Алгоритмы
и алгоритмические
языки

# ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Издательство «Наука»

АКАДЕМИЯ НАУК СССР
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

Алгоритмы
и алгоритмические
языки

# ЯЗЫКИ ПРОГРАММИРОВАНИЯ

МОСКВА
«НАУКА»
1985

УДК 519.682

**Языки программирования.**— М.: Наука, 1985. (Алгоритмы и алгоритмические языки).

Сборник содержит две статьи. В первой дано описание нового алгоритмического языка Модула-2, разработанного Н. Виртом. Существует уже несколько реализаций этого языка, в том числе и для прототипов эксплуатируемых у нас машин. Вторая статья посвящена атрибутным грамматикам, получающим все большее применение при реализации языков программирования и в системах построения трансляторов.
Для специалистов в области программирования.



**ЯЗЫКИ ПРОГРАММИРОВАНИЯ**

Алгоритмы и алгоритмические языки

Утверждено к печати Вычислительным центром АН СССР

Редактор издательства **А. А. Боровая**
Художественный редактор **Н. А. Фильчагина**
*Технические редакторы* **Т. С. Жарикова, М. Л. Анучина**
Корректоры **Н. Б. Габасова, Н. И. Казарина**

ИБ № 29133

Сдано в набор 11.02.85. Подписано к печати 12.07.85. Т-14834.
Формат 60×90¹/₁₆
Бумага книжно-журнальная. Гарнитура обыкновенная. Печать высокая. Усл. печ. л. 5. Усл. кр. отт. 5,25. Уч.-изд. л. 5,7. Тираж 10100 экз.
Тип. зак. 1210. Цена 35 к.

Ордена Трудового Красного Знамени Издательство «Наука»
117864 ГСП-7, Москва В-485, Профсоюзная ул., 90.
2-я типография издательства «Наука», 121099,
Москва, Г-99, Шубинский пер., 6

Я $\frac{1702070000\text{-}405}{042(02)\text{-}85}$ 109-85-II

УДК 519.682

# МОДУЛА-2 *

Н. Вирт

Модула-2 — язык программирования общего назначения, разработанный прежде всего для реализации систем. Данное сообщение определяет его точным, хотя и неформальным образом. Оно также описывает пользование реализацией для машин серии PDP-11.

## 1. Введение

Модула-2 возникла из практической необходимости в едином языке системного программирования, допускающем эффективную реализацию на мини-машинах. Ее прародители — Паскаль [1] и Модула [2]. От последней она унаследовала имя, важную концепцию модуля и систематический современный синтаксис, от Паскаля — структуры данных (т. е. массивы, записи, вариантные записи, множества и указатели) и составные операторы, включая привычные условный, выбирающий, циклический, присоединяющий. Синтаксис операторов таков, что каждая управляющая структура оканчивается явным завершающим символом.

Язык существенно машинно-независим, за исключением ограничений, обусловленных длиной слова. Кажется, что это противоречит концепции языка системного программирования, в котором должна быть возможность выразить любую операцию, свойственную для используемой ЭВМ. Это противоречие разрешено с помощью понятия модуля. Машинно-зависимые части можно ввести в особых модулях и таким образом эффективно ограничить и изолировать их использование. В таких случаях, в частности, язык позволяет ослабить правило совместимости типов. В хорошем языке системного программирования должна быть возможность выразить процедуры ввода/вывода, подпрограммы работы с файлами, распределения памяти, управления процессами и т. п. Эти возможности не включаются как элементы в сам язык, а возникают как (так называемые низкоуровневые) модули, которые являются частью большинства программ. Поэтому такой набор стандартных модулей является существенной частью реализации Модулы-2.

---

* Пер. с англ. Л. А. Захарова.

Концепция процессов и их синхронизации сигналами, включенная в Модулу, заменена в Модуле-2 понятием более низкого уровня — сопрограммой, но можно, однако, оформить (стандартный) модуль, который такие процессы и сигналы реализует. Преимущество этого заключается в том, что программисты могут подбирать алгоритм планировщика процессов, подходящий для их частных нужд, программируя этот модуль по-своему. Этот планировщик может быть даже полностью опущен в простых (но однако частых) случаях, например когда параллельные процессы появляются только как драйверы внешних устройств.

Современный язык системного программирования должен, в частности, также облегчать создание больших программ, разрабатываемых, возможно, несколькими людьми. Модули, написанные отдельными разработчиками, должны иметь хорошо определенный интерфейс, который может быть описан независимо от фактической реализации. Модула-2 поддерживает эту идею введением самостоятельных определяющих и реализующих модулей. Первые определяют все объекты, экспортированные из соответствующих реализующих модулей (в некоторых случаях, таких, как процедуры и типы, определяющий модуль задает только те части, которые имеют значение для интерфейса, т. е. для пользователя или клиента данного модуля).

Раздел 15 данного сообщения описывает пользование реализацией Модулы-2 на машинах серии PDP-11. Эта система программирования состоит из многопроходного транслятора, редактора связей и загрузчика. Компилятор позволяет транслировать отдельные модули, которые могут быть собраны редактором связей. Результат работы редактора связей загружается для исполнения с помощью загрузчика. Проверку совместимости, например соответствие типов, между отдельно оттранслированными модулями выполняет транслятор.

В разд. 16 приведен набор некоторых широко используемых вспомогательных модулей, в частности для управления вводом и выводом, перечислены соответствующие определяющие модули с объяснениями смысла экспортированных процедур.

Данное сообщение не предназначается в качестве учебника для программистов. Оно намеренно сделано точным и, мы надеемся, понятным. Его функция — служить эталоном для программистов, разработчиков трансляторов и авторов руководств и быть арбитром, если у них возникнут разногласия.

Мне хотелось бы отметить влияние, которое язык MESA [3] оказал на разработку Модулы-2. Длительная возможность пользоваться изощренной MESA-системой показала мне, как подойти ко многим проблемам и что некоторых из них благоразумнее вообще избежать. Так, следует выразить благодарность реализаторам Модулы-2 L. Geismann, A. Gorrengourt, Ch. Jacobi, S. E. Knudsen, которые тщательно проверили рукопись и неоценимая поддержка которых помогла поставить фантазии разработчика языка на твердую основу.

## 2. Обозначения для описания синтаксиса

Для описания синтаксиса используется расширенный формализм Бэкуса-Наура (РБНФ). Синтаксические понятия (нетерминальные символы) изображаются русскими (в оригинале — английскими) словами, выражающими их интуитивный смысл. Символы языка (терминальные символы) — это или слова, написанные прописными буквами, или строки, заключенные в кавычки. Каждое синтаксическое правило (продукция) имеет вид

S = E.

где S — синтаксическое понятие, а E — синтаксическое выражение, изображающее множество сентенциальных форм (последовательностей символов), которые обозначает S. Выражение E имеет вид

T1 | T2 | . . . | Tn (n > 0),

где Ti является термом выражения E. Каждое Ti обозначает некоторое множество сентенциальных форм, а E изображает их объединение. Каждый терм T имеет вид

F1 F2 . . . Fn (n > 0),

где Fi — множитель терма T. Каждое Fi обозначает некоторое множество сентенциальных форм, а T изображает их конкатенацию. Конкатенация двух множеств последовательностей символов — это множество последовательностей, состоящее из всех таких конкатенаций, когда последовательность из первого множителя продолжается последовательностью из второго множителя. Каждый множитель F — это или символ (терминальный или нетерминальный), или он имеет вид [E] и изображает объединение множества E и пустой последовательности или {E} и изображает объединение пустой последовательности и E, EE, EEE . . . Круглые скобки можно использовать для группировки термов и множителей.

РБНФ может описать свой собственный синтаксис, например, следующим образом:

```
синтаксис = { правило }.
правило = НТСимв "=" выражение ".".
выражение = терм { "|" терм }.
терм = множитель { множитель }.
множитель = ТСимв | НТСимв | "(" выражение ")" |
"[" выражение "]" | " { "выражение "} ".
```

## 3. Словарь и представление

Язык — это бесконечное множество предложений (программ), а именно тех предложений, которые построены в соответствии с синтаксисом языка. Каждое предложение (программа) — это конечная последовательность символов из ограниченного словаря. Словарь Модулы-2 состоит из идентификаторов, чисел, текстов,

знаков операций и разделителей. Они называются лексическими символами или лексическими единицами и, в свою очередь, составлены из последовательностей литер. (Отметьте разницу между символами и литерами.) Представление символов литерами зависит от используемого набора литер. В данной статье используется набор ASCII * и соблюдаются следующие лексические правила:

1. Идентификатор — это последовательность букв и цифр. Первым символом должна быть буква.

```
идент = буква { буква | цифра }.
```

Примеры.

```
X читай Модула ЕТН ДайСимвол первБуква
```

2. Числа — это целые или вещественные числа (без знака). Целое число — это последовательность цифр. Если после числа стоит буква B, то число рассматривается как восьмеричное, если буква H, то как шестнадцатеричное; если же стоит буква C, то оно изображает литеру с данным (восьмеричным) порядковым номером (и имеет тип CHAR, см. 6.1). Целое число i из отрезка 0 $<=$ $=$ i $<=$ МаксЦел можно рассматривать как имеющее тип INTEGER или CARDINAL, из отрезка МаксЦел $<$ i $<=$ МаксНат — тип CARDINAL. Для 16-разрядной машины: МаксЦел $=$ $=$ 32767, МаксНат $=$ 65535. Вещественное число всегда содержит десятичную точку. Оно также может содержать десятичный порядок. Буква E читается «на десять в степени». Вещественное число имеет тип REAL.

```
число = целое | вещественное.
целое = цифра { цифра } | восьмЦифра { восьмЦифра }.
        ( "B" | "C" ) | цифра { шестЦифра } "H".
вещественное = цифра { цифра } "." { цифра } [ порядок ].
порядок = "E" [ "+" | "-" ] цифра { цифра }.
шестЦифра = цифра | "A" | "B" | "C" | "D" | "E" | "F".
цифра = восьмЦифра | "8" | "9".
восьмЦифра = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7".
```

Примеры.

```
1980   3764B   7BCH   33C   12.3   45.57E-8
```

3. Тексты — это последовательности литер, заключенные в кавычки. В качестве кавычек можно использовать как двойные кавычки, так и апостроф. Однако открывающий и закрывающий символы должны совпадать и этот символ не может встречаться внутри текста. Текст не может продолжаться на следующей строке программы.

```
текст = "'" { литера } "'" | '"' { литера } '"'.
```

Текст, состоящий из одной литеры, имеет тип CHAR, а текст, состоящий из n $>$ 1 литер, имеет тип (см. 6.4)

```
ARRAY [0 .. n — 1 ] OF CHAR
```

* При переводе этот набор дополнен буквами русского алфавита.

Примеры.

"MODULA" "В'езд запрещен!" 'программа "Аполлон"'

4. Знаки операций и разделители — это специальные литеры, пары литер и перечисленные ниже зарезервированные слова. Зарезервированные слова состоят только из прописных букв, и эти слова запрещается использовать в качестве идентификаторов. Синонимами являются символы # и <>, как и символы & и AND.

| | | | | |
|---|---|---|---|---|
| + | = | AND | FOR | QUALIFIED |
| − | # | ARRAY | FROM | RECORD |
| * | < | BEGIN | IF | REPEAT |
| / | > | BY | IMPLEMENTATION | RETURN |
| := | <> | CASE | IMPORT | SET |
| & | <= | CONST | IN | THEN |
| . | >= | DEFINITION | LOOP | TO |
| , | .. | DIV | MOD | TYPE |
| ; | : | DO | MODULE | UNTIL |
| ( | ) | ELSE | NOT | VAR |
| [ | ] | ELSIF | OF | WHILE |
| { | } | END | OR | WITH |
| ↑ | \| | EXIT | POINTER | |
| | | EXPORT | PROCEDURE | |

5. Пробелы внутри символов (за исключением текстов) запрещены. Пробелы и концы строк игнорируются, они имеют значение только для разделения следующих друг за другом символов.

6. Между любыми двумя символами программы можно вставлять комментарии. Комментарий — это произвольная последовательность литер, которая открывается скобкой (* и закрывается *). Комментарии могут быть вложенными. Они не влияют на смысл программы.

## 4. Описания и правила локализации

Каждый идентификатор, встречающийся в программе, должен быть описан, если он не является стандартным. Последние рассматриваются как предописанные, они доступны во всех частях программы, и по этой причине их называют проникающими. Описания служат также для спецификации некоторых постоянных свойств объектов; например, является ли он константой, типом, переменной, процедурой или модулем.

Идентификатор используется для указания на связанный с ним объект. Это возможно только в тех частях программы, которые находятся внутри так называемой области видимости описания. В общем случае эта область распространяется на исходный блок (описание модуля или процедуры), которому данное описание принадлежит и в котором данный объект локален. К этому правилу локализации добавляются следующие случаи:

1. Если идентификатор X, определенный описанием D1, используется в другом описании (не операторе) D2, то D1 должно текстуально предшествовать D2.

2. Тип T1 можно использовать в описании типа указателя T (см. 6.7), который текстуально предшествует описанию T1, если оба описания T и T1 находятся в одном и том же блоке. Это ослабление правила 1.

3. Если идентификатор, определенный в модуле M1, экспортирован, то его область видимости расширяется на тот блок, в котором содержится M1. Если M1 является единицей компиляции (см. разд. 14), то область видимости распространяется на все те единицы, которые импортируют M1.

4. Идентификатор поля из описания записи (см. 6.7) доступен только в изображении поля и внутри присоединяющего оператора, который ссылается на переменную — запись этого типа. Идентификатор может быть квалифицирован. В этом случае к нему добавляется префикс — другой идентификатор, обозначающий тот модуль (см. разд. 11), в котором определен квалифицируемый идентификатор. Префикс и идентификатор разделяются точкой.

квалидент = идент {"." идент }.

Следующие идентификаторы являются стандартными:

| | | | |
|---|---|---|---|
| ABS | (10.2) | INCL | (10.2) |
| ADR | (10.2) | INTEGER | (6.1) |
| ASH | (10.2) | HALT | (10.2) |
| BITSET | (6.6) | HIGH | (10.2) |
| BOOLEAN | (6.1) | NEW | (10.2) |
| CAP | (10.2) | NIL | (6.7) |
| CARDINAL | (6.1) | ODD | (10.2) |
| CHAR | (6.1) | PROC | (6.8) |
| DEC | (10.2) | REAL | (6.1) |
| DISPOSE | (10.2) | ROUND | (10.2) |
| EXCL | (10.2) | SIZE | (10.2) |
| FALSE | (6.1) | TRUE | (6.1) |
| FLOAT | (10.2) | TSIZE | (10.2) |
| INC | (10.2) | | |

## 5. Описания констант

Описание константы связывает идентификатор с неким постоянным значением.

ОписаниеКонстанты = идент "=" КонстВыражение.
КонстВыражение = ПростоеКонстВыраж [ отношение ПростоеКонстВыраж ].
отношение = "=" | "#" | "<>" | "<" | "<= "|">" | ">=" | IN.
ПростоеКонстВыраж = [ "+" | "—" ] КонстТерм { ОперСлож КонстТерм }.
ОперСлож = "+" | "—" | OR.
КонстТерм = КонстМножитель { ОперУмнож КонстМножитель }.

```
ОперУмнож = "*" | "/" | DIV | MOD | AND | "&".
КонстМножитель = квалидент | число | текст | множество |
      "(" КонстВыражение ")" | NOT КонстМножитель.
множество = [ квалидент ] "{" [ элемент { "," элемент }] "}",
элемент = КонстВыражение [ ".." КонстВыражение ].
```

Смысл знаков операций объясняется в разд. 8. Идентификатор, стоящий перед левой скобкой множества, указывает тип этого множества. Если он опущен, то подразумевается стандартный тип BITSET (см. 6.6).

Примеры описаний констант

```
     N = 100
граница = 2*N — 1
   все = { 0 .. ДлинаСлова — 1 }
```

## 6. Описания типов

Тип данных определяет множество значений, которые могут принимать переменные данного типа, и связывает с этим типом идентификатор. В случае сложных типов определяется также и структура переменных этого типа. Имеется три различные структуры, а именно массивы, записи и множества.

```
ОписаниеТипа = идент "=" тип.
тип = ПростойТип | ТипМассива | ТипЗаписи | ТипМножества |
    ТипУказателя | ТипПроцедуры.
ПростойТип = квалидент | перечисление | ОтрезокТипа.
```

Примеры.

```
Цвет = (красный, зеленый, синий)
Индекс   = [ 1 .. 80 ]
Карта    = ARRAY Индекс OF CHAR
Вершина  = RECORD ключ : CARDINAL;
             левый, правый: УкДерев
             END
  Радуга = SET OF Цвет
УкДерев  = POINTER TO Вершина
Функция  = PROCEDURE ( CARDINAL ) : CARDINAL
```

### 6.1. Базовые типы

Следующие базовые типы считаются предописанными и обозначаются стандартными идентификаторами:

INTEGER — переменная этого типа принимает в качестве значений целые числа от МинЦел до МаксЦел;

CARDINAL — переменная этого типа принимает в качестве значений целые числа от 0 до МаксНат;

BOOLEAN — переменная этого типа принимает истинностные значения TRUE и FALSE. Это единственные значения данного типа, который предописан как перечисление
BOOLEAN = ( FALSE, TRUE );

CHAR — переменная этого типа принимает в качестве значений литеры набора ASCII ( ISO );

REAL — переменная этого типа принимает в качестве значений вещественные числа.

Для реализации на 16-разрядной ЭВМ МинЦел = —32768, МаксЦел = 32767 и МаксНат = 65535.

### 6.2. Перечисления

Перечисление — это список идентификаторов, которые изображают значения, образующие данный тип. В программе эти идентификаторы используются как константы. Только они — и никакие другие значения — принадлежат этому типу. Значения эти упорядочены: отношение порядка определено их последовательностью в перечислении.

перечисление = "(" СписокИдент ")".
СписокИдент = идент { "," идент }.

Примеры перечислений.

(красный, зеленый, синий)
(трефы, бубны, черви, пики)
(понедельник, вторник, среда, четверг, пятница, суббота, воскресенье)

### 6.3. Отрезки типов

Тип Т может быть определен как отрезок другого типа Т1, базового (за исключением REAL) или перечисления,— указанием наименьшего и наибольшего значений отрезка.

Отрезок Типа = "[" КонстВыражение".. КонстВыражение"]".

Первая константа задает нижнюю границу и должна быть не больше верхней границы. Тип границы Т1 называется исходным типом для Т, и все операции, применимые к операндам типа Т1, также применимы к операндам типа Т. Однако значение, присваиваемое переменной типа отрезка, должно лежать в заданном интервале. Если нижняя граница — неотрицательное число, то в качестве исходного типа для отрезка берется CARDINAL; а если она отрицательна, то INTEGER.

Говорят, что тип Т1 совместим с типом Т0, если Т0 = Т1, или Т1 — отрезок Т0, или Т0 — отрезок Т1, или Т0 и Т1 оба являются отрезками одного и того же (исходного) типа.

Примеры отрезков типов.

[0 .. N — 1]
["A" .. "Z"]
[ понедельник .. пятница ]

### 6.4. Типы массивов

Массив — это структура, состоящая из фиксированного числа компонент; все компоненты имеют один тип, называемый типом компонент. Элементы массива изображаются с помощью индексов — значений, принадлежащих так называемому типу индекса.

Описание типа массива задает как тип компонент, так и тип индекса. Последний должен быть перечислением, отрезком типа или одним из базовых типов BOOLEAN или CHAR.

ТипМассива = ARRAY ПростойТип { "," ПростойТип } OF тип.

Описание такого вида

ARRAY T1, T2, . . ., Tn OF T

с типами индексов T1 . . . Tn надо рассматривать как сокращение для описания

```
ARRAY T1 OF
    ARRAY T2 OF
        ...
        ARRAY Tn OF T
```

Примеры типов массивов.

```
ARRAY [ 0 .. N — 1 ] OF CARDINAL
ARRAY [ 1 .. 10 ], [ 1 .. 20 ] OF [ 0 .. 99 ]
ARRAY ДеньНедели OF Цвет
ARRAY Цвет OF ДеньНедели
```

### 6.5. Типы записей

Тип записи — это структура, состоящая из фиксированного числа компонент, возможно, разного типа. Описание типа записи задает для каждой компоненты, называемой полем, ее тип и обозначающий это поле идентификатор. Областью видимости этих идентификаторов полей является само определение записи, однако они доступны также в изображениях поля (см. 8.1), обозначающих компоненты переменных — записей.

Тип записи может иметь несколько вариантных разделов; в этом случае первое поле такого раздела называется полем признака: его значение показывает, какой вариант предполагается в этом разделе. Варианты структуры идентифицируются так называемыми метками выбора. Эти метки суть константы типа, указанного в поле признака.

```
ТипЗаписи = RECORD ПоследСписковПолей END.
ПоследСписковПолей = СписокПолей { ";" СписокПолей }.
СписокПолей = [ СписокИдент ":" тип |
    CASE [ идент ":" ] квалидент OF вариант { "|" вариант }
    [ ELSE ПоследСписковПолей ] END ].
вариант = СписокМетокВыбора ":" ПоследСписковПолей.
СписокМетокВыбора = МеткиВыбора { "," МеткиВыбора }.
МеткиВыбора = КонстВыражение [ ".." КонстВыражение ].
```

Примеры типов записей.

```
RECORD число : [ 1 .. 31 ];
       месяц : [ 1 .. 12 ];
       год :   [ 0 .. 2000 ]
END
```

```
RECORD
    Фамилия, имя : ARRAY [ 0 .. 9 ] OF CHAR;
    возраст : [ 0 .. 99 ];
    зарплата : REAL
END
RECORD X, Y : T0;
    CASE тэг0 : Цвет OF
        красный : А : Ткр1; Б : Ткр2 |
        зеленый : В : Тзл1 ; Г; Тзл2 |
        синий : Д : Тсн1; Е : Тсн2
    END
    Ж : T0;
    CASE тэг1 : BOOLEAN OF
        TRUE : К , Л : INTEGER |
        FALSE : М , Н : CARDINAL
    END
END
```

Пример, приведенный выше, содержит два вариантных раздела. Варианты первого раздела определяются значениями поля признака тэг0, а варианты второго раздела — полем признака тэг1.

```
RECORD
    CASE BOOLEAN OF
        TRUE : i : INTEGER (* со знаком *)
        FALSE : r : CARDINAL (* без  знака *)
    END
END
```

Этот пример показывает запись без фиксированной части и с вариантной частью, в которой опущено поле признака. Вид текущего варианта в таком случае не может быть определен из значения самой переменной. В некоторых случаях эта возможность удобна, но пользоваться ею нужно очень осторожно.

### 6.6. Типы множеств

Тип множества, определенный как SET OF T, включает в себя все множества из значений своего исходного типа T. Исходный тип должен быть или отрезком целых, лежащим между 0 и ДлинаСлова — 1; или перечислением, содержащим не более чем ДлинаСлова значений; или отрезком такого перечисления.

ТипМножества = SET OF ПростойТип.

Стандартный тип BITSET определен таким образом:

BITSET = SET OF [0. . ДлинаСлова — 1]

### 6.7. Типы указателей

Переменные указательного типа P принимают в качестве своих значений указатели на переменные некоторого другого типа T; в таком случае говорят, что указательный тип P связан с типом T.

Указательные значения генерируются вызовами стандартной процедуры NEW (см. 10.2).

ТипУказателя = POINTER TO тип.

Кроме этих указательных значений, переменная-указатель может принимать значение NIL, которое можно рассматривать как не указывающее ни на какую переменную.

### 6.8. Типы процедур

Переменные процедурного типа T могут принимать в качестве своего значения некую процедуру P; для этого типы формальных параметров P должны соответствовать списку формальных типов T. Кроме того, P не должна быть описана внутри другой процедуры и не может быть стандартной процедурой.

```
ТипПроцедуры= PROCEDURE [ СписокФормальныхТипов ].
СписокФормальныхТипов= "(" [ [ VAR ] ФормальныйТип
    { "," [ VAR ] ФормальныйТип } ] ")" [ ":" квалидент ].
```

Стандартный тип PROC обозначает процедуру без параметров:

PROC = PROCEDURE

## 7. Описания переменных

Описания переменных служат для введения переменных; они также связывают с каждой переменной уникальный идентификатор и фиксированные тип и структуру. Все переменные, идентификаторы которых перечислены в одном списке, получают один и тот же тип.

ОписаниеПеременных = СписокИдент ":" тип.

Тип переменной определяет, какое множество значений может принимать эта переменная и какие операции к ней применимы; он также определяет и структуру переменной.

Примеры (см. примеры в разд. 6).

```
i , j : CARDINAL
k :     INTEGER
p , q : BOOLEAN
s :     BITSET
F :     Функция
a :     ARRAY Индекс OF CARDINAL
w :     ARRAY [ 0 .. 7 ] OF
        RECORD лит : CHAR;
               счетчик : CARDINAL
        END
Д : УкДерев
```

## 8. Выражения

Выражения — это конструкции, указывающие правила вычислений значений переменных и способ получения новых значений применением операций. Выражения состоят из операндов и знаков операций. Для выделения связей между знаками операций и операндами можно использовать круглые скобки.

## 8.1. Операнды

За исключением литеральных констант,т. е. чисел, литер, текстов и множеств (см. разд. 5), операнды обозначаются так называемыми изображениями. Изображение состоит из идентификатора, указывающего на константу, переменную или процедуру, которую он изображает; этот идентификатор может быть квалифицирован идентификаторами модулей (см. разд. 4 и 11) и к нему могут быть добавлены селекторы в том случае, когда изображаемый объект является элементом какой-нибудь структуры. Если эта структура — некий массив A, то изображение A [E] обозначает ту компоненту A, индекс которой равен текущему значению E. Тип индекса массива A должен быть совместим по присваиванию с типом E (см. 9.1). Изображение вида A [E1, E2, . . ., En] является сокращением для A [E1] [E2] . . . [En]. Если эта структура — некая запись R, то изображение R.f обозначает поле f записи R. Изображение P ↑ обозначает ту переменную, на которую ссылается указатель P.

```
изображение = квалидент { "." идент } [ "[" СписокВыраж "]"
                                      | "↑" ].
СписокВыраж = выражение { "," выражение }.
```

Если изображаемый объект является переменной, то изображение ссылается на текущее значение этой переменной. Если этот объект является процедурой-функцией, то изображение без списка параметров ссылается на саму эту процедуру. Если же список параметров (возможно, пустой) имеется, то это подразумевает активацию процедуры, и такое изображение обозначает результат исполнения процедуры, т. е. так называемое «возвращаемое» значение. Типы фактических параметров обязаны соответствовать типам формальных параметров, заданным в описании процедуры (см. разд. 10).

Примеры изображений (см. примеры в разд. 7):

```
k                    ( INTEGER )
a [ i ]              ( CARDINAL )
w [ 3 ]. лит         ( CHAR )
Д ↑ . ключ           ( CARDINAL )|
Д ↑ . левый . правый   ( УкДерев )
```

## 8.2. Операции

Синтаксис выражений задает приоритеты знаков операций, разбивая их на четыре класса. Знак операции NOT имеет наивысший приоритет, затем идут так называемые знаки операций типа умножения, затем — типа сложения. Знаки отношений имеют наименьший приоритет. Последовательность операций одного и того же приоритета исполняется слева направо.

```
выражение = ПростоеВыражение [ отношение ПростоеВыражение ].
ПростоеВыражение = [ "+" | "—" ] терм { ОперСлож терм }.
терм = множитель { ОперУмнож множитель }.
```

```
множитель = число | текст | множество |
    изображение [ФактПараметры ] | "( выражение ")" | NOT
    множитель.
ФактПараметры = "(" [ СписокВыраж ] ")".
```

Все операции перечислены ниже. В некоторых случаях разные операции обозначаются одним и тем же знаком, в таких ситуациях конкретный вид операции определяется типами операндов.

### 8.2.1. Арифметические операции

| *символ* | *операция* |
|---|---|
| + | сложение |
| — | вычитание |
| * | умножение |
| / | вещественное деление |
| DIV | целое деление |
| MOD | взятие модуля |

Эти операции (за исключением /) применяются к операндам, имеющим тип INTEGER, CARDINAL или их отрезок. Оба операнда должны иметь тип CARDINAL или отрезок с исходным типом CARDINAL (в этом случае результат будет иметь тип CARDINAL) либо оба они должны иметь тип INTEGER или отрезок с исходным типом INTEGER (в этом случае результат будет иметь тип INTEGER).

Операции +, — и * применяются также и к операндам типа REAL. В этом случае оба операнда должны иметь тип REAL, и результат также будет иметь тип REAL. Операция деления / применяется только к операндам типа REAL.

Если знак операции используется с одним операндом, то — обозначает обращение знака, а + обозначает тождественную операцию. Обращение знака применяется к операндам типа INTEGER или REAL.

Операции DIV и MOD определяются следующими правилами:

x DIV y равно усеченному частному x / y
x MOD y равно остатку от деления x DIV y
x = ( x DIV y ) * y + ( x MOD y )

### 8.2.2. Логические операции

| *символ* | *операция* |
|---|---|
| OR | логическая конъюнкция |
| AND | логическая дизъюнкция |
| NOT | отрицание |

Эти операции применяются к операндам типа BOOLEAN и дают результат типа BOOLEAN.

p OR q означает «если p, то TRUE, иначе q»
p AND q означает «если p, то q, иначе FALSE».

### 8.2.3. Операции над множествами

| *символ* | *операция* |
|---|---|
| + | объединение множеств |
| − | разность множеств |
| * | пересечение множеств |
| / | *симметричная разность множеств* |

Эти операции применяются к операндам, имеющим любой тип множеств, и дают результат того же самого типа.

```
x IN ( s1 + s2 ) iff* ( x IN s1 ) OR      ( x IN s2 )
x IN ( s1 − s2 ) iff  ( x IN s1 ) AND NOT ( x IN s2 )
x IN ( s1 * s2 ) iff  ( x IN s1 ) AND     ( x IN s2)
x IN ( s1 / s2 ) iff  ( x IN s1 ) #       ( x IN s2 )
```

### 8.2.4. Отношения

Отношения дают результат типа BOOLEAN. Отношения порядка применимы к базовым типам INTEGER, CARDINAL, BOOLEAN, CHAR, REAL, к перечислениям и к отрезкам типов.

| символ | операция |
|---|---|
| = | равенство |
| # | неравенство |
| < | меньше |
| <= | меньше или равно (включение множеств) |
| > | больше |
| >= | больше или равно (включение множеств) |
| IN | *содержится в* (принадлежность множеству) |

Отношения = и # применяются также к множествам и указателям. Отношения <= и >=, примененные к множествам, обозначают (нестрогое) включение; IN обозначает отношение принадлежности множеству (в выражении вида x IN s выражение s должно иметь тип SET OF T, где T — тип x или совместимый с ним).

Примеры выражений (см. примеры в разд. 7):

```
1980                  ( CARDINAL )
k DIV 3               ( INTEGER)
NOT p OR q            ( BOOLEAN )
( i + j ) * ( i − j )  ( CARDINAL )
s − { 8 , 9 , 13 }    ( BITSET )
Д ↓ . ключ = 0        ( BOOLEAN )
{ 13 .. 15 } <= s      ( BOOLEAN )
i IN { 0 , 5 .. 8 , 15 }  ( BOOLEAN )
```

---

* Тогда и только тогда (Примеч. перев.)