Язык программирования OBERON-2M ORM

1. ВВЕДЕНИЕ

OBERON-2M ORM - строго типизированный язык программирования, является подмножеством языка Оберон-2. По сравнению с классическим Оберон-2, в нем сделаны несколько исключений и уточнений. При выборе подмножества использовался чистый эмпирический подход, продиктованный практикой использования языка для задач системного программирования. Данный текст описывает синтаксис и семантику языка.

2. СИНТАКСИС

Язык - это бесконечное множество предложений, удовлетворяющих его синтаксису. Такие предложения называются единицами компиляции. Каждая единица представляет собой конечную последовательность из конечного словаря. Словарь языка состоит из идентификаторов, чисел, строк, операторов, разделителей и комментариев. Все вместе они называются лексическими символами (лексемами) и состоят из последовательности литер.

Синтаксис языка описывается расширенными формами Бэкуса-Наура (РБНФ). Квадратные скобки [и] означают необязательность записанной в них формы, фигурные скобки { и } означают ее повторение (возможно 0 раз). Альтернативы разделяются литерой | и помещаются в круглые скобки (и). Синтаксические понятия (нетерминальные лексемы) обозначаются английскими словами, выражающими их интуитивный смысл. Символы словаря языка (терминальные символы) записываются строками, заключенными в кавычки, или заглавными буквами.

3. СЛОВАРЬ

Лексемами языка являются: идентификаторы, числа, строки, операторы, разделители и комментарии. Пробельные литеры (пробел, табуляция, перевод строки) игнорируются, если не используются для разделения лексем. Заглавные и строчные буквы считаются различными. Регистр букв имеет значение.

Идентификатор - это последовательность латинских букв, цифр и литеры подчеркивания "_", начинающаяся с буквы или подчеркивания.

```
идентификатор = (буква | "_") {буква | "_" | цифра}
```

Примеры:

```
x    _scan    Oberon_2M    GetSymbol
```

Числа могут быть целыми или действительными. Целые числа - последовательность цифр, возможно продолженная буквой. Если буквы нет, то число десятичное. Буква H или L указывает на шестнадцатеричное представление.

Действительное число всегда содержит десятичную точку. Кроме того, за числом может следовать порядок. Буква E означает "десять в степени". Действительное число имеет тип REAL.

```
число = целое | действительное
целое = цифра {цифра} |
                цифра {шестнадцатеричнаяЦифра} ("H"|"L")
```

```
действительное = цифра {цифра} "." {цифра} [Порядок]

Порядок = "E" ["+" | "-"] цифра {цифра}
шестнадцатеричнаяЦифра = цифра |"A"|"B"|"C"|"D"|"E"|"F"
цифра          = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

Примеры:

```
2025    100H = 256    12.3    4.567E8 = 456700000
```

Строка - это последовательность литер, заключенная в одинарные "'" или двойные '"' кавычки. При этом открывающей и закрывающей кавычкой должна быть одна и та же, не встречающаяся внутри строки кавычка. Строку из одной литеры можно задать её шестнадцатеричным кодом с суффиксом X.

```
строка = '"' {литера} '"' | "'" {литера} "'"
              | цифра {шестнадцатеричнаяЦифра} "X"
```

Примеры:

```
'OBERON-2M ORM'   "Don't worry!"   22X
```

Операции и разделители - специальные литеры, пары литер или зарезервированные слова, перечисленные ниже. Зарезервированные слова состоят только из заглавных букв и не могут использоваться как идентификаторы. Символы # и <> - синонимы, как и символы & и AND, ! и OR, ~ и NOT.

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

Многострочные комментарии - произвольная последовательность литер, заключенная в скобки (* и *). Комментарии могут быть вставлены между любой парой лексем. Комментарии не влияют на смысл программы. Комментарии могут быть вложенными.

Однострочные комментарии начинаются с последовательности литер // и действует на весь следующий текст до конца строки.

4. ОБЪЯВЛЕНИЯ И ПРАВИЛА ВИДИМОСТИ

Каждый идентификатор, встречающийся в программе и не являющийся стандартным, должен быть объявлен. Стандартные идентификаторы считаются предописанными и допускают использование в любой части программы. Объявления задают неизменные свойства идентификатора - константа, тип, переменная или процедура.

Область видимости идентификатора распространяется от точки объявления, до конца содержащего его блока (процедуры или модуля). Внутри области видимости идентификатор не может повторяться. Из этого правила есть

исключения.

Объявление типа T0, содержащее ссылку на другой тип T1, текстуально допускается, до объявления T1. В таком случае объявление типа T1 должно следовать далее и находиться в одном блоке видимости с типом T0.

Идентификаторы полей записей или методов видны только через селектор в составных именах.

```
    УточнИдент = идентификатор {"." идентификатор}
```

Идентификатор, экспортируемый из модуля, в своем объявлении должен быть помечен звездочкой "*". Такой, идентификатор можно использовать в других модулях, в списке импорта, которых присутствует текущий модуль. В таком случае, к идентификатору приставляется идентификатор его модуля. Приставка и идентификатор разделены точкой и вместе называются уточненным идентификатором.

```
    ИдентОпр  = идентификатор ["*"]
```

Следующие идентификаторы являются предопределенными. Их значение описано в разделе типы или процедуры:

```
    BOOLEAN     CHAR        SCHR
    INTEGER     BYTE        WORD
    LONGINT     REAL        BSET
    ADDRESS
    ABS   ADR   ASH   BIT   CHR   CAP   DEC
    ECL   FRE   FLT   GRG   GET   INC   INT
    ICL   LEN   LSH   MIN   MAX   MOV   NEW
    ODD   ORD   OUT   PRG   PUT   RSH   SET
    SYS   SQR   SIZE  TYP   VAL
```

5. ОБЪЯВЛЕНИЯ КОНСТАНТ

Объявление константы связывает ее идентификатор с результатом вычисления константного выражения.

```
    ОбъявлениеКонстанты = ИдентОпр "=" КонстВыражение
    КонстВыражение      = Выражение
```

Константное выражение вычисляется во время компиляции и может включать только другие константы и операторы. Примеры объявления констант:

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

6. ОБЪЯВЛЕНИЯ ТИПОВ

Тип данных определяет множество значений, которые могут принимать переменные этого типа, и связывает с этим типом идентификатор. Есть две разные структуры типов данных - массивы и записи. Они отличаются способом обращения к своим элементам.

```
    ОбъявлениеТипа = ИдентОпр "=" Тип
    Тип            = УточнИдент | ТипМассив | ТипЗапись
                                | ТипУказатель | ПроцедурныйТип
```

Примеры:

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

6.1 Базовые типы

Следующие основные типы предопределены и обозначаются стандартными идентификаторами:

```
    BOOLEAN    логические значения TRUE и FALSE
    CHAR       литеры стандартного набора ASCII
    BYTE       целые числа от -128 до 127
    WORD       целые числа от -32768 до 32767
    INTEGER    целые числа от -2147483648 до 2147483647
    LONGINT    целые числа от -9223372036854775808
                           до 9223372036854775807
    REAL       действительные числа IEEE 754
    BSET       множества целых чисел из диапазона от 0 до 31
    ADDRESS    = LONGINT
```

Целые типы и действительные типы вместе называются числовыми типами. Они образуют иерархию, в которой старший тип включает младший по значению:

```
    REAL >= LONGINT >= INTEGER >= WORD >= BYTE
```

6.2 Тип массив

Массив - структура данных, содержащая фиксированное количество элементов одного типа. Элементы массива обозначаются индексами, которые являются целыми числами от 0 до длины массива минус 1.

```
    ТипМассив = ARRAY длина {"," длина} OF Тип
    длина     = КонстВыражение
```

Объявление вида

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

сокращенное объявление

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

Примеры типа массив:

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

6.3 Тип запись

Запись - структура данных, состоящая из фиксированного числа элементов различных типов. Объявление типа запись определяет для каждого элемента тип и идентификатор, обозначающий этот элемент и называемый полем. Областью видимости идентификаторов полей является сама запись. Но они также доступны через обозначение поля в составном имени.

```
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 отмечены "*". Такие поля называются публичными, неотмеченные поля называются приватными.

Тип запись становиться расширяемой записью, если в объявление добавлены круглые скобки (). Такая запись допускает расширенное переопределение (наследование). Можно определить новый тип записи (потомок), как расширение существующего базового типа (предок). Такие записи в терминах объектно-ориентированного программирования называются объектами.

Все идентификаторы поля, объявленные в объекте потомке, должны быть отличны от идентификаторов полей, объявленных во всех его предках. Кроме того, в качестве предка можно использовать указательный тип. В таком случае в качестве типа предка расширенной записи используется базовый тип записи данного указателя.

Определение: тип T1 расширение типа T0, если он и есть T0, или если он расширение расширения T0. И наоборот, тип T0 является базовым типом для типа T1, если он и есть T1, или если он является базовым типом базового типа T1.

Примеры типов запись:

```
T0 = RECORD(  ) x : INTEGER END
T1 = RECORD(T0) y : REAL    END

date = RECORD day, month, year : INTEGER END
```

6.4 Тип указатель

Переменные указательного типа P, в качестве значений принимают указатели на переменные другого типа T. T - базовый тип для P. Базовый тип может быть только записью или массивом. Указательные типы расширяемых записей (объектов) наследуют отношение потомок-предок для своих базовых типов. Если тип T потомок типа T0, а P является указателем на T, то P также является потомком типа P0.

```
ТипУказатель = POINTER TO Тип
```

Объявление P = POINTER TO T, текстуально может предшествовать объявлению

идентификатора T, если при этом они находиться в одной области видимости. Любой указатель может принимать значение NIL. Значение NIL означает, что указатель ни на что не ссылается.

Вызов предопределенной процедуры NEW размещает переменную типа T в свободной памяти (куче) и присваивает этот адрес памяти переменной указательного типа. Все поля вновь размещенной записи или элементы массива очищаются (обнуляются). Вызов предопределенной процедуры FRE освобождает ранее выделенную память и устанавливает переменную указателя в NIL.

6.5 Тип процедура

Переменная процедурного типа T принимает в качестве значения процедуру P или NIL. Типы формальных параметров P должны совпадать с типами, указанными в списке формальных типов T. То же относится к типу результата в случае процедуры-функции. P не может быть стандартной процедурой, не может быть локальной процедурой, не может быть методом.

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

6.6 Тип нуль-строка

Тип нуль-строка это последовательности литер, оканчивающиеся литерой (0X). Длина такой строки - это количество литер в ней до первой нуль-литеры. Нуль-строка совместима по присваиванию с любой переменной объявленной как литерный массив.

Строки могут быть константами, или храниться в массиве литер. Для строк переменных нет предопределенных идентификаторов. Переменная массив из литерных элементов, становиться нуль-строкой, при добавлении суфикса $, к имени переменной.

Пример

```
string1 := 'Пример строки'
string2 := string1$
```

7. ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ

Объявление переменной добавляют переменную и связывают с ней уникальный идентификатор, тип данных и структуру. Все переменные, перечисленные в одном списке, получают одинаковый тип.

```
ОбъявлениеПеременных = СписокИдент ":" Тип
```

Тип данных определяет множество значений, которое может принимать переменная, операции, которые можно над ней выполнять и определяет структуру переменной.

Переменные указательных типов на тип расширяемая запись имеют, как статический тип (тип объявления), так и динамический тип (тип их значения при выполнении программы). Для указателей переменных типа расширяемая запись, динамический тип может быть потомком их статического типа. Статический тип определяет, какие поля записи доступны. Динамический тип определяет, какие динамические методы будут вызваны.

Примеры объявлений переменных:

```
    i,j,k : INTEGER
    x,y,z : ARRAY 8 OF REAL
```

8. ВЫРАЖЕНИЯ

Выражения - это конструкции, задающие правила вычисления значений с использованием констант и значений переменных, с применением операций и процедур-функций. Выражения состоят из операндов и операций. Круглые скобки используются для группировки операций и операндов.

8.1 Операнды

За исключением чисел, строк и множеств, то есть литерных констант, операнды определяются обозначениями. Обозначение может быть идентификатором константы, переменной или процедуры. Этот идентификатор может быть уточнен идентификатором модуля, а за ним могут следовать селекторы, для элемента структуры.

Если структура - массив A, то обозначение A[E] достает элемент A, индекс которого, текущее значение выражения E. Тип индекса A должен быть целым. Обозначение вида A[E1,E2,...,En] означает то же, что и A[E1][E2]...[En].

Если структура - запись R, то R.f обозначает поле f записи R. Обозначение P^ используется для переменной, на которую ссылается указатель P. Если P - составное имя, обозначающее указатель, то P^ обозначает переменную, на которую ссылается P. Составные имена P^.f, P^[e] и P^$ можно сокращать до P.f, P[e] и P$ - селекторы записи, массива и строки подразумевают автоматическое разыменование.

```
    Обозначение = УточнИдент {"." идентификатор
                        | "[" СписокВыражений "]"
              | "^" | "(" УточнИдент ")" } ["$"]
    СписокВыражений = Выражение {"," Выражение}
```

Если обозначение - переменная, то оно ссылается на текущее значение переменной. Если обозначение - процедура, то обозначение без списка параметров ссылается на эту процедуру. Если за обозначением следует список параметров (возможно, пустой), обозначение предполагает активацию процедуры и обозначает возвращаемый результат её исполнения. Фактические параметры (их типы) должны соответствовать формальным параметрам, указанным в объявлении процедуры.

Примеры обозначений:

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

8.2 Операции

Синтаксис операций определяет старшинство операций в соответствии с четырьмя классами операций. Операция отрицания "~" имеет наивысший приоритет, за ней следуют так называемые операции типа умножения, затем операции типа сложения и, наконец, с низшим приоритетом, операции отношения. Последовательности операций одного приоритета выполняются

слева направо.

```
Выражение = ПростоеВыражение [Отношение ПростоеВыражение]
Отношение = "=" | "#" | "<" | "<=" | ">" | ">=" | IN | IS

ПростоеВыражение = ["+"|"-"] Слагаемое
                              {ОперацияСложения Слагаемое}
ОперацияСложения = "+" | "-" | "!"
Слагаемое = 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] ")"
```

Имеющиеся операции перечислены ниже в таблице. В некоторых случаях одним и тем же знаком операции обозначаются несколько различных операций. В этих случаях конкретная операция определяется типами операндов.

8.2.1 Логические операции

Лексема	Операция
!	Логическое сложение
&	Логическое умножение
~	Отрицание

Эти операции применяются к операндам BOOLEAN и дают результат BOOLEAN.

```
p ! q    означает "если p, то TRUE, иначе q"
p & q    означает "если p, то q, иначе FALSE"
~ p      означает "не p"
```

Второй операнд логического ! вычисляется только если результат первого равен FALSE. Второй операнд логического & вычисляется, только если результат первого равен TRUE.

8.2.2 Арифметические операции

Лексема	Операция
+	сложение
-	вычитание
*	умножение
/	действительное деление
DIV	целое деление
MOD	остаток от деления

Операции +, -, * и / применимы к операндам числовых типов. Типом результата будет REAL в случае операции деления / или если один из операндов имеет тип REAL. Иначе типом результата будет LONGINT если тип одного из операндов - LONGINT, или INTEGER в любом другом случае. Для одноместной операции "-" обозначает перемену знака, а "+" тождественную операцию. Операции DIV и MOD применимы только к целочисленным операндам.

8.2.3 Операции над множествами

Лексема	Операция
+	объединение множеств
-	разность множеств
*	пересечение множеств
/	симметрическая разность множеств

Операции над множествами применимы к операндам типа BSET и дают результат типа BSET.

```
x IN (sl+s2)  эквивалентно  (x IN s1) ! (x IN s2)
x IN (si-s2)  эквивалентно  (x IN s1) & ~(x IN s2)
x IN (sl*s2)  эквивалентно  (x IN s1) & (x IN s2)
x IN (si/s2)  эквивалентно  (x IN s1) # (x IN s2)
```

Одноместный минус обозначает дополнение множества, то есть все элементы, которые не являются элементами множества операнда. Операции над множествами не являются ассоциативными

```
((a+b)-c # a+(b-c)).
```

Конструктор множества задает множество перечислением его элементов и диапазонов элементов a..b между фигурными скобками.

8.2.4 Отношения

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

Отношения - логические операции и дают результат BOOLEAN. Операции сравнения <, <=, >, >= применимы к строкам, числовым типам и CHAR. Отношения = и # применимы дополнительно к типам BOOLEAN, BSET, указателям и процедурным типам.

x IN s обозначает "x является элементом s". x должен иметь тип INTEGER и s должен быть типом BSET.

v IS T означает "v имеет тип T" и называется проверкой типа. Это применимо, если v является переменной типа расширяемая запись или указателем на тип расширяемая запись, и если T является потомком статического типа v, или что тоже самое v - предок расширяемой записи типа T.

Примеры выражений:

```
1987                    (INTEGER)
i DIV 3                 (INTEGER)
~ p ! q                 (BOOLEAN)
(I + j) * (i-j)         (INTEGER)
s - {8, 9, 13}          (BSET)
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)
```

8.2.5 Сложение строк

Лексема	Операция
+	конкатенация

Операция конкатенации применяется к операндам, имеющим тип строки. Результирующая строка состоит из литер первого операнда, за которыми следуют литеры второго операнда, замыкаемые нуль-символом.

Примеры сложения:

```
string := sring1$ + '    ' + string2$
```

9. ОПЕРАТОРЫ

Операторы означают действия. Имеются простые и структурные операторы. В простые операторы в качестве составных частей не входят другие операторы. Простые операторы - присваивание, вызов процедуры и возврат. Структурные операторы состоят из частей, являющихся операторами. Они используются для организации выполнения последовательности, условий, выбора и повторений. Оператор может быть пустым, что означает отсутствие действия. Пустой оператор позволяет ослабить правила пунктуации в операторных последовательностях.

```
Оператор = [Присваивание | ВызовПроцедуры | ОператорIF
      | ОператорCASE | ОператорWHILE | ОператорREPEAT
      | ОператорFOR | ОператорCODE | RETURN [Выражение]]
```

9.1 Присваивания

Присваивание служит для замены текущего значения переменной новым значением, определяемым значением выражения. Знаком операции присваивания служит ":=", который читается "присвоить".

```
Присваивание = Обозначение ":=" Выражение
```

Слева от знака присваивания находится переменная. После присваивания переменная имеет значение, полученное в результате вычисления выражения. Старое значение теряется. Тип переменной должен быть совместим по присваиванию с типом выражения.

Говорят, что типы операндов совместимы по присваиванию если они одинаковые, если переменная указатель или процедура, а выражение константа NIL. Если переменная массив литер, а выражение строка. Открытые массивы равного базового типа. В случае расширяемых записей тип выражения должен быть указательным расширением типа переменной.

Примеры присвоений:

```
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 служит для активации процедуры. Он может содержать список фактических параметров, которые заменяют формальные параметры, определенные в объявлении процедуры. Соответствие устанавливается в порядке следования параметров в списках фактических и формальных параметров. Имеются два вида параметров: параметры-переменные и параметры-значения.

Для параметров-переменных фактический параметр должен быть обозначением переменной. Если он обозначает элемент структурной переменной, селектор вычисляется при подстановке фактического параметра на место формального, до исполнения процедуры. В случае параметра—значения фактический параметр должен быть выражением. Это выражение вычисляется до активации процедуры, и получающееся значение присваивается формальному параметру. Типы соответствующих формальных и фактических параметров должны быть идентичны в случае параметров-переменных и совместимы по присваиванию в случае параметров-значений.

```
ВызовПроцедуры = Обозначение [ФактическиеПараметры]
```

Примеры вызова процедур:

```
WriteInt(i*2+1)
INC(w[k].count)
t.Insert("John")
```

9.3 Последовательность операторов

Последовательность операторов означает последовательное выполнение действий, определяемых входящими в нее операторами, разделяемых точкой с запятой.

```
ПоследовательностьОператоров = Оператор {";" Оператор}
```

9.4 Условный оператор

```
ОператорIF = IF Выражение THEN ПоследовательностьОператоров
             {EI Выражение THEN ПоследовательностьОператоров}
             [ELSE ПоследовательностьОператоров] END
```

Выражения, следующие за символами IF и EI, имеют тип BOOLEAN. Они вычисляются в порядке следования. Если значение логического выражения FALSE, последовательность операторов пропускается. Если TRUE, исполняется последовательность операторов этой ветки. Если присутствует предложение ELSE, его последовательность операторов исполняется тогда и только тогда, когда все логические выражения дали значение FALSE.

Пример:

```
IF (ch >= "A") & (ch <= "Z") THEN ReadIdentifier
EI (ch >= "0") & (ch <= "9") THEN ReadNumber
EI (ch =  "'") ! (ch =  '"') THEN ReadString
```

```
    ELSE SpecialCharacter END
```

9.5. Оператор выбора

Оператор выбора определяет выбор и исполнение последовательности операторов в соответствии со значением выражения. Сначала вычисляются выражения выбора, затем исполняется та последовательность операторов, список меток выбора которой содержит полученное значение. Если среди меток выбора нет значения выражения, выбирается последовательность операторов, следующая за символом ELSE. Выражение выбора должно иметь целый или литерный тип.

```
    ОператорCASE = CASE Выражение OF Вариант {"|" Вариант}
                   [ELSE ПоследовательностьОператоров] END

    Вариант = [СписокМетокВарианта ":"
                               ПоследовательностьОператоров]
    СписокМетокВарианта = МеткиВарианта {"," МеткиВарианта}

    МеткиВарианта = КонстантноеВыражение [".."
                                     КонстантноеВыражение]
```

Пример:

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

9.6 Цикл с условием продолжения

Цикл с условием продолжения определяет повторяющееся исполнение последовательности операторов в зависимости от значения логического выражения. Это выражение вычисляется до очередного исполнения последовательности операторов. Повторение прекращается, как только результатом этого вычисления становится значение FALSE.

```
    ОператорWHILE = WHILE Выражение DO
                            ПоследовательностьОператоров END
```

Примеры:

```
    WHILE i > 0 DO i := i DIV 2; k := k + 1 END
    WHILE (t # NIL) & (t.key # i) DO t := t^.left END
```

9.7 Цикл с условием окончания

Цикл с условием окончания определяет повторяемое исполнение последовательности операторов в зависимости от значения логического выражения. Это выражение вычисляется после каждого исполнения последовательности операторов, и повторение прекращается, как только результатом этого вычисления становится значение TRUE. Таким образом, последовательность операторов исполняется хотя бы один раз.

```
    ОператорREPEAT = REPEAT ПоследовательностьОператоров
                     UNTIL Выражение
```

Пример:

```
REPEAT k := i MOD j; i := j; j := k  UNTIL j = 0
```

9.8 Цикл с шагом

Цикл с шагом определяет повторяющееся выполнение последовательности операторов фиксированное число раз, для прогрессии значений, присваиваемых целой переменной, называемой управляющей переменной цикла.

```
ОператорFOR = FOR идентификатор
                          ":=" Выражение (TO | DT) Выражение
                        DO ПоследовательностьОператоров END
```

Значение управляющей цикла меняется с шагом 1 для TO и с шагом -1 для DT (DownTo).

Примеры:

```
FOR i := 0 TO 79 DO k := k + a[i] END
FOR i := 79 DT 0 DO k := k + a[i] END
```

9.9 Ассемблерные вставки

Ассемблерные вставки позволяют реализовать низкоуровневые особенности системного программирования, обеспечивающие взаимодействие с операционными системами и недоступные иным способом. Ниже приводится спецификация простого оператора, поддерживаемая компилятором для виртуального процессора ORM (Oberon Risc Machine):

```
ОператорCODE = CODE целое {целое} END
```

9.10 Операторы возврата

Оператор возврата состоит из слова RETURN, за которым, возможно, следует выражение. Он приводит к завершению процедуры (или тела модуля), а выражение определяет значение, возвращаемое как результат процедуры-функции. Тип этого выражения должен быть совместим по присваиванию с типом результата, определенным в заголовке процедуры.

Процедура-функция требует присутствия оператора возврата, задающего значение результата. Таких операторов может быть несколько, но выполняется только один из них. В собственно процедурах под оператором возврата неявно подразумевается конец тела процедуры. Поэтому явное задание оператора возврата рассматривается как дополнительная точка завершения.

10. ОБЪЯВЛЕНИЯ ПРОЦЕДУР

Объявление процедуры состоит из заголовка процедуры и тела процедуры. Заголовок определяет имя процедуры, формальные параметры и тип результата (если он есть). Для связанных с типом процедур в объявлении также определяется параметр-приемник. 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 после списка параметров указывается тип результата. Тело процедуры-функции должно содержать оператор возврата RETURN, который определяет ее результат.

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

В добавок к их формальным параметрам и локально объявленным переменным, идентификаторы объявленные глобально также видны в процедуре (за исключением, имеющих имена, совпадающие с именами локальных идентификаторов).

Вызов процедуры изнутри ее объявления подразумевает рекурсивную активацию.

```
ОбъявлениеПроцедуры = ЗаголовокПроцедуры ";"
                          ТелоПроцедуры идентификатор
ЗаголовокПроцедуры = PROCEDURE [Приемник]
                       ИдентОпр [ФормальныеПараметры]
ТелоПроцедуры = ПоследовательностьОбъявлений
                [BEGIN ПоследовательностьОператоров]
                                [RETURN Dыражение] END
ПоследовательностьОбъявлений =
                    {CONST {ОбъявлениеКонстант ";"} }
                    {TYPE {ОбъявлениеТипов ";"} }
                    {VAR {ОбъявлениеПеременных ";"} }
                            {ОбъявлениеПроцедуры ";"}
```

Если объявление процедуры содержит параметр-приемник, процедура рассматривается как связанная с типом и называется методом.

10.1 Формальные параметры

Формальные параметры - идентификаторы, обозначающие фактические параметры, задаваемые при вызове процедуры. Соответствие между формальными и фактическими параметрами устанавливается при вызове процедуры. Имеется два вида параметров: параметры-значения и параметры-переменные. Разновидность параметра указывается в списке формальных параметров. Параметры-значения представляют собой локальные переменные, которым в качестве начальных значений присваиваются результаты вычисления фактических параметров. Параметры-переменные соответствуют фактическим параметрам, являющимся переменными, и они подставляются вместо этих переменных. Параметры-переменные указываются символом VAR, у параметров-значений символа VAR нет.

Процедура-функция без параметров имеет пустой список параметров. И вызывается обозначением функции, чей список фактических параметров также пуст. Область видимости формального параметра начинается с места его объявления и заканчивается концом блока процедуры его объявления.

```
ФормальныеПараметры = "(" [СекцияФП
                    {";" СекцияФП}] ")" [":" УточненИдент]
СекцияФП = [VAR] идентификатор
                              {"," идентификатор} ":" Тип
```

Тип каждого формального параметра определен в списке параметров. Для параметров-переменных он должен быть идентичным типу фактического параметра, кроме расширяемых записей (объектов), когда формальный параметр должен быть базовым типом фактического параметра.

Если тип формального параметра определен как ARRAY OF T, то такой параметр называется открытым массивом, а подставляемый на его место фактический параметр может быть произвольной длины.

Если формальный параметр определен как процедурный тип, то его фактический параметр должен быть либо глобальной процедурой, либо переменной (или параметром) процедурного типа и не может быть стандартной процедурой или методом.

Тип результата процедуры может быть простым типом, указателем, процедурой, записью или массивом.

Пример объявления процедур:

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

10.2 Процедуры, связанные с типом (методы)

Глобальные процедуры могут быть связаны с типом расширяемая запись, объявленным в том же самом модуле. В этом случае считается, что процедура связана с типом объект и называется методом. Связь задается типом приемника в заголовке объявления процедуры. Приемник должен быть параметром-переменной для статического типа расширяемая запись, или параметром-значением для указательного объектного типа. Процедура, связанная с таким типом, рассматривается как локальная для него и доступ к ней осуществляется через селектор.

```
ЗаголовокПроцедуры = PROCEDURE [Приемник]
           ИдентОпр [ФормальныеПараметры]
Приемник = "(" [VAR] имя ":" имя ")"
```

Если процедура P связана с типом T0, она неявно также связана с любым типом T1, который является расширением T0. Однако процедура P? (с тем же самым именем, что и P) может быть явно связана с T1, перекрывая в этом случае связывание с P. P? рассматривается как переопределение P для T1. Формальные параметры P и P? должны совпадать. Если P и T1 экспортируются, P? также должна экспортироваться.

Если v — обозначение, а P — связанная процедура, то v.P обозначает процедуру P, связанную с динамическим типом v. Если динамический тип v отличается от статического типа v, то вызываемая процедура P может отличаться от процедуры, связанной со статическим типом v. Объект v передается приемнику процедуры P согласно правилам передачи параметров.

Если r — параметр-приемник, объявленный с типом T, r.P^ обозначает процедуру P непосредственного предка, связанную с базовым для T типом. Соответственно r.P^^ - обозначает процедуру пра-предка, r.P^^^ - пра-пра-предка и т.д.

Такие методы поддерживают полиморфизм. Вызов всегда будет выполнять метод, соответствующий динамическому типу переменной.

Примеры:

```
PROCEDURE (t : Tree) Insert(node : Tree);
   VAR p, father : Tree;
BEGIN
   p := t;
   REPEAT father := p;
      IF node.key = p.key THEN RETURN END;
      IF node.key < p.key THEN p := p.left
      ELSE p := p.right END
   UNTIL p = NIL;
   IF node.key < father.key THEN father.left := node
                            ELSE father.right := node
   END;
   node.left := NIL; node.right := NIL
END Insert;

PROCEDURE (t : CenterTree) Insert(node : Tree);
BEGIN (*переопределение*)
   WriteInt(node.width);
   (* процедура Insert, связанная с Tree *)
   t.Insert^(node)
END Insert;
```

10.3 Стандартные процедуры

Стандартные процедуры являются предопределенными. Некоторые процедуры - обобщенные, то есть они применимы к операндам нескольких типов. Стандартные процедуры перечислены ниже. Буква v обозначает переменную, x и n - выражения, T - тип.

Процедуры-функции:

Имя	Тип аргументов	Тип результата	Функция
ABS(x)	числовой тип	тип x	абсолютное значение
BIT(x)	INTEGER	BSET	{i | ODD(x DIV 2^i)}
SET(x)	INTEGER	BSET	{i | ODD(x DIV 2^i)}
CAP(x)	литерный тип	тип x	x - соответствующая букве большая буква
SQR(x)	числовой тип	REAL	корень квадратный
MIN(T)	T = элементарный тип	T	min значение типа T
	T = BSET	INTEGER	0
MAX(T)	T = элементарный тип	T	max значение типа T
	T = BSET	INTEGER	31
ASH(x,n)	x: INTEGER	INTEGER	арифметический
	x: LONGINT	LONGINT	n > 0 влево
	n: целый тип		n < 0 вправо
LSH(x,n)	x: INTEGER	INTEGER	логический сдвиг

	x: LONGINT	LONGINT	n > 0 влево
	n: целый тип		n < 0 вправо
RSH(x,n)	x: INTEGER	INTEGER	циклический сдвиг
	x: LONGINT	LONGINT	n > 0 влево
	n: целый тип		n < 0 вправо
LEN(v,x)	v: массив	INTEGER	длина v в измерении x,
	x: константа		первое измерение = 0
LEN(v)	массив	INTEGER	эквивалентно LEN(v,0)
LEN(v$)	строка	INTEGER	длина строки до 0X
MAX(x,y)	<= INTEGER	INTEGER	большее из x и y
	целый тип	LONGINT	
	числовой тип	REAL	
	литерный тип	CHAR	
MIN(x,y)	<= INTEGER	INTEGER	меньшее из x и y
	целый тип	LONGINT	
	числовой тип	REAL	
	литерный тип	CHAR	

Функции преобразования типов:

Имя	Тип аргументов	Тип результата	Функция
CHR(x)	целый тип	CHAR	литера с порядковым номером x
INT(x)	действительный	LONGINT	ближайшее целое, к x
FLT(x)	целый тип	REAL	тождественно
ODD(x)	целый тип	BOOLEAN	x MOD 2 = 1
ORD(x)	BOOLEAN	INTEGER	порядковый номер x
	CHAR	INTEGER	порядковый номер x
	BSET	INTEGER	SUM i: i IN x: 2^i
SIZE(T)	любой тип	INTEGER	количество байт, требуемое для T

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
ICL(v,x)	v: BSET; x: INTEGER	v := v + {x}
ECL(v,x)	v: BSET; x: INTEGER	v := v - {x}

NEW(v)	тип указатель	разместить v^
FRE(v)	тип указатель	освободить v^
OUT(v)	простой тип	печать в консоль

10.4 Системно-зависимые процедуры

Процедуры-функции:

Имя	Тип аргументов	Тип результата	Функция
ADR(v)	любой	ADDRESS	адрес переменной v
TYP(v)	объектный тип	INTEGER	tag тип объекта v
SYS(v)	INTEGER	INTEGER	зависит от v
VAL(T,x)	T, x: любой	T	x интерпретируется как значение типа T

Собственно процедуры:

Имя	Типы аргумента	Функция
GET(a,v)	a: LONGINT; v: любой основной тип, указатель, процедурный тип	v := Память[a]
PUT(a,x)	a: LONGINT; x: любой основной тип, указатель, процедурный тип	Память[a] := x
GRG(n,v)	n: целая константа; v: любой основной тип, указатель, процедурный тип	v := Регистр n
PRG(n,x)	n: целая константа; x: любой основной тип, указатель, процедурный тип	Регистр n := x
MOV(a0, a1,n)	a0, a1: LONGINT; n: целый	Память[a1..a1+n-1] := Память[a0..a0+n-1]

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 идентификатор ";" [СписокИмпорта]
                ПоследовательностьОбъявлений
               [BEGIN ПоследовательностьОператоров]
               [CLOSE ПоследовательностьОператоров]
                END идентификатор "."
СписокИмпорта = IMPORT Импорт {"," Импорт} ";"
Импорт        = [идентификатор ":="] идентификатор
```

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 идентификатор x, объявленный в M1, вызывается в импортирующем модуле как M.x.

Идентификаторы, которые должны быть видны в клиентских модулях, должны быть отмечены звездочкой (отметкой экспорта) в их объявлении. Переменные всегда экспортируются в режиме для чтения и изменения.

Последовательность операторов, следующая за лексемой BEGIN, выполняется, когда модуль добавляется в систему (загружается). Отдельные процедуры (без параметров) могут быть активированы из системы, и эти процедуры служат в качестве команд.

Последовательность операторов, следующая за лексемой CLOSE, выполняется, когда модуль удаляется из системы (выгружается).

Содержимое переменных, объявленных в модуле, очищается перед выполнением тела модуля. Это подразумевает, что в том числе все переменные указательных или процедурных типов инициализируются в NIL.

12. STD МОДУЛИ

STD-модуль - специальный модуль, предоставляющий бесшовный интерфейс для взаимодействия с внешними библиотеками и API операционной системы. Он содержит только объявления констант, типов и заголовков процедур (без реализации).

```
    МодульSTD = MODULE идентификатор "[" строка "]" ";"
                       {CONST {ОбъявлениеКонстант ";"}}
                       {TYPE {ОбъявлениеТипов ";"}}
                       {PROCEDURE
                          ИдентОпр [ФормальныеПараметры]}
                  END идентификатор "."
```

Как правило внешние библиотеки имеют расширение, зависящее от целевой платформы (операционной системы). В частности, dll - библиотеки Windows, so - Linux, dylib - Mac OS и js - Wasm.