



Лабораторная работа № 2

Программы с простейшей структурой

Терентьев Александр Николаевич

Национальный Технический Университет Украины
“КПИ”, Институт Прикладного Системного Анализа

Киев 2007

Цель работы:

Получение навыков работы с основными
управляющими конструкциями

План проведения занятия:

1. Ознакомление с теорией слайды 3-25.
2. Выполнение практического задания в среде программирования Turbo C++ слайды 26-29.

Управляющие конструкции языка С:

0. Фигурные скобки

1. конструкции ветвления:

1.1 if-else

1.2 if-else if

2. ЦИКЛЫ

2.1 Цикл for

2.2 Цикл while

2.3 Цикл do-while

Управляющие конструкции языка C:

3. Выход из цикла **break**

4. Оператор продолжения **continue**

5. Оператор перехода на метку **goto**

6. Оператор **switch**

7. Оператор возвращения **return**

0. Фигурные скобки

Фигурные скобки позволяют объединить несколько элементарных операторов в один составной оператор, или блок. Во всех синтаксических конструкциях составной оператор можно использовать вместо простого.

В C++ в начало блока можно помещать описания локальных переменных. **Локальные переменные, описанные внутри блока, создаются при входе в блок и уничтожаются при выходе из него.**

В C++ локальные переменные можно описывать где **угодно**, а не только в начале блока.

Приведем программу, обменивающую значения двух целых переменных.

Чтобы обменять значения двух переменных x и y , мы сначала запоминаем значение x во вспомогательной переменной $temp$. Затем в x записывается значение y , а в y - сохраненное в $temp$ предыдущее значение x . Поскольку переменная $temp$ нужна только внутри этого фрагмента, мы заключили его в блок и описали переменную $temp$ внутри этого блока. По выходу из блока память, занятая переменной $temp$, будет освобождена.

Пример:

```
#include<conio.h>
#include<stdio.h>
void main()
{ int x, y;
  clrscr();
  x=10;
  y=20;
  {
    int temp;
    temp=x;
    x=y;
    y=temp;
  }
  getch();
}
```

1.1 if-else

Оператор `if` ("если") позволяет организовать ветвление в программе. Он имеет две формы: оператор "если" и оператор "если...иначе".

оператор
"если"

ИМЕЕТ ВИД:

if (условие)
 действие;

оператор "если...
иначе"

ИМЕЕТ ВИД:

if (условие)
 действие1;
else
 действие2;

1.2 Выбор из нескольких возможностей

if-else if

Несколько условных операторов типа "если...иначе" можно записывать последовательно, то есть действие после else может снова представлять собой условный оператор.

В результате реализуется выбор из нескольких возможностей.

Синтаксис:

```
If (выражение 1)  
    оператор 1  
Else  
    if (выражение 2)  
        оператор 2  
    else  
        оператор 3
```

2.1 Цикл `for`

for (*выражение1*; *выражение2*; *выражение3*)
 {*оператор*}

выражение1 - описывает инициализацию цикла;
выражение2 - проверяет условие завершения цикла, если он истинный то выполняется оператор;
выражение3 – итератор, который вычисляется после каждой итерации.

Цикл повторяется до тех пор, пока *выражение* не станет ошибочным.

Пример:

```
#include <conio.h>
#include <stdio.h>
void main()
{
    int x;
    clrscr();
    for(x=1;x<=10;x++)
    {
        printf(“%i \n”,x);
    }
    getch();
}
```

2.2 Цикл **while**

while (выражение)
оператор

Если *выражение* является истинным, то *оператор* выполняется до тех пор, пока выражение не станет ошибочным. Если *выражение* ошибочное с самого начала, то управление передается следующему оператору. При этом цикл не выполняется совсем. Значение *выражения* определяется до выполнения оператора.

Пример:

```
#include<conio.h>
#include<stdio.h>
void main()
{
    int x;
    clrscr();
    x=1;
    while(x<=10)
    { printf(“%i \n”,x);
      x++; }
    getch();
}
```

2.3 Цикл **do-while**

do

оператор

while (выражение);

Если *выражение* истинное, то оператор *выполняется* и снова вычисляется значение *выражения*. Это повторяется, пока *выражение* не станет ошибочным.

Оператор выполняется не меньше одного раза .

Очень часто после

while (выражение)

забывают ставить точку с запятой.

Пример:

```
#include<conio.h>
```

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
int x;
```

```
clrscr();
```

```
x=1;
```

```
do {
```

```
printf(“%i \n”,x);
```

```
x++;
```

```
} while(x<=10);
```

```
getch();
```

```
}
```

Замечание о while и do-while

Оператор do-while проверяет условие после выполнения операторов цикла.

Оператор while проверяет условие перед выполнением операторов цикла.

3 Выход из цикла **break**

break;

Оператор *break* прерывает выполнение ближайшего вложенного внешнего оператора *switch*, *while*, *do* или *for*. Управление передается следующему оператору. Например в примере выход из бесконечного цикла "*while (true)*" осуществляется с помощью оператора "*break*".

Пример:

```
#include <conio.h>
void main()
{   int x, v;
    x=0; v=0;
    while (v==0)
        {   if x==10
            {   break;
                // условие выхода из
                // программы при x=10
            }
            else { x++; }
        }
    getch();
}
```

4 Оператор продолжения **continue**

continue;

Иногда требуется пропустить выполнение тела цикла при каких-либо значениях изменяющихся в цикле переменных, переходя к следующему набору значений и очередной итерации. Для этого используется оператор продолжения *continue*, который передает управление на начало ближайшего внешнего оператора цикла *switch*, *while*, *do* или *for*, и вызывает начало следующей итерации.

Оператор *continue*, так же, как и *break*, используется лишь в том случае, когда тело цикла состоит более чем из одного оператора и заключено в фигурные скобки. Его следует понимать как переход на фигурную скобку, закрывающую тело цикла.

Пример: пусть задано n натуральных чисел $i = 1, \dots, n$ и требуется найти их полную сумму за исключением $i=k$, то есть $L=1+2+\dots+(k-1)+(k+1)+\dots+n$

```
void main()  
{ int i, n, k, L;  
  i=1;  
  n=10;  
  k=7;  
  L=0;  
  while (i<=n)  
    { if x==k  
      { i++;  
        continue;  
        // Пропустить k-й член  
      }  
      L=L+i;  
      i++;  
    }  
}
```

5. Оператор перехода на метку **goto**

goto (метка);

Оператор перехода *goto* позволяет изменить естественный порядок выполнения программы и осуществить переход на другой участок программы, обозначенный *меткой*.

Переход может осуществляться только внутри функции, т. е. оператор *goto* не может ни выйти из функции, ни войти внутрь другой функции.

В качестве *метки* можно использовать любое имя, допустимое в Си (т.е. последовательность букв, цифр и знаков подчеркивания "_", начинающуюся не с цифры).

goto

Метка может стоять до или после оператора *goto*. *Метка* выделяется символом двоеточия ":". Лучше после него сразу ставить точку с запятой ";", помечая таким образом пустой оператор - это общепринятая программистская практика, согласно которой *метки* ставятся между операторами, а не на операторах.

Не следует увлекаться использованием оператора *goto* - это всегда запутывает программу. Большинство программистов считают применение оператора *goto* дурным стилем программирования. Вместо *goto* при необходимости можно использовать операторы выхода из цикла *break* и пропуска итерации цикла *continue*.

```
#include<conio.h>
void main()
{
int i,j;
i=1;
while(i<10)
{
j=1;
while(j<20)
{
if (i+j==25)
{ goto metka_1; }
j++;
}
i++;
}
metka_1: ;
getch();
}
```

goto

Единственная ситуация, в которой использование *goto* оправдано, - это выход из нескольких вложенных друг в друга циклов.

6. Оператор `switch`

switch (выражение)

```
{  
  case константа1: оператор  
  case константа2: оператор  
  case константа3: оператор  
  ...  
  default: оператор  
}
```

1. сначала вычисляется значение выражения в заголовке *switch*;
2. затем осуществляется переход на метку "*case константа L:*", где *константа L* совпадает с вычисленным значением *выражения* в заголовке;
3. если такого значения нет среди меток внутри тела *switch*, то
 - 3.1 если есть метка "*default:*", то осуществляется переход на нее;
 - 3.2 если метка "*default:*" отсутствует, то ничего не происходит.

Например, при выполнении фрагмента программы

```
int n, k;  
n = 2;  
switch (n)  
  {  
    case 1: k = 2;  
    case 2: k = 4;  
    case 3: k = 8;  
  }
```

переменной *k* будет присвоено значение 8, а не 4. Дело в том, что при переходе на метку "case 2:" будут выполнены сначала строка

k = 4;

и затем строка

k = 8;

что делает приведенный фрагмент совершенно бессмысленным, так как оптимизирующий компилятор вообще исключит строки " $k = 2;$ " и " $k = 4;$ " из кода готовой программы!!! Чтобы исправить этот фрагмент, следует использовать оператор *break*;

Так же, как и в случае цикла, оператор *break* приводит к выходу из фигурных скобок, обрамляющих тело оператора *switch*.

Приведенный фрагмент надо переписать следующим образом:

```
int n, k;  
n = 2;  
  
switch (n)  
{  
    case 1:  
        k = 2;  
        break;  
    case 2:  
        k = 4;  
        break;  
    case 3:  
        k = 8;  
        break;  
}
```

В результате выполнения этого фрагмента переменной k будет присвоено значение 4. Если бы значение n равнялось 1, то k было бы присвоено значение 2, если n равнялось бы 3, то 8. Если n не равно ни 1, ни 2, ни 3, то ничего не происходит. Оператор *switch* иногда совершенно необоснованно называют оператором выбора. На самом деле, для выбора следует использовать конструкцию *if – else – if*. Например, приведенный фрагмент лучше реализовать следующим образом:

```
if (n == 1)
    { k = 2; }
else if (n == 2)
    { k = 4; }
else if (n == 3)
    { k = 8; }
```

Замечание о *switch*

Оператор *switch* по сути своей является оператором перехода *goto* с вычисляемой меткой. Ему присущи многие недостатки *goto*, например, проблемы с инициализацией локальных переменных при входе в блок.

Кроме того, *switch* не позволяет записывать условия в виде логических выражений, что ограничивает сферу его применения.

Рекомендуется никогда не использовать оператор *switch*, так как выбор в стиле *if...else if...* во всех отношениях лучше!

7. Оператор возвращения **return**

return (выражение)

Перерывает выполнение текущей функции и возвращает управление программе, которая ее вызвала.

Пример функции вычисления квадрата:

```
double sqr(double x)  
{  
  return (x*x);  
}
```

Практическая часть

Постановка задачи:

Напишите программу которая выполняет нахождение всех простых несократимых дробей между 0 и 1, знаменатели которых не превышают 7. Дробь задаётся двумя натуральными числами – числителем и знаменателем.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
    int i, j, m, k, i1, j1, k2;
    clrscr();
    printf("Program beginning \n \n");
    /* tast of cycles */
    for (i1=2;i1<8;++i1)
        for (j1=1;j1<i1;++j1)
            {
                /* printing of fraction of kind 1/n */
                /* because all fractions of kind 1/n are irreducible */
                if (j1==1) printf("%d : %d \n",j1,i1);
                m=0; i=i1; j=j1;
```

Код
программы

<i>do</i>	<pre> { k=i-j; k2=i1%j1; if(((k==0)&&(j!=1)) (k2==0)) m=1; i=j; j=k; } while(k>0); /* printing of fraction if flag m<>1 */ if (m!=1) printf("%d : %d \n",j1,i1); } getch(); } </pre>	<p>Код программы продолжение</p>
-----------	---	--

Program beginning

1 : 2

1 : 3

2 : 3

1 : 4

3 : 4

1 : 5

2 : 5

3 : 5

4 : 5

1 : 6

5 : 6

1 : 7

2 : 7

3 : 7

4 : 7

5 : 7

6 : 7

Результат
выполнения
программы.

Замечание:

% - оператор нахождения остатка от деления.

Синтаксис:

*Число_1 % Число_2 = Остаток от
деления (Числа_1 на Число_2)*

Порядок выполнения лабораторной работы:

1. Получить задание
2. Проанализировать условие задачи
3. Разработать алгоритм и написать программу решения задачи в соответствии с номером полученного задания.
4. Оформить результаты работы протоколом

Контрольные вопросы

1. Что такое сложная конструкция?
2. Цикл с пред-условием и с пост-условием. Их синтаксис?
3. Что такое пошаговый цикл?
4. Когда используется переключатель?
Достоинства и недостатки?
5. Назначение и виды управляющих конструкций? Конкретизировать примерами.