Lecture-7

Control Structures

A program is usually not limited to a linear sequence of instructions. During its process it may bifurcate, repeat code or take decisions. For that purpose, C++ provides control structures that serve to specify what has to be done by our program, when and under which circumstances.

With the introduction of control structures we are going to have to introduce a new concept: the *compound-statement* or *block*. A block is a group of statements which are separated by semicolons (;) like all C++ statements, but grouped together in a block enclosed in braces: `{ }`:

```
{ statement1; statement2; statement3; }
```

Most of the control structures that we will see in this section require a generic statement as part of its syntax. A statement can be either a simple statement (a simple instruction ending with a semicolon) or a compound statement (several instructions grouped in a block), like the one just described. In the case that we want the statement to be a simple statement, we do not need to enclose it in braces (`{}`). But in the case that we want the statement to be a compound statement it must enclosed between braces (`{}`) forming a block.

7.1 Conditional structure: if and else

The `if` keyword is used to execute a statement or block only if a condition is fulfilled. Its form is:

```
if (condition) statement
```

Where `condition` is the expression that is being evaluated. If this condition is true, `statement` is executed. If it is false, `statement` is ignored (not executed) and the program continues right after this conditional structure. For example, the following code fragment prints `x is 100` only if the value stored in the `x` variable is indeed `100`:

```
if (x == 100)
  cout << "x is 100";
```

If we want more than a single statement to be executed in case that the condition is true we can specify a block using braces `{ }`:

```
if (x == 100)
{
   cout << "x is ";
   cout << x;
}
```

We can additionally specify what we want to happen if the condition is not fulfilled by using the keyword `else`. Its form used in conjunction with `if` is:

```
if (condition) statement1 else statement2
```

Program 7.1

```
// Program for Factorial of a number

#include <iostream> /* STanDard Input & Output Header file */
using namespace std;

int fact (int n)
{
if ((n == 0)||(n==1))
return 1;
else
return (n * fact(n - 1));
}

int main()
{
int n = 5;
cout<<"Factorial of "<<n<<"is="<<fact(n);//cout is defined in iostream
return 0;

}
```

The `if + else` structures can be concatenated with the intention of verifying a range of values. The following example shows its use telling if the value currently stored in `x` is positive, negative or none of them (i.e. zero):

```
if (x > 0)
  cout << "x is positive";
else if (x < 0)
  cout << "x is negative";
else
  cout << "x is 0";
```

Remember that in case that we want more than a single statement to be executed, we must group them in a block by enclosing them in braces `{ }`.

7.2 Iteration structures (loops)

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

The while loop

Its format 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 count down using a while-loop:

```
// custom countdown using while
```

```
Enter the starting number > 8
8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

```
#include <iostream>
using namespace std;

int main ()
{
  int n;
  cout << "Enter the starting
number > ";
  cin >> n;

  while (n>0) {
    cout << n << ", ";
    --n;
  }

  cout << "FIRE!";
  return 0;
}
```

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):

1. User assigns a value to `n`
2. The while condition is checked (`n>0`). At this point there are two posibilities:
   * condition is true: statement is executed (to step 3)
   * condition is false: ignore statement and continue after it (to step 5)
3. Execute statement:
   `cout << n << ", ";`
   `--n;`
   (prints the value of `n` on the screen and decreases `n` by `1`)
4. End of block. Return automatically to step 2
5. Continue the program right after the block: print FIRE! and end program.

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 format 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 a number : ";
    cin >> n;
    cout << "You entered: " << n <<
"\n";
  } while (n != 0);
  return 0;
}
```

```
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 a value `0` in the previous example you can be prompted for more numbers forever. To come out of the loop, one has to enter zero.

7.3 The for loop

Its format 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!";
  return 0;
}
```

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

//Program 7.2 : Sum and average of given numbers

```
#include <iostream>
using namespace std;

void main(void)
 {
   int n;
   cout<<"Ho many numbers?";
    cin>>n;
   float sum = 0;
    float a;
    for(int i=0; i<= n-1; ++i) {
    cout<<"enteer a number\n";
     cin>>a;
     sum+=a;
        }
     float av =sum/n;
     cout<<"Sum="<<sum<<endl;
     cout<<"Average="<<av<<endl;
}
```

Program 7.3 :Sum of first 100 natural numbers.

```
#include <iostream>
using namespace std;
void main(void)
 {
 int sum, digit;
  sum=0;
digit=1;
while (digit <=100){
sum+=digit;
digit++;
}
```

```
cout<<"1+2+3..+100="<<sum<<endl;
}
```

Prog 7.3

// Factorial without using function command

```
#include <iostream>
using namespace std;

int main(void)
 {
   int n;
   long int perm=1;
    while (n>=1)
     { perm=perm*n;
       n=n-1;
}
   cout<<"factorial of "<<n<<"is"<<perm<<endl;
}
```

Prog 7.4 :

```
//Summation of series
#include <iostream>
using namespace std;
int fact(int n){if((n==0)||(n==1))
   return 1;
else
return (n*fact(n-1));
}
int main()
{
      float sum=1.0;
      for(int n=1; n<=12; n++)
      {
            cout<<"factorial of "<<n<<"is="<<fact(n)<<"\n";
            //getch();
      sum+=n/fact(n);
      }
      cout<<"sum of 10 terms is="<<sum<<endl;
      return 0;
}
/*factorial of 1is=1
factorial of 2is=2
factorial of 3is=6
factorial of 4is=24
factorial of 5is=120
factorial of 6is=720
factorial of 7is=5040
factorial of 8is=40320
factorial of 9is=362880
factorial of 10is=3628800
sum of 10 terms is=3
Press any key to continue . . .*/
```