CHAPTER 6

DECISIONS AND REPETITIONS: BASIC ACTIVITIES IN PROGRAMS

So far, our programs have been made of statements that declare variables, assignment statements, input statements, and output statements. Assignment statements, which we studied in the previous chapter, are fundamentally important action statements in computer programs. With an assignment statement, we can copy the contents of one variable into another variable, or we can produce a new value for a variable by inserting an arithmetic expression to the right side of an assignment operation. Assignment statements always write to a location in the main memory of a computer.

In this chapter, we will study more fundamental statements used in computer programming. You will learn to write statements that make decisions (selections) and perform repetitions. Decisions made in a program result in some statements being executed and others not being executed. Performing repetitions means that one or more statements can be executed many times. C++ has **`if`** and **`switch`** statements for making decisions, and **`while`**, **`for`**, and **`do-while`** statements for performing repetitions. **`if`** statements are also called **`if`** constructs. **`while`**, **`for`**, and **`do-while`** statements are called loops.

6.1 Making decisions with keywords if and else

The word "if" in our natural language expresses a condition. We can say: "If the weather is warm and sunny tomorrow, let's go to the beach." The word "if" is used in a similar way in C++. `if` is a keyword that identifies the basic decision-making mechanism of C++.

`if` statements, which we also call `if` constructs, are used to make decisions in C++. The structure of the simplest `if` statement is described in Figure 6-1. An `if` statement always contains a boolean expression that can be either true or false. Boolean expressions are named after George Boole (1815 - 1864) whose ideas have deeply influenced computing and programming.

Boolean expressions define conditions. In `if` statements, the boolean expression is given in parentheses ( ) after the keyword `if`. Every boolean expression has a truth value which is always either true or false, but not both. Provided that the boolean expression is true, the statements inside the braces { } after the boolean expression will be executed. If the boolean expression is false (i.e. not true), the statements inside the braces will not be executed, and the execution of the program continues from the statement that follows the closing brace } of the `if` statement.

```
if  (  boolean expression  )
{
     One or more statements that will be executed if the boolean
     expression, given in parentheses above, is true. These statements
     will not be executed at all if the boolean expression  is false (i.e.
     not true).
}
```

Figure 6-1. The structure of a simple if statement

```
if  (  boolean expression  )
{
     One or more statements that will be executed if the boolean
     expression, given in parentheses above, is true.
}
else
{
     One or more statements that will be executed if the boolean
     expression, given in parentheses above, is false (i.e. not true).
}
```

Figure 6-2. The structure of if-else construct

A more advanced form of `if` statement is an `if-else` construct which contains two C++ keywords, `if` and `else`. The structure of `if-else` construct is explained in Figure 6-2. The `if-else` construct has two blocks of statements, and only one block of statements will be executed. When one or more statements are inside braces { }, we can call the group of statements an embraced block of statements, or simply a block. In `if-else` constructs, depending on the truth value of the boolean expression, either the first block of statements or the second block, but never both, will be executed. The `if-else` construct thus makes a decision as to which program block will be executed.

Program **largeint.cpp** is an example where two decisions are made with keywords `if` and `else`. The first decision is made with an `if-else` construct. The second decision is made with a simple `if` statement. The program is able to find the largest of three integers that the user types in from the keyboard. First the program decides which is larger of the first two integers. Then it decides whether the third integer is larger than the largest of the first two integers.

Boolean expressions have a truth value, either true or false. To write a boolean expression we need operators that can describe situations that are either true or false. Relational operators, which are listed in Table 6-1, are common in boolean expressions. In program **largeint.cpp**, the relational operator < is used in the boolean expression

```
( first_integer  <  second_integer )
```

to test whether it is true that the value of the variable `first_integer` is less than the value of the variable `second_integer`. A common use for relational operators is to compare values of variables, but relational operators can also take numerical values as operands. For example, the boolean expression

```
( some_variable  ==  0 )
```

tests whether the contents of `some_variable` is zero. If the contents of `some_variable` is zero, then the expression above is true, otherwise it is false.

Operator ==, like most of the relational operator symbols, consists of two characters. There should be no spaces between the two characters. Writing, for example, = = will result in a compilation error. The compiler would interpret the two equal signs separated with a space as two adjacent assignment operators.

Table 6-1: The relational operators of C++

Operator symbol	Operator name
`<`	less than
`<=`	less than or equal
`>`	greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal

This single input statement reads values to all three integer variables. The input operator >> can be used in sequence, in the same way as the output operator <<.

```
//  largeint.cpp  (c) 1997 Kari Laitinen

#include  <iostream.h>

int main()
{
   int  first_integer,  second_integer,  third_integer ;
   int  found_largest_integer ;

   cout <<  "\n This program can find the largest of three"
        <<  "\n integers you enter from the keyboard."
        <<  "\n Please, enter three integers:  " ;

   cin  >>  first_integer >> second_integer >> third_integer ;

   if ( first_integer  >  second_integer )
   {
      found_largest_integer  =  first_integer ;
   }
   else
   {
      found_largest_integer  =  second_integer ;
   }

   if ( third_integer  >  found_largest_integer )
   {
      found_largest_integer  =  third_integer ;
   }

   cout  <<  "\n The largest integer is "
         <<  found_largest_integer  <<  ".\n" ;
}
```

This is the simplest form of an `if` construct, containing only the keyword `if`. If the contents of `third_integer` is greater than the contents of `found_largest_integer`, the contents of variable `third_integer` will be copied to variable `found_largest_integer`.

This is an `if-else` construct. If the contents of `first_integer` is greater than the contents of `second_integer`, the contents of `first_integer` will be copied to `found_largest_integer`. Otherwise, the contents of `second_integer` will be copied to `found_largest_integer`.

largeint.cpp - 1.+ A program to find the largest of three integers.

The relational operator >, greater than, is used to define a boolean expression which determines which variable shall be copied to **`found_largest_integer`**.

Note that there are no semicolons ; following the boolean expressions of **`if`** constructs.

```
if ( first_integer  >  second_integer )
{
   found_largest_integer  =  first_integer ;
}
else
{
   found_largest_integer  =  second_integer ;
}

if ( third_integer  >  found_largest_integer )
{
   found_largest_integer  =  third_integer ;
}
```

largeint.cpp - 1-1. The if constructs that find the largest integer.

```
D:\book2cpp>largeint

 This program can find the largest of three
 integers you enter from the keyboard.
 Please, enter three integers:  111 222 211

 The largest integer is 222.

D:\book2cpp>
```

largeint.cpp - X. The program finds 222 to be the largest of 111, 222, and 211.