Простые и составные операторы

- Если ветвь условной конструкции или тело цикла содержит несколько операторов, то они объединяются в составной оператор при помощи “{“ и “}”.
- Begin в Паскале соответствует “{“ в Си, End - “}”.

Пример. Нахождение суммы первых 20 натуральных чисел и вывод сумм на экран.

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{int sum=0; //сумма
int c=0; //счетчик

while (c++<20) //увеличение после сравнения
{
   sum=sum+c;   // или sum+=c
   printf ("sum=%d\n", sum);
}
return 0;
}
```

Условная конструкция.

Различают три формы условной конструкции :

1. простейшая (в Паскале - if … then…);
2. расширенная (в Паскале - if … then … else …);
3. выбор (в Паскале - case … of…).

Простейшая условная конструкция.

Псевдокод :

если <условие> то

<оператор>

все

Си

if (<выражение>)

<оператор>

В отличие от Паскаля выражение может иметь любой тип. 0, ‘\0’ и NULL считаются ложью, остальные значения - истиной.

Расширенная условная конструкция

Псевдокод :

если <условие> то
    <оператор1>
иначе
    <оператор2>
все

Си

```
if(<выражение>)
    <оператор1>
else <оператор2>
```

Примеры.

```
if(a>b&&b>c)
   f=x*x-1;
else
   f=x+1;
```

‘;’ перед **else** является частью оператора присваивания в ветви **if**.

```
if(x>5||x<0)
{
   k++;
   y[k]=x;
}
else
{
   n++;
   z[n]=x;
}
```

Отличия от Паскаля:

1. Выражение обязательно заключается в скобки.
2. ”;” ставится перед **else**, если только нет составного оператора. (“;”- часть <оператора1>).
3. Другой приоритет вычисления логического выражения

- сначала операции отношения
- затем логические операции !, &&, || (не, и, или).

Приоритет операций отношения меньше, чем у операций “+” и “-” и больше, чем у операции присваивания:

x>y+2 то же, что и x>(y+2).

Допускается вложенность операторов **if**. Если нет составного оператора, **else** относится к ближайшему **if**.

Рассмотрим пример.

```
if (number>6)
  if (number<12)
    printf ("Конец игры!\n");
  else
    printf ("Потеря хода!\n");
```

Число	**Результат**
5	«нет результата»
10	Конец игры
15	Потеря хода

Если необходимо, чтобы **else** соответствовал первому **if**, добавим { }.

Рассмотрим пример.

```
if (number>6)
  {
  if (number<12)
    printf ("Конец игры!\n");
  }
else
  printf ("Потеря хода!\n");
```

Число	**Результат**
5	Потеря хода
10	Конец игры
15	«нет результата»

Операция условия - сокращённый способ записи if- else (тернарная).

В общем виде условное выражение записывается следующим образом:

(<*условие*>**)?**<*значение если истинно*>**:**<*значение если ложно*>**;**

Например:

```
x=(y<0)?-y:y;
```

эквивалентно:

```
if (y<0)
   x=-y;
else
   x=y;
```

Условное выражение удобно использовать в тех случаях, когда некоторой переменной надо присвоить одно из двух возможных значений, например:

```
max = (a > b)?a:b;
```

Множественный выбор: switch и break

```
switch (<выражение>)
    /*выражение может быть типа int
    или char*/
{
case <константа 1>: <операторы
    1> //операторы могут
    отсутствовать
case <константа 2>: <операторы
    2> //константы типа int или char
      …
default: <операторы>
    //ветвь не обязательна
}
```

Если значение выражения совпадает с одной из констант, выполняются операторы, расположенные после соответствующей константы.

Если подходящей метки не найдется, то, если существует строка с меткой "default", будет выполняться оператор, помеченный этой меткой. В противном случае произойдет переход к оператору, расположенно-му за

В каждой последовательности операторов последним должен быть оператор break.

Выполнение оператора break осуществляет выход из оператора switch и переход к следующему за ним оператору.

При отсутствии оператора break будут выполнены все операторы, начиная с помеченного данной меткой и заканчивая оператором default.

```
char ch='1';
switch (ch)
  {
  case '1': printf ("один\n");
  case '2': printf ("два \n");
  default : printf ("три \n");
  }
```

```
один
два
три
```

В качестве меток-констант используются выражения типа int или char.

В качестве метки запрещается использовать переменную.

Можно пометить оператор несколькими метками одновременно. Например,

```
case 'E':
case 'e':printf ("ель\n"); break;
```

Пример. Игра в города

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{setlocale(LC_ALL, "RUS");
char ch;

printf (“Введите букву а, б
    или в. \n");
printf ("Я назову город на
    эту букву\n");
scanf("%c", &ch);
switch (ch)
{case 'а': printf ("\n Ашхабад");
    break;
case ‘б': printf ("\n Белгород");
    break;
case 'в': printf ("\n Воронеж");
    break;
default: printf ("\n неизвестная
    буква"); break;
}
return 0;
}
```

Правила выбора условных операторов

1. Выбор из двух возможностей - выполнить оператор или пропустить его - оператор **if**.
2. Выбор одного из двух вариантов - **if**...**else**.
3. Выбор одного из нескольких - **else-if**, **switch**.

Если выбор вариантов основывается на вычислении значения переменной или выражения типа **float**, то **switch** применить **нельзя.**

Если значения переменной попадают в некоторый диапазон, использовать **switch** неудобно. Лучше записать:

```
if (i <1000 && i>2).
```

Циклы

Существует три базовых структуры цикла:

- цикл-пока (с предусловием)
  (в Паскале - **while** … **do**…);
- цикл-до (с постусловием)
  (в Паскале – **repeat** …**until** …);
- цикл от … до (со счетчиком)
  (в Паскале - **for** … **to**/**downto do** …).

Цикл-пока **(while)**

Псевдокод

цикл-пока <условие>

    <действия>

кц

Си

while (<выражение>)

    <операторы>

, где

<выражение> - любого типа.

Пока значение выражения отлично от 0 (т.е. истинно), повторяется выполнение операторов в теле цикла.

Оператор может быть простым или составным.

while является циклом с предусловием, поэтому возможно, что он не выполнится ни разу.

Пример 1. Напечатать целые числа от 1 до 100 и их квадраты.

```
/*Здесь и в следующих примерах пропущены операторы
#include <iostream>
#include <stdio.h>
using namespace std;
*/


int main()
{
setlocale(LC_ALL,"RUS");
int n=0;
printf("число  квадрат");
while(n++<100)
   printf("%6d %6d\n",n,n*n);
return  0;
}
```

Пример 2. Сколько членов гармонического ряда S=1+1/2+1/3+...+1/n надо взять, чтобы получить сумму, большую числа dano?

```
int main()
{
int i=0;
float dano, S=0.0;

printf ("введите число"); scanf("%f",&dano);
while (dano<=0.0)
   {
   printf("повторите ввод");
   scanf ("%f", &dano);
   }
while (S<=dano)
   S+=1.0/(float)(++i);
printf (“число членов ряда=%d\n",i);
return 0;
}
```

Цикл for

Псевдокод

цикл от i:=<н.з.> до <к.з.> [шаг <приращение>]
    <действия>
кц

Си

```
for (<выражение 1>; <выражение 2>; <выражение 3>)
    <оператор>
```

,где

- **<выражение 1>** – инициализирующее: вычисляется один раз до начала цикла;
- **<выражение 2>** – проверяемое: вычисляется перед каждым выполнением оператора. Тело цикла выполняется, если значение проверяемого выражения истина (или не равно нулю);
- **<выражение 3>** – корректирующее: вычисляется после каждого выполнения тела цикла.

Перед первым выполнением цикла проверяется <выражение 2>, т.е. тело цикла может не выполниться ни разу!

Любое выражение, а также оператор может отсутствовать, но точка с запятой сохраняется.

Оператор **for** эквивалентен следующей последовательности операторов:

```
<выражение 1>;
while(<выражение 2>)
   {
   <оператор>
   <выражение 3>;
   }
```

Пример 1. Найти сумму n членов гармонического ряда S=1+1/2+1/3+...+1/n.

a)
```
s=0.0;
for(i=1;i<=n;i++)
S=S+1.0/(float)i;
```

b)
```
for (i=1, S=0.0; i<=n; i++)
S+=1.0/(float)i;
```

Операция «запятая» связывает два выражения в одно и гарантирует, что самое левое будет вычисляться первым. Обычно используется для включения дополнительной информации в спецификацию цикла **for**, например:

<выражение 1>, < выражение 2>

Значением всего выражения является значение <выражения 2> .

c) `for (i=1, S=0.0; i<=n; S+=1.0/(float)i++);`
`//Здесь пустой цикл.`

Это не очень хороший стиль программирования - лучше не смешивать процесс изменения переменной цикла с алгебраическими вычислениями.

Пример 2. Найти минимальное n, при котором S=1+1/2+1/3+...+1/n > dano.

```
for (n=0, S=0.0; S<=dano; S+=1.0/++n);
```

- в операторе **for** проверяемое выражение не обязательно использует параметр цикла.

Пример 3. Счет в порядке убывания

```
for(n=10;n>0;n--)
   printf ("%d секунд !\n", n);
printf("пуск!\n");
```

Можно использовать любое значение шага цикла, например:

```
for(n=2; n<60;n+=13)
   printf("%d\n",n);
```

На экране получим 2 15 28 41 54.

Пример 4. Переменная цикла может быть не только числом, но и символом.

```
for(ch='a';ch<='z';ch++)
printf ("Величина кода для %c = %d\n", ch, ch);
```

При выполнении этого оператора будут выведены на печать все буквы от a до z и их коды ASCII.

Пример 5. Можно пропустить одно или более выражений, но при этом нельзя пропустить символ “;”.

1. `ans=2;  for (n=3; ans<=25;) ans=ans*n;`
2. `for(;;) printf ("работаю\n");`

Тело этого цикла будет выполняться бесконечное число раз, поскольку пустое условие всегда считается истинным.

Следующие записи эквивалентны:

while (<выражение>) <оператор>

и

for (; <выражение> ;) <оператор>

Применение операторов **for** или **while** - дело вкуса. Цикл **for** предпочтительнее, когда в цикле используется инициализация и коррекция переменной, в остальных случаях лучше использовать **while**.

Возможно изменение параметров, входящих в проверяемое и корректирующее выражения, в теле

Пример 6. Вложенные циклы.

Вычислить все совершенные числа, меньшие или равные заданному числу smax.

Совершенное число равно сумме своих делителей, исключая делитель, равный самому числу.

6=1+2+3.

```
int  main()
{int n, sum, del, smax;

printf ("введите предел");
scanf ("%d", &smax);
printf ("совершенное число\n");
for(n=6; n<=smax; n++)
   {
   sum= 1;
   for(del=2; del<n; del++)
      if(n%del==0) sum+=del;
   if (sum==n) printf(“%5d”,n);
   }
return 0;
}
```

Цикл с постусловием **do…while**

Псевдокод

цикл

    <действия>

до <условие>

кц

Си

```
do
    <оператор>
while(<выражение>)
```

Тело цикла всегда выполняется хотя бы один раз. Выполнение цикла продолжается до тех пор, пока выражение не станет ложным (равным 0).

В Паскале - наоборот, цикл продолжается пока выражение не станет истинным.

Пример.Найти минимальное число членов гармонического ряда с S > dano

```
i = 0; S=0.0;
do
    S+=1.0/(float)++i;
while (S< = dano);
```

При помощи этого цикла можно организовать ввод данных с проверкой их правильности, например.

```
do {
    printf ("введите положительное число");
    scanf("%f”, &dano);
    }
while (dano < =0);
```

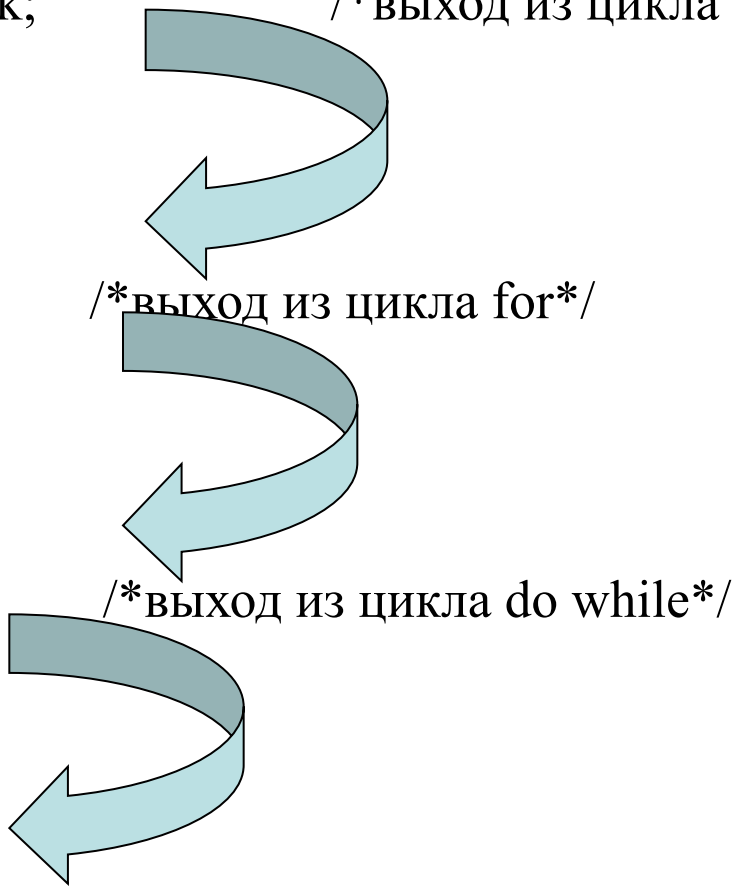
Управляющие операторы break, continue, goto

По возможности следует избегать их использования. Все эти операторы предназначены для безусловного перехода.

Частое применение этих операторов - признак низкой квалификации программиста и слабого владения структурным программированием.

break - выход из ближайшего цикла любого вида или switch и переход к следующему оператору программы.

```
do {…
      for(...)
    {…  while( … )
              { …
              if (…) break;                  /*выход из цикла while*/
                 …
              }
    <оператор 1>
    if(…) break;                 /*выход из цикла for*/
    …
    }
    <оператор 2>;
    if (…) break;                /*выход из цикла do while*/
    …
   }      while(…);
<оператор 3 >
```

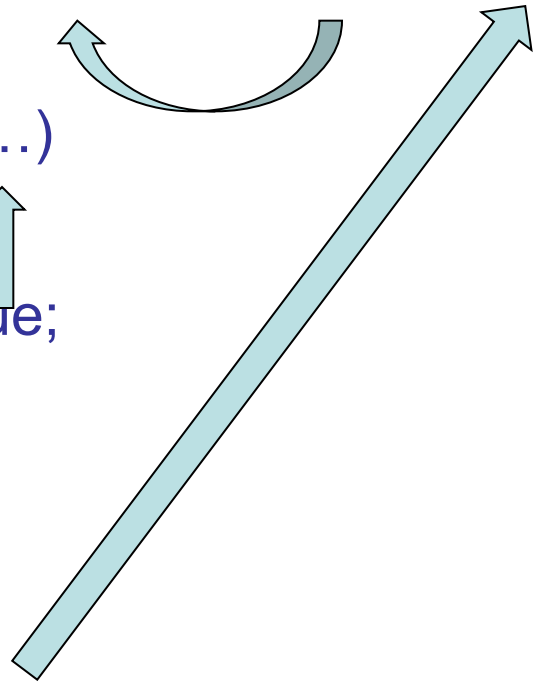


continue - окончание текущей итерации данного цикла.

В циклах **while** и **do** … **while** происходит переход к проверке условия продолжения цикла.

В цикле **for** – переход к вычислению корректирующего выражения, а затем к проверке.

```
for (<выражение 1>;
< выражение2>; <выражение3>)
{
    while(…)
    {…
    continue;
    …
    }
…
continue;
}
```



Оператор goto

goto метка;

Метка должна быть идентификатором, например: goto m1;

Допускается использовать в одном случае - выход из вложенного набора циклов при обнаружении каких-либо ошибок.

```
for (…)
    for (…)
     for (…)
     {
    if (<ошибка>)
                    goto Code;
    <операторы>
     }
                    …

Code: <операторы для
    исправления ошибки>
```