CHAPTER 4

Making Decisions

TOPICS

4.1 Relational Operators

CONCEPT: **Relational operators allow you to compare numeric and `char` values and determine whether one is greater than, less than, equal to, or not equal to another.**

So far, the programs you have written follow this simple scheme:

- Gather input from the user.
- Perform one or more calculations.
- Display the results on the screen.

Computers are good at performing calculations, but they are also quite adept at comparing values to determine if one is greater than, less than, or equal to the other. These types of operations are valuable for tasks such as examining sales figures, determining profit and loss, checking a number to ensure it is within an acceptable range, and validating the input given by a user.

Numeric data is compared in C++ by using relational operators. Each relational operator determines whether a specific relationship exists between two values. For example,

the greater-than operator (>) determines if a value is greater than another. The equality operator (==) determines if two values are equal. Table 4-1 lists all of C++'s relational operators.

Table 4-1

Relational Operators	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

All of the relational operators are binary, which means they use two operands. Here is an example of an expression using the greater-than operator:

```
x > y
```

This expression is called a *relational expression*. It is used to determine whether `x` is greater than `y`. The following expression determines whether `x` is less than `y`:

```
x < y
```

Table 4-2 shows examples of several relational expressions that compare the variables `x` and `y`.

Table 4-2

Expression	What the Expression Means
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y?
`x == y`	Is x equal to y?
`x != y`	Is x equal to y?

NOTE: All the relational operators have left-to-right associativity. Recall that associativity is the order in which an operator works with its operands.

The Value of a Relationship

So, how are relational expressions used in a program? Remember, all expressions have a value. Relational expressions are also known as *Boolean expressions*, which means their value can only be *true* or *false*. If `x` is greater than `y`, the expression `x > y` will be true, while the expression `y == x` will be false.

The == operator determines whether the operand on its left is equal to the operand on its right. If both operands have the same value, the expression is true. Assuming that `a` is 4, the following expression is true:

```
a == 4
```

But the following is false:

```
a == 2
```

WARNING! Notice the equality operator is two = symbols together. Don't confuse this operator with the assignment operator, which is one = symbol. The == operator determines whether a variable is equal to another value, but the = operator assigns the value on the operator's right to the variable on its left. There will be more about this later in the chapter.

A couple of the relational operators actually test for two relationships. The >= operator determines whether the operand on its left is greater than *or* equal to the operand on the right. Assuming that `a` is 4, `b` is 6, and `c` is 4, both of the following expressions are true:

```
b >= a
a >= c
```

But the following is false:

```
a >= 5
```

The <= operator determines whether the operand on its left is less than *or* equal to the operand on its right. Once again, assuming that `a` is 4, `b` is 6, and `c` is 4, both of the following expressions are true:

```
a <= c
b <= 10
```

But the following is false:

```
b <= a
```

The last relational operator is `!=`, which is the not-equal operator. It determines whether the operand on its left is not equal to the operand on its right, which is the opposite of the == operator. As before, assuming `a` is 4, `b` is 6, and `c` is 4, both of the following expressions are true:

```
a != b
b != c
```

These expressions are true because `a` is *not* equal to `b` and `b` is *not* equal to `c`. But the following expression is false because `a` *is* equal to `c`:

```
a != c
```

Table 4-3 shows other relational expressions and their true or false values.

Table 4-3 (Assume x is 10 and y is 7.)

Expression	Value
`x < y`	False, because x is not less than y.
`x > y`	True, because x is greater than y.
`x >= y`	True, because x is greater than or equal to y.
`x <= y`	False, because x is not less than or equal to y.
`y != x`	True, because y is not equal to x.

What Is Truth?

The question "What is truth?" is one you would expect to find in a philosophy book, not a C++ programming text. It's a good question for us to consider, though. If a relational expression can be either true or false, how are those values represented internally in a program? How does a computer store *true* in memory? How does it store *false*?

As you saw in Program 2-17, those two abstract states are converted to numbers. In C++, relational expressions represent true states with the number 1 and false states with the number 0.

NOTE: As you will see later in this chapter, 1 is not the only value regarded as true.

To illustrate this more fully, look at Program 4-1.

Program 4-1

```
1  // This program displays the values of true and false states.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       bool trueValue, falseValue;
8       int x = 5, y = 10;
9
10      trueValue = x < y;
11      falseValue = y == x;
12
13      cout << "True is " << trueValue << endl;
14      cout << "False is " << falseValue << endl;
15      return 0;
16 }
```

Program Output

```
True is 1
False is 0
```

Let's examine the statements containing the relational expressions, in lines 10 and 11, a little closer:

```
trueValue = x < y;
falseValue = y == x;
```

These statements may seem odd because they are assigning the value of a comparison to a variable. In line 10 the variable `trueValue` is being assigned the result of `x < y`. Since `x` is less than `y`, the expression is true, and the variable `trueValue` is assigned the value 1. In line 11 the expression `y == x` is false, so the variable `falseValue` is set to 0. Table 4-4 shows examples of other statements using relational expressions and their outcomes.

NOTE: Relational expressions have a higher precedence than the assignment operator. In the statement

```
z = x < y;
```

the expression `x < y` is evaluated first, and then its value is assigned to `z`.

Table 4-4 (Assume `x` is 10, `y` is 7, and `z`, `a`, and `b` are `int`s or `bool`s)

Statement	Outcome
`z = x < y`	`z` is assigned 0 because `x` is not less than `y`.
`cout << (x > y);`	Displays 1 because `x` is greater than `y`.
`a = x >= y;`	`a` is assigned 1 because `x` is greater than or equal to `y`.
`cout << (x <= y);`	Displays 0 because `x` is not less than or equal to `y`.
`b = y != x;`	`b` is assigned 1 because `y` is not equal to `x`.

When writing statements such as these, it sometimes helps to enclose the relational expression in parentheses, such as:

```
trueValue = (x < y);
falseValue = (y == x);
```

As interesting as relational expressions are, we've only scratched the surface of how to use them. In this chapter's remaining sections you will see how to get the most from relational expressions by using them in statements that take action based on the results of the comparison.

Checkpoint

4.1 Assuming `x` is 5, `y` is 6, and `z` is 8, indicate by circling the T or F whether each of the following relational expressions is true or false:

A) `x == 5` T F
B) `7 <= (x + 2)` T F
C) `z < 4` T F
D) `(2 + x) != y` T F
E) `z != 4` T F
F) `x >= 9` T F
G) `x <= (y * 2)` T F

4.2 Indicate whether the following statements about relational expressions are correct or incorrect.

A) `x <= y` is the same as `y > x`.
B) `x != y` is the same as `y >= x`.
C) `x >= y` is the same as `y <= x`.

4.3 Answer the following questions with a yes or no.

A) If it is true that `x > y` and it is also true that `x < z`, does that mean `y < z` is true?
B) If it is true that `x >= y` and it is also true that `z == x`, does that mean that `z == y` is true?
C) If it is true that `x != y` and it is also true that `x != z`, does that mean that `z != y` is true?

4.4 What will the following program display?

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

int main ()
{
    int a = 0, b = 2, x = 4, y = 0;

    cout << (a == b) << endl;
    cout << (a != y) << endl;
    cout << (b <= x) << endl;
    cout << (y > a) << endl;
    return 0;
}
```

4.2 The `if` Statement

CONCEPT: **The `if` statement can cause other statements to execute only under certain conditions.**

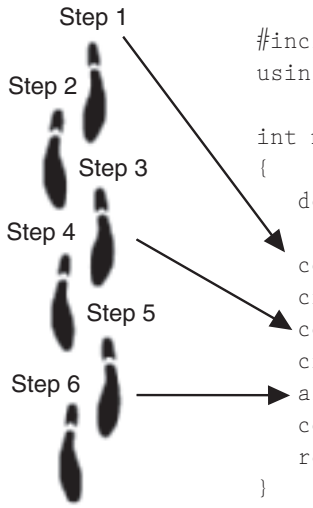

VideoNote
The `if` Statement

You might think of the statements in a procedural program as individual steps taken as you are walking down a road. To reach the destination, you must start at the beginning and take each step, one after the other, until you reach the destination. The programs you have written so far are like a "path" of execution for the program to follow.

The type of code in Figure 4-1 is called a *sequence structure* because the statements are executed in sequence, without branching off in another direction. Programs often need more than one path of execution, however. Many algorithms require a program to execute some statements only under certain circumstances. This can be accomplished with a *decision structure*.

In a decision structure's simplest form, a specific action is taken only when a specific condition exists. If the condition does not exist, the action is not performed. The flowchart in Figure 4-2 shows the logic of a decision structure. The diamond symbol represents a yes/no

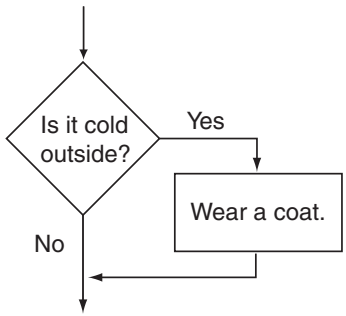
Figure 4-1




```
// A program to calculate the area of a rectangle

#include <iostream>
using namespace std;

int main()
{
   double length, width, area;

   cout << "Enter the length of the rectangle: ";
   cin >> length;
   cout << "Enter the width of the rectangle: ";
   cin >> width;
   area = length * width;
   cout << "The area is: " << area << endl
   return 0;
}
```

question or a true/false condition. If the answer to the question is yes (or if the condition is true), the program flow follows one path, which leads to an action being performed. If the answer to the question is no (or the condition is false), the program flow follows another path, which skips the action.

Figure 4-2

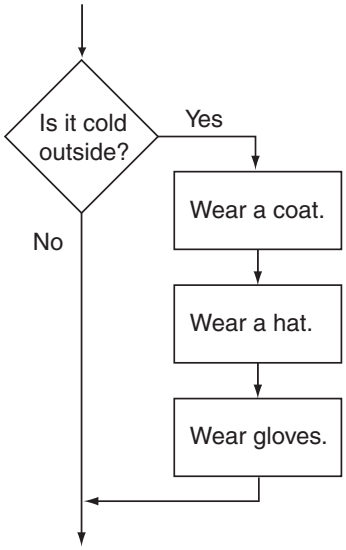


In the flowchart, the action "Wear a coat" is performed only when it is cold outside. If it is not cold outside, the action is skipped. The action is *conditionally executed* because it is performed only when a certain condition (cold outside) exists. Figure 4-3 shows a more elaborate flowchart, where three actions are taken only when it is cold outside.

We perform mental tests like these every day. Here are some other examples:

If the car is low on gas, stop at a service station and get gas.
If it's raining outside, go inside.
If you're hungry, get something to eat.

Figure 4-3



One way to code a decision structure in C++ is with the `if` statement. Here is the general format of the `if` statement:

```
if (expression)
   statement;
```

The `if` statement is simple in the way it works: If the value of the expression inside the parentheses is true, the very next *`statement`* is executed. Otherwise, it is skipped. The *`statement`* is *conditionally executed* because it only executes under the condition that the expression in the parentheses is true. Program 4-2 shows an example of an `if` statement. The user enters three test scores, and the program calculates their average. If the average is greater than 95, the program congratulates the user on obtaining a high score.

Program 4-2

```
1  // This program averages three test scores
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  int main()
7  {
8     const int HIGH_SCORE = 95;  // A high score is 95 or greater
9     int score1, score2, score3; // To hold three test scores
10    double average;             // TO hold the average score
11
```

```
12      // Get the three test scores.
13      cout << "Enter 3 test scores and I will average them: ";
14      cin >> score1 >> score2 >> score3;
15
16      // Calculate and display the average score.
17      average = (score1 + score2 + score3) / 3.0;
18      cout << fixed << showpoint << setprecision(1);
19      cout << "Your average is " << average << endl;
20
21      // If the average is a high score, congratulate the user.
22      if (average > HIGH_SCORE)
23         cout << "Congratulations! That's a high score!\n";
24      return 0;
25  }
```

Program Output with Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 80.0
```

Program Output with Different Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 100 100 100 [Enter]
Your average is 100.0
Congratulations! That's a high score!
```

Lines 22 and 23 cause the congratulatory message to be printed:

```
if (average > HIGH_SCORE)
   cout << "Congratulations! That's a high score!\n";
```

The cout statement in line 23 is executed only if the average is greater than 95, the value of the HIGH_SCORE constant. If the average is not greater than 95, the cout statement is skipped. Figure 4-4 shows the logic of this if statement.

Figure 4-4

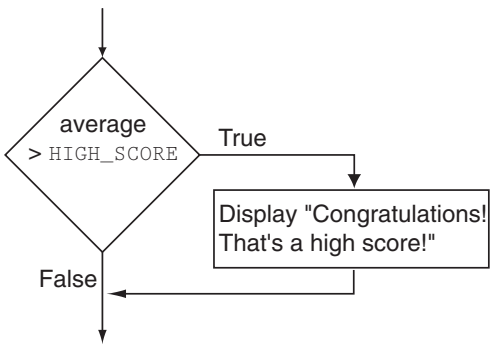


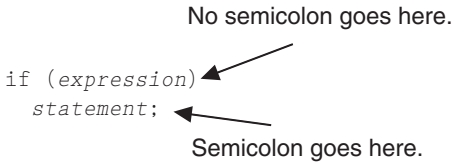
Table 4-5 shows other examples of if statements and their outcomes.

Table 4-5

Statement	Outcome
`if (hours > 40)` `  overTime = true;`	Assigns `true` to the `bool` variable `overTime` only if `hours` is greater than 40
`if (value > 32)` `  cout << "Invalid number\n";`	Displays the message "Invalid number" only if `value` is greater than 32
`if (overTime == true)` `  payRate *= 2;`	Multiplies `payRate` by 2 only if `overTime` is equal to `true`

Be Careful with Semicolons

Semicolons do not mark the end of a line, but the end of a complete C++ statement. The `if` statement isn't complete without the conditionally executed statement that comes after it. So, you must not put a semicolon after the `if (`*`expression`*`)` portion of an `if` statement.



If you inadvertently put a semicolon after the `if` part, the compiler will assume you are placing a null statement there. The *null statement* is an empty statement that does nothing. This will prematurely terminate the `if` statement, which disconnects it from the statement that follows it. The statement following the `if` will always execute, as shown in Program 4-3.

Program 4-3

```
1  // This program demonstrates how a misplaced semicolon
2  // prematurely terminates an if statement.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       int x = 0, y = 10;
9
10      cout << "x is " << x << " and y is " << y << endl;
11      if (x > y); // Error! Misplaced semicolon
12         cout << "x is greater than y\n"; //This is always executed.
13      return 0;
14 }
```

Program Output

```
x is 0 and y is 10
x is greater than y
```

Programming Style and the if Statement

Even though if statements usually span more than one line, they are technically one long statement. For instance, the following if statements are identical except in style:

```
if (a >= 100)
   cout << "The number is out of range.\n";
if (a >= 100) cout << "The number is out of range.\n";
```

In both the examples above, the compiler considers the if part and the cout statement as one unit, with a semicolon properly placed at the end. Indention and spacing are for the human readers of a program, not the compiler. Here are two important style rules you should adopt for writing if statements:

- The conditionally executed statement should appear on the line after the if statement.
- The conditionally executed statement should be indented one "level" from the if statement.

NOTE: In most editors, each time you press the tab key, you are indenting one level.

By indenting the conditionally executed statement you are causing it to stand out visually. This is so you can tell at a glance what part of the program the if statement executes. This is a standard way of writing if statements and is the method you should use.

NOTE: Indentation and spacing are for the human readers of a program, not the compiler. Even though the cout statement following the if statement in Program 4-3 is indented, the semicolon still terminates the if statement.

Comparing Floating-Point Numbers

Because of the way that floating-point numbers are stored in memory, rounding errors sometimes occur. This is because some fractional numbers cannot be exactly represented using binary. So, you should be careful when using the equality operator (==) to compare floating point numbers. For example, Program 4-4 uses two double variables, a and b. Both variables are initialized to the value 1.5. Then, the value 0.0000000000000001 is added to a. This should make a's contents different than b's contents. Because of a round-off error, however, the two variables are still the same.

Program 4-4

```
1  // This program demonstrates how floating-point
2  // round-off errors can make equality operations unreliable.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
```

(program continues)

Program 4-4 *(continued)*

```
 8       double a = 1.5;                 // a is 1.5.
 9       double b = 1.5;                 // b is 1.5.
10
11       a += 0.0000000000000001; // Add a little to a.
12       if (a == b)
13          cout << "Both a and b are the same.\n";
14       else
15          cout << "a and b are not the same.\n";
16
17       return 0;
18    }
```

Program Output

```
Both a and b are the same.
```

To prevent round-off errors from causing this type of problem, you should stick with greater-than and less-than comparisons with floating-point numbers.

And Now Back to Truth

Now that you've gotten your feet wet with relational expressions and `if` statements, let's look at the subject of truth again. You have seen that a relational expression has the value 1 when it is true and 0 when false. In the world of the `if` statement, however, the concept of truth is expanded. 0 is still false, but all values other than 0 are considered true. This means that any value, even a negative number, represents true as long as it is not 0.

Just as in real life, truth is a complicated thing. Here is a summary of the rules you have seen so far:

- When a relational expression is true it has the value 1.
- When a relational expression is false it has the value 0.
- Any expression that has the value 0 is considered false by the `if` statement. This includes the `bool` value `false`, which is equivalent to 0.
- Any expression that has any value other than 0 is considered true by the `if` statement. This includes the `bool` value `true`, which is equivalent to 1.

The fact that the `if` statement considers any nonzero value as true opens many possibilities. Relational expressions are not the only conditions that may be tested. For example, the following is a legal `if` statement in C++:

```
if (value)
    cout << "It is True!";
```

The `if` statement above does not test a relational expression, but rather the contents of a variable. If the variable, `value`, contains any number other than 0, the message "`It is True!`" will be displayed. If `value` is set to 0, however, the `cout` statement will be skipped. Here is another example:

```
if (x + y)
   cout << "It is True!";
```

In this statement the sum of x and y is tested like any other value in an if statement: 0 is false and all other values are true. You may also use the return value of function calls as conditional expressions. Here is an example that uses the pow function:

```
if (pow(a, b))
   cout << "It is True!";
```

This if statement uses the pow function to raise a to the power of b. If the result is anything other than 0, the cout statement is executed. This is a powerful programming technique that you will learn more about in Chapter 6.

Don't Confuse == With =

Earlier you saw a warning not to confuse the equality operator (==) with the assignment operator (=), as in the following statement:

```
if (x = 2) //Caution here!
   cout << "It is True!";
```

The statement above does not determine whether x is equal to 2, it assigns x the value 2! Furthermore, the cout statement will *always* be executed because the expression x = 2 is always true.

This occurs because the value of an assignment expression is the value being assigned to the variable on the left side of the = operator. That means the value of the expression x = 2 is 2. Since 2 is a nonzero value, it is interpreted as a true condition by the if statement. Program 4-5 is a version of Program 4-2 that attempts to test for a perfect average of 100. The = operator, however, was mistakenly used in the if statement.

Program 4-5

```
1  // This program averages 3 test scores. The if statement
2  // uses the = operator, but the == operator was intended.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9      int score1, score2, score3; // To hold three test scores
10     double average;             // TO hold the average score
11
12     // Get the three test scores.
13     cout << "Enter 3 test scores and I will average them: ";
14     cin >> score1 >> score2 >> score3;
15
16     // Calculate and display the average score.
17     average = (score1 + score2 + score3) / 3.0;
18     cout << fixed << showpoint << setprecision(1);
19     cout << "Your average is " << average << endl;
20
```

(program continues)

Program 4-5 *(continued)*

```
21       // Our intention is to congratulate the user
22       // for having a perfect score. But, this doesn't work.
23       if (average = 100) // WRONG! This is an assignment!
24          cout << "Congratulations! That's a perfect score!\n";
25       return 0;
26    }
```

Program Output with Example Input Shown in Bold

```
Enter three test scores and I will average them: 80 90 70 [Enter]
Your average is 80.0
Congratulations! That's a perfect score!
```

Regardless of the average score, this program will print the message congratulating the user on a perfect score.

Checkpoint

4.5 Write an `if` statement that performs the following logic: if the variable `x` is equal to 20, then assign 0 to the variable `y`.

4.6 Write an `if` statement that performs the following logic: if the variable `price` is greater than 500, then assign 0.2 to the variable `discountRate`.

4.7 Write an `if` statement that multiplies `payRate` by 1.5 if `hours` is greater than 40.

4.8 TRUE or FALSE: Both of the following `if` statements perform the same operation.

```
if (sales > 10000)
   commissionRate = 0.15;

if (sales > 10000) commissionRate = 0.15;
```

4.9 TRUE or FALSE: Both of the following `if` statements perform the same operation.

```
if (calls == 20)
   rate *= 0.5;

if (calls = 20)
   rate *= 0.5;
```

4.3 Expanding the `if` Statement

CONCEPT: **The `if` statement can conditionally execute a block of statements enclosed in braces.**

What if you want an `if` statement to conditionally execute a group of statements, not just one line? For instance, what if the test averaging program needed to use several `cout`

statements when a high score was reached? The answer is to enclose all of the conditionally executed statements inside a set of braces. Here is the format:

```
if (expression)
{
   statement;
   statement;
   // Place as many statements here as necessary.
}
```

Program 4-6, another modification of the test-averaging program, demonstrates this type of `if` statement.

Program 4-6

```
1  // This program averages 3 test scores.
2  // It demonstrates an if statement executing
3  // a block of statements.
4  #include <iostream>
5  #include <iomanip>
6  using namespace std;
7
8  int main()
9  {
10     const int HIGH_SCORE = 95;         // A high score is 95 or greater
11     int score1, score2, score3;        // To hold three test scores
12     double average;                    // TO hold the average score
13
14     // Get the three test scores.
15     cout << "Enter 3 test scores and I will average them: ";
16     cin >> score1 >> score2 >> score3;
17
18     // Calculate and display the average score.
19     average = (score1 + score2 + score3) / 3.0;
20     cout << fixed << showpoint << setprecision(1);
21     cout << "Your average is " << average << endl;
22
23     // If the average is high, congratulate the user.
24     if (average > HIGH_SCORE)
25     {
26        cout << "Congratulations!\n";
27        cout << "That's a high score.\n";
28        cout << "You deserve a pat on the back!\n";
29     }
30     return 0;
31 }
```

(program output continues)

Program 4-6 *(continued)*

Program Output with Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 100 100 100 [Enter]
Your average is 100.0
Congratulations!
That's a high score.
You deserve a pat on the back!
```

Program Output with Different Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 80.0
```

Program 4-6 prints a more elaborate message when the average score is greater than 95. The `if` statement was expanded to execute three `cout` statements when `highScore` is set to `true`. Enclosing a group of statements inside a set of braces creates a *block* of code. The `if` statement will execute all the statements in the block, in the order they appear, only when `average` is greater than 95. Otherwise, the block will be skipped.

Notice all the statements inside the braces are indented. As before, this visually separates the statements from lines that are not indented, making it more obvious they are part of the `if` statement.

NOTE: Anytime your program has a block of code, all the statements inside the braces should be indented.

Don't Forget the Braces!

If you intend to conditionally execute a block of statements with an `if` statement, don't forget the braces. Remember, without a set of braces, the `if` statement only executes the very next statement. Program 4-7 shows the test-averaging program with the braces inadvertently left out of the `if` statement's block.

Program 4-7

```
1  // This program averages 3 test scores. The braces
2  // were inadvertently left out of the if statement.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9       const int HIGH_SCORE = 95;          // A high score is 95 or greater
10      int score1, score2, score3;         // To hold three test scores
11      double average;                     // To hold the average score
12
```

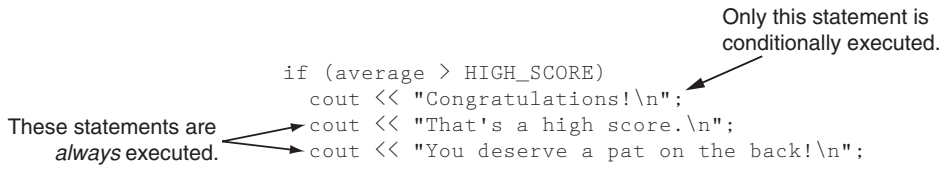
```
13      // Get the three test scores.
14      cout << "Enter 3 test scores and I will average them: ";
15      cin >> score1 >> score2 >> score3;
16
17      // Calculate and display the average score.
18      average = (score1 + score2 + score3) / 3.0;
19      cout << fixed << showpoint << setprecision(1);
20      cout << "Your average is " << average << endl;
21
22      // ERROR! This if statement is missing its braces!
23      if (average > HIGH_SCORE)
24         cout << "Congratulations!\n";
25         cout << "That's a high score.\n";
26         cout << "You deserve a pat on the back!\n";
27      return 0;
28  }
```

Program Output with Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 80
That's a high score.
You deserve a pat on the back!
```

The `cout` statements in lines 25 and 26 are always executed, even when `average` is not greater than 95. Because the braces have been removed, the `if` statement only controls execution of line 24. This is illustrated in Figure 4-5.

Figure 4-5



Checkpoint

4.10 Write an `if` statement that performs the following logic: if the variable `sales` is greater than 50,000, then assign 0.25 to the `commissionRate` variable, and assign 250 to the `bonus` variable.

4.11 The following code segment is syntactically correct, but it appears to contain a logic error. Can you find the error?

```
if (interestRate > .07)
   cout << "This account earns a $10 bonus.\n";
   balance += 10.0;
```

4.4 The `if/else` Statement

CONCEPT: The `if/else` statement will execute one group of statements if the expression is true, or another group of statements if the expression is false.

VideoNote
The `if/else` statement

The `if/else` statement is an expansion of the `if` statement. Here is its format:

```
if (expression)
    statement or block
else
    statement or block
```

As with the `if` statement, an expression is evaluated. If the expression is true, a statement or block of statements is executed. If the expression is false, however, a separate group of statements is executed. Program 4-8 uses the `if/else` statement along with the modulus operator to determine if a number is odd or even.

Program 4-8

```
1  // This program uses the modulus operator to determine
2  // if a number is odd or even. If the number is evenly divisible
3  // by 2, it is an even number. A remainder indicates it is odd.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9       int number;
10
11      cout << "Enter an integer and I will tell you if it\n";
12      cout << "is odd or even. ";
13      cin >> number;
14      if (number % 2 == 0)
15         cout << number << " is even.\n";
16      else
17         cout << number << " is odd.\n";
18      return 0;
19 }
```

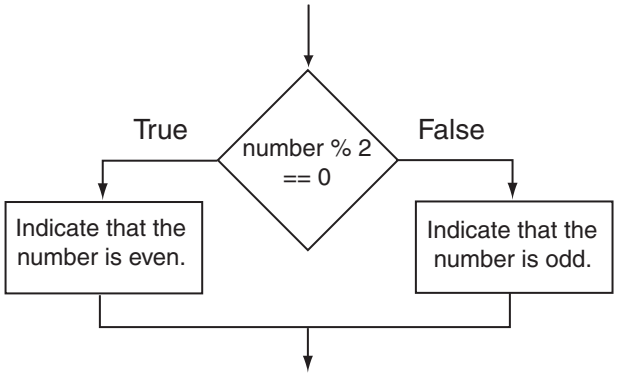
Program Output with Example Input Shown in Bold

```
Enter an integer and I will tell you if it
is odd or even. 17 [Enter]
17 is odd.
```

The `else` part at the end of the `if` statement specifies a statement that is to be executed when the expression is false. When `number % 2` does not equal 0, a message is printed indicating the number is odd. Note that the program will only take one of the two paths in the `if/else` statement. If you think of the statements in a computer program as steps

taken down a road, consider the `if/else` statement as a fork in the road. Instead of being a momentary detour, like an `if` statement, the `if/else` statement causes program execution to follow one of two exclusive paths. The flowchart in Figure 4-6 shows the logic of this `if/else` statement.

Figure 4-6



Notice the programming style used to construct the `if/else` statement. The word `else` is at the same level of indention as `if`. The statement whose execution is controlled by `else` is indented one level. This visually depicts the two paths of execution that may be followed.

Like the `if` part, the `else` part controls a single statement. If you wish to control more than one statement with the `else` part, create a block by writing the lines inside a set of braces. Program 4-9 shows this as a way of handling a classic programming problem: *division by zero*.

Division by zero is mathematically impossible to perform, and it normally causes a program to crash. This means the program will prematurely stop running, sometimes with an error message. Program 4-9 shows a way to test the value of a divisor before the division takes place.

Program 4-9

```
1  // This program asks the user for two numbers, num1 and num2.
2  // num1 is divided by num2 and the result is displayed.
3  // Before the division operation, however, num2 is tested
4  // for the value 0. If it contains 0, the division does not
5  // take place.
6  #include <iostream>
7  using namespace std;
8
9  int main()
10 {
11     double num1, num2, quotient;
12
13     // Get the first number.
14     cout << "Enter a number: ";
15     cin >> num1;
16
17     // Get the second number.
18     cout << "Enter another number: ";
```

(program continues)

Program 4-9 *(continued)*

```
19      cin >> num2;
20
21      // If num2 is not zero, perform the division.
22      if (num2 == 0)
23      {
24         cout << "Division by zero is not possible.\n";
25         cout << "Please run the program again and enter\n";
26         cout << "a number other than zero.\n";
27      }
28      else
29      {
30         quotient = num1 / num2;
31         cout << "The quotient of " << num1 << " divided by ";
32         cout << num2 << " is " << quotient << ".\n";
33      }
34      return 0;
35  }
```

Program Output with Example Input Shown in Bold

```
Enter a number: 10 [Enter]
Enter another number: 0 [Enter]
Division by zero is not possible.
Please run the program again and enter
a number other than zero.
```

The value of `num2` is tested in line 22 before the division is performed. If the user enters 0, the lines controlled by the `if` part execute, displaying a message that indicates that the program cannot perform a division by zero. Otherwise, the `else` part takes control, which divides `num1` by `num2` and displays the result.

Checkpoint

4.12 TRUE or FALSE: The following `if/else` statements cause the same output to display.

A)
```
if (x > y)
    cout << "x is the greater.\n";
else
    cout << "x is not the greater.\n";
```

B)
```
if (y <= x)
    cout << "x is not the greater.\n";
else
    cout << "x is the greater.\n";
```

4.13 Write an `if/else` statement that assigns 1 to `x` if `y` is equal to 100. Otherwise it should assign 0 to `x`.

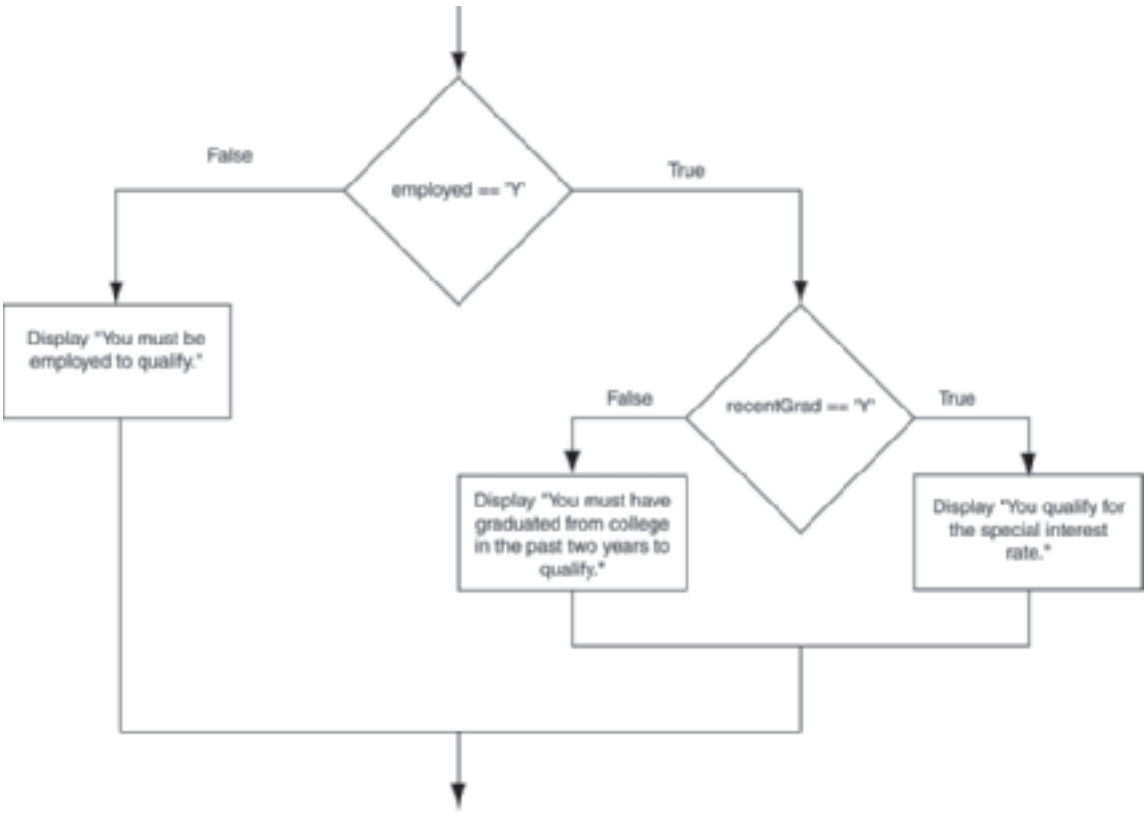
4.14 Write an `if/else` statement that assigns 0.10 to `commissionRate` unless `sales` is greater than or equal to 50000.00, in which case it assigns 0.20 to `commissionRate`.

4.5 Nested `if` Statements

CONCEPT: **To test more than one condition, an `if` statement can be nested inside another `if` statement.**

Sometimes an `if` statement must be nested inside another `if` statement. For example, consider a banking program that determines whether a bank customer qualifies for a special, low interest rate on a loan. To qualify, two conditions must exist: (1) the customer must be currently employed, and (2) the customer must have recently graduated from college (in the past two years). Figure 4-7 shows a flowchart for an algorithm that could be used in such a program.

Figure 4-7



If we follow the flow of execution in the flowchart, we see that the expression `employed == 'Y'` is tested. If this expression is false, there is no need to perform further tests; we know that the customer does not qualify for the special interest rate. If the expression is true, however, we need to test the second condition. This is done with a nested decision structure that tests the expression `recentGrad == 'Y'`. If this expression is true, then the customer qualifies for the special interest rate. If this expression is false, then the customer does not qualify. Program 4-10 shows the code for the complete program.

Program 4-10

```
1  // This program demonstrates the nested if statement.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      char employed,   // Currently employed, Y or N
8           recentGrad; // Recent graduate, Y or N
9
10     // Is the user employed and a recent graduate?
11     cout << "Answer the following questions\n";
12     cout << "with either Y for Yes or ";
13     cout << "N for No.\n";
14     cout << "Are you employed? ";
15     cin >> employed;
16     cout << "Have you graduated from college ";
17     cout << "in the past two years? ";
18     cin >> recentGrad;
19
20     // Determine the user's loan qualifications.
21     if (employed == 'Y')
22     {
23         if (recentGrad == 'Y') //Nested if
24         {
25             cout << "You qualify for the special ";
26             cout << "interest rate.\n";
27         }
28     }
29     return 0;
30 }
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? Y [Enter]
Have you graduated from college in the past two years? Y [Enter]
You qualify for the special interest rate.
```

Program Output with Different Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? Y [Enter]
Have you graduated from college in the past two years? N [Enter]
```

Look at the `if` statement that begins in line 21. It tests the expression `employed == 'Y'`. If this expression is true, the `if` statement that begins in line 23 is executed. Otherwise the program jumps to the `return` statement in line 29 and the program ends.

Notice in the second sample execution of Program 4-10 that the program output does not inform the user whether he or she qualifies for the special interest rate. If the user enters an 'N' (or any character other than 'Y') for `employed` or `recentGrad`, the program does not

print a message letting the user know that he or she does not qualify. An `else` statement should be able to remedy this, as illustrated by Program 4-11.

Program 4-11

```
// This program demonstrates the nested if statement.
#include <iostream>
using namespace std;

int main()
{
    char employed,   // Currently employed, Y or N
         recentGrad; // Recent graduate, Y or N

    // Is the user employed and a recent graduate?
    cout << "Answer the following questions\n";
    cout << "with either Y for Yes or ";
    cout << "N for No.\n";
    cout << "Are you employed? ";
    cin >> employed;
    cout << "Have you graduated from college ";
    cout << "in the past two years? ";
    cin >> recentGrad;

    // Determine the user's loan qualifications.
    if (employed == 'Y')
    {
        if (recentGrad == 'Y') // Nested if
        {
            cout << "You qualify for the special ";
            cout << "interest rate.\n";
        }
        else // Not a recent grad, but employed
        {
            cout << "You must have graduated from ";
            cout << "college in the past two\n";
            cout << "years to qualify.\n";
        }
    }
    else // Not employed
    {
        cout << "You must be employed to qualify.\n";
    }
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? N [Enter]
Have you graduated from college in the past two years? Y [Enter]
You must be employed to qualify.
```

(program output continues)

Program 4-11 *(continued)*

Program Output with Different Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? Y [Enter]
Have you graduated from college in the past two years? N [Enter]
You must have graduated from college in the past two years to qualify.
```

Program Output with Different Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? Y [Enter]
Have you graduated from college in the past two years? Y [Enter]
You qualify for the special interest rate.
```

In this version of the program, both `if` statements have `else` clauses that inform the user why he or she does not qualify for the special interest rate.

Programming Style and Nested Decision Structures

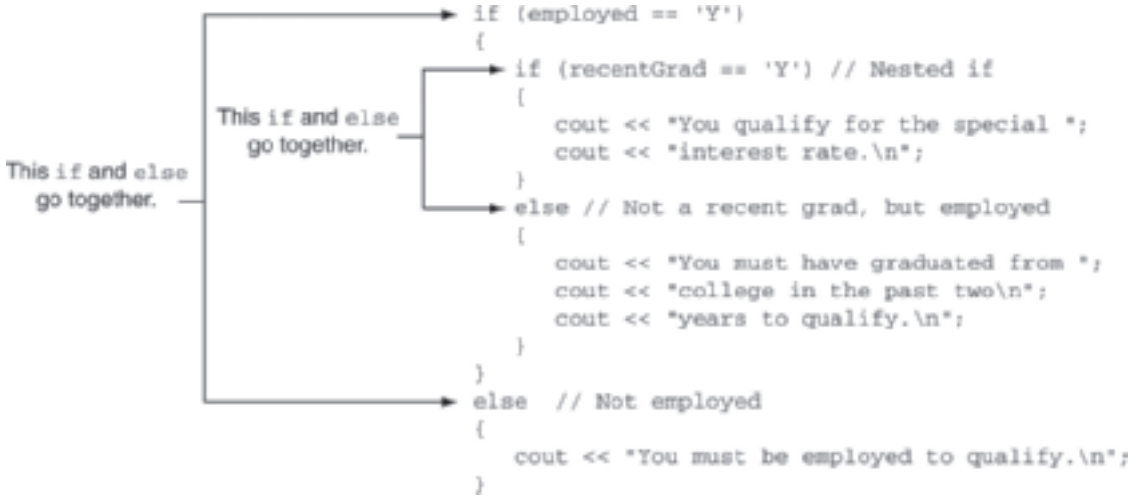
For readability and easier debugging, it's important to use proper alignment and indentation in a set of nested `if` statements. This makes it easier to see which actions are performed by each part of the decision structure. For example, the following code is functionally equivalent to lines 21 through 38 in Program 4-11. Although this code is logically correct, it is very difficult to read and would be very difficult to debug because it is not properly indented.

Don't write code like this!

```
if (employed == 'Y')
{
if (recentGrad == 'Y') // Nested if
{
cout << "You qualify for the special ";
cout << "interest rate.\n";
}
else // Not a recent grad, but employed
{
cout << "You must have graduated from ";
cout << "college in the past two\n";
cout << "years to qualify.\n";
}
}
else // Not employed
{
cout << "You must be employed to qualify.\n";
}
```

Proper indentation and alignment also makes it easier to see which `if` and `else` clauses belong together, as shown in Figure 4-8.

Figure 4-8



Testing a Series of Conditions

In the previous example you saw how a program can use nested decision structures to test more than one condition. It is not uncommon for a program to have a series of conditions to test and then perform an action depending on which condition is true. One way to accomplish this is to have a decision structure with numerous other decision structures nested inside it. For example, consider the program presented in the following *In the Spotlight* section.

In the Spotlight:

Multiple Nested Decision Structures

Dr. Suarez teaches a literature class and uses the following 10-point grading scale for all of his exams:

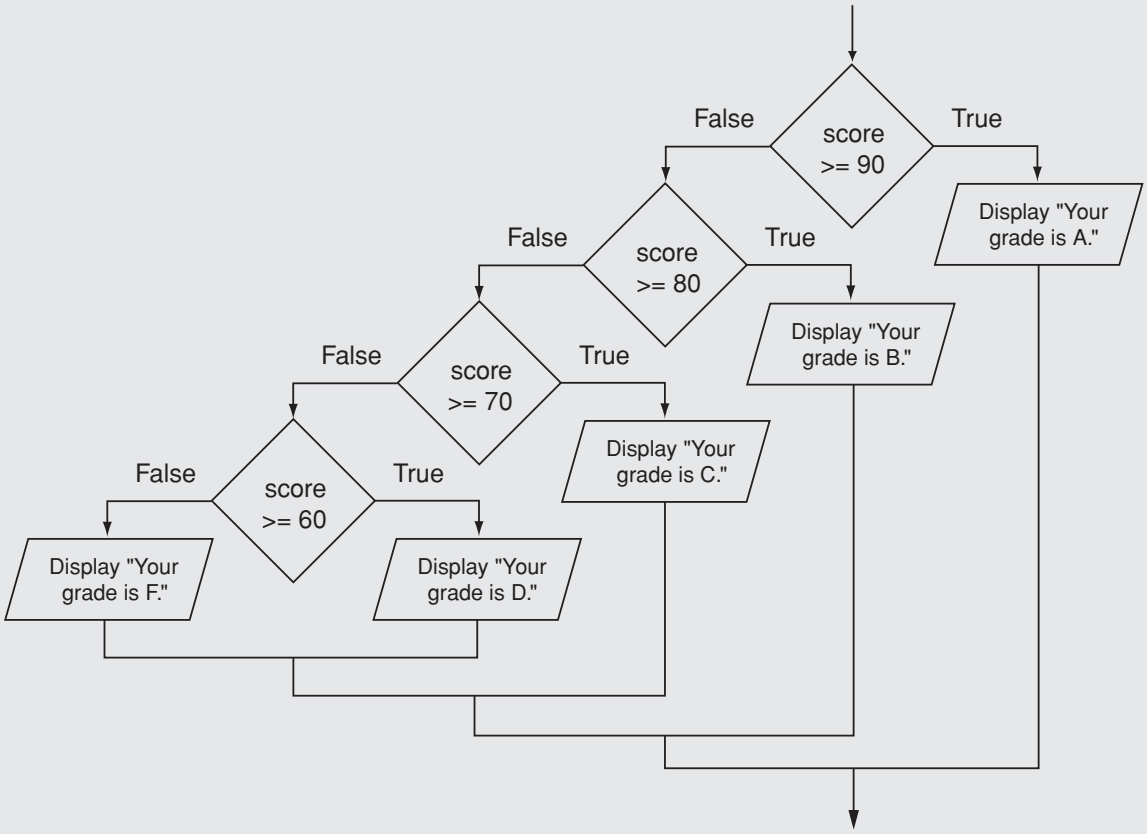
Test Score	Grade
90 and above	A
80–89	B
70–79	C
60–69	D
Below 60	F

He has asked you to write a program that will allow a student to enter a test score and then display the grade for that score. Here is the algorithm that you will use:

Ask the user to enter a test score.
Determine the grade in the following manner:
If the score is greater than or equal to 90, then the grade is A.
Otherwise, if the score is greater than or equal to 80, then the grade is B.
Otherwise, if the score is greater than or equal to 70, then the grade is C.
Otherwise, if the score is greater than or equal to 60, then the grade is D.
Otherwise, the grade is F.

You decide that the process of determining the grade will require several nested decisions structures, as shown in Figure 4-9. Program 4-12 shows the code for the complete program. The code for the nested decision structures is in lines 17 through 45.

Figure 4-9 Nested decision structure to determine a grade



Program 4-12

```
1  // This program uses nested if/else statements to assign a
2  // letter grade (A, B, C, D, or F) to a numeric test score.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       // Constants for grade thresholds
9       const int A_SCORE = 90,
10                B_SCORE = 80,
11                C_SCORE = 70,
12                D_SCORE = 60;
13
14      int testScore; // To hold a numeric test score
15
16      // Get the numeric test score.
17      cout << "Enter your numeric test score and I will\n";
```

```
18      cout << "tell you the letter grade you earned: ";
19      cin >> testScore;
20
21      // Determine the letter grade.
22      if (testScore >= A_SCORE)
23      {
24         cout << "Your grade is A.\n";
25      }
26      else
27      {
28          if (testScore >= B_SCORE)
29          {
30              cout << "Your grade is B.\n";
31          }
32          else
33          {
34              if (testScore >= C_SCORE)
35              {
36                  cout << "Your grade is C.\n";
37              }
38              else
39              {
40                  if (testScore >= D_SCORE)
41                  {
42                      cout << "Your grade is D.\n";
43                  }
44                  else
45                  {
46                      cout << "Your grade is F.\n";
47                  }
48              }
49          }
50      }
51
52      return 0;
53  }
```

Program Output with Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: 78 [Enter]
Your grade is C.
```

Program Output with Different Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: 84 [Enter]
Your grade is B.
```

Checkpoint

4.15 If you executed the following code, what would it display if the user enters 5? What if the user enters 15? What if the user enters 30? What if the user enters –1?

```
int number;
cout << "Enter a number: ";
cin >> number;

if (number > 0)
{
   cout << "Zero\n";

   if (number > 10)
   {
      cout << "Ten\n";

      if (number > 20)
      {
         cout << "Twenty\n";
      }
   }
}
```

4.6 The `if/else if` Statement

CONCEPT: The **`if/else if`** statement tests a series of conditions. It is often simpler to test a series of conditions with the **`if/else if`** statement than with a set of nested **`if/else`** statements.

VideoNote
The `if/else if` Statement

Even though Program 4-12 is a simple example, the logic of the nested decision structure is fairly complex. In C++, and many other languages, you can alternatively test a series of conditions using the `if/else if` statement. The `if/else if` statement makes certain types of nested decision logic simpler to write. Here is the general format of the `if/else if` statement:

```
if (expression_1)
{
   statement                    } If expression_1 is true these state-
   statement                    } ments are executed, and the rest of the
   etc.                         } structure is ignored.
}
else if (expression_2)
{
   statement                    } Otherwise, if expression_2 is true these
   statement                    } statements are executed, and the rest of the
   etc.                         } structure is ignored.
}
Insert as many else if clauses as necessary
else
{
   statement                    } These statements are executed
   statement                    } if none of the expressions above
   etc.                         } are true.
}
```

When the statement executes, *expression_1* is tested. If *expression_1* is true, the block of statements that immediately follows is executed, and the rest of the structure is ignored. If *expression_1* is false, however, the program jumps to the very next `else if` clause and tests *expression_2*. If it is true, the block of statements that immediately follows is executed, and then the rest of the structure is ignored. This process continues, from the top of the structure to the bottom, until one of the expressions is found to be true. If none of the expressions are true, the last `else` clause takes over, and the block of statements immediately following it is executed.

The last `else` clause, which does not have an `if` statement following it, is referred to as the *trailing* `else`. The trailing `else` is optional, but in most cases you will use it.

NOTE: The general format shows braces surrounding each block of conditionally executed statements. As with other forms of the `if` statement, the braces are required only when more than one statement is conditionally executed.

Program 4-13 shows an example of the `if/else` if statement. This program is a modification of Program 4-12, which appears in the previous *In the Spotlight* section.

Program 4-13

```
1  // This program uses an if/else if statement to assign a
2  // letter grade (A, B, C, D, or F) to a numeric test score.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      // Constants for grade thresholds
9      const int A_SCORE = 90,
10               B_SCORE = 80,
11               C_SCORE = 70,
12               D_SCORE = 60;
13
14     int testScore; // To hold a numeric test score
15
16     // Get the numeric test score.
17     cout << "Enter your numeric test score and I will\n"
18          << "tell you the letter grade you earned: ";
19     cin >> testScore;
20
21     // Determine the letter grade.
22     if (testScore >= A_SCORE)
23        cout << "Your grade is A.\n";
24     else if (testScore >= B_SCORE)
25        cout << "Your grade is B.\n";
26     else if (testScore >= C_SCORE)
27        cout << "Your grade is C.\n";
28     else if (testScore >= D_SCORE)
29        cout << "Your grade is D.\n";
30     else
31        cout << "Your grade is F.\n";
```

(program continues)

Program 4-13 *(continued)*

```
32
33       return 0;
34   }
```

Program Output with Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: 78 [Enter]
Your grade is C.
```

Program Output with Different Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: 84 [Enter]
Your grade is B.
```

Let's analyze how the `if/else if` statement in lines 22 through 31 works. First, the expression `testScore >= A_SCORE` is tested in line 22:

```
→ if (testScore >= A_SCORE)
      cout << "Your grade is A.\n";
  else if (testScore >= B_SCORE)
      cout << "Your grade is B.\n";
  else if (testScore >= C_SCORE)
      cout << "Your grade is C.\n";
  else if (testScore >= D_SCORE)
      cout << "Your grade is D.\n";
  else
      cout << "Your grade is F.\n";
```

If `testScore` is greater than or equal to 90, the message `"Your grade is A.\n"` is displayed and the rest of the `if/else if` statement is skipped. If `testScore` is not greater than or equal to 90, the `else` clause in line 24 takes over and causes the next `if` statement to be executed:

```
  if (testScore >= A_SCORE)
      cout << "Your grade is A.\n";

→ else if (testScore >= B_SCORE)
      cout << "Your grade is B.\n";
  else if (testScore >= C_SCORE)
      cout << "Your grade is C.\n";
  else if (testScore >= D_SCORE)
      cout << "Your grade is D.\n";
  else
      cout << "Your grade is F.\n";
```

The first `if` statement handles all of the grades greater than or equal to 90, so when this `if` statement executes, `testScore` will have a value of 89 or less. If `testScore` is greater than or equal to 80, the message `"Your grade is B.\n"` is displayed and the rest of the `if/else if` statement is skipped. This chain of events continues until one of the expressions is found to be true, or the last `else` clause at the end of the statement is encountered.

Notice the alignment and indentation that is used with the `if/else if` statement: The starting `if` clause, the `else if` clauses, and the trailing `else` clause are all aligned, and the conditionally executed statements are indented.

Using the Trailing else To Catch Errors

The trailing else clause, which appears at the end of the if/else if statement, is optional, but in many situations you will use it to catch errors. For example, Program 4-13 will assign a grade to any number that is entered as the test score, including negative numbers. If a negative test score is entered, however, the user has probably made a mistake. We can modify the code as shown in Program 4-14 so the trailing else clause catches any test score that is less then 0 and displays an error message.

Program 4-14

```
1  // This program uses an if/else if statement to assign a
2  // letter grade (A, B, C, D, or F) to a numeric test score.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       // Constants for grade thresholds
9       const int A_SCORE = 90,
10                B_SCORE = 80,
11                C_SCORE = 70,
12                D_SCORE = 60;
13
14      int testScore; // To hold a numeric test score
15
16      // Get the numeric test score.
17      cout << "Enter your numeric test score and I will\n"
18           << "tell you the letter grade you earned: ";
19      cin >> testScore;
20
21      // Determine the letter grade.
22      if (testScore >= A_SCORE)
23         cout << "Your grade is A.\n";
24      else if (testScore >= B_SCORE)
25         cout << "Your grade is B.\n";
26      else if (testScore >= C_SCORE)
27         cout << "Your grade is C.\n";
28      else if (testScore >= D_SCORE)
29         cout << "Your grade is D.\n";
30      else if (testScore >= 0)
31         cout << "Your grade is F.\n";
32      else
33         cout << "Invalid test score.\n";
34
35      return 0;
36 }
```

Program Output with Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: -1 [Enter]
Invalid test score.
```

The `if/else if` Statement Compared to a Nested Decision Structure

You never have to use the `if/else if` statement because its logic can be coded with nested `if/else` statements. However, a long series of nested `if/else` statements has two particular disadvantages when you are debugging code:

- The code can grow complex and become difficult to understand.
- Because indenting is important in nested statements, a long series of nested `if/else` statements can become too long to be displayed on the computer screen without horizontal scrolling. Also, long statements tend to "wrap around" when printed on paper, making the code even more difficult to read.

The logic of an `if/else if` statement is usually easier to follow than that of a long series of nested `if/else` statements. And, because all of the clauses are aligned in an `if/else if` statement, the lengths of the lines in the statement tend to be shorter.

Checkpoint

4.16 What will the following code display?

```
int funny = 7, serious = 15;
funny = serious % 2;

if (funny != 1)
{
   funny = 0;
   serious = 0;
}
else if (funny == 2)
{
   funny = 10;
   serious = 10;
}
else
{
   funny = 1;
   serious = 1;
}
cout << funny << "" << serious << endl;
```

4.17 The following code is used in a bookstore program to determine how many discount coupons a customer gets. Complete the table that appears after the program.

```
int numBooks, numCoupons;
cout << "How many books are being purchased? ";
cin >> numBooks;

if (numBooks < 1)
   numCoupons = 0;
else if (numBooks < 3)
   numCoupons = 1;
else if (numBooks < 5)
   numCoupons = 2;
```

```
else
   numCoupons = 3;
cout << "The number of coupons to give is "
     << numCoupons << endl;
```

If the customer purchases this many books	This many coupons are given.
1	
3	
4	
5	
10	

4.7 Flags

CONCEPT: **A flag is a Boolean or integer variable that signals when a condition exists.**

A *flag* is typically a `bool` variable that signals when some condition exists in the program. When the flag variable is set to `false`, it indicates that the condition does not exist. When the flag variable is set to `true`, it means the condition does exist.

For example, suppose a program that calculates sales commissions has a `bool` variable, defined and initialized as shown here:

```
bool salesQuotaMet = false;
```

In the program, the `salesQuotaMet` variable is used as a flag to indicate whether a salesperson has met the sales quota. When we define the variable, we initialize it with `false` because we do not yet know if the salesperson has met the sales quota. Assuming a variable named `sales` holds the amount of sales, code similar to the following might be used to set the value of the `salesQuotaMet` variable:

```
if (sales >= QUOTA_AMOUNT)
   salesQuotaMet = true;
else
   salesQuotaMet = false;
```

As a result of this code, the `salesQuotaMet` variable can be used as a flag to indicate whether the sales quota has been met. Later in the program we might test the flag in the following way:

```
if (salesQuotaMet)
   cout << "You have met your sales quota!\n";
```

This code displays *You have met your sales quota!* if the `bool` variable `salesQuotaMet` is `true`. Notice that we did not have to use the `==` operator to explicitly compare the `salesQuotaMet` variable with the value `true`. This code is equivalent to the following:

```
if (salesQuotaMet == true)
   cout << "You have met your sales quota!\n";
```

Integer Flags

Integer variables may also be used as flags. This is because in C++ the value 0 is considered false, and any nonzero value is considered true. In the sales commission program previously described, we could define the `salesQuotaMet` variable with the following statement:

```
int salesQuotaMet = 0;       // 0 means false.
```

As before, we initialize the variable with 0 because we do not yet know if the sales quota has been met. After the sales have been calculated, we can use code similar to the following to set the value of the `salesQuotaMet` variable:

```
if (sales >= QUOTA_AMOUNT)
   salesQuotaMet = 1;
else
   salesQuotaMet = 0;
```

Later in the program we might test the flag in the following way:

```
if (salesQuotaMet)
   cout << "You have met your sales quota!\n";
```

4.8 Logical Operators

CONCEPT: Logical operators connect two or more relational expressions into one or reverse the logic of an expression.

In the previous section you saw how a program tests two conditions with two `if` statements. In this section you will see how to use logical operators to combine two or more relational expressions into one. Table 4-6 lists C++'s logical operators.

Table 4-6

Operator	Meaning	Effect
`&&`	AND	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
`||`	OR	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
`!`	NOT	The ! operator reverses the "truth" of an expression. It makes a true expression false, and a false expression true.

The && Operator

The `&&` operator is known as the logical AND operator. It takes two expressions as operands and creates an expression that is true only when both sub-expressions are true. Here is an example of an `if` statement that uses the `&&` operator:

```
if (temperature < 20 && minutes > 12)
   cout << "The temperature is in the danger zone.";
```

In the statement above the two relational expressions are combined into a single expression. The `cout` statement will only be executed if `temperature` is less than 20 AND `minutes` is greater than 12. If either relational test is false, the entire expression is false, and the `cout` statement is not executed.

TIP: You must provide complete expressions on both sides of the `&&` operator. For example, the following is not correct because the condition on the right side of the `&&` operator is not a complete expression.

```
temperature > 0 && < 100
```

The expression must be rewritten as

```
temperature > 0 && temperature < 100
```

Table 4-7 shows a truth table for the `&&` operator. The truth table lists all the possible combinations of values that two expressions may have, and the resulting value returned by the `&&` operator connecting the two expressions.

Table 4-7

Expression	Value of Expression
`true && false`	`false (0)`
`false && true`	`false (0)`
`false && false`	`false (0)`
`true && true`	`true (1)`

As the table shows, both sub-expressions must be true for the `&&` operator to return a true value.

NOTE: If the sub-expression on the left side of an `&&` operator is false, the expression on the right side will not be checked. Since the entire expression is false if only one of the sub-expressions is false, it would waste CPU time to check the remaining expression. This is called *short circuit evaluation*.

The `&&` operator can be used to simplify programs that otherwise would use nested `if` statements. Program 4-15 performs a similar operation as Program 4-11, which qualifies a bank customer for a special interest rate. This program uses a logical operator.

Program 4-15

```
1  // This program demonstrates the && logical operator.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      char employed,   // Currently employed, Y or N
8           recentGrad; // Recent graduate, Y or N
9
10     // Is the user employed and a recent graduate?
11     cout << "Answer the following questions\n";
12     cout << "with either Y for Yes or N for No.\n";
13
```

(program continues)

Program 4-15 *(continued)*

```
14       cout << "Are you employed? ";
15       cin >> employed;
16
17       cout << "Have you graduated from college "
18            << "in the past two years? ";
19       cin >> recentGrad;
20
21       // Determine the user's loan qualifications.
22       if (employed == 'Y' && recentGrad == 'Y')
23       {
24          cout << "You qualify for the special "
25               << "interest rate.\n";
26       }
27       else
28       {
29          cout << "You must be employed and have\n"
30               << "graduated from college in the\n"
31               << "past two years to qualify.\n";
32       }
33       return 0;
34   }
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? Y [Enter]
Have you graduated from college in the past two years? N [Enter]
You must be employed and have
graduated from college in the
past two years to qualify.
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? N [Enter]
Have you graduated from college in the past two years? Y [Enter]
You must be employed and have
graduated from college in the
past two years to qualify.
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? Y [Enter]
Have you graduated from college in the past two years? Y [Enter]
You qualify for the special interest rate.
```

The message "`You qualify for the special interest rate`" is displayed only when both the expressions `employed == 'Y'` and `recentGrad == 'Y'` are true. If either of these is false, the message "`You must be employed and have graduated from college in the past two years to qualify.`" is printed.

NOTE: Although it is similar, Program 4-15 is not the logical equivalent of Program 4-11. For example, Program 4-15 doesn't display the message "You must be employed to qualify."

The || Operator

The || operator is known as the logical OR operator. It takes two expressions as operands and creates an expression that is true when either of the sub-expressions are true. Here is an example of an `if` statement that uses the || operator:

```
if (temperature < 20 || temperature > 100)
   cout << "The temperature is in the danger zone.";
```

The `cout` statement will be executed if `temperature` is less than 20 OR `temperature` is greater than 100. If either relational test is true, the entire expression is true and the `cout` statement is executed.

TIP: You must provide complete expressions on both sides of the || operator. For example, the following is not correct because the condition on the right side of the || operator is not a complete expression.

```
temperature < 0 || > 100
```

The expression must be rewritten as

```
temperature < 0 || temperature > 100
```

Table 4-8 shows a truth table for the || operator.

Table 4-8

Expression	Value of the Expression
`true || false`	`true (1)`
`false || true`	`true (1)`
`false || false`	`false (0)`
`true || true`	`true (1)`

All it takes for an OR expression to be true is for one of the sub-expressions to be true. It doesn't matter if the other sub-expression is false or true.

NOTE: The || operator also performs short circuit evaluation. If the sub-expression on the left side of an || operator is true, the expression on the right side will not be checked. Since it's only necessary for one of the sub-expressions to be true, it would waste CPU time to check the remaining expression.

Program 4-16 performs different tests to qualify a person for a loan. This one determines if the customer earns at least $35,000 per year, or has been employed for more than five years.

Program 4-16

```
1  // This program demonstrates the logical || operator.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       // Constants for minimum income and years
8       const double MIN_INCOME = 35000.0;
9       const int MIN_YEARS = 5;
10
11      double income; // Annual income
12      int years;     // Years at the current job
13
14      // Get the annual income
15      cout << "What is your annual income? ";
16      cin >> income;
17
18      // Get the number of years at the current job.
19      cout << "How many years have you worked at "
20           << "your current job? ";
21      cin >> years;
22
23      // Determine the user's loan qualifications.
24      if (income >= MIN_INCOME || years > MIN_YEARS)
25          cout << "You qualify.\n";
26      else
27      {
28          cout << "You must earn at least $"
29               << MIN_INCOME << " or have been "
30               << "employed more than " << MIN_YEARS
31               << " years.\n";
32      }
33      return 0;
34 }
```

Program Output with Example Input Shown in Bold

```
What is your annual income? 40000 [Enter]
How many years have you worked at your current job? 2 [Enter]
You qualify.
```

Program Output with Example Input Shown in Bold

```
What is your annual income? 20000 [Enter]
How many years have you worked at your current job? 7 [Enter]
You qualify.
```

Program Output with Example Input Shown in Bold

```
What is your annual income? 30000 [Enter]
How many years have you worked at your current job? 3 [Enter]
You must earn at least $35000 or have been employed more than 5 years.
```

The message "`You qualify\n.`" is displayed when either or both the expressions `income >= 35000` or `years > 5` are true. If both of these are false, the disqualifying message is printed.

The ! Operator

The `!` operator performs a logical NOT operation. It takes an operand and reverses its truth or falsehood. In other words, if the expression is true, the `!` operator returns false, and if the expression is false, it returns true. Here is an `if` statement using the `!` operator:

```
if (!(temperature > 100))
   cout << "You are below the maximum temperature.\n";
```

First, the expression `(temperature > 100)` is tested to be true or false. Then the `!` operator is applied to that value. If the expression `(temperature > 100)` is true, the `!` operator returns false. If it is false, the `!` operator returns true. In the example, it is equivalent to asking "is the temperature not greater than 100?"

Table 4-9 shows a truth table for the `!` operator.

Table 4-9

Expression	Value of the Expression
`!true`	`false (0)`
`!false`	`true (1)`

Program 4-17 performs the same task as Program 4-16. The `if` statement, however, uses the `!` operator to determine if the user does *not* make at least $35,000 or has *not* been on the job more than five years.

Program 4-17

```
1  // This program demonstrates the logical ! operator.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       // Constants for minimum income and years
8       const double MIN_INCOME = 35000.0;
9       const int MIN_YEARS = 5;
10
11    double income; // Annual income
12    int years;     // Years at the current job
13
14    // Get the annual income
15    cout << "What is your annual income? ";
16    cin >> income;
17
18    // Get the number of years at the current job.
19    cout << "How many years have you worked at "
20         << "your current job? ";
```

(program continues)

Program 4-17 *(continued)*

```
21       cin >> years;
22
23       // Determine the user's loan qualifications.
24       if (!(income >= MIN_INCOME || years > MIN_YEARS))
25       {
26          cout << "You must earn at least $"
27               << MIN_INCOME << " or have been "
28               << "employed more than " << MIN_YEARS
29               << " years.\n";
30       }
31       else
32          cout << "You qualify.\n";
33       return 0;
34    }
```

The output of Program 4-17 is the same as Program 4-16.

Precedence and Associativity of Logical Operators

Table 4-10 shows the precedence of C++'s logical operators, from highest to lowest.

Table 4-10

Logical Operators in Order of Precedence
!
&&
||

The `!` operator has a higher precedence than many of the C++ operators. To avoid an error, you should always enclose its operand in parentheses unless you intend to apply it to a variable or a simple expression with no other operators. For example, consider the following expressions:

```
!(x > 2)
!x > 2
```

The first expression applies the `!` operator to the expression `x > 2`. It is asking, "Is `x` not greater than 2?" The second expression, however, applies the `!` operator to `x` only. It is asking, "Is the logical negation of `x` greater than 2?" Suppose `x` is set to 5. Since 5 is nonzero, it would be considered true, so the `!` operator would reverse it to false, which is 0. The `>` operator would then determine if 0 is greater than 2. To avoid a catastrophe like this, always use parentheses!

The `&&` and `||` operators rank lower in precedence than the relational operators, so precedence problems are less likely to occur. If you feel unsure, however, it doesn't hurt to use parentheses anyway.

```
(a > b) && (x < y)     is the same as   a > b && x < y
(x == y) || (b > a)    is the same as   x == y || b > a
```

The logical operators have left-to-right associativity. In the following expression, `a < b` is evaluated before `y == z`.

```
a < b || y == z
```

In the following expression, `y == z` is evaluated first, however, because the `&&` operator has higher precedence than `||`.

```
a < b || y == z && m > j
```

The expression is equivalent to

```
(a < b) || ((y == z) && (m > j))
```

4.9 Checking Numeric Ranges with Logical Operators

CONCEPT: Logical operators are effective for determining whether a number is in or out of a range.

When determining whether a number is inside a numeric range, it's best to use the `&&` operator. For example, the following `if` statement checks the value in `x` to determine whether it is in the range of 20 through 40:

```
if (x >= 20 && x <= 40)
   cout << x << " is in the acceptable range.\n";
```

The expression in the `if` statement will be true only when `x` is both greater than or equal to 20 AND less than or equal to 40. `x` must be within the range of 20 through 40 for this expression to be true.

When determining whether a number is outside a range, the `||` operator is best to use. The following statement determines whether `x` is outside the range of 20 to 40:

```
if (x < 20 || x > 40)
   cout << x << " is outside the acceptable range.\n";
```

It's important not to get the logic of these logical operators confused. For example, the following `if` statement would never test true:

```
if (x < 20 && x > 40)
   cout << x << " is outside the acceptable range.\n";
```

Obviously, `x` cannot be less than 20 and at the same time greater than 40.

NOTE: C++ does not allow you to check numeric ranges with expressions such as `5 < x < 20`. Instead, you must use a logical operator to connect two relational expressions, as previously discussed.

Checkpoint

4.18 The following truth table shows various combinations of the values `true` and `false` connected by a logical operator. Complete the table by indicating if the result of such a combination is TRUE or FALSE.

Logical Expression	Result (true or false)
`true && false`	
`true && true`	
`false && true`	
`false && false`	
`true || false`	
`true || true`	
`false || true`	
`false || false`	
`!true`	
`!false`	

4.19 Assume the variables `a = 2`, `b = 4`, and `c = 6`. Indicate by circling the T or F if each of the following conditions is true or false:

```
a == 4 || b > 2          T  F
6 <= c && a > 3          T  F
1 != b && c != 3         T  F
a >= -1 || a <= b        T  F
!(a > 2)                 T  F
```

4.20 Write an `if` statement that prints the message "The number is valid" if the variable `speed` is within the range 0 through 200.

4.21 Write an `if` statement that prints the message "The number is not valid" if the variable `speed` is outside the range 0 through 200.

4.10 Menus

CONCEPT: **You can use nested `if/else` statements or the `if/else if` statement to create menu-driven programs. A *menu-driven* program allows the user to determine the course of action by selecting it from a list of actions.**

A menu is a screen displaying a set of choices the user selects from. For example, a program that manages a mailing list might give you the following menu:

1. Add a name to the list.
2. Remove a name from the list.
3. Change a name in the list.
4. Print the list.
5. Quit the program.

The user selects one of the operations by entering its number. Entering 4, for example, causes the mailing list to be printed, and entering 5 causes the program to end. Nested `if/else` statements or an `if/else if` structure can be used to set up such a menu. After the user enters a number, the program compares the number with the available selections and executes the statements that perform that operation.

Program 4-18 calculates the charges for membership in a health club. The club has three membership packages to choose from: standard adult membership, child membership, and

senior citizen membership. The program presents a menu that allows the user to choose the desired package and then calculates the cost of the membership.

Program 4-18

```
// This program displays a menu and asks the user to make a
// selection. An if/else if statement determines which item
// the user has chosen.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int choice;     // To hold a menu choice
    int months;     // To hold the number of months
    double charges; // To hold the monthly charges

    // Constants for membership rates
    const double ADULT = 40.0,
                 SENIOR = 30.0,
                 CHILD = 20.0;

    // Constants for menu choices
    const int ADULT_CHOICE = 1,
              CHILD_CHOICE = 2,
              SENIOR_CHOICE = 3,
              QUIT_CHOICE = 4;

    // Display the menu and get a choice.
    cout << "\t\tHealth Club Membership Menu\n\n"
         << "1. Standard Adult Membership\n"
         << "2. Child Membership\n"
         << "3. Senior Citizen Membership\n"
         << "4. Quit the Program\n\n"
         << "Enter your choice: ";
    cin >> choice;

    // Set the numeric output formatting.
    cout << fixed << showpoint << setprecision(2);

    // Respond to the user's menu selection.
    if (choice == ADULT_CHOICE)
    {
       cout << "For how many months? ";
       cin >> months;
       charges = months * ADULT;
       cout << "The total charges are $" << charges << endl;
    }
    else if (choice == CHILD_CHOICE)
    {
       cout << "For how many months? ";
```

(program continues)

Program 4-18 *(continued)*

```
48          cin >> months;
49          charges = months * CHILD;
50          cout << "The total charges are $" << charges << endl;
51       }
52       else if (choice == SENIOR_CHOICE)
53       {
54           cout << "For how many months? ";
55           cin >> months;
56           charges = months * SENIOR;
57           cout << "The total charges are $" << charges << endl;
58       }
59       else if (choice == QUIT_CHOICE)
60       {
61           cout << "Program ending.\n";
62       }
63       else
64       {
65           cout << "The valid choices are 1 through 4. Run the\n"
66                << "program again and select one of those.\n";
67       }
68       return 0;
69   }
```

Program Output with Example Input Shown in Bold

```
      Health Club Membership Menu

1. Standard Adult Membership
2. Child Membership
3. Senior Citizen Membership
4. Quit the Program

Enter your choice: 3 [Enter]
For how many months? 6 [Enter]
The total charges are $180.00
```

Let's take a closer look at the program:

- Lines 10–12 define the following variables:
  - The `choice` variable will hold the user's menu choice.
  - The `months` variable will hold the number of months of health club membership.
  - The `charges` variable will hold the total charges.
- Lines 15–17 define named constants for the monthly membership rates for adult, senior citizen, and child memberships.
- Lines 20–23 define named constants for the menu choices.
- Lines 26–32 display the menu and get the user's choice.
- Line 35 sets the numeric output formatting for floating point numbers.
- Lines 38–67 is an `if/else if` statement that determines the user's menu choice in the following manner:
  - If the user selected 1 from the menu (adult membership), the statements in lines 40–43 are executed.

- Otherwise, if the user selected 2 from the menu (child membership), the statements in lines 47–50 are executed.
- Otherwise, if the user selected 3 from the menu (senior citizen membership), the statements in lines 54–57 are executed.
- Otherwise, if the user selected 4 from the menu (quit the program), the statement in line 61 is executed.
- If the user entered any choice other than 1, 2, 3, or 4, the `else` clause in lines 63–67 executes, displaying an error message.

4.11 Focus on Software Engineering: Validating User Input

CONCEPT: As long as the user of a program enters bad input, the program will produce bad output. Programs should be written to filter out bad input.

Perhaps the most famous saying of the computer world is "Garbage in, garbage out." The integrity of a program's output is only as good as its input, so you should try to make sure garbage does not go into your programs. *Input validation* is the process of inspecting data given to a program by the user and determining if it is valid. A good program should give clear instructions about the kind of input that is acceptable and not assume the user has followed those instructions. Here are just a few examples of input validations performed by programs:

- Numbers are checked to ensure they are within a range of possible values. For example, there are 168 hours in a week. It is not possible for a person to be at work longer than 168 hours in one week.
- Values are checked for their "reasonableness." Although it might be possible for a person to be at work for 168 hours per week, it is not probable.
- Items selected from a menu or other sets of choices are checked to ensure they are available options.
- Variables are checked for values that might cause problems, such as division by zero.

Program 4-19 is a test scoring program that rejects any test score less than 0 or greater than 100.

Program 4-19

```
1  // This test scoring program does not accept test scores
2  // that are less than 0 or greater than 100.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      // Constants for grade thresholds
9      const int A_SCORE = 90,
```

(program continues)

Program 4-19 *(continued)*

```
10                  B_SCORE = 80,
11                  C_SCORE = 70,
12                  D_SCORE = 60,
13                  MIN_SCORE = 0,      // Minimum valid score
14                  MAX_SCORE = 100;    // Maximum valid score
15
16      int testScore; // To hold a numeric test score
17
18      // Get the numeric test score.
19      cout << "Enter your numeric test score and I will\n"
20           << "tell you the letter grade you earned: ";
21      cin >> testScore;
22
23      // Validate the input and determine the grade.
24      if (testScore >= MIN_SCORE && testScore <= MAX_SCORE)
25      {
26           // Determine the letter grade.
27           if (testScore >= A_SCORE)
28             cout << "Your grade is A.\n";
29           else if (testScore >= B_SCORE)
30             cout << "Your grade is B.\n";
31           else if (testScore >= C_SCORE)
32             cout << "Your grade is C.\n";
33           else if (testScore >= D_SCORE)
34             cout << "Your grade is D.\n";
35           else
36             cout << "Your grade is F.\n";
37      }
38      else
39      {
40           // An invalid score was entered.
41           cout << "That is an invalid score. Run the program\n"
42                << "again and enter a value in the range of\n"
43                << MIN_SCORE << " through " << MAX_SCORE << ".\n";
44      }
45
46      return 0;
47  }
```

Program Output with Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: -1 [Enter]
That is an invalid score. Run the program
again and enter a value in the range of
0 through 100.
```

Program Output with Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: 81 [Enter]
Your grade is B.
```

4.12 Comparing Characters and Strings

CONCEPT: **Relational operators can also be used to compare characters and `string` objects.**

Earlier in this chapter you learned to use relational operators to compare numeric values. They can also be used to compare characters and `string` objects.

Comparing Characters

As you learned in Chapter 3, characters are actually stored in memory as integers. On most systems, this integer is the ASCII value of the character. For example, the letter 'A' is represented by the number 65, the letter 'B' is represented by the number 66, and so on. Table 4-11 shows the ASCII numbers that correspond to some of the commonly used characters.

Table 4-11 ASCII Values of Commonly Used Characters

Character	ASCII Value
'0' – '9'	48 – 57
'A' – 'Z'	65 – 90
'a' – 'z'	97 – 122
blank	32
period	46

Notice that every character, even the blank, has an ASCII code associated with it. Notice also that the ASCII code of a character representing a digit, such as `'1'` or `'2'`, is not the same as the value of the digit itself. A complete table showing the ASCII values for all characters can be found in Appendix B.

When two characters are compared, it is actually their ASCII values that are being compared. `'A' < 'B'` because the ASCII value of `'A'` (65) is less than the ASCII value of `'B'` (66). Likewise `'1' < '2'` because the ASCII value of `'1'` (49) is less than the ASCII value of `'2'` (50). However, as shown in Table 4-11, lowercase letters have higher ASCII codes than uppercase letters, so `'a' > 'Z'`. Program 4-20 shows how characters can be compared with relational operators.

Program 4-20

```
1  // This program demonstrates how characters can be
2  // compared with the relational operators.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      char ch;
9
```

(program continues)

Program 4-20 *(continued)*

```
10       // Get a character from the user.
11       cout << "Enter a digit or a letter: ";
12       ch = cin.get();
13
14       // Determine what the user entered.
15       if (ch >= '0' && ch <= '9')
16            cout << "You entered a digit.\n";
17       else if (ch >= 'A' && ch <= 'Z')
18            cout << "You entered an uppercase letter.\n";
19       else if (ch >= 'a' && ch <= 'z')
20            cout << "You entered a lowercase letter.\n";
21       else
22            cout << "That is not a digit or a letter.\n";
23
24       return 0;
25  }
```

Program Output with Example Input Shown in Bold

```
Enter a digit or a letter: t [Enter]
You entered a lowercase letter.
```

Program Output with Example Input Shown in Bold

```
Enter a digit or a letter: V [Enter]
You entered an uppercase letter.
```

Program Output with Example Input Shown in Bold

```
Enter a digit or a letter: 5 [Enter]
You entered a digit.
```

Program Output with Example Input Shown in Bold

```
Enter a digit or a letter: & [Enter]
That is not a digit or a letter.
```

Comparing `string` Objects

`string` objects can also be compared with relational operators. As with individual characters, when two `string` objects are compared, it is actually the ASCII value of the characters making up the strings that are being compared. For example, assume the following definitions exist in a program:

```
string str1 = "ABC";
string str2 = "XYZ";
```

The `string` object `str1` is considered less than the `string` object `str2` because the characters `"ABC"` alphabetically precede (have lower ASCII values than) the characters `"XYZ"`. So, the following `if` statement will cause the message `"str1 is less than str2."` to be displayed on the screen.

```
if (str1 < str2)
     cout << "str1 is less than str2.";
```

One by one, each character in the first operand is compared with the character in the corresponding position in the second operand. If all the characters in both `string` objects match, the two strings are equal. Other relationships can be determined if two characters in corresponding positions do not match. The first operand is less than the second operand if the first mismatched character in the first operand is less than its counterpart in the second operand. Likewise, the first operand is greater than the second operand if the first mismatched character in the first operand is greater than its counterpart in the second operand.

For example, assume a program has the following definitions:

```
string name1 = "Mary";
string name2 = "Mark";
```

The value in `name1`, `"Mary"`, is greater than the value in `name2`, `"Mark"`. This is because the first three characters in `name1` have the same ASCII values as the first three characters in `name2`, but the `'y'` in the fourth position of `"Mary"` has a greater ASCII value than the `'k'` in the corresponding position of `"Mark"`.

Any of the relational operators can be used to compare two `string` objects. Here are some of the valid comparisons of `name1` and `name2`.

```
name1 > name2 // true
name1 <= name2 // false
name1 != name2 // true
```

`string` objects can also, of course, be compared to string literals:

```
name1 < "Mary Jane" // true
```

Program 4-21 further demonstrates how relational operators can be used with `string` objects.

Program 4-21

```
1  // This program uses relational operators to compare a string
2  // entered by the user with valid stereo part numbers.
3  #include <iostream>
4  #include <iomanip>
5  #include <string>
6  using namespace std;
7
8  int main()
9  {
10     const double PRICE_A = 249.0,
11                  PRICE_B = 299.0;
12
13     string partNum; // Holds a stereo part number
14
15     // Display available parts and get the user's selection
16     cout << "The stereo part numbers are:\n"
17          << "Boom Box: part number S-29A \n"
18          << "Shelf Model: part number S-29B \n"
19          << "Enter the part number of the stereo you\n"
```

(program continues)

Program 4-21 *(continued)*

```
20            << "wish to purchase: ";
21       cin >> partNum;
22
23       // Set the numeric output formatting
24       cout << fixed << showpoint << setprecision(2);
25
26       // Determine and display the correct price
27       if (partNum == "S-29A")
28            cout << "The price is $" << PRICE_A << endl;
29       else if (partNum == "S-29B")
30            cout << "The price is $" << PRICE_B << endl;
31       else
32            cout << partNum << " is not a valid part number.\n";
33       return 0;
34  }
```

Program Output with Example Input Shown in Bold

```
The stereo part numbers are:
Boom Box: part number S-29A
Shelf Model: part number S-29B
Enter the part number of the stereo you
wish to purchase: S-29A [Enter]
The price is $249.00
```

Checkpoint

4.22 Indicate whether each of the following relational expressions is true or false. Refer to the ASCII table in Appendix B if necessary.

A) `'a' < 'z'`

B) `'a' == 'A'`

C) `'5' < '7'`

D) `'a' < 'A'`

E) `'1' == 1`

F) `'1' == 49`

4.23 Indicate whether each of the following relational expressions is true or false. Refer to the ASCII table in Appendix B if necessary.

A) `"Bill" == "BILL"`

B) `"Bill" < "BILL"`

C) `"Bill" < "Bob"`

D) `"189" > "23"`

E) `"189" > "Bill"`

F) `"Mary" < "MaryEllen"`

G) `"MaryEllen" < "Mary Ellen"`

4.13 The Conditional Operator

CONCEPT: **You can use the conditional operator to create short expressions that work like `if/else` statements.**

The conditional operator is powerful and unique. It provides a shorthand method of expressing a simple `if/else` statement. The operator consists of the question-mark (?) and the colon (:). Its format is:

```
expression ? expression : expression;
```

Here is an example of a statement using the conditional operator:

```
x < 0 ? y = 10 : z = 20;
```

The statement above is called a *conditional expression* and consists of three sub-expressions separated by the ? and : symbols. The expressions are `x < 0`, `y = 10`, and `z = 20`, as illustrated here:

```
x < 0   ?   y = 10   :   z = 20;
```

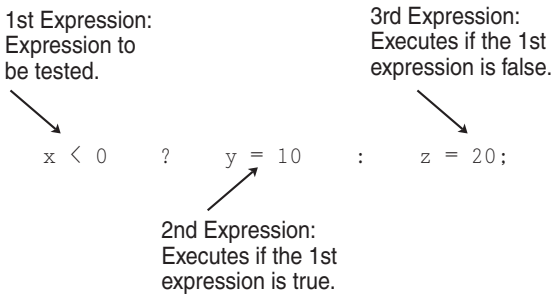
NOTE: Since it takes three operands, the conditional operator is considered a *ternary* operator.

The conditional expression above performs the same operation as the following `if/else` statement:

```
if (x < 0)
   y = 10;
else
   z = 20;
```

The part of the conditional expression that comes before the question mark is the expression to be tested. It's like the expression in the parentheses of an `if` statement. If the expression is true, the part of the statement between the ? and the : is executed. Otherwise, the part after the : is executed. Figure 4-10 illustrates the roles played by the three sub-expressions.

Figure 4-10



If it helps, you can put parentheses around the sub-expressions, as in the following:

```
(x < 0) ? (y = 10) : (z = 20);
```

Using the Value of a Conditional Expression

Remember, in C++ all expressions have a value, and this includes the conditional expression. If the first sub-expression is true, the value of the conditional expression is the value of the second sub-expression. Otherwise it is the value of the third sub-expression. Here is an example of an assignment statement using the value of a conditional expression:

```
a = x > 100 ? 0 : 1;
```

The value assigned to `a` will be either 0 or 1, depending upon whether `x` is greater than 100. This statement could be expressed as the following `if/else` statement:

```
if (x > 100)
   a = 0;
else
   a = 1;
```

Program 4-22 can be used to help a consultant calculate her charges. Her rate is $50.00 per hour, but her minimum charge is for five hours. The conditional operator is used in a statement that ensures the number of hours does not go below five.

Program 4-22

```
1  // This program calculates a consultant's charges at $50
2  // per hour, for a minimum of 5 hours. The ?: operator
3  // adjusts hours to 5 if less than 5 hours were worked.
4  #include <iostream>
5  #include <iomanip>
6  using namespace std;
7
8  int main()
9  {
10     const double PAY_RATE = 50.0; // Hourly pay rate
11     const int MIN_HOURS = 5;      // Minimum billable hours
12     double hours,                 // Hours worked
13            charges;               // Total charges
14
15     // Get the hours worked.
16     cout << "How many hours were worked? ";
17     cin >> hours;
18
19     // Determine the hours to charge for.
20     hours = hours < MIN_HOURS ? MIN_HOURS : hours;
21
22     // Calculate and display the charges.
23     charges = PAY_RATE * hours;
24     cout << fixed << showpoint << setprecision(2)
25          << "The charges are $" << charges << endl;
26     return 0;
27 }
```

Program Output with Example Input Shown in Bold

```
How many hours were worked? 10 [Enter]
The charges are $500.00
```

Program Output with Example Input Shown in Bold

```
How many hours were worked? 2 [Enter]
The charges are $250.00
```

Notice that in line 11 a constant named `MIN_HOURS` is defined to represent the minimum number of hours, which is 5. Here is the statement in line 20, with the conditional expression:

```
hours = hours < MIN_HOURS ? MIN_HOURS : hours;
```

If the value in `hours` is less than 5, then 5 is stored in `hours`. Otherwise `hours` is assigned the value it already has. The `hours` variable will not have a value less than 5 when it is used in the next statement, which calculates the consultant's charges.

As you can see, the conditional operator gives you the ability to pack decision-making power into a concise line of code. With a little imagination it can be applied to many other programming problems. For instance, consider the following statement:

```
cout << "Your grade is: " << (score < 60 ? "Fail." : "Pass.");
```

If you were to use an `if/else` statement, the statement above would be written as follows:

```
if (score < 60)
   cout << "Your grade is: Fail.";
else
   cout << "Your grade is: Pass.";
```

NOTE: The parentheses are placed around the conditional expression because the `<<` operator has higher precedence than the `?:` operator. Without the parentheses, just the value of the expression `score < 60` would be sent to `cout`.

Checkpoint

4.24 Rewrite the following `if/else` statements as conditional expressions:

A)
```
if (x > y)
      z = 1;
   else
      z = 20;
```

B)
```
if (temp > 45)
      population = base * 10;
   else
      population = base * 2;
```

C)
```
if (hours > 40)
      wages *= 1.5;
   else
      wages *= 1;
```

D)
```
if (result >= 0)
      cout << "The result is positive\n";
   else
      cout << "The result is negative.\n";
```

4.25 The following statements use conditional expressions. Rewrite each with an `if/else` statement.

A) `j = k > 90 ? 57 : 12;`
B) `factor = x >= 10 ? y * 22 : y * 35;`
C) `total += count == 1 ? sales : count * sales;`
D) `cout << (((num % 2) == 0) ? "Even\n" : "Odd\n");`

4.26 What will the following program display?

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

int main()
{
   const int UPPER = 8, LOWER = 2;
   int num1, num2, num3 = 12, num4 = 3;

   num1 = num3 < num4 ? UPPER : LOWER;
   num2 = num4 > UPPER ? num3 : LOWER;
   cout << num1 << " " << num2 << endl;
   return 0;
}
```

4.14 The `switch` Statement

CONCEPT: **The `switch` statement lets the value of a variable or expression determine where the program will branch.**

A branch occurs when one part of a program causes another part to execute. The `if/else if` statement allows your program to branch into one of several possible paths. It performs a series of tests (usually relational) and branches when one of these tests is true. The `switch` statement is a similar mechanism. It, however, tests the value of an integer expression and then uses that value to determine which set of statements to branch to. Here is the format of the `switch` statement:

```
switch (IntegerExpression)
{
   case ConstantExpression:
        // place one or more
        // statements here

   case ConstantExpression:
        // place one or more
        // statements here

    // case statements may be repeated as many
    // times as necessary

   default:
        // place one or more
        // statements here
}
```

The first line of the statement starts with the word `switch`, followed by an integer expression inside parentheses. This can be either of the following:

- a variable of any of the integer data types (including `char`)
- an expression whose value is of any of the integer data types

On the next line is the beginning of a block containing several `case` statements. Each `case` statement is formatted in the following manner:

```
case ConstantExpression:
     // place one or more
     // statements here
```

After the word `case` is a constant expression (which must be of an integer type), followed by a colon. The constant expression may be an integer literal or an integer named constant. The `case` statement marks the beginning of a section of statements. The program branches to these statements if the value of the `switch` expression matches that of the `case` expression.

WARNING! The expression of each `case` statement in the block must be unique.

NOTE: The expression following the word `case` must be an integer literal or constant. It cannot be a variable, and it cannot be an expression such as `x < 22` or `n == 50`.

An optional `default` section comes after all the `case` statements. The program branches to this section if none of the `case` expressions match the `switch` expression. So, it functions like a trailing `else` in an `if/else if` statement.

Program 4-23 shows how a simple `switch` statement works.

Program 4-23

```
1  // The switch statement in this program tells the user something
2  // he or she already knows: the data just entered!
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       char choice;
9
10      cout << "Enter A, B, or C: ";
11      cin >> choice;
12      switch (choice)
13      {
14           case 'A': cout << "You entered A.\n";
15                     break;
16           case 'B': cout << "You entered B.\n";
17                     break;
18           case 'C': cout << "You entered C.\n";
19                     break;
20           default:  cout << "You did not enter A, B, or C!\n";
21      }
22      return 0;
23 }
```

(program continues)

Program 4-23 *(continued)*

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: B [Enter]
You entered B.
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: F [Enter]
You did not enter A, B, or C!
```

The first `case` statement is `case 'A':`, the second is `case 'B':`, and the third is `case 'C':`. These statements mark where the program is to branch to if the variable `choice` contains the values `'A'`, `'B'`, or `'C'`. (Remember, character variables and literals are considered integers.) The `default` section is branched to if the user enters anything other than A, B, or C.

Notice the `break` statements that are in the `case 'A'`, `case 'B'`, and `case 'C'` sections.

```
switch (choice)
{
   case 'A':cout << "You entered A.\n";
            break;  <----------
   case 'B':cout << "You entered B.\n";
            break;  <----------
   case 'C':cout << "You entered C.\n";
            break;  <----------
   default: cout << "You did not enter A, B, or C!\n";
}
```

The `case` statements show the program where to start executing in the block and the `break` statements show the program where to stop. Without the `break` statements, the program would execute all of the lines from the matching `case` statement to the end of the block.

NOTE: The `default` section (or the last `case` section, if there is no `default`) does not need a `break` statement. Some programmers prefer to put one there anyway, for consistency.

Program 4-24 is a modification of Program 4-23, without the `break` statements.

Program 4-24

```
1  // The switch statement in this program tells the user something
2  // he or she already knows: the data just entered!
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      char choice;
9
```

```
10       cout << "Enter A, B, or C: ";
11       cin >> choice;
12       // The following switch is
13       // missing its break statements!
14       switch (choice)
15       {
16           case 'A': cout << "You entered A.\n";
17           case 'B': cout << "You entered B.\n";
18           case 'C': cout << "You entered C.\n";
19           default:  cout << "You did not enter A, B, or C!\n";
20       }
21       return 0;
22   }
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: A [Enter]
You entered A.
You entered B.
You entered C.
You did not enter A, B, or C!
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: C [Enter]
You entered C.
You did not enter A, B, or C!
```

Without the `break` statement, the program "falls through" all of the statements below the one with the matching `case` expression. Sometimes this is what you want. Program 4-25 lists the features of three TV models a customer may choose from. The Model 100 has remote control. The Model 200 has remote control and stereo sound. The Model 300 has remote control, stereo sound, and picture-in-a-picture capability. The program uses a `switch` statement with carefully omitted `break`s to print the features of the selected model.

Program 4-25

```
1   // This program is carefully constructed to use the "fall through"
2   // feature of the switch statement.
3   #include <iostream>
4   using namespace std;
5
6   int main()
7   {
8        int modelNum; // Model number
9
10       // Get a model number from the user.
11       cout << "Our TVs come in three models:\n";
12       cout << "The 100, 200, and 300. Which do you want? ";
13       cin >> modelNum;
14
```

(program continues)

Program 4-25 *(continued)*

```
15       // Display the model's features.
16       cout << "That model has the following features:\n";
17       switch (modelNum)
18       {
19            case 300: cout << "\tPicture-in-a-picture.\n";
20            case 200: cout << "\tStereo sound.\n";
21            case 100: cout << "\tRemote control.\n";
22                      break;
23            default:  cout << "You can only choose the 100,";
24                      cout << "200, or 300.\n";
25       }
26       return 0;
27  }
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 100 [Enter]
That model has the following features:
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 200 [Enter]
That model has the following features:
   Stereo sound.
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 300 [Enter]
That model has the following features:
   Picture-in-a-picture.
   Stereo sound.
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 500 [Enter]
That model has the following features:
You can only choose the 100, 200, or 300.
```

Another example of how useful this "fall through" capability can be is when you want the program to branch to the same set of statements for multiple `case` expressions. For instance, Program 4-26 asks the user to select a grade of pet food. The available choices are A, B, and C. The `switch` statement will recognize either upper or lowercase letters.

Program 4-26

```
1   // The switch statement in this program uses the "fall through"
2   // feature to catch both uppercase and lowercase letters entered
3   // by the user.
4   #include <iostream>
```

```
using namespace std;

int main()
{
     char feedGrade;

     // Get the desired grade of feed.
     cout << "Our pet food is available in three grades:\n";
     cout << "A, B, and C. Which do you want pricing for? ";
     cin >> feedGrade;

     // Display the price.
     switch(feedGrade)
     {
          case 'a':
          case 'A': cout << "30 cents per pound.\n";
                    break;
          case 'b':
          case 'B': cout << "20 cents per pound.\n";
                    break;
          case 'c':
          case 'C': cout << "15 cents per pound.\n";
                    break;
          default:  cout << "That is an invalid choice.\n";
     }
     return 0;
}
```

Program Output with Example Input Shown in Bold

```
Our pet food is available in three grades:
A, B, and C. Which do you want pricing for? b [Enter]
20 cents per pound.
```

Program Output with Example Input Shown in Bold

```
Our pet food is available in three grades:
A, B, and C. Which do you want pricing for? B [Enter]
20 cents per pound.
```

When the user enters `'a'` the corresponding `case` has no statements associated with it, so the program falls through to the next `case`, which corresponds with `'A'`.

```
case 'a':
case 'A': cout << "30 cents per pound.\n";
          break;
```

The same technique is used for `'b'` and `'c'`.

Using `switch` in Menu Systems

The `switch` statement is a natural mechanism for building menu systems. Recall that Program 4-18 gives a menu to select which health club package the user wishes to purchase. The program uses `if/else if` statements to determine which package the user has selected and displays the calculated charges. Program 4-27 is a modification of that program, using a `switch` statement instead of `if/else if`.

Program 4-27

```
// This program uses a switch statement to determine
// the item selected from a menu.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
     int choice;      // To hold a menu choice
     int months;      // To hold the number of months
     double charges;  // To hold the monthly charges

     // Constants for membership rates
     const double ADULT = 40.0,
                  CHILD = 20.0,
                  SENIOR = 30.0;

     // Constants for menu choices
     const int ADULT_CHOICE = 1,
               CHILD_CHOICE = 2,
               SENIOR_CHOICE = 3,
               QUIT_CHOICE = 4;

     // Display the menu and get a choice.
     cout << "\t\tHealth Club Membership Menu\n\n"
          << "1. Standard Adult Membership\n"
          << "2. Child Membership\n"
          << "3. Senior Citizen Membership\n"
          << "4. Quit the Program\n\n"
          << "Enter your choice: ";
     cin >> choice;

     // Set the numeric output formatting.
     cout << fixed << showpoint << setprecision(2);

     // Respond to the user's menu selection.
     switch (choice)
     {
          case ADULT_CHOICE:
               cout << "For how many months? ";
               cin >> months;
               charges = months * ADULT;
               cout << "The total charges are $" << charges << endl;
               break;

          case CHILD_CHOICE:
               cout << "For how many months? ";
               cin >> months;
               charges = months * CHILD;
               cout << "The total charges are $" << charges << endl;
               break;

```

```
53          case SENIOR_CHOICE:
54               cout << "For how many months? ";
55               cin >> months;
56               charges = months * SENIOR;
57               cout << "The total charges are $" << charges << endl;
58               break;
59
60          case QUIT_CHOICE:
61               cout << "Program ending.\n";
62               break;
63
64          default:
65               cout << "The valid choices are 1 through 4. Run the\n"
66                    << "program again and select one of those.\n";
67      }
68
69      return 0;
70  }
```

Program Output with Example Input Shown in Bold

```
             Health Club Membership Menu

1. Standard Adult Membership
2. Child Membership
3. Senior Citizen Membership
4. Quit the Program

Enter your choice: 2 [Enter]
For how many months? 6 [Enter]
The total charges are $120.00
```

Checkpoint

4.27 Explain why you cannot convert the following `if/else if` statement into a `switch` statement.

```
if (temp == 100)
   x = 0;
else if (population > 1000)
   x = 1;
else if (rate < .1)
   x = -1;
```

4.28 What is wrong with the following `switch` statement?

```
switch (temp)
{
   case temp < 0 : cout << "Temp is negative.\n";
                   break;
   case temp == 0: cout << "Temp is zero.\n";
                   break;
   case temp > 0 : cout << "Temp is positive.\n";
                   break;
}
```

4.29 What will the following program display?

```
#include <iostream>
using namespace std;
int main()
{
   int funny = 7, serious = 15;

   funny = serious * 2;
   switch (funny)
   {     case 0 : cout << "That is funny.\n";
                  break;
         case 30: cout << "That is serious.\n";
                  break;
         case 32: cout << "That is seriously funny.\n";
                  break;
         default: cout << funny << endl;
    }
    return 0;
}
```

4.30 Complete the following program skeleton by writing a `switch` statement that displays "one" if the user has entered 1, "two" if the user has entered 2, and "three" if the user has entered 3. If a number other than 1, 2, or 3 is entered, the program should display an error message.

```
#include <iostream>
using namespace std;
int main()
{
   int userNum;
   cout << "Enter one of the numbers 1, 2, or 3: ";
   cin >> userNum;
   //
   // Write the switch statement here.
   //
   return 0;
}
```

4.31 Rewrite the following program. Use a `switch` statement instead of the `if/else if` statement.

```
#include <iostream>
using namespace std;
int main()
{
   int selection;
   cout << "Which formula do you want to see?\n\n";
   cout << "1. Area of a circle\n";
   cout << "2. Area of a rectangle\n";
   cout << "3. Area of a cylinder\n"
   cout << "4. None of them!\n";
   cin >> selection;
   if (selection == 1)
      cout << "Pi times radius squared\n";
```

```
        else if (selection == 2)
           cout << "Length times width\n";
        else if (selection == 3)
           cout << "Pi times radius squared times height\n";
        else if (selection == 4)
           cout << "Well okay then, good bye!\n";
        else
           cout << "Not good with numbers, eh?\n";
        return 0;
    }
```

4.15 More About Blocks and Variable Scope

CONCEPT: **The scope of a variable is limited to the block in which it is defined.**

C++ allows you to create variables almost anywhere in a program. Program 4-28 is a modification of Program 4-17, which determines if the user qualifies for a loan. The definitions of the variables `income` and `years` have been moved to later points in the program.

Program 4-28

```
1  // This program demonstrates late variable definition
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       // Constants for minimum income and years
8       const double MIN_INCOME = 35000.0;
9       const int MIN_YEARS = 5;
10
11      // Get the annual income.
12      cout << "What is your annual income? ";
13      double income;   // Variable definition
14      cin >> income;
15
16      // Get the number of years at the current job.
17      cout << "How many years have you worked at "
18           << "your current job? ";
19      int years;       // Variable definition
20      cin >> years;
21
22      // Determine the user's loan qualifications.
23      if (income >= MIN_INCOME || years > MIN_YEARS)
24         cout << "You qualify.\n";
25      else
26      {
27          cout << "You must earn at least $"
28               << MIN_INCOME << " or have been "
29               << "employed more than " << MIN_YEARS
30               << " years.\n";
31      }
32      return 0;
33 }
```

It is a common practice to define all of a function's variables at the top of the function. Sometimes, especially in longer programs, it's a good idea to define variables near the part of the program where they are used. This makes the purpose of the variable more evident.

Recall from Chapter 2 that the scope of a variable is defined as the part of the program where the variable may be used.

In Program 4-28, the scope of the `income` variable is the part of the program in lines 13 through 32. The scope of the `years` variable is the part of the program in lines 19 through 32.

The variables `income` and `years` are defined inside function `main`'s braces. Variables defined inside a set of braces have *local scope* or *block scope*. They may only be used in the part of the program between their definition and the block's closing brace.

You may define variables inside any block. For example, look at Program 4-29. This version of the loan program has the variable `years` defined inside the block of the `if` statement. The scope of `years` is the part of the program in lines 21 through 31.

Program 4-29

```
1  // This program demonstrates a variable defined in an inner block.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      // Constants for minimum income and years
8      const double MIN_INCOME = 35000.0;
9      const int MIN_YEARS = 5;
10
11     // Get the annual income.
12     cout << "What is your annual income? ";
13     double income;   // Variable definition
14     cin >> income;
15
16     if (income >= MIN_INCOME)
17     {
18         // Get the number of years at the current job.
19         cout << "How many years have you worked at "
20              << "your current job? ";
21         int years;  // Variable definition
22         cin >> years;
23
24         if (years > MIN_YEARS)
25            cout << "You qualify.\n";
26         else
27         {
28            cout << "You must have been employed for\n"
29                 << "more than " << MIN_YEARS
30                 << " years to qualify.\n";
31         }
32     }
33     else
```

```
34       {
35            cout << "You must earn at least $" << MIN_INCOME
36                 << " to qualify.\n";
37       }
38       return 0;
39  }
```

Notice the scope of `years` is only within the block where it is defined. The variable is not visible before its definition or after the closing brace of the block. This is true of any variable defined inside a set of braces.

NOTE: When a program is running and it enters the section of code that constitutes a variable's scope, it is said that the variable *comes into scope*. This simply means the variable is now visible and the program may reference it. Likewise, when a variable *leaves scope*, it may no longer be used.

Variables with the Same Name

When a block is nested inside another block, a variable defined in the inner block may have the same name as a variable defined in the outer block. As long as the variable in the inner block is visible, however, the variable in the outer block will be hidden. This is illustrated by Program 4-30.

Program 4-30

```
1   // This program uses two variables with the name number.
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
7        // Define a variable named number.
8        int number;
9
10       cout << "Enter a number greater than 0: ";
11       cin >> number;
12       if (number > 0)
13       {
14            int number; // Another variable named number.
15            cout << "Now enter another number: ";
16            cin >> number;
17            cout << "The second number you entered was "
18                 << number << endl;
19       }
20       cout << "Your first number was " << number << endl;
21       return 0;
22  }
```

Program Output with Example Input Shown in Bold

```
Enter a number greater than 0: 2 [Enter]
Now enter another number: 7 [Enter]
The second number you entered was 7
Your first number was 2
```

Program 4-30 has two separate variables named `number`. The `cin` and `cout` statements in the inner block (belonging to the `if` statement) can only work with the `number` variable defined in that block. As soon as the program leaves that block, the inner `number` goes out of scope, revealing the outer `number` variable.

WARNING! Although it's perfectly acceptable to define variables inside nested blocks, you should avoid giving them the same names as variables in the outer blocks. It's too easy to confuse one variable with another.

Case Study: See the Sales Commission Case Study on this book's companion Web site at www.pearsonhighered.com/gaddis.

Review Questions and Exercises

Short Answer

1. Describe the difference between the `if/else if` statement and a series of `if` statements.
2. In an `if/else if` statement, what is the purpose of a trailing `else`?
3. What is a flag and how does it work?
4. Can an `if` statement test expressions other than relational expressions? Explain.
5. Briefly describe how the `&&` operator works.
6. Briefly describe how the `||` operator works.
7. Why are the relational operators called relational?
8. Why do most programmers indent the conditionally executed statements in a decision structure?

Fill-in-the-Blank

9. An expression using the greater-than, less-than, greater-than-or-equal to, less-than-or-equal-to, equal, or not-equal to operator is called a(n) _____ expression.
10. A relational expression is either _____ or _____.
11. The value of a relational expression is 0 if the expression is _____ or 1 if the expression is _____.
12. The `if` statement regards an expression with the value 0 as _____.
13. The `if` statement regards an expression with a nonzero value as _____.
14. For an `if` statement to conditionally execute a group of statements, the statements must be enclosed in a set of _____.
15. In an `if/else` statement, the `if` part executes its statement or block if the expression is _____, and the `else` part executes its statement or block if the expression is _____.
16. The trailing `else` in an `if/else if` statement has a similar purpose as the _____ section of a `switch` statement.

17. The `if/else if` statement is actually a form of the _____ `if` statement.
18. If the sub-expression on the left of the _____ logical operator is false, the right sub-expression is not checked.
19. If the sub-expression on the left of the _____ logical operator is true, the right sub-expression is not checked.
20. The _____ logical operator has higher precedence than the other logical operators.
21. The logical operators have _____ associativity.
22. The _____ logical operator works best when testing a number to determine if it is within a range.
23. The _____ logical operator works best when testing a number to determine if it is outside a range.
24. A variable with _____ scope is only visible when the program is executing in the block containing the variable's definition.
25. You use the _____ operator to determine whether one `string` object is greater then another `string` object.
26. An expression using the conditional operator is called a(n) _____ expression.
27. The expression that is tested by a `switch` statement must have a(n) _____ value.
28. The expression following a `case` statement must be a(n) _____ _____.
29. A program will "fall through" a `case` section if it is missing the _____ statement.
30. What value will be stored in the variable `t` after each of the following statements executes?

    A) `t = (12 > 1);` _____

    B) `t = (2 < 0);` _____

    C) `t = (5 == (3 * 2));` _____

    D) `t = (5 == 5);` _____

Algorithm Workbench

31. Write an `if` statement that assigns 100 to `x` when `y` is equal to 0.
32. Write an `if/else` statement that assigns 0 to `x` when `y` is equal to 10. Otherwise it should assign 1 to `x`.
33. Using the following chart, write an `if/else if` statement that assigns .10, .15, or .20 to `commission`, depending on the value in `sales`.

Sales	Commission Rate
Up to $10,000	10%
$10,000 to $15,000	15%
Over $15,000	20%

34. Write an `if` statement that sets the variable `hours` to 10 when the flag variable `minimum` is set.
35. Write nested `if` statements that perform the following tests: If `amount1` is greater than 10 and `amount2` is less than 100, display the greater of the two.

36. Write an `if` statement that prints the message "The number is valid" if the variable `grade` is within the range 0 through 100.
37. Write an `if` statement that prints the message "The number is valid" if the variable `temperature` is within the range –50 through 150.
38. Write an `if` statement that prints the message "The number is not valid" if the variable `hours` is outside the range 0 through 80.
39. Assume `str1` and `str2` are `string` objects that have been initialized with different values. Write an `if/else` statement that compares the two objects and displays the one that is alphabetically greatest.
40. Convert the following `if/else if` statement into a `switch` statement:

```
if (choice == 1)
{
    cout << fixed << showpoint << setprecision(2);
}
else if (choice == 2 || choice == 3)
{
    cout << fixed << showpoint << setprecision(4);
}
else if (choice == 4)
{
    cout << fixed << showpoint << setprecision(6);
}
else
{
    cout << fixed << showpoint << setprecision(8);
}
```

41. Match the conditional expression with the `if/else` statement that performs the same operation.

A) `q = x < y ? a + b : x * 2;`

B) `q = x < y ? x * 2 : a + b;`

C) `x < y ? q = 0 : q = 1;`

```
____ if (x < y)
         q = 0;
     else
         q = 1;

____ if (x < y)
         q = a + b;
     else
         q = x * 2;

____ if (x < y)
         q = x * 2;
     else
         q = a + b;
```

True or False

42. T F The = operator and the == operator perform the same operation when used in a Boolean expression.

43. T F A variable defined in an inner block may not have the same name as a variable defined in the outer block.
44. T F A conditionally executed statement should be indented one level from the `if` statement.
45. T F All lines in a block should be indented one level.
46. T F It's safe to assume that all uninitialized variables automatically start with 0 as their value.
47. T F When an `if` statement is nested in the `if` part of another statement, the only time the inner `if` is executed is when the expression of the outer `if` is true.
48. T F When an `if` statement is nested in the `else` part of another statement, as in an `if/else if`, the only time the inner `if` is executed is when the expression of the outer `if` is true.
49. T F The scope of a variable is limited to the block in which it is defined.
50. T F You can use the relational operators to compare `string` objects.
51. T F `x != y` is the same as `(x > y || x < y)`
52. T F `y < x` is the same as `x >= y`
53. T F `x >= y` is the same as `(x > y && x = y)`

Assume the variables `x = 5`, `y = 6`, and `z = 8`. Indicate by circling the T or F whether each of the following conditions is true or false:

54. T F `x == 5 || y > 3`
55. T F `7 <= x && z > 4`
56. T F `2 != y && z != 4`
57. T F `x >= 0 || x <= y`

Find the Errors

Each of the following programs has errors. Find as many as you can.

58.
```
// This program averages 3 test scores.
// It uses the variable perfectScore as a flag.
include <iostream>
using namespace std;

int main()
{
     cout << "Enter your 3 test scores and I will ";
          << "average them:";
     int score1, score2, score3,
     cin >> score1 >> score2 >> score3;
     double average;
     average = (score1 + score2 + score3) / 3.0;
     if (average = 100);
          perfectScore = true; // Set the flag variable
     cout << "Your average is " << average << endl;
     bool perfectScore;
     if (perfectScore);
     {
          cout << "Congratulations!\n";
          cout << "That's a perfect score.\n";
          cout << "You deserve a pat on the back!\n";
          return 0;
}
```

59.
```
// This program divides a user-supplied number by another
// user-supplied number. It checks for division by zero.
#include <iostream>
using namespace std;

int main()
{
     double num1, num2, quotient;

     cout << "Enter a number: ";
     cin >> num1;
     cout << "Enter another number: ";
     cin >> num2;
     if (num2 == 0)
        cout << "Division by zero is not possible.\n";
        cout << "Please run the program again ";
        cout << "and enter a number besides zero.\n";
     else
        quotient = num1 / num2;
        cout << "The quotient of " << num1 <<
        cout << " divided by " << num2 << " is ";
        cout << quotient << endl;
     return 0;
}
```

60.
```
// This program uses an if/else if statement to assign a
// letter grade (A, B, C, D, or F) to a numeric test score.
#include <iostream>
using namespace std;

int main()
{
     int testScore;

     cout << "Enter your test score and I will tell you\n";
     cout << "the letter grade you earned: ";
     cin >> testScore;
     if (testScore < 60)
          cout << "Your grade is F.\n";
     else if (testScore < 70)
          cout << "Your grade is D.\n";
     else if (testScore < 80)
          cout << "Your grade is C.\n";
     else if (testScore < 90)
          cout << "Your grade is B.\n";
     else
          cout << "That is not a valid score.\n";
     else if (testScore <= 100)
          cout << "Your grade is A.\n";
     return 0;
}
```

61.
```
// This program uses a switch-case statement to assign a
// letter grade (A, B, C, D, or F) to a numeric test score.
#include <iostream>
using namespace std;

int main()
{
    double testScore;

    cout << "Enter your test score and I will tell you\n";
    cout << "the letter grade you earned: ";
    cin >> testScore;
    switch (testScore)
    {
        case (testScore < 60.0):
                  cout << "Your grade is F.\n";
                  break;
        case (testScore < 70.0):
                  cout << "Your grade is D.\n";
                  break;
        case (testScore < 80.0):
                  cout << "Your grade is C.\n";
                  break;
        case (testScore < 90.0):
                  cout << "Your grade is B.\n";
                  break;
        case (testScore <= 100.0):
                  cout << "Your grade is A.\n";
                  break;
        default:
                  cout << "That score isn't valid\n";
    return 0;
}
```

62. The following statement should determine if `x` is not greater than 20. What is wrong with it?

```
if (!x > 20)
```

63. The following statement should determine if `count` is within the range of 0 through 100. What is wrong with it?

```
if (count >= 0 || count <= 100)
```

64. The following statement should determine if `count` is outside the range of 0 through 100. What is wrong with it?

```
if (count < 0 && count > 100)
```

65. The following statement should assign 0 to `z` if `a` is less than 10, otherwise it should assign 7 to `z`. What is wrong with it?

```
z = (a < 10) : 0 ? 7;
```

Programming Challenges

1. **Minimum/Maximum**

   Write a program that asks the user to enter two numbers. The program should use the conditional operator to determine which number is the smaller and which is the larger.

2. **Roman Numeral Converter**

   Write a program that asks the user to enter a number within the range of 1 through 10. Use a `switch` statement to display the Roman numeral version of that number.

   Input Validation: Do not accept a number less than 1 or greater than 10.

3. **Magic Dates**

   The date June 10, 1960 is special because when we write it in the following format, the month times the day equals the year.

   6/10/60

   Write a program that asks the user to enter a month (in numeric form), a day, and a two-digit year. The program should then determine whether the month times the day is equal to the year. If so, it should display a message saying the date is magic. Otherwise it should display a message saying the date is not magic.

4. **Areas of Rectangles**

   The area of a rectangle is the rectangle's length times its width. Write a program that asks for the length and width of two rectangles. The program should tell the user which rectangle has the greater area, or if the areas are the same.

5. **Body Mass Index**

   Write a program that calculates and displays a person's body mass index (BMI). The BMI is often used to determine whether a person with a sedentary lifestyle is overweight or underweight for his or her height. A person's BMI is calculated with the following formula:

   $BMI = weight \times 703 / height^2$

   where *weight* is measured in pounds and *height* is measured in inches. The program should display a message indicating whether the person has optimal weight, is underweight, or is overweight. A sedentary person's weight is considered to be optimal if his or her BMI is between 18.5 and 25. If the BMI is less than 18.5, the person is considered to be underweight. If the BMI value is greater than 25, the person is considered to be overweight.

6. **Mass and Weight**

   Scientists measure an object's mass in kilograms and its weight in newtons. If you know the amount of mass that an object has, you can calculate its weight, in newtons, with the following formula:

   $Weight = mass \times 9.8$

   Write a program that asks the user to enter an object's mass, and then calculates and displays its weight. If the object weighs more than 1,000 newtons, display a message indicating that it is too heavy. If the object weighs less than 10 newtons, display a message indicating that the object is too light.

7. Time Calculator


VideoNote
Solving the Time Calculator Problem

Write a program that asks the user to enter a number of seconds.

- There are 60 seconds in a minute. If the number of seconds entered by the user is greater than or equal to 60, the program should display the number of minutes in that many seconds.
- There are 3,600 seconds in an hour. If the number of seconds entered by the user is greater than or equal to 3,600, the program should display the number of hours in that many seconds.
- There are 86,400 seconds in a day. If the number of seconds entered by the user is greater than or equal to 86,400, the program should display the number of days in that many seconds.

8. Color Mixer

The colors red, blue, and yellow are known as the primary colors because they cannot be made by mixing other colors. When you mix two primary colors, you get a secondary color, as shown here:

When you mix red and blue, you get purple.
When you mix red and yellow, you get orange.
When you mix blue and yellow, you get green.

Write a program that prompts the user to enter the names of two primary colors to mix. If the user enters anything other than "red," "blue," or "yellow," the program should display an error message. Otherwise, the program should display the name of the secondary color that results by mixing two primary colors.

9. Change for a Dollar Game

Create a change-counting game that gets the user to enter the number of coins required to make exactly one dollar. The program should ask the user to enter the number of pennies, nickels, dimes, and quarters. If the total value of the coins entered is equal to one dollar, the program should congratulate the user for winning the game. Otherwise, the program should display a message indicating whether the amount entered was more than or less than one dollar.

10. Days in a Month

Write a program that asks the user to enter the month (letting the user enter an integer in the range of 1 through 12) and the year. The program should then display the number of days in that month. Use the following criteria to identify leap years:

1. Determine whether the year is divisible by 100. If it is, then it is a leap year if and only if it is divisible by 400. For example, 2000 is a leap year but 2100 is not.
2. If the year is not divisible by 100, then it is a leap year if and if only it is divisible by 4. For example, 2008 is a leap year but 2009 is not.

Here is a sample run of the program:

```
Enter a month (1-12): 2 [Enter]
Enter a year: 2008 [Enter]
29 days
```

11. Math Tutor

This is a modification of Programming Challenge 17 from Chapter 3. Write a program that can be used as a math tutor for a young student. The program should display two random numbers that are to be added, such as:

```
  247
+ 129
```

The program should wait for the student to enter the answer. If the answer is correct, a message of congratulations should be printed. If the answer is incorrect, a message should be printed showing the correct answer.

12. Software Sales

A software company sells a package that retails for $99. Quantity discounts are given according to the following table.

Quantity	Discount
10–19	20%
20–49	30%
50–99	40%
100 or more	50%

Write a program that asks for the number of units sold and computes the total cost of the purchase.

Input Validation: Make sure the number of units is greater than 0.

13. Book Club Points

Serendipity Booksellers has a book club that awards points to its customers based on the number of books purchased each month. The points are awarded as follows:

- If a customer purchases 0 books, he or she earns 0 points.
- If a customer purchases 1 book, he or she earns 5 points.
- If a customer purchases 2 books, he or she earns 15 points.
- If a customer purchases 3 books, he or she earns 30 points.
- If a customer purchases 4 or more books, he or she earns 60 points.

Write a program that asks the user to enter the number of books that he or she has purchased this month and then displays the number of points awarded.

14. Bank Charges

A bank charges $10 per month plus the following check fees for a commercial checking account:

$.10 each for fewer than 20 checks
$.08 each for 20–39 checks
$.06 each for 40–59 checks
$.04 each for 60 or more checks

The bank also charges an extra $15 if the balance of the account falls below $400 (before any check fees are applied). Write a program that asks for the beginning balance and the number of checks written. Compute and display the bank's service fees for the month.

Input Validation: Do not accept a negative value for the number of checks written. If a negative value is given for the beginning balance, display an urgent message indicating the account is overdrawn.

15. Shipping Charges

The Fast Freight Shipping Company charges the following rates:

Weight of Package (in Kilograms)	Rate per 500 Miles Shipped
2 kg or less	$1.10
Over 2 kg but not more than 6 kg	$2.20
Over 6 kg but not more than 10 kg	$3.70
Over 10 kg but not more than 20 kg	$4.80

Write a program that asks for the weight of the package and the distance it is to be shipped, and then displays the charges.

Input Validation: Do not accept values of 0 or less for the weight of the package. Do not accept weights of more than 20 kg (this is the maximum weight the company will ship). Do not accept distances of less than 10 miles or more than 3,000 miles. These are the company's minimum and maximum shipping distances.

16. Running the Race

Write a program that asks for the names of three runners and the time it took each of them to finish a race. The program should display who came in first, second, and third place.

Input Validation: Only accept positive numbers for the times.

17. Personal Best

Write a program that asks for the name of a pole vaulter and the dates and vault heights (in meters) of the athlete's three best vaults. It should then report, in order of height (best first), the date on which each vault was made and its height.

Input Validation: Only accept values between 2.0 and 5.0 for the heights.

18. Fat Gram Calculator

Write a program that asks for the number of calories and fat grams in a food. The program should display the percentage of calories that come from fat. If the calories from fat are less than 30% of the total calories of the food, it should also display a message indicating that the food is low in fat.

One gram of fat has 9 calories, so

Calories from fat = fat grams * 9

The percentage of calories from fat can be calculated as

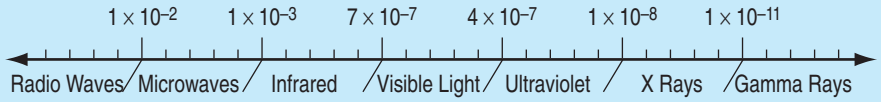
Calories from fat ÷ total calories

Input Validation: Make sure the number of calories and fat grams are not less than 0. Also, the number of calories from fat cannot be greater than the total number of calories. If that happens, display an error message indicating that either the calories or fat grams were incorrectly entered.

19. Spectral Analysis

If a scientist knows the wavelength of an electromagnetic wave, he or she can determine what type of radiation it is. Write a program that asks for the wavelength of an electromagnetic wave in meters and then displays what that wave is according to the

chart below. (For example, a wave with a wavelength of 1E-10 meters would be an X-ray.)



20. The Speed of Sound

The following table shows the approximate speed of sound in air, water, and steel.

Medium	Speed
Air	1,100 feet per second
Water	4,900 feet per second
Steel	16,400 feet per second

Write a program that displays a menu allowing the user to select air, water, or steel. After the user has made a selection, he or she should be asked to enter the distance a sound wave will travel in the selected medium. The program will then display the amount of time it will take. (Round the answer to four decimal places.)

Input Validation: Check that the user has selected one of the available choices from the menu. Do not accept distances less than 0.

21. The Speed of Sound in Gases

When sound travels through a gas, its speed depends primarily on the density of the medium. The less dense the medium, the faster the speed will be. The following table shows the approximate speed of sound at 0 degrees centigrade, measured in meters per second, when traveling through carbon dioxide, air, helium, and hydrogen.

Medium	Speed (Meters per Second)
Carbon Dioxide	258.0
Air	331.5
Helium	972.0
Hydrogen	1,270.0

Write a program that displays a menu allowing the user to select one of these four gases. After a selection has been made, the user should enter the number of seconds it took for the sound to travel in this medium from its source to the location at which it was detected. The program should then report how far away (in meters) the source of the sound was from the detection location.

Input Validation: Check that the user has selected one of the available menu choices. Do not accept times less than 0 seconds or more than 30 seconds.

22. Freezing and Boiling Points

The following table lists the freezing and boiling points of several substances. Write a program that asks the user to enter a temperature and then shows all the substances that will freeze at that temperature and all that will boil at that temperature. For example, if the user enters –20 the program should report that water will freeze and oxygen will boil at that temperature.

Substance	Freezing Point (°F)	Boiling Point (°F)
Ethyl alcohol	−173	172
Mercury	−38	676
Oxygen	−362	−306
Water	32	212

23. Geometry Calculator

Write a program that displays the following menu:

```
Geometry Calculator

    1. Calculate the Area of a Circle
    2. Calculate the Area of a Rectangle
    3. Calculate the Area of a Triangle
    4. Quit

    Enter your choice (1-4):
```

If the user enters 1, the program should ask for the radius of the circle and then display its area. Use the following formula:

area = πr^2

Use 3.14159 for π and the radius of the circle for *r*. If the user enters 2, the program should ask for the length and width of the rectangle and then display the rectangle's area. Use the following formula:

```
area = length * width
```

If the user enters 3 the program should ask for the length of the triangle's base and its height, and then display its area. Use the following formula:

```
area = base * height * .5
```

If the user enters 4, the program should end.

Input Validation: Display an error message if the user enters a number outside the range of 1 through 4 when selecting an item from the menu. Do not accept negative values for the circle's radius, the rectangle's length or width, or the triangle's base or height.

24. Long-Distance Calls

A long-distance carrier charges the following rates for telephone calls:

Starting Time of Call	Rate per Minute
00:00–06:59	0.05
07:00–19:00	0.45
19:01–23:59	0.20

Write a program that asks for the starting time and the number of minutes of the call, and displays the charges. The program should ask for the time to be entered as a floating-point number in the form HH.MM. For example, 07:00 hours will be entered as 07.00, and 16:28 hours will be entered as 16.28.

Input Validation: The program should not accept times that are greater than 23:59. Also, no number whose last two digits are greater than 59 should be accepted. Hint: Assuming num is a floating-point variable, the following expression will give you its fractional part:

```
num - static_cast<int>(num)
```

25. Mobile Service Provider

A mobile phone service provider has three different subscription packages for its customers:

Package A: For $39.99 per month 450 minutes are provided. Additional minutes are $0.45 per minute.

Package B: For $59.99 per month 900 minutes are provided. Additional minutes are $0.40 per minute.

Package C: For $69.99 per month unlimited minutes provided.

Write a program that calculates a customer's monthly bill. It should ask which package the customer has purchased and how many minutes were used. It should then display the total amount due.

Input Validation: Be sure the user only selects package A, B, or C.

26. Mobile Service Provider, Part 2

Modify the Program in Programming Challenge 25 so that it also displays how much money Package A customers would save if they purchased packages B or C, and how much money Package B customers would save if they purchased Package C. If there would be no savings, no message should be printed.

27. Mobile Service Provider, Part 3

Months with 30 days have 720 hours, and months with 31 days have 744 hours. February, with 28 days, has 672 hours. You can calculate the number of minutes in a month by multiplying its number of hours by 60. Enhance the input validation of the Mobile Service Provider program by asking the user for the month (by name), and validating that the number of minutes entered is not more than the maximum for the entire month. Here is a table of the months, their days, and number of hours in each.

Month	Days	Hours
January	31	744
February	28	672
March	31	744
April	30	720
May	31	744
June	30	720
July	31	744
August	31	744
September	30	720
October	31	744
November	30	720
December	31	744