CHAPTER 3

Expressions and Interactivity

TOPICS

3.1 The `cin` Object

CONCEPT: The `cin` object can be used to read data typed at the keyboard.

So far you have written programs with built-in data. Without giving the user an opportunity to enter his or her own data, you have initialized the variables with the necessary starting values. These types of programs are limited to performing their task with only a single set of starting data. If you decide to change the initial value of any variable, the program must be modified and recompiled.

VideoNote
Reading Input with `cin`

In reality, most programs ask for values that will be assigned to variables. This means the program does not have to be modified if the user wants to run it several times with different sets of data. For example, a program that calculates payroll for a small business might ask the user to enter the name of the employee, the hours worked, and the hourly pay rate. When the paycheck for that employee has been printed, the program could start over again and ask for the name, hours worked, and hourly pay rate of the next employee.

Just as `cout` is C++'s standard output object, `cin` is the standard input object. It reads input from the console (or keyboard) as shown in Program 3-1.

Program 3-1

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9      int length, width, area;
10
11     cout << "This program calculates the area of a ";
12     cout << "rectangle.\n";
13     cout << "What is the length of the rectangle? ";
14     cin >> length;
15     cout << "What is the width of the rectangle? ";
16     cin >> width;
17     area = length * width;
18     cout << "The area of the rectangle is " << area << ".\n";
19     return 0;
20 }
```

Program Output with Example Input Shown in Bold

```
This program calculates the area of a rectangle.
What is the length of the rectangle? 10 [Enter]
What is the width of the rectangle? 20 [Enter]
The area of the rectangle is 200.
```

Instead of calculating the area of one rectangle, this program can be used to get the area of any rectangle. The values that are stored in the `length` and `width` variables are entered by the user when the program is running. Look at lines 13 and 14:

```
cout << "What is the length of the rectangle? ";
cin >> length;
```

In line 13, the `cout` object is used to display the question "What is the length of the rectangle?" This question is known as a *prompt*, and it tells the user what data he or she should enter. Your program should always display a prompt before it uses `cin` to read input. This way, the user will know that he or she must type a value at the keyboard.

Line 14 uses the `cin` object to read a value from the keyboard. The `>>` symbol is the *stream extraction operator*. It gets characters from the stream object on its left and stores them in the variable whose name appears on its right. In this line, characters are taken from the `cin` object (which gets them from the keyboard) and are stored in the `length` variable.

Gathering input from the user is normally a two-step process:

1. Use the `cout` object to display a prompt on the screen.
2. Use the `cin` object to read a value from the keyboard.

The prompt should ask the user a question, or tell the user to enter a specific value. For example, the code we just examined from Program 3-1 displays the following prompt:

```
What is the length of the rectangle?
```

When the user sees this prompt, he or she knows to enter the rectangle's length. After the prompt is displayed, the program uses the `cin` object to read a value from the keyboard and store the value in the `length` variable.

Notice that the << and >> operators appear to point in the direction that data is flowing. In a statement that uses the `cout` object, the << operator always points toward `cout`. This indicates that data is flowing from a variable or a literal to the `cout` object. In a statement that uses the `cin` object, the >> operator always points toward the variable that is receiving the value. This indicates that data is flowing from `cin` to a variable. This is illustrated in Figure 3-1.

Figure 3-1

```
cout << "What is the length of the rectangle? ";
cin >> length;
```

Think of the << and >> operators as arrows that point in the direction that data is flowing.

```
cout ← "What is the length of the rectangle? ";
cin → length;
```

The `cin` object causes a program to wait until data is typed at the keyboard and the **[Enter]** key is pressed. No other lines in the program will be executed until `cin` gets its input.

`cin` automatically converts the data read from the keyboard to the data type of the variable used to store it. If the user types 10, it is read as the characters '1' and '0'. `cin` is smart enough to know this will have to be converted to an `int` value before it is stored in the `length` variable. `cin` is also smart enough to know a value like 10.7 cannot be stored in an integer variable. If the user enters a floating-point value for an integer variable, `cin` will not read the part of the number after the decimal point.

NOTE: You must include the `iostream` file in any program that uses `cin`.

Entering Multiple Values

The `cin` object may be used to gather multiple values at once. Look at Program 3-2, which is a modified version of Program 3-1.

Line 15 waits for the user to enter two values. The first is assigned to `length` and the second to `width`.

```
cin >> length >> width;
```

Program 3-2

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9       int length, width, area;
10
11      cout << "This program calculates the area of a ";
12      cout << "rectangle.\n";
13      cout << "Enter the length and width of the rectangle ";
14      cout << "separated by a space.\n";
15      cin >> length >> width;
16      area = length * width;
17      cout << "The area of the rectangle is " << area << endl;
18      return 0;
19 }
```

Program Output with Example Input Shown in Bold

```
This program calculates the area of a rectangle.
Enter the length and width of the rectangle separated by a space.
10 20 [Enter]
The area of the rectangle is 200
```

In the example output, the user entered 10 and 20, so 10 is stored in `length` and 20 is stored in `width`.

Notice the user separates the numbers by spaces as they are entered. This is how `cin` knows where each number begins and ends. It doesn't matter how many spaces are entered between the individual numbers. For example, the user could have entered

```
10          20
```

NOTE: The [Enter] key is pressed after the last number is entered.

`cin` will also read multiple values of different data types. This is shown in Program 3-3.

Program 3-3

```
1  // This program demonstrates how cin can read multiple values
2  // of different data types.
3  #include <iostream>
4  using namespace std;
5
```

```
 6  int main()
 7  {
 8       int whole;
 9       double fractional;
10       char letter;
11
12       cout << "Enter an integer, a double, and a character: ";
13       cin >> whole >> fractional >> letter;
14       cout << "Whole: " << whole << endl;
15       cout << "Fractional: " << fractional << endl;
16       cout << "Letter: " << letter << endl;
17       return 0;
18  }
```

Program Output with Example Input Shown in Bold

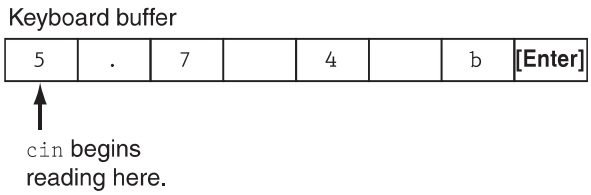
```
Enter an integer, a double, and a character: 4 5.7 b [Enter]
Whole: 4
Fractional: 5.7
Letter: b
```

As you can see in the example output, the values are stored in their respective variables. But what if the user had responded in the following way?

```
Enter an integer, a double, and a character: 5.7 4 b [Enter]
```

When the user types values at the keyboard, those values are first stored in an area of memory known as the *keyboard buffer*. So, when the user enters the values 5.7, 4, and b, they are stored in the keyboard buffer as shown in Figure 3-2.

Figure 3-2



When the user presses the Enter key, `cin` reads the value 5 into the variable `whole`. It does not read the decimal point because `whole` is an integer variable. Next it reads .7 and stores that value in the `double` variable `fractional`. The space is skipped, and 4 is the next value read. It is stored as a character in the variable `letter`. Because this `cin` statement reads only three values, the b is left in the keyboard buffer. So, in this situation the program would have stored 5 in `whole`, 0.7 in `fractional`, and the character '4' in `letter`. It is important that the user enters values in the correct order.

Checkpoint

3.1 What header file must be included in programs using `cin`?

3.2 TRUE or FALSE: `cin` requires the user to press the [Enter] key when finished entering data.

3.3 Assume `value` is an integer variable. If the user enters 3.14 in response to the following programming statement, what will be stored in `value`?

```
cin >> value;
```

A) 3.14
B) 3
C) 0
D) Nothing. An error message is displayed.

3.4 A program has the following variable definitions.

```
long miles;
int feet;
float inches;
```

Write one `cin` statement that reads a value into each of these variables.

3.5 The following program will run, but the user will have difficulty understanding what to do. How would you improve the program?

```
// This program multiplies two numbers and displays the result.
#include <iostream>
using namespace std;

int main()
{
   double first, second, product;

   cin >> first >> second;
   product = first * second;
   cout << product;
   return 0;
}
```

3.6 Complete the following program skeleton so it asks for the user's weight (in pounds) and displays the equivalent weight in kilograms.

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

int main()
{
   double pounds, kilograms;

   // Write code here that prompts the user
   // to enter his or her weight and reads
   // the input into the pounds variable.

   // The following line does the conversion.
   kilograms = pounds / 2.2;

   // Write code here that displays the user's weight
   // in kilograms.
   return 0;
}
```

3.2 Mathematical Expressions

CONCEPT: **C++ allows you to construct complex mathematical expressions using multiple operators and grouping symbols.**

In Chapter 2 you were introduced to the basic mathematical operators, which are used to build mathematical expressions. An *expression* is a programming statement that has a value. Usually, an expression consists of an operator and its operands. Look at the following statement:

```
sum = 21 + 3;
```

Since 21 + 3 has a value, it is an expression. Its value, 24, is stored in the variable `sum`. Expressions do not have to be in the form of mathematical operations. In the following statement, 3 is an expression.

```
number = 3;
```

Here are some programming statements where the variable `result` is being assigned the value of an expression:

```
result = x;
result = 4;
result = 15 / 3;
result = 22 * number;
result = sizeof(int);
result = a + b + c;
```

In each of these statements, a number, variable name, or mathematical expression appears on the right side of the = symbol. A value is obtained from each of these and stored in the variable `result`. These are all examples of a variable being assigned the value of an expression.

Program 3-4 shows how mathematical expressions can be used with the `cout` object.

Program 3-4

```
1  // This program asks the user to enter the numerator
2  // and denominator of a fraction and it displays the
3  // decimal value.
4
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     double numerator, denominator;
11
12     cout << "This program shows the decimal value of ";
13     cout << "a fraction.\n";
```

(program continues)

Program 3-4 *(continued)*

```
14       cout << "Enter the numerator: ";
15       cin >> numerator;
16       cout << "Enter the denominator: ";
17       cin >> denominator;
18       cout << "The decimal value is ";
19       cout << (numerator / denominator) << endl;
20       return 0;
21   }
```

Program Output with Example Input Shown in Bold

```
This program shows the decimal value of a fraction.
Enter the numerator: 3 [Enter]
Enter the denominator: 16 [Enter]
The decimal value is 0.1875
```

The `cout` object will display the value of any legal expression in C++. In Program 3-4, the value of the expression `numerator / denominator` is displayed.

NOTE: The example input for Program 3-4 shows the user entering 3 and 16. Since these values are assigned to `double` variables, they are stored as the `double` values 3.0 and 16.0.

NOTE: When sending an expression that consists of an operator to `cout`, it is always a good idea to put parentheses around the expression. Some advanced operators will yield unexpected results otherwise.

Operator Precedence

It is possible to build mathematical expressions with several operators. The following statement assigns the sum of 17, `x`, 21, and `y` to the variable `answer`.

```
answer = 17 + x + 21 + y;
```

Some expressions are not that straightforward, however. Consider the following statement:

```
outcome = 12 + 6 / 3;
```

What value will be stored in `outcome`? 6 is used as an operand for both the addition and division operators. `outcome` could be assigned either 6 or 14, depending on whether the addition operation or the division operation takes place first. The answer is 14 because the division operator has higher *precedence* than the addition operator.

Mathematical expressions are evaluated from left to right. When two operators share an operand, the operator with the highest precedence works first. Multiplication and division have higher precedence than addition and subtraction, so the statement above works like this:

A) 6 is divided by 3, yielding a result of 2
B) 12 is added to 2, yielding a result of 14

It could be diagrammed in the following way:

```
outcome = 12 + 6 / 3
                \ /
outcome = 12 +   2

outcome = 14
```

Table 3-1 shows the precedence of the arithmetic operators. The operators at the top of the table have higher precedence than the ones below them.

Table 3-1 Precedence of Arithmetic Operators (Highest to Lowest)

(unary negation) -
* / %
+ -

The multiplication, division, and modulus operators have the same precedence. This is also true of the addition and subtraction operators. Table 3-2 shows some expressions with their values.

Table 3-2 Some Simple Expressions and Their Values

Expression	Value
5 + 2 * 4	13
10 / 2 - 3	2
8 + 12 * 2 - 4	28
4 + 17 % 2 - 1	4
6 - 3 * 2 + 7 - 1	6

Associativity

An operator's *associativity* is either left to right, or right to left. If two operators sharing an operand have the same precedence, they work according to their associativity. Table 3-3 lists the associativity of the arithmetic operators. As an example, look at the following expression:

```
5 - 3 + 2
```

Both the - and + operators in this expression have the same precedence, and they have left to right associativity. So, the operators will work from left to right. This expression is the same as:

```
((5 - 3) + 2)
```

Here is another example:

```
12 / 6 * 4
```

Because the / and * operators have the same precedence, and they have left to right associativity, they will work from left to right. This expression is the same as:

```
((12 / 6) * 4)
```

Table 3-3 Associativity of Arithmetic Operators

Operator	Associativity
(unary negation) -	Right to left
* / %	Left to right
+ -	Left to right

Grouping with Parentheses

Parts of a mathematical expression may be grouped with parentheses to force some operations to be performed before others. In the following statement, the sum of `a + b` is divided by 4.

```
result = (a + b) / 4;
```

Without the parentheses, however, `b` would be divided by 4 and the result added to `a`. Table 3-4 shows more expressions and their values.

Table 3-4 More Simple Expressions and Their Values

Expression	Value
`(5 + 2) * 4`	`28`
`10 / (5 - 3)`	`5`
`8 + 12 * (6 - 2)`	`56`
`(4 + 17) % 2 - 1`	`0`
`(6 - 3) * (2 + 7) / 3`	`9`

Converting Algebraic Expressions to Programming Statements

In algebra it is not always necessary to use an operator for multiplication. C++, however, requires an operator for any mathematical operation. Table 3-5 shows some algebraic expressions that perform multiplication and the equivalent C++ expressions.

Table 3-5 Algebraic and C++ Multiplication Expressions

Algebraic Expression	Operation	C++ Equivalent
$6B$	`6 times B`	`6 * B`
$(3)(12)$	`3 times 12`	`3 * 12`
$4xy$	`4 times x times y`	`4 * x * y`

When converting some algebraic expressions to C++, you may have to insert parentheses that do not appear in the algebraic expression. For example, look at the following expression:

$$x = \frac{a + b}{c}$$

To convert this to a C++ statement, $a + b$ will have to be enclosed in parentheses:

```
x = (a + b) / c;
```

Table 3-6 shows more algebraic expressions and their C++ equivalents.

Table 3-6 Algebraic and C++ Expressions

Algebraic Expression	C++ Expression
$y = 3\frac{x}{2}$	`y = x / 2 * 3;`
$z = 3bc + 4$	`z = 3 * b * c + 4;`
$a = \frac{3x + 2}{4a - 1}$	`a = (3 * x + 2) / (4 * a - 1)`

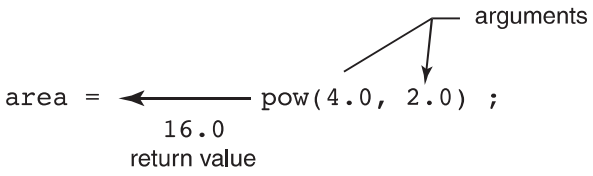
No Exponents Please!

Unlike many programming languages, C++ does not have an exponent operator. Raising a number to a power requires the use of a *library function*. The C++ library isn't a place where you check out books, but a collection of specialized functions. Think of a library function as a "routine" that performs a specific operation. One of the library functions is called `pow`, and its purpose is to raise a number to a power. Here is an example of how it's used:

```
area = pow(4.0, 2.0);
```

This statement contains a *call* to the `pow` function. The numbers inside the parentheses are *arguments*. Arguments are data being sent to the function. The `pow` function always raises the first argument to the power of the second argument. In this example, 4 is raised to the power of 2. The result is *returned* from the function and used in the statement where the function call appears. In this case, the value 16 is returned from `pow` and assigned to the variable `area`. This is illustrated in Figure 3-3.

Figure 3-3



The statement `area = pow(4.0, 2.0)` is equivalent to the following algebraic statement:

$$\text{area} = 4^2$$

Here is another example of a statement using the `pow` function. It assigns 3 times 6^3 to `x`:

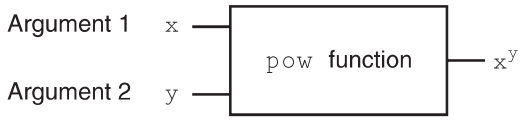
```
x = 3 * pow(6.0, 3.0);
```

And the following statement displays the value of 5 raised to the power of 4:

```
cout << pow(5.0, 4.0);
```

It might be helpful to think of `pow` as a "black box" that you plug two numbers into, and that then sends a third number out. The number that comes out has the value of the first number raised to the power of the second number, as illustrated in Figure 3-4:

Figure 3-4



There are some guidelines that should be followed when the pow function is used. First, the program must include the cmath header file. Second, the arguments that you pass to the pow function should be doubles. Third, the variable used to store pow's return value should be defined as a double. For example, in the following statement the variable area should be a double:

```
area = pow(4.0, 2.0);
```

Program 3-5 solves a simple algebraic problem. It asks the user to enter the radius of a circle and then calculates the area of the circle. The formula is

$$\text{Area} = \pi r^2$$

which is expressed in the program as

```
area = PI * pow(radius, 2.0);
```

Program 3-5

```
1  // This program calculates the area of a circle.
2  // The formula for the area of a circle is Pi times
3  // the radius squared. Pi is 3.14159.
4  #include <iostream>
5  #include <cmath> // needed for pow function
6  using namespace std;
7
8  int main()
9  {
10     const double PI = 3.14159;
11     double area, radius;
12
13     cout << "This program calculates the area of a circle.\n";
14     cout << "What is the radius of the circle? ";
15     cin >> radius;
16     area = PI * pow(radius, 2.0);
17     cout << "The area is " << area << endl;
18     return 0;
19 }
```

Program Output with Example Input Shown in Bold

```
This program calculates the area of a circle.
What is the radius of the circle? 10 [Enter]
The area is 314.159
```

NOTE: Program 3-5 is presented as a demonstration of the pow function. In reality, there is no reason to use the pow function in such a simple operation. The math statement could just as easily be written as

```
area = PI * radius * radius;
```

The pow function is useful, however, in operations that involve larger exponents.

In the Spotlight:

Calculating an Average

Determining the average of a group of values is a simple calculation: You add all of the values and then divide the sum by the number of values. Although this is a straightforward calculation, it is easy to make a mistake when writing a program that calculates an average. For example, let's assume that `a`, `b`, and `c` are `double` variables. Each of the variables holds a value, and we want to calculate the average of those values. If we are careless, we might write a statement such as the following to perform the calculation:

```
average = a + b + c / 3.0;
```

Can you see the error in this statement? When it executes, the division will take place first. The value in `c` will be divided by `3.0`, and then the result will be added to the sum of `a + b`. That is not the correct way to calculate an average. To correct this error we need to put parentheses around `a + b + c`, as shown here:

```
average = (a + b + c) / 3.0;
```

Let's step through the process of writing a program that calculates an average. Suppose you have taken three tests in your computer science class, and you want to write a program that will display the average of the test scores. Here is the algorithm in pseudocode:

Get the first test score.
Get the second test score.
Get the third test score.
Calculate the average by adding the three test scores and dividing the sum by 3.
Display the average.

In the first three steps we prompt the user to enter three test scores. Let's say we store those test scores in the `double` variables `test1`, `test2`, and `test3`. Then in the fourth step we calculate the average of the three test scores. We will use the following statement to perform the calculation and store the result in the `average` variable, which is a `double`:

```
average = (test1 + test2 + test3) / 3.0;
```

The last step is to display the average. Program 3-6 shows the program.

Program 3-6

```
1  // This program calculates the average
2  // of three test scores.
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  int main()
8  {
9       double test1, test2, test3;   // To hold the scores
10      double average;               // To hold the average
11
12      // Get the three test scores.
13      cout << "Enter the first test score: ";
14      cin >> test1;
15      cout << "Enter the second test score: ";
16      cin >> test2;
17      cout << "Enter the third test score: ";
18      cin >> test3;
19
20      // Calculate the average of the scores.
21      average = (test1 + test2 + test3) / 3.0;
22
23      // Display the average.
24      cout << "The average score is: " << average << endl;
25      return 0;
26 }
```

Program Output with Example Input Shown in Bold

```
Enter the first test score: 90 [Enter]
Enter the second test score: 80 [Enter]
Enter the third test score: 100 [Enter]
The average score is 90
```

Checkpoint

3.7 Complete the table below by writing the value of each expression in the "Value" column.

Expression	Value
`6 + 3 * 5`	
`12 / 2 - 4`	
`9 + 14 * 2 - 6`	
`5 + 19 % 3 - 1`	
`(6 + 2) * 3`	
`14 / (11 - 4)`	
`9 + 12 * (8 - 3)`	
`(6 + 17) % 2 - 1`	
`(9 - 3) * (6 + 9) / 3`	

3.8 Write C++ expressions for the following algebraic expressions:

$y = 6x$

$a = 2b + 4c$

$y = x^2$

$g = \dfrac{x + 2}{z^2}$

$y = \dfrac{x^2}{z^2}$

3.9 Study the following program and complete the table.

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

int main()
{
     double value1, value2, value3;

     cout << "Enter a number: ";
     cin >> value1;
     value2 = 2 * pow(value1, 2.0);
     value3 = 3 + value2 / 2 - 1;
     cout << value3 << endl;
     return 0;
}
```

If the User Enters...	The Program Will Display What Number (Stored in **value3**)?
2	
5	
4.3	
6	

3.10 Complete the following program skeleton so it displays the volume of a cylindrical fuel tank. The formula for the volume of a cylinder is

Volume = $\pi r^2 h$

where

π is 3.14159
r is the radius of the tank
h is the height of the tank

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

```
int main()
{
     double volume, radius, height;

     cout << "This program will tell you the volume of\n";
     cout << "a cylinder-shaped fuel tank.\n";
     cout << "How tall is the tank? ";
     cin >> height;
     cout << "What is the radius of the tank? ";
     cin >> radius;

     // You must complete the program.
}
```

3.3 When You Mix Apples and Oranges: Type Conversion

CONCEPT: When an operator's operands are of different data types, C++ will automatically convert them to the same data type. This can affect the results of mathematical expressions.

If an `int` is multiplied by a `float`, what data type will the result be? What if a `double` is divided by an `unsigned int`? Is there any way of predicting what will happen in these instances? The answer is yes. C++ follows a set of rules when performing mathematical operations on variables of different data types. It's helpful to understand these rules to prevent subtle errors from creeping into your programs.

Just like officers in the military, data types are ranked. One data type outranks another if it can hold a larger number. For example, a `float` outranks an `int`. Table 3-7 lists the data types in order of their rank, from highest to lowest.

Table 3-7 Data Type Ranking

`long double`
`double`
`float`
`unsigned long`
`long`
`unsigned int`
`int`

One exception to the ranking in Table 3-7 is when an `int` and a `long` are the same size. In that case, an `unsigned int` outranks `long` because it can hold a higher value.

When C++ is working with an operator, it strives to convert the operands to the same type. This automatic conversion is known as *type coercion*. When a value is converted to a higher data type, it is said to be *promoted*. To *demote* a value means to convert it to a lower data type. Let's look at the specific rules that govern the evaluation of mathematical expressions.

Rule 1: `chars`, `shorts`, and `unsigned shorts` are automatically promoted to `int`.

You will notice that `char`, `short`, and `unsigned short` do not appear in Table 3-7. That's because anytime they are used in a mathematical expression, they are automatically promoted to an `int`. The only exception to this rule is when an `unsigned short` holds a value larger than can be held by an `int`. This can happen on systems where `shorts` are the same size as `ints`. In this case, the `unsigned short` is promoted to `unsigned int`.

Rule 2: When an operator works with two values of different data types, the lower-ranking value is promoted to the type of the higher-ranking value.

In the following expression, assume that `years` is an `int` and `interestRate` is a `float`:

```
years * interestRate
```

Before the multiplication takes place, `years` will be promoted to a `float`.

Rule 3: When the final value of an expression is assigned to a variable, it will be converted to the data type of that variable.

In the following statement, assume that `area` is a `long int`, while `length` and `width` are both `ints`:

```
area = length * width;
```

Since `length` and `width` are both `ints`, they will not be converted to any other data type. The result of the multiplication, however, will be converted to `long` so it can be stored in `area`.

Watch out for situations where an expression results in a fractional value being assigned to an integer variable. Here is an example:

```
int x, y = 4;
float z = 2.7;
x = y * z;
```

In the expression `y * z`, `y` will be promoted to `float` and 10.8 will result from the multiplication. Since `x` is an integer, however, 10.8 will be truncated and 10 will be stored in `x`.

Integer Division

When you divide an integer by another integer in C++, the result is always an integer. If there is a remainder, it will be discarded. For example, in the following code, `parts` is assigned the value 2.0:

```
double parts;
parts = 15 / 6;
```

Even though 15 divided by 6 is really 2.5, the .5 part of the result is discarded because we are dividing an integer by an integer. It doesn't matter that `parts` is declared as a `double` because the fractional part of the result is discarded *before* the assignment takes place. In order for a division operation to return a floating-point value, at least one of the operands must be of a floating-point data type. For example, the previous code could be written as:

```
double parts;
parts = 15.0 / 6;
```

In this code the literal value 15.0 is interpreted as a floating-point number, so the division operation will return a floating-point number. The value 2.5 will be assigned to `parts`.

3.4 Overflow and Underflow

CONCEPT: **When a variable is assigned a value that is too large or too small in range for that variable's data type, the variable overflows or underflows.**

Trouble can arise when a variable is being assigned a value that is too large for its type. Here is a statement where `a`, `b`, and `c` are all short integers:

```
a = b * c;
```

If `b` and `c` are set to values large enough, the multiplication will produce a number too big to be stored in `a`. To prepare for this, `a` should have been defined as an `int`, or a `long int`.

When a variable is assigned a number that is too large for its data type, it *overflows*. Likewise, assigning a value that is too small for a variable causes it to *underflow*. Program 3-7 shows what happens when an integer overflows or underflows. (The output shown is from a system with two-byte short integers.)

Program 3-7

```
1  // This program demonstrates integer overflow and underflow.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       // testVar is initialized with the maximum value for a short.
8       short testVar = 32767;
9
10      // Display testVar.
11      cout << testVar << endl;
12
13      // Add 1 to testVar to make it overflow.
14      testVar = testVar + 1;
15      cout << testVar << endl;
16
17      // Subtract 1 from testVar to make it underflow.
18      testVar = testVar - 1;
19      cout << testVar << endl;
20      return 0;
21 }
```

Program Output

```
32767
-32768
32767
```

Typically, when an integer overflows, its contents wrap around to that data type's lowest possible value. In Program 3-7, `testVar` wrapped around from 32,767 to –32,768 when 1 was added to it. When 1 was subtracted from `testVar`, it underflowed, which caused its

contents to wrap back around to 32,767. No warning or error message is given, so be careful when working with numbers close to the maximum or minimum range of an integer. If an overflow or underflow occurs, the program will use the incorrect number and therefore produce incorrect results.

When floating-point variables overflow or underflow, the results depend upon how the compiler is configured. Your system may produce programs that do any of the following:

- Produces an incorrect result and continues running.
- Prints an error message and immediately stops when either floating point overflow or underflow occurs.
- Prints an error message and immediately stops when floating point overflow occurs, but stores a 0 in the variable when it underflows.
- Gives you a choice of behaviors when overflow or underflow occurs.

You can find out how your system reacts by compiling and running Program 3-8.

Program 3-8

```
1  // This program can be used to see how your system handles
2  // floating point overflow and underflow.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       float test;
9
10      test = 2.0e38 * 1000;     // Should overflow test.
11      cout << test << endl;
12      test = 2.0e-38 / 2.0e38;  // Should underflow test.
13      cout << test << endl;
14      return 0;
15 }
```

3.5 Type Casting

CONCEPT: Type casting allows you to perform manual data type conversion.

A *type cast expression* lets you manually promote or demote a value. The general format of a type cast expression is

```
static_cast<DataType>(Value)
```

where *Value* is a variable or literal value that you wish to convert and *DataType* is the data type you wish to convert *Value* to. Here is an example of code that uses a type cast expression:

```
double number = 3.7;
int val;
val = static_cast<int>(number);
```

This code defines two variables: `number`, a `double`, and `val`, an `int`. The type cast expression in the third statement returns a copy of the value in `number`, converted to an `int`. When a `double` is converted to an `int`, the fractional part is truncated so this statement stores 3 in `val`. The original value in `number` is not changed, however.

Type cast expressions are useful in situations where C++ will not perform the desired conversion automatically. Program 3-9 shows an example where a type cast expression is used to prevent integer division from taking place. The statement that uses the type cast expression is

```
perMonth = static_cast<double>(books) / months;
```

Program 3-9

```
1  // This program uses a type cast to avoid integer division.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       int books;          // Number of books to read
8       int months;         // Number of months spent reading
9       double perMonth;    // Average number of books per month
10
11      cout << "How many books do you plan to read? ";
12      cin >> books;
13      cout << "How many months will it take you to read them? ";
14      cin >> months;
15      perMonth = static_cast<double>(books) / months;
16      cout << "That is " << perMonth << " books per month.\n";
17      return 0;
18 }
```

Program Output with Example Input Shown in Bold

```
How many books do you plan to read? 30 [Enter]
How many months will it take you to read them? 7 [Enter]
That is 4.28571 books per month.
```

The variable `books` is an integer, but its value is converted to a `double` before the division takes place. Without the type cast expression in line 15, integer division would have been performed resulting in an incorrect answer.

WARNING! In Program 3-9, the following statement would still have resulted in integer division:

```
perMonth = static_cast<double>(books / months);
```

The result of the expression `books / months` is 4. When 4 is converted to a `double`, it is 4.0. To prevent the integer division from taking place, one of the operands should be converted to a `double` prior to the division operation. This forces C++ to automatically convert the value of the other operand to a `double`.

Program 3-10 further demonstrates the type cast expression.

Program 3-10

```
1  // This program uses a type cast expression to print a character
2  // from a number.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       int number = 65;
9
10      // Display the value of the number variable.
11      cout << number << endl;
12
13      // Display the value of number converted to
14      // the char data type.
15      cout << static_cast<char>(number) << endl;
16      return 0;
17 }
```

Program Output

```
65
A
```

Let's take a closer look at this program. In line 8 the `int` variable `number` is initialized with the value 65. In line 11, `number` is sent to `cout`, causing 65 to be displayed. In line 15, a type cast expression is used to convert the value in `number` to the `char` data type. Recall from Chapter 2 that characters are stored in memory as integer ASCII codes. The number 65 is the ASCII code for the letter 'A', so this statement causes the letter 'A' to be displayed.

NOTE: C++ provides several different type cast expressions. `static_cast` is the most commonly used type cast expression, so we will primarily use it in this book.

Checkpoint

3.11 Assume the following variable definitions:

```
int a = 5, b = 12;
double x = 3.4, z = 9.1;
```

What are the values of the following expressions?

A) `b / a`
B) `x * a`
C) `static_cast<double>(b / a)`
D) `static_cast<double>(b) / a`
E) `b / static_cast<double>(a)`
F) `static_cast<double>(b) / static_cast<double>(a)`
G) `b / static_cast<int>(x)`
H) `static_cast<int>(x) * static_cast<int>(z)`
I) `static_cast<int>(x * z)`
J) `static_cast<double>(static_cast<int>(x) * static_cast<int>(z))`

3.12 Complete the following program skeleton so it asks the user to enter a character. Store the character in the variable letter. Use a type cast expression with the variable in a cout statement to display the character's ASCII code on the screen.

```
#include <iostream>
using namespace std;
int main()
{
     char letter;

     // Finish this program
     // as specified above.
     return 0;
}
```

3.13 What will the following program display?

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

int main()
{
     int integer1, integer2;
     double result;
     integer1 = 19;
     integer2 = 2;
     result = integer1 / integer2;
     cout << result << endl;
     result = static_cast<double>(integer1) / integer2;
     cout << result << endl;
     result = static_cast<double>(integer1 / integer2);
     cout << result << endl;
     return 0;
}
```

3.6 Multiple Assignment and Combined Assignment

CONCEPT: **Multiple assignment means to assign the same value to several variables with one statement.**

C++ allows you to assign a value to multiple variables at once. If a program has several variables, such as a, b, c, and d, and each variable needs to be assigned a value, such as 12, the following statement may be constructed:

```
a = b = c = d = 12;
```

The value 12 will be assigned to each variable listed in the statement.*

* The assignment operator works from right to left. 12 is first assigned to d, then to c, then to b, then to a.

Combined Assignment Operators

Quite often, programs have assignment statements of the following form:

```
number = number + 1;
```

The expression on the right side of the assignment operator gives the value of `number` plus 1. The result is then assigned to `number`, replacing the value that was previously stored there. Effectively, this statement adds 1 to `number`. In a similar fashion, the following statement subtracts 5 from `number`.

```
number = number - 5;
```

If you have never seen this type of statement before, it might cause some initial confusion because the same variable name appears on both sides of the assignment operator. Table 3-8 shows other examples of statements written this way.

Table 3-8 (Assume x = 6)

Statement	What It Does	Value of x After the Statement
`x = x + 4;`	Adds 4 to x	10
`x = x - 3;`	Subtracts 3 from x	3
`x = x * 10;`	Multiplies x by 10	60
`x = x / 2;`	Divides x by 2	3
`x = x % 4`	Makes x the remainder of x / 4	2

These types of operations are very common in programming. For convenience, C++ offers a special set of operators designed specifically for these jobs. Table 3-9 shows the *combined assignment operators*, also known as *compound operators*, and *arithmetic assignment operators*.

Table 3-9

Operator	Example Usage	Equivalent to
`+=`	`x += 5;`	`x = x + 5;`
`-=`	`y -= 2;`	`y = y - 2;`
`*=`	`z *= 10;`	`z = z * 10;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`c %= 3;`	`c = c % 3;`

As you can see, the combined assignment operators do not require the programmer to type the variable name twice. Also, they give a clear indication of what is happening in the statement. Program 3-11 uses combined assignment operators.

Program 3-11

```
// This program tracks the inventory of three widget stores
// that opened at the same time. Each store started with the
// same number of widgets in inventory. By subtracting the
// number of widgets each store has sold from its inventory,
// the current inventory can be calculated.
#include <iostream>
using namespace std;

int main()
{
   int begInv,    // Beginning inventory for all stores
       sold,      // Number of widgets sold
       store1,    // Store 1's inventory
       store2,    // Store 2's inventory
       store3;    // Store 3's inventory

   // Get the beginning inventory for all the stores.
   cout << "One week ago, 3 new widget stores opened\n";
   cout << "at the same time with the same beginning\n";
   cout << "inventory. What was the beginning inventory? ";
   cin >> begInv;

   // Set each store's inventory.
   store1 = store2 = store3 = begInv;

   // Get the number of widgets sold at store 1.
   cout << "How many widgets has store 1 sold? ";
   cin >> sold;
   store1 -= sold; // Adjust store 1's inventory.

   // Get the number of widgets sold at store 2.
   cout << "How many widgets has store 2 sold? ";
   cin >> sold;
   store2 -= sold; // Adjust store 2's inventory.

   // Get the number of widgets sold at store 3.
   cout << "How many widgets has store 3 sold? ";
   cin >> sold;
   store3 -= sold; // Adjust store 3's inventory.

   // Display each store's current inventory.
   cout << "\nThe current inventory of each store:\n";
   cout << "Store 1: " << store1 << endl;
   cout << "Store 2: " << store2 << endl;
   cout << "Store 3: " << store3 << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
One week ago, 3 new widget stores opened
at the same time with the same beginning
inventory. What was the beginning inventory? 100 [Enter]
How many widgets has store 1 sold? 25 [Enter]
How many widgets has store 2 sold? 15 [Enter]
How many widgets has store 3 sold? 45 [Enter]

The current inventory of each store:
Store 1: 75
Store 2: 85
Store 3: 55
```

More elaborate statements may be expressed with the combined assignment operators. Here is an example:

```
result *= a + 5;
```

In this statement, `result` is multiplied by the sum of `a + 5`. When constructing such statements, you must realize the precedence of the combined assignment operators is lower than that of the regular math operators. The statement above is equivalent to

```
result = result * (a + 5);
```

which is different from

```
result = result * a + 5;
```

Table 3-10 shows other examples of such statements and their assignment statement equivalencies.

Table 3-10

Example Usage	Equivalent to
`x += b + 5;`	`x = x + (b + 5);`
`y -= a * 2;`	`y = y - (a * 2);`
`z *= 10 - c;`	`z = z * (10 - c);`
`a /= b + c;`	`a = a / (b + c);`
`c %= d - 3;`	`c = c % (d - 3);`

Checkpoint

3.14 Write a multiple assignment statement that assigns 0 to the variables `total`, `subtotal`, `tax`, and `shipping`.

3.15 Write statements using combined assignment operators to perform the following:

A) Add 6 to `x`.
B) Subtract 4 from `amount`.
C) Multiply `y` by 4.
D) Divide `total` by 27.
E) Store in `x` the remainder of `x` divided by 7.
F) Add `y * 5` to `x`.
G) Subtract `discount` times 4 from `total`.
H) Multiply `increase` by `salesRep` times 5.
I) Divide `profit` by `shares` minus 1000.

3.16 What will the following program display?

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

int main()
{
     int unus, duo, tres;

     unus = duo = tres = 5;
     unus += 4;
     duo *= 2;
     tres -= 4;
     unus /= 3;
     duo += tres;
     cout << unus << endl;
     cout << duo << endl;
     cout << tres << endl;
     return 0;
}
```

3.7 Formatting Output

CONCEPT: The `cout` object provides ways to format data as it is being displayed. This affects the way data appears on the screen.

The same data can be printed or displayed in several different ways. For example, all of the following numbers have the same value, although they look different:

```
720
720.0
720.00000000
7.2e+2
+720.0
```

The way a value is printed is called its *formatting*. The `cout` object has a standard way of formatting variables of each data type. Sometimes, however, you need more control over the way data is displayed. Consider Program 3-12, for example, which displays three rows of numbers with spaces between each one.

Program 3-12

```
1  // This program displays three rows of numbers.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       int num1 = 2897, num2 = 5,     num3 = 837,
```

```
 8           num4 = 34,   num5 = 7,    num6 = 1623,
 9           num7 = 390,  num8 = 3456, num9 = 12;
10
11       // Display the first row of numbers
12       cout << num1 << " " << num2 << " " << num3 << endl;
13
14       // Display the second row of numbers
15       cout << num4 << " " << num5 << " " << num6 << endl;
16
17       // Display the third row of numbers
18       cout << num7 << " " << num8 << " " << num9 << endl;
19       return 0;
20   }
```

Program Output

```
2897    5  837
  34    7 1623
 390 3456   12
```

Unfortunately, the numbers do not line up in columns. This is because some of the numbers, such as 5 and 7, occupy one position on the screen, while others occupy two or three positions. `cout` uses just the number of spaces needed to print each number.

To remedy this, `cout` offers a way of specifying the minimum number of spaces to use for each number. A stream manipulator, `setw`, can be used to establish print fields of a specified width. Here is an example of how it is used:

```
value = 23;
cout << setw(5) << value;
```

The number inside the parentheses after the word `setw` specifies the *field width* for the value immediately following it. The field width is the minimum number of character positions, or spaces, on the screen to print the value in. In the example above, the number 23 will be displayed in a field of 5 spaces. Since 23 only occupies 2 positions on the screen, 3 blank spaces will be printed before it. To further clarify how this works, look at the following statements:

```
value = 23;
cout << "(" << setw(5) << value << ")";
```

This will cause the following output:

```
(   23)
```

Notice that the number occupies the last two positions in the field. Since the number did not use the entire field, `cout` filled the extra 3 positions with blank spaces. Because the number appears on the right side of the field with blank spaces "padding" it in front, it is said to be *right-justified*.

Program 3-13 shows how the numbers in Program 3-12 can be printed in columns that line up perfectly by using `setw`.

Program 3-13

```
1  // This program displays three rows of numbers.
2  #include <iostream>
3  #include <iomanip>     // Required for setw
4  using namespace std;
5
6  int main()
7  {
8       int num1 = 2897, num2 = 5,     num3 = 837,
9           num4 = 34,   num5 = 7,     num6 = 1623,
10          num7 = 390,  num8 = 3456,  num9 = 12;
11
12      // Display the first row of numbers
13      cout << setw(6) << num1 << setw(6)
14           << num2 << setw(6) << num3 << endl;
15
16      // Display the second row of numbers
17      cout << setw(6) << num4 << setw(6)
18           << num5 << setw(6) << num6 << endl;
19
20      // Display the third row of numbers
21      cout << setw(6) << num7 << setw(6)
22           << num8 << setw(6) << num9 << endl;
23      return 0;
24 }
```

Program Output

```
 2897     5   837
   34     7  1623
  390  3456    12
```

By printing each number in a field of 6 positions, they are displayed in perfect columns.

NOTE: A new header file, `iomanip`, is included in Program 3-13. It must be used in any program that uses `setw`.

Notice how a `setw` manipulator is used with each value because `setw` only establishes a field width for the value immediately following it. After that value is printed, `cout` goes back to its default method of printing.

You might wonder what will happen if the number is too large to fit in the field, as in the following statement:

```
value = 18397;
cout << setw(2) << value;
```

In cases like this, `cout` will print the entire number. `setw` only specifies the minimum number of positions in the print field. Any number larger than the minimum will cause `cout` to override the `setw` value.

You may specify the field width of any type of data. Program 3-14 shows `setw` being used with an integer, a floating-point number, and a `string` object.

Program 3-14

```
1  // This program demonstrates the setw manipulator being
2  // used with values of various data types.
3  #include <iostream>
4  #include <iomanip>
5  #include <string>
6  using namespace std;
7
8  int main()
9  {
10     int intValue = 3928;
11     double doubleValue = 91.5;
12     string stringValue = "John J. Smith";
13
14     cout << "(" << setw(5) << intValue << ")" << endl;
15     cout << "(" << setw(8) << doubleValue << ")" << endl;
16     cout << "(" << setw(16) << stringValue << ")" << endl;
17     return 0;
18 }
```

Program Output

```
( 3928)
(    91.5)
(    John J. Smith)
```

Program 3-14 can be used to illustrate the following points:

- The field width of a floating-point number includes a position for the decimal point.
- The field width of a `string` object includes all characters in the string, including spaces.
- The values printed in the field are right-justified by default. This means they are aligned with the right side of the print field, and any blanks that must be used to pad it are inserted in front of the value.

VideoNote
Formatting Numbers with `setprecision`

The `setprecision` Manipulator

Floating-point values may be rounded to a number of *significant digits*, or *precision*, which is the total number of digits that appear before and after the decimal point. You can control the number of significant digits with which floating-point values are displayed by using the `setprecision` manipulator. Program 3-15 shows the results of a division operation displayed with different numbers of significant digits.

Program 3-15

```
1  // This program demonstrates how setprecision rounds a
2  // floating point value.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9       double quotient, number1 = 132.364, number2 = 26.91;
10
11      quotient = number1 / number2;
12      cout << quotient << endl;
13      cout << setprecision(5) << quotient << endl;
14      cout << setprecision(4) << quotient << endl;
15      cout << setprecision(3) << quotient << endl;
16      cout << setprecision(2) << quotient << endl;
17      cout << setprecision(1) << quotient << endl;
18      return 0;
19 }
```

Program Output

```
4.91877
4.9188
4.919
4.92
4.9
5
```

The first value is displayed in line 12 without the `setprecision` manipulator. (By default, the system in the illustration displays floating-point values with 6 significant digits.) The subsequent `cout` statements print the same value, but rounded to 5, 4, 3, 2, and 1 significant digits.

If the value of a number is expressed in fewer digits of precision than specified by `setprecision`, the manipulator will have no effect. In the following statements, the value of `dollars` only has four digits of precision, so the number printed by both `cout` statements is 24.51.

```
double dollars = 24.51;
cout << dollars << endl;                        // Displays 24.51
cout << setprecision(5) << dollars << endl;     // Displays 24.51
```

Table 3-11 shows how `setprecision` affects the way various values are displayed.

Table 3-11

Number	Manipulator	Value Displayed
28.92786	setprecision(3)	28.9
21	setprecision(5)	21
109.5	setprecision(4)	109.5
34.28596	setprecision(2)	34

Unlike field width, the precision setting remains in effect until it is changed to some other value. As with all formatting manipulators, you must include the header file `iomanip` to use `setprecision`.

Program 3-16 shows how the `setw` and `setprecision` manipulators may be combined to fully control the way floating point numbers are displayed.

Program 3-16

```
1  // This program asks for sales figures for 3 days. The total
2  // sales are calculated and displayed in a table.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9       double day1, day2, day3, total;
10
11      // Get the sales for each day.
12      cout << "Enter the sales for day 1: ";
13      cin >> day1;
14      cout << "Enter the sales for day 2: ";
15      cin >> day2;
16      cout << "Enter the sales for day 3: ";
17      cin >> day3;
18
19      // Calculate the total sales.
20      total = day1 + day2 + day3;
21
22      // Display the sales figures.
23      cout << "\nSales Figures\n";
24      cout << "-------------\n";
25      cout << setprecision(5);
26      cout << "Day 1: " << setw(8) << day1 << endl;
27      cout << "Day 2: " << setw(8) << day2 << endl;
28      cout << "Day 3: " << setw(8) << day3 << endl;
29      cout << "Total: " << setw(8) << total << endl;
30      return 0;
31 }
```

Program Output with Example Input Shown in Bold

```
Enter the sales for day 1: 321.57 [Enter]
Enter the sales for day 2: 269.62 [Enter]
Enter the sales for day 3: 307.77 [Enter]

Sales Figures
-------------
Day 1: 321.57
Day 2: 269.62
Day 3: 307.77
Total: 898.96
```

The `fixed` Manipulator

The `setprecision` manipulator can sometimes surprise you in an undesirable way. When the precision of a number is set to a lower value, numbers tend to be printed in scientific notation. For example, here is the output of Program 3-16 with larger numbers being input:

Program 3-16

Program Output with Example Input Shown in Bold

```
Enter the sales for day 1: 145678.99 [Enter]
Enter the sales for day 2: 205614.85 [Enter]
Enter the sales for day 3: 198645.22 [Enter]

Sales Figures
-------------
Day 1: 1.4568e+005
Day 2: 2.0561e+005
Day 3: 1.9865e+005
Total: 5.4994e+005
```

Another stream manipulator, `fixed`, forces `cout` to print the digits in *fixed-point notation*, or decimal. Program 3-17 shows how the `fixed` manipulator is used.

Program 3-17

```
1  // This program asks for sales figures for 3 days. The total
2  // sales are calculated and displayed in a table.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9       double day1, day2, day3, total;
10
11      // Get the sales for each day.
12      cout << "Enter the sales for day 1: ";
13      cin >> day1;
14      cout << "Enter the sales for day 2: ";
15      cin >> day2;
16      cout << "Enter the sales for day 3: ";
17      cin >> day3;
18
19      // Calculate the total sales.
20      total = day1 + day2 + day3;
21
```

```
22      // Display the sales figures.
23      cout << "\nSales Figures\n";
24      cout << "-------------\n";
25      cout << setprecision(2) << fixed;
26      cout << "Day 1: " << setw(8) << day1 << endl;
27      cout << "Day 2: " << setw(8) << day2 << endl;
28      cout << "Day 3: " << setw(8) << day3 << endl;
29      cout << "Total: " << setw(8) << total << endl;
30      return 0;
31  }
```

Program Output with Example Input Shown in Bold

```
Enter the sales for day 1: 1321.87 [Enter]
Enter the sales for day 2: 1869.26 [Enter]
Enter the sales for day 3: 1403.77 [Enter]

Sales Figures
-------------
Day 1:     1321.87
Day 2:     1869.26
Day 3:     1403.77
Total:     4594.90
```

The statement in line 25 uses the `fixed` manipulator:

```
cout << setprecision(2) << fixed;
```

When the `fixed` manipulator is used, all floating point numbers that are subsequently printed will be displayed in fixed point notation, with the number of digits to the right of the decimal point specified by the `setprecision` manipulator.

When the `fixed` and `setprecision` manipulators are used together, the value specified by the `setprecision` manipulator will be the number of digits to appear after the decimal point, not the number of significant digits. For example, look at the following code.

```
double x = 123.4567;
cout << setprecision(2) << fixed << x << endl;
```

Because the `fixed` manipulator is used, the `setprecision` manipulator will cause the number to be displayed with two digits after the decimal point. The value will be displayed as 123.46.

The `showpoint` Manipulator

By default, floating-point numbers are not displayed with trailing zeroes, and floating-point numbers that do not have a fractional part are not displayed with a decimal point. For example, look at the following code.

```
double x = 123.4, y = 456.0;
cout << setprecision(6) << x << endl;
cout << y << endl;
```

The cout statements will produce the following output.

```
123.4
456
```

Although six significant digits are specified for both numbers, neither number is displayed with trailing zeroes. If we want the numbers padded with trailing zeroes, we must use the showpoint manipulator as shown in the following code.

```
double x = 123.4, y = 456.0;
cout << setprecision(6) << showpoint << x << endl;
cout << y << endl;
```

These cout statements will produce the following output.

```
123.400
456.000
```

NOTE: With most compilers, trailing zeroes are displayed when the setprecision and fixed manipulators are used together.

The left and right Manipulators

Normally output is right justified. For example, look at the following code.

```
double x = 146.789, y = 24.2, z = 1.783;
cout << setw(10) << x << endl;
cout << setw(10) << y << endl;
cout << setw(10) << z << endl;
```

Each of the variables, x, y, and z, is displayed in a print field of 10 spaces. The output of the cout statements is

```
146.789
 24.2
  1.783
```

Notice that each value is right-justified, or aligned to the right of its print field. You can cause the values to be left-justified by using the left manipulator, as shown in the following code.

```
double x = 146.789, y = 24.2, z = 1.783;
cout << left << setw(10) << x << endl;
cout << setw(10) << y << endl;
cout << setw(10) << z << endl;
```

The output of these cout statements is

```
146.789
 24.2
  1.783
```

In this case, the numbers are aligned to the left of their print fields. The left manipulator remains in effect until you use the right manipulator, which causes all subsequent output to be right-justified.

Table 3-12 summarizes the manipulators we have discussed.

Table 3-12

Stream Manipulator	Description
`setw(n)`	Establishes a print field of *n* spaces.
`fixed`	Displays floating-point numbers in fixed point notation.
`showpoint`	Causes a decimal point and trailing zeroes to be displayed, even if there is no fractional part.
`setprecision(n)`	Sets the precision of floating-point numbers.
`left`	Causes subsequent output to be left justified.
`right`	Causes subsequent output to be right justified.

Checkpoint

3.17 Write `cout` statements with stream manipulators that perform the following:

A) Display the number 34.789 in a field of nine spaces with two decimal places of precision.

B) Display the number 7.0 in a field of five spaces with three decimal places of precision.
The decimal point and any trailing zeroes should be displayed.

C) Display the number 5.789e+12 in fixed point notation.

D) Display the number 67 left justified in a field of seven spaces.

3.18 The following program will not compile because the lines have been mixed up.

```
#include <iomanip>
}
cout << person << endl;
string person = "Wolfgang Smith";
int main()
cout << person << endl;
{
#include <iostream>
return 0;
cout << left;
using namespace std;
cout << setw(20);
cout << right;
```

When the lines are properly arranged the program should display the following:

```
    Wolfgang Smith
Wolfgang Smith
```

Rearrange the lines in the correct order. Test the program by entering it on the computer, compiling it, and running it.

3.19 The following program skeleton asks for an angle in degrees and converts it to radians. The formatting of the final output is left to you.

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

int main()
{
     const double PI = 3.14159;
     double degrees, radians;

     cout << "Enter an angle in degrees and I will convert it\n";
     cout << "to radians for you: ";
     cin >> degrees;
     radians = degrees * PI / 180;
     // Display the value in radians left justified, in fixed
     // point notation, with 4 places of precision, in a field
     // 5 spaces wide, making sure the decimal point is always
     // displayed.
     return 0;
}
```

3.8 Working with Characters and `string` Objects

CONCEPT: Special functions exist for working with characters and **`string`** objects.

Although it is possible to use `cin` with the >> operator to input strings, it can cause problems that you need to be aware of. When `cin` reads input, it passes over and ignores any leading *whitespace* characters (spaces, tabs, or line breaks). Once it comes to the first nonblank character and starts reading, it stops reading when it gets to the next whitespace character. Program 3-18 illustrates this problem.

Program 3-18

```
1  // This program illustrates a problem that can occur if
2  // cin is used to read character data into a string object.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9       string name;
10      string city;
11
12      cout << "Please enter your name: ";
13      cin >> name;
14      cout << "Enter the city you live in: ";
15      cin >> city;
16
```

```
17       cout << "Hello, " << name << endl;
18       cout << "You live in " << city << endl;
19       return 0;
20   }
```

Program Output with Example Input Shown in Bold

```
Please enter your name: Kate Smith [Enter]
Enter the city you live in: Hello, Kate
You live in Smith
```

Notice that the user was never given the opportunity to enter the city. In the first input statement, when `cin` came to the space between `Kate` and `Smith`, it stopped reading, storing just `Kate` as the value of `name`. In the second input statement, `cin` used the leftover characters it found in the keyboard buffer and stored `Smith` as the value of `city`.

To work around this problem, you can use a C++ function named `getline`. The `getline` function reads an entire line, including leading and embedded spaces, and stores it in a `string` object. The `getline` function looks like the following, where `cin` is the input stream we are reading from and `inputLine` is the name of the `string` object receiving the input.

```
getline(cin, inputLine);
```

Program 3-19 illustrates using the `getline` function.

Program 3-19

```
1   // This program demonstrates using the getline function
2   // to read character data into a string object.
3   #include <iostream>
4   #include <string>
5   using namespace std;
6
7   int main()
8   {
9        string name;
10       string city;
11
12       cout << "Please enter your name: ";
13       getline(cin, name);
14       cout << "Enter the city you live in: ";
15       getline(cin, city);
16
17       cout << "Hello, " << name << endl;
18       cout << "You live in " << city << endl;
19       return 0;
20  }
```

Program Output with Example Input Shown in Bold

```
Please enter your name: Kate Smith [Enter]
Enter the city you live in: Raleigh [Enter]
Hello, Kate Smith
You live in Raleigh
```

Inputting a Character

Sometimes you want to read only a single character of input. For example, some programs display a menu of items for the user to choose from. Often the selections are denoted by the letters A, B, C, and so forth. The user chooses an item from the menu by typing a character. The simplest way to read a single character is with `cin` and the `>>` operator, as illustrated in Program 3-20.

Program 3-20

```
1  // This program reads a single character into a char variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       char ch;
8
9       cout << "Type a character and press Enter: ";
10      cin >> ch;
11      cout << "You entered " << ch << endl;
12      return 0;
13 }
```

Program Output with Example Input Shown in Bold

```
Type a character and press Enter: A [Enter]
You entered A
```

Using `cin.get`

As with string input, however, there are times when using `cin >>` to read a character does not do what you want. For example, because it passes over all leading whitespace, it is impossible to input just a blank or [Enter] with `cin >>`. The program will not continue past the `cin` statement until some character other than the spacebar, tab key, or [Enter] key has been pressed. (Once such a character is entered, the [Enter] key must still be pressed before the program can continue to the next statement.) Thus, programs that ask the user to `"Press the Enter key to continue."` cannot use the `>>` operator to read only the pressing of the [Enter] key.

In those situations, the `cin` object has a built-in function named `get` that is helpful. Because the `get` function is built into the `cin` object, we say that it is a *member function* of `cin`. The `get` member function reads a single character, including any whitespace character. If the program needs to store the character being read, the `get` member function can be called in either of the following ways. In both examples, assume that `ch` is the name of a `char` variable that the character is being read into.

```
cin.get(ch);
ch = cin.get();
```

If the program is using the `cin.get` function simply to pause the screen until the [**Enter**] key is pressed and does not need to store the character, the function can also be called like this:

```
cin.get();
```

Program 3-21 illustrates all three ways to use the `cin.get` function.

Program 3-21

```
1  // This program demonstrates three ways
2  // to use cin.get() to pause a program.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       char ch;
9
10      cout << "This program has paused. Press Enter to continue.";
11      cin.get(ch);
12      cout << "It has paused a second time. Please press Enter again.";
13      ch = cin.get();
14      cout << "It has paused a third time. Please press Enter again.";
15      cin.get();
16      cout << "Thank you!";
17      return 0;
18 }
```

Program Output with Example Input Shown in Bold

```
This program has paused. Press Enter to continue. [Enter]
It has paused a second time. Please press Enter again. [Enter]
It has paused a third time. Please press Enter again. [Enter]
Thank you!
```

Mixing `cin >>` and `cin.get`

Mixing `cin >>` with `cin.get` can cause an annoying and hard-to-find problem. For example, look at Program 3-22.

Program 3-22

```
1  // This program demonstrates a problem that occurs
2  // when you mix cin >> with cin.get().
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       char ch;                    // Define a character variable
9       int number;                 // Define an integer variable
10
```

(program continues)

Program 3-22 *(continued)*

```
11       cout << "Enter a number: ";
12       cin >> number;                // Read an integer
13       cout << "Enter a character: ";
14       ch = cin.get();               // Read a character
15       cout << "Thank You!\n";
16       return 0;
17   }
```

Program Output with Example Input Shown in Bold

```
Enter a number: 100 [Enter]
Enter a character: Thank You!
```

When this program runs, line 12 lets the user enter a number, but it appears as though the statement in line 14 is skipped. This happens because `cin >>` and `cin.get` use slightly different techniques for reading data.

In the example run of the program, when line 12 executed, the user entered 100 and pressed the [**Enter**] key. Pressing the [**Enter**] key causes a newline character (`'\n'`) to be stored in the keyboard buffer, as shown in Figure 3-5. The `cin >>` statement in line 12 begins reading the data that the user entered, and stops reading when it comes to the newline character. This is shown in Figure 3-6. The newline character is not read, but remains in the keyboard buffer.

Figure 3-5

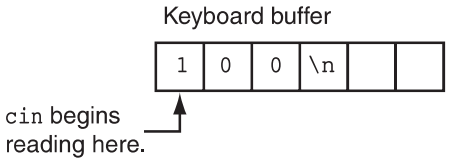
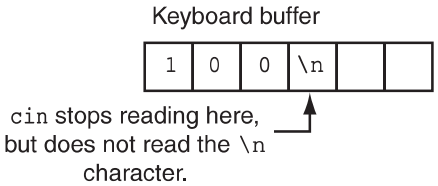


Figure 3-6



When the `cin.get` function in line 14 executes, it begins reading the keyboard buffer where the previous input operation stopped. That means that `cin.get` reads the newline character, without giving the user a chance to enter any more input. You can remedy this situation by using the `cin.ignore` function, described in the following section.

Using cin.ignore

To solve the problem previously described, you can use another of the cin object's member functions named ignore. The cin.ignore function tells the cin object to skip one or more characters in the keyboard buffer. Here is its general form:

```
cin.ignore(n, c);
```

The arguments shown in the parentheses are optional. If used, *n* is an integer and *c* is a character. They tell cin to skip n number of characters, or until the character *c* is encountered. For example, the following statement causes cin to skip the next 20 characters or until a newline is encountered, whichever comes first:

```
cin.ignore(20,'\n');
```

If no arguments are used, cin will skip only the very next character. Here's an example:

```
cin.ignore();
```

Program 3-23, which is a modified version of Program 3-22, demonstrates the function. Notice that a call to cin.ignore has been inserted in line 13, right after the cin >> statement.

Program 3-23

```
1  // This program successfully uses both
2  // cin >> and cin.get() for keyboard input.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       char ch;
9       int number;
10
11      cout << "Enter a number: ";
12      cin >> number;
13      cin.ignore();              // Skip the newline character
14      cout << "Enter a character: ";
15      ch = cin.get();
16      cout << "Thank You!\n";
17      return 0;
18 }
```

Program Output with Example Input Shown in Bold

```
Enter a number: 100 [Enter]
Enter a character: Z [Enter]
Thank You!
```

`string` Member Functions and Operators

C++ `string` objects also have a number of member functions. For example, if you want to know the length of the string that is stored in a `string` object, you can call the object's `length` member function. Here is an example of how to use it.

```
string state = "Texas";
int size = state.length();
```

The first statement creates a `string` object named `state` and initializes it with the string `"Texas"`. The second statement defines an `int` variable named `size` and initializes it with the length of the string in the `state` object. After this code executes, the size variable will hold the value 5.

Certain operators also work with `string` objects. One of them is the `+` operator. You have already encountered the `+` operator to add two numeric quantities. Because strings cannot be added, when this operator is used with string operands it *concatenates* them, or joins them together. Assume we have the following definitions and initializations in a program.

```
string greeting1 = "Hello ";
string greeting2;
string name1 = "World";
string name2 = "People";
```

The following statements illustrate how string concatenation works.

```
greeting2 = greeting1 + name1; // greeting2 now holds "Hello World"
greeting1 = greeting1 + name2; // greeting1 now holds "Hello People"
```

Notice that the string stored in `greeting1` has a blank as its last character. If the blank were not there, `greeting2` would have been assigned the string `"HelloWorld"`.

The last statement in the previous example could also have been written using the `+=` combined assignment operator, to achieve the same result:

```
greeting1 += name2;
```

You will learn about other useful `string` member functions and operators in Chapter 10.

3.9 More Mathematical Library Functions

CONCEPT: **The C++ runtime library provides several functions for performing complex mathematical operations.**

Earlier in this chapter you learned to use the `pow` function to raise a number to a power. The C++ library has numerous other functions that perform specialized mathematical operations. These functions are useful in scientific and special-purpose programs. Table 3-13 shows several of these, each of which requires the `cmath` header file.

Table 3-13

Function	Example	Description
`abs`	`y = abs(x);`	Returns the absolute value of the argument. The argument and the return value are integers.
`cos`	`y = cos(x);`	Returns the cosine of the argument. The argument should be an angle expressed in radians. The return type and the argument are `double`s.
`exp`	`y = exp(x);`	Computes the exponential function of the argument, which is `x`. The return type and the argument are `double`s.
`fmod`	`y = fmod(x, z);`	Returns, as a `double`, the remainder of the first argument divided by the second argument. Works like the modulus operator, but the arguments are `double`s. (The modulus operator only works with integers.) Take care not to pass zero as the second argument. Doing so would cause division by zero.
`log`	`y = log(x);`	Returns the natural logarithm of the argument. The return type and the argument are `double`s.
`log10`	`y = log10(x);`	Returns the base-10 logarithm of the argument. The return type and the argument are `double`s.
`sin`	`y = sin(x);`	Returns the sine of the argument. The argument should be an angle expressed in radians. The return type and the argument are `double`s.
`sqrt`	`y = sqrt(x);`	Returns the square root of the argument. The return type and argument are `double`s.
`tan`	`y = tan(x);`	Returns the tangent of the argument. The argument should be an angle expressed in radians. The return type and the argument are `double`s.

Each of these functions is as simple to use as the `pow` function. The following program segment demonstrates the `sqrt` function, which returns the square root of a number:

```
cout << "Enter a number: ";
cin >> num;
s = sqrt(num);
cout << "The square root of " << num << " is " << s << endl;
```

Here is the output of the program segment, with 25 as the number entered by the user:

```
Enter a number: 25
The square root of 25 is 5
```

Program 3-24 shows the `sqrt` function being used to find the hypotenuse of a right triangle. The program uses the following formula, taken from the Pythagorean theorem:

$$c = \sqrt{a^2 + b^2}$$

In the formula, c is the length of the hypotenuse, and a and b are the lengths of the other sides of the triangle.

Program 3-24

```
1  // This program asks for the lengths of the two sides of a
2  // right triangle. The length of the hypotenuse is then
3  // calculated and displayed.
4  #include <iostream>
5  #include <iomanip>     // For setprecision
6  #include <cmath>       // For the sqrt and pow functions
7  using namespace std;
8
9  int main()
10 {
11     double a, b, c;
12
13     cout << "Enter the length of side a: ";
14     cin >> a;
15     cout << "Enter the length of side b: ";
16     cin >> b;
17     c = sqrt(pow(a, 2.0) + pow(b, 2.0));
18     cout << "The length of the hypotenuse is ";
19     cout << setprecision(2) << c << endl;
20     return 0;
21 }
```

Program Output with Example Input Shown in Bold

```
Enter the length of side a: 5.0 [Enter]
Enter the length of side b: 12.0 [Enter]
The length of the hypotenuse is 13
```

The following statement, taken from Program 3-24, calculates the square root of the sum of the squares of the triangle's two sides:

```
c = sqrt(pow(a, 2.0) + pow(b, 2.0));
```

Notice that the following mathematical expression is used as the `sqrt` function's argument:

```
pow(a, 2.0) + pow(b, 2.0)
```

This expression calls the `pow` function twice: once to calculate the square of `a` and again to calculate the square of `b`. These two squares are then added together, and the sum is sent to the `sqrt` function.

Random Numbers

Random numbers are useful for lots of different programming tasks. The following are just a few examples:

- Random numbers are commonly used in games. For example, computer games that let the player roll dice use random numbers to represent the values of the dice. Programs that show cards being drawn from a shuffled deck use random numbers to represent the face values of the cards.

- Random numbers are useful in simulation programs. In some simulations, the computer must randomly decide how a person, animal, insect, or other living being will behave. Formulas can be constructed in which a random number is used to determine various actions and events that take place in the program.
- Random numbers are useful in statistical programs that must randomly select data for analysis.
- Random numbers are commonly used in computer security to encrypt sensitive data.

The C++ library has a function, `rand()`, that you can use to generate random numbers. (The `rand()` function requires the `cstdlib` header file.) The number returned from the function is an `int`. Here is an example of its usage:

```
y = rand();
```

After this statement executes, the variable `y` will contain a random number. In actuality, the numbers produced by `rand()`are pseudorandom. The function uses an algorithm that produces the same sequence of numbers each time the program is repeated on the same system. For example, suppose the following statements are executed.

```
cout << rand() << endl;
cout << rand() << endl;
cout << rand() << endl;
```

The three numbers displayed will appear to be random, but each time the program runs, the same three values will be generated. In order to randomize the results of `rand()`, the `srand()` function must be used. `srand()` accepts an `unsigned int` argument, which acts as a seed value for the algorithm. By specifying different seed values, `rand()` will generate different sequences of random numbers.

A common practice for getting unique seed values is to call the `time` function, which is part of the standard library. The `time` function returns the number of seconds that have elapsed since midnight, January 1, 1970. The `time` function requires the `ctime` header file, and you pass 0 as an argument to the function. Program 3-25 demonstrates. The program should generate three different random numbers each time it is executed.

Program 3-25

```
1  // This program demonstrates random numbers.
2  #include <iostream>
3  #include <cstdlib>     // For rand and srand
4  #include <ctime>       // For the time function
5  using namespace std;
6
7  int main()
8  {
9      // Get the system time.
10     unsigned seed = time(0);
11
12     // Seed the random number generator.
13     srand(seed);
14
```

(program continues)

Program 3-25 *(continued)*

```
15      // Display three random numbers.
16      cout << rand() << endl;
17      cout << rand() << endl;
18      cout << rand() << endl;
19      return 0;
20  }
```

Program Output

```
23861
20884
21941
```

If you wish to limit the range of the random number, use the following formula:

```
y = (rand() % (maxValue - minValue + 1)) + minValue;
```

In the formula, *minValue* is the lowest number in the range, and *maxValue* is the highest number in the range. For example, the following code assigns a random number in the range of 1 through 100 to the variable y:

```
const int MIN_VALUE = 1;
const int MAX_VALUE = 100;
y = (rand() % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
```

As another example, the following code assigns a random number in the range of 100 through 200 to the variable y:

```
const int MIN_VALUE = 100;
const int MAX_VALUE = 200;
y = (rand() % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
```

The following "In the Spotlight" section demonstrates how to use random numbers to simulate rolling dice.

In the Spotlight:

Using Random Numbers

Dr. Kimura teaches an introductory statistics class and has asked you to write a program that he can use in class to simulate the rolling of dice. The program should randomly generate two numbers in the range of 1 through 6 and display them. Program 3-26 shows the program, with three examples of program output.

Program 3-26

```
1  // This program simulates rolling dice.
2  #include <iostream>
3  #include <cstdlib>    // For rand and srand
4  #include <ctime>      // For the time function
5  using namespace std;
6
```

```
int main()
{
     // Constants
     const int MIN_VALUE = 1;  // Minimum die value
     const int MAX_VALUE = 6;  // Maximum die value

     // Variables
     int die1;  // To hold the value of die #1
     int die2;  // To hold the value of die #2

     // Get the system time.
     unsigned seed = time(0);

     // Seed the random number generator.
     srand(seed);

     cout << "Rolling the dice…\n";
     die1 = (rand() % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
     die2 = (rand() % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
     cout << die1 << endl;
     cout << die2 << endl;
     return 0;
}
```

Program Output

```
Rolling the dice...
5
2
```

Program Output

```
Rolling the dice...
4
6
```

Program Output

```
Rolling the dice...
3
1
```

Checkpoint

3.20 Write a short description of each of the following functions:

```
cos     log     sin
exp     log10   sqrt
fmod    pow     tan
```

3.21 Assume the variables `angle1` and `angle2` hold angles stored in radians. Write a statement that adds the sine of `angle1` to the cosine of `angle2` and stores the result in the variable `x`.

3.22 To find the cube root (the third root) of a number, raise it to the power of 1/3. To find the fourth root of a number, raise it to the power of ¼. Write a statement that will find the fifth root of the variable `x` and store the result in the variable `y`.

3.23 The cosecant of the angle a is

$$\frac{1}{\sin a}$$

Write a statement that calculates the cosecant of the angle stored in the variable `a`, and stores it in the variable `y`.

3.10 Focus on Debugging: Hand Tracing a Program

Hand tracing is a debugging process where you pretend that you are the computer executing a program. You step through each of the program's statements one by one. As you look at a statement, you record the contents that each variable will have after the statement executes. This process is often helpful in finding mathematical mistakes and other logic errors.

To hand trace a program you construct a chart with a column for each variable. The rows in the chart correspond to the lines in the program. For example, Program 3-27 is shown with a hand trace chart. The program uses the following four variables: `num1`, `num2`, `num3`, and `avg`. Notice that the hand trace chart has a column for each variable and a row for each line of code in function `main`.

Program 3-27

```
1  // This program asks for three numbers, then
2  // displays the average of the numbers.
3  #include <iostream>
4  using namespace std;
5  int main()
6  {
7       double num1, num2, num3, avg;
8       cout << "Enter the first number: ";
9       cin >> num1;
10      cout << "Enter the second number: ";
11      cin >> num2;
12      cout << "Enter the third number: ";
13      cin >> num3;
14      avg = num1 + num2 + num3 / 3;
15      cout << "The average is " << avg << endl;
16      return 0;
17 }
```

num1	num2	num3	avg

This program, which asks the user to enter three numbers and then displays the average of the numbers, has a bug. It does not display the correct average. The output of a sample session with the program follows.

Program Output with Example Input Shown in Bold

```
Enter the first number: 10 [Enter]
Enter the second number: 20 [Enter]
Enter the third number: 30 [Enter]
The average is 40
```

The correct average of 10, 20, and 30 is 20, not 40. To find the error we will hand trace the program. To hand trace this program, you step through each statement, observing the operation that is taking place, and then record the contents of the variables after the statement executes. After the hand trace is complete, the chart will appear as follows. We have written question marks in the chart where we do not know the contents of a variable.

Program 3-27 (with hand trace chart filled)

```
1  // This program asks for three numbers, then
2  // displays the average of the numbers.
3  #include <iostream>
4  using namespace std;
5  int main()
6  {
7       double num1, num2, num3, avg;
8       cout << "Enter the first number: ";
9       cin >> num1;
10      cout << "Enter the second number: ";
11      cin >> num2;
12      cout << "Enter the third number: ";
13      cin >> num3;
14      avg = num1 + num2 + num3 / 3;
15      cout << "The average is " << avg << endl;
16      return 0;
17 }
```

num1	num2	num3	avg
?	?	?	?
?	?	?	?
10	?	?	?
10	?	?	?
10	20	?	?
10	20	?	?
10	20	30	?
10	20	30	40
10	20	30	40

Do you see the error? By examining the statement that performs the math operation in line 14, we find a mistake. The division operation takes place before the addition operations, so we must rewrite that statement as

```
avg = (num1 + num2 + num3) / 3;
```

Hand tracing is a simple process that focuses your attention on each statement in a program. Often this helps you locate errors that are not obvious.

3.11 Focus on Problem Solving: A Case Study

General Crates, Inc. builds custom-designed wooden crates. With materials and labor, it costs GCI $0.23 per cubic foot to build a crate. In turn, they charge their customers $0.50 per cubic foot for the crate. You have been asked to write a program that calculates the volume (in cubic feet), cost, customer price, and profit of any crate GCI builds.

Variables

Table 3-14 shows the named constants and variables needed.

Table 3-14

Constant or Variable	Description
`COST_PER_CUBIC_FOOT`	A named constant, declared as a `double` and initialized with the value 0.23. This represents the cost to build a crate, per cubic foot.
`CHARGE_PER_CUBIC_FOOT`	A named constant, declared as a `double` and initialized with the value 0.5. This represents the amount charged for a crate, per cubic foot.
`length`	A `double` variable to hold the length of the crate, which is input by the user.
`width`	A `double` variable to hold the width of the crate, which is input by the user.
`height`	A `double` variable to hold the height of the crate, which is input by the user.
`volume`	A `double` variable to hold the volume of the crate. The value stored in this variable is calculated.
`cost`	A `double` variable to hold the cost of building the crate. The value stored in this variable is calculated.
`charge`	A `double` variable to hold the amount charged to the customer for the crate. The value stored in this variable is calculated.
`profit`	A `double` variable to hold the profit GCI makes from the crate. The value stored in this variable is calculated.

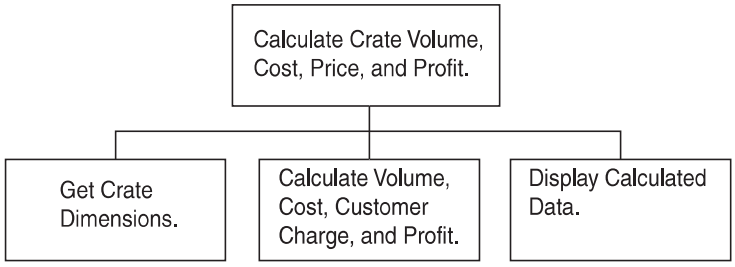
Program Design

The program must perform the following general steps:

1. Ask the user to enter the dimensions of the crate (the crate's length, width, and height).
2. Calculate the crate's volume, the cost of building the crate, the customer's charge, and the profit made.
3. Display the data calculated in Step 2.

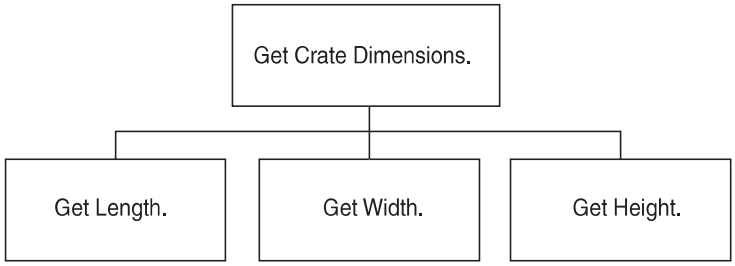
A general hierarchy chart for this program is shown in Figure 3-7.

Figure 3-7



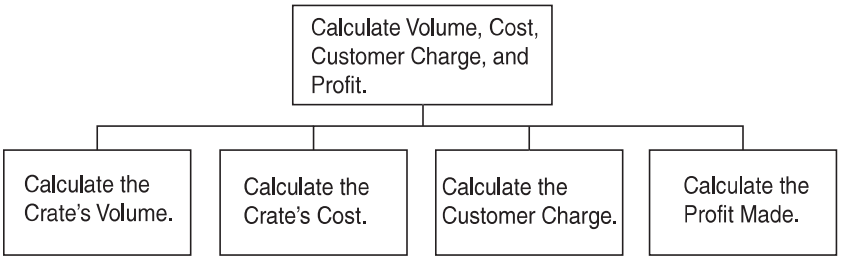
The "Get Crate Dimensions" step is shown in greater detail in Figure 3-8.

Figure 3-8



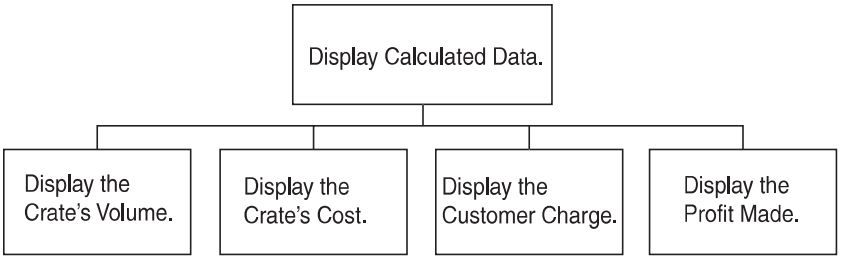
The "Calculate Volume, Cost, Customer Charge, and Profit" step is shown in greater detail in Figure 3-9.

Figure 3-9



The "Display Calculated Data" step is shown in greater detail in Figure 3-10.

Figure 3-10



Pseudocode for the program is as follows:

```
Ask the user to input the crate's length.
Ask the user to input the crate's width.
Ask the user to input the crate's height.
Calculate the crate's volume.
Calculate the cost of building the crate.
Calculate the customer's charge for the crate.
Calculate the profit made from the crate.
Display the crate's volume.
Display the cost of building the crate.
Display the customer's charge for the crate.
Display the profit made from the crate.
```

Calculations

The following formulas will be used to calculate the crate's volume, cost, charge, and profit:

volume = length × width × height

cost = volume × 0.23

charge = volume × 0.5

profit = charge − cost

The Program

The last step is to expand the pseudocode into the final program, which is shown in Program 3-28.

Program 3-28

```
1  // This program is used by General Crates, Inc. to calculate
2  // the volume, cost, customer charge, and profit of a crate
3  // of any size. It calculates this data from user input, which
4  // consists of the dimensions of the crate.
5  #include <iostream>
6  #include <iomanip>
7  using namespace std;
8
9  int main()
10 {
11     // Constants for cost and amount charged
12     const double COST_PER_CUBIC_FOOT = 0.23;
13     const double CHARGE_PER_CUBIC_FOOT = 0.5;
14
15     // Variables
16     double length,  // The crate's length
17            width,   // The crate's width
18            height,  // The crate's height
19            volume,  // The volume of the crate
20            cost,    // The cost to build the crate
```

```
21            charge,    // The customer charge for the crate
22            profit;    // The profit made on the crate
23
24      // Set the desired output formatting for numbers.
25      cout << setprecision(2) << fixed << showpoint;
26
27      // Prompt the user for the crate's length, width, and height
28      cout << "Enter the dimensions of the crate (in feet):\n";
29      cout << "Length: ";
30      cin >> length;
31      cout << "Width: ";
32      cin >> width;
33      cout << "Height: ";
34      cin >> height;
35
36      // Calculate the crate's volume, the cost to produce it,
37      // the charge to the customer, and the profit.
38      volume = length * width * height;
39      cost = volume * COST_PER_CUBIC_FOOT;
40      charge = volume * CHARGE_PER_CUBIC_FOOT;
41      profit = charge - cost;
42
43      // Display the calculated data.
44      cout << "The volume of the crate is ";
45      cout << volume << " cubic feet.\n";
46      cout << "Cost to build: $" << cost << endl;
47      cout << "Charge to customer: $" << charge << endl;
48      cout << "Profit: $" << profit << endl;
49      return 0;
50  }
```

Program Output with Example Input Shown in Bold

```
Enter the dimensions of the crate (in feet):
Length: 10 [Enter]
Width: 8 [Enter]
Height: 4 [Enter]
The volume of the crate is 320.00 cubic feet.
Cost to build: $73.60
Charge to customer: $160.00
Profit: $86.40
```

Program Output with Different Example Input Shown in Bold

```
Enter the dimensions of the crate (in feet):
Length: 12.5 [Enter]
Width: 10.5 [Enter]
Height: 8 [Enter]
The volume of the crate is 1050.00 cubic feet.
Cost to build: $241.50
Charge to customer: $525.00
Profit: $283.50
```

Review Questions and Exercises

Short Answer

1. Assume that the following variables are defined:

```
int age;
double pay;
char section;
```

   Write a single `cin` statement that will read input into each of these variables.

2. Assume a `string` object has been defined as follows:

```
string description;
```

   A) Write a `cin` statement that reads in a one-word string.

   B) Write a statement that reads in a string that can contain multiple words separated by blanks.

3. What header files must be included in the following program?

```
int main()
{
    double amount = 89.7;
    cout << showpoint << fixed;
    cout << setw(8) << amount << endl;
    return 0;
}
```

4. Complete the following table by writing the value of each expression in the Value column.

Expression	Value
`28 / 4 - 2`	
`6 + 12 * 2 - 8`	
`4 + 8 * 2`	
`6 + 17 % 3 - 2`	
`2 + 22 * (9 - 7)`	
`(8 + 7) * 2`	
`(16 + 7) % 2 - 1`	
`12 / (10 - 6)`	
`(19 - 3) * (2 + 2) / 4`	

5. Write C++ expressions for the following algebraic expressions:

   $a = 12x$

   $z = 5x + 14y + 6k$

   $y = x^4$

   $g = \frac{h + 12}{4k}$

   $c = \frac{a^3}{b^2k^4}$

6. Assume a program has the following variable definitions:

```
int units;
float mass;
double weight;
```

and the following statement:

```
weight = mass * units;
```

Which automatic data type conversion will take place?

A) `mass` is demoted to an `int`, `units` remains an `int`, and the result of `mass * units` is an `int`.

B) `units` is promoted to a `float`, `mass` remains a `float`, and the result of `mass * units` is a `float`.

C) `units` is promoted to a `float`, `mass` remains a `float`, and the result of `mass * units` is a `double`.

7. Assume a program has the following variable definitions:

```
int a, b = 2;
float c = 4.2;
```

and the following statement:

```
a = b * c;
```

What value will be stored in `a`?

A) 8.4

B) 8

C) 0

D) None of the above

8. Assume that `qty` and `salesReps` are both integers. Use a type cast expression to rewrite the following statement so it will no longer perform integer division.

```
unitsEach = qty / salesReps;
```

9. Rewrite the following variable definition so the variable is a named constant.

```
int rate;
```

10. Complete the following table by writing statements with combined assignment operators in the right-hand column. The statements should be equivalent to the statements in the left-hand column.

Statements with Assignment Operator	Statements with Combined Assignment Operator
`x = x + 5;`	
`total = total + subtotal;`	
`dist = dist / rep;`	
`ppl = ppl * period;`	
`inv = inv - shrinkage;`	
`num = num % 2;`	

11. Write a multiple assignment statement that can be used instead of the following group of assignment statements:

```
east = 1;
west = 1;
north = 1;
south = 1;
```

12. Write a `cout` statement so the variable `divSales` is displayed in a field of 8 spaces, in fixed point notation, with a precision of 2 decimal places. The decimal point should always be displayed.
13. Write a `cout` statement so the variable `totalAge` is displayed in a field of 12 spaces, in fixed point notation, with a precision of 4 decimal places.
14. Write a `cout` statement so the variable `population` is displayed in a field of 12 spaces, left-justified, with a precision of 8 decimal places. The decimal point should always be displayed.

Fill-in-the-Blank

15. The _____ library function returns the cosine of an angle.
16. The _____ library function returns the sine of an angle.
17. The _____ library function returns the tangent of an angle.
18. The _____ library function returns the exponential function of a number.
19. The _____ library function returns the remainder of a floating point division.
20. The _____ library function returns the natural logarithm of a number.
21. The _____ library function returns the base-10 logarithm of a number.
22. The _____ library function returns the value of a number raised to a power.
23. The _____ library function returns the square root of a number.
24. The _____ file must be included in a program that uses the mathematical functions.

Algorithm Workbench

25. A retail store grants its customers a maximum amount of credit. Each customer's available credit is his or her maximum amount of credit minus the amount of credit used. Write a pseudocode algorithm for a program that asks for a customer's maximum amount of credit and amount of credit used. The program should then display the customer's available credit.

    After you write the pseudocode algorithm, convert it to a complete C++ program.
26. Write a pseudocode algorithm for a program that calculates the total of a retail sale. The program should ask for the amount of the sale and the sales tax rate. The sales tax rate should be entered as a floating-point number. For example, if the sales tax rate is 6 percent, the user should enter 0.06. The program should display the amount of sales tax and the total of the sale.

    After you write the pseudocode algorithm, convert it to a complete C++ program.

27. Write a pseudocode algorithm for a program that asks the user to enter a golfer's score for three games of golf, and then displays the average of the three scores.

    After you write the pseudocode algorithm, convert it to a complete C++ program.

Find the Errors

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

28.
```
using namespace std;
int main ()
{
     double number1, number2, sum;

     Cout << "Enter a number: ";
     Cin << number1;
     Cout << "Enter another number: ";
     Cin << number2;
     number1 + number2 = sum;
     Cout "The sum of the two numbers is " << sum
     return 0;
}
```

29.
```
#include <iostream>
using namespace std;

int main()
{
     int number1, number2;
     float quotient;
     cout << "Enter two numbers and I will divide\n";
     cout << "the first by the second for you.\n";
     cin >> number1, number2;
     quotient = float<static_cast>(number1) / number2;
     cout << quotient
     return 0;
}
```

30.
```
#include <iostream>;
using namespace std;

int main()
{
     const int number1, number2, product;

     cout << "Enter two numbers and I will multiply\n";
     cout << "them for you.\n";
     cin >> number1 >> number2;
     product = number1 * number2;
     cout << product
     return 0;
}
```

31.
```
#include <iostream>;
using namespace std;

main
{
     int number1, number2;

     cout << "Enter two numbers and I will multiply\n"
     cout << "them by 50 for you.\n"
     cin >> number1 >> number2;
     number1 =* 50;
     number2 =* 50;
     cout << number1 << " " << number2;
     return 0;
}
```

32.
```
#include <iostream>;
using namespace std;

main
{
     double number, half;

     cout << "Enter a number and I will divide it\n"
     cout << "in half for you.\n"
     cin >> number1;
     half =/ 2;
     cout << fixedpoint << showpoint << half << endl;
     return 0;
}
```

33.
```
#include <iostream>;
using namespace std;

int main()
{
     char name, go;

     cout << "Enter your name: ";
     getline >> name;
     cout << "Hi " << name << endl;
     return 0;
}
```

Predict the Output

What will each of the following programs display? (Some should be hand traced and require a calculator.)

34. (*Assume the user enters 38700. Use a calculator.*)

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

```
int main()
{
    double salary, monthly;
    cout << "What is your annual salary? ";
    cin >> salary;
    monthly = static_cast<int>(salary) / 12;
    cout << "Your monthly wages are " << monthly << endl;
    return 0;
}
```

35.
```
#include <iostream>
using namespace std;
int main()
{
    long x, y, z;

    x = y = z = 4;
    x += 2;
    y -= 1;
    z *= 3;
    cout << x << " " << y << " " << z << endl;
    return 0;
}
```

36. (*Assume the user enters George Washington.*)
```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main()
{
    string userInput;
    cout << "What is your name? ";
    getline(cin, userInput);
    cout << "Hello " << userInput << endl;
    return 0;
}
```

37. (*Assume the user enters 36720152. Use a calculator.*)
```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    long seconds;
    double minutes, hours, days, months, years;

    cout << "Enter the number of seconds that have\n";
    cout << "elapsed since some time in the past and\n";
    cout << "I will tell you how many minutes, hours,\n";
    cout << "days, months, and years have passed: ";
    cin >> seconds;
```

```
    minutes = seconds / 60;
    hours = minutes / 60;
    days = hours / 24;
    years = days / 365;
    months = years * 12;
    cout << setprecision(4) << fixed << showpoint << right;
    cout << "Minutes: " << setw(6) << minutes << endl;
    cout << "Hours: " << setw(6) << hours << endl;
    cout << "Days: " << setw(6) << days << endl;
    cout << "Months: " << setw(6) << months << endl;
    cout << "Years: " << setw(6) << years << endl;
    return 0;
}
```

Programming Challenges

1. **Miles per Gallon**

   Write a program that calculates a car's gas mileage. The program should ask the user to enter the number of gallons of gas the car can hold and the number of miles it can be driven on a full tank. It should then display the number of miles that may be driven per gallon of gas.

2. **Stadium Seating**

   
   VideoNote
   Solving the Stadium Seating Problem

   There are three seating categories at a stadium. For a softball game, Class A seats cost $15, Class B seats cost $12, and Class C seats cost $9. Write a program that asks how many tickets for each class of seats were sold, then displays the amount of income generated from ticket sales. Format your dollar amount in fixed-point notation, with two decimal places of precision, and be sure the decimal point is always displayed.

3. **Test Average**

   Write a program that asks for five test scores. The program should calculate the average test score and display it. The number displayed should be formatted in fixed-point notation, with one decimal point of precision.

4. **Average Rainfall**

   Write a program that calculates the average rainfall for three months. The program should ask the user to enter the name of each month, such as June or July, and the amount of rain (in inches) that fell each month. The program should display a message similar to the following:

   The average rainfall for June, July, and August is 6.72 inches.

5. **Male and Female Percentages**

   Write a program that asks the user for the number of males and the number of females registered in a class. The program should display the percentage of males and females in the class.

   Hint: Suppose there are 8 males and 12 females in a class. There are 20 students in the class. The percentage of males can be calculated as 8 ÷ 20 = 0.4, or 40%. The percentage of females can be calculated as 12 ÷ 20 = 0.6, or 60%.

6. **Ingredient Adjuster**

   A cookie recipe calls for the following ingredients:

   - 1.5 cups of sugar
   - 1 cup of butter
   - 2.75 cups of flour

   The recipe produces 48 cookies with this amount of the ingredients. Write a program that asks the user how many cookies he or she wants to make, and then displays the number of cups of each ingredient needed for the specified number of cookies.

7. **Box Office**

   A movie theater only keeps a percentage of the revenue earned from ticket sales. The remainder goes to the movie distributor. Write a program that calculates a theater's gross and net box office profit for a night. The program should ask for the name of the movie, and how many adult and child tickets were sold. (The price of an adult ticket is $10.00 and a child's ticket is $6.00.) It should display a report similar to

Movie Name:	"Wheels of Fury"
Adult Tickets Sold:	382
Child Tickets Sold:	127
Gross Box Office Profit:	$ 4582.00
Net Box Office Profit:	$ 916.40
Amount Paid to Distributor:	$ 3665.60

NOTE: Assume the theater keeps 20 percent of the gross box office profit.

8. **How Many Widgets?**

   The Yukon Widget Company manufactures widgets that weigh 12.5 pounds each. Write a program that calculates how many widgets are stacked on a pallet, based on the total weight of the pallet. The program should ask the user how much the pallet weighs by itself and with the widgets stacked on it. It should then calculate and display the number of widgets stacked on the pallet.

9. **How Many Calories?**

   A bag of cookies holds 30 cookies. The calorie information on the bag claims that there are 10 "servings" in the bag and that a serving equals 300 calories. Write a program that asks the user to input how many cookies he or she actually ate and then reports how many total calories were consumed.

10. **How Much Insurance?**

    Many financial experts advise that property owners should insure their homes or buildings for at least 80 percent of the amount it would cost to replace the structure. Write a program that asks the user to enter the replacement cost of a building and then displays the minimum amount of insurance he or she should buy for the property.

11. **Automobile Costs**

    Write a program that asks the user to enter the monthly costs for the following expenses incurred from operating his or her automobile: loan payment, insurance, gas, oil, tires, and maintenance. The program should then display the total monthly cost of these expenses, and the total annual cost of these expenses.

12. Celsius to Fahrenheit

Write a program that converts Celsius temperatures to Fahrenheit temperatures. The formula is

$$F = \frac{9}{5}C + 32$$

F is the Fahrenheit temperature, and C is the Celsius temperature.

13. Currency

Write a program that will convert U.S. dollar amounts to Japanese yen and to euros, storing the conversion factors in the constants `YEN_PER_DOLLAR` and `EUROS_PER_DOLLAR`. To get the most up-to-date exchange rates, search the Internet using the term "currency exchange rate". If you cannot find the most recent exchange rates, use the following:

1 Dollar = 98.93 Yen
1 Dollar = 0.74 Euros

Format your currency amounts in fixed-point notation, with two decimal places of precision, and be sure the decimal point is always displayed.

14. Monthly Sales Tax

A retail company must file a monthly sales tax report listing the sales for the month and the amount of sales tax collected. Write a program that asks for the month, the year, and the total amount collected at the cash register (that is, sales plus sales tax). Assume the state sales tax is 4 percent and the county sales tax is 2 percent.

If the total amount collected is known and the total sales tax is 6 percent, the amount of product sales may be calculated as:

$$S = \frac{T}{1.06}$$

S is the product sales and *T* is the total income (product sales plus sales tax).

The program should display a report similar to

```
Month: October
--------------------
Total Collected:      $ 26572.89
Sales:                $ 25068.76
County Sales Tax:     $   501.38
State Sales Tax:      $  1002.75
Total Sales Tax:      $  1504.13
```

15. Property Tax

A county collects property taxes on the assessment value of property, which is 60 percent of the property's actual value. If an acre of land is valued at $10,000, its assessment value is $6,000. The property tax is then 75¢ for each $100 of the assessment value. The tax for the acre assessed at $6,000 will be $45. Write a program that asks for the actual value of a piece of property and displays the assessment value and property tax.

16. Senior Citizen Property Tax

Madison County provides a $5,000 homeowner exemption for its senior citizens. For example, if a senior's house is valued at $158,000 its assessed value would be $94,800,

as explained above. However, he would only pay tax on $89,800. At last year's tax rate of $2.64 for each $100 of assessed value, the property tax would be $2,370.72. In addition to the tax break, senior citizens are allowed to pay their property tax in four equal payments. The quarterly payment due on this property would be $592.68. Write a program that asks the user to input the actual value of a piece of property and the current tax rate for each $100 of assessed value. The program should then calculate and report how much annual property tax a senior homeowner will be charged for this property and what the quarterly tax bill will be.

17. Math Tutor

Write a program that can be used as a math tutor for a young student. The program should display two random numbers to be added, such as

```
  247
+129
```

The program should then pause while the student works on the problem. When the student is ready to check the answer, he or she can press a key and the program will display the correct solution:

```
  247
+129
 376
```

18. Interest Earned

Assuming there are no deposits other than the original investment, the balance in a savings account after one year may be calculated as

$$\texttt{Amount} = \texttt{Principal} * (1 + \frac{\texttt{Rate}}{\texttt{T}})^{\texttt{T}}$$

`Principal` is the balance in the savings account, `Rate` is the interest rate, and `T` is the number of times the interest is compounded during a year (`T` is 4 if the interest is compounded quarterly).

Write a program that asks for the principal, the interest rate, and the number of times the interest is compounded. It should display a report similar to

```
Interest Rate:             4.25%
Times Compounded:            12
Principal:            $ 1000.00
Interest:             $   43.34
Amount in Savings:    $ 1043.34
```

19. Monthly Payments

The monthly payment on a loan may be calculated by the following formula:

$$\texttt{Payment} = \frac{\texttt{Rate} * (1 + \texttt{Rate})^{\texttt{N}}}{((1 + \texttt{Rate})^{\texttt{N}} - 1)} * \texttt{L}$$

`Rate` is the monthly interest rate, which is the annual interest rate divided by 12. (12% annual interest would be 1 percent monthly interest.) `N` is the number of payments, and

`L` is the amount of the loan. Write a program that asks for these values and displays a report similar to

```
Loan Amount:              $ 10000.00
Monthly Interest Rate:             1%
Number of Payments:               36
Monthly Payment:          $   332.14
Amount Paid Back:         $ 11957.15
Interest Paid:            $  1957.15
```

20. Pizza Pi

Joe's Pizza Palace needs a program to calculate the number of slices a pizza of any size can be divided into. The program should perform the following steps:

A) Ask the user for the diameter of the pizza in inches.
B) Calculate the number of slices that may be taken from a pizza of that size.
C) Display a message telling the number of slices.

To calculate the number of slices that may be taken from the pizza, you must know the following facts:

- Each slice should have an area of 14.125 inches.
- To calculate the number of slices, simply divide the area of the pizza by 14.125.
- The area of the pizza is calculated with this formula:

Area = πr^2

NOTE: π is the Greek letter pi. 3.14159 can be used as its value. The variable r is the radius of the pizza. Divide the diameter by 2 to get the radius.

Make sure the output of the program displays the number of slices in fixed point notation, rounded to one decimal place of precision. Use a named constant for pi.

21. How Many Pizzas?

Modify the program you wrote in Programming Challenge 18 (Pizza Pi) so that it reports the number of pizzas you need to buy for a party if each person attending is expected to eat an average of four slices. The program should ask the user for the number of people who will be at the party and for the diameter of the pizzas to be ordered. It should then calculate and display the number of pizzas to purchase.

22. Angle Calculator

Write a program that asks the user for an angle, entered in radians. The program should then display the sine, cosine, and tangent of the angle. (Use the `sin`, `cos`, and `tan` library functions to determine these values.) The output should be displayed in fixed-point notation, rounded to four decimal places of precision.

23. Stock Transaction Program

Last month Joe purchased some stock in Acme Software, Inc. Here are the details of the purchase:

- The number of shares that Joe purchased was 1,000.
- When Joe purchased the stock, he paid $45.50 per share.
- Joe paid his stockbroker a commission that amounted to 2% of the amount he paid for the stock.

Two weeks later Joe sold the stock. Here are the details of the sale:

- The number of shares that Joe sold was 1,000.
- He sold the stock for $56.90 per share.
- He paid his stockbroker another commission that amounted to 2% of the amount he received for the stock.

Write a program that displays the following information:

- The amount of money Joe paid for the stock.
- The amount of commission Joe paid his broker when he bought the stock.
- The amount that Joe sold the stock for.
- The amount of commission Joe paid his broker when he sold the stock.
- Display the amount of profit that Joe made after selling his stock and paying the two commissions to his broker. (If the amount of profit that your program displays is a negative number, then Joe lost money on the transaction.)

24. Word Game

Write a program that plays a word game with the user. The program should ask the user to enter the following:

- His or her name
- His or her age
- The name of a city
- The name of a college
- A profession
- A type of animal
- A pet's name

After the user has entered these items, the program should display the following story, inserting the user's input into the appropriate locations:

```
There once was a person named NAME who lived in CITY. At the age of
AGE, NAME went to college at COLLEGE. NAME graduated and went to work
as a PROFESSION. Then, NAME adopted a(n) ANIMAL named PETNAME. They
both lived happily ever after!
```

This page intentionally left blank