STEP computer ACADEMY

PROGRAMMING **C**

Lesson No. 5

Nested Loops

Contents

1. Nested Construction

In previous lessons you have acquainted with the construction, which is called loop and loop realization variants in language C. as you might have noticed a loop is one of the fundamental construction in programming. You can resolve a big number of tasks using a loop. You have also faced the fact that we can nest logical choice constructions within the loop, for example if and switch. Though, we won't stop and go an extra mile and will try to nest within a loop a similar construction, i.e. other loop. Review a simple example:

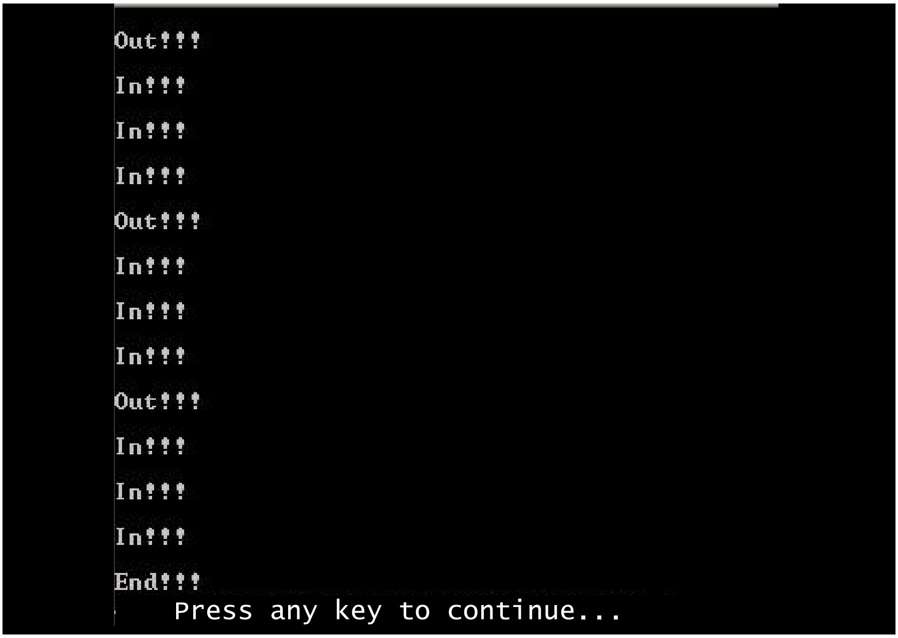
```
#include <iostream>
using namespace std;
void main ()
{
    int i=0,j;
    while(i<3){
        cout<<"\nOut!!!\n";
        j=0;
        while(j<3){
            cout<<"\nIn!!!\n";
            j++;
        }
        i++;
    }
    cout<<"\nEnd!!!\n";

}
```

Project name is NestedLoop.

Let's analyze the example:

1. A program checks the condition i<3, since 0 is less than 3 the condition is true and the program enters an external loop.
2. Is displayed on a screen ***Out!!!***
3. J variable is reset to zero.
4. Now the condition j<3 is checked, since 0 is less than 3 the condition is true and the program enters an internal loop.
5. ***In!!!*** Is displayed on a screen
6. J variable is changed.
7. And again the condition j<3 is checked, and because 1 is less than 3 the condition is true and the program enters the internal loop.
8. ***In!!!*** Is displayed on a screen.
9. J variable is changed.
10. And again the condition j<3 is checked, because 2 is less than 3 condition is true and the program enters an internal loop.
11. ***In!!!*** Is displayed on a screen.
12. J variable is changed.
13. And again the condition j<3 is checked, because 3 in not less than 3 condition is false and the program exits an internal loop.

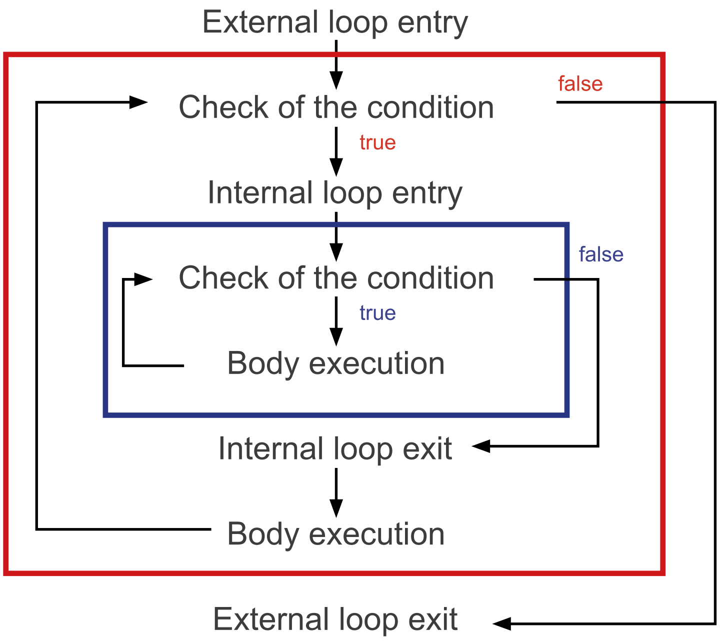
Afterwards the code returns to the step 1. All described above steps (1–13) will be repeated 3 times, i.e. until i becomes equal to 3. Afterwards the program exits the external loop and ***End!!!*** will be displayed on a screen.



```
Out!!!
In!!!
In!!!
In!!!
Out!!!
In!!!
In!!!
In!!!
Out!!!
In!!!
In!!!
In!!!
End!!!
    Press any key to continue...
```

Principle of the program operation realizing a nested loop is based on the fact that the internal loop is completely executed on each step of the external loop from the beginning to the end. In other words until the program exits the nested loop, there will be no continuation of execution of the external loop. Here is a scheme of operation of nested loops.

Operation scheme



As you can see everything is quite easy, but despite it, the nested constructions facilitate a realization of the majority of complex algorithms. We can fully realize this after we review the next lesson unit, in which we have prepared a few examples for you.

2. Use Cases

Use Case 1

Problem statement

Write a program displaying the multiplication table on a screen. Project name is MultiplicationTable.

Realization code:

```
#include <iostream>
using namespace std;
void main ()
{
    for(int i=1;i<10;i++)
    {
        for(int j=0;j<10;j++)
        {
            cout<<i*j<<"\t";
        }
        cout<<"\n\n";
    }
}
```

Comments to the code

1. Control variables of external and internal loops exercise the functions of multipliers.
2. Control variable i is created and initialized by the value 1.
3. The program checks the condition i<10, since 1 is less than 10 the condition is true and the program enters the external loop.
4. Control variable j is created and initialized by the value 1.

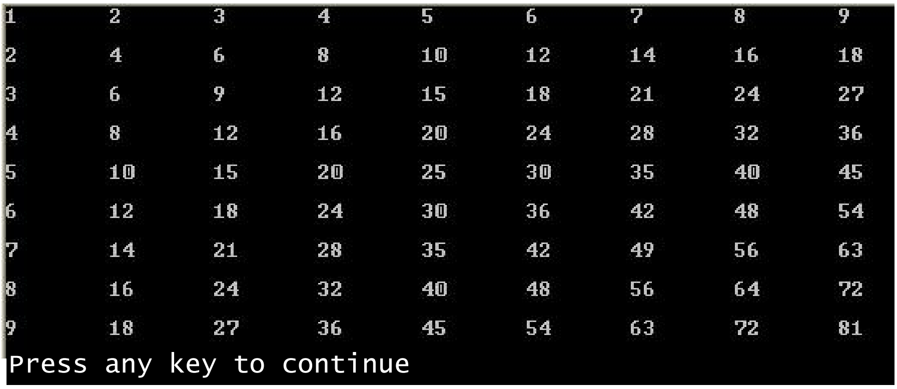
5. The program checks the condition j<10, since 1 is less than 10 the condition is true and the program enters the internal loop.
6. The product i by j — 1 is displayed on a screen
7. J variable is changed.
8. And again the condition j<10 is checked, since 2 is less than 10 the condition is true and the program enters во the internal loop again.
9. Production of i by j — 2 is displayed on a screen
10. J variable is changed.

. . .

Actions 5–7 are being repeated until j becomes equal to 10, where the current value i (1) multiplies by each j value (from 1 to 9 included), the result will be displayed on a screen. We get a line of one times table.

Then the program exits the internal loop and moves a screen cursor two lines lower. Afterwards, the variable i is increased by one and enters the internal loop again in order to display the line of two times table.

Thus, at last we get the complete multiplication table displayed on a screen.



```
1       2       3       4       5       6       7       8       9

2       4       6       8       10      12      14      16      18

3       6       9       12      15      18      21      24      27

4       8       12      16      20      24      28      32      36

5       10      15      20      25      30      35      40      45

6       12      18      24      30      36      42      48      54

7       14      21      28      35      42      49      56      63

8       16      24      32      40      48      56      64      72

9       18      27      36      45      54      63      72      81
Press any key to continue
```

Use Case 2

Problem statement

To display a rectangle made out of symbols 20 by 20. Project name is Rectangle.

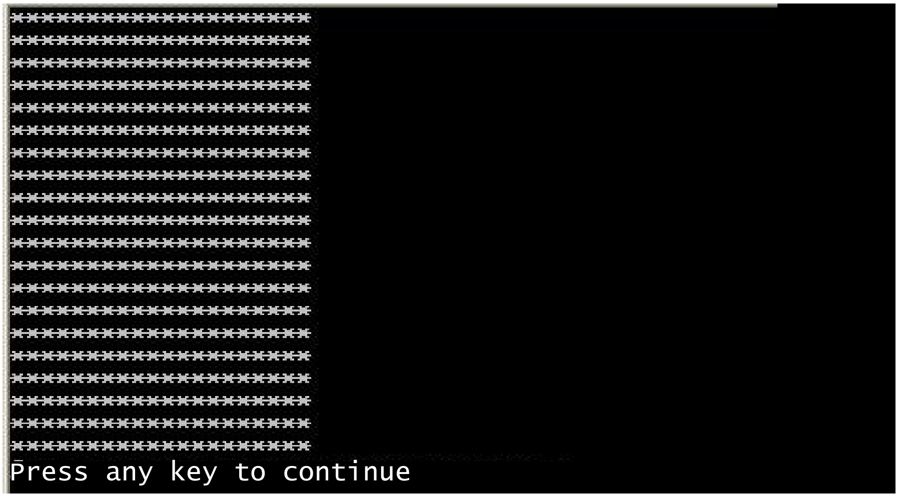
Code realization:

```
#include <iostream>
using namespace std;
void main(){
    int str;
    int star_count;
    int length=20;
    str=1;

    while(str<=length)
    {
        star_count=1;
        while(star_count<=length)
        {
            cout<<"*";
            star_count++;
        }
        cout<<"\n";
        str++;
    }
}
```

Comment to the code

1. External loop control variable str controls the number of lines within a rectangle.
2. Internal loop control variable star_count controls the number of symbols in each line.
3. Length — is a length of a rectangle side.

4. After drawing of each line there is exercised a transition to the next line in a rectangle within the external loop.
5. The result is as follows:



```
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
Press any key to continue
```

> ***Note:*** *Pay attention that in spite of the fact that the number of the lines corresponds to the number of symbols within a line, it does not look like a square on a screen! It happens because symbol height and width are different.*

That's it! Now you have the complete information about the loops, their types and operation principles. But before making your home assignment we would also get acquainted with another lesson unit. This unit will help you both to write programs and analyze their work.

3. Use of Integrated Debugger Microsoft Visual Studio

Concept of Debugging. Necessity to Use a Debugger

As you already know there are two types of program errors.

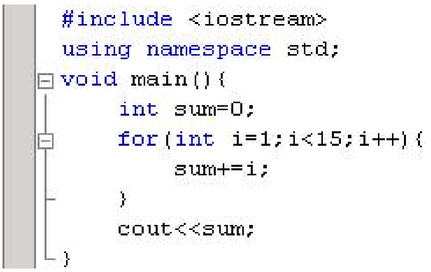
Error at the compile stage — is an error of programming language syntax. Such errors or mistypes are controlled by a compiler. Program containing this kind of a mistake will not star up and a compiler will show the string with a mistake.

Error at the execution stage — is an error resulting in incorrect work of the program, or to its full shutdown. In this regard we should take into account that such an error will not be controlled by a compiler. Only in rare cases a compiler may notify about an incorrect instruction, but in general in such cases a developer should bail out alone.

Now we will talk about the errors at the execution stage. Very often to find such an error we need to follow a program fragment step by step, the way the program would have been executed. Certainly it is recommended to calculate what value particular variables have at particular moment. Of course we could make such calculations on a paper analyzing step-by-step, though in Visual Studio programming environment there is a special tool for program analysis and this tool is a debugger. This lesson unit is devoted to the fundamentals of working with the debugger.

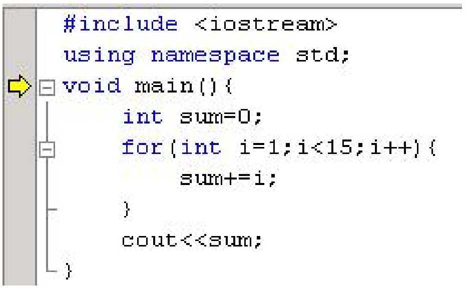
Step-by-step Program Execution

Assume that we are going to analyze the following code:



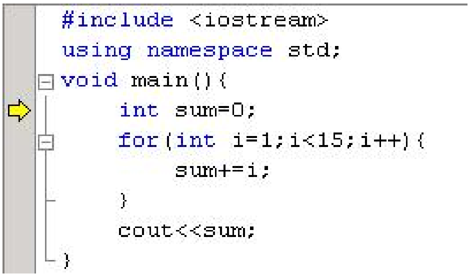
```
#include <iostream>
using namespace std;
void main(){
        int sum=0;
        for(int i=1;i<15;i++){
                sum+=i;
        }
        cout<<sum;
}
```

At first create the project and type in the code. Compile it and make sure there are no syntactic errors. Now let's start. Press a functional key F10 on a keyboard. Next to the executed code string you will see a yellow arrow on a screen.



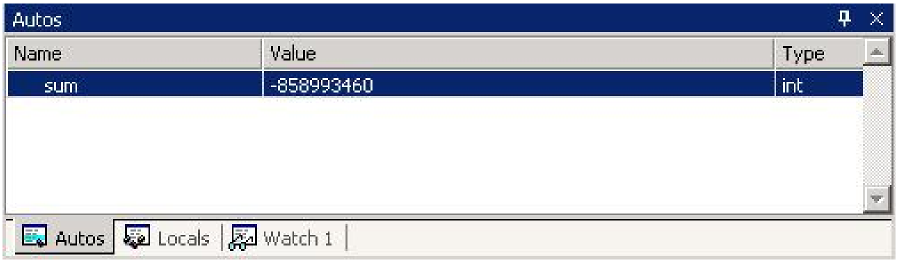
```
#include <iostream>
using namespace std;
void main(){
        int sum=0;
        for(int i=1;i<15;i++){
                sum+=i;
        }
        cout<<sum;
}
```

This very arrow indicates which code string is being «executed» at the moment. To move to the next step of the program press F10 again and you get to the next string:



```
#include <iostream>
using namespace std;
void main(){
        int sum=0;
        for(int i=1;i<15;i++){
                sum+=i;
        }
        cout<<sum;
}
```

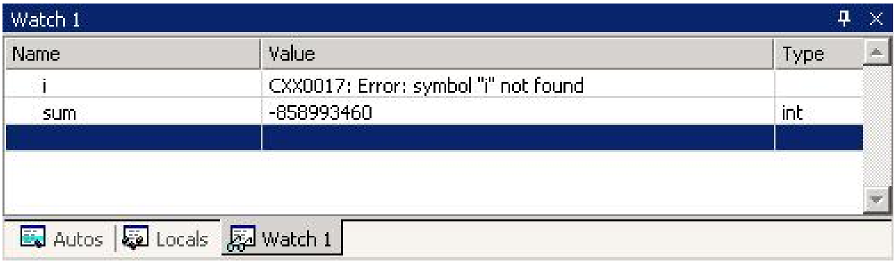
Pay attention that there is a tab set in the lower part of the screen for variables analysis:



Autos

Name	Value	Type
sum	-858993460	int

Autos — tab is designated for review of variable values that exist at the moment of execution of the current code string. To write in a random text on the tab is not allowed, for this is an automatic function.



Watch 1

Name	Value	Type
i	CXX0017: Error: symbol "i" not found	
sum	-858993460	int

Watch is designated exactly for the cases where it is necessary to choose a variable for preview at once. You just write in a variable name within the field Name and it will be displayed regardless of the executed code.

Now we just press F10, «skip through» the code and notice how the data is changed within the tabs.

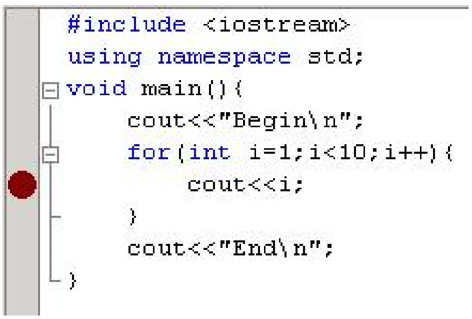
Note: *If you want to stop the debugger earlier than code analysis terminates press a combination of keys Shift+F5.*

Note*: You might have noticed that the debugger starts an analysis from the first string of the program. If you want to run the debugger from the particular string of a program install a cursor at the required string and press a combination of the keys Ctrl+F10.*

Breakpoint

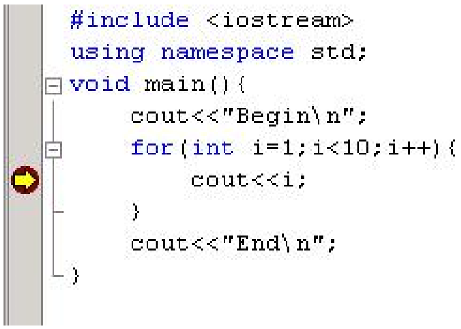
Let's review the situation when we need to execute a code part and having stopped in the particular place we should run the debugger and the breakpoint is used for this purpose.

Type in the next code and put a cursor at the string cout<<i; press F9 key. Next to the string there will appear a red dot and this is a breakpoint.



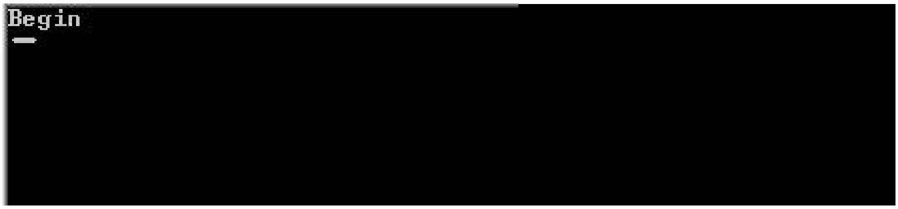
```
#include <iostream>
using namespace std;
void main(){
    cout<<"Begin\n";
    for(int i=1;i<10;i++){
        cout<<i;
    }
    cout<<"End\n";
}
```

Now press F5 and the program will be run and executed till the moment where a breakpoint is installed and then transits into the debugger mode.



```
#include <iostream>
using namespace std;
void main(){
	cout<<"Begin\n";
	for(int i=1;i<10;i++){
		cout<<i;
	}
	cout<<"End\n";
}
```

Pay attention to the console state (program window). Everything that has happened is displayed therein:



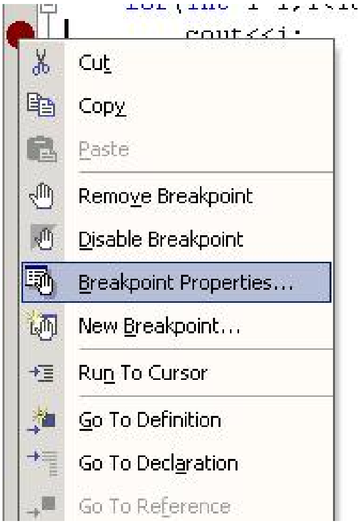
```
Begin
_
```

Afterwards a debugger exercises its common work. Move a yellow arrow using F10 and follow variables behavior. Besides, have a look onto a console window, because all code changes are displayed there.

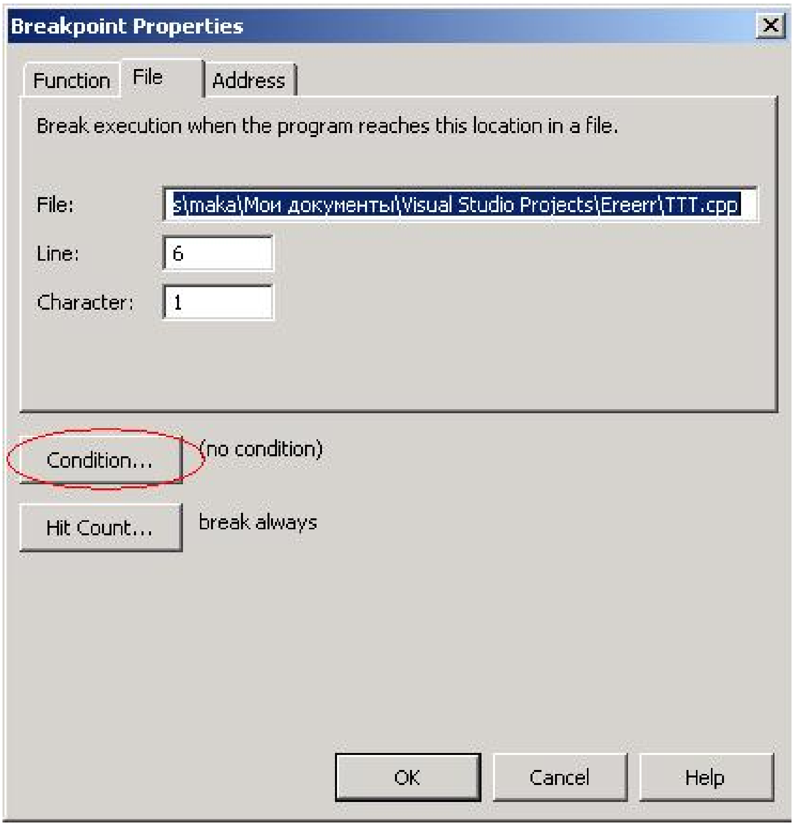
«Smart» Breakpoint

We have just run program analysis from the particular place. Though can notice that debugger was initiated at the moment of the body loop execution start, meaning at the first iteration. It is inconvenient in case if there is a big number of iterations and we have to skip some of them. In other words, if you want to start the analysis from the 5th iteration of a loop. It is easy to solve the problem — we should make a breakpoint «smart».

With this purpose click the right mouse button on the very point and in the menu choose Breakpoint Properties.



In the appeared window File, Line and Character — are a file, line and position where a breakpoint is installed. We are interested in the button Condition with a sign (no condition). Press it.


Breakpoint Properties

Function | File | Address

Break execution when the program reaches this location in a file.

File: s\maka\Мои документы\Visual Studio Projects\Ereerr\TTT.cpp

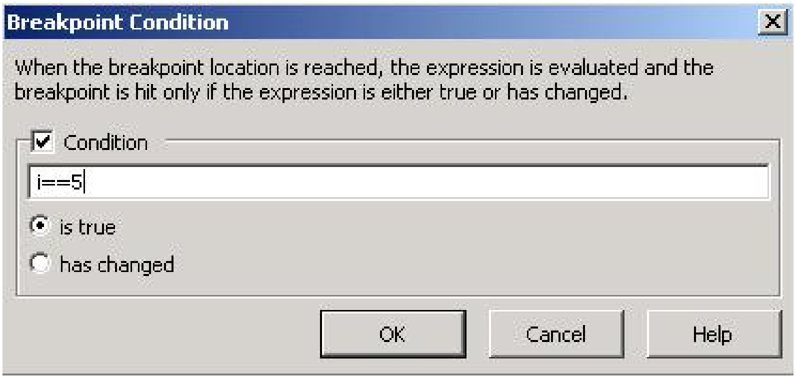
Line: 6

Character: 1

Condition... (no condition)

Hit Count... break always

In an appeared window we need to write in a condition under which a debugger will be initiated. Our condition i==5. Choose is true — meaning to stop when a condition becomes true and press OK.


Breakpoint Condition

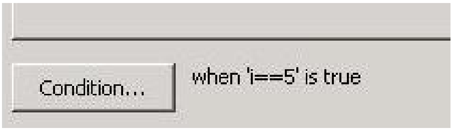
When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is either true or has changed.

[✓] Condition

i==5

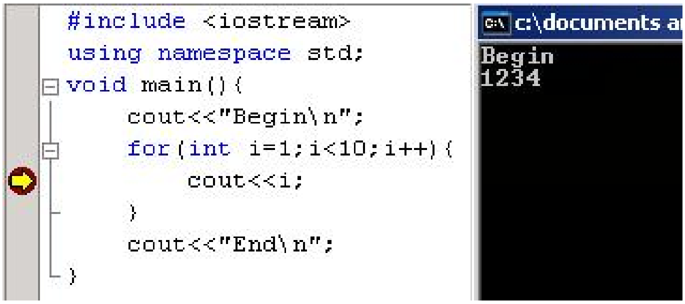
[✓ is true]
has changed

Now there is another sign next to the button. Press OK in the main menu.



when 'i==5' is true

When everything is ready press F5 and wait what happens next. As you can see everything went successfully — the debugger was initiated at the right moment.



```
#include <iostream>
using namespace std;
void main(){
     cout<<"Begin\n";
     for(int i=1;i<10;i++){
          cout<<i;
     }
     cout<<"End\n";
}
```

```
c:\documents a
Begin
1234
```

Note: *You can cancel a breakpoint pressing F9 in the line containing it.*

Today we have discussed not all debugger possibilities. We will talk about it in future when studying functions. And now we would recommend practicing your skills of working with the debugger on the lesson examples and your home assignment. Good luck!

4. Home Assignment

1. Create a program displaying on a screen prime numbers in a range from 2 to 1000. (a number is called a prime when it can be divided by itself or by 1 without the excess 1; whereas numbers 1 and 2 are not considered prime).
2. Write a program displaying on a screen the following figure:

```
*********************
*                   *
*                   *
*                   *
*                   *
*                   *
*                   *
*********************
```

   Width and height of the figure are assigned by a user from a keyboard.

3. Using a loop to display a calendar of the current month.

Lesson No. 5

Nested Loops