Chapter 5

Making Decisions and Repetition

Second Thoughts. Don’t have them. Switch over.

Programming is mainly concerned with getting a computer program to demonstrate a little intelligence or to make decisions. The basis of a decision in computing lies ultimately in testing whether an expression is true or false. This chapter introduces the C++ tools available for making decisions: the if, if-else and switch statements and the conditional operator, ?, which is a specialised form of the if-else statement.

The great power of computers is their ability to perform a task over and over again (namely repetition) with phenomenal speed, accuracy and reliability. Repetition is something that we humans find very boring and difficult to do. In this chapter we introduce the three facilities offered by C++ for performing repetition: the for loop, the while loop and the do-while loop.

The chapter concludes by examining the bitwise and shift operators.

5.1 Decisions

A program frequently requires the capability to make a decision based on some known information. For example, if a car’s speed is below the speed limit then accelerate; otherwise, if the car’s speed is above the speed limit then decelerate. Given the speed of the car and the speed limit we can make this kind of accelerate or decelerate decision. Decisions occur in a variety of disguises: on-off, yes-no, true-false etc.

C++ offers us two key ways of making decisions: the if-else and switch statements. We shall begin by examining the if statement alone, followed by the if-else decision statement, after which we shall discuss the switch statement, finishing by examining the conditional operator (? :), which is a specialised form of the if-else statement.

1 Note the emphasis placed on known information. We can always make a purely random or a blind decision, but we shall not be concerned with such decisions.
5.1.1 The if Statement

Consider the program:

```cpp
// if.cpp
// illustrates the if statement
#include <iostream.h> // C++ I/O

void main ()
{
    float number ;

    // get user-entered number
    cout << "enter a positive or negative number: " ;
    cin >> number ;

    // compare number to zero
    if (number > 0)
    
        cout << number << " is greater than zero" << endl ;
}
```

The following shows a typical program–user interaction:

```
enter a positive or negative number: 10.12
10.12 is greater than zero
```

Upon execution the program extracts a number from the user and then proceeds to test whether the number is greater than zero. The keyword `if` directly followed by the required expression in parentheses `(number > 0)` performs the comparison. The general syntax of the `if` statement is:

```
if (test expression)
    statement ;  // single statement if-body
```

Figure 5.1 illustrates the `if` statement schematically. Think of the `if` statement as a box in which we have only an input and an output. If the input 'passes the test' then the box `fires` and the box produces an action. Alternatively, if the box 'fails the test' then the box produces no output.

Try running the above program but enter a negative number. What happens? Nothing. The program is so simple that it can only test whether the number entered is greater than zero.

OK: let's make the above program a bit more intelligent:

```cpp
// if.if.cpp
// illustrates the if statement
#include <iostream.h> // C++ I/O

void main ()
{
    float number ;

    // get user-entered number
```