе в машинное представление располагается начиная с младших разрядов. При наличии сдвига двоичное представление константы сдвигается влево на число разрядов, указанное после буквы *м*.

Если компонента константы начинается со знака минус, то она представляется в дополнительном коде. Например, константа «*в* '—*1*'» эквивалентна «*в* '*7777 7777 7777 7777*'».

Значение константы получается сложением по модулю 2 всех компонент константы с учетом сдвигов. Например, «*м15в'1'в'1'*» эквивалентна «*в'1 00001'*».

Константы в операторе *бемш* делятся на длинные и короткие. Длинные константы имеют один из следующих видов: *конд*, *К*; *конд*, *а*(*И*); *конд*, *а*(*И*(*ИР*)). Короткие константы: *конк*, *К*; *конк*, *а*(*И*); *конк*, *а*(*И*(*ИР*)).

Здесь *К* — изображение константы; *а* — буква «*а*», означающая, что эта константа — адресная; *И* — идентификатор или идентификатор плюс (минус) восьмеричное число; *ИР* — индекс-регистр БЭСМ-6.

Длинная константа занимает ячейку памяти, короткая — половину ячейки. Если короткая константа помечена меткой, то она всегда располагается в левой половине ячейки.

Изображение константы (не адресной) имеет такой же вид, как изображение литеральной константы, но без знака равенства. Короткая адресная константа формируется аналогично длинной команде с нулевым кодом операции, базирование в этом случае не производится. Иными словами, индекс-регистр (*ИР*) записывается в поле *РЕГ*, а адрес операнда *И* — в поле *АДРЕС* (15 разрядов) команды БЭСМ-6 (см. рис. 2.10).

Длинная адресная константа формируется аналогично короткой адресной константе в младших разрядах ячейки (старшие 24 разряда равны нулю).

Пример.

бемш (*ржа*, *3*; *и*, = *в'1'*; *по*, *чет*; *сч*, = *м42в'64'*; *пб*, *М*; *чет*:*сч*, =*м42 в '64' в'1'*; *М*:*зп*, *четнечет*).

Список команд оператора *бемш* с комментариями приведен в Приложении 13.

2.22. Подпрограммы и средства их комплексации

К средствам комплексации альфа-программ относятся:

— описания внешних процедур или процедур-функций (с описателями **альфа**, **фортран**, **алгол**),

— оператор *librari* (п. 2.23),

— общая память (п. 2.24).

Связь между альфа-подпрограммами может осуществляться посредством параметров процедур (процедур-функций) и общей памяти.

Оператор *library* служит для динамического вызова подпрограмм (т. е. загрузки и выполнения подпрограмм в процессе счета) и является наряду с метками *PART* дополнительным средством сегментации программ.

2.22.1. Комплекс альфа-программ. В альфа-программе можно обратиться к другой альфа-программе. Программа, к которой происходит обращение, называется подпрограммой и оформляется как описание процедуры или процедуры-функции языка альфа-6. Параметры процедуры или процедуры-функции (параметры подпрограммы) в этом случае должны иметь спецификации. В отличие от языка алгол-60 в спецификации массива должна быть указана размерность массива. Например, запись

вещественный массив *A*, *B*—*2*+*1*; **массив** *C*—*3*; **массив** *D*;

означает, что массивы *A*, *B* имеют абсолютную размерность 3, внешнюю — 2, внутреннюю — 1, массив *C* имеет абсолютную размерность 3, внутреннюю — *0*, внешняя размерность массива *D* — 1, внутренняя — *0*.

Альфа-программа, которая содержит обращение к альфа-подпрограмме, должна иметь описание имени данной подпрограммы в виде:

альфа ⟨тип⟩ ⟨идентификатор⟩,

где ⟨идентификатор⟩ есть имя процедуры, которая реализует данную подпрограмму, или

альфа ⟨тип⟩ ⟨идентификатор⟩,

если ⟨идентификатор⟩ есть имя процедуры-функции. Описатель **альфа**, как и все другие описатели языка альфа-6, может специфицировать несколько подпрограмм, идентификаторы которых в этом случае перечисляются через запятую, например,

альфа *ПРОГ1*, **вещественный** *ПРОГ2*, *ПРОГ3*.

Пример.

```
ПРОБА:
    начало массив A [1:2, 1:2, 1:3]; альфа ПРОГ1;
      . . . . . .
      ПРОГ1(A);
      . . . . . .
    конец
    процедура ПРОГ1(A); массив A — 2 + 1;
      начало
      . . . . . .
      A[i, j]: = A[i, j] ↑ 2;
      . . . . . .
      конец
```

Таким образом, можно говорить о связанных между собой альфа-программах, предназначенных для решения одной задачи. Каждая программа комплекса или содержит обращение к некоторой программе этого комплекса, или вызывается из какой-либо программы комплекса.

Головной программой комплекса называется программа, с которой начинается выполнение задачи. Головная программа является блоком в смысле языка альфа-6 и не может быть вызвана из других программ комплекса.

Отметим, что в качестве фактического параметра при обращении к подпрограмме нельзя употреблять идентификатор переключателя или процедуры (процедуры-функции), не являющейся именем подпрограммы.

2.22.2. Комплекс из альфа-программ, фортран-программ и алгол-программ. Комплекс программ может содержать кроме альфа-программ также программы, написанные на языках фортран и алгол-60, трансляторы с которых входят в состав МС Дубна [13], [12].

Альфа-программа, которая содержит обращение к фортран-подпрограмме или алгол-подпрограмме, должна иметь описание имени этой подпрограммы, вид описания аналогичен описанию имен альфа-подпрограмм, а именно:

фортран ⟨идентификатор⟩;

или

алгол ⟨идентификатор⟩;

Обращение к фортран-подпрограмме или алгол-подпрограмме имеет вид оператора процедуры или указателя функции (если задан ⟨тип⟩ перед ⟨идентификатором⟩) в смысле языка альфа-6.

Головной программой комплекса может быть также программа, написанная на языке фортран или алгол-60.

Обращение к альфа-подпрограмме из фортран-программы (алгол-программы) осуществляется также как к фортран-подпрограмме (алгол-подпрограмме). Отличие есть лишь в использовании фактического параметра-идентификатора массива, после которого в списке параметров нужно указать для каждого измерения массива его нижнюю границу и длину. Например, для вызова альфа-подпрограммы P с фактическим параметром A, где A — массив с граничными парами «*1:2*, *3:5*», в фортран-программе нужно написать следующий оператор:

$call\ \_ P(A, 1, 2, 3, 2)$.

2.22.3. Использование мадлен-подпрограмм и бемш-подпрограмм в комплексе программ.

Комплекс программ может также включать в себя подпрограммы, написанные на автокодах мадлен и бемш, которые составляются по правилам их использования в фортран-программах [13]. При комплексации в альфа-программах мадлен-подпрограммы и бемш-подпрограммы рассматриваются как фортран-подпрограммы (используется описатель **фортран**).

2.23. Оператор *library*

Если имеется оттранслированная альфа-подпрограмма, находящаяся в библиотеке МС Дубна, например, *ПРОГ1* с параметрами X и Y, где X — вещественный скаляр, Y — массив, то к ней можно обратиться в альфа-программе с помощью оператора *library*. Для этого необходимо поместить в начале блока, в котором есть обращение, описание процедуры, например, P с параметрами, которые соответствуют параметрам подпрограммы *ПРОГ1*, и телом которой является оператор *library*:

процедура $P(X, Y)$; **вещественный** X; **массив** Y;
library (*'ПРОГ1'*);

Оператор процедуры $P(a, в)$ служит обращением к подпрограмме *ПРОГ1*. Если *ПРОГ1* является процедурой-функцией, то перед описателем **процедура** необходимо указать ее тип. Спецификация формальных параметров может отсутствовать.

Особенности использования оператора *library* при комплексации альфа-программ см. в гл. 9.

2.24. Общая память

Общая память, наряду с механизмом формальных и фактических параметров, является средством передачи значений переменных из одной подпрограммы в другую, эта передача обеспечивается совмещением по памяти переменных из различных подпрограмм. Общая память — участки оперативной памяти машины, используемые при совместном выполнении отдельно транслируемых подпрограмм [13].

Пусть, например, для совместной работы подпрограмм *ПР1* и *ПР2* необходимо иметь общую память длиной 5 ячеек. В подпрограмме *ПР1* на этом участке ОЗУ располагаются: комплексные переменные z, $z1$ и вещественный скаляр C. В подпрограмме *ПР2* на этой общей памяти располагаются:

вещественный скаляр *B* и массив *A* длиной 4 ячейки (см. рис. 2.11).

Во время выполнения подпрограмм *ПР1* и *ПР2* переменные, которые оказались совмещенными по памяти (скаляр *B* совмещен с действительной частью переменной *z*, элемент массива *A* [*4*] — со скаляром *C* и т. п.), имеют одинаковые значения.

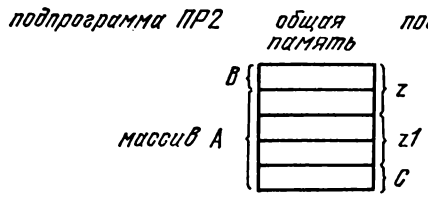


Рис. 2.11. Пример общего массива.

Изменение значения переменной в одной подпрограмме ведет к изменению значения совмещенной с ней переменной из другой подпрограммы и наоборот.

Порядок расположения элементов на общей памяти может задавать также частичное совмещение переменных, например, скаляр может быть совмещен с элементом массива или часть массива может быть совмещена с частью другого массива и т. п.

Если массив *A* описан в подпрограмме *ПР2* как целый, то это может привести к неправильным вычислениям при выполнении программы, например, при вычислении совмещенных с ним по памяти переменных *z1*, *C* в массив *A* будут засылаться вещественные значения.

Общая память определяется при помощи описания общих массивов, которое имеет вид:

общий '*имя*', *A1*, ..., *Ak*

либо

общий *A1*, ..., *Ak*,

где

имя — имя общего массива, по которому идентифицируются участки общей памяти для разных подпрограмм. Если в описании отсутствует имя общего массива, то это означает, что задан непомеченный общий массив.

'*имя*' — строка длиной не более 8 (с учетом кавычек) символов. Этими символами могут быть только буквы, цифры и знаки +, —, /, *.

A1, ..., *Ak* — список переменных, определяющий порядок расположения переменных на общей памяти.

Ai — общий объект ($i = 1, \ldots, k$) имеет вид *И* либо *И*(*Ц*), где *И* — идентификатор массива или скаляра, *Ц* — целое число (или ноль), указывающее количество ячеек в общем массиве, зарезервированных для переменной *И*.

Если длина резервируемой области не указана, то общий объект занимает в общем массиве столько ячеек, сколько требует описание данного объекта, т. е. для целых, вещественных и логических скаляров резервируется одна ячейка, для комплексного скаляра — две ячейки, для статических массивов — количество ячеек, необходимое для его размещения в общем массиве. Для динамических массивов резервирование ячеек не производится.

Указатель длины (*Ц*) является также удобным средством для совмещения различных переменных по памяти в пределах одной подпрограммы.

Пример 1.

Пусть в подпрограмме описан вещественный скаляр *B* и вещественный массив *A* длиной 4 ячейки. Тогда при помощи описания

общий *'ПРИМ'*, *A*(*3*), *B*

определяется общий массив с именем *ПРИМ* длиной 4 ячейки, причем скаляр *B* будет совмещен по памяти с четвертым элементом массива *A*, так как под *A* выделено 3 ячейки.

Пример 2.

Пусть *K* — комплексный, а *X* и *Y* — вещественные скаляры. Использованием нулевого указателя длины при помощи описания

общий *K*(*0*), *X*, *Y*

определяется непомеченный общий массив длиной 2 ячейки. Скаляр *X* совмещен по памяти с действительной частью комплексного скаляра *K*, а *Y* — с мнимой частью этого же скаляра.

Пример 3.

Пусть *Д* — динамический массив, *A* и *B* — скаляры и имеется описание

общий *'ДИН'*, *Д*, *A*, *B*.

В момент трансляции подпрограммы с этим описанием длина массива *Д* неизвестна, поэтому это описание считается эквивалентным следующему:

общий *'ДИН'*, *Д*(*0*), *A*, *B*.

Все переменные, входящие в список переменных описания общего массива, должны иметь свои описания, а само описание общего массива должно находиться в области действия всех перечисленных в нем переменных.

В программе допускается несколько описаний общих массивов, в том числе, с одним и тем же именем общего массива. Однако объекты *A1*, ..., *Ak* в списках после имен общих массивов должны быть различными во всех описаниях общих массивов. Если в программе имеется несколько описаний общего массива с одним и тем же именем (в одном или в разных блоках), то все объекты объединяются в один общий массив с этим именем в том порядке, в каком следуют в описаниях общего массива.

Пример 4.

```
начало вещественный  c, b; массив A [1:2, 1:3];
общий 'ОБЩ', A(2), c;
начало комплексный Z; общий 'ОБЩ', Z;
начало массив A1 [1:2]; общий 'ОБЩ', A1, b;
конец
конец
конец
```

В этом случае для общего массива *ОБЩ* в оперативной памяти отведется 8 ячеек:

ОБЩ [*1*] ≡ *A* [*1, 1*]
ОБЩ [*2*] ≡ *A* [*1, 2*]
ОБЩ [*3*] ≡ *C* ≡ *A* [*1, 3*]
ОБЩ [*4*] ≡ действительная часть *Z* ≡ *A* [*2, 1*]
ОБЩ [*5*] ≡ мнимая часть *Z* ≡ *A* [*2, 2*]
ОБЩ [*6*] ≡ *A1* [*1*] ≡ *A* [*2, 3*]
ОБЩ [*7*] ≡ *A1* [*2*]
ОБЩ [*8*] ≡ *b*

Если для совместной работы альфа- и фортран-подпрограмм требуется общая память, то идентификатор общего блока в операторе *common* (фортран-программы) должен совпадать с именем общего массива в описании общего массива (альфа-программы). Непомеченный общий блок в фортран-программе и непомеченный общий массив в альфа-программе во время совместного выполнения этих программ будут располагаться на общем участке ОЗУ.

Для идентификации общих блоков в программе на автокоде мадлен и общих массивов в альфа-программе, нужно в мадлен-программе использовать имя общего массива, заключенное между двумя символами * (непомеченному общему массиву будет соответствовать общий блок с именем **).

2.25. Упакованные массивы

В альфа-программе логические массивы располагаются по одному элементу в ячейке Например, логический массив *A* [*1:4*, *1:30*] занимает 120 ячеек памяти.

Для экономии памяти при хранении логических массивов в язык альфа-6 введен описатель типа **упакованный**, который может стоять непосредственно перед описателем **логический** в описании массива или в описании типа и структуры.

Например,

упакованный логический массив *C* [*1:4*, *1:30*];
упакованный логический массив *B*-**вектор** *100*.

В этом случае массив *C* займет 4 ячейки памяти, а вектор *B* — 3 ячейки Расположение элементов упакованного логического массива в памяти покажем на следующем примере.

Пусть имеется описание

упакованный логический массив *A* [*1:2*, *1:3*, *1:50*].

Расположение в памяти этого массива показано на рис. 2.12. Следует отметить, что использование логических упа-

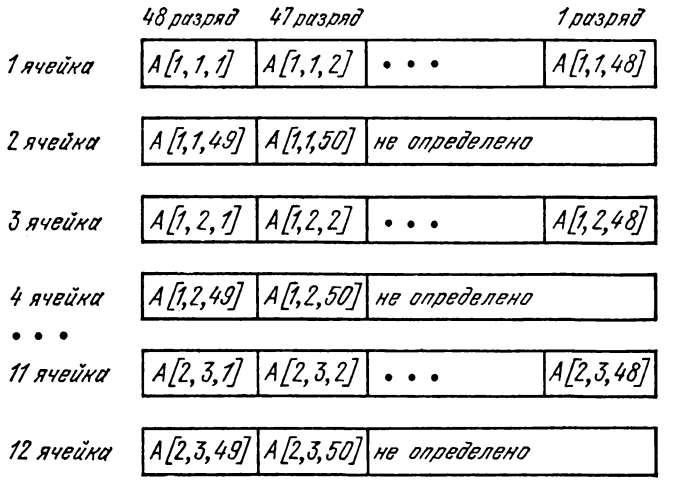


Рис. 2.12. Расположение элементов логического упакованного массива в ячейках БЭСМ-6.

кованных массивов ведет, как правило, к увеличению времени работы программы и увеличению ее длины. Исключение со-

ставляет случай использования упакованных массивов с пустым индексным выражением по последнему измерению. Например, $C[1,]:=\neg C[2,]$. В этом случае операция будет производиться не над отдельными элементами массива C, а над векторами.

ГЛАВА 3

КАК РАБОТАТЬ С СИСТЕМОЙ (10 ПРИМЕРОВ)

В этой главе мы рассмотрим примеры комплектации пакетов для запуска системы Альфа-6 в наиболее часто встречающихся на практике режимах ее эксплуатации.

Пример 1. Написать альфа-программу, вычисляющую и печатающую значения функции

$$y(x)=\sin(x^2/(x^2+1))$$

с шагом 0.1 на отрезке [0, 1].

Воспользуемся разложением в ряд

$$\sin x=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\ldots \qquad (1)$$

и прекратим суммирование, когда очередной член ряда по модулю станет меньше 10^{-6}.

Нетрудно убедиться, что программа *ПРОБА*, записанная ниже, решает поставленную задачу.

```
ПРОБА:
◊1◊  начало массив y [0:10]; целый i;
       вещественный x;
◊2◊  вещественный процедура si(x); значение x;
       вещественный x;
◊3◊  начало целый i; вещественный s, a;
◊4◊  s:=a:=x; i:=3;
◊5◊  для a:=-a×x×x/(i×(i-1))
◊6◊  пока abs(a) ≥ 10-6 цикл
◊7◊  начало s:=s+a; i:=i+2 конец;
◊8◊  si:=s конец процедуры si;
◊9◊  для i:=0 шаг 1 до 10 цикл
◊10◊ начало x:=(i×0.1)↑2;
       y[i]:=si(x/(x+1)) конец;
◊11◊ output (0, 'e', y) конец
```

Предположим, что программа *ПРОБА* отперфорирована в кодировке УПП для БЭСМ-6. Тогда следующий пакет служит для трансляции, загрузки и выполнения полученной программы.

```
ШИФР _777700 ЗС +¬
ЕЕВ1А3
```

```
*NAME _ ПРОБА
*ASSIGN _ LIBRARY _ N
* CALL _ ALPHA/6
◇ ◇ ◇ ПРОГРАММА
(программа ПРОБА на перфокартах)
◇ ◇ ◇ КНЦ
* EXECUTE
* END _ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Здесь N — номер библиотеки подпрограмм Альфа-6.

При запуске системы Альфа-6 с этим пакетом программа *ПРОБА* будет переведена с языка альфа-6 на язык модуля загрузки МС Дубна, загружена и выполнена, на АЦПУ будут напечатаны 11 значений: $y[0]$, ..., $y[10]$. Если нужно распечатать текст программы *ПРОБА* на языке альфа-6, то в пакет после карты * CALL _ ALPHA/6 необходимо вставить карту ◇ ◇ ◇ СИСТЕМА и карту * ПЕЧАТАТЬ _ ПРОГРАММУ.

Пример 2. Записать программу *ПРОБА* после трансляции на ленту с номером 42, начиная с зоны 100.

```
ШИФР _ 777700 ЗС +⁻
ЛЕНТА _ 42 (199 — ЗП)⁻
ЕЕВ1А3
* NAME _ ПРОБА
* ASSIGN _ LIBRARY _ N
* CALL _ ALPHA/6
◇ ◇ ◇ ПРОГРАММА
(программа ПРОБА на перфокартах)
◇ ◇ ◇ КНЦ
* TO _ PERSONAL _ LIBRARY:42100, 100
* END _ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

При запуске системы с этим пакетом программа *ПРОБА* будет переведена на язык модуля загрузки МС Дубна и (без выполнения) записана, начиная с зоны 100, в личную библиотеку пользователя МС Дубна длиной 100 зон, которая находится на бобине с номером 199. При дальнейших запусках загрузка и выполнение программы *ПРОБА* может происходить без вызова системы Альфа-6 средствами МС Дубна. Например, следующий пакет вызывает программу *ПРОБА* из личной библиотеки, загружает и выполняет полученную программу.

```
ШИФР_ 777700 ЗС+-
ЛЕНТА_ 42 (199)-
ЕЕВ1А3
* NAME_ ПРОБА
* ASSIGN_ LIBRARY_N
* PERSONAL_ LIBRARY:42100
* MAIN_ ПРОБА
* EXECUTE
* END_ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Замечание. Номер библиотеки, указанный в перфокарте * ASSIGN_ LIBRARY устанавливается организацией, эксплуатирующей систему Альфа-6.

Пример 3. Текст программы *ПРОБА* на языке альфа-6 можно хранить во внутренней библиотеке (архиве) системы Альфа-6. Создание библиотеки с именем *АЛЬФА* на ленте с математическим номером 42 (бобина 199) осуществляется следующим пакетом.

```
ШИФР_777700 ЗС +-
ЛЕНТА_ 42 (199 — ЗП)-
ЕЕВ1А3
* NAME_ ЗАПОГЛ
* ASSIGN_ LIBRARY_N
* CALL_ ALPHA/6
◇ ◇ ◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ЗАПИСАТЬ_ ОГЛАВЛЕНИЕ (АЛЬФА)
◇ ◇ ◇ КНЦ
* END_ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Запись программы *ПРОБА* (без трансляции) в архив с именем *АЛЬФА* производится следующим пакетом

```
ШИФР _777700 ЗС +-
ЛЕНТА_ 42 (199 — ЗП)-
ЕЕВ1А3
* NAME_ ПРОБА
* ASSIGN_ LIBRARY _N
* CALL_ ALPHA/6
◇ ◇ ◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
```

```
* ЗАПИСАТЬ _ ПРОГРАММУ (ПРОБА, АЛЬФА)
* БЕЗ _ ТРАНСЛЯЦИИ
◇ ◇ ◇ ПРОГРАММА

⟨программа ПРОБА на перфокартах⟩

◇ ◇ ◇ КНЦ
* END _ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Пример 4. Пусть программа *ПРОБА* из примера 1 находится в архиве *АЛЬФА* (см. пример 3). Рассмотрим программу *ПРОБА1*, которая отличается от программы *ПРОБА* тем, что суммирование прекращается, когда очередной член разложения (I) станет по модулю меньше 10^{-n}, где n — некоторый входной параметр программы, задающий точность вычислений.

```
ПРОБА1:
◇1◇   начало массив y[0:10]; целый i;
      вещественный x;
◇2◇   вещественный процедура si(x, n); значения x, n;
      вещественный x; целое n;
◇3◇   начало целый i; вещественный s, a;
◇4◇   s:=a:=x; i:=3;
◇5◇   для a:=—a×x×x/(i×(i—1))
◇6◇   пока abs(a) ≥ 10↑(—n) цикл
◇7◇   начало s:=s+a; i:=i+2 конец;
◇8◇   si:=s конец процедуры si;
◇8.1◇ целое n; input (0, n);
◇9◇   для i: = 0 шаг 1 до 10 цикл
◇10◇  начало x: = (i×0.1)↑2;
      y [i]:=si(x/(x+1), n) конец;
◇11◇  output (0, 'e', y) конец
```

Следующий пакет производит редактирование программы *ПРОБА*, превращая ее в программу *ПРОБА1*

```
ШИФР _777700 ЗС +⁻
ЛЕНТА _ 42 (199 — ЗП)⁻
ЕЕВ1А3
* NAME _ ПРОБА
* ASSIGN _ LIBRARY _ N
* CALL _ ALPHA/6
◇ ◇ ◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА, АЛЬФА)
```

```
* ВСТАВИТЬ (8, 8 1)
* ЗАМЕНИТЬ (2, 2)
* ЗАМЕНИТЬ (6, 6)
* ЗАМЕНИТЬ (10, 10)
* ЗАПИСАТЬ_ПРОГРАММУ (ПРОБА1, АЛЬФА)
* БЕЗ_ТРАНСЛЯЦИИ
◇◇◇ ЗАМЕНЫ
◇8.1◇ целое n; input (0, n);
◇2◇   вещественный процедура si(x, n); значения
        x, n;
      вещественный x; целое n;
◇6◇   пока abs(a)≥10↑(—n) цикл
◇10◇  начало x:=(i×0.1)↑2;
        y[i]:=si(x/(x+1), n)
конец;
◇◇◇ КНЦ
* END_FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Пример 5. Трансляция и выполнение программы *ПРОБА1* из примера 4 осуществляется следующим пакетом

```
ШИФР_777700 ЗС +⁻
ЛЕНТА_42 (199)⁻
ЕЕВ1А3
* NAME_ПРОБА1
* ASSIGN _ LIBRARY _ N
*CALL _ ALPHA/6
◇◇◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА1, АЛЬФА)
◇◇◇ ЗАДАНИЕ
◇◇◇ ДАННЫЕ
n=10;
◇◇◇ КНЦ
* EXECUTE
* END_ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Разумеется, что программа *ПРОБА1*, как и в примере 1, может находиться на перфокартах.

Пример 6. Пусть программу *ПРОБА1* из примера 4 после трансляции необходимо выполнить дважды: при $n=10$ и при $n=11$. Наиболее просто это можно сделать так.

```
ШИФР _ 777700 ЗС +-
ЛЕНТА _ 42 (199)-
ЕЕВ1А3
* NAME _ ПРОБА1
* ASSIGN _ LIBRARY _ 7
* CALL _ ALPHA/6
◇ ◇ ◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА1, АЛЬФА)
◇ ◇ ◇ ЗАДАНИЕ
◇ ◇ ◇ ДАННЫЕ
n=10; 11;
◇ ◇ ◇ КНЦ
* EXECUTE
* EXECUTE
* END _ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Пример 7. Исходные данные для вычислений могут также быть записаны в архив. Например, следующий пакет осуществляет действия, аналогичные пакету из примера 6, но одновременно записывает данные $n=10;\ 11;$ в архив *АЛЬФА*, используя эту запись в следующем задании.

```
ШИФР _ 777700 ЗС +-
ЛЕНТА _ 42 (199 — ЗП)-
ЕЕВ1А3
* NAME _ ПРОБА1
* ASSIGN _ LIBRARY _ 7
* CALL _ ALPHA/6
◇ ◇ ◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ЗАПИСАТЬ _ ДАННЫЕ (ПРОБА1, АЛЬФА)
* БЕЗ _ ТРАНСЛЯЦИИ
◇ ◇ ◇ ДАННЫЕ
n=10; 11;
◇ ◇ ◇ ЗАДАНИЕ
◇ ◇ ◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА1, АЛЬФА)
◇ ◇ ◇ ЗАДАНИЕ
◇ ◇ ◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ДАННЫЕ (ПРОБА1, АЛЬФА)
```

```
◊ ◊ ◊ КНЦ
* EXECUTE
* EXECUTE
* END␣ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Пример 8. Рассмотрим следующую программу.

ПРОБА2:

```
◊1◊   начало массив y [0:10]; целый i;
      вещественный x;
◊2◊   вещественный процедура cos i (x, n); значения
      x, n; вещественный x; целое n;
◊3◊   начало целый i; вещественный s, a;
◊4◊   s:=a:=1; i:=2;
◊5◊   для a:=−a×x×x/(i×(i−1));
◊6◊   пока abs(a) ≥ 10↑(−n) цикл
◊7◊   начало s:=s+a; i:=i+2 конец;
◊8◊   cos i:=s конец процедуры cos i;
◊ 8.1 ◊ целый n; input (0, n);
◊9◊   для i: = 0 шаг 1 до 10 цикл
◊10◊  начало x:=(i×0.1)↑2;
      y [i]:=cos i(x/(x+1), n) конец;
◊11◊  output (0, 'e', y) конец
```

Программа *ПРОБА2* вычисляет и печатает значения функции

$$y(x) := cos[x^2/(x^2+1)]$$

с шагом 0,1 на отрезке [0, 1]. Следующий пакет производит текстовое редактирование программы *ПРОБА1* из примера 4, в результате которого получается программа *ПРОБА2*, и последняя записывается в архив *АЛЬФА*.

```
ШИФР ␣777700 ЗС +−
ЛЕНТА␣ 42 (199 — ЗП)−
ЕЕВ1А3
* NAME␣ ПРОБА1
* ASSIGN␣ LIBRARY ␣N
* CALL␣ ALPHA/6
◊◊◊ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА1, АЛЬФА)
* ЗАМЕНИТЬ␣ ТЕКСТ ('si', 'cos i')
* ЗАМЕНИТЬ␣ ТЕКСТ ('x; i:=3', '1; i:=2')
```

```
* ЗАПИСАТЬ _ ПРОГРАММУ (ПРОБА2, АЛЬФА)
* БЕЗ _ ТРАНСЛЯЦИИ
◇◇◇ КНЦ
* END_ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Пример 9. Следующий пакет печатает на АЦПУ фрагмент программы *ПРОБА1*, находящийся на перфокартах 2—8 и выдает текст всей программы *ПРОБА1* на перфорацию в кодировке УПП для БЭСМ-6.

```
ШИФР _ 777000 ЗС +⁻
ЛЕНТА _42 (199 — ЗП)⁻
ЕЕВ1А3
* NAME _ ПРОБА1
* ASSIGN _ LIBRARY _ N
* CALL _ ALPHA/6
◇◇◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА1, АЛЬФА)
* ПЕЧАТАТЬ _ ПРОГРАММУ (2:8)
* ПЕРФОРИРОВАТЬ _ ПРОГРАММУ
* БЕЗ _ ТРАНСЛЯЦИИ
◇◇◇ КНЦ
* END _ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Пример 10. Пусть функция DIV(x, y), написанная на языке фортран, реализует деление x на y с двойной точностью. Мы хотим воспользоваться этой функцией для деления внутри процедуры $si(x, n)$ из программы *ПРОБА1* (пример 4). Это осуществляется запуском следующего пакета.

```
ШИФР _ 777700 ЗС +⁻
ЛЕНТА    42 (199)⁻
ЕЕВ1А3
* NAME _ ПРОБА1
* ASSIGN _ LIBRARY _ N
          _ FUNCTION _ DIV(x, y)   ⎫
. . . . . . . . . . . . . . . .    ⎬ фортран-функция
          _ END                    ⎭ DIV (x, y) на перфокартах
* CALL _ ALPHA/6
◇◇◇ СИСТЕМА
```

```
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА1, АЛЬФА)
* ВСТАВИТЬ (1, 1.0)
* ЗАМЕНИТЬ (5, 5)
◇◇◇ ЗАМЕНЫ
◇1.0◇  фортран вещественный DIV;
◇5◇    для a:=DIV(-a×x×x, i×(i-1))
◇◇◇ КНЦ
* EXECUTE
* END_FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

РАЗДЕЛ II

СИСТЕМА АЛЬФА-6

ГЛАВА 4

ФУНКЦИОНИРОВАНИЕ СИСТЕМЫ АЛЬФА-6

4.1. Выполнение последовательности заданий

Инициация работы системы Альфа-6 осуществляется управляющей картой МС Дубна

```
* CALL _ALPHA/6.
```

Непосредственно за этой картой в пакет помещается последовательность заданий (альфа-задача), заканчивающаяся картой конца альфа-задачи (◇◇◇ КНЦ). Соседние задания разделяются картой ◇◇◇ ЗАДАНИЕ.

Функционирование системы Альфа-6 состоит в последовательном выполнении всех ее заданий, присутствующих в альфа-задаче. После обработки последнего задания происходит автоматический вызов МС Дубна, которая настраивается на выполнение управляющей карты, располагающейся в пакете непосредственно за картой конца альфа-задачи. При этом, если в результате работы системы Альфа-6 произошла трансляция, то полученный модуль записывается во временную библиотеку МС Дубна и становится доступным для обработки лишь после окончания работы системы Альфа-6 в соответствии с общими правилами обработки модулей в рамках МС Дубна. Отметим также, что данные, подготовленные для системы Альфа-6, перерабатываются и передаются МС Дубна посредством каналов данных для их использования при выполнении программы после ее загрузки. Необходимо помнить, что других средств обмена информацией между Альфа-6 и МС Дубна не существует. То есть в процессе своей работы система Альфа-6 может использовать только свои системные команды, только свои средства редактирования и только свои библиотеки. Интерфейс между системой Альфа-6 и МС Дубна графически

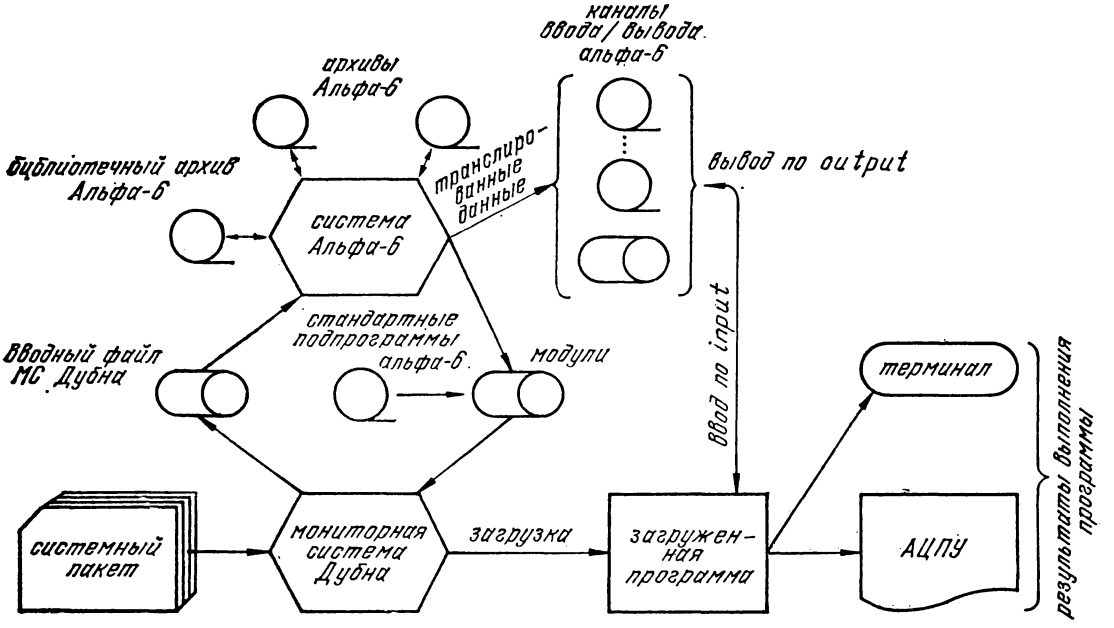


Рис. 4.1. Схема взаимодействия системы Альфа-6 и МС Дубна.

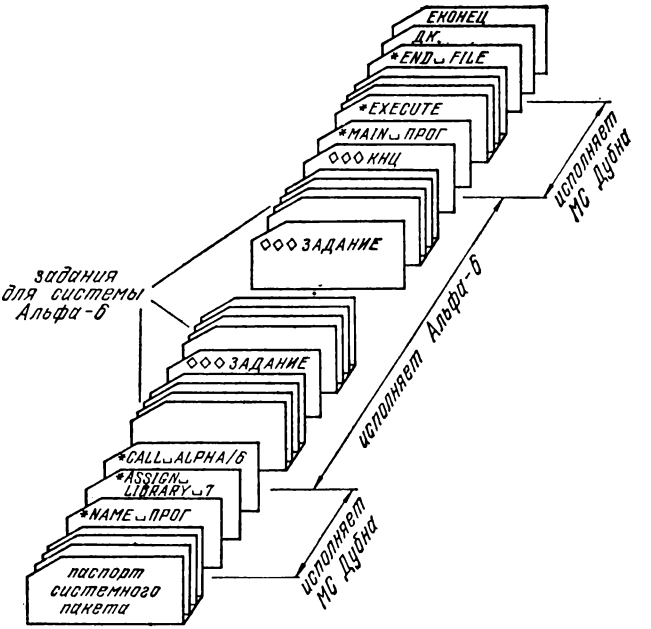


Рис. 4.2. Комплектация пакета для запуска системы Альфа-6.

представлен на рис. 4.1. Комплектация пакета для пуска системы Альфа-6 в общем виде изображена на рис. 4.2.

Пакет в целом должен отвечать правилам комплектации пакетов в МС Дубна. В частности, он всегда начинается паспортом и заканчивается картами ⟨диспетчерский конец⟩, ЕКОНЕЦ.

4.2. Выполнение задания

Выполнение отдельного задания в системе Альфа-6 состоит в выполнении некоторой совокупности из следующих основных работ: трансляция программы, подготовка данных, редактирование программы и данных, печати и перфорации, библиотечные работы (в дальнейшем библиотеки Альфа-6, для отличия от библиотек МС Дубна, называются архивами). Состав и порядок выполнения работ в задании определяется структурой пакета и перфокарт, составляющих это задание. Задание собирается из фрагментов четырех типов: системная программа, программа, данные, фрагмент замен. Фрагмент каждого типа может входить в задание не более одного раза.

Фрагмент «системная программа» начинается картой ◇◇◇ СИСТЕМА и состоит из совокупности системных команд Альфа-6. Каждая системная команда из этой совокупности задает некоторую работу, выполняемую в задании.

Фрагмент «программа» начинается картой ◇◇◇ ПРОГРАММА и состоит из альфа-программы, структурированной номерами перфокарт. Если в задании присутствует фрагмент «программа», то система выполняет работы, связанные с альфа-программой и заданные в системной программе задания. При отсутствии системной программы осуществляется только трансляция программы (см. п. 4.3).

Фрагмент «данные» начинается картой ◇◇◇ ДАННЫЕ и состоит из альфа-данных, структурированных номерами перфокарт. Если в задании присутствует фрагмент «данные», то система выполняет работы, связанные с данными (подготовка данных, печать, редактирование, перфорация и т. п.).

В одном задании фрагменты «программа» и «данные» одновременно присутствовать не могут.

Фрагмент «замены» начинается картой ◇◇◇ ЗАМЕНЫ и содержит некоторые тексты на языке альфа-6 или языке данных, структурированные номерами перфокарт. Эти тексты система использует для замены или вставки частей текстов программы или данных при их редактировании.

Порядок вхождения фрагментов в задание не существен. Информацию о совместимости отдельных работ в задании можно извлечь из графа работ (Приложение 12). Если работы в одном задании не совместимы, то часто их все же можно выполнить за один вызов системы Альфа-6, поместив эти работы в различные задания одной и той же альфа-задачи.

При выполнении каждого задания система обнаруживает ошибки в комплектации задания, а также синтаксические и семантические ошибки в системной программе, в исходной альфа-программе (в случае ее трансляции) и в данных (в случае их обработки). При обнаружении ошибки на АЦПУ выдается сообщение об ошибке, система прекращает выполнение работ ошибочного задания, но продолжает поиск других ошибок в задании до тех пор, пока это возможно. После этого происходит переход к выполнению последующих заданий, которые обрабатываются в обычном режиме. Однако, если системная программа одного из последующих заданий содержит системную команду

* ОСТАНОВ _ ПРИ _ ОШИБКЕ

то это и следующие за ним задания не обрабатываются и происходит выход в МС Дубна.

4.3. Выполнение работ

Трансляция программы состоит в переводе исходной программы, записанной на входном языке альфа-6, в эквивалентную ей программу, записанную на языке модуля загрузки МС Дубна. При выходе из системы после обработки последовательности заданий все оттранслированные программы находятся во временной библиотеке МС Дубна. Оттранслированную программу можно выполнять, загружать совместно с другими программами, заносить в личную библиотеку в соответствии с общими правилами выполнения этих работ в системе Дубна.

Подготовка данных заключается в переводе исходных альфа — данных в данные, воспринимаемые МС Дубна. Подготовленные данные записываются в каналы данных, которые располагаются на барабанах, дисках или лентах и воспринимаются при выполнении программ, использующих эти данные.

Редактирование программы или данных заключается в изменении текста программы или данных в соответствии с правилами редактирования в системе Альфа-6. Пользователю предоставляются следующие возможности редактирования.

При *поперфокартном* редактировании программа или данные рассматриваются как последовательность перфокарт. Под перфокартой при этом понимается часть текста программы или данных, расположенная между двумя соседними ограничителями — номерами перфокарт (см. гл. 13). Поперфокартное редактирование состоит либо в замене совокупности рядом стоящих перфокарт на другую совокупность (может быть и пустую), взятую из фрагмента замен, либо во вставке между двумя соседними перфокартами некоторой совокупности перфокарт из фрагмента замен.

При *текстовом* редактировании все вхождения некоторого текста в программу или данные заменяются на вхождения некоторого другого текста. Заменяемым текстом может быть и идентификатор в программе или в данных.

Отладочное редактирование состоит во вставке в исходную альфа-программу некоторых операторов, которые помогают программисту осуществлять отладку программы после ее трансляции.

Печать и *перфорация* позволяют выдать на АЦПУ или на перфокарты программу, данные и модуль загрузки (частично или полностью).

Библиотечные работы производят следующие операции с архивами Альфа-6:

— создание архива,
— запись программы или данных в архив,
— уничтожение программы или данных из архива,
— печать оглавления архива,
— изменение имени архива.

4.4. Виды заданий

По составу выполняемых работ различаются следующие виды заданий:
— задание с программой,
— задание с данными,
— задание с архивами.

В задании с программой присутствует либо фрагмент «программа» (альфа-программа на перфокартах), либо системная команда * ПРОГРАММА с именем альфа-программы (программа в архиве). Задание с программой состоит из следующей последовательности работ. Сначала происходит предварительная обработка программы, состоящая из редактирования и вставки в текст программы библиотечной части (п. 12.3.). Затем, в соответствии с режимом печати, определяемым системными командами печати, на АЦПУ выводятся системная программа, фрагмент замен и альфа-программа. Программа, прошедшая предварительную обработку, может быть записана в архив и выведена на перфокарты. Далее происходит трансляция, в ходе которой в программу могут быть внесены отладочные изменения. Модуль загрузки, полученный в результате трансляции, может быть отперфорирован, распечатан, записан в личную библиотеку МС Дубна и загружен (отдельно или совместно с другими модулями). Большинство из перечисленных работ может как входить, так и не входить в задание с программой, что зависит от наличия соответствующих команд в системной программе задания. При отсутствии в задании фрагмента «системная программа» производится трансляция альфа-программы с записью полученного модуля во временную библиотеку МС Дубна.

В задании с данными присутствует либо фрагмент «данные» либо системная команда * ДАННЫЕ с именем альфа-данных. Задание с данными состоит из следующей последовательности работ. Сначала данные редактируются. Затем, в соответствии с режимом печати, определяемым системными командами печати, на АЦПУ выводятся системная программа, фрагмент замен и отредактированные данные. После этого данные могут быть записаны в архив и выведены на перфо-

карты. Далее производится подготовка данных, то есть преобразование данных в машинное представление, используемое операторами ввода при выполнении оттранслированной программы. Подготовленные данные записываются в каналы. При отсутствии в задании системной программы производится подготовка данных с записью в каналы.

Задание с архивами состоит из произвольной совокупности следующих вспомогательных работ с архивами (см. гл. 12): создание архива, уничтожение программы или данных из архива, печать оглавления, изменение имени архива.

Существует общее правило, согласно которому работы, входящие в различные виды заданий, не могут быть заданы в одном задании Например, трансляция программы несовместима с любой работой из задания с данными.

Работы, выполняемые системой Альфа-6, задаются, как отмечалось, системными командами. Любая системная команда начинается символом * и содержит идентификатор системной команды с возможными параметрами, заключаемыми в круглые скобки и перечисляемыми через запятую. Наиболее часто употребляемыми видами параметров являются имена и диапазоны.

Имя — это либо ⟨имя программы⟩, либо ⟨имя данных⟩, либо ⟨имя архива⟩ Синтаксически имя является идентификатором в смысле языка альфа-6 не более чем из шести символов.

Диапазон обозначает часть текста программы или данных и синтаксически является либо просто номером перфокарты (если текст располагается на одной перфокарте), либо двумя номерами перфокарт, разделенными двоеточием (если текст располагается на нескольких последовательных перфокартах). Номер перфокарты — это последовательность не более чем шести символов, среди которых могут быть цифры и десятичная точка, причем десятичная точка не может находиться в начале или в конце последовательности.

Если диапазон обозначает текст всей программы или всех данных, то соответствующий ему параметр в системной команде может быть опущен Такую возможность указывает параметр ⟨пусто или диапазон⟩. Например, команда

```
* ПЕЧАТАТЬ_ПРОГРАММУ (⟨пусто или диапазон⟩)
```

определяет две команды:

```
* ПЕЧАТАТЬ_ПРОГРАММУ (⟨диапазон⟩)
* ПЕЧАТАТЬ_ПРОГРАММУ
```

В последующих главах раздела II альфа-программа или альфа-данные часто рассматриваются как последовательность перфокарт. При этом последовательность номеров перфокарт называется пумерацией программы или данных. Нумерация называется пормальной, если она совпадает с некоторым отрезком натурального ряда.

ГЛАВА 5

ТРАНСЛЯЦИЯ

5.1. Задание на трансляцию

Задать трансляцию некоторой альфа-программы можно тремя способами:

1. Альфа-программа, находящаяся на перфокартах, оформляется как фрагмент «программа» (п. 4.2). Трансляция программы осуществляется при обработке задания, содержащего этот фрагмент.

2. Системная программа задания содержит системную команду

    * ПРОГРАММА (⟨имя альфа-программы⟩, ⟨имя архива⟩).

Трансляция программы осуществляется при обработке задания, содержащего данную системную команду.

3. Системная программа задания содержит системную команду

    * ПРОГРАММА (⟨имя альфа-программы⟩).

При обработке задания, содержащего данную системную команду, производится трансляция программы, заданной на перфокартах в некотором другом задании из данной последовательности заданий.

Если в некотором задании описана альфа-программа и тем не менее необходимо избежать ее трансляции, в системную программу задания нужно вставить команду

    * БЕЗ _ ТРАНСЛЯЦИИ

5.2. Контроль ошибок и выдача сообщений

При трансляции производится полный синтаксический контроль альфа-программы и обнаруживается большинство семантических ошибок. Сообщения об ошибках распечатываются на АЦПУ.

Отдельные сообщения (начинающиеся символом *) являются предупреждающими: они указывают на «подозрительные» места в программе. Например, распечатываются тексты примечаний после служебного слова конец, что служит предупреждением о возможном пропуске символа «;» после конец. При этом распечатку таких примечаний можно отменить, если использовать команду * БЕЗ_ ПРИМЕЧАНИЙ. Отметим, что предупреждающие сообщения не прекращают трансляцию программы.

§ 5.3. Оптимизирующие преобразования при трансляции

Во время трансляции в программе производятся оптимизирующие преобразования. Для целей отладки иногда оказывается полезным знать специфику оптимизирующих преобразований. Отметим наиболее важные из них.

В трансляторе Альфа-6 предусмотрено несколько способов трансляции формальных и фактических параметров процедур. На основе анализа описаний и операторов процедур (указателей функций) для каждого параметра происходит выбор наиболее эффективного способа трансляции. Следует учитывать, что при трансляции описание процедуры или функции помещается в конец того блока, в котором оно описано.

Подвыражения из тела и заголовка цикла, которые при выполнении данного цикла не меняют своего значения, будут вычисляться непосредственно перед входом в цикл.

Подвыражения из некоторой линейной последовательности операторов, которые совпадают (с точностью до коммутативности и ассоциативности), вычисляются только один раз. Это может привести к изменению порядка выполнения операций в оттранслированной программе по отношению к исходной.

Для массивов альфа-программы, описанных в непересекающихся блоках, может отводиться одно и то же место в оперативной памяти.

При оптимизации учитывается структура системы команд ЭВМ БЭСМ-6. Например, параметры циклов и переменные с индексами в теле цикла часто программируются с использованием индекс — регистров. Поэтому значение скаляра — параметра цикла в отведенной для него ячейке памяти в поле скаляров (п. 8.2) может не соответствовать действительному значению этого скаляра.

5.4. Режимы трансляции

Иногда возникает необходимость зарезервировать некоторые индекс-регистры для их использования только в той части альфа-программы, которая записывается в кодах ЭВМ БЭСМ-6 (то есть в операторах *бемш* (п. 2.21)). Такие индекс-регистры задаются системной командой

* РЕГИСТРЫ (⟨список регистров⟩),

где параметром ⟨список регистров⟩ через запятую перечисляются восьмеричные номера индекс-регистров. Регистры с номерами 15, 16, 17 используются транслятором всегда и поэтому не могут быть заняты программистом.

Внешние массивы (*EX*-массивы) модуля загрузки, полученного в результате трансляции, образуют *внешнюю область*, которая размещается во внешней памяти на барабанах, ленте или диске. Длина внешней области определяется суммой длин всех внешних массивов модуля, причем внешние массивы непересекающихся блоков в памяти не совмещаются. Начало внешней области модуля задается командой

* ОБЛАСТЬ (⟨номер носителя⟩, ⟨зона⟩, ⟨сдвиг⟩),

где ⟨номер носителя⟩ задается двумя восьмеричными цифрами, определяющими математический номер направления и номер устройства;

⟨зона⟩ — десятичный номер зоны (тракта);

⟨сдвиг⟩ — десятичное целое, указывающее начальный адрес в зоне (тракте).

При отсутствии команды * ОБЛАСТЬ внешняя область модуля размещается на барабанах.

Пример 1.

Проиллюстрируем характер оптимизирующих преобразований для следующей альфа-программы.

```
Задача:
◇1◇  начало вещественный a1, a2, a3;
     целый i; массив A [1:10];
◇2◇  процедура Ф(x1, x2); значение x2;
     вещественный x2; x1:=x1+x2↑(3/2);
     . . . . . .
◇60◇ Ф(a1, a2+a3);
     . . . . . .
◇71◇ a2:=a1+a2+a3;
◇72◇ для i:=1, ..., 10 цикл
◇73◇ A [i]:=a1+a3;
◇74◇ конец
```

Преобразования текста состоят в следующем:

— описываются новые переменные *b1*, *b2*;

— тело цикла заключается в скобки блока;

— из тела цикла выносится вычисление выражения *a1*+*a3*;

— формальные параметры *x1*, *x2* в теле процедуры *Ф* заменяются фактическими параметрами *a1*, *b1*, где *b1* хранит вычисленное значение выражения *a2*+*a3*;

— тело процедуры программируется и помещается в конец того блока, где описана процедура.

Полученный текст условно можно записать следующим образом:

начало вещественный *b1*, *b2*; Задача:
◇1◇ **начало вещественный** *a1*, *a2*, *a3*; **целый** *i*;
массив *A* [*1*:*10*];
. . . . . .
◇60◇ *b1*: = *a2*+*a3*; *Ф*;
. . . . . .
◇71◇ *b2*: = *a1*+*a3*; *a2*: = *b2*+*a2*;
◇72◇ **для** *i*: = *1*, . . ., *10* **цикл**
◇73◇ **начало** *A* [*i*] : = *b2* **конец**;
◇2◇ **процедура** *Ф*; *a1*: = *a1*+*b1*↑(*3*/*2*);
◇74◇ **конец конец**

В оттранслированной программе заголовок цикла и индексное выражение программируются через индекс-регистры БЭСМ-6, а вычисление вещественной степени — как обращение к стандартной подпрограмме.

Пример 2.

Пусть системная программа задания имеет следующий вид:

* ПРОГРАММА (ПРОБА, АЛЬФА)
* АРХИВ (АЛЬФА, 42)
* ОБЛАСТЬ (53, 9, 0)
* БЕЗ_ ПРИМЕЧАНИЙ
* ПЕЧАТАТЬ_ МОДУЛЬ (2.3:5.1)
* РЕГИСТРЫ (1)
* РЕГИСТРЫ (2, 3)
* ПЕЧАТАТЬ _ МАССИВЫ

Задание определяет трансляцию альфа-программы *ПРОБА* из архива *АЛЬФА*, который находится на ленте с математическим номером 42. Команда * БЕЗ _ ПРИМЕЧАНИЙ блокирует выдачу текстов примечаний, стоящих после служебного слова **конец**. Программа *ПРОБА* содержит операторы *бемш*, где используются индекс-регистры 1, 2, 3. Значение этих регистров не должно изменяться при выполнении других частей оттран-

слированной программы, поэтому они указываются в командах * РЕГИСТРЫ. Внешние массивы модуля будут размещаться на ленте с математическим номером 53, начиная с зоны 9. После трансляции произойдет распечатка части модуля, которая соответствует тексту альфа-программы от перфокарты 2.3 до 5.1 включительно. Если при выполнении программы *ПРОБА* произойдет авост, то сбойная распечатка, будет содержать значения массивов программы *ПРОБА* в момент прерывания (п. 11.3).

Отметим, что распечатка на АЦПУ текста альфа-программы *ПРОБА* не производится (п. 7.1).

ГЛАВА 6

ПОДГОТОВКА ДАННЫХ

Под альфа-данными понимаются параметры альфа-программы, задающие при ее выполнении значения переменных. Язык представления альфа-данных описан в гл. 16.

Подготовить данные для альфа-программы можно тремя способами:

1) альфа-данные находятся на перфокартах и оформляются как фрагмент «данные» (п. 4.2). Подготовка данных осуществляется при обработке задания, содержащего этот фрагмент.

2) Системная программа задания содержит системную команду

* ДАННЫЕ (⟨имя альфа-данных⟩, ⟨имя архива⟩).

Подготовка альфа-данных осуществляется при обработке задания, содержащего эту системную команду.

3) Системная программа задания содержит системную команду

* ДАННЫЕ (⟨имя альфа-данных⟩).

При обработке задания, содержащего эту системную команду, производится подготовка данных, заданных на перфокартах в некотором другом задании из данной последовательности заданий.

Если в некотором задании описаны альфа-данные и по каким то причинам необходимо избежать подготовки данных, то в системную программу задания нужно вставить команду

* БЕЗ _ ТРАНСЛЯЦИИ.

Система производит полный контроль альфа-данных. Синтаксический контроль данных осуществляется при подготовке

данных, а семантический контроль — во время выполнения оттранслированной программы.

При выполнении программы данные считываются из канала оператором ввода (оператор *input*). Носителем нулевого канала является барабан. Поэтому данные, записанные в нулевой канал, сохраняются только на время обработки пакета МС Дубна.

Для каналов с номерами 1—14 носителями являются ленты или диски. Данные, записанные в эти каналы, можно использовать при выполнении альфа-программ в других пакетах МС Дубна без повторной подготовки данных.

Отметим, что информация, записанная в каналы оператором вывода (*output*) при выполнении программы, может быть использована в качестве исходных данных при выполнении другой программы.

Пример использования данных в другом пакете.

Сформируем два пакета.

```
Пакет 1: ШИФР _ 777700 _ ЗС+-
         ЛЕНТА _ 53 (199 — ЗП)-
         ЕЕВ1А3
         * NAME _ ПРОБА
         * ASSIGN _ LIBRARY _ N
         * CALL _ ALPHA/6
         ◇◇◇ ДАННЫЕ
         канал 3 * x = 14; x = 356;
         ◇◇◇ КНЦ
         * END _ FILE
         ⟨диспетчерский конец⟩
         ЕКОНЕЦ
Пакет 2: ШИФР _ 777700 _ ЗС+-
         ЛЕНТА _ 53 (199 — ЗП)-
         ЕЕВ1А3
         * NAME _ ПРОБА
         * ASSIGN _ LIBRARY _ N
         * CALL _ ALPHA/6
         ◇◇◇ ПРОГРАММА
         ПРОБА: начало целое x; coxp(3);
         input (3, x);
       M: если x=1 то стоп;
         если x=(x÷2)×2 то
              x:=x÷2 иначе
              x:=3×x+1; на M;
                  конец
         ◇◇◇ КНЦ
```

```
* EXECUTE
* EXECUTE
* END_FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Пакет 1 переводит альфа-данные в машинное представление ($x=14$; $x=356$), и записывает их в канал 3, находящийся на ленте с математическим номером 53.

Пакет 2 транслирует альфа-программу *ПРОБА* и выполняет оттранслированную программу два раза: сначала при $x=14$, а затем при $x=356$.

ГЛАВА 7

ПЕЧАТЬ И ПЕРФОРАЦИЯ

7.1. Задание печати

Задание печатей производится с помощью системных команд печати. Если системная программа задания не содержит описанных ниже команд печати, то производится распечатка только системной программы и фрагмента замен.

Для распечатки программ и данных (частично или полностью) служат команды:

```
* ПЕЧАТАТЬ_ ПРОГРАММУ (⟨пусто или диапазон⟩)
* ПЕЧАТАТЬ _ ДАННЫЕ (⟨пусто или диапазон⟩)
```

Необходимо учитывать, что при распечатке программы текст библиотечной части этой программы (п. 12.3.) распечатываться не будет. Для распечатки библиотечной части необходимо использовать команду

```
* ПЕЧАТАТЬ_БИБЛИОТЕЧНУЮ _ ЧАСТЬ
```

Иногда требуется отменить все печати, в том числе системной программы и фрагмента замен. В этом случае используется команда

```
* БЕЗ _ ПЕЧАТЕЙ
```

Для отмены печати фрагмента замен используется команда

```
* БЕЗ _ ПЕЧАТИ _ ЗАМЕН
```

Для распечатки модуля загрузки, полученного в результате трансляции, используется команда

```
* ПЕЧАТАТЬ _ МОДУЛЬ (⟨пусто или диапазон⟩)
```

По этой команде распечатывается часть текста модуля, соответствующая исходной альфа-программе и задаваемая параметром ⟨пусто или диапазон⟩. При задании диапазона необходимо учитывать тот факт, что описания процедур при трансляции выносятся в конец того блока, в котором они находятся, причем порядок описаний процедур может измениться.

7.2. Вывод на перфорацию

Для перфорации альфа-программы и альфа-данных используются команды:

```
* ПЕРФОРИРОВАТЬ_ПРОГРАММУ
* ПЕРФОРИРОВАТЬ_ДАННЫЕ
```

На перфокартах текст программы или данных представляется в кодировке УПП. Отметим, что при перфорации программы библиотечная часть этой программы на перфорацию не выдается.

Для перфорации модуля загрузки используется управляющая карта МС Дубна [19]

```
* PUNCH
```

которая помещается в пакет непосредственно перед картой

```
* CALL_ALPHA/6
```

Пример.

Пусть задание содержит следующую системную программу.

```
* ПРОГРАММА (ПРОБА, АЛЬФА)
* АРХИВ (АЛЬФА, 42)
* ПЕЧАТАТЬ_ПРОГРАММУ (63:76)
*ПЕЧАТАТЬ_ПРОГРАММУ (1:40)
* ПЕЧАТАТЬ_БИБЛИОТЕЧНУЮ_ЧАСТЬ
* ПЕРФОРИРОВАТЬ_ПРОГРАММУ
```

На АЦПУ распечатывается системная программа, библиотечная часть программы *ПРОБА*, а также два диапазона программы: перфокарты с 1 по 40 и с 63 по 76. На выходной перфоратор ЭВМ БЭСМ-6 выдается пакет перфокарт, содержащий текст альфа-программы *ПРОБА* без библиотечной части.

ГЛАВА 8
ВЫПОЛНЕНИЕ ПРОГРАММЫ

8.1. Инициация выполнения программы

Пусть альфа-задача содержит задание, в котором производится трансляция некоторой альфа-программы. Для того, чтобы выполнить эту программу, необходимо поместить в пакет после карты конца альфа-задачи (◇◇◇ КНЦ) управляющую карту * EXECUTE. При обработке пакета оттранслированная альфа-программа будет считана из временной библиотеки МС Дубна, загружена в оперативную память и выполнена. Для повторного выполнения программы необходимо поместить в пакет еще одну карту * EXECUTE.

Пример 1.

Рассмотрим пакет примера 6 из гл. 3.

```
ШИФР_777700 _ ЗС+-
ЛЕНТА _42 (199)-
ЕЕВ1А3
* NAME_ПРОБА1
* ASSIGN _ LIBRARY _N
* CALL _ ALPHA/6
◇◇◇ СИСТЕМА
* АРХИВ (АЛЬФА, 42)
* ПРОГРАММА (ПРОБА1, АЛЬФА)
◇◇◇ ЗАДАНИЕ
◇◇◇ ДАННЫЕ
n=10; 11;
◇◇◇ КНЦ
* EXECUTE
* EXECUTE
* END_ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

При обработке пакета программа *ПРОБА* будет выполнена два раза, один раз с данным $n=10$, второй раз с данным $n=11$.

Если альфа-задача содержит задания на трансляцию нескольких альфа-программ, то для выполнения некоторой из этих программ, например, с именем *ПРОБА*, необходимо между картами ◇◇◇ КНЦ и * EXECUTE поместить управляющую карту * MAIN _ ПРОБА.

Если при выполнении программы обнаружена ошибка или произошел авост, то выдается сбойная распечатка, состав которой описан в (п. 11.3).

8.2. Распределение памяти при выполнении программы

Распределение оперативной памяти при выполнении загруженной программы изображено на рис. 8.1.

Длина поля программы вместе с ее динамическими массивами не может превышать 52400 (восьмеричных) ячеек.

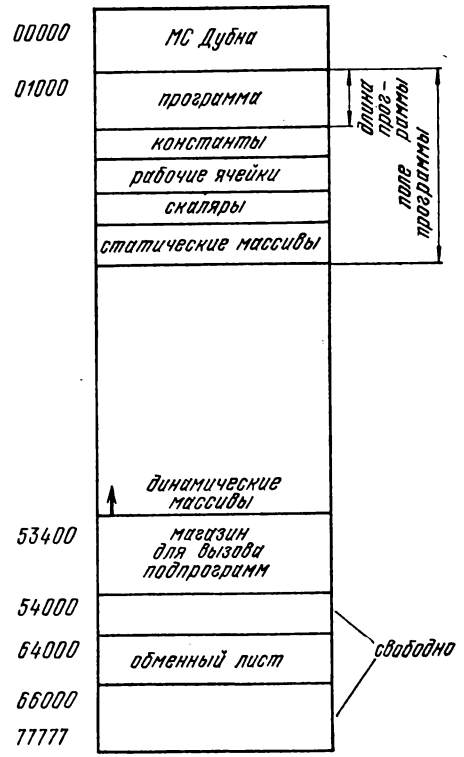


Рис. 8.1. Распределение ОЗУ при выполнении программы.

Для увеличения этого поля (до 72400) необходимо использовать управляющую карту * CALL _ FICMEMORY [13, 19], которая помещается в пакет непосредственно перед картой * EXECUTE.

Начальные значения рабочих ячеек — 0, скаляров и статических массивов $+922337203695_{10} + 19$. Начальное значение динамических массивов не определено.

Отведение памяти под динамические массивы начинается с ячейки 53 377.

Некоторая информация, связанная с выполнением программы, располагается на магнитных барабанах. Свободны для использования операторами обмена (*copy*) барабаны с математическими номерами 15—17.

8.3. Многоканальный ввод/вывод

Программы, оттранслированные системой Альфа-6, могут вводить и выводить информацию одновременно по нескольким каналам (до 16-ти), располагающимся на магнитном барабане (канал 0), дисках или лентах (каналы 1—15). Это дает возможность использовать информацию, выводимую одной программой, в качестве исходных данных для другой.

Пример 2.

```
ШИФР␣ 777700␣ ЗС+⁻
ЛЕНТА␣51 (199 — ЗП)⁻
ЕЕВ1А3
* NAME␣ПРОБА
* ASSIGN␣LIBRARY␣N
* CALL␣ALPHA/6
◇◇◇ ПРОГРАММА
ПРОБА1:
. . . . . .
OUTPUT(1, 'e', a)
. . . . . .
◇◇◇ КНЦ
* EXECUTE
. . . . . .
* CALL␣ALPHA/6
◇◇◇ПРОГРАММА
ПРОБА2:
. . . . . .
coxp (1);
input (1, a);
. . . . . .
◇◇◇ КНЦ
* END␣FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Программа *ПРОБА1* генерирует *n* случайных чисел, которые используются в качестве входных данных в программе

ПРОБА2. После обработки пакета информации на ленте 192 сохраняется и доступна для ввода при выполнении других программ. Математический номер 51 ленты 199 соответствует каналу с номером 1 (п 2.14).

ГЛАВА 9

КОМПЛЕКСАЦИЯ ПРОГРАММ

Пусть комплекс альфа-программ состоит из головной программы *ПРОБА* и двух подпрограмм *ПРОГ1* и *ПРОГ2*. Пакет для трансляции программ *ПРОБА ПРОГ1* и *ПРОГ2*, совместной их загрузки и выполнения комплекса составляется следующим образом.

Пакет 1:

```
ШИФР _ 777700 _ ЗС+-
ЕЕВ1А3
* NAME  ПРОБА
* ASSIGN _ LIBRARY _ N
* CALL _ ALPHA/6
◇◇◇ПРОГРАММА
      ПРОБА:                          ┐
       начало альфа ПРОГ1, ПРОГ2;     │
       . . . . . .                    │ текст
       ПРОГ1 (a, c);                  │ программы
       . . . . . .                    │ ПРОБА
       ПРОГ2 (c);                     │
       . . . . . .                    ┘
конец
◇◇◇ ЗАДАНИЕ
◇◇◇ ПРОГРАММА
(текст программы ПРОГ1)
◇◇◇ ЗАДАНИЕ
◇◇◇ ПРОГРАММА
(текст программы ПРОГ2)
◇◇◇ ЗАДАНИЕ
◇◇◇ ДАННЫЕ
(текст данных)
◇◇◇ КНЦ
* MAIN _ ПРОБА
*EXECUTE
* END _ FILE
```

⟨диспетчерский копец⟩
ЕКОНЕЦ

В этом случае все три программы *ПРОБА*, *ПРОГ1*, *ПРОГ2* будут одновременно загружены в оперативную память и выполнены с данными из последнего задания (см. [19] стр. 37). Отметим, что любая из программ комплекса может быть оттранслирована предварительно и записана в личную библиотеку МС Дубна ([19] стр. 91, [11] стр. 38) и в дальнейшем быть использована при совместной загрузке программ комплекса. Пусть, например, альфа-подпрограмма *ПРОГ1* находится в личной библиотеке с математическим номером 51, находящейся на бобине 199, начиная с зоны 200. Тогда перед картой * CALL␣ALPHA/6 в пакет необходимо: поместить карту * PERSONAL␣LIBRARY:51200, в паспорте пакета описать библиотеку картой

ЛЕНТА␣51 (199)⁻

и из последовательности заданий Альфа-6 убрать задание на трансляцию программы *ПРОГ1*.

Предположим теперь, что суммарная длина всех программ комплекса превышает длину поля программы, т. е. 52400 восьмеричных ячеек в стандартном режиме или 72400 ячеек в режиме FICMEMORY (см. гл. 8). В этом случае возможно организовать динамический вызов подпрограмм в оперативную память в процессе выполнения комплекса.

Например, если длина программы *ПРОБА* равна 10000, *ПРОГ1* — 30000, *ПРОГ2* — 40000, то пакет 1 можно использовать для загрузки и выполнения комплекса, если текст программы *ПРОБА*

альфа *ПРОГ1*, *ПРОГ2*;

заменить на следующий:

процедура *ПРОГ1*(*x*, *y*); **начало** *library* (*'ПРОГ1'*) **конец**;
процедура *ПРОГ2*(*z*); **начало** *library* (*'ПРОГ2'*) **конец**

В этом случае при выполнении комплекса в оперативной памяти постоянно находится лишь программа *ПРОБА*, а программы *ПРОГ1* и *ПРОГ2* располагаются во временной библиотеке МС Дубна и при каждом обращении к ним из программы *ПРОБА* загружаются и вызываются в оперативную память для их выполнения.

Для оптимизации времени обращения к подпрограммам можно использовать режим статической загрузки в МС Дубна ([19] стр. 216). Рассмотрим следующий пакет.

Пакет 2:
ШИФР _ 777700 _ ЗС +⁻
ЛЕНТА _ 60 (199 — ЗП)⁻
ЕЕВ1А3
* NAME_ ПРОБА
* ASSIGN_ LIBRARY _ N
* CALL _ ALPHA/6
◇◇◇ ПРОГРАММА
(текст программы *ПРОБА*)
◇◇◇ ЗАДАНИЕ
◇◇◇ ПРОГРАММА
(текст подпрограммы *ПРОГ1*)
◇◇◇ ЗАДАНИЕ
◇◇◇ ПРОГРАММА
(текст подпрограммы *ПРОГ2*)
◇◇◇ КНЦ
* CALL_ OVERLAY
_*ПРОБА* (*ПРОГ1*, *ПРОГ2*)
* END_ RECORD
* END _ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ

При выполнении пакета 2 на ленте с математическим номером 60 записывается комплекс, состоящий из загруженных программ *ПРОБА*, *ПРОГ1* и *ПРОГ2*.

Для выполнения этого комплекса необходимо собрать следующий пакет.

Пакет 3:
ШИФР _ 777700 _ ЗС +⁻
ЛЕНТА _ 60 (199)⁻
ЕЕВ1А3
* NAME_ ПРОБА
* ASSIGN _ LIBRARY _ N
* CALL _ ALPHA/6
◇◇◇ ДАННЫЕ
(текст данных комплекса)
◇◇◇КНЦ
* CALL _ EXECUTE
* END _ FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ

Сделаем несколько замечаний относительно использования в комплексе фортран-программ.

1) В фортран-программе массивы располагаются в оперативной памяти по столбцам, а в альфа-программе — по строкам. Поэтому, если альфа-программа содержит вызов фортран-подпрограммы с фактическим параметром — массивом, то последний должен быть предварительно транспонирован.

2) Для организации динамического вызова фортран-подпрограммы из альфа-программы вместо процедуры *library* необходимо использовать процедуру *loadfa* (аналог *loadgo*, см. [19] стр. 215), предварительно специфицировав эту процедуру в альфа-программе описателем **фортран**. Например, если подпрограмма *ПРОГ1* записана на языке фортран, то в программе *ПРОБА* текст

процедура *ПРОГ1*(*x*, *y*); **начало** *library* ('*ПРОГ1*') **конец**;

необходимо заменить на следующий текст:

фортран *loadfa*;

и текст

ПРОГ1 (*a*, *c*)

заменить текстом:

loadfa (*a*, *c*, '*ПРОГ1*')

ГЛАВА 10
РЕДАКТИРОВАНИЕ

10.1. Поперфокартное редактирование

По системной команде

* ВСТАВИТЬ (⟨номер⟩, ⟨диапазон⟩)

в текст альфа-программы или данных после перфокарты с номером ⟨номер⟩ вставляется совокупность перфокарт из фрагмента замен, определяемая параметром ⟨диапазон⟩.

По системной команде

* ВЫБРОСИТЬ (⟨диапазон⟩)

из текста программ или данных удаляется совокупность перфокарт, определяемая параметром ⟨диапазон⟩.

По системной команде

* ЗАМЕНИТЬ (⟨диапазон1⟩, ⟨диапазон2⟩)

совокупность перфокарт из текста программы или данных, определяемая параметром ⟨диапазон1⟩, заменяется на совокупность перфокарт из фрагмента замен, определяемую параметром ⟨диапазон2⟩. Если параметры ⟨диапазон1⟩ и ⟨диапазон2⟩

совпадают, то в качестве эквивалента можно использовать команду

* ЗАМЕНИТЬ (⟨диапазон1⟩)

Изменения в программе или данных, задаваемые всеми командами поперфокартного редактирования из одной системной программы, производятся одновременно. Отсюда следует ограничение: части текстов программы или данных, определяемые диапазонами различных команд поперфокартного редактирования из одной системной программы, не должны пересекаться. Например, для программы с нормальной нумерацией (п. 4.4) системная программа не может содержать одновременно следующие пары команд:

А) * ЗАМЕНИТЬ (1:5)
* ЗАМЕНИТЬ (2:10)

Б) * ВЫБРОСИТЬ (1:5)
* ВСТАВИТЬ (5, 5.1)

10.2. Текстовое редактирование

По системной команде

* ЗАМЕНИТЬ_ТЕКСТ (⟨строка1⟩, ⟨строка2⟩, ⟨пусто или диапазон⟩)

в части программы или данных, определяемой третьим параметром, все вхождения текста, задаваемые параметром ⟨строка1⟩ заменяются на текст, задаваемый параметром ⟨строка2⟩. При этом параметры ⟨строка1⟩, ⟨строка2⟩ являются строками в смысле языка альфа-6 (см. гл. 15), не содержащими символа *.

Параметр ⟨строка2⟩ может иметь вид ' '. В этом случае все вхождения текста, задаваемые параметром ⟨строка1⟩, удаляются.

Изменения в программе или данных, задаваемые всеми командами * ЗАМЕНИТЬ_ТЕКСТ из одной системной программы, производятся одновременно. Отсюда следует ограничение: любая часть текста программы или данных, измененная одной из команд * ЗАМЕНИТЬ_ТЕКСТ, не может быть повторно изменена другими командами текстовой замены из той же системной программы.

Некоторые особенности алгоритма текстового редактирования иллюстрируют следующие примеры.

Пример 1. Если системная программа задания содержит команды

* ЗАМЕНИТЬ_ТЕКСТ ('*A* +', '*K*')
* ЗАМЕНИТЬ_ТЕКСТ ('*A*', '*K*'),

то на АЦПУ будет выдано предупреждающее сообщение

* ПЕРЕСЕКАЮТСЯ ЗАМЕНЯЕМЫЕ ТЕКСТЫ,

а результат текстовой замены не определен.

Пример 2. Если системная программа задания содержит команды

* ЗАМЕНИТЬ_ТЕКСТ ('*A*+', '*K*')
* ЗАМЕНИТЬ_ТЕКСТ ('+', ' '),

то любой входящий в программу текст «*A*+» будет изменяться только первой командой.

10.3. Замена идентификаторов

По системной команде

* ЗАМЕНИТЬ_ИДЕНТИФИКАТОР (⟨идент⟩, ⟨строка⟩, ⟨пусто или диапазон⟩)

в части программы или данных, определяемой третьим параметром, все вхождения идентификатора, задаваемые параметром ⟨идент⟩, заменяются на текст, задаваемый параметром ⟨строка⟩.

Для совокупности команд замены идентификаторов в одной системной программе имеет место ограничение, аналогичное ограничению для команд текстовой замены (п. 10.2.).

10.4. Перенумерация

Если задание содержит фрагмент «программа», в котором задана альфа-программа без номеров перфокарт, то на начальном этапе обработки программы ей приписывается нормальная нумерация (п. 4.4.), начинающаяся с единицы и определяемая перфокартами пакета.

По системной команде

* НОМЕР (⟨номер⟩)

отредактированной альфа-программе приписывается нормальная нумерация, начинающаяся с целого десятичного числа, указанного в параметре ⟨номер⟩. Если параметр отсутствует, то нумерация производится, начиная с единицы. Если исходная альфа-программа задается фрагментом «программа», то нумерация определяется перфокартами пакета. Если же альфа-программа находится в архиве, то перенумерация заключается в том, что на место старых номеров перфокарт, содержащихся в тексте программы, вставляются номера нормальной нумерации.

Пример.

Рассмотрим альфа-программу *РИТМ*, которая перед отладкой имела следующий вид.

```
РИТМ:
◇1◇  начало массив A, B [1:4, 1:4];
◇2◇  вещественный p; целый n1, n2;
◇3◇  процедура ЯКОВ (вектор, A, B, p);
◇4◇  значение вектор;
◇5◇  начало целый K, T;
     . . . . . .
◇9◇  если вектор то
◇10◇ для K:=1, ..., n1 цикл
◇11◇ для T: = 1, ..., n1 цикл
◇12◇ A [K, T]:= если K≠T то 1 иначе 0;
     . . . . . .
◇20◇ конец процедуры ЯКОВ;
     . . . . . .
◇30◇ ЯКОВ (вектор, A, B, p);
     . . . . . .
◇70◇ конец
```

В ходе отладки в программу *РИТМ* внесены изменения, которые задаются ниже с помощью системных команд редактирования и фрагмента замен.

Системная программа:

```
* ВСТАВИТЬ (2, 2.1)
* ВСТАВИТЬ (4, 2.1)
* ЗАМЕНИТЬ_ТЕКСТ ('≠', ':=', 12)
* ЗАМЕНИТЬ _ ТЕКСТ ('процедура ЯКОВ (',
'процедура ЯКОВ (n, ')
* ЗАМЕНИТЬ_ИДЕНТИФИКАТОР (n1, 'n', 5:20)
* ЗАМЕНИТЬ (30, 30:30.1)
* ЗАМЕНИТЬ_ТЕКСТ ('n', 'n1', 30)
* НОМЕР
```

Фрагмент замен:

```
◇2.1◇  логический вектор;
◇30◇   ЯКОВ (n, вектор, A, B, p);
◇30.1◇ ЯКОВ (n2, истина, A, B, p);
```

Отредактированная программа имеет следующий вид.

```
РИТМ:
◇1◇  начало массив A, B [1:4, 1:4];
◇2◇  вещественный p; целый n1, n2;
```

```
◇3◇  логический вектор;
◇4◇  процедура ЯКОВ (n, вектор, A, B, p);
◇5◇  значение вектор;
◇6◇  логический вектор;
◇7◇  начало целый K, T;
     . . . . . .
◇11◇ если вектор то
◇12◇ для K:=1, ..., n цикл
◇13◇ для T:=1, ..., n цикл
◇14◇ A [K, T]:= если K=T то 1 иначе 0;
     . . . . . .
◇22◇ конец процедуры ЯКОВ;
     . . . . . .
◇32◇ ЯКОВ (n1, вектор, A, B, p)
◇33◇ ЯКОВ (n2, вектор A, B, p);
     . . . . . .
◇73◇ конец
```

Сделаем два заключительных замечания. Изменения в альфа-программе или в данных, определяемые всеми командами редактирования в системной программе задания, производятся в следующем порядке:

— все поперфокартные редактирования;
— все текстовые замены;
— все замены идентификаторов;
— перенумерация.

Библиотечная часть программы не может быть отредактирована.

ГЛАВА 11
СРЕДСТВА ОТЛАДКИ

11.1. Трассировка программы

По системной команде

* ЗНАЧЕНИЕ (⟨идент⟩, ⟨пусто или диапазон⟩)

при выполнении программы в момент перевычисления переменной, задаваемой параметром ⟨идент⟩, на АЦПУ печатается новое значение этой переменной. Значения печатаются только при выполнении той части программы, которая определяется вторым параметром. Отметим, что переменная, задаваемая параметром ⟨идент⟩, может быть либо простой переменной,

либо переменной с индексами (в том числе с ненулевой внутренней размерностью). Параметр ⟨идент⟩ не может содержать более шести символов. Ограничение: если переменная, задаваемая параметром ⟨идент⟩, является фактическим параметром некоторого оператора процедуры и перевычисляется этим оператором, то при его выполнении печать новых значений не производится. Используя команду * ЗНАЧЕНИЕ можно проследить за изменением переменных при выполнении программы.

По системной команде

* ПЕЧАТАТЬ_МЕТКИ (⟨пусто или диапазон⟩)

при выполнении любого помеченного оператора из части программы, определяемой параметром команды, на АЦПУ печатается идентификатор метки оператора. Используя команду * ПЕЧАТАТЬ_МЕТКИ можно проследить за передачами управления при выполнении программы.

Отметим, что библиотечная часть программы (п. 12.3) не входит в зону действия команд * ЗНАЧЕНИЕ и * ПЕЧАТАТЬ_МЕТКИ, определяемую параметром ⟨пусто или диапазон⟩.

11.2. Контроль за выполнением программы

По системной команде

* ГРАНИЦЫ

при выполнении программы осуществляется контроль за значениями индексных выражений в переменных с индексами. Выполнение программы прекращается, если чтение или запись в массив выводит за область памяти, отведенной под массив в соответствии с его описанием. При этом выдается сообщение о выходе за границы массива (см. гл. 22) и сбойная распечатка. Необходимо помнить, что использование команды * ГРАНИЦЫ существенно увеличивает длину оттранслированной программы и время ее выполнения. Команду удобно использовать в случае «затирания» памяти, отведенной под массивы или программу.

Известна следующая особенность центрального процессора БЭСМ-6. Пусть при выполнении некоторой команды с адресом *А* в программе произошел авост, а в числе следующих нескольких команд (до 8 команд) программы находится команда перехода на адрес *Б*. Тогда в сообщении об ошибке иногда будет указан не адрес *А*, а адрес *Б*.

Системная команда

* ЛОКАЛИЗОВАТЬ_ АВОСТ

позволяет с точностью до 8 команд БЭСМ-6 указать место авоста, который произошел при выполнении оттранслированной программы. Необходимо помнить, что команда * ЛОКАЛИЗОВАТЬ_АВОСТ существенно увеличивает время выполнения программы.

При трансляции альфа-программы осуществляется автоматический контроль соответствия формальных и фактических параметров процедур по виду, типу и структуре. Аналогичный контроль для обращений к альфа-подпрограммам (п. 2.22.) можно произвести с помощью системной команды

* КОНТРОЛИРОВАТЬ_ПАРАМЕТРЫ

Она помещается в заданиях на трансляцию для всех альфа-подпрограмм, из которых комплектуется итоговая программа (см. гл. 9).

11.3. Сбойная распечатка

При обнаружении ошибки во время выполнения альфа-программы может произойти прерывание. В этом случае на АЦПУ выдается следующая информация.

1. Сообщение о характере ошибки, вызвавшей прерывание и адрес команды, на которой произошло прерывание. В случае авоста (диспетчерского прерывания) дополнительно выдается содержимое индекс-регистров.

2. Имя альфа-подпрограммы, при выполнении которой произошло прерывание.

3. Таблица перфокарт, в которой указывается соответствие между номерами перфокарт в исходной альфа-программе и адресами в оттранслированной программе.

4. Таблица скалярных переменных, которая для каждой переменной в исходной альфа-программе содержит ее идентификатор, значение в момент прерывания и адрес ее размещения в ОЗУ.

5. Таблица меток, в которой указывается соответствие между идентификаторами меток в исходной альфа-программе и их адресами в оттранслированной программе.

6. Таблица массивов, которая для каждого массива в исходной альфа-программе содержит его идентификатор и начальный адрес массива в момент прерывания.

7. Таблица общих массивов, в которой для каждого общего объекта (скаляра или массива) исходной альфа-программы

указывается его идентификатор и начальный адрес объекта относительно начала общего массива, в котором он содержится.

Отметим, что сбойная распечатка выдается для альфа-подпрограмм, входящих в выполняемую программу, которые не закончили свою работу в момент прерывания.

Если задание на трансляцию некоторой альфа-подпрограммы содержит системную команду

* БЕЗ_СБОЙНОЙ,

то при прерывании во время выполнения скомплексированной программы информация 2—7 для данной подпрограммы не распечатывается.

Если задание на трансляцию некоторой альфа-подпрограммы содержит системную команду

* ПЕЧАТАТЬ_МАССИВЫ,

то при прерывании во время выполнения скомплексированной программы сбойная распечатка для данной подпрограммы содержит также значения массивов, описанных в этой подпрограмме. Каждое значение печатается в формате '*e*' с указанием адреса памяти, занимаемого значением. Если несколько последовательных значений совпадают, то печатается только первое из них, за которым следует строка звездочек (см. приложение 18).

Пример.

Пусть задание содержит следующую системную программу.

* ПРОГРАММА (ПРОБА, АЛЬФА)
* АРХИВ (АЛЬФА, 42)
* ЗНАЧЕНИЕ (*A*, 10:13)
* ЗНАЧЕНИЕ (*B*, 10:13)
* ЗНАЧЕНИЕ (*C*, 10:13)
* ПЕЧАТАТЬ_МЕТКИ (10:13)
* ПЕЧАТАТЬ_ПРОГРАММУ (10:13)

На АЦПУ будет распечатан приведенный ниже фрагмент программы.

◇10◇ М: **начало целый** *a*, *b*, *c*; *a*=*0*;
◇11◇ **процедура** *Ф* (*a*, *i*); **начало**
◇12◇ *K*:*i*: = *i* + *1*; *a*: = *a* + 2 **конец** *Ф*;
◇13◇ *b*: = *c*: = *1*; *Ф*(*a*, *c*); *a*: = *a* + *b*; **конец**

При выполнении указанного фрагмента программы отладочная информация печатается на АЦПУ в виде следующей строки:

...⁻М А = 0 В = 1 С = 1 ⁻К А = 2 Л = 3 ....

ГЛАВА 12

БИБЛИОТЕКИ

Модули загрузки, полученные после трансляции альфа-программ, хранятся в библиотеках МС Дубна, которые описаны в [13, стр. 97]. Для хранения альфа-программ и альфа-данных используются внутренние библиотеки системы Альфа-6, которые в отличие от библиотек МС Дубна называются архивами и описываются в настоящей главе. Отдельные альфа-программы или альфа-данные, находящиеся в архиве, называются записями.

12.1. Архивы с оглавлением

Архив с оглавлением содержит оглавление с последующими записями. Он размещается на ленте или диске, начиная с некоторой зоны (начальной зоны архива). Зоны архива нумеруются относительно начальной зоны, которая получает номер 0. Оглавление архива находится в зоне 5, копия оглавления — в зоне 4. Зоны с номерами 0—3 в архиве не заняты. Записи располагаются, начиная с зоны с номером 6. Для каждой записи оглавление содержит: имя, номер начальной зоны записи, тип записи (программа или данные), дату записи (год, месяц, число). Запись идентифицируется именем и типом, то есть в одном архиве могут содержаться программа и данные с одним и тем же именем.

Архив с оглавлением определяется в задании с помощью команды

> * АРХИВ (⟨имя⟩, ⟨номер носителя⟩, ⟨начальная зона⟩)

где ⟨имя⟩ — имя архива; ⟨номер носителя⟩ — две восьмеричные цифры (математический номер направления и номер устройства); ⟨начальная зона⟩ — восьмеричный номер начальной зоны архива. Если номер начальной зоны есть 0, то третий параметр может отсутствовать.

Создание архива с оглавлением осуществляется системной командой

> * ЗАПИСАТЬ_ОГЛАВЛЕНИЕ (⟨имя⟩, ⟨длина архива⟩)

где ⟨имя⟩ — имя архива; ⟨длина архива⟩ — восьмеричное число зон, отводимое под архив. Если второй параметр опущен, то под архив будет отведено 300 зон.

Предостережение: после создания архива команда *ЗАПИСАТЬ_ОГЛАВЛЕНИЕ должна быть удалена из пос-

ледовательности заданий, так как повторное ее выполнение приведет к уничтожению всех записей в архиве.

Для записи в архив программы или данных используются соответствующие системные команды:

```
* ЗАПИСАТЬ_ ПРОГРАММУ (<имя программы>,
<имя архива>)
* ЗАПИСАТЬ_ДАННЫЕ (<имя данных>,
<имя архива>)
```

По этим командам программа или данные, определенные в задании, записываются в архив с именем <имя архива> под именем, задаваемым первым параметром. Повторная запись в архив в течение одних суток программы или данных под одним и тем же именем блокируется по системной команде

```
* КОНТРОЛЬ_ДАТЫ
```

С помощью этой команды можно избежать повторного редактирования программы или данных из архива, которое иногда происходит из-за сбоя ЭВМ и повторного пуска пакета.

Для определения программы (данных) из архива в задании с программой (данными) используются системные команды:

```
* ПРОГРАММА (<имя программы>, <имя архива>)
* ДАННЫЕ (<имя данных>, <имя архива>)
```

Для вычеркивания программы или данных из оглавления архива используются системные команды:

```
* УНИЧТОЖИТЬ _ ПРОГРАММУ (<имя программы>,
<имя архива>)
* УНИЧТОЖИТЬ_ДАННЫЕ (<имя данных>,
<имя архива>)
```

По системной команде

```
* ПЕЧАТАТЬ_ОГЛАВЛЕНИЕ (<имя архива>)
```

распечатывается оглавление архива.

По системной команде

```
* ИЗМЕНИТЬ_ИМЯ_АРХИВА (<старое имя>,
<новое имя>)
```

происходит смена имени архива.

12.2. Примитивные архивы

Примитивный архив содержит единственную запись, занимающую несколько последовательных зон на ленте или диске. Примитивный архив определяется в задании с помощью системной команды.

* АРХИВ (⟨имя архива⟩, ⟨адрес архива⟩),

⟨адрес архива⟩ — это последовательность из шести восьмеричных цифр $y_1y_2x_1x_2x_3x_4$, где y_1y_2 — математический номер направления и номер устройства, $x_1x_2x_3x_4$ — номер начальной зоны архива.

Для записи в примитивный архив программы или данных используются команды:

* ЗАПИСАТЬ_ПРОГРАММУ (⟨имя программы⟩, ⟨имя архива⟩)
* ЗАПИСАТЬ_ДАННЫЕ (⟨имя данных⟩, ⟨имя архива⟩)

Отметим, что для записи в примитивный архив не требуется дополнительного задания, создающего этот архив, так как он создается в момент записи в него программы или данных.

Для определения программы (данных) из примитивного архива в задании со схемой (данными) используются, как и в случае архива с оглавлением, системные команды:

* ПРОГРАММА (⟨имя программы⟩, ⟨имя архива⟩)
* ДАННЫЕ (⟨имя данных⟩, ⟨имя архива⟩)

Запись, хранящаяся в архиве с оглавлением, имеет ту же структуру, что и примитивный архив. Поэтому в случае «порчи» оглавления запись из архива с оглавлением может быть использована как запись примитивного архива.

12.3. Библиотечный архив

В состав системы Альфа-6 входит библиотека стандартных процедур, написанных на языке альфа-6, которые можно использовать в любой альфа-программе, не описывая. Для хранения таких библиотечных процедур выделен специальный архив с оглавлением со стандартным именем *LIBRA*, за которым закреплен математический номер 31. В архиве *LIBRA* находятся библиотечные описания, каждое из которых является

последовательностью произвольных описаний в смысле языка альфа-6. В частности, библиотечное описание может быть просто описанием процедуры или процедуры-функции языка альфа-6. Для использования библиотечного описания в альфа-программе необходимо, чтобы задание содержало системную команду

* ПРОЦЕДУРА (⟨имя библиотечного описания⟩)

При наличии такой команды в начало текста альфа-программы вставляется текст соответствующего библиотечного описания. Все вставленные в программу библиотечные описания называются библиотечной частью этой программы.

Библиотечное описание может также храниться в любом архиве. Для использования такого библиотечного описания в программе необходимо употребить системную команду

* ПРОЦЕДУРА (⟨имя библиотечного описания⟩,
⟨имя архива⟩)

С помощью одной команды * ПРОЦЕДУРА можно задавать несколько библиотечных описаний, имена которых перечисляются через запятую в числе параметров команды.

Отметим, что программа записывается в архив вместе со своей библиотечной частью.

Пример 1.

Пусть задание содержит следующую системную программу.

```
* АРХИВ (СИГМА, 520140)
* ПРОГРАММА (ПРОБА, СИГМА)
* ПРОЦЕДУРА (АРЕС, FIND)
* ПРОЦЕДУРА (ГАУСС)
* ЗАПИСАТЬ_ПРОГРАММУ (ПРОБА1, АЛЬФА)
* АРХИВ (АЛЬФА, 42)
* ЗАМЕНИТЬ_ИДЕНТИФИКАТОР (РК, '200')
* ПЕЧАТАТЬ_ПРОГРАММУ.
```

В задании определена альфа-программа *ПРОБА*, хранящаяся в примитивном архиве *СИГМА*, который размещается на ленте с математическим номером 52, начиная с зоны 140. Всюду в тексте программы *ПРОБА*, за исключением библиотечной части, идентификатор *РК* заменяется числом 200. Библиотечная часть отредактированной программы состоит из описаний библиотечных процедур *АРЕС*, *FIND* и *ГАУСС*, хранящихся в архиве *LIBRA*. Отредактированная программа вместе с библиотечной частью записывается под именем *ПРОБА1* в архив с оглавлением *АЛЬФА*, после чего производится транс-

ляция этой программы. Распечатка на АЦПУ будет содержать текст отредактированной программы без библиотечной части.

Пример 2.

Пусть задание состоит из следующей системной программы:

```
* ЗАПИСАТЬ_ОГЛАВЛЕНИЕ (ПОЛЮС)
* АРХИВ (ПОЛЮС, 42, 200)
* УНИЧТОЖИТЬ_ПРОГРАММУ (ЦИРК, АЛЬФА)
* УНИЧТОЖИТЬ_ДАННЫЕ (ЦИРК, АЛЬФА)
* АРХИВ (АЛЬФА, 51)
* ИЗМЕНИТЬ_ИМЯ_АРХИВА (АЛЬФА, СИГМА)
* ПЕЧАТАТЬ_ОГЛАВЛЕНИЕ (СИГМА)
* АРХИВ (СИГМА, 51).
```

На ленте с математическим номером 52 создается архив *ПОЛЮС*, под который отводятся зоны с 200 по 477; оглавление будет записано в зоны 204 и 205. Из оглавления архива *АЛЬФА* вычеркивается информация о хранящихся в архиве программе и данных с одинаковым именем *ЦИРК*. Далее архив *АЛЬФА* получает новое имя *СИГМА*, его оглавление распечатывается на АЦПУ. Отметим, что вычеркнутые из оглавления программа и данные *ЦИРК* могут быть прочитаны через примитивный архив, если после их вычеркивания не было записей в архив *СИГМА*.

Пример 3.

Для записи в архив *LIBRA* библиотечного описания *INVER*, подготовленного на перфокартах, можно использовать следующий пакет:

```
ШИФР_777777 ЗС+-
ЛЕНТА_42 (9999 — ЗП)-
EEB1A3
* NAME_INVER
* ASSIGN_LIBRARY_N
* CALL_ALPHA/6
◇◇◇СИСТЕМА
* ЗАПИСАТЬ_ПРОГРАММУ (INVER, LIBRA)
* АРХИВ (LIBRA, 42)
* БЕЗ_ТРАНСЛЯЦИИ
* НОМЕР (10000)
◇◇◇ПРОГРАММА
inver:
◇1◇  начало процедуры INVER (E, A, n);
◇2◇ начало . . . . . .
   . . . . . .
```

```
◊33◊ конец inver; конец
◊◊◊ КНЦ
* END_FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

Текст библиотечного описания должен завершаться символом точка с запятой. При записи в архив *LIBRA* рекомендуется нумеровать текст четырех- или пятизначными номерами перфокарт, используя для этого команду * НОМЕР. Отметим, что запись библиотечного описания в личный архив производится аналогично.

РАЗДЕЛ III
ЯЗЫКИ СИСТЕМЫ

Для описания синтаксиса языков системы мы будем использовать формализм, близкий к принятому в [3] (называемый БНФ). Отличия нашего формализма от БНФ состоят в следующем.

1) Отождествление металингвистических переменных, имена которых различаются только в падеже и числе.

2) Фиксация значения металингвистических переменных.

Пусть ⟨x⟩ — некоторая металингвистическая переменная, значение которой образуется соединением двух одинаковых идентификаторов. Очевидно, что в этом случае применение формулы

⟨x⟩::=⟨идентификатор⟩ ⟨идентификатор⟩

не достигает цели, поскольку по такой формуле допустимыми значениями ⟨x⟩ будут также слова, являющиеся соединением двух любых идентификаторов. Для указания того, что металингвистическая переменная, входящая несколько раз в металингвистическую формулу, принимает одно и то же значение для разных вхождений, используются двойные угловые скобки ⟨⟨ ⟩⟩, т. е. если имеется некоторая переменная ⟨a⟩, то употребление в правой части некоторой металингвистической формулы обозначения ⟨⟨a⟩⟩ означает, что переменная ⟨a⟩ во всех отмеченных таким образом ее вхождениях в правую часть принимает любое, но одно и то же значение. При таком условии вышеуказанная переменная ⟨x⟩ будет задаваться формулой

⟨x⟩::=⟨⟨идентификатор⟩⟩ ⟨⟨идентификатор⟩⟩.

В то же время формула

⟨y⟩::=⟨⟨идентификатор⟩⟩ ⟨идентификатор⟩
⟨⟨идентификатор⟩⟩

будет задавать переменную ⟨y⟩, значением которой является

соединение трех идентификаторов, первый и последний из которых совпадают.

3) Операция подстановки. Операция подстановки определяется для трех металингвистических переменных ⟨x⟩, ⟨y⟩ и ⟨p⟩. Записывается операция подстановки в виде:

подстановка ⟨p⟩ в ⟨y⟩ на место ⟨x⟩.

Результат операции подстановки определяется следующим образом. Пусть P, Y и X — значения переменных ⟨p⟩, ⟨y⟩ и ⟨x⟩ соответственно. В слове Y слева направо выделяются все непересекающиеся вхождения слова X*), т. е. слово Y представляется в виде

$$A_1XA_2X\ldots XA_k,$$

где A_1, A_2, ... A_k — какие-то слова, возможно, пустые. Тогда результатом подстановки будет:

$$A_1PA_2P\ldots PA_k.$$

В описании языков полужирный шрифт используется для выделения независимых основных символов (см. пп. 15.2.2 и 15.2.3). Подразумевается, что эти основные символы не имеют никакого отношения к отдельным буквам и цифрам, из которых они составлены.

В описании языков системы Альфа-6 используется русский вариант служебных слов, однако языки допускают как русское, так и английское представление служебных слов (см. Приложение 3).

ГЛАВА 13

КОМПЛЕКТАЦИЯ СИСТЕМНОГО ПАКЕТА

Информация для работы системы задается с помощью колоды перфокарт (см. рис. 13.1), которая называется системным пакетом. Металингвистические переменные ⟨КОНЕЦ ПАС-

---

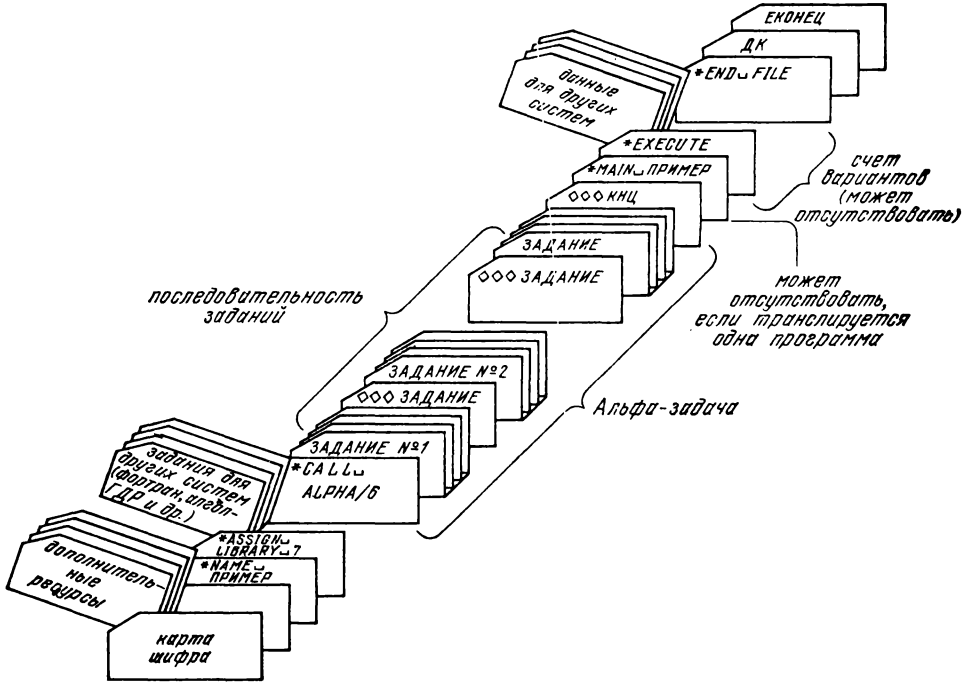
*) В отношении выражения «все вхождения» необходимо сделать некоторые пояснения. Пусть X есть индексное выражение, Y и P суть выражения с соответствующими значениями:

$P = a + b$,

$Y = c[iks, i]$,

$X = i$.

В этом случае нельзя выделять первое вхождение i в Y, поскольку оно не является значением индексного выражения.


ЕКОНЕЦ
ДК
*END FILE
данные для других систем
*EXECUTE
*MAIN ПРИМЕР
◇◇◇ КНЦ
ЗАДАНИЕ
◇◇◇ ЗАДАНИЕ
ЗАДАНИЕ №2
◇◇◇ ЗАДАНИЕ
ЗАДАНИЕ №1
*CALL ALPHA/6
задания для других систем (фортран, алгол- ГДР и др.)
*ASSIGN LIBRARY 7
*NAME ПРИМЕР
дополнительные ресурсы
карта шифра
счет вариантов (может отсутствовать)
может отсутствовать, если транслируется одна программа
последовательность заданий
Альфа-задача

ПОРТА⟩, ⟨ДК⟩ и ⟨ЕКОНЕЦ⟩ обозначают вспомогательные карты Альфа-6, изображенные в приложении 9.

В описании синтаксиса комплектации системного пакета используются основные символы языка альфа-6 (см. гл. 15), а также символ «⁻» (надчеркивание).

Синтаксис.

⟨системный пакет⟩::=⟨паспорт⟩ ⟨тело пакета⟩ ⟨конец пакета⟩

⟨паспорт⟩::=⟨карта шифра⟩ ⟨карты конца паспорта⟩|⟨карта шифра⟩ ⟨дополнительные ресурсы⟩ ⟨карты конца паспорта⟩

⟨карта шифра⟩::= *шифр* _ ⟨шесть цифр шифра⟩ _ *зс*+⁻| *шифр* _ ⟨двенадцать цифр шифра⟩ _ *зс*+⁻

⟨шесть цифр шифра⟩::= ⟨десят цифра⟩ ⟨десят цифра⟩ ⟨десят цифра⟩ ⟨десят цифра⟩ ⟨десят цифра⟩ ⟨десят цифра⟩

⟨десят цифра⟩::= 1|2|3|4|5|6|7|8|9|0

⟨двенадцать цифр шифра⟩::=⟨шесть цифр шифра⟩ ⟨шесть цифр шифра⟩

⟨дополнительные ресурсы⟩::=⟨ресурс⟩ | ⟨ресурс⟩ ⟨дополнительные ресурсы⟩

⟨ресурс⟩::=⟨заказ времени⟩|⟨заказ трактов⟩|⟨заказ ленты⟩|⟨заказ метров АЦПУ⟩|⟨заказ перфорирующего устройства⟩|⟨раздел ТЕЛЕ⟩

⟨заказ времени⟩::= *время* _ ⟨ччммсс⟩⁻

⟨ччммсс⟩::=⟨целое без знака⟩

⟨заказ трактов⟩::=*тракты* _ ⟨количество трактов⟩⁻

⟨количество трактов⟩::=⟨целое без знака⟩

⟨заказ ленты⟩::=лента _ ⟨мф⟩ (⟨бобина⟩)⁻| *лента* _ ⟨мф⟩ (⟨бобина⟩ — *зп*)⁻

⟨бобина⟩::=⟨целое без знака⟩|⟨рабочая область на диске⟩

⟨рабочая область на диске⟩::=⟨число кусков по 32 зоны⟩ *с*

⟨число кусков по 32 зоны⟩::=1|2|3|4|5|6|7

⟨мф⟩::=⟨номер направления⟩ ⟨номер магнитофона⟩

⟨номер направления⟩::=3|4|5|6

⟨номер магнитофона⟩::=⟨восьм цифра⟩

⟨восьм цифра⟩::=0|1|2|3|4|5|6|7

⟨заказ метров АЦПУ⟩::= *ацпу* _ ⟨количество метров⟩⁻

⟨количество метров⟩::=⟨целое без знака⟩

⟨заказ перфорирующего устройства⟩::= *вывод*⁻

⟨раздел теле⟩::= *теле*⁻

⟨карты конца паспорта⟩::=⟨КОНЕЦ ПАСПОРТА⟩ ⟨карта NAME⟩ ⟨карта ASSIGN LIBRARY⟩

⟨карта NAME⟩::=* *name* ⟨последовательность от 1 до 18 основных символов языка⟩

⟨карта ASSIGN LIBRARY⟩::=* *assign* ⎵ *library* ⎵ ⟨номер библиотеки⟩

⟨номер библиотеки⟩::=⟨восьм цифра⟩|⟨мф⟩

⟨конец пакета⟩::=⟨карта END FILE⟩ ⟨ДК⟩ ⟨ЕКОНЕЦ⟩

⟨карта END FILE⟩::=* *end* ⎵ *file*

⟨тело пакета⟩::=⟨задание для систем⟩ ⟨задание на счет⟩| ⟨тело пакета⟩ ⟨задание для систем⟩ ⟨задание на счет⟩

⟨задание на счет⟩::=⟨пусто⟩ | ⟨счет вариантов⟩ | ⟨карта MAIN⟩ ⟨счет вариантов⟩

⟨карта MAIN⟩::=* *main* ⎵ ⟨имя главной программы⟩

⟨счет вариантов⟩::=⟨карта EXECUTE⟩|⟨карта EXECUTE⟩ ⟨данные для других систем⟩ | ⟨счет вариантов⟩ ⟨счет вариантов⟩

⟨карта EXECUTE⟩::=* *execute*

⟨задание для систем⟩::=⟨альфа-задача⟩ | ⟨задание для других систем⟩ | ⟨задание для систем⟩ ⟨задание для систем⟩

⟨альфа-задача⟩::=⟨карта вызова системы Альфа-6⟩ ⟨последовательность заданий⟩ ⟨карта конца альфа-задачи⟩

⟨карта вызова системы Альфа-6⟩::=* *call* ⎵ *alpha/6*

⟨карта конца альфа-задачи⟩::=◇◇◇ *кнц*

⟨последовательность заданий⟩::=⟨задание⟩|⟨задание⟩ ⟨карта задание⟩ ⟨последовательность заданий⟩

⟨карта задание⟩::=◇◇◇ *задание*

⟨задание⟩::=⟨фрагмент системная программа⟩ ⟨фрагмент программа⟩ ⟨фрагмент замен⟩|⟨фрагмент системная программа⟩ ⟨фрагмент данные⟩ ⟨фрагмент замен⟩

⟨фрагмент системная программа⟩::=⟨карта система⟩ ⟨системная программа⟩|⟨пусто⟩

⟨карта система⟩::=◇◇◇ *система*

⟨фрагмент программа⟩::=⟨карта программа⟩ ⟨подпрограмма⟩|⟨карта программа⟩ ⟨карта алгибр⟩ ⟨алгибр-программа⟩|⟨пусто⟩

⟨карта программа⟩::=◇◇◇ *программа*

⟨карта алгибр⟩::=◇◇◇ *алгибр*

⟨фрагмент данные⟩::=⟨карта данные⟩ ⟨альфа-данные⟩|⟨карта данные⟩ ⟨карта алгибр⟩ ⟨алгибр-данные⟩|⟨пусто⟩

⟨карта данные⟩::=◇◇◇ *данные*

⟨фрагмент замен⟩::=⟨карта замены⟩ ⟨массив изменений⟩| ⟨пусто⟩

⟨карта замены⟩::=◇◇◇ *замены*

⟨номер перфокарты⟩::= ⟨целое без знака⟩|⟨целое без знака⟩, ⟨целое без знака⟩

⟨вхождение номера перфокарты в текст⟩::= ◇ ⟨номер перфокарты⟩ ◇

Семантика. Семантика карты шифра и дополнительных ресурсов дается в [17] и в Приложении 9. Дополнительные ресурсы необходимы, когда работа системы не обеспечивается ресурсами, задаваемыми стандартным паспортом (нужна лента архива, не хватает метров АЦПУ и т. д.).

Рабочая область на диске — это область памяти на рабочем диске ОС Диспак, ее состояние сохраняется только на время выполнения системного пакета. Рабочие области удобно использовать для многоканального ввода/вывода (2.14).

Перфокарты: карта шифра, дополнительные ресурсы и все вспомогательные карты (КОНЕЦ ПАСПОРТА, ДК, ЕКОНЕЦ) перфорируются на УПП (Приложение 9).

На карте NAME могут быть пробиты любые (от 1 до 18) символы языка, среди которых хотя бы один должен быть отличен от пробела. Эта последовательность символов распечатывается на АЦПУ крупными символами в начале листинга данной задачи.

Карта ASSIGN _ LIBRARY заказывает общую библиотеку подпрограмм Альфа-6, в которой находятся стандартные подпрограммы, необходимые для вызова и работы системы, а также для загрузки и выполнения альфа-программ.

Карта MAIN задает имя главной подпрограммы, т. е. подпрограммы, с которой начинается загрузка задачи в оперативную память для ее выполнения. Если системный пакет задает трансляцию одной подпрограммы, то эта карта может отсутствовать.

Перфокарты: NAME, ASSIGN _ LIBRARY, MAIN, END _ FILE, EXECUTE — являются управляющими картами МС Дубна, правила их перфорации описаны в [11], [13].

В системном пакете кроме альфа-задачи могут находиться «задания для других систем» и «данные для других систем», работающих в МС Дубна: Фортран, Алгол-ГДР, Мадлен, Бемш. Правила комплектации системного пакета для этих систем описываются в соответствующей литературе ([14], [12], [16], [15]).

Если альфа-задача содержит несколько заданий, то они выполняются в порядке их вхождения в пакет. Соседние задания разделяются «картой задание».

В одном задании могут быть следующие фрагменты: фрагмент системная программа, фрагмент программа, фрагмент

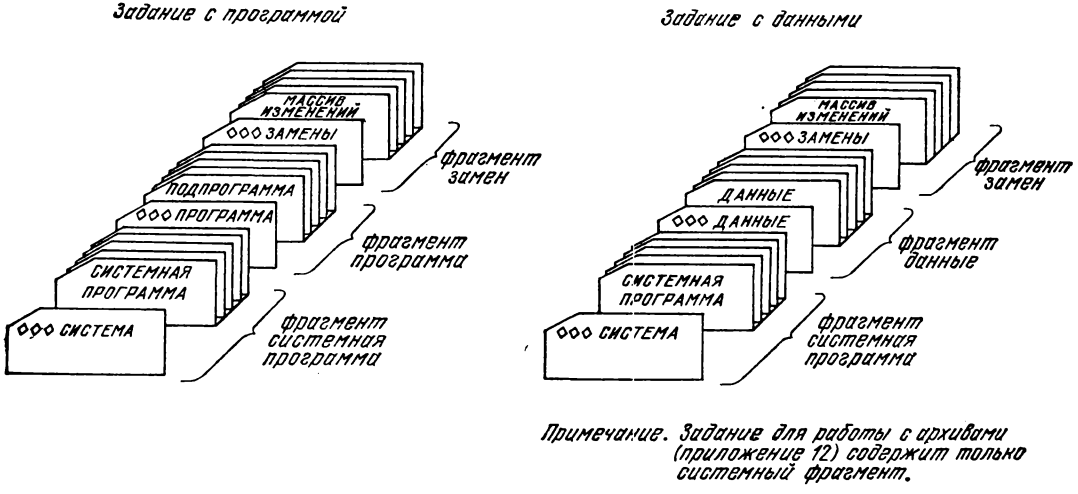


Рис. 13.2. Структура одного задания.

данные, фрагмент замен. Фрагмент каждого типа может входить в задание не более одного раза. В одном задании не могут присутствовать одновременно фрагмент программа и фрагмент данные, не могут быть пустыми одновременно фрагмент системная программа и фрагмент программа, а также фрагмент системная программа и фрагмент данные. Порядок вхождения фрагментов в задание не существен. Структура одного такого задания схематично изображена на рис. 13.2.

Фрагмент системная программа начинается «картой система» и состоит из последовательности системных команд, синтаксис и семантика которых описываются в гл. 14.

Фрагмент программа начинается «картой программа» и состоит из подпрограммы (п. 15.4.1), написанной на языке альфа-6 и структурированной номерами перфокарт. Номера перфокарт введены для удобства отладки. Они могут входить в текст подпрограммы или альфа-данных между любыми двумя символами языка. Обычно номерами перфокарт нумеруются части текста, расположенные на одной физической перфокарте. Однако отметим, что подпрограмма (данные) может не содержать ни одного номера перфокарты.

Фрагмент данные начинается «картой данные» и состоит из альфа-данных, структурированных номерами перфокарт. Синтаксис и семантика альфа-данных приведены в гл. 16.

Во фрагменте программа и во фрагменте данные могут быть программа (алгибр-программа) и данные (алгибр-данные) для системы Алгибр (см. Приложение 16).

В одном задании должна быть задана работа с одной программой или с одними данными. Программа (данные) берется либо с перфокарт, либо из архива, либо из другого задания.

Фрагмент замен начинается «картой замены» и содержит перфокарты (массив изменений), которые вставляются в подпрограмму или альфа-данные при помощи системных команд редактирования * ВСТАВИТЬ и * ЗАМЕНИТЬ. Он отсутствует в задании тогда и только тогда, когда фрагмент системная программа таких команд не содержит.

Перфокарты: карта система, карта задание, карта программа, карта данные, карта замены, карта алгибр — это стандартные карты Альфа-6. Они могут быть отперфорированы на одном из допустимых перфорирующих устройств (Приложение 1) в виде, задаваемом синтаксисом. Кроме того, существует макеты стандартных карт, вид и правила использования которых описаны в Приложении 10.

ГЛАВА 14

ЯЗЫК СИСТЕМНЫХ КОМАНД

В главе описываются синтаксис и семантика системных команд.

Металингвистические переменные, начинающиеся со слова «имя», задают идентификаторы в смысле алгола-60 не более чем из шести символов.

14.1. Системная программа

Синтаксис.

⟨системная программа⟩::=⟨имя системной программы⟩:⟨последовательность системных команд⟩

⟨последовательность системных команд⟩::=⟨системная команда⟩|⟨системная команда⟩ ⟨последовательность системных команд⟩

⟨системная команда⟩::=⟨задание программы или данных⟩|⟨задание архива⟩|⟨вывод программы или данных⟩|⟨команда печати⟩|⟨команда редактирования⟩|⟨команда архивных работ⟩|⟨команда режима⟩|⟨команда спецификации⟩|⟨отладочная команда⟩

⟨имя системной программы⟩::=⟨идентификатор⟩

Семантика. Системная программа задает совокупность работ, которые должна выполнить система при обработке задания (4.2). Каждая системная команда задает либо отдельную работу, либо режимы работ, либо информацию для работы.

14.2. Задание программы и данных

Синтаксис.

⟨задание программы или данных⟩::=⟨задание программы или данных из пакета⟩|⟨задание программы или данных из архива⟩

⟨задание программы или данных из пакета⟩::=*программа(⟨имя программы⟩)|*данные(⟨имя данных⟩)

⟨задание программы или данных из архива⟩::=*программа(⟨имя программы⟩, ⟨имя архива⟩)|*данные(⟨имя данных⟩, ⟨имя архива⟩)

⟨имя данных⟩::=⟨идентификатор⟩

⟨имя программы⟩::=⟨идентификатор⟩

⟨имя архива⟩::=⟨идентификатор⟩

⟨задание архива⟩::=⟨задание архива с оглавлением⟩|⟨задание примитивного архива⟩

⟨задание архива с оглавлением⟩::=* архив (⟨имя архива⟩, ⟨адрес архива⟩)
⟨адрес архива⟩::=⟨номер носителя⟩|⟨номер носителя⟩, ⟨начальная зона архива⟩
⟨задание примитивного архива⟩::=* архив (⟨имя архива⟩, ⟨номер носителя⟩, ⟨начальная зона примитивного архива⟩)
⟨номер носителя⟩::=⟨номер направления⟩ ⟨номер магнитофона⟩
⟨номер направления⟩:: = ⟨восьм цифра⟩
⟨номер магнитофона⟩:: = ⟨восьм цифра⟩
⟨начальная зона примитивного архива⟩:: = ⟨восьм цифра⟩ ⟨восьм цифра⟩ ⟨восьм цифра⟩ ⟨восьм цифра⟩
⟨начальная зона архива⟩:: = ⟨восьм целое⟩
⟨восьм целое⟩:: = ⟨восьм цифра⟩ | ⟨восьм целое⟩ ⟨восьм цифра⟩
⟨восьм цифра⟩:: = 0|1|2|3|4|5|6|7

Семантика. Команда * ПРОГРАММА (⟨имя программы⟩) или * ДАННЫЕ (⟨имя данных⟩) определяет в задании программу (или данные), которая находится на перфокартах в другом задании из выполняемой последовательности заданий. Параметр команды есть имя фрагмента, находящееся в начале текста программы (данных).

Команда, задающая программу или данные из архива, определяет программу (или данные), хранящуюся в архиве. Имя архива указано вторым параметром команды. Имя программы (или данных) — первый параметр.

Архив, имя которого указано в командах * ПРОГРАММА или * ДАННЫЕ, должен быть задан с помощью команды * АРХИВ. Команда задания архива с оглавлением указывает математический номер МЛ (МД) и номер начальной зоны архива (п. 12.1). Если третий параметр команды отсутствует, то номер начальной зоны архива=0. Значение номера начальной зоны архива должно быть $\leqslant$1770.

Команда задания примитивного архива задает математический номер МЛ (или МД) и номер начальной зоны примитивного архива (п. 12.2). Значение номера начальной зоны примитивного архива должно быть $\leqslant$ 1777.

14.3. Вывод программы или данных

Синтаксис.

⟨вывод программы или данных⟩::=⟨запись программы или данных⟩ | ⟨перфорация программы или данных⟩

⟨запись программы или данных⟩::=* записать _ программу (⟨имя программы⟩, ⟨имя архива⟩) | * записать _ данные (⟨имя данных⟩, ⟨имя архива⟩)

⟨перфорация программы или данных⟩::=* перфорировать _ программу | * перфорировать _ данные.

Семантика. Команды записи программы или данных задают запись программы или данных в архив с именем, указанным вторым параметром. В первом параметре задается имя, под которым эта программа (данные) будет записана в архив. Архив, в который производится запись, нужно задавать в системной программе с помощью команды * АРХИВ.

Команды перфорации программы или данных задают выдачу на перфокарты программы или данных.

Вывод программы (или данных) происходит после редактирования и вставки в программу библиотечной части (п. 12.3).

14.4. Печать

Синтаксис.

⟨команда печати⟩::=⟨печать программы или данных⟩ | ⟨печать библиотечной части⟩ | ⟨частичная печать⟩ | ⟨отмена печати⟩ | ⟨печать модуля⟩

⟨печать программы или данных⟩::=* печатать _ программу | * печатать _ данные

⟨печать библиотечной части⟩::=* печатать _ библиотечную _ часть

⟨частичная печать⟩::=* печатать _ программу (⟨диапазон⟩) | * печатать _ данные (⟨диапазон⟩)

⟨диапазон⟩::=⟨начало диапазона⟩ : ⟨конец диапазона⟩ | ⟨номер перфокарты⟩

⟨начало диапазона⟩::=⟨номер перфокарты⟩

⟨конец диапазона⟩::=⟨номер перфокарты⟩

⟨номер перфокарты⟩::=⟨целое без знака⟩ | ⟨целое без знака⟩.⟨целое без знака⟩

⟨отмена печати⟩::=* без _ печатей | * без _ печати _ замен

⟨печать модуля⟩::=* печатать _ модуль | * печатать _ модуль (⟨диапазон⟩).

Семантика. Команда печати программы (или данных) задает распечатку на АЦПУ текста программы (данных) после редактирования. При отсутствии команды программа (данные) не печатаются. Отметим, что по команде * ПЕЧАТАТЬ _ ПРОГРАММУ текст библиотечной части распечатываться не будет. Для распечатки библиотечной части следует использовать

команду *ПЕЧАТАТЬ _ БИБЛИОТЕЧНУЮ _ ЧАСТЬ. Команда частичной печати программы (данных) задает распечатку текста указанного диапазона программы (данных). Таких команд может быть несколько.

Если системная программа не содержит ни одной печати, то произойдет распечатка только системной программы и фрагмента замен. Для отмены печати фрагмента замен используется команда * БЕЗ _ ПЕЧАТИ _ ЗАМЕН. Для отмены печати системной программы и фрагмента замен используется команда * БЕЗ _ ПЕЧАТЕЙ.

Команда * ПЕЧАТАТЬ _ МОДУЛЬ задает распечатку модуля загрузки на языке мадлен, полученного в результате трансляции программы. Если указывается диапазон печати, то происходит распечатка части модуля, соответствующая указанному диапазону программы. При задании диапазона необходимо учитывать, что тела процедур при трансляции выносятся в конец того блока, в котором описаны эти процедуры. При этом порядок следования процедур может измениться. В системной программе может быть несколько команд * ПЕЧАТАТЬ _ МОДУЛЬ (⟨диапазон⟩).

14.5. Редактирование

Синтаксис.

⟨команда редактирования⟩::=⟨команда поперфокартного редактирования⟩ | ⟨команда замены текста⟩ | ⟨команда замены идентификатора⟩ | ⟨команда перенумерации⟩

⟨команда поперфокартного редактирования⟩::=* вставить (⟨номер перфокарты⟩, ⟨диапазон⟩) | * выбросить (⟨диапазон⟩) | * заменить (⟨диапазон⟩, ⟨диапазон⟩) | * заменить (⟨диапазон⟩)

⟨команда замены текста⟩::=* заменить _ текст (⟨строка⟩, ⟨строка⟩, ⟨диапазон⟩) | * заменить _ текст (⟨строка⟩, ⟨строка⟩)

⟨команда замены идентификатора⟩::=* заменить _ идентификатор (⟨идентификатор⟩, ⟨строка⟩, ⟨диапазон⟩) | * заменить _ идентификатор (⟨идентификатор⟩, ⟨строка⟩)

⟨команда перенумерации⟩::=* номер | * номер (⟨целое без знака⟩)

Семантика. По команде * ВСТАВИТЬ в текст программы (или данных) после перфокарты с номером ⟨номер перфокарты⟩ вставляется совокупность перфокарт из фрагмента замен, задаваемая параметром ⟨диапазон⟩.

По команде * ВЫБРОСИТЬ из текста программы (или данных) удаляется совокупность перфокарт, определяемая параметром ⟨диапазон⟩.

Первый параметр команды * ЗАМЕНИТЬ указывает диапазон, удаляемый из текста программы. На место этого диапазона помещается диапазон из фрагмента замен, определенный во втором параметре команды. Если второй параметр в команде * ЗАМЕНИТЬ отсутствует, то совокупность перфокарт из фрагмента замен указывается первым параметром.

Имеется следующее ограничение на совокупность команд поперфокартного редактирования из одной системной программы: части текстов программы (данных), определяемые диапазонами различных команд поперфокартного редактирования, не должны пересекаться (п. 10.1).

По команде * ЗАМЕНИТЬ _ ТЕКСТ все вхождения текста, указанного первым параметром, заменяются на текст, указанный вторым параметром, причем в той части программы (данных), которая определяется третьим параметром. При отсутствии третьего параметра замена производится во всей программе (данных). Второй параметр может иметь вид ''. Это означает удаление вхождений текста, задаваемого первым параметром.

Параметр ⟨строка⟩ определен в (п. 15.2.6), он не может содержать символа *.

Имеется ограничение на всю совокупность команд замены текста из одной системной программы: любая часть программы (данных), изменяемая одной командой * ЗАМЕНИТЬ _ ТЕКСТ, не может быть повторно изменена другими командами замены текста (п. 10.2).

Действие команды * ЗАМЕНИТЬ _ ИДЕНТИФИКАТОР определяется аналогично действию команды * ЗАМЕНИТЬ _ ТЕКСТ. Отличие в том, что производится замена в тексте программы (данных) идентификатора, задаваемого первым параметром. Команда применима только к тем идентификаторам, которые состоят не более чем из шести символов.

По системной команде * НОМЕР производится замена номеров перфокарт в тексте программы (данных) на новые номера. Новая нумерация является нормальной (п. 4.4) и начинается с целого десятичного числа, указанного параметром команды * НОМЕР; при отсутствии параметра нумерация начинается с единицы. Если исходная программа задается с перфокарт (в виде фрагмента программа), то нумерация по команде * НОМЕР производится по физическим перфокартам. Если же программа задается из архива, то на место старых

номеров перфокарт помещаются новые номера с нормальной нумерацией.

Изменения в программе (данных), определяемые всеми командами редактирования из одной системной программы, осуществляются в следующем порядке: сначала выполняются все команды поперфокартного редактирования; к полученному тексту применяются все команды замены текста; в полученном тексте производится замена идентификаторов; после этого производится перенумерация.

Текст библиотечной части программы не может быть отредактирован.

14.6. Вспомогательные работы с архивами

Синтаксис.

⟨команда архивных работ⟩::=⟨создание архива⟩ | ⟨печать оглавления⟩ | ⟨уничтожение программы или данных⟩ | ⟨изменение имени архива⟩

⟨создание архива⟩::=* записать _ оглавление (⟨имя архива⟩, ⟨длина архива⟩) | * записать _ оглавление (⟨имя архива⟩)

⟨печать оглавления⟩::=* печатать _ оглавление (⟨имя архива⟩)

⟨уничтожение программы или данных⟩::=* уничтожить _ программу (⟨имя программы⟩, ⟨имя архива⟩) | * уничтожить _ данные (⟨имя данных⟩, ⟨имя архива⟩)

⟨изменение имени архива⟩::=* изменить _ имя _ архива (⟨имя старое⟩, ⟨имя новое⟩)

⟨длина архива⟩::=⟨восьм целое⟩

⟨имя новое⟩::=⟨идентификатор⟩

⟨имя старое⟩::=⟨идентификатор⟩

Семантика. Архив, имя которого указано хотя бы в одной из перечисленных команд, должен быть определен в системной программе командой * АРХИВ (п. 14.2).

С помощью команды * ЗАПИСАТЬ _ОГЛАВЛЕНИЕ на МЛ (МД), указанной в команде * АРХИВ, создается архив с оглавлением. Второй параметр команды * ЗАПИСАТЬ _ ОГЛАВЛЕНИЕ задает число зон на МЛ (МД), которое отводится архиву. При отсутствии второго параметра под архив отводится 300 зон (в восьмеричной системе счисления).

По команде * ПЕЧАТАТЬ _ОГЛАВЛЕНИЕ производится распечатка на АЦПУ оглавления архива, указанного параметром команды.

По команде уничтожения программы или данных из оглавления архива, имя которого указано вторым параметром, вычеркивается программа (или данные) с именем, указанным первым параметром.

По команде * ИЗМЕНИТЬ_ИМЯ _АРХИВА происходит изменение имени архива. Первый параметр указывает имя, под которым архив определен в команде * АРХИВ. Второй параметр указывает новое имя, которое присваивается архиву.

14.7. Режимы

Синтаксис.

⟨команда режима⟩::=⟨запрет комментариев⟩ | ⟨отмена трансляции⟩ | ⟨останов при ошибке⟩ | ⟨контроль даты записи⟩ | ⟨отмена сбойной распечатки⟩ | ⟨отмена константных действий⟩

⟨запрет комментариев⟩::=* без_ примечаний

⟨отмена трансляции⟩::=* без_трансляции

⟨останов при ошибке⟩::=* останов_при_ошибке

⟨контроль даты записи⟩::=* контроль_даты

⟨отмена сбойной распечатки⟩::=* без_сбойной

⟨отмена константных действий⟩::=* без_константных_действий

Семантика. Команды режима предписывают системе перейти на новые режимы работ, отличные от стандартных.

По команде * БЕЗ_ПРИМЕЧАНИЙ блокируется выдача на АЦПУ предупреждающих сообщений о комментариях после служебного слова конец.

По команде * БЕЗ_ПРИМЕЧАНИЙ блокируется трансляция программы в задании с программой (п. 5.1), либо подготовка данных в задании с данными (гл. 6).

Если задание содержит команду * ОСТАНОВ_ ПРИ_ ОШИБКЕ и при выполнении предыдущих заданий альфа-задачи были обнаружены ошибки, то отменяется выполнение задания, содержащего эту команду. Отменяется также и выполнение последующих за ним заданий в альфа-задаче.

Команда * КОНТРОЛЬ_ДАТЫ отменяет запись в архив программы (данных) в случае, когда архив уже содержит программу (данные) с этим именем и дата записи совпадает с датой выполнения данного задания.

Команда * БЕЗ_СБОЙНОЙ в задании с программой отменяет выдачу на АЦПУ сбойной распечатки в случае прерывания во время выполнения этой программы. Если программа выполняется в комплексе с другими программами (является

подпрограммой), то в случае прерывания при выполнении скомплексированной программы сбойная распечатка не будет содержать информацию об этой подпрограмме (п. 11.3).

Команда * БЕЗ_КОНСТАНТНЫХ_ДЕЙСТВИЙ блокирует оптимизирующие преобразования в программе при трансляции, связанные с выполнением операций с константами в выражениях.

14.8. Спецификации

Синтаксис.

⟨команда спецификации⟩::=⟨описание регистров⟩ | ⟨задание библиотечных описаний⟩ | ⟨задание *EX*-области⟩

⟨описание регистров⟩::=* регистры (⟨список регистров⟩)

⟨список регистров⟩::=⟨номер регистра⟩ | ⟨список регистров⟩, ⟨номер регистра⟩

⟨номер регистра⟩::=⟨восьм цифра⟩ | ⟨двоичная цифра⟩ ⟨восьм цифра⟩

⟨задание библиотечных описаний⟩::=* процедура (⟨список библиотечных описаний⟩) | * процедура (⟨список библиотечных описаний⟩, ⟨имя архива⟩)

⟨список библиотечных описаний⟩::=⟨имя библиотечного описания⟩ | ⟨список библиотечных описаний⟩, ⟨имя библиотечного описания⟩

⟨задание *EX*-области⟩::=* область (⟨номер носителя⟩, ⟨зона⟩, ⟨сдвиг в зоне⟩)

⟨номер носителя⟩::=⟨номер направления⟩ ⟨номер магнитофона⟩

⟨номер направления⟩::=⟨восьм цифра⟩

⟨номер магнитофона⟩::=⟨восьм цифра⟩

⟨зона⟩::=⟨целое без знака⟩

⟨сдвиг в зоне⟩::=⟨целое без знака⟩

Семантика. Команда * РЕГИСТРЫ указывает системе номера тех регистров, которые математик запрещает ей использовать в оттранслированной программе, помимо тех мест, где математик сам их использует (в операторе *беми*). В команде * РЕГИСТРЫ нельзя указывать регистры с номерами 15, 16, 17, поскольку они всегда используются системой в оттранслированной программе.

Команда * ПРОЦЕДУРА перечисляет имена тех библиотечных описаний, которые используются в программе, и должны быть включены в ее текст. Перечисленные библиотечные описания должны храниться под указанными именами в биб-

лиотечном архиве *LIBRA*. Если последним параметром команды является имя архива (определенного в задании командой * АРХИВ), то библиотечные описания будут взяты из этого архива, а не из архива *LIBRA*.

Команда * ОБЛАСТЬ задает начало внешней области программы (п. 5.4) на МЛ, МБ или МД. В этой области располагаются *EX*-массивы в порядке следования их описаний в программе. Первый параметр команды задает математический номер носителя. Второй параметр указывает десятичный номер зоны (тракта). Третий параметр указывает адрес ячейки относительно начала зоны (тракта), начиная с которой размещается внешняя область.

Если в системной программе нет команды * ОБЛАСТЬ, то система располагает *EX*-массивы — в стандартном фиксированном месте на МБ. (п. 8.2).

14.9. Средства отладки

Синтаксис.

⟨отладочная команда⟩:: = ⟨печать меток⟩ | ⟨печать значения идентификатора⟩ | ⟨контроль границ⟩ | ⟨локализация авоста⟩ | ⟨контроль параметров⟩ | ⟨печать массивов⟩

⟨печать меток⟩ :: = * печатать_метки | * печатать_метки (⟨диапазон⟩)

⟨печать значения идентификатора⟩:: = * значение (⟨идентификатор⟩) | * значение (⟨идентификатор⟩, ⟨диапазон⟩)

⟨контроль границ⟩:: = * границы

⟨локализация авоста⟩:: = * локализовать_авост

⟨контроль параметров⟩:: = * контролировать_параметры

⟨печать массивов⟩:: = * печатать_массивы

Семантика. Команда * ПЕЧАТАТЬ_МЕТКИ задает печать на АЦПУ идентификаторов меток при выполнении помеченного оператора. Если в команде задан ⟨диапазон⟩, то распечатываются только те метки, которые описаны в пределах данного диапазона в программе.

По команде * ПЕЧАТАТЬ_ЗНАЧЕНИЕ при выполнении программы в момент присваивания переменной (заданной первым параметром команды) производится печать на АЦПУ нового значения этой переменной. Если в команде указан ⟨диапазон⟩, то печать значений происходит только при выполнении той части программы, которая задается диапазоном. Если нет параметра ⟨диапазон⟩, то печать значения происходит при любом присваивании переменной.

Переменная, задаваемая в первом параметре, может быть либо простой, либо переменной с индексами. Идентификатор этой переменной не может содержать более шести символов.

Ограничение: если переменная, указанная в команде * ЗНАЧЕНИЕ, является фактическим параметром процедуры, то при выполнении оператора процедуры печать значений не производится.

Печать меток и значений определяется для отредактированной программы, но перед ее перенумерацией, задаваемой командой * НОМЕР.

Библиотечная часть недоступна действию команд печати меток и значений.

По команде * ГРАНИЦЫ при выполнении программы осуществляется контроль значений индексных выражений в переменных с индексами. Выполнение программы прекращается, если значение одного из индексов выходит за верхнюю или нижнюю границу индекса, указанную в описании массива (п. 11.2).

Команда * ЛОКАЛИЗОВАТЬ_АВОСТ задает такой режим выполнения программы, при котором в случае авоста гарантируется выдача адреса места авоста с точностью до 8 команд БЭСМ-6 (п. 11.2).

По команде * КОНТРОЛИРОВАТЬ_ПАРАМЕТРЫ при выполнении программы, скомплексированной из подпрограмм (гл. 9), производится контроль соответствия формальных и фактических параметров альфа-подпрограмм по виду, типу и структуре. Команда * КОНТРОЛИРОВАТЬ_ПАРАМЕТРЫ должна быть помещена в задании на трансляцию всех альфа-подпрограмм, из которых комплексируется итоговая программа.

По команде * ПЕЧАТАТЬ_МАССИВЫ при выполнении альфа-программы в случае прерывания сбойная распечатка дополнительно содержит значения массивов, описанных в этой программе. Если команда * ПЕЧАТАТЬ_МАССИВЫ содержится в задании на трансляцию альфа-подпрограммы, то при прерывании во время выполнения скомплексированной программы, сбойная распечатка дополнительно содержит значения массивов, описанных в этой подпрограмме.

ГЛАВА 15

ВХОДНОЙ ЯЗЫК АЛЬФА-6

За основу входного языка системы Альфа-6 взят Входной Язык, формальное описание которого содержится в работе [8]. После реализации в ВЦ СО АН СССР на ЭВМ типа М-20 этот

язык получил название альфа [4]. Язык альфа — это расширение языка алгол-60, обладающее большим набором выразительных средств. Например, в языке альфа допускаются комплексные величины, многомерные переменные и действия над ними, описания начальных значений и функции-выражения.

Основные отличия языка альфа-6 от языка альфа следующие:

1. Синтаксические конструкции, совпадающие с языком алгол-60, изменены в соответствии с пересмотренным сообщением [3].

2. Введены средства ввода/вывода и обмена языка алгамс вместо соответствующих средств языка альфа.

3. Расширен список стандартных функций с включением в него стандартных функций языка алгамс.

4. Введены средства записи команд БЭСМ-6 с мнемоникой автокода бемш.

15.1. Структура языка

Назначением алгоритмического языка является описание вычислительных процессов. Основной концепцией, используемой для описания правил вычислений, является хорошо известное понятие арифметического выражения, содержащего в качестве составных частей числа, переменные и функции. Из таких выражений путем применений правил арифметической композиции образуются самостоятельные единицы языка — явные формулы, называемые операторами присваивания.

Для того, чтобы указать ход вычислительного процесса, добавляются некоторые неарифметические и условные операторы, которые могут, например, описывать альтернативы или циклические повторения вычислительных операторов. Ввиду того, что для функционирования этих операторов возникает необходимость их взаимосвязи, операторы могут снабжаться метками. Чтобы образовать составной оператор, последовательность операторов можно заключить в операторные скобки **начало** и **конец**.

Операторы дополняются описаниями, которые сами по себе не являются предписаниями о вычислениях, но информируют транслятор о существовании и некоторых свойствах объектов, фигурирующих в операторах. Этими свойствами могут быть, например, класс чисел, используемых в качестве значений переменных, размерность массива чисел или даже совокупность правил, определяющих некоторую функцию. Последова-

тельность описаний и следующая за ней последовательность операторов, заключенные между **начало** и **конец**, составляют блок. Каждое описание должно располагаться в начале блока и действительно только для этого блока.

Подпрограмма — это помеченный блок, помеченный составной оператор или описание процедуры (процедуры-функции). Все идентификаторы в подпрограмме должны быть описаны.

Всякий раз, когда утверждается, что точность арифметических действий, вообще говоря, не указана, или когда результат некоторого процесса остается или объявляется неопределенным, это должно пониматься так, что программа станет полностью определять некоторый вычислительный процесс только тогда, когда дополнительная информация укажет как подразумеваемые точность и вид арифметических действий, так и последовательность выполняемых действий для всех случаев, которые могут встретиться в процессе вычислений.

15.2. Основные символы, идентификаторы, числа и строки

Язык альфа-6 строится из следующих основных символов:

⟨основной символ⟩::=⟨буква⟩ | ⟨цифра⟩ | ⟨логическое значение⟩ | ⟨ограничитель⟩

15.2.1. Буквы.

⟨буква⟩::=⟨русская буква⟩ | ⟨латинская буква⟩ | ⟨греческая буква⟩

⟨русская буква⟩:: = *а*|*б*|*в*|*г*|*д*|*е*|*ж*|*з*|*и*|*й*|*к*|*л*|*м*|*н*|*о*|*п*|*р*|*с*|*т*|*у*|*ф*|*х*|*ц*|*ч*|*ш*|*щ*|*ь*|*ы*|*э*|*ю*|*я*|*А*|*Б*|*В*|*Г*|*Д*|*Е*|*Ж*|*З*|*И*|*Й*|*К*|*Л*|*М*|*Н*|*О*|*П*|*Р*|*С*|*Т*|*У*|*Ф*|*Х*|*Ц*|*Ч*|*Ш*|*Щ*|*Ь*|*Ы*|*Э*|*Ю*|*Я*

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

⟨греческая буква⟩::=α|β|γ|δ|ε|ζ|η|ϑ|ϰ|λ|μ|ν|ξ|ο|π|ρ|σ|τ|φ|χ|ψ|ω|A|B|Γ|Δ|E|Z|H|Θ|K|Л|M|N|Ξ|O|П|P|Σ|T|Φ|X|Ψ|Ω

Совпадающие по написанию буквы отождествляются.

Буквы не имеют индивидуального смысла. Они используются для образования идентификаторов и строк (пп. 15.2.4, 15.2.6).

15.2.2. Цифры, логические значения.

Цифры

⟨цифра⟩::=0|1|2|3|4|5|6|7|8|9

Цифры используются для образования чисел, идентификаторов и строк.

Логические значения.

⟨логическое значение⟩::=**истина** | **ложь**

Логические значения имеют фиксированный очевидный смысл.

15.2.3. Ограничители.

⟨ограничитель⟩::=⟨знак операции⟩ | ⟨разделитель⟩ | ⟨скобка⟩ | ⟨описатель⟩ | ⟨спецификатор⟩

⟨знак операции⟩::=⟨знак арифметической операции⟩ | ⟨знак операции отношения⟩ | ⟨знак логической операции⟩ | ⟨знак операции следования⟩

⟨знак арифметической операции⟩::=+|−|×|/|÷|↑.

⟨знак операции отношения⟩::=<|≤|=|>|≥|≠

⟨знак логической операции⟩::=≡|⊃|∨|∧|¬|⊕

⟨знак операции следования⟩::=**на**|**если**|**то**|**иначе**|**для**|**цикл**|**стоп**

⟨разделитель⟩::=,|.|$_{10}$|:|;|:=| _ |**шаг**|**до**|**пока**|**примечание**|**раз**|*i*|=|...|*

⟨скобка⟩::=(|)|[|]|'|'|**начало**|**конец**

⟨описатель⟩::=**собственный**|**логический**|**упакованный**|**целый**|**вещественный**|**массив**|**переключатель**|**процедура**|**комплексный**|**функция**

⟨спецификатор⟩::=**строка**|**метка**|**значение**|**результат**

Ограничители имеют фиксированный смысл, который в большинстве случаев очевиден, а в остальных случаях будет пояснен в соответствующем месте.

Такие типографские особенности, как пробел и переход на новую строку, в языке не принимаются во внимание. Однако для облегчения чтения их можно свободно использовать.

Для возможности включения текста между символами программы имеют место следующие правила для примечаний.

Последовательность основных символов:	Эквивалент
; **примечание** ⟨любая последовательность, не содержащая символа«;»⟩ ;	;
начало примечание ⟨любая последовательность, не содержащая символа«;»⟩ ;	**начало**
конец ⟨любая последовательность, не содержащая **ни конец, ни«;», ни иначе**⟩ ■	**конец** ■

Знак ■ обозначает один из символов: **конец**, ; или **иначе**.

Эквивалентность здесь означает, что любая из трех конструкций, указанных в левой колонке, если она встречается вне некоторой строки, может заменяться соответствующим ей символом, указанным в правой колонке; эта замена не оказывает никакого влияния на выполнение программы. При этом считается, что конструкцию примечания, встретившуюся ранее при чтении текста слева направо, следует заменять прежде, нежели более поздние конструкции, содержащиеся в этой последовательности.

15.2.4. Идентификаторы.

Синтаксис.

⟨идентификатор⟩::=⟨буква⟩ | ⟨идентификатор⟩ ⟨буква⟩ | ⟨идентификатор⟩ ⟨цифра⟩

Примеры.

q
soup
r17a
a34kTMNS
MARILIN

Семантика. Идентификаторы не имеют неизменно присущего им смысла, а служат для обозначения простых переменных, массивов, меток, переключателей функций и процедур. Они могут выбираться произвольно (см. однако п. 15.3.2, стандартные функции).

Один и тот же идентификатор нельзя использовать для обозначения двух различных величин, за исключением того случая, когда эти величины, согласно описаниям программы, имеют несовместимые области действия (п.п. 15.2.7, 15.5).

15.2.5. Числа.

Синтаксис.

⟨целое без знака⟩::=⟨цифра⟩ | ⟨целое без знака⟩ ⟨цифра⟩

⟨целое⟩::=⟨целое без знака⟩ | +⟨целое без знака⟩ | −⟨целое без знака⟩

⟨правильная дробь⟩::=.⟨целое без знака⟩

⟨порядок⟩::=$_{10}$⟨целое⟩

⟨десятичное число⟩::=⟨целое без знака⟩ | ⟨правильная дробь⟩ | ⟨целое без знака⟩ ⟨правильная дробь⟩

⟨число без знака⟩::=⟨десятичное число⟩ | ⟨порядок⟩ | ⟨десятичное число⟩ ⟨порядок⟩

⟨число⟩::=⟨число без знака⟩ | + ⟨число без знака⟩ | −⟨число без знака⟩

Примеры.

0	-200.084	$-083_{10}-02$
177	$+07.43_{10}8$	$_{10}-7$
.5384	$9.34_{10}+10$	$_{10}-4$
$+0.7300$	$2_{10}-4$	$+_{10}+5$

Семантика. Десятичные числа имеют свой обычный смысл. Порядок — это масштабный множитель, выраженный как целая степень десяти.

Типы. Целые числа имеют тип **целый**. Все остальные числа имеют тип **вещественный**, (см. п. 15.5.1).

15.2.6. Строки.

Синтаксис.

⟨чистая строка⟩::=⟨любая последовательность основных символов, не содержащая ни ', ни '⟩ | ⟨пусто⟩
⟨открытая строка⟩::=⟨чистая строка⟩ | '⟨открытая строка⟩'| ⟨открытая строка⟩ ⟨открытая строка⟩
⟨строка⟩::='⟨открытая строка⟩'

Примеры.

'$5k_{\sqcup}$ — '[[['$\Lambda = /:'Tt$"
'$.This_{\sqcup} is_{\sqcup} a_{\sqcup}$'$string$"

Семантика. Для обозначения строк в языке используются кавычки (' и '). Это позволяет работать с произвольными последовательностями основных символов языка. Символ $_{\sqcup}$ обозначает пробел. Вне строки он не имеет значения.

Строки используются в качестве фактических параметров процедур (п.п. 15.3.2, 15.5.7).

15.2.7. Величины, классы и области действия.

Различаются следующие классы величин: простые переменные, массивы, метки, переключатели, функции и процедуры.

Область действия величины — это совокупность операторов и выражений, внутри которой имеет силу описание идентификатора, связанного с этой величиной. Вопросы, касающиеся меток, см. в п. 15.4.1.

15.2.8. Значения, структуры и типы.

Значение — это некоторое упорядоченное множество чисел (частный случай: отдельное число), некоторое упорядоченное множество логических значений (частный случай: отдельное логическое значение) или некоторая метка. Числа или логические значения соответствующего упорядоченного множества

называются (скалярными) компонентами этого значения. Структура значения характеризуется прежде всего размерностью значения, задаваемой некоторым натуральным числом m.

Простейшей структурой обладают значения размерности ноль, называемые скалярными.

Если значение имеет положительную размерность m, то это означает, что значение имеет m упорядоченных измерений. Каждое измерение $i (i = 1, 2, \ldots, m)$ характеризуется положительным числом t_i, называемым порядком значения по данному измерению.

m-мерное значение с порядками по измерениям $t_1, t_2, \ldots, t_m$ содержит $t_1 \times t_2 \times \ldots \times t_m$ скалярных компонент. Предполагается, что эти компоненты образуют m-мерную таблицу со «сторонами», содержащими соответственно по $t_1, t_2, \ldots, t_m$ членов. Понятие многомерного значения является обобщением понятия прямоугольной матрицы.

Размерность m m-мерного значения будет иногда называться абсолютной размерностью, а его скалярные компоненты — абсолютными компонентами. Одномерные значения называются также векторами, двумерные — матрицами, а значения любой положительной размерности — массивами.

В некоторых случаях есть смысл говорить не только о скалярных компонентах какого-либо массива, но и о компонентах более высокой размерности. Например, трехмерный массив можно рассматривать как матрицу векторов или как вектор матриц. Такие компоненты массива, имеющие положительную размерность, будут называться относительными компонентами, а размерность массива по отношению к этим компонентам — относительной размерностью.

О некоторых синтаксических единицах говорится, что они принимают значения. Вообще говоря, во время выполнения программы эти значения изменяются. Значения выражений и их составных частей определяются в (п. 15.3). Значение идентификатора массива есть упорядоченное множество значений соответствующего массива переменных с индексами (п. 15.3.1, индексы).

Различные типы (**целый**, **вещественный**, **логический** и **комплексный**) обозначают свойства значений. Типы, связанные с синтаксическими единицами, относятся к значениям этих единиц.

Правила построения синтаксических единиц обеспечивают то, что все скалярные компоненты их значения имеют один и тот же тип. Этот же тип присваивается по определению всему значению в целом.

15.3. Выражения

В языке альфа-6 первичными составными частями программ, описывающих алгоритмические процессы, являются арифметические, логические и именующие выражения. Составными частями этих выражений, помимо некоторых ограничителей, являются логические значения, числа, переменные, указатели функций, элементарные арифметические и логические операции, а также некоторые операции отношения и следования. Поскольку синтаксическое определение как переменных, так и функций содержит выражения, определение выражений и их составных частей по необходимости является рекурсивным.

⟨выражение⟩::=⟨арифметическое выражение⟩ | ⟨логическое выражение⟩ | ⟨именующее выражение⟩

15.3.1. Переменные.

Синтаксис.

⟨идентификатор переменной⟩::=⟨идентификатор⟩
⟨простая переменная⟩::=⟨идентификатор переменной⟩
⟨индексное выражение⟩::=⟨арифметическое выражение⟩|⟨пусто⟩
⟨список индексов⟩::=⟨индексное выражение⟩ | ⟨список индексов⟩, ⟨индексное выражение⟩
⟨идентификатор массива⟩::=⟨идентификатор⟩
⟨внешний идентификатор⟩::=*EX*|⟨внешний идентификатор⟩⟨буква⟩|⟨внешний идентификатор⟩⟨цифра⟩
⟨внешний идентификатор массива⟩::=⟨внешний идентификатор⟩
⟨внешняя переменная с индексами⟩::=⟨внешний идентификатор массива⟩ [⟨список индексов⟩]
⟨переменная с индексами⟩::=⟨идентификатор массива⟩ [⟨список индексов⟩]
⟨верхний индекс⟩::=⟨индексное выражение⟩
⟨переменная с верхним индексом⟩::=⟨идентификатор переменной⟩↑[⟨верхний индекс⟩]|⟨идентификатор массива⟩↑[⟨верхний индекс⟩] [⟨список индексов⟩]
⟨перечисление переменных⟩::=
подстановка ⟨арифметического выражения⟩ в ⟨⟨переменную⟩⟩ на место ⟨⟨индексного выражения⟩⟩, ..., подстановка ⟨арифметического выражения⟩ в ⟨⟨переменную⟩⟩ на место ⟨⟨индексного выражения⟩⟩
⟨элемент списка переменных⟩::=⟨переменная⟩|⟨перечисление переменных⟩

⟨список переменных⟩::=⟨элемент списка переменных⟩|⟨список переменных⟩, ⟨элемент списка переменных⟩
⟨сформированная переменная⟩::=|⟨список переменных⟩|
⟨скомпонованная переменная⟩::=|[⟨целое без знака⟩] ⟨список переменных⟩|
⟨переменная⟩::=⟨простая переменная⟩|⟨переменная с индексами⟩|⟨переменная с верхним индексом⟩|⟨сформированная переменная⟩|⟨скомпонованная переменная⟩

Примеры.

epsilon
dela
a17
EXA1[*i*, *j*]
Q [7, 2]
x [*sin*(*n* × *pi*/2), *Q* [3, *n*, 4]]
Q [, 2]
Q [,]
x↑ [0] [*l*, *m*]
y↑[*i* + 2]
|*x*, *y*, *z*, *a*, *b*, *c*|
|*m* [10], ... , *m* [10 + 2 × *V*]|
||*a* [1, 1], ... , *a* [1, *n*]|, ... , |*a* [*n*, 1], ... , *a* [*n*, *n*]||
|[2] *x* [1], ... , *x* [20]|
|[2]|[1] *A*, *B*|, |[1] *C*, *D*||

Семантика. Переменная — это наименование, данное некоторому значению. Это значение может использоваться в выражениях для образования других значений и может произвольным образом изменяться посредством операторов присваивания (п. 15.4.2).

Тип значения данной переменной определяется описанием самой переменной (п. 15.5.1) или соответствующего идентификатора массива (п. 15.5.2). Все переменные из списка переменных, образующих сформированную или скомпонованную переменную, должны быть одного типа и этот тип присваивается сформированной или скомпонованной переменной.

Массивы, обозначенные внешними идентификаторами, располагаются во внешней памяти БЭСМ-6. Внешняя переменная с индексами указывает элемент внешнего массива, как начальный адрес обмена для процедуры *copy*. Доступ к элементам внешнего массива возможен только через оператор *copy*, в связи с чем употребление внешней переменной с индексами возможно только в качестве фактического параметра процедуры *copy*, (15.4.13, семантика).

Семантика процедуры *copy* запрещает употребление пустых позиций во внешней переменной с индексами.

Индексы. Переменные с индексами именуют значения, которые являются относительными компонентами многомерных массивов (п. 15.5.2). Каждое арифметическое выражение из списка индексов занимает одну индексную позицию переменной с индексами и называется индексом. Полный список индексов заключается в индексные скобки [ ]. Какая именно компонента массива упоминается с помощью переменной с индексами, определяется по фактическому числовому значению ее индексов (п. 15.3.3).

Каждая непустая индексная позиция воспринимается как скалярная переменная типа целый, и вычисление индекса понимается как присваивание значения этой фиктивной переменной (п. 15.4.2, типы и структуры). Значение переменной с индексом определено только в том случае, когда значение индексного выражения находится в пределах границ индексов массива (п. 15.5.2).

Переменной с непустыми позициями индексов приписывается внешняя размерность, упорядоченная совокупность внешних измерений и порядки по внешним измерениям. Внешняя размерность равна числу позиций индекса. Внешние измерения упорядочиваются по порядку соответствующих позиций индексов. Порядок по некоторому внешнему измерению равен количеству возможных различных значений индексного выражения в соответствующей позиции индекса. Чтобы не смешивать внешнюю размерность с размерностью, введенной в п. 15.2.8, последняя будет иногда называться внутренней размерностью.

Пусть $I\,[E_1, E_2, \ldots, E_n]$ — переменная внутренней размерности m с n непустыми позициями индексов и пусть порядки этой переменной по внутренним измерениям равны $t_1, t_2, \ldots, t_m$, а по внешним измерениям равны $S_1, S_2, \ldots, S_n$. Тогда переменная с этим же идентификатором и с r пустыми позициями индексов, имеющими номера $l_1, l_2, \ldots, l_r$, будет изображать массив с абсолютной внутренней размерностью, равной $r+m$. Порядками этого массива по внутренним измерениям будут числа $S_{l_1}, S_{l_2}, \ldots, S_{l_r}, t_1, t_2, \ldots, t_m$. Переменную $I\,[E_1, E_2, \ldots, E_n]$ можно рассматривать как обозначение m-мерных компонент этого массива. Относительная размерность массива по отношению к компонентам будет равна r.

Примеры.

1) Пусть дана скалярная переменная $x[i, j]$ с индексами, меняющимися от 1 до 10 каждый. Тогда переменная x [,] будет

обозначать матрицу 10-го порядка, компонентами которой являются переменые $x\,[i, j]$, а переменная $x\,[i,]$ будет обозначать i-ю строку матрицы $x\,[,]$.

2) Пусть дана матрица 20-го порядка $M[r]$ с порядком по внешнему измерению, равным 10. Тогда $M\,[\,]$ будет обозначать трехмерный массив с порядками по измерениям 10, 20, 20.

Верхние индексы. Верхний индекс вводится для обозначения переменных, последовательные значения которых вычисляются посредством рекурсивных соотношений. Таким образом, наличие верхнего индекса говорит о том, что значения переменной образуют временну́ю последовательность. Текущее значение выражения, стоящего в позиции верхнего индекса, указывает номер значения переменной в данной последовательности. О правилах, регулирующих применение верхних индексов, см. в п. 15.4.6, операторы цикла, управляющие верхними индексами.

Перечисления переменных. Пусть перечисление переменных имеет вид

$$V\,[E1],\ \ldots,\ V\,[E2].$$

Пусть текущее значение $E1$ равно α, а текущее значение $E2$ равно ω. Тогда это перечисление обозначает список переменных, получаемых из $V\,[i]$ подстановкой в индекс i его последовательных значений, начиная с α и кончая ω. Число переменных в этом списке равно $\omega - \alpha + 1$. Перечисление переменных имеет смысл только в том случае, если $\omega - \alpha + 1 \geqslant 1$. Длина перечисления λ определяется по формуле

$$\lambda = \omega - \alpha + 1.$$

Сформированные переменные. Сформированная переменная — это переменная, значение которой есть относительный вектор с относительными компонентами, являющимися значениями переменных, входящих в список переменных. Сформированная переменная имеет смысл тогда, когда список переменных состоит из переменных одинаковой структуры. Пусть список переменных образован переменными размерности m с порядками по измерениям $t_1,\ t_2,\ \ldots,\ t_m$. В этом случае размерность сформированной переменной будет равна $m+1$ с порядками по измерениям $M,\ t_1,\ t_2,\ \ldots,\ t_m$, где M — длина списка, вычисляемая по формуле

$$M = \sum_{i=1}^{r} \lambda_i + S_1$$

где $\lambda_1, \lambda_2, \ldots, \lambda_r$ — значения длин r перечислений, входящих в список переменных ($r \geqslant 0$), и S — число переменных, входящих в список переменных ($S \geqslant 0$).

Все переменные из списка переменных должны иметь один и тот же тип, который и присваивается сформированной переменной.

Скомпонованные переменные. По определению скомпонованная переменная имеет вид

|[⟨целое без знака⟩] ⟨список переменных⟩|.

Пусть значение целого без знака равно G. Скомпонованная переменная имеет смысл тогда, когда все переменные из списка имеют одинаковую положительную размерность m и $G \leqslant m$. Все переменные, входящие в список, должны иметь одинаковые порядки по всем измерениям, кроме, может быть, измерения с номером G. Скомпонованная переменная обозначает переменную, значением которой является массив, полученный сращиванием массивов значений переменных из списка. Сращивание происходит по измерению с номером G.

Пусть в список переменных входят τ перечислений с порядками по измерению $\lambda_1, \lambda_2, \ldots, \lambda_r$, образованных переменными с порядками по G-му измерению $t^{(1)}, t^{(2)}, \ldots, t^{(n)}$, соответственно. Пусть также в список переменных входят S переменных с порядками по G-му измерению $\tau_1, \tau_2, \ldots, \tau_S$. Пусть, наконец, порядки всех переменных из списка по остальным $m-1$ измерениям суть $t_1, t_2, \ldots, t_{G-1}, t_{G+1}, \ldots, t_m$. Тогда скомпонованной переменной приписывается размерность m с порядками по измерениям

$$t_1, t_2, \ldots, t_{G-1}, T, t_{G+1}, \ldots, t_m,$$

где T — порядок по измерению с номером G — вычисляется по формуле

$$T = \sum_{i=1}^{r} \lambda_i t^{(i)} + \sum_{j=1}^{S} \tau_j.$$

15.3.2. Указатели функций.

Синтаксис.

⟨идентификатор функции⟩::=⟨идентификатор⟩

⟨функция-выражение⟩::=⟨идентификатор функции⟩ ⟨совокупность фактических параметров⟩

⟨идентификатор процедуры⟩::=⟨идентификатор⟩

⟨фактический параметр⟩::=⟨строка⟩|⟨выражение⟩|⟨идентификатор массива⟩|⟨идентификатор переключателя⟩|⟨идентификатор процедуры⟩|⟨идентификатор функции⟩

⟨строка букв⟩::=⟨буква⟩|⟨строка букв⟩ ⟨буква⟩

⟨ограничитель параметра⟩::=,|) ⟨строка букв⟩:(

⟨список фактических параметров⟩::=⟨фактический параметр⟩|⟨список фактических параметров⟩ ⟨ограничитель параметра⟩ ⟨фактический параметр⟩

⟨совокупность фактических параметров⟩::=⟨пусто⟩| (⟨список фактических параметров⟩)

⟨указатель функции⟩::=⟨идентификатор процедуры⟩ ⟨совокупность фактических параметров⟩ | ⟨функция-выражение⟩

Примеры.

$sin(a - b)$

$\tau(v+s,\ n)$

R

$S(s - 5)$ Температура: (T) Давление: (P)

$compile$ (': = ') $stack$:(Q)

СЛЕД (‖ $a\,[1,\ 11], \ldots, a\,[10,\ 11]$ | , ..., | $a\,[1,\ 20]$, ..., $a\,[10,\ 20]$ ‖)

Ортогонализация (n) Векторов: (| $x\,[1,], \ldots, x\,[n,]$ |) порядка: (S)

Семантика. Указатели функции определяют числовые или логические значения, которые получаются в результате применения заданных совокупностей правил, определяемых описанием функции (п. 15.5.5) к фиксированным совокупностям фактических параметров. Если значение описанной функции не является скалярным, то указатели этой функции могут использоваться только в качестве правых частей операторов присваивания. Структура функции при этом должна совпадать со структурой переменных левых частей соответствующих операторов присваивания. Правила, регулирующие задания фактических параметров, даны в (п. 15.4.7) и в (п. 15.5.5).

Стандартные функции. Некоторые идентификаторы закреплены за стандартными функциями. Ниже приводится полный список стандартных функций, используемых в языке альфа-6.

Естественно, что обращения к ним возможны только внутри тех блоков, где идентификаторы не определены в другом смысле. Все аргументы стандартных функций (кроме первого аргумента функции Σ) являются выражениями. Для указания того, какой тип (целый, вещественный или комплексный) может принимать аргумент и результат функции, будем записывать ее в следующем виде:

$функ\ (ABC) = ABC,$

где *функ* — имя стандартной функции, и *А*, *В*, *С* — одна из начальных букв слов **целый**, **вещественный**, **комплексный**, которая указывает, какой тип может принимать аргумент или результат. Причем букве в некоторой позиции аргумента соответствует буква в той же позиции результата. Например, запись *sin*(*ЦВК*) = *ВВК* указывает, что аргумент *sin* может быть целым, вещественным или комплексным, причем комплексному аргументу соответствует комплексный результат, а целому или вещественному — вещественный. Обозначения аргумента и результата могут иметь по две или по одной букве.

Все стандартные функции, кроме *det*, *shift*, *mod*, *max* и *min* являются покомпонентными операциями. Под компонентной операцией понимается операция, результатом которой является массив такой же размерности как и у аргумента, а элементы равны результату применения операции над соответствующими скалярными компонентами аргумента.

abs(*ЦВК*) = *ЦВК*	— абсолютная величина аргумента
mod(*ЦВК*) = *ВВВ*	— модуль аргумента, вычисляемый в общем случае как арифметическое значение квадратного корня из суммы произведений компонент массива на комплексно-сопряженные им величины
sin(*ЦВК*) = *ВВК*	— синус
cos(*ЦВК*) = *ВВК*	— косинус
tg(*ЦВК*) = *ВВК*	— тангенс
tan(*ЦВК*) = *ВВК*	— тангенс
arcsin (*ЦВК*) = *ВВК*	— главное значение арксинуса
arccos(*ЦВК*) = *ВВК*	— главное значение арккосинуса
arctan(*ЦВК*) = *ВВК*	— главное значение арктангенса
arctg(*ЦВК*) = *ВВК*	— главное значение арктангенса
sh(*ЦВК*) = *ВВК*	— гиперболический синус
ch(*ЦВК*) = *ВВК*	— гиперболический косинус
th(*ЦВК*) = *ВВК*	— гиперболический тангенс
arsh(*ЦВК*) = *ВВК*	— гиперболический арксинус
arch(*ЦВК*) = *ВВК*	— гиперболический арккосинус
arth(*ЦВК*) = *ВВК*	— гиперболический арктангенс
exp(*ЦВК*) = *ВВК*	— экспоненциальная функция
ln(*ЦВК*) = *ВВК*	— главное значение натурального логарифма
sqrt(*ЦВК*) = *ВВК*	— главное значение квадратного корня
Re(*К*) = *В*	— действительная часть комплексного значения аргумента

$Im(К) = В$ — мнимая часть комплексного значения аргумента

$arg(К) = В$ — главное значение аргумента комплексного значения

$con(К) = К$ — значение комплексно-сопряженного комплексному значению аргумента

$entier(ЦВ) = ЦЦ$ — целая часть вещественного значения аргумента, т. е. наибольшего целого, не превосходящего значения аргумента

$frac(ЦВ) = ВВ$ — дробная часть вещественного значения аргумента

$окр(ЦВ) = ЦЦ$ — округление значения аргумента

$$окр(Е) = \begin{cases} entier(E + 0.5), \\ \text{если } E \geqslant 0 \\ -entier(abs(E) + 0.5)), \\ \text{если } E \leqslant 0 \end{cases}$$

$sign(ЦВ) = ЦЦ$ — знак значения аргумента

$$sign(Е) = \begin{cases} 1, \text{ если } E > 0 \\ 0, \text{ если } E = 0 \\ -1, \text{ если } E < 0 \end{cases}$$

$det(\overline{Ц}В) = ВВ$ — определитель квадратной матрицы; функция определена для аргумента, имеющего абсолютную размерность 2, результат — скалярное значение

$shift(E, n)$ — сдвиг логического вектора E на n позиций. Функция определена для первого аргумента, имеющего абсолютную размерность типа логический и второго аргумента размерности 0 типа целый. Значением функции является логический вектор с длиной по измерению l, равной длине E. Если k-я компонента вектора E есть e_k $(1 \leqslant k \leqslant l)$, то k-ой компонентой вектора $shift(E, n)$ будет e_{k+n}, если $1 \leqslant k + n \leqslant l$ и ложь в остальных случаях

$\Sigma(Ц, Ц, Ц, ЦВК) = ЦВК$ — суммирование: $\Sigma(i, н, k, E) = \sum_{i=н}^{k} E_i$. Тип результата определяется типом четвертого аргумента. Первые три аргумента должны быть размерно-

сти 0, причем, первый аргумент — скалярная переменная, второй и третий — в общем случае выражения

arc(*ЦВ*, *ЦВ*) = *ВВ* — полярный угол точки с координатами *E1*, *E2*, где *E1* — значение первого аргумента, *E2* — второго. Значение результата берется из интервала $0 \leqslant arc < 2\pi$

div(*Ц*, *Ц*) = *Ц* — деление нацело *div* (*E1*, *E2*) = *sign*(*E1*/*E2*) × *entier*(*abs*(*E1*/*E2*))

res(*Ц*, *Ц*) = *Ц* — остаток от деления нацело. *res*(*E1*, *E2*) = *E1* — *div*(*E1*, *E2*)

min(*ЦВ*, *ЦВ*, ..., *ЦВ*) = *ЦВ* — наименьшее из значений, принимаемых аргументами. Для аргументов со структурой понимается как *min* совокупности всех значений, принимаемых аргументами функции

max(*ЦВ*, *ЦВ*, ..., *ЦВ*) = *ЦВ* — наибольшее из значений, принимаемых аргументами. Для аргументов со структурой понимается как *max* от совокупности значений, принимаемых аргументами функции.

15.3.3. Арифметические выражения.

Синтаксис.

⟨знак операции типа сложения⟩::=+|−

⟨знак операции типа умножения⟩::=×|/|÷|◦

⟨перечисление арифметических выражений⟩::= подстановка ⟨арифметического выражения⟩ в ⟨⟨арифметическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩, ..., подстановка ⟨арифметического выражения⟩ в ⟨⟨арифметическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩

⟨элемент списка арифметических выражений⟩::=⟨арифметическое выражение⟩ | ⟨перечисление арифметических выражений⟩

⟨список арифметических выражений⟩::=⟨элемент списка арифметических выражений⟩ | ⟨список арифметических выражений⟩, ⟨элемент списка арифметических выражений⟩

⟨формирование арифметического выражения⟩::=|⟨список арифметических выражений⟩|

⟨компоновка арифметического выражения⟩::=|[⟨целое без знака⟩] ⟨список арифметических выражений⟩|

⟨первичное выражение⟩::=⟨число без знака⟩ | ⟨переменная⟩ | ⟨указатель функции⟩ | ⟨формирование арифметического выражения⟩ | ⟨компоновка арифметического выражения⟩ | (⟨арифметическое выражение⟩)

⟨множитель⟩::=⟨первичное выражение⟩ | ⟨множитель⟩ ↑ ⟨первичное выражение⟩

⟨терм⟩::=⟨множитель⟩ | ⟨терм⟩ ⟨знак операции типа умножения⟩ ⟨множитель⟩

⟨простое арифметическое выражение⟩::=⟨терм⟩ | ⟨знак операции типа сложения⟩ ⟨терм⟩ | ⟨простое арифметическое выражение⟩ ⟨знак операции типа сложение⟩ ⟨терм⟩

⟨условие⟩::=**если** ⟨логическое выражение⟩ **то**

⟨**арифметическое** выражение⟩::=⟨простое арифметическое выражение⟩ | ⟨условие⟩ ⟨простое арифметическое выражение⟩ **иначе** ⟨арифметическое выражение⟩

Примеры.

Первичные выражения:

$+07.43_{10}8$

sum

$w\ [i+2.8]$

$cos\ (y+z\times 3)$

$(a-3/y+vu\uparrow 8)$

$z\uparrow[5]$

$|x,\ y,\ 1+a,\ N|$

$\|1,\ 2,\ 3|,\ |1,\ 4,\ 9|,\ |1,\ 8,\ 27\|$

$|x[i]\uparrow 2,\ \ldots,\ x|100|\uparrow 2|$

$|[1]|[2]\ A,\ B,\ C|,\ |[2],\ D,\ E,\ F|,\ |[2]\ G,\ H,\ K\|$

Множители:

omega

$sum\uparrow cos\ (y+z\times 3)$

$7.394_{10}-8\uparrow w\ [i+2]\uparrow(a-3/y+vu\uparrow 8)$

$\|1,\ 2,\ 3|,\ |1,\ 4,\ 9|,\ |1,\ 8,\ 27\|\uparrow(-1)$

Термы:

$omega\times sum\uparrow cos\ (y+z\times 3)/7.394_{10}-8\uparrow w\ [i+2]$

$(a-3/y+vu\uparrow 8)/2$

$|x,\ y,\ s.+a,\ N|\circ|x[97],\ \ldots,\ x|100||$

Простое арифметическое выражение:

$U-Yu+omega\times sum\uparrow cos(v+z\times 3)/7.394_{10}-8\uparrow w[i+2]$

Арифметические выражения:

$W \times u - Q\ (S + C u) \uparrow 2$

если $q > 0$ **то** $S + 3 \times Q/A$ **иначе** $2 \times S + 3 \times 8$

если $a < 0$ **то** $U + V$ **иначе если** $a \times b > 17$
то v/V **иначе если** $k \neq y$ **то** V/v **иначе** 0

$a \times sin(omega \times t)$

$0.57_{10} + 2 \times a\ [N \times (N-1)/12{,}0]$

$(A \times arctan\ (y) + z) \uparrow (7 + Q)$

если q **то** $n-1$ **иначе если** $B = 0$ **то** B/A **иначе** z

Семантика. Арифметическое выражение является правилом для вычисления числового значения. В случае простых арифметических выражений это значение получается путем выполнения указанных арифметических операций над фактическими числовыми значениями первичных выражений, входящих в данное выражение. Детально это объяснено в (п. 15.3.3, операции и типы). Что такое фактическое числовое значение первичного выражения, ясно в случае чисел. Для переменных оно является текущим значением (последнее по времени присвоенное значение), а для функций оно является значением, полученным по правилам вычислений, определяющих функцию (п. 15.5.5), применимым к текущим значениям параметров указателя функции, заданных в выражении. Значения формирования и компоновки арифметических выражений определяются точно так же, как значения сформированной и скомпонованной переменных в п. 15.3.1 (сформированные переменные, скомпонованные переменные) соответственно. Разница только в определении типа. Если все выражения из списка арифметических выражений имеют один и тот же тип, то этот тип присваивается сформированному (или скомпонованному) выражению. Если в списке фигурируют выражения типа **целый и вещественный**, то сформированное (или скомпонованное) выражение получает тип **вещественный**. Другие комбинации типов недопустимы.

Наконец, для арифметических выражений, заключенных в скобки, их значение должно определяться посредством рекурсивного анализа, исходя из значений остальных пяти видов первичных выражений.

В более общих арифметических выражениях, включающих в себя условия, выбор одного из нескольких простых арифметических выражений происходит, основываясь на фактических значениях логических выражений (15.3.4). Этот выбор производится следующим образом: значения логических выражений, входящих в условия, вычисляются последовательно

слева направо до тех пор, пока не найдется выражение, имеющее значение **истина** (если логическое выражение в условии является массивом, оно принимает значение **истина**, только если все его компоненты принимают значение **истина**). Значением арифметического выражения тогда будет значение первого арифметического выражения, следующего за этим логическим (при этом имеется в виду максимальное арифметическое выражение, стоящее в этой позиции).
Конструкция:

иначе ⟨простое арифметическое выражение⟩

эквивалентна конструкции:

иначе если истина то ⟨простое
арифметическое выражение⟩

Операции и типы. Составные части простых арифметических выражений (за исключением логических выражений, употребляемых в условиях) должны быть типов **вещественный**, **комплексный** или **целый** (15.5.1, описания типа). Смысл основных операций и типы выражений, к которым они приводят, даются следующими правилами.

Операции +, — имеют обычный смысл (сложение, вычитание). При этом оба операнда должны иметь одинаковую структуру, а сами операции выполняются покомпонентно. Типом выражения является **целый**, если оба операнда имеют тип **целый**, в противном случае — **вещественный** или **комплексный** (последнее в том случае, если хотя бы один из операндов имеет тип **комплексный**).

Операция × определена в случаях, приведенных в табл. 15.1. Тип выражения определяется так же, как для операций +, —.

Операция • означает покомпонентное умножение, определяемое для операндов одинаковой структуры. Структура выражения совпадает со структурой операндов. Тип выражения определяется так же, как для операций +, —.

Операции ⟨терм⟩/⟨множитель⟩ и ⟨терм⟩ ÷ ⟨множитель⟩ означают покомпонентное деление, понимаемое как умножение компоненты терма на обратную величину компоненты множителя с соответствующим учетом правил старшинства (см. п. 15.3.3, старшинство операций). Таким образом, например,

$$a/b\times 7/(p-q)\times v/S$$

означает

$$(((a\times(b^{-1})\times 7)\times((p-q)^{-1}))\times v)\times(S^{-1})$$

Таблица 15.1

Комбинация структуры операндов A и B	Как понимается операция
A и B — скаляры	обычным образом
A и B — векторы одного порядка	скалярное произведение векторов
A — вектор порядка n B — матрица порядка $n \times m$	произведение вектора на матрицу, результат вектор порядка m
A — матрица порядка $n \times m$ B — вектор порядка m	произведение матрицы на вектор, результат вектор порядка n
A — матрица порядка $n \times m$ B — матрица порядка $m \times k$	произведение матриц, результат матрица порядка $n \times k$
A — массив B — скаляр	покомпонентное произведение массива на скаляр
A — скаляр B — массив	

Оба операнда операций / и ÷ должны иметь одинаковую структуру. Операция / определена для всех четырех комбинаций типов **вещественный** и **целый** и в любом случае дает результат типа **вещественный**. Если хотя бы один из операндов имеет тип **комплексный**, то результат также определен и имеет тип **комплексный**. Операция ÷ определена только для того случая, когда оба операнда имеют тип **целый**, и дает результат типа **целый**, определяемый следующим образом:

$$a \div b = sign(a/b) \times entier(abs(a/b)),$$

(п. 15.3.2, стандартные функции).

Операция ⟨множитель⟩ ↑ ⟨первичное выражение⟩ означает возведение в степень, где множитель есть основание, а первичное выражение есть показатель степени. Таким образом, например,

$2\uparrow n\uparrow k$ означает $(2^n)^k$

тогда как

$2\uparrow(n\uparrow m)$ означает $2^{(n)^m}$

Показатель степени может быть только скалярным. Если ти-

сать i вместо числа типа **целый**, r — вместо числа типа **вещественный**, c — вместо числа типа **вещественный** или **комплексный** и a — вместо числа типа **вещественный**, **целый** или **комплексный**, то результат определяется следующими правилами:

$a\uparrow i$ если $i>0$, то $a\uparrow i=a\times a\times\ldots\times a$ (i раз) того же типа, что и a;
если $i=0$, то при $a\neq 0$ $a\uparrow 0=1$ того же типа, что и a, при $a=0$ не определено;
если $i<0$, то при $a\neq 0$ $a\uparrow i=1/(a\times a\times\ldots\times a)$ (знаменатель имеет i множителей) типа **вещественный** (и типа **комплексный** для a типа **комплексный**), при $a=0$ не определено.

$a\uparrow r$ (для a типа **вещественный** или **целый**),
если $a>0$, то $a\uparrow r=exp(r\times ln(a))$ типа **вещественный**,
если $a=0$, то при $r>0$, $0\uparrow r=0.0$ типа **вещественный**,
при $r\leqslant 0$ не определено,
если $a<0$, то всегда не определено.

$a\uparrow c$ (для a типа **комплексный**).
если $a\neq 0$, то $a\uparrow c=exp\,(c\times ln\,(a))$ типа **комплексный**.
если $a=0$, то всегда не определено.

В общем случае возведение в степень понимается как покомпонентное действие. Результат имеет ту же структуру, что и основание степени. Из общего случая с целым показателем выделяется случай, когда основанием степени является квадратная матрица. В этом случае умножение, к которому сводится возведение в степень, понимается как матричное умножение. Отрицательная степень понимается как положительная степень обратной матрицы.

В условном арифметическом выражении типы подвыражений, стоящих после ограничителя **то** и ограничителя **иначе** должны совпадать. Тип условного выражения определяется типом подвыражения, следующего после ограничителя **то**.

Старшинство операций.

Операции в пределах одного выражения выполняются, вообще говоря, в последовательности слева направо с учетом следующих добавочных правил.

Согласно синтаксису, данному в п. 15.3.3, выдерживается следующий порядок старшинства:

первый: $\uparrow$
второй: $\times$ / $\div$ $\circ$
третий: $+$ $-$

Выражение между левой скобкой и соответствующей правой скобкой вычисляется самостоятельно, и полученное зна-

чение используется в дальнейших вычислениях. Следовательно, желаемый порядок выполнения операций в пределах выражения всегда может быть достигнут соответствующей расстановкой скобок.

Арифметика величин типов **вещественный и комплексный**. Числа и переменные типа **вещественный**, а также переменные типа **комплексный** должны интерпретироваться с присущей им конечной точностью. Аналогично, в любом арифметическом выражении подразумевается возможность конечного отклонения от математически определяемого результата. Никакая точная арифметика для выражений этих типов не определяется, и, конечно, имеется в виду, что различные конкретные представления могут вычислять значения арифметических выражений по-разному. Контролировать возможные последствия таких различий необходимо численными методами.

15.3.4. Логические выражения.

Синтаксис.

⟨операция типа меньше⟩::= < | ≤ | =
⟨операция типа больше⟩::= > | ≥ | =
⟨перечисление отношений типа меньше⟩::=
подстановка ⟨арифметического выражения⟩ в ⟨⟨простое арифметическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩ ⟨⟨операция типа меньше⟩⟩... ⟨⟨операция типа меньше⟩⟩
подстановка ⟨арифметического выражения⟩ в ⟨⟨простое арифметическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩
⟨перечисление отношений типа больше⟩::=
подстановка ⟨арифметического выражения⟩ в ⟨⟨простое арифметическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩ ⟨⟨операция типа больше⟩⟩... ⟨⟨операция типа больше⟩⟩
подстановка ⟨арифметического выражения⟩ в ⟨⟨простое арифметическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩
⟨элемент отношения типа меньше⟩::= ⟨простое арифметическое выражение⟩ | ⟨перечисление отношений типа меньше⟩
⟨элемент отношения типа больше⟩::= ⟨простое арифметическое выражение⟩ | ⟨перечисление отношений типа больше⟩
⟨отношение типа меньше⟩::= ⟨простое арифметическое выражение⟩ ⟨операция типа меньше⟩ ⟨простое арифмети-

ческое выражение⟩ | ⟨перечисление отношений типа меньше⟩

⟨отношение типа больше⟩::=⟨простое арифметическое выражение⟩ ⟨операция типа больше⟩ ⟨простое арифметическое выражение⟩ | ⟨перечисление отношений типа больше⟩

⟨цепочка отношений типа меньше⟩::=⟨отношение типа меньше⟩ | ⟨цепочка отношений типа меньше⟩ ⟨операция типа меньше⟩ ⟨элемент отношения типа меньше⟩

⟨цепочка отношений типа больше⟩::=⟨отношение типа больше⟩ | ⟨цепочка отношений типа больше⟩ ⟨операция типа больше⟩ ⟨элемент отношений типа больше⟩

⟨цепочка отношений⟩::=⟨цепочка отношений типа меньше⟩| ⟨цепочка отношений типа больше⟩

⟨отношение⟩::=⟨цепочка отношений⟩ | ⟨простое арифметическое выражение⟩ ≠ ⟨простое арифметическое выражение⟩

⟨перечисление логических выражений⟩::=
подстановка ⟨арифметического выражения⟩ в ⟨⟨логическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩, . . .,
подстановка ⟨арифметического выражения⟩ в ⟨⟨логическое выражение⟩⟩ на место ⟨⟨индексного выражения⟩⟩

⟨элемент списка логических выражений⟩::=⟨логическое выражение⟩ | ⟨перечисление логических выражений⟩

⟨список логических выражений⟩::=⟨элемент списка логических выражений⟩ | ⟨список логических выражений⟩, ⟨элемент списка логических выражений⟩

⟨формирование логического выражения⟩::=|⟨список логических выражений⟩|

⟨компоновка логического выражения⟩::=|[⟨целое без знака⟩] ⟨список логических выражений⟩|

⟨первичное логическое выражение⟩::=⟨логическое значение⟩| (⟨логическое выражение⟩) |⟨переменная⟩| ⟨указатель функции⟩ | ⟨отношение⟩ | ⟨формирование логического выражения⟩|⟨компоновка логического выражения⟩

⟨вторичное логическое выражение⟩::=⟨первичное логическое выражение⟩| ¬⟨первичное логическое выражение⟩

⟨логический одночлен⟩::=⟨вторичное логическое выражение⟩| ⟨логический одночлен⟩∧⟨вторичное логическое выражение⟩

⟨логический терм⟩::=⟨логический одночлен⟩|⟨логический терм⟩∨⟨логический одночлен⟩

⟨импликация⟩::=⟨логический терм⟩|⟨импликация⟩⊃⟨логический терм⟩

⟨простое логическое выражение⟩::=⟨импликация⟩|⟨простое логическое выражение⟩≡⟨импликация⟩|⟨простое логическое выражение⟩⊕⟨импликация⟩

⟨логическое выражение⟩::=⟨простое логическое выражение⟩|⟨условие⟩ ⟨простое логическое выражение⟩ **иначе** ⟨логическое выражение⟩

Примеры.

$x = -2$

$a < x \leqslant b$

$d[1] = \ldots = d[n] = 0$

$M > T \geqslant t \geqslant m$

$Y > V \vee Z < q$

$a + b > -5 \wedge z - d > q \uparrow 2$

$p \wedge q \vee x \neq y$

$g \equiv \neg a \wedge b \wedge \neg c \vee d \vee e \supset \neg f$

если $k < l$ **то** $s > w$ **иначе** $h \leqslant c$

если если a **то** b **иначе** f **то** q **иначе** $h < k$

Семантика. Логическое выражение является правилом для вычисления логического значения. Принципы вычисления полностью аналогичны правилам, данным в п. 15.3.3 (семантика) для арифметических выражений.

Типы. Переменным и указателям функции, используемым в качестве первичных логических выражений, должен приписываться тип **логический** (п. 15.5.1, п. 15.5.5, значения указателей функции).

Операции. Отношение принимает значения **истина** в том случае, когда соответствующее отношение удовлетворяется для входящих в него выражений; в противном случае оно принимает значение **ложь**. Отношения, связывающие массивы, определяются только для массивов одинаковой структуры. Исключение составляют только скалярные выражения 0, 0.0 и .0 (нуль). Им может быть приписана любая структура, которую требует другой операнд отношения. Для всех отношений, кроме $\neq$, оно имеет значение **истина** только в том случае, когда это отношение выполняется по всем компонентам массива. Отношение $\neq$ имеет значение **истина**, если оно выполняется хотя бы для одной пары соответствующих компонент. Значения логических операций $\neg$ (не), $\wedge$ (и), $\vee$ (или), $\supset$ (влечет), $\equiv$ (эквивалентно) и $\oplus$ (сложение по модулю два) даются следующей функциональной таблицей 15.2.

Таблица 15.2

в1	ложь	ложь	истина	истина
в2	ложь	истина	ложь	истина
¬ *в1*	истина	истина	ложь	ложь
в1 ∧ *в2*	ложь	ложь	ложь	истина
в1 ∨ *в2*	ложь	истина	истина	истина
в1 ⊃ *в2*	истина	истина	ложь	истина
в1 ≡ *в2*	истина	ложь	ложь	истина
в1 ⊕ *в2*	ложь	истина	истина	ложь

Старшинство операций. Операции в пределах одного выражения выполняются, вообще говоря, в последовательности слева направо с учетом следующих добавочных правил.

Согласно синтаксису, данному в п. 15.3.4 (синтаксис) выдерживается следующий порядок старшинства:

первый: арифметические выражения
согласно п. 15.3.3 (старшинство операций)
второй: $<$, $\leqslant$, $=$, $>$, $\geqslant$, $\neq$
третий: ¬
четвертый: ∧
пятый: ∨
шестой: ⊃
седьмой: ≡, ⊕

Применение скобок интерпретируется в смысле, данном в п. 15.3.3 (старшинство операций).

15.3.5. Именующие выражения.

Синтаксис.

⟨метка⟩::=⟨идентификатор⟩|⟨целое без знака⟩
⟨идентификатор части⟩::=PART | ⟨идентификатор части⟩ ⟨буква⟩|⟨идентификатор части⟩ ⟨цифра⟩
⟨метка PART⟩::=⟨идентификатор части⟩
⟨идентификатор переключателя⟩::=⟨идентификатор⟩
⟨указатель переключателя⟩::=⟨идентификатор переключателя⟩ [⟨индексное выражение⟩]
⟨составная метка⟩::=⟨метка⟩.⟨метка⟩|⟨составная метка⟩.⟨метка⟩ | ⟨метка⟩ ⟨метка PART⟩ | ⟨метка PART⟩.⟨метка⟩
⟨простое именующее выражение⟩::=⟨метка⟩|⟨указатель переключателя⟩|⟨составная метка⟩| (⟨именующее выражение⟩)|⟨метка PART⟩

⟨именующее выражение⟩::=⟨простое именующее выражение⟩|⟨условие⟩ ⟨простое именующее выражение⟩ **иначе** ⟨именующее выражение⟩

Примеры.

17

*p*9

Выбирать [*n*—1]

Town [**если** $y < 0$ **то** N **иначе** $N + 1$]

если $Ab < C$ **то** *17* **иначе** *q* [**если** $w < 0$ **то** *2* **иначе** *n*]

Семантика. Именующее выражение является правилом для получения метки оператора (п. 15.4). Принципы вычисления значения именующего выражения по-прежнему полностью аналогичны правилам, приведенным для арифметических выражений в п. 15.3.3 (семантика). В общем случае логические выражения, содержащиеся в условиях, выбирают простое именующее выражение. Если эта метка, или составная метка, то желаемый результат уже получен. Указатель переключателя отсылает к соответствующему описанию переключателя (п. 15.5.3) и по фактическому числовому значению своего индексного выражения выбирает одно из именующих выражений, перечисленных в описании переключателя, отсчитывая эти выражения слево направо. Так как выбранное таким образом именующее выражение может, в свою очередь, оказаться указателем переключателя, то вычисление значения, очевидно, представляет собой рекурсивный процесс.

Метки PART используются для определения внешних блоков (15.4.1).

Индексное выражение. Вычисление индексного выражения, которое должно быть непустым, производится точно так же как и для переменных с индексами (15.3.1, индексы). Значение указателя переключателя определено только в том случае, когда индексное выражение принимает положительные значения 1, 2, 3, ..., *n*, где *n* — есть число членов в списке переключателей.

Целые без знака в качестве меток. Целые без знака, используемые в качестве меток, обладают тем свойством, что впереди стоящие нули не изменяют их значения. Например, 00217 означает ту же метку, что и 217.

15.4. Операторы

Единицы действий в языке называются операторами. Обычно они выполняются в той последовательности, в которой написаны. Однако эта последовательность выполнения может

прерываться операторами перехода, которые явно определяют своего преемника, и условными операторами, которые могут вызывать пропуск некоторых операторов.

Для того, чтобы имелась возможность указывать фактический порядок следования операторов в процессе работы, операторы могут снабжаться метками.

Ввиду того, что последовательности операторов можно группировать в составные операторы и блоки, определение оператора по необходимости должно быть рекурсивным. Кроме того, поскольку описания, о которых говорится в п. 15.5, существенно входят в синтаксическую структуру, синтаксическое определение операторов должно предполагать, что описания уже определены.

15.4.1. Составные операторы и блоки.

Синтаксис.

⟨непомеченный основной оператор⟩::=⟨оператор присваивания⟩|⟨оператор перехода⟩|⟨пустой оператор⟩|⟨оператор процедуры⟩|⟨оператор останова⟩|⟨оператор разметки⟩|⟨оператор вывода⟩|⟨оператор ввода⟩|⟨оператор текст⟩|⟨оператор обмена⟩|⟨оператор БЕМШ⟩|⟨оператор канала⟩

⟨основной оператор⟩::=⟨непомеченный основной оператор⟩|⟨метка⟩:⟨основной оператор⟩

⟨безусловный оператор⟩::=⟨основной оператор⟩|⟨составной оператор⟩|⟨блок⟩

⟨оператор⟩::=⟨безусловный оператор⟩|⟨условный оператор⟩|⟨оператор цикла⟩

⟨конец составного⟩::=⟨оператор⟩ **конец**|⟨оператор⟩; ⟨конец составного⟩

⟨начало блока⟩::=**начало** ⟨описание⟩|⟨начало блока⟩ ⟨описание⟩

⟨непомеченный составной⟩::=**начало** ⟨конец составного⟩

⟨непомеченный блок⟩::=⟨начало блока⟩; ⟨конец составного⟩

⟨составной оператор⟩::=⟨непомеченный составной⟩|⟨метка⟩:⟨составной оператор⟩

⟨блок⟩::=⟨непомеченный блок⟩|⟨метка⟩:⟨блок⟩|⟨внешний блок⟩

⟨внешний блок⟩::=⟨метка PART⟩:⟨непомеченный блок⟩

⟨тело программы⟩::=⟨непомеченный составной⟩|⟨непомеченный блок⟩

⟨программа⟩::=⟨подпрограмма⟩

⟨подпрограмма⟩::=⟨имя программы⟩:⟨тело программы⟩|⟨описание процедуры⟩|⟨описание процедуры-функции⟩

⟨имя программы⟩::=⟨идентификатор⟩

Этот синтаксис можно проиллюстрировать следующим образом. Обозначим произвольные операторы, описания и метки буквами S, D и L соответственно. Тогда основные синтаксические единицы примут следующий вид:

Составной оператор:

$L{:}L{:}\ \ldots$ **начало** S; S; $\ldots$ S; S **конец**

Блок:

$L{:}L{:}\ \ldots$ **начало** D; D; $\ldots$ D; S;
S; $\ldots$ S; S **конец**

При этом нужно помнить, что каждый из операторов S может, в свою очередь, быть полным составным оператором или блоком.

Примеры.

Основные операторы:

$a:=p+q$
на *Naples*
$start{:}continue{:}V:=7.993$

Составной оператор:

начало $x:=0$; **для** $y:=1$ **шаг** 1 **до** n **цикл** $x:=x+A[y]$;
если $x>q$ **то на** *stop* **иначе если** $x>w-2$ **то на** S
конец

Блок:

Q: **начало целый** i, k; **вещественный** N;
для $i:=1$ **шаг** 1 **до** m **цикл**
для $k:=i+1$ **шаг** 1 **до** m **цикл**
начало $w:=A[i, k]$; $A[i, k]:=A[k, i]$;
$A[k, i]:=w$
конец для i **и** k
конец блока Q.

Семантика. Каждый блок автоматически вводит новый уровень обозначений. Это реализуется следующим образом. Любой идентификатор, встречающийся внутри блока, можно определить посредством соответствующего описания (п. 15.5) как локальный внутри соответствующего блока. Это означает: а) что объект, представленный этим идентификатором внутри данного блока, не существует вне блока и б) что любой объект, представленный тем же идентификатором вне данного блока, нельзя использовать внутри блока.

Идентификаторы (за исключением тех, которые изображают метки), встречающиеся внутри блока и не описанные в

нем, не локализуются в блоке, т. е. представляют одни и те же объекты как внутри блока, так и в объемлющих блоках. Метка, отделенная двоеточием от оператора, т. е. помечающая этот оператор, действует так, как будто она описана в наименьшем объемлющем блоке, т. е. наименьшем блоке, скобки которого **начало** и **конец** заключают этот оператор. В этом отношении тело процедуры нужно рассматривать так, как если бы оно было заключено между **начало** и **конец**, и трактовать как блок.

Тело цикла трактуется как блок, поэтому метки, описанные в теле цикла, недоступны извне тела цикла.

Так как некоторый оператор в блоке может, в свою очередь, тоже быть блоком, то концепцию локальности и нелокальности в блоке следует понимать рекурсивно. Таким образом, идентификатор, не локальный в блоке A, может быть локальным или не локальным в блоке B, для которого A является одним из его операторов. Внешние блоки, т. е. блоки, помеченные метками PART, при выполнении программы размещаются во внешней памяти и вызываются в оперативную память при входе в эти блоки.

Подпрограмма — это конструкция языка альфа-6 (блок, составной оператор, описание процедуры или процедуры функции), которая не входит составной частью в другую конструкцию языка и является замкнутой в том смысле, что все объекты этой конструкции локализованы внутри нее. Если подпрограмма есть описание процедуры или процедуры-функции, то все ее формальные параметры должны быть специфицированы, а для формальных параметров, являющихся массивами, должны быть указаны их размерности (п. 15.5.4).

15.4.2. Операторы присваивания.

Синтаксис.

⟨перечисление присваиваний⟩::=подстановка ⟨арифметического выражения⟩ в ⟨⟨переменную⟩⟩ на место ⟨⟨индексного выражения⟩⟩: = ... : = подстановка ⟨арифметического выражения⟩ в ⟨⟨переменную⟩⟩ на место ⟨⟨индексного выражения⟩⟩

⟨левая часть⟩::=⟨переменная⟩:=|⟨перечисление присваиваний⟩: = |⟨идентификатор процедуры⟩: =

⟨список левой части⟩::=⟨левая часть⟩|⟨список левой части⟩ ⟨левая часть⟩

⟨оператор присваивания⟩::=⟨список левой части⟩ ⟨арифметическое выражение⟩|⟨список левой части⟩ ⟨логическое выражение⟩

Примеры.

$s:=p[0]:=n:=n+1+s$

$n:=n+1$

$A:=B/C-v-q+S$

$S\,[v,\ k+2]:=3-arctg\,(3\times z)$

$v:=a>y\wedge z$

$k\uparrow[n+1]:=f(x+a[n]\times h,\ y\times b[n]\times k\uparrow[n])$

$delta:=det(E-A\uparrow(-1)\times B\times A)$

$z:=|\,t[i(1)],\ \ldots,\ t[i(n)]\,|\circ|\,t[j(1)],\ \ldots,\ t[j(n)]\,|$

$y[1]:=\ldots:=y[n]:=1$

Семантика. Операторы присваивания служат для присваивания значения выражения одной или нескольким переменным. Подразумевается, что в общем случае этот процесс происходит в три этапа:

1) Значения всех индексных выражений, встречающихся в переменных левой части, вычисляются в порядке слева направо. Перечисления присваиваний заменяются на те фактические последовательности переменных, которые обозначены этими перечислениями.

2) Вычисляется значение выражения в операторе.

3) Значение выражения присваивается всем переменным левой части с любым индексным выражением, имеющим значение, вычисленное на шаге первого этапа.

Типы и структуры. Все переменные из списка левой части должны по описанию иметь один и тот же тип и структуру, совпадающую со структурой значения вычисляемого выражения. Однако значения скалярных выражений 0, 0.0 и .0 (нуль) могут присваиваться переменным любой структуры. По соглашению считается, что нуль принимает ту же структуру, что и переменные списка левой части. Это правило служит и для логического выражения **ложь**. Если переменные имеют тип **логический** (**комплексный**), выражение также должно быть типа **логический** (**комплексный**). Если переменные имеют тип **вещественный** или **целый**, то выражение должно быть арифметическим типа **вещественный** или **целый**. Если тип арифметического выражения отличается от типа переменных и идентификаторов процедур, то подразумевается, что автоматически нужно применить соответствующие преобразующие функции. Имеется в виду, что для преобразования из типа **вещественный** в тип **целый** функция преобразования выдает результат, эквивалентный

$$entier(E+0.5),$$

где E — значение выражения.

В случае, когда в левую часть оператора присваивания входит идентификатор процедуры, его тип задается описателем, который является первым символом описания процедуры-функции (п. 15.5.5).

15.4.3. Операторы перехода.

Синтаксис.

⟨оператор перехода⟩::=**на** ⟨именующее выражение⟩

Примеры.

на *8*

на *exit* [*n*+*1*]

на *Town* [**если** *y*<*0* **то** *N* **иначе** *N*+*1*]

на если *Ab* < *C* **то** *17* **иначе** *q* [**если** *w* **то** *2* **иначе** *n*]

на 15.3.5 примеры

Семантика. Оператор перехода прерывает естественную последовательность выполнения операторов, задаваемую порядком их написания, явно определяя своего преемника по значению именующего выражения. Таким образом, следующим выполняемым оператором будет тот, который имеет это значение в качестве своей метки.

Случай составной метки. Если значением именующего выражения оказывается составная метка, например, *M1. M2. M3*, то преемника оператора перехода находят по правилу: сначала по общему правилу находят блок с меткой *M1*, потом в этом блоке находят блок с меткой *M2* и в последнем — оператор с меткой *M3*.

Переход при неопределенном указателе переключателя. В том случае, когда именующее выражение есть указатель переключателя, значение которого неопределено, оператор перехода также неопределен.

15.4.4. Пустые операторы.

Синтаксис.

⟨пусто оператор⟩::=⟨пусто⟩

Примеры.

L:

начало ...; *John*: **конец**

Семантика. Пустой оператор не выполняет никаких действий. Он может служить для помещения метки.

15.4.5. Условные операторы.

Синтаксис.

⟨условие⟩::=**если** ⟨логическое выражение⟩ **то**

⟨безусловный оператор⟩::=⟨основной оператор⟩|⟨составной оператор⟩|⟨блок⟩

⟨оператор *если*⟩::=⟨условие⟩ ⟨безусловный оператор⟩
⟨условный оператор⟩::=⟨оператор *если*⟩|⟨оператор *если*⟩ **иначе** ⟨оператор⟩ | ⟨условие⟩ ⟨оператор цикла⟩ | ⟨метка⟩:⟨условный оператор⟩

Примеры.

если $x>0$ **то** $n:=n+1$
если $v>u$ **то** $v:=q:=n+m$ **иначе на** R
если $s<0 \vee P<Q$ **то** AA: **начало если**
$q<v$ **то** $a:=v/s$
иначе $y:=2\times a$ **конец**
иначе если $v>s$ **то** $a:=v-q$
иначе если $v>s-1$ **то на** S

Семантика. Условные операторы приводят к пропуску или выполнению некоторых операторов в зависимости от текущих значений указанных логических выражений.

Оператор *если*. Безусловный оператор, входящий в оператор *если* выполняется, когда логическое выражение, входящее в условие, является истинным. В противном случае он пропускается и процесс выполнения продолжается, начиная со следующего оператора.

Условный оператор. Согласно синтаксису, возможны две различные формы условных операторов. Эти формы иллюстрируются следующими примерами:

если *B1* **то** *S1* **иначе если** *B2* **то** *S2* **иначе** *S3*; *S4*;

и

если *B1* **то** *S1* **иначе если** *B1* **то** *S2* **иначе если** *B3* **то** *S3*; *S4*;

Здесь *B1*, *B2*, *B3* — логические выражения, а *S1*, *S2*, *S3* — безусловные операторы. *S4* — оператор, следующий за условным оператором с логическим выражением *B1*.

Выполнение условного оператора может быть описано следующим образом. В порядке слева направо вычисляются значения логических выражений, стоящих в условиях. Вычисления продолжаются до тех пор, пока не будет найдено выражение, имеющее значение **истина**. После этого выполняется безусловный оператор, следующий непосредственно за этим логическим выражением. Если этот оператор не определит своего преемника явно, то следующим выполняемым оператором будет *S4*, т. е. оператор, следующий за внешним условным оператором. Таким образом, действие ограничителя **иначе** можно охарактеризовать, сказав, что он определяет в качестве преемника

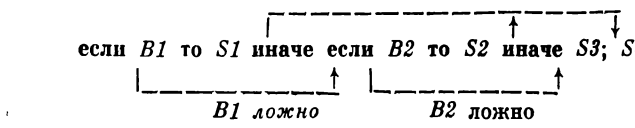
оператора, за которым он следует, оператор, следующий за внешним условным оператором. Конструкция

иначе ⟨безусловный оператор⟩

эквивалентна конструкции

иначе если истина то ⟨безусловный оператор⟩

Если ни одно из логических выражений, входящих в условия, не является истинным, то результат работы всего условного оператора будет эквивалентен работе пустого оператора. Для дальнейших пояснений может быть полезна следующая схема:



Переход внутрь условного оператора. Результат работы оператора перехода, ведущего внутрь условного оператора, непосредственно следует из объясненного выше действия ограничителя **иначе**.

15.4.6. Операторы цикла.

Синтаксис.

⟨элемент списка цикла⟩:: = ⟨арифметическое выражение⟩ | ⟨арифметическое выражение⟩ **шаг** ⟨арифметическое выражение⟩ **до** ⟨арифметическое выражение⟩ | ⟨арифметическое выражение⟩ **пока** ⟨логическое выражение⟩ | ⟨арифметическое выражение⟩, ..., ⟨арифметическое выражение⟩

⟨список цикла⟩:: = ⟨элемент списка цикла⟩ | ⟨список цикла⟩, ⟨элемент списка цикла⟩

⟨параметр цикла⟩:: = ⟨простая переменная⟩ | ⟨переменная с индексами⟩

⟨заголовок цикла⟩:: = **для** ⟨параметр цикла⟩: = ⟨список цикла⟩ **цикл**

⟨заголовок цикла без параметра⟩:: = ⟨первичное выражение⟩ **раз цикл** | **пока** ⟨логическое выражение⟩ **цикл**

⟨оператор цикла без параметра⟩:: = ⟨заголовок цикла без параметра⟩ ⟨оператор⟩

⟨оператор цикла⟩:: = ⟨заголовок цикла⟩ ⟨оператор⟩ | ⟨метка⟩: ⟨оператор цикла⟩ | ⟨оператор цикла без параметра⟩

Примеры.

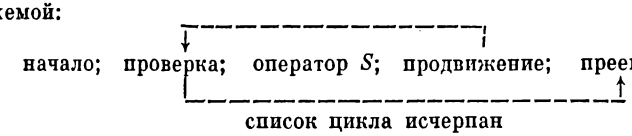
для $q:=1$ **шаг** s **до** n **цикл** $A[q]:=B[q]$

для $K:=1$, $v1\times 2$ **пока** $v1<N$ **цикл**

для *j*: = *y* + *G*, *L*, *1* **шаг** *1* **до** *N*, *C* + *D* **цикл**
A[*k*, *j*]: = *B*[*k*, *j*]
для *i*: = *1*, ..., *200* **цикл** *f*↑[*i* + *1*]: = *f*↑[*i*] + *f*↑[*i*—*1*]
200 **раз цикл** *x*: = *A*×*x*+*B*
пока (*x*—*y*) > $_{10}$—*3* **цикл начало** *y*: = *x*;
x: = *k*×(*x*×*sin*(*x*))×
b×*exp*(*x*/*2*) **конец**

Семантика. Все выражения и переменные, входящие в заголовок цикла, должны быть скалярными, вещественного или целого типа. Заголовок цикла без параметра вида «*E* **раз**» заставляет следующий за ним оператор повторно выполняться столько раз, каково текущее значение выражения *E*, вычисленное перед выполнением цикла. Выражение *E* вычисляется так же, как индексное выражение. Если значение *E* меньше нуля, то оператор не выполняется ни разу. Цикл без параметра с заголовком вида «**пока** *B*» выполняется следующим образом. Проверяется значение *B*. Если значение *B* равно **ложь**, то оператор, следующий за заголовком цикла, пропускается. В противном случае этот оператор выполняется, затем снова проверяется значение *B* и т. д. Рассмотрим теперь циклы с параметром.

Заголовок цикла заставляет стоящий за ним оператор повторно выполняться нуль или более раз. Кроме того, он осуществляет последовательные присваивания значений параметру цикла. Этот процесс может быть пояснен следующей схемой:



В этой схеме слово «начало» означает: произвести первое присваивание в заголовке цикла. «продвижение» означает произвести очередное присваивание в заголовке цикла; «проверка» определяет, было ли сделано последнее присваивание. Если оно сделано, то выполнение продолжается с преемника оператора цикла. В противном случае выполняется оператор, следующий за заголовком цикла.

Элементы списка цикла. Список цикла дает правило для получения значений, которые последовательно присваиваются параметру цикла. Эта последовательность значений получается из элементов списка цикла путем их последовательного перебора в порядке их написания. Последовательность значений, порождаемая каждой из четырех разно-

видностей элементов списка цикла, и соответствующее выполнение оператора S определяются следующими правилами:

1) Арифметическое выражение. Этот элемент задает только одно значение, а именно значение данного арифметического выражения, вычисленное непосредственно перед соответствующим выполнением оператора S.

2) Элемент типа арифметической прогрессии. Элемент, имеющий вид A **шаг** B **до** C, где A, B и C — арифметические выражения, задает порядок выполнения, который наиболее четко можно описать с помощью дополнительных операторов языка следующим образом:

$V:=4;$
$L1$: **если** $(V-C)\times sign(B)>0$
то на элемент исчерпан;
Оператор S;
$V:=V+B$;
на $L1$;

где V — параметр цикла, и «элемент исчерпан» указывает на переход либо к вычислениям, соответствующим следующему элементу списка цикла, либо, если данный элемент типа арифметической прогрессии стоит последним в списке, следующий оператор программы.

3) Элемент типа пересчета. Порядок выполнения, определяемый элементом списка цикла вида «E **пока** F», где E — арифметическое, а F — логическое выражение, наиболее четко описывается с помощью дополнительных операторов языка следующим образом:

$L3$: $V:=E$;
если $\neg F$ **то на** элемент исчерпан;
оператор S;
на $L3$;

где обозначения те же, что и для элемента типа арифметической прогрессии.

4) Элемент типа перечисления. Элемент, имеющий вид «E_1, ..., E_2», где E_1 и E_2 — арифметические выражения, в точности эквивалентен элементу вида «E_1 **шаг** 1 **до** E_2».

Значение параметра цикла после окончания работы цикла. После выхода из оператора S посредством какого-либо оператора перехода значение параметра цикла будет таким, каким оно было непосредственно перед выполнением оператора перехода.

С другой стороны, если выход из цикла вызван исчерпанием списка цикла, то значение параметра цикла после выхода из цикла не определено.

Оператор перехода, ведущий в оператор цикла. Тело цикла трактуется как блок, поэтому метки, описанные в теле цикла локализованы в нем. Переход на метку внутри оператора цикла извне оператора цикла запрещен.

Операторы цикла, управляющие верхними индексами. Как уже указывалось в п. 15.3.1 (верхние индексы), переменные с верхним индексом обозначают значения, образующие рекуррентную временну́ю последовательность. Текущее значение верхнего индекса указывает номер члена последовательности. Описание переменной с верхним индексом задает для позиции верхнего индекса номер начального члена рекуррентной последовательности N, длину рекурсии L (т. е. количество членов последовательности, необходимое для нахождения нового члена последовательности) и метку оператора цикла, управляющего данной позицией. С позицией верхнего индекса связывается понятие времени t, определенное в блоке, в котором описана данная переменная с верхним индексом. В момент входа в блок время для всех верхних индексов переменных, описанных в данном блоке, равно нулю. Время t, сопоставляемое некоторой позиции верхнего индекса, может изменяться только в процессе выполнения оператора цикла, управляющего данной позицией. По окончании очередного выполнения оператора S, повторяемого управляющим оператором цикла, время: увеличивается на единицу, если заголовок цикла не исчерпан; становится равным нулю, если заголовок цикла исчерпан, а в позиции верхнего индекса стоит арифметическое выражение; и сохраняет свое значение, если заголовок цикла исчерпан, а в позиции верхнего индекса стоит ⟨пусто⟩.

Текущее значение F верхнего индекса, в позиции которого стоит арифметическое выражение с текущим значением E, определено для переменных, находящихся вовне управляющего оператора цикла, только тогда, когда

$$N+t\leqslant E\leqslant N+L-1+t,$$

и для переменных, находящихся внутри управляющего оператора цикла, когда

$$N+t\leqslant E\leqslant N+L+t.$$

В этом случае $F=E$.

Текущее значение F верхнего индекса, в позиции которого стоит ⟨пусто⟩, определено только тогда, когда соответствующая переменная используется в операторе, выполняющемся динамически после исчерпания заголовка цикла в операторе цикла, управляющего данной позицией. В последнем случае $F=N+L+t-1$. Это значит, что пустая позиция верхнего индекса обозначает взятие последнего по времени значения верхнего индекса в данной позиции.

15.4.7. Операторы процедуры.

Синтаксис.

⟨фактический параметр⟩::=⟨строка⟩|⟨выражение⟩|⟨идентификатор массива⟩|⟨идентификатор процедуры⟩|⟨идентификатор переключателя⟩|⟨идентификатор функции⟩

⟨строка букв⟩::=⟨буква⟩|⟨строка букв⟩ ⟨буква⟩

⟨ограничитель параметра⟩::=,|) ⟨строка букв⟩:(

⟨список фактических параметров⟩::=⟨фактический параметр⟩|⟨список фактических параметров⟩ ⟨ограничитель параметра⟩ ⟨фактический параметр⟩

⟨совокупность фактических параметров⟩::=⟨пусто⟩|(⟨список фактических параметров⟩)

⟨оператор процедуры⟩::=⟨идентификатор процедуры⟩ ⟨совокупность фактических параметров⟩

Примеры.

След (A) порядок: (7) результат: (v)
транспонирование (w, $v+1$)
абсмакс (A, N, M, Yy, J, k)
скалярное произведение (A [t, p, u], B [p], *10*, p, V)

Эти примеры соответствуют примерам, данным в (п. 15.5.4).

Семантика. Оператор процедуры служит для обеспечения выполнения тела процедуры (п. 15.5.4). В том случае, когда тело процедуры является оператором, записанным на языке альфа-6, результат его выполнения будет эквивалентен результату осуществления следующих действий в программе.

1) Присваивание значений (вызов значением). Всем формальным параметрам, перечисленным в списке значений заголовка описания процедуры, присваиваются значения (п. 15.2.8) соответствующих фактических параметров. Эти присваивания следует рассматривать как выполняемые непосредственно перед входом в тело процедуры. Это происходит так, как будто создается объемлющий тело процедуры дополнительный блок, в котором делаются присваивания переменным, локальным в

этом фиктивном блоке и имеющим типы, заданные соответствующими спецификациями (п. 15.5.4, спецификации). В результате переменные, вызываемые значением, следует рассматривать как не локальные в теле процедуры, но локальные в этом фиктивном блоке (п. 15.5.4, семантика). Значения всех формальных параметров процедуры, перечисленных в списке результатов, будут присвоены соответствующим фактическим параметрам. Это следует понимать таким образом, что между телом процедуры и скобкой **конец** фиктивного блока, объемлющего тело процедуры, вставляются соответствующие операторы присваивания. Эти формальные параметры также будут рассматриваться как локальные в упомянутом фиктивном блоке.

2) Замена наименований (вызов по наименованию). Любой формальный параметр не перечисленный в списке значений и списке результатов, всюду в теле процедуры заменяется на соответствующий фактический параметр после заключения последнего там, где это синтаксически возможно, в скобки. Возможность противоречий между идентификаторами, вставляемыми в тело процедуры в результате такого процесса, и идентификаторами, уже присутствующими в теле процедуры, устраняется соответствующими систематическими изменениями формальных или локальных идентификаторов, затронутых такими противоречиями.

3) Подстановка и выполнение тела процедуры. Наконец, тело процедуры, преобразованное так, как это было описано выше, помещается на место оператора процедуры и выполняется. Если обращение к процедуре производится извне области действия любой величины, не локальной в теле процедуры, то противоречия между идентификаторами, включенными посредством этого процесса подстановки тела, и идентификаторами, описания которых имеют силу там, где расположен оператор процедуры или указатель функции, устраняются посредством соответствующих систематических изменений последних идентификаторов.

Соответствие между формальными и фактическими параметрами. Соответствие между фактическими параметрами оператора процедуры и формальными параметрами заголовка процедуры устанавливается следующим образом. Список фактических параметров оператора процедуры должен иметь то же число членов, что и список формальных параметров заголовка описания процедуры. Соответствие получается путем сопоставления членов этих двух списков в одном и том же порядке.

Ограничения. Чтобы оператор был определен, необходимо, чтобы действия над телом процедуры, определенные в п. 15.4.7 (семантика 1), 2)) приводили бы к правильному оператору в языке альфа-6. Это накладывает на любой оператор процедуры ограничение, заключающееся в том, что класс, структура и тип каждого фактического параметра должен быть совместим с классом, структурой и типом соответствующего формального параметра. Некоторые важные частные случаи этого общего правила приведены ниже.

а) Строка может использоваться только как фактический параметр процедуры. Формальный параметр, специфицированный как **строка**, может использоваться в качестве фактического параметра в операторе обращения к другой процедуре (в частности, в операторе вывода) или в операторе *беши*.

б) Формальному параметру, не вызываемому значением и встречающемуся в теле процедуры в виде переменной левой части некоторого оператора присваивания, может соответствовать в качестве фактического параметра только переменная (частный случай выражения).

в) Формальному параметру, используемому в теле процедуры в качестве идентификатора массива, может соответствовать в качестве фактического параметра только идентификатор массива той же самой внешней размерности. Кроме того, если формальный параметр вызывается значением, то локальный массив, возникающий в теле процедуры во время обращения, получит те же самые границы индексов, что и фактический массив.

г) Формальному параметру, вызываемому значением, не может, вообще говоря, соответствовать какой-либо идентификатор переключателя, функции или процедуры или строка, так как последние не обладают значениями (исключение составляет идентификатор такой функции, описание которой имеет пустую совокупность формальных параметров (п. 15.5.5, синтаксис) и которая определяет значение указателя функции (п. 15.5.5, значения указателей функции). Такой идентификатор функции сам по себе является законченным выражением).

д) Любой формальный параметр может налагать ограничения на тип и структуру соответствующего связанного с ним фактического параметра (эти ограничения могут быть как указаны, так и не указаны посредством спецификаций в заголовке процедуры). Очевидно, что в операторе процедуры эти ограничения должны быть соблюдены.

е) Элемент упакованного логического массива не может быть фактическим параметром процедуры (функции). Если эта

процедура (функция) внешняя, то и идентификатор упакованного массива не может быть фактическим параметром.

Ограничители параметров. Все ограничители параметров считаются эквивалентными. Не устанавливается никакого соответствия между ограничителями параметров, используемыми в операторе, процедуры, и ограничителями, фигурирующими в заголовке процедуры, кроме лишь того, что их количество должно быть одинаковым. Таким образом, вся информация, которая вносится употреблением сложных ограничителей, полностью избыточна.

15.4.8. Оператор останова.

Синтаксис.

⟨оператор останова⟩::=**стоп**

Пример.

начало программы: **стоп**

Семантика. Оператор останова вызывает прекращение выполнения программы. Преемник оператора останова не определяется.

15.4.9. Оператор разметки.

Синтаксис.

⟨оператор разметки⟩::=⟨установка разметки⟩|⟨выдача разметки⟩|⟨установка шаблона⟩|⟨задание ширины листа⟩

⟨установка шаблона⟩::=⟨открыть шаблон⟩; ⟨последовательность операторов задания шаблона⟩ ⟨закрыть шаблон⟩

⟨последовательность операторов задания шаблона⟩::=⟨пусто⟩|⟨оператор⟩; ⟨последовательность операторов задания шаблона⟩

⟨установка разметки⟩::=*marg* (⟨номер канала⟩, ⟨размер левого поля⟩, ⟨длина печатаемой строки⟩, ⟨размер правого поля⟩, ⟨размер верхнего поля⟩, ⟨число печатаемых на странице строк⟩, ⟨размер нижнего поля⟩, ⟨позиция для номера страницы⟩)

⟨позиция для номера страницы⟩::=, ⟨начальный номер страницы⟩|⟨пусто⟩

⟨размер левого поля⟩::=⟨арифметическое выражение⟩

⟨размер правого поля⟩::=⟨арифметическое выражение⟩

⟨длина печатаемой строки⟩::=⟨арифметическое выражение⟩

⟨размер верхнего поля⟩::=⟨арифметическое выражение⟩

⟨число печатаемых на странице строк⟩::=⟨арифметическое выражение⟩

⟨размер нижнего поля⟩::=⟨арифметическое выражение⟩

⟨начальный номер страницы⟩::=⟨арифметическое выражение⟩

⟨выдача разметки⟩:: = *lmarg* (⟨номер канала⟩, ⟨левое поле⟩, ⟨длина строки⟩, ⟨правое поле⟩, ⟨верхнее поле⟩, ⟨число строк⟩, ⟨нижнее поле⟩)
⟨левое поле⟩:: = ⟨переменная⟩
⟨длина строки⟩:: = ⟨переменная⟩
⟨правое поле⟩:: = ⟨переменная⟩
⟨верхнее поле⟩:: = ⟨переменная⟩
⟨число строк⟩:: = ⟨переменная⟩
⟨нижнее поле⟩:: = ⟨переменная⟩
⟨открыть шаблон⟩:: = *откшаб* (⟨номер канала⟩)
⟨закрыть шаблон⟩:: = *закшаб* (⟨номер канала⟩)
⟨задания ширины листа⟩:: = *креж* (⟨ширина листа⟩)
⟨ширина листа⟩:: = ⟨арифметическое выражение⟩
⟨номер канала⟩:: = ⟨арифметическое выражение⟩

Примеры.

процедура *P*;
начало целый левоеполе, длинастроки, правоеполе, верхнееполе, числострок, нижнееполе;
lmarg (0, левоеполе, длинастроки, правоеполе, верхнееполе, числострок, нижнееполе);
marg (0, 10, 108, 10, 4, 100, 4); *output* (0, '×');
примечание: оператор *lmarg* запоминает в соответствующих переменных величины, определяющие разметку, которая была до входа в процедуру. Оператор *marg* устанавливает разметку, необходимую для данной процедуры, а оператор *output* обеспечивает переход на следующую страницу, с новой разметкой;
началовычислений:
. . . . . .
конецвычислений:
ma g (*0*, левоеполе, длинастроки, правоеполе, верхнее поле, числострок, нижнееполе);
output (*0*, '×');
примечание: оператор *marg* перед выходом из процедуры восстанавливает ту разметку, которая была до входа в процедуру, а оператор *output* обеспечивает переход на следующую страницу, со старой разметкой;
конец: процедуры *P*;

Семантика. Вся информация, выводимая с помощью оператора *output*, размещается на АЦПУ в постраничном виде (см. рис. 2.6), причем выводимая информация заполняет только внутреннее поле. Это размещение для каждого канала (п. 15.4.15) характеризуется шестью величинами, которые на-

зываются разметкой страницы:

размер левого поля (*x1*),
длина печатаемой страницы (*x2*),
размер правого поля (*x3*),
размер верхнего поля (*x4*),
число строк внутреннего поля (*x5*),
размер нижнего поля (*x6*).

Шириной страницы будем называть сумму размеров левого поля, правого поля и длины строки внутреннего поля: $x1+x2+x3$.

Длиной страницы будем называть сумму размеров верхнего поля, нижнего поля и числа строк внутреннего поля: $x4+x5+x6$.

При выполнении оператора *lmarg* перечисленным в нем скалярным переменным присваиваются значения указанных выше шести величин в том порядке, в каком они выписаны. Для изменения указанных стандартных размеров служит оператор *marg*. После выполнения этого оператора шесть параметров, определяющих размеры страницы, становятся равными значениям фактических параметров (со второго по седьмой) процедуры *marg*. Соответствие между параметрами разметки и фактическими параметрами процедуры *marg* сохраняется таким же, как и в случае процедуры *lmarg*.

Восьмой фактический параметр процедуры *marg* задает начальное значение номера страницы. Если этого параметра нет, то нумерация страниц начинается с 1. Если пятый фактический параметр $\leqslant 2$, то на странице, выводимой на АЦПУ, будет отсутствовать также строка, состоящая из 128 символов надчеркивания, ограничивающих страницу разметки. Если в программе не задан оператор *marg*, то устанавливается стандартная разметка: $x1=10$, $x2=108$, $x3=10$, $x4=4$, $x5=60$, $x6=3$.

Расположение страниц на листе АЦПУ зависит как от размеров страницы, так и от размеров листа АЦПУ. Лист АЦПУ — это двумерное поле со стандартной шириной в 128 позиций, неограниченное по длине. Если $128/x<2$ (где x — ширина страницы), то страницы располагаются последовательно одна за другой (т. е. сверху вниз).

Если же $128/x\geqslant 2$, то при одной и той же разметке для всех выводимых страниц они будут располагаться по правилу «слева направо сверху вниз». При этом количество страниц, распечатываемых в один ряд, равно *entier* $(128/x)$. В случае, когда размеры страниц не одинаковые, то дополнительно к

правилу «слева направо сверху вниз» необходимо учитывать правило упаковки страниц, которое состоит в следующем:

1) длина любой страницы, размещенной справа, меньше либо равна длине страницы, размещенной слева;

2) страница располагается так, что ее верхняя и левая границы примыкают к левому краю листа АЦПУ или к предыдущим страницам;

3) если перенумеровать подряд все выводимые страницы, то страница с большим номером не может располагаться выше страницы с меньшим номером.

Стандартная процедура *креж* задает новое значение ширины листа АЦПУ. Аргумент *креж* — выражение типа целый задает ширину листа АЦПУ, которая должна быть >0 и $\leqslant 128$.

Установка шаблона задает начальное заполнение страниц, одинаковое для всех страниц, задаваемых текущей разметкой. Начальное заполнение страниц осуществляется операторами *output*, выполняемыми после оператора *откшаб* и до *закшаб*. Причем, для этих операторов *output* доступны все области страницы, т. е. если текущая разметка определяется величинами (*x1*, *x2*, ..., *x6*), то действие операторов *output* осуществляется в разметке (*0*, *x1*+*x2*+*x3*, *0*, *0*, *x4*+*x5*+*x6*, *0*).

В последовательности операторов задания шаблона не должны появляться операторы *marg* и те операторы *output*, которые задают переход на новую страницу.

Новая установка шаблона отменяет старую. Оператор *marg* также отменяет текущую установку шаблона.

15.4.10. Оператор вывода.

Синтаксис.

⟨оператор вывода⟩::=*output* (⟨номер канала⟩, ⟨список элементов вывода с размещением⟩)

⟨список элементов вывода с размещением⟩::=⟨элемент вывода с размещением⟩|⟨список элементов вывода с размещением⟩, ⟨элемент вывода с размещением⟩

⟨элемент вывода с размещением⟩::=⟨элемент размещения⟩|⟨элемент числового вывода⟩|⟨элемент логического вывода⟩|⟨элемент вывода текста⟩|⟨элемент восьмеричного вывода⟩|⟨элемент вывода номера страницы⟩

⟨элемент размещения⟩::=⟨общий формат размещения⟩|⟨общий формат размещения⟩, ⟨кратность элемента⟩

⟨элемент числового вывода⟩::=⟨общий числовой формат⟩, ⟨список объектов вывода⟩

⟨элемент логического вывода⟩::=⟨общий логический формат⟩, ⟨список объектов вывода⟩

⟨элемент вывода текста⟩::=⟨общий текстовый формат⟩, ⟨текстовый объект⟩|⟨общий текстовый формат⟩, ⟨текстовый объект⟩, ⟨кратность элемента⟩

⟨элемент восьмеричного вывода⟩::=⟨общий восьмеричный формат⟩, ⟨список объектов вывода⟩

⟨элемент вывода номера страницы⟩::='⟨формат вывода номера страницы⟩'|⟨неявный формат⟩

⟨кратность элемента⟩::=⟨арифметическое выражение⟩

⟨список объектов вывода⟩::=⟨объект вывода⟩|⟨список объектов вывода⟩, ⟨объект вывода⟩

⟨объект вывода⟩::=⟨выражение⟩|⟨идентификатор массива⟩

⟨общий формат размещения⟩::='⟨формат размещения⟩'|⟨неявный формат⟩

⟨общий числовой формат⟩::='⟨числовой формат⟩'|⟨неявный формат⟩

⟨общий логический формат⟩::='⟨логический формат⟩'|⟨неявный формат⟩

⟨общий текстовый формат⟩::='⟨текстовый формат⟩'|⟨неявный формат⟩

⟨общий восьмеричный формат⟩::='⟨восьмеричный формат⟩'|⟨неявный формат⟩

⟨текстовый объект⟩::=⟨строка⟩|⟨переменная с индексами⟩|⟨формальный параметр⟩

⟨неявный формат⟩::=⟨переменная с индексами⟩|⟨формальный параметр⟩

⟨формат вывода номера страницы⟩::=*s* ⟨числовой формат⟩

Примеры.

output (*0*, '*e*', *a*, *b*, *c*+*k*, *A*)
output (*1*, '*l16в5ј6в*', ЛОГ)
output (*0*, '*т*', 'ПЕЧАТАТЬ МЕТКИ')
output (*0*, '*/*')
outpyt (*0*, '*/*', '*т*', 'массив_*A*', '*e*', *A*, '×')

Семантика. Операторы вывода задают вывод числовых, логических или текстовых данных через канал, номер которого определяется первым фактическим параметром. О каналах см. в п. 15.4.15.

Выводимая информация размещается последовательно на строках внутреннего поля (см. рис. 2.6). При заполнении всех строк внутреннего поля осуществляется переход к началу следующей страницы с той же разметкой. Здесь и ниже при упоминании номера строки и номера позиции в строке будет пониматься (если не оговорено особо) номер строки и номер позиции в строке на внутреннем поле страницы. Термин

«переход к началу следующей страницы» везде понимается как переход к первой позиции первой строки внутреннего поля следующей страницы.

Числовой, логический, текстовый и восьмеричный форматы определяют вид, в котором выводятся указанные вслед за форматом объекты. Все объекты одного списка вывода выдаются в одном и том же формате.

Формат размещения служит для указания размещения последующих единиц вывода информации на странице. Формат вывода номера страницы служит для вывода номера текущей страницы.

Формат может быть задан явно, в виде строки, либо неявно, в виде переменной с индексами типа целый размерности ноль или в виде формального параметра. Если формат является переменной с индексами, то она указывает на элемент массива, начиная с которого располагаются целые числа, соответствующие последовательным символам формата (соответствие между символами формата и целыми числами см. в Приложении 7). Предполагается, что указанный элемент массива соответствует открывающей кавычке, конец формата определяется элементом массива, соответствующим закрывающей кавычке.

Если в операторе *output* имеется элемент числового вывода или элемент восьмеричного вывода, то следующий за ним элемент вывода не может иметь формат — переменную с индексом.

Формальный параметр может употребляться в качестве формата только в том случае, когда ему соответствует фактический параметр — строка.

Кратность элемента задает число повторений вывода текстового объекта или формата размещения. Кратность элемента может быть любым арифметическим выражением типа целый размерности ноль, за исключением переменной с индексами.

Объектом числового, логического и восьмеричного вывода может быть выражение любой размерности или идентификатор массива. Если объект вывода — идентификатор массива или выражение ненулевой размерности, то выводятся все его элементы. Объекты вывода, имеющие тип целый, вещественный или комплексный, должны выводиться в числовом или восьмеричном формате. Объекты вывода, имеющие тип логический, должны выводиться в логическом или восьмеричном формате.

Текстовым объектом вывода является строка, переменная с индексом типа целый нулевой размерности или формальный параметр. Если текстовый объект вывода есть строка, то вы-

дается эта строка без внешних кавычек. Если текстовый объект есть переменная с индексами, то она указывает на элемент массива, начиная с которого располагаются целые числа, соответствующие последовательным символам выводимой строки (см. Приложение 7). Предполагается, что указанный элемент массива соответствует открывающей кавычке, которая не будет выведена. Конец строки определяется элементом массива, соответствующим закрывающей кавычке, которая также не выводится.

Следует обратить внимание, что отдельные символы строки кодируются несколькими последовательными числами.

Формальный параметр может употребляться в качестве текстового объекта вывода лишь в том случае, когда ему соответствует фактический параметр-строка (15.4.7, 15.5.5).

Формат размещения.

Синтаксис.

⟨компонента размещения⟩::=/|×| ⟨шаговый сдвиг курсора⟩| ⟨запись курсора⟩|⟨восстановление курсора⟩|⟨сдвиг курсора на границу⟩|⟨сдвиг внутренней границы⟩| ⟨сдвиг луча⟩ | ⟨сдвиг лучевого поля⟩

⟨шаговый сдвиг курсора⟩::=*в*|*ю*|*р*|*п*

⟨запись курсора⟩::=*з*

⟨восстановление курсора⟩::=↑

⟨сдвиг курсора на границу⟩::=>|<|∧|∨

⟨сдвиг внутренней границы⟩::=|>||<||∧||∨

⟨сдвиг луча⟩::=⟨сдвиг луча от курсора⟩|⟨сдвиг луча к курсору⟩

⟨сдвиг луча от курсора⟩::=−>|−<|−∧|−∨

⟨сдвиг луча к курсору⟩::=+>|+<|+∧|+∨

⟨сдвиг лучевого поля⟩::=⟨сдвиг поля от курсора⟩|⟨сдвиг поля к курсору⟩

⟨сдвиг поля от курсора⟩::=(−>)|(−<)|(−∧)|(−∨)

⟨сдвиг поля к курсору⟩::=(+>)|(+<)|(+∧)|(+∨)

⟨формат размещения⟩::=⟨повторитель⟩ ⟨компонента размещения⟩|⟨формат размещения⟩ ⟨повторитель⟩ ⟨компонента размещения⟩

⟨повторитель⟩::=⟨целое без знака⟩|⟨пусто⟩

Семантика. Конструкция ⟨повторитель⟩ ⟨компонента размещения⟩ эквивалентна соответствующее число раз повторенной компоненте размещения, например, *3*+*р* эквивалентно +*р*+*р*+*р*. Назовем курсором место на странице, куда будет размещен очередной выводимый символ. Значением курсора являются координаты (*x*, *y*) этого места на странице относи-

тельно первой позиции первой строки верхнего поля страницы (см. рис. 2.6.), где x — номер позиции, y — номер строки. Если задана разметка ($x1$, $x2$, $x3$, $x4$, $x5$, $x6$), то значение курсора при выводе ограничено следующими соотношениями:

$$x1 < x \leqslant x1+x2, \quad x4 < y \leqslant x4+x5. \tag{1}$$

При обычном выводе курсор переводится на одну позицию вправо (в пределах внутреннего поля страницы), отмечая первую незаполненную позицию в строке. После вывода символа на правой границе внутреннего поля (т. е. $x=x1+x2$), курсор переводится на следующую строку внутреннего поля (т. е. $x:=x1+1$; $y:=y+1$). Если же строка внутреннего поля была последняя ($y=x4+x5$), то осуществляется переход на начало следующей страницы. Расположением информации на странице можно управлять, меняя значение курсора.

Символы изменения значения курсора следующие:

/ — переход к началу следующей строки: **если** $y \neq x4+x5$ **то начало** $x:=x1+1$; $y:=y+1$ **конец иначе** переход к началу следующей страницы;

$\times$ — переход к началу следующей страницы: $x:=x1+1$; $y:=x4+1$;

$в$ — переход на следующую позицию: **если** $x \neq x1+x2$ **то** $x:=x+1$ **иначе** переход к началу следующей строки;

$ю$ — переход на предыдущую позицию: **если** $x \neq x1+1$ **то** $x:=x-1$ **иначе если** $(y \neq x4+1) \wedge (x=x1+1)$ **то начало** $x:=x1+x2$; $y:=y-1$ **конец**;

$р$ — переход на предыдущую строку: **если** $y \neq x4+1$ **то** $y:=y-1$;

$п$ — переход на следующую строку: **если** $y \neq x4+x5$ **то** $y:=y+1$ **иначе** переход к началу следующей страницы;

$>$ — переход на правую границу внутреннего поля: $x:=x1+x2$;

$<$ — переход на левую границу внутреннего поля: $x:=x1+1$;

$\wedge$ — переход на верхнюю границу внутреннего поля: $y:=x4+1$;

$\vee$ — переход на нижнюю границу внутреннего поля: $y:=x4+x5$.

Компонента $з$ указывает на то, что текущее значение курсора должно быть запомнено. Это значение курсора может быть восстановлено с помощью компоненты ↑. Если значение курсора не запоминалось, то компонента ↑ устанавливает курсор на начало текущей страницы ($x:=x1+1$; $y:=x4+1$). Допус-

кается запоминание нескольких значений курсора, которые могут использоваться для последующих восстановлений курсора в качестве его значений в порядке, обратном запоминанию.

Компоненты сдвига внутренней границы сдвигают внутреннюю границу к текущему расположению курсора, изменяя разметку страницы. При этом сам курсор не движется и внешние размеры страницы не изменяются, т. е. ширина страницы и длина страницы остаются неизмененными. Компоненты сдвига внутренних границ следующие:

$|>$ — сдвиг правой границы внутреннего поля: $x2 := x - x1$; $x3 := x1 + x2 + x3 - x$;

$|<$ — сдвиг левой границы внутреннего поля: $x1 := x - 1$; $x2 := x1 + x2 - x + 1$;

$|\wedge$ — сдвиг верхней границы внутреннего поля: $x4 := y - 1$; $x5 := x4 + x5 - y + 1$;

$|\vee$ — сдвиг нижней границы внутреннего поля: $x5 := y - x4$; $x6 := x4 + x5 + x6 - y$.

Возврат к прежней разметке возможен и осуществляется с помощью компоненты $\uparrow$. Для этого нужно перед сдвигом внутренней границы запомнить значение курсора на этой границе. Если при восстановлении запомненного значения курсора не выполняются указанные выше соотношения (1), то восстановление курсора, сопровождается изменением размеров внутреннего поля по правилам сдвига внутренних границ. А именно: если $x \leqslant x1$, то сдвигается левая граница, если $x > x1 + x2$, то сдвигается правая граница, если $y \leqslant x4$, то сдвигается верхняя граница, и, наконец, если $y > x4 + x5$, то сдвигается нижняя граница.

Назовем *лучом* часть строки, на которой находится курсор, расположенную справа или слева от курсора (не включая позицию курсора); либо часть столбца, на котором находится курсор, расположенную сверху или снизу от курсора, не включая позицию курсора. В каждом из этих случаев луч называется правым, левым, верхним или нижним соответственно.

Сдвинуть правый луч на одну позицию вправо — это значит переместить все символы луча на их соседние позиции вправо, позицию после курсора заполнить пробелом, а выдвинутый (за правую границу внутреннего поля) символ убрать. Сдвинуть правый луч влево на одну позицию — это значит переместить все символы луча на их соседние позиции влево, позицию на правой границе внутреннего поля заполнить пробелом, а выдвинутый символ на позицию курсора убрать (в позиции курсора оставить тот же символ, какой был до сдвига). Ана-

логично определяются сдвиги левого луча (вправо и влево), верхнего и нижнего лучей (вверх и вниз). В компоненте сдвига луча знак задает направление сдвига, — от курсора, + к курсору, а символы $>$, $<$, $\wedge$, $\vee$ указывают, какой луч сдвигается: правый, левый, верхний или нижний соответственно.

Назовем правым лучевым полем часть страницы, которая находится между правой внутренней границей и столбцом, на котором находится курсор, включая этот столбец. Аналогично определяется левое лучевое поле, верхнее лучевое поле и нижнее.

Сдвиги полей от курсора и к курсору определяются так же как и сдвиги лучей. Например, при сдвиге правого лучевого поля от курсора все его символы перемещаются на соседнюю правую позицию, выдвигающиеся символы за правую внутреннюю границу пропадают, а самый левый столбец заполняется пробелами. Задание сдвигов лучевых полей осуществляется аналогично заданию сдвигов лучей, при этом знак означает направление сдвига (— от курсора, + к курсору), а символы $>$, $<$, $\wedge$, $\vee$ определяют лучевое поле: правое, левое, верхнее и нижнее соответственно.

Пример 1.

Пусть длина выводимой строки равна 60, а число строк, помещающихся на одной странице — 40. Предположим, что курсор находится в начале страницы (т. е. на эту страницу не был выведен ни один объект вывода и не выполнялся ни один оператор размещения), тогда переход к началу следующей страницы можно осуществить одним из следующих операторов размещения:

output (*0*, '$\times$')
output (*0*, '40/')
output (*0*, '19/60*в*', 2)

Пример 2.

Оператор размещения

output (*0*, '2/19*в*', 2)

указывает на то, что информацию надо располагать с 20 позиции 3-ей строки, начиная с новой страницы.

Числовой формат.

Синтаксис.

⟨повторитель⟩::=⟨целое без знака⟩|⟨пусто⟩

⟨*B*-часть⟩::=⟨пусто⟩ | ⟨повторитель⟩ *в* |⟨*B*-часть⟩ ⟨повторитель⟩ *в*

⟨*D*-часть⟩::=⟨пусто⟩ | ⟨повторитель⟩ *d* | ⟨*D*-часть⟩ ⟨повторитель⟩ *d* | ⟨*D*-часть⟩ ⟨*B*-часть⟩

⟨целый формат⟩::=⟨*B*-часть⟩ ⟨знаковая часть⟩ ⟨*D*-часть⟩
⟨дробный формат⟩::=⟨целый формат⟩|⟨целый формат⟩.⟨*D*-часть⟩
⟨экспоненциальный формат⟩::=⟨дробный формат⟩$_{10}$⟨целый формат⟩
⟨знаковая часть⟩::=+|−|⟨пусто⟩
⟨числовой формат⟩::=*e*|*y*|*z*|*yi*|*zi*|*e* ⟨экспоненциальный формат⟩| *y* ⟨дробный формат⟩ | *z* ⟨дробный формат⟩

Семантика. Числовой формат определяет вид, в котором выводятся числа (в десятичной системе счисления). Числа перед выводом округляются.

Буквы в формате обозначают:

d — десятичную цифру выводимого числа,

в — пропуск (интервал) между выводимыми символами,

z — замену незначащих нулей в выводимом числе пропусками (так называемое подавление нулей),

y — вывод без подавления нулей, т. е. вывод такого количества цифр, какое указано в формате,

e — вывод числа в экспоненциальной форме с отличной от нуля первой цифрой (если число не равно нулю).

Символы в формате обозначают:

точка — вывод десятичной точки в числе,

\+ — вывод знака числа (плюс, если число положительное и минус, если отрицательное),

— — вывод знака только у отрицательных чисел.

Знак числа помещается на место последнего подавленного нуля выводимого числа. Если знаковая часть пуста, то знак не печатается (выводится абсолютное значение числа).

Если в дробном формате с подавлением нулей перед десятичной точкой нет значащих цифр (или число, выводимое в целом формате, оказывается равным нулю), то сохраняется один нуль. Конструкция ⟨повторитель⟩ *в* или ⟨повторитель⟩ *d* эквивалентна соответствующее число раз повторенной букве *в* или *d*. Так, например, *5в* эквивалентно *ввввв*, *4d.3d* эквивалентно *dddd.ddd*.

Числовые форматы вида *e*, *y*, *z*, *yl*, *zl* обозначают вывод чисел в стандартной форме и эквивалентны выводам со следующими числовыми форматами:

e эквивалентно e — $10d_{10}+2d2в$
y эквивалентно *y* — *6d.6d4в*
z эквивалентно *z* — *6d.6d4в*

yl эквивалентно *y — 6dв6d4в*

zi эквивалентно *6dв6d4в*

Примеры числовых форматов. Требуется вывести массив $A = (10.37412, 17.92421, -25.34193)$. Это можно осуществить следующими способами.

Оператор вывода

output (0, 'y5в + 3d.3d', A)

выдает массив *A* в виде:

␣␣␣␣␣+010.374␣␣␣␣␣ + 017.924 ␣␣␣␣␣−025.342

Оператор вывода

output (0, 'e5в — d.5d$_{10}$ *+ 2d', A)*

выдает массив *A* в виде:

␣␣␣␣␣1.03741$_{10}$+01 ␣␣␣␣␣ 1.7924$_{10}$+
01␣␣␣␣␣−2.53419$_{10}$+01

Оператор вывода

output (0, 'z5в+3d.3d', A)

выводит массив *A* в виде:

␣␣␣␣␣+10.374␣␣␣␣␣+ 17.924␣␣␣␣␣−25.342

Оператор вывода

output (0, 'z — d5в', A)

выводит массив *A* в виде:

␣0␣␣␣␣␣␣8␣␣␣␣␣␣␣5␣␣␣␣␣␣

Логический формат.

Синтаксис.

⟨*F*-часть⟩::=⟨пусто⟩|⟨повторитель⟩*f*|⟨*F*-часть⟩ ⟨повторитель⟩ *f*|⟨*F*-часть⟩ ⟨*B*-часть⟩

⟨логический формат⟩::=*l*|*l* ⟨*F*-часть⟩

Семантика. Формат *l* означает вывод в стандартной форме, т. е. единицей, если выводимое значение — **истина** или нулем, если выводимое значение есть **ложь**.

⟨*F*-часть⟩ задает вывод логического значения в виде слов TRUE или FALSE при этом число букв *F* с учетом повторителей указывает, сколько букв слова надо печатать. В этом случае, если число букв *F* с учетом повторителей больше длины слова (TRUE или FALSE), то недостающие символы дополняются пробелами справа.

Примеры логических форматов. Пусть переменные *Л*, *B*, *C* имеют значения **истина**, **ложь**, **ложь** соответ-

ственно. Тогда оператор вывода

output (*0*, '*l5в f4в*', *Л*, *B*, *C*)

выведет логические значения *Л*, *B*, *C* в виде:

␣␣␣␣␣T␣␣␣␣␣␣␣␣␣F␣␣␣␣␣␣␣␣␣ F

Оператор вывода

output (*0*, '*l5в5f5в*', *Л*, *B*, *C*)

выведет логические значения *Л*, *B* и *C* в виде:

␣␣␣␣␣FALSE␣␣␣␣␣␣␣␣␣␣TRUE

␣␣␣␣␣␣␣␣␣␣␣FALSE

Текстовый формат.

Синтаксис.

⟨текстовый формат⟩:: = *т* ⟨*A*-часть⟩

⟨*A*-часть⟩:: = ⟨пусто⟩ | ⟨повторитель⟩ *а* | ⟨*A*-часть⟩ ⟨повторитель⟩ *а* | ⟨*A*-часть⟩ ⟨*B*-часть⟩

Семантика. Задание текстового формата позволяет печатать текст вразрядку с нужным числом пробелов между символами строки. Буква *а* в текстовом формате задает вывод символа строки, буква *в* — пропуск (интервал) между символами. Общее число *а* и *в* в текстовом формате, с учетом повторителей, назовем интервалом печати.

Если при выводе текста количество позиций до конца строки страницы меньше интервала печати, то последняя часть текста распечатывается с новой строки. Если число символов в строке не кратно количеству *а* в формате, с учетом повторителей, то при выводе этой строки недостающие символы дополняются пробелами.

В выводимых строках двоеточие используется для изменения смысла следующего за ним символа. А именно, комбинация

:1 — задает вывод символа ';

:2 — вывод символа ';

:3 — вывод символа :;

:4 — вывод предыдущего символа столько раз, сколько закодировано следующим за комбинацией символом (кодировку символов см. в таблице П7.1);

:5 — указывает переход к позиции текущей строки, номер которой закодирован следующим за комбинацией символом;

:6 — задает вывод символа %;

:7 — вывод символа ◇;

:8 — вывод символа !.

Комбинации *:/*, *:×*, *:в*, *:ю*, *:р*, *:п* задают изменение значения курсора и эквивалентны по действию форматам размещения: '/', '×', '*в*', '*ю*', '*р*' и '*п*' соответственно (п. 15.4.10).

Пример. Каждый из операторов

output (*0*, '*т*', '—_—_—_Т_А_Б_Л_И_Ц_А_—_—_—')
output (*0*, '*тав*', '— —ТАБЛИЦА— —')

распечатывает текст строки в следующем виде:

—_—_—_Т_А_Б_Л_И_Ц_А_—_—_—

Восьмеричный формат.

Синтаксис.

⟨восьмеричный формат⟩::=*о*⟨*А*-часть⟩

Семантика. Восьмеричный формат задает вывод машинного представления значения в восьмеричном виде. Буква *а* означает вывод восьмеричной цифры, *в* — пропуск (интервал) между цифрами. Общее число букв *а* с учетом повторителей в формате определяет количество младших восьмеричных цифр машинного представления. Если формат задает вывод больше 16 цифр, то недостающие цифры в распечатке дополняются нулями слева.

Пример. Оператор *output* (*0*, '*оав3ав4ав4ав4а*', *А*) позволит выдать значение числа *А* в таком виде:

а_ааа_аааа_аааа_аааа

15.4.11. Оператор ввода.

Синтаксис.

⟨оператор ввода⟩::=*input* (⟨номер канала⟩, ⟨список объектов ввода⟩)
⟨список объектов ввода⟩::=⟨объект ввода⟩|⟨список объектов ввода⟩, ⟨объект ввода⟩
⟨объект ввода⟩::=⟨идентификатор массива⟩|⟨простая переменная⟩|⟨переменная с индексами⟩
⟨номер канала⟩::=⟨арифметическое выражение⟩

Семантика. Оператор ввода задает ввод числовых, логических и текстовых данных через канал (п. 15.4.15), номер которого определяется первым фактическим параметром. Форма, в которой должны быть подготовлены эти данные, определяется в главе 16. Каждому объекту ввода должна соответствовать одна группа данных. Простым переменным и идентификаторам массива соответствуют группы числовых или логических данных. Скалярной переменной типа целый, вещественный или логический соответствует группа данных,

состоящая из одного значения. Комплексной скалярной переменной соответствует группа данных, состоящая из двух значений, первое значение для действительной части, второе — для мнимой. Если группа данных соответствует идентификатору массива (переменной ненулевой размерности), то значения этой группы данных присваиваются элементам массива, начиная с первого, согласно лексикографической упорядоченности элементов массива. Комплексному массиву соответствует группа данных из нескольких пар значений, причем первая пара соответствует первому элементу массива, вторая — второму и т. д.

Если значений в группе данных (или пар значений для комплексного массива) оказывается меньше числа элементов массива, то оставшиеся незаполненными элементы массива будут иметь те же значения, которые они имели до ввода.

Если значений в группе данных (или пар значений для комплексного массива) больше числа элементов массива, то результат выполнения оператора ввода не определен.

Целой переменной с индексами с абсолютной размерностью, равной нулю, должна соответствовать группа текстовых данных. Переменной с абсолютной размерностью, не равной нулю, соответствуют только числовые или логические данные.

При вводе текста последовательным элементам массива, начиная с указанного в объекте ввода, присваиваются целые значения в соответствии с приложением 7. Следует заметить, что некоторые символы строки представляются несколькими элементами массива.

Пример. Пусть *a*, *b*, *c* — скаляры, *ЛОГ* — логический вектор длины 3, *M* — вещественный вектор длины 5, *k* — целый вектор длины 3. Пусть данные имеют следующий вид:

канал *1***a*=*1*; *b*=*2*; *c*=*3.14*; **истина**, **ложь**, **ложь**;
M = *0.1*, *0.2*, *0.3*; '*a*';

Тогда в результате работы оператора

input (*1*, *a*, *b*, *c*, *ЛОГ*, *M*, *k* [*1*])

переменные примут следующие значения:

a = *1*, *b* = *2*, *c* = *3.14*;
ЛОГ = (**истина**, **ложь**, **ложь**)
M [*1*] = *0.1*; *M* [*2*] = *0.2*; *M* [*3*] = *0.3*;
M [*4*] и *M* [*5*] останутся не заданными.
k [*1*] = *26* — код открывающей кавычки,
k [*2*] = *32* — код буквы *a*,
k [*3*] = *27* — код закрывающей кавычки.

15.4.12. Оператор текст.

Синтаксис.

⟨оператор текст⟩::=*text* (⟨строка⟩, ⟨переменная с индексами⟩)

Семантика. Оператор *text* присваивает последовательным элементам массива типа целый, начиная с элемента, указанного вторым фактическим параметром, целые значения, соответствующие последовательным символам строки (включая внешние кавычки). Символы, не представимые одним числом, занимают несколько элементов массива (Приложение 7).

Пример. Пусть A — целый вектор длины 5. В результате выполнения оператора

text ('*abc*', *A* [*1*])

массив A получит следующие значения:

A [*1*] = *26* — код открывающей кавычки,
A [*2*] = *32* — код буквы *a*,
A [*3*] = *33* — код буквы *b*,
A [*4*] = *34* — код буквы *c*,
A [*5*] = *27* — код закрывающей кавычки.

15.4.13. Оператор обмена.

Синтаксис.

⟨оператор обмена⟩::=⟨оператор обмена с МБ⟩|⟨оператор физобмена⟩

⟨оператор обмена с МБ⟩::=*copy* (⟨внешняя переменная с индексами⟩, ⟨идентификатор массива⟩)|*copy* (⟨идентификатор массива⟩, ⟨внешняя переменная с индексами⟩)

⟨оператор физобмена⟩::=*copy* (⟨признак обмена⟩, ⟨идентификатор массива⟩, ⟨номер направления внешнего носителя⟩, ⟨номер устройства⟩, ⟨номер зоны⟩, ⟨относительный адрес ячейки в зоне⟩)

⟨признак обмена⟩::=⟨признак чтения⟩|⟨признак записи⟩

⟨признак чтения⟩::=⟨арифметическое выражение⟩

⟨признак записи⟩::=⟨арифметическое выражение⟩

⟨номер направления внешнего носителя⟩::=⟨арифметическое выражение⟩

⟨номер устройства⟩::=⟨арифметическое выражение⟩

⟨номер зоны⟩::=⟨арифметическое выражение⟩

⟨относительный адрес ячейки в зоне⟩::=⟨арифметическое выражение⟩

Примеры.

copy (*EXA* [*k*+*1*], *A*)
copy (*B*, *EXA* [*k*+*6*])
copy (*1*, *A*, *5*, *1*, *6*, *0*)
copy (*0*, *B*, *4*, *м*, *k*+*1*, *entier* (*a*+*b*))

Семантика. Оператор обмена с МБ служит для обмена между внутренними и внешними массивами. Элементы массива считаются упорядоченными лексикографически по индексам.

Внешняя переменная с индексами задает начальный адрес обмена во внешней памяти. Идентификатор массива всегда внутренний и первый элемент этого массива задает начальный адрес обмена во внутренней памяти, длина этого внутреннего массива определяет объем обмена. Передача значений идет от первого фактического параметра ко второму. Типы внешнего и внутреннего массивов в операторе обмена с МБ должны совпадать.

Оператор физобмена служит для записи (или чтения) массива на внешний носитель: ленту, диск или магнитный барабан. Причем распределение памяти на внешнем носителе возлагается на самого программиста.

Все параметры (кроме второго) могут быть арифметическими выражениями и должны иметь тип целый. Если значение первого параметра равно нулю, то осуществляется запись значений, которые имеет второй параметр, во внешнюю память, в противном случае осуществляется считывание значений из внешней памяти во внутренний массив.

Значение третьего параметра должно находиться в пределах от 1 до 6; значение четвертого параметра — от 0 до 7. Если значение третьего параметра (номер направления внешнего устройства) равно 1 или 2, то обмен осуществляется с МБ. Если значение третьего параметра равно 3, 4, 5 или 6, то обмен осуществляется с лентой или диском.

Запрещается использовать следующие комбинации значений третьего и четвертого параметра соответственно: 01—14, 20—33.

Следует быть осторожным при использовании оператора физобмена для обмена с магнитным барабаном, особенно при наличии в программе внешних массивов, так как при этом возможно наложение внешних массивов друг на друга.

Относительный адрес ячейки в зоне (шестой параметр) оператора физобмена) служит для указания смещения записываемого или читаемого массива относительно начала зоны (пятый параметр). Это смещение может превышать длину зоны, в этом случае обмен будет вестись, начиная

с зоны $з + N \div 1024$, где $з$ — номер зоны, N — относительный адрес ячейки в зоне.

15.4.14. Оператор *бемш.*

Синтаксис.

⟨оператор *бемш*⟩:: = *бемш* (⟨тело оператора *бемш*⟩)

⟨тело оператора *бемш*⟩:: = ⟨команда⟩|⟨константа⟩|⟨тело оператора *бемш*⟩; ⟨команда⟩|⟨тело оператора *бемш*⟩; ⟨константа⟩

⟨команда⟩:: = ⟨непомеченная команда⟩|⟨метка⟩ : ⟨непомеченная команда⟩

⟨непомеченная команда⟩::=⟨код операции⟩, ⟨операнд⟩|⟨код операции⟩, ⟨операнд⟩, ⟨индекс-регистр⟩

⟨код операции⟩::=⟨четв цифра⟩ ⟨восьм цифра⟩ ⟨восьм цифра⟩ | ⟨восьм цифра⟩ ⟨восьм цифра⟩ | ⟨восьм цифра⟩ | ⟨имя команды *бемш*⟩

⟨имя команды *бемш*⟩:: = ⟨буква⟩|⟨буква⟩ ⟨имя команды *бемш*⟩

⟨четв цифра⟩::=0|1|2|3

⟨восьм цифра⟩::=0|1|2|3|4|5|6|7

⟨двоичная цифра⟩::=0|1

⟨индекс-регистр⟩::=⟨восьм цифра⟩|⟨двоичная цифра⟩ ⟨восьм цифра⟩

⟨операнд⟩::=⟨простой операнд⟩|=⟨простая константа⟩

⟨простой операнд⟩::=⟨восьм адрес⟩|⟨простая переменная⟩|⟨идентификатор массива⟩|⟨идентификатор массива⟩ ⟨операция типа сложения⟩ ⟨восьм адрес⟩|⟨метка⟩|⟨метка⟩ ⟨операция типа сложения⟩ ⟨восьм адрес⟩

⟨восьм адрес⟩::=⟨последовательность восьм цифр⟩

⟨последовательность восьм цифр⟩::=⟨восьм цифра⟩|⟨последовательность восьм цифр⟩ ⟨восьм цифра⟩

⟨константа⟩::=⟨непомеченная константа⟩|⟨метка⟩:⟨непомеченная константа⟩

⟨непомеченная константа⟩::=⟨короткая константа⟩|⟨длинная константа⟩

⟨короткая константа⟩:: = *конк* ⟨простая константа⟩ | *конк*, ⟨адресная константа⟩

⟨длинная константа⟩::=*конд*, ⟨простая константа⟩| *конд*, ⟨адресная константа⟩

⟨простая константа⟩::=⟨составная компонента константы⟩|⟨простая константа⟩ ⟨составная компонента константы⟩

⟨составная компонента константы⟩::=⟨тело константы⟩|⟨сдвиг⟩ ⟨тело константы⟩

⟨сдвиг⟩::=*м* ⟨цифра⟩|*м* ⟨цифра⟩ ⟨цифра⟩
⟨тело константы⟩::=*в* '⟨восьм константа⟩'| *т* '⟨текстовый элемент константы⟩'| *в* '— ⟨восьм константа⟩'
⟨восьм константа⟩::=⟨последовательность восьм цифр⟩
⟨текстовый элемент константы⟩::=⟨последовательность не более шести основных символов не содержащая кавычек⟩
⟨последовательность не более шести основных символов не содержащая кавычек⟩::=⟨основной символ⟩|⟨основной символ⟩ ⟨последовательность не более шести основных символов не содержащая кавычек⟩
⟨адресная константа⟩::=*а* (⟨простой операнд⟩)| *а* (⟨простой операнд⟩ (⟨индекс-регистр⟩))

Примеры.

бемш (*зп*, *а*, *13*; *пб*, *M1*; *M2*: *конк*, *а* (*M2*))
бемш (*010*, =*в* '*1*'; *012*, *Б*; *260*, *цикл*; *слиа*, *1*, *5*)

Семантика. Оператор *бемш* позволяет непосредственно использовать систему команд машины БЭСМ-6. Для обозначения машинных команд принята мнемоника автокода *бемш* [15]. Таблица имен команд оператора *бемш* дается в Приложении 13.

Команды, соответствующие оператору *бемш*, вставляются в программу на место оператора *бемш*. Каждая команда оператора *бемш* занимает после трансляции ровно половину ячейки БЭСМ-6 и помещается вслед за предыдущей командой. Если команда помечена меткой и должна быть размещена в правой половине ячейки, то эта половина заполняется командой 002200000, а помеченная команда располагается в левой половине следующей ячейки. В операторе *бемш* запрещается использование целого без знака в качестве метки.

Восьмеричная константа в представлении тела константы не может состоять более чем из 16 восьмеричных цифр.

Идентификаторы простой переменной, массива или метки, являющиеся операндами команд, понимаются соответственно как адреса переменной, адрес первого элемента массива и адрес оператора, помеченного этой меткой.

Восьмеричный адрес не может состоять более чем из 5 восьмеричных цифр.

Литеральная константа, т. е. операнд, взятый в кавычки и начинающийся со знака равенства, заводится транслятором в таблице констант, а в соответствующую команду подставляется адрес этой константы.

Короткая константа занимает половину ячейки. Размещение коротких констант производится аналогично размещению

команд. Таким образом, если мы хотим заполнить ячейку двумя короткими константами, то сначала надо употребить константу, которая должна быть размещена в старших разрядах, т. е. в левой половине ячейки.

Длинная константа занимает всю ячейку.

Если простая константа состоит из нескольких констант, то результирующая константа образуется поразрядным сложением по модулю 2 всех соответствующих компонент.

Тело константы при переводе в машинное (двоичное) представление располагается начиная с младших разрядов. При наличии сдвига, двоичное представление константы сдвигается влево на число разрядов, указанное вслед за *м*.

Например,

конд, *м24в'1'*

определяет константу, имеющую единичку в 25 разряде и нули во всех остальных.

Если в теле константы типа *в* перед восьмеричной константой стоит знак минус, то внутреннее представление этой константы записывается в обратном коде. Например, константе

конд, *в'* — *1'*

будет сопоставлено машинное слово с единицами во всех 48 разрядах. Адресная константа формируется аналогично длинной команде с нулевым кодом операции, базирование [14 стр. 221] в этом случае не производится. Под текстовую константу отводится одно машинное слово, под каждый символ — по 8 разрядов, которые располагаются в ячейке слева направо. Если в текстовой константе меньше 6 символов, то свободное место в слове оставляется слева и заполняется нулями.

Формальные параметры, встречающиеся в операндах команд и адресных констант, должны быть специфированы значением.

15.4.15. Операторы управления каналами.

Синтаксис.

⟨оператор канала⟩:: = ⟨изменение указателя на *N* записей⟩|⟨считывание указателя⟩|⟨установка указателя⟩|⟨коммутация канала⟩|⟨задание ширины листа⟩|⟨задание длины канала⟩|⟨выдача информации о канале⟩|⟨использование записей на листе канала⟩

⟨изменение указателя на *N* записей⟩ ::= *мод* (⟨указатель⟩, ⟨номер канала⟩, ⟨число записей⟩)

⟨указатель⟩ :: = ⟨указатель записи по *output*⟩|⟨указатель чтения по *input*⟩|⟨указатель конца чтения и начала записи⟩

⟨указатель записи по *output*⟩ :: = 0
⟨указатель чтения по *input*⟩:: = 1
⟨указатель конца чтения и начала записи⟩ :: = 2
⟨считывание указателя⟩ :: = *изм* (1, ⟨указатель⟩, ⟨номер канала⟩, ⟨простая переменная⟩)
⟨установка указателя⟩:: = *изм* (0, ⟨указатель⟩, ⟨номер канала⟩, ⟨арифметическое выражение⟩)
⟨коммутация канала⟩:: = *коммут* (⟨номер канала⟩, ⟨номер направления внешнего носителя⟩, ⟨номер устройства⟩)
⟨номер направления внешнего носителя⟩ :: = ⟨арифметическое выражение⟩
⟨номер устройства⟩:: = ⟨арифметическое выражение⟩
⟨задание ширины листа⟩ :: = *режк* (⟨ширина листа⟩)
⟨ширина листа⟩ :: = ⟨арифметическое выражение⟩
⟨задание длины канала⟩ :: = *длинк* (⟨номер канала⟩, ⟨арифметическое выражение⟩)
⟨выдача информации о канале⟩ :: = *инфк* (⟨номер канала⟩, ⟨идентификатор массива⟩)
⟨использование записей на ленте канала⟩ :: = *сохр* (⟨номер канала⟩)
⟨номер канала⟩ :: = ⟨арифметическое выражение⟩
⟨число записей⟩ :: = ⟨арифметическое выражение⟩

Семантика. Канал состоит из последовательности записей (переменной длины), размещенной во внешней памяти (МБ, МЛ, МД). Запись канала — группа данных, предназначенная для ввода по *input*, или совокупность значений выражения или массива, выведенных оператором *output* для последующей распечатки на АЦПУ, или совокупность значений разметки (п. 15.4.9), установленной оператором *marg*.

Канал состоит из двух частей: в первой (начальной) содержатся записи групп данных в том порядке, как они следуют в альфа-данных, во второй (вслед за первой)) записи операторов *output* и *marg*. Канал с номером 0 располагается на МБ, остальные каналы с номерами 1, 2, ..., 14, размещаются на лентах (дисках), при использовании которых в паспорте системного пакета необходимо заказывать соответствующие бобины (Приложение 9). Математический номер ленты (диска) определяется по номеру канала t и равен $50_8 + t_8$, где подстрочная цифра 8 указывает, что 50 и номер канала взяты в восьмеричном представлении (например, 10 равно 12_8).

Максимальное число ячеек во внешней памяти, которое может занимать информация канала, назовем длиной канала. В стандартном режиме длина канала с номером 0 равна

32 768 ячеек, длина каждого из остальных каналов (1—14) равна 524 288 ячеек. Длина любого канала может быть уменьшена с помощью процедуры *длинк*.

При выполнении оператора *input* записи из канала считываются последовательно. Имеется возможность изменять порядок чтения записей, пропускать записи или возвращаться к уже прочитанным. Для этих целей в язык вводится указатель чтения текущей записи IN. При выполнении операторов *output* и *marg* каждые новые записи присоединяются к уже имеющимся в канале вслед за ними. Место очередной записи, которая должна быть записана в канал, отмечается указателем текущей записи OUT. Имеется также указатель *НО*, фиксирующий одновременно конец записей вводимых по *input* и начало записей операторов *output* и *marg*. Значениями указателей IN, НО и OUT являются адреса записей в канале во внешней памяти.

Если при выполнении программы объем записей, помещаемых в канал превышает длину канала, то перед продолжением работы программы производится распечатка информации, находящейся в канале, от указателя НО до конца (так называемое освобождение канала). При этом значение указателя OUT устанавливается равным значению указателя НО. Освобождение канала осуществляется также при окончании выполнения программы.

В операторах каналов подразумевается, что все параметры — арифметические выражения имеют тип целый и размерность ноль.

Процедура *мод* осуществляет сдвиг указателя, вид которого задан первым параметром, на количество записей, указанное третьим параметром. Пусть T — номер текущей записи, тогда после выполнения оператора процедуры *мод* значение указателя будет соответствовать $T + N$ записи, где N — значение третьего параметра. Причем, если $N < 0$, то значение указателя уменьшается, при $N > 0$ — увеличивается, при $N = 0$ — не изменяется. Если $T + N < 0$ или $T + N$ больше номера записи, соответствующей текущему значению указателя OUT, то действие оператора *мод* не определено. Отсюда следует, что значение указателя OUT может только уменьшаться.

Процедура *изм* позволяет пересылать в скалярную переменную типа целый значение одного из указателей, либо устанавливает указатель на значение, указанное третьим параметром.

Процедура *коммут* позволяет установить для указанного первым параметром номера канала новый внешний носитель, задав для него номер направления и номер носителя.

Процедура *режк* задает значение ширины листа, возможно, изменяя порядок расположения страниц на листе АЦПУ (или терминала) (пп. 15.4.9 и 2.16).

Процедура *длинк* задает длину канала в ячейках внешней памяти. Если при этом оказывается, что значение 2-го параметра меньше или равно текущему значению указателя OUT, то производится распечатка накопленной в канале информации. При этом значение указателя OUT устанавливается равным значению указателя НО.

Процедура *инфк* выдает информацию о текущем состоянии канала, причем в массив *A* типа целый, идентификатор которого задан вторым параметром, засылаются целые значения в следующем порядке:

A[*1*] — номер направления и номер устройства ленты (диска, барабана), на котором располагается канал,
A[*2*] — длина канала в ячейках,
A[*3*] — количество оставшихся (незаполненных) ячеек в канале,
A[*4*] — значение указателя IN,
A[*5*] — значение указателя OUT,
A[*6*] — значение указателя НО,
A[*7*] — режим выдачи (0 — на АЦПУ, 1 — на терминал),
A[*8*] — ширина листа АЦПУ (терминала) (количество позиций).

Процедура *сохр* устанавливает указатели НО и OUT на конец последней записи канала, а IN на начало первой записи. Тем самым все записи канала становятся доступными для ввода операторами *input*. Поскольку каналы 1, ..., 14 размещаются на МЛ (МД), то они могут использоваться в других задачах.

15.5. Описания

Описания служат для того, чтобы определить некоторые свойства величин, используемых в программе и связать эти величины с идентификаторами. Описание идентификатора имеет силу только для одного блока. Вне этого блока тот же идентификатор может использоваться для других целей (см. п. 15.4.1, семантика).

При входе в блок все идентификаторы, описанные в блоке, приобретают смысл, вытекающий из описаний, указанных в начале этого блока. Если эти идентификаторы уже были определены другими описаниями, находящимися вне блока, то на некоторое время они получают новый смысл.

С другой стороны, те идентификаторы, которые не были описаны в блоке, сохраняют прежний смысл.

В момент выхода из блока (через **конец** или оператор перехода) все идентификаторы, которые описаны в блоке, теряют свой локальный смысл.

Описание можно снабдить добавочным описателем **собственный**. Это приводит к следующему: к моменту повторного входа в блок значения собственных величин сохраняются такими же, какими они были при последнем выходе из этого блока, в то время как значения величин, которые были специфицированы без описателя **собственный**, будут не определены. Все идентификаторы программы, за исключением меток, формальных параметров в описаниях процедур и функций и идентификаторов стандартных функций (пп. 15.3.2, 15.3.2) должны быть описаны. В начале любого блока никакой идентификатор нельзя описывать более одного раза.

Синтаксис.

⟨описание⟩ :: = ⟨описание типа и структуры⟩|⟨описание массива⟩|⟨описание переключателя⟩|⟨описание процедуры⟩|⟨описание функции⟩|⟨описание начального значения⟩ | ⟨описание тождества⟩ | ⟨общий массив⟩ | ⟨описание внешней подпрограммы⟩

15.5.1. Описания типа и структуры.

Синтаксис.

⟨порядки по внутренним измерениям⟩ :: = ⟨первичное выражение⟩|⟨порядки по внутренним измерениям⟩ × ⟨первичное выражение⟩

⟨структура⟩ :: = **массив** ⟨порядки по внутренним измерениям⟩

⟨начало рекурсии⟩ :: = ⟨арифметическое выражение⟩

⟨длина рекурсии⟩:: = ⟨арифметическое выражение⟩

⟨метка управляющего оператора цикла⟩ :: = ⟨метка⟩|⟨составная метка⟩

⟨описание верхнего индекса⟩ :: = ⟨начало рекурсии⟩ : ⟨длина рекурсии⟩ : ⟨метка управляющего оператора цикла⟩

⟨список типа⟩:: = ⟨простая переменная⟩|⟨простая переменная⟩, ⟨список типа⟩|⟨простая переменная⟩ ↑ [⟨описание верхнего индекса⟩]|⟨простая переменная⟩ ↑ [⟨описание верхнего индекса⟩], ⟨список типа⟩

⟨тип⟩ :: = **вещественный** | **целый** | **логический** | **комплексный**

⟨локальный или собственный тип⟩ :: = ⟨тип⟩ | **собственный** ⟨тип⟩

⟨описание типа и структуры⟩ :: = ⟨локальный или собственный тип⟩ ⟨список типа⟩|⟨локальный или собственный

тип⟩ ⟨список типа⟩ — ⟨структура⟩ | **упакованный** ⟨локальный или собственный тип⟩ ⟨список типа⟩ — ⟨структура⟩

Примеры.

целый *p*, *q*, *s*
собственный логический *Acryl*, *n*
вещественный *A*, *B* — **массив** *n* × *n*
целый *t*, *r*, *Fibonacci* ↑ [1:2:*Fibonacci formula*]

Семантика. Описания типа служат для указания того, что некоторые идентификаторы представляют простые переменные (или простые переменные с верхним индексом) данного типа. Переменные, которым описанием дан тип **вещественный**, могут принимать только положительные и отрицательные значения, включая нуль. Переменные, которым описанием дан тип **логический**, могут принимать только значения **истина** и **ложь**. Переменные, которым описанием дан тип **комплексный**, могут принимать только значения, являющиеся парой значений типа **вещественный**.

В арифметических выражениях любая позиция, которая может быть занята переменной типа **вещественный**, может быть занята переменной типа **целый**.

Семантика описателя **собственный** приведена выше (п. 15.5).

Элементы логического массива (не упакованного) располагаются по одному элементу в ячейке. Описатель **упакованный** в описании логического массива предписывает транслятору располагать по нескольку элементов (до 48) в ячейке. При этом упаковка элементов в ячейке осуществляется по последнему измерению, например, вектор $Л[i_1, \ldots, i_k]$ займет *entier* $((l-1)/48)+1$ ячеек, где l — длина последнего измерения массива Л.

Структура. Внутренняя размерность определяется как число первичных выражений, образующих список порядков по внутренним измерениям. Первичные выражения, задающие порядки по внутренним измерениям, вычисляются так же, как границы индексов (п. 15.5.2, выражения для границ). Если структура не указана, то переменная считается скалярной.

Описание верхних индексов. Задает информацию о верхнем индексе простой переменной, непосредственно предшествующей ограничителю ↑. Начало рекурсии задает в виде арифметического выражения номер начального члена рекуррентной последовательности. Длина рекурсии задает в виде арифметического выражения (значение которого

должно быть $\geqslant 1$) число последовательных членов рекуррентной последовательности, необходимых для вычисления очередного члена. Значения выражений, задающих начало и длину рекурсии, вычисляются так же, как границы индексов (п. 15.5.2, выражения для границ). Метка управляющего оператора цикла указывает оператор цикла, при выполнении которого вычисляются очередные члены рекуррентной последовательности.

Переменные с верхними индексами не могут помечаться описателем **собственный** и не могут использоваться в качестве формальных параметров процедур и функций (п. п. 15.5.4, 15.5.5). В остальном семантика переменных с верхним индексом описана в (п. 15.4.6, семантика).

15.5.2. Описания массивов.

Синтаксис.

⟨нижняя граница⟩ :: = ⟨арифметическое выражение⟩
⟨верхняя граница⟩ :: = ⟨арифметическое выражение⟩
⟨граничная пара⟩:: = ⟨нижняя граница⟩ : ⟨верхняя граница⟩
⟨список граничных пар⟩ :: = ⟨граничная пара⟩ | ⟨список граничных пар⟩, ⟨граничная пара⟩
⟨сегмент массива⟩ :: = ⟨идентификатор массива⟩ [⟨список граничных пар⟩] | ⟨идентификатор массива⟩ ↑ [⟨описание верхнего индекса⟩] [⟨список граничных пар⟩] | ⟨идентификатор массива⟩, ⟨сегмент массива⟩ | ⟨идентификатор массива⟩ ↑ [⟨описание верхнего индекса⟩], ⟨сегмент массива⟩ | ⟨внешний идентификатор⟩ [⟨список граничных пар⟩] | ⟨внешний идентификатор⟩, ⟨сегмент массива⟩
⟨список массивов⟩ :: = ⟨сегмент массива⟩ | ⟨список массивов⟩, ⟨сегмент массива⟩
⟨описание массива⟩ :: = **массив** ⟨список массивов⟩ | ⟨обобщенный тип⟩ **массив** ⟨список массивов⟩ | **массив** ⟨список массивов⟩ — ⟨структура⟩ | ⟨обобщенный тип⟩ **массив** ⟨список массивов⟩ — ⟨структура⟩
⟨обобщенный тип⟩:: = ⟨локальный или собственный тип⟩ | **упакованный** ⟨локальный или собственный тип⟩

Примеры.

массив *a*, *b*, *c* [*7*:*n*, *2*:*m*], *S*[— *2*:*10*]
собственный целый массив *A* [**если** *c*<*0* **то** *2* **иначе** *1*:*20*]
вещественный массив *q*[— *7*:— *1*]
целый массив *p*, *q*[*7*:*n*, *2*:*m*], *r*[— *2*:*10*] — **массив** *4* × *4*
комплексный массив *A*, *B*, *S* ↑ [*0*:*n*:*label*] [*1*:*20*]

массив *a, EXb, c*[7:*n*, 2:*m*], *EX*[—2:20]
комплексный массив *A, S* ↑ [0:*n*:*метка*], *EXB*[1:20]

Семантика. Описание массива указывает, что один или несколько идентификаторов представляют многомерные массивы переменных с индексами, и задает внешние размерности этих массивов, границы индексов, типы и структуры переменных. Для переменных с индексами, имеющих также и верхний индекс, указывается описание верхнего индекса, по тем же правилам, что и для простых переменных.

Описание внешнего массива задает размерность, тип этого массива и границы индексов. Внешний массив располагается во внешней памяти машины и доступ к его элементам возможен только через оператор *copy* (п. 15.4.13).

Внешний массив не может иметь внутреннюю структуру, верхний индекс и описатель **упакованный**.

Границы индексов. Границы индексов любого массива указываются в первых индексных скобках, не стоящих непосредственно вслед за ограничителем ↑ и следующих за идентификатором данного массива в виде списка граничных пар. Каждый член этого списка определяет нижнюю и верхнюю границы индексов в виде двух арифметических выражений, разделенных ограничителем :. Список граничных пар задает границы всех индексов в порядке их перечисления слева направо.

Размерности. Внешние размерности определяются как число членов в списках граничных пар.

Типы. Все массивы, описанные в одном описании, имеют один и тот же тип. Если описатель типа отсутствует, то подразумевается тип **вещественный**.

Выражения для нижних и верхних границ. Значения этих выражений вычисляются так же, как значения индексных выражений (п. 15.3.1, индексы).

Эти выражения могут зависеть только от переменных и функций, не локальных в том блоке, для которых имеет силу данное описание массива. В самом внешнем блоке программы могут быть описаны массивы только с постоянными границами.

Массив определен только в том случае, когда значения всех верхних границ индексов не меньше, чем значения соответствующих нижних границ.

Значения выражений для границ вычисляются один раз при каждом входе в блок.

Идентичность переменных с индексами.

Идентичность переменных с индексами не связана с границами индексов, задаваемых в описании массива. Даже если

массив описан как **собственный**, значения соответствующих переменных с индексами будут в любой момент времени определены только для той части переменных, у которых индексы находятся в пределах границ индексов, вычисленных последний раз.

15.5.3. Описания переключателей.

Синтаксис.

⟨переключательный список⟩ :: = ⟨именующее выражение⟩|
⟨переключательный список⟩, ⟨именующее выражение⟩
⟨описание переключателя⟩:: = **переключатель** ⟨идентификатор переключателя⟩: = ⟨переключательный список⟩

Примеры.

переключатель *S*: = *S1*, *S2*, *Q*[*m*],
если *v* > —*5* **то** *S3* **иначе** *S4*
переключатель *Q*: = *p1*, *W*

Семантика. В описании переключателя указывается совокупность значений соответствующих указателей переключателя. Эти значения записываются одно за другим, и являются значениями именующих выражений, перечисленных в переключательном списке. С каждым из этих именующих выражений связано положительное целое число 1, 2, ..., *n*, получаемое при пересчете членов списка слева направо. Значением указателя переключателя, соответствующим данному значению индексного выражения (п. 15.3.5), является значение именующего выражения в переключательном списке, имеющего данное значение индексного выражения своим порядковым номером.

Вычисление значений выражений в переключательном списке. Значение выражения, входящего в переключательный список вычисляется каждый раз, когда происходит обращение к члену списка, в который входит данное выражение. При вычислении значения выражения используются текущие значения всех входящих в него переменных.

Влияние областей действия. Если указатель переключателя встречается вне области действия какой-либо величины, входящей в именующее выражение в переключательном списке, и при вычислении значения этого указателя переключателя, выбирается именно это именующее выражение, то противоречия между идентификаторами величин в этом выражении и идентификаторами, описания которых имеют силу там, где расположен указатель переключателя, устраняются путем соответствующих систематических изменений последних идентификаторов.

15.5.4. Описания процедур.

Синтаксис.

⟨формальный параметр⟩ :: = ⟨идентификатор⟩

⟨список формальных параметров⟩ :: = ⟨формальный параметр⟩ | ⟨список формальных параметров⟩ ⟨ограничитель параметра⟩ ⟨формальный параметр⟩

⟨совокупность формальных параметров⟩ :: = ⟨пусто⟩ | (⟨список формальных параметров⟩)

⟨список идентификаторов⟩ :: = ⟨идентификатор⟩ | ⟨список идентификаторов⟩, ⟨идентификатор⟩

⟨список значений⟩ :: = **значение** ⟨список идентификаторов⟩; | ⟨пусто⟩

⟨список результатов⟩ :: = **результат** ⟨список идентификаторов⟩; | ⟨пусто⟩

⟨спецификация⟩ :: = **строка** | ⟨тип⟩ | **массив** | ⟨тип⟩ **массив** | **метка** | **переключатель** | **процедура** | ⟨тип⟩ **процедура** | **функция** | ⟨тип⟩ **функция**

⟨спецификация массива⟩ :: = **массив** | ⟨тип⟩ **массив**

⟨совокупность спецификаций⟩ :: = ⟨пусто⟩ | ⟨спецификация⟩ ⟨список идентификаторов⟩; | ⟨спецификация массива⟩ ⟨список идентификаторов массива⟩ — ⟨внешняя размерность ФРП⟩; | ⟨спецификация массива⟩ ⟨список идентификаторов массива⟩ — ⟨внешняя размерность ФРП⟩ + ⟨внутренняя размерность ФРП⟩; | ⟨совокупность спецификаций⟩ ⟨совокупность спецификаций⟩

⟨список идентификаторов массива⟩ :: = ⟨идентификатор массива⟩ | ⟨список идентификаторов массива⟩, ⟨идентификатор массива⟩

⟨внешняя размерность ФРП⟩ :: = ⟨целое без знака⟩

⟨внутренняя размерность ФРП⟩ :: = ⟨целое без знака⟩

⟨заголовок процедуры⟩ :: = ⟨идентификатор процедуры⟩ ⟨совокупность формальных параметров⟩; ⟨список значений⟩ ⟨список результатов⟩ ⟨совокупность спецификаций⟩

⟨тело процедуры⟩ :: = ⟨оператор⟩ | *library* ('⟨имя подпрограммы⟩')

⟨описание процедуры⟩ :: = **процедура** ⟨заголовок процедуры⟩ ⟨тело процедуры⟩

⟨имя подпрограммы⟩ :: = ⟨символ имени⟩ | ⟨имя подпрограммы⟩ ⟨символ имени⟩

⟨символ имени⟩ :: = ⟨буква⟩ | ⟨цифра⟩ | + | − | / | *

Примеры (см. также примеры в конце главы).

процедура След (*a*) Порядок: (*n*) Результат: (*S*);

```
    значение n; массив a; целый n;
    вещественный S;
начало целый k;
    S: = 0;
для k: = 1 шаг 1 до n цикл S: = S + a[k, k]
конец
процедура Транспонирование (a) Порядок: (n);
значение n; массив a; целый n;
начало вещественный w; целый i, k;

    для i: = 1 шаг 1 до n цикл
        для k: = 1 + i шаг 1 до n цикл
      начало w: = [i, k];
             a[i, k]: = a[k, i];
             a[k, i]: = w
        конец
конец Транспонирования.

процедура Абсмакс (a) Порядок: (n, m) Результат: (y)
    индексы: (i, k);
примечание Наибольший по абсолютной величине элемент матрицы a порядка n × m передается в y. Индексы этого элемента передаются в i и k;
массив a; целый n, m, i, k; вещественный y;
начало целый p, q;
y: = 0; для p: = 1 шаг 1 до n цикл для q: = 1 шаг 1
до m цикл если mod (a[p, q]) > y
то начало y: = mod(a[p, q]);
    i: = p; k: = q конец конец Абсмакса


процедура Скалярное произведение (a, b)
    Порядок: (k, p) Результат: (y);
значение k; целый k, p; вещественный y, a, b;
начало вещественный s;
    s: = 0;
для p: = 1 шаг 1 до k цикл s: = s + a × b;
    y: = s
конец Скалярного произведения
```

Семантика. Главной составной частью описания процедуры является оператор, называемый телом процедуры. К этому оператору можно обратиться посредством операторов процедур из различных мест блока, в начале которого находится описание данной процедуры. С телом процедуры связан

заголовок, который указывает, что некоторые идентификаторы, встречающиеся в теле процедуры, представляют формальные параметры. В момент обращения к процедуре (п. п. 15.3.2, 15.4.7) формальным параметрам в теле процедуры будут присвоены значения фактических параметров или же они будут заменены фактическими параметрами. Те идентификаторы в теле процедуры, которые не являются формальными параметрами, являются либо локальными, либо нелокальными в теле процедуры в зависимости от того, описаны они в теле процедуры или нет. Тело процедуры всегда действует подобно блоку независимо от того, имеет оно форму блока или нет. Следовательно, область действия метки, помечающей оператор внутри тела или само тело, никогда не может распространяться за тело процедуры. Кроме того, если идентификатор формального параметра заново описан внутри тела процедуры (включая случай использования его в качестве метки, как это указано в п. 15.4.1 (семантика)), ему придается тем самым локальное значение, и фактические параметры, которые ему соответствуют, недоступны во всей области действия этой внутренней локальной величины.

Спецификации. В заголовок процедуры может быть включена совокупность спецификаций, задающая с помощью очевидных обозначений информацию о классах и типах формальных параметров. В этой части заголовка ни один формальный параметр не может встречаться более одного раза. Формальные параметры, вызываемые по значению или результату, всегда следует специфицировать, в то время как спецификации формальных параметров, вызываемых по наименованию, можно опускать (п. 15.4.7). Если описание процедуры является подпрограммой, то все формальные параметры должны быть специфицированы. Для формальных параметров, являющихся массивами, могут быть указаны их внутренние и внешние размерности. Если внутренняя размерность не указана, то считается, что она равна нулю, если не указана также и внешняя, то последняя считается равной единице.

library в качестве тела процедуры. Процедура, тело которой состоит из оператора *library*, позволяет использовать библиотеку подпрограмм МС Дубна. Формальные параметры такой процедуры должны соответствовать формальным параметрам подпрограммы, имя которой указано в аргументе оператора *'library*. Количество символов в имени (причем для греческих и больших букв учитывается их представление несколькими символами, приведенное в Приложении 7) не должно превышать 8.

15.5.5. Описания функций.

Синтаксис.

⟨вычисляющее выражение⟩ :: = ⟨арифметическое выражение⟩|⟨логическое выражение⟩
⟨формальный параметр⟩:: = ⟨идентификатор⟩
⟨список формальных параметров функции⟩ :: = ⟨формальный параметр⟩|⟨список формальных параметров функции⟩, ⟨формальный параметр⟩
⟨совокупность формальных параметров функции⟩ :: = ⟨пусто⟩|(⟨список формальных параметров функции⟩)
⟨описатель функции⟩ :: = **функция**|⟨тип⟩ **функция**
⟨описание функции-выражения⟩ :: = ⟨описатель функции⟩ ⟨идентификатор функции⟩ ⟨совокупность формальных параметров функции⟩ = ⟨вычисляющее выражение⟩
⟨описание функции-процедуры⟩:: = ⟨тип⟩ ⟨описание процедуры⟩
⟨описание функции⟩:: = ⟨описание функции-выражения⟩ | ⟨описание функции-процедуры⟩

Примеры (см. также примеры в конце главы).

функция *delta* = *0.5* × *a* × *b* × *cos* (*c*)
целая функция *c*(*i*, *j*) = | *a*[*i*, *1*], ..., *a*[*i*, *n*]| × |*b*[*1*, *j*], ..., *b*[*n*, *j*]|;
функция *y* = *f*(*sqrt*(*m*↑*2* + *n*↑*2*))
целая процедура *step*(*u*); **вещественный** *u*;
step: = **если** *0* ⩽ *u* ⩽ *1* **то** *1* **иначе** *0*;

Семантика. Описание функции задает правило вычисления некоторого значения, связываемого с идентификатором функции или процедуры. Способ нахождения значения задается вычисляющим выражением для функций-выражений и описанием процедуры для функций-процедур. Обращение к описанию некоторой функции производится при необходимости вычислить выражение, содержащее данную функцию. Если значения данной функции не являются скалярами, то функция не может быть составной частью более сложного выражения, чем сама функция. Описание функции-процедуры может быть подпрограммой, в этом случае для нее справедливы замечания пункта (п. 15.5.4, спецификации).

Значения указателей функций. Вычисление функции-выражения происходит по правилу: фактические параметры подставляются в вычисляющее выражение на места

соответствующих формальных параметров (после заключения последних в скобки). Значение видоизмененного таким образом вычисляющего выражения берется в качестве значения функции. Задание фактических параметров должно удовлетворять естественному требованию, чтобы их подстановка приводила к правильному выражению. Если описатель типа у функции-выражения отсутствует, то считается, что функция имеет тип **вещественный.**

Для того, чтобы описание процедуры определяло значение указателя функции, необходимо, чтобы внутри тела процедуры встречался один или несколько явных операторов присваивания с идентификатором этой процедуры в левой части; по крайней мере один из них должен выполняться, и тип идентификатора процедуры должен быть указан включением описателя типа в качестве самого первого символа описания функции. Последнее значение, присвоенное таким образом, используется для дальнейшего вычисления выражения, в котором встречается указатель функции. Любое другое появление идентификатора процедуры внутри ее тела, но не в левой части оператора присваивания означает обращение к этой процедуре.

15.5.6. Описания начальных значений.

Синтаксис.

⟨описание начального значения⟩ :: = ⟨переменная⟩ = ⟨вычисляющее выражение⟩

Примеры.

pi = 3.1415926536
a[ ] = |1/6, 1/3, 1/3, 1/6|
net ↑ [0] [, ,] = 0

Семантика. Описание начального значения указывает, что переменной, заданной в левой части этого описания, должно быть присвоено значение вычисляющего выражения, стоящего в правой части. Для собственной переменной это присваивание осуществляется при первом входе в блок. Для несобственной переменной присваивание производится при каждом входе в блок. Если левая часть описания начального значения есть формальный параметр, то присваивание ему осуществляется при каждом входе в блок. Переменная из левой части описания начального значения должна иметь тип (с точностью до целое-вещественное) и структуру, совпадающие с типом и структурой вычисляющего выражения. Вычисляющим выражением 0, 0.0 и .0 (нуль), а также **ложь** может быть приписана любая структура, которую требует переменная из элемента равенства. Все переменные, входящие в вычисляю-

щее выражение из правой части описания начального значения, должны быть описаны в объемлющем блоке.

15.5.7. Описания тождества.

⟨описание тождества⟩ :: = ⟨описание действительной и мнимой частей⟩|⟨описание компонент массива⟩

Описание действительной и мнимой частей. Синтаксис.

⟨совокупность индексов⟩:: = ⟨идентификатор⟩|⟨совокупность индексов⟩, ⟨идентификатор⟩

⟨описание действительной и мнимой частей⟩ :: = ⟨простая переменная⟩ = ⟨простая переменная⟩ + *i*⟨простая переменная⟩|⟨идентификатор массива⟩ [«совокупность индексов»] = ⟨идентификатор массива⟩ [«совокупность индексов»] + *i*⟨идентификатор массива⟩ [«совокупность индексов»]

Примеры.

$z = x + iy$

$lambda = Relambda + iImlambda$

$a[i] = b[i] + ic[i]$

Семантика. Описание действительной и мнимой частей используется как средство для введения обозначений действительной и мнимой частей комплексной переменной. Таким образом, описание вида $z = x + iy$ декларирует, что текущее значение действительной части переменной z тождественно равно текущему значению переменной x, а текущее значение мнимой части тождественно равно текущему значению переменной y.

Переменная, стоящая в левой части описания, должна быть описана и иметь тип **комплексный.** Переменные, стоящие в правой части описания, локализуются данным описанием в том смысле, что в блоке, в котором находится данное описание, они получают структуру переменной из левой части и им приписывается тип **вещественный**. Идентификаторы, образующие совокупность индексов, локализуются описанием действительной и мнимой частей только в пределах данного описания. Вне его эти идентификаторы могут свободно использоваться.

Описания компонент массивов.

Синтаксис.

⟨описание компонент массива⟩ :: = ⟨простая переменная⟩ = ||⟨идентификатор массива⟩ [⟨совокупность индексов⟩] || |⟨идентификатор массива⟩ [«совокупность индексов»] = ||⟨идентификатор массива⟩ [«совокупность индексов», ⟨совокупность индексов⟩] ||

Примеры.

$A = \|a[l, j]\|$
$x = [m, n] = \|s[m, n, k, z]\|$

Семантика. Описание компонент массива используется как средство для введения обозначения скалярных компонент многомерной переменной. Переменная, стоящая в левой части, должна быть описана. Избыточное количество индексов у переменной из правой части описания должно равняться внутренней размерности переменной из левой части. Переменная из правой части описывается и локализуется данным описанием в том смысле, что в блоке, в котором находится данное описание, она получает тип переменной из левой части и внутреннюю размерность, равную нулю. Граничные пары по каждой дополнительной позиции индекса равны $1{:}n$, где n — порядок по соответствующему внутреннему измерению переменной из левой части.

Идентификаторы, образующие совокупности индексов, локализуются описанием компонент массива только в пределах данного описания. Следовательно, вне его эти идентификаторы могут быть свободно использованы для любых целей.

15.5.8. Описание общей памяти.

Общая память определяется с помощью описаний общих массивов, включающих в себя описанные в программе простые переменные и идентификаторы массивов. Описания общих массивов задают расположение указанных в них объектов на общих участках оперативной памяти, доступных также другим отдельно транслируемым программам.

Синтаксис.

⟨общий массив⟩ :: = **общий** '⟨имя общего⟩', ⟨список общих объектов⟩| **общий** ⟨список общих объектов⟩

⟨имя общего⟩:: = ⟨символ имени⟩|⟨имя общего⟩ ⟨символ имени⟩

⟨список общих объектов⟩ :: = ⟨общий объект⟩|⟨общий объект⟩ (⟨длина⟩)|⟨список общих объектов⟩, ⟨общий объект⟩|⟨список общих объектов⟩, ⟨общий объект⟩ (⟨длина⟩)

⟨общий объект⟩ :: = ⟨простая переменная⟩|⟨идентификатор массива⟩

⟨длина⟩ :: = ⟨целое без знака⟩

⟨символ имени⟩ :: = ⟨буква⟩|⟨цифра⟩|+|−|/|*

Семантика. Имя общего массива должно содержать не более шести символов (причем для греческих и больших букв

необходимо учитывать то, что они представляются несколькими символами, см. Приложение 7).

Имя общего массива используется только для идентификации участков общей памяти в отдельно транслируемых подпрограммах. При этом переменные из разных подпрограмм, для которых указан один и тот же участок общей памяти, оказываются совмещенными в памяти, т. е. изменение значения переменной одной подпрограммы вызывает такое же изменение значения соответствующей переменной в другой подпрограмме и наоборот. Порядок расположения общих объектов в описании общего массива может задавать также частичное совмещение переменных или совмещение переменных разных типов и структур. Вся ответственность при этом возлагается на программиста.

При комплексации альфа-программы и фортран-программы общему массиву альфа-программы соответствует *common*-блок фортран-программы с таким же именем. При комплексации альфа-программы и мадлен-программы общему массиву альфа-программы с именем «И» соответствует общий массив мадлен-программы с именем «* И *». Если в описании общего массива альфа-программы отсутствует имя общего, то участок общей памяти идентифицируется как общий массив без имени. В фортран-программе ему соответствует непомеченный *common*-блок, в мадлен-программе — имя общего массива, состоящего из двух звездочек.

Все идентификаторы, входящие в список общих объектов, должны быть описаны в блоке, где находится данное описание общего массива (или в объемлющих блоках).

Порядок расположения общих объектов в общем массиве соответствует их порядку вхождения в описание общего массива, причем каждый последующий общий объект располагается в ячейках общей памяти, непосредственно следующих за ячейками, резервированными для предыдущего общего объекта.

Если в программе имеется несколько описаний общего массива с одним и тем же именем (или вовсе без имени), то предполагается, что все общие объекты располагаются в одном участке общей памяти, который идентифицируется указанным именем (или как общий массив без имени). Все общие объекты, входящие в эти описания, размещаются последовательно в порядке их вхождения в описания общих массивов в программе. Если описание общего массива входит в описание процедуры (функции-процедуры), то считается, что перед размещением общих объектов в общей памяти тело процедуры (функции-процедуры) как бы выносится в конец блока. Если для общего объекта в общем массиве явно не указана длина то резерви-

руется количество ячеек, определяемое описанием этого общего массива. Для скалярной переменной типа целый, вещественный или логический резервируется одно слово. Для комплексной скалярной переменной — два слова. Для статистического массива типа целый, вещественный или логический (не упакованный) — количество слов, равное числу элементов массива. Для статистического комплексного массива выделяется количество слов, равное удвоенному числу элементов. Для статистических упакованных логических массивов выделяется объем памяти, необходимый для его размещения в оперативной памяти (п. 2.25).

Для динамического массива память не резервируется, так что общий объект, следующий за идентификатором динамического массива в списках общих объектов, будет совмещен по памяти с первым элементом этого массива. В связи с этим рекомендуется для динамических массивов явно указывать необходимый объем памяти, равный максимальному числу слов, которые может занимать динамический массив во время выполнения программы.

15.5.9. Описание внешних подпрограмм.

Синтаксис.

⟨описание альфа-подпрограммы⟩ :: = **альфа** ⟨список имен⟩
⟨описание фортран-подпрограммы⟩ :: = **фортран** ⟨список имен⟩
⟨описание алгол-подпрограммы⟩ :: = **алгол** ⟨список имен⟩
⟨список имен⟩: = ⟨имя подпрограммы⟩ | ⟨тип⟩ ⟨имя подпрограммы⟩ | ⟨список имен⟩, ⟨имя подпрограммы⟩ | ⟨список имен⟩, ⟨тип⟩ ⟨имя подпрограммы⟩
⟨имя подпрограммы⟩ :: = ⟨идентификатор⟩
⟨тип⟩:: = **вещественный** | **целый** | **логический** | **комплексный**
⟨описание внешней подпрограммы⟩ :: = ⟨описание альфа-подпрограммы⟩ | ⟨описание фортран-подпрограммы⟩ | ⟨описание алгол-подпрограммы⟩

Семантика. Описание внешних подпрограмм устанавливает соответствие между именами отдельно транслируемых процедур или функций (подпрограмм) и именами процедур или функций, используемыми в данном блоке.

Если в списке имен описания подпрограмм перед именем подпрограммы указан тип, то это имя может использоваться как идентификатор указателя функции. Если тип не указан, то данное имя может использоваться только в качестве идентификатора оператора процедуры. Если подпрограмма является функцией, то ее результатом может быть только скалярное значение.

Области действия имен подпрограмм, перечисленных в описании внешней подпрограммы, определяются по правилам алгола-60.

Имена подпрограмм, перечисленные в описании альфа-подпрограмм, определяются как имена процедур или функций, транслируемых трансляторов Альфа-6.

Имена подпрограмм, перечисленные в описании фортран-подпрограмм, определяются как имена фортранных подпрограмм, либо подпрограмм, написанных на автокоде мадлен. И, наконец, имена подпрограмм, указанные в описании алгол-подпрограмм, определяются как имена процедур или функций, транслируемых трансляторов алгол-ГДР.

В качестве фактического параметра (*ФКП*) внешней подпрограммы в операторе процедуры или функции могут быть объекты, определяемые языком альфа-6, причем считается, что все соответствующие им формальные параметры (*ФРП*) в теле процедуры (функции) специфицированы, а для массивов заданы внешняя и внутренняя размерность.

Это накладывает ограничение на использование *ФКП* в операторах внешних подпрограмм, которое состоит в том, что *ФКП* должны иметь тип, вид и структуру соответствующих им *ФРП*, определенных во внешней подпрограмме. В противном случае результат действия подстановки *ФКП* на место *ФРП* в момент выполнения данного оператора окажется неопределенным. Исключением является случай, когда тип *ФРП* — вещественный (и *ФРП* не перевычисляется в теле процедуры), а *ФКП* может принимать тип целый. Следует также учитывать ограничения, указанные в Приложении 14.

15.6. Примеры описаний процедур и функций

Пример 1.

целая процедура $fi(n)$ с начальным значением: (q) и определяющей функцией: (hi); **значение** n, q; **целый** n, q; **целая функция** hi;

примечание: Данная процедура реализует схему примитивной рекурсии функции fi для неотрицательного значения аргумента n с определяющей функцией $hi(p,\ fi(p))$ и начальным значением $fi(0) = q$ (см. С. К. Клини. Введение в метаматематику.— М., 1957);

начало

Схема примитивной рекурсии:

если $n = 0$ **то** $fi := q$ **иначе**

```
        начало целый Ф, i; Ф: = q;
        для i: = 0, ..., n — 1 цикл
            Ф: = hi(i, Ф);
        fi: = Ф
        конец
    конец процедуры fi
```

Пример 2.

```
процедура jacobi (n, eivec) tranc:(a) res:(d, v, rot);
значение n, eivec;
целый n, rot; логический eivec; массив a, d, v;
начало
```

примечание: Метод Якоби для действительных симметрических матриц. Описание метода и процедуры *jacobi* см. Уилкинсон, Райнш. Справочник алгоритмов на языке алгол: Линейная алгебра. /Пер. с англ./ Под ред. Ю. И. Топчиева.— М.: Машиностроение, 1976, 389 с.

Входные параметры процедуры *jacobi*:

n — порядок исходной матрицы *A*;

eivec — логическая переменная, имеющая значение **истина**, если необходимо вычисление собственных векторов, и **ложь** в противном случае;

a — массив *a*[1:*n*, 1:*n*], содержащий элементы матрицы *A*. При работе процедуры будут использованы только диагональные и наддиагональные элементы (*a*[*i*, *k*], *k* ⩾ *i*).

Выходные параметры процедуры *jacobi*:

a — массив *a*[1:*n*, 1:*n*]. Наддиагональные элементы этого массива использованы при работе процедуры для записи других переменных. Диагональные и поддиагональные элементы оставлены без изменения (поэтому в массиве *a* будет содержаться полная информация об исходной матрице *A*, если поддиагональные элементы были введены в память);

d — массив *d*[1:*n*] диагональных элементов матрицы *D*, т. е. массив вычисленных собственных значений матрицы *A*;

v — массив *v*[1:*n*, 1:*n*], содержащий (если *eivec* ≡ **истина**) элементы матрицы *V*, *k*-й столбец которой есть нормированный собственный вектор, соответствующий собственному значению *d*[*k*];

rot — параметр, фиксирующий количество проведенных плоских вращений при приведении исходной матрицы к диагональному виду;

```
вещественный sm, c, s, t, h, g, tau, theta, tresh;
целый p, q, i, j;
массив b, z[1:n];
program:
если eivec то
   начало v[,]: = 0.0;
   для p: = 1, ..., n цикл v[p, p]: = 1.0;
   конец;
b[ ]: = d[ ]: = |a[1, 1], ..., a[n, n]|;
z[ ]: = 0;
rot: = 0;
для i: = 1, ..., 50 цикл
swp:
начало
   sm: = 0;
   для p: = 1, ..., n — 1 цикл
   для q: = p + 1, ..., n цикл
   sm: = sm + abs(a[p, q]);
   если sm = 0 то на out;
   tresh: = если i < 4
   то 0.2 × sm/n↑2 иначе 0.0;
   для p: = 1, ..., n — 1 цикл
   для q: = p + 1, ..., n цикл
   начало
      g: = 100 × abs(a[p, q]);
      если i > 4 ∧ abs(d[p]) + g =
      = abs(d[p]) ∧ abs(d[p]) + g =
      = abs(d[q]) то a[p, q]: = 0
      иначе
      если abs(a[p, q]) > tresh то
   rotate: начало
      h: = d[q] — d[p];
      если abs(h) + g = abs(h) то t: = a[p, q]/h
      иначе
      начало
      theta: = 0.5 × h/a[p, q];
      t: = 1/abs(theta) + sqrt(1 + theta ↑ 2));
      если theta < 0 то t: = —t
      конец вычисления тангенса угла вращения;
      c: = 1/sqrt(1 + t ↑ 2);
      s: = t × c;
```

```
        tau: = s/(1 + c);
        h: = t × a[p, q];
        z[p]: = z[p] — h;
        z[q]: = z[q] + h;
        d[p]: = d[p] — h;
        d[q]: = d[q] + h;
        a[p, q]: = 0;
        для j: = 1, ..., p — 1 цикл
        начало
        g: = a[j, p]; h: = a[j, q];
        a[j, p]: = g — s × (h + g × tau);
        a[j, q]: = h + s × (g — h × tau);
        конец варианта 1 ⩽ j < p;
        для j: = p + 1, ..., q — 1 цикл
        начало
        g: = a[p, j]; h: = a[j, q];
        a[p, j]: = g — s × (h + g × tau);
        a[j, p]: = h + s × (g — h × tau);
        конец варианта p < j < g;
        для j: = q + 1, ..., n цикл
        начало g: = a[p, j]; h: = a[—p, j];
        a[p, j]: = g — s × (h + g × tau);
        a[q, j]: = h + s × (g — h × tau);
        конец варианта q < j ⩽ n;
        если eivec то
        для j: = 1, ..., n цикл
        начало g: = v[j, p]; h: = v[j, q];
        v[j, p]: = g — s × (h + g × tau);
        v[j, q]: = h + s × (g — h × tau);
        конец вычисления массива v;
        rot: = rot + 1;
        конец rotate;
    конец;
    d[ ]: = b[ ]: = b[ ] + z[ ];
    z[ ]: = 0;
    конец swp;
out:
конец jacobi;
```

Пример 3.

```
процедура Рунге Кутта с начальными данными (x0,
y0) Правая часть: (f) Шаг: (h) Результаты: (x, y);
значение x0, y0, h;
результат x, y;
```

вещественный *x0*, *y0*, *h*, *x*, *y*;
вещественная функция *f*;
начало
функция $K1 = h \times f(x0, y0)$;
функция $K2 = h \times f(x0 + 0.5 \times h, y0 + 0.5 \times K1)$;
функция $K3 = h \times f(x0 + 0.5 \times h, y0 + 0.5 \times K2)$;
функция $K4 = h \times f(x0 + h, y0 + K3)$;
$x := x0 + h$;
$y := y0 + 1/6 \times K1 + 1/3 \times K2 + 1/3 \times K3 + 1/6 \times K4$
конец Рунге Кутта

Пример 4.

процедура Собственное значение (*lambda*) флаттера панели
Приведенная скорость: (*u*)
Парабола устойчивости: (*c*)
Начальные приближения: (*x0*, *x1*)
Граница ошибки: (*eps*);
значение *u*, *c*, *eps*, *x0*, *x1*; **результат** *lambda*;
вещественный *u*, *c*, *eps*;
комплексный *lambda*, *x0*, *x1*;
примечание: Процедура вычисляет одно собственное значение *lambda*, соответствующее флаттеру тонкой упругой прямоугольной панели. Описание метода см. А. А. Мовчан.— ПММ, т. 21, вып. 2, 1957 и Инженерный сборник, т. 27, 1960, стр. 70—76. Решение трансцендентного уравнения $f(x) = 0$, к которому сводится задача, производится методом последовательных приближений. Если заданные начальные приближения приводят к собственному значению, соответствующему области устойчивости, происходит останов;
начало комплексный $x \uparrow [0:2$:Итерация]; $x \uparrow [0] = x0$; $x \uparrow [1] = x1$; **целый** *i*;
комплексная функция $f(x) = ch(2 \times x) - cos(sqrt(u/(4 \times x) + x \uparrow 2)) \times ch(sqrt(u/(4 \times x) - x \uparrow 2)) - 3 \times x \uparrow 2/sqrt(u \uparrow 2/(16 \times x \uparrow 2) - x \uparrow 4) \times sin(sqrt(u/4 \times x) + x \uparrow 2)) \times sh(sqrt(u/(4 \times x) - x \uparrow 2))$;
комплексная функция $sh(y) = (exp(y) - exp(-y))/2$;
комплексная функция $ch(y) = (exp(y) + exp(-y))/2$;
Итерация:
для $i := 0, i + 1$ **пока** $mod(x \uparrow [i + 1] - x \uparrow [i]) \geqslant eps$ **цикл** $x \uparrow [i + 2] := x \uparrow [i + 1] - f(x \uparrow [i + 1]) \times (x \uparrow [i + 1] - x \uparrow [i])/(f(x \uparrow [i + 1]) - f(x \uparrow$

[*i*])); *lambda*: = *u* ↑ 2/(16 × *x* ↑ [ ] ↑ 2) − 4 × *x* ↑ [ ] ↑ 4;
если *Re*(*lambda*) ⩾ *c* × (*Im*(*lambda*)) ↑ 2 ∨ *Im*(*lambda*) = 0 **то**
область устойчивости: **стоп иначе**
конец *Флаттера*

ГЛАВА 16
ЯЗЫК ДАННЫХ

Данные состоят из совокупности групп данных, которые вводятся по оператору *input* при выполнении программы.

Синтаксис.

⟨данные альфа-6⟩:: = ⟨имя данных⟩:⟨данные каналов⟩;
⟨данные каналов⟩:: = ⟨данные канала⟩|⟨данные каналов⟩; ⟨данные канала⟩
⟨данные канала⟩:: = **канал** ⟨номер канала ввода⟩*⟨последовательность групп данных⟩|⟨последовательность групп данных⟩
⟨последовательность групп данных⟩:: = ⟨группа данных⟩|⟨последовательность групп данных⟩; ⟨группа данных⟩
⟨номер канала ввода⟩:: = ⟨целое без знака⟩
⟨группа данных⟩:: = ⟨группа числовых данных⟩|⟨группа логических данных⟩|⟨группа текстовых данных⟩|⟨группа БЭСМ данных⟩
⟨группа числовых данных⟩:: = ⟨элемент числового ввода⟩|⟨группа числовых данных⟩, ⟨элемент числового ввода⟩
⟨элемент числового ввода⟩:: = ⟨число⟩|⟨комментарий⟩ ⟨число⟩
⟨комментарий⟩:: = ⟨открытый комментарий⟩:|⟨открытый комментарий⟩ = |⟨открытый комментарий⟩: =
⟨открытый комментарий⟩:: = ⟨буква⟩|⟨открытый комментарий⟩ ⟨любой символ кроме двоеточия знака равенства знака присваивания точки с запятой⟩
⟨группа логических данных⟩:: = ⟨элемент логического ввода⟩|⟨группа логических данных⟩, ⟨элемент логического⟩ ввода
⟨элемент логического ввода⟩:: = ⟨логическое значение⟩|⟨комментарий⟩ ⟨логическое значение⟩
⟨группа текстовых данных⟩:: = ⟨строка⟩
⟨группа БЭСМ данных⟩:: = ⟨комментарий⟩ ⟨список элементов БЭСМ-данных⟩|⟨список элементов БЭСМ-данных⟩
⟨список элементов БЭСМ-данных⟩:: = ⟨элемент БЭСМ-данных⟩|⟨список элементов БЭСМ-данных⟩ ⟨элемент БЭСМ-данных⟩

⟨элемент БЭСМ-данных⟩:: = ⟨БЭСМ-число⟩|⟨БЭСМ-константа⟩|⟨БЭСМ-команда⟩|⟨БЭСМ-шкала⟩
⟨БЭСМ-число⟩:: = *ч* ⟨число⟩
⟨БЭСМ-константа⟩:: = *с* ⟨восьм цифра⟩|⟨БЭСМ-константа⟩ ⟨восьм цифра⟩
⟨БЭСМ-команда⟩:: = *к* ⟨двоичная цифра⟩ ⟨восьм цифра⟩ ⟨четв цифра⟩ ⟨шесть восьм цифр⟩ ⟨двоичная цифра⟩ ⟨восьм цифра⟩ ⟨четв цифра⟩ ⟨шесть восьм цифр⟩
⟨БЭСМ-шкала⟩:: = *б* ⟨двоичная цифра⟩|⟨БЭСМ-шкала⟩ ⟨двоичная цифра⟩
⟨имя данных⟩:: = ⟨идентификатор⟩
⟨шесть восьм цифр⟩:: = ⟨восьм цифра⟩ ⟨восьм цифра⟩ ⟨восьм цифра⟩⟨восьм цифра⟩ ⟨восьм цифра⟩ ⟨восьм цифра⟩

Примеры.

канал 1∗1; 2; *с* = 3.14; *Р* = **истина, ложь, ложь**; 'данные для счета';
канал 2∗*с* = *0*; *с73145*; *601100001*; *ч1 ч0.2 ч0.3*;

Семантика. Каждая группа данных присваивается одному объекту ввода (простой переменной, массиву или переменной с индексами), указанному в операторе *input*.

Соответствие между группами данных одного канала и объектами ввода определяется порядком выполнения операторов *input* с данным каналом и порядком следования объектов ввода в операторе *input*.

Номером канала является целое десятичное число от 0 до 14. При выполнении программы данные нулевого канала хранятся на МБ. Данные других каналов хранятся на МЛ (МД) (пп. 2.14, 15.4).

Отсутствие номера канала в начале текста альфа-данных означает, что все группы данных, стоящие перед первым указателем канала, вводятся по нулевому каналу.

Описанные объекты числового и логического ввода — числа и логические значения — понимаются в смысле языка альфа-6.

Список элементов БЭСМ-данных может содержать только однородные элементы, т. е. либо одни БЭСМ-числа, либо одни БЭСМ-команды, либо одни БЭСМ-константы, либо одни БЭСМ-шкалы.

Элемент БЭСМ-данных определяет значение, занимающее одну ячейку БЭСМ-6.

БЭСМ-команда определяет пару команд БЭСМ-6, размещенных в одной ячейке.

БЭСМ-константа размещается в ячейке, начиная со старших разрядов, оставшиеся справа разряды заполняются нулями. Аналогично представляется БЭСМ-шкала.

Число восьмеричных цифр в БЭСМ-константе должно быть $\leqslant 16$, а число двоичных цифр в БЭСМ-шкале $\leqslant 48$.

Используя БЭСМ-данные можно вводить в программу подпрограммы в машинных кодах, а также восьмеричные массивы БЭСМ-данных и десятичные массивы БЭСМ-чисел.

Пример 1.

Рассмотрим оператор

если *P* **то** *input* (*1*, *c*) **иначе** *input* (*2*, *c*);

и группу данных:

канал *1*1*; *2*; *c = 3.14*; *P* = **истина**, **ложь**, **ложь**;
'ДАННЫЕ ДЛЯ СЧЕТА';
канал *2* c = 0*; *c73145*; *601100001*; *ч1 ч0.2 ч0.3*;

Пусть оператор ввода по первому каналу проработал уже два раза, а по второму ни одного. Тогда если *P* истинно, то в результате работы рассмотренного оператора *c* получит значение, равное 3.14, в противном случае — *c* станет равным 0.

Пример 2.

Пусть *a*, *b*, *c* — скаляры. *ЛОГ* — логический вектор длины 3, и по первому каналу не считано ни одной группы данных из предыдущего примера. Тогда в результате работы оператора:

input (*1*, *a*, *b*, *c*, *ЛОГ*)

переменные примут следующие значения:

a = 1, *b* = 2, *c* = 3.14;
ЛОГ = (**истина**, **ложь**, **ложь**);

Если теперь выполнится оператор

input (1, *B* [*a*, *b*]),

где *B* — целый массив, то в *B*, начиная с элемента *B* [1, 2], расположится текст:

'ДАННЫЕ ДЛЯ СЧЕТА'

представленный целыми числами (Приложение 7).

РАЗДЕЛ IV
ДИАГНОСТИКА ОШИБОК

ГЛАВА 17
УКАЗАНИЯ К ИСПОЛЬЗОВАНИЮ СПИСКА СООБЩЕНИЙ

Перед выполнением последовательности заданий система проверяет правильность комплектации и перфорации альфа-задачи. Перед обработкой очередного задания проверяется системная программа, задание работ и данные. Все перечисленные проверки относятся к этапу предварительного контроля. Далее происходит полный синтаксический контроль программы. При отсутствии синтаксических ошибок выполняется семантический контроль программы, при котором обнаруживаются семантические ошибки. Контролируется использование архивов, а также фиксируются ошибки при выполнении программы.

При обнаружении ошибки система выдает на АЦПУ сообщение о ней. Сообщениям указанных трех этапов контроля предшествует один из заголовков:

ПРЕДВАРИТЕЛЬНЫЙ КОНТРОЛЬ
СИНТАКСИЧЕСКИЙ КОНТРОЛЬ
СЕМАНТИЧЕСКИЙ КОНТРОЛЬ

Заголовок идентифицирует этап контроля и отсылает пользователя к соответствующей главе настоящего раздела, где можно найти описание выданных сообщений. Сообщения о неправильном использовании архивов и об ошибках, возникающих в процессе выполнения программы, выдаются без заголовков. Их описание можно найти в последних двух главах настоящего раздела. Внутри каждой главы сообщения упорядочены по алфавиту.

Сообщение, выдаваемое на АЦПУ, может начинаться номером перфокарты, либо диапазоном номеров. Номер перфокарты в сообщении указывает ту перфокарту программы (данных или системной программы), к которой относится сообщение.

Диапазон номеров в сообщении означает, что ошибка находится на перфокартах программы указанного диапазона.

Обнаружив ошибку, система печатает сообщение о ней и, в зависимости от вида ошибки, определяет дальнейший ход выполнения задания. Ниже перечисляются виды ошибок и реакция системы в случае их возникновения.

а) Обнаруженная ошибка делает невозможной дальнейшую обработку задания. Сюда относятся грубые ошибки и нарушения количественных ограничений.

б) Обнаруженная ошибка позволяет выполнить лишь часть работ задания. Как правило, дальнейшая работа системы направлена на выявление других ошибок.

в) Сообщение предупреждает о возможной ошибке. Тексты таких сообщений начинаются символом *. Дальнейшая работа системы от выдачи таких сообщений не зависит. Сюда же относятся сообщения о завершении некоторых работ системы.

Все сообщения описываются в следующих главах настоящего раздела. В общем случае описание сообщения содержит:

— текст сообщения;

— описание причины выдачи сообщения под заголовком «комментарий»;

— рекомендацию к поиску ошибки и ее исправлению под заголовком «рекомендация»;

— пример, относящийся к комментарию или рекомендации.

При выдаче сообщения на АЦПУ в его текст могут быть включены параметры входной информации задания либо выполняемой программы, которые называются параметрами сообщений. Ниже приводится перечень всех параметров:

«адрес» — последовательность из пяти восьмеричных цифр;

«восьм длина» — последовательность восьмеричных цифр;

«восьм номер» — последовательность восьмеричных цифр;

«идентификатор» — последовательность букв и цифр, начинающаяся буквой;

«имя» — идентификатор не более чем из шести символов;

«имя команды» — имя системной команды;

«номер» — последовательность десятичных цифр;

«номер канала» — последовательность десятичных цифр;

«номер перфокарты» — последовательность десятичных цифр, внутри которой может находиться десятичная точка;

«диапазон» — есть «номер перфокарты» либо конструкция: «номер перфокарты» — «номер перфокарты»;

«символ» — основной символ языка альфа-6;

«символ или простая конструкция» — см. описание группы сообщений (Г);

«синт» — см. описание группы сообщений (Г);

«текст» — последовательность символов языка альфа-6;

«тип записи» — одно из слов: «программа» или «данные»;

«фрагмент» — один из текстов: «программы», «данных», «фрагмента замен» либо «системной программы».

Сообщения, относящиеся к указанным выше трем этапам контроля, весьма разнородны по своему назначению и реакции системы.

Поэтому ниже мы приводим другую классификацию, основанную на группах родственных сообщений.

(А) Ошибки комплектации альфа-задачи и ошибки перфорации.

(Б) Ошибки в задании управления системой.

(В) Ошибки в данных.

(Г) Синтаксические ошибки в программе.

(Д) Семантические ошибки.

(Е) Нарушение количественных ограничений.

(Ж) Ошибки, обнаруживаемые при выполнении программы.

(З) Сообщения ОС Диспак, связанные с работой системы Альфа-6.

(И) Архивные сообщения.

Каждая группа сообщений идентифицируется заглавной буквой в круглых скобках. Эта буква указывается на полях слева при описании каждого сообщения (но не в выдаче этого сообщения на АЦПУ), отсылая пользователя к описанию этой группы в настоящей главе.

Ниже следует общая информация для каждой группы сообщений:

Сообщения группы (А) об ошибках комплектации альфа-задачи и перфорации фрагментов выдаются на первом этапе работы системы и относятся сразу ко всей последовательности заданий.

Для локализации места ошибки в некоторых сообщениях указывается номер стандартной карты (Приложение 10). Нужная стандартная карта отсчитывается от начала системного пакета.

Сообщениям об ошибках в комплектации и перфорации одного фрагмента предшествует подзаголовок:

ОШИБКИ фрагмент имя ПОСЛЕ номер СТАНДАРТНОЙ КАРТЫ.

Подзаголовок выдается на середине строки (Приложение 18). Первые два параметра идентифицируют тип и имя фрагмента.

Некоторые сообщения содержат следующий контекст:

...ФИЗИЧЕСКАЯ ПЕРФОКАРТА номер...

что указывает на номер физической перфокарты во фрагменте, если первой считать карту, следующую за стандартной.

Тексты сообщений об ошибках перфорации фрагмента содержат контекст:

...ФИЗИЧЕСКАЯ ПЕРФОКАРТА номер, СТРОКА номер, НОМЕР СИМВОЛА номер ...

Три параметра в контексте означают: номер физической перфокарты фрагмента; номер строки на указанной перфокарте; номер символа на указанной строке перфокарты.

При ошибке перфорации дальнейшее выполнение задания зависит от характера работ и типа ошибочного фрагмента. На месте ошибки перфорации в текст фрагмента вставляется символ ! как сигнал ошибки.

При ошибке перфорации в программе трансляция прекращается после синтаксического контроля программы. Такая ошибка не отменяет записи в архив и перфорации. Однако использование таких программ требует замены перфокарт, содержащих символы !, с помощью команд * ЗАМЕНИТЬ или * ЗАМЕНИТЬ _ТЕКСТ.

Аналогичным образом можно исправлять ошибки перфорации в данных. Выполнение программы по ошибочным данным не производится, однако запись в архив и перфорация не отменяется.

При ошибке перфорации в системной программе задание не выполняется. Реакция на ошибку перфорации во фрагменте замен производится так же, как на ошибку в редактируемой программе (или данных).

После выдачи всех сообщений группы (А) происходит распечатка текстов тех перфокарт последовательности заданий, в которых были обнаружены ошибки перфорации.

Сообщения группы (Б) об ошибках в задании работ выдаются монитором системы Альфа-6. При ошибке в задании с программой выполнение задания обычно прекращается после синтаксического контроля. При наличии серьезных ошибок в задании редактирования запись в архив и перфорация программ (данных) отменяется.

Сообщения группы (В) об ошибках в данных выдаются после распечатки системной программы и данных. Выполнение программы по ошибочным данным не производится.

Сообщения группы (Г) о синтаксических ошибках в программе выдаются при предварительном и синтаксическом контроле, соответственно, на первом и втором просмотре программы.

При обнаружении синтаксических ошибок при первом просмотре в общем случае осуществляется редактирование, распечатка программы, запись программы в архив, перфорация программы, после чего выполнение задания прекращается.

В сообщениях синтаксического контроля могут использоваться параметры:

> «символ или простая конструкция» — есть символ языка альфа-6, либо одна из конструкций: ⟨ИДЕНТИФИКАТОР⟩, ⟨СТРОКА⟩, ⟨ЧИСЛО⟩, ⟨ЛОГИЧЕСКОЕ ЗНАЧЕНИЕ⟩, , ..., , = ... =, < ... <, ⩽ ... ⩽, > ... >, ⩾ ... ⩾, : = ... : =.
>
> «синт» — ошибочный контекст программы в виде последовательности символов и металингвистических переменных.

Пример 1. Переменная с индексом $A[i, j]$ будет представлена параметром *синт* в виде:

⟨ИДЕНТИФИКАТОР⟩ [⟨СПИСОК ПЕРЕМЕННЫХ⟩]

Как правило, обнаружить место ошибки в программе можно по номеру перфокарты (или диапазону номеров) в сообщении. Параметр *синт* уточняет вид синтаксически неверной конструкции и дает дополнительную информацию для поиска ошибки.

Пример 2. На перфокарте 7 в программе пропущено второе слагаемое:

◇7◇ $x: = a +;$

Система выдает следующее сообщение:

7 СИНТАКСИЧЕСКИ НЕВЕРНАЯ КОНСТРУКЦИЯ: ⟨ИДЕНТИФИКАТОР⟩ +.

Сообщение с таким текстом, как правило, означает пропуск части текста в программе.

Пример 3. Пусть перфокарта 5 в программе содержит текст:

◇5◇ $x: = B(y,);$

Система выдает следующее сообщение:

5 ОШИБКА В СПИСКЕ ФАКТИЧЕСКИХ ПАРАМЕТРОВ: ⟨ИДЕНТИФИКАТОР⟩ (⟨СПИСОК ИНДЕКСОВ⟩).

Если $B(y,)$ есть обращение к процедуре-функции, то ошибка в том, что отсутствует второй фактический параметр; список фактических параметров принимает здесь вид списка индексов. Если же $B(y,)$ есть переменная с пустым вторым индексом, то тогда вместо скобок [и] ошибочно используются круглые скобки.

Необходимо отметить, что представление конструкции в параметре *синт* иногда не соответствует синтаксису языка альфа-6.

Пример 4. Пусть в программе имеется оператор цикла:

◇9◇ **для** $J := 1, \ldots, N$ **цикл начало** $(a + B) < N$ **то**...

Из текста видно, что после **начало** пропущен символ **если**. Система выдает следующее сообщение:

9. НЕСОВМЕСТНЫ СИМВОЛЫ ';' И '<' В КОНТЕКСТЕ: ⟨ПЕРВИЧНОЕ ВЫРАЖЕНИЕ⟩ <.

На самом деле должна быть выдана несовместность символов **начало** и <. Указанное несоответствие вызвано тем, что при синтаксическом контроле символ **начало** для тела цикла заменяется конструкцией **начало** ⟨НАЧАЛО БЛОКА⟩;.

Реакция системы на ошибки при синтаксическом контроле.

При обнаружении ошибки оператор или описание, в котором допущена ошибка, пропускается без контроля. Дальнейший просмотр программы производится с целью выявления других ошибок. При завершении синтаксического контроля выполнение задания прекращается.

Сообщения группы (Д) о семантических ошибках в программе выдаются при синтаксическом и семантическом контроле. Если в операторе (или описании) обнаруживается ошибка, то контроль других ошибок в этом операторе не производится. Дальнейший просмотр программы направлен на выявление других ошибок. При завершении просмотра обработка задания прекращается.

Сообщения группы (Е) о нарушении количественных ограничений выдаются при предварительном, синтаксическом и семантическом контроле и могут относиться к программе, данным, системной программе и заданию в целом. При нарушении количественного ограничения выполнение задания прекращается. Запись в архив и перфорация отменяются.

Сообщения группы (Ж) выдаются при выполнении программы. По типу реакции системы обнаруживаемые ошибки делятся на два класса. Если сообщение об ошибке начинается символом *, то обнаружение этой ошибки не прерывает выполнения программы. Сообщения всех других ошибок начинаются с !. При их обнаружении выполнение программы прекращается и выдается сбойная распечатка (11.3).

Рекомендация при исчерпании ресурсов ОЗУ под задачу.

1) В блоке, содержащем несколько длинных массивов, следует попытаться образовать несколько параллельных блоков, распределив между ними массивы исходного блока. Измененная таким образом программа будет использовать меньше оперативной памяти, поскольку для массивов параллельных блоков используется один и тот же участок оперативной памяти.

2) Если в статическом массиве при выполнении программы используются не все элементы, то для размещения массива потребуется меньше памяти, если описать его с динамическими границами.

3) Если задачу невозможно разместить в оперативной памяти, следует использовать средства сегментации Альфа-6 (пп. 2.19, 2.20) или МС Дубна [11, 13].

Сообщениями группы (З) являются те сообщения ОС Диспак, которые имеют специфическое толкование для системы Альфа-6:

EXP(x) $x \geqslant 44$.

(З) Комментарий. Сообщение может означать, что при возведении в степень получено число больше $0.9223372036_{10} + 19$ — максимально представимого в БЭСМ-6.

ИСТЕК. ВРЕМЯ ПО ЭК.

(З) Комментарий. Исчерпано время, заказанное в паспорте задачи. Оставшиеся 10 сек зарезервированы для сбойной распечатки.

НЕТ ПРИЗН. ВЫВ

(З) Комментарий. В паспорте системного пакета отсутствует раздел ВЫВОД, который необходим при наличии системных команд * ПЕРФОРИРОВАТЬ_ПРОГРАММУ или * ПЕРФОРИРОВАТЬ_ДАННЫЕ

Сообщения группы (И), выдаваемые при записи в архив и при считывании из архива, содержат в тексте идентифицирующую часть:

тип записи имя АРХИВ имя

В ней указывается тип записи (программа или данные) и ее имя в архиве, а также имя архива, к которому производится обращение. В случае обнаружения ошибки при обращении к архиву выполнение задания прекращается.

Замечание. При работе системы могут выдаваться сообщения:

ОШИБКА СИСТЕМЫ ИЛИ СБОЙ МАШИНЫ.
В БЛОКЕ номер ПО АДРЕСУ адрес ОБНАРУЖЕНА ОШИБКА

Эти сообщения выдаются при сбое БЭСМ-6, либо при ошибке системы; последнее часто выдается при переполнении таблиц транслятора на больших программах.

ГЛАВА 18

СООБЩЕНИЯ ПРЕДВАРИТЕЛЬНОГО КОНТРОЛЯ

БОЛЬШОЕ ВОСЬМЕРИЧНОЕ ЧИСЛО В СИСТЕМНОЙ КОМАНДЕ

(Б) Комментарий. Число содержит больше 16 восьмеричных цифр.

БОЛЬШОЕ ЧИСЛО В ДАННЫХ.

(Б) Комментарий. Значение числа в данных по абсолютной величине $> 0.9223372036_{10}19$ — максимально представимого числа в БЭСМ-6.

В ЗАДАНИИ С ПРОГРАММОЙ НЕДОПУСТИМЫ ВСПОМОГАТЕЛЬНЫЕ РАБОТЫ С АРХИВАМИ.

(Б) Комментарий. В задании определена программа (п. 5.1), а также имеется одна из команд:

* ЗАПИСАТЬ _ ОГЛАВЛЕНИЕ, * ПЕЧАТАТЬ _ ОГЛАВЛЕНИЕ, * УНИЧТОЖИТЬ _ ПРОГРАММУ, * УНИЧТОЖИТЬ _ ДАННЫЕ, * ИЗМЕНИТЬ _ ИМЯ _ АРХИВА, что несовместимо (п. 4.4).

В ЗАДАНИИ ЧИСЛО ОПЕРАЦИЙ С АРХИВАМИ > 50.

(Е) Комментарий. Каждая из операций определяется одной из следующих команд:

* ЗАПИСАТЬ _ ОГЛАВЛЕНИЕ, * ЗАПИСАТЬ _ ПРОГРАММУ, * ЗАПИСАТЬ _ ДАННЫЕ, * ПРОГРАММА, * ДАННЫЕ, * УНИЧТОЖИТЬ ПРОГРАММУ, * УНИЧТОЖИТЬ ДАННЫЕ, * ПЕЧАТАТЬ _ ОГЛАВЛЕНИЕ, * ИЗМЕНИТЬ _ ИМЯ _ АРХИВА

* В ОДНОМ ЗАДАНИИ ЗАПРЕЩАЕТСЯ ПОВТОРНАЯ ЗАПИСЬ В АРХИВ ПРОГРАММЫ ИЛИ ДАННЫХ.

(Б) Комментарий. В задании имеется несколько команд * ЗАПИСАТЬ_ПРОГРАММУ (или несколько команд * ЗАПИСАТЬ_ДАННЫЕ). Все команды, кроме первой, игнорируются системой.

ВО ФРАГМЕНТЕ ЗАМЕН НЕТ ПЕРФОКАРТЫ С НОМЕРОМ, УКАЗАННЫМ В СИСТЕМНОЙ КОМАНДЕ * ЗАМЕНИТЬ ИЛИ * ВСТАВИТЬ.

(Б) Комментарий. В массиве изменений отсутствует перфокарта с номером, являющимся началом диапазона вставляемых в программу (данные) перфокарт по системной команде * ЗАМЕНИТЬ (либо * ВСТАВИТЬ)

ВСПОМОГАТЕЛЬНЫЕ РАБОТЫ С АРХИВОМ К ПРИМИТИВНОМУ АРХИВУ НЕ ПРИМЕНИМЫ.

(Б) Комментарий. В одной из команд * ЗАПИСАТЬ_ОГЛАВЛЕНИЕ, * УНИЧТОЖИТЬ_ПРОГРАММУ, * УНИЧТОЖИТЬ_ДАННЫЕ, * ПЕЧАТАТЬ_ОГЛАВЛЕНИЕ, * ИЗМЕНИТЬ_ИМЯ_АРХИВА в качестве имени архива указано имя примитивного архива.

* ДВЕ ПЕРФОКАРТЫ С ОДИНАКОВЫМ НОМЕРОМ ВО ФРАГМЕНТЕ ЗАМЕН.

(Б) Комментарий. Пусть номер перфокарты *N3* дважды встречается во фрагменте замен, и допустим, что в системной программе имеется одна или несколько команд вида * ЗАМЕНИТЬ (*N1* : *N2*, *N3* : *N4*) или * ВСТАВИТЬ (*N1*, *N3* : *N4*). Для любой из этих команд вставка из фрагмента замен происходит, начиная с первой перфокарты с номером *N3*.

* ДЛИННОЕ ИМЯ фрагмент 'имя'.

(А) Комментарий. Имя фрагмента — идентификатор не более, чем из 6 символов. Если число символов имени больше 6 и меньше 12, то оставляются первые 6 символов, а остальные удаляются. Если же число символов в имени > 12, то перед текстом фрагмента ставится ИМЯ *N*: ; где *N* — десятичная цифра.

ДЛИННЫЙ ТЕКСТ В СИСТЕМНОЙ КОМАНДЕ * ЗАМЕНИТЬ ИДЕНТИФИКАТОР ИЛИ * ЗАМЕНИТЬ ТЕКСТ.

(Е) Комментарий. Строка, являющаяся параметром указанных команд, не может содержать > 128 символов.

ДЛИННЫЙ ТЕКСТ ПРОГРАММЫ ИЛИ ДАННЫХ.

(Е) Комментарий. Число символов текста программы (или данных) после редактирования и вставки библиотечных описаний должно быть < 196600.

* ЗАДАНИЕ НЕ ИСПОЛЬЗУЕТ ФРАГМЕНТА ЗАМЕН.

(Б) Комментарий. В задании имеется фрагмент замен, но нет системной программы, либо в системной программе нет системных команд * ВСТАВИТЬ и * ЗАМЕНИТЬ.

ЗАМЕНЯЕМЫЙ ТЕКСТ (ИЛИ ИДЕНТИФИКАТОР) РАСПОЛОЖЕН БОЛЕЕ ЧЕМ НА 10 ПЕРФОКАРТАХ.

(Б) Комментарий. Нарушено ограничение: заменяемый текст или идентификатор может разбиваться не более, чем 10-ю номерами перфокарт.

Рекомендация. Убрать часть номеров перфокарт.

ЗАПРЕЩАЕТСЯ ИСПОЛЬЗОВАТЬ ДАННЫЕ В ЗАДАНИИ С ПРОГРАММОЙ.

(Б) Комментарий. В задании определены программа и данные. Однако работы с программой и работы с данными несовместимы в одном задании (4.4).

* ЗАПРЕЩЕН НОМЕР ПЕРФОКАРТЫ В СИСТЕМНОЙ ПРОГРАММЕ: ФИЗИЧЕСКАЯ ПЕРФОКАРТА номер, СТРОКА номер.

(А) Комментарий. В системной программе нельзя употреблять номера перфокарт. Ошибочно вставленный номер перфокарты удаляется.

ЗАПРЕЩЕННЫЙ МАТЕМАТИЧЕСКИЙ НОМЕР МАГНИТОФОНА ИЛИ ДИСКА.

(Б) Комментарий. Запрещается использовать математические номера 30, 31, 32, 33. Эти номера заняты системой.

ЗАТЕРТА ПРОГРАММА В АРХИВЕ.

(Б) Комментарий. Испорчены одна или несколько зон на МЛ или МД, отведенных под программу в архиве.

* ЗНАЧАЩАЯ ЧАСТЬ ЧИСЛА В ДАННЫХ СОДЕРЖИТ > 13 ЦИФР.

(В) Комментарий. Точность представления чисел в БЭСМ-6 определяется первыми 13-ю значащими цифрами числа. Это означает, что все остальные цифры в числе не влияют на значение числа в машинном представлении.

ИДЕНТИФИКАТОР ОДНОВРЕМЕННО ЗАМЕНЯЕТСЯ НА РАЗНЫЕ ТЕКСТЫ.

(Б) Комментарий. Имеется две команды * ЗАМЕНИТЬ _ ИДЕНТИФИКАТОР, у которых первый параметр — идентификатор — совпадает, а вторые параметры различны. Кроме того, в тексте программы этот идентификатор принадлежит диапазонам обеих команд.

ИСПОРЧЕНЫ ДАННЫЕ В АРХИВЕ.

(Б) Комментарий. Испорчены одна или несколько зон на МЛ или МД, отведенных под данные в архиве.

КОМАНДА * ЗАПИСАТЬ ДАННЫЕ НЕДОПУСТИМА В ЗАДАНИИ С ПРОГРАММОЙ.

(Б) Комментарий. В задании определена программа и имеется команда * ЗАПИСАТЬ _ ДАННЫЕ.

КОМАНДА * ЗАПИСАТЬ ОГЛАВЛЕНИЕ ЗАДАЕТ ЧИСЛО ЗОН В АРХИВЕ > 1777.

(Б) Комментарий. Восьмеричное число во втором параметре команды > 1777, что превышает максимально возможное число зон на МЛ или МД.

КОМАНДА * ЗАПИСАТЬ ПРОГРАММУ НЕДОПУСТИМА В ЗАДАНИИ С ДАННЫМИ.

(Б) Комментарий. В задании определены данные и имеется команда * ЗАПИСАТЬ _ ПРОГРАММУ.

КОМАНДА * ПРОЦЕДУРА ЗАДАЕТ НЕСКОЛЬКО БИБЛИОТЕЧНЫХ ОПИСАНИЙ ИЗ ПРИМИТИВНОГО АРХИВА 'имя'.

(Б) Комментарий. Последний параметр команды * ПРОЦЕДУРА является именем примитивного архива. Число параметров в команде больше двух. Это значит, что в архиве должно храниться более одного библиотечного описания, что противоречит определению примитивного архива (п. 12.2).

ЛИШНЯЯ ЗАКРЫВАЮЩАЯ КАВЫЧКА СТРОКИ.

(Г) Комментарий. В программе (данных) пропущена открывающая кавычка строки, либо присутствует лишняя закрывающая кавычка строки.

ЛИШНЯЯ номер-я СКОБКА 'КОНЕЦ'.

(Г) Комментарий. Нарушен баланс скобок **начало — конец** в программе. Параметр *номер* в тексте сообщения указывает порядковый номер скобки **конец** в программе.

ЛИШНЯЯ ОТКРЫВАЮЩАЯ КАВЫЧКА СТРОКИ.

(Г) Комментарий. Число открывающих кавычек строки больше числа закрывающих кавычек. Номер перфокарты в сообщении указывает местонахождение лишней открывающей кавычки.

ЛИШНЯЯ СКОБКА 'НАЧАЛО'.

(Г) Комментарий. В программе число скобок **начало** больше числа скобок **конец**. Номер перфокарты в сообщении указывает местоположение лишней скобки **начало**.

МНОГО БИБЛИОТЕЧНЫХ ОПИСАНИЙ.

(Е) Комментарий. Число команд * ПРОЦЕДУРА в сумме с числом параметров этих команд должно быть < 250.

МНОГО ДАННЫХ.

(Е) Комментарий. Для размещения данных, определенных в задании, требуется > 32768 ячеек. Необходимо учитывать, что число, логическое значение и БЭСМ-данные занимают по одной ячейке памяти. Каждый символ строки занимает одну или несколько ячеек. Представление символов строки см. в Приложении 7.

МНОГО СИМВОЛОВ ТЕКСТА В ПОСЛЕДОВАТЕЛЬНОСТИ ЗАДАНИЙ.

(А) Комментарий. Общее число символов на перфокартах для альфа-задачи (последовательность заданий), исключая стандартные карты, должно быть $\leqslant 98200$.

МНОГО СИСТЕМНЫХ КОМАНД РЕДАКТИРОВАНИЯ.

(Е) Комментарий. Число системных команд редактирования в одном задании должно быть < 128.

НЕВЕРЕН ИДЕНТИФИКАТОР СИСТЕМНОЙ КОМАНДЫ.

(Б) Комментарий. Последовательность символов в системной программе от * до (либо от * до * не является идентификатором системной команды. Причиной ошибки может оказаться неправильное сокращение идентификатора системной команды (Приложение 11). Возможен также пропуск скобки (либо отсутствие * у идентификатора следующей системной команды.

НЕВЕРНАЯ КОМПЛЕКТАЦИЯ ЗАДАНИЯ.

(Б) Комментарий. Возможны следующие ошибки:

— более одной программы в задании;

— несколько системных программ в задании;

— несколько фрагментов замен в задании;

— нет одновременно альфа-программы и системной программы в задании.

Причиной таких ошибок может быть отсутствие стандартной карты ЗАДАНИЕ, либо ее неправильное вхождение в последовательность заданий.

НЕВЕРНОЕ ИМЯ В ОПЕРАНДЕ СИСТЕМНОЙ КОМАНДЫ.

(Б) Комментарий. Возможны следующие ошибки:

— вместо имени другая конструкция;

— в имени больше шести символов;

— первый символ имени не буква.

НЕВЕРНО ИСПОЛЬЗУЕТСЯ СТАНДАРТНАЯ КАРТА АЛГИБР.

(А) Комментарий. Возможны следующие ошибки:

— после стандартной карты СИСТЕМА следует карта АЛГИБР;

— после стандартной карты ЗАМЕНЫ следует карта АЛГИБР.

НЕВЕРНЫЙ НОМЕР ИНДЕКС-РЕГИСТРА В СИСТЕМНОЙ КОМАНДЕ * РЕГИСТРЫ.

(Б) Комментарий. В операнде системной команды * РЕГИСТРЫ употребляется номер регистра $> 17_8$.

НЕВЕРНЫЙ НОМЕР ПЕРФОКАРТЫ В ОПЕРАНДЕ СИСТЕМНОЙ КОМАНДЫ.

(Б) Комментарий. Синтаксически неверный номер перфокарты в операнде системной команды редактирования, отладки и печати. Возможно также, что в операнде находится не номер перфокарты, а другая конструкция, хотя по смыслу должен быть номер перфокарты.

Пример. Данное сообщение будет выдано для команды * ЗАМЕНИТЬ (*'a5'*, *'5'*, *30*).

Ошибка в том, что вместо команды * ЗАМЕНИТЬ — ТЕКСТ употребляется команда * ЗАМЕНИТЬ, а для нее первым операндом должен быть номер перфокарты.

НЕВЕРНЫЙ СИМВОЛ: ФИЗИЧЕСКАЯ ПЕРФОКАРТА номер, СТРОКА номер, НОМЕР СИМВОЛА номер.

(А) Комментарий. Это сообщение фиксирует ошибку перфорации. Данного символа нет в алфавите языка альфа-6. Таблицы кодировок для УПП и КУ-3 см. в Приложении 1.

НЕ ВОСЬМЕРИЧНАЯ ЦИФРА В ОПЕРАНДЕ СИСТЕМНОЙ КОМАНДЫ.

(Б) Комментарий. Сообщение может быть выдано для команд * АРХИВ, * ОБЛАСТЬ, * ЗАПИСАТЬ_ОГЛАВЛЕНИЕ, * РЕГИСТРЫ, операнды которых содержат восьмеричные константы.

Пример. Ошибочна команда * РЕГИСТРЫ (9, 10).

НЕКОРРЕКТНО ИСПОЛЬЗУЕТСЯ КОМАНДА * ПЕРФОРИРОВАТЬ ПРОГРАММУ ИЛИ * ПЕРФОРИРОВАТЬ ДАННЫЕ.

(Б) Комментарий. В задании с программой используется команда * ПЕРФОРИРОВАТЬ_ДАННЫЕ, либо в задании с данными используется команда * ПЕРФОРИРОВАТЬ_ПРОГРАММУ.

НЕ ОПИСАН АРХИВ 'имя'.

(Б) Комментарий. В системной программе есть системные команды, которые обращаются к архиву. Однако нет команды * АРХИВ, описывающей архив с указанным именем.

НЕ ОПРЕДЕЛЕН ВИД КОДИРОВКИ номер ПОСЛЕ номер СТАНДАРТНОЙ КАРТЫ.

(А) Комментарий. В карте вида кодировки неверно указан вид устройства перфорации, либо неверная стандартная карта (приложение 1).

НЕПОЛНОЕ ЗАДАНИЕ.

(Б) Комментарий. В задании не определено ни программы, ни данных и нет ни одной команды вспомогательных архивных работ (п. 4.4).

НЕСКОЛЬКО ВАРИАНТОВ ДАННЫХ В ОДНОМ ЗАДАНИИ НЕДОПУСТИМЫ.

(Б) Комментарий. В задании допустим либо один фрагмент-данные, либо один вариант данных из архива.

Рекомендация. Если необходимо просчитать по нескольким вариантам данных, то следует использовать способ, предложенный в п. 8.1.

НЕСКОЛЬКО ПРОГРАММ В ОДНОМ ЗАДАНИИ.

(Б) Комментарий. В задании определено более одной программы, что недопустимо.

НЕСОВМЕСТИМЫЕ РАБОТЫ В ЗАДАНИИ С АРХИВАМИ.

(Б) Комментарий. Нарушено требование о недопустимости определения в задании работ, принадлежащих различным видам заданий (п. 4.4). В задании вместе с командами вспомогательных работ с архивами (п. 4.4) используются команды, задающие работы с программой или данными.

НЕСОВМЕСТИМЫЕ РАБОТЫ В ЗАДАНИИ С ДАННЫМИ.

(Б) Комментарий. Нарушено требование о недопустимости определения в задании работ, принадлежащих различным видам заданий (п. 4.4). В задании определены данные, а также используются команды, задающие работы с программой, либо команды, задающие вспомогательные работы с архивами (п. 4.4).

* НЕТ ИМЕНИ фрагмент ПОСЛЕ номер СТАНДАРТНОЙ КАРТЫ.

(А) Комментарий. Указывается порядковый номер стандартной карты в последовательности заданий. Система ставит имя ИМЯ*N*, где *N* — десятичная цифра.

НЕТ КАРТ В ЗАДАНИИ.

(А) Комментарий. Ошибочная ситуация: после карты * CALL _ ALPHA/6 в пакете следует карта ◇◇◇ КНЦ или карта * END _ FILE.

НЕТ КОДА ОПЕРАЦИИ В ОПЕРАНДЕ КОД: ФИЗИЧЕСКАЯ ПЕРФОКАРТА номер, СТРОКА номер, НОМЕР СИМВОЛА номер.

(Г) Комментарий. Сообщение фиксирует ошибку в операторе КОД в алгибр-программе. Возможно указывается неверный код операции, либо пропущена запятая после кода операции.

НЕТ ОТКРЫВАЮЩЕЙ КАВЫЧКИ СТРОКИ В ОПЕРАНДЕ СИСТЕМНОЙ КОМАНДЫ.

(Б) Комментарий. Нет открывающей кавычки строки в первом или втором операнде команды * ЗАМЕНИТЬ — ТЕКСТ, либо во втором операнде команды * ЗАМЕНИТЬ— ИДЕНТИФИКАТОР.

НЕТ ПЕРВОЙ СКОБКИ 'НАЧАЛО' ПОСЛЕ ИМЕНИ ПРОГРАММЫ.

(Г) Комментарий. В альфа-программе после имени программы (и, возможно, номера перфокарты) должен следовать символ **начало**.

НЕТ СТАНДАРТНОЙ КАРТЫ ПОСЛЕ КОНЦА ФРАГМЕНТА, ФИЗИЧЕСКАЯ ПЕРФОКАРТА номер.

(А) Комментарий. После ◇ * следует нестандартная карта.

НЕТ ФРАГМЕНТА ЗАМЕН.

(Б) Комментарий. В задании нет фрагмента замен, однако в системной программе есть системные команды * ЗАМЕНИТЬ или * ВСТАВИТЬ.

НЕТ фрагмент ПОСЛЕ номер СТАНДАРТНОЙ КАРТЫ.

(А) Комментарий. После стандартной карты СИСТЕМА, ПРОГРАММА, ДАННЫЕ ЗАМЕНЫ или АЛГИБР следует карта ◇◇◇ КНЦ или карта * END — FILE.

НЕТ ';' В КОНЦЕ ДАННЫХ.

(В) Комментарий. В конце последней группы данных нет «;» (либо нет контрольной суммы в случае алгибр-данных).

* ОДНОВРЕМЕННО ЗАДАНА ПЕЧАТЬ ПРОГРАММЫ И ДАННЫХ.

(Б) Комментарий. В задании имеются команды * ПЕЧАТАТЬ— ПРОГРАММУ и * ПЕЧАТАТЬ — ДАННЫЕ.

ОШИБКА В БИБЛИОТЕЧНОМ ОПИСАНИИ.

(Б) Комментарий. Возможны следующие ошибки:

— затирание личного архива, куда была записана программа с библиотечной частью;

— затирание библиотечного архива;

— нет баланса скобок **начало** — **конец** в библиотечном описании.

ОШИБКА В БЭСМ-ДАННЫХ.

(В) Комментарий. Возможны следующие ошибки:

— первая или десятая цифра в БЭСМ-команде не двоичная;

— в БЭСМ-шкале цифры, отличные от двоичных;

— недопустимый символ в БЭСМ-данных;

— нет ; в конце группы БЭСМ-данных;

— в БЭСМ-константе, либо в БЭСМ-команде встретилась не восьмеричная цифра.

ОШИБКА В БЭСМ-КОМАНДЕ В ДАННЫХ.

(В) Комментарий. Возможны следующие ошибки:

— первая или десятая цифра БЭСМ-команды не двоичная;

— третий или двенадцатый символ БЭСМ-команды > 3; после восемнадцати цифр за символом *к* нет символа ; , либо *к*.

ОШИБКА В БЭСМ-КОНСТАНТЕ В ДАННЫХ.

(В) Комментарий. Возможны следующие ошибки:

— БЭСМ-константа содержит более шестнадцати восьмеричных цифр;

— недопустимый символ в БЭСМ-константе;

— нет ; , либо символа *с* после БЭСМ-константы.

ОШИБКА В БЭСМ-ШКАЛЕ В ДАННЫХ.

(В) Комментарий. Возможны следующие ошибки:

— в БЭСМ-шкале $>$ 48 цифр;

— недопустимый символ в БЭСМ-шкале;

— после БЭСМ-шкалы нет ни ; , ни символа *б*.

ОШИБКА В ДАННЫХ.

(В) Комментарий. Возможны следующие ошибки:

— первый символ после ; не соответствует ни комментарию, ни какому-либо виду данных;

— данные разделяет не ; и не , , либо есть недопустимый символ внутри данных;

— после закрывающей кавычки строковых данных нет ;.

ОШИБКА В ДАННЫХ ЛИБО НЕВЕРНЫЙ КОММЕНТАРИЙ.

(В) Комментарий. Возможны следующие ошибки:

— недопустимый символ в БЭСМ-данных;

— недопустимый символ в числовых данных;

— комментарий не завершается символами: : = , = , : .

ОШИБКА В ДИАПАЗОНЕ СИСТЕМНОЙ КОМАНДЫ * ВЫБРОСИТЬ ИЛИ * ЗАМЕНИТЬ.

(Б) Комментарий. В программе (данных) не встретился номер перфокарты, соответствующий концу диапазона удаляемых перфокарт в одной из команд * ВЫБРОСИТЬ или * ЗАМЕНИТЬ. Поэтому были удалены перфокарты от начала диапазона до конца программы (данных).

ОШИБКА В ДИАПАЗОНЕ СИСТЕМНОЙ КОМАНДЫ РЕДАКТИРОВАНИЯ, НЕ ВЫПОЛНЯЕТСЯ СИСТЕМНАЯ КОМАНДА имя команды.

(Б) Комментарий. В программе (данных) не встретился номер перфокарты, соответствующий началу диапазона данной команды. Поэтому редактирование, предписываемое этой командой, не производилось. Возможно, что номер начала диапазона ошибочно оказался в диапазоне одной из команд поперфокартного редактирования (п. 10.1).

ОШИБКА В НОМЕРЕ КАНАЛА В ДАННЫХ.

(В) Комментарий. Номер канала в данных должен быть целым положительным числом (без знака) < 15. Возможно пропущена звездочка после номера канала.

ОШИБКА В НОМЕРЕ ПЕРФОКАРТЫ: ФИЗИЧЕСКАЯ ПЕРФОКАРТА номер, СТРОКА номер.

(А) Комментарий. Синтаксически неверный номер перфокарты заменяется системой на стандартный номер ◇999999◇. Возможно также неверное представление греческих букв в кодировке УПП.

Пример. Неверны номера:

◇1.2.3◇

◇1111111◇

ОШИБКА В номер-й СТАНДАРТНОЙ КАРТЕ ИЛИ НЕВЕРНО СОСТАВЛЕНА ПОСЛЕДОВАТЕЛЬНОСТЬ ЗАДАНИЙ.

(А) Комментарий. Система не может определить тип стандартной карты. В сообщении указывается порядковый номер стандартной карты в пакете.

ОШИБКА В ПОРЯДКЕ ЧИСЛА В ДАННЫХ.

(В) Комментарий. Возможны следующие ошибки:

— после знака порядка $_{10}$ нет ни цифры, ни +, ни —;

— больше шести цифр в порядке числа.

ОШИБКА В СИСТЕМНОЙ КОМАНДЕ * АРХИВ.

(Б) Комментарий. Возможны следующие ошибки:

— нет запятой после первого операнда, либо недопустимый символ в первом операнде;

— номер направления МЛ или МД не в диапазоне от 3 до 6;

— восьмеричный номер начальной зоны архива на МЛ (или МД), указанный в третьем операнде, $>$ 1777;

— ошибка в номере начальной зоны примитивного архива — третий символ во втором параметре не 0 и не 1;

— нет скобки).

ОШИБКА В СИСТЕМНОЙ КОМАНДЕ * ОБЛАСТЬ.

(Б) Комментарий. Возможны следующие ошибки:

— неверный номер направления носителя (должен быть от 1 до 6);

— после двух восьмеричных цифр в первом параметре нет запятой;

— указан номер зоны $\geqslant$ 1024;

— последовательность цифр номера зоны не заканчивается запятой;

— указан сдвиг $\geqslant$ 1024.

ОШИБОЧНЫЙ СИМВОЛ В ПРОГРАММЕ (ДАННЫХ) ИЗ АРХИВА.

(Г) Комментарий. В программе (данных), читаемой из архива, имеется символ !. Это означает, что ранее программа (данные) была записана в архив с ошибочным символом.

Пример. Командой

* ЗАМЕНИТЬ— ТЕКСТ ('_кон!', 'конец', 145)

можно заменить текст '_кон!' служебным словом конец. Текст _'кон!' возник в архиве из-за ошибки четности при перфорации символа — на УПП.

ПЕРВОЕ ЗАДАНИЕ НЕ НАЧИНАЕТСЯ СТАНДАРТНОЙ КАРТОЙ.

(А) Комментарий. После карты * CALL— ALPHA/6 нет стандартной карты АЛЬФА-6.

ПЕРЕД ◇ * НЕТ СИМВОЛА 'конец', ЛИБО НЕВЕРНО ВСТАВЛЕНА СТАНДАРТНАЯ КАРТА.

(А) Комментарий. Текст программы должен кончаться символом 'конец' — не допускается комментарий после этого символа.

ПЕРЕСЕКАЮТСЯ ДИАПАЗОНЫ В СИСТЕМНЫХ КОМАНДАХ * ВСТАВИТЬ, * ВЫБРОСИТЬ, * ЗАМЕНИТЬ.

(Б) Комментарий. В системной программе имеются две команды поперфокартного редактирования, у которых номер перфокарты начала диапазона (в первом параметре команды) совпадает.

* ПЕРЕСЕКАЮТСЯ ЗАМЕНЯЕМЫЕ ТЕКСТЫ.

(Б) Комментарий. Конец заменяемого текста одной команды * ЗАМЕНИТЬ— ТЕКСТ является началом заменяемого текста другой команды * ЗАМЕНИТЬ— ТЕКСТ.

Пример. В командах

* ЗАМЕНИТЬ—ТЕКСТ ('тори', '0', 10 : 30)

* ЗАМЕНИТЬ—ТЕКСТ ('мотор', 'п', 3 : 20)

текст «тор» является началом заменяемого текста одной команды и концом другой. Если в перфокартах программы 10—20 встретится текст «моторика», то редактирование произойдет по второй команде.

Результирующий текст: «пика».

ПЕРФОКАРТА КОНЦА ДИАПАЗОНА ВСТРЕТИЛАСЬ РАНЬШЕ ПЕРФОКАРТЫ НАЧАЛА ДИАПАЗОНА.

(Б) Комментарий. Это сообщение может быть выдано для системных команд * ЗАМЕНИТЬ— ТЕКСТ, * ЗАМЕНИТЬ— ИДЕНТИФИКАТОР (пп. 10.2, 10.3).

ПОВТОРНОЕ ОПИСАНИЕ АРХИВА 'имя'.

(Б) Комментарий. Две системные команды * АРХИВ в одной системной программе описывают архивы с одним и тем же именем.

СИНТАКСИЧЕСКАЯ ОШИБКА В СИСТЕМНОЙ ПРОГРАММЕ.

(Б) Комментарий. Возможны следующие ошибки:

— пропуск скобки) после операндов системной команды;

— недопустимое вхождение * в операндах системной команды;

— лишний операнд в системной команде;

— недопустимый символ в последнем операнде.

СИСТЕМНЫХ КОМАНД * ПЕЧАТАТЬ МОДУЛЬ > 64.

(Е) Комментарий. В системной программе задания число команд * ПЕЧАТАТЬ _ МОДУЛЬ должно быть $\leqslant 64$.

ФРАГМЕНТА 'имя' НЕТ В ДРУГИХ ЗАДАНИЯХ.

(Б) Комментарий. В задании имеется команда * ПРОГРАММА (⟨имя⟩) или * ДАННЫЕ (⟨имя⟩). Согласно (п. 5.1) указанная программа (или данные) должна находиться на перфокартах в другом задании из последовательности заданий.

ФРАГМЕНТ ЗАМЕН НЕ НАЧИНАЕТСЯ НОМЕРОМ ПЕРФОКАРТЫ.

(Б) Комментарий. Текст фрагмента замен должен начинаться номером перфокарты, который задается математиком. Отметим, что в начале фрагмента замен недопустимо ⟨имя⟩:.

ЧИСЛО В ДАННЫХ ПО АБСОЛЮТНОЙ ВЕЛИЧИНЕ $<$ МИНИМАЛЬНОГО ПРЕДСТАВИМОГО В БЭСМ-6.

(В) Комментарий. Число в данных по абсолютной величине $<_{10} - 19$.

ЧИСЛО КОМАНД * АРХИВ В ЗАДАНИИ > 100.

(Е) Комментарий. Нельзя определять в одном задании более 100 различных архивов.

ЧИСЛО СИМВОЛОВ В ИДЕНТИФИКАТОРЕ > 96.

(Е) Комментарий. Нарушено ограничение на длину идентификатора.

ЧИСЛО СИМВОЛОВ В СТРОКЕ > 512 ЛИБО НЕТ ЗАКРЫВАЮЩЕЙ КАВЫЧКИ.

(Е) Комментарий. Нарушено ограничение на длину строки в программе.

ЧИСЛО СИМВОЛОВ СТРОКИ В ДАННЫХ > 512 ЛИБО НЕТ ЗАКРЫВАЮЩЕЙ КАВЫЧКИ.

(Е) Комментарий. Нарушено ограничение на длину строки в данных.

ЧИСЛО СИСТЕМНЫХ КОМАНД ОТЛАДКИ > 128.

(Е) Комментарий. Совокупное число системных команд * ЗНАЧЕНИЕ и * ПЕЧАТАТЬ — МЕТКИ в одном задании должно быть $\leqslant 128$.

ГЛАВА 19

СООБЩЕНИЯ СИНТАКСИЧЕСКОГО КОНТРОЛЯ

БОЛЬШАЯ ВЛОЖЕННОСТЬ БЛОКОВ И ЦИКЛОВ.

(Е) Комментарий. Глубина вложенности блоков и циклов должна быть < 64. Понятие вложенности иллюстрируется на примере.

Пример. Блок, помеченный меткой М , содержит цикл по K, в котором содержится цикл по J. Следовательно, глубина вложенности $= 3$.

М: **начало целый** K, J; **для** K: $= 1, \ldots, N$ **цикл для** J: $= 1, \ldots, 5$ **цикл** AC **конец**;

БОЛЬШАЯ ВЛОЖЕННОСТЬ СИНТАКСИЧЕСКИХ КОНСТРУКЦИЙ.

(Е) Комментарий. Глубина вложенности синтаксических конструкций в программе должна быть < 100. Понятие вложенности синтаксических конструкций иллюстрируется на примере.

Пример. Рассмотрим вхождение индекса $J - 1$ в следующем фрагменте программы:

◇5◇ **начало для** J: $= 1, \ldots, N$ **цикл если** $J < N - 3$ **то**

◇6◇ $A(C[J - 1])$ **иначе** $B(J)$ **конец**;

Относительно отмеченного вхождения $J - 1$ имеются следующие уровни вложенности: составной оператор, оператор цикла, условный оператор, оператор процедуры, переменная с индексами, простое арифметическое выражение. Глубина вложенности конструкций $= 6$.

* В ЗНАЧАЩЕЙ ЧАСТИ ЧИСЛА $>$ 13 ЦИФР.

(Д) Комментарий. Число в тексте исходной программы транслируется в число БЭСМ-6. На значение числа в БЭСМ-6 влияют только первые 13 значащих цифр числа в тексте программы.

В ПОРЯДКЕ ЧИСЛА $>$ 2 ЦИФР.

(Г) Комментарий. В конструкции $_{10}$⟨целое⟩ употребляется $>$ 2 цифр.

В ТЕКСТОВОМ ЭЛЕМЕНТЕ КОНСТАНТА ОПЕРАТОРА БЕМШ $>$ 6 СИМВОЛОВ.

(Д) Комментарий. В оттраслированной программе символы текстового элемента представляются в кодировке УПП.

Те символы, которых нет в кодировке УПП, представляются несколькими символами (Приложение 7). С учетом этого число символов в представлении текстового элемента не должно превышать 6.

В ЧИСЛЕ $>$ 60 ЦИФР.

(Е) Комментарий. В числе нельзя употреблять $>$ 60 цифр.

ДЛИННЫЙ ЭЛЕМЕНТАРНЫЙ ФРАГМЕНТ ТЕКСТА.

(Е) Комментарий. Нарушено одно из ограничений:

— число символов, заключенных между ближайшими символами ; и ;, должно быть $<$ 500;

— в списке граничных пар описания массивов должно быть $<$ 250 символов;

— число символов в описании верхнего индекса должно быть $<$ 100;

— в перечислении должно быть $<$ 250 символов.

ЗАПРЕЩАЕТСЯ ЛЕКСИКОГРАФИЧЕСКОЕ СОВПАДЕНИЕ ОБЕИХ ЧАСТЕЙ ПЕРЕЧИСЛЕНИЯ.

(Г) Комментарий. Правая и левая части перечисления совпадают посимвольно, что недопустимо.

Пример. Недопустимо перечисление $a[1]$, ..., $a[1]$

ИДЕНТИФИКАТОР идентификатор В ЗАГОЛОВКЕ ПРОЦЕДУРЫ — НЕ ФОРМАЛЬНЫЙ ПАРАМЕТР.

(Д) Комментарий. В заголовке процедуры возможны следующие ошибки:

— в списке значений или списке результатов встречается идентификатор, не являющийся формальным параметром;

— специфицирован идентификатор, не являющийся формальным параметром.

Пример. Скаляр y должен быть описан после скобки **начало**:

◇10◇ **процедура** $B(x)$; **целый** y; **начало**

* КОММЕНТАРИЙ В ПРОГРАММЕ: текст.

(Г) Комментарий. В качестве параметра распечатывается текст после символа **конец** до первого из символов **конец**, **иначе**, ;. Этот текст удаляется из программы при трансляции. Сообщение предупреждает о возможном пропуске символа ; после **конец**.

КОРОТКАЯ КОНСТАНТА В ОПЕРАТОРЕ БЕМШ ЗАНИМАЕТ БОЛЬШЕ ПОЛОВИНЫ ЯЧЕЙКИ.

(Д) Комментарий. Значение константы «*конк*» выходит за пределы 24 разрядов.

Пример. Константа, имеющая единицу в 25-м разряде и нули во всех остальных, не может быть короткой:

◊4◊ БЕМШ (*конк*, *м20в'20'*);

МНОГО ДЛИННЫХ ИДЕНТИФИКАТОРОВ.

(Е) Комментарий. Общее число символов во всех различающихся идентификаторах программы, состоящих из четырех символов и более, должно быть < 2000.

МНОГО ИДЕНТИФИКАТОРОВ.

(Е) Комментарий. Число различных констант в программе должно быть <1024.

МНОГО КОНСТАНТ В ПРОГРАММЕ.

(Е) Комментарий. Число различных констант в программе должно быть < 2048. Под константами понимаются числа, а также литеральные константы, короткие константы, длинные константы и восьмеричные адреса в операторах *бемш*.

МНОГО ОГРАНИЧИТЕЛЕЙ НАЧАЛО, ЦИКЛ, ...

(Е) Комментарий. Общее число ограничителей **начало**, **цикл**, **процедура**, **функция**, а также операторов *бемш* должно быть < 1000.

МНОГО ПАРАЛЛЕЛЬНЫХ ИНДЕКСОВ В ПЕРЕЧИСЛЕНИИ.

(Е) Комментарий. Число параллельных индексов в перечислении должно быть < 15. Если в перечислении имеется несколько индексов, по которым одновременно происходит пересчет, то такие индексы называются параллельными. Ниже приведен пример перечисления с тремя параллельными индексами.

Пример. Вектор, составленный из диагональных элементов трехмерного массива, можно представить следующим образом:

$|A[I, I, I], \ldots, A[N, N, N]|$

МНОГО ПЕРЕЧИСЛЕНИЙ В ПЕРЕЧИСЛЕНИИ.

(Е) Комментарий. Число перечислений, включенных в состав выражений исходного перечисления, должно быть < 16.

МНОГО ПЕРФОКАРТ В ПРОГРАММЕ.

(Е) Комментарий. Число перфокарт в программе $+n >$ 4096, где n — число простых операторов (не содержащих в себе других операторов), в тексте которых находится более одного номера перфокарты.

МНОГО СПЕЦИАЛЬНЫХ ОБЪЕКТОВ.

(Е) Комментарий. Число описанных во всех блоках программы специальных объектов должно быть < 220. Специальными объектами являются:

— метки, метящие блок;
— переменные с внутренней размерностью;
— переменные, определяемые описаниями тождества;
— собственные переменные;
— переменные с верхним индексом.

МНОГО СТРОК.

(Е) Комментарий. Общее число символов во всех строках программы (не считая строк — стандартных форматов вывода) должно быть < 4000. При подсчете числа символов в строке учитываются также кавычки строки.

Рекомендация. Как правило, в программе имеются операторы с повторяющимися форматами вывода. В этом случае рекомендуется описать несколько процедур, осуществляющих вывод объектов с общим форматом.

Пример.

процедура вывод (S, x); **строка** S; **вещественный** x;
примечание перед значением x печатается текст строки S;
output (0, '$т$', S, '*z2в3d.4d*', x);

НЕВЕРНАЯ КОНСТАНТА В ОПЕРАТОРЕ БЕМШ.

(Г) Комментарий. В короткой, длинной или литеральной константе возможна одна из следующих ошибок:

— нет цифры после $м$;
— элемент константы не начинается с $м$, $в$ либо $т$;
— значение сдвига, задаваемое числом после $м$, больше 47;

— недопустимый символ в восьмеричной или адресной константе;
— нет закрывающей кавычки в восьмеричной константе;
— ошибка в индекс-регистре адресной константы;
— нет) в адресной константе.

НЕВЕРНАЯ КОНСТАНТА, ЗАДАННАЯ ОПЕРАТОРОМ КОД АЛГИБРА.

(Г) Комментарий. В операторе *код* встречается символ, отличный от восьмеричной цифры, запятой и). Возможен также пропуск символа).

НЕВЕРНАЯ СТРУКТУРА В ОПИСАНИИ ПЕРЕМЕННЫХ С ВНУТРЕННЕЙ РАЗМЕРНОСТЬЮ.

(Г) Комментарий. В описании типа и структуры, либо в описании массивов после разделителя «—» следует символ, отличный от **массив, вектор, матрица.**

НЕВЕРНАЯ СТРУКТУРА В ОПИСАНИИ ПЕРЕМЕННЫХ С ВНУТРЕННЕЙ РАЗМЕРНОСТЬЮ: 'синт'.

(Г) Комментарий. В описании типа и структуры, либо в описании массивов порядок по внутреннему измерению задается не первичным выражением. В параметре *синт* непосредственно после символа **массив** указывается ошибочный порядок по измерению.

НЕВЕРНОЕ ОПИСАНИЕ ВЕРХНЕГО ИНДЕКСА.

(Г) Комментарий. Возможны следующие ошибки:
— нет [после ↑;
— начало рекурсии или длина рекурсии не является арифметическим выражением;
— нет];
— ошибка в метке управляющего цикла.

НЕВЕРНОЕ СЛУЖЕБНОЕ СЛОВО.

(Г) Комментарий. Конструкция _⟨последовательность букв⟩_ не соответствует ни одному из служебных символов языка альфа-6 (Приложение 3). Возможно также ошибочное вхождение в тексте символа _ (подчеркивание).

НЕВЕРНО ИСПОЛЬЗУЕТСЯ СИМВОЛ 'символ'.

(Г) Комментарий. Возможны следующие ошибки:
— символ * употребляется вне строки;

— спецификаторы **значение, результат, метка, строка** употребляются вне заголовка процедуры, либо пропущен символ **процедура** в описании процедуры;

— перед символом троеточие употребляется не запятая или $=$, $<$, $\leqslant$, $>$, $\geqslant$, $:=$.

НЕВЕРНЫЙ ИНДЕКС-РЕГИСТР В ОПЕРАТОРЕ БЕМШ.

(Г) Комментарий. В позиции индекс-регистра употребляется не восьмеричная цифра, либо указан номер индекс-регистра $> 17_8$.

НЕВЕРНЫЙ ОГРАНИЧИТЕЛЬ ПАРАМЕТРА ЛИБО ПРОПУЩЕН СИМВОЛ ПОСЛЕ ')'.

(Г) Комментарий. Возможны следующие ошибки:

— после) употребляется идентификатор, что является следствием пропуска символа;

— отсутствуют символы :, либо (в ограничителе параметра вида «) ⟨строка букв⟩ : (».

Пример. Перед обращением к процедуре *A* пропущена ; :

```
◇8◇ output (0, 'e', x, y) A(x, y);
```

НЕВЕРНЫЙ ОГРАНИЧИТЕЛЬ ПЕРЕД ОПИСАНИЕМ НАЧАЛЬНОГО ЗНАЧЕНИЯ ЛИБО ОШИБКА В ОТНОШЕНИИ.

(Г) Комментарий. Перед синтаксической конструкцией ⟨переменная⟩ = ⟨логическое выражение⟩ употребляется не ; и не **начало**.

Пример. Неверно записан условный переход по значению логического скаляра *L*:

```
◇4◇ если L = истина то на M;
```

Правильно будет:

```
◇4◇ если L то на M;
```

НЕВЕРНЫЙ СИМВОЛ В ОПИСАНИИ ПОСЛЕ ТИПА.

(Г) Комментарий. После символа **вещественный, целый, логический или комплексный** не следует буква, **массив, процедура, функция.**

НЕДОПУСТИМАЯ КОНСТРУКЦИЯ В СКОБКАХ **НАЧАЛО И КОНЕЦ**: синт.

(Г) Комментарий. В скобках **начало** и **конец** не тело блока и не последовательность операторов.

НЕДОПУСТИМОЕ ИМЯ КОМАНДЫ В ОПЕРАТОРЕ БЕМШ.

(Г) Комментарий. Идентификатора, встречающегося в качестве кода операции, нет в таблице имен команд оператора *бемш* (Приложение 13).

НЕДОПУСТИМОЕ ОПИСАНИЕ ВНЕШНЕГО МАССИВА.

(Д) Комментарий. В описании массива внешний массив должен быть описан простым массивом, т. е. не может иметь верхний индекс, быть собственным или иметь структуру.

Пример. Внешний массив *EXa* не является простым, так как имеет структуру:

◇6◇ **массив** *EXa*, *a*[*1* : *1024*] — **вектор** *3*;

НЕДОПУСТИМОЕ ПОВТОРНОЕ ВХОЖДЕНИЕ ФОРМАЛЬНОГО ПАРАМЕТРА идентификатор В ЗАГОЛОВОК ПРОЦЕДУРЫ.

(Д) Комментарий. Возможны следующие ошибки:

— повторное вхождение параметра в списке значений или в списке результатов;

— повторная спецификация параметра.

Пример. Повторно специфицирован параметр *y*:

◇7◇ **процедура** *B*(*x*, *y*); **целый** *x*, *y*; **целый** *y*;

* НЕПРАВИЛЬНО ВСТАВЛЕН КОММЕНТАРИЙ.

(Г) Комментарий. Перед символом **примечание** нет ; либо **начало**. Из программы удаляется текст комментария до символа; включительно и трансляция продолжается.

НЕСОВМЕСТИМЫ 'символ или простая конструкция' И 'символ или простая конструкция'.

(Г) Комментарий. Указанные в сообщении символы или конструкции расположены рядом в тексте программы, что синтаксически неверно.

Пример. После оператора *K*: = *4* пропущена ; :

◇5◇ *K*: = *4* **для** *J*: = *1*, ..., *N* **цикл**

Здесь будет выдано сообщение о несовместимости конструкции ⟨число⟩ и символа **для**.

НЕСОВМЕСТИМЫ 'символ или простая конструкция' И 'символ или простая конструкция' В КОНТЕКСТЕ: синт.

(Г) Комментарий. Указанные в первых двух параметрах символы или конструкции разделены в тексте программы

некоторой конструкцией. Параметр *синт* содержит эту конструкцию, окаймленную указанными символами (или конструкциями). Контекст, указанный параметром *синт*, не может встретиться в синтаксически правильной программе и является ошибочным.

Пример. У переменной с индексами $c\ [J-1]$ пропущена] :

$\Diamond 9\Diamond \quad c[1] := (1 + c[J-1) \times a;$

Система выдает сообщение:

9 НЕСОВМЕСТИМЫ '[' И ')' В КОНТЕКСТЕ:
[⟨ПРОСТОЕ АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ⟩)

НЕ СПЕЦИФИЦИРОВАН ФОРМАЛЬНЫЙ ПАРАМЕТР идентификатор, ПЕРЕЧИСЛЕННЫЙ В СПИСКЕ ЗНАЧЕНИЙ ИЛИ РЕЗУЛЬТАТОВ.

(Д) Комментарий. Параметры, описанные значением или результатом, должны быть специфицированы.

Пример. Не специфицирован целый параметр x:

процедура $B(x, y)$; **значение** x; **начало**

Должно быть:

процедура $B(x, y)$; **значение** x; **целый** x: **начало**

НЕТ МЕТКИ ПЕРЕД ДВОЕТОЧИЕМ.

(Г) Комментарий. Перед двоеточием употребляется не идентификатор и не целое без знака.

НЕТ РАЗДЕЛИТЕЛЯ '=' В ОПИСАНИИ ФУНКЦИИ-ВЫРАЖЕНИЯ.

(Г) Комментарий. После совокупности формальных параметров функции нет разделителя = .

Пример. Вместо = ошибочно используется : =

функция $A(x, y) := sqrt\ (x \uparrow 2 + y \uparrow 2)$;

НЕ ';' И НЕ 'НАЧАЛО' ПЕРЕД ОПИСАТЕЛЕМ.

(Г) Комментарий. Перед описанием употребляется символ, отличный от ; и от **начало**.

НЕТ ';' В КОНЦЕ БИБЛИОТЕЧНОГО ОПИСАНИЯ.

(Г) Комментарий. Текст библиотечного описания, указанного командой * ПРОЦЕДУРА, не завершается символом ; . Эта ошибка была допущена при записи библиотечного опи-

сания в архив *LIBRA* либо в личный архив математика (гл. 12, пример 3).

ОПИСАНИЕ ЗАДАЕТ АБСОЛЮТНУЮ РАЗМЕРНОСТЬ ПЕРЕМЕННЫХ > 14.

(Е) Комментарий. Нарушено одно из следующих ограничений:

— в описании типа и структуры число порядков по внутренним измерениям должно быть $\leqslant 14$;

— в описании массивов число граничных пар в сумме с числом внутренних измерений должно быть $\leqslant 14$;

— размерность переменной в описании компонент должна быть $\leqslant 14$.

ОПИСАНИЕ ЗАДАЕТ > 15 ВНЕШНИХ МАССИВОВ.

(Е) Комментарий. Описание массивов может задавать $\leqslant 15$ внешних массивов.

ОПИСАНИЕ ЗАДАЕТ > 64 ПЕРЕМЕННЫХ.

(Е) Комментарий. Описание типа и структуры или описание массивов должно определять $\leqslant 64$ переменных.

Рекомендация. Описание, которое вводит много переменных, следует разбить на несколько описаний.

ОПИСАНИЕ ЗАДАЕТ > 7 ПЕРЕМЕННЫХ С ВЕРХНИМ ИНДЕКСОМ.

(Е) Комментарий. В описании типа и структуры либо в описании массивов может описываться $\leqslant 7$ переменных с верхним индексом.

ОПИСАНИЕ ПОСЛЕ ОПЕРАТОРА: синт.

(Г) Комментарий. Как правило, ошибка заключается в том, что описание встречается после оператора. Однако возможен случай, когда символ ; разделяет две конструкции, одна из которых не является оператором или описанием.

Пример. Процедура *арк* описана после оператора $x := 4.75$.

```
◇6◇ начало вещественный x, y;
◇7◇   x := 4.75;
◇8◇ процедура арк (a); начало
◇70◇ конец арк;
```

Система выдаст сообщение:

6—70 ОПИСАНИЕ ПОСЛЕ ОПЕРАТОРА: ⟨ТЕЛО БЛОКА⟩; ⟨ОПИСАНИЕ⟩.

ОШИБКА В ВЫЧИСЛЯЮЩЕМ ВЫРАЖЕНИИ: синт.

(Г) Комментарий. Возможны следующие ошибки:

— операндами арифметической операции не являются арифметические выражения;

— операндами логической операции не являются логические выражения.

ОШИБКА В ГРАНИЧНОЙ ПАРЕ ОПИСАНИЯ МАССИВОВ: синт.

(Г) Комментарий. Нижняя либо верхняя граница — не арифметическое выражение. Менее вероятным является ошибочное вхождение символа [перед идентификатором метки.

ОШИБКА В ЗАГОЛОВКЕ ЦИКЛА.

(Г) Комментарий. Возможны следующие ошибки:

— в элементе списка цикла встречается недопустимый символ;

— после элемента списка цикла нет символа **цикл** или запятая;

— в заголовке цикла без параметра после конструкции **пока** ⟨логическое выражение⟩ вместо символа **цикл** следует запятая.

ОШИБКА В ЗАГОЛОВКЕ ЦИКЛА: синт.

(Г) Комментарий. Возможны следующие ошибки:

— вместо арифметического выражения в элементе списка цикла употребляется другая конструкция;

— после символа **пока** употребляется не логическое выражение.

ОШИБКА В ЗАДАНИИ РАЗМЕРНОСТИ ФОРМАЛЬНЫХ ПАРАМЕТРОВ ПОДПРОГРАММЫ.

(Г) Комментарий. В спецификациях заголовка процедуры-подпрограммы возможны следующие ошибки:

— после — либо + не следует цифра;

— значение абсолютной размерности > 14.

ОШИБКА В ИНДЕКСАХ ПЕРЕМЕННОЙ: синт.

(Г) Комментарий. Один из индексов переменной не является арифметическим выражением.

ОШИБКА В КОМПОНОВКЕ.

(Г) Комментарий. Возможны следующие ошибки:

— после | [нет цифры;

— после конструкции « |[⟨целое без знака⟩ » нет скобки] ;

— номер измерения, по которому производится компоновка, > 14.

ОШИБКА В ОПЕРАТОРЕ БЕМШ.

(Г) Комментарий. Возможны следующие ошибки:

— код операции команды начинается не с буквы и не с восьмеричной цифры;

— операнд команды начинается не с буквы, не с восьмеричной цифры и не с = ;

— после *конд* или *конк* нет запятой;

— в команде или константе больше 16 восьмеричных цифр подряд;

— в операнде команды или адресной константы после знака + или — нет восьмеричной цифры;

— восьмеричный адрес > 77777;

— операнд адресной константы начинается не с буквы и не с восьмеричной цифры;

— восьмеричный код операции > 377.

ОШИБКА В ОПЕРАТОРЕ ПЕРЕХОДА: синт.

(Г) Комментарий. Конструкция после символа **на** не является именующим выражением.

ОШИБКА В ОПЕРАТОРЕ ПРИСВАИВАНИЯ: синт.

(Г) Комментарий. Конструкция после : = не является арифметическим или логическим выражением. Возможен также случай, когда вместо переменной в левой части употребляется формирование или компоновка арифметических (логических) выражений.

ОШИБКА В ОПЕРАТОРЕ ЦИКЛА: синт.

(Г) Комментарий. Возможны следующие случаи:

— после символа **цикл** следует не оператор;

— в заголовке цикла без параметра перед символом **раз** употребляется не первичное выражение.

Пример. В операторе присваивания $A[J] := A[J] + 1$ пропущен текст $:= A[J] + 1$

◇7◇ **для** $J := 1, \ldots, N$ **цикл** $A[J]$;

В результате в качестве тела цикла воспринята переменная с индексами.

ОШИБКА В ОПИСАНИИ НАЧАЛЬНОГО ЗНАЧЕНИЯ ЛИБО ОШИБКА В ОТНОШЕНИИ: синт.

(Г) Комментарий. Возможны следующие ошибки:

— в отношении символ = разделяет две конструкции, одна из которых не является арифметическим выражением;

— в описании начального значения конструкция перед = не является переменной;

— после = в описании начального значения употребляется не арифметическое и не логическое выражение;

— в отношении типа больше употребляется перечисление отношений типа меньше;

— в отношении типа меньше употребляется перечисление отношений типа больше.

ОШИБКА В ОПИСАНИИ ПЕРЕКЛЮЧАТЕЛЯ.

(Г) Комментарий. После описателя **переключатель** нет идентификатора переключателя либо после идентификатора переключателя нет знака : = .

ОШИБКА В ОПИСАНИИ ПЕРЕКЛЮЧАТЕЛЯ: синт.

(Г) Комментарий. В переключательном списке употребляется не именующее выражение. В параметре синт непосредственно после символа **переключатель** указывается ошибочный элемент переключательного списка.

ОШИБКА В ОПИСАНИИ СОБСТВЕННЫХ ПЕРЕМЕННЫХ.

(Г) Комментарий. В описании типа и структуры либо в описании массивов возможны следующие ошибки:

— после описателя **собственный** не следует символ **целый**, **вещественный**, **логический** либо **комплексный**;

— после конструкции **собственный** ⟨тип⟩ не следует буква или символ **массив**.

ОШИБКА В ОПИСАНИИ УПАКОВАННЫХ.

(Г) Комментарий. Возможны следующие ошибки:

— после служебного слова **упакованный** следует символ, отличный от **собственный** и от **логический**;

— после **упакованный собственный** не следует описателя **логический**.

ОШИБКА В ОПИСАНИИ ФУНКЦИИ — ВЫРАЖЕНИЯ: синт.

(Г) Комментарий. Ошибка в выражении после разделителя $=$.

Пример. В условном выражении опущена вторая альтернатива после **иначе**: ◇11◇ **функция** $B(x) =$ **если** $x < 0$ **то** $SG(-x)$; Параметр *синт* в сообщении примет следующий вид: ФУНКЦИЯ ⟨ОПЕРАТОР⟩.

ОШИБКА В ОТНОШЕНИИИ: синт.

(Г) Комментарий. Вместо простого арифметического выражения в отношении употребляется другая конструкция.

Пример. Вместо знака $+$ ошибочно отперфорирован знак циклического сложения:

◇8◇ **если** $(a \oplus B) < 0$ **то** $C(a, B)$;

Параметр *синт* в сообщении примет следующий вид:

⟨ЛОГИЧЕСКОЕ ВЫРАЖЕНИЕ⟩ < ⟨ПЕРВИЧНОЕ ВЫРАЖЕНИЕ⟩

ОШИБКА В ПЕРЕМЕННОЙ С ВЕРХНИМ ИНДЕКСОМ: синт.

(Г) Комментарий. Возможны следующие ошибки:

— верхний индекс переменной не является арифметическим выражением;

— у переменной несколько верхних индексов, что недопустимо;

— нет идентификатора переменной с верхним индексом перед ↑[.

ОШИБКА В ПЕРЕЧИСЛЕНИИ: синт.

(Г) Комментарий. Возможны следующие ошибки:

— в перечислении арифметических выражений или в перечислении отношений правая или левая часть не являются арифметическим выражением;

— в перечислении логических выражений правая или левая часть не является логическим выражением;

— в правой или левой части перечисления присваиваний вместо сформированной (скомпонованной) переменной употребляется формирование или компоновка выражений.

ОШИБКА В ПОМЕЧЕННОМ ОПЕРАТОРЕ: синт.

(Г) Комментарий. После конструкции ⟨метка⟩: не следует оператор.

Пример. Перед ; пропущен текст « : = *0* »:

◇12◇ M : *x*[*1*];

ОШИБКА В СПЕЦИФИКАЦИИ.

(Г) Комментарий. В спецификации заголовка процедуры возможны следующие ошибки:

— после символа **целый, вещественный, логический или комплексный** не следует буква, **массив, процедура** либо **функция**;

— после одного **из описателей массив, процедура, функция, переключатель, метка,** либо **строка** нет идентификатора;

— нет идентификатора после запятой.

ОШИБКА В СПИСКЕ: синт.

(Г) Комментарий. Список есть конструкция, в которой несколько элементов разделены между собой запятыми. Ошибка заключается в том, что имеется недопустимый элемент в одном из следующих списков: списке граничных пар, списке индексов переменной, списке фактических параметров, списке арифметических выражений, списке логических выражений.

Пример. Ошибка во второй граничной паре:

◇15◇ массив *A*[*1*:*40*, *1*];

Система выдает сообщение:

15 ОШИБКА В СПИСКЕ: ⟨ГРАНИЧНАЯ ПАРА⟩, ⟨ПЕРВИЧНОЕ ВЫРАЖЕНИЕ⟩

ОШИБКА В СПИСКЕ ФАКТИЧЕСКИХ ПАРАМЕТРОВ: синт.

(Г) Комментарий. В списке фактических параметров встречается пустой параметр, либо конструкция, которая не может быть фактическим параметром.

ОШИБКА В СПИСКЕ ФОРМАЛЬНЫХ ПАРАМЕТРОВ.

(Г) Комментарий. В списке формальных параметров описания процедуры или функции возможны следующие ошибки:

— после скобки (, либо после запятой нет идентификатора;

— после идентификатора не следует), либо запятая.

ОШИБКА В СПИСКЕ ФОРМИРОВАНИЯ ИЛИ КОМПОНОВКИ: синт.

(Г) Комментарий. Возможны следующие ошибки:

— в списке арифметических выражений в качестве эле-

мента списка употребляется не арифметическое выражение и не перечисление арифметических выражений;

— в списке логических выражений в качестве элемента списка употребляется не логическое выражение и не список логических выражений;

— в качестве элемента списка употребляется пустой элемент.

ОШИБКА В ТРОЕТОЧИИ ИЛИ ДЕСЯТИЧНОМ ЧИСЛЕ.

(Г) Комментарий. В тексте программы (не в строке и не в комментарии) две десятичные точки подряд.

ОШИБКА В УСЛОВНОМ ВЫРАЖЕНИИ ЛИБО В УСЛОВНОМ ОПЕРАТОРЕ: синт.

(Г) Комментарий. Возможны следующие ошибки:

— между символами если и то употребляется не логическое выражение;

— в условном операторе после символа то или иначе употребляется не оператор;

— в условном арифметическом выражении после символа то или иначе употребляется не арифметическое выражение;

— в условном логическом выражении после символа то или иначе употребляется не логическое выражение.

ОШИБКА В ЧИСЛЕ ИЛИ СОСТАВНОЙ МЕТКЕ.

(Г) Комментарий. Возможны следующие ошибки:

— после точки не следует цифра;

— после знака порядка $_{10}$ не следует +, —, либо цифра;

— после конструкции «⟨целое без знака⟩.⟨целое без знака⟩.» не следует буква, либо цифра, что означает ошибку в числе или составной метке.

ПЕРЕМЕННАЯ С ВЕРХНИМ ИНДЕКСОМ 'идентификатор' ОПИСАНА КАК СОБСТВЕННАЯ.

(Д) Комментарий. В языке альфа-6 запрещено использовать собственные переменные с верхним индексом.

ПОВТОРНОЕ ОПИСАНИЕ В БЛОКЕ ИДЕНТИФИКАТОРА 'идентификатор'.

(Д) Комментарий. В блоке имеются два описания, определяющих указанный в сообщении идентификатор. Кроме

того, возможно ошибочное повторение идентификатора в одном описании. К описаниям необходимо также причислить вхождения идентификатора в качестве метки некоторого оператора.

ПРОПУСК ИДЕНТИФИКАТОРА ПЕРЕД '(' ЛИБО НЕВЕРНОЕ ВЫРАЖЕНИЕ В КРУГЛЫХ СКОБКАХ: синт.

(Г) Комментарий. Возможны следующие ошибки:

— перед (нет идентификатора процедуры или функции;

— ошибка в арифметическом или логическом выражении в круглых скобках.

Пример. Перед скобкой (отсутствует идентификатор процедуры A:

$$\Diamond 17\Diamond\ x := x + y;\ (x,\ y);$$

ПРОЦЕДУРА С ЧИСЛОМ ФОРМАЛЬНЫХ ПАРАМЕТРОВ > 35.

(Е) Комментарий. Число формальных параметров процедуры или функции не должно быть > 35.

СИНТАКСИЧЕСКАЯ ОШИБКА В ЗАГОЛОВКЕ ПРОЦЕДУРЫ.

(Г) Комментарий. Возможны следующие ошибки:

— после описателя **процедура** нет идентификатора процедуры;

— после конструкции **процедура** « ⟨идентификатор процедуры⟩» не следует скобка (, либо точка с запятой;

— после скобки) списка формальных параметров нет символа точка с запятой;

— нет идентификатора после спецификатора **значение** (или **результат**);

— в списке значений или результатов нет идентификатора после запятой;

— в списке значений или результатов после идентификатора не следует запятая, либо точка с запятой;

СИНТАКСИЧЕСКАЯ ОШИБКА В ОПИСАНИИ ВНЕШНИХ ПОДПРОГРАММ.

(Г) Комментарий. Возможны следующие ошибки:

— после описателя **альфа**, **фортран** или **алгол**, либо после запятой не следует описатель типа или идентификатор;

— после идентификатора следует не , и не ;.

СИНТАКСИЧЕСКАЯ ОШИБКА В ОПИСАНИИ ДЕЙСТВИТЕЛЬНОЙ И МНИМОЙ ЧАСТИ.

(Г) Комментарий. Возможны следующие ошибки:

— перед і нет +;

— нет разделителя =, либо недопустимый текст после =;

— после і нет идентификатора;

— нет], либо ошибка в списке индексов;

— нет [перед списком индексов;

— перед [нет идентификатора;

— нет ; после описания.

СИНТАКСИЧЕСКАЯ ОШИБКА В ОПИСАНИИ МАССИВОВ.

(Г) Комментарий. Возможны следующие ошибки:

— после описателя **массив**, либо после запятой не следует идентификатор;

— после идентификатора массива нет запятой либо [;

— после сегмента массивов не следует запятая, ; либо —.

Пример. Пропущена ; в описании массивов:

◇3◇ **массив** A, $B[J : 100]$ **целый** T;

СИНТАКСИЧЕСКАЯ ОШИБКА В ОПИСАНИИ ОБЩЕГО МАССИВА.

(Г) Комментарий. Возможны следующие ошибки:

— нет запятой после закрывающей кавычки, ограничивающей имя общего массива;

— после идентификатора встречается символ, отличный от (, запятой или точки с запятой;

— неверное описание длины объекта в общем массиве, либо пропущена).

СИНТАКСИЧЕСКАЯ ОШИБКА В ОПИСАНИИ ТИПА И СТРУКТУРЫ.

(Г) Комментарий. После запятой нет идентификатора, либо после идентификатора не следует запятая, ; или —.

СИНТАКСИЧЕСКАЯ ОШИБКА В ПЕРЕЧИСЛЕНИИ.

(Г) Комментарий. Выражение (или переменная) в правой части перечисления не может быть получено из левой части заменой текста индексов, что противоречит определению перечисления.

Пример. В перечислении вместо $a[J]$ ошибочно употребляется $B[J]$;

◇22◇ $|a[I], \ldots, B[J]| := 0$;

СИНТАКСИЧЕСКИ НЕВЕРНАЯ КОНСТРУКЦИЯ: синт.

(Г) Комментарий. В синтаксической конструкции, указанной параметром *синт*, пропущена часть текста. Все другие причины ошибки менее вероятны. Вся информация о характере ошибки заключена в параметре *синт*.

Пример. В заголовке цикла пропущен символ **для** :

◇7◇ *J*: = *1*, ...,*10* **цикл начало**

Параметр *синт* в сообщении будет иметь вид:

⟨ИДЕНТИФИКАТОР⟩: = ⟨ПЕРВИЧНОЕ ВЫРАЖЕНИЕ⟩, ..., ⟨ПЕРВИЧНОЕ ВЫРАЖЕНИЕ⟩

СЛОЖНЫЙ ЭЛЕМЕНТАРНЫЙ ФРАГМЕНТ ТЕКСТА.

(Е) Комментарий. В тексте программы, расположенном между ближайшими символами ; и ;, нарушено одно из следующих ограничений:

— не допускается > 30 вхождений идентификаторов, указанных системными командами * ЗНАЧЕНИЕ;

— не допускается > 16 описаний меток, т. е. конструкций вида « ⟨метка⟩ : » ;

— в совокупности не должно быть слишком много ограничителей **начало** и **цикл**, переменных с внутренней размерностью, определенных в описании массивов, переменных с верхним индексом, определенных в описании типа и структуры.

ТЕЛО ПРОЦЕДУРЫ — НЕ ОПЕРАТОР: синт.

(Г) Комментарий. На месте тела процедуры вместо оператора находится другая конструкция, которая распечатана в параметре *синт* непосредственно после символа **процедура**.

ТРОЕТОЧИЕ ОКАЙМЛЯЕТСЯ РАЗНЫМИ СИМВОЛАМИ.

(Г) Комментарий. Наиболее вероятная причина ошибки — пропуск ограничителя после троеточия.

Пример. Пропущена запятая после троеточия:

◇12◇ **для** *J*: = *1*, ... *N* **цикл**

ЧИСЛО ОБЪЕКТОВ В ПРОГРАММЕ > 1250.

(Е) Комментарий. Число различных объектов, описанных во всех блоках программы, > 1250. Объектами являются: простая переменная, массив, процедура, функция, переключатель, метка, формальный параметр.

ЧИСЛО ПО АБСОЛЮТНОЙ ВЕЛИЧИНЕ > МАКСИМАЛЬНО ПРЕДСТАВИМОГО В БЭСМ-6.

(Е) Комментарий. В тексте программы встречается число, значение которого по абсолютной величине $> 0.92233720369_{10}+$ $+19$ — максимально представимого в БЭСМ-6.

ЧИСЛО ПО АБСОЛЮТНОЙ ВЕЛИЧИНЕ < МИНИМАЛЬНО ПРЕДСТАВИМОГО В БЭСМ-6.

(Е) Комментарий. В тексте программы встречается число, значение которого по абсолютной величине $<_{10}-19$.

ГЛАВА 20

СООБЩЕНИЯ СЕМАНТИЧЕСКОГО КОНТРОЛЯ

БОЛЬШАЯ И СЛОЖНАЯ ПРОГРАММА — МНОГО РАБОЧИХ СКАЛЯРОВ БЕТА.

(Е) Комментарий. В программе много длинных выражений, регулярных циклов, обращений к процедурам, многомерных операций, цепочек присваиваний, отношений и пр.

Рекомендация. Уменьшить сложность программы, сделав частично следующие преобразования:

— устранить цепочки присваиваний и (или) отношений;

— устранить геометрические операции;

— часть динамических массивов перевести в статические.

БОЛЬШАЯ И СЛОЖНАЯ ПРОГРАММА — МНОГО РАБОЧИХ СКАЛЯРОВ ТЭ.

(Е) Комментарий. В программе много сложных выражений, обращений к процедурам, массивам и пр.

Рекомендация. Уменьшить сложность программы, сделав частично следующие преобразования:

— устранить длинные выражения;

— не употреблять выражения в качестве фактических параметров, вычислять их перед обращением к процедуре (или функции).

ВЕЛИК ЛИНЕЙНЫЙ УЧАСТОК ИЛИ МНОГО ВЫНЕСЕНИЙ ИЗ ЦИКЛА.

(Е) Комментарий. Линейным участком называется последовательность операторов альфа-программы, не прерываемая метками и операторами перехода. Это сообщение может быть выдано также, когда в теле цикла много выражений, не пере-

вычисляемых в цикле. Такие выражения система помещает перед заголовком цикла.

Рекомендация. Разбить линейный участок метками, где это возможно, либо вынести выражения, не перевычисляемые в теле цикла перед циклом и разбить метками.

ВИД ИЛИ ТИП номер-го ФАКТИЧЕСКОГО ПАРАМЕТРА НЕ СООТВЕТСТВУЕТ СПЕЦИФИКАЦИИ В ОПИСАНИИ ПРОЦЕДУРЫ: идентификатор.

(Д) Комментарий. Для специфицированного формального параметра виды или типы (с точностью до целый-вещественный) соответствующих ему фактических параметров должны совпадать со спецификацией.

Пример. **проц** $P(x)$; **массив** x; **начало** x: = 0 **конец**; **вещ** a; $P(a)$; В процедуре P формальный параметр x специфицирован как массив. Соответствующий ему параметр a является скаляром.

В ИМЕНУЮЩЕМ ВЫРАЖЕНИИ НЕВЕРЕН ВИД ИДЕНТИФИКАТОРА идентификатор.

(Д) Комментарий. Вместо метки, составной метки либо указателя переключателя в программе после символа **на** употребляется идентификатор переменной, либо осуществляется неверное обращение к процедуре, параметрами которой являются именующие выражения.

Пример. **начало проц** $P(x)$; **если** $a < 0$ **то на** M **иначе на** x, $P(2)$;

$P(2)$ — неверное обращение к процедуре, так как целое без знака в качестве метки не может быть фактическим параметром процедуры (Приложение 14).

В КАЧЕСТВЕ ФАКТИЧЕСКОГО ПАРАМЕТРА АЛГОЛ-ПРОЦЕДУРЫ УПОТРЕБЛЯЕТСЯ СТРОКА — ФОРМАЛЬНЫЙ ПАРАМЕТР ИЛИ ИДЕНТИФИКАТОР ПРОЦЕДУРЫ — ФОРМАЛЬНОГО ПАРАМЕТРА.

(Д) Комментарий. Процедура, специфицированная описателем **алгол** не может иметь в качестве фактического параметра строку — формальный параметр или процедуру (функцию) — формальный параметр.

Пример. **алгол** P; **проц** $Q(x, y)$; **проц** x; $P(x)$;

Для алгол-процедуры P в качестве фактического параметра ошибочно употребляется x — идентификатор процедуры, являющийся формальным параметром.

ВМЕСТО УКАЗАТЕЛЯ ПЕРЕКЛЮЧАТЕЛЯ УПОТРЕБЛЯЕТСЯ ИДЕНТИФИКАТОР ПЕРЕКЛЮЧАТЕЛЯ идентификатор.

(Д) Комментарий. Пропущен индекс в указателе переключателя.

Пример. **перекл** *SW*: = *N*, *M*, *K*; **на** *SW*;

В ОПИСАНИИ МАССИВА ГРАНИЦА номер ИЗМЕРЕНИЯ ИМЕЕТ НЕДОПУСТИМЫЙ ТИП.

(Д) Комментарий. Граница измерения может иметь тип только **целый** либо **вещественный**.

Пример. Ошибочно описание **лог** *E*; **массив** *A* [*1* : *E*];

В ОПИСАНИИ МАССИВА ГРАНИЦА номер ИЗМЕРЕНИЯ НЕНУЛЕВОЙ РАЗМЕРНОСТИ.

(Д) Комментарий. Граница измерения в описании массива должна быть скалярной.

Пример. Ошибочно описание массива *B*
целый *A* — **матрица** $N \times N$; **массив** *B* [*1* : *A*];

В ОПИСАНИИ МАССИВА ИЛИ В ПЕРЕЧИСЛЕНИИ НИЖНЯЯ ГРАНИЦА БОЛЬШЕ ВЕРХНЕЙ.

(Д) Комментарий. Если границы по измерению в описании массива или перечисления являются константами, то нижняя граница должна быть меньше верхней.

Пример. Ошибочно описание **массив** *A* [*10* : *1*]. Ошибочно перечисление *B* [*10*]: = ... : = *B* [*1*]: = *0*;

ГЛУБИНА ВЛОЖЕННОСТИ ИНДЕКСНЫХ ВЫРАЖЕНИЙ > 5.

(Е) Комментарий. Глубина вложенности скобок [должна быть $\leqslant 5$.

ГРАНИЦА ПЕРЕЧИСЛЕНИЯ ИМЕЕТ НЕНУЛЕВУЮ РАЗМЕРНОСТЬ.

(Д) Комментарий. Граница перечисления должна быть скалярным выражением.

Пример. **вещественный** *a* — **массив** *2*; **массив** *B* [*1* : *2*]; *B* [*1*]: = ... : = *B* [*a*]: = *0*; Ошибка в том, что граница перечисления *a* размерности 1.

ДЛИНА МАССИВА идентификатор $>$ 32000.

(Е) Комментарий. Число элементов статического массива должно быть $<$ 32 000.

Рекомендация. Поместить массив во внешнюю память, описав его как *EX*-массив.

ИДЕНТИФИКАТОР ПЕРЕКЛЮЧАТЕЛЯ В КАЧЕСТВЕ ФАКТИЧЕСКОГО ПАРАМЕТРА АЛГОЛ-ПРОЦЕДУРЫ.

(Д) Комментарий. Процедура (функция), специфицированная описателем **алгол**, не может иметь фактический параметр — идентификатор переключателя. В качестве параметров алгол-процедуры допускается переменная, массив, константа, метка, процедура, скалярное выражение.

МЕТКА идентификатор ИЗ СОСТАВНОЙ МЕТКИ НЕ ОПИСАНА В ТОМ БЛОКЕ, КОТОРЫЙ МЕТИТ ПРЕДШЕСТВУЮЩАЯ ЕЙ МЕТКА В ЭТОЙ СОСТАВНОЙ.

(Д) Комментарий. *M1*: **начало вещ** *a*, *B*; *a*: = *0*; *M2*: **начало цел** *c*; *M3* : *c* : = *0*; **конец конец**; **на** *M1*. *M3*. Неверно задан переход по метке *M1* . *M3*. Правильно было бы **на** *M1.M2.M3*.

МНИМАЯ ЧАСТЬ ДИНАМИЧЕСКОГО МАССИВА — ФОРМАЛЬНОГО ПАРАМЕТРА идентификатор В АДРЕСНОЙ КОНСТАНТЕ КОНК ИЛИ КОНД ОПЕРАТОРА БЕМШ.

(Д) Комментарий. Запрещается употреблять идентификатор мнимой части динамического массива в адресной константе.

Пример. Ошибочна ситуация

```
начало цел k; k: = 2;
  начало комплексный массив A[1 : k];
    проц P(a); массив a;
      начало
    a[k] = ax[k] + iay[k];
    M: бемш (конк, a(ay));
      . . . . . .
      конец.
  P(A);
  конец;
конец
```

ay — мнимая часть массива — формального параметра и ему соответствует фактический параметр — динамический массив.

МНОГО АРГУМЕНТОВ В ОПЕРАТОРЕ идентификатор.
(Е) Комментарий. Это сообщение относится только к стандартным операторам *input* и *output*. Число аргументов в этих операторах должно быть < 38.

Рекомендация. Ввод (или вывод) объектов осуществить несколькими операторами *input* (*output*).

МНОГО БЛОКОВ.
(Е) Комментарий. Суммарное количество блоков, циклов и процедур в программе должно быть < 1024.

МНОГО МЕТОК.
(Е) Комментарий. В программе много меток, условных выражений, циклов и процедур так, что общее число меток, включая заводимые транслятором, оказалось > 4096.

МНОГО ОБРАЩЕНИЙ К ПРОЦЕДУРАМ И БОЛЬШАЯ ВЛОЖЕННОСТЬ ПРОЦЕДУР.
(Е) Комментарий. Нарушено ограничение на величину интегральной характеристики, зависящей от следующих параметров:

— числа обращений к процедурам и функциям в тексте программы;

— глубины вложенности процедур и функций;

— числа параметров в процедурах и функциях;

— числа переменных, которым производится присваивание в теле вызываемых процедур и функций.

Рекомендация. Рекомендуется упростить программу, уменьшив:

— глубину вложенности процедур;

— число обращений к процедурам;

— число перевычисляемых переменных в телах процедур.

МНОГО ОБРАЩЕНИЙ К ПРОЦЕДУРАМ И ФУНКЦИЯМ.
(Е) Комментарий. Суммарное число обращений к процедурам и функциям, с учетом расклейки тел процедур (см. сообщение * РАСКЛЕИВАЕТСЯ ТЕЛО ПРОЦЕДУРЫ 'идентификатор'), и нерегулярных циклов должно быть $\leqslant 1024$.

МНОГО ОТЛАДОЧНЫХ РЕДАКТИРОВАНИЙ.

(Е) Комментарий. Суммарное число идентификаторов, печатаемых по системным командам *ПЕЧАТАТЬ— МЕТКИ и * ЗНАЧЕНИЕ, должно быть < 512.

Рекомендация. Уменьшить диапазон действия команды * ПЕЧАТАТЬ — МЕТКИ.

МНОГО ПРОЦЕДУР С ФОРМАЛЬНЫМИ ПАРАМЕТРАМИ-ПРОЦЕДУРАМИ.

(Е) Комментарий. Формальный параметр-процедура есть формальный параметр, вместо которого подставляется идентификатор процедуры или идентификатор функции, у которой есть параметры.

МНОГО ЦИКЛОВ И ПРОЦЕДУР.

(Е) Комментарий. Число циклов в программе должно быть < 512.
Число процедур должно быть < 256.

НЕВЕРЕН ПАРАМЕТР ЦИКЛА 'идентификатор'.

(Д) Комментарий. Параметр цикла ненулевой размерности, либо имеет недопустимый тип (не **целый** и не **вещественный**).

НЕВЕРНОЕ ИНДЕКСНОЕ ВЫРАЖЕНИЕ.

(Д) Комментарий. Индексное выражение имеет ненулевую размерность, либо недопустимый тип (не **целый** и не **вещественный**).

НЕВЕРНОЕ ЧИСЛО АРГУМЕНТОВ СТАНДАРТНОЙ ФУНКЦИИ ИЛИ ПРОЦЕДУРЫ имя.

(Д) Комментарий. Сообщение относится к стандартным процедурам и функциям *marg*, *lmarg*, *det*, *shift*, *text* и *copy*. Возможно, в списке аргументов пропущен символ запятая.

НЕВЕРНЫЙ номер-Й АРГУМЕНТ ОПЕРАТОРА имя оператора.

(Д) Комментарий. Сообщение относится к стандартным операторам *input*, *output* и *marg*, выдается при семантическом анализе аргументов.

НЕДОПУСТИМЫЙ ТИП НОМЕРА КАНАЛА.

(Д) Комментарий. Номером канала, указанного в операторе *input* (*output*) может быть только выражение типа **целый**.

НЕДОПУСТИМЫЙ ТИП номер-ГО АРГУМЕНТА ФУНКЦИИ MAX ИЛИ MIN.

(Д) Комментарий. Аргументы функции *max* (или *min*) должны иметь тип целый, либо вещественный.

Пример. **комплексный** *b*; **вещественный** *a*; *a*: = *max*(*b*, *2*, *3*);

НЕНУЛЕВАЯ РАЗМЕРНОСТЬ В ЭЛЕМЕНТЕ СПИСКА ЦИКЛА.

(Д) Комментарий. Элемент списка цикла должен быть скалярным выражением.

Пример. **вещественный** *a* — **массив** 5; **для** *j*: = *1* **шаг** *a* **до** *10* **цикл** *a* — имеет размерность, равную 1, что неверно.

НЕ ОПИСАНА МЕТКА ДЛЯ ПЕРЕМЕННОЙ С ВЕРХНИМ ИНДЕКСОМ: 'идентификатор'.

(Д) Комментарий. Метка управляющего цикла (указанная в описании переменной с верхним индексом) не описана в том блоке, где описана данная переменная с верхним индексом.

Пример. **начало вещественный** *a* ↑ [1:5:M]; **конец**; Метка M не описана.

НЕОПИСАННЫЙ ИДЕНТИФИКАТОР идентификатор.

(Д) Комментарий. Нет описания идентификатора ни в том блоке, в котором он используется, ни в объемлющих блоках. Следует отметить, что описанием метки является ее вхождение в качестве метки, метящей оператор или блок.

Примеры. 1. **начало** *a* = ‖*b*[*k*]‖; *b*[*1*]: = *0* **конец**; Не описан идентификатор *a*.

2. **начало вещественный** *B*; *B*: = *a*; **начало вещественный** *a*; **конец конец**. Не описан идентификатор *a*.

3. **начало** *a* = *x* + *iy*; *x*: = 1 **конец**

Не описан идентификатор *a*.

НЕПРАВИЛЬНОЕ ПРИСВАИВАНИЕ номер-му ФОРМАЛЬНОМУ ПАРАМЕТРУ ПРОЦЕДУРЫ идентификатор.

(Д) Комментарий. Формальному параметру процедуры есть присваивание в теле процедуры, а соответствующий фактический параметр является константой или выражением.

Возможно ошибка в том, что формальный параметр не специфицирован значением.

Пример. **процедура** $F(x)$; **начало** $x := a$ **конец**; $F(a+B)$; Формальный параметр x необходимо специфицировать значением, так как ему соответствует фактический параметр $a+B$ — вычисляющее выражение. Правильно было бы: **процедура** $F(x)$; **значение** x; **вещественный** x; **начало** $x := a$ **конец**; $F(a+B)$;.

НЕПРАВИЛЬНОЕ УПОТРЕБЛЕНИЕ СТРОКИ.

(Д) Комментарий. Формальный параметр, которому соответствует фактический параметр-строка, используется там, где вхождение строки синтаксически недопустимо.

НЕСОВМЕСТИМЫ РАЗМЕРНОСТИ ОПЕРАНДОВ В АЛГЕБРАИЧЕСКОЙ ОПЕРАЦИИ.

(Д) Комментарий. Сообщение выдается для операций умножения и возведения в целую степень. Умножение определено для комбинаций операндов, приведенных в табл. 2.1. Целая степень массива определяется для квадратных матриц и понимается как матричное умножение. Отрицательная степень понимается как положительная степень обратной матрицы.

НЕСОВМЕСТИМЫ ТИПЫ ЭЛЕМЕНТОВ СПИСКА ГЕОМЕТРИЧЕСКОЙ ОПЕРАЦИИ.

(Д) Комментарий. Если сформированная или скомпонованная переменная стоит в левой части оператора присваивания, то элементы списка должны совпадать по типу; если в правой части, то совпадение типов должно быть с точностью до **целый — вещественный**.

Пример. **целый** i; **вещественный** j; $|i, j| := 0$; Элементы списка в левой части оператора присваивания имеют разный тип — это ошибка. Однако будет верным случай **целый** i; **вещественный** j; **массив** $a[1:2]$; $a[\,] := |i, j|$.

НЕ СОВПАДАЕТ ВИД, ТИП И СТРУКТУРА ФАКТИЧЕСКИХ ПАРАМЕТРОВ ДЛЯ РАЗНЫХ ОБРАЩЕНИЙ К ФОРМАЛЬНОЙ ПРОЦЕДУРЕ идентификатор.

(Д) Комментарий. Поясним эту ошибку на примере:
процедура $F(x)$; **начало** $x(a)$; $x(B)$; **конец**;
вещественный a; **массив** $B[1:2]$; **процедура** $F(y)$;
начало $y := 0$ **конец**; $F(Q)$;

Процедура *x* является формальной, так как она не описана явно, *a* является формальным параметром процедуры. Причем структуры фактических параметров *a* и *B* для разных обращений к процедуре *x* — различны (*a* — скаляр, *B* — массив). Такая ситуация является ошибочной.

НЕСОВПАДЕНИЕ РАЗМЕРНОСТЕЙ В ОПЕРАТОРЕ ПРИСВАИВАНИЯ.

(Д) Комментарий. Левая и правая части оператора присваивания должны иметь одинаковую абсолютную размерность.

Пример. Ошибка в том, что левая часть оператора присваивания имеет размерность 1, а правая — 0: **вещественный** *a*; **массив** *B*[*1* : *2*]; *B*[ ]: = *a*; *B*[ ]: = *1*;. Однако будет верным **массив** *a*[*1* : *3*, *1* : *4*]; *a*[,]: = 0;

НЕСОВПАДЕНИЕ РАЗМЕРНОСТЕЙ ОПЕРАНДОВ В ПОКОМПОНЕНТНОЙ ОПЕРАЦИИ.

(Д) Комментарий. По языку альфа-6 операнды в покомпонентной операции должны иметь одинаковую абсолютную размерность. К покомпонентным операциям относятся:

— арифметические операции +, —, /, +, ↑ (для нецелого показателя);

— логические операции ∨, ∧, ¬, ⊃, ≡;

— операции покомпонентного умножения и операция логического сложения по модулю 2.

Пример. **массив** *a*[*1* : *2*], *B*[*1* : *2*, *1* : *2*]; *a*[ ]: = *a*[ ] + *B*[,]; Операнд *a* имеет размерность, равную 1, а операнд *B* — равную 2.

НЕСОВПАДЕНИЕ РАЗМЕРНОСТЕЙ У АРГУМЕНТОВ СТАНДАРТНОЙ ФУНКЦИИ идентификатор.

(Д) Комментарий. Сообщение относится к стандартным функциям *arc*, *div* и *res*, для которых должны совпадать размерности первого и второго аргументов.

Пример. **вещественный** *a*; **массив** *B*[*1* : *10*]; *div* (*a*, *B*[ ]); Размерность первого аргумента функции *div* равна 0, а второго — 1.

НЕСОВПАДЕНИЕ РАЗМЕРНОСТЕЙ У ЭЛЕМЕНТОВ СПИСКА ГЕОМЕТРИЧЕСКОЙ ОПЕРАЦИИ.

(Д) Комментарий. Элементы списка формирования и компоновки должны иметь одинаковую абсолютную размерность.

НЕСОВПАДЕНИЕ ТИПОВ В ОПЕРАТОРЕ ПРИСВАИВАНИЯ ИНОЕ, ЧЕМ ЦЕЛЫЙ — ВЕЩЕСТВЕННЫЙ.

(Д) Комментарий. Левая и правая части оператора присваивания должны совпадать по типу с точностью до **целый — вещественный.**

Пример. Ошибочно: **логический** *a*; **вещественный** *B*; *a*: = *B*; Верно: **вещественный** *a*; **целый** *B*; *a*: = *B*;

НЕСООТВЕТСТВИЕ МЕЖДУ ОПИСАНИЕМ И ИСПОЛЬЗОВАНИЕМ ПЕРЕМЕННОЙ идентификатор В КОЛИЧЕСТВЕ ИНДЕКСНЫХ ПОЗИЦИЙ.

(Д) Комментарий. При использовании переменной (формального параметра) число индексных позиций должно совпадать с числом индексных позиций в описании данной переменной (фактического параметра).

Пример. **процедура** *P*(*x*); **начало** *a*: = *x*[*1*]; **конец; целый** *i*; **массив** *A*[*1*:*2*, *1*:*3*]; *P*(*A*); Массив *A* по описанию является двумерным. Поэтому обращение к процедуре *P*(*A*) ошибочно, так как фактический параметр процедуры *P* должен быть одномерным.

НЕСООТВЕТСТВИЕ ЧИСЛА ФАКТИЧЕСКИХ ПАРАМЕТРОВ ЧИСЛУ ФОРМАЛЬНЫХ ПАРАМЕТРОВ ПРОЦЕДУРЫ идентификатор.

(Д) Комментарий. Число фактических параметров в обращении к процедуре должно строго совпадать с числом формальных параметров в описании этой процедуры.

Пример. Ошибочны обращения *P*(*2*, *1*) и *P*:
процедура *P*(*x*); **начало** *x*: = *1* **конец**; *P*(*2*, *1*); *P*;

* НЕТ ОБРАЩЕНИЯ К ПРОЦЕДУРЕ 'идентификатор'.

(Д) Комментарий. Нет обращения к указанной процедуре (или функции). Кроме того, идентификатор процедуры нигде не используется как параметр при обращении к другим процедурам (или функциям). Поэтому описание указанной процедуры устраняется из программы при трансляции.

НЕТ РЕГИСТРА ДЛЯ БАЗЫ.

(Б) Комментарий. Для базирования модуля загрузки система употребляет один из регистров с 1 по 7-й. В системной команде * РЕГИСТРЫ указаны все регистры с 1 по 7-й — нет свободного регистра для базирования [14, стр. 221].

ОПЕРАТОР ПРОЦЕДУРЫ ВМЕСТО УКАЗАТЕЛЯ ФУНКЦИИ: идентификатор.

(Д) Комментарий. В выражении ошибочно используется оператор процедуры. Возможно, в описании процедуры-функции перед символом **процедура** отсутствует описатель **типа**.

Пример. **процедура** $F(x)$; $F:=x+1$; **вещественный** a; $a:=F(a)+1$; Перед символом **процедура** пропущен описатель **вещественный**.

ПЕРЕКЛЮЧАТЕЛЬ ИЛИ МЕТКА 'идентификатор' В ВЫЧИСЛЯЮЩЕМ ВЫРАЖЕНИИ.

(Д) Комментарий. В вычисляющее выражение не могут входить метка, переключатель или указатель переключателя.

Пример. **начало вещественный** a, B; $M: a:=B+M$; **конец**.

ПОКАЗАТЕЛЬ СТЕПЕНИ — ПЕРЕМЕННАЯ НЕНУЛЕВОЙ РАЗМЕРНОСТИ: 'идентификатор'.

(Д) Комментарий. **вещественный** a; **вещественный** B — **массив** 2; $a:=a\uparrow B$; Показатель степени B — размерности 1.

ПРОПУСК ЗНАКА ОПЕРАЦИИ ПЕРЕД ОТКРЫВАЮЩЕЙ КРУГЛОЙ СКОБКОЙ: 'символ'.

(Д) Комментарий. Как правило, сообщение относится к пропуску знака умножения перед открывающей скобкой.

Пример. **вещественный** a; $a:=a(X+1)$;
Между символами a и (отсутствует знак $\times$.

* РАСКЛЕИВАЕТСЯ ТЕЛО ПРОЦЕДУРЫ 'идентификатор'.

(Д) Комментарий. Один из формальных параметров процедуры (функции) не специфицирован. Соответствующие ему фактические параметры в двух обращениях к процедуре различаются по виду, типу или структуре. Транслятор дублирует описание процедуры; и связывает первую копию тела процедуры с первым обращением к процедуре, а вторую копию — со вторым обращением.

Пример.

процедура $F(y)$; **начало** $a:=y\times y$ **конец**;
вещественный a; **массив** $b[1:2]$; $F(a)$; $F(b[\,])$;

Для двух обращений к процедуре F не совпадает структура фактических параметров (a — скаляр, b — массив). Тело процедуры F дублируется. Для обращения $F(a)$ в теле процедуры

обычное умножение скаляров. Для обращения $F(b[\,])$ в теле процедуры — скалярное произведение векторов.

Рекомендация. Следует избегать расклейки процедур, так как это увеличивает длину программы.

РЕКУРСИВНАЯ ПРОЦЕДУРА 'идентификатор'.

(Д) Комментарий. Запрещаются рекурсивные процедуры и процедуры-функции. Запрещаются рекурсивные обращения к процедурам и процедурам-функциям.

Примеры.

1. **процедура** $F(x)$; **процедура** x; **начало** $x(a)$
   конец; **процедура**; $Q(y)$; $F(y)$; $F(Q)$;
   $F(Q)$ — рекурсивное обращение к процедуре F.
2. **целый процедура** $F(x)$; **начало** $y := x + F(x-1)$ **конец**;
   F — рекурсивная процедура-функция.

СЛОЖНАЯ ПРОГРАММА.

(Е) Комментарий. В программе много меток, циклов, условных выражений, описаний процедур, обращений к процедурам.

СТАНДАРТНАЯ ФУНКЦИЯ 'идентификатор' ИМЕЕТ НЕДОПУСТИМЫЕ АРГУМЕНТЫ.

(Д) Комментарий. Аргумент стандартной функции имеет недопустимый вид, тип или структуру (п. 15.3).

СУММИРОВАНИЕ ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ.

(Д) Комментарий. Операндами оператора суммирования Σ являются логические выражения, что недопустимо.

Пример. **логический массив** $a[1:10]$; $B := \Sigma(i, 1, 10, a[i])$;

* ТЕЛО ПРОЦЕДУРЫ идентификатор ВЫНОСИТСЯ В ОБЪЕМЛЮЩИЙ БЛОК.

(Д) Комментарий. Сообщение может быть выдано в тех случаях, когда к процедуре-функции нет явных обращений и идентификатор процедуры используется в качестве фактического параметра. Сообщение указывает, что описание процедуры помещается в объемлющий блок, это может стать причиной неописанных идентификаторов, либо изменения смысла идентификаторов.

Рекомендация. Чтобы избежать вынесения тела процедуры, необходимо вставить фиктивное обращение к процедуре.

начало процедура $Q(x)$; $x(a)$;
целый a; **начало процедура** $F(y)$;
начало a: $= 1$ **конец**;
целый a; $Q(F)$ **конец**; a: $= 0$ **конец**;

В блоке, где описана процедура F, к ней нет явного обращения. Чтобы тело процедуры не выносилось в объемлющий блок, следует вставить фиктивное обращение.

начало процедура $Q(x)$; $x(a)$;
целый a; **начало процедура** $F(y)$;
начало a: $= 1$ **конец**; **целый** a; $Q(F)$: **на** M; $F(a)$;
M: **конец**; a: $= 0$ **конец**;

УПОТРЕБЛЕНИЕ АРИФМЕТИЧЕСКОЙ ПЕРЕМЕННОЙ 'идентификатор' В ЛОГИЧЕСКОЙ ОПЕРАЦИИ.

(Д) Пример. **вещественный** a; **если** $(a < 0) \bigwedge a$ **то на** M **иначе на** N;

В операции логического умножения употребляется вещественная переменная a, что неверно.

УПОТРЕБЛЕНИЕ ИДЕНТИФИКАТОРА СТАНДАРТНОЙ ФУНКЦИИ 'идентификатор' В КАЧЕСТВЕ ФАКТИЧЕСКОГО ПАРАМЕТРА.

(Д) Комментарий. Запрещено использование идентификаторов стандартных функций в качестве фактических параметров процедур или функций.

Пример. Ошибочно обращение к процедуре $F(sin)$

УПОТРЕБЛЕНИЕ ЛОГИЧЕСКОЙ ПЕРЕМЕННОЙ 'идентификатор' В АРИФМЕТИЧЕСКОЙ ОПЕРАЦИИ.

(Д) Пример. **логический** $Л$; **вещественный** a; a: $= Л + 1$;

УПОТРЕБЛЕНИЕ НЕЦЕЛОГО ОПЕРАНДА 'идентификатор' В ОПЕРАЦИИ '÷'.

(Д) Комментарий. Операнды операции деления нацело (÷) должны иметь тип **целый**.

Пример. Ошибочное употребление операнда B: **вещественный** B; **целый** a; a: $= B \div a$;

УПОТРЕБЛЕНИЕ ПЕРЕМЕННОЙ 'идентификатор' НУЛЕВОЙ РАЗМЕРНОСТИ В ОПЕРАЦИИ ПОКОМПОНЕНТНОГО УМНОЖЕНИЯ.

(Д) Комментарий. В операции покомпонентного умножения операнды должны иметь ненулевую размерность.

Пример. **вещественный** a, B; $a \circ B$;

УПОТРЕБЛЕНИЕ СКАЛЯРА 'идентификатор' ВМЕСТО МАССИВА.

(Д) Комментарий. В программе встретился текст « ⟨идентификатор⟩ [» ; в то же время данный идентификатор описан как скаляр.

Пример. Возможно также неверное обращение к процедуре.

процедура $P(a)$; **массив** a; **начало** $a := 0$ **конец**;
вещественный B; $P(B)$;

Для процедуры P фактический параметр должен быть массивом и обращение $P(B)$ — неверно, так как B — скаляр.

УПОТРЕБЛЕНИЕ С КОМПЛЕКСНЫМИ ОПЕРАНДАМИ ОПЕРАЦИИ ПОРЯДКА, ОТЛИЧНОЙ ОТ $=$ и $\neq$.

(Д) Комментарий. Для комплексных величин нельзя применять операции $<$, $\leqslant$, $>$, $\geqslant$.

Пример. Ошибочные операнды для операции $<$:
комплексный a, B; **если** $a < B$ **то** $a := B$; .

ФАКТИЧЕСКИЙ ПАРАМЕТР АЛГОЛ-ПРОЦЕДУРЫ ЯВЛЯЕТСЯ ИДЕНТИФИКАТОРОМ ПРОЦЕДУРЫ (ФУНКЦИИ), НЕ СПЕЦИФИЦИРОВАННОЙ КАК АЛГОЛ-ПРОЦЕДУРА (ФУНКЦИЯ).

(Д) Комментарий. Процедура (функция), специфицированная описателем **алгол**, может иметь фактический параметр — идентификатор процедуры, только если эта процедура тоже специфицирована описателем **алгол**.

ФОРМИРОВАНИЕ ИЛИ КОМПОНОВКА ВЫЧИСЛЯЮЩИХ ВЫРАЖЕНИЙ В ЛЕВОЙ ЧАСТИ ОПЕРАТОРА ПРИСВАИВАНИЯ.

(Д) Комментарий. Ошибка в том, что в обращении к процедуре фактический параметр есть формирование или компоновка вычисляющих выражений и имеется присваивание соответствующему формальному параметру в теле процедуры.

ЦИКЛЫ С ОДИНАКОВЫМИ ПАРАМЕТРАМИ ВЛОЖЕНЫ ДРУГ В ДРУГА.

(Д) Примеры: 1. **для** $i := c1$ **шаг** $c2$ **до** $c3$ **цикл начало для** $i := B1$ **шаг** $B2$ **до** $B3$ **цикл начало** $i := i + 1$ **конец конец**;

2. **процедура** F; **начало для** $i := 1$ **шаг** 2 **до** 3 **цикл начало** $B := B + 1$ **конец конец**; **для** $i := c1$ **шаг** $c2$ **до** $c3$ **цикл начало** F **конец**;

Здесь вложенность циклов динамическая (через процедуру).

ЧИСЛО БЛОКОВ PART > 128.

(Е) Комментарий. Нарушено количественное ограничение.

ЧИСЛО ВХОЖДЕНИЙ НЕОПИСАННОГО ИДЕНТИФИКАТОРА 'идентификатор' ⩾ целое число.

(Д) Комментарий. Если в программе есть неописанный идентификатор, то для первых 4-х его вхождений выдается текст: НЕОПИСАННЫЙ ИДЕНТИФИКАТОР 'идентификатор'. В дальнейшем подсчитывается общее число вхождений неописанного идентификатора и выдается данное сообщение.

ЧИСЛО СИМВОЛОВ АРИФМЕТИЧЕСКОГО ИЛИ ЛОГИЧЕСКОГО ВЫРАЖЕНИЯ > 512.

(Е) Комментарий. Нарушено ограничение на длину выражения.

ГЛАВА 21

АРХИВНЫЕ СООБЩЕНИЯ

ЗАПИСЬ В АРХИВЕ РАЗМЕЩАЕТСЯ В ЗОНЕ С НОМЕРОМ > 1777.

(И) Комментарий. Носители — МЛ и МД содержат не более 1024 зон с восьмеричными номерами от 0 до 1777. Сообщение указывает, что номера зон, в которых хранится или в которые должна быть помещена запись, выходят за допустимые границы. Возможно и так, что начальная зона записи ⩽ 1777, а конечная > 1777. Причиной такой ситуации для архива с оглавлением может быть перепись архива в начальные зоны, при которой часть архива оказалась бы за пределами носителя.

ЗАПРЕЩЕНО ИЗМЕНЯТЬ ИМЯ АРХИВА БУФЕР.

(Б) Комментарий. Первым параметром команды * ИЗМЕНИТЬ␣ИМЯ␣АРХИВА указывается имя системного архива *БУФЕР*, что недопустимо.

НА МЕСТЕ ЗАПИСИ НАХОДИТСЯ ЗАПИСЬ С ДРУГИМ ИМЕНЕМ. тип записи имя АРХИВ имя архива.

(И) Комментарий. Для архива с оглавлением сообщение означает, что в зонах на МЛ (или МД), где должна храниться

запись, создан примитивный архив. Для примитивного архива возможны следующие случаи:

— неверно указано имя программы или данных в команде * ПРОГРАММА или * ДАННЫЕ;

— неверно указана начальная зона примитивного архива;

— на место примитивного архива был записан другой примитивный архив;

— в качестве архивной используется не та бобина, на которой был создан примитивный архив.

НА МЕСТЕ ЗАПИСИ ХРАНИТСЯ ЗАПИСЬ С ДРУГИМ ТИПОМ. тип записи имя АРХИВ имя архива.

(И) Комментарий. Для архива с оглавлением сообщение означает, что в зонах на МЛ (или МД), где должна храниться запись, создан примитивный архив. Для примитивного архива возможны следующие случаи:

— неверно указана начальная зона примитивного архива;

— на месте примитивного архива создан другой примитивный архив;

— в качестве архивной используется не та бобина, на которой ранее был создан примитивный архив.

НЕТ БИБЛИОТЕЧНОГО ОПИСАНИЯ имя В ОГЛАВЛЕНИИ АРХИВА имя архива.

(И) Комментарий. В указанном архиве нет программы, имя которой задано параметром в системной программе * ПРОЦЕДУРА. Возможно неверно указано имя библиотечного описания в команде * ПРОЦЕДУРА.

НЕТ ЗАПИСИ В ОГЛАВЛЕНИИ. тип записи имя АРХИВ имя архива.

(И) Комментарий. В архиве с оглавлением нет записи с указанным типом и именем. Возможны следующие случаи:

— неверно указано имя записи в команде * ПРОГРАММА или * ДАННЫЕ;

— запись не была ранее записана в данный архив;

— запись была вычеркнута при помощи команды * УНИЧТОЖИТЬ — ПРОГРАММУ или * УНИЧТОЖИТЬ — ДАННЫЕ.

НЕТ ЗАПИСИ тип записи имя АРХИВ имя архива.

(И) Комментарий. Для архива с оглавлением сообщение означает, что оглавление содержит указанную запись, однако ее нет в тех зонах на МЛ (или МД), которые указаны в ог-

лавлении. Такое возможно при затирании записи. Для примитивного архива возможны следующие случаи:

— запись не была ранее записана в примитивный архив;

— неверно указана начальная зона примитивного архива в системной команде * АРХИВ;

— ранее произошло затирание записи;

— в качестве архивной используется не та бобина, на которую ранее была сделана запись.

НЕТ МЕСТА В АРХИВЕ. тип записи имя АРХИВ имя архива.

(И) Комментарий. При записи в архив программы или данных нет требуемого числа свободных зон из общего числа зон, отведенных архиву по команде * ЗАПИСАТЬ — ОГЛАВЛЕНИЕ.

НЕТ МЕСТА В ОГЛАВЛЕНИИ АРХИВА. тип записи имя АРХИВ имя архива.

(И) Комментарий. При записи в архив новой программы или данных оглавление содержало предельное число — 509 записей.

НЕТ ОГЛАВЛЕНИЯ. АРХИВ имя архива.

(И) Комментарий. Возможны следующие случаи:

— архив не был предварительно создан по команде * ЗАПИСАТЬ — ОГЛАВЛЕНИЕ;

— неверно указана начальная зона архива в команде * АРХИВ;

— в качестве архивной используется не та бобина, на которой был создан архив;

— затерта 5-ая зона (относительно начальной зоны архива), на которой хранится оглавление.

Рекомендация. При затирании 5-й зоны оглавление может быть восстановлено, если сохранилась 4-я зона, в которой всегда хранится копия 5-й зоны. При затирании 4-й и 5-й зоны сохранившиеся записи могут быть прочитаны, если их описать как примитивные архивы (п. 12.2).

ОГЛАВЛЕНИЕ ЧУЖОГО АРХИВА. В ЗАДАНИИ ИСПОЛЬЗУЕТСЯ АРХИВ имя архива.

(И) Комментарий. Имя архива, указанное в системной команде * АРХИВ, не совпадает с именем архива в оглавлении. Возможны следующие случаи:

— неверное имя архива в системной команде * АРХИВ;

— используется не та бобина, на которой был создан архив с именем, указанным в системной команде * АРХИВ;

— на месте архива создан архив с другим именем.

ОТМЕНЯЕТСЯ ПОВТОРНАЯ ЗАПИСЬ В АРХИВ. тип записи имя АРХИВ имя архива.

(И) Комментарий. Задание содержит команду * КОНТРОЛЬ _ ДАТЫ. До начала обработки данного задания архив уже содержал запись с указанным именем и типом. Кроме того, дата этой записи, указанная в оглавлении, совпадает с текущей датой решения данной задачи на БЭСМ-6. Тогда, в соответствии с назначением команды * КОНТРОЛЬ _ ДАТЫ, запись в архив, которая задается данным заданием, отменяется.

* ПРОИЗОШЛА ЗАПИСЬ В АРХИВ. НАЧАЛЬНАЯ ЗОНА восьм номер ДЛИНА восьм длина. тип записи имя АРХИВ имя архива.

(И) Комментарий. Первый параметр — математический номер той зоны на МЛ (или МД), начиная с которой размещается запись. Отметим, что этот параметр отличен от номера начальной зоны записи, если номер начальной зоны архива $\neq 0$ (п. 12.1). Второй параметр указывает длину записи в ячейках; в каждой ячейке размещается по шесть символов текста программы или данных.

* УНИЧТОЖЕНА ЗАПИСЬ В ОГЛАВЛЕНИИ. тип записи имя АРХИВ имя архива.

(И) Комментарий. По команде * УНИЧТОЖИТЬ _ ПРОГРАММУ или * УНИЧТОЖИТЬ _ ДАННЫЕ произошло вычеркивание записи из оглавления архива. Тем самым освобождаются зоны на МЛ (МД), где хранилась запись. Эти зоны будут заняты при дальнейших записях в архив. Отметим, что запись, вычеркнутая из оглавления, некоторое время может быть доступна, если она будет описана как примитивный архив (п. 12.2).

ГЛАВА 22

СООБЩЕНИЯ ПРИ ВЫПОЛНЕНИИ ПРОГРАММЫ

!ВЫХОД ПО номер-му ИНДЕКСУ МАССИВА идентификатор ЗА ВЕРХНЮЮ ИЛИ НИЖНЮЮ ГРАНИЦУ, ПЕРФОКАРТА диапазон, ИНДЕКС = номер, ГРАНИЦЫ ИНДЕКСА номер: номер.

(Ж) Комментарий. При обращении к переменной с индексами значение указанного индекса находится вне границ индекса, которые задаются описанием массива.

! ГРУППА ДАННЫХ НЕ ПОМЕЩАЕТСЯ ВО ВВОДИМЫЙ МАССИВ, ПЕРФОКАРТА диапазон, АДРЕС МАССИВА = адрес, КАНАЛ номер канала.

(Ж) Комментарий. Число значений в группе данных, которая вводится оператором *input* для массива, больше числа скалярных элементов в массиве.

! ЗАТЕРТА ЗАПИСЬ В КАНАЛЕ номер канала.

(Ж) Комментарий. Запись в канале оказалась затертой до выдачи записей операторами *output* на АЦПУ. Возможно, был неверно сдвинут указатель чтения канала.

! ЗАТЕРТ КАНАЛ номер канала, ПЕРФОКАРТА диапазон.

(Ж) Комментарий. При вводе по *input* оказалось, что текущая ячейка канала, на которую указывает стрелка чтения, не является указателем записи. Возможно, что эта область канала затерта другими обменами.

! НЕДОПУСТИМЫЙ НОМЕР КАНАЛА номер канала, ПЕРФОКАРТА диапазон.

(Ж) Комментарий. Первым аргументом в операторе *input*, *output*, *marg* и *lmarg* должен быть указан номер канала, значение которого должно находиться в пределах от 0 до 15. Возможно, этот аргумент ошибочно пропущен.

* НЕДОПУСТИМЫЙ номер АРГУМЕНТ ОПЕРАТОРА MARG, ПЕРФОКАРТА диапазон.

(Ж) Комментарий. Нарушено одно из условий: $0 \leqslant E2 \leqslant 127$, $0 < E3 \leqslant 128$, $0 \leqslant E4 \leqslant 127$, $E2 + E3 + E4 \leqslant 128$, $0 \leqslant E5 \leqslant 127$, $0 < E6 < 2048$, $0 \leqslant E7 \leqslant 256$, где $E2, \ldots E7$ — значения аргументов оператора *marg* со 2-го по 7-й.

! НЕДОПУСТИМЫЙ ТИП ВВОДИМОГО ДАННОГО, ПЕРФОКАРТА диапазон, КАНАЛ номер канала.

(Ж) Комментарий. Текущее вводимое данное не соответствует по типу объекту ввода, указанному в операторе *input*.

! НЕПАРНЫЕ ДАННЫЕ ДЛЯ ВВОДИМОГО КОМПЛЕКСНОГО МАССИВА, ПЕРФОКАРТА диапазон, АДРЕС МАССИВА = адрес, КАНАЛ = номер канала.

(Ж) Комментарий. Группа данных, которая вводится по оператору *input* для комплексного массива, состоит из нечетного числа вещественных значений. Отметим, что комплексному значению должна соответствовать в данных пара вещественных значений.

! НЕТ ДАННЫХ В КАНАЛЕ номер канала, ПЕРФОКАРТА диапазон, АДРЕС ВВОДА = адрес.

(Ж) Комментарий. Возможны следующие ошибки:

— исчерпаны данные в канале;

— данные не были оттранслированы в канал, например, из-за ошибки в данных;

— не установлен указатель чтения из канала.

! НЕТ МЕСТА В ПАМЯТИ ДЛЯ ДИНАМИЧЕСКОГО МАССИВА ДЛИНОЙ восьм. длина, АДРЕС В ПРОГРАММЕ = адрес.

(Ж) Комментарий. Исчерпана область оперативной памяти, отведенная для загружаемых модулей и динамических массивов (п. 8.2).

Рекомендация. Если адрес последней ячейки области — 52377, то перед картой * EXECUTE следует поставить карту * CALL — FICMEMORY [13], что увеличит область до ячейки с адресом 72377.

* НЕТ ФОРМАТА ДЛЯ ОБЪЕКТА ВЫВОДА, НОМЕР АРГУМЕНТА = номер, ПЕРФОКАРТА диапазон.

(Ж) Комментарий. Объекту вывода — массиву или выражению не предшествует формат вывода. При обнаружении этой ошибки отменяется вывод, который задается остальными параметрами, следующими за указанным объектом вывода. Выполнение программы продолжается с оператора, который должен выполняться после оператора *output*

Пример. В операторах: *output* (*0*, *a*, *e*) *output* (*0*, '*r*', '*a* = ', *a*) пропущен формат вывода для переменной *a*.

! ОБРАЩЕНИЕ ВЫРОЖДЕННОЙ МАТРИЦЫ идентификатор, ПЕРФОКАРТА диапазон.

(Ж) Комментарий. Матрица с нулевым детерминантом не может возводиться в отрицательную степень.

* ОШИБКА В ФОРМАТЕ ОПЕРАТОРА OUTPUT, ПЕРФОКАРТА диапазон, ФОРМАТ: текст.

(Ж) Комментарий. При обнаружении этой ошибки отменяется вывод, который задается всеми параметрами оператора *output*, следующими за ошибочным форматом. Выполнение программы продолжается с оператора, который должен выполняться после оператора *output*.

* РАБОТА С ТЕРМИНАЛОМ НЕВОЗМОЖНА.

Комментарий. Если задан режим вывода на терминал по оператору *output* и устройство дает отказ, то на АЦПУ выдается это сообщение и последующий вывод операторами *output* будет производиться на АЦПУ.

ПРИЛОЖЕНИЕ 1

ВИДЫ КОДИРОВОК И ИХ ЗАДАНИЕ

Текст заданий для системы Альфа-6 может быть отперфорирован на различных устройствах подготовки перфокарт.

Стандартный режим. В стандартном режиме карты последовательности заданий Альфа-6 воспринимаются МС Дубна, которая допускает следующие виды кодировок: УПП (БЭСМ-6), IBM, М-220. Для использования других кодировок МС Дубна, например, CDC, применяют специальную карту вида кодировки (см. [13], [19]).

Системный пакет готовится по следующим правилам.

1) На одной перфокарте должно быть отперфорировано не более 80 символов.

2) Незначащие позиции символов на перфокарте (пустые позиции), стоящие перед позициями с пробивками значащих символов, не допускаются.

3) Вложенные строки не допускаются, открывающая кавычка отождествляется с закрывающей.

4) Последовательность символов, кодирующая один основной символ языка альфа-6 должна быть отперфорирована на одной перфокарте.

5) Пробелы в конце перфокарты при вводе игнорируются (кроме пробела, используемого в кодировке служебного слова).

6) Макеты стандартных карт Альфа-6 не допускаются, необходимо использовать их текстовое представление, отперфорированное в любой из перечисленных выше кодировок (см. Приложение 10).

Режим альфа-6.

Имеется режим, в котором карты последовательности заданий для системы Альфа-6 воспринимает не МС Дубна, а система Альфа-6. В этом режиме допускаются входные кодировки системы Альфа-6 и используются правила подготовки

перфокарт, принятые для системы Альфа-6. Для указания того, что последовательность заданий воспринимается системой Альфа-6, необходимо эту последовательность заключить между картами: * BINARY и *END_BINARY, а также вместо карты * CALL_ALPHA/6 поставить карты: * MAIN_*ALFA/6 * и * EXECUTE. Все эти карты перфорируются по правилам МС Дубна, в любой из указанных выше кодировок.

Правила подготовки карт, воспринимаемых системой Альфа-6, следующие:

а) перечисленные выше правила 1—6 могут не выполняться,

б) в строках требуется соответствие открывающих и закрывающих кавычек,

в) текстовое представление стандартной карты может быть отперфорировано только на УПП (БЭСМ-6) с начала перфокарты.

В системе Альфа-6 воспринимаются следующие виды кодировок: УПП (БЭСМ-6), КУ-3, УПП (БЭСМ-4), ЕС. Для подготовки данных можно использовать устройство подготовки чисел М-220. При использовании кодировок УПП (БЭСМ-4), ЕС и М-220 необходимо перед картами в этих кодировках в каждом фрагменте ставить соответствующую карту вида кодировки, отперфорированную на УПП (БЭСМ-6):

◇◇◇—— БЭСМ — 4 (УПП)
◇◇◇—— ЕС
◇◇◇—— ЧИСЛА — М220

Любой фрагмент задания может быть составлен из перфокарт, отперфорированных на разных устройствах подготовки перфокарт. В этом случае перед каждой последовательностью карт одного вида кодировки необходимо ставить соответствующую карту вида кодировки. Для указания кодировок УПП и КУ-3 необходимо использовать карты:

◇◇◇——УПП или ◇◇◇—— КУ — 3

Вид кодировки необходимо задавать в начале каждого фрагмента с помощью карты указания вида кодировки. Исключением являются кодировки УПП (БЭСМ-6) и КУ-3, для которых не обязательно задавать вид кодировки. Вид кодировки может устанавливаться также признаком в стандартной карте, если она задается макетом. Значение признака (восьмеричное число) следующее: 0 — для УПП (БЭСМ-6) или КУ-3; 30 — для ЕС; 303 — БЭСМ-4 (УПП); 44 — ЧИСЛА М-220; 102 — КУ-3; 201 — УПП (БЭСМ-6).

ПРИЛОЖЕНИЕ 2

КОДИРОВКА ОСНОВНЫХ СИМВОЛОВ НА АЦПУ (БЭСМ-6) (ГОСТ—(10859-64), АЦПУ-128-2)

Таблица П2.1

Восьмеричный код	Символ	Восьмеричный код	Символ	Восьмеричный код	Символ	Восьмеричный код	Символ
000	0	027	[	056	о	105	n
001	1	030	]	057	п	106	q
002	2	031	*	060	р	107	r
003	3	032	‘	061	с	110	s
004	4	033	’	062	т	111	u
005	5	034	≠	063	у	112	v
006	6	035	<	064	ф	113	w
007	7	036	>	065	х	114	z
010	8	037	:	066	ц	115	¯надчерк.
011	9	040	а	067	ч	116	⩽
012	+	041	б	070	ш	117	⩾
013	—	042	в	071	щ	120	∨
014	/	043	г	072	ы	121	∧
015	,	044	д	073	ь	122	⊃
016	.	045	е	074	э	123	⌊
017	⌊	046	ж	075	ю	124	÷
020	10	047	з	076	я	125	≡
021	↑	050	и	077	d	126	%
022	(	051	й	100	f	127	◇
023	)	052	к	101	g	130	|
024	×	053	л	102	i	131	—
025	=	054	м	103	j	132	_подчерк.
026	;	055	н	104	l		

Примечание: Большие буквы кодируются двумя символами: надчеркиванием и буквой, например, большая буква *А* кодируется $\overline{a}$.

ПРИЛОЖЕНИЕ 3
ПРЕДСТАВЛЕНИЕ СЛУЖЕБНЫХ СЛОВ В КОДИРОВКАХ УПП (БЭСМ-6), ЕС, УПП (БЭСМ-4), IBM, CDC

Таблица П3.1

Служебное слово	Кодировка	Служебное слово	Кодировка
°	—×—	если	—ес■—
градус	— x —	if	—if—
примечание	—прим■—	то	—то—
comment	—комм■—	then	—then—
целый	—цел■—	иначе	—ин■—
integer	—int■—	else	—els■—
вещественный	—вещ■—	на	—на—
		goto	—go■—
real	—real—	процедура	—проц■—
комплексный	—ком■—	procedure	—pr■—
complex	—c mp■—	результат	—рез■—
логический	—лог■—	result	—res■—
boolean	—boo■—	значение	—зн■—
собственный	—соб■—	value	—val—
own	—own—	метка	—мет■—
начало	—нач■—	label	—lab■—
begin	—beg■—	⊕	—+—
конец	—кон■—	вектор	—век■—
end	—end—	матрица	—мат■—
массив	—мас■—	строка	—стр■—
array	—ar■—	string	—str■—
стоп	—сто ■—	функция	—фун■—
stop	—stop—	funct on	—fun■—
для	—для—	истина	—ист■—
for	—for—		—и—
шаг	—шаг—	true	—tru■—
step	—step—		—t—
до	—до—	ложь	—лож■—
until	—unt■—		—л—
цикл	—цик—	false	—fal■—
do	—do—		—f—
раз	—раз—	i	—i—
times	—tim■—	канал	—кан■—
пока	—пок■—	canal	—can■—
while	—whil■—	переключатель	—пер■—
		switch	—sw■—

ПРИЛОЖЕНИЕ 4

КОДИРОВКА ОСНОВНЫХ СИМВОЛОВ И СЛУЖЕБНЫХ СЛОВ НА КУ-3

Таблица П4.1

Восьмеричный код	Символ	Восьмеричный код	Символ	Восьмеричный код	Символ	Восьмеричный код	Символ	Восьмеричный код	Символ	Восьмеричный код	Символ	Восьмеричный код	Символ
000		032	'	064	фФ	116	$\leqslant$	150	β	202	**цел**	234	
001	1	033	'	065	хХ	117	$\geqslant$	151	γ	203	**вещ**	235	**вектор**
002	2	034	$\neq$	066	цЦ	120	$\prec$	152	δ Δ	204	**компл**	236	**матр**
003	3	035	<	067	чЧ	121	$\succ$	153	ε	205	**логич**	237	**строка**
004	4	036	>	070	шШ	122	$\supset$	154	ζ	206	**собств**		
005	5	037	:	071	щЩ	123	$\neg$	155	η	207	**нач**	315	**функ**
006	6	040	аА	072	ыЫ	124	÷	156	ϑθ	210	**конец**	316	**истина**
007	7	041	бБ	073	ьЬ	125	≡	157	ϰ	211	**масс**	317	**ложь**
010	8	042	вВ	074	эЭ	126	%	160	λΛ	212	**стоп**	326	**перекл**
011	9	043	гГ	075	юЮ	127	◇	161	μ	213	**для**	331	:=
012	+	044	дД	076	яЯ	130	|	162	ν	214	**шаг**	337	, . ,
013	—	045	еЕ	077	dD	131	—	163	ξΞ	215	**до**	340	{
014	/	046	жЖ	100	fF	132	подчерк.	164	π	216	**цикл**	341	}

015	,	047	зЗ	101	gG	133		165	ρ	217	раз	345	i
016	.	050	иИ	102	iI	134		166	σΣ	220	пока		
017	—	051	йЙ	103	jJ	135		167	τ	221	если		
020	10	052	кК	104	lL	136	°градус	170	φ	222	то		
021	↑	053	лЛ	105	nN	137		171	χ	223	иначе		
022	(	054	мМ	106	qQ	140		172	ψΨ	224	на		
023	)	055	нН	107	rR	141		173	ωΩ	225	проц		
024	×	056	оО	110	sS	142		174	hH	226	результ		
025	=	057	пП	111	uU	143		175	mM	227	знач		
026	;	060	рР	112	vV	144	∅	176	tT	230	метка		
027	[	061	сС	113	wW	145		177		231	⊕		
030	]	062	тТ	114	zZ	146		200	0	232			
031	*	063	уУ	115		147	α	201	прим	233			

Примечание: Код больших букв получается добавлением к коду соответствующей малой буквы 1 в 8 разряде (т. е. код большой буквы больше малой на 200). Служебные слова, которые отсутствуют в таблице, кодируются с помощью символа подчеркивание также, как в кодировке УПП (БЭСМ-6) (см. Приложение 3).

ПРИЛОЖЕНИЕ 5

ПРЕДСТАВЛЕНИЕ ГРЕЧЕСКИХ БУКВ В КОДИРОВКАХ УПП (БЭСМ-6), УПП (БЭСМ-4), IBM, CDC

Таблица П5.1

Греческая буква	Кодировка	Большая греческая буква		Кодировка
α	◇a	A	*	⁻a
β	◇b	B	*	⁻в
γ	◇g	Γ	*	⁻г
δ	◇d	Δ		◇⁻d
ε	◇e	E	*	⁻e
η	◇h	H	*	⁻h
ν	◇u	Θ		◇⁻u
κ	◇k	K	*	
λ	◇l	Λ		◇⁻l
μ	◇m	M	*	⁻m
ν	◇n	N	*	⁻n
ξ	◇q	Ξ		◇⁻q
π	◇p	P	*	⁻p
ρ	◇r	R	*	⁻r
ζ	◇z	Z	*	⁻z
σ	◇s	Σ		◇⁻s
τ	◇t	T	*	⁻т
φ	◇f	Φ	*	⁻ф
χ	◇x	X	*	⁻x
ψ	◇y	Ψ		◇⁻y
ω	◇w	Ω		◇⁻w

Примечание: Отмеченные * большие буквы отождествляются с соответствующими большими буквами латинского алфавита.

ПРИЛОЖЕНИЕ 6

ОТОЖДЕСТВЛЕНИЯ В КОДИРОВКЕ КУ-3

В таблице П6.1. содержатся пары символов (столбцы 1 и 2), которые отождествляются транслятором альфа-6. Кодировки отождествляемых символов указаны в столбцах 2 и 4.

Таблица П6.1

Символ входной кодировки	Кодировка символа	Символ отождествления	Кодировка символа отождествления
∅	144	0	200
{	340	начало	207
}	341	конец	210
вектор	235	массив	211
матрица	236	массив	211
$m_{лат}$	175	м	054
$t_{лат}$	176	т	062
$h_{лат}$	174	н	055
$M_{лат}$	375	М	254
$T_{лат}$	376	Т	262
$H_{лат}$	374	Н	255
$A_{греч}$	347	А	240
$B_{греч}$	350	В	242
$Г_{греч}$	351	Г	243
$E_{греч}$	353	Е	245
$Z_{греч}$	354	Z	314
$H_{греч}$	355	Н	255
$K_{греч}$	357	К	252
$M_{греч}$	361	М	254
$N_{греч}$	362	N	305
$П_{греч}$	364	П	257
$P_{греч}$	365	Р	260
$T_{греч}$	367	Т	262
$Ф_{греч}$	370	Ф	264
$X_{греч}$	371	Х	265

ПРИЛОЖЕНИЕ 7

ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ СИМВОЛОВ В СТРОКАХ

В языке альфа-6 различают два вида основных символов: базовые символы и прочие (не базовые) символы. Любой базовый символ в строке при выполнении программы соответствует одному определенному целому числу. Символ, не являющийся базовым (прочий), представляется последовательностью целых чисел, соответствующих базовым символам в этом представлении.

Таблица П7.1

Соответствие между базовыми символами строки и целыми числами

Символ	Число	Символ	Число	Символ	Число	Символ	Число	Символ	Число
0	0	(	18	д	36	ц	54	s	72
1	1	)	19	е	37	ч	55	u	73
2	2	×	20	ж	38	ш	56	v	74
3	3	=	21	з	39	щ	57	w	75
4	4	;	22	и	40	ы	58	z	76
5	5	[	23	й	41	ь	59	⁻надч.	77
6	6	]	24	к	42	э	60	⩽	78
7	7	*	25	л	43	ю	61	⩾	79
8	8	‘	26	м	44	я	62	ˇ	80
9	9	’	27	н	45	d	63	^	81
+	10	≠	28	о	46	f	64	⊃	82
—	11	<	29	п	47	g	65	¬	83
/	12	>	30	р	48	i	66	÷	84
,	13	:	31	с	49	j	67	≡	85
.	14	а	32	т	50	l	68	◇	87
⎵	15	б	33	у	51	n	69	|	88
10	16	в	34	ф	52	q	70	⁻подч.	90
↑	17	г	35	х	53	r	71	—тире	11

Таблица П7.2

Представление заглавных букв русского и латинского алфавитов в строках последовательностями базовых символов

Заглавная буква	Представление последовательностью базовых символов	Заглавная буква	Представление последовательностью базовых символов	Элемент строки	Представление последовательностью базовых символов
А	-а	Р	-р	F	-f
Б	-б	С	-с	G	-g
В	-в	Т	-т	I	-i
Г	-г	У	-у	J	-j
Д	-д	Ф	-ф	L	-l
Е	-е	Х	-х	N	-n
Ж	-ж	Ц	-ц	Q	-q
З	-з	Ч	-ч	R	-r
И	-и	Ш	-ш	S	-s
Й	-й	Щ	-щ	U	-u
К	-к	Ы	-ы	W	-w
Л	-л	Ь	-ь	V	-v
М	-м	Э	-э	Z	-z
Н	-н	Ю	-ю	:=	:=
О	-о	Я	-я	...	...
П	-п	D	-d		

Примечание: Заглавные буквы русского и латинского алфавитов кодируются надчеркиванием и строчной буквой алфавитов; символ:=(знак присваивания) кодируется двоеточием и равенством, а символ ... (троеточие) — тремя десятичными точками.

Таблица П7.3

Представления букв греческого алфавита в строках последовательностями базовых символов

Греческая буква	Представление последовательностью базовых символов	Большая греческая буква	Представление последовательностью базовых символов
α	*альф*	Α	~а
β	*бета*	Β	~в
γ	*гам*	Γ	~г
δ	*дел*	Δ	~*дел*
ε	*эпс*	Ε	~е
η	*эта*	Η	~н
θ	*тэта*	Θ	~*тэт*
κ	*кап*	Κ	~к
λ	*лам*	Λ	*~лам*
μ	*мю*	Μ	~м
ν	*ню*	Ν	~п
ξ	*кси*	Ξ	~*кси*
π	*пи*	Π	~п
ρ	*ро*	Ρ	~р
ζ	*дзет*	Ζ	~z
σ	*сиг*	Σ	~*сиг*
τ	*тау*	Τ	~т
φ	*фи*	Φ	~ф
χ	*хи*	Χ	~х
ψ	*пси*	Ψ	~*пси*
ω	*ом*	Ω	~*ом*

Примечание: Прописная греческая буква в строках представляется последовательностью прописных букв, окаймленной символом * (звездочка). Заглавные греческие буквы кодируются (кроме тех, которые совпадают по написанию с заглавными буквами русского или латинского алфавитов, последние кодируются надчерком и прописной буквой соответствующего алфавита) надчерком и последовательностью базовых символов, соответствующей представлению прописной греческой буквы.

Таблица П7.4

Представление служебных слов и некоторых знаков операций последовательностями базовых символов

Служебное слово	Представление последовательностью базовых символов	Элемент строки	Представление последовательностью базовых символов
вещественный	_вещ_	**примечание**	_прим_
для	_для_	**процедура**	_проц_
до	_до_	**раз**	_раз_
если	_если_	**результат**	_рез_
значение	_знач_	**собственный**	_соб_
иначе	_инач_	**стоп**	_стоп_
истина	_ист_	**строка**	_стр_
конец	_кон_	**то**	_то_
комплексный	_комп_	**функция**	_функ_
логический	_лог_	**целый**	_цел_
ложь	_ложь_	**цикл**	_цикл_
массив	_мас_	**шаг**	_шаг_
на	_на_	**метка**	_мет_
начало	_нач_	$\circ$	-×_
переключатель	_пер_	$\oplus$	-+_
пока	_пока_	i	_i_

Примечание: Служебные слова в строках кодируются подчерком, последовательностью прописных букв или знаков и пробелом. Знак операции $\circ$ (покомпонентное умножение) кодируется подчерком, знаком умножения и пробелом. Знак $\oplus$ (сложение по модулю два) кодируется подчерком, знаком плюс и пробелом.

ПРИЛОЖЕНИЕ 8

РЕСУРСЫ БЭСМ-6, ТРЕБУЮЩИЕСЯ ДЛЯ РАБОТЫ СИСТЕМЫ АЛЬФА-6

Ленты (диски) с математическими номерами:
33 — система Альфа-6 (4—200 зоны),
31 — библиотечный архив *LIBRA*,
30 — МС Дубна,
32 — рабочий буфер Альфа-6.
Могут использоваться тракты на МБ с номерами:
01—07, 20—27 — МС Дубна [19, стр. 228],
10—14 — система Альфа-6.
ОЗУ: 0—37 — система Альфа-6.
Перфоратор, если есть выдача на ПИ.
Терминал (телетайп или видеотон), если есть заказ в паспорте «теле», в этом случае выдача операторами *output* ведется на терминал.
АЦПУ — одно.
Стандартное время ЦП (центрального процессора), заказанное для работы системы Альфа-6 — 2 мин 43 сек.
Стандартное количество трактов, заказанное для работы системы Альфа-6 — 96.
Максимальное время одной трансляции — 9 мин.
Максимальное количество трактов, занимаемых системой Альфа-6 — 165.

ПРИЛОЖЕНИЕ 9

ПАСПОРТ И КАРТЫ ВЫЗОВА СИСТЕМЫ

Информация для системы Альфа-6 задается с помощью пакета перфокарт, изображенного на рис. 13.1.
Системный пакет состоит из следующих частей:
1)*) Карта шифра задачи.
ШИФР _ XXXXXX ЗС+⁻ или,
ШИФР _ XXXXXXXXXXXX ЗС+⁻
где X...X — десятичные цифры — шифр задачи.
2) Карта КОНЕЦ ПАСПОРТА (вспомогательная карта).
3) Карта * NAME.
4) Карта * ASSIGN _ LIBRARY _ N (где N — число — номер библиотеки подпрограмм Альфа-6).

*) Карта шифра заказывает стандартный паспорт Альфа-6 в операционной системе Диспак [9], эксплуатирующейся в ВЦ СО АН СССР.

5) Карта * CALL — ALPHA/6.

6) Последовательность заданий, подлежащих выполнению системой, разделенных стандартной картой ЗАДАНИЕ.

7) Карта ◇◇◇КНЦ.

8) Карты задания на счет, режимы счета (* MAIN, * CALL — FICMEMORY, * EXECUTE).

9) Карта * END — FILE.

10) Карта ДК (диспетчерский конец), служащая признаком конца задания для МС Дубна (вспомогательная карта).

11) Карта ЕКОНЕЦ, являющаяся концом системного пакета (вспомогательная карта).

Карта шифра и вспомогательные карты перфорируются во входной кодировке ОС Диспак [9]. Карты частей 3—9 могут быть отперфорированы в одной из кодировок МС Дубна (см. Приложение 1).

А. Паспорт.

Стандартный паспорт, заказываемый картой шифра, имеет следующие разделы:

ЛИСТЫ — 0 — 37⁻
ТРАКТЫ — 96⁻
ВРЕМЯ — 243⁻
АЦПУ — 7⁻
АВОСТ⁻
ЛЕНТА — 30 (БОБ1) 31 (БОБ2) 32 (БОБ3) 33 (БОБ4)⁻

где: БОБ1 — номер бобины, с которой считывается МС Дубна
БОБ2 — номер бобины библиотечного архива,
БОБ3 — номер рабочей бобины системы Альфа-6,
БОБ4 — номер бобины с системой Альфа-6.

В зависимости от задачи пользователя стандартный паспорт может быть дополнен картами следующих разделов паспорта, вставляемых после карты шифра и отперфорированных на УПП.

1) ВРЕМЯ — ЧЧММСС⁻,

где Ч, М, С — десятичные цифры, ЧЧ — часы, ММ — минуты, СС — секунды.

В этом разделе пользователь указывает время работы системы. Нули слева можно опустить.

2) АЦПУ — ХХ⁻,

где ХХ — десятичное число, указывающее количество метров бумаги, необходимое для вывода на АЦПУ ($1 \leqslant XX \leqslant 64$). Нуль слева можно опускать.

3) ТРАКТЫ — ХХХ⁻

где ХХХ — десятичное число, обозначающее количество трактов

МБ, необходимое для работы системы ($65 \leqslant$ XXX $\leqslant 144$). Нуль слева можно опускать.

4) ЛЕНТА—АБ (УУУУ — ЗП)⁻ или ЛЕНТА— АБ (УУУУ)⁻
где А — математический номер направления ($3 \leqslant А \leqslant 6$),

Б — математический номер магнитофона ($0 \leqslant Б \leqslant 7$),
УУУУ — десятичный номер бобины ($1 \leqslant$ УУУУ $\leqslant 2047$).

Буквы ЗП обозначают разрешение записи на данную МЛ. При отсутствии пункта ЗП (ЛЕНТА—АБ (УУУУ)⁻) информацию можно только считывать с данной ленты. В качестве разделителя между УУУУ и ЗП используется символ минус.

Раздел ЛЕНТА может служить для заказа лент архивов (см. гл. 12), лент каналов ввода/вывода (п. 2.14), а также лент, используемых для хранения премежуточных значений с помощью операторов *copy* (2.20). В одном паспорте может быть несколько разделов ЛЕНТА.

Пример. Раздел: лента _52 (496—ЗП)⁻ заказывает бобину 496, разрешает запись на нее при выполнении программы и приписывает ей математический номер направления и номер магнитофона — 52.

5) ВЫВОД⁻

Этот раздел указывается только в том случае когда задан вывод на перфокарты данных, программы (с помощью системных команд ПЕРФОРИРОВАТЬ—...).

6) Можно пополнить паспорт другими разделами, не описанными в стандартном паспорте, в соответствии с правилами пользования операционной системой Диспак [9].

Б. Вспомогательные карты.

1) КОНЕЦ ПАСПОРТА

```
Е Е В 00 000-0001 00 000 0000 А3
```

2) ДК (диспетчерский конец)

Перфокарта имеет пробивки в 40-й и 80-й колонках.

3) ЕКОНЕЦ

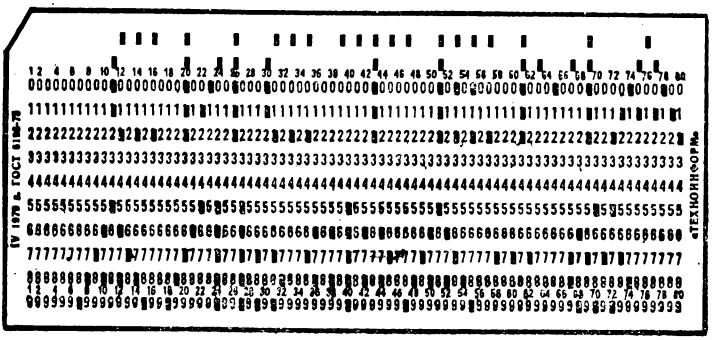
```
ЕКОНЕЦ
```

ПРИЛОЖЕНИЕ 10

МАКЕТЫ СТАНДАРТНЫХ КАРТ

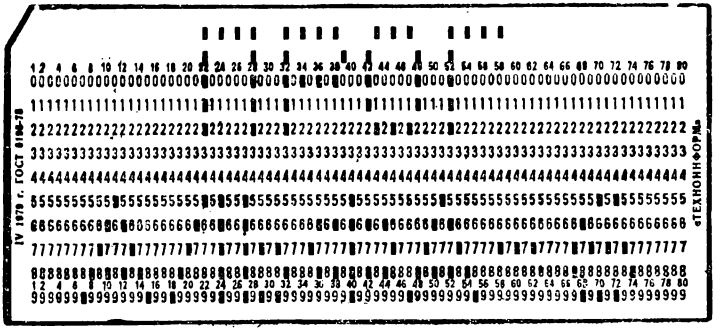
Любая из стандартных карт имеет два представления: в виде макета и в виде текста, отперфорированного на отдельной карте (см. приложение 1). Перфорируемый текст стандартной карты указан под макетом соответствующей стандартной карты.

1) Карта С И С Т Е М А .



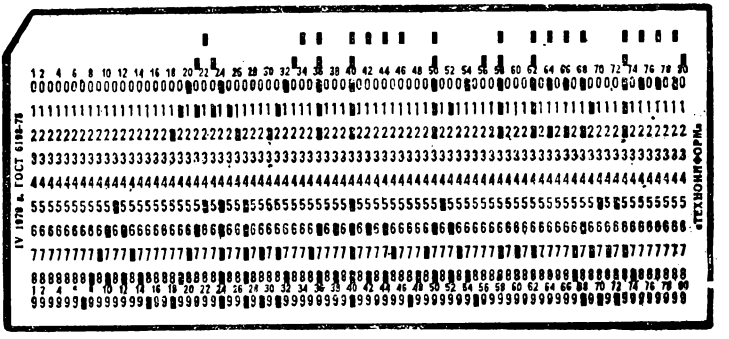
◊ ◊ ◊ СИСТЕМА

2) Карта П Р О Г Р А М М А .



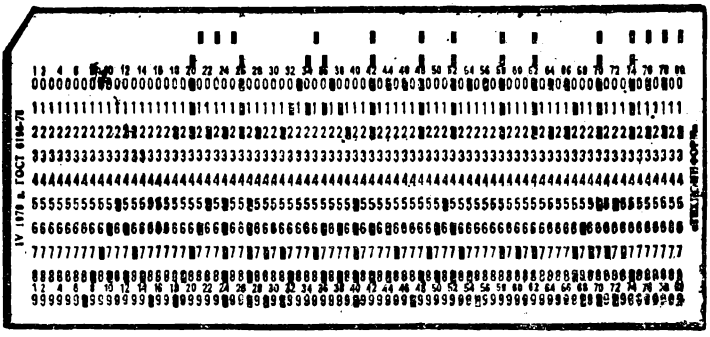
◊ ◊ ◊ ПРОГРАММА

3) Карта А Л Г И Б Р .



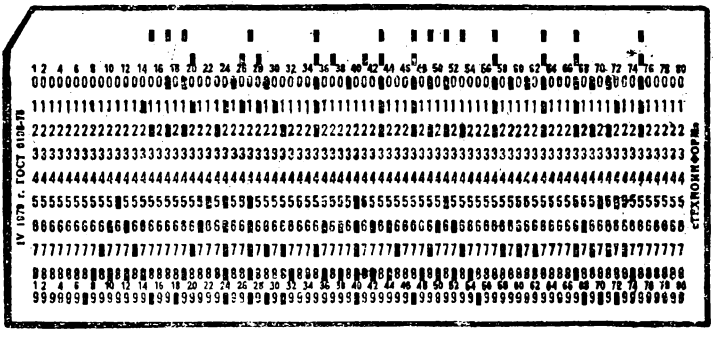
◊ ◊ ◊ АЛГЕБР

4) Карта Д А Н Н Ы Е .



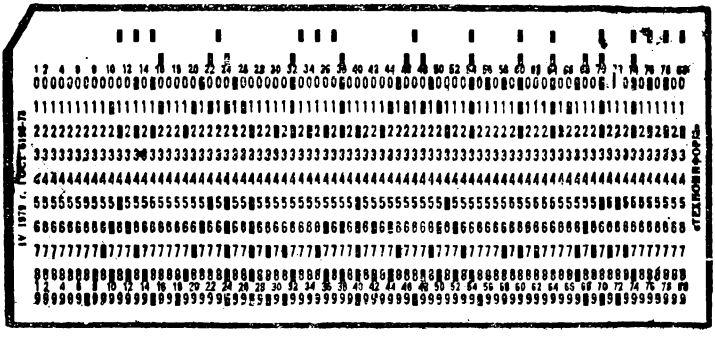
◊◊◊ ДАННЫЕ

5) Карта З А М Е Н Ы .



◊◊◊ ЗАМЕНЫ

6) Карта З А Д А Н И Е .



◊◊◊ ЗАДАНИЕ

ПРИЛОЖЕНИЕ 11
СПИСОК СИСТЕМНЫХ КОМАНД

Таблица П11.1

Минимальное сокращение имени команд	Системная команда	Вид	Назначение
*П	*ПРОГРАММА (ИМЯ)	Задание программы и данных	Указывает, что программа с именем *ИМЯ* находится в другом задании системного пакета (на перфокартах).
* П	*ПРОГРАММА (ИМЯ, АРХ)		Указывает, что программу с именем *ИМЯ* нужно взять из архива с именем *АРХ*.
*Д	* ДАННЫЕ (ИМЯ)		Указывает, что данные с именем *ИМЯ* находятся в другом задании системного пакета (на перфокартах).
*Д	*ДАННЫЕ (ИМЯ, АРХ)		Указывает, что данные с именем *ИМЯ* нужно взять из архива с именем *АРХ*.
*А	* АРХИВ (АРХ, МЕСТО) *АРХИВ (АРХ, МЕСТО, СДВИГ)		Описывает архив с именем *АРХ*; *МЕСТО* — это либо математический номер-направление МЛ архива с оглавлением, либо математический номер-направление — зона МЛ примитивного архива (см. главу 12). Третий аргумент команды *СДВИГ* задает восьмеричный номер начальной зоны архива на носителе.

Таблица П11.1 (продолжение)

Минимальное сокращение имени команд	Системная команда	Вид	Назначение
*З_П	*ЗАПИСАТЬ_ ПРОГРАММУ (ИМЯ, АРХ)	Запись или перфорация программы (данных)	Запись программы или данных с именем *ИМЯ* в архив с именем *АРХ*.
* З_П	*ЗАПИСАТЬ ДАННЫЕ (ИМЯ, АРХ)		
*К_Д	* КОНТРОЛЬ_ ДАТЫ		Отменяет повторную запись в архив программы или данных в течение одних суток.
*ПЕР_П	* ПЕРФОРИРОВАТЬ_ ПРОГРАММУ		Перфорация программы или данных в кодировке УПП (БЭСМ-6).
*ПЕР_Д	* ПЕРФОРИРОВАТЬ_ ДАННЫЕ		
*Б_ПЕ	* БЕЗ_ПЕЧАТЕЙ	Печати	Запрет печати фрагментов (системной программы, данных программы, фрагмента замен).
*Б_П_З	*БЕЗ_ ПЕЧАТИ_ ЗАМЕН		Запрет печати фрагмента замен.
* П_П *П_Д	* ПЕЧАТАТЬ_ ПРОГРАММУ * ПЕЧАТАТЬ_ ДАННЫЕ * ПЕЧАТАТЬ_ ПРОГРАММУ (ДИАП) * ПЕЧАТАТЬ_ ДАННЫЕ (ДИАП)		Задает печать программы или данных, *ДИАП* — номер одной перфокарты или диапазон перфокарт,
* П_МО	* ПЕЧАТАТЬ_ МОДУЛЬ * ПЕЧАТАТЬ_ МОДУЛЬ (ДИАП)		Задает печать модуля программы после трансляции.

Таблица П11.1 (продолжение)

Минимальное сокращение имени команд	Системная команда	Вид	Назначение
* П_МА	* ПЕЧАТАТЬ_ МАССИВЫ	Печати	Задает печать поля массивов в сбойной распечатке.
* П_Б_Ч	* ПЕЧАТАТЬ_ БИБЛИОТЕЧНУЮ_ЧАСТЬ		Задает распечатку вставляемых (по указанию системной команды * *ПРОЦЕДУРЫ*) библиотечных описаний.
* ВС	* ВСТАВИТЬ (НОМЕР, ДИАП)	Редактирование	Вставить в программу или данные после перфокарты с номером *НОМЕР* текст перфокарт, хранящихся во фрагменте замен, и указанных *ДИАП* (номером одной перфокарты или диапазоном).
* ВЫ	* ВЫБРОСИТЬ (ДИАП)		Выбросить из программы или данных текст перфокарт, указанных *ДИАП*.
*З	* ЗАМЕНИТЬ (Д1, Д2) * ЗАМЕНИТЬ (Д1)		Заменить текст перфокарт в программе или данных, указанный *Д1* на текст перфокарт, находящийся во фрагменте замен, и указанный *Д2*. *Д1*, *Д2* — номера перфокарт или диапазоны перфокарт. Если *Д1* равно *Д2*, то можно указать один аргумент.
*З_Т	* ЗАМЕНИТЬ_ ТЕКСТ (КОНТЕКСТ, ТЕКСТ ЗАМЕН)		Во всей программе или в данных *КОНТЕКСТ* заменить на *ТЕКСТ ЗАМЕН*. Оба аргумента системной команды являются строками.

Таблица П11.1 (продолжение)

Минимальное сокращение имени команд	Системная команда	Вид	Назначение
*З_Т	* ЗАМЕНИТЬ_ ТЕКСТ (КОНТЕКСТ, ТЕКСТ ЗАМЕН, ДИАП)	Редактирование	То же, что и предыдущая команда, только замены осуществляются в пределах *ДИАП* (номер одной перфокарты или диапазон перфокарт).
* З—И	* ЗАМЕНИТЬ_ ИДЕНТИФИКАТОР (ИДЕНТ, ТЕКСТ ЗАМЕН)		Заменить идентификатор в программе или данных на *ТЕКСТ ЗАМЕН*. Второй аргумент системной команды — строка.
	* ЗАМЕНИТЬ_ ИДЕНТИФИКАТОР (ИДЕНТ, ТЕКСТ ЗАМЕН, ДИАП)		То же, что и в предыдущей команде, только замены осуществляются в пределах *ДИАП* (номер одной перфокарты или диапазон перфокарт).
* Н	* НОМЕР		Программа или данные нумеруются в конце редактирования, заданного командами системной программы. Перенумерация заключается в том, что вместо старых номеров перфокарт, содержащихся в тексте, вставляются номера нормальной нумерации, начиная с десятичного числа, указанного в аргументе. Если аргумент отсутствует, то нумерация производится с единицы. Текст программы или данных на перфокартах, не содержащий номеров перфокарт эквивален-
	* НОМЕР (НАЧ НОМЕР ПК)		

Таблица П 11.1 (продолжение)

Минимальное сокращение имени команд	Системная команда	Вид	Назначение
		Редактирование	тен соответствующему тексту, у которого в начале каждой карты стоит номер перфокарты, определяемой нормальным номером этой карты во фрагменте (имя фрагмента не нумеруется).
* З_О	* ЗАПИСАТЬ_ОГЛАВЛЕНИЕ (АРХ) * ЗАПИСАТЬ_ОГЛАВЛЕНИЕ (АРХ, ДЛ)	Архивные работы	Создает оглавление архива с именем *АРХ* на той МЛ, которая указана в системной команде * *АРХИВ*. Второй аргумент задает восьмеричное число зон, занимаемое архивом.
* П_О	* ПЕЧАТАТЬ_ОГЛАВЛЕНИЕ (АРХ)		Задает печать оглавления архива с именем *АРХ*.
* У_П	* УНИЧТОЖИТЬ_ПРОГРАММУ (ИМЯ, АРХ)		Удаляет из оглавления архива *АРХ* программу или данные с именем *ИМЯ*.
* У_Д	* УНИЧТОЖИТЬ_ДАННЫЕ (ИМЯ, АРХ)		
* И_И_А	* ИЗМЕНИТЬ_ИМЯ_АРХИВА (СТАРОЕ ИМЯ, НОВОЕ ИМЯ)		Изменяет *СТАРОЕ ИМЯ* оглавления на *НОВОЕ ИМЯ*.
* Б_Т	* БЕЗ_ТРАНСЛЯЦИИ	Режимы	Блокирует трансляцию программы или данных (см. граф работ).

Таблица П11.1 (продолжение)

Минимальное сокращение имени команд	Системная команда	Вид	Назначение
* Б_ПР	* БЕЗ_ ПРИМЕЧАНИЙ	Режимы	Блокирует печать предупреждающих сообщений о наличии комментариев в программе.
* Б_К_Д	* БЕЗ_ КОНСТАНТНЫХ_ ДЕЙСТВИЙ		Блокирует оптимизирующие действия при трансляции, связанные с вычислением константных выражений в программе.
* Р	* РЕГИСТРЫ (СПИСОК РЕГИСТРОВ)		Запрещает транслятору использовать в модуле индекс-регистры, перечисленные в списке аргументов системной команды.
* ОБ	* ОБЛАСТЬ (МЕСТО, ЗОНА, СДВИГ)	Спецификации	Задает начало внешней области для *EX*-массивов программы, где *МЕСТО* — восьмеричный номер и направление носителя, *ЗОНА* — десятичный номер начальной зоны, *СДВИГ* — десятичный адрес относительно начала начальной зоны.
* ПРОЦ	* ПРОЦЕДУРА (СПИСОК ИМЕН) * ПРОЦЕДУРА (СПИСОК ИМЕН, АРХ)		Сообщает системе имена библиотечных описаний перечисленных в списке аргументов системной команды. Если последний аргумент команды является именем архива (*АРХ* описан в команде * АРХИВ), то все библиотечные описания по списку берутся из указанного архива.

Таблица П11.1 (продолжение)

Минимальное сокращение имени команд	Системная команда	Вид	Назначение
* П_МЕ	* ПЕЧАТАТЬ_ МЕТКИ * ПЕЧАТАТЬ_ МЕТКИ (ДИАП)	Отладочные команды	Инициирует вставку в модуль команд, осуществляющих печать имен меток при передаче управления на них в процессе выполнения программы. Если указан *ДИАП* (диапазон перфокарт или номер перфокарты), то вставки осуществляются в пределах *ДИАП*.
* ЗН	* ЗНАЧЕНИЕ (ИДЕНТ)		Инициирует вставку в модуль команд печати *ИДЕНТ* и его значения после каждого присваивания данному идентификатору какого-то значения (точнее каждого вхождения идентификатора *ИДЕНТ* в левую часть оператора присваивания).
	* ЗНАЧЕНИЕ (ИДЕНТ, ДИАП)		То же, что и для предыдущей команды, только в пределах *ДИАП* (номер перфокарты или диапазон перфокарт).
* Г	* ГРАНИЦЫ		Инициирует вставку в модуль команд, контролирующих выход за границы массивов.
* Л_А	* ЛОКАЛИЗОВАТЬ_ АВОСТ		Инициирует вставку в модуль специальных кодов, позволяющих локализовать авост при выполнении программы с точностью до 8 команд БЭСМ-6.

Таблица П11.1 (окончание)

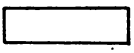
Минимальное сокращение имени команд	Системная команда	Вид	Назначение
* К_П	* КОНТРОЛИРОВАТЬ_ ПАРАМЕТРЫ	Отладочные команды	Задает режим автоматического контроля соответствия формальных и фактических параметров внешних процедур по типу, виду и структуре.
* Б_СБ	* БЕЗ_СБОЙНОЙ		Задает режим трансляции, по которому в модуль не вставляется сбойная программа, т. образом при прерывании выполняемой программы не будут выданы таблицы распределения памяти, значений переменных, полей и т. д.
* О_П_О	* ОСТАНОВ_ ПРИ_ОШИБКЕ		Если при выполнении предыдущих заданий были ошибки, обнаруженные системой, то задание, в котором находится данная системная команда, и все последующие не выполняются.

Примечание: В системной программе можно пользоваться сокращенными именами системных команд, которые получаются приписыванием к словам в минимальном сокращении (1-й столбец таблицы) любых строчных букв русского алфавита. Например, системная команда *ЗАПИСАТЬ_ПРОГРАММУ может употребляться в следующих сокращениях: * З_П, * ЗА_П, * ЗАП_ПРОГ и т. д.

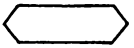
ПРИЛОЖЕНИЕ 12

ГРАФ РАБОТ.

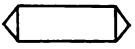
Обозначения.



Прямоугольник — обозначает работу системы, смысл работы дается текстом внутри прямоугольника.

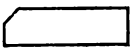


Шестиугольник — обозначает наличие в системной программе хотя бы одной из системных команд, перечисленных внутри фигуры.



Шестиугольник перечеркнутый — обозначает необходимость наличия в системной программе всех системных команд, перечисленных внутри фигуры.

— Обозначает коммептарий к дуге графа, текст комментария помещен внутри фигуры.



Пятиугольник — обозначает наличие в системном пакете фрагмента, тип которого указан текстом внутри фигуры.

Круг — обозначает вершину графа.

— Указание па продолжение дуги графа на другой странице. Внутри фигуры ссылка на обозначенный участок ветви, либо обозначение ветви, на которую указывает данная фигура.

Сокращения системных команд даются в приложении 11.

Каждый путь графа указывает на список работ, выполняемых системой в одном задании, а также на набор фрагментов и системных команд, обеспечивающих этот список работ.

Непустой список системных команд указывает на наличие в задании системного фрагмента.

Рекомендации к использованию графа работ. Граф работ может оказаться полезным при использовании его в следующих целях.

А. Определение списка системных команд и набора фрагментов в задании для обеспечения данной последовательности работ. Для этой цели выбирают путь, начиная от начала графа, следуя по стрелкам через вершины, обозначающие требуемые виды работ. Каждая вершина может пополнить задание указанным в ней фрагментом (данные, программа) и (или) необходимым списком системных команд, выбранным в соответствии с их семантикой.

B. Определение совместности в одном задании данного набора работ, системных команд, фрагментов. Для этого достаточно убедиться в существовании пути, вершины которого охватили бы весь данный набор работ.

C. Определение последовательности работ по наличию фрагментов и списка команд в одном задании. В этом случае поступают как и в случае A, выбирая вершины, содержащие указание на имеющиеся системные команды и (или) фрагменты.

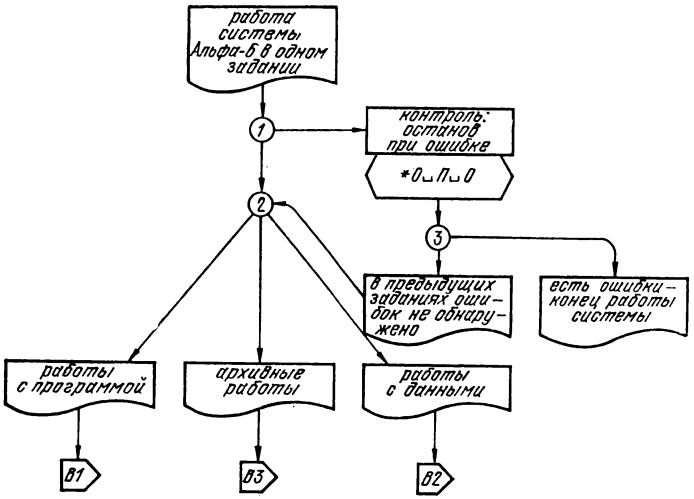


Рис. П12.1. Работы системы Альфа-6 в одном задании.

Примечание 1. Часть графа работ, изображенная на рис. П12.1., показывает, что работы системы делятся на три группы, и отсылает к соответствующим частям графа. Никакие две работы из разных групп не могут быть выполнены в одном задании, но они могут выполняться в последовательности заданий одного системного пакета. Правая ветвь узла 1 показывает, что пользуясь системной командой * ОСТАНОВ — ПРИ — ОШИБКЕ, можно прекратить выполнение последовательности заданий, если в одном из них обнаружена ошибка. Например, при выполнении последовательности из двух заданий: запись оглавления архива и запись программы в архив обнаруживается, что в паспорте системного пакета не заказана лента архива, тогда записи оглавления не произойдет и,

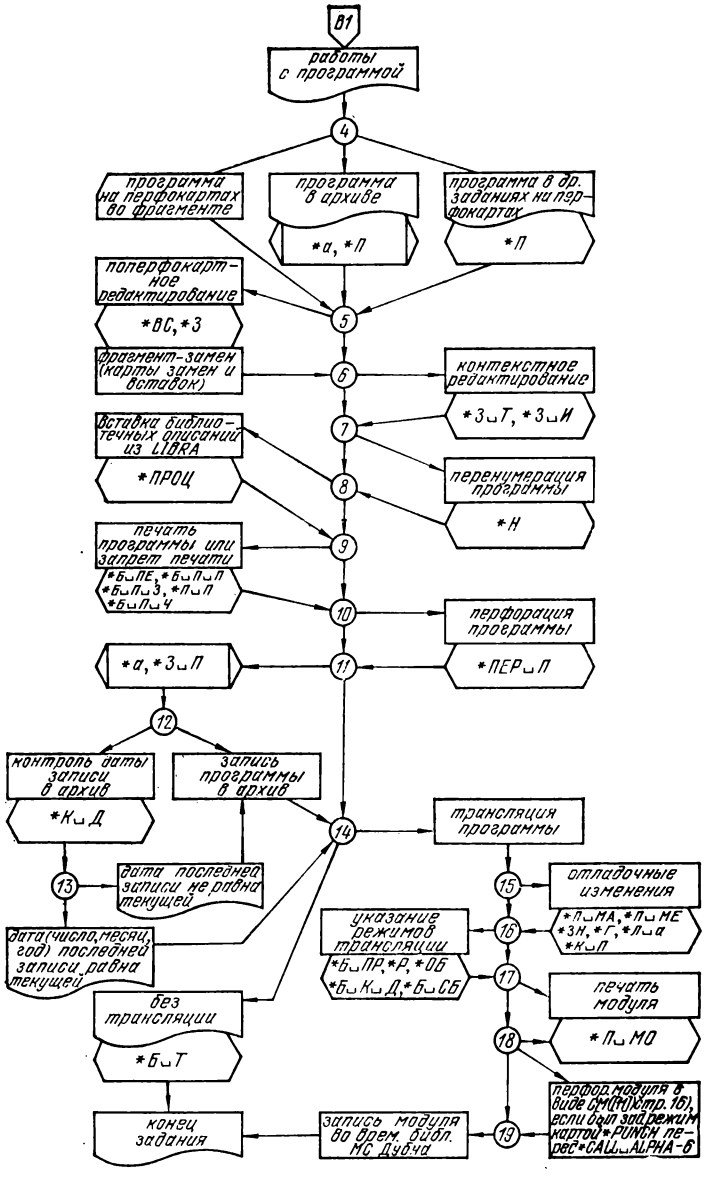


Рис. П12.2. Работы с программой.

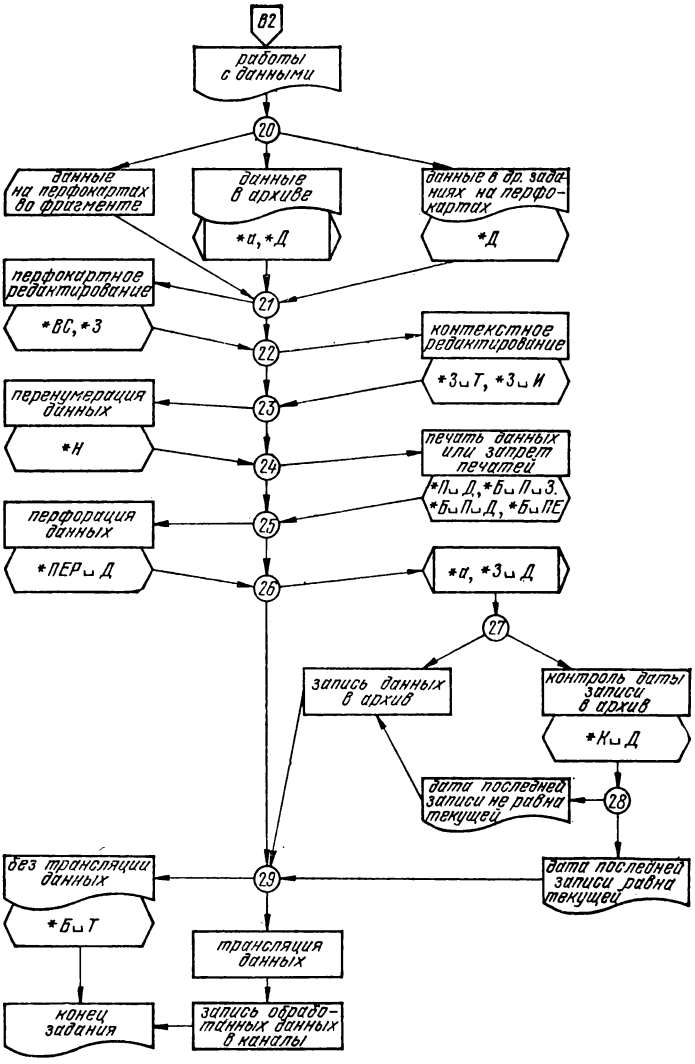


Рис. П12.3. Работы с данными.

следовательно, второе задание выполнять не нужно. Для этого во второе задание необходимо вставить системную команду * ОСТАНОВ␣ ПРИ ␣ОШИБКЕ.

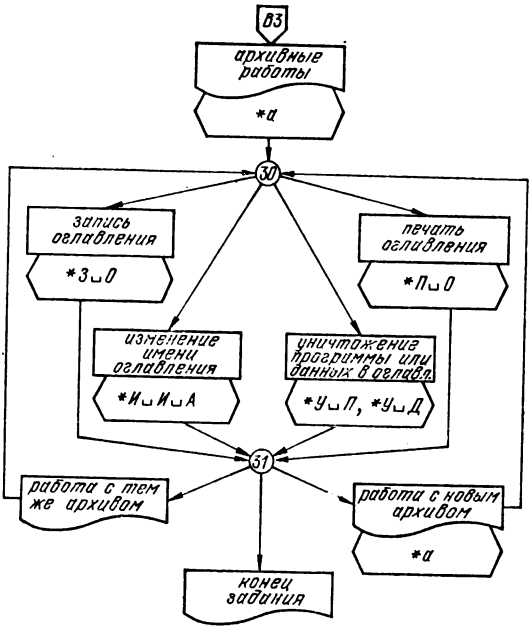


Рис. П 12.4. Архивные работы.

Примечание 2 к рис. П12.2. Часть графа, изображенная на рис. П12.2., показывает состав и последовательность выполнения работ над программой.

Проиллюстрируем использование графа для примера 3 гл. 3. Необходимо составить задание для записи программы на перфокартах в архив без ее трансляции. Проследим граф работ от B1 до «конец задания». Из трех ветвей узла 4 нашим целям соответствует крайняя левая ветвь (до узла 5). Она не содержит шестиугольника и следовательно не требует системных команд дополнительно к программному фрагменту. Далее следуем вниз по узлам 5—10 до 11. От узла 11 нужно пройти по левой ветви через узел 12, так как необходимо выполнить работу «запись программы в архив». В этой ветви требуется сразу две системные команды: * АРХИВ и * ЗАПИСАТЬ —

ПРОГРАММУ. При первой записи программы в архив контроль даты смысла не имеет, поэтому системную команду * КОНТРОЛЬ — ДАТЫ ставить не требуется. От узла 14 идем по левой ветви, потому что трансляцию осуществлять не нужно. Для этого режима требуется системная команда * БЕЗ — ТРАНСЛЯЦИИ.

Таким образом, задание состоит из системной программы и фрагмент — программы. Системная программа состоит из трех системных команд: * АРХИВ, * ЗАПИСАТЬ — ПРОГРАММУ и * БЕЗ — ТРАНСЛЯЦИИ. Если бы нам требовалось только трансляция программы, то системная программа могла бы отсутствовать (в узлах 11 и 14 следуем правым ветвям, см. пример 1 гл. 3).

Примечание 3 к рис. П12.4. Особенностью данного подграфа ВЗ является то, что левая и правая ветви узла 31 заканчиваются на узле 30, образуя циклы. Это означает, что в одном задании могут выполняться любые из указанных работ, в любом порядке и любом количестве. Порядок выполнения архивных работ определяется порядком вхождения системных команд в системном фрагменте. Об ограничении количества системных команд в одном задании см. Приложение 15.

ПРИЛОЖЕНИЕ 13

СПИСОК КОМАНД ОПЕРАТОРА *БЕМШ*

Обозначения [17].

А — адресная часть команды.

*А*2 — адресная часть следующей команды.

i — номер индексного регистра.

Иi — содержимое индексного регистра *i*.

*И*15 — содержимое индексного регистра 15.

$И_В$ — содержимое индексного регистра, номер которого указан в *В*.

С — сумматор.

РМР — регистр младших разрядов.

РК — регистр режима работы арифметического устройства, если *РК* = 1, то режим логической операции *РК* = 2 — типа умножения, *РК* = 3 — типа сложения.

В — рабочая величина, используемая для описания семантики команд.

НК — адрес текущей команды (регистр номера команды).

ПБО — признак блокировки округления, если *ПБО* = 16, то округление не производится, *ПБО* = 0 — производится.

ПБН — признак блокировки нормализации, если *ПБН* = 1, то нормализация не производится, *ПБН* = 0 — производится.

ПБА — признак блокировки аварийного останова (авост) при переполнении порядков или делении на нуль, если *ПБА* = 1, то авост выполняется, *ПБА* = 0 — не выполняется.

С[*К1* : *К2*] — разряды с К1 по К2 сумматора.

С[*К*] — *К*-й разряд сумматора.

В[*К1* : *К2*] — разряды с К1 по К2 рабочей величины *В*.

В[*К*] — *К*-й разряд рабочей величины *В*.

П[*А*] — ячейка памяти ОЗУ с адресом *А* (*А* может быть выражением в алголовском смысле).

П[*А*, *К1* : *К2*] — разряды с *К1* по *К2* ячейки памяти ОЗУ с адресом *А*.

П[*А*, *К*] — *К*-й разряд ячейки памяти ОЗУ с адресом *А*.

С * *РМР* [*1* : *40*] — конкатенация сумматора и *РМР* [*1* : *40*], т. е. код *РМР* с 1 по 40 разряды рассматривается как продолжение кода сумматора.

⊕ — сложение по модулю 2.

Ниже описываются две процедуры *норм* (нормализации числа, находящегося на сумматоре) и *нормд* (нормализации после деления), которые используются при описании команд.

```
процедура норм; начало целый М; М: = 0;
   если ПБН = 1 то на Округление;
   для м: = м пока С[40] = С[41] цикл
        начало С[2 : 40]: = С[1 : 39];
        если РМР[40] = 1 то м: = 1;
        С[1]: = РМР[40]; РМР[2 : 40]: = РМР[1 : 39];
        РМР[1]: = 0; С[42 : 48]: = С[42 : 48] — 1;
        если С[42 : 48] = — 64 то начало С[1 : 48]: = 0;
        на выход конец
        конец;
   округление: если ПБО = 1 то на выход;
               если м = 1 то на выход;
               если РМР[1 : 40] = 0 то на выход;
               С[1]: = 1;
   выход: конец процедуры норм;
процедура нормд; начало если ПБН = 1 то на выход;
   для ПБН: = ПБН пока С[40] = С[41] цикл
        начало С[2 : 20]: = С[1 : 39]; С[1]: = 0;
        С[42 : 48]: = С[42 : 48] — 1;
        если С[42 : 48] = —64 то С: = 0; на выход;
        конец;
выход: конец процедуры нормд;
```

Таблица П13.1

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
000	зп (ЗГ)	АТХ	Запись в ОЗУ: **если** *i=15 & A=0* **то начало** *П[И15] : =C;* *И15 : =И15+1* **конец** **иначе** *П[A+Ии] : =C;*
001	зпм (ЗМ)	STX	Запись в ОЗУ и магазинное считывание: *П[A+Ии] : = C; И15 : = И15—1;* *C : = П[И15]; РК : =1;*
003	счп (СМ)	XTS	Считывание па сумматор и магазинная запись: *П[И15] : =C; И15:=И15+1;* *C : =П[A+Ии]; РК : =1;*
004	сл (АС)	A+X	Арифметическое сложение: **если** *i=15 & A=0* **то начало** *И15 : =И15—1;* *C * РМР[1 : 40] : =C+П[И15]* **конец** **иначе** *C * РМР[1 : 40] : =C+П[A+Ии];* *РК : =3; норм;*
005	вч (АВ)	A—X	Арифметическое вычитание: **если** *i=15 & A=0* **то начало** *И15 : =И15—1;* *C * РМР[1 : 40] : =C—П[И15];* **конец** **иначе** *C * РМР[1 : 40]: =C—П[A+Ии];* *РК : =3; норм;*
006	вчоб ОВ	X—A	Обратное вычитание: **если** *i=15 & A=0* **то начало** *И15 : =И15—1;* *C * РМР[1 : 40]: =П[И15]—C;* **конец** **иначе** *C * РМР[1 : 40] : =П[A+Ии]—C;* *РК : =3; норм;*

Таблица П13.1 (продолжение)

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
007	вчаб МВ	АМХ	Вычитание модулей: **если** $i=15$ & $A=0$ **то начало** $И15 := И15-1$; $C * РМР[1:40] :=$ $= abs(C)-abs(П[И15])$ **конец** **иначе** $C * РМР[1:40] :=$ $= abs(C)-abs(П[A+Иi])$; $РК := 3$; *норм*;
010	сч СЧ	ХТА	Считывание из ОЗУ на сумматор: **если** $i=15$ & $A=0$ **то начало** $И15 := И15-1$; $C := П[И15]$ **конец** **иначе** $C := П[A+Иi]$; $РК := 1$;
011	и ЛУ	ААХ	Логическое умножение: $РМР := 0$; **если** $i=15$ & $A=0$ **то** **начало** $И15 := И15-1$; $C := C$ & $П[И15]$ **конец** **иначе** $C := C$ & $П[A+Иi]$; $РК := 1$;
012	нтж СР	АЕХ	Сравнение по модулю 2: $РМР := C$; **если** $i=15$ & $A=0$ **то** **начало** $И15 := И15-1$; $C := C \oplus П[И15]$ **конец** **иначе** $C := C+П[A+Иi]$; $РК := 1$;
013	слц ЦС	ARX	Циклическое сложение: $РМР := 0$; **если** $i=15$ & $A=0$ **то** **начало** $И15 := И15-1$; $C[1:48] :=$ $= C[1:48] + П[И15]$ **конец** **иначе** $C[1:48] :=$ $= C[1:48]+П[A+Иi]$; $РК := 1$;

Таблица П13.1 (продолжение)

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
014	знак И З	AVX	Изменение знака: *РМР* : =0; **если** *i=15* & *A*=0 **то начало** *И15* : =*И15—1*; *C* : =**если** *П*[*И15*, *41*]=0 **то** *C* **иначе** —*C* **конец иначе** *C* : = **если** *П*[*A*+*Иi*, *41*]=0 **то** *C* **иначе** —*C*; *РК* : =*3*; *норм*;
015	или ЛС	AOX	Логическое сложение: *РМР* : =0; **если** *i=15* & *A*=0 **то начало** *И15* : =*И15—1*; *C* : =*C*∨*П*[*И15*] **конец иначе** *C* : =*C*∨*П*[*A*+*Иi*]; *РК* : =*1*;
016	дел А Д	A/X	Арифметическое деление : **если** *i*=15 & *A*=0 **то начало** *И15* : =*И15—1*; *C* : =*C*/*П*[*И15*] **конец** **иначе** *C* : =*C*/*П*[*A*+*Иi*]; *РК* : =*2*; *нормд*; **примечание** *РМР* не определен;
017	умн А У	A*X	Арифметическое умножение: **если** *i=15* & *A*=0 **то начало** *И15* : =*И15* —*1*; *C* * *РМР*[*1* : *40*] : =*C*×*П*[*И15*] **конец** **иначе** *C* *РМР*[*1* : *40*] : =*C*×*П*[*A*+*Иi*]; *РК* : =*2*; *норм*;
020	сбр С Б	APX	Сборка по маске: **начало процедура** *сборка* (*x*); **начало целый** *м*, *к*; *м* : =*48*; **для** *к*: =*48* **шаг** —*1* **до** *1* **цикл** **если** *П*[*x*, *к*]=*1* **то начало** *C*[*м*] : =*C*[*к*]; *м* : =*м—1* **конец**; **если** *м*≠*0* **то** *C*[1 : *м*] : =0 **конец проц** *сборка*; **если** *i=15* & *A*=0 **то начало** *И15* : =*И15—1*; *сборка* (*И15*) **конец иначе** *сборка* (*A*+*Иi*); *РК* : =*1* **конец**;

Таблица П13.1 (продолжение)

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
021	рзб РБ	AUX	Разборка по маске: **начало процедура** *разборка* (*x*); **начало целый** *м*, *к*; *м* : =*48*; **для** *к* : =*48* **шаг** —*1* **до** *1* **цикл** **если** *П*[*x*, *к*]=*1* **то начало** *С*[*к*] : =*РМР*[*м*]; *м* : =*м*—*1* **конец**; **конец** процедуры *разборка*; *РМР* : =*С*; *С* : =0; **если** *i*=15 & *А*=0 **то начало** *И15* : =*И15*—*1*; разборка (*И15*) **конец иначе** *разборка* (*А*+*Иi*); *РК* : =*1*; *РМР* : =0;
022	чед ВС	ACX	Выдача числа единиц в коде: **Начало процедура** *чед* (*x*); **начало целый** *к*; **для** *к* : =*48* **шаг** —*1* **до** *1* **цикл** **если** *РМР* [*к*]=*1* **то** *С*[*1* : *48*] : =*С*[*1* : *48*]+*1*; *С*[*1* : *48*] : =*С*[*1* : *48*]+*П*[*x*, *1* : 48] **конец** процедуры *чед*; *РМР* : =*С*; *С* : =0; **если** *i*=15 & *А*=0; **то начало** *И15* : =*И15* —*1*; *чед* (*И15*) **конец иначе** *чед* (*А*+*Иi*); *РМР* : =0; *РК* : =*1*;
023	ьед ВИ	ANX	Вычисление номера старшей единицы: **начало процедура** *номед* (*x*); **начало целый** *к*; **для** *к* : =*48* **шаг** —*1* **до** *1* **цикл** **начало** *С*[*1* : =*48*] := *С*[*1* : *48*] +*1*; **если** *РМР*[*48*]=*1* **то начало** *РМР*[*2* : *48*] : =*РМР*[*1* : *47*]; *РМР*[*1*] : =0; **на выход** **конец**; *РМР*[*2* : *48*] : =*РМР*[*1* : *47*]; *РМР*[*1*] : =0 **конец**; выход : *С*[*1* : *48*] : =*С*[*1* : *48*]+ +*П*[*x*, *1* : *48*] **конец** процедуры *номед*; *РМР* : =*С*; *С* : =0; **если** *i*=15 & *А*=0 **то начало** *И15* : =*И15*—*1*; *номед* (*И15*) **конец иначе** *номед* (*А*+*Иi*); *РК* : =*1*; **примечание** на сумматоре двоичное число в младших разрядах: **конец**;

Таблица П13.1 (продолжение)

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
024	слп СП	Е+Х	Сложение порядков: *РМР* : =0; **если** *i=15* & *А=0* **то начало** *И15* : =*И15—1*; *С[48 : 42]* : =*С[48 : 42]*+*П[И15, 48 : 42]* **конец иначе** *С [48 : 42]* : = = *С[48 : 42]*+ + *П[А+Иi, 48 : 42]*; *норм*; *РК* : =*2*;
025	вчп ВП	Е—Х	Вычитание порядков: *РМР* : =0; **если** *i=15* & *А=0* **то начало** *И15*: = = *И15—1*; *С[48 : 42]* : =*С[48 :42]* —*П[И15, 48 : 42]* **конец иначе** *С[48 : 42]* : = =*С[48 : 42]*— —*П[А+Иi, 48 : 42]*; *норм*; *РК* : =*2*;
026	сд СК	ASX	Сдвиг по коду: **начало процедура** *сдвиг* (*х*); **начало целый** *к*; **если** *х* ⩾ 0 **то начало** **для** *к* : =*х* **шаг** —*1* **до** *1* **цикл начало** *РМР[1 : 47]* : =*РМР[2 : 48]*; *РМР[48]* : =*С[1]*; *С[1 : 47]* : =*С[2 : 48]*; *С[48]* : =0 **конец**; **конец иначе** **для** *к* : =*х* **шаг 1 до** —*1* **цикл начало** *РМР[2 : 48]* : =*РМР[1 : 47]*; *РМР[1]* : =*С[48]*; *С[2 : 48]* : =*С[1 : 47]*; *С[1]*: =0 **конец** **конец процедуры** *сдвиг*; *РМР* : =0; **если** *i=15* & *А=0* **то начало** *И15* : =*И15—1*; *сдвиг* (*П[И15, 48 : 42]*) **конец иначе** *сдвиг* (*П[А+Пi, 48 : 42]*); *РК* : =*1*; **конец**;

Таблица П13.1 (продолжение)

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
027	рж РК	XTR	Установка по коду числа режима выполнения АУ: *ПБН* : =*П*[*А*+*Иi*, *42*]; *ПБО* : = *П*[*А*+*Иi*, *43*]; *ПБА* : =*П*[*А*+*Иi*, *47*]; **если** *П*[*А*+*Иi*, *46*]=*1* **то** *РК* : =*3* **иначе если** *П*[*А*+*Иi*, *45*]=*1* **то** *РК* : =*2* **иначе если** *П*[*А*+*Иi*, *44*]=*1* **то** *РК* : =*1*;
030	счрж ВР	PTE	Выдача содержимого регистра признаков режима АУ: *В* : =*А*+*Иi*; *С* : =0; *С*[*42*]:=*ПБП*; *С*[*43*] : =*ПБО*; *С*[*47*] : =*ПБА*; *С*[*РК*+*43*] : =1; *С*[*42* : *47*] : =*С*[*42* : *47*] & *В*[*1* : *6*];
031	счмр МР	УТА	Выдача младших разрядов: *В* : =*А*+*Иi*; **если** *РК*=*1*, **то** *С*[*1* : *48*] : =*РМР*[*1*:*48*] **иначе начало** *С*[*1* : *41*] : =0; *С*[*42* : *48*] : =*С*[*42* : *48*] + *В*[*1* : *7*]; *С*[*1* : *40*] : =*РМР*[*1* : *40*]; *норм*; **конец**;
034	слпа КС	E+N	Корректировка порядка сложением: *В* : = *А* + *Иi*; *РМР* : = 0; *С*[*42* : *48*] : = *С*[*42* : *48*] + *В*[*1* : *7*]; *норм*; *РК* : = *2*;
035	вчпа КВ	E—N	Корректировка порядка вычитанием: *В* : =*А*+*Иi*; *РМР* : =0; *С*[*42* : *48*] : =*С*[*42* : *48*]—*В*[*1* : *7*]; *норм*; *РК* : =*2*;
036	сда СД	ASN	Сдвиг по адресу: *РМР* : =0; *В* : =*А*+*Иi*; *сдвиг* (*В*[*1* : *7*]); *РК* : =*1*; **примечание** процедура *сдвиг* описана в команде 026;

Таблица П13.1 (продолжение)

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
037	ржа РА	NTR	Установка по коду адреса режима выполнения команд АУ: $B := A+Ии$; $ПБН := B[1]$; $ПБО := B[2]$; $ПБА := B[6]$; **если** $B[5]=1$ **то** $РК := 3$ **иначе** **если** $B[4]=1$ **то** $РК := 2$ **иначе** **если** $B[3]=1$ **то** $РК := 1$ **иначе** $РК := 0$;
040	уи УИ	ATI	Установка кода на индексный регистр: $B := A+Ии$; $B := B[1:4]$; $И_в := C[1:15]$;
041	уим УМ	STI	Установка кода на индексный регистр и магазинное считывание: $B := A+Ии$; $B := B[1:4]$; $И15 := И15-1$; $И_в := C[1:15]$; $C := П[И15]$; $РК := 1$;
042	счи ВИ	ITA	Считывание индексного регистра: $B := A+Ии$; $B := B[1:4]$; $C[16:48] := 0$; $C[1:15] := И_в$; $РК := 1$;
043	счим ВМ	ITS	Считывание индексного регистра и магазинная запись: $B := A+Ии$; $B := B[1:4]$; $П[И15] := C$; $И15 := И15+1$; $C[16:48] := 0$; $C[1:15] := И_в$; $РК := 1$;
044	уии ПИ	MTI	Передача кода из индексного регистра в индексный регистр: $B := A[1:4]$; $И_в := Ии$;
045	сли СИ	J+M	Сложение индексных регистров: $B := A[1:4]$; $И_в := И_в + Ии$;
220	мода ИА	UTC	Изменение команды адресом: **примечание** к коду адресной части следующей команды прибавляет 15 младших разрядов исполнительного адреса текущей команды; $B := A+Ии$; $A2 := A2+B[1:15]$;

Таблица П13.1 (продолжение)

КОП	Мнемоника *бемш*	Мнемоника *младен*	Название команды и семантика выполнения
230	мод ИК	WTC	Изменение команды кодом: **примечание** к коду адресной части следующей команды прибавляются 15 младших разрядов содержимого ячейки ОЗУ по исполнительному адресу текущей команды; *B* : =*П*[*A*+*Ии*, *1* : *15*]; *A2* : =*A2*+*B*;
240	уиа ПА	VTM	Передача адреса в индексный регистр: *Ии* : =*A*;
250	слиа ИР	UTM	Сложение индексного регистра с адресом: *Ии* : =*Ии*+*A*;
260	по УО	UZA	Условный переход по нулевому коду в сумматоре: *РМР* : =*C*; **если** *РК=0* **то на вых**; **если** *РК=1* **то начало если** *C*[*1* : *48*]=0 **то на** *П*[*A*+*Ии*] **конец иначе** **если** *РК=2* **то начало если** *C*[*48*]=*1* **то на** *П*[*A*+*Ии*] **конец иначе** **если** *РК=3* **то начало если** *C*[*41*]=0 **то на** *П*[*A*+*Ии*] **конец; вых**;
270	пе У1	UIA	Условный переход по ненулевому коду в сумматоре; *РМР* : =*C*; **если** *РК=0* **то на** *П*[*A*+*Ии*] **если** *РК=1* **то начало если** *C*[*1* : *48*]≠0 **то на** *П*[*A*+*Ии*] **конец иначе** **если** *РК=2* **то начало если** *C*[*48*]=0 **то на** *П*[*A*+*Ии*] **конец иначе** **если** *РК=3* **то начало если** *C*[*41*]=*1* **то на** *П*[*A*+*Ии*] **конец;**
300	пб ПБ	VIM	Безусловный переход: **на** *П*[*A*+*Ии*];

Таблица П13.1 (окончание)

КОП	Мнемоника *бемш*	Мнемоника *мадлен*	Название команды и семантика выполнения
310	пв ПВ	VIM	Безусловный переход с запоминанием адреса возврата: *Ии* : =*НК*+*1*; на *В*[*А*];
340	пио ИО	VZM	Условный переход по нулевому коду в индексном регистре: если *Ии*=*0* то на *П*[*А*];
350	пино И1	V1M	Условный переход по ненулевому коду в индексном регистре: если *Ии*≠*0* то на *П*[*А*];
370	цикл КУ	VLM	Конец цикла: если *Ии*≠*0* то начало *Ии* : =*Ии*+*1*; на *П*[*А*] конец;
—	конк	ZOO	Короткая константа
—	конд	LOG	Длинная константа

Примечание. В операторе *бемш* допускается использование как цифрового обозначения кодов операций, так и в мнемонике *бемш* (колонки 1 и 2 таблицы). Короткие команды оператора *бемш*, у которых в адресную часть входят скалярные переменные, идентификаторы массивов или метки, базируются, т. е., если *КОП* — код операции, *А* — адресная часть команды, *КО* — адрес базы (адрес нулевой константы см. гл. 8), база — номер индексного регистра, взятого для базирования, то базирование выглядит как следующее преобразование команды:

КОП, *А*; *в КОП*, *А* — *КО*, *база*;
КОП, *А*, *i*; в *мода*, *А*; *КОП*, 0, *i*;

Допускается использование в качестве кодов операций любых других (не перечисленных в списке) цифровых кодов операций в соответствии с системой команд БЭСМ-6 (экстракоды и др.).

ПРИЛОЖЕНИЕ 14

ОГРАНИЧЕНИЯ НА АЛЬФА-ПРОГРАММУ

А. Ограничения по отношению к языку альфа-6.

1) Не допускаются рекурсивные процедуры.

2) Побочный эффект в процедурах-функциях игнорируется.

3) Должны совпадать вид, тип (с точностью до целого и вещественного) и размерность идентификаторов и (или) выражений, выступающих в роли фактических параметров, соответствующих одному и тому же формальному параметру процедуры, идентификатор которой сам является формальным параметром.

Пример. Запрещена конструкция:

```
начало вещественный c;
  процедура P(x) начало вещественный a; массив M[1:5];
            a:=2; M[]:=0;
            x(a); x(M);
      конец;
  процедура Ф(y); начало c:=y×y конец;
  P(Ф);
конец
```

Здесь размерности фактического параметра a и фактического параметра M формальной процедуры x не совпадают.

4) Фактический параметр, подставляемый именем и являющийся вычисляющим выражением, таким, что хотя бы одна переменная, графически в него входящая, перевычисляется при выполнении оператора процедуры, не должен содержать указатель функции.

Пример. Запрещена конструкция:

```
начало вещественный а, в;
   вещественный функция ф = 5;
   процедура P(x); начало a:=в:=1 конец
   P(a+ф); P(в+ф);
конец
```

5) Фактическими параметрами оператора процедуры (указателя функции)
не могут быть:

а) целые без знака, используемые в качестве метки,

б) идентификаторы стандартных функций и стандартных процедур,

в) элементы упакованного логического массива.

6) Фактическими параметрами оператора внешней процедуры (указателя функции) (т. е. специфицированной **альфа**, **фортран**, **алгол**) не могут быть:

а) идентификаторы упакованных логических массивов и их элементы,

б) идентификаторы переключателей,

в) переменная с пустыми позициями индексов левее непустых, например, $A[, 1]$.

7) В теле некоторого цикла с параметром не может ни статически, ни динамически содержаться оператор цикла с тем же параметром.

Пример 1. Запрещена конструкция:

начало целый k; **вещественный** a; $a := 1$;
  для $k := 1, \ldots, 10$ **цикл для** $k := k + 1$ **пока** $k < 3$
  цикл $a := a \uparrow k$;
конец

Пример 2. Запрещена конструкция:

начало целый i, k; **логический** $Л$;
  процедура P; **начало** $Л := $ **истина**;
    для $i := 1, \ldots, k$ **цикл** $a[i] := a[i + 1]$;
    конец;
  $k := 10$; $Л :=$ **ложь**;
  для $i := k$ **пока** $Л$ **цикл** P;
конец

8) Левая и правая части перечисления не должны совпадать графически.

9) Первая метка составной метки должна быть описана в том блоке, в котором используется эта составная метка.

Пример. Запрещена конструкция:

начало *M1*: **начало** *M2*: **начало на** *M1. M2. M3*;
M3: **конец конец конец**

где M1 и M2 — метки, метящие блоки.

10) Результат действия переключателя неопределен при неопределенном значении указателя переключателя.

11) В отличие от алгола-60 осуществляется иной порядок вычисления индексных выражений при выполнении оператора присваивания со списком левой части, состоящим более чем из одной переменной. Если написано: $i := j := 1$; $B[i, j] := i := j := 2$; то присваивание произойдет не $B[1, 1]$, а $B[2, 2]$.

12) Описание переключателя или функции (процедуры-функции) должно предшествовать их использованию. Исклю-

чение составляет случай вхождения указателя функции или указателя переключателя в тело процедуры (функции, процедуры-функции), входящей в тот же блок, что и описание соответствующей переменной.

Пример. Запрещена конструкция:

начало вещественный a; $a = p + 1$;
функция $p = 5$;
. . .
конец

13) Запрещено употребление метки вне тела цикла, в котором она метит оператор.

Пример 1. Запрещена конструкция:

начало
5 **раз цикл начало** *M*: ... **конец**;
на *M*; ...
конец

Пример 2. Запрещена конструкция:

начало переключатель *S*: = *M1*, *M2*; **цел** *K*;
M2: **для** *K*: = 1, ..., *5* **цикл начало** *M1*: ... **на** *S*[*I*] **конец**
. . .
конец

14) Если фактический параметр (ФКП) оператора внешней подпрограммы есть идентификатор процедуры, то этот идентификатор может быть только идентификатором внешней подпрограммы либо формальным параметром подпрограммы. Кроме того, запрещается использовать в качестве *ФКП* внешней подпрограммы:

— идентификатор логического упакованного массива или его элемент;

— идентификатор переключателя, являющегося формальным параметром какой-либо процедуры или функции;

— переменную с непустыми индексами не нулевой абсолютной размерности ($A[, i]$, $A[i,]$ и т. д.).

15) Идентификатор внешней процедуры (функции) запрещается использовать в качестве *ФКП*, описанной в подпрограмме процедуры или функции.

16) При подстановке *ФКП* на место формальных параметров внешней процедуры (функции) языком не учитывается возможное перевычисление компонент *ФКП* в теле указанной

процедуры (функции). Например, если $i = 1$ в момент обращения к процедуре P оператором $P(i, A[i])$ и тело P выглядит следующим образом:

процедура $P(x, y)$; **целый** x; **вещественный** y;
начало $x := 0$; $y := 2$ **конец**,

то после выполнения оператора $P(i, A[i])$ перевычислится элемент не $A[0]$, а $A[1]$.

Б. Количественные ограничения на альфа-программу.

17) Число различных идентификаторов (включая целые числа, используемые в качестве меток) не должно превышать 1024.

18) Общее число символов в различных идентификаторах, состоящих более чем из 3 символов, должно быть меньше 3000.

19) Максимальная длина идентификатора или целого без знака в качестве метки равна 96 символам.

20) Число символов в строке не должно превышать 512.

21) Общее число символов во всех строках с учетом кавычек, не считая строк — стандартных форматов вывода, должно быть меньше 4000.

22) Число различных констант с учетом констант, вводимых транслятором, должно быть меньше 2048.

23) Число блоков PART должно быть не больше 128.

24) Специальных объектов (метка, метящая блок; переменная с внутренней размерностью; переменная с верхним индексом; собственная переменная; переменная, определяемая посредством описания тождества) может быть описано не более 220.

25) Число символов в описании верхнего индекса должно быть меньше 100.

26) Одно описание типа и структуры или описание массива должно содержать не более 64 переменных.

27) В одном описании типа и структуры либо описании массива должно содержаться не более 7 переменных с верхним индексом.

28) Число внешних массивов в одном описании массива не должно превышать 15.

29) Абсолютная размерность переменных должна быть меньше 15.

30) Число массивов в одном сегменте описания массива должно быть меньше 36.

31*) Общее число массивов не должно превышать 256.

32)*) Суммарное число измерений для всех масивов не должно превышать 1024.

33) В списке граничных пар описания массивов должно быть меньше 250 символов.

34) Длина массива (не внешнего) должна быть не больше 32000.

35) Максимальная вложенность блоков и циклов равна 63.

36) Общее число операторов *бегш* и ограничителей: **начало**, **процедура**, **функция**, **цикл** **) должно быть меньше 1000.

37) Число символов между двумя ближайшими точками с запятой должно быть меньше 500.

38) Число описаний меток между двумя ближайшими точками с запятой не должно превышать 16.

39) Число перечислений, входящих в состав перечисления, должно быть меньше 16.

40) Общее количество описанных процедур и библиотечных процедур не должно превышать 255.

41) Общее число формальных параметров процедур (включая библиотечные процедуры) не должно превышать 512.

42) Число формальных параметров одной процедуры должно быть меньше 36.

43) Суммарное число обращений к процедурам и функциям с учетом расклейки тел процедур (функций) и циклов, программируемых как процедура не должно превышать 1024.

44) Число циклов должно быть меньше 512 ***).

45) Число параллельных индексов в перечислении должно быть меньше 8, где под параллельными индексами понимаются одинаковые индексные выражения, по которым ведется перечисление. Например, в перечислении

$$a[i+1] \times в[i+1], \ldots, a[j] \times в[j]$$

индексы у массивов a и $в$ будут параллельными. В перечислении

$$a[i, 5, i], \ldots, a[i+4, 5, i+4]$$

два параллельных индекса.

---

*) Здесь под массивом понимается как описанная переменная ненулевой размерности, так и величина ненулевой размерности, заводимая транслятором.

**) Если тело цикла заключено в скобки

то ограничитель **цикл** заголовка данного цикла не учитывается в числе данных ограничителей.

***) Имеются в виду также и циклы, возникшие при программировании многомерных операций.

46) Глубина вложенности индексных выражений в индексные выражения не может превышать 5.

47) Значение индексного выражения или параметра регулярного цикла*) типа целый не должно превышать по абсолютной величине 32768. Исключение составляют индексные выражения переменных, являющихся внешними массивами. Это ограничение не относится к индексному выражению в элементе внешнего массива.

48) Длина арифметического или логического выражения должна быть меньше 512 символов.

49) Число аргументов оператора *input* или *output* должно быть меньше 38.

50) Глубина вложенности синтаксических конструкций должна быть меньше 100.

51) Любое значение, задаваемое константой или вырабатываемое арифметическим выражением при выполнении программы, не может превышать по абсолютной величине числа $.922337203695_{10} + 19$.

ПРИЛОЖЕНИЕ 15

ОГРАНИЧЕНИЯ НА ПАРАМЕТРЫ СИСТЕМНОГО ПАКЕТА И ПРЕДСТАВЛЕНИЕ ДАННЫХ

А. Количественные ограничения на системный пакет.

1) Число фрагментов в последовательности заданий Альфа-6 плюс число заданий должно быть меньше 120.

2) Число символов в последовательности заданий должно быть меньше 98200 (1230 перфокарт из расчета 80 символов на одной перфокарте).

Б. Количественные ограничения на задание.

3) Максимальное число перфокарт программы или данных в задании Альфа-6—600 (из расчета 80 символов на одной перфокарте).

4) Число перфокарт в программе плюс n должно быть меньше 4096, где n — число простых операторов (не содержащих в себе других операторов), в тексте которых находится более одного номера перфокарты.

---

*) К регулярным циклам относятся циклы с пересчетом типа «шаг» и шагом типа целый, который не пересчитывается в теле цикла.

5) В один архив можно записать не более 509 программ и данных.

6) Суммарное число символов во фрагменте замен и в фактических параметрах команд редактирования не должно превышать 24500.

7) Число символов в фактических параметрах одной системной команды не должно превышать 128.

8) Число системных команд редактирования (* ЗАМЕНИТЬ, * ВСТАВИТЬ, * ЗАМЕНИТЬ – ТЕКСТ и т. д.) не должно превышать 128.

9) Число системных команд * АРХИВ не должно превышать 100.

10) Число системных команд отладки (* ЗНАЧЕНИЕ, * ПЕЧАТАТЬ – МЕТКИ и т. д.) не должно превышать 128.

11) Число системных команд * ПЕЧАТАТЬ – МОДУЛЬ не должно превышать 64.

12) Число библиотечных описаний в сумме с числом команд * ПРОЦЕДУРА должно быть меньше 250.

13) Суммарное число идентификаторов, печатаемых с помощью системных команд * ПЕЧАТАТЬ – МЕТКИ и * ЗНАЧЕНИЕ не должно превышать 512.

14) Число различных идентификаторов, входящих в программу между ближайшими точками с запятой и распечатываемых по системным командам * ПЕЧАТАТЬ – МЕТКИ и * ЗНАЧЕНИЕ, не должно превышать 30.

15) Заменяемый текст в исходной программе может разбиваться не более чем десятью номерами перфокарт.

В. Ограничения на представление данных.

16) Общее число символов в данных (в архиве) не должно превышать 180000.

17) Значение числа в данных не должно превышать по абсолютной величине $.922337203695_{10} + 19$.

18) Количество значащих цифр мантиссы числа может быть не более 12.

ПРИЛОЖЕНИЕ 16

ТРАНСЛЯЦИЯ АЛГИБР-ПРОГРАММ

Система Альфа-6 имеет режим, позволяющий использовать программу или данные, приготовленные для системы Алгибр. При этом алгибр-программа должна быть отперфорирована на КУ-3, а алгибр-данные — на числовом перфораторе для М-220. Допускаются комбинации: алгибр-программы с альфа-данными и альфа-программы с алгибр-данными.

Ниже приводится пример системного пакета, инициирующего трансляцию алгибр-программы и выполнения программы с данными, приготовленными для системы Алгибр (см. гл. 3). В программе есть обращение к двум библиотечным процедурам: *apec* и *lint*.

Для выполнения программы требуется 6 мин. центрального процессора БЭСМ-6.

```
ШИФР—777777 — ЗС +⁻
ВРЕМЯ — 600⁻
ЕЕВ1А3
* NAME — АЛГИБР
* ASSIGN — LIBRARY — N
* MAIN — * ALFA/6 *
* EXECUTE
* BINARY
◇ ◇ ◇ СИСТЕМА
* ПРОЦЕДУРА (APEC, LINT)
◇◇◇ ПРОГРАММА
◇◇◇ АЛГИБР
      ⟨пакет перфокарт с алгибр-программой (без
      контрольных сумм, без шифра, кончается звез-
      дочкой)⟩
◇◇◇ ЗАДАНИЕ
◇◇◇ ДАННЫЕ
◇◇◇ АЛГИБР
      (пакет перфокарт с данными для М-20 (с конт-
      рольными суммами, без разделителей, кончается
      контрольной суммой))
◇◇◇ КНЦ
* END — BINARY
* EXECUTE
* END — FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

При вводе алгибр-программы система Альфа-6 преобразует ее в альфа-программу, осуществляя следующие замены:

ввод (...) на *input* (0, ...)
вывод (...) на *output* (0, 'e', ...)
КОД (⟨коп⟩, ⟨А⟩) на *беми* (⟨коп⟩, ⟨А⟩)
КОД (⟨ир⟩, ⟨коп⟩, ⟨А⟩) на *беми* (⟨коп⟩, ⟨А⟩, ⟨*ир*⟩)
КОД (..., ..., ..., ...) на *беми* (..., ..., ..., ...)

— перед каждой физической перфокартой программы ставится порядковый номер перфокарты (◇ N◇);

— * в конце программы заменяется на ◇ *;

— перед программой ставится имя алгибр-программы: «алгибр:».

При вводе алгибр-данные переводятся из кодировки числового перфоратора М-220 в алголовский вид. Причем контрольная сумма заменяется символом ; и после каждого числа, за которым не следует контрольная сумма, ставится символ ,. В конце алгибр-данных вставляется ◇ *. Перед данными ставится имя алгибр-данных алгибр:.

Следует обратить внимание на различие отождествлений основных символов, осуществляемых в системах Алгибр и Альфа-6. Отождествления в системе Алгибр, отсутствующие в Альфа-6: (строчные буквы)

$$д_{р} = d_{лат}$$
$$и_{р} = u_{лат}$$
$$п_{р} = n_{лат}$$
$$т_{р} = m_{лат}$$
$$ч_{р} = r_{лат}$$

Отождествления в системе Альфа-6, отсутствующие в Алгибре: (строчные буквы)

$$т_{лат} = т_{р}$$
$$м_{лат} = м_{р}$$
$$h_{лат} = н_{р}$$

Примечание. В индексах: р — русская буква, лат — латинская буква.

ПРИЛОЖЕНИЕ 17

ТРАНСЛЯЦИЯ АЛГОЛ-ПРОГРАММ

Система Альфа-6 позволяет транслировать и считать задачи, подготовленные для системы БЭСМ-алгол. Для этого в программе необходимо осуществить ряд изменений (систем-

ными командами редактирования), которые приводят ее к альфа-программе.

Ниже приводится пример системного пакета, содержащий такой список команд редактирования, и осуществляющий трансляцию и выполнение программы (см. гл. 3).

```
ШИФР — 777 777 — ЗС+-
ЕЕВ1А3
* NAME— АЛГОЛ
* ASSIGNE— LIBRARY — N
* CALL — ALPHA/6
◇◇◇ СИСТЕМА
* ЗАМЕНИТЬ — ТЕКСТ ('input (' , ' input(0, ')
* ЗАМЕНИТЬ— ТЕКСТ ('output (' , 'output (0, ')
* ЗАМЕНИТЬ— ТЕКСТ ('marg (' , 'marg (0, ')
* ЗАМЕНИТЬ— ТЕКСТ ('read (' , 'copy (1, ')
* ЗАМЕНИТЬ— ТЕКСТ ('write (' , 'copy (0, ')
◇◇◇ ПРОГРАММА
      (пакет перфокарт с программой)
◇◇◇ ЗАДАНИЕ
◇◇◇ ДАННЫЕ
      (пакет перфокарт с данными)
◇◇◇ КНЦ
* EXECUTE
* END — FILE
⟨диспетчерский конец⟩
ЕКОНЕЦ
```

ТЕРМИНОЛОГИЧЕСКИЙ СЛОВАРЬ

АБСОЛЮТНАЯ РАЗМЕРНОСТЬ ПЕРЕМЕННОЙ. В общем случае переменная может быть описана как массив (такое описание задает внешнюю размерность переменной), элементы которого, в свою очередь, тоже являются массивами (обладают внутренней размерностью). Абсолютная размерность переменной является суммой внешней и внутренней размерностей. Скаляр является переменной нулевой абсолютной размерности, а верхний индекс увеличивает абсолютную размерность на единицу.

АВОСТ. Диспетчерское прерывание, т. е. прекращение выполнения программы ввиду нарушения ограничений, установленных ЭВМ БЭСМ-6 или операционной системой. Печатается сообщение о причине прерывания и сбойная информация.

АДРЕС ПРЕРЫВАНИЯ В ПРОГРАММЕ. Адрес в загруженной программе, где произошло прерывание.

АЛГИБР-ПРОГРАММА. Программа, которая написана на входном языке альфа, и подготовлена к трансляции для системы Алгибр.

АЛГОЛ-ПРОГРАММА. Программа, которая написана на входном языке для транслятора БЭСМ-алгол [7].

АЛГОЛ-ПРОЦЕДУРА. Процедура, которая написана на входном языке алгол-ГДР [12]. Термин употребляется в связи с обращением к ней из альфа-программы.

АЛЬФА-ДАННЫЕ. Данные, предназначенные для ввода оператором *input* при выполнении альфа-программы.

АЛЬФА-ЗАДАЧА. Последовательность заданий для системы Альфа-6, начинающаяся картой вызова системы * CALL — ALPHA/6 и заканчивающаяся картой конца заданий ◇◇◇ КНЦ (рис. 13.1).

АЛЬФА-ПРОГРАММА (ПОДПРОГРАММА). Программа, которая написана на входном языке альфа-6. Если при комплексации программ к ней есть обращение из другой программы, то говорят, что она является альфа-подпрограммой. В этом случае она имеет вид описания процедуры или описания процедуры-функции.

АРХИВ. Совокупность программ и данных, определенным образом структурированная. Различают архивы с оглавлением и примитивные архивы. Носителем архива является магнитная лента (диск).

АРХИВ С ОГЛАВЛЕНИЕМ. Архив, содержащий произвольное число программ и данных, информация о которых хранится в оглавлении.

АРХИВ *LIBRA*. То же, что и библиотечный архив.

БИБЛИОТЕЧНАЯ (СТАНДАРТНАЯ) ПРОЦЕДУРА. Процедура, написанная на входном языке альфа-6 и хранящаяся в библиотечном архиве. При использовании библиотечной процедуры необходимо указать ее имя в системной команде. * ПРОЦЕДУРА (п. 12.3), см. «библиотечное описание».

БИБЛИОТЕЧНАЯ ЧАСТЬ ПОДПРОГРАММЫ. Совокупность библиотечных описаний, вставляемых в альфа-программу.

БИБЛИОТЕЧНОЕ ОПИСАНИЕ. Последовательность произвольных описаний в смысле языка альфа-6 (в частном случае, описание процедуры). Библиотечное описание хранится в архиве *LIBRA* (или личном архиве) и используется в программе по команде * ПРОЦЕДУРА.

БИБЛИОТЕЧНЫЙ АРХИВ. Архив, носящий стандартное имя *LIBRA*, находящийся на МЛ с математическим номером 31, служит для хранения библиотеки стандартных процедур.

БОБИНА. Магнитная лента (диск), предназначенная для хранения информации. Запись и чтение осуществляются по

номеру бобины, который указывается в паспорте задачи в разделе «лента» [9].

БЭСМ-6. Сокращенное название электронно-вычислительной машины: быстродействующая электронно-счетная машина серии 6.

ВВОД (ДАННЫХ). Присваивание переменным и массивам, объектам оператора *input*, значений очередных групп данных из альфа-данных.

ВИД ЗАДАНИЯ. Определяет набор работ, выполняемых в одном задании. Имеются следующие виды заданий: задание с программой, задание с данными, задание с архивами.

ВИД КОДИРОВКИ. Название перфорирующего устройства, см. «входные кодировки».

ВНЕШНИЕ ПОДПРОГРАММЫ. Отдельно транслируемые программы, обращение к которым осуществляется из альфа-программы. В ней имена этих внешних подпрограмм должны быть указаны в одном из описаний, задаваемых описателями **альфа, алгол или фортран** (п. 2.22).

ВНЕШНИЙ БЛОК. Блок в программе, который помечен меткой, начинающейся с букв PART. Такой блок при выполнении программы находится во временной библиотеке МС Дубна и вызывается в ОЗУ только в момент входа в него.

ВНЕШНИЙ МАССИВ. Массив в программе, идентификатор которого начинается с букв *EX*. При выполнении программы такой массив помещается системой на МБ, а обмен между элементами массива, МБ и ОЗУ осуществляется с помощью стандартной процедуры *copy*.

ВНЕШНЯЯ ПАМЯТЬ. Общее название для «медленной памяти» ЭВМ — барабанов, дисков, лент.

ВНУТРЕННЕЕ ПОЛЕ СТРАНИЦЫ. Вся выводимая операторами *output* информация располагается на листе АЦПУ в постраничном виде (п. 2.17). Прямоугольная область страницы, на которой размещается выводимая информация, называется

внутренним полем страницы. Эта область ограничена левым, правым, верхним и нижним полями страницы (п. 2.16).

ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ СИМВОЛОВ В СТРОКАХ. Строки при выполнении программы представляются последовательностью целых чисел. Каждому символу строки может соответствовать одно целое число или последовательность целых чисел. Символ, которому соответствует одно число, называется базовым символом. Символ, не являющийся базовым, представляется последовательностью чисел, которые соответствуют базовым символам. См. Приложение 7.

ВНУТРЕННИЙ МАССИВ. Массив, не являющийся внешним (см. «внешний массив»), расположен в ОЗУ.

ВНУТРЕННЯЯ ПАМЯТЬ. «Быстрая память» ЭВМ, к элементам которой центральный процессор машины имеет непосредственный доступ. То же, что и ОЗУ.

ВНУТРЕННЯЯ РАЗМЕРНОСТЬ ПЕРЕМЕННОЙ. Размерность переменной, которую она получает в описании типа и структуры (п. 15.5.1).

ВСПОМОГАТЕЛЬНЫЕ КАРТЫ АЛЬФА-6. Участвуют в комплексации системного пакета. К ним относятся карты КОНЕЦ ПАСПОРТА, ДК (диспетчерский конец) и ЕКОНЕЦ. Эти карты пробиваются на УПП (БЭСМ-6).

ВСПОМОГАТЕЛЬНЫЕ РАБОТЫ С АРХИВАМИ. Работы, входящие в задание с архивами (п. 4.4): создание архива, уничтожение программы или данных в архиве, печать оглавления, изменение имени архива.

ВХОДНЫЕ КОДИРОВКИ. Информация для системы может вадаваться на различных устройствах подготовки перфокарт. Способ представления символов на перфокарте в виде последовательности перфорационных отверстий называется кодировкой устройства, которая идентифицируется названием устройства подготовки карт. Кодировки, в которых может представляться информация для системы, называются ее входными кодировками. Входными кодировками системы Альфа-6 являются УПП (БЭСМ-6), КУ-3, ЕС, УПП (БЭСМ-4), числовой перфоратор М-220 (см. Приложение 1).

ВЫВОД. Распечатки на АЦПУ получаемых при выполнении программы значений переменных, массивов и выражений, а также текстовой информации.

ВЫВОД НА ПЕРФОРАЦИЮ. Выдача программы (данных) на перфокарты. Задается системными командами * ПЕРФОРИРОВАТЬ — ПРОГРАММУ, * ПЕРФОРИРОВАТЬ — ДАННЫЕ. Перфорация осуществляется в кодировке УПП (БЭСМ-6).

ВЫПОЛНЕНИЕ ПРОГРАММЫ. Выполнение программы, осуществляемое после ее трансляции и загрузки в ОЗУ, состоит в последовательном выполнении операторов программы согласно семантике их выполнения, определяемой языком альфа-6 (п. 15.4). Задание на выполнение программы осуществляется картой * EXECUTE (см. главы 3, 8, 13).

ВЫЧИСЛЯЮЩЕЕ ВЫРАЖЕНИЕ. Логическое или арифметическое выражение. В большинстве случаев речь идет о выражениях, не являющихся переменными.

ГРАФ РАБОТ. Графическое изображение состава и последовательности выполнения работ системы в зависимости от характера входной информации (см. Приложение 12).

ГРУППА ДАННЫХ. Одно или несколько значений, предназначенных для ввода оператором *input* очередного объекта ввода. В данных одна группа от другой отделяется точкой с запятой (контрольной суммой в случае алгибр-данных).

ДАТА ЗАПИСИ. Время (год, число и месяц) записи программы или данных в архив.

ДАННЫЕ см. «альфа-данные».

ДИАПАЗОН. Это либо номер перфокарты, либо пара номеров, разделенных двоеточием. Диапазон выделяет некоторую область в тексте программы или данных (перфокарту или группу перфокарт, номера которых принадлежат диапазону).

ДИНАМИЧЕСКИЕ МАССИВЫ. Массивы, в описании которых хотя бы одна из границ по измерению не является константой.

ДОПОЛНИТЕЛЬНЫЕ РЕСУРСЫ ЗАДАЧИ. Ресурсы, которые должны быть заданы в дополнение к ресурсам, указанным в стандартном паспорте системы. (гл. 13, Приложение 9).

ДРОБНЫЙ ВИД ВЫВОДА. Вывод оператором *output* числового значения в виде десятичного числа с указанием дробной и целой части. Дробный вид вывода задается дробными форматами, начинающимися с букв *z* или *y* (пп. 2.17, 15.4.10).

ЕХ-МАССИВ. То же, что и внешний массив.

ЕХ-ОБЛАСТЬ. Область внешней памяти, на которой размещаются внешние массивы программы. В стандартном режиме ЕХ-область находится на МБ. С помощью системной команды * ОБЛАСТЬ можно задавать ее расположение на ленте или диске (п. 14.8).

ЗАГРУЗКА (ПРОГРАММЫ В ОЗУ). Размещение оттранслированной программы в ОЗУ. При этом осуществляется настройка относительных адресов программы на абсолютные адреса расположения в ОЗУ, а также вычисление адресов внешних подпрограмм и общих объектов.

ЗАДАНИЕ. Часть альфа-задачи, определяющая независимую совокупность работ.

ЗАДАНИЕ С ДАННЫМИ. Один из видов заданий, определяющий совокупность работ над данными (п. 4.4, Приложение 12).

ЗАДАНИЕ С ПРОГРАММОЙ. Один из видов заданий, определяющий совокупность работ над программой (п. 4.4, Приложение 12).

ЗАПИСЬ В АРХИВЕ. Программа или данные, хранящиеся в архиве.

ЗАПИСЬ В КАНАЛЕ. Порция информации в канале ввода-вывода, соответствующая группе данных, или совокупность значений объекта вывода.

ЗОНА. ОЗУ и внешняя память в ЭВМ БЭСМ-6 разбита на участки одинаковой длины, называемые для ОЗУ листами,

для МБ — трактами, для лент и дисков — зонами. Обмен между лентой и ОЗУ возможен только порциями информации, занимающими лист или зону целиком. Каждая зона (лист, тракт) содержит по 1024 слов (ячеек) БЭСМ-6.

ИНДЕКС-РЕГИСТР. Элемент наиболее быстрой памяти ЭВМ БЭСМ-6. В большинстве команд машины используется для модификации адресной части.

КАНАЛ. Специальным образом организованный участок внешней памяти (МБ, МЛ, МД), информация в которую записывается (или считывается) по номеру канала. Различают каналы ввода/вывода, запись в которые производится процедурой *output*, чтение — *input*, и каналы МС Дубна, запись осуществляется процедурой *write*, чтение *read*.

КАРТА УКАЗАНИЯ ВИДА КОДИРОВКИ. Специальная карта, которая указывает вид кодировки, в которой отперфорированы следующие за ней перфокарты. Текст этой перфокарты, содержащий 3 ромбика и название входной кодировки системы Альфа-6, должен быть отперфорирован на УПИ (БЭСМ-6).

КОММУТАЦИЯ КАНАЛА. Переключение номера канала на другой внешний носитель осуществляется процедурой *коммут* (п. 15.4.15).

КОМПЛЕКСАЦИЯ. Сборка задачи из нескольких отдельно транслируемых подпрограмм (гл. 9) для их совместного выполнения. Комплексация осуществляется в момент загрузки в ОЗУ перед выполнением этих задач. Связь подпрограмм может осуществляться посредством формальных параметров и общей памяти. Допускается комплексация альфа-программ, фортран-программ, алгол-программ и мадлен-программ.

КОМПЛЕКТАЦИЯ СИСТЕМНОГО ПАКЕТА. Правила, по которым должен быть собран системный пакет для МС Дубна.

КУРСОР. Текущая позиция на странице, начиная с которой будет продолжен вывод информации по оператору *output*. Идентифицируется номером позиции на странице x и номером строки на странице — y (п. 15.4.10).

КУ-3. Клавишное устройство подготовки серии 3. Характерным является маркер (перфорационное отверстие) в 19 позиции строки.

ЛЕКСИКОГРАФИЧЕСКАЯ УПОРЯДОЧЕННОСТЬ. Упорядоченность элементов по написанию: правый элемент младше левого. Например, элементы массива упорядочиваются вначале по младшему измерению, затем по более старшему и т. д.

ЛЕНТОЧНЫЕ КАНАЛЫ. Каналы ввода/вывода, которые располагаются на лентах (их номера 1—14) (пп. 2.14, 15.4.15).

ЛОКАЛИЗАЦИЯ. Определение области программы, в которой идентификатор понимается в том смысле, который предписывает ему соответствующее описание объекта с данным идентификатором. Если в некотором блоке есть описание объекта с данным идентификатором, то говорят, что объект локализован в данном блоке (п. 15.5).

ЛОКАЛЬНЫЙ ИДЕНТИФИКАТОР (ПЕРЕМЕННАЯ). Идентификатор (переменная) — локальный в некотором блоке, если в начале данного блока есть описание объекта (переменной) с данным идентификатором. Если описание находится в объемлющем блоке, то по отношению к данному блоку этот идентификатор (переменная) называется глобальным (п. 15.5).

МАДЛЕН-ПРОГРАММА (ПОДПРОГРАММА). Программа (подпрограмма), которая написана на автокоде мадлен [16].

МБ. Сокращение: магнитный барабан — носитель внешней памяти ЭВМ БЭСМ-6. Содержит 32 тракта по 1024 слова БЭСМ-6 каждый.

МД. Сокращение: магнитный диск — носитель внешней памяти ЭВМ БЭСМ-6. Содержит 901 зону по 1024 слова БЭСМ-6 каждое.

МАКСИМАЛЬНОЕ ПРЕДСТАВИМОЕ В БЭСМ-6 ЧИСЛО. Число, равное $+0.922337203695_{10} + 19$. Это значение присваивается скалярам и всем элементам статических массивов перед выполнением программы.

МАТЕМАТИЧЕСКИЙ НОМЕР ЛЕНТЫ. Число, состоящее из двух восьмеричных цифр, идентифицирующее ленту (диск).

Соответствие математического номера ленты и номера бобины задается в паспорте системного пакета в разделе «лента» (гл. 13).

МАТЕМАТИЧЕСКИЙ НОМЕР НАПРАВЛЕНИЯ. Первая цифра математического номера ленты.

МАТЕМАТИЧЕСКИЙ НОМЕР УСТРОЙСТВА. Вторая цифра математического номера ленты.

МЕТАЛИНГВИСТИЧЕСКАЯ ПЕРЕМЕННАЯ. Обозначение синтаксической конструкции, используемое для формального описания синтаксиса языка. Значением металингвистической переменной является последовательность символов, полученная применением металингвистических формул.

МЕТАЛИНГВИСТИЧЕСКАЯ ФОРМУЛА. Используется для формального описания синтаксиса языка и представляет собой определение некоторой металингвистической переменной.

МЕТКА PART. Метка, начинающаяся с заглавных букв PART, и стоящая непосредственно перед скобкой блока, который она метит. Этот блок называется внешним блоком.

МЛ. Сокращение: магнитная лента — носитель внешней памяти ЭВМ БЭСМ-6. Содержит 512 зон по 1024 слова БЭСМ-6 каждая.

МНЕМОНИЧЕСКОЕ ОБОЗНАЧЕНИЕ. Буквенное или символическое обозначение некоторого объекта.

МНОГОМЕРНОЕ ЗНАЧЕНИЕ. Упорядоченное множество значений (п. 15.2.8).

МОДУЛЬ. Результат трансляции программы с языка альфа-6. То же, что и стандартный массив МС Дубна [14, стр. 226], [16, стр. 25].

МОНИТОР. Программа, которая анализирует входную информацию системы и в зависимости от ее характера управляет работой системы.

МС ДУБНА. Мопиторная система Дубна. Имеет в своем составе монитор, трансляторы с языков алгол-ГДР, альфа-6, бемш, фортран, и ряд сервисных программ [13], [11], [14].

НАЧАЛЬНАЯ ЗОНА АРХИВА. Зона на ленте (диске), начиная с которой располагается архив с оглавлением (п. 12.1).

НЕРЕГУЛЯРНЫЙ ЦИКЛ. Цикл, не являющийся регулярным (см. «регулярный цикл»).

НОМЕР КАНАЛА. Десятичное число h, $0 \leqslant h \leqslant 14$. Указывается первым параметром в операторах *input*, *output*, *marg* и т. д. В данных указывается вслед за описателем **канал.**

НОМЕР НАЧАЛЬНОЙ ЗОНЫ ЗАПИСИ В АРХИВЕ. Номер зоны, начиная с которой размещается программа или данные в архиве с оглавлением. Этот номер считывается относительно начальной зоны архива (п. 12.1).

НОМЕР ПЕРФОКАРТЫ. Конструкция языка вида $\Diamond\, N\, \Diamond$, которая служит для разбиения текста программы, данных и фрагмента замен на части, называемые перфокартами.

НОРМАЛЬНАЯ НУМЕРАЦИЯ ПРОГРАММЫ (ДАННЫХ). Нумерация, совпадающая с отрезком натурального ряда. Нумерацией называется последовательность номеров перфокарт, встречающаяся в тексте программы (данных).

ОБЛАСТЬ ДИНАМИЧЕСКОЙ ПАМЯТИ. Область ОЗУ, где во время выполнения программы размещаются динамические массивы (п. 8.2).

ОЗУ. Сокращенно от Оперативное Запоминающее Устройство — то же, что «внутренняя память».

ОБЩИЙ МАССИВ. Участок ОЗУ, идентифицируемый именем общего массива и доступный (для чтения и записи) в нескольких подпрограммах скомплексированной программы (п. 2.24).

ОБЩАЯ ПАМЯТЬ. Совокупность общих массивов скомплексированной программы (п. 2.24).

ОБЪЕКТ ВЫВОДА. Выражение или массив, аргумент оператора *output*, значение которого выводится на страницу (п. 2.17).

ОГЛАВЛЕНИЕ АРХИВА. Для каждой записи, хранящейся в архиве, оглавление содержит: имя, номер начальной зоны записи, тип записи (программа или данные), дату записи (год, число, месяц) (п. 12.1).

ОПЕРАЦИОННАЯ СИСТЕМА ДИСПАК (ОС ДИСПАК). Операционная система, предназначенная для пакетной обработки задач в ЭВМ БЭСМ-6 [9].

ОПИСАНИЕ КАНАЛА ДАННЫХ. Языковое средство Н **«канал** ⟨номер канала⟩ **»**, предназначенное для соотнесения групп данных конкретному каналу данных. См. **«канал данных».**

ОПИСАНИЕ МЕТКИ. Вхождение идентификатора в качестве метки, метящей оператор или блок.

ОПТИМИЗАЦИЯ. Совокупность преобразований исходной программы (на внутреннем языке транслятора), которая позволяет запрограммировать основные конструкции языка альфа-6 наилучшим для данной программы способом (с точки зрения быстродействия при выполнении программы).

ОСВОБОЖДЕНИЕ КАНАЛА. Распечатка на АЦПУ всей информации, которая была выведена в канал операторами *output*. Все распечатанные записи удаляются из канала (п. 2.14).

ОСНОВНОЙ СИМВОЛ ЯЗЫКА. Простейшая единица языка, из которых строятся все конструкции языка. В языке альфа-6 основные символы — буквы, цифры, знаки, служебные слова, ограничители и др. (п. 15.2).

ОТДЕЛЬНО ТРАНСЛИРОВАННАЯ ПРОГРАММА (ПОДПРОГРАММА). В общем случае программа составляется из нескольких подпрограмм, которые транслируются независимо друг от друга.

ОТЛАДОЧНЫЕ ИЗМЕНЕНИЯ. Такие изменения в программе, которые сделаны в соответствии с системными командами отладки (см. гл. 11).

ПАРАМЕТР СООБЩЕНИЯ. Информация из выполняемого задания, которая помещается в текст сообщения. Например, идентификатор, ошибочный контекст программы и т. д.

ПАСПОРТ (ЗАДАЧИ). Информация о задаче, необходимая операционной системе для выделения задаче ресурсов, определения ее приоритетности и т. д.

ПЕРЕВЫЧИСЛЯЕМОСТЬ ПЕРЕМЕННЫХ. Изменение значений переменных (вхождение в левую часть оператора присваивания).

ПЕРЕНУМЕРАЦИЯ. Замена нумерации программы (данных) на нормальную нумерацию по команде * НОМЕР.

ПЕРФОКАРТА. Часть текста, начинающаяся с номера перфокарты и оканчивающаяся следующим по порядку вхождения в текст номером, не включая его.

ПОДГОТОВКА ДАННЫХ. То же, что и трансляция данных.

ПОДПРОГРАММА. Программа, обращение к которой осуществляется из другой отдельно-транслируемой программы.

ПОЛЕ СТРАНИЦЫ. Прямоугольная область страницы; различают правое, левое, верхнее, нижнее и внутреннее поля страницы (п. 2.16).

ПОЛЕ (СТАТИЧЕСКИХ или ДИНАМИЧЕСКИХ) МАССИВОВ. Участок ОЗУ, в котором располагаются значения массивов. Различают: поле статических массивов, т. е. таких, у которых границы в описаниях заданы целыми числами, и поле динамических массивов.

ПОЛЬЗОВАТЕЛЬ. Программист или организация, эксплуатирующая систему.

ПОЛЯ. Сплошная область ячеек памяти.

ПОПЕРФОКАРТНОЕ РЕДАКТИРОВАНИЕ. Изменение текста программы или данных с помощью системных команд редактирования * ВСТАВИТЬ, * ВЫБРОСИТЬ, * ЗАМЕНИТЬ (п. 10.1).

ПОСЛЕДОВАТЕЛЬНОСТЬ ЗАДАНИЙ. Часть альфа-задачи, находящаяся между картами * CALL— ALPHA/6 и ◇◇◇ КНЦ (см. гл. 13).

ПРЕДВАРИТЕЛЬНЫЙ КОНТРОЛЬ. Проверка правильности комплектации системного пакета, выявление ошибок перфорации, проверка правильности системной программы и т. д.

ПРЕДУПРЕЖДАЮЩЕЕ СООБЩЕНИЕ. Сообщение системы, предупреждающее о возможной ошибке, которое не изменяет выполнение задания.

ПРЕРЫВАНИЕ. Прекращение выполнения программы ввиду обнаружения ошибки. Если ошибка обнаруживается ОС Диспак, то прерывание называется авостом.

ПРИМИТИВНЫЙ АРХИВ. Архив (без оглавления), содержащий одну запись: программу или данные (п. 12.2).

ПРОГРАММА. Запись алгоритма задачи на языке программирования.

ПРОГРАММА НА ПЕРФОКАРТАХ. Программа, находящаяся на перфокартах во фрагменте задания.

ПРОСТАЯ ПЕРЕМЕННАЯ. См. п. 15.3.1.

РАБОТА СИСТЕМЫ. Независимое действие в выполнении задания. Выполнение задания состоит из совокупности работ.

РАБОЧИЕ ВЕЛИЧИНЫ. Величины (скаляры), вводимые транслятором для хранения при выполнении программы промежуточных результатов вычислений, или возникающих при выполнении оптимизирующих преобразований (п. 5.3).

РАЗДЕЛ ПАСПОРТА. Часть паспорта задачи, определяющая заказ машинных ресурсов или режим пропуска задачи (например, ТРАКТЫ, АЦПУ, ТЕЛЕ).

РАЗМЕРНОСТЬ ПЕРЕМЕННОЙ. Различают абсолютную размерность, внешнюю и внутреннюю размерность (см. «абсолютная размерность переменной»). В Руководстве, если специально не оговорено, под размерностью понимается ее абсолютная размерность.

РАЗМЕТКА (СТРАНИЦЫ). Набор из шести величин ($x1$, $x2$, $x3$, $x4$, $x5$, $x6$), указывающих размеры полей страницы вывода: $x1$ — размер левого поля, $x2$ — длина строки, $x3$ — размер правого поля, $x4$ — размер верхнего поля, $x5$ — число печатаемых на странице строк, $x6$ — размер нижнего поля (п. 2.16). Разметка устанавливается оператором *marg*.

РЕАКЦИЯ СИСТЕМЫ НА ОШИБКУ. Изменение системой режима выполнения задания из-за обнаруженной в нем ошибки. Выполнение задания может прекратиться, либо будет выполнена часть работ задания. В некоторых случаях выполнение задания будет продолжено без изменений (см. гл. 17).

РЕГУЛЯРНЫЙ ЦИКЛ. Цикл типа **для** *i*: = *A* **шаг** *B* **до** *C* **цикл**, где *A*, *B*, *C* — выражения типа целый, у которых входящие в них переменные не изменяют своего значения при выполнении тела цикла. Заголовок такого цикла программируется с использованием индекс-регистра БЭСМ-6 (п. 15.4.6, Приложение 14).

РЕДАКТИРОВАНИЕ. Изменение системой первоначального текста программы или данных. Производится с помощью системных команд редактирования (см. гл. 10).

РЕЖИМ АЛГИБР. Режим системы, позволяющий использовать программу и данные, подготовленные для системы Алгибр. Задается с помощью стандартной карты АЛГИБР (см. Приложение 16).

РЕЖИМ РАБОТЫ СИСТЕМЫ. Одна из нескольких возможностей выполнения задания или отдельной работы.

РЕСУРСЫ. Имеющиеся в памяти ЭВМ БЭСМ-6 ленты, листы ОЗУ, тракты и т. д. (см. Приложения 8 и 9). В их число входят ресурсы, которые занимает система Альфа-6 при своей работе; стандартные ресурсы — те, которые заказаны в стандартном паспорте.

РОДСТВЕННЫЕ ФАКТИЧЕСКИЕ ПАРАМЕТРЫ. Фактические параметры, соответствующие одному и тому же формальному параметру.

СБОЙНАЯ ПРОГРАММА. Программа транслятора, которая работает после прерывания для получения сбойной распечатки

и освобождения каналов ввода-вывода. Выполнение сбойной программы можно отменить системной командой * БЕЗ — СБОЙНОЙ (п. 11.3).

СБОЙНАЯ РАСПЕЧАТКА. Печатаемая на АЦПУ информация после прерывания, состоящая, как правило, из сообщения о причине прерывания, распределения памяти и текущих значений скаляров и массивов.

СЕГМЕНТАЦИЯ ПРОГРАММЫ. Разбиение программы на части, расположенные на различных уровнях памяти.

СЕМАНТИКА. Смысл, который несут в себе синтаксические конструкции.

СЕМАНТИЧЕСКИЙ КОНТРОЛЬ. Проверка транслятором соблюдения семантических правил в программе.

СЕРВИСНАЯ РАСПЕЧАТКА. Печать входной информации, сообщения системы, сбойная информация и т. д. (см. Приложение 18).

СИНТАКСИС. Набор формальных правил, определяющих написание программ на языке программирования.

СИНТАКСИЧЕСКИЙ КОНТРОЛЬ. Проверка системой соблюдения синтаксических правил в программе (данных).

СИСТЕМНАЯ КОМАНДА. Команда системной программы, задающая либо отдельную работу системы, либо режим работ, либо информацию для работы (гл. 14).

СИСТЕМНАЯ ПРОГРАММА. Часть задания (фрагмент системная программа без стандратной карты СИСТЕМА), представляющая собой последовательность системных команд. Задает совокупность работ, которые должна выполнить система при обработке задания (гл. 14).

СИСТЕМНЫЙ ПАКЕТ. Пакет перфокарт, подготовленный для МС Дубна и содержащий альфа-задачу.

СКАЛЯРЫ. Величины нулевой размерности (не массивы), описанные в программе.

СЛУЖЕБНОЕ СЛОВО. Основной символ языка, имеющий вид подчеркнутого слова (п. 15.2).

COMMON-БЛОК. Участок общей памяти, определяемый операторами *common* в фортран-программе [14].

СООБЩЕНИЕ СИСТЕМЫ. Распечатываемая на АЦПУ информация о работе системы Альфа-6. Различают архивные сообщения, предупреждающие сообщения и сообщения об ошибках (гл. 17).

СОСТАВНАЯ ПЕРЕМЕННАЯ. Сформированная или скомпонованная переменная (п. 2.4).

СРЕДСТВА КОМПЛЕКСАЦИИ (ОТДЕЛЬНО ТРАНСЛИРУЕМЫХ ПРОГРАММ В РАМКАХ МС ДУБНА). Возможности связи отдельно транслируемых подпрограмм, представляемые входным языком. В языке альфа-6 к этим возможностям относятся: подпрограммы и описание внешних подпрограмм (п. 2.22), общая память (п. 2.24), оператор *library* (п. 2.23), каналы ввода/вывода (пп. 2.14, 2.15), операторы обмена (п. 2.20).

СТАНДАРТНАЯ РАЗМЕТКА СТРАНИЦЫ. Принятый в системе стандартный формат размещения информации на АЦПУ (информация выводится операторами *output*) (п. 2.16).

СТАНДАРТНЫЕ КАРТЫ АЛЬФА-6. Карты, используемые для комплектации последовательности заданий (см. Приложение 10).

СТАНДАРТНЫЕ ПОДПРОГРАММЫ СИСТЕМЫ. К ним относятся стандартные подпрограммы, реализующие *input*, *output*, подпрограмма вычисления вещественной степени и т. д., обращения к которым транслятор вставляет в модуль, а сами подпрограммы загружаются в ОЗУ вместе с программой перед ее выполнением (гл. 12, п. 15.3.2). Стандартные подпрограммы системы находятся в общей библиотеке МС Дубна, заказываемой управляющей картой * ASSIGN _LIBRARY [19, стр. 19].

СТАНДАТНЫЕ ПРОЦЕДУРЫ (ФУНКЦИИ). Процедуры и функции, определяемые в языке. Обозначаются закрепленными за ними стандартными идентификаторами.

СТАНДАРТНЫЙ ПАСПОРТ СИСТЕМЫ. Паспорт со стандартными ресурсами, предназначенными для типичного использования системы Альфа-6. Для ВЦ СО АН СССР признаком стандартного паспорта служит sc_+.

СТАНДАРТНЫЙ РЕЖИМ. Режим работы системы, который устанавливается, когда нет указания о специальном режиме работ. Например, для задания с программой стандартный режим состоит в трансляции программы (без ее распечатки) и записи полученного модуля во временную библиотеку МС Дубна.

СТАТИЧЕСКИЕ МАССИВЫ. Массив, у которого все границы по изменениям в описании массива заданы целыми числами.

СТРАНИЦА (ВЫВОДА). Прямоугольная область листа АЦПУ (рис. 2.6), внутри которой выделяются: подобласть, занятая выводимой информацией (внутреннее поле) и не занятые подобласти (левое, правое, верхнее, нижнее поля) (п. 2.16).

СТРОКА. Синтаксическая конструкция языка альфа-6, состоящая из последовательности основных символов, обрамленная кавычками (п. 15.2.6).

СТРОКА ПЕРФОКАРТЫ. Позиции физической перфокарты, расположенные одна возле другой в направлении длинной части перфокарты.

СТРУКТУРА ПЕРЕМЕННОЙ. Включает в себя абсолютную размерность переменной и порядки по каждому измерению.

ТЕКСТОВОЕ ПРЕДСТАВЛЕНИЕ СТАНДАРТНОЙ КАРТЫ. Представление стандартной карты в виде: ◇◇◇ ⟨название карты⟩. Например, ◇◇◇ ДАННЫЕ или ◇◇◇ СИСТЕМА.

ТЕКСТОВЫЙ РЕДАКТОР. Часть системы, осуществляющая редактирование программы или данных.

ТЕРМИНАЛ. Внешнее устройство БЭСМ-6, предназначенное для ввода (вывода) информации: телетайп или видеотон.

ТРАНСЛЯТОР. Программа, переводящая запись алгоритма на входном языке в программу на языке ЭВМ.

ТРАНСЛЯЦИЯ ДАННЫХ (ИЛИ ПОДГОТОВКА ДАННЫХ). Процесс перевода данных с языка данных в машинное представление (для ввода оператором *input*).

ТРАНСЛЯЦИЯ ПРОГРАММЫ. Процесс перевода записи алгоритма со входного языка на язык ЭВМ.

УКАЗАТЕЛЬ КАНАЛА. Указатель *IN*, *OUT* или *НО*, *IN* (или *OUT*) указывает на текущую запись, которая вводится по *input* (выводится по *output*). *но* указывает одновременно на последнюю запись для чтения по *input* и на начальную запись для вывода по *output* (п. 2.14).

УПП. Устройство подготовки перфокарт для БЭСМ-6. Имеет кодировку ГОСТ (АЦПУ-128). Обозначается также УПП (БЭСМ-6).

УРОВНИ ПАМЯТИ. Оперативная память, к которой центральный процессор имеет непосредственный доступ, и внешняя память (магнитные барабаны, ленты, диски).

ФИЗИЧЕСКАЯ ПЕРФОКАРТА. Карта установленного стандарта, содержащая 12 строк и 80 столбцов. Является носителем информации, представленной в виде перфорационных отверстий.

ФОРМАТЫ ВЫВОДА. Параметры оператора *output*, задающие вид, в котором выводятся объекты вывода.

ФОРТРАН-ПРОГРАММА (ПОДПРОГРАММА). Программа, написанная на языке фортран.

ФРАГМЕНТ. Часть задания. Виды фрагмента: фрагмент программа, фрагмент данные, фрагмент замен, фрагмент системная программа.

ФРАГМЕНТ ЗАМЕН. Часть задания, начинающаяся стандартной картой ЗАМЕНЫ и содержащая последовательность перфокарт, вставляемых в текст программы (данных) в процессе редактирования.

ФУНКЦИОНИРОВАНИЕ СИСТЕМЫ. Обработка системой Альфа-6 части системного пакета, образованного альфа-задачей.

ШАБЛОН. Начальное заполнение страницы вывода, которое производится до начала вывода на эту страницу (например, сетка для размещения таблицы).

ЭКСПОНЕНЦИАЛЬНЫЙ ВИД ВЫВОДА. Вывод оператором *output* числового значения в виде десятичного числа с указанием мантиссы и порядка числа. Экспоненциальный вид вывода задается экспоненциальным форматом, начинающимся с буквы *e* (п. 2.17).

ЯЗЫК АЛЬФА-6. Входной язык для системы Альфа-6. Является расширением алгола-60 языками альфа, алгамс и операторами *бемш* (гл. 15).

ЯЗЫК ДАННЫХ. Язык, на котором записываются входные данные программы, вводимые оператором *input* (гл. 16).

ЯЗЫК СИСТЕМНЫХ КОМАНД. Язык, на котором записывается системная программа (гл. 14).

ЯЧЕЙКА ОПЕРАТИВНОЙ ПАМЯТИ. Элементарная единица ОЗУ БЭСМ-6, состоит из 48 разрядов. В ячейке может храниться одно число.

ЛИТЕРАТУРА

1. Ершов А. П., Кожухин Г. И., Поттосин И. В. Руководство к пользованию системой Альфа.— Новосибирск: ВЦ СО АН СССР, 1968.
2. Описание языка алгамс./В кн.: Алгоритмы и алгоритмические языки, вып. 3.— М., 1968.
3. Алгоритмический язык алгол-60: Пересмотренное сообщение.— М., 1965.
4. Альфа — система автоматизации программирования: Сб. статей под ред. А. П. Ершова.— Новосибирск, 1967.
5. Руководство к пользованию системой Алгибр: Материалы по математическому обеспечению машины БЭСМ-6.— Л., 1973.
6. Штаркман В. С. Автокод бемш.— М.: ИПМ, 1969.
7. Система БЭСМ-алгол: Методическое руководство по программированию.— М., 1969.
8. Ершов А. П., Кожухин Г. И., Волошин Ю. М. Входной язык для систем автоматического программирования.— Новосибирск: ИМ СО АН СССР, 1964.
9. Бокова И. Д., Зельдинова С. А., Зуев В. И. и др. Инструкция пользователю по работе с операционной системой Диспак для БЭСМ-6.— М.: ИПМ, 1973.
10. Буда А. О., Васючкова Т. С., Грановский А. А., Козловский С. Э., Шелехов В. И. Руководство к пользованию системой автоматизации программирования Альфа-6./Под ред. А. П. Ершова.— Новосибирск: ВЦ СО АН СССР, 1974.
11. Митюшова Л. Л., Пономарева Л. С. Мониторная система «Дубна» в ОС Диспак (инструкция пользователю).— Свердловск: ИММ УНЦ АН СССР, 1976.
12. Хирр Р., Штребель Р. Алгол в мониторной системе «Дубна»./Перевод с нем. Б. П. Примочкина./Под ред. Н. С. Мирономановой, И. Г. Пасынкова, изд. 2.— М., 1973.
13. Мазный Г. Л. Мониторная система «Дубна»: Руководство для пользователей./Под ред. В. Ю. Веретенова и др.— Дубна: ОИЯИ, 1972.

14. Салтыков А. И., Макаренко Г. И. Программирование на языке фортран.— М.: Наука, 1976.
15. Морозова Л. Б. Транслятор с автокода бемш в мониторной сисгеме «Дубна».— М.: ИПМ АН СССР, 1973.
16. Волков А. И. Автокод *madlen*.— Дубна: ОИЯИ 1969.
17. Инструкция по программированию на БЭСМ-6.— М.: МЗСАМ, 1971.
18. Входной язык для системы Альфа-6: Информационно-оперативный материал (руководство к пользованию).— Новосибирск: ВЦ СО АН СССР, 1976.
19. Мазный Г. Л. Программирование на БЭСМ-6 в системе «Дубна».— М.; Наука, 1978.

АЛФАВИТНЫЙ УКАЗАТЕЛЬ ОПРЕДЕЛЯЕМЫХ ПОНЯТИЙ И СИНТАКСИЧЕСКИХ ЕДИНИЦ РАЗДЕЛА III

Ссылки разбиты на две группы:

опр.— ссылка, стоящая с сокращением «опр», отсылает к соответствующему синтаксическому определению;

текст — ссылки, стоящие за словом «текст», отсылают к определениям, данным в тексте;

Аникеева Ирина Николаевна
Буда Анатолий Олегович
Васючкова Татьяна Сергеевна
Кожухина Светлана Константиновна
Козловский Сергей Энеевич
Шелехов Владимир Иванович

ТРАНСЛЯТОР АЛЬФА-6
В СИСТЕМЕ ДУБНА

(Серия: «Библиотечка программиста»)

М., 1979 г., 352 стр. с илл.

Редакторы *Н. Н. Васина, Р. Л. Смелянский*

Техн. редактор *Л. В. Лихачева*

Корректоры *М. Л. Медведская,*
Н. Б. Румянцева

ИБ № 11391

Сдано в набор 4.06.79. Подписано к печати 28.11.79. Т-21505. Бумага 84×108$^{1}/_{32}$, тип. № 2. Обыкновенная гарнитура. Высокая печать. Условн. печ. л. 18,48. Уч.-изд. л. 18,82. Тираж 15 000 экз. Заказ № 555. Цена книги 1 р. 10 к.

Издательство «Наука»
Главная редакция физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15

4-я типография издательства «Наука»
630077, Новосибирск, 77, Станиславского, 25

1 р. 10 к.