Лекция 12. Процедуры в языке Delphi

Важной составной частью процедурного программирования в *Delphi* является использование подпрограмм – специальным образом оформленных и логически законченных блоков инструкций. В языке Delphi подпрограммы называют процедурами, и они бывают двух видов: процедуры общего вида и функции.

Использование подпрограмм позволяет сделать исходный код более стройным и наглядным, а также повторно использовать уже написанный код, поскольку подпрограмму можно вызывать любое число раз из других мест программы, или из других подпрограмм.

Использование подпрограммы состоит из 2 этапов: сначала подпрограмму описывают, а затем, уже в разделе операторов программы или подпрограммы, вызывают. Ттакая программа так и называется – вызывающая программа, а вызывающая программа самого верхнего уровня – главной программой, подпрограмма может вызывать и саму себя, но об этом в лекции про рекурсию во второй части курса.

В библиотеках *Delphi* имеется описание тысяч подпрограмм, их вызовом вы уже неоднократно занимались – достаточно взглянуть на код любой из программ этого семестра и там встретятся как минимум вызовы процедур *write* и *read*, или *writeln* и *readln*. Также, уже на первом лабораторном занятии вы познакомились с математическими функциями, такими как *sin, cos, arctan, abs, exp, ln, sqr, sqrt*.

Синтаксис

Структура подпрограммы похожа на программу в миниатюре: она обязательно содержит **заголовок**, необязательные разделы объявления констант, типов и переменных, и даже процедур и обязательный **блок операторов**. Из отличий можно выделить лишь невозможность подключать модули (раздел *uses*), а так же ограничения на объявления типов данных: если локальные простые и даже составные типы в подпрограммах вполне допустимы, то более сложные типы – классы и интерфейсы (эти типы не изучаются в данном курсе), локальными быть не могут, а потому в подпрограммах их объявлять нельзя.

Процедура общего вида:

```
{заголовок процедуры: }
procedure  <имя процедуры>(<список формальных параметров>);
const {раздел может отсутствовать}
<описание локальных констант>
type {раздел может отсутствовать}
<описание локальных типов>
var {раздел может отсутствовать}
<описание локальных переменных>

<описание локальных процедур/функций> {раздел может отсутствовать}

begin {раздел операторов обязателен}
<операторы>
end;
```

Вызов процедуры общего вида располагается в списке операторов вызывающей программы и выглядит так:

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

Функция:

```
{ заголовок функции: }
function <имя функции>(<список формальных параметров>): <тип
                                                  возвращаемого значения>;
const {раздел может отсутствовать}
<описание локальных констант>
type {раздел может отсутствовать}
<описание локальных типов>
var {раздел может отсутствовать}
<описание локальных переменных>

<описание внутренних процедур/функций> {раздел может отсутствовать}

begin {раздел операторов обязателен}
  <операторы>

  <имя функции>:=<возвращаемое значение>;
  {либо Result:= <возвращаемое значение> }
end;
```

Возвращаемое значение должно быть простого типа, хотя в последних версиях *Delphi* это ограничение снято, но практически при этом происходит неявное преобразование такого результата в параметр, так как результат функции возвращается через регистры.

Возвращаемое значение присваивается имени функции или стандартной переменной ***Result*** (она неявно описана, и имеет тип возвращаемого значения), имеющейся у каждой функции в *Delphi*. Хороший стиль предполагает наличие единственного такого присваивания в конце раздела операторов (для нерекурсивных процедур, о рекурсивных процедурах речь пойдет во второй части курса).

Вызов функции располагается в арифметическом или логическом выражении в вызывающей программе выглядит так:

```
<имя функции>(<список фактических параметров>)
```

в качестве операнда арифметического или логического выражения:

```
<переменная> := <имя функции>(<список фактических параметров>);
```
{переменной присваивается возвращаемое функцией в точку вызова значение}

или

<переменная>:=2* <имя функции>(<список фактических параметров>) + 5;
{переменной присваивается арифметическое выражение, получаемое умножением возвращаемого функцией значения на 2 и увеличением этого произведения на 5}

или

<переменная>:=(n>0) and <имя функции>(<список фактических параметров>);
{переменной присваивается результат вычисления логического выражения, которое имеет значение
истина (*True*), если n – положительно и возвращаемое функцией значение является истиной (*True*); и
ложь (*False*), если n – не положительно (при этом функция даже не вызывается при включенной оптимизации (по умолчанию)), либо
если n – положительно, а функция возвращает в точку вызова значение ложь (*False*)}

или

if <имя функции>(<список фактических параметров>) then
{возвращаемое функцией значение используется в качестве логического выражения в операторе ветвления}

Имена процедуры и функции подчиняются правилам для имен всех идентификаторов – начинаются с латинской буквы или знака подчеркивания и могут состоять только букв латинского алфавита, цифр и знаков подчеркивания, общей длиной не более 256.

Формальные и фактические параметры

Список формальных параметров состоит из списка имен параметров, обычно с указанием их типов и иногда значений по умолчанию. Разделителем описаний является **точка с запятой** «;». Формальные параметры указывают «место» в коде процедуры, куда будут подставлены **фактические параметры** при вызове этой процедуры (общего вида или функции).

Перед параметрами, получающими начальное значения извне, которые НЕ будут изменяться в теле процедуры, обычно ставится слово ***const***. Это **параметры-константы**.

Перед параметрами, получающими начальное значения извне, которые вы хотите использовать как локальные переменные в теле процедуры, обычно **ничего** не ставится (способ передачи *по умолчанию*). Это **параметры-значения**.

Перед параметрами, получающими начальное значения извне и изменения которых в теле процедуры должны отразиться в вызывающей программе ставиться ***var***. Это **параметры-переменные**.

Перед параметрами, которые НЕ получают начальное значения извне и изменения которых в теле процедуры должны отразиться в вызывающей программе ставиться ***out***. Это тоже **параметры-переменные**. Для *статических* переменных действует аналогично ***var***, для *динамических* (рассматриваются во второй части курса) – при передаче параметра память, выделенная для переменной освобождается.

Список фактических параметров является списком имен переменных, арифметических или логических выражений, по **типам** (для параметров значений и параметров-констант достаточно *совместимости* типов по присваиванию), **порядку** и **количеству** (за редким исключением случаев перегрузки процедур или наличия значений по умолчанию) точно соответствующих параметрам из **списка формальных параметров**. Разделителем описаний фактических параметров является **запятая.**

Списки параметров могут отсутствовать, и тогда процедура является процедурой без параметров. В этом случае пустые скобки тоже можно опустить.

Параметры пользовательских типов

Важнейшее синтаксическое правило: при описании процедур в спецификациях формальных параметров принципиально указываются только *имена* типов:

. . . <список идентификаторов>:<имя типа> . . .

Поэтому необходимо для любого параметра пользовательского (а не базового) типа, то есть для **массива, записи, множества, перечисляемого типа, интервального типа и т.д.**, до описания процедуры (вне процедуры) определить тип этого параметра в разделе ***type***. К этому же типу (с тем же *именем*) должен принадлежать и соответствующий фактический параметр в вызове процедуры.

Про определение разных типов можете прочитать в лекциях про типы данных (простые типы, структурированные типы: массивы, записи, множества).

Механизмы передачи параметров-констант, параметров-значений и параметров-переменных.

Формальные и фактические параметры-переменные обязательно должны принадлежать к одному типу, а формальные и фактические параметры-значения и параметры-константы должны быть совместимы по присваиванию. То есть правила подстановки следующие:

Формальный параметр	**Фактический параметр**
параметр-константа (*const*)	Выражение
параметр-значение	Выражение
параметр-переменная (*var*, *out*)	Переменная того же (по имени!) типа

Механизмы передачи в процедуру *параметров-констант*, *параметров-значений* и *параметров-переменных* принципиально отличаются.

Передача параметра по значению:

1) При передаче **параметров-значений** для их обработки **создается локальная переменная процедуры, в которую копируется значение из фактического параметра**. По завершении выполнения процедуры значение этой переменной недоступно. Соответствующий **фактический параметр не меняется**.

2) При передаче **параметров-констант** для их обработки **создается локальная константа, в которую копируется значение из фактического параметра**. Значение константы нельзя менять в теле процедуры. Соответствующий **фактический параметр не меняется**. В отличие от параметров-значений параметры-константы **пользовательских типов** (массив, запись, строки) используются для генерации компилятором более оптимального кода для передачи этих параметров (могут передаваться и по ссылке – решает компилятор). Параметры-**константы базовых типов** на эффективность передачи не влияют, но больше говорят о характере использования такого параметра.

Передача параметра по ссылке:

При передаче **параметров-переменных** перед выполнением процедуры устанавливается ссылка на переменную (фактический параметр); иначе говоря, в процедуру ***передается адрес фактического параметра*. Все действия процедуры, таким образом, выполняются над фактическим параметром**. Если начальное значение фактического параметра должно быть использовано в процедуре, то параметр следует передавать с ***var***. Если указать перед параметром ***out***, то память, выделенная для переданной **динамической** переменной (изучаются во второй части курса), будет освобождена (*nil*) при входе в процедуру, но для **статической переменной указание *out* аналогично *var*.**

Следовательно, **выходные** параметры процедуры необходимо специфицировать как параметры-переменные (передавать по ссылке с ***var*** или ***out***). А **входные** – как параметры-значения или параметр-константы (передавать по значению).

Следствие. Как *var* можно описывать входные **массивы-параметры** с целью **экономии памяти** (чтобы явно указать, что не нужно создавать локальную копию значения всех элементов массива).

Файловые переменные тоже следует передавать с ***var***, так как при работе с файлом (чтение/запись, открытие/закрытие) меняется позиция в файле или его состояние, или даже имя связанного с файловой переменной файла.

Достоинства процедур и функций

1) ***Повышают читабельность*** кода за счет сокращения длины кода его отдельных модулей (автономных частей). **Старайтесь, чтобы тексты каждой из процедур/функций и главной программы умещались на экране целиком;**

2) Процедуры с параметрами позволяют ***повторно использовать*** их код с другими фактическими параметрами, что к тому же ***избавляет от дублей кода***, к каждом из дублей могут при отладке появляться/оставаться разные ошибки, поскольку можно легко перепутать один код с другим.

Проиллюстрируем сказанное:

<table>
<tr><th colspan="2">Цикл сокращает код за счет введения параметра i</th><th colspan="2">Функция сокращает код (в программе остается только ее вызов) и появляется возможность повторного использования при введении параметров nx и X (одна функция вместо двух)</th></tr>
<tr>
<td>

```
Readln(A[1]);
Readln(A[2]);
Readln(A[3]);
Readln(A[4]);
Readln(A[5]);
Readln(A[6]);
Readln(A[7]);
Readln(A[8]);
Readln(A[9]);
Readln(A[10]);
```

</td>
<td>

```
For i:=1 to 10 do
  Readln(A[i]);
```

</td>
<td>

```
Function SumA: real;
Var i:byte; S: real;
Begin
  S:=0;
  For i:=1 to na do S:=S+A[i];
  SumA:=S;
End;

Function SumB: real;
Var i:byte; S: real;
Begin
  S:=0;
  For i:=1 to nb do S:=S+B[i];
  SumB:=S;
End;
      Вызовы в программе:
SA:=SumA;
SB:=SumB;
```

</td>
<td>

```
Type
  massiv = array [1..10] of
                      real;

Function Sum(const nx:byte;
        var X: massiv): real;
Var i:byte; S: real;
Begin
  S:=0;
  For i:=1 to nx do S:=S+X[i];
  Sum:=S;
End;


      Вызовы в программе:
SA:=Sum(na, A);
SB:=Sum(nb, B);
```

</td>
</tr>
</table>

Прерывание работы программы и процедуры

Любая процедура (общего вида или функция) выполняется до тех пор, пока не будет выполнена последняя инструкция в блоке операторов этой процедуры, или пока в ее теле не встретится вызов специальной процедуры ***exit***. Процедура *exit* досрочно прерывает выполнение процедуры и возвращает управление инструкции, следующей за вызовом данной процедуры.

Аналогично, процедура ***halt***(0) прерывает работу всей программы, при этом ещё и передавая код завершения программы, в программу, которая её запустила, если таковая имеется. (Можно запустить одну программу из другой, передать через строку параметров программы значения и вернуть с помощью *halt* число-результат.)