Глава 8. ПОДПРОГРАММЫ И МОДУЛИ

- Процедуры и функции
- Область видимости переменных
- Формальные и фактические параметры
- Параметры-значения и параметры-переменные
- Особенности функций
- Параметры структурированных типов
- Рекурсия
- Процедурный тип
- Параметры-функции и параметры-процедуры
- Стандартные математические функции и процедуры
- Модуль, структура модуля
- Стандартные модули

ПРОЦЕДУРЫ И ФУНКЦИИ

Подпрограммы (процедуры или функции) представляют собой относительно самостоятельные фрагменты программы, оформленные особым способом и снабженные именем. Упоминание этого имени в тексте программы называется **вызовом** процедуры (функции).

Подпрограммы представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд относительно независимых частей, что имеет смысл по двум причинам.

Во-первых, это средство экономии памяти и других ресурсов: каждая подпрограмма существует в программе в единственном экземпляре, в то время как обращаться к ней можно многократно из разных точек программы.

Во-вторых, при таком разбиении максимально реализуется принцип нисходящего проектирования. В этом случае алгоритм представляется в виде последовательности относительно крупных подпрограмм, реализующих более или менее самостоятельные смысловые части алгоритма.

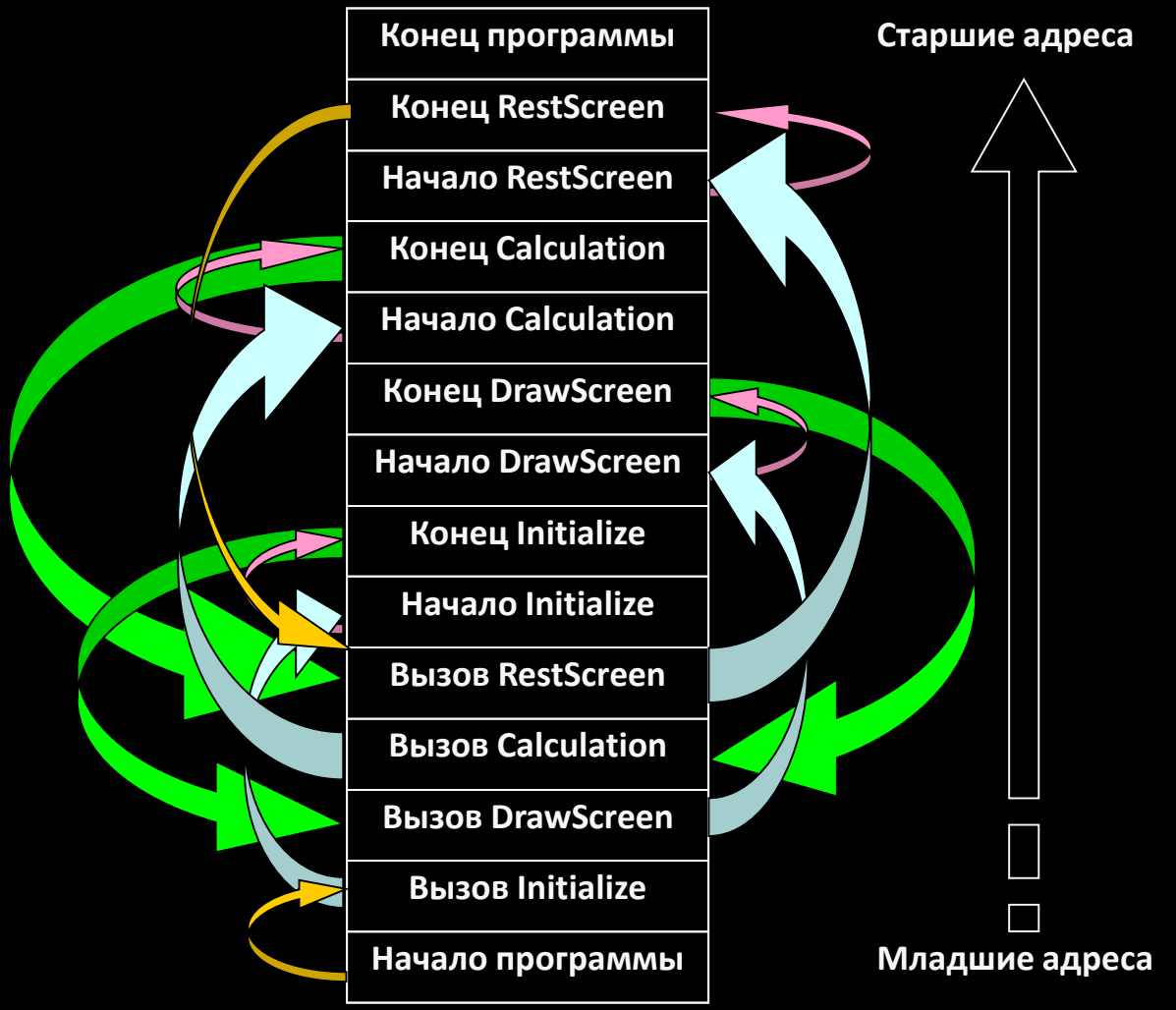
```
Program Calculator;
   Begin
      Initialize;
      DrawScreen;
      Calculation;
      RestScreen;
   End.
```

СТРУКТУРА ПОДПРОГРАММЫ, ЗАГОЛОВКИ ПРОЦЕДУРЫ И ФУНКЦИИ

ВЫЗОВ ПОДПРОГРАММ

Машинные команды, из которых состоит любая программа, хранятся в ОЗУ. Выполнение программы происходит путем последовательного чтения команд, начиная с первой; следующей выполняется команда, расположенная "выше" только что выполненной ("естественный" порядок выполнения команд"). Адрес байта, где находится текущая команда, называется **активной точкой программы**.

Естественный порядок выполнения команд нарушается при обращении к подпрограммам.



СТЕК В ПАМЯТИ

Адрес точки возврата (адрес машинной команды в момент вызова подпрограммы) хранится в ячейках специальной области ОЗУ, которая называется **стеком**. Кроме того, стек в памяти используется для хранения значений локальных переменных подпрограммы во время выполнения операторов, входящих в ее тело. Принцип его работы совпадает с организацией динамической структуры данных, имеющей такое же имя – стек. Стек – это LIFO-устройство (Last In First Out). Стек в памяти заполняется сверху вниз.

После выполнения всех операторов подпрограммы стек освобождается для хранения данных следующей вызванной подпрограммы. В последнюю очередь считывается адрес команды, которая стоит после вызова выполненной подпрограммы в основном блоке (адрес точки возврата).

ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ

Глобальные константы, типы, переменные – это те, которые объявлены в программе вне процедур или функций. Они доступны из любого места программы.

Локальные – это константы, типы, переменные, существующие только внутри процедур или функций, и объявленные либо в соответствующих разделах *Const*, *Type*, *Var* внутри данной процедуры или функции, либо в списке формальных параметров (как параметры-значения).
Глобальные переменные хранятся в области данных, локальные – в стеке.

Область видимости переменной (константы, типа) – это ряд операторов, в которых переменная "видна". Все имена, описанные внутри подпрограммы (локальные данные), локализуются в ней, т.е. они как бы "невидимы" снаружи подпрограммы. При наличии вложенных подпрограмм переменные данного уровня являются "видимыми", т.е. доступными для всех нижних уровней описания процедур. Подпрограммы, наряду со своими локальными данными, могут использовать и модифицировать и глобальные. Для этого лишь нужно, чтобы описание процедуры (функции) стояло в тексте программы ниже, чем описание соответствующих глобальных типов, констант и переменных.

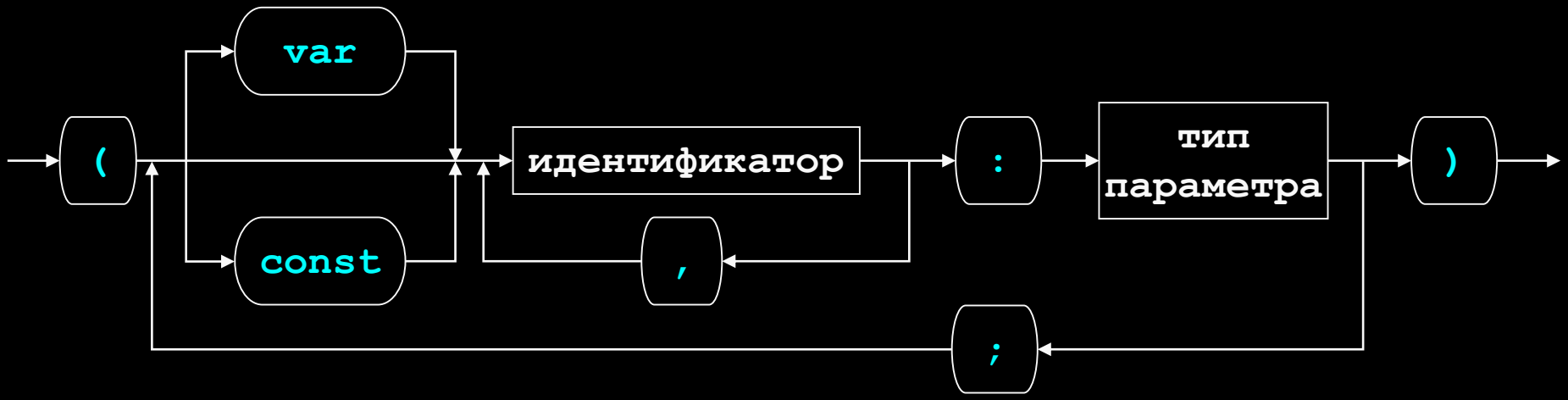
Время жизни переменной – это время, в течение которого переменная связана с определенной ячейкой памяти.

ГЛОБАЛЬНЫЕ И ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

```
Program Main;
Var
   Xmain, Ymain : Real; {глобальные переменные}
Procedure P1;                {описание процедуры}
   Var
      Res : Real;            {локальная переменная}
   begin
      Res := 0.5;
      Ymain := Res + Xmain*Ymain;
      Xmain := Xmain + 1
   end;
...
Var          {глобальные переменные, недоступные в P1)
   Zmain : Extended;
...
Begin
   ...
   P1;                       {вызов процедуры}
   ...
End.
```

ПАРАМЕТРЫ

Параметры обеспечивают обмен значениями между вызывающей частью программы и вызываемой подпрограммой. Могут быть параметры-значения, параметры-переменные или параметры-константы.



Описываемые в заголовке подпрограммы параметры называются **формальными**:

```
Procedure P1 (A : Integer; Var B : Real);
```

а те, которые подставляются на их место при вызове – **фактическими**, т. к. они при выполнении программы замещают все вхождения в подпрограмму своих формальных "двойников":

```
P1(A_fact,B_fact);
```

ПАРАМЕТРЫ

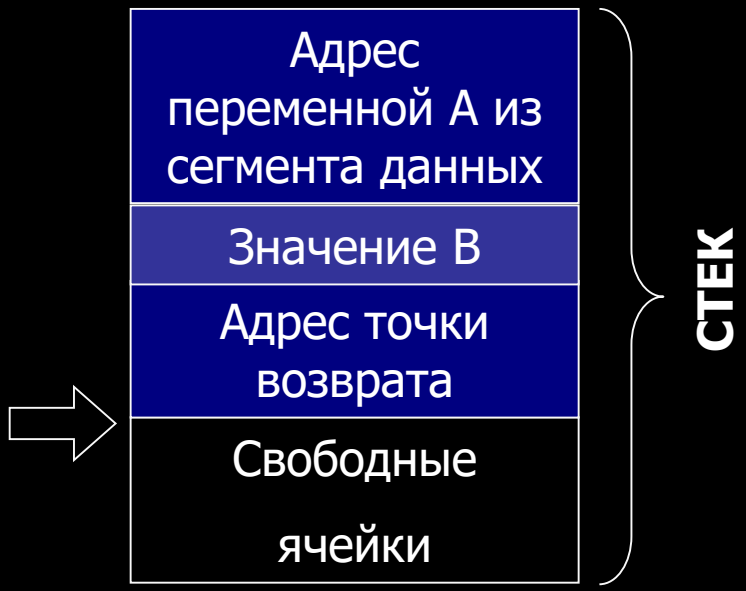
Если параметр описан как **параметр-значение**, то в подпрограмму передается копия значения этого фактического параметра, и никакие изменения этой копии не возвращаются в вызывающий блок. Параметр-значение – это локальная переменная подпрограммы, стартовое значение которых задается при вызове подпрограммы из внешних блоков.

Если параметр описан как **параметр-переменная** (служебное слово Var в заголовке) или как **параметр-константа** (слово Const), то в подпрограмму передаются адреса фактических параметров. Все изменения параметра-переменной в подпрограмме происходят с переменной, переданной в качестве фактического параметра. Параметр-константа не может изменяться в подпрограмме.

На место фактического параметра-значения можно подставлять литерал, выражение, идентификатор (переменную или константу), а на место параметра-переменной и параметра-константы – только идентификатор.

ПАРАМЕТРЫ

```
                                    {$J+}
Const
    A : Integer = 5; {типизированные константы}
    B : Integer = 7;
Procedure Plus (Var C : Integer; B : Integer);
    begin
        C := C + C;
        B := B + B;
        Writeln(C,B)
    end;
Begin
    Writeln(A,B);
    Plus(A,B);
    Writeln(A,B)
End.
```



ФУНКЦИИ

Кроме обмена значениями через параметры функция возвращает в вызываемый блок свое значение в виде скаляра, строки или указателя. Для присвоения функции значения ее имя должно появиться хотя бы однажды в левой части оператора присваивания в теле самой функции (альтернатива – *Result*). Функция имеет побочный эффект, если она производит другие действия, не связанные с вычислениями своего значения.

```
Var                                    {функция вычисления X^Y}
   X,Y : Real;
Function Power (Arg, P : Real) : Real;
   begin
      if Arg <> 0 then
         Power := exp(P * ln(abs(Arg))) {Result :=...}
      else
         if P = 0 then
            Power := 1                 {Result :=...}
         else   Power := 0             {Result :=...}
   end;
Begin
   Readln(X,Y);   Writeln(Power(X,Y))
End.
```

ПАРАМЕТРЫ СТРУКТУРИРОВАННЫХ ТИПОВ

Чтобы передать в подпрограмму массив (строку, запись,...), нужно сначала описать его тип. Следствие – возможные затраты памяти.

```
Type
      vector = array [1..100] of Real;
Procedure Sum (A : vector; Var B : vector);
      begin...end;
```

Открытые массивы:

```
Procedure Sum (A : array of Real; Var B : array of Real);
      Var i : Word;
      begin
            for i := 0 to High(A) do B[i] := A[i]+B[i]
      end;
Var A1,A2 : array [1..100] of Real;
Begin
      {инициализация массивов A1 и A2}
      Sum(A1,A2);
      {вывод A2}
End.
```

РЕКУРСИЯ

Применительно к практическому программированию под **рекурсией** понимается вызов функции (процедуры) из тела этой же самой функции (процедуры). Необходимым условием работоспособности рекурсивных подпрограмм является наличие условия окончания рекурсивных вызовов.

```
Var                              {вычисление n!}
   N : Word;
Function Fact (m : Word) : Int64;
   begin
      if m = 0 then
         Fact := 1
      else
         Fact := m*Fact(m-1)
   end;
Begin
   Readln(N);
   Writeln('n! = ',Fact(N))
End.
```

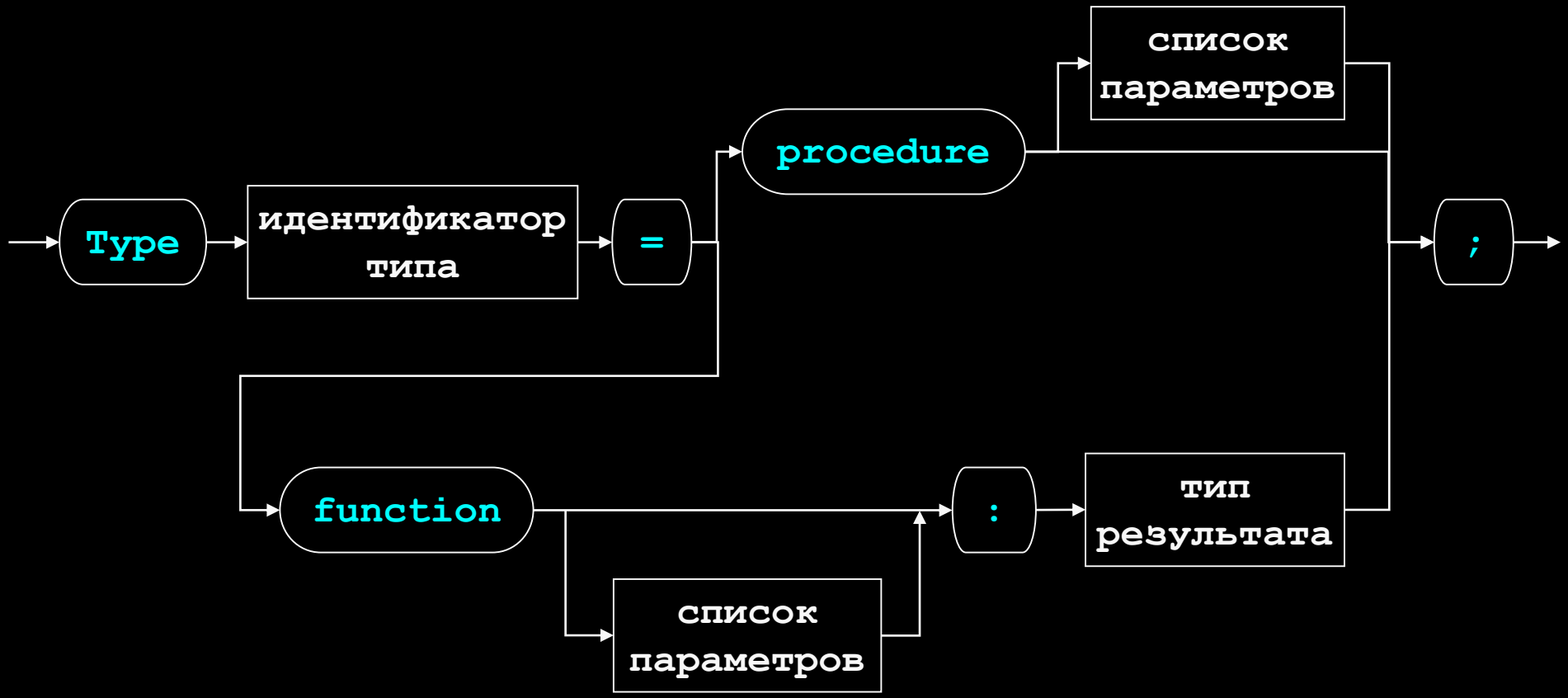
РЕКУРСИЯ

Реализация рекурсивных алгоритмов при большом числе итераций ограничена доступным объемом памяти; каждый отложенный вызов процедуры или функции – это свой набор значений всех локальных переменных этой подпрограммы (фрейм активации), размещенных в стеке.

```
                {вычисление n! итерационным методом}
Function Fact (m : Word) : Int64;
   Var
      I : Word; P : Int64;
   begin
      P := 1;
      if m = 0 then
         Fact := 1
      else
         begin
            for I := 1 to m do P:=P*I;
            Fact := P
         end
   end;
```

ПРОЦЕДУРНЫЙ ТИП

Процедурный тип позволяет интерпретировать процедуры и функции как данные-объекты, которые можно использовать, в частности, при передаче параметров. Значением переменной процедурного типа является конкретная процедура или функция.



ПАРАМЕТРЫ-ФУНКЦИИ И ПАРАМЕТРЫ-ПРОЦЕДУРЫ

```
                {вычисление интеграла функции}
Type
   FuncType = Function (X : Real) : Real;
Procedure Integral (LowerLimit, UpperLimit : Real;
                    Var Result : Real;
                    Funct : FuncType);
   begin ... end;
Function SinExp (Arg : Real) : Real;
   begin
      SinExp := Sin(Arg) + Exp(Arg)
   end;
Var
   Res : Real;
...
Begin
   ...
   Integral (0, 1, Res, SinExp);
   ...
End.
```

ПРОЦЕДУРА – ПОЛЕ ЗАПИСИ

```
Type
   ProcType = Procedure;
   RecType = record
             X,Y : Integer;
             P : ProcType;
             end;
Var
   Rec1,Rec2 : RecType;
Procedure P1;
   begin
   ...
   end;
Begin
   ...
   with Rec1 do P := P1;
   ...
End.
```

Процедурные переменные совместимы с переменными типа *Pointer* (их значения – адреса). Процедуры и функции – это адреса.

СТАНДАРТНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ И ПРОЦЕДУРЫ

ABS(X) – возвращает абсолютное значение аргумента X (X – целое/вещественное, результат – как у аргумента);
SQR(X) – возвращает квадрат X (X – целое/вещественное, результат – как у аргумента);

SIN(X), **COS(X)**, **ArcTan(X)** – возвращают значения синуса, косинуса и арктангенса (X – целое/вещественное, результат – *Extended*);
SQRT(X) – возвращает квадратный корень из X (X > 0 – целое/вещественное, результат – *Extended*);
EXP(X), **LN(X)** – возвращает экспоненту или натуральный логарифм X (X – целое/вещественное, результат – *Extended*);
FRAC(X) – дробная часть числа X (X – целое/вещественное, результат – *Extended*);

INT(X) – целая часть числа X (X – целое/вещественное, результат – *Extended*);
TRUNC(X) – целая часть числа X (X – целое/вещественное, результат – *Int64*);
ROUND(X) – округление X до ближайшего целого (X – целое/вещественное, результат – *Int64*);

СТАНДАРТНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ И ПРОЦЕДУРЫ

RANDOM(X) – возвращает случайное целое число из диапазона от 0 до X (X и результат – *Integer*);
RANDOM – возвращает случайное число от 0 до 1 (результат – *Extended*);
RANDOMIZE – процедура гарантирует несовпадение последовательностей случайных чисел, выдаваемых функцией Random;

ODD(X) – возвращает *True*, если X – нечетное число, и *False*, если X – четное (X – целое, результат – *Boolean*);

INC(Var X : <целое>), **DEC(Var X : <целое>)** – увеличивает или уменьшает значение X на 1 (процедуры);
INC(Var X : <целое>; N : <целое>), **DEC(Var X : <целое>; N : <целое>)** – увеличивает или уменьшает значение X на N (процедуры);

```
Var
   X : Integer;
Begin
   X := 2;
   INC(X,4)          {X → 6}
End.
```

МОДУЛИ

Модуль – это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры, функции) и, возможно, некоторые исполняемые операторы, которая предназначена для использования другими модулями и программами.

```
Unit <имя модуля>;
        Interface
           <секция интерфейса>
        Implementation
           <секция реализации>
        Initialization
           <секция инициализации>
        Finalization
           <секция завершения>
        End.
```

Имя модуля (например, Unit_1) должно совпадать с именем дискового физического файла, в который помещается исходный текст модуля (unit_1.pas). Для подключения ресурсов модуля к другим программным единицам (программа, модуль) необходимо указать его имя в спецификации Uses раздела описаний этой программной единицы (Uses Unit_1;).

```
Unit Unit_1;
Interface
   Type
      Complex = record
                Re, Im : Real
                end;
   Var
      Zmain : Complex;
   Procedure P_1 (X,Y : Complex; Var Z : Complex);
Implementation
   Procedure P_1;
      begin
         Z.Re := X.Re + Y.Re;
         Z.Im := X.Im + Y.Im
      end;
Initialization
   Zmain.Re := 1; Zmain.Im := -1
End.
```

СЕКЦИИ МОДУЛЯ

В **интерфейсной секции** содержатся объявления всех глобальных элементов модуля (типов, констант, переменных, функций, процедур), которые должны стать доступными основной программе и другим модулям, к которым подключен данный модуль. При объявлении глобальных подпрограмм указывается только их заголовок. Если при объявлении типов, данных и подпрограмм используются элементы, введенные в других модулях, то они должны быть указаны в разделе Uses сразу после слова Interface.

Секция реализации содержит описание подпрограмм, объявленных в интерфейсной части, и описание внутренних ресурсов модуля (локальных переменных, типов, подпрограмм). Обращение к этим ресурсам возможно только из подпрограмм, описанных в этом же модуле.

Секция инициализации (может отсутствовать) включает программные действия, которые будут произведены перед выполнением основной программы, к которой подключен данный модуль. Обычно в разделе инициализации происходит заполнение стартовыми значениями библиотечных переменных, а также одноразовые действия, которые должны выполниться в начале программы (например, открываться нужные файлы и т.д.).

Секция завершения (может отсутствовать) содержит операторы, которые будут выполняться при завершении работы приложения (основной программы).

СТАНДАРТНЫЕ МОДУЛИ DELPHI PASCAL

- **Системные модули:**

- **SYSTEM** – включает все стандартные математические процедуры и функции, обеспечивает работу с файлами, с динамической памятью, с другими модулями и т.д.; подключается автоматически к каждой программе.
- **SYSUTILS** – содержит дополнительные процедуры и функции для работы с файлами, дисками, строками, по обработке исключительных ситуаций и др.
- **MATH** – содержит множество дополнительных математических функций и процедур.

- **Модули визуальных компонентов** (VCL – Visual Component Library) используются для визуальной разработки полнофункциональных GUI-приложений - приложений с графическим пользовательским интерфейсом (Graphical User Interface).