Script for the vhb lecture

Programming in C++
Part 1

Prof. Dr. Herbert Fischer
Deggendorf Institute of Technology

Table of Contents

3 Control Structures

Chapter 3 teaches you how to program conditional control structures and loops in C++.

3.1 Conditional control structures

3.1.1 if statement

The *if statement* is used to check a condition and then, depending on whether the condition is "true" or "false", to execute certain blocks of code.

Syntax:
```
if(condition) {
        statement;
}
```

To specify the condition, relational operators are required (see **Arithmetic operators in 2.4**).

Important:
- Please note the special notation "==" when checking for equality.
- If you put the !-operator in front of a condition, you can negate conditions.
- A condition may be in an arithmetic expression. It assumes the value 1 for "true" and 0 for "false"

Example:
```
int x;
cout << "Please enter a number: ";
cin >> x;
if (x > 10)
{
        cout << " The number entered is greater than 10." << endl;
}
```

Explanation:
First, the user is prompted to enter a number. If the number is greater than 10, the condition x > 10 is evaluated as "true" and the text is displayed; in the other case, nothing is displayed. Note that if the conditional statement results in "true", the statement that follows immediately after the if statement is executed. If the statement block consists of only one statement, the curly brackets can be omitted.

You can use an if statement to perform different operations depending on whether the condition is true or false. There are six relational operators. They return the value "true" if the two operands (left and right of the operator) satisfy the comparison criteria, otherwise they return the value "false".

==	equal
<=	is less than (or equal to)
>=	is greater than (or equal to)
<	is less than
>	is greater than
!=	is unequal

Example:

```
if (x == 20) {
        cout << "The number entered is 20." << endl;
        }
else if (x <= 19) {
        cout << "The number entered is less than or equal to 19." << endl;
else {
        cout << "The number entered is greater than 20." << endl;
        }
```

Explanation:
The keyword *else* extends the possible applications of the conditional control structure. The code that follows the keyword *else* is only executed if the previous conditions are false. You can specify more than two alternatives in a conditional control statement. The *else if* statement is used to have another block of code executed if the first condition is false. You can use as many *else if* statements as you want.

You can also link several conditions to one expression. You have the following options:

!	NOT	Result is true if the operand is false
&&	AND	Result is true if both operands are true
||	OR	Result is true if at least one operand is true

Example:

```
int i = 10, j = 20;
if ((i == 10) && (j == 10))
{
        cout << "Both values are equal to ten" << endl;
}
else if ((i == 10) || (j == 10))
{
        cout << "One value or both values are equal to ten " << endl;
}
else if ((i != 10) && !(j == 10))
{
        cout << "Both values are unequal to ten " << endl;
}
```

Note:
In C/C++ there are many ways to abbreviate statements. One way is to use only the name of the variable to check whether the condition is “true”.

Example:

```
bool fileOK;
if (fileOK) ReadData();

This is the abbreviation of the previous lines:
if (fileOK == true) ReadData ();
```

3.1.2 Conditional expression

Conditional control structures are often used to make an assignment, depending on the value of an expression. You can also use the ?-operator to express this is in a simpler way.

Example:
```
int min;
min = (a < b) ? a : b;
```

Explanation:

The syntax for this is:

Value = Condition ("Question") **?** Expression for TRUE **:** Expression for FALSE;

In the above example: Is a less than b? If true (i.e. a is less than), then assign a to min, if false (i.e. b is less than), then assign b to min.

3.1.3 switch statements

The *switch statement* can also be considered as an extended *if statement*. It allows you to switch/jump to different statement blocks (to execute alternate blocks of codes), depending on the value of a particular expression. The expression can be a variable, the result of a function call, or a valid C/C++ expression. The default statement is optional.

Syntax:
```
switch(expression) {
    case x1 :   <statement;>
                <break;>
    case x2 :   <statement;>
                <break;>
    …
    default :   <statement;>
                <break;>
}
```

Explanation:

- The x_i (case label) of a case statement must be integer constants (numbers or single letters, e.g.: case 'A').
- You can combine several case statements with the same purpose by using or omitting break statements.
- You can have any number of case statements within a switch loop.
- If there is no case label x_i for an expression of the switch statement, the default case is executed (if existent).

Task: Extend the following source code excerpt to create an executable program and experiment with it!

```
switch(AboveSpeedLimit) {
    case 0 : {
    Penalty = 0;
    break;
    }
    case 10 : {
    Penalty = 20;
    break;
    }
    case 15 : {
    Penalty = 50;
    break;
    }
    case 20 :
    case 25 :
    case 30 : {
    Penalty = AboveSpeedLimit * 10;
    break;
    }
    default : {
    Penalty = Subpoena();
    PenaltyDegree = Verdict();
    }
}
```

Explanation:
The switch loop consists of several parts. The first part consists of the expression – in our case the *AboveSpeedLimit* variable. This is followed by the case statements, which check the expression for equality.
If the *AboveSpeedLimit* variable is equal to 0 (case 0 :), the *Penalty* variable is assigned the value 0. If the *AboveSpeedLimit* variable is equal to 10, the *Penalty* variable is assigned the value 20, and so forth.
In each of the first three case statements you can find a break statement. When a break statement is reached, the switch block terminates.
Once a match has been found with one of the cases (the case is true), the rest of the switch statement is skipped. At the end of the switch loop is the default statement. The default case is executed if none of the cases is true.
Note that there are no statements behind case 20 and case 25. If the expression *AboveSpeedLimit* is 20 or 25, these cases are skipped and the following block is executed. This implies that the same code is executed for the values 20, 25 and 30.

3.2 Loop types

With a loop you can access arrays, perform an action several times, read a file, etc. We will discuss the *for*, *while*, and *do-while* loops.

Loops consist of the following basic elements:
- an entry point,
- a loop body that is enclosed by brackets; it contains the statement block that is executed each time the processing passes through the loop,
- the exit of the loop,
- the evaluation of a condition that determines when the loop is to be exited.

3.2.1 *for* loops

for loops are the most commonly used loops.
They require three parameters: the initial value (init), the condition statement (condition) that determines when the loop terminates, and the increment statement (increment).

Syntax:
```
for (init; condition; increment) {
        statement;
}
```

The *for loop* executes the statement block as long as the condition statement "condition" is true (unequal zero). The initial value “init” initializes any loop control variables. After the statements (body of the loop) have been executed, the status of the loop is changed according to the “increment” statement.

Example:

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

int main()
{
    cout << endl << "Program start FOR..." << endl << endl;
    int i;
    cout << "   1. FOR loop " << endl;
          for (i=0;i<12;i++)
          {
                    cout << "Increase  " << i << endl;
          }
    cout << "   2. FOR loop " << endl;
          for (int j=10; j>0; j--)
          {
                    cout << "Decrease  " << j << endl;
          }
   system("pause");
}
```

Explanation:
The first loop starts with the initial value 0. The statement in the body of the loop is executed until variable i has reached the value 12. During each execution, variable i increases by 1. In the second loop, a new variable named j is declared and initialized with the initial value 10. The statement in the body of the loop is executed until variable j has reached the value 0. After each execution, variable j decreases by 1.

3.2.2 *while* loops

while loops only contain one condition statement that is checked at the beginning of each iteration. The body of the loop is executed as long as the condition is “true”.

Syntax:

```
while (condition) {
        statement;
}
```

Example:

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

int main()
{
        cout << endl << " Program start WHILE ..." << endl << endl;
        int i = 6;

        while (i > 0)
        {
                i--;
                cout << endl << "I have to send" << i;
                cout << " parcels.";
        }

        cout << "DONE !" << endl << endl;

        system("pause");
}
```

Explanation:
With each loop pass, the value of variable i decreases by 1. The body of the loop is executed until variable i has reached the value 0. When the variable has reached the value 0, the condition is false, and the loop is terminated.

3.2.3 *do-while* loops

The *do-while loop* checks its condition at the end of the loop (unlike for and while loops which test the condition at the beginning of the loop).

Syntax:

```
do {
        statement;
        }

        while (condition)
```

The *do* loop executes the statement block as long as the condition is *true* (unequal zero). The status of the loop must be initialized before using the *do* loop and the status change must be explicitly specified within the statement block. The statement block is executed until the given condition becomes false.

Example:

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

int main()
{
        cout << endl << "Program start DO WHILE ..." << endl << endl;
        int i = 6;

        do {
                i--;
                cout << endl << "I have to send " << i;
                cout << " parcels.";
        }
        while (i>0);

        cout << "DONE !" << endl << endl;

        system("pause");
}
```

Difference between *while* loop and *do-while* loop:

While	**Do-While**

int i = 1;
while(i<0) {
  cout << i << endl;
  i++;
}
	int i = 1;
	do {
	  cout << i << endl;
	  i++;
	} while(i<0);

Console output: Press any key to continue…	Console output: 1 Press any key to continue…
Checks condition before output. The condition is false, termination at the beginning of the loop, no output.	Output before condition is checked. The condition is checked at the end of the loop, termination at the end of the loop.