**ITCS 1212L**

**Lab lessons 6**

# Iteration structures (Loops) / More on Random Numbers

**Iteration structures (loops)**

Loops have a purpose to repeat a statement a certain number of times or while a condition is fulfilled.

# The while loop:

Its syntax is:

while (expression) statement

and its functionality is simply to repeat statement while the condition set in expression is true. For example, we are going to make a program to countdown using a while-loop:

//custom countdown using while

#include <iostream>

using namespace std;

int main ()

{

int n;

cout << "Enter the starting number > ";

cin >> n;

while (n>0) {

cout << n << ", ";

--n;

}

cout << "FIRE!\n";

return 0;

}

Sample Input:

Enter the starting number > 7

Output:

7, 6, 5, 4, 3, 2, 1, FIRE!

When the program starts the user is prompted to insert a starting number for the countdown. Then the while loop begins, if the value entered by the user fulfills the condition n>0 (that n is greater than zero) the block that follows the condition will be executed and repeated while the condition (n>0) remains being true.

The whole process of the previous program can be interpreted according to the following script (beginning in main):

Step 1: Start

Step 2: User assigns a value to n.

Step 3: The while condition is checked (n>0). At this point there are two possibilities:

\*condition is true: statement is executed (to step 3)

\*condition is false: ignore statement and continue after it (to step 6)

Step 4: Execute statement:

cout << n << ", ";

--n;

(prints the value of n on the screen and decreases n by 1)

Step 5: End of block. Return automatically to step 3.

Step 6: Continue the program right after the block: print FIRE!

Step 7: Stop.

When creating a while-loop, we must always consider that it has to end at some point, therefore we must provide within the block some method to force the condition to become false at some point, otherwise the loop will continue looping forever. In this case we have included --n; that decreases the value of the variable that is being evaluated in the condition (n) by one - this will eventually make the condition (n>0) to become false after a certain number of loop iterations: to be more specific, when n becomes 0, that is where our while-loop and our countdown end.

Of course this is such a simple action for our computer that the whole countdown is performed instantly without any practical delay between numbers.

# The do-while loop:

Its syntax is:

do statement while (condition);

Its functionality is exactly the same as the while loop, except that condition in the do-while loop is evaluated after the execution of statement instead of before, granting at least one execution of statement even if condition is never fulfilled. For example, the following example program echoes any number you enter until you enter 0.

// number echoer

#include <iostream>

using namespace std;

int main ()

{

unsigned long n;

do {

cout << "Enter number (0 to end): ";

cin >> n;

cout << "You entered: " << n << "\n";

} while (n != 0);

return 0;

}

Sample Input/Output:

Enter number (0 to end): 12345

You entered: 12345

Enter number (0 to end): 160277

You entered: 160277

Enter number (0 to end): 0

You entered: 0

The do-while loop is usually used when the condition that has to determine the end of the loop is determined within the loop statement itself, like in the previous case, where the user input within the block is what is used to determine if the loop has to end. In fact if you never enter the value 0 in the previous example you can be prompted for more numbers forever.

# The for loop:

Its syntax is:

for (initialization; condition; increase) statement;

and its main function is to repeat statement while condition remains true, like the while loop. But in addition, the for loop provides specific locations to contain an initialization statement and an increase statement. So this loop is specially designed to perform a repetitive action with a counter which is initialized and increased on each iteration.

It works in the following way:

1. initialization is executed. Generally it is an initial value setting for a counter variable. This is executed only once.
2. condition is checked. If it is true the loop continues, otherwise the loop ends and statement is skipped (not executed).
3. statement is executed. As usual, it can be either a single statement or a block enclosed in braces { }.
4. Finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

Here is an example of countdown using a for loop:

// countdown using a for loop

#include <iostream>

using namespace std;

int main ()

{

for (int n=10; n>0; n--) { cout << n << , ";

}

cout << "FIRE!\n";

return 0;

}

Sample Output:

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!

**NOTE:** The initialization and increase fields are optional. They can remain empty, but in all cases the semicolon signs between them must be written. For example we could write: for (;n<10;) if we wanted to specify no initialization and no increase; or for (;n<10;n++) if we wanted to include an increase field but no initialization (maybe because the variable was already initialized before).

Optionally, using the comma operator (,) we can specify more than one expression in any of the fields included in a for loop, like in initialization, for example. The comma operator (,) is an expression separator, it serves to separate more than one expression where only one is generally expected. For example, suppose that we wanted to initialize more than one variable in our loop:

for ( n=0, i=100 ; n!=i ; n++, i-- )

{

// whatever here...

}

This loop will execute for 50 times if neither n or i are modified within the loop:



n starts with a value of 0, and i with 100, the condition is n!=i (that n is not equal to i). Because n is increased by one and i decreased by one, the loop's condition will become false after the 50th loop, when both n and i will be equal to 50.

**Random Numbers**

In order to generate a random number we make a call to function rand(). If we want the rand() function to generate a new number, we need to call srand() function with time() function as argument as shown below:

srand (time(NULL));

To generate a random number between two specific integers we can use the following formula.

//Generate a random number between a and b

s = a + rand() % (b-a+1);srand (time(NULL));

The generated random number mod b will result to a number between 0 to (b - a). By adding a to this random number, the range of the numbers will change from a to b.

For example to generate a number between 1 and 13, we can write the following equation.

//Generate a random number between 1 and 13

number = 1 + rand() % 13;