*Лабораторная работа №2.*

# *Структурные операторы.*

Время: 180 мин.
Что нужно освоить:
– виды и особенности использования структурных операторов языка Паскаль.

## ОПИСАНИЕ СТРУКТУРНЫХ ОПЕРАТОРОВ.

Операторы языка Turbo Pascal можно разделить на два класса: простые и сложные. Простые операторы называют простыми, так как они не содержат внутри себя других операторов. Сложные операторы чаще называют структурными, так как они представляют собой определенным образом организованную структуру, в которую по определенным правилам могут включаться простые операторы. К простым операторам относят операторы ввода и вывода информации, присваивания, перехода, а также пустой оператор (может быть обозначен отдельно стоящим символом «;»).

**1. Составной оператор.**

Наверное, чаще всего из структурных операторов в текстах программ можно встретить составной оператор, представляющий собой операторные скобки, то есть, по сути, скобки для объединения других операторов. В качестве открывающей скобки используется зарезервированное слово begin, а в качестве закрывающей end. Между скобками можно расположить произвольное число простых и структурных операторов, при этом составной оператор будет восприниматься как один цельный оператор. Именно для этого он и предназначен, то есть он обычно используется в том месте программного кода, где по правилам языка может стоять только один оператор, а для реализации алгоритма требуется использование нескольких. Ранее мы уже встречались с такой организацией кода – любая программа обрамлена операторными скобками, что может выглядеть, например, так:

```
begin
  write('Сидоров');
end.
```

или так:

```
begin
  write('Сидоров');
  readln;
end.
```

Синтаксис языка предполагает, что точкой заканчивается текст программы, но в иных местах не следует ставить точку после закрывающей скобки end. Обычно после end ставят символ «;»:

```
begin
  оператор 1;
  оператор 2;
  …
  оператор n;
end;
```

Синтаксис языка обязывает ставить символ «;» не только после end, но и после любого иного оператора, но перед закрывающей скобкой этот символ необязателен:

```
begin
  write('Сидоров');
  readln      ' тут символ ";" можно не ставить
end;
```

Допускается вложение составных операторов друг в друга:

```
begin
```

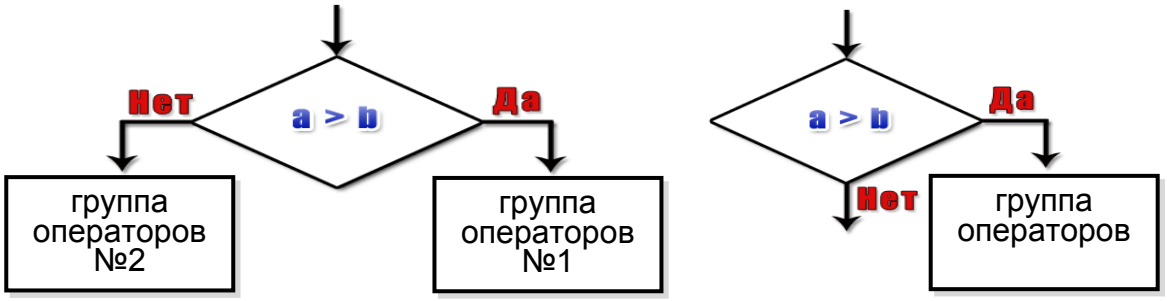
```
  оператор a.1;
  оператор a.2;
  begin
    оператор b.1;
    оператор b.2;
    …
    оператор b.n;
  end;
  …
  оператор a.n;
end;
```

Составной оператор обычно используется в составе других структурных операторов.

**2. Условный оператор.**

Оператор называют условным, если он обеспечивает выполнение или невыполнение некоторого оператора (в том числе и составного) в зависимости от проверяемого условия. Условный оператор позволяет выполнять нелинейные алгоритмы, такие в которых предусмотрено прерывание линейной последовательности действий в зависимости от обрабатываемых данных и выполнение одной из предусмотренных программистом ветвей алгоритма.

Конструкция условного оператора позволяет организовать два варианта его использования: с двумя нагруженными ветвями алгоритма и с одной.




**<u>Вариант 1</u>** (с двумя ветвями):

```
if условие
then
  begin
    {группа операторов №1}
    {эти операторы выполняются, если условие истинно}
  end
else
  begin
    {группа операторов №2}
    {эти операторы выполняются, если условие ложно}
  end
end;
```

Этот вариант можно интерпретировать так – если условие истинно, то выполняй первую группу операторов, иначе выполняй вторую группу операторов.

Создайте папку для сохранения программ лабораторной работы. Запустите Turbo Pascal и апробируйте первую программу:

```
program prog01;
uses crt;
const parol='prog01';
var p: String;
begin
  clrscr;
```

```
  write('Введите пароль - ');
  readln(p);
  if p=parol
    then writeln('Верно!')
    else writeln('Неверно!');
  readln;
end.
```

Эта программа проверяет пароль, введенный пользователем, сравнивая его со значением константы `parol`. В данной программе в структуре условного оператора не использовался составной оператор (`begin end`) так как в случае истинности условия выполняется только один оператор `writeln('Верно!')`, и при ложности условия также только один оператор – `writeln('Неверно!')`. Составной оператор обычно используют в тех случаях, когда при выполнении/невыполнении условия необходимо обеспечить реализацию сразу нескольких операторов:

```
program prog02;
uses crt;
const parol='prog02';
var p: String;
begin
  clrscr;
  write('Введите пароль - ');
  readln(p);
  if p=parol
  then
   begin
     writeln('Пароль введен верно!');
     writeln('Вы допущены к работе.');
   end
  else
   begin
     writeln('Пароль введен неверно!');
     writeln('Вы не допущены к работе.');
   end;
  readln;
end.
```

► Самостоятельно разработайте программу p01.pas, которая запрашивает у пользователя любое натуральное (целое положительное) число и проверяет его на четность/нечетность (используйте либо mod либо odd / см. Help), результат выводится на экран. ◄

**Вариант 2** (только одна ветвь нагружена):

```
if условие
then
  begin
    {группа операторов}
    {эти операторы выполняются, если условие истинно}
  end;
```

Этот вариант можно интерпретировать так – если условие истинно, то выполняй группу операторов, иначе ничего не выполняй. То есть в случае ложности проверяемого условия происходит непосредственный переход на оператор, следующий сразу же за условным оператором.

```
program prog04;
uses crt;
const parol='prog04';
var p,n: String;
```

```
begin
  clrscr;
  write('Введите пароль - ');
  readln(p);
  if p<>parol then n:=' не';
  writeln('Пароль введен',n,' верно!');
  writeln('Вы',n,' допущены к работе.');
  readln;
end.
```

Программа prog04 реализует тот же алгоритм, что и предыдущая, но с использованием условного оператора с одной нагруженной ветвью.

Следует отметить, что в условном операторе (if then) на позиции условие не обязательно должна стоять логическая операция, а может также стоять и переменная логического типа (ведь множество её значений также ограничено – true и false):

```
program prog05;
uses crt;
const parol='prog05';
var p,n: String;
    b: boolean;
begin
  clrscr;
  write('Введите пароль - ');
  readln(p);
  b:=p<>parol;
  if b then n:=' не';
  writeln('Пароль введен',n,' верно!');
  writeln('Вы',n,' допущены к работе.');
  readln;
end.
```

В программе prog05 в логическую переменную записывается результат выполнения логической операции отношения – **b:=p<>parol;**, и, в дальнейшем, этот результат используется в условном операторе при проверке условия – **if b then n:=' не';**.

Естественно сам условный оператор может быть использован в структуре другого условного оператора на равных основаниях с любым иным оператором. Пусть при входе в программу необходимо проверить не только пароль, но и логин пользователя:

```
program prog06;
uses crt;
const login='log06';
      parol='prog06';
var par,log: String;
begin
  clrscr;
  write('Введите логин - '); readln(log);
  write('Введите пароль - '); readln(par);
  if log=login
  then
    if par=parol
    then
      begin
        writeln('Пара Логин и Пароль введены верно!');
        writeln('Вы допущены к работе.');
      end
    else
      begin
        writeln('Пароль введен неверно!');
        writeln('Вы не допущены к работе.');
```

```
      end
  else
    begin
      writeln('Логин введен неверно!');
      writeln('Вы не допущены к работе.');
    end;
  readln;
end.
```

В программе `prog06` во внешнем условном операторе проверяется правильность введенного пользователем логина, если логин введен верно, то управление передается внутреннему условному оператору для проверки пароля, а если логин введен неверно, то на экран выводится соответствующее сообщение. Внутренний условный оператор аналогично проверяет правильность введенного пароля.

Зачастую при наличии нескольких проверяемых условий не требуется дифференцировать место, где именно допущена ошибка. В этих случаях можно не использовать несколько вложенных друг в друга условных операторов, заменив их проверкой одного, но сложного отношения ( `(log=login) and (par=parol)` ):

```
program prog07;
uses crt;
const login='log07';
      parol='prog07';
var par,log,n: String;
begin
  clrscr;
  write('Введите логин - '); readln(log);
  write('Введите пароль - '); readln(par);
  if (log=login) and (par=parol)
  then
    begin
      writeln('Пара Логин и Пароль введены верно!');
      writeln('Вы допущены к работе.');
    end
  else
    begin
      writeln('Допущена ошибка при вводе пары Логин/Пароль!');
      writeln('Вы не допущены к работе.');
    end;
  readln;
end.
```

► Разработайте программу p02.pas, которая проверяет возможность существования треугольника с заданными сторонами (треугольник может быть изображен на плоскости, если сумма длин любой пары сторон больше длины третьей стороны). Программа запрашивает у пользователя длины сторон треугольника и на основании этих данных делает вывод о возможности существования треугольника с такими сторонами. ◄

### 3. Оператор выбора.

Один условный оператор способен обеспечить разветвление алгоритма на два пути. Однако, зачастую, необходимо иметь оператор, способный обеспечить развилку на большее количество путей. Например, выбрать процент скидки на покупку в зависимости от общей стоимости покупки (до 200р. – 0%; до 500р. – 1%; до 1000р. – 2%; до 5000р. – 3%; свыше 5000р. – 5%). Такой алгоритм можно, конечно, организовать с помощью нескольких вложенных условных операторов, но такая конструкция будет громоздкой и сложной

для понимания. Для реализации такого алгоритма следует использовать специально созданный для этих целей оператор выбора – case:

```
case селектор of
  список 1: begin
              {группа операторов 1}
            end;
  список 2: begin
              {группа операторов 2}
            end;
  ...
  список n: begin
              {группа операторов n}
            end;
  else
    begin
      {группа операторов}
    end;
end;
```

Селектор в операторе case может быть как переменной, так и целым выражением. Тип значения селектора может быть только порядковым (целочисленным, символьным, логическим), но не может быть вещественным или строковым. Допустимый диапазон значений селектора: от –32768 до 32767. В зависимости от значения переменной или результата вычисления выражения будет выполнена только одна группа операторов, представленных в структурном операторе case. Группа операторов, которая будет выполнена, определяется в зависимости от того в какой список констант входит значение селектора. Если ни в один из обозначенных списков селектор не входит, тогда выполняются инструкции из блока else. Список может быть представлен и отдельной константой. Рассмотрим примеры использования оператора выбора.

```
program prog08;
uses crt;
var i: longint;
    p: byte;
    r: real;
begin
  clrscr;
  write('Введите сумму покупки в рублях - '); readln(i);
  case i of
    0..199:     p:=0;
    200..499:   p:=1;
    500..999:   p:=2;
    1000..4999: p:=3;
  else          p:=5;
  end;
  writeln('Ваша скидка - ',p,'%.');
  r:=i-i*p/100;
  writeln('С учетом скидки сумма к оплате - ',r:9:2,'р.');
  readln;
end.
```

В программе prog08 использовался селектор интервального типа, так как организована проверка вхождения селектора в списки констант. В операторе выбора не обязательно использовать именно интервалы, можно использовать и конкретные отдельные значения селектора:

```
program prog09;
uses crt;
var k,p: byte;
    r: real;
```

```
begin
  clrscr;
  write('Введите количество предметов в покупке - '); readln(k);
  case k of
    0,1: p:=0;
    2:   p:=1;
    3:   p:=2;
    4,5: p:=3;
  else   p:=5;
  end;
  writeln('Ваша скидка - ',p,'%.');
  readln;
end.
```

Синтаксис языка также позволяет совмещать интервалы с конкретными значениями:

```
program prog10;
uses crt;
var k,p: byte;
    r: real;
begin
  clrscr;
  write('Введите количество предметов в покупке - '); readln(k);
  case k of
    0..2:   p:=0;
    3,4:    p:=1;
    5,6:    p:=2;
    6,7..9: p:=3;
  else      p:=5;
  end;
  writeln('Ваша скидка - ',p,'%.');
  readln;
end.
```

Операторы выбора можно вкладывать друг в друга. Блок else в операторе выбора необязательный (может отсутствовать), при этом, естественно, может сложиться ситуация когда не будет выполнена ни одна группа операторов, так как селектор может не войти ни в один из обозначенных интервалов. Зарезервированное слово end в обязательном порядке ставится в конце оператора выбора и, по существу, является закрывающей скобкой в этом структурном операторе. Открывающей скобкой, в свою очередь, можно считать слово case.

► Разработайте программу p03.pas, которая запрашивает у пользователя возраст мужчины и, в зависимости от введенного значения, выводит на экран результат из списка: до 9 лет – дитя; до 12 лет – отрок; до 18 лет – юноша; до 24 лет – молодой человек; до 60 лет – мужчина; старше 60 лет – старик. ◄

### 4. Операторы цикла.

Довольно часто в программах возникает необходимость организовать циклическую обработку данных. Оператор цикла условно можно разбить на две составляющие: заголовок (шапка цикла) и тело цикла. Заголовок определяет количество повторов или условие окончания повторов, а тело цикла содержит список повторяющихся действий (группу операторов).

#### 4.1. Оператор for

Если необходимое количество повторов заранее известно, то удобнее всего воспользоваться оператором for. В структуре этого оператора предусмотрен специальный параметр, который и определяет количество повторов. На каждом шаге цикла этот параметр

будет изменять своё значение на единицу до достижения предельного значения, установленного в заголовке цикла. Возможны два варианта организации оператора for.

Вариант 1 – организация цикла с увеличением счетчика:

```
for счетчик := начальное_значение to конечное_значение do
begin
  {тело цикла}
end;
```

Счетчик – это переменная порядкового типа, которая последовательно принимает значения от `начальное_значение` до `конечное_значение`. Предполагается, что начальное значение меньше, чем конечное и счетчик с каждым шагом увеличивается. Если начальное и конечное значения равны, то тело цикла выполнится один раз. Если начальное значение выбрано большим, чем конечное, то тело цикла ни разу не выполнится. Тело цикла (группа операторов) располагается между операторными скобками `begin end`. Если оператор один, то его можно не помещать в операторные скобки.

Пример:

```
for i := 0 to 9 do
  write(i);
```

Этот цикл выведет на экран последовательно все цифры от 0 до 9.

Вариант 2 – организация цикла с уменьшением счетчика:

```
for счетчик := начальное_значение downto конечное_значение do
begin
  {тело цикла}
end;
```

Данный цикл организован и работает аналогично первому варианту, но значения счетчика с каждым шагом уменьшаются.

Пример:

```
for i := 9 downto 0 do
  write(i);
```

Этот цикл выведет на экран последовательно все цифры от 9 до 0.

► Напишите программу p04.pas, которая выводит на экран все символы, коды которых в таблице ASCII лежат в диапазоне от 40 до 80 и являются четными. ◄

### 4.2. Оператор repeat

Оператор цикла repeat не имеет заранее заданного количества повторений, достижение которого и означает выход из цикла. В организации оператора repeat предусмотрена проверка условия окончания цикла, поэтому программисту следует самостоятельно организовать либо счетчик числа повторов, либо позаботиться о том, чтобы условие окончания цикла когда-нибудь наступило.

```
repeat
  {группа операторов – тело цикла}
until условие выхода из цикла;
```

Здесь в качестве операторных скобок выступают зарезервированные слова `repeat` (повторяй) и `until` (до тех пор пока).

Пример:

```
program prog11;
uses crt;
var i: byte;
begin
  clrscr;
  i:=0;
  repeat
    write(i);
    inc(i); {счетчик, аналогично i:=i+1}
  until i>9;
```

```
  readln;
end.
```

Эта программа выведет на экран последовательно все цифры от 0 до 9 аналогично оператору for, рассмотренному ранее.

► Напишите программу p05.pas (с использованием оператора `repeat`), которая выводит на экран цифры в обратном порядке – от 9 до 0. ◄

Оператор `repeat` удобно использовать для организации меню, если в теле цикла производится проверка нажатых клавиш. Например, при нажатии клавиши `Esc` – выход из программы; при нажатии клавиши «`S`» – сохранить результаты и т.п.

Все предыдущие программы завершались процедурой `readln`, которая задерживала экран до нажатия клавиши `Enter`. Это смотрится как-то неказисто, ведь обычно, когда мы хотим покинуть программу, мы нажимаем `Esc`.

Для начала напишем код, который будет содержать цикл `repeat`. В теле цикла будет считываться символ с клавиатуры и размещаться в переменной `c`. Условием выхода из цикла пусть пока будет нажатие клавиши пробел (`c=' '`).

```
program prog12;
uses crt;
var c: char;
begin
  clrscr;
  repeat
    c:=readkey;
  until c=' ';
end.
```

Когда вы запустите эту программу, то увидите пустой экран – он будет отображаться до тех пор пока вы не нажмете пробел. Эта программа совсем не функциональна, но зато мы уже вплотную подошли к избирательной обработке клавиатурных событий. Дело в том, что в переменной `c` хранится символ, введенный с клавиатуры и его можно обрабатывать. Давайте программу немного функционально дополним – пусть она будет выводить нажатый нами символ до тех пор пока не будет нажат пробел:

```
program prog12;
uses crt;
var c: char;
begin
  clrscr;
  repeat
    c:=readkey;
    write(c);
  until c=' ';
end.
```

Апробируйте программу. Попробуйте её доработать так, чтобы выход из программы был по нажатию клавиши «q». Тут может возникнуть проблема, которая заключается в том, что с помощью одной и той же клавиши можно ввести 4 символа. В частности – «q», «Q», «й», «Й». Вводимый символ зависит от нажатой клавиши Caps Lock и установленного языка ввода. Чтобы обеспечить безошибочную работу алгоритма выхода из программы следует в условии выхода из цикла проверять совпадение введенного символа с одним из четырех возможных вариантов (в нашем случае – «q», «Q», «й», «Й»):

```
program prog13;
uses crt;
var c: char;
begin
  clrscr;
  repeat
```

```
    c:=readkey;
    write(c);
  until (c='q') or (c='Q') or (c='й') or (c='Й');
end.
```

Полагаю, что у вас уже созрел вопрос: а как же проверить нажатие служебных клавиш таких как `Backspace`, `Tab` или `Esc`? Можно сделать следующим образом – в качестве условия проверять совпадение не с самим символом, а с его порядковым номером (кодом). Для этого воспользуемся функцией `ord(c)`, возвращающей код символа:

```
program prog14;
uses crt;
var c: char;
begin
  clrscr;
  repeat
    c:=readkey;
    writeln(c,' - ',ord(c));
  until (c='q') or (c='Q') or (c='й') or (c='Й');
end.
```

Апробируйте программу. Давайте изменим её так, чтобы выход из программы был по нажатию «1». Вы без труда определите, что код единицы 49 – это её порядковый номер в таблице ASCII.

```
program prog15;
uses crt;
var c: char;
begin
  clrscr;
  repeat
    c:=readkey;
    writeln(c,' - ',ord(c));
  until ord(c)=49;
end.
```

► Апробируйте программу и ***самостоятельно*** измените её так, чтобы программа p06.pas завершала свою работу по нажатию клавиши Esc. ◄

В дальнейшем в конце программы вместо невнятной процедуры `readln` уже будем вставлять более адекватный код:

```
repeat c:=readkey until ord(c)=код_Esc;
```

Его можно записывать именно так – в одну строку. Вместо **`код_Esc`** вы будете писать целое число (код клавиши Esc), которое сами нашли. И символ «;» после `readkey` можно не вставлять, так как слово `until` является, по сути, закрывающей скобкой.

### 4.3. Оператор while

Оператор цикла `while` не имеет заранее заданного количества повторений, достижение которого и означает выход из цикла. В организации оператора while предусмотрена проверка условия, при истинности которого тело цикла следует выполнить. Затем снова проверяется условие, при истинности которого снова выполняется тело цикла и так до тех пор пока условие не станет ложным.

```
while условие do
begin
  {группа операторов – тело цикла}
end;
```

Здесь в качестве операторных скобок выступают зарезервированные слова `begin` и `end`.

Пример:

```
program prog16;
uses crt;
```

```
var i: byte;
begin
  clrscr;
  i:=0;
  while i<10 do
  begin
    write(i);
    inc(i);
  end;
  readln;
end.
```

Аналогичные программы мы уже создавали с использованием операторов for и repeat.

► Попробуйте самостоятельно изменить текст программы prog16 так, чтобы цепочка цифр выводилась в обратной последовательности – от 9 до 0. (это программа p07.pas)◄

Все три оператора цикла взаимозаменяемы. Однако, тело цикла оператора `repeat` вне зависимости от проверяемого условия (в любом случае) выполнится хотя бы один раз, так как проверка условия производится в конце. В то время как тело цикла оператора `while` (а также и оператора `for`) при определенных условиях может и ни разу не выполнится, так как проверка условия производится в начале.

Операторные скобки не являются обязательным атрибутом оператора while и используются только когда тело цикла состоит из группы операторов. Рассмотрим ещё один пример:

```
program prog17;
uses crt;
const n=99;
var i: byte;
begin
  clrscr;
  randomize; {инициализация генератора случайных чисел}
  i:=1+random(n);
  while i mod 3 <> 0 do
    i:=1+random(n);
  writeln('i=',i);
  readln;
end.
```

Эта программа осуществляет генерацию целого числа в диапазоне от 1 до 99 кратного 3. Используемая в программе функция `random` возвращает значение целого случайного числа в диапазоне от 0 до величины параметра, указанного в скобках (не включая его). Например, `random(5)` может вернуть любое значение из списка – 0, 1, 2, 3, 4. Строка кода `i:=1+random(99);` как раз и обеспечивает генерацию числа в диапазоне от 1 до 99. Функция `random(99)`, конечно, возвращает значение в диапазоне от 0 до 98, но в строке предусмотрено добавление 1, чтобы избежать ситуации когда число будет равно 0. Это добавление 1 сдвигает диапазон с `0..98` на `1..99`. В качестве условия стоит выражение: `i mod 3 <> 0` – это проверка числа на кратность трем. Тело цикла состоит из одного оператора (`i:=1+random(99);`), поэтому использование операторных скобок (`begin end`) необязательно. Цикл будет выполняться до тех пор пока не будет получено целое число в диапазоне от 1 до 99 кратное 3.

► Разработайте игровую программу p08.pas. Компьютер загадывает случайное число в диапазоне от 0 до 999. Пользователь должен угадать это число за минимальное количество попыток. Диалог пользователя с компьютером организован так: пользователь вводит свой вариант числа, на что компьютер может ответить – «загаданное число больше (меньше, равно)». Ответ «равно» означает, что число угадано. Перед началом игры у игрока 1000$. За каждую попытку с игрока списывается 100$. ◄