Процедуры

---

1. Процедура - средство повышения уровня программирования

Процедура - поименованная часть программы, может быть очень большая, для ***активации*** которой достаточно выполнить небольшой ***оператор вызова процедуры***. Процедура определяется ***описанием процедуры***, в составе ***раздела описания процедур*** блока.

Упрощенный синтаксис описания процедуры:
procedure <Имя процедуры>; <Блок>;

Синтаксис соответствующего оператора вызова процедуры:
<Имя процедуры>

Пример.

```
program SwapTst1;
var x,y,t: Integer;

procedure Swap1;
begin t:=x; x:=y; y:=t end; {Swap1}

begin
x:=3; y:=7;
WriteLn('x=',x,' y=',y);
Swap1;
WriteLn('x=',x,' y=',y);
Swap1;
WriteLn('x=',x,' y=',y);
end.
```

Выход программы SwapTst1:

```
x=3 y=7
x=7 y=3
x=3 y=7
```

Преимущества использования процедур:

- ***Повышение уровня программирования***, т.е. возможность использования абстракции, как метода рассуждения при построении сложных программ. Следствием является возможность придания программам более понятной структуры и возможность разработки сложной программы несколькими относительно независимыми разработчиками.

- ***Возможность использования библиотек процедур*** (в АЯ TurboPascal реализуется с помощью понятия ***модуля***).

- ***Сокращения размера программы*** в случае многократных вызовов.

Недостатки использования процедур:

- Некоторое ***увеличение времени выполнения программы*** из-за необходимости организации вызова процедур (затраты на ***связывание***). Это означает, что дробить программу на множество мелких процедур не целесообразно.

2. Локальность

Обратим внимание на то, что в программе SwapTst1 переменная t предназначена только для использования в процедуре Swap1. В основной программе она не имеет никакого смысла, кроме как "переменная для процедуры Swap1". Для исключения подобных неудобств в современных языках программирования вводится понятие локальности. Локальными будем называть те программные объекты (типы, переменные, ...), которые описываются в блоке процедуры. Пример.

```
program SwapTst2;
var x,y: Integer;

procedure Swap2;
var t: Integer;
begin t:=x; x:=y; y:=t end; {Swap2}

begin
x:=3; y:=7;
WriteLn('x=',x,' y=',y);
Swap2;
WriteLn('x=',x,' y=',y);
Swap2;
WriteLn('x=',x,' y=',y);
end.
```

Локальность ещё в большей степени повышает абстрактность понятия процедуры, так как позволяет "спрятать" в тексте процедуры детали её реализации, которые не существенны для использования процедуры на внешнем уровне.

Попутно появляется понятие ***области видимости имени*** объекта программы, т.е. той части программы, в которой можно обращаться к объекту. ***Имя считается видимым в том блоке, в котором оно описано, за исключением тех вложенных блоков, в которых описано то же имя***. Кроме того, надо следить за тем, чтобы всякое имя было описано до его использования. Пример.

```
program SwapTst3;
var x,y,t: Integer;

procedure Swap3;
var t: Integer;
begin t:=x; x:=y; y:=t end; {Swap3}

begin
x:=3; y:=7; t:=0;
WriteLn('x=',x,' y=',y);
Swap3; Inc(t);
```

```
WriteLn('x=',x,' y=',y);
Swap3; Inc(t);
WriteLn('x=',x,' y=',y);
WriteLn('t=',t);
end.
```

Выход программы SwapTst3:

```
x=3 y=7
x=7 y=3
x=3 y=7
t=2
```

В программе SwapTst3 используются две переменные с именем t. Одна переменная описана в блоке программы и имеет смысл счётчика вызовов процедуры Swap3. Другая - описана в блоке процедуры Swap3 и используется просто как рабочая переменная. Это две совершенно разные переменные! Они даже могут иметь совершенно различные типы. Область видимости первой переменной это блок всей программы, за исключением вложенного блока процедуры. Область видимости второй - только блок процедуры.

Понятие видимости имени может показаться при первом знакомстве достаточно надуманным и сложным. Но, оказывается, оно очень полезно, так как позволяет вести разработку процедуры не заботясь о возможном конфликте имён, т.е. относительно независимо от разработки других частей программы.

Кроме того, появляется ещё одно понятие - понятие глобального имени. ***Имя называется глобальным в блоке его использования, если оно описано в одном из охватывающих блоков***. В программе SwapTst3 переменные x и y являются глобальными в блоке процедуры Swap3. Назначение глобальных переменных - передача значений между блоком процедуры и её окружением, или, как говорят, её ***контекстом***.

Интересно отметить, что сама программа, самый внешний её блок, тоже имеет контекст. Это множество имён стандартных констант, типов, переменных и процедур. Их следует отличать от ключевых слов - в отличие от последних их можно переопределить во вложенных блоках.

3. Параметры

Недостатком процедуры Swap3 можно считать то, что она "умеет" обменивать значения только двух переменных - x и y. Для устранения этого недостатка вводятся понятия ***формальных*** и ***фактических параметров*** процедуры, которые записываются сразу за именем процедуры в круглых скобках. Пример.

```
program SwapTst4;
var a,b,c: Integer;

procedure Swap4(var x,y: Integer);
var t: Integer;
begin t:=x; x:=y; y:=t end; {Swap4}

procedure WrABC;
begin WriteLn('a=',a,' b=',b,' c='c) end; {WrABC}

begin
a:=1; b:=2; c:=3;
WrABC;
```

```
Swap4(a,b);
WrABC;
Swap4(b,c);
WrABC;
end.
```

Выход программы SwapTst4:

```
a=1 b=2 c=3
a=2 b=1 c=3
a=2 b=3 c=1
```

Механизм параметров существенно расширяет возможности процедуры, ещё более повышая уровень абстрактности этого понятия - так как параметр можно рассматривать как обобщение понятия возможного множества переменных.

Параметры в описании процедуры называют формальными. Параметры в операторе вызова процедуры - фактическими. Между ними существует строгое ***позиционное соответствие***. Это означает, что число параметров должно быть одинаковым, а их типы должны попарно совпадать друг с другом. Синтаксис списка формальных параметров:

*[***var***]* <Имя параметра> *{* , <Имя параметра> *}* : <Имя типа>
{ ; *[***var***]* <Имя параметра> *{* , <Имя параметра>*}* : <Имя типа> *}*

Синтаксис списка формальных параметров:

<Выражение> *{* , <Выражение> *}*

Очень важно правильно понять, каким образом происходит передача параметров между ***вызывающей программной единицей*** (содержащей оператор вызова процедуры) и ***вызываемой процедурой***. ***В Паскале существуют два способа передачи параметра*****:**

1. ***По значению*** (без **var**). В этом случае при вызове процедуры ей передаётся значение параметра. Параметр без **var** фактически является локальной переменной в блоке процедуры, инициализируемой при её вызове. Изменения этого параметра в ходе выполнения процедуры ни в коем случае не могут привести к каким-либо изменениям в вызывающей. Таким образом, характер передачи параметра без **var** всегда односторонний - значение передаётся только в одну сторону. Важно отметить, что ***фактический параметр, соответствующий такому формальному параметру, может быть выражением*** - его значение автоматически вычисляется перед фактическим вызовом процедуры, в ходе которого оно передаётся формальному параметру.

2. ***По адресу*** (с **var**). Сразу отметим, что ***в этом случае фактический параметр не может быть выражением и обязательно должен быть переменной***. Процедуре в этом случае ***передаётся адрес этой переменной***. Процедура устроена таким образом, что все её действия с параметром с **var** означают по сути эти же действия с соответствующим фактическим параметром-переменной. Можно считать, что ***на время выполнения процедуры фактический параметр-переменная просто замещает формальный параметр***.

Важным остаётся вопрос о том, когда какой способ следует использовать. Сразу выделим один особый случай, а именно: ***если процедура должна возвратить значение вызывающей, то сделать это можно только с помощью параметра, передаваемого по адресу!*** Само собой разумеется, что такой параметр может служить и для передачи

значения процедуре. Но, если мы хотим защитить фактический параметр-переменную от возможных изменений при выполнении процедуры, то следует использовать передачу по значению. Кроме всего прочего, следует учитывать, что передача по значению происходит медленнее и требует большего объёма основной памяти, чем передача по адресу. Особенно это заметно, при передаче параметров большого размера, например, массивов.

Мы должны вспомнить и о другой возможности передачи значений между вызывающей и вызываемой программными единицами - с помощью глобальных переменных. Таким образом, возникает вопрос и о том, когда следует предпочесть этот способ. Передача через параметры всегда явно заметна для того, кто изучает программу. Обращения же к глобальным переменным в тексте процедуры легко не заметить. Использование механизма передачи значений через параметры способствует большей абстрактности понятия процедуры. На практике это означает, что в этом случае не надо заводить каких-либо дополнительных переменных в вызывающей. С другой стороны, излишне длинные списки параметров тоже могут отрицательно сказаться на понимаемости программы, не говоря о простом увеличении размера текста программы. В этой связи желателен некоторый компромисс - если некоторые структуры данных занимают в программе центральное положение, достаточно хорошо всем понятны и их количество не очень велико, то они вполне могут быть глобальными.

4. Процедуры-функции

Процедура-функция, или просто функция, очень похожа на процедуру общего вида, и отличается от последней возможностью возвратить одно скалярное значение через имя процедуры. Это даёт возможность использовать оператор вызова функции в выражениях. Синтаксис описания функции:

function <Имя функции>*[*(<Список формальных параметров>)*]*:<Имя типа возвращаемого значения>;

В разделе операторов блока процедуры обязательно должен быть оператор вида:

<Имя функции> := <Выражение>

Их может быть несколько. Тот, который будет выполнен последним перед завершением функции и определит, какое значение будет ею возвращено.

Пример.

```
program MaxTst;
var a,b,c,y: Integer;

function Max(x,y: Integer): Integer;
begin if x>y then Max:=x else Max:=y end; {Max}

begin
a:=1; b:=2; c:=3;
y:=Max(a,Max(b,c));
WriteLn('y=',y);
end.
```

Выход программы TstMax:

```
y=3
```

Последний пример иллюстрирует также тот факт, что ***имя параметра процедуры, локально в описании этой процедуры*** (y).

5. Рекурсия

Процедура называется рекурсивной, если в её описании используется вызов самой этой процедуры.

Пример.

```
program FactTst;
function Fact(n: Integer): Integer;
begin
if n=1 then begin Fact:=1; Exit end;
Fact:=Fact(n-1)*n
end; {Fact}
begin WriteLn('5!=',Fact(5)) end.
```

Выход программы FactTst:

```
5!=120
```

В этой программе вы также видите пример вызова стандартной процедуры Exit. Вызов этой процедуры немедленно прекращает текущую активацию данной программной единицы, в том числе и главной.

Различают ***прямую*** и ***общую*** рекурсию. Процедура Fact иллюстрирует прямую рекурсию - в описании процедуры происходит вызов самой себя. Если же определяемая процедура вызывается косвенно, т.е. через посредничество вызова каких-либо других процедур, то такая рекурсия называется общей.

Следует отметить, что организация рекурсивных процедур подчиняется строгому правилу, а именно: хотя бы одна из ветвей рекурсивной процедуры не должна содержать рекурсивного вызова. Эта ветвь определяет ***ограничение рекурсии***, без которого произойдёт ***рекурсивное зацикливание*** с последующим аварийным завершением программы из-за "переполнения стека".

Рекурсивные процедуры эффективны при работе со сложными динамическими структурами данных, такими как списки, деревья, графы.

6. Пошаговая детализация программ с применением процедур

Процедуры очень удобно использовать для реализации абстрактных блоков программ. Программирование с применением процедур рассмотрим на примере уже известной нам задачи о нахождении среднего по значению из трёх различных чисел входа.

Практически сразу мы можем написать такую программу:

```
program MiddleTst;
var a,b,c,r: Integer;
begin
ReadLn(a,b,c);
r := Middle(a,b,c);
WriteLn(r);
end.
```

Функция Middle вычисляет среднее по значению из переменных a, b и c с различными значениями. Реализуем её.

```
function Middle(a,b,c: Integer): Integer;
begin
if a > b then
    Middle := MiddleGt(a,b,c)
else
    Middle := MiddleGt(b,a,c);
end; {Middle}
```

Функция MiddleGt вычисляет среднее по значению из переменных a, b и c с различными значениями, причём a > b. Реализуем функцию MiddleGt.

```
function MiddleGt(a,b,c: Integer): Integer;
begin
if c > a then
    MiddleGt := a
else
    MiddleGt := max(b,c);
end; {MiddleGt}
```

Теперь реализуем функцию max.

```
function max(a,b: Integer): Integer;
begin
if a > b then max := a else max := b;
end; {max}
```

Осталось составить все фрагменты в единую программу.

```
program MiddleTst;
var a,b,c,r: Integer;

function max(a,b: Integer): Integer;
begin
if a > b then max := a else max := b;
end; {max}

function MiddleGt(a,b,c: Integer): Integer;
begin
if c > a then
    MiddleGt := a
else
    MiddleGt := max(b,c);
end; {MiddleGt}

function Middle(a,b,c: Integer): Integer;
begin
if a > b then
    Middle := MiddleGt(a,b,c)
else
    Middle := MiddleGt(b,a,c);
end; {Middle}

begin
ReadLn(a,b,c);
r := Middle(a,b,c);
WriteLn(r);
end.
```

Порядок следования процедур в разделе описания процедур должен быть таким, чтобы их описания текстуально предшествовали операторам их вызова. При использовании общей рекурсии такой порядок в принципе не достижим. Для таких случаев предусмотрена возможность ***предварительных описаний процедур***. Предварительное описание процедуры состоит из её заголовка и ключевого слова **forward**, которое записывается вместо блока процедуры. Само описание процедуры должно присутствовать где-то ниже в тексте программы. Заголовки предописания процедуры и её описания должны полностью совпадать. Пример предописания: **function** MiddleGt(a,b,c: Integer): Integer; **forward**; .
