8 Control Statements

In most of the programs presented above, the statements have been sequential, executed in the order they appear in the main program.

In many programs the values of variables need to be tested, and depending on the result, different statements need to be executed. This facility can be used to **select** among alternative courses of action. It can also be used to build **loops** for the **repetition** of basic actions.

8.1 Boolean expressions and relational operators

In C++ the testing of *conditions* is done using **boolean expressions**, which yield `bool` values that are either `true` or `false`. The simplest and most common way to construct such an expression is to use the so-called **relational operators**.

`x == y`	true if `x` is equal to `y`
`x != y`	true if `x` is not equal to `y`
`x > y`	true if `x` is greater than `y`
`x >= y`	true if `x` is greater than or equal to `y`
`x < y`	true if `x` is less than `y`
`x <= y`	true if `x` is less than or equal to `y`

Be careful to avoid mixed-type comparisons. If `x` is a floating point number and `y` is an integer the equality tests will **not** work as expected.

8.2 Compound boolean expressions using logical operators

If you need to test more than one relational expression at a time, it is possible to combine the relational expressions using the **logical operators**.

Operator	C++ symbol	Example
AND	&&, and	expression1 && expression2
OR	||, or	expression1 || expression2
NOT	!	!expression

The meaning of these will be illustrated in examples below.

8.3 The IF selection control statement

The simplest and most common selection structure is the `if` statement, which is written in a statement of the form:

`if(`*`boolean-expression`*`)` *`statement`*`;`

The **if** statement tests for a particular condition (expressed as a boolean expression) and only executes the following statement(s) if the condition is true. An example follows of a fragment of a program that tests if the denominator is not zero before attempting to calculate `fraction`.

```
if(total != 0)
   fraction = counter/total;
```

If the value of `total` is `0`, the boolean expression above is false and the statement assigning the value of `fraction` is ignored.

If a sequence of statements is to be executed, this can be done by making a **compound statement** or **block** by enclosing the group of statements in braces.

```
if(boolean-expression)
{
    statements;
}
```

An example of this is:

```
if(total != 0)
{
   fraction = counter/total;
   cout << "Proportion = " << fraction <<  endl;
}
```

8.4 The IF/ELSE selection control statement

Often it is desirable for a program to take one branch if the condition is true and another if it is false. This can be done by using an **if/else** selection statement:

```
if(boolean-expression)
    statement-1;
else
    statement-2;
```

Again, if a sequence of statements is to be executed, this is done by making a compound statement by using braces to enclose the sequence:

```
if(boolean-expression)
{
    statements;
}
else
{
    statements;
}
```

An example occurs in the following fragment of a program to calculate the roots of a quadratic equation.

```
// testing for real solutions to a quadratic
d = b*b - 4*a*c;
if(d >= 0.0)
{
   // real solutions
   root1 = (-b + sqrt(d)) / (2.0*a);
   root2 = (-b - sqrt(d)) / (2.0*a);
   real_roots = true;
}
else
{
   // complex solutions
   real = -b / (2.0*a);
   imaginary = sqrt(-d) / (2.0*a);
   real_roots = false;
}
```

If the boolean condition is `true`, i.e. (`d` ≥ 0), the program calculates the roots of the quadratic as two real numbers. If the boolean condition is `false`, then a different sequence of statements is executed to calculate the real and imaginary parts of the complex roots. Note that the variable `real_roots` is of type `bool`. It is assigned the value `true` in one of the branches and `false` in the other.

8.5 ELSE IF multiple selection statement

Occasionally a decision has to be made on the value of a variable that has more than two possibilities. This can be done by placing `if` statements within other `if-else` constructions. This is commonly known as *nesting* and a different style of indentation is used to make the multiple-selection functionality much clearer. This is given below:

```
if(boolean-expression-1)
    statement-1;
else if(boolean-expression-2)
    statement-2;
⋮
else
    statement-N;
```

For compound statement blocks, braces must be used. Here follows a dull example of ELSE IF multiple selection in use:

```
// Program to assess the user's for suitability as a major college donor:

// Find out how many houses are owned:
int howManyHousesSheOwns;
cout << "How many houses do you own?" << endl;
cin >> howManyHousesSheOwns;

// Give feedback
if (howManyHousesSheOwns<=0)
{
  cout << "And have you come far today?" << endl;
}
else if (howManyHousesSheOwns<10)
{
  cout << "Student hardship funds make a real difference to
           many students lives!" << endl;
}
else if (howManyHousesSheOwns<100)
{
  cout << "Have I told you about the business park the college
           would like to build in the Cotswolds?" << endl;
}
else
{
  // I'm speechless!
}
```

Note that since whitespace outside of strings is irrelevant in C++, you also see formatting like the following, which is entirely equivalent and preferred by many:

```
// Program to assess the user's for suitability as a major college donor:

// Find out how many houses are owned:
int howManyHousesSheOwns;
cout << "How many houses do you own?" << endl;
cin >> howManyHousesSheOwns;

// Give feedback
if (howManyHousesSheOwns<=0) {
  cout << "And have you come far today?" << endl;
} else if (howManyHousesSheOwns<10) {
  cout << "Student hardship funds make a real difference to
           many students lives!" << endl;
} else if (howManyHousesSheOwns<100) {
  cout << "Have I told you about the business park the college
           would like to build in the Cotswolds?" << endl;
} else {
  // I'm speechless!
}
```

8.6 SWITCH multiple selection statement

Instead of using multiple `if/else` statements C++ also provides a special control structure, `switch`.

For an integer variable `x`, **see footnote**[11], the `switch(x)` statement tests whether `x` is equal to the constant values `x1`, `x2`, `x3`, etc. and takes appropriate action. The `default` option is the action to be taken if the variable does not have any of the values listed.

```
switch(x)
{
   case x1:
      statements1;
       break;

   case x2:
      statements2;
       break;

   case x3:
      statements3;
       break;

   default:
      statements4;
       break;
}
```

The `break` statement causes the program to proceed to the first statement after the `switch` structure. Note that the `switch` control structure is different to the others in that braces are not required around multiple statements. If you forget to put in one of the `break` statements, then execution can continue from one case statement into another. Sometimes you intend this to happen, but more often than not it is not intended and a source of many bugs. Be careful to include all the `break` statements unless you are very sure there are some you do not need.

The following example uses the `switch` statement to produce a simple calculator that branches depending on the value of the operator being typed in. The operator is read and stored as a character value (`char`). The values of `char` variables are specified by enclosing them between single quotes. The program is terminated (`return -1`) if two numbers are not input or the simple arithmetic operator is not legal. The return value of -1 instead of 0 signals that an error took place.

[11]The variable `x` **must** be either an integer type, or one that can evaluate to an integer. In practice this means you can successfully use `switch` on `int`s and `char`s, since single characters are represented by their so-called ASCII code and thus are on an equal footing as integers – indeed are the most basic of the integer types. Do not attempt to use `switch` on a variable which is a `float` or a `double` or a string. It almost certainly won't work, and even if it appears to work, it could fail at some later point.

```
// CalculatorSwitch.cc
// Simple arithmetic calculator using switch() selection.

#include <iostream>
using namespace std;

int main()
{
   float a, b, result;
   char operation;

   // Get numbers and mathematical operator from user input
   cin >> a >> operation >> b;

   // Character constants are enclosed in single quotes
   switch(operation)
   {
   case '+':
         result = a + b;
         break;

   case '-':
         result = a - b;
         break;

   case '*':
         result = a * b;
         break;

   case '/':
         result = a / b;
         break;

   default:
         cout << "Invalid operation. Program terminated." << endl;
         return -1;
   }

   // Output result
   cout << result << endl;
   return 0;
}
```