**Подпрограммы (процедуры и функции)**

При решении новых задач можно попытаться воспользоваться ранее написанными программами. Алгоритм, ранее разработанный и целиком используемый в составе других алгоритмов, называется вспомогательным. Применение вспомогательных алгоритмов позволяет разбить задачу на части, структурировать ее.

Вся программа условно может быть разделена на две части: основную и вспомогательную. В основной части производится простейшая обработка информации, организуется обращение к разным вспомогательным модулям (подпрограммам).

Вспомогательный алгоритм тоже может вызывать другие вспомогательные, длина такой цепочки вызовов теоретически не ограничена. Здесь и далее следующие пары слов используются как синонимы: алгоритм и программа, вспомогательный алгоритм и подпрограмма, команда и оператор, программа и модуль. Вспомогательными и основными алгоритмы являются не сами по себе, а по отношению друг к другу.

При использовании вспомогательных алгоритмов необходимо учитывать способ передачи значений исходных данных для них и получения результата от них. Аргументы вспомогательного алгоритма — это переменные, в которых должны быть помещены исходные данные для решения соответствующей подзадачи. Результаты вспомогательного алгоритма — это также переменные, где содержаться результаты решения этих подзадач, а также результатом может быть конкретное действие, которое совершает компьютер под действием подпрограммы.

Подпрограммы могут быть двух видов: подпрограмма без параметров и подпрограмма с параметрами. Обращение к подпрограмме может быть организовано из любого места основной программы или другой подпрограммы сколько угодно раз.

При работе с подпрограммами важными являются понятия формальных и фактических параметров. Формальные параметры — это идентификаторы входных данных для подпрограммы. Если формальные параметры получают конкретные значения, то они называются фактическими. Формальные параметры могут получить конкретные значения только в той программе, где производится обращение к данному модулю-подпрограмме. Тип и порядок записи фактических параметров должны быть такими же, как и формальных параметров. В противном случае результат работы программы будет непредсказуемым. Из этого следует, что фактические параметры используются при обращении к подпрограмме из основной, а формальные параметры — только в самом модуле.

Подпрограмма с параметрами используется для записи многократно повторяющихся действий при разных исходных данных. Подпрограммы с параметрами можно разделить на два типа: подпрограммы-функции и просто подпрограммы с параметрами (их называют процедурами).

При составлении подпрограмм с параметрами надо соблюдать следующие правила:

1) каждая подпрограмма имеет свое имя и список формальных параметров;

2) процедура из основной программы вызывается командой вызова, которая по форме ничем не отличается от вызова команды исполнителя. Результат присваивается одной или нескольким переменным, которые находятся в списке формальных параметров. Но результатом могут быть, конечно, не только значения переменных, но какое либо действие, выполненное ЭВМ.

Пример 1. Используем алгоритм нахождения наибольшего общего делителя двух натуральных чисел в качестве вспомогательного при решении задачи: составить программу вычитания дробей (a, b, c, d — натуральные числа). Результат представить в виде обыкновенной несократимой дроби.

Подпрограмма.

1) Ввести натуральные числа M, N.

2) Если M=N, перейти к п. 5, иначе к следующему пункту.

3) Если M>N, то M:=M-N, иначе N:=N-M.

4) Перейти к п. 2.

5) Передать значение M в основную программу.

6) Конец подпрограммы.

Основная программа.

1) Ввести значения A, B, C, D.

2) E:=A\*D - B\*C.

3) F:= B\*D.

4) Если E=0, вывести значение E и перейти к п. 9, иначе перейти к следующему пункту.

5) M:=|E|, N:=F, перейти к подпрограмме вычисления НОД.

6) G := M.

7) E и F нацело разделить на G.

8) Вывести значения E и F на печать.

9) Конец программы.

Program Sub;

Var A, B, C, D, G, E, F : Integer;

Procedure Nod(M, N : Integer; Var K : Integer);

Begin

While M <> N Do

If M > N Then M := M - N Else N := N - M;

K := M

End;

Begin

Write('Введите числители и знаменатели дробей:');

ReadLn(A, B, C, D);

E := A \* D - B \* C;

F := B \* D;

If E = 0 Then WriteLn(E)

Else

Begin

Nod(Abs(E), F, G);

E := E Div G;

F := F Div G;

WriteLn('Ответ: ', E, '/', F)

End

End.

Как видно из примера, объявление и тело подпрограмм находится в разделе описаний. В заголовке подпрограммы содержится список формальных параметров с указанием их типа, которые условно можно разделить на входные и выходные (перед ними стоит служебное Var). При обращении к процедуре указывается ее имя и список фактических параметров. Формальные и фактические параметры должны соответствовать по количеству и по типу.

Вызов процедуры осуществляется следующим образом:

<Идентификатор (имя) процедуры>(<список фактических параметров>);

Например,

Nod(Abs(E), F, G);

По способу передачи фактических значений в подпрограмму в Turbo Pascal 7.0 выделяют параметры-переменные, параметры-значения, параметры-константы и массивы открытого типа, строки открытого типа, параметры-процедуры, параметры-функции (подробности — в литературе).

Функция (в отличие от процедуры) всегда возвращает единственное значение.

Покажем, как изменится подпрограмма из примера, если ее записать в виде функции.

Function Nod(M, N : Integer) : Integer;

Begin

While M <> N Do

If M > N Then M := M - N Else N := N - M;

Nod := M

End;

Итак, после списка параметров указывается тип значения функции, а в теле функции хотя бы один раз встречается присваивание переменной, имя которой совпадает с именем функции, соотответствующего значения.

Вызов функции будет следующим:

G := Nod(Abs(E), F);

Вообще, вызов функции может присутствовать в выражении, стоящем: в правой части оператора присваивания, в процедуре вывода, в качестве фактического параметра в вызове другой подпрограммы и т.д.

При решении задач целесообразно проанализировать условие, записать решение в крупных блоках (не являющихся операторами Pascal), детализировать каждый из блоков (записав в виде блоков, возможно, по-прежнему не операторов Pascal), и т.д., продолжать до тех пор, пока каждый из блоков не будет реализован с помощью операторов языка.

Пример 2. Дано натуральное число n. Переставить местами первую и последнюю цифры этого числа.

Program Integ;

Var N : Integer;

Begin

Write('Введите натуральное число: ');

ReadLn(N);

If Impossible(N)

Then WriteLn('Невозможно переставить цифры, возникнет переполнение')

Else Begin

Change(N);

WriteLn('Ответ: ', N)

End;

End.

Можно заметить, что необходимо детализировать логическую функцию Impossible, которая диагностирует, возможна ли перестановка, и процедуру Change, которая эту перестановку (в случае, если она возможна) выполняет.

Function Impossible(N : Integer) : Boolean;

Begin

If Number(N) < 5

Then Impossible := False

Else Impossible := (N Mod 10 > 3) Or

(N Mod 10 = 3) And

(N Mod 10000 Div 10 \* 10 + N Div 10000 > MaxInt Mod 10000)

End;

Здесь необходимо детализировать функцию Number, возвращающую количество цифр в записи натурального числа (т.к. функция Impossible содержит ее вызов, то в разделе описаний функция Number должна ей предшествовать).

Function Number(N : Integer) : Integer;

Var Vsp : Integer;

Begin

Vsp := 0;

While N > 0 Do

Begin

Vsp := Vsp + 1; N := N Div 10

End;

Number := Vsp

End;

Наконец, последняя процедура.

Procedure Change(Var N : Integer);

Var Kol, P, S, R : Integer;

Begin

Kol := Number(N);

P := N Mod 10; {последняя цифра}

If Kol > 1 Then

S := N Div Round(Exp((Kol - 1) \* Ln(10)))

Else S := 0; {первая цифра}

R := N Mod Round(Exp((Kol - 1) \* Ln(10))) Div 10;

N := P \* Round(Exp((Kol - 1) \* Ln(10))) + R \* 10 + S

End;

Возможны также подпрограммы, которые вызывают сами себя. Они называются рекурсивными. Создание таких подпрограмм является красивым приемом программирования, но не всегда целесообразно из-за чрезмерного расхода памяти ЭВМ.

Пример 3. Найти максимальную цифру в записи данного натурального числа.

Program MaxDigit;

Type NaturLong = 1..(High(LongInt));

Digit = 0..9;

Var A : LongInt;

Function Maximum(N : LongInt) : Digit;

Begin

If N < 10

Then Maximum := N

Else If N Mod 10 > Maximum(N Div 10)

Then Maximum := N mod 10

Else Maximum := Maximum(N Div 10)

End;

Begin

Write('Введите натуральное число: ');

ReadLn(A);

WriteLn('Максимальная цифра равна ', Maximum(A))

End.

При создании функции Maximum было использовано следующее соображение: если число состоит из одной цифры, то она является максимальной, иначе если последняя цифра не является максимальной, то ее следует искать среди других цифр числа. При написании рекурсивного алгоритма следует позаботиться о граничном условии, когда цепочка рекурсивных вызовов обрывается и начинается ее обратное «раскручивание». В нашем примере это условие N < 10.

Более подробно о рекурсии говорится в следующей статье.