Подпрограммы-процедуры

Подпрограммы-функции могут иметь несколько аргументов, но возвращают в вызвавшую её основную программу или другую подпрограмму единственный результат. Это не всегда удобно. Иногда результатов требуется тоже несколько, а иногда не нужен ни один. (Например, если подпрограмма должна просто изменить значения некоторых глобальных переменных или вывести нечто на экран.) В таких случаях вместо подпрограмм-функций используют подпрограммы-процедуры. Процедуры имеют следующий формат описания

```
Procedure Имя_процедуры(список_формальных_параметров);
  Описание_локальных_переменных;
Begin
  операторы
End;
```

В число формальных параметров могут входить как входные параметры — аргументы, так и выходные — результаты. Некоторые параметры могут одновременно использоваться и как входные, и как выходные.

Для вызова процедуры просто указывается её имя со списком фактических параметров в скобках. Например, так мы вызывали Read или Write.

Имя процедуры является идентификатором. Список формальных параметров, как и у функций, — это последовательность разделённых точками с запятыми описаний формальных параметров. Например,

```
Procedure MyProc(a,b : Integer; x : Real; flag : Boolean);
```

или

```
Procedure MyProc2;
```

если и входные, и выходные параметры процедуры отсутствуют.

Классификация параметров

Каждый параметр классифицируется как значение, переменная, константа или выходной.

Параметры-значения

Для параметров-значений всё так же, как описывалось в предыдущем занятии для аргументов функций. Все использованные нами параметры, как при рассмотрении функций, так и в процедуре MyProc — параметры-значения. При вызове процедуры на место формальных параметров-значений будут *скопированы* фактические, и какие бы с ними в вызванной процедуре ни произошли изменения, вызвавшая подпрограмма об этом никогда не узнает. Эти параметры используют в качестве входных. Если размер параметра велик, например, параметром является массив большого размера, использование параметра-значения может оказаться не только нецелесообразным, но и невозможным. В качестве фактического параметра может быть использовано выражение того же типа, что и формальный параметр.

Параметры-переменные

Однако есть и другой способ — передача по адресу. Он применяется, например, для параметров-переменных. Если при описании формальных параметров перед их именами написать модификатор **Var**, то в подпрограмму будет передана *не копия* фактического *параметра*, *а копия* его *адреса* в оперативной памяти. А это значит, что и вызванная подпрограмма, и вызвавшая работают с одной и той же областью памяти. Таким образом, если процедура изменит значение формального параметра, переданного по адресу, фактический параметр будет иметь то же значение, что и формальный. При помощи параметра-переменной можно и передавать значение в процедуру, и получать результат на выходе. Переменные файлового типа передаются в подпрограмму *только* как параметры-переменные.

Пример применения параметров-переменных.

```
Program Sample1;

Procedure Arithmetic(a,b : Integer; Var sum, diff : Integer);
Begin
  Sum:=a+b;
  Diff:=a-b
End;

Var m,n,s,r : Integer;
Begin
  WriteLn('Введите 2 целых числа');
  Read(m,n);
  Arithmetic(m,n,s,r);
  WriteLn('Их сумма ',s);
  WriteLn('Разность ',r)
End.
```

В этом примере первые 2 параметра процедуры являются параметрами-значениями. В процедуру *Arithmetic* в параметр *a* будет скопировано значение переменной *m*, в параметр *b* — значение *n*. Адресом переменной *sum* станет адрес переменной *s*, а адресом переменной *diff* — адрес переменной *r*. Поэтому во время работы процедуры значение *s* будет совпадать со значением *sum*, а значение *r* — со значением *diff*. После окончания работы процедуры значения параметров-переменных сохранятся в *s* и *r*.

Выходные параметры

Если параметр не должен передавать значение в процедуру, а используется только как выходной, при его описании можно вместо **Var** в списке формальных параметров написать **out**. Эффект будет почти тот же. Так можно было поступить в приведённом выше примере Sample1.

```
Procedure Arithmetic(a,b : Integer; out sum, diff : Integer);
```

Параметры-константы

Если параметр не будет изменяться в процедуре, его можно описать как параметр-константу, используя модификатор **const**. При этом если размер параметра велик, то он будет передан в подпрограмму по адресу, что сэкономит память (не нужна будет копия) и время (на копирование). Присвоение значения параметру, переданному как константа, запрещено. Можно, конечно, было описать такой параметр как параметр-переменную, но использование **const** вместо **var** гарантирует, что в подпрограмме переменная никак изменена не будет.

Всё, что выше говорилось о параметрах процедур, справедливо и для параметров функций. Однако в случае функций лучше использовать параметры-значения и константы. А результат у функций желательно чтобы был всё-таки один и возвращался стандартным для функций способом.

Задача 1

Даны 3 матрицы[1] одинакового размера *m* строк на *n* столбцов. Вывести ту из них, которая содержит максимальный элемент из всех, встречающихся в трёх матрицах. Первая строка входного файла input.txt содержит числа *m* и *n*. Далее следуют элементы матриц: *m* строк по *n* элементов для каждой из них.

Решение

[1] Числовой матрицей в математике называется прямоугольная таблица чисел.

Один из возможных вариантов решения представлен ниже. Поочерёдно вводятся матрицы и находятся их максимальные элементы, среди которых выбирается наибольший. Потом найденная матрица выводится.

```
{$mode delphi}
Program Sample2;
Const MAXN=1000;

Type TMatrix = array[1..MAXN,1..MAXN] of Integer;

//Ввод из файла f матрицы из m строк и n столбцов
Procedure Input(m,n : Integer; Var a : TMatrix; Var f : Text);
Var i,j : Integer;
Begin
  For i:=1 To m Do
  For j:=1 To n Do
  Read(f,a[i,j])
End;

//Вывод матрицы a размером M x N в файл f
Procedure Output(m,n : Integer; const a : TMatrix; Var f : Text);
Var i,j : Integer;
Begin
  For i:=1 To m Do
  Begin
    For j:=1 To n Do
     Write(f,a[i,j]:7,' ');
    WriteLn(f) //переход к новой строке
  End
End;

//Нахождение наибольшего элемента матрицы
Function MaxElem(m,n : Integer; const a : TMatrix) : Integer;
Var i,j,max : Integer;
Begin
  max:=a[1,1];
  For i:=1 To m Do
  For j:=1 To n Do
   If a[i,j] > max Then
   max:=a[i,j];
  MaxElem:=max
End;

//Основная программа
Var a       : array[1..3] of TMatrix;
    m,n     : Integer;
    f       : Text;
    i,k,max : Integer;
    curr    : Integer;
Begin
  Assign(f,'input.txt');
  Reset(f);
  Read(f,m,n);

  max:=-MaxInt-1;//Это значение вряд ли будет максимальным элементом
  k:=1; //Номер матрицы с максимальным значением
  For i:=1 To 3 Do
  Begin
    Input(m,n,a[i],f);
    curr:=MaxElem(m,n,a[i]);
    If curr > max Then
    Begin
      max:=curr;
      k:=i
```

```
      End
    End;
    Close(f);

    Assign(f,'output.txt');
    Rewrite(f);
    Output(m,n,a[k],f);
    Close(f)
  End.
```

Эта программа просто иллюстрирует применение процедур и функций. Матрица a в процедуру ввода Input передаётся как параметр-переменная, чтобы при выходе из процедуры сохранить введённые значения. Можно было бы расширить процедуру ввода с тем, чтобы она сразу при вводе искала максимальный и минимальный элементы матрицы. Тогда функция MaxElem была бы не нужна, следовательно, не нужен был бы и дополнительный проход по матрице. Программа работала бы быстрее. Реализуйте этот подход сами.

Обратите внимание, что матрица передавалась в функцию как параметр-константа. Это избавляет от необходимости копирования массива из 1000000 элементов (MAXN * MAXN) в эту функцию. То же относится и к процедуре вывода Output.

Все подпрограммы в этом примере независимы друг от друга, все они вызывались из основной программы. Поэтому их можно было располагать друг за другом в произвольном порядке. Главное, чтобы они предшествовали основной программе. В следующем примере показан вызов одной подпрограммы из другой.

Задача 2

Дано натуральное число $a \leq 10^{18}$. Определить, является ли оно палиндромом. (Палиндром — это число или слово, читающееся одинаково слева направо и справа налево. Например, числа 12321, 1991, слово «казак».)

Решение

Можно предложить следующий алгоритм решения этой задачи.

1. Найти реверсную запись числа a.
2. Если эта запись совпадает с a, то перед нами палиндром, в противном случае не палиндром.

А как найти реверсную запись числа x?

1. Взять число r, равное нулю.
2. Пока в записи числа x ещё есть цифры
   a. «Отрезать» от числа x последнюю цифру c
   b. «Приклеить» цифру c в конец записи числа r
3. Полученное r и есть реверсная запись числа x

А как «отрезать» последнюю цифру c от числа x?

1. $c \coloneqq x \ mod\ 10$
2. $x \coloneqq x \ div\ 10$

А как «приклеить» цифру c в конец записи числа x?

1. $x \coloneqq x * 10 + c$

Итак, нам удалось разбить исходную задачу на подзадачи. Для решения каждой из них составлен вспомогательный алгоритм. Два последних из них очень короткие и, наверное, следовало бы объединить их с алгоритмом нахождения реверсной записи числа. Но мы этого делать не будем, и реализуем каждый из них в виде подпрограммы.

```
{$mode delphi}
Program Palindrom;
Type Digit = 0..9;

(**********************************************
 *"Отрезание" последней цифры от числа x      *
 * Вход:  x - исходное число.                 *
 * Выход: x - число без последней цифры,      *
 *        last - последняя цифра исходного числа *
 **********************************************)
Procedure Cut(Var x : Int64; Var last : Digit);
Begin
  last := x mod 10;
  x := x div 10
End;

(**********************************************
 * "Приклеить" цифру с в конец записи числа x *
 **********************************************)
Procedure Paste(Var x : Int64; c : Digit);
Begin
  x := x * 10 + c
End;

(**********************************************
 * Получение числа, цифры которого следуют в   *
 * обратном порядке относительно цифр числа x. *
 * Например, 12345 --> 54321                   *
 **********************************************)
Function Revers(x : Int64) : Int64;
Var  r : Int64;
     d : Digit;
Begin
  r:=0;  //Новое число - будущая реверсная запись числа x
  While x > 0 Do
  Begin
    Cut(x,d);   //"Отрезать" цифру от исходного числа
    Paste(r,d)  //"Прицепить" её к новому
  End;
  Revers:=r
End;

(********* Основная программа *********)

Var a : Int64;
Begin
  WriteLn('Введите число');
  Read(a);
  If a=Revers(a) Then
    WriteLn('Палиндром')
  Else
    WriteLn('Не палиндром')
End.
```

Обратите внимание, что процедуры Cut и Paste вызываются не из основной программы, а из функции Revers. Это накладывает требование на порядок следования подпрограмм в тексте программы. К моменту своего первого вызова подпрограмма должна быть описана. Сначала в тексте программы идут процедуры Cut и Paste, а уже потом[2] подпрограмма, которая их использует — функция Revers.

[2] Этот порядок можно изменить, используя опережающее описание. Однако здесь рассматривать опережающее описание мы не будем.

Обсудим теперь передачу параметров в подпрограммы. В процедуре Cut параметр x является одновременно и входным, и выходным. Отсюда следует, что он должен быть параметром-переменной. Параметр *last* является только выходным. Никакой информации в подпрограмму при помощи него не передаётся. Поэтому его можно было описать либо при помощи модификатора **Var**, либо при помощи модификатора **out**.

В процедуре Cut параметр x тоже является и входным, и выходным. Поэтому он описан как параметр-переменная. А вот входной параметр c описан как параметр-значение. Описывать его как параметр-переменную не стоит. В случае больших подпрограмм вы только запутаетесь, гадая, изменялся ли этот параметр в подпрограмме или нет.

Задачи для самостоятельного решения

1. Составить программу, определяющую, в каком из двух данных натуральных чисел больше цифр. Каждое из чисел не превышает 10^9. В записи чисел лидирующие нули отсутствуют. Для определения количества цифр числа используйте
   a) функцию;
   b) процедуру.
2. Какое наибольшее число можно составить из цифр заданного натурального числа N? Каждую цифру *нужно* использовать столько раз, сколько раз она входит в запись числа N. $N \leq 10^{18}$. (*Указание*. Разложите N на цифры, а потом отсортируйте их. Используйте подпрограммы.)