Язык программирования Оберон

Ревизия 1.10.2013 / 3.5.2016 Редакция перевода от 2017-11-21

Никлаус Вирт

Все должно быть сделано максимально простым, но не примитивно. (А. Эйнштейн)

1. История и введение

Оберон — язык программирования общего назначения развился из **Модулы-2**. Его принципиальная новая особенность — концепция расширения типов. Он позволяет конструировать новые типы данных на основе существующих и устанавливать между ними отношения.

Этот документ не является учебником программирования. Он преднамеренно краток. Его назначение — служить эталоном для программистов, разработчиков компиляторов и авторов руководств. Если о чём-то не сказано, то обычно сознательно: или потому, что это следует из других правил языка, или потому, что это может нежелательно ограничить свободу разработчикам компиляторов.

Этот документ описывает язык определённый в 1988/90 и пересмотренный в 2007 / 2016.

2. Синтаксис

Язык представляет собой бесконечное множество предложений, а именно предложений правильно сформированных в соответствии с его синтаксисом. В **Обероне** эти предложения называются *единицами компиляции*. Каждая единица представляет собой конечную последовательность *символов* из конечного словаря. Словарь **Оберона** состоит из имён, чисел, строк, предписаний, разделителей и комментариев. Они называются *лексическими символами* и состоят из последовательностей *литер*. (Обратите внимание на разницу между символами и литерами.)

Для описания синтаксиса используются расширенная форма Бэкуса — Наура (РБНФ). Квадратные скобки [и] означают необязательность записанного внутри них выражения, а фигурные скобки { и } означают его повторение (возможно 0 раз). Синтаксические единицы (нетерминальные символы) обозначаются английскими словами, выражающими их интуитивное назначение. Символы словаря языка программирования (терминальные символы) обозначаются строками, заключенными в кавычки или заглавными буквами.

3. Словарь

Для составления терминальных символов предусматривается использование следующих правил. Пробелы и переносы не должны встречаться внутри символов (исключая комментарии и пробелы в строках). Они игнорируются, если они не существенны для отделения двух последовательных символов. Заглавные и строчные буквы считаются различными.

Имена — последовательности букв и цифр. Первая литера должен быть буквой.

```
имя = литера {литера | цифра}.
```

Примеры:

```
x  Сканер Оберон Симв_Получ Лит_Первая
```

Числа — (беззнаковые) целые или вещественные числа. Целые являются последовательностью цифр и могут быть представлены литерой суффикса. Если суффикса нет, то представление десятичное. Суффикс **H** обозначает шестнадцатеричное представление.

Вещественное число всегда содержит десятичную точку. Допускается чтобы оно было представлено целым десятичным числом. Литера **E** означает «умножить на десять в степени».

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

Примеры:

```
1987
100H  = 256
12.3
4.567E8 = 456700000
```

Строки — последовательность литер, заключенных в *двойные* кавычки ("). Ограничивающая кавычка не должна встречаться внутри строки. Допускается, что строка из одной литеры может быть определена порядковым номером литеры в шестнадцатеричной нотации с последующей литерой "X". Число литер в строке называется *длиной* строки.

```
строка = '"' {символ} '"' | цифра {шестнЦифра} "X".
```

Примеры:

```
"Оберон"    "Не беспокойся!"    22X
```

Предписания и *разделители* — это специальные литеры, пары литер или ключевые слова, перечисленные ниже. *Ключевые слова* состоят исключительно из заглавных букв и не могут использоваться в качестве имён.

```
+     :=    ARRAY      IMPORT     THEN
-     ^     BEGIN      IN         TO
*     =     BY         IS         TRUE
/     #     CASE       MOD        TYPE
~     <     CONST      MODULE     UNTIL
&     >     DIV        NIL        VAR
.     <=    DO         OF         WHILE
,     >=    ELSE       OR
;     ..    ELSIF      POINTER
|     :     END        PROCEDURE
(     )     FALSE      RECORD
[     ]     FOR        REPEAT
{     }     IF         RETURN
```

Комментарии могут быть вставлены между любыми двумя символами в программе. Они являются произвольными последовательностями литер, которые открываются скобкой (* и закрываются с помощью *). Комментарии не влияют на смысл программы. Они могут быть вложенными.

4. Объявления имён и область видимости

Каждое встречающееся в программе имя должно быть объявлено, если это не встроенное имя (например, ключевое слово или разделитель). Имена также служат для задания определённых постоянных свойств объекта, например, являются ли они константами, обозначением типа, переменной или процедурой.

Имя используется для ссылки на соответствующий объект. Это возможно в тех частях программы, которые находятся в пределах области видимости. Имя не может обозначать больше чем один объект внутри данной области. Область распространяется текстуально от места объявления до конца блока (процедуры или модуля), к которому принадлежит имя и, следовательно -- по отношению к которому, объект является локальным.

Имя, объявленное в блоке модуля, может сопровождаться меткой экспорта сразу после имени(*), чтобы указать, что оно экспортируется из определяющего модуля. В этом случае имя может быть использовано и в других модулях, если эти модули импортируют объявляющий модуль. Имя затем предваряется другим именем (*префиксом*), обозначающим его модуль (см. Гл 11

Модули). Префикс и имя разделены точкой и вместе называются *уточнённым именем* (или *квалифицированным именем*).

```
УточнИмя   =  [имя "."] имя.
ИмяОпр     =  имя ["*"].
```

Следующие имена являются стандартными (группа ключевых слов и встроенных процедур); их значение определено в разделе 6.1 Базовые типы и 10.2 Встроенные процедуры :

```
ABS       ASR       ASSERT    BOOLEAN       BYTE
CHAR      CHR       DEC       EXCL          FLOOR
FLT       INC       INCL      INTEGER       LEN
LSL       NEW       ODD       ORD           PACK
REAL      ROR       SET       UNPK
```

5. Объявление констант

Объявление константы связывает её имя с её значением.

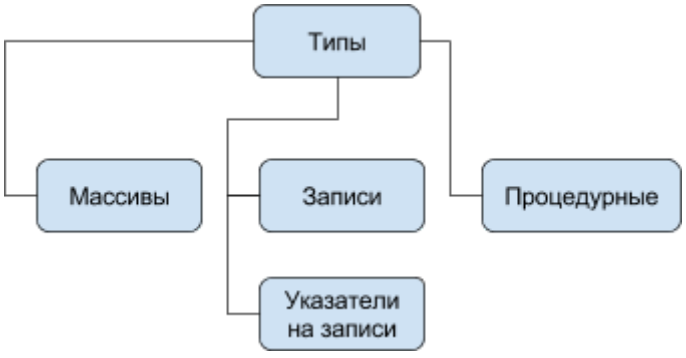
```
ОбъявлениеКонстанты  = ИмяОпр "=" КонстантноеВыражение.
КонстантноеВыражение = Выражение.
```

Константное выражение может быть вычислено по его тексту без фактического выполнения программы. Его операнды — константы (см. Гл. 8 Выражения). Примеры объявлений констант:

```
N     = 100
предел = 2*N - 1
мир2  = {0 .. мир.ширина -1}
имя   = "Оберон"
```

6. Объявление типов

Тип данных определяет множество значений, которые переменные этого типа могут принимать и операторов, которые к ним применимы. Объявление типа используется для связывания имени с типом. Типы определяют структуру переменных этого типа и, косвенно, операторы, которые применимы к компонентам самих типов. Есть две разных вида типов, а именно массивы и записи, с различным способом выбора компонентов.




```
ОбъявлениеТипа = ИмяОпр "=" Тип.
    Тип = УточнИмя | ТипМассив | ТипЗапись | ТипУказатель | ПроцедурныйТип.
```

Примеры:

```
Таблица= ARRAY 255 OF REAL
Дерево= POINTER TO Узел
Узел= RECORD
```

```
        ключ: INTEGER;
        левый, правый: Дерево
    END
    ДеревоСередина= POINTER TO УзелСередина
    УзелСередина= RECORD (Узел)
        имя     : ARRAY 32 OF CHAR;
        подузел: Дерево
    END
    Функция = PROCEDURE (x: INTEGER): INTEGER
```

6.1 Базовые типы

Следующие *базовые* типы обозначаются заранее объявленными именами. Связанные операторы определены в 8.2 Операторы, а встроенные функции - в 10.2 Встроенные функции. Значения заданного базового типа следующие:

`BOOLEAN` -- принимает значения `TRUE` и `FALSE`
`CHAR` -- литеры стандартного набора символов.
`INTEGER` -- целые числа
`REAL` -- Действительные числа
`LONGREAL` -- Действительные числа
`BYTE` -- целые числа от 0 до 255
`SET` -- набор целых чисел между 0 и пределом, зависящим от реализации

Тип `BYTE` совместим с типом `INTEGER`, и наоборот.

6.2 Тип массив

Массив - это структура, состоящая из фиксированного числа элементов одного типа, обозначаемых через тип элемента. Число элементов массива называется его длиной. Элементы массива обозначаются индексами, которые являются целыми числами от 0 до (длины - 1).

```
ТипМассива = ARRAY {"," длина} OF тип_элемента.
длина      = КонстантноеВыражение.
```

Форма объявления

```
ARRAY N0, N1, ..., Nk OF T
```

понимается как аббревиатура объявления

```
ARRAY N0 OF
    ARRAY N1 OF
    ...
        ARRAY Nk OF  T
```

Примеры типов массивов:

```
ARRAY NN OF INTEGER
ARRAY 10, 20 OF REAL
```

6.3 Тип запись

Тип записи - это структура, состоящая из фиксированного числа элементов возможно разных типов. Объявление типа записи задаёт для каждого элемента, которое называется *полем*, его тип и имя, которое ссылается на своё поле. Область видимости этих имён полей - само определение записи, но они также видны как части экземпляра записи в форме "через точку" (см. 8.1), которая ссылается на элементы самого экземпляра записи.

```
ТипЗаписи = RECORD ["(" БазовыйТип")"] [СписокПолейТипа] END.
БазовыйТип = имя.
СписокПолейТипа= СписокПолей{";" СписокПолей}.
СписокПолей = СписокИмён":" тип.
```

```
СписокИмён = ИмяОбъявление{"," ИмяОбъявление}.
```

Если тип записи экспортируется, имена полей, которые должны быть видимыми вне модуля объявления, должны быть помечены. Они называются публичными полями; неотмеченные поля называются приватными полями. Типы записей являются расширяемыми, т. е. тип записи может быть определён как расширение другого типа записи. В приведенных выше примерах `УзелСередина` (непосредственно) расширяет `Узел`, который является (прямым) базовым типом `УзелСередина`. Более конкретно, `УзелСередина` расширяет `Узел` с полями `имя` и `подузел`. Определение. Тип T расширяет тип T0, если он равен T0, или если он непосредственно является расширением T0. И наоборот, тип T0 является базовым типом для типа T, если он равен T, или если он является прямым базовым типом базового типа T. Примеры типов записей:

```
RECORD день, месяц, год: INTEGER
END
RECORD
  имя, фамилия: ARRAY 32 OF CHAR;
  возраст: INTEGER;
  зарплата: REAL
END
```

6.4 Тип указатель

Переменные указателя типа P принимают в качестве значений указатели на переменные некоторого типа T. Этот тип должен быть типом записи. Указатель типа P называется *связанным* с T, а T - базовым типом указателя P. Типы указателей наследуют отношение расширения их базовых типов (если они есть). Если тип T является расширением T0 и P является типом указателя, связанным с T, то P также является расширением P0, тип указателя, связанный с T0.

```
ТипУказателя = POINTER TO тип .
```

Если тип P определён как `POINTER TO T`, идентификатор T *может* быть текстуально объявлен *после* объявления P, но [если это так] он должен находиться в пределах одной области. Если p - переменная типа `P = POINTER TO T`, то вызов предопределённой процедуры `NEW (p)` имеет следующий эффект (см. 10.2 Встроенные процедуры): переменная типа T выделяется в свободном хранилище, а указатель на неё присваивается p. Этот указатель p имеет тип P, а ссылочная переменная `p^` имеет тип T. Отказ распределения хранилища под структуры приводит к тому, что p получает значение `NIL`. Каждой переменной-указателю *может быть* присвоено значение `NIL`, которое вообще не указывает на переменную.

6.5 Процедурный тип

Переменные процедурного типа T имеют процедуру (или `NIL`) в качестве значения. Если процедуре P присвоена переменная процедурного типа T, формальные параметры (типа) P должны быть такими же, как те, что указаны в формальных параметрах T. То же самое справедливо для типа результата в случае (См. 10.1). P не должен быть объявлен локальным для другой процедуры, и не может быть стандартной процедурой.

```
ПроцедурныйТип = PROCEDURE [ФормальныеПараметры].
```

7. Объявление переменных

Объявления переменных служат для введения переменных и связывания их с именами, которые не должны повторяться в пределах данной области. Они также служат для связывания фиксированных типов данных с переменными. `ОбъявлениеПеременной = СписокИмён":" тип`. Переменные, имена которых отображаются в одном списке, имеют один и тот же тип. Примеры объявления переменных (см. Примеры в главе 6):

```
i, j, k: INTEGER
x, y: REAL
```

```
p, q: BOOLEAN
s: SET
f: Function
a: ARRAY 100 OF REAL
w: ARRAY 16 OF
     RECORD
          лит: CHAR;
          счётчик: INTEGER
     END
t:  Дерево
```

8. Выражения

Выражения - это конструкции, обозначающие правила вычисления, в которых константы и текущие значения переменных объединяются для получения других значений посредством применения операторов и функциональных процедур. Выражения состоят из операндов и операторов. Круглые скобки могут использоваться для выражения определённых правил вычисления операторов и операндов.

8.1 Операнды

За исключением наборов и литеральных констант, то есть чисел и строк, операнды обозначаются "указателями". Обозначение состоит из имени, относящегося к константе, переменной или процедуре, которая будет обозначена. Такое имя может быть определено как имя модуля (см. Главы 4 и 11), и за ним могут следовать селекторы, если назначенный объект является элементом структуры.

Если A обозначает массив, то `A[E]` обозначает тот элемент массива A, индекс которого является текущим значением выражения E. Тип E должен иметь тип `INTEGER`. Обозначение вида `A[E1, E2, ..., En]` обозначает `A[E1][E2]...[En]`. Если p обозначает переменную-указатель, `p^` обозначает переменную, на которую ссылается p. Если r обозначает запись, то `r.f` обозначает поле f из записи r. Если p обозначает указатель, `p.f` обозначает поле f записи `p^`, то есть точка подразумевает разыменование, а `p.f` означает `p^.f`.

Охрана типа `v(T0)` проверяет, что v имеет тип T0, то есть охрана типа прекращает выполнение программы, если v не типа T0. Охранник применим, если

1. T0 является расширением объявленного типа T в v, и если
2. v - переменный параметр типа записи, или v - указатель.

```
указатель  = Квалификатор {селектор}
селектор   = "." Имя | "["СписокВыражения"]" | «^» | "(" Квалификатор
")"
СписокВыражения = выражение {"," выражение}.
```

Если назначенный объект является переменной, то обозначение ссылается на текущее значение переменной. Если объект является процедурой, указатель без списка параметров ссылается на эту процедуру. Если за ним следует список параметров (возможно, пустой), указатель подразумевает активацию процедуры и обозначает значение, полученное в результате его выполнения. Действительные параметры (используемых типов) должны соответствовать формальным параметрам, указанным в объявлении процедуры (см. Главу 10).

Примеры обозначений (см. Примеры в главе 7):

```
i                    (INTEGER)
a[I]                 (REAL)
w[3].ch              (CHAR)
t.ключ               (INTEGER)
t.левый.правый       (Дерево)
```

```
t(УзелСредний).подузел(Дерево)
```

8.2 Операторы

Синтаксис выражений различает четыре вида операторов с разными приоритетами (сильные связи). Оператор ~ имеет наивысший приоритет, за которым следуют операторы умножения, сложения и отношения. Операторы одного и того же приоритета ассоциируются слева направо. Например, x-y-z означает (x-y) -z.

```
выражение = ПростоеВыражение [отношение ПростоеВыражение].
отношение  =  "=" | "#" | "<" | "<=" | ">" | ">=" | IN | IS.
ПростоеВыражение =  ["+"|"-"] терминал {ОператорАддикции терминал}.
ОператорАддикции=  "+" | "-" | OR.
терминал =  множитель {ОператорМультипликации множитель}.
ОператорМультипликации =  "*" | "/" | DIV | MOD | "&" .
множитель =  число| строка| NIL | TRUE | FALSE | set | указатель
[ФактическийПараметр] | "(" выражение ")" | "~" множитель.
set  =  "{" [элемент {"," элемент}] "}".
элемент =  выражение [".." выражение].
ФактическийПараметр =  "(" [СписокВыражения] ")" .
```

Множество {m..n} обозначает {m, m + 1, ..., n-1, n}, а если m>n, то пустое множество. Доступные операторы перечислены в следующих таблицах. В некоторых случаях несколько разных операций обозначаются одним и тем же символом оператора. В этих случаях фактическая операция определяется типом операндов.

8.2.1 Логические операторы

Символ	Результат
OR	логическая дизъюнкция
&	логическое соединение
~	логическое отрицание

Эти операторы применяются к операндам BOOLEAN и дают результат BOOLEAN.

p OR q	означает «если p, то TRUE, иначе q»
p & q	обозначает «если p, то q, иначе FALSE»
~ P	означает "не p"

8.2.2 Арифметические операторы

Символ	Результат
+	сумма
-	разница
*	умножение
/	деление
DIV	целое частное
MOD	модуль

Операторы +, -, * и / применяются к операндам числовых типов. Оба операнда должны быть одного типа, что также определяет тип результата. При использовании в качестве унарных операторов "-" обозначает инверсию знака, а "+" обозначает операцию **идентичности**. Операторы DIV и MOD применяются только к целочисленным операндам. Пусть q = x DIV y и r = x MOD y. Тогда **множитель** q и остаток r определяются уравнением

```
X = q * y + r          0 <= r < y
```

8.2.3 Операторы наборов

Символ	Результат
+	объединение
-	разница
*	пересечение
/	симметричная разность наборов

.

Когда используется с одним операндом типа SET, знак минус обозначает дополнение набора.

8.2.4 Отношения

Символ	Отношение
=	равно
#	неравно
<	меньше
<=	меньше или равно
>	больше
=	больше или равно
IN	членство в наборе
IS	проверка типа

Булевы отношения. Отношение упорядочения <, <=,>,> = применяется к числовым типам, `CHAR` и литерным массивам. Отношения = и # применимы также к типам `BOOLEAN`, `SET`, указателям и процедурным типам.

`x IN s` обозначает "x является элементом s". x должен иметь тип `INTEGER` и s должен быть типом `SET`. `v IS T` означает «v имеет тип T» и вызывает проверку типа. Это применимо, если

1. T - расширение объявленного типа T0 для v, и если
2. v - переменный параметр типа записи или v - указатель.

Предполагая, например, что T является расширением T0 и что v является указателем на тип T0, тогда проверка `v IS T` определяет, является ли фактически назначенная переменная (также, а не только T0) типом T. Значение `NIL IS T` не определено.

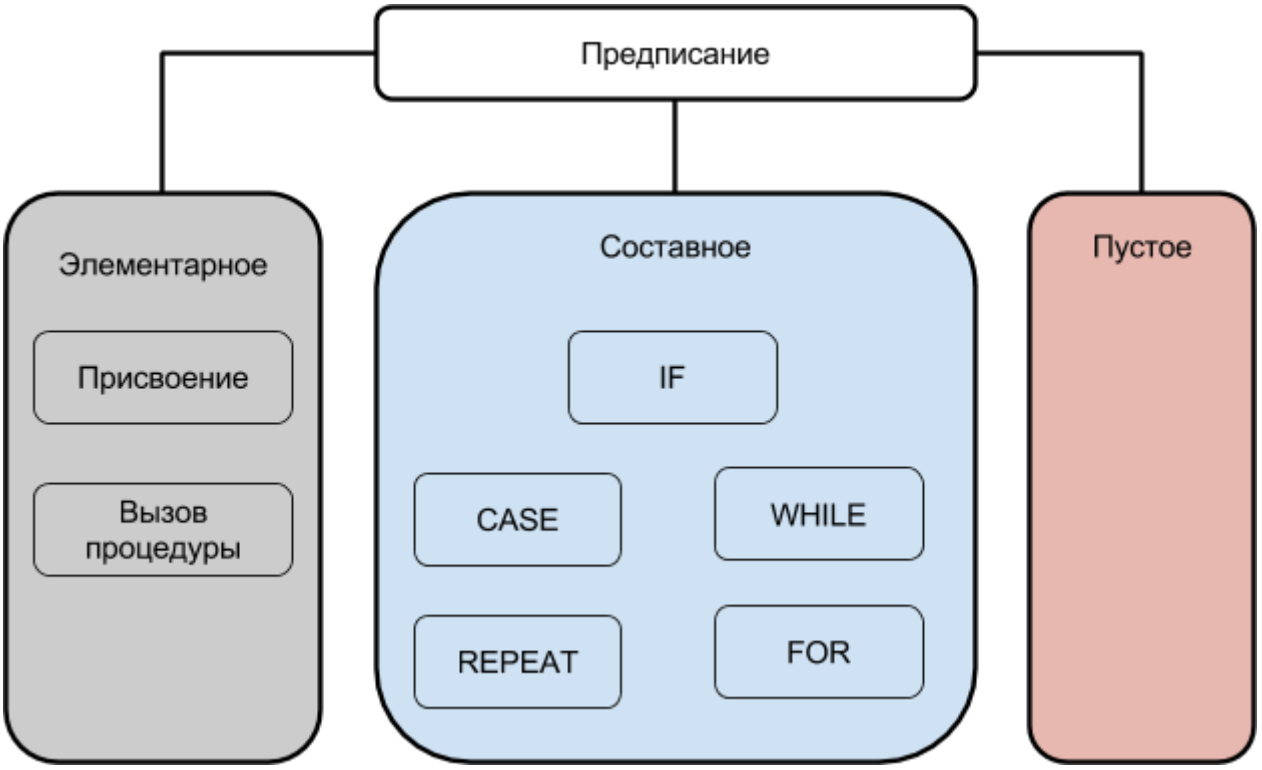
Примеры выражений (см. Примеры в главе 7):

```
1987                    (INTEGER)
I DIV 3                 (INTEGER)
~ P OR q                (BOOLEAN)
(I + j) * (i-j)         (INTEGER)
S - {8, 9, 13}          (SET)
A [i + j] * a [i-j]     (REAL)
(0 <= i) & (i <100)     (BOOLEAN)
T.ключ = 0              (BOOLEAN)
K IN {i .. j-1}         (BOOLEAN)
T IS УзелСредний              (BOOLEAN)
```

9. Предписания

Предписания обозначают действия. Есть элементарные и составные предписания. Элементарные предписания не состоят из каких-либо частей, которые сами являлись бы предписаниями. Они выражаются либо в присваивании, либо в вызове процедуры. Составные предписания состоят из частей, которые сами являются предписаниями. Они используются, чтобы выразить последовательное и условное, выборочное и повторное действие. Предписания также

могут быть пустыми, и в этом случае они не обозначают никаких действий. Пустое предписание включено в язык для того, чтобы ослабить правила пунктуации в последовательностях предписаний.




```
Предписание =  [Присвоение | ВызовПроцедур | Предписание_IF |
Предписание_CASE | Предписание_WHILE | Предписание_REPEAT | Предписание_FOR].
```

9.1 Предписание Присвоение

Элементарное предписание Присвоение служит для замены текущего значения переменной на новое значение, заданное выражением. Предписание присвоения записывается как «:=» и произносится как «становится».

```
присвоение  =  переменная ":=" выражение.
```

Если значение параметра структурировано (массив или тип записи), параметру не разрешено присваивать его или его элементам. Импортируемые переменные также не могут быть присвоены.

Тип выражения должен быть таким же, как у переменной. Имеют место следующие исключения:

1. Константу `NIL` можно присвоить переменным любого типа указателя или процедуры.
2. Строки могут быть назначены любому массиву литер, если количество литер в строке меньше, чем количество литер в массиве. (Добавляется нулевой символ). Однолитерные строки также могут быть присвоены переменным типа CHAR.
3. В случае записей тип источника должен быть расширением типа адресата.
4. Открытый массив может быть назначен массиву равного базового типа.

Примеры присвоений (см. Примеры в главе 7):

```
i := 0
p := i = j
x := FLT(i + 1)
k := (i + j) DIV 2
f := log2
s := {2, 3, 5, 7, 11, 13}
a[i] := (x+y) * (x-y)
t.key := i
w[i+1].ch := "A"
```

9.2 Предписание Вызов процедур

Элементарное предписание вызов процедуры служит для активации процедуры. Вызов процедуры может содержать список фактических параметров (аргументов), которые заменяются вместо соответствующих формальных параметров (параметров), определённых в объявлении процедуры (см. Главу 10). Соответствие устанавливается положениями параметров в списках фактических и формальных параметров соответственно. Существуют два вида параметров: переменные и значения.

В случае переменных параметров фактический параметр должен быть обозначением, обозначающим переменную. Если он обозначает элемент структурной переменной, селектор вычисляется, когда фактическая формальная замена/параметр имеет место, то есть перед выполнением процедуры. Если параметр является параметром значения, соответствующий фактический параметр должен быть выражением. Это выражение вычисляется до активации процедуры, и результирующее значение присваивается формальному параметру, который теперь представляет собой локальную переменную (см. Также 10.1.).

```
Вызов_Процедуры  =  Переменная [Актуальные_Параметры].
```

Примеры вызова процедур:

```
ReadInt(i)       (см. Главу 10)
WriteInt(2*j + 1, 6)
INC(w[k].count)
```

9.3 Последовательность предписаний

Последовательности предписаний обозначают последовательность действий, заданную компонентами оператора, разделенными точкой с запятой.

```
Последовательность_Предписаний  =  Предписание{";" Предписание}.
```

9.4 Предписание IF

```
Предписание_IF  =  IF Выражение THEN
     Последовательность_Предписаний
{ELSIF Выражение THEN
     Последовательность_Предписаний}
[ELSE
     Последовательность_Предписаний]
END.
```

Составное предписание `IF` определяет условное выполнение охраняемых подчинённых предписаний. Логическое выражение, предшествующее предписанию называется охрана. Охрана оценивает выражение в порядке встречи, пока выражение имеет значение `TRUE`, после чего выполняется связанная с охраной последовательность предписаний. Если охраны не выполнена, то выполняется последовательность предписаний после символа `ELSE`, если такая ветка существует. Пример:

```
IF (литера >= "A") & (литера <= "Z") THEN
     Сущность_Читать
ELSIF (литера >= "0") & (литера <= "9") THEN
     Число_Читать
ELSIF литера = 22X THEN
     Строку_Читать
END
```

9.5 Предписание CASE

Составное предписание `CASE` определяет выбор и выполнение последовательности предписаний в соответствии со значением выражения. Сначала вычисляется выражение `CASE`, затем выполняется последовательность предписаний, чей список меток содержит полученное

значение. Если выражение CASE имеет тип INTEGER или CHAR, все метки должны быть целыми или однолитерными строками, соответственно.

```
Предписание_CASE=  CASE выражение OF выбор{"|" выбор} END.
выбор =  [СписокМеток_CASE ":" ПоследовательностьПредписаний].
СписокМеток_CASE =  МеткиДляВыбора{"," МеткиДляВыбора}.
МеткиДляВыбора =  Метка[".." Метка].
Метка =  целое | строка | квалификатор.
```

Пример:

```
CASE k OF
      0: x := x + y
    | 1: x := x - y
    | 2: x := x * y
    | 3: x := x / y
END
```

Тип T выражения Case (Case переменная) также может быть типом записи или указателя. Тогда метки Case должны быть расширениями T, а в предписаниях Si, помеченных Ti, переменная Case рассматривается как тип Ti.

Пример:

```
TYPE R_ = RECORD a:
     INTEGER END;
     R0 = RECORD (R_)
          b: INTEGER
     END;
     R1 = RECORD (R_)
          b: REAL
     END;
      R2 = RECORD (R_)
          b: SET
     END;
     P = POINTER TO R_;
     P0 = POINTER TO R0;
     P1 = POINTER TO R1;
     P2 = POINTER TO R2;
VAR
     p: P;
CASE p OF
  P0: p.b := 10 |
  P1: p.b := 2.5 |
  P2: p.b := {0, 2}
END
```

9.6 Предписание While

Составное предписание While определяет повторение. Если любое из булевых выражений (охранников) дает TRUE, выполняется соответствующая последовательность предписаний. Оценка выражения и выполнение предписаний повторяются до тех пор, пока ни одно из булевых выражений не даст TRUE.

```
Предписание_While  =  WHILE Выражение DO
                            ПоследовательностьПредписаний
                {ELSIF Выражение DO
                     ПоследовательностьПредписаний}
                END.
```

Пример:

```
WHILE j > 0 DO
      j := j DIV 2;
      i := i+1
END
WHILE (t # NIL) & (t.key # i) DO
      t := t.left
END
WHILE m > n DO
      m := m - n
ELSIF n > m DO
      n := n - m
END
```

9.7 Предписание Repeat

Составное предписание Repeat указывает повторное выполнение последовательности предписаний до тех пор, пока условие не будет выполнено. Последовательность предписаний выполняется хотя бы один раз.

```
Предписание_Repeat  =  REPEAT
                         ПоследовательностьПредписаний
                  UNTIL Выражение.
```

9.8 Предписание For

Составное предписание For указывает повторное выполнение последовательности предписаний для заданного количества раз, в то время как последовательное увеличение значений присваивается целочисленной переменной, называемой управляющей переменной для предписания For.

```
Предписание_For =
FOR Имя ":=" Выражение TO Выражение[BY КонстантаВаражения] DO
  ПоследовательностьПредписаний
END.
```

Предписание For

```
FOR v := beg TO end BYinc DO
      S
END
```

есть, если `inc>0`, что эквивалентно

```
v := beg;
WHILE v <= end DO
      S;
      v := v + inc
END
```

и если `inc<0`, то это эквивалентно

```
v := beg;
WHILE v >= end DO
      S;
      v := v + inc
END
```

Типы `v`, `beg` и `end` должны быть `INTEGER`, а `inc` должен быть целым (константное выражение). Если шаг не указан, предполагается, что он равен 1.

10. Объявление процедур

Объявления процедур состоят из заголовка процедуры и тела процедуры. Заголовок указывает имя процедуры, формальные параметры и тип результата (если таковой есть). Тело содержит объявления и предписания. Имя процедуры повторяется в конце объявления процедуры.

Существует два типа процедур, а именно простые процедуры и процедуры-функции. Последние активируются указателем процедуры как составляющей выражения и возвращают результат, являющийся операндом в выражении. Простые процедуры активируются вызовом процедуры. Процедура-функция отличается в объявлении путём указания типа её результата после списка параметров. Её тело должно заканчиваться ключевым словом `RETURN`, которое определяет результат процедуры-функции.

Все константы, переменные, типы и процедуры, объявленные в теле процедуры, являются локальными для этой процедуры. Значения локальных переменных не определены при входе в процедуру. Поскольку процедуры могут быть объявлены как локальные объекты, объявления процедур могут быть вложенными.

В дополнение к своим формальным параметрам и локально объявленным объектам -- объекты, объявленные глобально, также видны в процедуре.

Использование имени процедуры в вызове в её объявлении подразумевает рекурсивную активацию процедуры.

```
ОБъявлениеПроцедуры =  ЗаголовокПроцедуры";" ТелоПроцедуры Имя.
ЗаголовокПроцедуры =  PROCEDURE ИмяОпр[ФормальныеПараметры].
ТелоПроцедуры =  ПоследОбъявлений[BEGIN ПоследПредписаний]
[RETURN Выражение] END.
ПоследОбъявлений =  [CONST {ОбъявлениеКонстант ";"}]
  [TYPE {ОбъявлениеТипов ";"}] [VAR {ОбъявлениеПеременных ";"}]
  {ОбъявлениеПроцедур ";"}.
```

10.1 Формальные параметры

Формальные параметры - это имена, которые обозначают фактические параметры, указанные в вызове процедуры. Соответствие между формальными и фактическими параметрами устанавливается при вызове процедуры. Существует два типа параметров, а именно значения и переменные параметры. Параметр переменной соответствует фактическому параметру, который является переменной, и он обозначает эту переменную. Параметр значения соответствует фактическому параметру, который является выражением, и он обозначает его значение, которое невозможно изменить при назначении. Однако, если параметр значения имеет базовый тип, он представляет собой локальную переменную, которой первоначально присваивается значение фактического выражения.

Тип параметра указывается в списке формальных параметров: переменные параметров обозначаются ключевым словом `VAR` и значениями параметров не имеют префикса.

Функция-процедура без параметров должна иметь пустой список параметров. Она должна быть вызвана через имя функции, список фактических параметров которой также пуст.

Формальные параметры являются локальными для процедуры, т. е. их область действия - это текст программы, который представляет собой объявление процедуры.

```
Формальныйпараметр =  "(" [Секция_FP {";" Секция_FP}] ")" [":" квалификатор].
Секция_FP =  [VAR] имя{"," имя} ":" ФормальныйПараметр.
ФормальныйПараметр =  {ARRAY OF} квалификатор.
```

Тип каждого формального параметра указан в списке параметров. Для переменных параметров он должен быть идентичен типу соответствующего фактического параметра, за исключением случая записи, где он должен быть базовым типом соответствующего типа фактического параметра.

Если тип формального параметра указан как

```
ARRAY OF T
```

параметр называется открытым массивом, а соответствующий фактический параметр может быть произвольной длины.

Если формальный параметр указывает тип процедуры, то соответствующий фактический параметр должен быть либо объявленный глобально, либо переменной (или параметром) этого типа процедуры. Это не может быть встроенная процедура. Тип результата процедуры не может быть ни записью, ни массивом.

Примеры объявлений процедур:

```
PROCEDURE Целое_Читать(VAR x: INTEGER);
VAR
     i : INTEGER;
     ch: CHAR;
BEGIN i := 0; Read(ch);
  WHILE ("0" <= ch) & (ch <= "9") DO
    i := 10*i + (ORD(ch)-ORD("0"));
     Read(ch)
  END ;
  x := i
END Целое_Читать

PROCEDURE Целое_Писать(x: INTEGER);  (* 0 <= x < 10^5 *)
VAR
     i: INTEGER;
     buf: ARRAY 5 OF INTEGER;
BEGIN
     i := 0;
     REPEAT
          buf[i] := x MOD 10;
          x := x DIV 10;
          INC(i)
     UNTIL x = 0;
     REPEAT
          DEC(i);
          Write(CHR(buf[i] + ORD("0")))
     UNTIL i = 0
END Целое_Писать

PROCEDURE log2(x: INTEGER): INTEGER;
VAR
     y: INTEGER;  (*assume x>0*)
BEGIN y := 0;
     WHILE x > 1 DO
          x := x DIV 2;
          INC(y)
     END ;
     RETURN y
END log2
```

10.2. Встроенные процедуры

В следующей таблице перечислены предопределенные процедуры. Некоторые из них являются общими процедурами, то есть они применяются к нескольким типам операндов. v обозначает переменную, x и n для выражений, T - тип.

Процедуры-функции:

Имя	Тип аргумента	Тип результата	Функция
ABS(x)	x: числовой тип	соответствует типу x	абсолютное значение числа
ODD(x)	x: INTEGER	BOOLEAN	x MOD 2 = 1
LEN(v)	v: ARRAY	INTEGER	длина v
LSL(x, n)	x, n: INTEGER	INTEGER	логический сдвиг влево, $x * 2^n$
ASR(x, n)	x, n: INTEGER	INTEGER	знаковый сдвиг вправо, $x \text{ DIV } 2^n$
ROR(x, n)	x, n: INTEGER	INTEGER	x сдвигается вправо на n бит

Функции преобразования типов:

Имя	Тип аргумента	Тип результата	Функция
FLOOR(x)	REAL	INTEGER	округление вниз
FLT(x)	INTEGER	REAL	преобразование типа
ORD(x)	CHAR, BOOLEAN, SET	INTEGER	порядковый номер x
CHR(x)	INTEGER	CHAR	литера по порядковому номеру x

Безопасные процедуры:

Имя	Тип аргумента	Функция
INC(v)	INTEGER	v := v + 1
INC(v, n)	INTEGER	v := v + n
DEC(v)	INTEGER	v := v - 1
DEC(v, n)	INTEGER	v := v - n

INCL(v, x)	v: SET; x: INTEGER	v := v + {x}
EXCL(v, x)	v: SET; x: INTEGER	v := v - {x}
NEW(v)	тип указателя	размещение v^
ASSERT(b)	BOOLEAN	прервать, если ~b
PACK(x, n)	REAL; INTEGER	упаковать x и n в x
UNPK(x, n)	REAL; INTEGER	распаковать x в x и n

Функция FLOOR (x) дает наибольшее целое число, не большее x.
`FLOOR(1.5) = 1  FLOOR(-1.5) = -2`
Параметр n в `PACK` представляет экспоненту `x.PACK (x, y)` эквивалентен x: = x * 2^y
`UNPK` - это обратная операция. Получаемый x нормализуется, так что 1.0 <= x <2.0.

11. Модули

Модуль представляет собой набор объявлений констант, типов, переменных и процедур и последовательностей предписаний с целью присвоения начальных значений переменным. Модуль обычно представляет собой текст, который можно скомпилировать как единое целое.

```
Модуль =  MODULE Имя ";"  [СписокИмпорта] ПоследовательностьОбъявлений
     [BEGIN ПоследовательностьПредписаний] END имя"." .
СписокИмпорта =  IMPORT Импорт{"," Импорт} ";" .
Импорт =  Имя[":=" Имя].
```

В списке импорта указаны модули, для которых текущий модуль является клиентом. Если имя x экспортируется из модуля M, и если M указан в списке импорта модуля, тогда к x обращаются как `M.x`. Если в списке импорта используется форма «M: = M1», экспортируемый объект x, объявленный в M1, ссылается в импортируемом модуле как `M.x`.

Имена, которые должны быть видны в клиентских модулях, то есть должны быть экспортированы, должны быть отмечены звездочкой (отметкой экспорта) в их объявлении. Переменные всегда экспортируются в режиме только для чтения.

Последовательность предписаний, следующая за символом `BEGIN`, выполняется, когда модуль добавляется в систему (загружается). Последующие индивидуальные (без параметров) процедуры могут быть активированы из системы, и эти процедуры служат в качестве команд.

Пример:

```
MODULE  Вывод;              (*экспортирует процедуры:   Write, WriteInt,
WriteLn*)
            IMPORT Текст,
               Oberon;
            VAR
               W: Текст.Writer;
            PROCEDURE Write*(ch: CHAR);
               BEGIN
                    Текст.Write(W, ch)
               END ;
```

```
    PROCEDURE WriteInt*(x, n: INTEGER);
        VAR
            i: INTEGER;
            a: ARRAY 16 OF CHAR;
        BEGIN
            i := 0;
            IF x < 0 THEN
                Текст.Текст(W, "-");
                x := -x
            END ;
            REPEAT
                a[i] := CHR(x MOD 10 + ORD("0"));
                x := x DIV 10;
                INC(i)
            UNTIL x = 0;
            REPEAT
                Текст.Write(W, " ");
                DEC(n)
            UNTIL n <= i;
            REPEAT
                DEC(i);
                Текст.Write(W, a[i])
            UNTIL i = 0
        END WriteInt;
  PROCEDURE WriteLn*;
    BEGIN
        Текст.WriteLn(W);
        Текст.Append(Oberon.Log, W.buf)
    END WriteLn;
BEGIN
    Текст.OpenWriter(W)
END Out.
```

11.1 Модуль SYSTEM

Необязательный модуль SYSTEM содержит определения, необходимые для программирования операций низкого уровня, ссылающихся непосредственно на ресурсы, специфичные для данного компьютера и/или реализации языка/компьютера.

К ним относятся, например, средства доступа к устройствам, которые контролируются компьютером, и, возможно, низкоуровневым средствам для нарушения правил совместимости типов данных (иначе пришлось бы такие средства языка вводить явно).

В модуле SYSTEM есть две причины для упрощения процедур:

1) Их значение зависит от реализации, то есть значение не выводится из определения языка, и

2) они могут повредить систему (например, PUT). Настоятельно рекомендуется ограничить их использование конкретными низкоуровневыми модулями, поскольку такие модули по своей сути являются не переносимыми, и не «безопасными по типу». Однако они легко распознаются из-за идентификатора SYSTEM, появляющегося в списках импорта модуля.

Следующие определения обычно применимы без дополнительных изменений. Однако отдельные реализации языка могут включать в свои модули дополнительные определения SYSTEM, которые относятся к конкретному, находящемуся в использовании компьютеру. В дальнейшем v обозначает переменную, x, a и n для выражений.

Процедуры-функции:

Имя	Тип аргумента	Тип результата	Функция
ADR(v)	любой	INTEGER	адрес переменной v
SIZE(T)	любой тип	INTEGER	размер в байтах
BIT(a, n)	a, n: INTEGER	BOOLEAN	n бит в mem[a]

Собственные процедуры:

Имя	Тип аргумента	Тип результата	Функция
GET(a, v)	a: INTEGER;	v: any basic type	v := mem[a]
PUT(a, x)	a: INTEGER;	x: any basic type	mem[a] := x
COPY(src, dst, n)	all	INTEGER	копировать n последовательных слов из src в dst

Ниже приводятся дополнительные процедуры, принятые компилятором для RISC-процессора:

Процедуры-функции:

Имя	Тип аргумента	Тип результата	Функция
VAL(T, n)	скалярный	T	преобразование
ADC(m, n)	INTEGER	INTEGER	сложение с флагом переноса C
SBC(m, n)	INTEGER	INTEGER	вычитание с флагом переноса C
UML(m, n)	INTEGER	INTEGER	беззнаковое умножение
COND(n)	INTEGER	BOOLEAN	IF Cond(n) THEN ...

Собственные процедуры:

Имя	Тип аргумента	Функция
LED(n)	INTEGER	отображает n на LED-экране

Приложение: Синтаксис Оберона

```
     литера     =  "A" | "B" | … | "Z" | "a" | "b" | … | "z".
     цифра      =    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
"9".
     шестнЦифра =  digit | "A" | "B" | "C" | "D" | "E" | "F".
     имя        =  литера {литера | цифра}.
```

```
квалификатор =  [имя "."] имя.
ОпрИмени   = имя ["*"].
целое      =  цифра {цифра} | цифра {шестнЦифра} "H".
вещественное  =  цифра{цифра} "." {цифра} [экспонента].
экспонента =  "E" ["+" | "-"] цифра{цифра}.
число      =  целое | вещественное.
строка     =  """ {литера} """ | цифра {шестнЦифра } "X".
ОпрКонстанты =  ОпрКонстанты "=" КонстВыражение.
КонстВыражение =  выражение.
ОпрТипа    =  ОпрИмени "=" тип.
тип        =   квалификатор| ТипМассива | ТипЗаписи | ТипУказателя |
ТипПроцедуры.
ТипМассива =  ARRAY длина {"," длина} OF тип.
длина      =  КонстВыражение .
ТипЗаписи       =  RECORD ["(" БазовыйТип ")"] [ПоследСписокПолей] END.
БазовыйТип =  квалификатор.
ПоследСписокПолей =  СписокПолей {";" СписокПолей }.
СписокПолей  =  СписокИмён ":" тип.
СписокИмён      =  ОпрИмени{"," ОпрИмени}.
ТипУказателя  =  POINTER TO тип.
ТипПроцедуры =  PROCEDURE [ФормальныеПараметры].
ОпрПеременных =  СписокИмён ":" тип.
выражение  =  ПростоеВыражение [отношение ПростоеВыражение ].
отношение  =  "=" | "#" | "<" | "<=" | ">" | ">=" | IN | IS.
ПростоеВыражение  =  ["+" | "-"] терминал {ОперАддикции терминал }.
ОперАддикции =  "+" | "-" | OR.
терминал   =  множитель {ОперМультипликации  множитель}.
ОперМультипликации =  "*" | "/" | DIV | MOD | "&".
множитель  =   число | строка | NIL | TRUE | FALSE | set | указатель
[АктуальныеПараметры] | "(" выражение")" | "~" множитель.
указатель  =  квалификатор {селектор}.
селектор   =     "."  имя|  "["  СписокВыражений"]"  |  "^"  |     "("
квалификатор")".
набор      =  "{" [элемент {"," элемент}] "}".
элемент    =  выражение [".." выражение ].
СписокВыражений =  выражение {"," выражение }.
АктуальныеПараметры =  "(" [СписокВыражений ] ")" .
предписание     =    [присвоение  |  ВызовПроцедуры|  Предписание_If  |
Предписание_Case |
   Предписание_While| Предписание_Repeat | Предписание_For].

присвоение      =  указатель ":=" выражение.
ВызовПроцедуры =  указатель [АктуальныеПараметры].
ПоследПредписаний =  предписание {";" предписание }.
Предписание_If  =  IF выражение THEN ПоследПредписаний
  {ELSIF выражение THEN ПоследПредписаний }
  [ELSE ПоследПредписаний ] END.
Предписание_Case  =  CASE выражение OF вариант {"|" вариант } END.
вариант    =  [СписокМеток_Case ":"  ПоследПредписаний ].
СписокМеток_Case  = СписокМеток {"," СписокМеток }.
СписокМеток  =  метка [".." метка ].
метка           =  целое | строка | квалификатор .
Предписание_While =  WHILE выражение DO ПоследПредписаний
```

```
    {ELSIF выражение DO ПоследПредписаний } END.
    Предписание_Repeat  =  REPEAT ПоследПредписаний UNTIL выражение .

    Предписание_For     =     FOR   имя   ":="   выражение   TO   выражение   [BY
КонстВыражение ]
    DO ПоследПредписаний END.

    ОпрПроцедуры =  ЗаголовокПроцедуры ";" ТелоПроцедуры имя.
    ЗаголовокПроцедуры  =  PROCEDURE ОпрИмени [ФормальныеПараметры].

    ТелоПроцедуры  =  ПоследОпределений[BEGIN ПоследПредписаний ]
        [RETURN выражение ] END.

    ПоследОпределений =  [CONST {ОпрКонстант ";"}]
      [TYPE {ОпрТипа ";"}]
      [VAR {ОпрПеременной ";"}]
      {ОпрПроцедуры  ";"}.
    ФормальныеПараметры  =     "("  [Секция_FP  {";"  Секция_FP}]  ")"  [":"
квалификатор].
    Секция_FP  =  [VAR] имя {"," имя } ":" ФормальныйТип.
    ФормальныйТип =  {ARRAY OF} квалификатор.

    модуль     =  MODULE имя ";" [СписокИмпорта] ПоследОпределений
      [BEGIN ПоследПредписаний ] END имя "."

    СписокИмпорта =  IMPORT импорт {"," импорт } ";".
    импорт     =  имя [":=" имя].
```

Использован [перевод С.Свердлова].

Перевод выполнили:

Денисов И.
Шипков В.

Технические консультанты:

Ершов А. П.
Волков С.
Ермаков И.
Ефимов А.