ОБЪЕДИНЕННЫЙ ИНСТИТУТ ЯДЕРНЫХ ИССЛЕДОВАНИЙ
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

Бэкус Д.В. и др.

R 1139

ОБ АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ АЛГОЛ-60

Дубна 1962

R 1139

ОБ АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ АЛГОЛ-60

Дубна 1962

ЖУРНАЛ
ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ и МАТЕМАТИЧЕСКОЙ ФИЗИКИ
Том I Март 1961 Апрель № 2

СООБЩЕНИЕ ОБ АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ АЛГОЛ-60

Дж. В. БЭКУС, Ф. Л. БАУЭР, Дж. ГРИН, С. КЭТЦ, Дж. МАК-КАРТИ, П. НАУР, Э. Дж. ПЕРЛИС, Х. РУТИСХАУЗЕР, К. ЗАМЕЛЬЗОН, Б. ВОКУА, Дж. УЭГСТЕЙН, А. ВАН-ВЭЙНГААРДЕН, М. ВУДЖЕР

*под редакцией ПЕТЕРА НАУРА**

(перевод Г. И. КОЖУХИНА под редакцией А. П. ЕРШОВА)

Посвящается памяти
*УИЛЬЯМА ТУРАНСКОГО***

Введение

Предыстория. После опубликования (см. [1], [2]) предварительного отчета об алгоритмическом языке АЛГОЛ, подготовленного Цюрихской конференцией в 1958 г., к этому языку был проявлен значительный интерес.

В результате неофициальной встречи в Майнце в ноябре 1958 г. около сорока заинтересованных лиц из различных европейских стран собрались на конференцию по АЛГОЛу в Копенгагене в феврале 1959 г. Была организована рабочая группа для совместной работы по нахождению способа записи языка на перфолентах. Эта конференция привела к публикации Вычислительным центром в Копенгагене «АЛГОЛ-бюллетеня» (ALGOL Bulletin) под редакцией Петера Наура (Peter Naur) с тем, чтобы это издание явилось форумом для дальнейших обсуждений. В течение июня 1959 г. в Париже на конференции ICIP (Международная конференция по обработке информации) состоялось несколько как официальных, так и неофициальных встреч. Эти встречи выявили некоторое недопонимание целей группы, которая первоначально была ответственна за разработку языка. С другой стороны, стало ясно, что имеется широкое одобрение предпринятых усилий. В результате обсуждений было решено созвать в январе 1960 г. Международную конференцию для улучшения языка АЛГОЛ и подготовки окончательного отчета. В ноябре 1959 г. в Париже на европейской конференции по АЛГОЛу, на которой присутствовало около пятидесяти человек, были выбраны семь европейских представителей для участия в январской конференции 1960 г. Они представляли следующие орга-

* J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur (editor), A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, M. Woodger. Report on the algorithmic language ALGOL 60. *Numer. Math.*, 1960, 2, № 2, 106—136.

** Уильям Туранский — представитель американской группы на январской конференции 1960 г.— погиб при автомобильной катастрофе как раз перед этой конференцией.

низации: Французскую вычислительную ассоциацию (Association Francaise de Calcul), Британское вычислительное общество (British Computer Society), Общество по прикладной математике и механике (Германия) (Gesellschaft für Angewandte Mathematik und Mechanik) и Голландское общество по вычислительной математике (Nederlands Rekenmachine Genootschap). Эти представители встретились в декабре 1959 г. в Майнце для заключительной подготовки.

Тем временем в США всем желающим высказать свои замечания по языку АЛГОЛ было предложено направлять их в журнал ACM Communications, где они публиковались. Эти замечания стали затем основой обсуждений для изменения АЛГОЛа. Обе организации (SHARE и USE), создавшие рабочую группу по АЛГОЛу, были представлены в Комитете Ассоциации по вычислительной технике (ACM) по языку программирования. Комитет ACM собрался в Вашингтоне в ноябре 1959 г. и рассмотрел все замечания, присланные в ACM Communications. Было также избрано семь представителей для участия в Международной конференции в январе 1960 г. Заключительная подготовительная встреча этих представителей состоялась в Бостоне в декабре 1959 г.

Январская конференция 1960 г. Тринадцать представителей из Дании, Англии, Франции, Германии, Голландии, Швейцарии и США собрались на конференцию в Париже, которая проходила с 11 по 16 января 1960 г.

Перед этой встречей Петер Наур, исходя из предварительного описания языка и рекомендаций подготовительных встреч, выработал совершенно новый проект описания. Конференция приняла эту форму как основу для своего сообщения. Затем конференция перешла к работе по согласованию каждой из частей сообщения. Настоящее сообщение представляет собой объединение концепций комитета и перечисление его соглашений.

Как и в предыдущем сообщении о языке АЛГОЛ, различаются три уровня языка, а именно: эталонный язык, язык публикаций и различные конкретные представления.

Эталонный язык:

1) является рабочим языком комитета;

2) является определяющим языком;

3) знаки языка определены, исходя из желания облегчить взаимопонимание, а не из каких-либо машинных ограничений, кодовых или чисто математических обозначений;

4) является основой и руководством для создателей программирующих программ;

5) является образцом для всех конкретных представлений;

6) является основой для перевода с языка публикаций на любое данное конкретное представление;

7) основные публикации самого языка АЛГОЛ будут использовать эталонное представление.

Язык публикаций:

1) допускает изменения эталонного языка, связанные с удобством

печати или написания (например, индексы, пробелы, показатели степени, греческие буквы);

2) используется для взаимного обмена информацией;

3) знаки языка, используемые в различных странах, могут быть различными, но должно быть гарантировано однозначное соответствие с эталонным языком.

Конкретные представления:

1) каждое из них является таким сокращением эталонного языка, которое вызвано ограниченным числом символов в стандартном оборудовании ввода;

2) каждое из них использует совокупность символов конкретной машины и является входным языком программирующей программы для данной машины;

3) каждое из них должно сопровождаться правилами для перевода с языка публикаций или эталонного языка.

Для перевода между эталонным языком и языком публикаций, наряду с другими, рекомендуются следующие правила:

Эталонный язык	Язык публикаций
Индексные скобки []	Понижение строки, заключенной в скобки, и удаление скобок.
Возведение в степень↑	Поднятие показателя степени.
Скобки ()	Любая форма скобок.
Основание десять$_{10}$	Поднятие десяти и следующего за ним целого числа; вставляется знак умножения.

Описание эталонного языка

То, что вообще может быть сказано, должно быть сказано ясно, а о чем невозможно говорить — о том следует молчать.

ЛЮДВИГ ВИТГЕНШТЕЙН
«Логико-философский трактат»

1. Структура языка

Как было сказано во введении, алгоритмический язык имеет три различных уровня представления — эталонное, конкретное и для публикаций. Основное описание дается в терминах эталонного представления. Это означает, что все определяемые в языке объекты представляются посредством фиксированной совокупности символов и разница для других двух представлений состоит только в выборе символов. Структура и содержание остаются для всех представлений одними и теми же.

Назначением алгоритмического языка является описание вычислительных процессов. Основной концепцией, используемой для описания правил вычисления, является хорошо известное понятие арифметического выражения, содержащее в качестве компонент числа переменные и функции. Из таких выражений путем применения правил композиции образуются замкнутые единицы языка — полные формулы, называемые операторами присваивания.

Для того чтобы указать течение вычислительного процесса, добавляются некоторые неарифметические операторы, а также условия выбора операторов, которые могут, например, описывать выбор альтернативы

или циклическое повторение вычислительных операторов. Ввиду того что возникает необходимость обращения одного оператора к другому, операторы могут снабжаться метками. Последовательность операторов может комбинироваться в составные операторы путем заключения ее в операторные скобки.

Операторы могут быть снабжены описаниями, которые сами по себе не являются предписаниями о вычислениях, но информируют программирующую программу о существовании и некоторых свойствах объектов, появляющихся в операторах. Этими свойствами могут быть, например, класс чисел, используемых в качестве значений переменной, размерность массива чисел или даже совокупность правил, определяющих функцию. Каждое описание связано с некоторым составным оператором и имеет силу только для него. Составной оператор, который включает описания, называется блоком.

Программа является замкнутым составным оператором, т. е. составным оператором, который не содержится ни в каком другом составном операторе и не использует никакой составной оператор, не содержащийся в нем.

Ниже будут описаны синтаксис и семантика языка *.

1.1. *Формализм для синтаксических описаний*

Синтаксис будет описываться с помощью металингвистических формул [3]. Их интерпретация лучше всего объясняется с помощью примера:

$$\langle ab\rangle ::= (\,|\,[\,|\,\langle ab\rangle\,(\,|\,\langle ab\rangle\langle d\rangle$$

Последовательность знаков, заключенных в скобки $\langle\rangle$, представляет собой металингвистические переменные, значения которых суть последовательности символов. Знаки $::=$ и $|$ (последний со значением «или») ** суть 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 означает соединение обозначаемых последовательностей. Таким образом, приведенная формула дает рекурсивное правило для образования значений переменной $\langle ab\rangle$. Она указывает, что $\langle ab\rangle$ может иметь значения (или [, а если дано некоторое допустимое значение $\langle ab\rangle$, то новое его значение может быть получено путем постановки вслед за ним символа (или значения $\langle d\rangle$. Если значениями $\langle d\rangle$ являются десятичные цифры, то некоторые из значений $\langle ab\rangle$ суть:

[(((*1*(*37*(
(*1 2 3 4 5*(
(((
[*86*

* Всякий раз, когда утверждается, что точность арифметических действий, вообще говоря, не указана, или когда говорится, что результат некоторого процесса точно не определен, это должно пониматься так, что программа станет точно определять некоторый вычислительный процесс только тогда, когда дополнительная информация точно укажет как подразумеваемые точность и вид арифметических действий, так и последовательность выполняемых действий для всех случаев, которые могут встретиться в процессе вычислений.

** Знак $::=$ означает «равно по определению». (*Прим. ред. перевода.*)

С целью облегчения запоминания в качестве символов, используемых для металингвистических переменных (т. е. последовательностей знаков, появляющихся внутри скобок ⟨⟩, подобно *ab* в вышеприведенном примере) были выбраны слова, приблизительно описывающие природу соответствующих переменных. Там, где слова, которые были введены таким способом, используются где-либо в тексте, они будут относиться к соответствующему синтаксическому определению. Кроме того, некоторые формулы будут приведены по нескольку раз.

Определение:

⟨пусто⟩ : : =

(т. е. нулевая строка символов).

2. Основные символы, идентификаторы, числа и строки. Основные понятия

Эталонный 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*
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*

Этот алфавит может быть произвольно сужен или расширен добавлением любых других символов (т. е. символов, не совпадающих с любой цифрой, логическим значением или ограничителем).

Буквы не имеют индивидуального значения. Они используются для образования идентификаторов и строк* (см. раздел 2.4. ***Идентификаторы*** и 2.6. *Строки*).

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 очевидный смысл**.

2.3. *Ограничители*

⟨ограничитель⟩ : : = ⟨операция⟩ | ⟨разделитель⟩ | ⟨скобка⟩ | ⟨описатель⟩ | ⟨спецификатор⟩

⟨операция⟩ : : = ⟨арифметическая операция⟩ | ⟨операция отношения⟩ | ⟨логическая операция⟩ | ⟨операция следования⟩

⟨арифметическая операция⟩ : : = + | − | × | / | ÷ | ↑

⟨операция отношения⟩ : : = < | ≤ | = | ≥ | > | ≠

* Следует заметить, что всюду в настоящем описании выделение в тексте (не в заголовках) полужирным шрифтом используется для обозначения независимых основных символов (см. разделы 2.2.2, 2.3 и т. д.). Подразумевается, что основные символы не имеют никакого отношения к буквам, из которых они составлены. В этом сообщении выделение текста полужирным шрифтом для других целей не используется.

** **true** — истина, **false** — ложь. (*Прим. ред. перевода.*)

⟨логическая операция⟩ : : = ≡ | ⊃ | ∨ | ∧ | ¬
⟨операция следования⟩ : : = **go to** | **if** | **then** | **else** | **for** | **do** *
⟨разделитель⟩ : : = , | · | ₁₀ | : | ; | : = | ␣ | **step** | **until** | **while** | **comment**
⟨скобка⟩ : : = (|) | [|] | ‘ | ’ | **begin** | **end**
⟨описатель⟩ : : = **own** | **Boolean** | **integer** | **real** | **array** | **switch** | **procedure**
⟨спецификатор⟩ : : = **string** | **label** | **value**

Ограничители имеют фиксированное значение, которое в большинстве случаев очевидно **, в остальных случаях будет пояснено в соответствующем месте.

Такие типографские особенности, как пробел или переход к новой строке, в эталонном языке не обозначаются. Однако для облегчения чтения их можно свободно использовать.

Для возможности включения текста между символами программы имеют место следующие правила для примечаний:

Последовательность основных символов	Ее эквивалент
; **comment** ⟨любая последовательность, не содержащая; ⟩;	;
begin comment ⟨любая последовательность, не содержащая; ⟩;	**begin**
end ⟨любая последовательность, не содержащая **end** или ; или **else**⟩	**end**

Эквивалентность здесь означает, что любой из трех символов, показанных в правой колонке, если он встречается вне некоторой строки, может заменяться любой последовательностью символов структуры, показанной слева, не оказывая при этом никакого влияния на программу.

2.4. *Идентификаторы*

2.4.1. Синтаксис.

⟨идентификатор⟩ : : = ⟨буква⟩ | ⟨идентификатор⟩ ⟨буква⟩ | ⟨идентификатор⟩ ⟨цифра⟩

2.4.2. Примеры***.

q
Soup
v17a
a34kTMNs
MARILYN

---

* **do** используется в операторах цикла. Это **do** не имеет никакого отношения к **do**, употребляемому в предварительном сообщении и не включенному в АЛГОЛ-60.

** **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, **own** — собственный, **Boolean** — булевский, **integer** — целый, **real** — вещественный, **array** — массив, **switch** — переключатель, **procedure** — процедура, **string** — строка, **label** — метка, **value** — значение. (*Прим. ред. перевода.*)

*** В дальнейшем при описании примеров в некоторых случаях, когда изображение идентификатора имеет мнемоническое значение, английское слово будет заменяться соответствующим русским переводом. При этом будет неявно предполагаться, что алфавит АЛГОЛа расширен введением русского алфавита. (*Прим. переводчика.*)

2.4.3. Семантика.

Идентификаторы не имеют внутреннего смысла, а служат только для обозначения простых переменных, массивов, меток, переключателей и процедур. Они могут выбираться произвольно (см., однако, раздел 3.2.4. Стандартные функции).

Один и тот же индентификатор не может использоваться для обозначения различных величин, за исключением того случая, когда эти величины, согласно описаниям программы, имеют несовместные области действия (см. раздел 2.7. *Величины, классы и области действия* и раздел 5. Описания).

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⟩

⟨десятичная дробь⟩ : : = . ⟨целое без знака⟩

⟨десятичный порядок⟩ : : = $_{10}$ ⟨целое⟩

⟨десятичное число⟩ : : = ⟨целое без знака⟩ | ⟨десятичная дробь⟩ | ⟨целое без знака⟩ ⟨десятичная дробь⟩

⟨число без знака⟩ : : = ⟨десятичное число⟩ | ⟨десятичный порядок⟩ | ⟨десятичное число⟩ ⟨десятичный 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. Десятичный порядок — это масштабный множитель, выраженный как целая степень десяти.

2.5.4. Типы.

Целые числа имеют тип **integer**. Все остальные числа имеют тип **real** (см. раздел 5.1. *Описания типов*).

2.6. *Строки*

2.6.1. Синтаксис.

⟨правильная строка⟩ : : = ⟨любая последовательность основных символов, не содержащая **'или'** ⟩ | ⟨пусто⟩

⟨открытая строка⟩ : : = ⟨правильная строка⟩ | '⟨открытая строка⟩' | ⟨открытая строка⟩ ⟨открытая строка⟩

⟨строка⟩ : : = '⟨открытая строка⟩'

2.6.2. Примеры.

$$'5k,, -'[[['\wedge = /:' \ Tt''$$
$$'..This \sqcup is \sqcup a \sqcup 'string''$$

2.6.3. Семантика.

Для того чтобы обеспечить возможность в языке иметь дело с произвольной последовательностью основных символов, введены кавычки для строк ' и '. Символ $\sqcup$ обозначает пробел. Вне строки он не имеет смысла.

Строки используются в качестве фактических параметров процедур (см. разделы 3.2. *Функции* и 4.7. *Операторы процедур*).

2.7. *Величины, классы и области действия*

Различаются следующие классы величин: простые переменные, массивы, метки, переключатели и процедуры.

Область действия величины — это совокупность операторов, для которых имеют силу описания идентификатора, связанного с этой величиной, или — для метки — совокупность операторов, которые могут иметь своим преемником оператор, содержащий данную метку.

2.8. *Значения и типы*

Значение — это некоторое упорядоченное множество чисел (специальный случай: одно число), некоторое упорядоченное множество логических значений (специальный случай: одно логическое значение) или некоторая метка.

Некоторые синтаксические единицы могут принимать какие-либо значения. Вообще говоря, во время работы программы эти значения будут изменяться. Значение выражений и их компонент определяются в разделе 3. Значение идентификатора массива есть упорядоченное множество значений соответствующего массива переменных с индексами (см. раздел 3.1.4.1).

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

3. Выражения

В данном языке первичными компонентами программ, описывающих алгоритмические процессы, являются арифметические, булевские и именующие выражения. Компонентами этих выражений, помимо некоторых ограничителей, являются логические значения, числа, переменные, функции и элементарные арифметические и логические операции, а также некоторые операции следования и отношения. Ввиду того что синтаксические определения как переменных, так и функций содержат выражения, определение выражений и их компонент может быть только рекурсивным:

⟨выражение⟩ : : = ⟨арифметическое выражение⟩ | ⟨булевское 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⟩
⟨переменная с индексами⟩ : : = ⟨идентификатор массива⟩
[⟨список индексов⟩]
⟨переменная⟩ : : = ⟨простая переменная⟩ | ⟨переменная с индексами⟩

3.1.2 Примеры.

epsilon
detA
a17
Q[7,2]
x[sin(n × pi/2), Q[3, n, 4]]

3.1.3. Семантика.

Переменная — это наименование, данное некоторому одному значению. Это значение может быть использовано в выражениях для образования других значений и может быть изменено впоследствии посредством операторов присваивания (раздел 4.2). Тип значения данной переменной определяется описанием самой переменной (см. раздел 5.1. *Описания типов*) или соответствующего идентификатора массива (см. раздел 5.2. *Описания массивов*).

3.1.4. Индексы.

3.1.4.1. Переменные с индексами именуют значения, которые являются компонентами многомерных массивов (см. раздел 5.2. *Описания массивов*). Каждое арифметическое выражение из списка индексов занимает одну позицию индекса переменной с индексами и называется индексом. Полный список индексов заключается в индексные скобки [ ]. Фактическая компонента массива, упоминаемая с помощью переменной с индексами, определяется по фактическим численным значениям ее индексов (см. раздел 3.3. *Арифметические выражения*).

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

3.2. *Функции*

3.2.1. Синтаксис.

⟨идентификатор процедуры⟩ : : = ⟨идентификатор⟩
⟨фактический параметр⟩ : : = ⟨строка⟩ | ⟨выражение⟩ |
⟨идентификатор массива⟩ | ⟨идентификатор процедуры⟩ |
⟨идентификатор переключателя⟩
⟨строка букв⟩ : : = ⟨буква⟩ | ⟨строка букв⟩ ⟨буква⟩
⟨ограничитель параметра⟩ : : = , |) ⟨строка буки⟩ : (
⟨список фактических параметров⟩ : : = ⟨фактический параметр⟩ |
⟨список фактических параметров⟩ ⟨ограничитель параметра⟩
⟨фактический параметр⟩
⟨совокупность фактических параметров⟩ : : = ⟨пусто⟩ |
(⟨список фактических параметров⟩)
⟨функция⟩ : : = ⟨идентификатор процедуры⟩ ⟨совокупность фактических параметроа⟩

3.2.2. Примеры.

sin(*a* − *b*)
J(*v* + *s*, *n*)
R
Compile (' : = ') *Stack* : (*Q*)
S (*s* — 5) *Температура* : (*T*) *Давление*: (*P*)

3.2.3. Семантика.

Функция определяет одно числовое или логическое значение, которое является результатом применения заданной совокупности правил, определяемых описанием процедуры (см. раздел 5.4. *Описания процедур*) к фиксированной совокупности фактических параметров. Правила, регулирующие задание фактических параметров, даны в разделе 4.7. *Операторы процедур*. Следует отметить, что не каждое описание процедуры определяет значение какой-либо функции.

3.2.4. Стандартные функции.

Желательно, чтобы некоторые идентификаторы были сохранены для стандартных функций анализа, которые будут задаваться процедурами. Рекомендуется, чтобы список закреплений содержал:

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 значения E
sin (E) для синуса значения E
cos (E) для косинуса значения E
arctan (E) для главного значения арктангенса значения E
ln (E) для натурального логарифма значения E
exp (E) для экспоненциальной функции значения E(e^E)

Эти функции определены как для аргументов типа **real**, так и для аргументов типа **integer**. Функция *sign* (E) будет иметь значение типа **integer**. Все остальные дают значения типа **real**. В отдельных конкретных представлениях эти функции могут использоваться без точных описаний (см. раздел 5. Описания).

3.2.5. Функции преобразования.

Ясно, что между любой парой величин и выражений можно определить функцию преобразования. Среди стандартных функций рекомендуется иметь одну, называемую *entier* (E), которая «преобразует» выражение типа **real** в выражение типа **integer** и присваивает ему значение, являющееся наибольшим целым, не превышающим значение E.

3.3. *Арифметические выражения*

3.3.1. Синтаксис.

⟨операция типа сложения⟩ : : = + | −
⟨операция типа умножения⟩ : : = × | / | ÷
⟨первичное выражение⟩ : : = ⟨число без знака⟩ | ⟨переменная⟩ |
⟨функция⟩ | (⟨арифметическое выражение⟩)
⟨множитель⟩ : : = ⟨первичное выражение⟩ |
⟨множитель⟩ ↑ ⟨первичное выражение⟩
⟨терм⟩ : : = ⟨множитель⟩ | ⟨терм⟩ ⟨операция типа умножения⟩ ⟨множитель⟩

⟨простое арифметическое выражение⟩ : : = ⟨терм⟩ |
⟨операция типа сложения⟩ ⟨терм⟩ |
⟨простое арифметическое выражение⟩ ⟨операция типа сложения⟩
⟨терм⟩
⟨условие⟩ : : = **if** ⟨булевское 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
$w[i + 2,8]$
$cos(y + z \times 3)$
$(a - 3/y + vu \uparrow 8)$

Множители:

omega
$sum \uparrow cos(y + z \times 3)$
$7.394_{10} - 8 \uparrow w[i + 2,8] \uparrow (a - 3/y + vu \uparrow 8)$

Термы:

U
$omega \times sum \uparrow cos(y + z \times 3)/7.394_{10} - 8 \uparrow w[i + 2,8] \uparrow (a - 3/y + vu \uparrow 8)$

Простое арифметическое выражение:

$U - Yu + omega \times sum \uparrow cos(y + z \times 3)/7.394_{10} - 8 \uparrow w[i + 2,8] \uparrow$
$(a - 3/y + vu \uparrow 8)$

Арифметические выражения:

$w \times u - Q(S + Cu) \uparrow 2$
if $q > 0$ **then** $S + 3 \times Q/A$ **else** $2 \times S + 3 \times q$
if $a > 0$ **then** $u + v$ **else if** $a \times b > 17$ **then** u/v **else if** $k \neq y$ **then** v/u **else** 0
$a \times sin(omega \times t)$
$0.57_{10}\,12 \times a[N \times (N - 1)/2,0]$
$(A \times arctan(y) + z) \uparrow (7 + Q)$
if q **then** $n - 1$ **else** n
if $a < 0$ **then** A/B **else if** $b = 0$ **then** B/A **else** z

3.3.3. Семантика.

Арифметическое выражение является правилом для вычисления числового значения. В случае простого арифметического выражения это значение получается путем выполнения указанных арифметических операций над фактическими числовыми значениями первичных выражений. Детально это объяснено ниже в разделе 3.3.4. Что такое фактическое числовое значение первичного выражения — ясно в случае чисел. Для переменных оно является текущим значением (последнее по времени присвоенное значение), а для функций оно является значением, полученным по правилам вычислений, определяющих процедуру (см. раздел 5.4. *Описания процедур*), примененным к текущим значениям параметров процедуры, заданных в выражении. Наконец, для арифметических выражений, заключенных в скобки, их значение должно быть выражено путем рекур-

сивного процесса в терминах значений остальных трех видов первичных выражений.

В более общем арифметическом выражении, включающем в себя условия, нахождение значения определяется выбором одного из нескольких арифметических выражений на основе фактических значений булевских выражений. Это происходит следующим образом: значения булевских выражений, входящих в условия, вычисляются последовательно слева направо до тех пор, пока не найдется выражение, имеющее значение **true**. Значением арифметического выражения тогда будет значение первого арифметического выражения, следующего за этим булевским (при этом имеется в виду максимальное арифметическое выражение, найденное в этой позиции). Конструкция

else ⟨простое арифметическое выражение⟩

эквивалентна конструкции

else if true then ⟨простое арифметическое выражение⟩

3.3.4. Операции и типы.

Компоненты простого арифметического выражения (за исключением булевских выражений, употребляемых в условиях) должны быть типа **real** или **integer**. Смысл основных операций и типы выражений, к которым они приводят, даются следующими правилами.

3.3.4.1. Операции $+$, $-$ и $\times$ имеют соответствующий смысл (сложение, вычитание и умножение). Тип выражения будет **integer**, если оба операнта имеют тип **integer**, в противном случае оно будет иметь тип **real**.

3.3.4.2. Операции ⟨терм⟩ / ⟨множитель⟩ и ⟨терм⟩ ÷ ⟨множитель⟩ обозначают деление, понимаемое как умножение терма на обратную величину множителя с соответствующим учетом правил старшинства (см. раздел 3.3.5). Таким образом, например,

$$a / b \times 7 / (p - q) \times v / s$$

означает

$$((((a \times (b^{-1})) \times 7) \times ((p - q)^{-1})) \times v) \times (s^{-1})$$

Операция / определена для всех четырех комбинаций типов **real** и **integer** и в любом случае дает результат типа **real**. Операция ÷ определена только для того случая, когда оба операнта имеют тип **integer**. Результат последней операции также имеет тип **integer**, определенный следующим образом:

$$a \div b = sign\,(a/b) \times entier\,(abs\,(a/b))$$

(см. разделы 3.2.4 и 3.2.5).

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

$2 \uparrow n \uparrow k$ означает $(2^n)^k$

тогда как

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

Если писать i вместо числа типа **integer**, r вместо числа типов **real** и t вместо числа типа **real** или **integer**, то результат операции определяется следующими правилами:

$t \uparrow i$ равно, если $i > 0$: $a \times a \times \ldots \times a$ (i раз), того же типа, что и a;

если $i = 0$, если $a \neq 0$: 1, того же типа, что и a;
если $a = 0$, неопределенно;
если $i < 0$, если $a \neq 0$: $1/(a \times a \times \ldots \times a)$ (знаменатель имеет i множителей), типа **real**;
если $a = 0$: неопределенно;
$a \uparrow r$ равно, если $a > 0$: $exp(r \times ln(a))$, типа **real**;
если $a = 0$, если $r > 0$: 0.0, типа **real**;
если $r \leqslant 0$: неопределенно;
если $a < 0$: всегда неопределенно.

3.3.5. Старшинство операций.

Последовательность операций в выражении выполняется, вообще говоря, слева направо с учетом следующих правил.

3.3.5.1. По отношению к синтаксису, данному в разделе 3.3.1, выдерживается следующий порядок старшинства:

первый: $\uparrow$
второй: $\times / \div$
третий: $+ -$

3.3.5.2. Выражение между левой скобкой и соответствующей правой скобкой вычисляется самостоятельно, и его значение используется в дальнейших вычислениях. Желаемый порядок выполнения операций 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, т. е. как объекты, определенные с присущей им конечной точностью. Точно так же понимается возможность конечных отклонений от математически определяемого результата в любом арифметическом выражении. Никакой точной арифметики не будет определяться, и ясно, что различные конкретные представления могут приводить к различным значениям арифметического выражения. Контроль возможных следствий таких различий должен проводиться методами численного анализа. Этот контроль должен рассматриваться как часть описываемого процесса и, следовательно, сам будет выражаться в терминах данного языка.

3.4. *Булевские выражения*

3.4.1. Синтаксис.

⟨операция отношения⟩ : : = $< | \leqslant | = | \geqslant | > | \neq$

⟨отношение⟩ : : = ⟨арифметическое выражение⟩ ⟨операция отношения⟩ ⟨арифметическое выражение⟩

⟨первичное булевское выражение⟩ : : = ⟨логическое значение⟩ | (⟨булевское выражение⟩) | ⟨переменная⟩ | ⟨функция⟩ | ⟨отношение⟩

⟨вторичное булевское выражение⟩ : : = ⟨первичное булевское выражение⟩ | $\neg$ ⟨первичное булевское выражение⟩

⟨булевский множитель⟩ : : = ⟨вторичное булевское выражение⟩ | ⟨булевский множитель⟩ $\wedge$ ⟨вторичное булевское выражение⟩

⟨булевский терм⟩ : : = ⟨булевский множитель⟩ | ⟨булевский терм⟩ $\vee$ ⟨булевский множитель⟩

⟨импликация⟩ : : = ⟨булевский терм⟩ | ⟨импликация⟩ $\supset$ ⟨булевский терм⟩

⟨простое булевское выражение⟩ : : = ⟨импликация⟩|
⟨простое булевское выражение⟩ ≡ ⟨импликация⟩
⟨булевское выражение⟩ : : = ⟨простое булевское выражение⟩|
⟨условие⟩⟨простое булевское выражение⟩ **else** ⟨булевское выражение⟩

3.4.2. Примеры.

$x = -2$
$Y > V \vee z < q$
$a \div b > -5 \wedge z - d > q \uparrow 2$
$p \wedge q \vee x \neq y$
$g \equiv \neg a \wedge b \wedge \neg c \vee d \vee e \supset \neg f$
if $k < 1$ **then** $s > w$ **else** $h \leqslant c$
if if if a **then** b **else** c **then** d **else** f **then** g **else** $h < k$

3.4.3. Семантика.

Булевское выражение является правилом для вычисления логического значения. Принципы вычисления полностью аналогичны правилам, данным в разделе 3.3.3 для арифметического выражения.

3.4.4. Типы.

Переменным и функциям, используем в качестве первичных булевских выражений, должен приписываться тип **Boolean** (см. раздел 5.1. *Описания типов* и раздел 5.4.4. Значения функций).

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**.

Значение логических операций $\neg$ (не), $\wedge$ (и), $\vee$ (или), $\supset$ (влечет) и $\equiv$ (эквивалентно) даются следующей функциональной таблицей.

b 1 *b 2*	**false** **false**	**false** **true**	**true** **false**	**true** **true**
$\neg b\,1$	**true**	**true**	**false**	**false**
$b\,1 \wedge b\,2$	**false**	**false**	**false**	**true**
$b\,1 \vee b\,2$	**false**	**true**	**true**	**true**
$b\,1 \supset b\,2$	**true**	**true**	**false**	**true**
$b\,1 \equiv b\,2$	**true**	**false**	**false**	**true**

3.4.6. Старшинство операций.

Операции в выражении выполняются слева направо с учетом следующих добавочных правил.

3.4.6.1. По отношению к синтаксису, данному в разделе 3.4.1, выдерживается следующий порядок старшинства:

первый: арифметическое выражение согласно разделу 3.3.5
второй: $< \leqslant = \geqslant > \neq$
третий: $\neg$
четвертый: $\wedge$
пятый: $\vee$
шестой: $\supset$
седьмой: $\equiv$

3.4.6.2. Применение скобок будет интерпретироваться в смысле, данном в разделе 3.3.5.2.

3.5. *Именующие выражения*

3.5.1. Синтаксис.

⟨метка⟩ : : = ⟨идентификатор⟩|⟨целое без знака⟩

⟨идентификатор переключателя⟩ : : = ⟨идентификатор⟩

⟨указатель переключателя⟩ : : = ⟨идентификатор переключателя⟩
[⟨индексное выражение⟩]

⟨простое именующее выражение⟩ : : = ⟨метка⟩|⟨указатель переключателя⟩|
(⟨именующее выражение⟩)

⟨именующее выражение⟩ : : = ⟨простое именующее выражение⟩|
⟨условие⟩⟨простое именующее выражение⟩**else**⟨именующее выражение⟩

3.5.2. Примеры

17

p9

выбрать[*n* — *1*]

Town [**if** *y* < *0* **then** *N* **else** *N* + *1*]

if *Ab* < *c* **then** *17* **else** *q* [**if** *w* ⩽ *0* **then** *2* **else** *n*]

3.5.3. Семантика.

Именующее выражение является правилом для определения метки оператора (см. раздел 4. Операторы). Принципы вычисления значения именующего выражения по-прежнему полностью аналогичны правилам, приведенным для арифметического выражения (раздел 3.3.3.). В общем случае булевское выражение, содержащееся в условии, выбирает простое именующее выражение. Если это метка, то желаемый результат уже получен. Указатель переключателя отсылает к описанию соответствующего переключателя (см. раздел 5.3. *Описание переключателей*) и по числовому значению его индексного выражения выбирает одно из именующих выражений, перечисленных в описании переключателя. Выбор осуществляется путем пересчета этих выражений слева направо. Так как выбранное таким образом именующее выражение может вновь оказаться переключателем, то вычисление значения, очевидно, представляет собой рекурсивный процесс.

3.5.4. Индексное выражение.

Вычисление индексного выражения аналогично такому же вычислению для переменной с индексом (см. раздел 3.1.4.2). Значение указателя переключателя определено только в том случае, когда индексное выражение принимает положительные значения *1*, *2*, *3*, ..., *n*, где *n* есть число членов в переключательном списке.

3.5.5. Целые без знака в качестве меток.

Целые без знака, используемые в качестве меток, обладают тем свойством, что приписывание впереди нулей не изменяет их значения, т. е. *00217* обозначает ту же метку, что и *217*.

4. Операторы

Единицы действий в языке называются операторами. Обычно они выполняются в той последовательности, в которой они написаны. Однако эта последовательность выполнения может прерываться операторами перехода, которые точно определяют преемника данного оператора. Последовательность выполняемых операторов может сокращаться условными

операторами, которые иногда заставляют пропустить некоторые операторы. Для того чтобы имелась возможность фактически указывать порядок следования операторов в процессе работы, операторы должны снабжаться метками. Ввиду того что последовательность операторов может группироваться в составные операторы и блоки, определение операторов, по необходимости, должно быть рекурсивным. Так как описания, уточняемые в разделе 5, существенно входят в синтаксическую структуру, то синтаксическое описание операторов предполагает, 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⟩ | ⟨оператор цикла⟩ |
⟨составной оператор⟩ | ⟨блок⟩ ⟨оператор⟩ : : = ⟨безусловный оператор⟩ |
⟨условный оператор⟩ ⟨конец составного⟩ : : = ⟨оператор⟩ **end** |
⟨оператор⟩; ⟨конец составного⟩
⟨начало блока⟩ : : = **begin** ⟨описание⟩ | ⟨начало блока⟩; ⟨описание⟩
⟨непомеченный составной⟩ : : = **begin** ⟨конец составного⟩
⟨непомеченный блок⟩ : : = ⟨начало блока⟩; ⟨конец составного⟩
⟨составной оператор⟩ : : = ⟨непомеченный составной⟩ |
⟨метка⟩ : ⟨составной оператор⟩
⟨блок⟩ : : = ⟨непомеченный блок⟩ | ⟨метка⟩ : ⟨блок⟩

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

Составной оператор:

$L : L : \ldots$ **begin** S; S; $\ldots S$; S **end**

Блок:

$L : L : \ldots$ **begin** D; D; $\ldots D$; S; S; $\ldots S$; S **end**

При этом нужно помнить, что каждый из операторов S может, в свою очередь, быть полным составным оператором и блоком.

4.1.2. Примеры.

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 and k*

end *block Q*

4.1.3. Семантика.

Каждый блок автоматически вводит новый уровень обозначений. Это реализуется таким образом, что любой идентификатор, встречающийся внутри блока, может быть путем соответствующего описания (см. раздел 5. Описания) локализован внутри блока. Это означает следующее: а) объект, представленный этим идентификатором внутри данного блока, не существует вне блока; б) любой объект, представленный тем же идентификатором вне данного блока, недоступен для блока.

Идентификаторы (за исключением тех, которые изображают метки), встречающиеся внутри блока и не описанные в нем, не будут локализованы в блоке, т. е. будут представлять те же самые объекты как внутри блока, так и на уровне, который следует непосредственно за данным блоком. Исключение из этого правила представляют только метки, которые всегда локализуются внутри того блока, в котором они встречаются.

Так как оператор блока может в свою очередь тоже быть блоком, то концепция локализации и нелокализации понимается рекурсивно. Таким образом, идентификатор, который не локализован в блоке А, может быть локализован или не локализован в блоке В, для которого А является одним из его операторов.

4.2. *Операторы присваивания*

4.2.1. Синтаксис.

⟨левая часть⟩ : : = ⟨переменная⟩ : =

⟨список левой части⟩ : : = ⟨левая часть⟩|⟨список левой части⟩ ⟨левая часть⟩

⟨оператор присваивания⟩ : : = ⟨список левой части⟩ ⟨арифметическое выражение⟩|⟨список левой части⟩ ⟨булевское выражение⟩

4.2.2. Примеры.

$s := p[0] := n := n + 1 \uparrow s$

$n := n + 1$

$A := B/C - v - q \times S$

$s[v, k + 2] := 3 - arctan(s \times zeta)$

$V := Q > Y \wedge Z$

4.2.3. Семантика.

Операторы присваивания служат для присваивания значения выражения одной или нескольким переменным. Этот процесс в общем случае понимается как проходящий в 3 этапа:

4.2.3.1. Любые индексные выражения, встречающиеся в переменных левой части, вычисляются последовательно слева направо.

4.2.3.2. Вычисляется значение выражения.

4.2.3.3. Значение выражения присваивается всем переменным левой части с любым индексным выражением, имеющим значение, вычисленное на шаге 4.2.3.1.

4.2.4. Типы.

Все переменные списка левой части должны по описанию иметь одинаковые типы. Если переменные имеют тип **Boolean**, выражение также

должно быть типа **Boolean**. Если переменные имеют тип **real** или **integer**, выражение должно быть арифметическим. Если тип арифметического выражения отличается от типа переменных, то считается, что автоматически включается соответствующая преобразующая функция. Для преобразования из типа **real** к типу **integer** функция преобразования выдает результат, эквивалентный

entier (E + 0.5),

где E — значение выражения.

4.3. *Операторы перехода*

4.3.1. Синтаксис.

⟨оператор перехода⟩ : : = **go to** ⟨именующее выражение⟩

4.3.2. Примеры.

go to *8*
go to *exit* [*n* + *1*]
go to *Town* [**if** *y* < *0* **then** *N* **else** *N* + *1*]
go to if *Ab* < *c* **then** *17* **else** *q* [**if** *w* < *0* **then** *2* **else** *n*]

4.3.3. Семантика.

Операторы перехода прерывают естественную последовательность выполнения операторов, определяемую порядком их написания. Они определяют следующий выполняемый оператор по значению именующего выражения. Таким образом, следующим выполняемым оператором будет тот, который имеет это значение в качестве своей метки.

4.3.4. Ограничение.

Так как меткам присуща локализованность, то ни один оператор перехода не может вести извне в блок.

4.3.5. Переход при неопределенном указателе переключателя.

В том случае, когда именующее выражение есть указатель переключателя, значение которого не определено, оператор перехода эквивалентен пустому оператору.

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 никакой операции. Он может служить для помещения метки.

4.5. *Условные операторы*

4.5.1. Синтаксис.

⟨условие⟩ : : = **if** ⟨булевское выражение⟩ **then**
⟨безусловный оператор⟩ : : = ⟨основной оператор⟩ | ⟨оператор цикла⟩ |
⟨составной оператор⟩ | ⟨блок⟩
⟨оператор «если»⟩ : : = ⟨условие⟩ ⟨безусловный оператор⟩ |
⟨метка⟩ : ⟨оператор «если»⟩
⟨условный оператор⟩ : : = ⟨оператор «если»⟩ |
⟨оператор «если»⟩ **else** ⟨оператор⟩

4.5.2. Примеры.

if $x > 0$ **then** $n := n + 1$

if $v > u$ **then** $V : q := n + m$ **else go to** R

if $s < 0 \vee P \leqslant Q$ **then** AA : **begin** *if* $q < v$ **then** $a := v/s$
else $y := 2 \times a$ **end else if** $v > s$ **then** $a := v - q$
else if $v > s - 1$ **then go to** S

4.5.3. Семантика.

Условные операторы приводят к пропуску или выполнению некоторых операторов в зависимости от текущего значения указанных булевских выражений.

4.5.3.1. Оператор «если».

Безусловный оператор, входящий в оператор «если», будет выполняться в том случае, когда булевское выражение, входящее в условие, принимает значение **true**; в противном случае он будет пропускаться, так что процесс выполнения продолжится начиная со следующего оператора.

4.5.3.2. Условные операторы.

Согласно синтаксису, возможны два различных типа условных операторов. Эти типы могут быть проиллюстрированы следующими примерами:

if *B1* **then** *S1* **else if** *B2* **then** *S2* **else** *S3*; *S4*

и

if *B1* **then** *S1* **else if** *B2* **then** *S2* **else if** *B3* **then** *S3*; *S4*

Здесь *B1*, *B2*, *B3* — булевские выражения, а *S1*, *S2*, *S3* — безусловные операторы. *S4* — оператор, следующий за полным условным оператором.

Выполнение условного оператора может быть описано следующим образом: последовательно, слева направо, вычисляются значения булевских выражений, стоящих в условиях. Вычисления продолжаются до тех пор, пока не будет найдено выражение, имеющее значение **true**. В последнем случае выполняется безусловный оператор, следующий непосредственно за этим выражением. Если этот оператор сам не определяет себе своего преемника, то следующим выполняемым оператором будет *S4*, т. е. оператор, следующий за полным условным оператором. Таким образом, действие ограничителя **else** можно описать, сказав, что он определяет в качестве преемника оператора, за которым он (ограничитель.— *Прим. ред. перевода*) следует, оператор, следующий за полным условным оператором.

Конструкция

else ⟨безусловный оператор⟩

эквивалентна конструкции

else if true then ⟨безусловный оператор⟩

Если ни одно из булевских выражений, входящих в условия, не имеет значения **true**, то результат работы всего условного оператора будет эквивалентен рвботе пустого оператора.

Для дальнейших пояснений может быть полезна следующая схема:

if *B1* **then** *S1* **else if** *B2* **then** *S2* **else** *S3*; *S4*

B1 имеет значение **false** *B2* имеет значение **false**

4.5.4. Переход внутрь условного оператора.

Результат работы оператора перехода, ведущего внутрь условного

оператора, непосредственно следует из объясненного выше действия разделителя **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** ⟨булевское выражение⟩

⟨список цикла⟩ : : = ⟨элемент списка цикла⟩ |
⟨список цикла⟩, ⟨элемент списка цикла⟩

⟨заголовок цикла⟩ : : = **for** ⟨переменная⟩ : = ⟨список цикла⟩ **do**

⟨оператор цикла⟩ : : = ⟨заголовок цикла⟩ ⟨оператор⟩ |
⟨метка⟩ : ⟨оперетор цикла⟩

4.6.2. Примеры.

for $q := 1$ **step** s **until** n **do** $A[q] := B[q]$
for $k := 1$, $V1 \times 2$ **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 заставлнет следующий за ним оператор повторно выполняться нуль или более раз. Кроме того, он осуществляет последовательные присвоения значений переменной, управляемой данным заголовком *. Этот процесс может быть пояснен следующей схемой:

начало; проверка; оператор S; продвижение; следующий оператор
если список цикла исчерпан

В этой схеме слово «начало» означает: произвести первое присваивание заголовком цикла. «Продвижение» означает: произвести следующее присваивание заголовком цикла. «Проверка» проверяет, было ли сделано последнее присваивание. Если это так, то выполнение продолжается с оператора, следующего за оператором цикла; в противном случае выполняется оператор, следующий 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. Последовательность значений, порождаемая каждой из трех синтансически возможных компонент среди элементов списка цикла, и соответствующее выполнение оператора S даются следующими правилами:

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

* В дальнейшем это выражение будет заменено привычным для нашей терминологии выражением «параметр цикла». (*Прим. ред. перевода.*)

4.6.4.2. Элемент типа арифметической прогрессии. Элемент цикла, имеющий вид *A* **step** *B* **until** *C*, задает порядок выполнения, который наиболее точно можно описать в терминах операторов АЛГОЛа следующим образом:

$V := A$;
L1: **if** $(V - C) \times sign\,(B) > 0$ **then go to** *элемент исчерпан*;
Оператор S;
$V := V + B$;
go to *L1*;

где V — параметр цикла и «элемент исчерпан» означает переход либо к следующему члену списка элементов цикла, либо, если данный элемент стоит последним в списке цикла, к следующему оператору программы.

4.6.4.3. Элемент типа пересчета. Выполнение, управляемое элементом списка цикла вида E **while** F, наиболее точно может быть описано в терминах операторов АЛГОЛа следующим образом:

L3: $V := E$;
if ¬ F **then go to** *элемент исчерпан*;
Оператор S;
go to *L3*;

где обозначения те же, что и в 4.6.4.2.

4.6.5. Значение параметра цикла после окончания.

После выхода из оператора *S* (если предполагается, что он является составным оператором) посредством какого-либо оператора перехода значение параметра цикла будет таким, каким оно было непосредственно перед выполнением оператора перехода.

С другой стороны, если выход из цикла вызван исчерпанием списка цикла, то значение параметра цикла не определено.

4.6.6. Операторы перехода, ведущие в оператор цикла.

Результат действия оператора перехода, обращающегося к метке в операторе цикла и действующего извне цикла, не определен.

4.7. *Операторы процедур*

4.7.1. Синтаксис.

⟨фактический параметр⟩ : : = ⟨строка⟩ | ⟨выражение⟩ |
⟨идентификатор массива⟩ | ⟨идентификатор процедуры⟩ |
⟨идентификатор переключателя⟩
⟨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 фактичесиих параметров⟩

4.7.2. Примеры.

След (*A*) *Порядок*: (7) *Результат*: (*V*)

Транспонирование (*W*, *v* + *I*)

Абсмакс (*A*, *N*, *M*, *Yy*, *I*, *K*)

Скалярное произведение (*A*[*t*, *P*, *u*], *B*[*P*], *10*, *P*, *Y*)

Эти примеры соответствуют примерам, данным в 5.4.2.

4.7.3. Семантика.

Оператор процедуры служит для обращения к выполнению тела процедуры (см. раздел 5.4. *Описания процедур*). В том случае, когда тело процедуры является оператором, написанным в языке АЛГОЛ, результат его выполнения будет эквивалентен результату следующих действий в программе:

4.7.3.1. Присваивание значений (подстановка значений). Всем формальным параметрам, перечисленным в списке значений заголовка описания процедуры, присваиваются значения (см. раздел 2.8. *Значения и типы*) соответствующих фактических параметров. Предполагается, что эти присвоения производятся непосредственно перед входом в тело процедуры. Слодовательно, эти формальные параметры будут рассматриваться так, как если бы они были локализованы в блоке, образующем тело процедуры.

4.7.3.2. Замена наименований (подстановка наименований). Любой формальный параметр, не перечисленный в списке значений, повсюду в теле процедуры заменяется на соответствующий фактический параметр после заключения последнего там, где это синтаксически возможно, в скобки. Возможность коллизий между идентификаторами, вставляемыми в тело процедуры и результате такого процесса, и идентификаторами, уже присутствующими в теле процедуры, устраняется соответствующими систематическими изменениями тех формальных или локализованных идентификаторов, которые могут быть задеты такими коллизиями.

4.7.3.3. Подстановка тела процедуры и выполнение. Наконец, тело процедуры, модифицированное так, как это было описано выше, помещается на место оператора процедуры и выполняется.

4.7.4. Соответствие между формальными и фактическими параметрами.

Соответствие между фактическими параметрами оператора процедуры и формальными параметрами заголовка процедуры устанавливается следующим образом. Список фактических параметров оператора процедуры должен иметь то же число членов, что и список формальных параметров заголовка описания процедуры. Соответствие получается путем сопоставления членов этих двух списков в одном и том же порядке.

4.7.5. Ограничения.

Очевидно, что для определяемого оператора процедуры необходимо, чтобы действия над телом процедуры, определенные в 4.7.3.1 и 4.7.3.2, приводили бы к правильному оператору в языке АЛГОЛ.

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 пара-

метра. Некоторые важные частные случаи этого общего правила приведены ниже:

4.7.5.1. Строки не могут фигурировать в качестве фактических параметров в тех операторах процедурах, которые обращаются к процедурам, описанным в терминах операторов АЛГОЛа (см. раздел 4.7.8).

4.7.5.2. Формальному параметру, входящему в тело процедуры в виде левой части некоторого оператора присваивания и не указанному в списке значений в заголовке процедуры, может соответствовать в качестве фактического параметра только переменная (частный случай выражения).

4.7.5.3. Формальному параметру, входящему в тело процедуры в качестве идентификатора массива, может соответствовать в качестве фактического параметра только идентификатор массива той же самой размерности. Кроме того, если формальный параметр указан в списке значений в заголовке процедуры, то локализованный массив, который появится в теле процедуры в результате обращения, получит те же самые границы индексов, что и фактический массив.

4.7.5.4. Формальному параметру, который указан в списке значений в заголовке процедуры, не может соответствовать какой-либо идентификатор указателя переключателя или процедуры, поскольку последние не могут принимать каких-либо значений. (Исключение составляют идентификаторы процедур, описания которых содержат пустую совокупность формальных параметров (см. раздел 5.4.1) и которые определяют значение функции (см. раздел 5.4.4). Идентификаторы таких процедур сами по себе являются законченными выражениями).

4.7.5.5. Любой формальный параметр может налагать ограничения на тип соответствующего связанного с ним фактического параметра (эти ограничения могут быть как указаны, так и не указаны в заголовке процедуры). Очевидно, что в операторе процедуры эти ограничения должны быть соблюдены.

4.7.5. Нелокализованные величины тела процедуры.

Оператор процедуры, который записан вне области действия любой нелокализованной величины процедуры, не определен.

4.7.7. Ограничители параметров.

Предусматривается, что все ограничители параметров эквивалентны. Не устанавливается никакого соответствия между ограничителями параметров, используемыми в операторе процедуры, и ограничителями, фигурирующими в заголовке процедуры, кроме лишь того, что их количество должно быть одинаковым. Таким образом, вся информация, которая извлекается из сложных ограничителей, является избыточной.

4.7.8. Тело процедуры, описанное в других кодах.

Ограничения, налагаемые на операторы процедуры, производящие обращения к процедурам, тело которых выражено не на языке АЛГОЛ, очевидно, могут следовать лишь из характеристик применяемого языка и из особых соображений работающих с языком, так что все они остаются вне рамок эталонного языка.

5. Описания

Описания служат для определения некоторых свойств идентификаторов программы. Описание идентификатора имеет силу только в одном блоке. Вне данного блока конкретный идентификатор может использоваться для других целей (см. раздел 4.1.3.).

В динамике работы программы это влечет за собой следующее: с момента входа в блок (через **begin**, так как внутренние метки локализованы и, следовательно, недостижимы извне) все идентификаторы, описанные в блоке, употребляются в значении, соответствующем природе 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 прежний смысл.

К моменту выхода из блока (через **end** или оператор перехода) все идентификаторы, которые были описаны в блоке, вновь теряют свой смысл.

Описания могут отмечаться добавочным описателем (собственный). Это приводит к следующему действию: к моменту возвращения в блок значения собственных величин сохраняются такими же, какими они были после последнего выхода, в то время как значения описанных величин, которые не были отмечены описателем **own**, будут не определены. Все идентификаторы программы, за исключением меток и, возможно, стандартных функций (см. раздел 3.2.4), должны быть описаны. Ни один идентификатор в блоке не должен быть описан более чем один раз.

Синтаксис.

⟨**описание**⟩ : : = ⟨описание типа⟩ | ⟨описание массива⟩ |
⟨описание переключателя⟩ | ⟨описание процедуры⟩

5.1. *Описание типов*

5.1.1. Синтаксис.

⟨список типа⟩ : : = ⟨простая переменная⟩ |
⟨простая переменная⟩,⟨список типа⟩

⟨**тип**⟩ : : = **real** | **integer** | **Boolean**

⟨**локализованный или собственный тип**⟩ : : = ⟨**тип**⟩ | **own** ⟨**тип**⟩

⟨**описание типа**⟩ : : = ⟨**локализованный или собственный тип**⟩ ⟨список типа⟩

5.1.2. Примеры.

integer *p*, *q*, *s*
own Boolean *Acryl*, *n*

5.1.3. Семантика.

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

В арифметическом выражении любая позиция, которая может быть

занята переменной типа **real**, может быть занята и переменной типа **integer**.

Семантика описателя **own** приведена в четвертом абзаце раздела 3.

5.2. *Описание 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⟩
⟨сегмент массива⟩ : : = ⟨идентификатор массива⟩ [⟨список граничных пар⟩] |
⟨идентификатор массива⟩, ⟨сегмент массива⟩
⟨список массивов⟩ : : = ⟨сегмент массива⟩ |
⟨список массивов⟩, ⟨сегмент массива⟩
⟨описание массивов⟩ : : = **array** ⟨список массивов⟩ |
⟨локализованный или собственный тип⟩ **array** ⟨список массивов⟩

5.2.2. Примеры.

array *a*, *b*, *c* [*7* : *n*, *2* : *m*], *s* [— *2* : *10*]
own integer array *A* [**if** *c* < *0* **then** *2* **else** *1* : *20*]
real array *q* [— *7* : — *1*]

5.2.3. Семантика.

Описания массивов описывают один или несколько идентификаторов, представляющих многомерные массивы переменных с индексами, и указывают размерности массивов, границы индексов и типы переменных.

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

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

5.2.3.3. Типы. Все массивы, описанные в одном описании, имеют один и тот же предписываемый тип. Если описание типа не дано, то считается, что предписан тип **real**.

5.2.4. Выражения для границ индексов.

5.2.4.1. Эти выражения вычисляются так же, как индексное выражение (см. раздел 3.1.4.2).

5.2.4.2. Каждое выражение может зависеть только от тех переменных и процедур, которые не локализованы в том блоке, для которого имеет силу описание массива. Из этого следует, что в наибольшем блоке программы могут быть описания массивов только с постоянными границами.

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

5.2.4.4. Выражения для границ вычисляются заново при каждом новом входе в блок.

5.2.5. Идентичность переменных с индексами.

Идентичность переменных с индексами не связана с границами индексов, данными в описании массива. Однако даже если массиву приписано свойство **own**, значения соответствующих переменных с индексами будут в любой момент времени определены только для тех из этих переменных, у которых значения индексов лежат в пределах границ, вычисленных в последний раз.

5.3. *Описания переключателей*

5.3.1. Синтаксис.

⟨переключательный список⟩ : : = ⟨именующее выражение⟩ |
⟨переключательный список⟩,⟨именующее выражение⟩

⟨описание переключателя⟩: : = **switch** ⟨идентификатор переключателя⟩ : =
⟨переключательный список⟩

5.3.2. Примеры.

switch $S := S1, S2, Q\,[m]$, **if** $v > -5$ **then** $S3$ **else** $S4$

switch $Q := p1, w$

5.3.3. Семантика.

Описание переключателя задает значения, соответствующие идентификатору переключателя. Эти значения даны одно за другим, как значения именующих выражений, перечисленных в переключательном списке. С каждым из этих именующих выражений связано положительное целое число $1,2,\ldots$, равное порядковому номеру этого выражения, получаемому при счете слева направо. Значением указателя переключателя, соответствующим данному значению индексного выражения (см. раздел 3.5. *Именующие выражения*), является значение именующего выражения в переключательном списке, имеющее данное значение индексного выражения своим порядковым номером.

5.3.4. Вычисление выражений в переключательном списке.

Выражение, входящее в переключательный список, будет вычисляться каждый раз, когда происходит обращение к члену списка, в который входит данное выражение. При вычислении выражения используются текущие значения всех входящих в него переменных.

5.3.5. Влияние области действия.

Любое обращение к значению указателя переключателя извне области действия любой величины, входящей в именующее выражение для этого значения, не определено.

5.4. *Описание 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⟩ | ⟨код⟩ *

⟨описание процедуры⟩ : : = **procedure** ⟨заголовок процедуры⟩ ⟨тело процедуры⟩ | ⟨тип⟩ **procedure** ⟨заголовок процедуры⟩ ⟨тело процедуры⟩

5.4.2. Примеры (см. также примеры в конце сообщения).

```
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 := 1 + i 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) Порядок: (n, m) Результат: (y) Индексы (i, k);
comment: Наибольший по абсолютной величине элемент матрицы a,
размером n на m, передается в y. 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;
```

* ⟨код⟩ — не определенная синтаксическая единица. Подразумевается произвольная запись тела процедуры средствами, лежащими вне языка АЛГОЛ-60. (*Прим. ред. перевода.*)

begin real s;
$s := 0$;
for $p := 1$ **step** 1 **until** k **do** $s := s + a \times b$;
$y := s$
end *скалярного произведения* *

5.4.3. Семантика.

Описание процедуры служит для задания процедуры, связанной с идентификатором процедуры. Главной компонентой описания процедуры является оператор или группа кодов, представляющие тело процедуры, к которой может быть произведено обращение посредством функции или (и) оператора процедуры. Это обращение может быть произведено из различных частей блока, в начале которого находится описание данной процедуры. С каждым телом процедуры связан заголовок процедуры, который снабжает определенной информацией некоторые идентификаторы, встречающиеся в теле процедуры и представляющие формальные параметры. В момент обращения к процедуре (см. раздел 3.2. *Функции* и раздел 4.7. *Операторы процедур*) формальным параметрам в теле процедуры будут присвоены некоторые значения или же они будут заменены некоторыми фактическими параметрами. Те идентификаторы в теле процедуры, которые не являются формальными параметрами, могут быть либо локализованы, либо не локализованы, в зависимости от того, описаны они в теле процедуры или нет. Те из них, которые не локализованы в теле процедуры, могут быть локализованы в блоке, в начале которого находится описание данной процедуры.

5.4.4. Значения функций.

Для того чтобы описание процедуры определяло функцию, необходимо, чтобы в теле процедуры встречалось присваивание значения идентификатору процедуры. Кроме того, первым символом описания процедуры должен быть в этом случае описатель типа, описывающий тип значения процедуры.

Любое другое вхождение идентификатора некоторой процедуры P в тело данной процедуры означает обращение к процедуре P.

5.4.5. Спецификации.

В заголовок процедуры может быть включена совокупность спецификаций, задающая с помощью очевидных обозначений информацию о классах и типах формальных параметров. В этой части заголовка ни одна формальный параметр не может встречаться более одного раза. Кроме того, формальные параметры, которые не входят в список значений в заголовке процедуры (см. раздел 4.7.3.2), всегда могут быть опущены.

5.4.6. Код в качестве тела процедуры.

Ясно, что тело процедуры может быть выражено и не в языке АЛГОЛ. Так как предполагается, что использование этой возможности является целиком вопросом конкретного представления, то никаких дальнейших правил относительно языка этого кода в эталонном языке не может быть дано.

---

* Ограничитель параметра)Порядок:(может ввести читателя в заблуждение, поскольку порядком является только k; p — параметр, обозначающий индекс суммирования. (*Прим. ред. перевода.*)

Примеры описаний процедур

Пример 1

procedure *эйлер* (*fct*, *sum*, *eps*, *tim*); **value** *eps*, *tim*; **integer** *tim*; **real procedure** *fct*; **real** *sum*, *eps*;
comment *Процедура эйлер вычисляет сумму fct (i) для i от нуля до бесконечности посредством надлежащим образом усовершенствованного метода эйлеровской трансформации. Суммирование прекращается, как только подряд tim раз абсолютные значения членов преобразованных рядов будут меньше, чем eps. Следовательно, необходимо задать функцию fct с одним целым аргументом, нижнюю границу eps и целую величину tim. Результатом является сумма sum. Процедура эйлер особенно эффективна в случае медленно сходящихся и расходящихся рядов*;
begin integer *i*, *k*, *n*, *t*; **array** *m* [*0* : *15*]; **real** *mn*, *mp*, *ds*;
i := *n* := *t* := *0*; *m* [*0*] := *fct* (*0*); *sum* := *m* [*0*] / *2*;
продолжение: *i* := *i* + *1*; *mn* := *fct* (*i*);
for *k* := *0* **step** *1* **until** *n* **do**
begin *mp* := (*mn* + *m* [*k*]) / *2*; *m* [*k*] := *mn*; *mn* := *mp* **end**
if (*abs* (*mn*) < *abs* (*m* [*n*])) ∧ (*n* < *15*) **then**
begin *ds* := *mn* / *2*; *n* := *n* + *1*; *m* [*n*] := *mn* **end**
else *ds* := *mn*;
sum := *sum* + *ds*;
if *abs* (*ds*) < *eps* **then** *t* := *t* + *1* **else** *t* := *0*;
if *t* < *tim* **then go to** *продолжение*
end *эйлер*

Пример 2*

procedure *RK* (*x*, *y*, *n*, *FKT*, *eps*, *eta*, *xE*, *yE*, *fi*); **value** *x*, *y*; **integer** *n*; **Boolean** *fi*; **real** *x*, *eps*, *eta*, *xE*; **array** *y*, *yE*; **procedure** *FKT*;
comment: *RK интегрирует систему* $y'_k = f_k(x, y_1, y_2, \ldots, y_n)$ $(k = 1, 2, \ldots, n)$ *дифференциальных уравнений методом Рунге — Кутта с автоматическим выбором шага. Параметрами являются: начальные начения x и y* [*k*] *для x и неизвестных функций* $y_k(x)$; *порядок системы n; процедура FKT* (*x*, *y*, *n*, *z*), *представляющая интегрируемую систему, т. е. совокупность функций* f_k; *подходящие значения eps и eta, которые определяют точность численного интегрирования; конец интервала интегрирования xE; выходной параметр yE, который представляет решение в точке* $x = xE$; *булевская переменная fi, которой при изолированном или первом обращении должно быть присвоено значение* **true**. *Если, однако, необходимо получить значения функции в нескольких промежуточных точках* $x_0, x_1, \ldots, x_n$, *то процедура должна вызываться повторно* (*с* $x = x_k$, $xE = x_{k+1}$ *для* $k = 0, 1, \ldots, n - 1$), *при этом последующие обращения для экономии машинного времени можна осуществить с fi* = **false**. *Входными параметрами FKT должны быть x, y, n. Выход-*

* Эта *RK*-программа содержит некоторые новые идеи, связанные с идеями Гилла [4] и Фрёберга [5]. Однако должно быть ясно, что с точки зрения времени вычислений и ошибок округления она, возможно, не будет оптимальной. Кроме того, она не была фактически проверена на машине.

ной параметр z представляет собой совокупность производных $z[k] = f_k(x, y[1], y[2], \ldots, y[n])$ *для x и фактических y. Процедура comp входит как нелокализованный идентификатор;*

```
begin
    array z, y1, y2, y3[1:n]; real x1, x2, x3, H; Boolean out;
    integer k, j; own real s, Hs;
    procedure RK1ST (x, y, h, xe, ye); real x, h, xe; array y, ye;
        comment: RK1ST интегрирует один шаг методом Рунге — Кутта с
        начальными значениями x, y[k]. Входными параметрами являются
        xe = x + h и ye[k], причем последний вектор есть решение в точке xe.
          Важно: параметры n, FKT, z входят в RK1ST как нелокализованные объекты;
        begin
            array w[1:n], a[1:5]; integer k, j;
            a[1]:=a[2]:=a[5]:=h/2; a[3]:=a[4]:=h; xe:=x;
            for k:=1 step 1 until n do ye[k]:=w[k]:=y[k];
            for j:=1 step 1 until 4 do
            begin
                FKT (xe, w, n, z);
                xe:=x+a[j];
                for k:=1 step 1 until n do
                begin
                    w[k]:=y[k]+a[j]×z[k];
                    ye[k]:=ye[k]+a[j+1]×z[k]/3
                end k
            end j
        end RK1ST;
Начало программы:
    if fi then begin H:=xE-x; s:=0 end else H:=Hs;
    out:=false;
AA: if (x+2.01×H-xE>0)≡(H>0) then
    begin Hs:=H; out:=true; H:=(xE-x)/2 end if;
    RK1ST (x, y, 2×H, x1, y1);
BB: RK1ST (x, y, H, x2, y2); RK1ST (x2, y2, H, x3, y3);
    for k:=1 step 1 until n do
        if comp(y1[k], y3[k], eta)>eps then go to CC;
    comment: comp(a, b, c) — функция, значением которой является абсолютное
    значение разности мантисс a и b, после того как порядки этих величин
    выравнены до наибольшего порядка параметров a, b, c;
    x:=x3; if out then go to DD;
    for k:=1 step 1 until n do y[k]:=y3[k];
    if s=5 then begin s:=0; H:=2×H end if;
    s:=s+1; go to AA;
CC: H:=0.5×H; out:=false; x1:=x2;
    for k:=1 step 1 until n do y1[k]:=y2[k];
    go to BB;
```

DD: **for** *k* := *1* **step** *1* **until** *n* **do** *yE* [*k*] := *y3* [*k*]
end *RK*

Алфавитный указатель определяемых понятий и синтаксических единиц*

Все ссылки даются на номера разделов. Ссылки разбиты на три группы:

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

синт — ссылка, стоящая за сокращением «синт», отсылает к вхождению в металингвистическую формулу. Ссылки, перечисленные в группе «опр», не повторяются.

текст — ссылка, стоящая за словом «текст», отсылает к определению, данному в тексте.

При составлении указателя примеры не принимались во внимание.

I. Указатель символов и терминов языка

+	см. плюс
−	см. минус
×	см. умножение
/ ÷	см. деление
↑	см. возведение в степень
<, ⩽, =, ⩾, >, ≠	см. ⟨операция отношения⟩
≡, ⊃, ∨, ∧, ¬	см. ⟨логическая операция⟩
,	см. запятая
.	см. десятичная точка
10	см. десять
:	см. двоеточие
;	см. точка с запятой
: =	см. двоеточие равенство
␣	см. пробел
( )	см. скобки
[ ]	см. индексные скобки
' '	см. кавычки для строк
array	синт 2.3, 5.2.1, 5.4.1
begin	синт 2.3, 4.1.1
Boolean	синт 2.3, 5.1.1, текст 5.1.3
comment	синт 2.3
do	синт 2.3, 4.6.1
else	синт 2.3, 3.3.1, 3.4.1, 3.5.1, 4.5.1, текст 4.5.3.2
end	синт 2.3, 4.1.1
false	синт 2.2.2
for	синт 2.3, 4.6.1
go to	синт 2.3, 4.3.1

* Ввиду того что лексикографический порядок английских и русских терминов не совпадает, данный указатель разделен на две части: I — указатель символов и терминов языка (последние расположены в порядке латинского алфавита) и II — указатель определяемых металингвистических терминов и терминов, употребляемых в тексте. (*Прим. перев.*)

if синт 2.3, 3.3.1, 4.5.1
integer синт 2.3, 5.1.1, текст 5.1.3
label синт 2.3, 5.4.1
own синт 2.3, 5.1.1, текст 5, 5.2.5
procedure синт 2.3, 5.4.1
real синт 2.3, 5.1.1, текст 5.1.3
step синт 2.3, 4.6.1, текст 4.6.4.2
string синт 2.3, 5.4.1
switch синт 2.3, 5.3.1, 5.4.1
then синт 2.3, 3.3.1, 4.5.1
true синт 2.2.2
until синт 2.3, 4.6.1, текст 4.6.4.2
value синт 2.3, 5.4.1
while синт 2.3, 4.6.1, текст 4.6.4.3

II. Указатель определяемых металингвистических терминов и терминов, употребляемых в тексте

алфавит — текст 2.1
арифметический — текст 3.3.6
⟨арифметическая операция⟩ — опр 2.3, текст 3.3.4
⟨арифметическое выражение⟩ — опр 3.3.1, синт 3, 3.1.1, 3.3.1, 3.4.1, 4.2.1, 4.6.1, 5.2.1, текст 3.3.3
⟨безусловный оператор⟩ — опр 4.1.1, 4.5.1
⟨блок⟩ — опр 4.1.1, синт 4.5.1, текст 1, 4.1.3, 5
⟨буква⟩ — опр 2.1, 2, 2.4.1, 3.2.1, 4.7.1
⟨булевский множитель⟩ — опр 3.4.1
⟨булевский терм⟩ — опр 3.4.1
⟨булевское выражение⟩ — опр 3.4.1, синт 3, 3.3.1, 4.2.1, 4.5.1, 4.6.1, текст 3.4.3
величина — текст 2.7
⟨верхняя граница⟩ — опр 5.2.1, текст 5.2.4
возведение в степень ↑ — синт 2.3, 3.3.1, текст 3.3.4.3
⟨вторичное булевское выражение⟩ — опр 3.4.1
⟨выражение⟩ — опр 3, синт 3.2.1, 4.7.1, текст 3 (весь раздел)
границы индексов — текст 5.2.3.1
⟨граничная пара⟩ — опр 5.2.1
двоеточие : — синт 2.3, 3.2.1, 4.1.1, 4.5.1, 4.6.1, 4.7.1, 5.2.1
двоеточие равенство : = — синт 2.3, 4.2.1, 4.6.1, 5.3.1
деление /÷ — синт 2.3, 3.3.1, текст 3.3.4.2
⟨десятичная дробь⟩ — опр 2.5.1
десятичная точка . — синт 2.3, 2.5.1
⟨десятичное число⟩ — опр 2.5.1, текст 2.5.3
⟨десятичный порядок⟩ — опр 2.5.1, текст 2.5.3
десять $_{10}$ — синт 2.3, 2.5.1
⟨заголовок процедуры⟩ — опр 5.4.1, текст 5.4.3
⟨заголовок цикла⟩ — опр 4.6.1, текст 4.6.3
запятая ,— синт 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

значение — текст 2.8, 3.3.3
⟨идентификатор⟩ — опр. 2.4.1, синт 3.1.1, 3.2.1, 3.5.1, 5.4.1, текст 2.4.3
⟨идентификатор массива⟩ — опр 3.1.1, синт 3.2.1, 4.7.1, 5.2.1, текст 2.3
⟨идентификатор переменной⟩ — опр 3.1.1
⟨идентификатор переключателя⟩ — опр 3.5.1, синт 3.2.1, 4.7.1, 5.3.1
⟨идентификатор процедуры⟩ — опр 3.2.1, синт 3.2.1, 4.7.1, 5.4.1, текст 4.7.5.4
⟨именующее выражение⟩ — опр 3.5.1, синт 3, 4.3.1, 5.3.1, текст 3.5.3
⟨импликация⟩ — опр 3.4.1
индекс — текст 3.1.4.1
⟨индексное выражение⟩ — опр 3.1.1, синт 3.5.1
индексные скобки [ ] — синт 2.3, 3.1.1, 3.5.1, 5.2.1
кавычки для строк ‘ ’ — синт 2.3, 2.6.1, текст 2.6.3
⟨код⟩ — синт 5.4.1, текст 4.7.8, 5.4.6
⟨конец составного⟩ — опр 4.1.1
⟨левая часть⟩ — опр 4.2.1
⟨логическая операция⟩ — опр 2.3, синт 3.4.1, текст 3.4.5
⟨логическое значение⟩ — опр 2.2.2, синт 2, 3.4.1
локализованный — текст 4.1.3
⟨локализованный или собственный тип⟩ — опр 5.1.1, синт 5.2.1
массив — текст 3.1.4.1.
⟨метка⟩ — опр 3.5.1, синт 4.1.1, 4.5.1, 4.6.1, текст 1, 4.1.3
минус — — синт 2.3, 2.5.1, 3.3.1, текст 3.3.4.1
⟨множитель⟩ — опр 3.3.1
⟨начало блока⟩ — опр 4.1.1
нелокализованный — текст 4.1.3
⟨непомеченный блок⟩ — опр 4.1.1
⟨непомеченный основной оператор⟩ — опр 4.1.1
⟨непомеченный составной⟩ — опр 4.1.1
⟨нижняя граница⟩ — опр 5.2.1, текст 5.2.4
область действия — текст 2.7
⟨ограничитель⟩ — опр 2.3, синт 2
⟨ограничитель параметра⟩ — опр 3.2.1, 4.7.1, синт 5.4.1, текст 4.7.7
⟨оператор⟩ — опр 4.1.1, синт 4.5.1, 4.6.1, 5.4.1, текст 4 (весь раздел)
⟨оператор «если»⟩ — опр 4.5.1, текст 4.5.3.1
⟨оператор перехода⟩ — опр 4.3.1, синт 4.1.1, текст 4.3.3
⟨оператор присваивания⟩ — опр 4.2.1, синт 4.1.1, текст 1, 4.2.3
⟨оператор процедуры⟩ — опр 4.7.1, синт 4.1.1, текст 4.7.3
⟨оператор цикла⟩ — опр 4.6.1, синт 4.1.1, 4.5.1, текст 4.6 (весь раздел)
операторные скобки — см. **begin end**
⟨операция⟩ — опр 2.3
⟨операция отношения⟩ — опр 2.3, 3.4.1
⟨операция следования⟩ — опр 2.3
⟨операция сложения⟩ — опр 3.3.1
⟨операция умножения⟩ — опр 3.3.1
⟨описание⟩ — опр 5, синт 4.1.1, текст 1, 5 (весь раздел)
⟨описание массива⟩ — опр 5.2.1, синт 5, текст 5.2.3
⟨описание переключателя⟩ — опр 5.3.1, синт 5, текст 5.3.3

⟨описание процедуры⟩ — опр 5.4.1, синт 5, текст 5.4.3
⟨описание типа⟩ — опр 5.1.1, синт 5, текст 5.1.3
⟨описатель⟩ — опр 2.3
⟨основной оператор⟩ — опр 4.1.1, синт 4.5.1
⟨основной символ⟩ — опр 2
⟨открытая строка⟩ — опр. 2.6.1
⟨отношение⟩ — опр 3.4.1, текст 3.4.5
⟨первичное булевское выражение⟩ — опр 3.4.1
⟨первичное выражение⟩ — опр 3.3.1
⟨переключательный список⟩ — опр 5.3.1
⟨переменная⟩ — опр. 3.1.1, синт 3.3.1, 3.4.1, 4.2.1, 4.6.1, текст 3.1.3
⟨переменная с индексом⟩ — опр 3.1.1, текст 3.1.4.1
плюс+ — синт 2.3, 2.5.1, 3.3.1, текст 3.3.4.1
правила примечаний — текст 2.3
⟨правильная строка⟩ — опр 2.6.1
преемник — текст 4
преобразующая функция — текст 3.2.5
пробел ␣ — синт 2.3, текст 2.3, 2.6.3
программа — текст 1
⟨простая переменная⟩ — опр. 3.1.1, синт 5.1.1, текст 2.4
⟨простое арифметическое выражение⟩ — опр. 3.3.1, текст 3.3.3
⟨простое булевское выражение⟩ — опр 3.4.1
⟨простое именующее выражение⟩ — опр 3.5.1
⟨пусто⟩ — опр 1.1, синт 2.6.1, 3.2.1, 4.4.1, 4.7.1, 5.4.1
⟨пустой оператор⟩ — опр 4.4.1, синт 4.1.1, текст 4.4.3
⟨разделитель⟩ — опр 2.3
размерность — текст 5.2.3.2
⟨сегмент массива⟩ — опр 5.2.1
⟨скобка⟩ — опр 2.3
скобки ( ) — синт 2.3, 3.2.1, 3.3.1, 3.4.1, 3.5.1, 4.7.1, 5.4.1, текст 3.3.5.2
⟨совокупность спецификаций⟩ — опр 5.4.1, текст 5.4.5
⟨совокупность фактических параметров⟩ — опр. 3.2.1, 4.7.1
⟨совокупность формальных параметров⟩ — опр 5.4.1
⟨составной оператор⟩ — опр 4.1.1, синт 4.5.1, текст 1
⟨спецификатор⟩ — опр 2.3
⟨спецификация⟩ — опр 5.4.1
⟨список граничных пар⟩ — опр. 5.2.1
⟨список значений⟩ — опр. 5.4.1, текст 4.7.3.1
⟨список индексов⟩ — опр 3.1.1
⟨список идентификаторов⟩ — опр 5.4.1
⟨список левой части⟩ — опр 4.2.1
⟨список массивов⟩ — опр 5.2.1
⟨список типа⟩ — опр 5.1.1
⟨список фактических параметров⟩ — опр 3.2.1, 4.7.1
⟨список формальных параметров⟩ — опр 5.4.1
⟨список цикла⟩ — опр 4.6.1, текст 4.6.4
стандартные функции — текст 3.2.4, 3.2.5
⟨строка⟩ — опр 2.6.1, синт 3.2.1, 4.7.1, текст 2.6.3

⟨строка букв⟩ — опр 3.2.1, 4.7.1
⟨тело процедуры⟩ — опр 5.4.1
⟨терм⟩ — опр 3.3.1
⟨тип⟩ — опр 5.1.1, синт 5.4.1, текст 2.8
точка с запятой ; — синт 2.3, 4.1.1, 5.4.1
⟨указатель переключателя⟩ — опр 3.5.1, текст 3.5.3
умножение × — синт 2.3, 3.3.1, текст 3.3.4.1
⟨условие⟩ — опр 3.3.1, 4.5.1, синт 3.4.1, 3.5.1, текст 3.3.3, 4.5.3.2
⟨условный оператор⟩ — опр 4.5.1, синт 4.1.1, текст 4.5.3
⟨фактический параметр⟩ — опр 3.2.1, 4.7.1
⟨формальный параметр⟩ — опр 5.4.1, текст 5.4.3
⟨функция⟩ — опр 3.2.1, синт 3.3.1, 3.4.1, текст 3.2.3, 5.4.4
целая часть — текст 3.2.5
⟨целое⟩ — опр 2.5.1, текст 2.5.4
⟨цифра⟩ — опр 2.2.1, синт 2, 2.4.1, 2.5.1
⟨целое без знака⟩ — опр 2.5.1, 3.5.1
⟨число⟩ — опр 2.5.1, текст 2.5.3, 2.5.4
⟨число без знака⟩ — опр 2.5.1, синт 3.3.1
⟨элемент списка цикла⟩ — опр 4.6.1, текст 4.6.4.1, 4.6.4.2, 4.6.4.3

Цитированная литература

1. International Algebraic Language. Communs Assoc. Comput. Machinery, 1958, **1**, № 12, 8.
2. Report on the Algorithmic Langnage ALGOL by the ACM Committee on Programming Languages and GAMM Committee on Programming. Ed. A. J. Perlis, K. Samelson. Numerische Math., 1959, **1**, 41—60; Сообщение об алгоритмическом языке АЛГОЛ. Перев. с англ. Ред. Перлис А. Дж., Замельзон К. М. ВЦ АН СССР, 1959.
3. J. W. Backns. The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM conference. ICIP Paris, June, 1959.
4. S. Gill. A process for the step by step integration of differential equation in an automatic computing machine. Proc. Cambridge Philos. Soc., 1951, **47**, 96.
5. E. Fröberg. On the solution of ordinary differential equations with digital computing machines. Fisiograf. Sällsk. Lund, Förhd., 1950, **20**, № 11, 136—152.