Chapter 4

SELECTION STATEMENTS

The flow of control refers to the order in which a program's statements are executed. Unless directed otherwise, the normal flow of control for all programs is *sequential*. This means that statements are executed in sequence, one after another, in the order in which they are placed within the program.

Selection, repetition and function invocation structures permit the flow of control to be altered in defined ways. This chapter introduces to you C++'s selection statements. Repetition and invocation techniques are presented in Chapter 5 and 6.

4.1 SELECTION CRITERIA

Relational Operators

Relational operators are used to compare two operands for equality and to determine if one numeric value is greater than another. A Boolean value of *true* or *false* is returned after two operands are compared. The list of relational operators is given in Table 4.1

Table 4.1 Relational Operators

Operator	Description
= =	equal
!=	not equal
>	greater than
<	less than
<=	less than or equal
>=	greater than or equal

Example:

```
a = = b
(a*b) != c
s == 'y'
x<= 4
```

The value of a relational expression such as *a > 40* depends on the value stored in the variable *a*.

Logical Operators

Logical operators, AND, OR and NOT are used for creating more complex conditions. Like relational operators, a Boolean value of true or false is returned after the logical operation is executed.

When the AND operator, &&, is used with two simple expressions, the condition is true only if both individual expressions are true by themselves.

The logical OR operator, ||, is also applied with two expressions. When using the OR operator, the condition is satisfied if either one or both of the two expressions are true.

The NOT operator,!, is used to change an expression to its opposite state; thus, if the expression has any nonzero value (true),! *expression* produces a zero value (false). If an expression is false,! *expression* is true (and evaluates to false).

Example:

```
(age > 40) && (term < 10)
(age > 40) || (term < 10)
!(age > 40)
( i==j) || (a < b) || complete
```

The relational and logical operators have a hierarchy of execution similar to the arithmetic operators. The following table lists the precedence of these operators in relation to the other operators we have used.

Level	Operator	Associativity
1.	! unary - ++ --	Right to left
2.	* / %	Left to right
3.	+ -	Left to right
4.	< <= > >=	Left to right
5.	= = !=	Left to right
6.	&&	Left to right
7.	||	Left to right
8.	= += -= *= /=	Right to left

Example: Assume the following declarations:

```
char key = 'm';
int i = 5, j = 7, k = 12;
double x = 22.5;
```

Expression	Equivalent expression	Value	Interpretation
i + 2 = = k-1	(i + 2) = = (k –1)	0	false
'a' +1 = = 'b'	('a' +1) = = 'b'	1	true
25 >= x + 1.0	25 >= (x + 1.0)	1	true
key –1 > 20	(key –1) > 20	0	false

By evaluating the expressions within parentheses first, the following compound condition is evaluated as:

(6*3 = = 36/2) || (13<3*3 + 4) && !(6-2 < 5)
(18 = = 18) || (13 < 9 + 4) && !(4 < 5)
1 || (13 < 13) && ! 1
1 || 0 && 0
1 || 0
1

The *bool* Data Type

As specified by the ANSO/ISO standard, C++ has a built-in Boolean data type, *bool*, containing the two values *true* and *false*. As currently implemented, the actual values represented by the bool values, *true* and *false*, are the integer values 1 and 0, respectively. For example, consider the following program, which declares two Boolean variables:

Example 4.1.1

```
#include<iostream.h>
int main()
{
   bool t1, t2;
   t1 = true;
   t2 = false;
   cout << "The value of t1 is "<< t1
        << "\n and the value of t2 is "<< t2 << endl;
   return 0;
}
```

The output of the program is:

```
The value of t1 is 1
 and the value of t2 is 0
```

4.2 THE if-else STATEMENT

The *if-else* statement directs the computer to select a sequence of one or more statements based on the result of a comparison.

The syntax for an *if .. else* statement:

```
if (conditional expression) {
   statements;
}
else {
  statements;
}
```

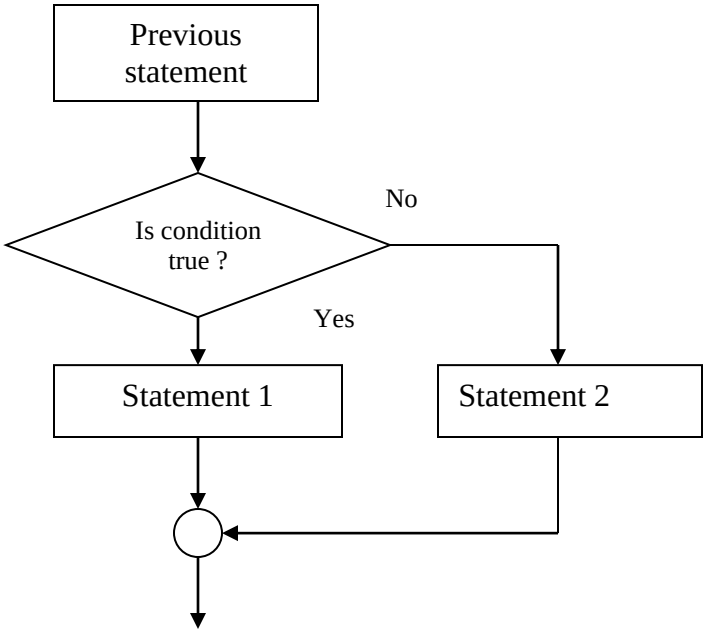

Figure 4.1 The flow chart of *if-else* statement

Example 4.2.1

We construct a C++ program for determining income taxes. Assume that these taxes are assessed at 2% of taxable incomes less than or equal to $20,000. For taxable income greater than $20,000, taxes are 2.5% of the income that exceeds $20,000 plus a fixed amount of $400. (The flowchart of the program is given in Figure 4.2.)

```
#include <iostream.h>
#include <iomanip.h>

const float LOWRATE = 0.02;     // lower tax rate
const float HIGHRATE = 0.025;   // higher tax rate
const float CUTOFF = 20000.0;   // cut off for low rate
const float FIXEDAMT = 400;     // fixed dollar amount for higher rate amounts
int main()
{
  float taxable, taxes;
  cout << "Please type in the taxable income: ";
  cin  >> taxable;

  if (taxable <= CUTOFF)
    taxes = LOWRATE * taxable;
  else
    taxes = HIGHRATE * (taxable - CUTOFF) + FIXEDAMT;
    // set output format
```

```
 cout << setiosflags(ios::fixed)
      << setiosflags(ios::showpoint)
      << setprecision(2);
 cout << "Taxes are $ " << taxes << endl;
 return 0;
}
```

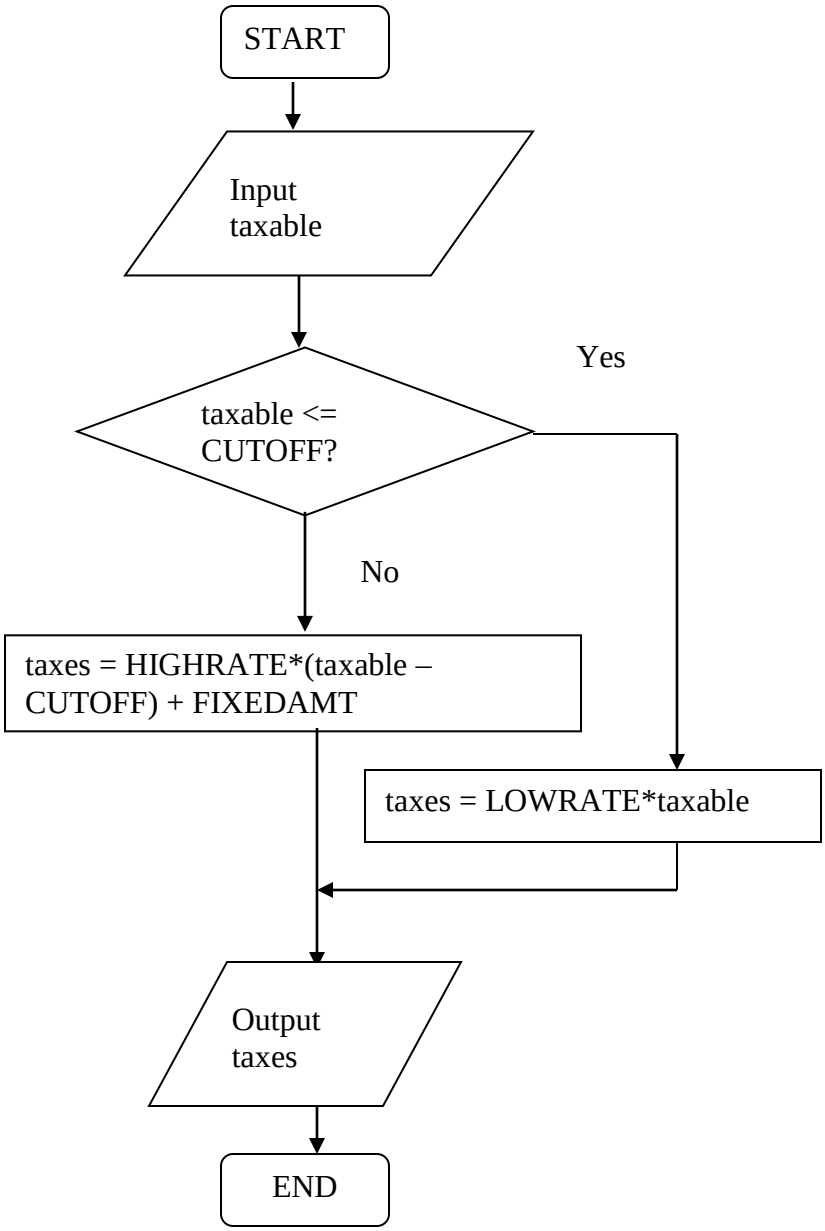

Figure 4.2 Flowchart of the Example 4.2.1

The results of the above program:

```
Please type in the taxable income: 10000
Taxes are $ 200
```

and

```
Please type in the taxable income: 30000
Taxes are $ 650
```

Block Scope

All statements within a compound statement constitute a single *block* of code, and any variable declared within such a block only is valid within the block.

The location within a program where a variable can be used formally referred to as the *scope* of the variable.

Example:

```
{ // start of outer block
   int a = 25;
   int b = 17;
   cout << "The value of a is " << a
          << " and b is "  << b << endl;
   {   // start of inner block
      float a = 46.25;
      int c = 10;
      cout << " a is now " << a
             << "b is now " << b
             << " and c is " << c << endl;
    }
    cout << " a is now " << a
             << "b is now " << b << endl;
} // end of outer block
```

The output is

```
The value of a is  25   and b is 17
a is now  46.25 b is now 17 and c is 10
a is now  25 b is now  17
```

One-way Selection

A useful modification of the *if-else* statement involves omitting the *else* part of the statement. In this case, the *if* statement takes a shortened format:

if (conditional expression) {
   statements;
}

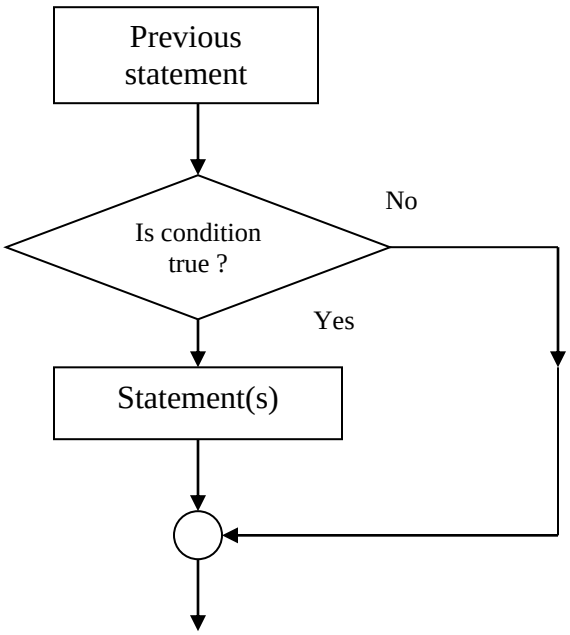

Figure 4.3 The flow chart of one-way *if* statement

Example 4.2.2
The following program displays an error message for the grades that is less than 0 or more than 100.

```
#include <iostream.h>
int main()
{
   int grade;

  cout << "\nPlease enter a grade: ";
  cin  >> grade;

  if(grade < 0 || grade > 100)
    cout << " The grade is not valid\n";
  return 0;
}
```

4.3 NESTED if STATEMENT

An *if-else* statement can contain simple or compound statements. Any valid C++ statement can be used, including another *if-else* statement. Thus, one or more *if-else*

statements can be included within either part of an *if-else* statement. The inclusion of one or more *if* statement within an existing if statement is called a *nested if statement*.

The if-else Chain

When an *if* statement is included in the *else* part of an existing *if* statement, we have an *if-else chain*.

```
if (expression-1)
  statement-1
else if (expression-2)
  statement-2
else
  statement-3
```

Example 4.3.1

```
// This program can solve quadratic equation
#include <iostream.h>
#include <math.h>
#include <iomanip.h>
int main()
{
  double a, b, c, del, x1, x2;
   cout << "Enter the coefficients of the equation: "<< endl;
   cin  >> a >> b >> c;
   del = b*b – 4.0*a*c;
   if (del == 0.0)
   {
      x1 = x2 = -b/(2*a);
      cout << "x1 = " << x1 << setw(20) << "x2 = " << x2 << endl;
    }
   else if (del > 0.0)
   {
      x1 = (-b + sqrt(del))/(2*a);
      x2 = (-b – sqrt(del))/(2*a);
      cout << "x1 = " << x1 << setw(20) << "x2 = " << x2 << endl;
    }
   else
   cout << "There is no solution\n";
  return 0;
}
```

The output of the above program:

```
Enter the coefficients of the equation:
1   5   6
```

x1 = -2.0 x2 = -3.0

Example 4.3.2

The following program calculates the monthly income of a computer salesperson using the following commission schedule:

Monthly Sales	Income
Greater than or equal to $50,000	$375 plus 16% of sales
Less than $50,000 but greater than or equal to $40,000	$350 plus 14% of sales
Less than $40,000 but greater than or equal to $30,000	$325 plus 12% of sales
Less than $30,000 but greater than or equal to $20,000	$300 plus 9% of sales
Less than $20,000 but greater than or equal to $10,000	$250 plus 5% of sales
Less than $10,000	$200 plus 3% of sales

```
#include <iostream.h>
#include <iomanip.h>

int main()
{
 float monthlySales, income;

 cout << "\nEnter the value of monthly sales: ";
 cin  >> monthlySales;

 if (monthlySales >= 50000.00)
   income = 375.00 + .16 * monthlySales;
 else if (monthlySales >= 40000.00)
   income = 350.00 + .14 * monthlySales;
 else if (monthlySales >= 30000.00)
   income = 325.00 + .12 * monthlySales;
 else if (monthlySales >= 20000.00)
   income = 300.00 + .09 * monthlySales;
 else if (monthlySales >= 10000.00)
   income = 250.00 + .05 * monthlySales;
 else
   income = 200.00 + .03 * monthlySales;

  // set output format
 cout << setiosflags(ios::fixed)
      << setiosflags(ios::showpoint)
      << setprecision(2);

 cout << "The income is $" << income << endl;

 return 0;
```

}

The output of the program:

Enter the value of monthly sales: 36243.89
The income is $4674.27

4.4 THE switch STATEMENT

The *switch* statement controls program flow by executing a set of statements depending on the value of an expression.

Note: The value of expression must be an integer data type, which includes the *char*, *int*, *long int*, and *short* data types.

The syntax for the *switch* statement:

```
switch(expression){
    case label:
        statement(s);
        break;
   case label;
     statement(s);
        break;
  default:
     statement(s);
}
```

The expression in the *switch* statement must evaluate to an integer result. The *switch* expression's value is compared to each of these case values in the order in which these values are listed until a match is found. When a match occurs, execution begins with the statement following the match.

If the value of the expression does not match any of the case values, no statement is executed unless the keyword *default* is encountered. If the value of the expression does not match any of the case values, program execution begins with the statement following the word *default*.

The *break* statement is used to identify the end of a particular case and causes an immediate exit from the *switch* statement. If the *break* statements are omitted, all cases following the matching case value, including the default case, are executed.

Example 4.4.1

```
#include <iostream.h>
int main()
```

```
{
  int iCity;

  cout << "Enter a number to find the state where a city is located. "<< endl;
  cout << “1. Boston” << endl;
  cout << "2. Chicago" << endl;
  cout << "3. Los Angeles” << endl;
  cout << "4. Miami” << endl;
  cout << "5. Providence” << endl;
  cin  >> iCity;
  switch (iCity)
  {
    case 1:
      cout << "Boston is in Massachusetts " << endl;
      break;
    case 2:
      cout << "Chicago is in Illinois " << endl;
      break;
    case 3:
      cout << "Los Angeles is in California " << endl;
      break;
    case 4:
      cout << "Miami is in Florida " << endl;
      break;
    case 5:
      cout << "Providence is in Rhode Island " << endl;
      break;
    default:
      cout << “You didn’t select one of the five cities” << endl;
  }     // end of switch
  return 0;
}
```

The output of the above program:

```
Enter a number to find the state where a city is located.
1. Boston
2. Chicago
3. Los Angeles
4. Miami
5. Providence

3
Los Angeles is in California
```

The *switch* statement is a clean way to implement multi-way selection (i.e., selecting from among a number of different execution paths), but it requires an expression that evaluates to an integral value at compile-time.

When writing a *switch* statement, you can use multiple case values to refer to the same set of statements; the *default* label is optional. For example, consider the following example:

```
switch(number)
{
   case 1:
      cout << "Have a Good Morning\n";
      break;
   case 2:
      cout << "Have a Happy Day\n";
      break;
   case 3:
   case 4:
   case 5:
      cout << "Have a Nice Evening\n";
  }
```

4.5 CONDITIONAL EXPRESSIONS

A conditional expression uses the conditional operator,?, and provides an alternative way of expression a simple *if-else* statement.

The syntax of a conditional expression is:

expression1 ? expression2 : expression3

If the value of *expression1* is nonzero (true), *expresson2* is evaluated; otherwise, *expression3* is evaluated. The value for the complete conditional expression is the value of either *expression2* or *expression3* depending on which expression was evaluated.

Example: The *if* statement:

```
if (hours > 40)
   rate = 0.45;
else
    rate = 0.02;
```

can be replaced with the following one-line statement:

```
rate = (hours > 40) ? 0.45 : 0.02;
```

4.6 THE enum SPECIFIER

An enumerated data type is a way of attaching names to numbers, thereby giving more meaning to anyone reading the code. The *enum* specifier creates an enumerated data type, which is simply a user-defined list of values that is given its own data type name. Such data types are identified by the reserved word *enum* followed by an optional user-selected name for the data type and a listing of acceptable values for the data type.

Example:

```
enum day { mon, tue, wed, thu, fri, sat, sun}
enum color {red, green, yellow};
```

Any variable declared to be of type *color* can take only a value of *red* or *green* or *yellow*. Any variable declared to be of type *day* can take only a value among seven given values.

The statement

```
enum day a, b,c;
```

declares the variables *a*, *b*, and *c* to be of type *day*.

Internally, the acceptable values of each enumerated data type are ordered and assigned sequential integer values beginning with 0. For example, for the values of the user-defined type *color*, the correspondences created by the C++ compiler are that *red* is equivalent to 0, *green* is equivalent to 1, and *yellow* is equivalent to 2. The equivalent numbers are required when inputting values using *cin* or displaying values using *cout*.

Example 4.6.1

```
#include <iostream.h>
int main()
{
   enum color{red, green, yellow};
   enum color crayon = red;
   cout << "\nThe color is " << crayon << endl;
   cout << "Enter a value:  ";
   cin >> crayon;
   if (crayon == red)
     cout << "The crayon is  red." << endl;
   else if (crayon == green)
     cout << "The crayon is green." << endl;
   else if (crayon== yellow)
     cout << "The crayon is yellow." << endl;
   else
     cout << "The color is not defined. \n" << endl;
   return 0;
```

```
}
```

The output of the above program:

```
The color is 0
Enter a value: 2
The crayon is yellow.
```