*В. М. Гуровиц. Записки уроков по программированию для восьмых классов в ЦО 218 и в ФМШ 2007 г. Москвы, версия 18 сентября 2008 г.*

До сих пор все операторы в наших программах выполнялись последовательно, один за одним, сверху вниз. Но далеко не любую программу можно написать, используя лишь такой порядок выполнения операторов. Иногда какой-то оператор необходимо повторять несколько раз, другой нужно выполнять не всегда, а только при определенных условиях. Сейчас мы познакомимся с конструкциями языка Паскаль (впрочем, и в других языках есть такие же операторы), которые позволят нам управлять порядком выполнения других операторов.

## Условный оператор

Рассмотрим работу нового оператора сразу на примере:

```
if x>10 then
  write(x);
```

Переведем дословно:

```
если x>10 тогда
  напечатать x
```

Именно так это оператор и работает: проверяется условие `x>10`, и если оно выполняется, то печатается `x`.

Приведем другой пример:

```
if x=y then
  z:=1;
```

Прочитаем дословно:

```
если x=y тогда
  присвоить переменной z единицу
```

В общем виде условный оператор `if` можно записать так:

```
if условие then
  оператор;
```

Здесь в качестве оператора может выступать абсолютно любой оператор языка Паскаль (из тех что мы уже изучили и будем изучать далее), например:

```
read(password);
if password<>12345 then
  read(password);
```

В этом фрагменте пользователя просят ввести пароль, и если он отличается от правильного (12345), то требуется ввести его еще раз. Здесь `password` — это переменная, и, поскольку мы пока не знаем других переменных, кроме переменных типа `integer`, то вводить можно только числовые пароли. Вскоре мы избавимся от этого ограничения, изучив тип `string`.

А что делать, если перед тем, как ввести пароль, мы хотим сообщить пользователю, что он ошибся? Хотелось бы написать примерно такой код:

```
{НЕПРАВИЛЬНЫЙ КОД}
write('Enter password:');
read(password);
if password<>12345 then
  write('Wrong! Enter password again:');
  read(password);
```

Но в операторе `if` выполняется *только один оператор, следующий после* `then`. То есть в случае, если пароль не равен 12345, выполнится команда `write`. Следующая за ней команда `read` будет выполнена в любом случае, независимо от того, сработало ли условие в операторе `if`.

Для того, чтобы объяснить оператору `if`, что требуется выполнить несколько операторов в том случае, если условие верно, используются ключевые слова `begin ... end`:

```
if password<>12345 then begin
  write('Wrong! Enter password again:');
  read(password);
end;
```

В этом случае `begin` обозначает начало блока операторов, которые требуется выполнить внутри условного оператора, а `end` — конец этого блока. Обратите внимание, что после `end` ставится точка с запятой в отличие от слова `end` в конце программы, после которого ставится точка.

Для выполнения нескольких операторов внутри условного оператора используется конструкция `begin ... end`:

```
if условие then begin
  оператор1;
  оператор2;
  ...
end;
```

Оператор `if` имеет еще одну, более продвинутую форму:

```
if условие then
  оператор
else
  другой оператор;
```

что переводится дословно как

```
если верно условие, тогда
  выполнять оператор
иначе
  выполнять другой оператор;
```

Таким образом, первый оператор выполняется только если условие верно, второй же — наоборот: только если условие не верно. Например,

```
if x=y then
  write('equal')
else
  write('not equal');
```

Если значения переменных `x` и `y` равны, будет напечатано `equal` («равны»), в противном случае будет напечатано `not equal` («не равны»). Обратите внимание, что перед `else` точка с запятой не ставится, поскольку это продолжение оператора `if`.

В этой конструкции также можно использовать несколько операторов, окружая их ключевыми словами `begin ... end`, причем как в секции `then`, так и в секции `else`:

```
if условие then begin
  оператор1;
  оператор2;
  ...
end
else begin
  оператор3;
  оператор4;
  ...
end;
```

Перечислим теперь условия, которые можно использовать в операторе `if` (более подробно об этом читайте в главе, посвященной типу `boolean`).

Целые числа, с которыми мы работаем, можно сравнивать, используя следующие операции:

| | |
|---|---|
| $<$ | меньше |
| $>$ | больше |
| $=$ | равно |
| $<=$ | меньше либо равно |
| $>=$ | больше либо равно |
| $<>$ | не равно |

Обратите внимание, что в операциях, которые состоят из двух символов, этим символы нужно писать именно в таком порядке и нельзя ставить между ними пробел. В операциях $<=$ и $>=$ знаки ставятся в таком же порядке, как и произносятся: «меньше либо равно», «больше либо равно».

С помощью такого, казалось бы, небольшого набора операций можно записывать самые разнообразные условия. К примеру, давайте попробуем записать условие «`x` делится на 5». Для этого вспомним, что под термином «делится» в математическом жаргоне подразумевают «делится без остатка», то есть остаток равен нулю. Как только в определении появилось слово «равен», записать условие уже не составляет труда:

```
x mod 5 = 0
```

Попытаемся записать более сложное условие: «числа `x` и `y` одной четности», то есть оба числа либо четные, либо нечетные. Иначе говоря, они дают равные остатки при делении на 2:

```
x mod 2 = y mod 2
```

Это же условие можно записать и по другому, если вспомнить, что сумма двух чисел является четной только в том случае, когда оба числа одной четности:

```
(x + y) mod 2 = 0
```

Обратите внимание: скобки в этом примере обязательны, поскольку операция `mod` выполняется до операции сложения (говорят, что она имеет больший приоритет).

## Вложенные условные операторы

В операторе `if` можно использовать не только операторы ввода-вывода и присваивания, но и любой другой оператор, в частности, другой оператор `if`:

```
if x<1000 then
  if x>0 then
    write('Your password accepted')
  else
    write('Enter positive number');
else
  write('Enter number less than 1000');
```

Данная программа проверяет введенное вами число-пароль, и если оно оказывается в пределах от 1 до 999 — принимает его, если введено число не больше 0 — просит ввести положительное число, а если введено наоборот слишком большое число — просит ввести число меньше 1000. Обратите внимание: `else` относится к ближайшему `if` над ним, поэтому верхний `else` относится ко второму `if`'у, а нижний — к первому.

## О пробелах, концах строк и стиле программирования

Вы наверное заметили, что в наших примерах в начале некоторых строк мы делали отступы:

```
if x=y then
  write('equal')
else
  write('not equal');
```

Хотя ничего не мешало нам написать так:

```
if x=y then
write('equal')
else
write('not equal');
```

Или даже так:

```
if x=y then write('equal') else write('not equal');
```

Паскаль не обращает внимания на лишние переводы строк, а также не принимает в расчет лишние пробелы. Обязательные пробелы ставятся лишь для отделения одного элемента от другого, например, нельзя не ставить пробел между `if` и `x=y`. При этом между `x` и `=` пробел можно и не ставить: компилятор сам определит, где закончилось имя переменной и началась операция сравнения.

Тем не менее, очень важно правильно расставлять переносы строк и отступы, чтобы сделать программу читабельной для человека. Ведь большую часть времени программист тратит не на написание кода, а на его отладку (поиск и устранение ошибок), улучшение и модернизацию (добавление новых возможностей).

Поэтому обычно при написании программ придерживаются следующих соглашений.

1. Каждый оператор пишется на отдельной строке. Это позволяет при пошаговой отладке легко понять, какие операторы в какой последовательности выполняются.

2. Если несколько операторов выполняются друг за другом, то они пишутся с одинаковыми отступами:

   ```
   read(x);
   y:=2*x;
   write(y);
   ```

3. Если один оператор выполняется внутри другого, то он пишется с бóльшим отступом:

   ```
   if x>5 then begin
     y:=x-5;
     write(y);
   end
   else
     write(x);
   ```

   При такой расстановке отступов сразу видно, какие операторы выполняются, если `x>5`, а какие — в противном случае.

## Упражнения

Напишите на языке паскаль следующие условия.

1. `x` — четное.
2. `x` делится на `y`.
3. Последняя цифра числа `x` нечетная.
4. Среди чисел `x`, `y`, `z` есть четное число.
5. Числа `x`, `y`, `z` все нечетные.
6. Числа `x`, `y`, `z` все четные.

## Практическое задание

1. Напишите программу, которая выводит большее из а) двух; б) трех введенных чисел.
2. Напишите программу, которая по координатам (номеру столбца и номеру строки) двух клеток шахматной доски определяет, одного они цвета или разного.
3. В час пик на остановку одновременно подъехали три маршрутных такси, следующие по одному маршруту, в которые тут же набились пассажиры. Водители обнаружили, что количество людей в разных маршрутках разное, и решили пересадить часть пассажиров так, чтобы в каждой маршрутке было поровну пассажиров. Требуется по количествам людей в трех маршрутках определить, какое наименьшее количество пассажиров придется при этом пересадить (если это вообще возможно).

## Ответы к упражнениям

1. `x mod 2 = 0`
2. `x mod y = 0`
3. Заметим, что это условие равносильно более простому: `x` — нечетное число. Поэтому достаточно написать `x mod 2 = 0` (хотя и выражение `x mod 10 mod 2 = 0` приведет к тому же результату).
4. Это условие выполняется тогда и только тогда, когда произведение чисел `x`, `y`, `z` четно: `x*y*z mod 2 = 0`
5. А здесь, наоборот, произведение чисел должно быть нечетным: `x*y*z mod 2 = 1`.
6. Это упражнение, как ни странно, сложнее остальных. Неверным будет написать, что произведение четно: это означает лишь, что хотя бы одно из чисел четное. Также неверно пытаться проверить, что произведение делится на `2*2*2=8`: может оказаться, что одно из чисел делится на 8, а остальные — нет. Можно предложить разные способы решения данного упражнения. Например, можно воспользоваться тем, что числа `x`, `y`, `z` четные тогда и только тогда, когда числа `(x+1)`, `(y+1)` и `(z+1)` нечетные и свести задачу к предыдущей: `(x+1)*(y+1)*(z+1) mod 2 = 1`.