

Подпрограммы; работа с файлами.

1. Подпрограммы

Подпрограмма представляет из себя набор команд, обособленный от остальной программы. Этот блок команд выполняется целиком при *вызове* подпрограммы, который осуществляется через её имя. После выполнения подпрограммы, программа продолжит выполняться с места вызова. Подпрограммы бывают двух типов – процедуры и функции.

```

1  program Project1;
.
.
.  var global_variable_1, a, b, p3: type1;
.    global_2, d, e: type2;
5    f: type3;
.
.  ... //
.
.  procedure sample(p1,p2,p3:type1; var p4,p5:type2; p6:type3);
.0 var local_variable_1: type4;
.    local_2, local_3: type5;
.  begin
.    команда1;
.    команда2;
.    ...
.    ...
.  end;
.
.  begin
.0    ...
.    sample(a,b,p3, d,e, f);
.2    ...
.  end.

```

Рис. 1: Синтаксис использования процедур.

Синтаксис написания процедуры и её вызов показаны на рис. 1. При написании процедуры после зарезервированного слова **procedure** указывается её имя и в скобках набор *формальных параметров*. Формальные параметры показывают в каком порядке и какого типа переменные процедура должна получить на вход при вызове, и по их именам можно работать с переданными переменными. То есть внутри подпрограммы её параметры являются локальными переменными, начальные значения которых указываются при вызове. Однако передаваться в процедуру они могут по-разному: параметры могут быть значениями или переменными.

Параметры-значения передаются в подпрограмму через стек в виде их копий. Поэтому любые изменения таких параметров в теле процедуры ни как не отражаются на значениях внешних переменных. Параметры-значения используются для передачи в подпрограмму исходных данных, используемых для расчета. Такие параметры при описании подпрограммы не предваряются никакими зарезервированными словами. Для каждого из них указывается соответствующий тип данных. В качестве такого параметра может передаваться не только переменная, но и константа или просто фиксированное число (если параметр численного типа), символ (если параметр символьный) и т.п. Про такие параметры говорят, что они передаются по значению.

Параметры-переменные указываются в описании подпрограммы после зарезервированного слова **var**, действие которого имеет силу до ближайшей точки с запятой. При вызове подпрограммы на месте параметра-переменной указывается именно переменная того же типа. При использовании параметра-переменной в подпрограмму передается сама переменная (её адрес), что позволяет вернуть в точку вызова

подпрограммы измененные значения внешних переменных, которые были указаны в качестве параметров. Иными словами, изменения значений параметров-переменных в теле подпрограммы приводит также к изменению значений внешних переменных, указанных в качестве параметров при вызове подпрограммы. Про параметры-переменные говорят, что они передаются по ссылке.

В данном примере параметры p1, p2, p3 и p6 процедура получает по значению, а параметры p4 и p5 — по ссылке.

После имени процедуры и формальных параметров можно объявить локальные переменные, которые будут видны только в этой процедуре (области видимости локальных и глобальных переменных уже обсуждались ранее). Дальше в блоке **begin** — **end**; идёт *тело процедуры* — то есть собственно список обособленных команд, которые к ней относятся. В теле процедуры могут использоваться глобальные переменные, локальные переменные, а так же формальные переменные. Так же, как глобальные переменные заменяются локальными с тем же именем в области их видимости, так и при использовании формальных параметров, совпадающих по имени с глобальными переменными, эти глобальные параметры станут не видны. То есть при использовании в теле процедуры переменной p3 в примере на рис. 1, будет использоваться параметр, а не глобальная переменная с тем же именем. Соответственно при изменении значения p3 глобальная переменная не изменится.

Для выполнения процедуры она должна быть вызвана командой, совпадающей с её именем. В команде вызова указываются *фактические параметры* — то есть параметры, значения которых будут подставлены в формальные параметры. При этом фактические параметры, передаваемые по ссылке изменятся при выполнении процедуры, а передаваемые по значению — нет.

```

1  program Project1;
.
.  var global_variable_1, a, b: type1;
.      global_2, c: type2;
5      f: type3; x,i:integer;
.
.  ... //
.
.  function sample(p1,p2:type1; var p4:type2): type3;
10 var local_variable_1: type4;
.    local_2, local_3: type5;
.  begin
.      команда1;
.      команда2;
15  ...
.      ...
.      result:=...; // либо sample:=...;
.  end;
.
20 begin
.      ...
.      f:=sample(a,b, c); // типы должны совпадать
.
.      f:=(2 + sample(a,b,c) ) / f; { Если type3 - численный
25      (если использовать деление - то с плавающей точкой),
.      то возвращаемое функцией значение может быть
.      использовано в выражении }
.
.      x:=ord( sample(a,b,c) ); // Если type3 - символьный.
30
.      if sample(a,b,c) then ... // Если type3 - boolean.
.
.      for i:=1 to sample(a,b,c) do ... // если type3 - integer
34
35  ...
.  end.

```

Рис. 2: Синтаксис использования функций.

Синтаксис написания функции и её вызов показаны на рис. 2. Функция отличается от процедуры тем, что в результате её выполнения должно быть получено некоторое значение, сохранённое в «переменную», имеющую одновременно два имени: **result** и имя функции. Говорят, что функция *возвращает* это значение при вызове. Это означает, что в отличие от процедуры, которая вызывалась сама по себе, вызов функции должен происходить внутри команды, которая будет использовать возвращаемое функцией значение. Примеры таких команд и показаны на рис. 2. Обратите внимание, что при объявлении функции в конце указывается какой-либо тип данных – его называют типом функции, и он определяет тип возвращаемого значения. Функция может использоваться в составе любой команды, требующей параметр того типа, который возвращает функция. В остальном использование функций совпадает с использованием процедур, так же можно передавать параметры по значению или по ссылке и т.п.

При использовании подпрограмм иногда полезна команда **exit**;, которая осуществляет выход из подпрограммы. При этом программа продолжает выполняться, начиная с точки вызова подпрограммы, из которой был осуществлён выход. При выходе из функции будет возвращено то значение, которое на момент выхода содержалось в переменной **result**.

Кстати говоря, это команду можно использовать и в основной программе, что приведёт к её закрытию.

2. Работа с файлами.

Во-первых вкратце упомяну, что в консольном режиме Delphi 7.0 работает способ работы с файлами, показанный на примере программы a+b на рис. 3.

```
program Project1;

{$APPTYPE CONSOLE}

var a,b:integer;

begin
  reset(input, 'filename1.txt'); // Открытие на чтение
  rewrite(output, 'filename2.out'); // Открытие на перезапись
  readln(a, b); // Чтение из файла
  writeln(a, ' + ', b, ' = ', a+b); // Запись в файл
  close(input); // Закрытие файлов
  close(output); // Закрытие файлов
end.
```

Рис. 3: Первый способ работы с файлами.

Из примера наглядно видно его использование, нужно только упомянуть, что при указании имени файла надо либо указывать его с каталогом, либо без, но тогда программа будет искать этот файл в той же папке, в которой находится сама. При открытии файла на чтение он должен существовать и программа должна его найти; при открытии на перезапись, если программа не найдёт файл, то он будет создан.

Второй способ работы с файлами достаточно хорошо описан на этом [сайте](#). Этот способ одинаково работает и в Delphi, и в Lazarus-e, что в графическом, что в консольном режиме. В качестве примера на следующей странице на рис. 4 приведёно решение [задачи А](#), реализованное с использованием файлов.

При работе с файлами иногда оказывается нужна команда **EOF(file_variable)**, которая позволяет определять конец файла при чтении. Подробнее об её использовании так же написано на указанном выше сайте. Кроме того есть аналогичным образом используемая функция **EOLN(file_variable)**, которая позволяет определять конец считываемой строки файла. О ней на этом сайте не написано, однако её использование полностью аналогично, а при надобности вы без труда найдёте подробности в интернете.

```

1  program project1; |
.
.  var fin, fout : textfile;
.    a,b,c,d:integer;
5
.  function min4(a,b,c,d: integer):integer;
.  var min:integer;
.  begin
.    min:=a;
10   if min>b then min:=b;
.    if min>c then min:=c;
.    if min>d then min:=d;
.    result:=min;
.  end;
15
.  begin
.    assignfile(fin,'input.txt');
.    assignfile(fout,'output.txt');
.    reset(fin);
20   rewrite(fout);
.    read(fin, a,b,c,d);
.    writeln(fout, min4(a,b,c,d) );
.    closefile(fin);
.    closefile(fout);
25  end.

```

Рис. 4: Второй способ работы с файлами – [задача А](#) .