The "for" loop

W*hile* and *do-while* provide all the looping mechanisms needed for programming. In fact, either one of them would be enough since, as we have seen, anything written using *while* can be translated into a *do-while* and viceversa. However, the fact that one construct would be enough does not mean that it would be desirable to have only one looping command. As we have discussed, we want our programs to express our algorithms in a clear fashion; therefore our programming language should provide enough variety of "idioms" to allow us to choose the ones that best express our ideas.

This need for enhanced expressiveness justifies the existence of yet another loop form, itself equivalent to the previous ones, but with its own expressive power: the *for* loop.

The basic format of the *for* loop is:

for (< initial commands> ; < condition> ; <update commands>)

where the parts inside < > are provided by the programmer, and the rest, including the intervening ';' are mandatory.

The *for* loop works as follows:

- The <initial commands> are executed first, and only once.
- The <condition> is tested before entering the body of the loop. If it fails, the loop stops.
- The < update commands> are executed every time we finish the body of the loop, before we retest the <condition> to check whether the loop should continue.

For example:

```
for ( int i = 0 ; i < 4; i++)

cout << i << " ";
```

where the initial command defines and initializes variable i , where the condition --tested every time the body of the loop is considered for execution-- is *i < 4* and, finally, where the update part is represented by *i*++, which increments the variable every time through the loop, before the condition is retested.

Note that any variable defined in the initialization part of the *for* loop, will exist only within the loop. Upon exiting the *for*, that variable "does not exist": you will get an error if you try to refer to it after the loop.

It is common to find that the initialization and the update sections of the *for* are made up of more than one command. In this case, the commands comprising those sections would be separated by commas ','.

```
for (int i = 0, j = 1 ; i < 4 && j < 5; i++, j++)

cout << i << " " << j << '\n';
```

In this example, the initialization part defines and initializes i and j, and the update part increments i and j in two commands separated by a comma.

Finally, the following code, albeit ugly, is correct:

```
for (int i = 0, j = 1 ; i < 4 && j < 5; i++, cout << j << '\n')

{

cout << i << " ";

j++;

}
```

What would this print?

Notice that, in this case, the "update" section is made up of the increment of i and, separated by a comma, the output command for j.