STEP computer ACADEMY

PROGRAMMING C

Lesson No. 3

Programming C

Contents

1. Concept of a loop

Very often both in real life and while writing a program we need to repeat an action several times. For example, let's assume an algorithm realizing dish washing.

```
0. Take a plate out of the sink.

1. Soap a plate using dishwashing liquid.

2. Rub the plate with a washup.

3. Wash off soap foam from the plate.

4. Dry up the plate.

5. Put the plate on a shelf.

6. End of the program.
```

In this seemingly good algorithm there is one tiny — if there more than one plate in the result only one will be washed. This is because the program executes all the actions by linear method — upside down in sequence. Consequently we need to think up how to make a program repeat a set of particular actions, and also define a necessary number of повторов. Correct algorithm will look like that.

```
0.Take a plate out of the sink.
1. Soap a plate using dishwashing liquid.
2. Rub the plate with a washup.
3. Wash off soap foam from the plate.
4. Dry up the plate.
5. Put the plate on a shelf.
6. If there are any other dirty plates return to step 0.
7. End of the program.
```

Lat's pay attention to the fact that in order to define whether to repeat actions at first we use a condition «If there are dirty plates». If the condition is true — actions repeat, if false then the 7th step of the algorithm is executed.

Thus, we have concluded that we need a construction comprising a set of actions for repetition. Wherein the number of repetitions should depend on a condition that is within this construction.

Unwillingly we have just given a definition of a so called LOOP. Let's revise again!!!

Loop is a special programming language operator with the help thereof one or another action can be executed a necessary number of times depending on a condition.

Note: By the way — another name of a loop is — repetition construction. And each action repetition is — LOOP STEP or ITERATION.

In language C there are several realizations of such a form as loop. In the present lesson we will talk about two realizations — **while** and do **while**.

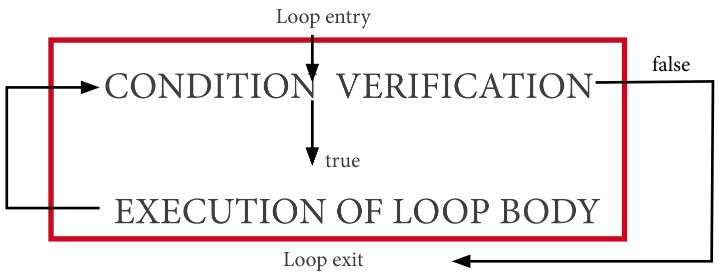
2. Loop while

General syntax and execution order of loop while.

```
while(expression)
{
                              action for repetition;
}
```

1. First of all the first expression is checked.
2. If an assertion in parentheses is true an action within the curly braces is executed.
3. If an assertion within the parentheses is false, a program will shift to the next string behind the closing curly brace of a loop.
4. If an assertion in in parentheses was true and action was executed, and then an assertion is checked.

As you can see expression check repeats upon each cycle execution. As soon as it ceases to be true, a loop stops. Pay attention that if an expression is false from the very beginning, the action inside the loop will not be executed at all.



Review an example

Assume that a person needs to write an assay about 7 Wonders of the World. Before doing that he has to go and see each of the wonders with his own eyes and only afterwards to write about them. Project name **Miracles**.

```
#include <iostream>
using namespace std;
void main()
{
)//declaration of control variable
 int counter=0;
while(counter<7))// assertion verification
              {
                  counter++;// change of control variable

                  //action for repetition

                  // you saw ... Wonder of the World
                  cout<<"You seen»<<counter<<" miracle of world!!!\n";

              }
cout<<"Now, you can start your work.\n";
}
```

Now we have to analyze how the example works.

1. We declare a variable initially equal to 0
2. Afterwards we check a variable value within cycle condition. So far as this is the value an execution of the loop depends on, and such variable is called a control variable.
3. We increase a variable value by one.

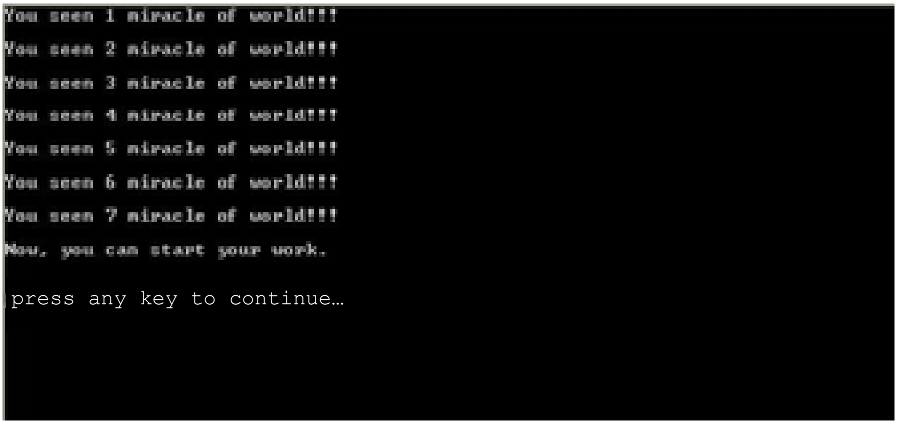
Note: This action is necessary because if you do not change the value controlling the loop, the assertion verification outcome also will never change. This can lead to a very common error called — the eternal loop. If loop assertion is true, and a control variable always has the same value, consequently the statement is always true. Imagine dirty dishes never end — their number is always constant. How long will the dishwasher last long?! Not for long, right? The program also cannot withstand such an onslaught, and some time after the start of the eternal loop it will generate an error at the execution time. To avoid such errors, you need to watch closely that a control variable is constantly changing within the loop body.

4. Afterwards we display on a screen a current value of our variable in a form of the message about a number of reviewed Wonder of the World.

5. Then we return to the condition and check the value of control variable.

Loop will continue its operation until a variable value is equal to 7. In this case a string «You have seen 7 Wonders of World!!!» will be displayed on a screen, and then the program returns to condition check. 7<7 — is false. Программа больше в цикл не войдет и перейдет к строке «Now you can start your work».

During the program run we will see the following picture on a screen:



```
You seen 1 miracle of world!!!
You seen 2 miracle of world!!!
You seen 3 miracle of world!!!
You seen 4 miracle of world!!!
You seen 5 miracle of world!!!
You seen 6 miracle of world!!!
You seen 7 miracle of world!!!
Now, you can start your work.

press any key to continue…
```

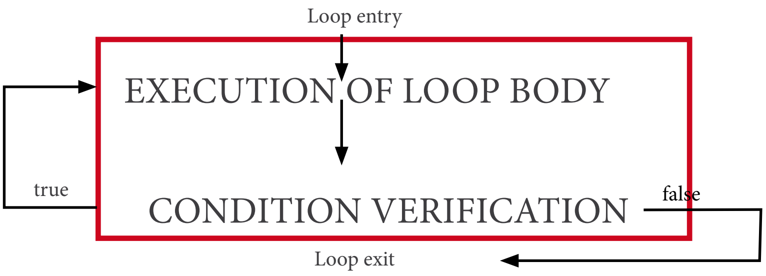
Now we have acquainted with one of the language C loop types. We hope it was not difficult for you. The next lesson unit will acquaint you with the loop which is an alternative to construction while.

3. Construction do while

General syntax and principle of operation of do while:

```
do
{
                        action;
}
while(condition);
```

Loop do while is similar with the loop while. The difference is that within the while a condition is checked right after loop entry and then if the condition is true an action is executed. Within do while in any case an action is executed in the first place and only then there is a condition check. If a condition is true an action is kept on executing and if no its execution passes to the next to while operator. In other words, if compared to «while» within «do while» an action is executed at least once. Let's review it on the scheme:



Practical application of do while

Assume we have to write a program where a user has a choice of any action to be repeated several times in succession. We realize this task at first using while, and then using do while. Project name is Calc.

```
#include <iostream>
using namespace std;
void main()
{
int answer,A,B,RES;

// request for operation choice
cout<<»\nSelect operation:\n»;
cout<<»\n 1 - if you want to see SUM.\n»;
cout<<»\n 2 - if you want to see DIFFERENCE.\n»;
cout<<»\n 3 - if you want to exit.\n»;
cin>>answer;

while(answer!=3){ // check of condition
        switch(answer){
                 case  1:    // if user chooses adding
                 cout<<»Enter first digit:\n»;
                 cin>>A;
                 cout<<»Enter second digit:\n»;
                 cin>>B;
                 RES=A+B;
                 cout<<»\nAnswer: «<<RES<<»\n»;
                 break; // termination of switch
                 case  2:     // if user chooses deducting
                 cout<<»Enter first digit:\n»;
                 cin>>A;
                 cout<<»Enter second digit:\n»;
                 cin>>B;
                 RES=A-B;
                 cout<<»\nAnswer: «<<RES<<»\n»;
                 break; // termination of switch
        case  3: // if user chooses exit
                 cout<<»\nEXIT!!!\n»;
                 break;
        default:    // if a chosen action is incorrect
                 cout<<»\nError!!! This operator isn't correct\n»;
}
         // request for operation choice
         cout<<»\nSelect operation:\n»;
         cout<<»\n 1 - if you want to see SUM.\n»;
         cout<<»\n 2 - if you want to see DIFFERENCE.\n»;
         cout<<»\n 3 - if you want to exit.\n»;
                  cin>>answer;
          }
        cout<<»\nBye....\n»;
}
```

This example proposes to a user to choose an action. After input the program checks: if the action — program exit — program ends, if not, then there is loop entry and analysis of a desirable action are carried out.

This code is not an optimal solution. As you can see the fragment.

```
// request for operation choice
   cout<<»\nSelect operation:\n»;
   cout<<»\n 1 - if you want to see SUM.\n»;
   cout<<»\n 2 - if you want to see DIFFERENCE.\n»;
   cout<<»\n 3 - if you want to exit.\n»;
   cin>>answer;
```

repeats a few times. In this case we should use do while. This construction will modify the code into a proper form. Project name CalcDoWhile.

```
##include <iostream>
using namespace std;
void main()
{
int answer,A,B,RES;

do{ // loop entry
        // request for operation choice
        cout<<»\nSelect operation:\n»;
        cout<<»\n 1 - if you want to see SUM.\n»;
        cout<<»\n 2 - if you want to see DIFFERENCE.\n»;
        cout<<»\n 3 - if you want to exit.\n»;
                               cin>>answer;

        // action analysis
        switch(answer){
                 case  1:    // if user chooses adding
                 cout<<»Enter first digit:\n»;
                 cin>>A;
                 cout<<»Enter second digit:\n»;
                 cin>>B;
                 RES=A+B;
                 cout<<»\nAnswer: «<<RES<<»\n»;
                 break; // termination of switch
                 case  2:     // if user chooses deducting
                 cout<<»Enter first digit:\n»;
                 cin>>A;
                 cout<<»Enter second digit:\n»;
                 cin>>B;
                 RES=A-B;
                 cout<<»\nAnswer: «<<RES<<»\n»;
                 break; // termination of switch
        case  3: // if user chooses exit
                 cout<<»\nEXIT!!!\n»;
                 break;
        default:    // if a chosen action is incorrect
                 cout<<»\nError!!! This operator isn't correct\n»;
        }

        } while(answer!=3);
cout<<»\nBye....\n»;

}
```

In view of the above you should understand that all constructions described in the present lesson are useful. You should only learn how to use one or another depending on the task.

Now when we are already familiar with the loops, you can pass to the next lesson unit. We have prepared for you a few examples on the current topic.

4. Examples to the lesson

Example 1.

Problem statement.

To write a program, which finds an amount of all integer digits from 1 to 5 included. Project name is Summ.

Realization code.

```
##include <iostream>
using namespace std;
void main(){
        int BEGIN=1; // range onset of summable values
        int END=5; // range end of summable values
        int SUMM=0; // variable for amount accumulation
        int i=BEGIN; // control variable

        // condition check
        while(i<=END){ //(comparing control variable with range end)
                 SUMM+=i;// amount accumulation
                 i++;// change of control variable
        }

        // display of a result
        cout<<«Result - «<<SUMM<<«\n\n»;
}
```

Comment to the code.

As a comment to the code, we decided to present a table that thoroughly describes each loop iteration:

INPUT DATA			
BEGIN=1		END=5	
LOOP OPERATION			
Iteration number	i	condition	SUMM
1	1	1≤5 — true	0+1=1
2	2	2≤5 — true	1+2=3
3	3	3≤5 — true	3+3=6
4	4	4≤5 — true	6+4=10
5	5	5≤5 — true	10+5=15
6	6	6≤5 — true	x
SUMM=15			

Studying the table it is easy to notice that control variable also performs a role of a variable that sequentially searches the values for adding.

Note: A common mistake is that control variable can change its value only by one, but this is not true. It is important that the variable changed in any logical way.

EXAMPLE 2.

Problem statement.

Write a program displaying a line of 5 asterisks. Project name Line.

REALIZATION CODE.

```
#include <iostream>
using namespace std;
void main(){
        int COUNT=5; // number of asterisks (line length)
        int i=0; // control variable

        while(i<=COUNT){ // condition check

         cout<<»*»;// display of asterisk
         i++;// change of control variable
        }
        cout<<»\n\n»;
}
```

Commends to the code.

1.Control variable at the moment of condition check is equal to the number of already drawn asterisks. It happens because i variable increases by one after each output *.

2. The loop will stop only when i=5, which is equal to the number of drawn *

How we pass to home assignment!

5. Home assignment

1. Develop a program displaying on a screen a horizontal line of symbols. Number and type of symbols, as well as type of line: horizontal or vertical is up to a user to decide.

2. Write a program, which calculates a sum of all integer odd numbers in a range set up by a user.

3. A natural number n is given. Write a program, which calculates a factorial of non-negative integers n (i.e. the number is integer and greater than 0). Factorial calculation formula is given below.

```
n! = 1*2*3*....*n, (calculation formula of factorial of number n)
0! = 1 (factorial 0 is equal to 1 (by factorial definition)
```