

ГОСУДАРСТВЕННЫЙ СТАНДАРТ
СОЮЗА ССР

ЯЗЫК ПРОГРАММИРОВАНИЯ
АЛГАМС

ГОСТ 21551—76

Издание официальное

Цена 15 коп.

ГОСУДАРСТВЕННЫЙ КОМИТЕТ СССР ПО СТАНДАРТАМ
Москва

gost.e-ecolog.ru

УДК 681.3.065.4:006.354 Группа П85

ГОСУДАРСТВЕННЫЙ СТАНДАРТ СОЮЗА ССР

ЯЗЫК ПРОГРАММИРОВАНИЯ АЛГАМС

Programming languages ALGAMS

ГОСТ 21551—76

Постановлением Государственного комитета стандартов Совета Министров СССР от 6 февраля 1976 г. № 353 срок действия установлен

с 01.07.1977 г.
до 01.07.1982 г.

Настоящий стандарт устанавливает описание языка программирования АЛГАМС*, предназначенного для автоматизации программирования при решении научных и инженерно-технических задач, а также для обмена алгоритмами.

Установленное настоящим стандартом описание языка АЛГАМС должно применяться при создании алгоритмов на языке АЛГАМС и трансляторов с АЛГАМСа.

В алгоритмах на АЛГАМСе должны использоваться только те языковые средства, которые установлены настоящим стандартом.

Транслятор с АЛГАМСа должен обеспечивать трансляцию любого алгоритма, разработанного в соответствии с настоящим стандартом.

Указатель определенных в стандарте понятий и синтаксических единиц приведен в справочном приложении 1.

1. СТРУКТУРА ЯЗЫКА

Назначением алгоритмического языка является описание вычислительных процессов. Описания правил вычислений основываются на хорошо известном понятии арифметического выражения, содержащего в качестве составных частей числа, переменные и функции. Из таких выражений путем применения правил арифме-

* История создания языка АЛГАМС и его отличия от языка АЛГОЛ-60 приведены в справочных приложениях 4 и 5.

Издание официальное **Перепечатка воспрещена**

Переиздание. Февраль 1979 г.

★

©Издательство стандартов, 1979

тической композиции образуются самостоятельные единицы языка — явные формулы, называемые операторами присваивания.

Для того, чтобы указать ход вычислительных процессов, 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 **begin*** и **end.**

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 переменной, размерность массива чисел или даже совокупность правил, определяющих некоторую функцию. Последовательность описаний и следующая за ней последовательность операторов, заключенные между **begin** и **end**, составляют блок. Каждое описание вводится в блоки таким путем и действительно только для этого блока.

Программа является блоком или составным оператором, который не содержится внутри другого оператора и который не использует других операторов, не содержащихся в нем.

Ниже будут приведены синтаксис и семантика языка**.

1.1. Формализм для синтаксического описания

Синтаксис описывается с помощью металингвистических формул. Их интерпретацию лучше всего можно объяснить на примере:

<ab>::= (|||<ab>(|<ab><d>

Последовательности знаков, заключенные в скобки <>, представляют собой металингвистические переменные, значениями которых являются последовательности символов. Знаки ::= и | (последний со значением «или») — это металингвистические связки. Любой знак в формуле, который не является переменной или связкой, обозначает самого себя (или класс знаков, ему подобных). Соединение знаков и (или) переменных в формуле означает соединение обозначаемых последовательностей. Таким образом, формула, приведенная выше, задает рекурсивное правило для обра-

* Соответствие между английскими и русскими служебными словами указано в справочном приложении 2.

** Если утверждается, что точность арифметических действий, вообще говоря, не указана, или когда результат некоторого процесса остается или объявляется неопределенным, следует понимать, что программа станет полностью определять некоторый вычислительный процесс только в том случае, если дополнительная информация укажет как подразумеваемые точность и вид арифметических действий, так и последовательность выполняемых действий для всех случаев, которые могут встретиться в процессе вычислений.

зования значений переменной <ab>. Она указывает, что <ab> может иметь значение либо (, либо [, или же, если дано некоторое допустимое значение <ab>, то еще одно значение можно получить, поставив за <ab> символ (, или некоторое значение переменной <*d*>. Если значениями <d> являются десятичные цифры, то некоторые из значений <ab> суть:

[(((1 (3 7 (
(1 2 3 4 5 (
(((
[8 6

Чтобы облегчить изучение, символы, используемые для различения металингвистических переменных (то есть последовательностей знаков, стоящих внутри скобок <>, подобно ab, в приведенном выше примере), выбраны в виде слов, приблизительно описывающих природу соответствующей переменной. Там, где слова, введенные таким способом, используются где-либо в тексте, они всегда, если не оговорено противное, относятся к соответствующему синтаксическому определению. Кроме того, некоторые формулы приведены по нескольку раз.

Определение:

<пусто>::=

(то есть строка, не содержащая символов).

2. ОСНОВНЫЕ СИМВОЛЫ, ИДЕНТИФИКАТОРЫ, ЧИСЛА И СТРОКИ. ОСНОВНЫЕ ПОНЯТИЯ

Язык строится из следующих основных символов:

<основной символ>::=<буква>|<цифра>|<логическое значение>|<ограничитель>

2.1. Буквы

<буква>::=

A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

Буквы не имеют индивидуального смысла. Они используются для образования идентификаторов и строк (см. пп. 2.4 и 2.6).

2.2. Цифры. Логические значения

2.2.1. *Цифры*

<цифра>::=0|1|2|3|4|5|6|7|8|9

Цифры используются для образования чисел, идентификаторов и строк.

2.2.2. *Логические значения*

<логическое значение>::=**true** | **false**

Логические значения имеют фиксированный очевидный смысл*.

* **true** — истина, **false** — ложь.

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>::= **go to** | **if** | **then** | **else** | **for** | **do**

<разделитель>::= , | . | 10 | : | ; | := | — | **step** | **until** | **while** | **comment**

<скобка>::=( ) | [ ] | ' ' | **begin** | **end**

<описатель>::= **Boolean** | **integer** | **real** | **array** | **switch** | **procedure**

<спецификатор>::= **string** | **label** | **value**

Ограничители имеют фиксированный смысл, который в большинстве случаев очевиден*, а в остальных случаях будет указан ниже. Пробел или 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 новых символов:	Эквиваленты
: **comment** <любая последовательность, не содержащая символа ";" >;	;
begin comment <любая последовательность, не содержащая символа ";" >;	**begin**
end <любая последовательность, не содержащая ни символа „end", ни символа ";", ни символа „else" >	**end**

Эквивалентность здесь означает, что любую из трех конструкций, указанных в левой колонке, если она встречается вне некоторой строки, можно заменять соответствующим ей символом, указанным в правой колонке. Эта замена не оказывает никакого влияния на работу программы. При этом считается, что конструкцию примечания, встретившуюся раньше при чтении текста слева направо, следует заменять прежде, нежели более поздние конструкции, содержащиеся в этой последовательности.

* Перевод английских слов, изображающих эти основные символы: **go to** — перейти к, **if** — если, **then** — то, **else** — иначе, **for** — для, **do** — выполнить, **step** — шаг, **until** — до, **while** — пока, **comment** — примечание, **begin** — начало, **end** — конец, **Boolean** — булевский, или логический, **integer** — целый, **real** — вещественный, **array** — массив, **switch** — переключатель, **procedure** — процедура, **string** — строка, **label** — метка, **value** — значение

2.4. Идентификаторы

2.4.1. *Синтаксис*

<идентификатор>::=<буква>|<идентификатор><буква>|
<идентификатор><цифра>|

2.4.2. *Примеры*

Q
SOUP
V17A
A34KTMNS
MARILYN

2.4.3. *Семантика*

Для описания языка удобно под термином идентификатор понимать не только определенную в п. 2.4.1 и дальнейшим синтаксисом синтаксическую единицу, но и специальные последовательности букв и цифр, начинающиеся с букв EX или PART и изображающие синтаксические единицы <внешний идентификатор> (пп. 3.1.1 и 5.2.6) и <идентификатор части> (пп. 4.1.1 и 4.1.3).

Идентификаторы не имеют неизменно присущего им смысла, а служат для обозначения простых переменных, массивов, меток, идентификаторов части, процедур и формальных параметров. Их можно выбирать произвольно (рекомендуется при этом учитывать разд. 6, а также указанные в пп. 4.1.3 и 5.2.6 ограничения на употребление идентификаторов с первыми буквами EX или PART).

Один и тот же идентификатор нельзя использовать для обозначения двух различных объектов, за исключением случая, когда эти объекты имеют несовместные области действия (п. 2.7 и разд. 5).

2.5. Числа

2.5.1. *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>

2.5.2. *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

2.5.3. *Семантика*

Десятичные числа имеют свой обычный смысл. Порядок — это масштабный множитель, выраженный как целая степень десяти.

2.5.4. *Типы*

Целые имеют тип **integer**. Остальные числа имеют тип **real** (п. 5.1).

2.6. Строки

2.6.1. *Синтаксис*

<строка>::='<любая последовательность символов, не содержащая ' или '>'|

2.6.2. *Пример*

'.. THIS ␣ IS ␣ A ␣ STRING'

2.6.3. *Семантика*

Для того, чтобы в языке можно было иметь дело с произвольными последовательностями основных символов, введены кавычки для строк: 'и'. Символ ␣ обозначает пробел. Вне строк он не имеет смысла.

Строки используют в качестве фактических параметров процедур (см. пп. 3.2 и 4.7).

2.7. Величины, классы и области действия

Различают следующие классы величин: простые переменные, массивы, метки, идентификаторы части, переключатели и процедуры.

Область действия величины — совокупность операторов и выражений, внутри которых определена связь этой величины и изображающего ее идентификатора. Для всех величин, кроме стандартных процедур и функций, меток и идентификаторов частей,— это область действия соответствующего описания (см. разд. 5 и п. 4.1.3).

2.8. Значения и типы

Значение — некоторое упорядоченное множество чисел (частный случай: отдельное число), некоторое упорядоченное множество логических значений (частный случай: отдельное логическое значение) или некоторая метка (идентификатор части).

Некоторые синтаксические единицы обладают значениями. Во время выполнения программы эти значения могут изменяться. Значения выражений и их составных частей определяются в разд. 3. Значение идентификатора массива есть упорядоченное множество значений соответствующего массива переменных с индексами (см. п. 3.1.4.1).

Различные типы **(integer, real, Boolean)** в основном обозначают свойства значений. Типы, связанные с синтаксическими единицами, относятся к значениям этих единиц.

3. ВЫРАЖЕНИЯ

В языке АЛГАМС 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 указателей функций содержит выражения, определение выражений и их составных частей по необходимости является рекурсивным.

<выражение>::=<арифметическое выражение>|<логическое выражение>|<именующее выражение>

3.1. Переменные

3.1.1. *Синтаксис*

<идентификатор переменной>::=<идентификатор>

<простая переменная>::=<идентификатор переменной>

<индексное выражение>::=<арифметическое выражение>

<список индексов>::=<индексное выражение>|<список индексов>, <индексное выражение>

<идентификатор массива>::=<идентификатор>|<внешний идентификатор>

<внешний идентификатор>::=EX|<внешний идентификатор><буква>|<внешний идентификатор><цифра>

<переменная с индексами>::=<идентификатор массива>[<список индексов>]

<переменная>::=<простая переменная>|<переменная с индексами>

3.1.2. *Примеры*

```
EPSILON
DELTA
A17
Q
X[SIN(N×P1/2), Q[3, N, 4]]
```

3.1.3. *Семантика*

Переменная — наименование, данное некоторому отдельному значению. Это значение может использоваться в выражениях для образования других значений. Его можно изменять посредством операторов присваивания (см. п. 4.2, а также ограничения на использование внешних идентификаторов в п. 5.2.6).

Тип значения данной переменной определяется описанием самой переменной (см. п. 5.1) или соответствующего идентификатора массива (см. п. 5.2).

3.1.4. *Индексы*

3.1.4.1. Переменные с индексами именуют значения, которые являются компонентами многомерных массивов (см. п. 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 при помощи переменной с индексами, определяется по фактическому числовому значению ее индексов (см. п. 3.3)

3.1.4.2. Каждая индексная позиция воспринимается как переменная типа **integer**, и вычисление индекса понимается как присваивание значения этой фиктивной переменной (см. п. 4.2.4). Значение переменной с индексами определено только в том случае, когда значение индексного выражения находится в пределах границ индексов массива (см. п. 5.2).

3.2. Указатели функций

3.2.1. *Синтаксис*

<идентификатор процедуры>::=<идентификатор>

<фактический параметр>::=<строка>|<выражение>|<идентификатор массива>|<идентификатор переключателя>|<идентификатор процедуры>

<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>

3.2.2. *Примеры**

```
SIN (A, B)
J (V+S, N)
R
S(T—5) температура: (T) давление: (P)
COMPILE (':=') STACK : (Q)
```

3.2.3. *Семантика*

Указатели функций определяют отдельные числовые или логические значения, которые получаются в результате применения заданных совокупностей правил, определяемых описанием процедуры (см. п. 5.4), к фиксированным совокупностям фактических параметров. Правила, регулирующие задание фактических парамет-

* В данном примере и далее в некоторых случаях, когда идентификатор имеет мнемоническое значение, он записывается русскими словами. При этом будет предполагаться, что алфавит расширен введением строчных русских букв.

ров, даны в п. 4.7. Не каждое описание процедуры определяет значение какого-либо указателя функции.

3.2.4. *Стандартные функции*

(См. п. 6.1).

3.3. Арифметические выражения

3.3.1. *Синтаксис*

<знак операции типа сложения>::=+|—

<знак операции типа умножения>::=×|/

<первичное выражение>::=<число без знака>|<переменная>|<указатель функции>|(<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 сложения><терм>

<условие>::=**if** <логическое выражение> **then**

<арифметическое выражение>::=<простое арифметическое выражение>|<условие><простое арифметическое выражение> **else** <арифметическое выражение>

3.3.2. *Примеры*

Первичные выражения

7.394_{10}—8
SUM
W[I+2,8]
COS (Y+X×3)
(A—3/Y+VU↑8)

Множители:

OMEGA
SUM↑COS(Y+Z×3)
7.394_{10}—8↑W[I+2,8]↑(A—3/Y+VU↑8)

Термы:

U
OMEGA×SUM↑COS(Y+Z×3)/7.394_{10}—8↑W[I+2,8]↑(A—3/Y+VU↑8)

Простое арифметическое выражение:

U—YU+OMEGA×SUM↑COS(Y+Z×3)/7.394_{10}—8↑W[I+2,8]↑(A—3/Y+VU↑8)

Арифметические выражения:

W×U—Q(S+CU)↑2

if Q>0 **then** S+3×Q/A **else** 2×S+3×Q

if A<0 **then** U+V **else if** A×B>17 **then** U/V **else if** K≠Y **then** V/U **else** 0

A×SIN (OMEGA×T)

$0.57_{10}12\times A[N\times(N-1)/2,0]$
(A×ARCTAN (Y)+Z) ↑ (7+Q)
if Q **then** N- -1 **else** N
if A<0 **then** A/B **else if** D=0 **then** B/A **else** Z

3.3.3. *Семантика*

Арифметическое выражение является правилом для вычисления числового значения. В случае простых арифметических выражений это значение получается посредством выполнения указанных арифметических операций над фактическими числовыми значениями первичных выражений, входящих в данное выражение (см. п. 3.3.4).

Что такое фактическое числовое значение первичного выражения, ясно в случае чисел. Для переменных оно является текущим значением (последним по времени присвоенным значением), а для указателей функций оно является значением, полученным по правилам вычислений, определяющих процедуру (см. п. 5.4.4), примененным к текущим значениям параметров процедуры, заданных в выражении. Наконец, значение арифметического выражения, заключенного в скобки, совпадает со значением арифметического выражения, полученного из исходного удалением заключающих его скобок. В конечном счете это значение должно выразиться посредством рекурсивного анализа, исходя из значений остальных трех видов первичных выражений. Значение арифметического выражения <условие> <простое арифметическое выражение> **else** <арифметическое выражение> определяется следующим образом.

Вычисляется фактическое значение логического выражения (см. п. 3.4), входящего в условие. Если это значение есть **true**, то значение рассматриваемого арифметического выражения определяется как значение простого арифметического выражения, стоящего между условием и ограничителем **else**. Если же вычисленное значение логического выражения есть **false**, то значение исходного арифметического выражения определяется как значение арифметического выражения, следующего за ограничителем **else**. Однако в обоих случаях значению исходного арифметического выражения приписывается тип согласно п. 3.3.4.4.

3.3.4. *Операции и типы*

Составные части простых арифметических выражений (не считая логических выражений, употребляемых в условиях) должны иметь тип **real** или **integer** (см. п. 5.1). Смысл основных операций и типы выражений, к которым они приводят, определяются следующими правилами.

3.3.4.1. Знаки операций +, – – и × имеют обычный смысл (сложение, вычитание и умножение). Результат имеет тип **integer**, если оба операнта имеют тип **integer**, в противном случае — **real**.

3.3.4.2. Операция <терм>/<множитель> означает деление, понимаемое как умножение терма на обратную величину множителя с соответствующим учетом правил старшинства (см. п. 3.3.5). Таким образом, например

$$a/b\times7/(p-q)\times v/s$$

означает

$$(((a\times(b^{-1}))\times7)\times((p-q)^{-1}))\times v)\times(s^{-1}).$$

Знак операции / определен для всех четырех комбинаций типов **real** и **integer** и в любом случае результат типа **real.**

3.3.4.3. Операция <множитель> ↑ <первичное выражение> означает возведение в степень, где множитель есть основание, а первичное выражение — показатель степени. Таким образом, например,

$$2\uparrow n\uparrow k \text{ означает } (2^n)^k,$$

тогда как

$$2\uparrow(n\uparrow m) \text{ означает } 2^{(n^m)}.$$

Если писать I вместо выражения типа **integer,** R вместо выражения типа **real,** A вместо выражения типа **integer** или **real,** а соответствующими малыми буквами (i, r, a) обозначить значения этих выражений, то результат возведения в степень (конечно, в предположении, что I и R — первичные выражения, а A — множитель) определяется следующими правилами:

A ↑ I Если i>0, то a×a× ... ×a (i раз) того же типа, что и A, если I — целое без знака, и типа **real** в противном случае

Если i=0 и a≠0, то 1 того же типа, что и A, если I — целое без знака, и типа **real** в противном случае.

Если i<0 и a=0, то не определено.

Если i<0 и a≠0, то 1/(a×a× ... ×a) (знаменатель имеет –i множителей) типа **real.**

A ↑ R Если a>0, то EXP (r×LN(a)) типа **real.**

Если a=0 и r>0, то 0.0 типа **real.**

Если a=0 и r≤0, то не определено.

Если a<0, то не определено.

3.3.4.4. Тип выражения **if** B **then** A1 **else** A2 есть **integer,** если A1 и A2 оба типа **integer,** и **real** в противном случае.

3.3.5. *Старшинство операций*

Операции в пределах одного выражения выполняются в последовательности слева направо с учетом следующих дополнительных правил.

3.3.5.1. Согласно синтаксису (см. п. 3.3.1), выдерживается следующий порядок старшинства:

первый: ↑

второй: ×, /

третий: +, –

3.3.5.2. Значение выражения между левой скобкой и соответствующей правой скобкой вычисляется самостоятельно и используется в дальнейших вычислениях. Следовательно, желаемый порядок выполнения операций в пределах выражения всегда может быть достигнут соответствующей расстановкой скобок.

3.3.6. *Арифметика величин типа* **real**.

Числа и переменные типа real должны интерпретироваться в смысле численного анализа, то есть как объекты, определенные с присущей им конечной точностью. Аналогично в любом арифметическом выражении явно подразумевается возможность отклонения от математически определяемого результата. Тем не менее никакая точная арифметика не определяется и, конечно, считается, что в зависимости от различных конкретных представлений значения арифметических выражений могут вычисляться по-разному. Контроль за возможными последствиями таких различий должен проводиться методами численного анализа. Этот контроль должен рассматриваться как часть описываемого процесса и, следовательно, выражаться в терминах самого языка.

3.4. Логические выражения

3.4.1. *Синтаксис*

<знак операции отношения>::=<|<|=|≥|>|≠

<отношение>::=<простое арифметическое выражение>

<знак операции отношения> <простое арифметическое выражение>

<первичное логическое выражение>::=<логическое значение>|<переменная>|<указатель функции>|<отношение>|(<логическое выражение>)

<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>

<логическое выражение>::=<простое логическое выражение>|<условие><простое логическое выражение> else <логическое выражение>

3.4.2. *Примеры*

X=.2

Y V.Z Q

A;B 5;Z D Q↑2

P.Q∨X≠Y

Q ¬|A∧B∧ C∨D∨E ↑ F
if K<1 **then** S>W **else** H≤ C
if if if A **then** B **else** C **then** D **else** F **then** G **else** H<K

3.4.3. *Семантика*

Логическое выражение является правилом для вычисления логического значения. Принципы вычисления полностью аналогичны правилам, данным в п. 3.3.3 для арифметических выражений.

3.4.4. *Типы*

Переменные и указатели функций, используемые в качестве первичных логических выражений, должны быть описаны как имеющие тип Boolean (см. пп. 5.1 и 5.4.4).

3.4.5. *Операции*

Отношения принимают значение **true** в том случае, когда соответствующее отношение удовлетворяется для входящих в него выражений; в противном случае они принимают значение **false.**

Значения знаков логических операций ¬ (не), ∧ (и), ∨ (или), ⊃ (влечет) и ≡ (эквивалентно) даются следующей функциональной таблицей:

b1	false	false	true	true
b2	false	true	false	true
¬ b1	true	true	false	false
b1 ∧ b2	false	false	false	true
b1 ∨ b2	false	true	true	true
b1 ⊃ b2	true	true	false	true
b1 ≡ b2	true	false	false	true

3.4.6. *Старшинство операций*

Операции в пределах одного выражения выполняются в последовательности слева направо с учетом следующих дополнительных правил:

3.4.6.1. Согласно синтаксису (см. п. 3.4.1), выдерживается следующий порядок старшинства:

первый: арифметические выражения согласно п. 3.3.5

второй: ≤ = ≠
третий: ¬
четвертый: ∧
пятый: ∨
шестой: ⊃
седьмой: ≡

3.4.6.2. Применение скобок интерпретируется в смысле, данном в п. 3.3.5.2.

3.5. Именующие выражения

3.5.1. *Синтаксис*

<метка> : : = <идентификатор>

<идентификатор части>::=PART / <идентификатор части><буква> / <идентификатор части><цифра>

<идентификатор переключателя>::=<идентификатор>

<указатель переключателя>::=<идентификатор переключателя>[<индексное выражение>]

<именующее выражение>::=<метка> / <идентификатор части> / <указатель переключателя>

3.5.2. *Примеры*

```
Z
P9
выбрать [N—1]
TOWN [if Y<0 then N else N+1]
```

3.5.3. *Семантика*

Именующее выражение является правилом для определения метки или идентификатора части. Если именующее выражение есть метка или идентификатор части, то желаемый результат уже получен. Указатель переключателя отсылает к описанию соответствующего переключателя (см. п. 5.3) и по числовому значению его индексного выражения выбирает одну из меток, содержащихся в переключательном списке описания переключателя. Выбор осуществляется пересчетом этих меток слева направо.

3.5.4. *Индексное выражение*

Вычисление значения индексного выражения аналогично такому же вычислению для переменных с индексами (см. п. 3.1.4.2). Значение указателя переключателя определено только в том случае, когда индексное выражение принимает одно из положительных значений 1, 2, 3, ..., n, где n — число членов в переключательном списке.

4. ОПЕРАТОРЫ

Единицы действий в языке называются операторами. Обычно они выполняются в той последовательности, в которой написаны. Однако эта последовательность действий может прерываться операторами перехода, которые явно определяют своего преемника, и сокращаться условными операторами, которые могут вызывать пропуск некоторых операторов.

Для того, чтобы имелась возможность указать фактический порядок следования операторов в процессе работы, оператор может быть снабжен метками.

Для возможной сегментации программы операторы, являющиеся блоками, могут, кроме того, помечаться особыми метками, называемыми идентификаторами частей.

Ввиду того, что последовательности операторов могут группироваться в составные операторы и блоки, определение оператора

по необходимости должно быть рекурсивным. Кроме того, поскольку описания, которые даны в разд. 5, существенно входят в синтаксическую структуру, синтаксическое определение операторов должно предполагать, что описания уже определены.

4.1. Составные операторы и блоки

4.1.1. *Синтаксис*

<непомеченный основной оператор>::=<оператор присваивания>|<оператор перехода>|<пустой оператор>|<оператор процедуры>

<основной оператор>::=<непомеченный основной оператор>|<метка>:<основной оператор>

<безусловный оператор>::=<основной оператор>|<составной оператор>|<блок>

<оператор>::=<безусловный оператор>|<условный оператор>|<оператор цикла>

<тело составного>::<оператор>|<тело составного>;<оператор>

<начало блока>::=**begin** <описание>|<начало блока>;<описание>

<непомеченный составной>::=**begin** <тело составного>**end**

<непомеченный блок>::=<начало блока>;<тело составного>**end**

<составной оператор>::=<непомеченный составной>|<метка>:=<составной оператор>

<блок>::=<непомеченный блок>|<идентификатор части>:<непомеченный блок>|<метка>:=<блок>

<программа>::=<блок>|<составной оператор>

Этот синтаксис можно проиллюстрировать следующим образом. Обозначим произвольные операторы, описания, метки и идентификаторы частей буквами S, D, L и P соответственно. Тогда основные синтаксические единицы примут следующий вид

составной оператор:

L:L: . . . **begin** S; S; . . . S; S **end**

блок:

L:L: . . . : P: **begin** D; D; . . D; S; S; . . . S; S **end**

При этом нужно помнить, что каждый из операторов S может в свою очередь, быть составным оператором или блоком. Метка переключателя в любом операторе (основном, составном, блоке, условном и цикле), а также идентификатор части в блоке можно использовать соответствующим образом.

4.1.2. *Пример*

основные операторы:

A:=P+Q

go to NAPLES

START : CONTINUE : W : = 7.993

составной оператор:

```
begin X:=0; for Y:=1 step 1 until N do X=X+A[Y];
      if X>Q then go to STOP else if X>W-2 then go to S;
      AW:ST:W:=X+BOB end
```

блок:

```
      Q: begin integer I, K; real W;
      for I:=1 step 1 until M do
      for K:=I+1 step 1 until M do
      begin W:=A[I, K]; A[I, K]:=A[K, I]; A[K, I]:=W
      end FOR I ANDK
      end BLOCK Q
```

4.1.3. *Семантика*

Каждый блок вводит новый уровень обозначений. Это означает, что некоторые идентификаторы, встречающиеся внутри блока, то есть между соответствующими скобками **begin** и **end** определяются как локальные в данном блоке, то есть объект, представленный таким идентификатором внутри данного блока, существует только внутри этого блока, а любой объект, представленный тем же идентификатором вне внутренности данного блока, нельзя непосредственно использовать внутри блока (о косвенном использовании см. пп. 4.7.3.2, 4.7.3.3, 5.3.4).

Поскольку в языке имеются стандартные процедуры и функции (см. разд. 6), и наряду с блоком новый уровень обозначений может быть введен как описанием процедуры, так и телом процедуры, то для объяснения правил локализации введем некоторые фиктивные блоки. Во-первых, будем считать, что программа содержится в некотором объемлющем фиктивном блоке, внутренность которого и есть вся программа. Во-вторых, будем считать каждое описание процедуры (см. п. 5.4) фиктивным блоком, внутренность которого начинается с совокупности формальных параметров в заголовке описания этой процедуры, точнее, что начинающие описание процедуры описатель типа (если он есть), описатель **procedure** и идентификатор описываемой процедуры как бы составляет открывающую скобку **begin** фиктивного блока, в то время, как закрывающая скобка **end** этого блока подразумевается непосредственно перед точкой с запятой, следующей за описанием рассматриваемой процедуры. В-третьих, будем считать каждое тело процедуры внутренностью фиктивного блока, подразумевая непосредственно перед телом процедуры и непосредственно за ним соответствующие скобки **begin** и **end**. Для двух любых блоков, включая фиктивные, справедливо утверждение о том, что либо они не пересекаются, либо один из них содержится в другом. Понимая под термином блок как определенные синтаксисом п. 4.1.1 блоки, так и только что описанные фиктивные блоки, можно сформулировать правила локализации идентификаторов следующим образом.

Каждое обозначение, то есть связь идентификатора с объектом (локализация), вводится в некотором блоке. Введенная в блоке связь идентификатора с каким-либо объектом действует внутри этого блока всюду, но не внутри содержащихся в этом блоке блоков, в которых этот же идентификатор связан с другим объектом.

Идентификатор, встречающийся внутри блока и нелокальный в нем, должен быть локальным в одном из блоков, объемлющем данный блок. Таким образом идентификатор, нелокальный в блоке А, может быть локальным или не локальным в блоке В, для которого А является одним из его операторов.

Специальный идентификатор LIBRARY (см. пп. 5.4.1 и 5.4.3) и идентификаторы стандартных процедур и функций (см. разд. 6) локальны в самом внешнем фиктивном блоке. Локализация идентификаторов простых переменных, массивов, переключателей и процедур (кроме стандартных) осуществляется описаниями (см. разд. 5) в начале соответствующего блока. В блоке локализуются также идентификаторы меток и идентификаторы частей, помечающие те операторы, которые лежат внутри данного блока, но не лежат внутри блока, содержащегося внутри данного блока. Наконец, в фиктивном блоке, возникающем из описания процедуры, локализуются идентификаторы формальных параметров из соответствующей совокупности формальных параметров. В фиктивном блоке, возникающем из тела процедуры, могут быть локальны лишь идентификаторы меток и идентификаторы частей.

Идентификатор части изображается идентификатором, начинающимся с букв PART, и служит для указания транслятору о сегментации составляемой программы. Предполагается, что команды составляемой программы, которые соответствуют блоку, помеченному идентификатором части (за исключением команд, соответствующих блокам, входящим в данный, и также помеченным идентификаторами части), располагаются в отдельном участке внешней памяти машины и целиком вызываются в оперативную память при входе в этот блок. Считается, что части программы, не содержащиеся в блоках, помеченных идентификаторами части, постоянно находятся в оперативной памяти.

4.2. Операторы присваивания

4.2.1. *Синтаксис*

<левая часть>::= <переменная>:= | <идентификатор процедуры>:=

<список левой части>::=<левая часть> | <список левой части> <левая часть>

<оператор присваивания>::=<список левой части> <арифметическое выражение> | <список левой части> <логическое выражение>

4.2.2. *Примеры*

S:=P[0]:=N:=N+1+S
N:=N+1
A:=B/C—V—Q×S
S[V, K+2] : =3 ARCTAN (T×ZETA)
V:=Q>Y ∨ Z

4.2.3. *Семантика*

Операторы присваивания служат для присваивания значения выражения одной или нескольким переменным или идентификаторам процедур. Присваивание идентификатору процедуры может встречаться только внутри тела процедуры, определяющей значение указателя функции (см. п. 5.4.4). Подразумевается, что в общем случае этот процесс проходит в следующие три этапа.

4.2.3.1. Значения всех индексных выражений, встречающихся в переменных левой части, вычисляются в порядке слева направо.

4.2.3.2. Вычисляется значение выражения в операторе.

4.2.3.3. Значение выражения присваивается всем переменным левой части, при этом индексные выражения имеют значения, вычисленные на шаге 4.2.3.1.

4.2.4. *Типы*

Переменные и идентификаторы процедур списка левой части должны по описанию иметь один и тот же тип. Если это тип **Boolean**, то выражение также должно быть типа **Boolean**. Если этот тип **real** или **integer**, то выражение должно быть арифметическим. Если тип арифметического выражения отличается от типа переменных и идентификаторов процедур, то считают, что автоматически применяется соответствующая функция преобразования. Имеется в виду, что для преобразования из типа **real** в тип **integer** функция преобразования выдаст результат, эквивалентный

ENTIER (E+0.5),

где E - значение выражения. Тип идентификатора процедуры выдается описателем, который является первым символом соответствующего описания процедуры (см. п. 5.4.4).

4.3. Операторы перехода

4.3.1. *Синтаксис*

<оператор перехода>::= **go to** <именующее выражение>

4.3.2. *Примеры*

go to L8
go to EXIT [N+1]
go to TOWN[if Y<0 **then** N **else** N+1]

4.3.3. *Семантика*

Оператор перехода прерывает естественную последовательность действий, задаваемую порядком написания операторов, явно определяя своего преемника по значению именующего выражения. Та-

ким образом, следующим выполняемым оператором будет тот, который имеет это значение в качестве своей метки.

4.3.4. *Ограничение*

В силу правил локализации меток ни один оператор перехода не может извне вести к метке внутри блока. Однако оператор перехода может вести извне к метке внутри составного оператора или внутри условного оператора (см. пп. 4.5.4 и 4.6.6).

4.4. Пустые операторы

4.4.1. *Синтаксис*

<пустой оператор>::=<пусто>

4.4.2. *Примеры*

```
L:
begin . . . ; JOHN : end
```

4.4.3. *Семантика*

Пустой оператор не выполняет никакого действия. Он может служить для помещения метки.

4.5. Условные операторы

4.5.1. *Синтаксис*

<условие>::= **if** <логическое выражение> **then**

<безусловный оператор>::=<основной оператор>|<составной оператор>|<блок>

<оператор «если»>::=<условие> <безусловный оператор>

<условный оператор>::=<оператор «если»>|<оператор «если»> **else** <оператор>|<условие><оператор цикла>|<метка>:<условный оператор>

4.5.2. *Примеры*

```
if X>0 then N:=N+1
if V>U then W:Q:=N+M else go to R
if S<0 ∨ P<Q then AA:begin if Q<V then A:=V/S
  else Y:=2×A end
  else if V>S then A:=V—Q
  else if V>S—1 then go to ST
```

4.5.3. *Семантика*

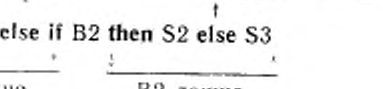
Условные операторы приводят к пропуску или выполнению некоторых операторов в зависимости от текущих значений указанных логических выражений. Согласно синтаксису возможны две различные формы условных операторов: укороченный условный оператор **if** B **then** S полный условный оператор **if** B **then** S1 **else** S2. Здесь B — логическое выражение, S — безусловный оператор или оператор цикла, S1 — безусловный оператор и S2 — оператор.

Если текущее значение логического выражения B есть **true**, то выполнение укороченного условного оператора сводится к выполнению оператора S, а выполнение полного условного оператора к

выполнению оператора S1. Если же текущее значение логического выражения B есть **false** , то в случае укороченного условного оператора действия продолжаются так, как будто он был пустым оператором, а в случае полного условного оператора его выполнение сводится к выполнению оператора S2. Во всех случаях преемник условного оператора определяется общими правилами, то есть так, как будто на месте условного оператора стоял один из операторов S, S1, S2 или пустой оператор в соответствии с тем или иным из описанных выше случаев.

В силу описанного действие ограничителя **else** в полном условном операторе можно охарактеризовать, сказав, что он определяет в качестве преемника оператора, за которым этот ограничитель следует, оператор, который надо выполнять за соответствующим полным условным оператором.

Для дальнейших пояснений используем следующую схему с очевидными обозначениями



4.5.4. *Переход внутрь условного оператора*

Результат работы оператора перехода, ведущего внутрь условного оператора, непосредственно следует из объясненного выше действия ограничителя **else.**

4.6. Операторы цикла

4.6.1. *Синтаксис*

<элемент списка цикла>::=<арифметическое выражение> | <арифметическое выражение> **step** <арифметическое выражение> **until** <арифметическое выражение> | <арифметическое выражение> **while** <логическое выражение>

<список цикла>::=<элемент списка цикла> | <список цикла>, <элемент списка цикла>

<заголовок цикла>::=**for** <идентификатор переменной>:=<список цикла> **do**

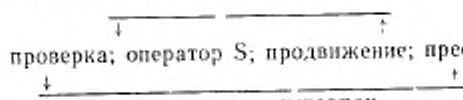
<оператор цикла>::=<заголовок цикла><оператор> | <метка>:<оператор цикла>

4.6.2. *Примеры*

```
for Q:=1 step S until N do A[Q]:=B[Q]
for K:=1, V1×2 while V1<N do
for J:=I+G, L, 1 step 1 until N, C+D do
                    A[K, J]:=B[K, J]
```

4.6.3. *Семантика*

Заголовок цикла заставляет стоящий за ним оператор S повторно выполняться нуль или более раз. Кроме того, он осуществляет последовательные присваивания значений переменной, управляемой данным заголовком, так называемому параметру цикла. Этот процесс может быть пояснен следующей схемой:



В этой схеме слова означают: «начало» — произвести первое присваивание в заголовке цикла; «продвижение» — произвести очередное присваивание в заголовке цикла; слово «проверка» определяет, было ли сделано последнее присваивание. Если оно сделано, то выполнение продолжается с преемника оператора цикла. В противном случае выполняется оператор, стоящий за заголовком цикла.

4.6.4. *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 трех разновидностей элементов списка цикла, и соответствующее выполнение оператора S определяются следующими правилами

4.6.4.1. Арифметическое выражение. Этот элемент задает только одно значение, а именно значение данного арифметического выражения, вычисленное непосредственно перед соответствующим выполнением оператора S.

4.6.4.2. Элемент типа арифметической прогрессии. Элемент, имеющий вид A **step** L **until** C, где A, B и C — арифметические выражения, задает порядок выполнения, который наиболее четко можно описать при помощи дополнительных операторов АЛГАМСа следующим образом:

```
        V:=A;
L1:if (V—(C))×SIGN(B)>0 then go to элемент исчерпан,
        оператор S;
        V:=V+(B);
        go to L1;
```

где V — параметр цикла и «элемент исчерпан» указывает на переход к вычислениям, соответствующим следующему элементу списка цикла, или (если данный элемент типа арифметической прогрессии стоит последним в списке) к следующему оператору программы.

4.6.4.3. Элемент типа пересчета. Порядок выполнения, определяемый элементом списка цикла вида A **while** B, где A — арифметическое выражение, B — логическое выражение, наиболее четко описывается при помощи дополнительных операторов АЛГАМСа следующим образом:

```
L3:V:=A;
if  | (B)then go to элемент исчерпан;
оператор S;
go to L3,
```

Обозначения те же, что и в п. 4.6.4.2.

4.6.5. *Значение параметра цикла после выхода*

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 вызван исчерпанием списка цикла, то значение параметра цикла после выхода не определено.

4.6.6. *Оператор перехода, ведущий в оператор цикла*

Результат действия оператора перехода, стоящего вне оператора цикла и обращающегося к метке внутри оператора цикла, не определен.

4.7. Операторы процедур

4.7.1. *Синтаксис*

<фактический параметр>::=<строка>|<выражение>|<идентификатор массива>|<идентификатор переключателя>|<идентификатор процедуры>

<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>

4.7.2. *Примеры*

след (A) порядок: (7) результат: (V)
транспонирование (W, V+1)
абсмакс (A, N, M, YY, I, K)
скалярное произведение (A[T, P, U], B[P], 10, P, Y)

Эти примеры соответствуют примерам, данным в п. 5.4.2.

4.7.3. *Семантика*

Оператор процедуры служит для обращения к выполнению тела процедуры (см. п. 5.4). Кроме случаев, когда тело процедуры имеет вид LIBRARY (<строка>), результат его выполнения будет эк-

вивалентен результату осуществления следующих действий в программе во время выполнения оператора процедуры.

4.7.3.1. Присваивание значений (вызов значением). Всем формальным параметрам, перечисленным в списке значений заголовка описания процедуры, присваиваются значения (см. п. 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 (см. п. 4.1.3), в котором делаются присваивания переменным, локальным в этом фиктивном блоке и имеющим типы, заданные соответствующими спецификациями (см. п. 5.4.5). В результате переменные, вызываемые значением, следует рассматривать как локальные в этом фиктивном блоке и нелокальные в теле процедуры (см. п. 5.4.3).

4.7.3.2. Замена наименований (вызов по наименованию). Любой формальный параметр, не перечисленный в списке значений, повсюду в теле процедуры заменяется на соответствующий фактический параметр, после того, как последний там, где это синтаксически возможно, заключен в круглые скобки. Возможность противоречий между идентификаторами, вставляемыми в тело процедуры в результате такого процесса, и идентификаторами, уже присутствующими в теле процедуры, устраняется соответствующими систематическими изменениями локальных идентификаторов, затронутых такими противоречиями.

4.7.3.3. Подстановка и выполнение тела процедуры. Тело процедуры, преобразованное как описано выше, помещается на место оператора процедуры и выполняется. Если обращение к процедуре производится извне области действия любой величины, нелокальной 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.

4.7.4. *Соответствие между фактическими и формальными 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 сопоставлением членов этих двух списков в одном и том же порядке.

4.7.5. *Ограничения*

Чтобы оператор процедуры был определен, очевидно, необходимо, чтобы действия над телом процедуры, определенные в пп. 4.7.3.1 и 4.7.3.2, приводили бы к правильному оператору в языке АЛГАМС.

Это накладывает на любой оператор процедуры ограничения, заключающиеся в том, что класс и тип каждого фактического параметра должен быть совместим с классом и типом соответствующего формального параметра. Некоторые важные частные случаи этого общего правила приведены ниже.

4.7.5.1. Если строка является фактическим параметром оператора процедуры или указателя функции, для которых соответствующее тело процедуры является оператором в смысле языка АЛГАМС (а не LIBRARY (<строка>)), то эту строку можно использовать в теле процедуры только как фактический параметр в дальнейших обращениях к процедурам. В конечном итоге строку можно использовать только в теле процедуры вида LIBRARY (<строка>) либо в соответствующих стандартных процедурах.

4.7.5.2. 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).

4.7.5.3. Формальному параметру, используемому в теле процедуры в качестве идентификатора массива, может соответствовать в качестве фактического параметра только идентификатор массива той же размерности. Кроме того, если формальный параметр вызывается значением, то локальный массив, возникающий в теле процедуры во время обращения, получает те же границы индексов, что и фактический массив.

4.7.5.4. Формальному параметру, вызываемому значением, не может, вообще говоря, соответствовать какой-либо идентификатор переключателя или идентификатор процедуры, или строка, так как последние не обладают значениями. (Исключение составляет идентификатор такой процедуры, описание которой имеет пустую совокупность формальных параметров (см. п. 5.4.1) и которая определяет значение указателя функции (см. п. 5.4.4). Такой идентификатор процедуры сам по себе является законченным выражением).

4.7.5.5. При вызове по наименованию класс и тип фактического параметра должны совпадать с классом и типом соответствующего формального параметра. Однако, если при выполнении процедуры не происходит присвоения значения этому параметру, то фактический параметр типа **integer** может соответствовать формальному параметру типа **real**.

4.7.5.6. Не может произойти обращение к какой-либо процедуре при выполнении операторов тела этой же самой процедуры или

при вычислении тех ее фактических параметров, которым соответствуют формальные параметры, вызываемые по наименованию, или при вычислении выражений, встречающихся в описаниях внутри этой же процедуры.

4.7.6. *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 того лишь, что их количество должно быть одинаковым. Таким образом, вся информация, которая вносится употреблением сложных ограничителей, полностью избыточна.

5. ОПИСАНИЯ

Описания служат для определения некоторых свойств величин, используемых в программе, и связи этих величин с идентификаторами. Описание идентификатора имеет силу только в одном блоке. Вне этого блока то же идентификатор можно использовать для других целей (см. п. 4.1.3).

В процессе работы это влечет за собой следующее: с момента входа в блок (через **begin**, так как внутренние метки локальны и, следовательно, недостижимы извне) все идентификаторы, описанные в блоке, приобретают смысл, вытекающий из природы данных описаний. 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 описаны в блоке, сохраняют свой прежний смысл.

В момент выхода из блока (через **end** или оператор перехода) все идентификаторы, которые описаны в блоке, теряют свой локальный смысл.

В программе все идентификаторы простых переменных, массивов, переключателей и процедур (кроме стандартных процедур и функций, (см. разд. 6) следует описывать при помощи описаний. Ни один идентификатор в блоке не должен быть описан более чем один раз. Идентификатор, связанный с величиной некоторым описанием, не может более одного раза встретиться, обозначая эту величину, между **begin** блока, в начале которого стоит это описание, и точкой с запятой, которой оканчивается это описание, за исключением случая, когда имеет место появления идентификатора процедуры в списке левой части оператора присваивания в смысле п. 5.4.4.

Синтаксис

<описание>::=<описание типа>|<описание массивов>|
<описание переключателя>|<описание процедуры>

5.1. Описание типа

5.1.1. *Синтаксис*

<список типа>::=<простая переменная>|<список типа>, <простая переменная>

<тип>::=**real**|**integer**|**Boolean**

<описание типа>::=<тип><список типа>

5.1.2. *Примеры*

integer P, Q, S
Boolean ACRYL, N

5.1.3. *Семантика*

Описания типа служат для указания того, что некоторые идентификаторы представляют простые переменные данного типа. Переменные, которым описанием дан тип **real**, могут принимать только положительные и отрицательные значения, включая нуль. Переменные, которым описанием дан тип **integer**, могут принимать только целые значения. Переменные, которым описанием дан тип **Boolean**, могут принимать только значения **true** и **false**.

В арифметических выражениях любая позиция, занятая переменной типа **real**, может быть занята и переменной типа **integer**.

5.2. Описания массивов

5.2.1. *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>

<список идентификаторов массивов>::=<идентификатор массива>|<список идентификаторов массивов>, <идентификатор массива>

<сегмент массивов>::=<список идентификаторов массивов>[<список граничных пар>]

<список массивов>::=<сегмент массивов>|<список массивов>, <сегмент массивов>

<описание массивов>::=**array** <список массивов>|<тип> **array**<список массивов>

5.2.2. *Примеры*

array A, B, C [7:N, 2:M], S [−2:10]
integer array A[**if** C<0 **then** 2 **else** 1:20]
real array Q[− 7:− 1]

5.2.3. *Семантика*

В описании массивов определяется, что один или несколько идентификаторов представляют многомерные массивы переменных с индексами, и задается размерность этих массивов, границы индексов и типы переменных.

5.2.3.1. Границы индексов. Границы индексов любого массива задаются в первых индексных скобках, следующих за идентификатором данного массива, в виде списка граничных пар. Каждый член этого списка задает нижнюю и верхнюю границы индекса в виде двух арифметических выражений, разделенных ограничителем. Список граничных пар задает границы всех индексов в порядке их перечисления слева направо.

5.2.3.2. Размерности. Размерности определяются как число членов в списках граничных пар.

5.2.3.3. Типы. Все массивы, данные в одном описании, имеют один и тот же заданный для них тип. Если описатель типа отсутствует, то подразумевается тип **real**.

5.2.4. *Выражения для нижних и верхних границ*

5.2.4.1. Значения этих выражений вычисляются аналогично значениям индексных выражений (см. п. 3.1.4.2).

5.2.4.2. Эти выражения могут зависеть только от переменных и процедур, не локальных в том блоке, для которого имеет силу данное описание массивов. Из этого следует, что в самом внешнем блоке программы могут быть описаны массивы только с постоянными границами.

5.2.4.3. Массив определен только в том случае, когда значения всех верхних границ индексов не меньше значений соответствующих нижних границ.

5.2.4.4. Значения выражений для границ вычисляются один раз при каждом входе в блок.

5.2.5. *Идентичность переменных с индексами*

Идентичность переменных с индексами не связана с границами индексов, задаваемыми в описании массивов. Однако значения соответствующих переменных с индексами в любой момент времени определены только для той части этих переменных, у которых индексы находятся в пределах границ индексов, вычисленных в последний раз.

5.2.6. *Внешние массивы*

Массивы, идентификаторы которых являются внешними идентификаторами (идентификаторами, начинающимися с букв EX), могут быть размещены транслятором во внешней памяти. Внешний идентификатор массива, так же как и переменная с индексами, имеющая внешний идентификатор в качестве идентификатора массива, может употребляться только как фактический параметр. Доступ к элементам внешних массивов возможен только через оператор процедуры COPY (см. п. 6.2).

5.3. Описания переключателей

5.3.1. *Синтаксис*

<переключательный список>::=<метка>|<переключательный список>, <метка>

<описание переключателя>::=**switch**<идентификатор переключателя>::=<переключательный список>.

5.3.2. *Пример*

switch Q:=P1, W

5.3.3. *Семантика*

Описание переключателя задает значения, соответствующие идентификатору переключателя. Эти значения задаются как метки, входящие в переключательный список, с каждой из которых сопоставляется целое положительное число 1, 2, . . . , получаемое пересчетом элементов списка слева направо. Значение указателя переключателя, соответствующее заданному значению индексного выражения (см. п. 3.5), есть метка в переключательном списке, имеющая заданное значение своим порядковым номером.

5.3.4. *Влияние областей действия*

Если указатель переключателя встречается вне области действия метки в переключательном списке и вычисление указателя переключателя выбирает эту метку, то возможная коллизия между идентификатором, использованным для обозначения этой метки, и идентификатором, описание которого действует на месте указателя переключателя, устраняется подходящим изменением этого последнего идентификатора.

5.4. Описания процедур

5.4.1. *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>::=**value**<список идентификаторов>;|<пусто>

<спецификация>::=**string**|<тип>|**array**|<тип>**array**|**label**|**switch**|**procedure**|<тип>**procedure**

<совокупность спецификаций>::=<пусто><спецификация><список идентификаторов>;|<совокупность спецификаций><спецификация><список идентификаторов>;

<заголовок процедуры>::=<идентификатор процедуры><совокупность формальных параметров>;<список значений><совокупность спецификаций>

<тело процедуры>::=<оператор>|LIBRARY (<строка>)

<описание процедуры>::=**procedure**<заголовок процедуры><тело процедуры>|<тип>**procedure**<заголовок процедуры><тело процедуры>

5.4.2. *Примеры*

```
procedure след (A) порядок: (N) результат: (S); value N;
array A; integer N; real S;
begin integer K;
S:=0;
for K:=1 step 1 until N do S:=S+A[K, K]
end
procedure транспонирование (A) порядок: (N); value N;
array A; integer N;
begin real W; integer I, K;
for I:=1 step 1 until N do
   for K:=I+1 step 1 until N do
   begin W:=A[I, K];
         A[I, K]:=A[K, I];
         A[K, I]:=W end
end транспонирования
integer procedure шаг (U); real U;
шаг:=if 0≤U∧U<1 then 1 else 0
procedure абсмакс (A) размер: (N, M) результат: (Y) индексы:
          (I, K);
comment наибольшая из абсолютных величин элементов матри-
          цы A размером N на M передается в Y, а индексы
          этого элемента передаются в I и K;
array A; integer N, M, I, K; real Y;
begin integer P, Q;
Y:=0;
for P:=1 step 1 until N do for Q:=1 step 1 until M do
if ABS (A[P, Q])>Y then begin Y:=ABS (A[P, Q]);
I:=P; K:=Q end end абсмакс
procedure скалярное произведение (A, B )порядок: (K, P)
результат: (Y); value K;
integer K, P; real Y, A, B;
begin real S; S:=0;
for P:=1 step 1 until K do S:=S+A×B;
Y:=S
end скалярного произведения.
```

5.4.3. *Семантика*

Описание процедуры служит для задания процедуры, связанной с идентификатором процедуры. Главной составной частью описания процедуры является оператор или LIBRARY (<строка>), называемые телом процедуры, к которому может быть произведено обращение посредством указателей функций и (или) операторов процедуры из других 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 (см. пп. 3.2 и 4.7) формальным параметрам в теле процедуры будут присвоены значения фактических параметров, или же они будут заменены фактическими параметрами. Если идентификатор формального параметра заново локализован внутри тела процедуры (как это указано в п. 4.1.3), то ему придается тем самым локальный смысл и фактические параметры, которые соответствуют такому формальному параметру, недоступны во всей области действия этого внутреннего локального идентификатора.

Идентификаторы, нелокальные в теле процедуры, могут быть локальными в блоке, в начале которого находится описание данной процедуры. Ни один идентификатор не может встречаться более одного раза в списке формальных параметров. Идентификатор LIBRARY, если ему не придано другого смысла, употребляется для указания того, что тело процедуры является кодом. В этом случае строка является названием кода (то есть библиотечной программы). Результат обращения к этой процедуре определяется фактическими параметрами и библиотечной подпрограммой, название которой помещено в строке. Тело процедуры всегда действует подобно блоку (см. п. 4.1.3). Следовательно, область действия метки, помечающей оператор внутри тела или само тело, никогда не может распространяться за тело процедуры.

5.4.4. *Значения указателей функций*

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 процедуры. Последнее значение, присвоенное таким образом, используется для дальнейшего вычисления выражения, в котором встречается указатель функции. Указатели функций в программе должны быть такими, чтобы все возможные использования этих указателей функций в форме операторов процедуры были бы эквивалентны пустым операторам.

5.4.5. *Спецификации*

В заголовок процедуры включается совокупность спецификаций, задающая с помощью очевидных обозначений информацию о классах и типах формальных параметров. В эту часть ни один формальный параметр нельзя вносить более одного раза. Каждый формальный параметр должен быть специфицирован.

6. СТАНДАРТНЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

В каждой программе на языке АЛГАМС подразумеваются описанными некоторые стандартные процедуры и функции, перечисленные ниже. Естественно, что обращения к ним возможны только внутри тех блоков, где их идентификаторы не определены в другом смысле.

Списки стандартных процедур и функций в дальнейшем могут быть расширены.

6.1. Стандартные функции

ABS(E) для модуля (абсолютной величины) значения выражения E

SIGN(E) для знака значения E(+1 для E>0, 0 для E=0, −1 для E<0)

SQRT(E) для квадратного корня из значения E

LN(E) для натурального логарифма значения E

EXP(E) для экспоненциальной функции значения E (e^E)

SIN(E) для синуса значения E

COS(E) для косинуса значения E

TAN(E) для тангенса значения E

ARCSIN(E) для главного значения арксинуса значения E

ARCCOS(E) для главного значения арккосинуса значения E

ARCTAN(E) для главного значения арктангенса значения E

ARC(E1, E2) для полярного угла точки с координатами E1, E2; значение берется из интервала 0 ARC<2π

ENTIER(E) для целой части значения E

DIV(E1, E2) для SIGN((E1)/(E2))×ENTIER(ABS((E1)/(E2)))

RES(E1, E2) для (E1)−DIV(E1, E2)×(E2)

MAX(E1, E2, . . . , EN) для наибольшего из значений выражений E1, E2, . . . , EN

MIN(E1, E2, . . . , EN) для наименьшего из значений выражений E1, E2, . . . , EN.

Функции DIV и RES определены для аргументов типа **integer** и принимают значения типа **integer**. Остальные функции определены как для аргументов типа **real**, так и для аргументов типа **integer**, и принимают значения типа **real**, кроме функций SIGN и ENTIER, которые принимают значения типа **integer**.

6.2. Процедура обмена

6.2.1. *Синтаксис*

<оператор обмена>::=COPY (<переменная с индексами>, <идентификатор массива>)|COPY (<идентификатор массива>, <переменная с индексами>)

Примеры

COPY(EXA[I+1,0], A1)

COPY(B, EXT[K])

6.2.2. *Семантика*

Процедура COPY служит для обмена между внутренними и внешними массивами. Элементы массивов считаются упорядоченными лексикографически по индексам. Переменная с индексами всегда принадлежит внешнему массиву и задает начальный адрес обмена во внешней памяти. Идентификатор массива всегда внутренний. Первый элемент этого массива задает начальный адрес обмена во внутренней памяти. Количество передаваемых элементов динамически определяется описанием внутреннего массива. Передача данных идет от первого фактического параметра ко второму. Типы внешнего и внутреннего массивов должны совпадать.

6.3. Процедуры вывода

6.3.1. *Синтаксис*

<оператор вывода>::=<оператор вывода чисел>|<оператор вывода логических значений>|<оператор вывода текста>|<оператор размещения>

<оператор вывода чисел>::=OUTPUT (<канал>, <числовой формат>, <список объектов вывода>)|OUTPUT (<канал>, <переменная с индексами>, <список объектов вывода>)

<оператор вывода логических значений>::=OUTPUT (<канал>, <логический формат>, <список объектов вывода>)|OUTPUT (<канал>, <переменная с индексами>, <список объектов вывода>)

<список объектов вывода>::=<объект вывода>|<список объектов вывода>, <объект вывода>

<объект вывода>::=<выражение>|<идентификатор массива>

<оператор вывода текста>::=OUTPUT (<канал>, 'T', <список текстовых объектов вывода>)|OUTPUT (<канал>, <переменная с индексами>, <список текстовых объектов вывода>)

<список текстовых объектов вывода>::=<текстовый объект вывода>|<список текстовых объектов вывода>,<текстовый объект вывода>

<текстовый объект вывода>::=<строка>|<переменная с индексами>

<оператор размещения>::=OUTPUT (<канал>, <формат размещения>)|OUTPUT (<канал>,<формат размещения>,<арифметическое выражение>)|OUTPUT (<канал>, <переменная с индексами>)|OUTPUT (<канал>, <переменная с индексами>, <арифметическое выражение>)

<канал>::=<арифметическое выражение>.

6.3.2. *Семантика операторов вывода*

Операторы вывода задают вывод числовых, логических или тек-

стовых данных через канал, номер которого определяется первым фактическим параметром. Второй фактический параметр определяет формат вывода (см. пп. 6.3.3—6.3.6), а все следующие – объекты вывода. Исключение представляет собой оператор размещения (см. п. 6.3.6), который не имеет объектов вывода. Если второй фактический параметр есть переменная с индексами, то она указывает на элемент массива, начиная с которого располагаются целые числа, соответствующие последовательным символам формата в смысле процедуры TEXT (см. п. 6.6).

6.3.3. *Оператор вывода чисел*

Оператор вывода чисел задает вывод значений целых и действительных выражений и массивов, перечисленных в списке объектов вывода. Все числа выводятся в одном и том же формате, определяемом вторым фактическим параметром оператора.

6.3.3.1. Синтаксис числового формата

<повторитель>::=<целое без знака>|<пусто>

<B-часть>::=<повторитель>B|<B-часть><повторитель>B|<пусто>

<D-часть>::=<повторитель>D|<D-часть><повторитель>D|<D-часть><B-часть>|<пусто>

<знаковая часть>::=+|-|<пусто>

<целый формат>::=<B-часть><знаковая часть><D-часть>

<дробный формат>::=<целый формат>|<целый формат>.<D-часть>

<экспоненциальный формат>::=<дробный формат>$_{10}$<целый формат>

<числовой формат>::='E'|'Y'|'Z'|'ZI'|'ZI'|
'E<экспоненциальный формат>'|
'Y<дробный формат>'
'Z<дробный формат>'

6.3.3.2. Семантика числового формата. Числовой формат определяет вид, в котором выводятся числа (в десятичной системе счисления). Числа перед выводом округляются. Буква D означает десятичную цифру, буква B – пропуск (интервал) между выводимыми символами, точка – десятичную точку.

Буква Z означает замену незначащих нулей пропусками (так называемое подавление нулей).

Если при этом перед десятичной точкой нет значащих цифр (или число, выводимое в целом формате, оказывается равным нулю), то сохраняется один нуль. Знак помещается на место последнего подавленного нуля выводимого числа. Буква Y означает вывод без подавления нулей, то есть вывод такого количества цифр, какое указано в формате. Буква E означает вывод числа в экспоненциальной форме с отличной от нуля первой цифрой. Знак + означа-

ет вывод знака числа во всех случаях, знак — означает вывод знака только у отрицательных чисел.

Если знаковая часть пустая, то знак не печатается (выводится абсолютное значение числа). Конструкция <повторитель> B или <повторитель> D эквивалентна соответствующее число раз повторенной букве B или D. Так, например, 5B эквивалентно BBBBB, 4D.3D эквивалентно DDDD.DDD. Числовой формат вида 'E', 'Y', 'Z', 'YI' или 'Z' означает вывод чисел в стандартной форме. Для различных машин стандартные формы могут быть разными; они зависят от особенностей соответствующих выводных устройств. Буквы Z, Y и E в числовых форматах 'Z', 'Y' и 'E' имеют указанный выше смысл. 'YI' означает вывод целых чисел в стандартной форме без подавления нулей, 'ZI' означает вывод целых чисел в стандартной форме с подавлением нулей.

6.3.4. *Оператор вывода логических значений*

Оператор вывода логических значений задает вывод значений булевских выражений и массивов, перечисленных в списке объектов вывода. Все значения выдаются в одном и том же формате, определяемом вторым фактическим параметром оператора.

6.3.4.1. Синтаксис логического формата

<F - - часть>::=5F | F

<логический формат>::='L' | 'L' <B - часть><F - часть><B часть>'

6.3.4.2. Семантика логического формата. 5F означает вывод значения логического выражения в виде FALSE или TRUE, причем при выводе TRUE после этого слова делается один пропуск. F означает вывод значения логического выражения, соответственно, в виде F или T. 'L' означает вывод в стандартной форме.

6.3.5. *Оператор вывода текста*

Этот оператор задает вывод текстов, определяемых списком текстовых объектов вывода. Если текстовый объект есть строка, то выдается 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 в смысле процедуры TEXT (см. п. 6.6). Предполагается, что указанный элемент массива соответствует открывающей кавычке, которая не будет выведена. Конец строки определяется первым элементом массива, соответствующим закрывающей кавычке, которая также не выводится. Элементы массива, имеющие значение нуль, исключаются из рассмотрения, то есть нуль трактуется как отсутствие информации. Отличные от нуля значения, не предусмотренные в таблице из п. 6.6.2, трактуются как ошибочные. В выводимых строках двоеточие используется

только для изменения смысла следующего за ним символа. А именно,

комбинация: В задает пробел при выводе
комбинация: / задает переход к началу следующей строки
комбинация: × задает переход к началу следующей страницы
комбинация: 1 задает вывод символа'
комбинация: 2 задает вывод символа'
комбинация: 3 задает вывод символа:

6.3.6. *Оператор размещения*

6.3.6.1. Синтаксис формата размещения.

<знак размещения>::=B | / | ×

<указатель размещения>::=<повторитель><знак размещения> | <указатель размещения><повторитель><знак размещения>

<формат размещения<::='<указатель размещения>'

6.3.6.2. Семантика. В формате размещения знак В задает пробел, знак / задает переход к началу следующей строки, знак × задает переход к началу следующей страницы.

Все эти операции выполняются в том порядке, как они заданы в формате (слева направо). Если перед какой-либо операцией помещен повторитель, то она выполняется соответствующее количество раз.

Третий фактический параметр оператора размещения (если он имеется) определяет число повторений всей совокупности операций, задаваемых форматом размещения.

6.4. Оператор разметки

6.4.1. *Синтаксис*

<оператор разметки>::=MARG (<канал>, <арифметическое выражение>, <арифметическое выражение>, <арифметическое выражение>, <арифметическое выражение>, <арифметическое выражение>) | MARG (<канал>, <арифметическое выражение>, <арифметическое выражение>, <арифметическое выражение>, <арифметическое выражение>, <арифметическое выражение>, <арифметическое выражение>) | LMARG(<канал>, <переменная>, <переменная>, <переменная>, <переменная>, <переменная>)

6.4.2. *Семантика*

С каждым каналом вывода связан некоторый стандарт разметки выводимой информации (для каждой машины свой). Эта разметка характеризуется шестью величинами: размер левого поля, длина печатаемой строки, размер правого поля, размер верхнего

поля, число печатаемых на одной странице строк, размер нижнего поля (см. чертеж). При выполнении оператора LMARG перечисленным в нем переменным присваиваются значения указанных выше шести величин в том порядке, в каком они выписаны. Для изменения указанных выше стандартных размеров служит оператор процедуры MARG. После выполнения этого оператора шесть размеров, определяющих вид страницы, делаются равными значениям фактических параметров со второго по седьмой в таком же порядке, как и для процедуры LMARG.



Восьмой фактический параметр процедуры MARG задает начальное значение номера страницы. Если этого параметра нет, то нумерация страниц начинается с 1.

6.5. Процедуры ввода

6.5.1. *Синтаксис*

<оператор ввода> : : = INPUT (<канал>, <список объектов ввода>)

<список объектов ввода> : : = <объект ввода> | <список объектов ввода>, <объект ввода>

<объект ввода> : : = <идентификатор массива> | <простая переменная> | <переменная с индексом>

6.5.2. *Семантика операторов ввода*

Операторы ввода задают ввод числовых, логических или текстовых данных через канал, номер которого определяется первым фактическим параметром. Форма, в которой должны быть подготовлены эти данные, определяется в п. 6.5.3. Каждому объекту ввода должна соответствовать одна группа данных (см. п 6.5.3). Простым переменным и идентификаторам массивов соответствуют группы числовых или логических данных, переменным с индексами соответствуют группы текстовых данных. Если количество данных в группе превосходит динамически определяемые описаниями размеры объектов ввода, то результат выполнения оператора ввода не определен.

При вводе текста последовательным элементам массива, начиная с указанного в объекте ввода, присваиваются целые значения, соответствующие последовательным символам вводимой строки (включая внешние кавычки) в смысле процедуры TEXT (см. п. 6.6).

6.5.3. *Группы данных*

<элемент числового ввода> : : = <число> | <комментарий>

<число>

<список элементов числового ввода>::=<элемент числового ввода>|<список элементов числового ввода>,<элемент числового ввода>

<группа числовых данных>::=<список элементов числового ввода>;

<элемент логического ввода>::=<логическое значение>|<комментарий><логическое значение>

<список элементов логического ввода>::=<элемент логического ввода>|<список элементов логического ввода>,<элемент логического ввода>

<группа логических данных>::=<список элементов логического ввода>;

<открытый комментарий>::=<буква>|<открытый комментарий><любой символ, кроме символа «:», символа «=» или «:=»>

<комментарий>::=<открытый комментарий>:|<открытый комментарий>=|<открытый комментарий>:=

<группа текстовых данных>::=<строка>

6.6. Процедура TEXT

6.6.1. *Синтаксис*

<оператор текст>::=TEXT (<строка>, <переменная с индексами>)

6.6.2. *Семантика*

Процедура TEXT присваивает последовательным элементам массива, начиная с элемента, указанного вторым фактическим параметром, целые значения, соответствующие последовательным символам строки, являющейся первым фактическим параметром (включая внешние кавычки).

Соответствие между символами строки и целыми числами определяется следующей таблицей.

A	1	K	11	U	21	4	81	×	136		146
B	2	L	12	V	22	5	83	/	137	∨	147
C	3	M	13	W	23	6	86	^	138	∧	148
D	4	N	14	X	24	7	87	≤	139	¬	149
E	5	O	15	Y	25	8	88	≥	140	go to	200
F	6	P	16	Z	26	9	89	≠	141	if	201
G	7	Q	17	0	80	true	130	≡	142	then	202
H	8	R	18	1	81	false	131	⊃	143	else	203
I	9	S	19	2	82	≡	134	≡	144	for	204
J	10	T	20	3	83	⊃	135	≡	145	do	205

'	210	comment	220	integer	230
.	211	(	221	real	231
10	212	)	222	array	232
:	213	[	223	switch	233
;	214	]	224	procedure	234
: =	215	'	225	string	235
⎵	216	,	226	label	236
step	217	begin	227	value	237
until	218	end	228		
while	219	Boolean	229		

Представление основных символов языка АЛГАМС через символы по ГОСТ 19767—74 приведено в справочном приложении 3.

ПРИЛОЖЕНИЕ 1
Справочное

АЛФАВИТНЫЙ УКАЗАТЕЛЬ ОПРЕДЕЛЯЕМЫХ ПОНЯТИЙ СИНТАКСИЧЕСКИХ ЕДИНИЦ

Указатель разделен на две части: I – указатель основных символов и II – указатель металингвистических переменных и терминов, употребляемых в тексте. Все ссылки даются на номера разделов. Ссылки разбиты на три группы:

опр — ссылка, стоящая за сокращением «опр», отсылает к соответствующему синтаксическому определению,

синт — ссылки, стоящие за сокращением «синт», отсылают к вхождению в металингвистическую формулу. Ссылки, перечисленные в группе «опр», не повторяются,

текст — ссылки, стоящие за словом «текст», отсылают к определению, данному в тексте.

При составлении указателя номера не принимались во внимание

I. УКАЗАТЕЛЬ ОСНОВНЫХ СИМВОЛОВ

+	см. плюс
−	см. минус
×	см. умножение
/	см. деление
↑	см. возведение в степень
<, ≤, =, ≠, ≥, >, ≡, ⊃, ∨, ∧, ¬	см. <знак операции отношения> или <знак логической операции>
,	см. запятая
.	см. десятичная точка
10	см. десять
:	см. двоеточие
;	см. точка с запятой
:=	см. двоеточие равенство
_	см. пробел
( )	см. скобки
' '	см. видовые скобки
" "	см. кавычки для строк
array	синт пп. 2.3, 5.2.1, 5.4.1
begin	синт пп. 2.3, 4.1.1.
Boolean	синт пп. 2.3, 5.1.1. текст п. 5.1.3
comment	синт п. 2.3
do	синт пп. 2.3, 4.6.1
else	синт пп. 2.3, 3.3.1, 3.4.1, 4.5.1, текст п. 4.5.3
end	синт пп. 2.3, 4.1.1
false	синт п. 2.2.2.
for	синт пп. 2.3, 4.6.1
go to	синт пп. 2.3, 4.3.1
if	синт пп. 2.3, 3.3.1, 4.5.1
integer	синт пп. 2.3, 5.1.1, текст п. 5.1.3
label	синт пп. 2.3, 5.4.1
procedure	синт пп. 2.3, 5.4.1
real	синт пп. 2.3, 5.1.1, текст п. 5.1.3
step	синт пп. 2.3, 4.6.1, текст п. 4.6.4.2
string	синт пп. 2.3, 5.4.1

switch синт пп. 2.3, 5.3.1, 5.4.1
then синт пп. 2.3, 3.3.1, 4.5.1
true синт п. 2.2.2
until синт пп. 2.3, 4.6.1, текст п. 4.6.4.2
value синт пп. 2.3, 5.4.1
while синт пп. 2.3, 4.6.1, текст п. 4.6.4.3

II. УКАЗАТЕЛЬ МЕТАЛИНГВИСТИЧЕСКИХ ПЕРЕМЕННЫХ И ТЕРМИНОВ, УПОТРЕБЛЯЕМЫХ В ТЕКСТЕ

алфавит текст п. 2.1
арифметический — текст п. 3.3.6
<арифметическое выражение> — опр п. 3.3.1, синт пп. 3.3.1.1, 4.2.1, 4.6.1, 5.2.1, 6.3.1, 6.4.1, текст п. 3.3.3
<безусловный оператор> — опр пп. 4.1.1, 4.5.1
<блок> — опр 4.1.1, синт п. 4.5.1, разд. 1, п. 4.1.3,5
<буква> опр 2.1, синт пп. 2, 2.4.1, 3.1.1, 3.2.1, 3.5.1, 4.7.1, 6.5.3
величина — текст п. 2.7
<верхняя граница> — опр п. 5.2.1, текст п. 5.2.4
<внешний идентификатор> — опр п. 3.1.1, текст п. 3.2.6
возведение в степень ↑ — синт. пп. 2.3, 3.3.1, текст п. 3.3.4.3
<вторичное логическое выражение> — опр п. 3.4.1
<выражение> — опр 3, синт пп. 3.2.1, 4.7.1, 6.3.1, разд. 3
границы индексов — текст п. 5.2.3.1
<граничная пара> — опр п. 5.2.1
<группа логических данных> — опр п. 6.5.3, текст п. 6.5.2
<группа текстовых данных> — опр п. 6.5.3, текст п. 6.5.2
<группа числовых данных> — опр п. 6.5.3 текст п. 6.5.2
двоеточие : — синт пп. 2.3, 3.2.1, 4.1.1, 4.5.1, 4.6.1, 4.7.1, 5.2.1, 6.5.3.
двоеточие, равенство: := — синт пп. 2.3, 4.2.1, 4.6.1, 5.3.1, 6.5.3
деление / — синт п.п. 2.3, 3.3.1, текст п. 3.3.4.2
десятичная точка — синт пп. 2.3, 2.5.1, 6.3.3.1
<десятичное число> — опр п. 2.5.1, текст п. 2.5.3
десять$_{10}$ — синт пп. 2.3, 2.5.1, 6.3.3.1
<дробный формат> — опр п. 6.3.3.1, текст п. 6.3.3.2
<заголовок процедуры> — опр п. 5.4.1, текст п. 5.4.3
<заголовок цикла> — опр п. 4.6.1, текст п. 4.6.3
запятая, — синт пп. 2.3, 3.1.1, 3.2.1, 4.6.1, 4.7.1, 5.1.1, 5.2.1, 5.3.1, 5.4.1, 6.2.1, 6.3.1, 6.4.1, 6.5.1, 6.5.3, 6.6.1
<знак арифметической операции> — опр п. 2.3 текст п. 3.3.4
<знак логической операции> — опр п. 2.3, синт п. 3.4.1, текст п. 3.4.5
<знак операции> — опр п. 2.3
<знак операции отношения> опр пп. 2.3, 3.4.1
<знак операции следования> — опр п. 2.3
<знак операции типа сложения> — опр п. 3.3.1
<знак операции типа умножения> — опр п. 3.3.1
<знак размещения> — опр п. 6.3.6.1, текст п. 6.3.6.2
<знаковая часть> опр п. 6.3.3.1, текст п. 6.3.3.2
значение — текст пп. 2.8, 3.3.3
<идентификатор> — опр п. 2.4.1, синт пп. 3.1.1, 3.2.1, 5.4.1, текст п. 2.4.3
<идентификатор массива> — опр п. 3.1.1, синт пп. 3.2.1, 4.7.1, 5.2.1, 6.2.1, 6.3.1, 6.5.1
<идентификатор переменной> — опр п. 3.1.1, синт п. 4.6.1
<идентификатор переключателя> — опр п. 3.5.1, синт п. 3.2.1, 4.7.1, 5.3.1
<идентификатор процедуры> — опр п. 3.2.1, синт пп. 4.2.1, 4.7.1, 5.4.1, текст п. 4.7.5.4

<идентификатор части> — опр п. 3.5.1, синт п. 4.1.1, текст п. 4.1.3
<именующее выражение> — опр п. 3.5.1, синт разд. 3, п. 4.3.1, текст п. 3.5.3
<импликация> — опр п. 3.4.1
индекс — текст п. 3.1.4.1
<индексное выражение> — опр п. 3.1.1, синт п. 3.5.1
индексные скобки [] — синт пп. 2.3, 3.1.1, 3.5.1, 5.2.1
кавычки для строк ' ' — синт пп. 2.3, 2.6.1, 6.3.1, 6.3.3.1, 6.3.4.1, 6.3.6.1, текст п. 2.6.3
<канал> — опр п. 6.3.1, синт пп. 6.4.1, 6.5.1, текст пп. 6.3.2, 6.4.2, 6.5.2
<комментарий> — опр п. 6.5.3
<левая часть> — опр п. 4.2.1
<логический одночлен> — опр п. 3.4.1
<логический терм> — опр п. 3.4.1
<логический формат> — опр п. 6.3.4.1, синт п. 6.3.1, текст п. 6.3.4.2
<логическое выражение> — опр п. 3.4.1, синт пп. 3, 3.3.1, 4.2.1, 4.5.1, 4.6.1, текст п. 3.4.3
<логическое значение> — опр п. 2.2.2, синт пп. 2, 3.4.1, 6.5.3
локальный — текст п. 4.1.3
массив — текст п. 3.1.4.1
<метка> — опр п. 3.5.1, синт пп. 4.1.1, 4.5.1, 4.6.1, 5.3.1, текст разд. 1, п. 4.1.3
минус — — синт пп. 2.3, 2.5.1, 3.3.1, 6.3.3.1, текст п. 3.3.4.1
<множитель> — опр п. 3.3.1
<начало блока> — опр п. 4.1.1
нелокальный — текст п. 4.1.3
<непомеченный блок> — опр п. 4.1.1
<непомеченный основной оператор> — опр п. 4.1.1
<непомеченный составной> — опр п. 4.1.1
<нижняя граница> — опр п. 5.2.1, текст п. 5.2.4
область действия — текст п. 2.7
<объект ввода> — опр п. 6.5.1, текст п. 6.5.2
<объект вывода> опр п. 6.3.1, текст п. 6.3.2
<ограничитель> — опр п. 2.3, синт п. 2
<ограничитель параметра> — опр пп. 3.2.1, 4.7.1, синт п. 5.4.1, текст п. 4.7.6
<оператор> — опр п. 4.1.1, синт пп. 4.5.1, 4.6.1, 5.4.1, текст разд. 4
<оператор ввода> — опр п. 6.5.1, текст п. 6.5.2
<оператор вывода> — опр п. 6.3.1, текст п. 6.3.2
<оператор вывода логических значений> — опр п. 6.3.1, текст пп. 6.3.2, 6.3.4
<оператор вывода текста> — опр п. 6.3.1, текст п. 6.3.2, 6.3.5
<оператор вывода чисел> — опр п. 6.3.1, текст пп. 6.3.2, 6.3.3
<оператор «если»> — опр п. 4.5.1, текст п. 4.5.3
<оператор обмена> — опр п. 6.2.1, текст п. 6.2.2
<оператор перехода> — опр п. 4.3.1, синт п. 4.1.1, текст п. 4.3.3
<оператор присваивания> — опр п. 4.2.1, синт п. 4.1.1, текст разд. 1, п. 4.2.3
<оператор процедуры> — опр п. 4.7.1, синт п. 4.1.1, текст п. 4.7.3
<оператор разметки> — опр п. 6.4.1, текст п. 6.4.2
<оператор размещения> — опр п. 6.3.1, текст пп. 6.3.2, 6.3.6
<оператор текст> опр п. 6.6.1, текст п. 6.6.2
<оператор цикла> — опр п. 4.6.1, синт пп. 4.1.1, 4.5.1, текст п. 4.6
операторные скобки — см. **begin end**
<описание> — опр разд. 5, синт п. 4.1.1, текст разд. 1, 5
<описание массивов> — опр п. 5.2.1, синт разд. 5, текст п. 5.2.3
<описание переключателя> — опр п. 5.3.1, синт разд. 5, текст п. 5.3.3
<описание процедуры> — опр п. 5.4.1, синт разд. 5, текст п. 5.4.3
<описание типа> — опр п. 5.1.1, синт разд. 5, текст п. 5.1.3
<описатель> — опр п. 2.3
<основной оператор> — опр п. 4.1.1, синт п. 4.5.1
<основной символ> — опр разд. 2

<открытый комментарий> — опр п. 6.5.3
<отношение> — опр п. 3.4.1, текст п. 3.4.5
<первичное выражение> — опр п. 3.3.1
<первичное логическое выражение> — опр п. 3.4.1
<переключательный список> — опр п. 5.3.1
<переменная> — опр п. 3.1.1, синт пп. 3.3.1, 3.4.1, 4.2.1, 6.4.1, текст п. 3.1.3
<переменная с индексами> — опр п. 3.1.1, синт пп. 6.2.1, 6.3.1, 6.5.1, 6.6.1, текст п. 3.1.4.1
плюс + — синт пп. 2.3, 2.5.1, 3.3.1, 6.3.3.1, текст п. 3.3.4.1
<повторитель> — опр п. 6.3.3.1, синт п. 6.3.6.1, текст п. 6.3.3.2
<порядок> — опр п. 2.5.1, текст п. 2.5.3
правила примечаний — текст п. 2.3
<правильная дробь> — опр п. 2.5.1
преемник — текст разд. 4.
пробел ␣ — синт п. 2.3, текст п. 2.3, разд. 2, п. 6.3
<программа> — опр п. 4.1.1, текст разд. 1
<простая переменная> — опр п. 3.1.1, синт пп. 5.1.1, 6.5.1, текст п. 5.1.3
<простое арифметическое выражение> — опр п. 3.3.1, синт п. 3.4.1, текст п. 3.3.3
<простое логическое выражение> — опр п. 3.4.1
<пусто> — опр 1.1, синт пп. 3.2.1, 4.4.1, 4.7.1, 5.4.1, 6.3.3.1
<пустой оператор> — опр п. 4.4.1, синт п. 4.1.1, текст п. 4.4.3
<разделитель> — опр п. 2.3
размерность — текст п. 5.2.3.2
<сегмент массива> — опр п. 5.2.1
<скобка> — опр п. 2.3
скобки ( ) — синт пп. 2.3, 3.2.1, 3.3.1, 3.4.1, 4.7.1, 5.4.1, 6.2.1, 6.3.1, 6.4.1, 6.5.1, 6.6.1, текст п. 3.3.5.2
<совокупность спецификаций> — опр п. 5.4.1, текст п. 5.4.5
<совокупность фактических параметров> — опр пп. 3.2.1, 4.7.1
<совокупность формальных параметров> — опр п. 5.4.1
<составной оператор> — опр 4.1.1, синт п. 4.5.1, текст разд. 1
<спецификатор> — опр п. 2.3
<спецификация> — опр п. 5.4.1
<список граничных пар> — опр п. 5.2.1
<список значений> — опр п. 5.4.1, текст п. 4.7.3.1
<список идентификаторов> — опр п. 5.4.1
<список идентификаторов массивов> — опр п. 5.2.1
<список индексов> — опр п. 3.1.1
<список левой части> — опр п. 4.2.1
<список массивов> — опр п. 5.2.1
<список объектов ввода> — опр п. 6.5.1
<список объектов вывода> — опр п. 6.3.1
<список текстовых объектов вывода> — опр п. 6.3.1
<список типа> — опр п. 5.1.1
<список фактических параметров> — опр пп. 3.2.1, 4.7.1
<список формальных параметров> — опр п. 5.4.1
<список цикла> — опр п. 4.6.1, текст п. 4.6.4
<список элементов логического ввода> — опр п. 6.5.3
<список элементов числового ввода> — опр п. 6.5.3
стандартные функции — текст п. 6.1
<строка> — опр п. 2.6.1, синт пп. 3.2.1, 4.7.1, 5.4.1, 6.3.1, 6.5.3, 6.6.1, текст п. 2.6.3
<строка букв> — опр пп. 3.2.1, 4.7.1
<текстовый объект вывода> — опр п. 6.3.1, текст п. 6.3.5
<тело процедуры> — опр п. 5.4.1
<тело составного> — опр п. 4.1.1
<терм> — опр п. 3.3.1

<тип> — опр п. 5.1.1, синт пп. 5.2.1, 5.4.1, текст п. 2.8
точка с запятой ; — синт пп. 2.3, 4.1.1, 5.4.1, 6.5.3
<указатель переключателя> — опр п. 3.5.1, текст п. 3.5.3
<указатель размещения> — опр п. 6.3.6.1
<указатель функции> — опр. п. 3.2.1, синт пп. 3.3.1, 3.4.1, текст пп. 3.2.3, 5.4.4
умножение × — синт пп. 2.3, 3.3.1, текст п. 3.3.4.1
<условие> — опр пп. 3.3.1, 4.5.1, синт п. 3.4.1, текст пп. 3.3.3, 4.5.3
<условный оператор> — опр п. 4.5.1, синт п. 4.1.1, текст п. 4.5.3
<фактический параметр> — опр пп. 3.2.1, 4.7.1
<формальный параметр> — опр п. 5.4.1, текст п. 5.4.3
<формат размещения> — опр п. 6.3.6.1, синт п. 6.3.1, текст п. 6.3.6.2
функция преобразования — текст п. 4.2.4
целая часть — текст п. 6.1
<целое> — опр п. 2.5.1, текст п. 2.5.4
<целое без знака> — опр п. 2.5.1, синт п. 6.3.3.1
<целый формат> — опр п. 6.3.3.1, текст п. 6.3.3.2
<цифра> — опр п. 2.2.1, синт разд. 2, пп. 2.4.1, 2.5.1, 3.1.1, 3.5.1
<число> — опр п. 2.5.1, синт п. 6.5.3, текст пп. 2.5.3, 2.5.4
<число без знака> — опр п. 2.5.1, синт п. 3.3.1
<числовой формат> — опр п. 6.3.3.1, синт п. 6.3.1, текст п. 6.3.3.2
<экспоненциальный формат> — опр п. 6.3.3.1, текст п. 6.3.3.2
<элемент логического ввода> — опр п. 6.5.3
<элемент списка цикла> — опр п. 4.6.1, текст пп. 4.6.4.1, 4.6.4.2, 4.6.4.3
<элемент числового ввода> — опр п. 6.5.3
<B — часть> — опр п. 6.3.3.1, синт п. 6.3.4.1, текст п. 6.3.3.2
<D — часть> — опр п. 6.3.3.1, текст п. 6.3.3.2
<F — часть> — опр п. 6.3.4.1, текст п. 6.3.4.2

ПРИЛОЖЕНИЕ 2
Справочное

СООТВЕТСТВИЕ МЕЖДУ АНГЛИЙСКИМИ И РУССКИМИ СЛУЖЕБНЫМИ СЛОВАМИ

Некоторые трансляторы с АЛГАМСа могут допускать как входные тексты с английскими служебными словами, изображающими основные символы языка АЛГАМС, так и входные тексты с русскими служебными словами того же назначения. При этом рекомендуется придерживаться следующего соответствия между английскими и русскими служебными словами:

array — массив
begin — начало
Boolean — лог
comment — прим
do — цикл
else — иначе
end — конец
false — ложь
for — для
go to — на
if — если
integer — цел
label — метка
procedure — проц
real — вещ
step — шаг
string — строк
switch — перекл
then — то
true — истина
until — до
value — знач
while — пока

ПРИЛОЖЕНИЕ 3
Справочное

ПРЕДСТАВЛЕНИЕ ОСНОВНЫХ СИМВОЛОВ ЯЗЫКА АЛГАМС ЧЕРЕЗ СИМВОЛЫ ПО ГОСТ 19767—74

Основные символы языка АЛГАМС, представляющие из себя слова, кодируются большими латинскими буквами, взятыми в апострофы. Например: begin кодируется 'BEGIN'.

Основные символы, не имеющие эквивалента в наборе символов по ГОСТ 19767—74, кодируются в соответствии с приведенной ниже таблицей.

Основной символ	Представление	Основной символ	Представление
×	*	∨	'OR'
↑	**	∧	'AND'
≤	'LE'	¬	'NOT'
≥	'GE'	10	'10'
≠	'NE'	'	''
≡	'EQV'	,	,,
⊃	'IMP'		

Примечание. Открывающая и закрывающая кавычки кодируются с помощью двух апострофов. Остальные основные символы языка АЛГАМС кодируются соответствующими символами по ГОСТ 19767—74.

ПРИЛОЖЕНИЕ 4
Справочное

ИСТОРИЯ ЯЗЫКА АЛГАМС

Предлагаемый алгоритмический язык АЛГАМС разработан группой ГАМС (группой по автоматизации программирования для машин среднего типа).

Группа ГАМС была создана по инициативе Польской академии наук Комиссией многостороннего сотрудничества академий наук социалистических стран по проблеме «Научные вопросы вычислительной техники» в июне 1963 г. Перед группой ГАМС была поставлена задача создания эффективных средств автоматизации программирования в странах-участниках и, в частности, соответствующего языка для описания алгоритмов с ориентацией на машины средней мощности. На первом же рабочем совещании группы ГАМС в октябре 1963 г. в Софии было решено, что основой такого языка должно быть подмножество языка АЛГОЛ-60 и что язык будет называться АЛГАМС.

Хорошо известно, что при всех своих несомненных достоинствах АЛГОЛ 60 обладает свойствами, затрудняющими его использование на машинах с небольшим быстродействием и малой оперативной памятью, особенно в тех случаях, когда желательно иметь достаточно эффективные программы. Идея преодоления этих затруднений на пути выделения подмножества языка не нова, и в каком-то смысле, даже указана в первоначальном сообщении о языке АЛГОЛ-60 при упоминании о конкретных представлениях. В связи с трудностями реализации тех или иных свойств языка АЛГОЛ-60, по пути ограничений, то есть выделений некоторых подмножеств, пошло подавляющее большинство авторов трансляторов.

Поскольку появление самых различных вариантов языка противоречило идее унификации, послужившей толчком к созданию АЛГОЛа, совершенно естественными были усилия по стандартизации подмножеств.

Второй важной проблемой, возникшей на пути внедрения языка АЛГОЛ-60, была необходимость введения в конкретные представления и, в конечном счете, в язык средств, обеспечивающих ввод и вывод информации.

Эти задачи выбора подмножества и введения в язык средств, обеспечивающих ввод и вывод информации, возникли и перед группой ГАМС. Представители Польской академии наук переработали предложенный ими на первом рабочем совещании проект языка, и на втором рабочем совещании в апреле 1964 г. в Бухаресте А. Мазуркевич (Польша) доложил вариант АЛГАМСа, признанный основой для построения языка. На этом совещании были рассмотрены проект SUBSET ALGOL и сокращения языка АЛГОЛ-60, предложенные Академией наук СССР, и были приняты текст эталонного языка АЛГАМС, конкретное представление языка на телетайпе с пятидорожечной перфолентой и конкретное представление в коде RFT.

Недостатком принятого варианта эталонного языка было отсутствие в нем достаточно развитых стандартных процедур для описания ввода-вывода информации. Уже на следующем рабочем совещании группы ГАМС в октябре 1964 г. в Варшаве наряду с обсуждениями методов трансляции пришлось вернуться к тексту языка для внесения в него дополнений, связанных с вводом-выводом. Эта часть языка подвергалась уточнениям и дополнениям как на совещании ГАМС в марте 1965 г. в Берлине, так и в октябре 1965 г. в Ташкенте, где было принято решение о создании редакционной подгруппы, которая в феврале 1966 г. в Варшаве в составе И. Концевич (Польша), В. М. Курочкина (СССР), Э. З. Любимского (СССР), Л. Чайа (Польша) и Т. Шорца (Польша) составила текст языка АЛГАМС в форме поправок и дополнений к пересмотренному сообщению о языке АЛГОЛ-60. Текст поправок и дополнений, предложенный редакционной подгруппой, был одобрен на рабочем совещании ГАМС в апреле 1966 г. в Будапеште.

Работа группы ГАМС пересекалась во времени с работой над SUBSET ALGOL-60 (IFIP) в WG2.1. Предложения и решения рабочей группы IFIP/WG2.1 учитывались и оказывали влияние на работу группы ГАМС. Выход в апреле 1964 г. окончательной реакции сообщения о SUBSET ALGOL-60 (IFIP) заставил группу ГАМС заново проанализировать уже принятые ею решения с целью возможного исключения отличий языка АЛГАМС от SUBSET ALGOL-60. Конечно, речь шла не об устранении всех имеющихся отличий, поскольку в язык АЛГАМС уже были включены важные свойства, которые полностью отсутствовали в SUBSET ALGOL-60.

В конечном счете, если не считать запрещения строчных скобок внутри строк в АЛГАМСе, SUBSET ALGOL-60 оказался подмножеством языка АЛГАМС. В последнем по сравнению с SUBSET ALGOL-60 нет ограничения на изображение идентификаторов, нет ограничений на употребление операции возведения в степень, в определенных случаях разрешена подстановка фактических параметров типа **integer** на место формальных параметров типа **real** и, наконец, разрешена подстановка выражений по наименованию.

Настоящее приложение не ставит перед собой цели обоснования тех или иных решений, принятых группой ГАМС. Однако в качестве иллюстрации мотивов для таких решений можно рассмотреть отличие SUBSET ALGOL-60 от АЛГАМСа в части изображения идентификаторов. Принятое в первом ограничение, согласно которому результат появления различных идентификаторов с совпадающими первыми шестью символами неопределен, может быть по существу использовано транслятором лишь при отказе от анализа этой неопределенной ситуации и индикации ошибки. Ведь для обнаружения такой ошибки необходимо хранить все символы встречающихся идентификаторов. В то же время группа ГАМС считала с одной стороны невозможным отказ от индикации ошибок, связанных с неправильным изображением идентификаторов, а с другой стороны стремилась к тому, чтобы каждое ограничение на язык давало заметную выгоду при трансляции или в эффективности получаемых программ.

В языке АЛГАМС заметно расширен набор стандартных функций и процедур, а также предложен некоторый синтаксис для использования библиотечных подпрограмм. При этом исключено синтаксически неопределенное понятие <код>.

Процедуры ввода—вывода языка АЛГАМС являются переработанным и упрощенным вариантом известных предложений комиссии Д. В. Кнута для языка АЛГОЛ-60. При выборе этого варианта были подробно исследованы возможные реализации с учетом имеющегося оборудования.

В язык АЛГАМС включены средства, позволяющие дать указания о возможной сегментации программы, так называемые идентификаторы части, а также средства, дающие возможность эффективно использовать буферные памяти машины путем описаний некоторых из массивов особыми идентификаторами (начинающимися с букв EX). Доступ к таким массивам осуществляется при помощи специальной стандартной процедуры COPY. Все эти свойства должны заметно повысить эффективность использования языка как средства автоматизации программирования

Поскольку основой языка АЛГАМС был принят АЛГОЛ-60, группа ГАМС сочла возможным широко использовать текст пересмотренного сообщения о языке АЛГОЛ-60* и его русский перевод**, изменяя его, как правило, лишь в тех местах, которые были связаны с изменениями и дополнениями. Однако в некоторых случаях текст Revised **report** был изменен не для придания ему нового смысла, а с целью упрощения формы изложения. Эти изменения не отражены в помещенном в качестве дополнения списке отличий АЛГАМСа от языка АЛГОЛ-60, предназначенном для лиц, хорошо знакомых с языком АЛГОЛ-60.

* Revised report on the algorithmic language ALGOL 60 by J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. Mc Carthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. von Wijngaarden, M. Woodger. Edited by Peter Naur. International Federation for Information Processing, 1962.

** Алгоритмический язык АЛГОЛ-60. Пересмотренное сообщение. [Пер. с англ.]. Под ред. А. П. Ершова, С. С. Лаврова, М. Р. Шура-Бура. М. Изд-во «Мир», 1965 г

В работе группы ГАМС систематически принимали участие М. Апостолова (Болгария), Т. Бакош (Венгрия), Д. Вайда (Румыния), Ф. Грунд (ГДР), Б. Домелки (Венгрия), И. Кернер (ГДР), Е. Киндлер (Чехословакия), П. Константинеску (Румыния), И. Концевич (Польша), Я. Крал (Чехословакия), Р. Кретер (ГДР), В. М. Курочкин (СССР), Э. З. Любимский (СССР), А. Мазуркевич (Польша), Ю. Мароньский (Польша), М. Русева (Болгария), Б. Сендов (Болгария), Д. Стайку (Румыния), В. Фолтени (Венгрия), Л. Чайа (Польша), П. Шорц (Польша), Р. Штробель (ГДР), М. Р. Шура-Бура (СССР).

В 1971 г. в институте прикладной математики АН СССР был разработан транслятор с АЛГАМСа на универсальный машинно-ориентированный язык программирования АЛМО, что позволяет осуществлять трансляцию с языка АЛГАМС

в код любой вычислительной машины, оснащенной транслятором с АЛМО. (В настоящее время эксплуатируются трансляторы с АЛМО для БЭСМ-6, М-220 и некоторых других типов машин и завершается разработка транслятора для ЕС ЭВМ).

ПРИЛОЖЕНИЕ 5
Справочное

ОТЛИЧИЯ АЛГАМСа ОТ ЯЗЫКА АЛГОЛ-60

1. Используется один регистр букв (в вышеприведенном тексте языка используются лишь прописные латинские буквы).
2. Выброшен знак ÷ (деление нацело реализуется с помощью стандартной функции).
3. Нет понятия own.
4. Упрощена конструкция строк.
5. Если для определения типа арифметического выражения необходимо выполнять какие-либо вычисления, проверки условий или присваивания, то считается, что выражение имеет тип real (см. пп 3.3.4.3 и 3.3.4.4.).
6. Именующими выражениями могут быть только метки и указатели переключателя. Переключательный список может состоять только из меток. Целое без знака не может быть меткой. При неопределенном указателе переключателя оператор перехода также неопределен.
7. Управляемой переменной цикла (параметром цикла) может быть только простая переменная.
8. Все формальные параметры процедуры должны быть специфицированы. При вызове по наименованию класс и тип фактического параметра, вообще говоря, должен совпадать с классом и типом формального параметра (п. 4.7.5.5).
9. Не допускается рекурсивное использование процедур.
10. Не допускается побочный эффект.
11. Уточнено использование кода в качестве тела процедуры пп. 5.4.1, 5.4.3).
12. Описание идентификатора (за исключением меток) должно предшествовать его использованию.
13. Введены понятия внешнего идентификатора и идентификатора части. Добавлен раздел 6 о стандартных функциях и стандартных процедурах (процедуры, осуществляющие ввод, вывод и обмен информацией).

Редактор *С. Г. Вилькина*
Технический редактор *В. Ю. Смирнова*
Корректор *Е. И. Евтеева*

Сдано в наб. 31.05.79 Подп. в печ. 31.05.79 3,0 п. л. 2,84 уч.-изд. л. Тир. 6000 Цена 15 коп.

Ордена «Знак Почета» Издательство стандартов, Москва, Д-557, Новопресненский пер., 3.
Тип. «Московский печатник» Москва, Лялин пер., д. 6. Зак. 1485

gost.e-ecolog.ru