# Язык программирования Оберон

Ревизия 1.10.2013 / 3.5.2016

Никлаус Вирт

*Делай настолько просто, насколько возможно, но не проще. (А. Эйнштейн)*

## Содержание



## 1. История и введение

Оберон — язык программирования общего назначения, являющийся развитием языка Модула-2. Его принципиальная новая особенность — концепция расширения типов. Эта особенность позволяет конструировать новые типы данных на основе существующих типов и устанавливать между ними отношения.

Этот документ не является учебником по программированию. Он преднамеренно краток. Его назначение — служить эталоном для программистов, разработчиков компиляторов и авторов руководств. Если о чем-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.

Настоящий документ описывает язык, определенный в 1988/90 в редакции 2007 / 2016 годов.

## 2. Синтаксис

Язык представляет собой бесконечное множество предложений, а именно предложений, правильно оформленных 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 идентификаторов, чисел, строк, операторов, разделителей и комментариев. Они называются *лексическими символами* и состоят из последовательностей *литер*. (Обратите внимание на разницу между символами и литерами.)

Для описания синтаксиса используются расширенный формализм Бэкуса — Наура (расширенная Бэкус — Наурова форма (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 (в переводе — русскими) словами, выражающими их интуитивное значение. Символы словаря языка программирования (терминальные символы) обозначаются строками, заключенными в кавычки, или — заглавными буквами.

## 3. Словарь

Для составления 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.

*Идентификаторы* — это последовательности букв и цифр. Первая литера идентификатора должна быть буквой.

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

Примеры:

```
x  Scan  Oberon  GetSymbol  firstLetter
```

*Числа* — это (беззнаковые) целые или вещественные числа. Целые числа являются последовательностью цифр и могут быть продолжены буквой суффикса. Если суффикса нет, то представление десятичное. Суффикс H указывает на шестнадцатеричное представление.

*Вещественное число* всегда содержит десятичную точку. Опционально оно может также содержать десятичный порядок. Буква 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* — это последовательность литер, заключенных в двойные кавычки ("). Ограничивающая кавычка не должна встречаться внутри строки. Альтернативно, строка из одной литеры может быть определена порядковым номером буквы в шестнадцатеричной нотации с последующей литерой "X". Число литер в строке называется *длиной* строки.

```
строка = """ {литера} """ | цифра {шестнЦифра} "X".
```

Примеры:

```
"OBERON"    "Don't worry!"    22X
```

*Операции* и *разделители* — это специальные литеры, пары литер или зарезервированные слова, перечисленные ниже. Зарезервированные слова состоят исключительно из заглавных букв и не могут использоваться в качестве идентификаторов.

```
+       :=      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 и правила области видимости

Каждый идентификатор, встречающийся в программе, должен быть объявлен заранее, если это не предопределенный идентификатор. Объявления также служат для задания определенных постоянных свойств объекта, например, является ли он константой, типом, переменной или процедурой.

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

Идентификатор, объявленный в блоке модуля, может сопровождаться меткой экспорта (*), чтобы указать, что он экспортируется из определяющего модуля. В этом случае, идентификатор может быть использован в других модулях, если они импортируют объявляющий модуль. В таком случае, идентификатор предваряется идентификатором, обозначающим его модуль (см. Гл. 11). Префикс и идентификатор разделены точкой и вместе называются *уточненным идентификатором*.

```
уточнИдент  =  [идент "."] идент.
идентОпр    =  идент ["*"].
```

Следующие идентификаторы являются предопределенными; их значение описано в разделе 6.1 (типы) и 10.2 (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. Объявления констант

Объявление константы связывает ее идентификатор с ее значением.

```
ОбъявлениеКонстанты     = идентОпр "=" КонстантноеВыражение.
КонстантноеВыражение    = выражение.
```

Константное выражение может быть вычислено при транслировании текста программы без ее выполнения. Операнды константного выражения также являются константами (см. Гл. 8). Примеры объявления констант:

```
N = 100
limit = 2*N-1
all = {0 .. WordSize-1}
name = "Oberon"
```

## 6. Объявления типов

Тип данных определяет множество значений, которые могут принимать переменные этого типа, и применимые операций. Объявление типа используется для связи идентификатора с типом. Типы определяют структуру переменных этого типа и, как следствие, операции, которые применимы к компонентам. Существуют две разные структуры данных, а именно массивы и записи, с различными способами обращения к их компонентам (селекторами).

```
ОбъявлениеТипа = идентОпр "=" тип.
тип = уточнИдент | ТипМассив | ТипЗапись | ТипУказатель | ПроцедурныйТип.
```

Примеры:

```
Table = ARRAY N OF REAL
Tree = POINTER TO Node
Node = RECORD key : INTEGER;
        left, right: Tree
END
CenterNode = RECORD (Node)
        name: ARRAY 32 OF CHAR;
        subnode: Tree
END
Function = PROCEDURE (x: INTEGER): INTEGER
```

## 6.1 Элементарные типы

Следующие элементарные типы обозначаются предопределенными идентификаторами. Соответствующие операции определены в 8.2, а предопределенные процедуры-функции — в 10.2. Значения элементарных типов таковы:

```
BOOLEAN   логические значения TRUE и FALSE
CHAR      литеры стандартного набора литер
INTEGER   целые числа
REAL      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 Типы массив

Массив — это структура, состоящая из фиксированного числа элементов одинакового типа, называемого *типом элементов*. Количество элементов массива называется его *длиной*. Элементы массива обозначаются индексами, которые являются целыми числами от 0 до длины массива минус 1.

```
ТипМассив = ARRAY длина {"," длина} OF тип.
длина = КонстантноеВыражение.
```

Объявление вида

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

понимается как сокращение объявления

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

Примеры типа массив:

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

## 6.3 Типы запись

Запись — это структура, состоящая из фиксированного числа элементов, которые могут иметь различные типы. Объявление типа запись задает для каждого элемента, называемого *полем*, его тип и идентификатор, который обозначает это поле. Область видимости идентификаторов полей — само определение записи, но они также доступны через обозначения поля (см. 8.1), указывающее на элементы переменных типа запись.

```
ТипЗапись = RECORD ["(" БазовыйТип ")"] [ПоследСпискаПолей] END.
БазовыйТип = уточнИдент.
ПоследСпискаПолей = СписокПолей {";" СписокПолей}.
СписокПолей = СписокИдент ":" тип.
СписокИдент = идентОпр {"," идентОпр}.
```

Если тип запись экспортируется, идентификаторы полей, которые должны быть видимыми вне модуля объявления, должны быть помечены. Они называются *общедоступными полями*; не отмеченные поля называются *приватными полями*.

Записи являются расширяемыми, т.е. запись может быть определена как расширение другой записи. В приведенных выше примерах *CenterNode* (напрямую) расширяет *Node*, который является его (прямым) базовым типом *CenterNode*. А именно, *CenterNode* расширяет *Node* полями *name* и *subnode*.

*Определение*: тип *T* расширяет тип *T0*, если он и есть *T0*, или если он непосредственно расширяет расширение *T0*. И наоборот, тип *T0* является базовым типом для типа *T*, если он и есть *T*, или если он является прямым базовым типом базового типа *T*.

Примеры типов запись:

```
RECORD day, month, year: INTEGER
END
RECORD
  name, firstname: ARRAY 32 OF CHAR;
  age: INTEGER;
  salary: REAL
END
```

### 6.4 Типы указатель

Переменные типа указатель *P* принимают в качестве значений указатели на переменные некоторого типа *T*. Этот тип должен быть записью. Тип указатель *P* называется *связанным с типом T*, а *T* — является *базовым типом указателя P*. Типы указатель наследуют отношение расширения базовых типов, если они есть. Если тип *T* является расширением *T0* и *P* является указателем, связанным с *T*, то тогда *P* также является расширением *P0*, который является указателем, связанным с *T0*.

```
ТипУказатель = 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): в свободной памяти выделяется место для переменной типа *T*, и указатель на нее присваивается к *p*. Этот указатель *p* типа *P* ссылается на переменную *p^* типа *T*. Ошибка выделения памяти под структуру приводит к тому, что *p* присваивается значение *NIL*. Каждой переменной типа указатель может быть присвоено значение *NIL*, которое не указывает ни на какую переменную вообще.

### 6.5 Процедурные типы

Переменные процедурного типа *T* принимают в качестве значения процедуру или NIL. Если процедура *P* связана с переменной процедурного типа *T*, то типы формальных параметров процедуры *P* должны совпадать с типами соответствующих формальных параметров типа *T*. То же самое справедливо для типа результата в случае процедур-функций (см. 10.1). *P* не может быть локальной в другой процедуре, и также не может быть стандартной процедурой.

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

## 7. Объявления переменных

Объявления переменных служат для введения переменных и связывания их с идентификаторами, которые должны быть уникальными в пределах заданной области определения. Они также служат для связывания фиксированных типов данных с переменными.

```
ОбъявлениеПеременных = СписокИдент ":" тип.
```

Переменные, идентификаторы которых отображаются в одном списке, имеют один общий тип. Примеры объявления переменных (отсылка к примерам в Гл. 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 ch: CHAR;
               count: INTEGER
            END
t:       Tree
```

# 8. Выражения

Выражения — это конструкции, которые задают правила вычисления значений с использованием констант и текущих значений переменных, применяя операции и процедуры-функции. Выражения состоят из операндов и операций. Круглые скобки могут использоваться для группировки операций и операндов.

## 8.1 Операнды

За исключением множеств и литерных констант, то есть чисел и строк, операнды определяются *обозначениями*. Обозначение может быть идентификатором константы, переменной или процедуры. Этот идентификатор может быть уточнен идентификатором модуля (см. Гл. 4 и 11) и может сопровождаться селекторами, если обозначенный объект — часть структуры.

Если *A* обозначает массив, то *A[E]* обозначает тот элемент *A*, индекс которого является текущим значением выражения *E*. Результат *E* должен иметь тип INTEGER. Обозначение вида *A[E1, E2, ..., En]* значит *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, и *p.f* означает *p^.f*.

Охрана типа *v(T0)* обеспечивает, чтобы *v* имел тип *T0*, то есть охрана типа прекращает выполнение программы, если *v* имеет тип отличный от *T0*. Охрана применима, если:

1. *T0* является расширением объявленного типа *T* в *v*, и если
2. *v* — переменный параметр типа запись, или *v* — указатель.

```
обозначение = уточнИдент {селектор}.
селектор = "." идент | "[" СписокВыражений "]" | "^" | "(" уточнИдент ")".
СписокВыражений = выражение {"," выражение}.
```

Если обозначенный объект является переменной, то обозначение ссылается на текущее значение переменной. Если объект является процедурой, обозначение без списка параметров ссылается на эту процедуру. Если за обозначением следует список параметров (возможно, пустой), обозначением подразумевает активацию процедуры и обозначает возвращаемый результат её исполнения. Фактические параметры (и их типы) должны соответствовать формальным параметрам, указанным в объявлении процедуры (см. Гл. 10).

Примеры обозначений (см. примеры в Гл. 7):

```
i                      (INTEGER)
a[i]                   (REAL)
w[3].ch                (CHAR)
t.key                  (INTEGER)
t.left.right           (Tree)
t(CenterNode).subnode  (Tree)
```

## 8.2 Операции

В синтаксисе выражений различаются четыре класса операций с различными приоритетами (порядком выполнения). Операция ~ имеет наивысший приоритет, за которой следуют мультипликативные операции, аддитивные операции и отношения. Операции одного приоритета выполняются слева направо. Например, x–y–z означает (x–y)–z.

```
выражение = ПростоеВыражение [отношение ПростоеВыражение].
отношение = "=" | "#" | "<" | "<=" | ">" | ">=" | IN | IS.
ПростоеВыражение  =  ["+"|"-"] выражение {ОперацияСложения выражение}.
ОперацияСложения  =  "+" | "-" | OR.
выражение = множитель {ОперацияУмножения множитель}.
ОперацияУмножения = "*" | "/" | DIV | MOD | "&" .
множитель = число | строка | NIL | TRUE | FALSE | множество |
              обозначение [ФактическиеПараметры] | "(" выражение ")" | "~" множитель
множество = "{" [элемент {"," элемент}] "}".
элемент = выражение [".." выражение].
ФактическиеПараметры = "(" [СписокВыражений] ")" .
```

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

### *8.2.1 Логические операции*

| Символ | Результат |
|---|---|
| OR | логическая дизъюнкция |
| & | логическое соединение |
| ~ | отрицание |

Эти операции применяются к операндам BOOLEAN и дают результат BOOLEAN.

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

### *8.2.2 Арифметические операции*

| Символ | Результат |
|---|---|
| + | сумма |
| - | разность |
| * | произведение |
| / | вещественное деление |
| DIV | деление нацело |
| MOD | остаток |

Операции +, -, * и / применяются к операндам числовых типов. Оба операнда должны быть одного типа, что также определяет тип результата. При использовании в качестве унарных операций "-" обозначает инверсию знака, а "+" обозначает операцию идентичности.

Операции 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 Операции над множествами*

| Символ | Результат |
|---|---|
| + | объединение |
| - | разность |
| * | пересечение |
| / | симметрическая разность множеств |

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

### *8.2.4 Отношения*

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

Отношения являются логическими операциями. Отношение упорядочения <, <=, >, >= применяется к числовым типам, CHAR и символьным 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 (не только T0, но и также) типом T. Значение *NIL IS T* не определено.

Примеры выражений (см. примеры в Гл. 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.key = 0            (BOOLEAN)
K IN {i .. j-1}      (BOOLEAN)
T IS CenterNode      (BOOLEAN)
```

# 9. Операторы

Операторы обозначают действия. Есть элементарные и структурные операторы. Элементарные операторы не состоят из каких-либо частей, которые сами являлись бы операторами. Это такие операторы как присваивание и вызов процедуры. Структурные операторы состоят из частей, которые сами являются операторами. Они используются, чтобы выразить последовательное и условное, выборочное и повторное исполнение. Операторы также могут быть пустыми, и в этом случае они не означают никаких действий. Пустой оператор включен для того, чтобы ослабить правила пунктуации в последовательностях операторов.

```
оператор  =
        [присваивание | ВызовПроцедуры
        | ОператорIf | ОператорCase | ОператорWhile
        | ОператорRepeat | ОператорFor].
```

## 9.1 Присваивания

Присваивание служит для замены текущего значения переменной на новое значение, заданное выражением. Оператор присваивания записывается как «:=» и произносится как «становится».

```
присваивание  =  переменная ":=" выражение.
```

Если значение параметра структурировано (имеет тип массив или запись), никакого присваивания ему или его элементам не допускаются. Для импортированных переменных также не допускаются присваивания.

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

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

Примеры присвоений (см. примеры в Гл. 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 Вызовы процедур

Вызов процедуры активирует процедуру. Он может содержать список фактических параметров, которые заменяют соответствующие формальные параметры, определенные в объявлении процедуры (см. Гл. 10).

Соответствие устанавливается в порядке следования параметров в списках фактических и формальных параметров. Имеются два вида параметров: параметры-переменные и параметры-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).

```
ВызовПроцедуры  =  обозначение [ФактическиеПараметры].
```

Примеры вызова процедур:

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

## 9.3 Последовательность операторов

Последовательность операторов, разделенных точкой с запятой, означает поочередное выполнение действий, заданных составляющими операторами.

```
ПослОператоров  =  оператор {";" оператор}.
```

## 9.4 Операторы If

```
ОператорIf = IF выражение THEN
        ПослОператоров
{ELSIF выражение THEN
        ПослОператоров}
[ELSE
        ПослОператоров]
END.
```

Операторы If определяют условное выполнение охраняемых операторов. Логическое выражение, предшествующие последовательности операторов, будем называть *условием*. Условия проверяются последовательно одно за другим, пока очередное не окажется равным TRUE, после чего выполняется связанная с этим условием последовательность операторов. Если ни одно условие не удовлетворено, выполняется последовательность операторов, записанная после слова ELSE, если оно имеется.

Пример:

```
IF (ch >= "A") & (ch <= "Z") THEN ReadIdentifier
ELSIF (ch >= "0") & (ch <= "9") THEN ReadNumber
ELSIF ch = 22X THEN ReadString
END
```

## 9.5 Операторы Case

Операторы Case определяют выбор и выполнение последовательности операторов по значению выражения. Сначала вычисляется выбирающее выражение, а затем выполняется та последовательность операторов, чей список меток варианта содержит полученное значение. Если выбирающее выражение имеет тип INTEGER или CHAR, то все метки должны быть целыми или строками из одной литеры, соответственно.

```
ОператорCase = CASE выражение OF вариант {" | " вариант} END.
вариант = [СписокМетокВарианта ":" ПослОператоров].
СписокМетокВарианта = МеткиВарианта {"," МеткиВарианта }.
МеткиВарианта = метка [".." метка].
метка = целое | строка | уточнИдент.
```

Пример:

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

```
  | 3: x := x / y
END
```

Тип *T* оценочного выражения (оценочной переменной) также может быть записью или указателем. Тогда метки должны быть расширениями *T*, и в операторах *Si* помечены как *Ti*, и оценочные переменные рассматриваются 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 Операторы While

Операторы While задают повторное выполнение. Если любое из логических вырождений (условий) возвращает TRUE, соответствующая последовательность операторов выполняется. Условия вычисляются и выполнение продолжается пока хотя бы одно из логических выражений возвращает TRUE.

```
ОператорWhile = WHILE выражение DO ПослОператоров
   {ELSIF выражение DO ПослОператоров} END.
```

Примеры:

```
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 Операторы Repeat

Операторы Repeat определяют повторное выполнение последовательности операторов пока условие, заданное логическим выражением, не удовлетворено. Последовательность операторов выполняется по крайней мере один раз.

```
ОператорRepeat = REPEAT ПослОператоров UNTIL выражение.
```

### 9.8 Операторы For

Операторы For определяют повторное выполнение последовательности операторов фиксированное число раз для прогрессии значений целочисленной переменной, называемой *управляющей переменной* оператора for.

```
ОператорFor =
        FOR идент ":=" выражение TO выражение [BY КонстантноеВыражение] DO
        ПослОператоров END.
```

Оператор

```
FOR v := beg TO end BY inc DO S END
```

если inc > 0, эквивалентен

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

и если inc < 0, то эквивалентен

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

Типы *v*, *beg* и *end* должны быть INTEGER, и *inc* должно быть целым (константным выражением). Если шаг не определен, то он предполагается равным 1.

# 10. Объявления процедур

Объявление процедуры состоит из заголовка процедуры и тела процедуры. Заголовок определяет имя процедуры, формальные параметры и тип результата (если он есть). Тело содержит объявления и операторы. Имя процедуры повторяется в конце объявления процедуры.

Имеются два вида процедур: собственно процедуры и процедуры-функции. Последние активизируются обозначением функции как части выражения и возвращают результат, который является операндом выражения. Собственно процедуры активизируются вызовом процедуры. Процедура-функция отличается в объявлении тем, что обозначает тип результата после списка параметров. Тело процедуры-функции должно содержать оператор возврата RETURN, который определяет ее результат.

Все константы, переменные, типы и процедуры, объявленные внутри тела процедуры, локальны в процедуре. Значения локальных переменных неопределены до входа в процедуру. Поскольку процедуры тоже могут быть объявлены как локальные объекты, объявления процедур могут быть вложенными.

В добавок к их формальным параметрам и локально объявленным объектам, объекты объявленные глобально также видны в процедуре.

Вызов процедуры изнутри ее объявления подразумевает рекурсивную активацию.

```
ОбъявлениеПроцедуры = ЗаголовокПроцедуры ";" ТелоПроцедуры идент.
ЗаголовокПроцедуры  = PROCEDURE идентОпр [ФормальныеПараметры].
ТелоПроцедуры       = ПослОбъявлений [BEGIN ПослОператоров]
                      [RETURN выражение] END.
ПослОбъявлений      = [CONST {ОбъявлениеКонстант ";"}]
                      [TYPE {ОбъявлениеТипов ";"}] [VAR {ОбъявлениеПеременных ";"}]
                      {ОбъявлениеПроцедуры ";"}.
```

## 10.1 Формальные параметры

Формальные параметры — это идентификаторы фактических параметров, которые будут конкретизированы при вызове процедуры. Соответствие между формальными и фактическими параметрами устанавливается, когда процедура вызывается. Имеются два вида параметров: параметры-значения и параметры-переменные. Параметры-переменные соответствуют фактическим параметрам, которые являются переменными, и означают эти переменные. Параметр-значение соответствует фактическому параметру, который является выражением, и отвечает его значению, которое не может быть изменено присвоением. Однако, если параметр-значение имеет элементарный тип, то он представляет локальную переменную, к которой изначально присвоено значение фактического выраженя.

Тип параметра обозначен в списке формальных параметров: параметр-переменная обозначается символом VAR, а параметр-значение отсутствием такого префикса.

Процедура-функция без параметров должна иметь пустой список параметров. Она должна вызываться обозначением функции, чей список фактических параметров также пуст.

Формальные параметры локальны для процедуры, т.е. их область действия простирается в области текста программы, представляющего объявление процедуры.

```
ФормальныеПараметры = "(" [СекцияФП {";" СекцияФП }] ")" [":" уточнИдент].
СекцияФП            = [VAR] идент {"," идент} ":" ФормальныйТип.
ФормальныйТип       = {ARRAY OF} уточнИдент.
```

Тип каждого формального параметра определен в списке параметров. Для параметров-переменных он должен быть идентичным соответствующему типу фактического параметра, кроме записей, когда он должен базовым типом соответствующего фактического параметра. Если тип формального параметра определен как

```
ARRAY OF T
```

то такой параметр называется открытым массивом, и соответствующий фактический параметр может быть произвольной длины.

Если формальный параметр определен как процедурный тип, то соответствующий фактический параметр должен быть либо процедурой, которая объявлена глобально, либо переменной (или параметром) процедурного типа. Также в этом случае фактический параметр не может быть предопределенной процедурой. Тип результата процедуры не может быть ни записью, ни массивом.

Пример объявления процедур:

```
PROCEDURE ReadInt(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 ReadInt

PROCEDURE WriteInt(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 WriteInt

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

## 10.2 Предопределенные процедуры-функции

Следующая таблица содержит список предопределенных процедур. Некоторые процедуры — обобщенные, то есть они применимы к операндам нескольких типов. Буква v обозначает переменную, x и n — выражения, T — тип.

*Процедуры-функции:*

| Имя | Типы аргумента | Тип результата | Функция |
|---|---|---|---|
| ABS(x) | x: числовой тип | тип x | абсолютное значение |
| ODD(x) | x: INTEGER | BOOLEAN | x MOD 2 = 1 |
| LEN(v) | v: массив | INTEGER | длина v |
| LSL(x, n) | x, n: INTEGER | INTEGER | логический сдвиг влево, $x * 2^n$ |
| ASR(x, n) | x, n: INTEGER | INTEGER | signed сдвиг вправо, $x \text{ DIV } 2^n$ |
| ROR(x, n) | x, n: INTEGER | INTEGER | x вращается вправо на n бит |

*Функции преобразования типов:*

| Имя | Типы аргумента | Тип результата | Функция |
|---|---|---|---|
| FLOOR(x) | REAL | INTEGER | округлить до меньшего |
| FLT(x) | INTEGER | REAL | тождественно |
| ORD(x) | CHAR, BOOLEAN, SET | INTEGER | порядковый номер x |
| CHR(x) | INTEGER | CHAR | литера с порядковым номером x |

*Собственно процедуры:*

| Имя | Типы 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 указатель | разместить v^ |
| ASSERT(b) | BOOLEAN | прерывает программу, если ~b |

```
PACK(x, n)    REAL; INTEGER        упаковывает x и n в x
UNPK(x, n)    REAL; INTEGER        распаковывает из x в x и n
```

Функция FLOOR(x) возвращает наибольшее целое не большее чем x.

```
FLOOR(1.5) = 1     FLOOR(-1.5) = -2
```

Параметр n процедуры PACK представляет экспоненту x. PACK(x, y) равнозначно x := x * $2^y$. UNPK является обратной операцией. Результирующее x нормализовано, так что 1.0 <= x < 2.0.

## 11. Модули

Модуль — совокупность объявлений констант, типов, переменных и процедур вместе с последовательностью операторов, предназначенных для присваивания начальных значений переменным. Модуль обычно представляет собой текст, который возможно компилировать как целое (единица компиляции).

```
    Модуль        = MODULE идент ";" [СписокИмпорта] ПослОбъявлений
                    [BEGIN ПослОператоров] END идент ".".
    СписокИмпорта = IMPORT Импорт {"," Импорт} ";".
    Импорт        = [идент ":="] идент.
```

В списке импорта указаны модули, клиентом которых является модуль. Если идентификатор x экспортируется из модуля M, и если M указан в списке импорта модуля, тогда обращение к x осуществляется как M.x. Если в списке импорта используется форма «M := M1», экспортируемый объект x, объявленный в M1, вызывается в импортирующем модуле как M.x.

Идентификаторы, 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 операторов после символа BEGIN выполняется, когда модуль добавляется к системе (загружается). Это происходит после загрузки импортируемых модулей. Отсюда следует, тот циклический импорт модулей запрещен. Отдельные (не имеющие параметров и экспортированные) процедуры могут быть активированы из системы. Эти процедуры служат командами (см. Приложение D1).
Последовательность операторов, следующая за символом BEGIN, выполняется, когда модуль добавляется в систему (загружается). Отдельные процедуры (без параметров) могут быть активированы из системы, и эти процедуры служат в качестве команд.

Пример:

```
MODULE Out;    (*exported procedures: Write, WriteInt, WriteLn*)
        IMPORT Texts, Oberon;
        VAR W: Texts.Writer;

        PROCEDURE Write*(ch: CHAR);
        BEGIN Texts.Write(W, ch)
        END Write;

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

        PROCEDURE WriteLn*;
        BEGIN Texts.WriteLn(W); Texts.Append(Oberon.Log, W.buf)
        END WriteLn;

BEGIN Texts.OpenWriter(W)
END Out.
```

### 11.1 Модуль SYSTEM

Опциональный модуль SYSTEM содержит определения, которые необходимы для программирования низко-уровневых операций, прямо указывающих на ресурсы конкретного компьютера и/или реализации.

Они содержат, к примеру, средства для доступа к устройствам, которые управляются компьютером, и, возможно, — средства для нарушения правил совместимости типов данных, иначе запрещенных по определению языка программирования.

Есть две причины для предоставления таких средств в Модуле SYSTEM; (1) Их значения зависят от реализации, то есть не выводятся из определения языка, и (2) они могут повредить систему (например PUT). Настоятельно рекомендуется ограничить использование этих средств специфическими модулями (модулями низкого уровня), так как такие модули по своей природе являются непереносимыми и небезопасны с точки зрения преобразования типов. Однако они легко распознаются по идентификатору SYSTEM в списке импорта. Следующие определения обычно применимы. Однако отдельные реализации могут включать в свои модули дополнительные определения SYSTEM, которые относятся к конкретному, находящемуся в основе компьютеру. В дальнейшем *v* обозначает переменную, *x*, *a* и *n* для выражений.

*Процедуры-функции:*

| Имя | Типы 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 | бит n из mem[a] |

*Собственно процедуры:*

| Имя | Типы аргумента | Функция |
|---|---|---|
| GET(a, v) | a: INTEGER; v: любой элементарный тип | v := mem[a] |
| PUT(a, x) | a: INTEGER; x: любой элементарный тип | mem[a] := x |
| COPY(src, dst, n) | все INTEGER | копирует n последовательных слова из src в dst |

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

*Процедуры-функции:*

| Имя | Типы аргумента | Тип результата | Функция |
|---|---|---|---|
| VAL(T, n) | скаляр | T | тождество |
| ADC(m, n) | INTEGER | INTEGER | сложение с учётом флага переноса C |
| SBC(m, n) | INTEGER | INTEGER | вычитания с учётом флага переноса C |
| UML(m, n) | INTEGER | INTEGER | беззнаковое умножение |
| COND(n) | INTEGER | BOOLEAN | IF Cond(n) THEN ... |

*Собственно процедуры:*

| Имя | Типы аргумента | Функция |
|---|---|---|
| LED(n) | INTEGER | показать n на светодиодах |

## Приложение: Синтаксис Оберона

```
буква        =  "A" | "B" | ... | "Z" | "a" | "b" | ... | "z".
цифра        =  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
шестнЦифра   =  цифра | "A" | "B" | "C" | "D" | "E" | "F".

идент        =  буква {буква | цифра}.
уточнИдент   =  [идент "."] идент.
идентОпр     =  идент ["*"].

целое        =  цифра {цифра} | цифра {шестнЦифра} "H".
вещественное =  цифра {цифра} "." {цифра} [порядок].
порядок      =  ("E") ["+" | "-"] цифра {цифра}.
число        =  целое | вещественное.
строка       = """ {литера} """ | цифра {шестнЦифра} "X".

ОбъявлениеКонстанты  =  идентОпр "=" КонстантноеВыражение.
КонстантноеВыражение =  выражение.

ОбъявлениеТипа  =  идентОпр "=" тип.
тип             =  уточнИдент | ТипМассив | ТипЗапись | ТипУказатель | ПроцедурныйТип.
ТипМассив       =  ARRAY длина {"," длина} OF тип.
длина           =  КонстантноеВыражение.
ТипЗапись       =  RECORD ["(" БазовыйТип ")"] [ПоследСпискаПолей] END.
БазовыйТип      =  уточнИдент.
ПоследСпискаПолей =  СписокПолей {";" СписокПолей}.
СписокПолей     =  СписокИдент ":" тип.
СписокИдент     =  идентОпр {"," идентОпр}.
```

```
ТипУказатель    =  POINTER TO тип.
ПроцедурныйТип  =  PROCEDURE [ФормальныеПараметры].

ОбъявлениеПеременных = СписокИдент ":" тип.

выражение           = ПростоеВыражение [отношение ПростоеВыражение].
отношение           = "=" | "#" | "<" | "<=" | ">" | ">=" | IN | IS.
ПростоеВыражение    =  ["+"|"-"] выражение {ОперацияСложения выражение}.
ОперацияСложения    =  "+" | "-" | OR.
выражение           = множитель {ОперацияУмножения множитель}.
ОперацияУмножения   = "*" | "/" | DIV | MOD | "&" .
множитель           = число | строка | NIL | TRUE | FALSE | множество |
                      обозначение [ФактическиеПараметры] | "(" выражение ")" | "~" множитель
обозначение         = уточнИдент {селектор}.
селектор            = "." идент | "[" СписокВыражений "]" | "^" | "(" уточнИдент ")".
множество           = "{" [элемент {"," элемент}] "}".
элемент             = выражение [".." выражение].
СписокВыражений     = выражение {"," выражение}.
ФактическиеПараметры  = "(" [СписокВыражений] ")" .

оператор            = [присвоение | ВызовПроцедуры | ОператорIf | ОператорCase |
                      ОператорWhile | ОператорRepeat | ОператорFor].
присвоение          = обозначение ":=" выражение
ВызовПроцедуры      = обозначение [ФактическиеПараметры].
ПослОператоров      = оператор {";" оператор}.
ОператорIf          = IF выражение THEN
                         ПослОператоров
                      {ELSIF выражение THEN
                         ПослОператоров}
                      [ELSE
                         ПослОператоров]
                      END.
ОператорCase        = CASE выражение OF вариант {" | " вариант} END.
вариант             = [СписокМетокВарианта ":" ПослОператоров].
СписокМетокВарианта = МеткиВарианта {"," МеткиВарианта }.
МеткиВарианта       = метка [".." метка].
метка               = целое | строка | уточнИдент.
ОператорWhile       = WHILE выражение DO ПослОператоров
                      {ELSIF выражение DO ПослОператоров} END.
ОператорRepeat      = REPEAT ПослОператоров UNTIL выражение.
ОператорFor         = FOR идент ":=" выражение TO выражение [BY КонстантноеВыражение] DO
                          ПослОператоров END.

ОбъявлениеПроцедуры = ЗаголовокПроцедуры ";" ТелоПроцедуры идент.
ЗаголовокПроцедуры  = PROCEDURE идентОпр [ФормальныеПараметры].
ТелоПроцедуры       = ПослОбъявлений [BEGIN ПослОператоров]
                      [RETURN выражение] END.
ПослОбъявлений      = [CONST {ОбъявлениеКонстант ";"}]
                      [TYPE {ОбъявлениеТипов ";"}] [VAR {ОбъявлениеПеременных ";"}]
                      {ОбъявлениеПроцедуры ";"}.

ФормальныеПараметры = "(" [СекцияФП {";" СекцияФП }] ")" [":" уточнИдент].
СекцияФП            = [VAR] идент {"," идент} ":" ФормалныйТип.
ФормалныйТип        = {ARRAY OF} уточнИдент.

модуль              = MODULE идент ";" [СписокИмпорта] ПослОбъявлений
                      [BEGIN ПослОператоров] END идент ".".
СписокИмпорта       = IMPORT импорт {"," импорт} ";".
импорт              = [идент ":="] идент.
```

Перевод выполнил Иван Денисов.

Использован перевод Сергея Свердлова.
Использован перевод пакета Дельта.
Использован перевод Бурцева Вадима.
Использован перевод Валерия Шипкова.
Благодарность за замечания по переводу

Александру Ширяеву, Артуру Ефимову, Александру Легалову и Константину (comdivbyzero).