Понятие подпрограммы. Подпрограммы-функции

Зачем нужны подпрограммы

С увеличением объёма программы становится невозможным удержать в памяти все детали. Естественным способом борьбы со сложностью любой задачи является её разбиение на части, т.е. на подзадачи. Для решения каждой из таких подзадач составляют вспомогательный алгоритм. Вспомогательный алгоритм, записанный на языке программирования, называют подпрограммой. Саму программу для решения исходной задачи можно рассматривать в более укрупнённом виде — в виде взаимодействующих подпрограмм. Это важно, так как человек способен помнить ограниченное число фактов. Использование подпрограмм ведёт к упрощению структуры программы. Процесс отладки[1] программы, использующей подпрограммы, тоже упрощается.

Разделение программы на подпрограммы позволяет также избежать избыточного кода (уменьшить размер программы), поскольку подпрограмму записывают один раз, а вызывать её на выполнение можно столько раз, сколько это будет нужно, из разных точек программы. Более того, одни и те же вспомогательные задачи могут возникать при решении разных задач. Это значит, что стоит написать подпрограмму один раз, а потом её можно использовать уже в готовом виде при создании различных программ. Чтобы избежать копирования текста подпрограмм из одной программы в другую, можно оформить подпрограммы в виде отдельных файлов, а потом просто подключать их к основной программе по мере надобности.

Дальнейшим развитием этой идеи является создание библиотек часто используемых подпрограмм. Любая система программирования на языках высокого уровня, в том числе и Паскаль, поставляется с набором стандартных библиотек, содержащих большое количество подпрограмм. Например, подпрограммами являются постоянно применявшиеся нами для ввода и вывода команды Read и Write, а также команды вычисления модуля числа abs, квадратного корня sqrt, длины строки length, кода символа Ord или символа по коду Chr, работы с файлами Assign, Reset, Rewrite, Close и многие другие. Кроме того, программист сам может создавать дополнительные библиотеки из своих собственных подпрограмм.

Подпрограммы-функции

В языке Паскаль определены два вида подпрограмм: процедуры и функции. Они различаются количеством выходных данных, а также способом обращения к ним. Мы начнём изучение подпрограмм с функций.

Подпрограмма-функция аналогична математической функции. И та, и другая могут иметь один или более аргументов, но значением функции (её результатом) является *единственная* величина. В языке Паскаль допускается также использование функций, не имеющих ни одного аргумента. Но их результат тоже только один.

Описание функций

Пока что будем предполагать, что наша программа состоит из единственного файла.

При использовании подпрограмм их код помещается в файл перед кодом основной программы. Код подпрограммы-функции имеет следующую структуру.

```
Function Имя_функции(арг1 : тип1; арг2 : тип2;…; аргN : типN) : тип_результата;
Var … //Описание локальных переменных, если они есть. Иначе эта строка отстутствует.
Begin
      //Реализация вспомогательного алгоритма
      …
```

[1] Отладка программы — это поиск и исправление ошибок в ней.

```
    Имя_функции := результат
End;
```

Имя функции и имена аргументов арг1, арг2,…, аргN являются идентификаторами[2]. Количество аргументов определяет сам программист. Их называют формальными параметрами. Имена этих аргументов действуют только в пределах кода данной подпрограммы. В вызывающей программе имена соответствующих параметров могут быть иными. Важно только, чтобы соответствовал их тип. Покажем это на примере.

Пример 1.

Треугольник ABC задан координатами своих вершин. Определить его периметр. Координаты вершин — вещественные числа. Случай, когда точки A, B и C лежат на одной прямой тоже считать треугольником (вырожденным).

Алгоритм решения этой задачи очевиден: найти длины сторон AB, AC и BC, а потом их сложить.

Определение расстояния между двумя точками плоскости по их координатам выделим в подзадачу Dist с входными параметрами x1, y1, x2, y2 — декартовыми прямоугольными координатами этих точек. Результатом будет одно число — искомое расстояние. Значит, эту подзадачу можно решить с использованием функции

Function Dist(x1, y1, x2, y2 : real) : real;

Реализующей вычисления по известной формуле $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Эту функцию предстоит нам создать. Вот программа решения поставленной задачи.

```
{$mode delphi}
Program Perim;

// Функция вычисления расстояния между 2 точками
Function Dist(x1,y1,x2,y2 : real) : real;
Var dx, dy : real;
Begin
  dx:=x2-x1;
  dy:=y2-y1;
  Dist:=sqrt(dx*dx+dy*dy)
End;

//Основная программа
Var xa,ya,xb,yb,xc,yc : real;
    ab, ac, bc, p     : real;
Begin
  WriteLn('Введите координаты 3 точек: x y');
  Read(xa,ya,xb,yb,xc,yc);
  ab := Dist(xa,ya,xb,yb);      //Вызовы подпрограмм-функций
  ac := Dist(xa,ya,xc,yc);
  bc := Dist(xb,yb,xc,yc);
  p := ab + bc + ac;
  WriteLn('Периметр треугольника равен ',p:7:3)
End.
```

Давайте в этой программе немного разберёмся. Обратите внимание, что блок описания переменных Var появляется в этой программе дважды: в функции и в основной программе. Это не случайно. Каждая подпрограмма может иметь собственный набор переменных. Их называют локальными. Они действуют только в пределах той подпрограммы, в которой они описаны, и существуют только в период выполнения этой подпрограммы. Причём совершенно неважно, есть ли ещё где-нибудь в программе переменные с такими же именами. Если таковые имеются, то это будут *другие* переменные. Для их хранения выделены *другие* участки памяти. Изменение значения

[2] Идентификатор — последовательность букв латинского алфавита и цифр, начинающаяся с буквы.

локальных переменных никак не отразится на значении их «однофамильцев». То же касается и формальных параметров[3] x1, y1, x2, y2. Это позволяет выбирать имена локальных переменных и формальных параметров подпрограмм, не оглядываясь на остальные подпрограммы или основную программу. Формальные параметры можно рассматривать как продолжение списка локальных переменных и использовать во всех выражениях внутри тела подпрограммы.

Теперь давайте обратим внимание на последнюю строку функции

```
Dist:=sqrt(dx*dx+dy*dy)
```

Значение, присвоенной имени функции и будет результатом её работы. Если эту строку забыть написать, результат будет утерян. В Delphi[4] вместо присваивания значения имени функции допустимо присваивание значения специальной переменной Result. (Эту переменную не надо нигде описывать.) Т.е. только что рассмотренную строку можно заменить следующей

```
Result:=sqrt(dx*dx+dy*dy)
```

В отличие от системы программирования TurboPascal, в Delphi и FreePascal функция может возвращать значения любых предварительно объявленных типов, в том числе и массивов. Однако тип массива должен быть объявлен в разделе объявления типов.

Вызов подпрограммы-функции

Теперь рассмотрим вызов подпрограммы-функции. Чаще всего это делается так

```
Имя_переменной := имя_функции(список_фактических_параметров)
```

Список фактических параметров разделён запятыми. В нашем примере один из вызовов такой:

```
ab := Dist(xa,ya,xb,yb)
```

Здесь фактическими параметрами являются xa, ya, xb, yb. Их значения копируются в формальные параметры подпрограммы: первый фактический в первый формальный, второй фактический во второй формальный и т.д. В нашем случае xa скопируется в x1, ya в y1, xb в x2, yb в y2. При совпадении имён фактического и формального параметров копирование произойдёт всё равно. Вспомните, что это всё же разные переменные, расположенные в разных участках памяти, «однофамильцы». При создании подпрограммы надо позаботиться о том, чтобы количество и типы её формальных и фактических параметров соответствовали друг другу. Нельзя, например, чтобы тип формального параметра x1 был integer, а фактического параметра xa — real: real нельзя скопировать в integer. В приведённом примере все эти типы — real. Поскольку скопировать (присвоить) переменную типа integer в переменную типа real можно, то допустима ситуация, когда тип фактического параметра (xa) integer, а тип формального параметра (x1) real. Если формальным параметром является массив, то его тип должен быть определён в разделе описания типов Type.

После окончания работы функции значение переменной (в нашем случае ab) станет равным значению, вычисленному, или, говорят, возвращённому, функцией. Значения же всех локальных переменных и формальных параметров функции станет неопределённым, так как память для их хранения выделяется временно, на период работы этой функции.

Вызов функции можно также использовать в выражениях там, где допустимо использование значения того типа, который возвращает функция. Например, для функции

```
Function f(x : Integer) : Integer
```

это могут быть следующие выражения:

```
z:=f(x)+f(y);
If f(p) > k Then …
```

---

[3] По крайней мере, описанных так, как говорится на этом занятии. О нюансах поговорим при изучении процедур

[4] И в FreePascal в режиме совместимости с Delphi

В качестве фактического параметра при вызове функции можно использовать не только переменную, но и константу или выражение, тип которых совпадает с типом требуемого формального параметра. Например,

```
z:=f(x+5);
m:=f(100) div f(5);
m:=f(x+f(10))
```

Здесь имеется в виду, что переменные *x*, *m*, *z* имеют тип integer.

Пример 2.

В следующем примере создаётся и выводится на экран массив случайных целых чисел.

```
{$mode delphi}
Program TestFunc;
Const MAXN = 1000; //Предельный размер массива
      LINE = 7;    //Количество элементов в строке при выводе на экран

Type massiv = array[1..MAXN] of Integer;

Var N : Integer; //Количество элементов массива

//Функция создания массива случайных целых чисел
Function RandArray(max : Integer) : massiv;
Var i : Integer;
    a : massiv;
Begin
  Randomize; //Начать со случайного числа
  For i:=1 To N Do
   a[i]:=Random(max); //Заполняем массив случайными числами, меньшими max
  RandArray := a
End;

//Основная программа
Var x    : massiv;
    m, i : Integer;
Begin
  //Ввод данных
  Write('Количество элементов массива ');
  Read(N);
  If N > MAXN Then
  Begin
    WriteLn('Ошибка. Количество элементов должно быть не больше ',MAXN);
    Halt
  End;
  Write('Верхняя граница элементов массива ');
  Read(m);

  x:=RandArray(m);  //Вызов функции

  //Вывод массива на экран по LINE элементов в строке
  For i:=1 To N Do
  Begin
   Write(x[i]:10,' ');
   If i mod LINE = 0 Then
     WriteLn
  End;
  WriteLn
End.
```

Для создания массива в этой программе применяется функция RandArray. Хотя для этих целей обычно применяют процедуры, здесь показано, что в Dephli и FreePascal функция может

возвращать даже массив. Функция RandArray имеет один формальный параметр *max* типа Integer. При вызове этой функции в него копируется значение фактического параметра *m* тоже типа Integer. Обратим внимание на переменную *N*, описанную в самом начале программы. Мы её используем для хранения количества элементов массива. Она описана вне всяких подпрограмм (и вне основной программы). Такие переменные являются глобальными. Они существуют в оперативной памяти всё время выполнения программы и доступны из любой её части: из любой подпрограммы и из основной программы. Однако, описана эта переменная должна быть до первого её использования. Сразу обратим внимание на то, что объявлять глобальную переменную следует, только если на то есть действительно веские причины. Глобальных переменных не должно быть много. Иначе легко запутаться и в их именах, и в том, в каких местах программы значения этих переменных могут изменяться.

В программе может использоваться и несколько подпрограмм.

Пример 3.

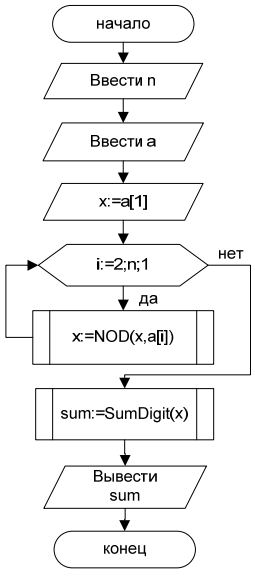
Дан массив $\boldsymbol{a}$ *из* $N \leq 1000$ *натуральных чисел, каждое из которых не превышает* 10^{18}. *Найти сумму цифр наибольшего общего делителя всех этих чисел.*

Для решения этой задачи выделим 2 подзадачи: нахождения наибольшего общего делителя двух натуральных чисел и вычисления суммы цифр натурального числа. Обе они могут быть решены при помощи подпрограмм-функций, т.к. имеют только по одному результату. Пусть эти функции

```
Function NOD(a,b : int64) : int64;
Function SumDigit(n : int64) : Byte;
```

Ясно, что сумма цифр числа типа int64 меньше 255: количество цифр этого числа не превышает 19, каждая цифра не больше 9. Поэтому функция вычисления суммы цифр числа возвращает значение типа byte.

Чтобы найти НОД нескольких чисел, поступим следующим образом. Сначала найдём НОД первых двух чисел, потом результата и третьего числа, потом нового результата и четвёртого числа и т.д. Основной алгоритм получается следующим.



Прямоугольники с двойными боковыми стенками на блок-схеме обозначают обращение к подпрограммам. Алгоритмы для решения подзадач нахождения НОД и суммы цифр числа рассматривались нами ранее, на занятиях 4 и 5. Воспользуемся ими.

Теперь пишем программу.

```
{$mode delphi}
Program TwoFunc;
```

```
Const MAXN = 1000;
Type massiv = array[1..MAXN] of int64;

//Вычисление наибольшего общего делителя чисел a и b
Function NOD(a,b : int64) : int64;
Begin
  While (a>0) and (b>0) Do
  If a>b Then
    a:=a mod b
  Else
    b:=b mod a;
  NOD := a + b
End;

//Вычисление суммы цифр числа
Function SumDigit(n : int64) : Byte;
Var sum : Byte;
Begin
  sum := 0;
  While n > 0 Do
  Begin
    sum:=sum+N mod 10;
    N:=N div 10
  End;
  SumDigit:=sum
End;

//Основная программа
Var n : Integer; //Количество элементов массива
    a : massiv;
    i : Integer;
    x : int64;  //Наибольший общий делитель
    sum : Byte;
Begin
  Write('Количество элементов массива ');
  Read(n);
  WriteLn('Введите ',n,' натуральных чисел');
  For i:=1 To n Do
  Read(a[i]);

  x:=a[1];
  For i:=2 To n Do
    x:=NOD(x,a[i]);

  sum:=SumDigit(x);

  WriteLn(sum)
End.
```

В этой программе нет ни одной глобальной переменной. Они просто не нужны. Идентификатор *a* в функции NOD обозначает числовую переменную, а в основной программе — целый массив. Переменная *n* в функции SumDigit имеет тип int64, а в основной программе — integer. Несмотря на совпадение имён, эти переменные по сути разные, поэтому никакого противоречия здесь нет. В функции SumDigit значение формального параметра *n* изменяется, но значение фактического параметра *x* при вызове `sum:=SumDigit(x)` остаётся неизменным. Ведь при вызове функции значение фактического параметра *x* копируется в формальный параметр *n* и изменениям подвергается именно эта копия.

Задачи для самостоятельного решения

1. Напишите программу сложения и умножения двух простых дробей $\frac{a}{b}+\frac{c}{d}$ и $\frac{a}{b}\cdot\frac{c}{d}$, где *a*, *c* — целые, а *b*, *d* — натуральные числа. Ответ должен быть несократимой простой дробью.

2. Напишите программу вычисления значения выражения $\left(\frac{a^m b^n}{c^k}\right)^p$, где a, b, c — вещественные, а m, n, k, p — натуральные числа, не превосходящие $4 \cdot 10^9$.
3. Найдите натуральное число из отрезка $[a, b]$, имеющее наибольшее количество делителей. Если таких чисел несколько, выведите наименьшее из них. (a, b — целые числа, модуль которых не превышает 10^6.)