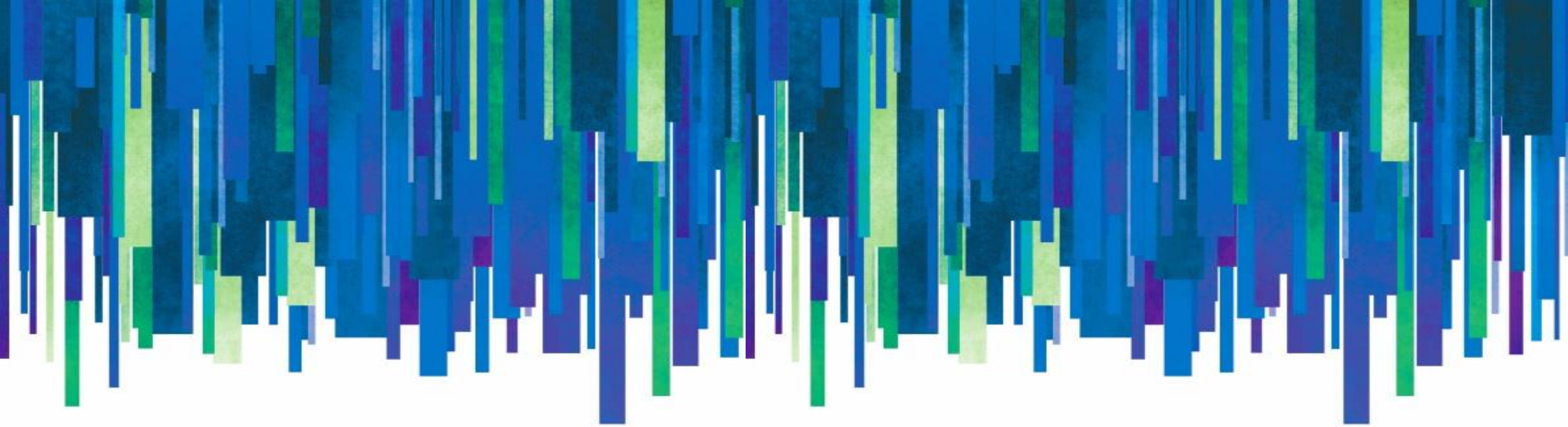


Introduction to Programming in C++ Seventh Edition

Chapter 9: Value-Returning Functions

Objectives

- Use the `sqrt` function to return the square root of a number
- Generate random numbers
- Create and invoke a function that returns a value
- Pass information *by value* to a function
- Write a function prototype
- Understand a variable's scope and lifetime

Functions

- A function is a block of code that performs a task
- Every C++ program contains at least one function (main)
  - Most contain many functions
- Some functions are built-in functions (part of C++): defined in language libraries
- Others, called program-defined functions, are written by programmers; defined in a program
- Functions allow for blocks of code to be used many times in a program without having to duplicate code

Functions (cont'd.)

- Functions also allow large, complex programs to be broken down into small, manageable sub-tasks
- Each sub-task is solved by a function, and thus different people can write different functions
- Many functions can then be combined into a single program
- Typically, `main` is used to call other functions, but any function can call any other function

Functions (cont'd.)

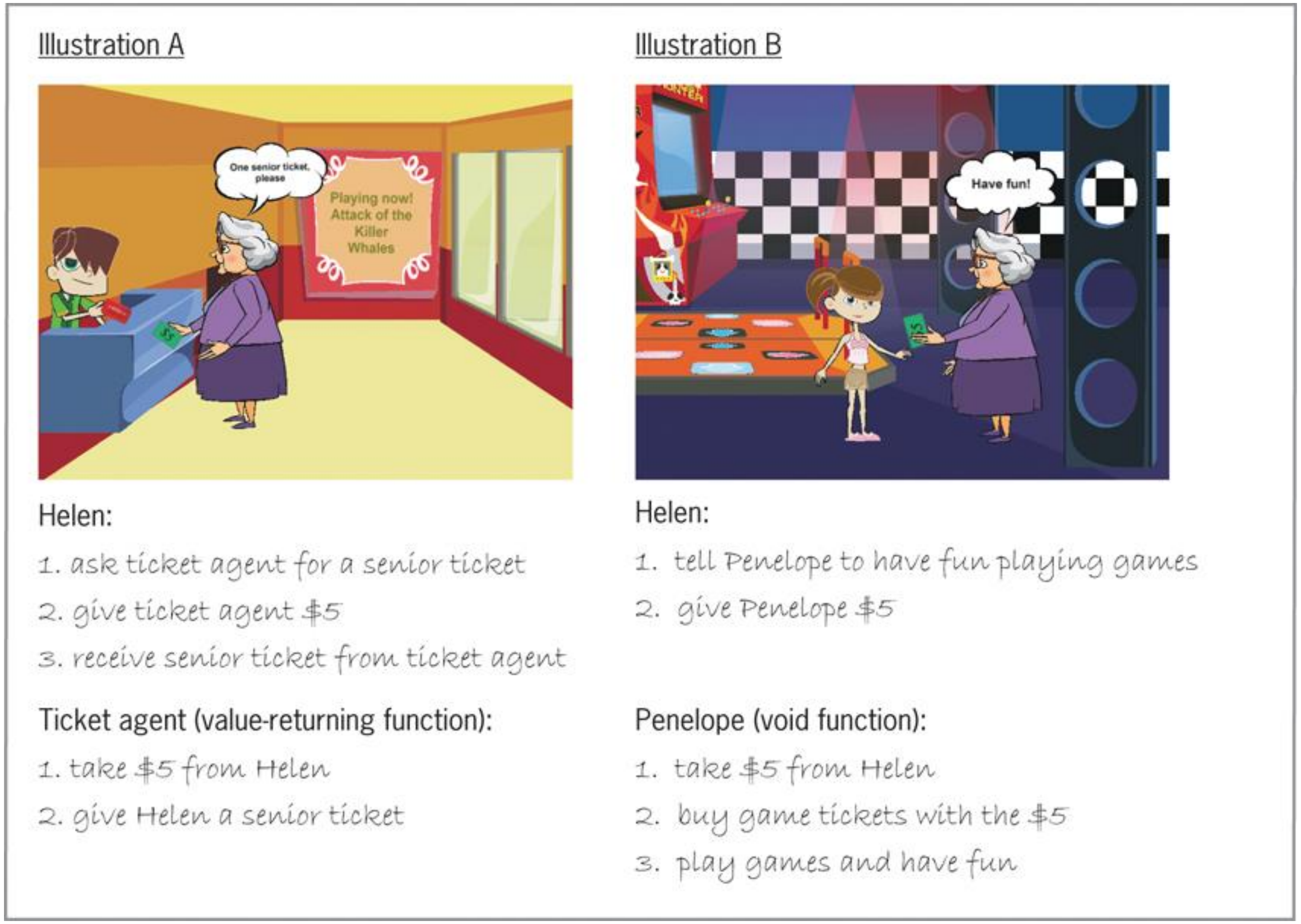

One senior ticket, please
Playing now! Attack of the Killer Whales
Have fun!
$5

Illustration A

Helen:

1. ask ticket agent for a senior ticket
2. give ticket agent $5
3. receive senior ticket from ticket agent

Ticket agent (value-returning function):

1. take $5 from Helen
2. give Helen a senior ticket

Illustration B

Helen:

1. tell Penelope to have fun playing games
2. give Penelope $5

Penelope (void function):

1. take $5 from Helen
2. buy game tickets with the $5
3. play games and have fun

Figure 9-1 Illustrations of value-returning and void functions

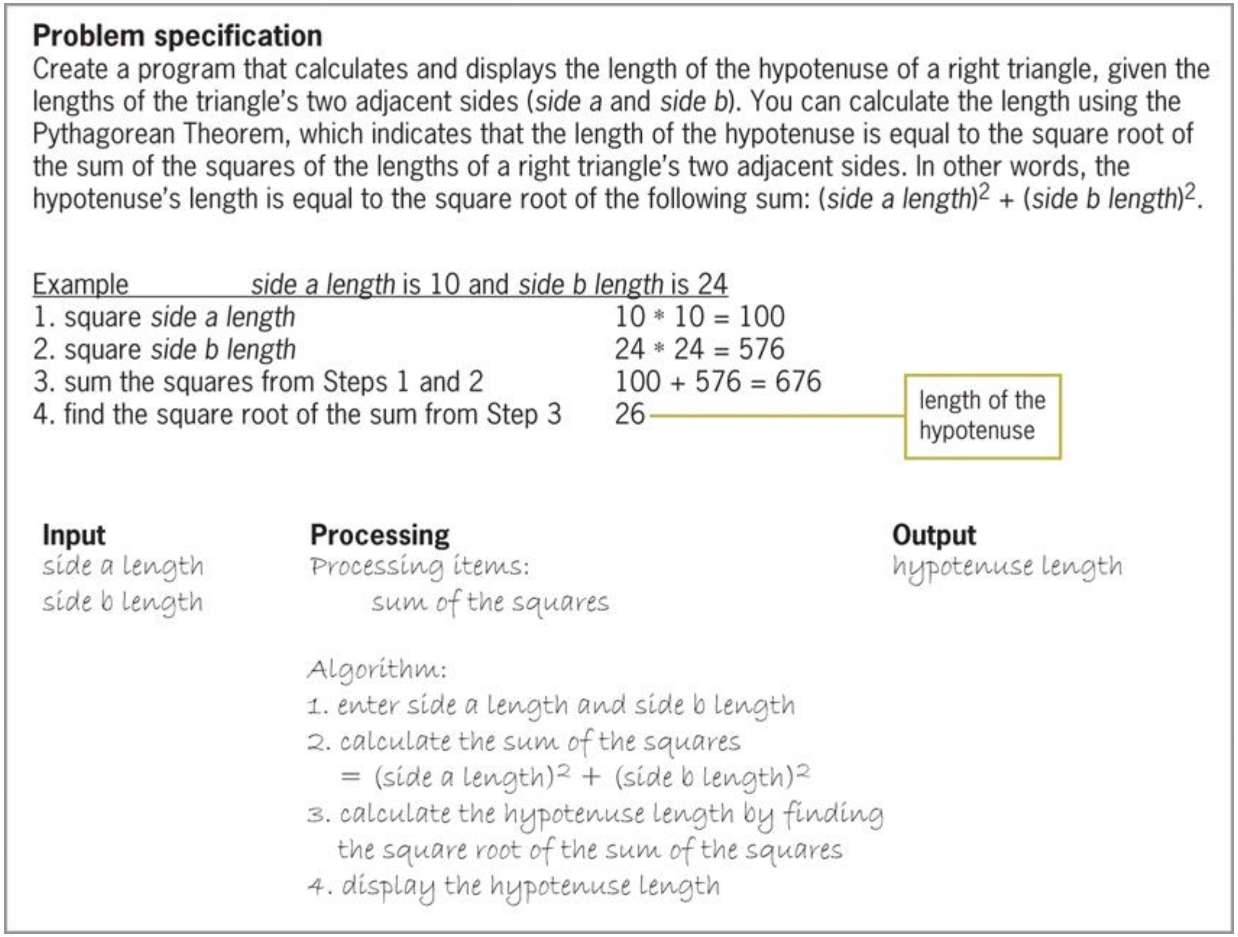
Value-Returning Functions

- All functions are either value-returning or void
- All **value-returning functions** perform a task and then return precisely one value
- In most cases, the value is returned to the statement that called the function
- Typically, a statement that calls a function assigns the return value to a variable
  - However, a return value could also be used in a comparison or calculation or could be printed to the screen

The Hypotenuse Program

- Program that calculates and displays the length of a right triangle hypotenuse
- Program uses Pythagorean theorem
  - Requires squaring and taking square root
- `pow` function can be used to square
- `sqrt` function can be used to take square root
- Both are built-in value-returning functions

The Hypotenuse Program (cont'd.)



Problem specification

Create a program that calculates and displays the length of the hypotenuse of a right triangle, given the lengths of the triangle's two adjacent sides (*side a* and *side b*). You can calculate the length using the Pythagorean Theorem, which indicates that the length of the hypotenuse is equal to the square root of the sum of the squares of the lengths of a right triangle's two adjacent sides. In other words, the hypotenuse's length is equal to the square root of the following sum: (*side a length*)^2 + (*side b length*)^2.

Example	*side a length* is 10 and *side b length* is 24
1. square *side a length*	10 * 10 = 100
2. square *side b length*	24 * 24 = 576
3. sum the squares from Steps 1 and 2	100 + 576 = 676
4. find the square root of the sum from Step 3	26

length of the hypotenuse

Input	Processing	Output
side a length side b length	Processing items: sum of the squares	hypotenuse length

```
Algorithm:
1. enter side a length and side b length
2. calculate the sum of the squares
   = (side a length)$^2$ + (side b length)$^2$
3. calculate the hypotenuse length by finding
   the square root of the sum of the squares
4. display the hypotenuse length
```

Figure 9-2 Problem specification, calculation example, and IPO chart for the hypotenuse program

The Hypotenuse Program (cont'd.)

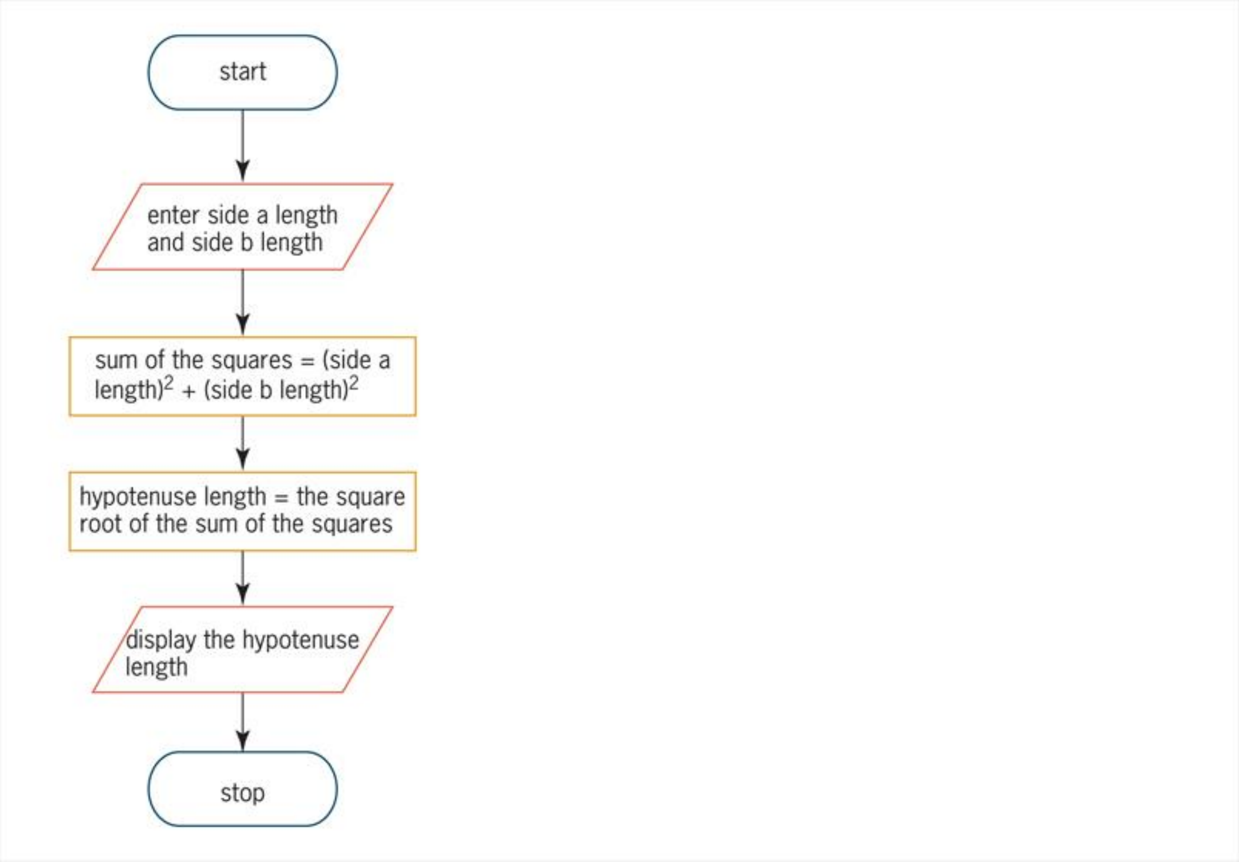
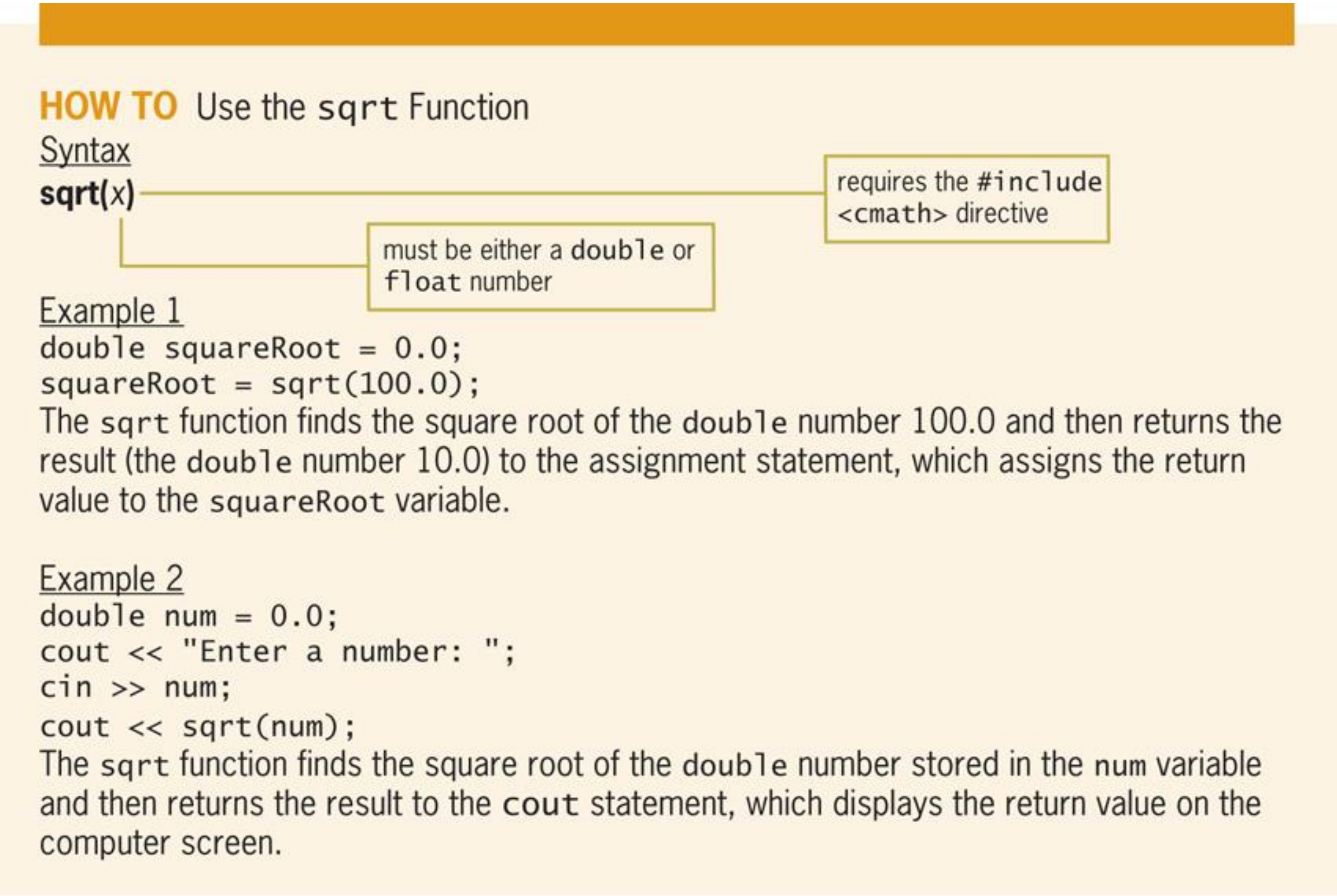

start
enter side a length and side b length
sum of the squares = (side a length)2 + (side b length)2
hypotenuse length = the square root of the sum of the squares
display the hypotenuse length
stop

Figure 9-3 Flowchart of the hypotenuse program

Finding the Square Root of a Number

- **`sqrt` function** is a built-in value-returning function that returns a number's square root as a `double`
- Definition contained in `cmath` library
  - Program must contain `#include <cmath>` to use it
- Syntax: `sqrt(x)`, in which `x` is a `double` or `float`
  - Here, `x` is an **actual argument**, which is an item of information a function needs to perform its task
- Actual arguments are passed to a function when called

Finding the Square Root of a Number (cont'd.)



HOW TO Use the `sqrt` Function

Syntax

sqrt(*x*)

requires the `#include <cmath>` directive

must be either a `double` or `float` number

Example 1

```
double squareRoot = 0.0;
squareRoot = sqrt(100.0);
```

The `sqrt` function finds the square root of the `double` number 100.0 and then returns the result (the `double` number 10.0) to the assignment statement, which assigns the return value to the `squareRoot` variable.

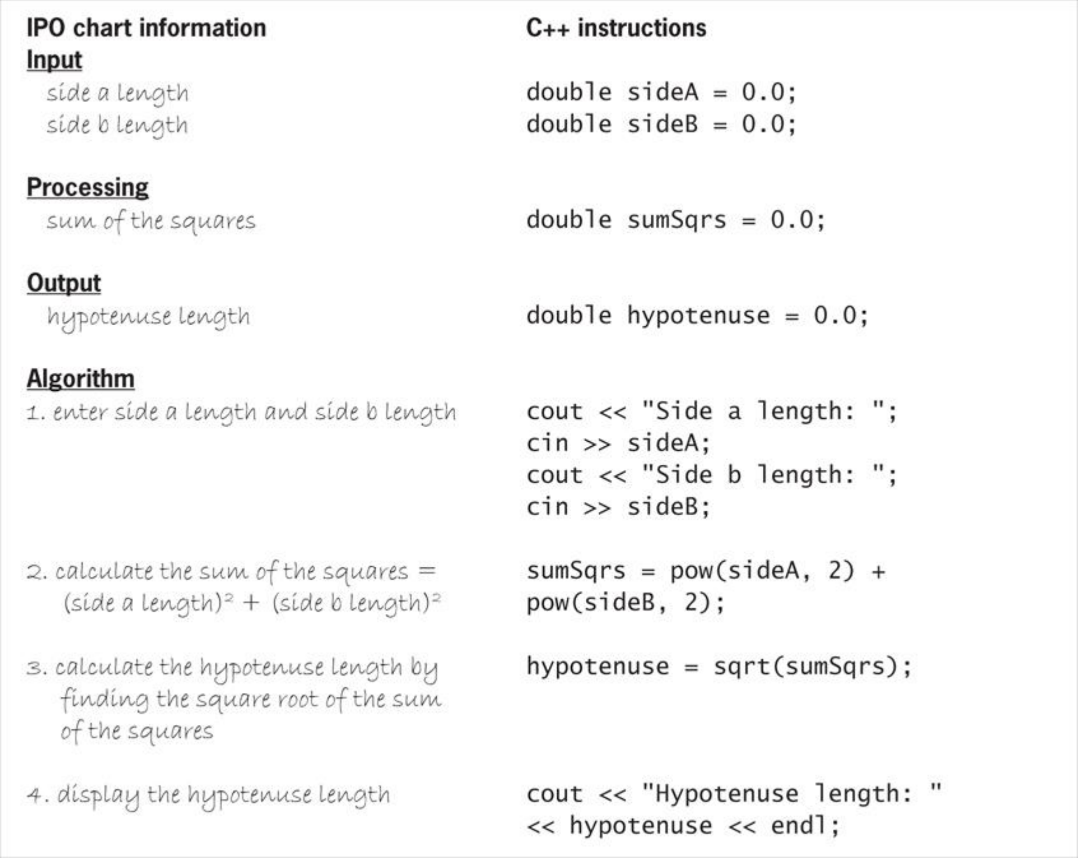
Example 2

```
double num = 0.0;
cout << "Enter a number: ";
cin >> num;
cout << sqrt(num);
```

The `sqrt` function finds the square root of the `double` number stored in the `num` variable and then returns the result to the `cout` statement, which displays the return value on the computer screen.

Figure 9-4 How to use the `sqrt` function

Finding the Square Root of a Number (cont'd.)



IPO chart information	C++ instructions
Input	
side a length	`double sideA = 0.0;`
side b length	`double sideB = 0.0;`
Processing	
sum of the squares	`double sumSqrs = 0.0;`
Output	
hypotenuse length	`double hypotenuse = 0.0;`
Algorithm	

1. enter side a length and side b length	`cout << "Side a length: ";`
	`cin >> sideA;`
	`cout << "Side b length: ";`
	`cin >> sideB;`

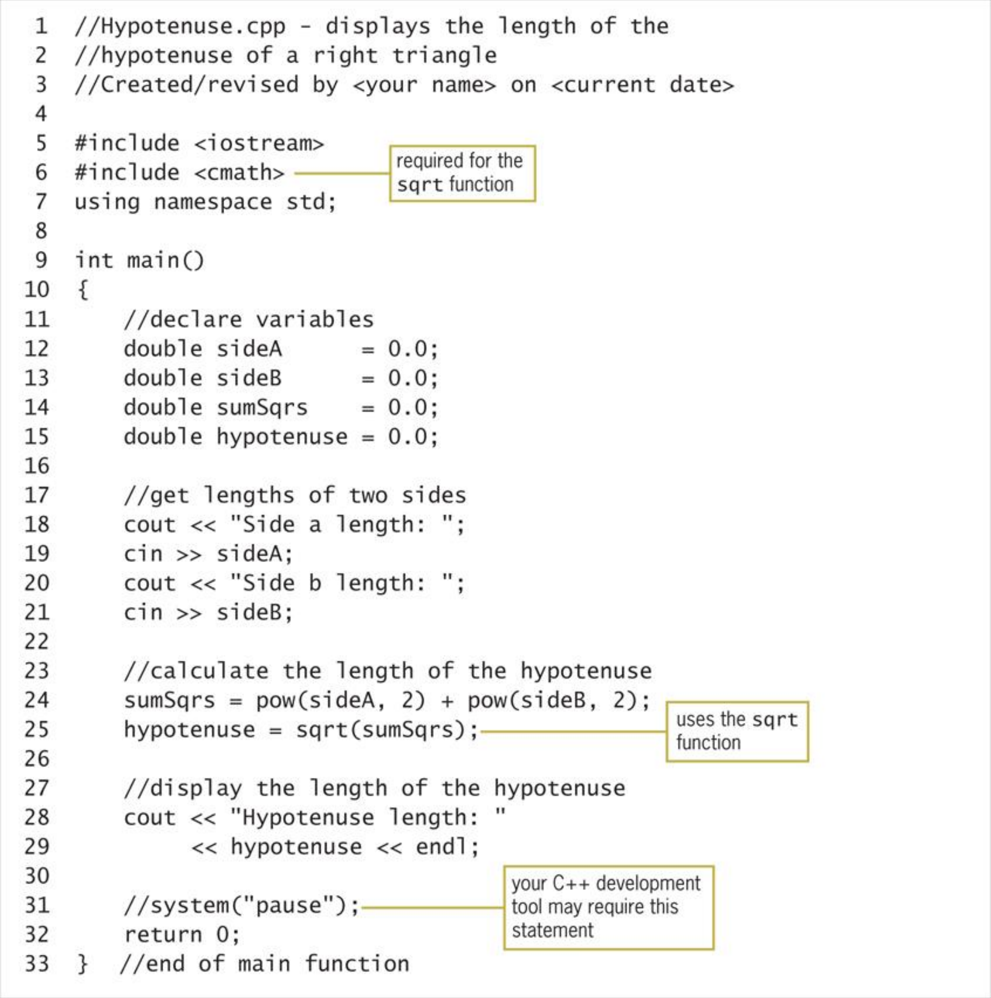
2. calculate the sum of the squares = (side a length)^2 + (side b length)^2	`sumSqrs = pow(sideA, 2) +`
	`pow(sideB, 2);`

3. calculate the hypotenuse length by finding the square root of the sum of the squares	`hypotenuse = sqrt(sumSqrs);`

4. display the hypotenuse length	`cout << "Hypotenuse length: "`
	`<< hypotenuse << endl;`

Figure 9-5 IPO chart information and C++ instructions for the hypotenuse program

Finding the Square Root of a Number (cont’d.)



```
//Hypotenuse.cpp - displays the length of the
//hypotenuse of a right triangle
//Created/revised by <your name> on <current date>

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

int main()
{
    //declare variables
    double sideA      = 0.0;
    double sideB      = 0.0;
    double sumSqrs    = 0.0;
    double hypotenuse = 0.0;

    //get lengths of two sides
    cout << "Side a length: ";
    cin >> sideA;
    cout << "Side b length: ";
    cin >> sideB;

    //calculate the length of the hypotenuse
    sumSqrs = pow(sideA, 2) + pow(sideB, 2);
    hypotenuse = sqrt(sumSqrs);

    //display the length of the hypotenuse
    cout << "Hypotenuse length: "
         << hypotenuse << endl;

    //system("pause");
    return 0;
}   //end of main function
```

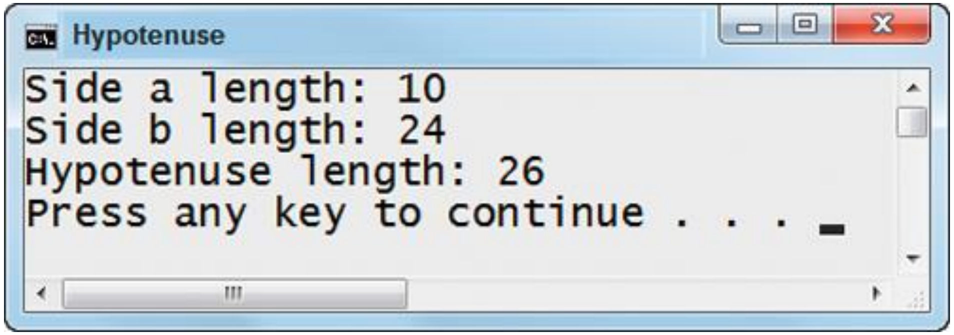
required for the sqrt function

uses the sqrt function

your C++ development tool may require this statement

Figure 9-6 Hypotenuse program

Finding the Square Root of a Number (cont’d.)



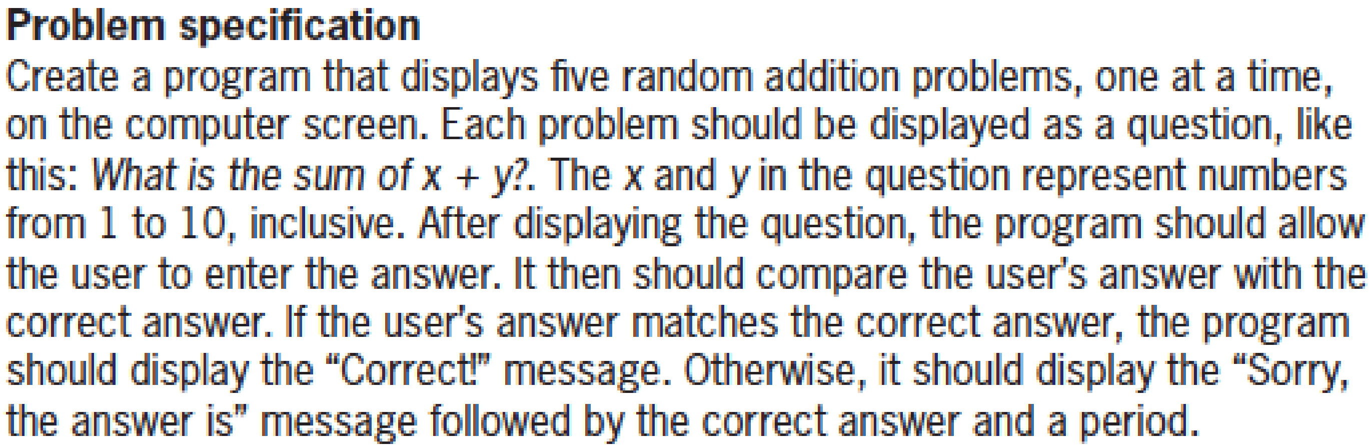
```
Side a length: 10
Side b length: 24
Hypotenuse length: 26
Press any key to continue . . .
```

Figure 9-7 Sample run of hypotenuse program

The Random Addition Problems Program

- Program that generates addition problems of the form “What is the sum of x and y?”
- Asks user to input answer, compares answer to correct answer, and displays whether correct or not
- Program requires generating random integers between 1 and 10

The Random Addition Problems Program (cont'd.)

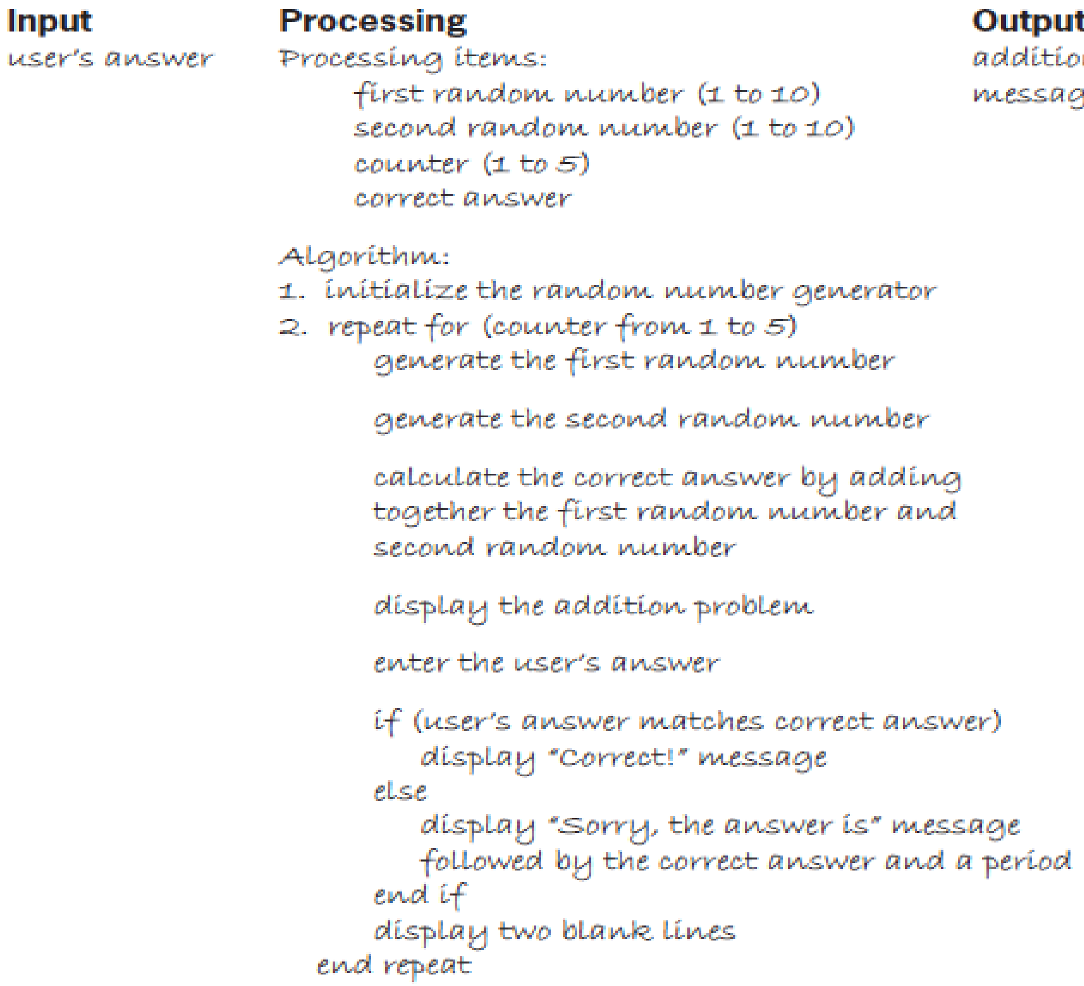


Problem specification

Create a program that displays five random addition problems, one at a time, on the computer screen. Each problem should be displayed as a question, like this: *What is the sum of* x + *y?*. The *x* and *y* in the question represent numbers from 1 to 10, inclusive. After displaying the question, the program should allow the user to enter the answer. It then should compare the user's answer with the correct answer. If the user's answer matches the correct answer, the program should display the "Correct!" message. Otherwise, it should display the "Sorry, the answer is" message followed by the correct answer and a period.

Figure 9-8 Problem specification for random addition problems program

The Random Addition Problems Program (cont'd.)



Input	Processing	Output
user's answer	Processing items: first random number (1 to 10) second random number (1 to 10) counter (1 to 5) correct answer	addition problem message

```
Algorithm:
1. initialize the random number generator
2. repeat for (counter from 1 to 5)
     generate the first random number

     generate the second random number

     calculate the correct answer by adding
     together the first random number and
     second random number

     display the addition problem

     enter the user's answer

     if (user's answer matches correct answer)
         display "Correct!" message
     else
         display "Sorry, the answer is" message
         followed by the correct answer and a period
     end if
     display two blank lines
   end repeat
```

Figure 9-8 IPO chart for random addition problems program

The Random Addition Problems Program (cont'd.)

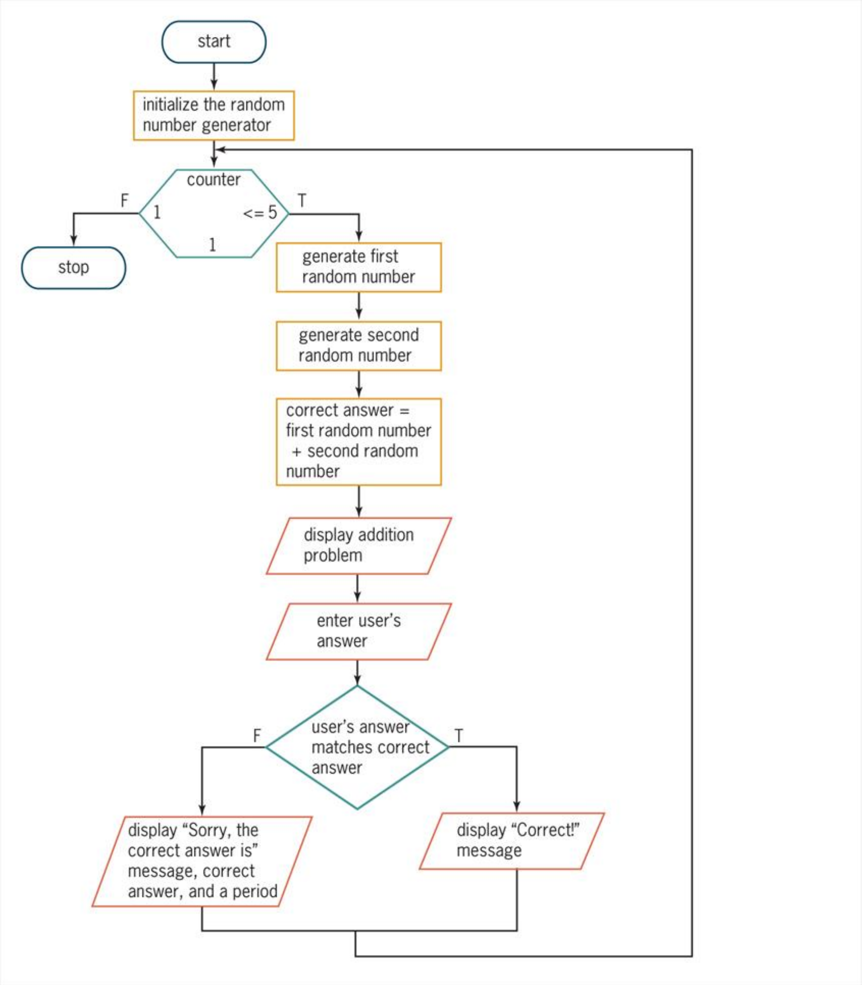

start
initialize the random number generator
counter
1
<= 5
1
F
T
stop
generate first random number
generate second random number
correct answer = first random number + second random number
display addition problem
enter user's answer
user's answer matches correct answer
F
T
display "Sorry, the correct answer is" message, correct answer, and a period
display "Correct!" message

Figure 9-9 Flowchart for random addition problems program

Generating Random Integers

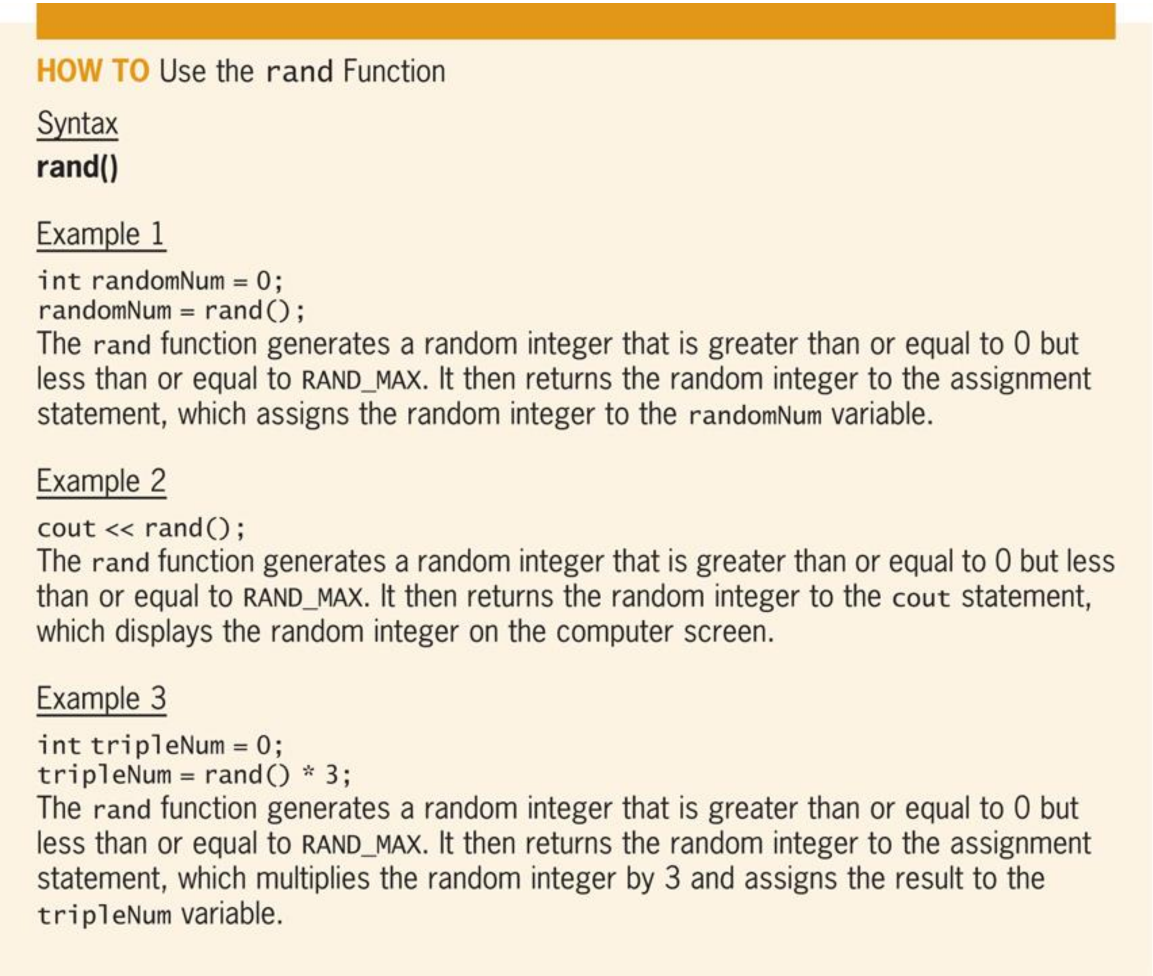
- C++ provides a pseudo-random number generator
  - Produces a sequence of numbers that meet certain statistical requirements for randomness
  - Numbers chosen uniformly from finite set of numbers
  - Not truly random but sufficient for practical purposes
- Random number generator in C++: `rand` function
  - Returns an integer between 0 and `RAND_MAX`, inclusive
  - `RAND_MAX` is a built-in constant (>= 32767)

Generating Random Integers (cont'd.)

- `rand` function's syntax: `rand()`
  - Doesn't require any actual arguments, but parentheses are still required
- Expression:

  lowerBound **+** `rand() % (`*upperBound* **–** *lowerBound* `+ 1)`
  - Allows ranges other than 0 to `RAND_MAX` to be used
  - Range is *upperBound* to *lowerBound*
- Initialize random number generator each time
  - Otherwise, will produce the same sequence

Generating Random Integers (cont'd.)



HOW TO Use the `rand` Function

Syntax

rand()

Example 1

```
int randomNum = 0;
randomNum = rand();
```

The `rand` function generates a random integer that is greater than or equal to 0 but less than or equal to `RAND_MAX`. It then returns the random integer to the assignment statement, which assigns the random integer to the `randomNum` variable.

Example 2

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

The `rand` function generates a random integer that is greater than or equal to 0 but less than or equal to `RAND_MAX`. It then returns the random integer to the `cout` statement, which displays the random integer on the computer screen.

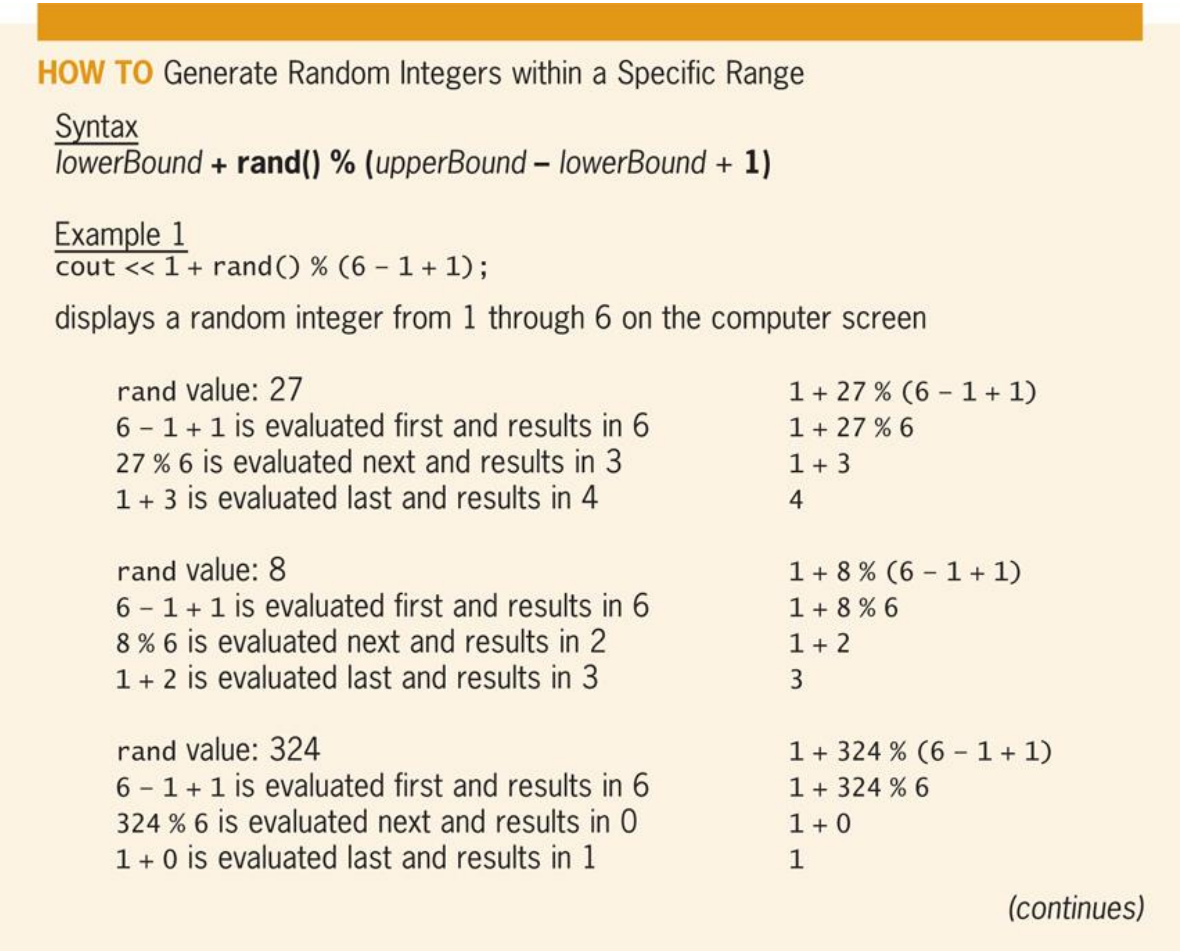
Example 3

```
int tripleNum = 0;
tripleNum = rand() * 3;
```

The `rand` function generates a random integer that is greater than or equal to 0 but less than or equal to `RAND_MAX`. It then returns the random integer to the assignment statement, which multiplies the random integer by 3 and assigns the result to the `tripleNum` variable.

Figure 9-10 How to use the `rand` function

Generating Random Integers (cont'd.)



HOW TO Generate Random Integers within a Specific Range

Syntax

lowerBound **+ rand() % (***upperBound* **–** *lowerBound* + **1)**

Example 1

```
cout << 1 + rand() % (6 - 1 + 1);
```

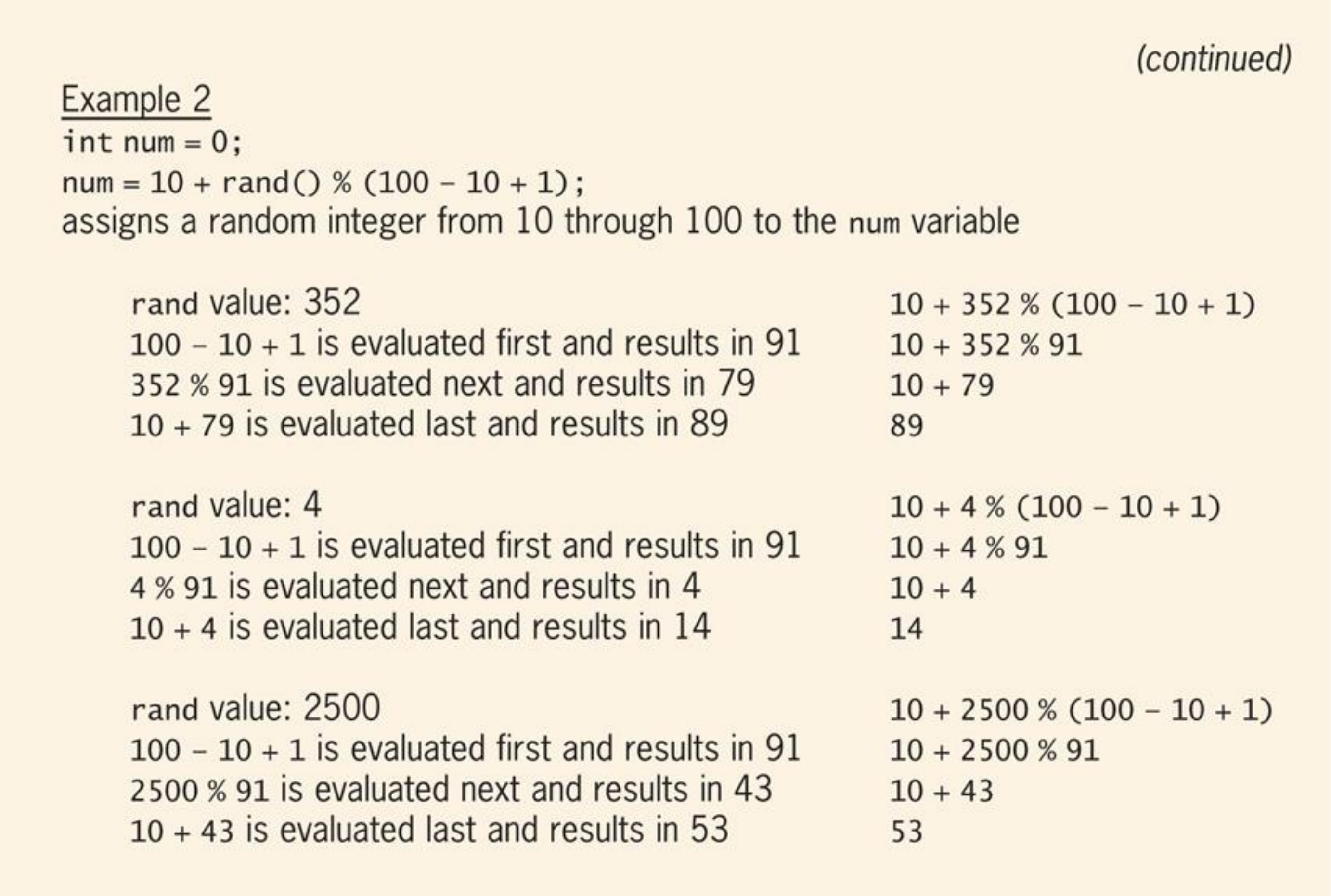
displays a random integer from 1 through 6 on the computer screen

rand value: 27	1 + 27 % (6 – 1 + 1)
6 – 1 + 1 is evaluated first and results in 6	1 + 27 % 6
27 % 6 is evaluated next and results in 3	1 + 3
1 + 3 is evaluated last and results in 4	4
rand value: 8	1 + 8 % (6 – 1 + 1)
6 – 1 + 1 is evaluated first and results in 6	1 + 8 % 6
8 % 6 is evaluated next and results in 2	1 + 2
1 + 2 is evaluated last and results in 3	3
rand value: 324	1 + 324 % (6 – 1 + 1)
6 – 1 + 1 is evaluated first and results in 6	1 + 324 % 6
324 % 6 is evaluated next and results in 0	1 + 0
1 + 0 is evaluated last and results in 1	1

(continues)

Figure 9-11 How to generate random integers within a specific range

Generating Random Integers (cont’d.)



(continued)

Example 2

```
int num = 0;
num = 10 + rand() % (100 – 10 + 1);
```

assigns a random integer from 10 through 100 to the num variable

rand value: 352	10 + 352 % (100 – 10 + 1)
100 – 10 + 1 is evaluated first and results in 91	10 + 352 % 91
352 % 91 is evaluated next and results in 79	10 + 79
10 + 79 is evaluated last and results in 89	89
rand value: 4	10 + 4 % (100 – 10 + 1)
100 – 10 + 1 is evaluated first and results in 91	10 + 4 % 91
4 % 91 is evaluated next and results in 4	10 + 4
10 + 4 is evaluated last and results in 14	14
rand value: 2500	10 + 2500 % (100 – 10 + 1)
100 – 10 + 1 is evaluated first and results in 91	10 + 2500 % 91
2500 % 91 is evaluated next and results in 43	10 + 43
10 + 43 is evaluated last and results in 53	53

Figure 9-11 How to generate random integers within a specific range (cont’d.)

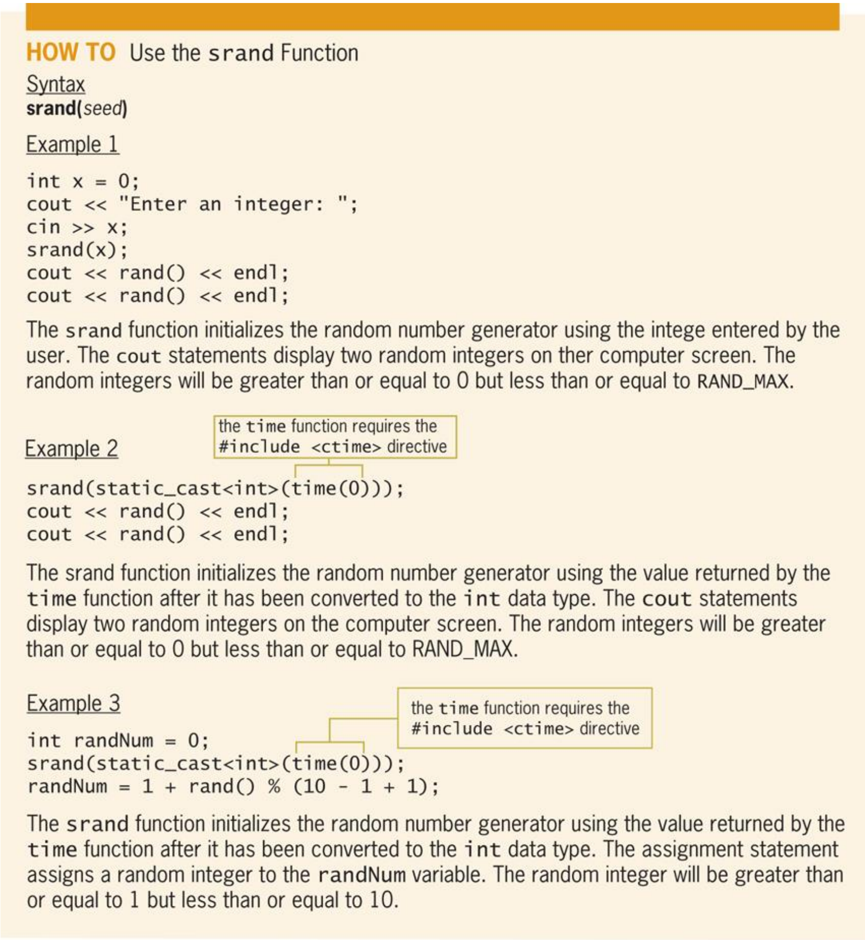
Generating Random Integers (cont'd.)

- Use `srand` function (a void function) to initialize random number generator
- Syntax: `srand(seed)`, in which `seed` is an integer actual argument that represents the starting point of the generator
  - Commonly initialized using the `time` function
    - Ensures unique sequence of numbers for each program run

Generating Random Integers (cont'd.)

- **`time` function** is a value-returning function that returns current time in number of seconds since January 1, 1970
  - Returns a `time_t` object, so must be cast to an integer before passing to `srand`
  - Program must contain `#include <ctime>` directive to use it

Generating Random Integers (cont'd.)



HOW TO Use the `srand` Function

Syntax

srand(*seed***)**

Example 1

```
int x = 0;
cout << "Enter an integer: ";
cin >> x;
srand(x);
cout << rand() << endl;
cout << rand() << endl;
```

The `srand` function initializes the random number generator using the intege entered by the user. The `cout` statements display two random integers on ther computer screen. The random integers will be greater than or equal to 0 but less than or equal to RAND_MAX.

Example 2

the `time` function requires the `#include <ctime>` directive

```
srand(static_cast<int>(time(0)));
cout << rand() << endl;
cout << rand() << endl;
```

The srand function initializes the random number generator using the value returned by the `time` function after it has been converted to the `int` data type. The `cout` statements display two random integers on the computer screen. The random integers will be greater than or equal to 0 but less than or equal to RAND_MAX.

Example 3

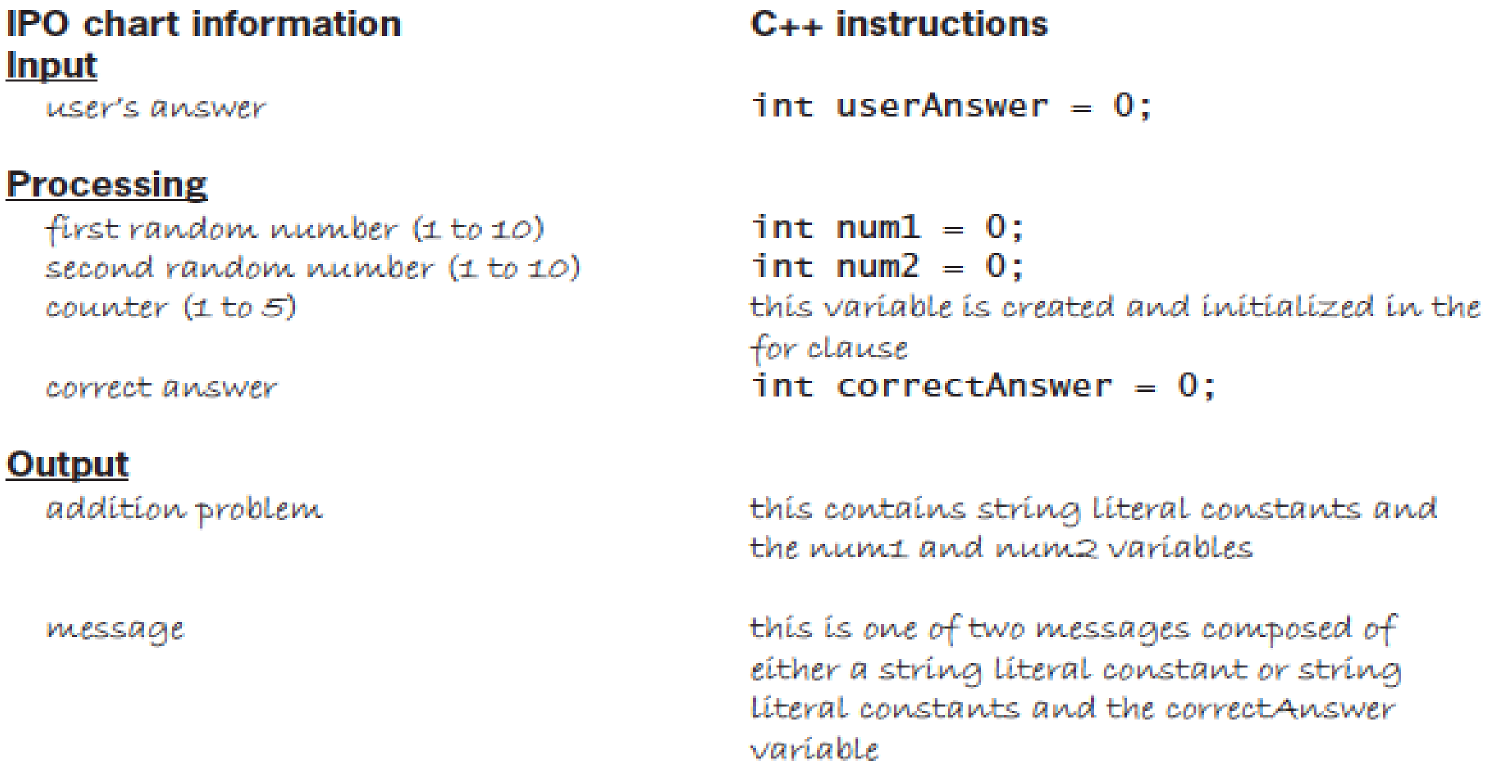
the `time` function requires the `#include <ctime>` directive

```
int randNum = 0;
srand(static_cast<int>(time(0)));
randNum = 1 + rand() % (10 - 1 + 1);
```

The `srand` function initializes the random number generator using the value returned by the `time` function after it has been converted to the `int` data type. The assignment statement assigns a random integer to the `randNum` variable. The random integer will be greater than or equal to 1 but less than or equal to 10.

Figure 9-12 How to use the `srand` function

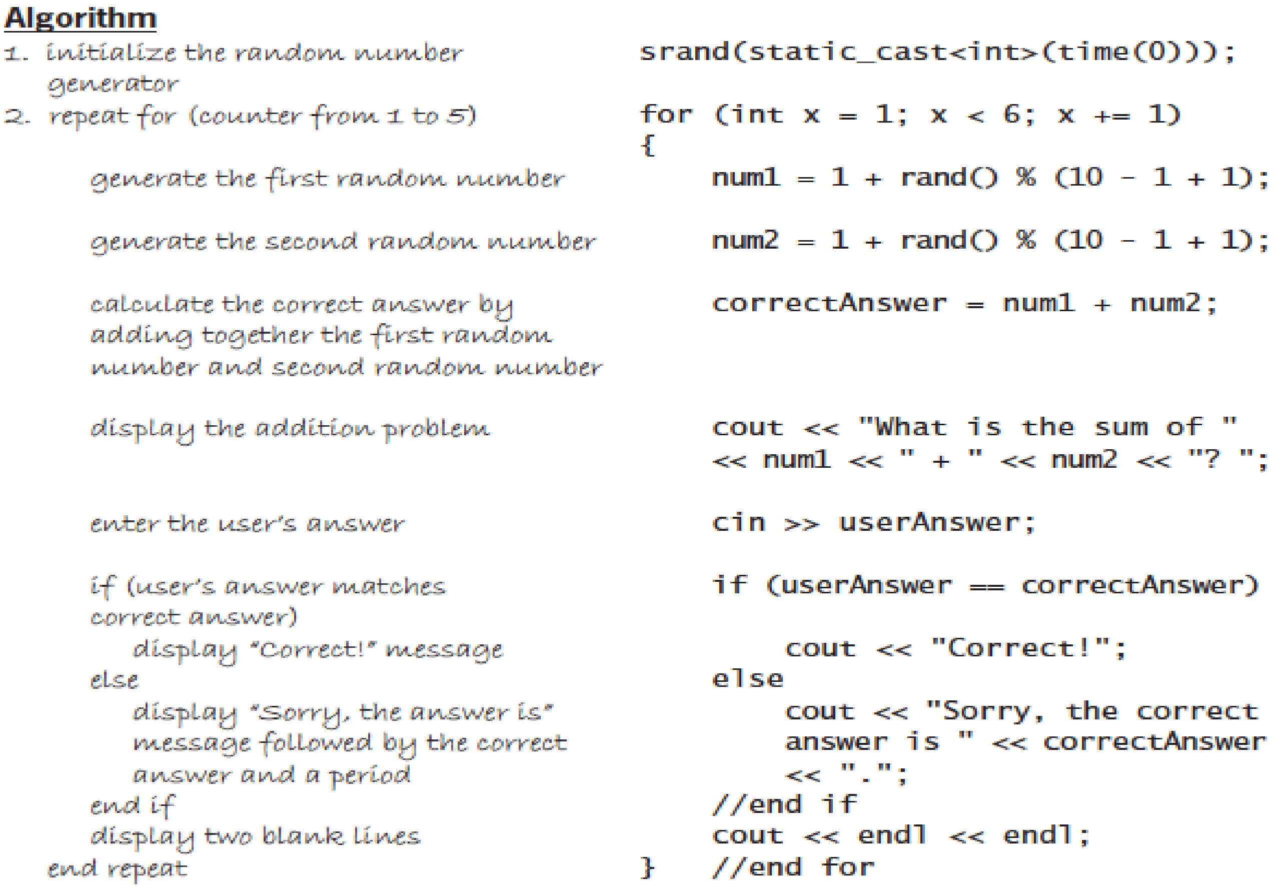
Generating Random Integers (cont'd.)



IPO chart information	C++ instructions
Input	
user's answer	`int userAnswer = 0;`
Processing	
first random number (1 to 10)	`int num1 = 0;`
second random number (1 to 10)	`int num2 = 0;`
counter (1 to 5)	this variable is created and initialized in the for clause
correct answer	`int correctAnswer = 0;`
Output	
addition problem	this contains string literal constants and the num1 and num2 variables
message	this is one of two messages composed of either a string literal constant or string literal constants and the correctAnswer variable

Figure 9-13 IPO chart information and C++ instructions for the random addition problems program

Generating Random Integers (cont'd.)



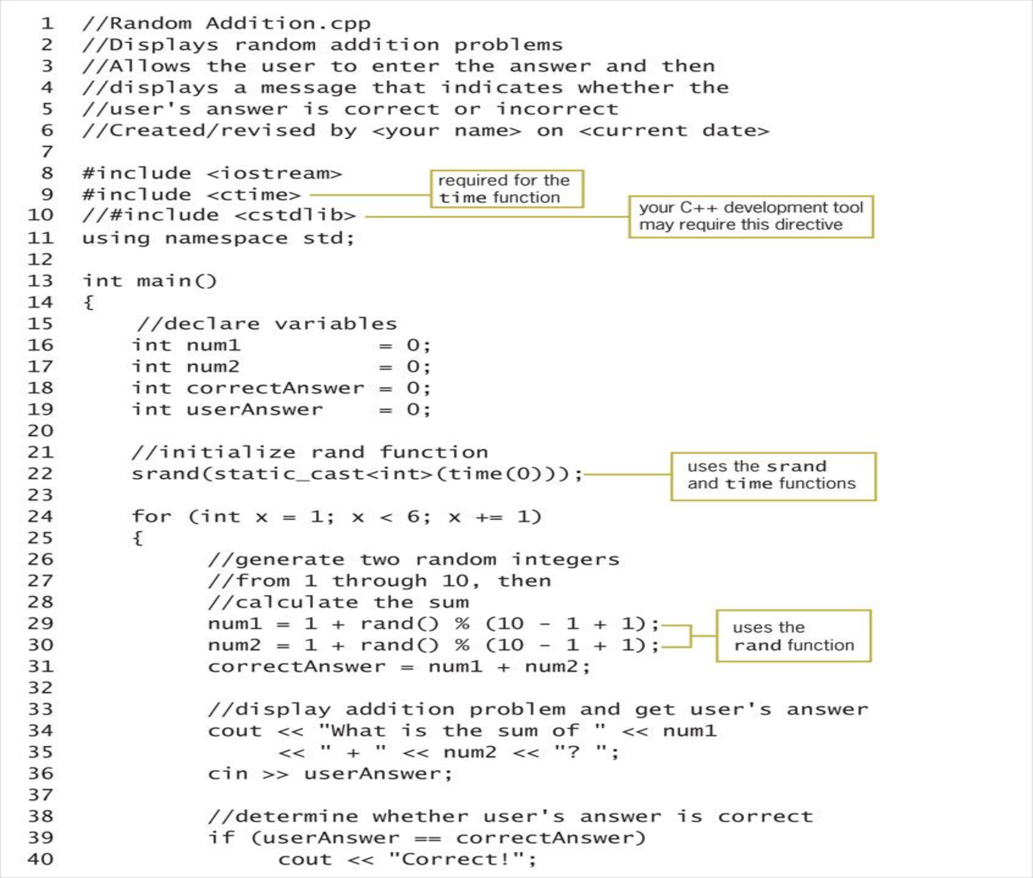
Algorithm

```
1. initialize the random number
   generator
2. repeat for (counter from 1 to 5)

      generate the first random number

      generate the second random number

      calculate the correct answer by
      adding together the first random
      number and second random number

      display the addition problem


      enter the user's answer

      if (user's answer matches
      correct answer)
         display "Correct!" message
      else
         display "Sorry, the answer is"
         message followed by the correct
         answer and a period
      end if
      display two blank lines
   end repeat
```

```
srand(static_cast<int>(time(0)));

for (int x = 1; x < 6; x += 1)
{
    num1 = 1 + rand() % (10 - 1 + 1);

    num2 = 1 + rand() % (10 - 1 + 1);

    correctAnswer = num1 + num2;



    cout << "What is the sum of "
    << num1 << " + " << num2 << "? ";

    cin >> userAnswer;

    if (userAnswer == correctAnswer)

        cout << "Correct!";
    else
        cout << "Sorry, the correct
        answer is " << correctAnswer
        << ".";
    //end if
    cout << endl << endl;
}   //end for
```

Figure 9-13 IPO chart information and C++ instructions for the random addition problems program (cont'd.)

Generating Random Integers (cont'd.)



```
//Random Addition.cpp
//Displays random addition problems
//Allows the user to enter the answer and then
//displays a message that indicates whether the
//user's answer is correct or incorrect
//Created/revised by <your name> on <current date>

#include <iostream>
#include <ctime>
//#include <cstdlib>
using namespace std;

int main()
{
    //declare variables
    int num1          = 0;
    int num2          = 0;
    int correctAnswer = 0;
    int userAnswer    = 0;

    //initialize rand function
    srand(static_cast<int>(time(0)));

    for (int x = 1; x < 6; x += 1)
    {
         //generate two random integers
         //from 1 through 10, then
         //calculate the sum
         num1 = 1 + rand() % (10 - 1 + 1);
         num2 = 1 + rand() % (10 - 1 + 1);
         correctAnswer = num1 + num2;

         //display addition problem and get user's answer
         cout << "What is the sum of " << num1
              << " + " << num2 << "? ";
         cin >> userAnswer;

         //determine whether user's answer is correct
         if (userAnswer == correctAnswer)
              cout << "Correct!";
```

required for the time function

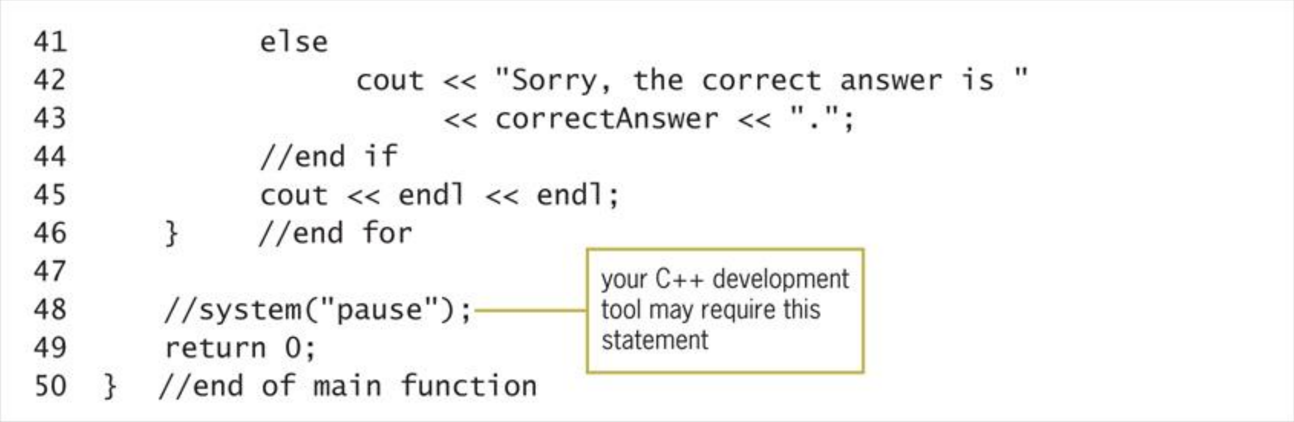
your C++ development tool may require this directive

uses the srand and time functions

uses the rand function

Figure 9-14 Random addition problems program

Generating Random Integers (cont’d.)

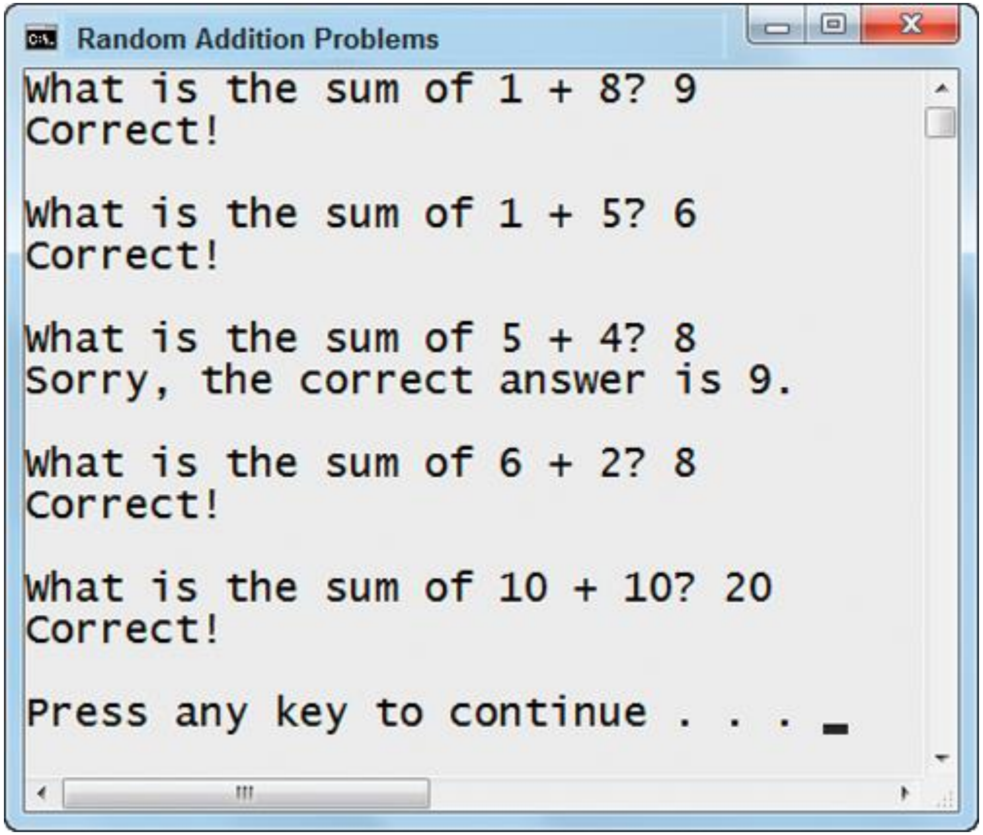


```
         else
              cout << "Sorry, the correct answer is "
                   << correctAnswer << ".";
         //end if
         cout << endl << endl;
    }    //end for

    //system("pause");
    return 0;
}   //end of main function
```

your C++ development tool may require this statement

Figure 9-14 Random addition problems program (cont’d.)

Generating Random Integers (cont'd.)



```
What is the sum of 1 + 8? 9
Correct!

What is the sum of 1 + 5? 6
Correct!

What is the sum of 5 + 4? 8
Sorry, the correct answer is 9.

What is the sum of 6 + 2? 8
Correct!

What is the sum of 10 + 10? 20
Correct!

Press any key to continue . . . _
```

Figure 9-15 Sample run of random addition problems program

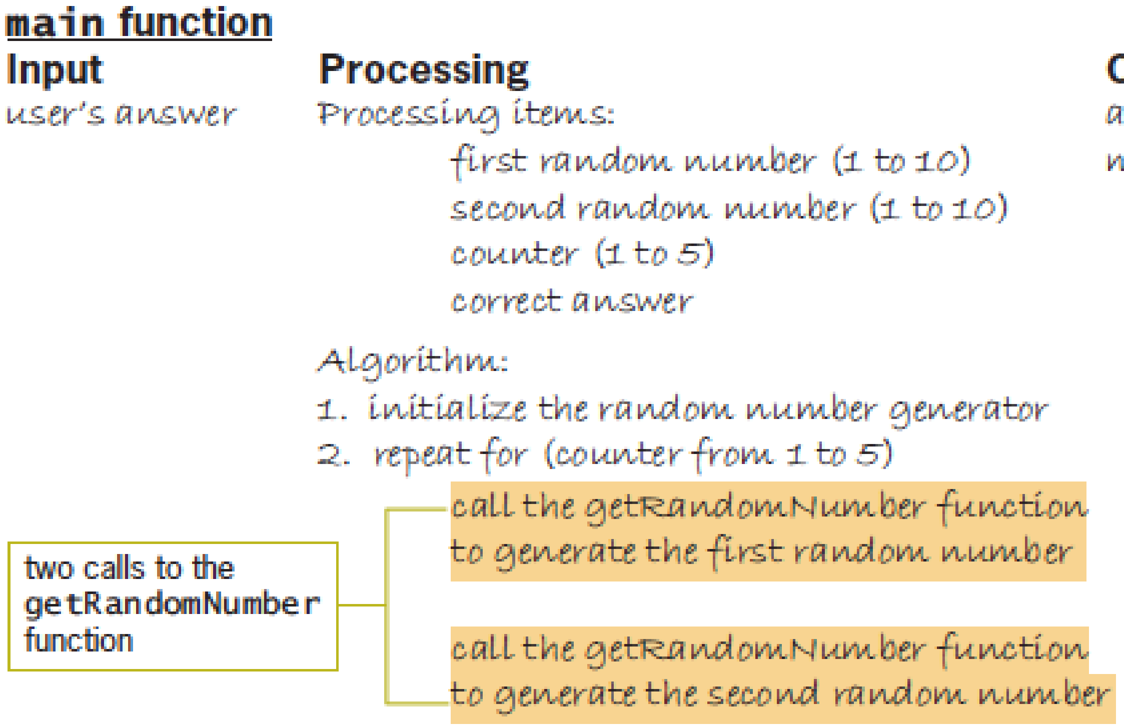
Creating Program-Defined Value-Returning Functions

- A program-defined value-returning function definition is composed of a header and a body
- Header (first line) contains return data type, name of function, and an optional *parameterList*
  - Rules for function names are same as for variables
  - Good idea to use meaningful names that describe function's purpose
  - Memory locations in *parameterList* are called **formal parameters**
    - Each stores an item of information passed to the function when it is called

Creating Program-Defined Value-Returning Functions (cont'd.)

- Function body contains instructions for performing the function's assigned task
- Surrounded by braces (`{ }`)
- Last statement is usually the **`return` statement**
  - Returns one value (must match return data type in function header)
- After `return` statement is processed, program execution continues in calling function
- Good idea to include comment (such as `//end of` *functionName*) to mark end of function

Creating Program-Defined Value-Returning Functions (cont'd.)



main function

Input	Processing	Output
user's answer	Processing items: first random number (1 to 10) second random number (1 to 10) counter (1 to 5) correct answer	addition problem message

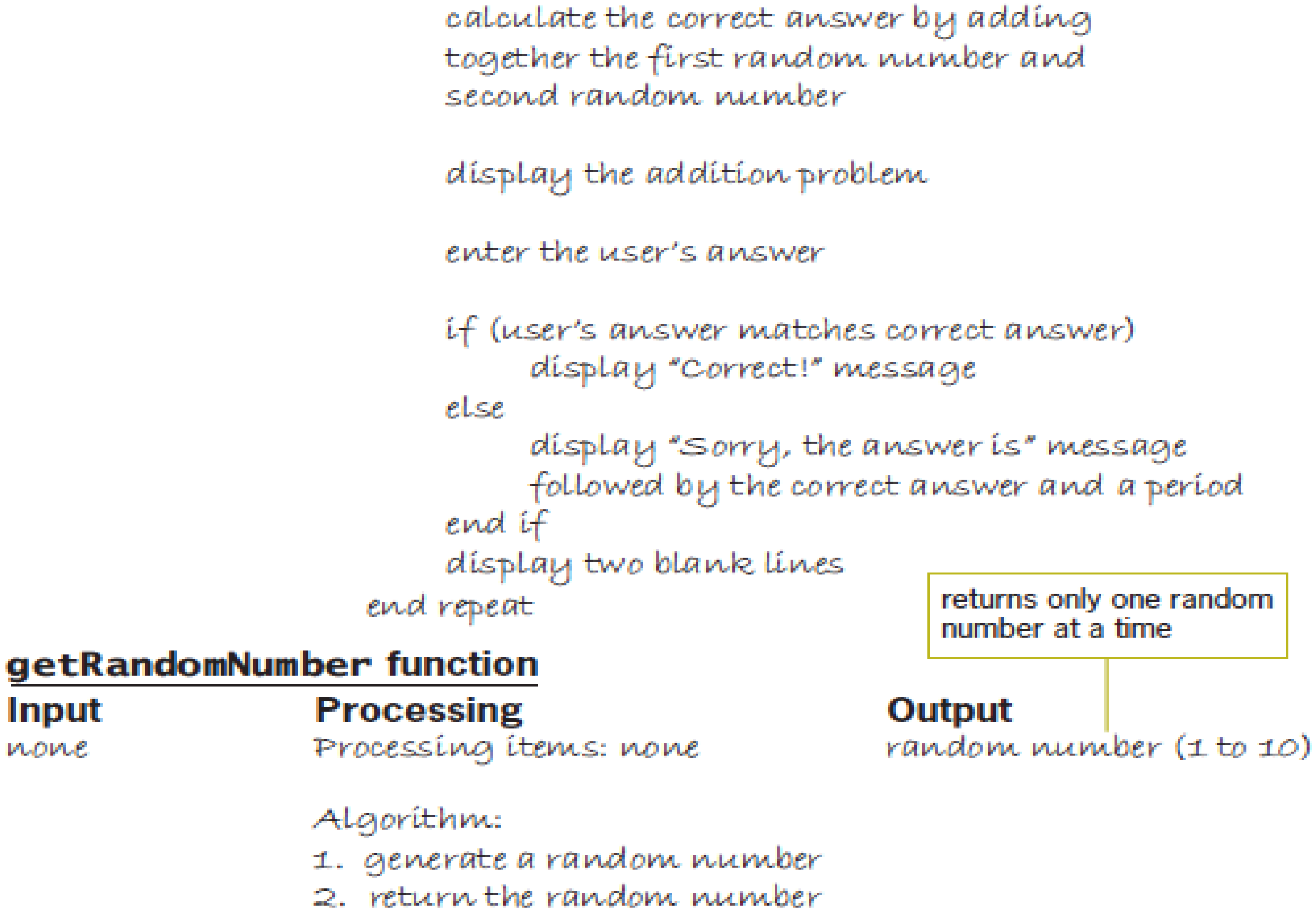
Algorithm:

```
1. initialize the random number generator
2. repeat for (counter from 1 to 5)
       call the getRandomNumber function
       to generate the first random number

       call the getRandomNumber function
       to generate the second random number
```

two calls to the getRandomNumber function

Figure 9-16 IPO charts for modified random addition problems program

Creating Program-Defined Value-Returning Functions (cont'd.)



```
        calculate the correct answer by adding
        together the first random number and
        second random number

        display the addition problem

        enter the user's answer

        if (user's answer matches correct answer)
            display "Correct!" message
        else
            display "Sorry, the answer is" message
            followed by the correct answer and a period
        end if
        display two blank lines
    end repeat
```

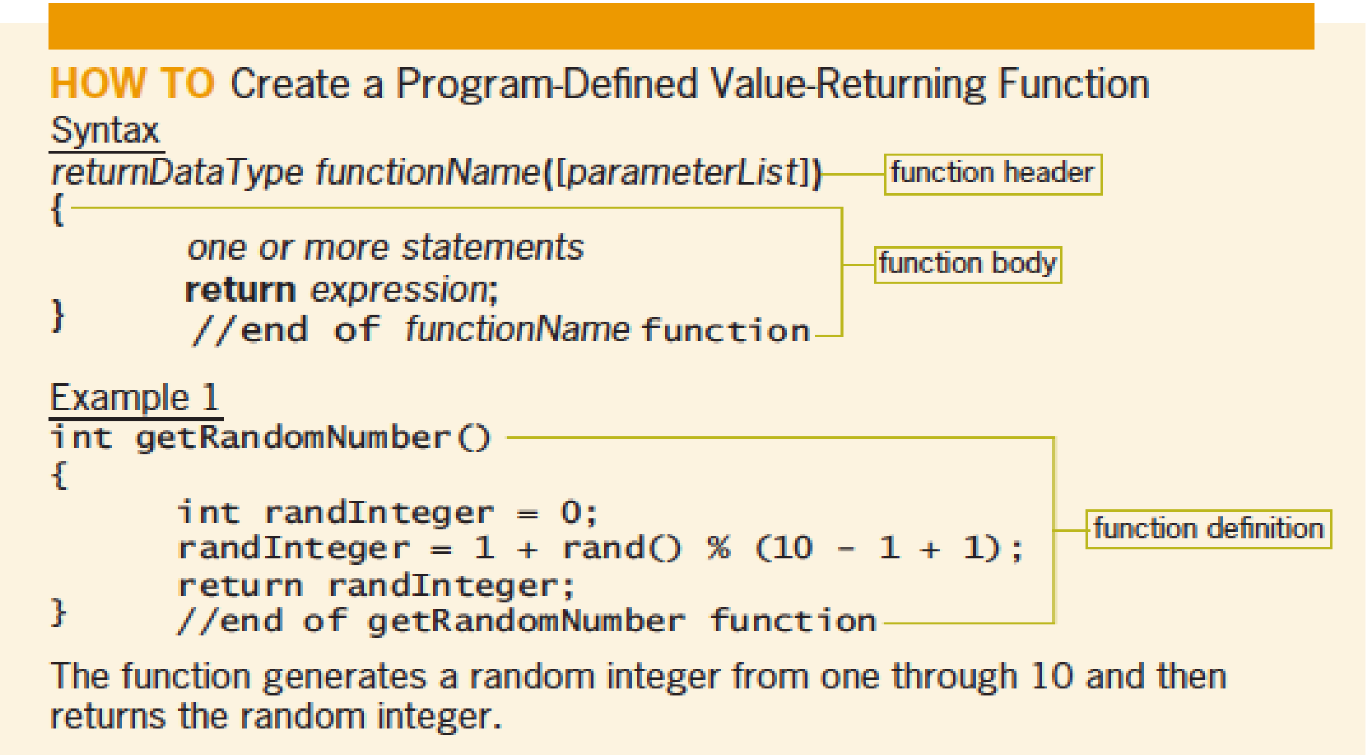
returns only one random number at a time

getRandomNumber function

Input	Processing	Output
none	Processing items: none Algorithm: 1. generate a random number 2. return the random number	random number (1 to 10)

Figure 9-16 IPO charts for modified random addition problems program (cont'd.)

Creating Program-Defined Value-Returning Functions (cont'd.)



HOW TO Create a Program-Defined Value-Returning Function

Syntax

```
returnDataType functionName([parameterList])
{
        one or more statements
        return expression;
}       //end of functionName function
```

function header
function body

Example 1

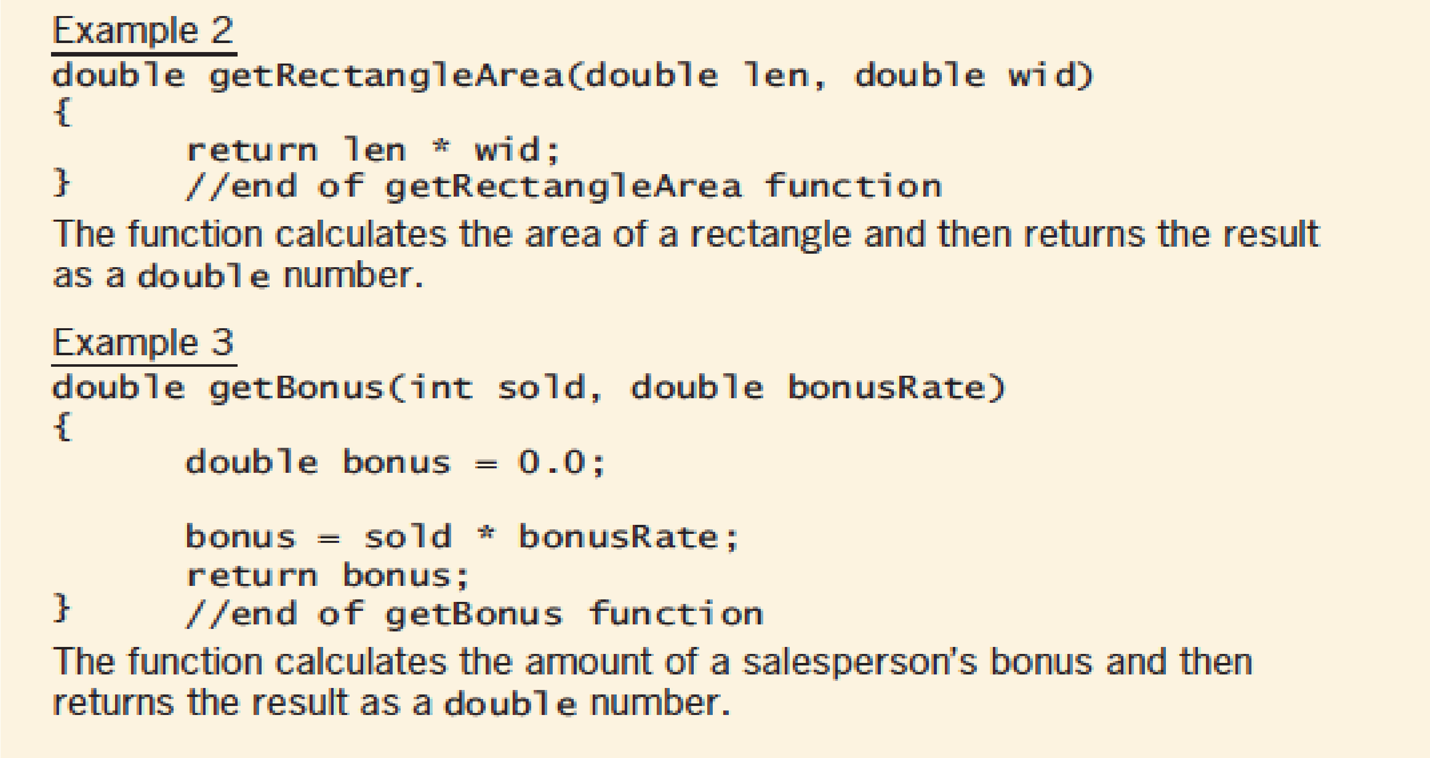
```
int getRandomNumber()
{
        int randInteger = 0;
        randInteger = 1 + rand() % (10 - 1 + 1);
        return randInteger;
}       //end of getRandomNumber function
```

function definition

The function generates a random integer from one through 10 and then returns the random integer.

Figure 9-17 How to create a program-defined value-returning function

Creating Program-Defined Value-Returning Functions (cont'd.)



Example 2

```
double getRectangleArea(double len, double wid)
{
      return len * wid;
}     //end of getRectangleArea function
```

The function calculates the area of a rectangle and then returns the result as a `double` number.

Example 3

```
double getBonus(int sold, double bonusRate)
{
      double bonus = 0.0;

      bonus = sold * bonusRate;
      return bonus;
}     //end of getBonus function
```

The function calculates the amount of a salesperson's bonus and then returns the result as a `double` number.

Figure 9-17 How to create a program-defined value-returning function (cont'd.)

Calling a Function

- A function must be called (invoked) to perform its task
- `main` is automatically called when program is run
- Other functions must be called by a statement
- Syntax for calling a function:
  functionName`(`[*argumentList*]`);`
  - *argumentList* is list of actual arguments (if any)
  - An actual argument can be a variable, named constant, literal constant, or keyword

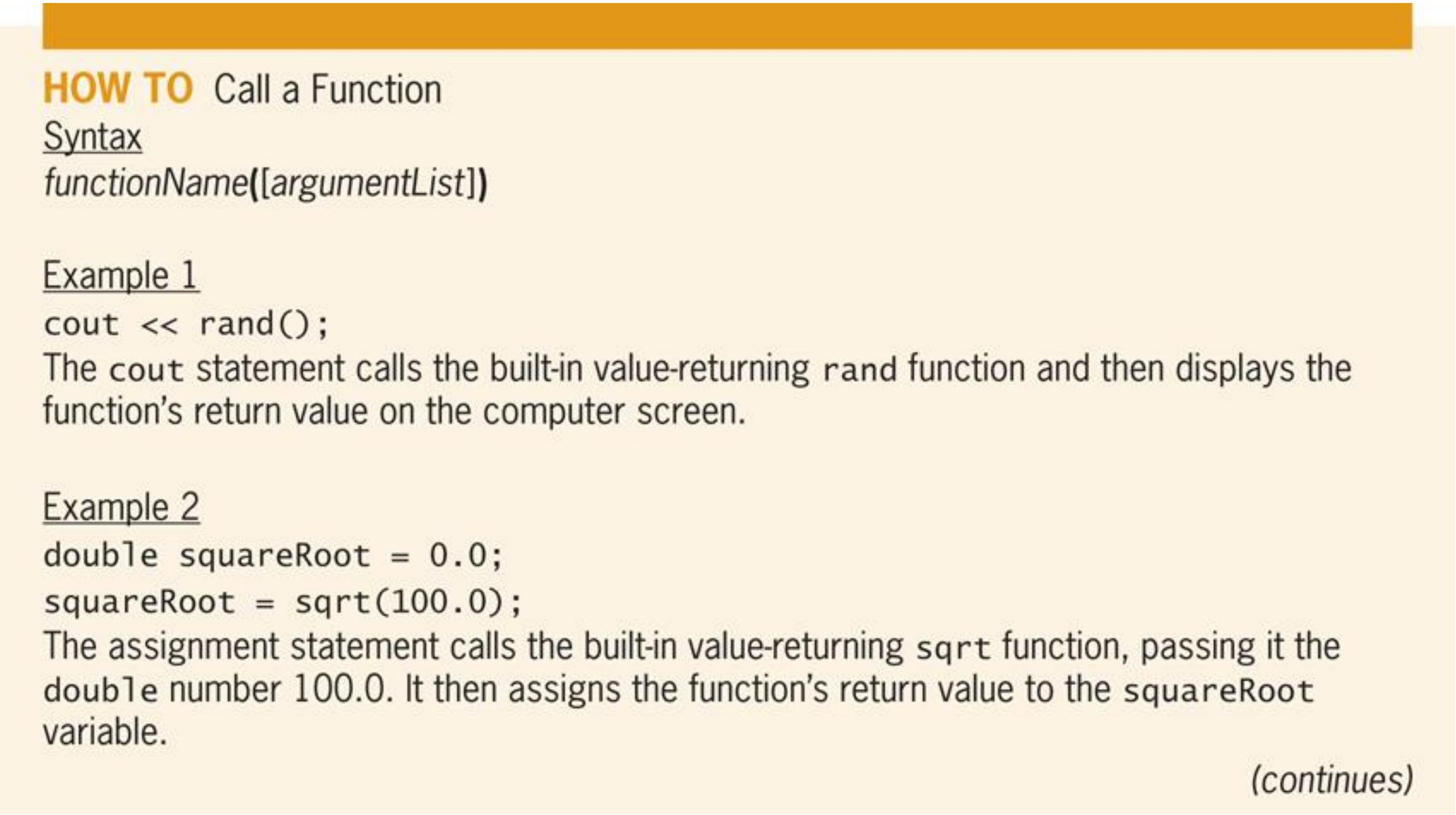
Calling a Function (cont'd.)

- Value-returning functions are typically called from statements that:
  - Assign the return value to a variable
  - Use the return value in a calculation or comparison
  - Display the return value
- A call to a void function is an independent statement because void functions do not return values

Calling a Function (cont'd.)

- C++ allows you to pass either a variable's value or its address to a function
- Passing a variable's value is referred to as **passing *by value***
- Passing a variable's address is referred to as **passing *by reference***
- Default is passing *by value*
- Number, data type, and ordering of actual arguments must match the formal parameters in function header
  - Names do not need to match (different names are better)

Calling a Function (cont'd.)



HOW TO Call a Function

Syntax

*functionName***(**[*argumentList*]**)**

Example 1

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

The `cout` statement calls the built-in value-returning `rand` function and then displays the function's return value on the computer screen.

Example 2

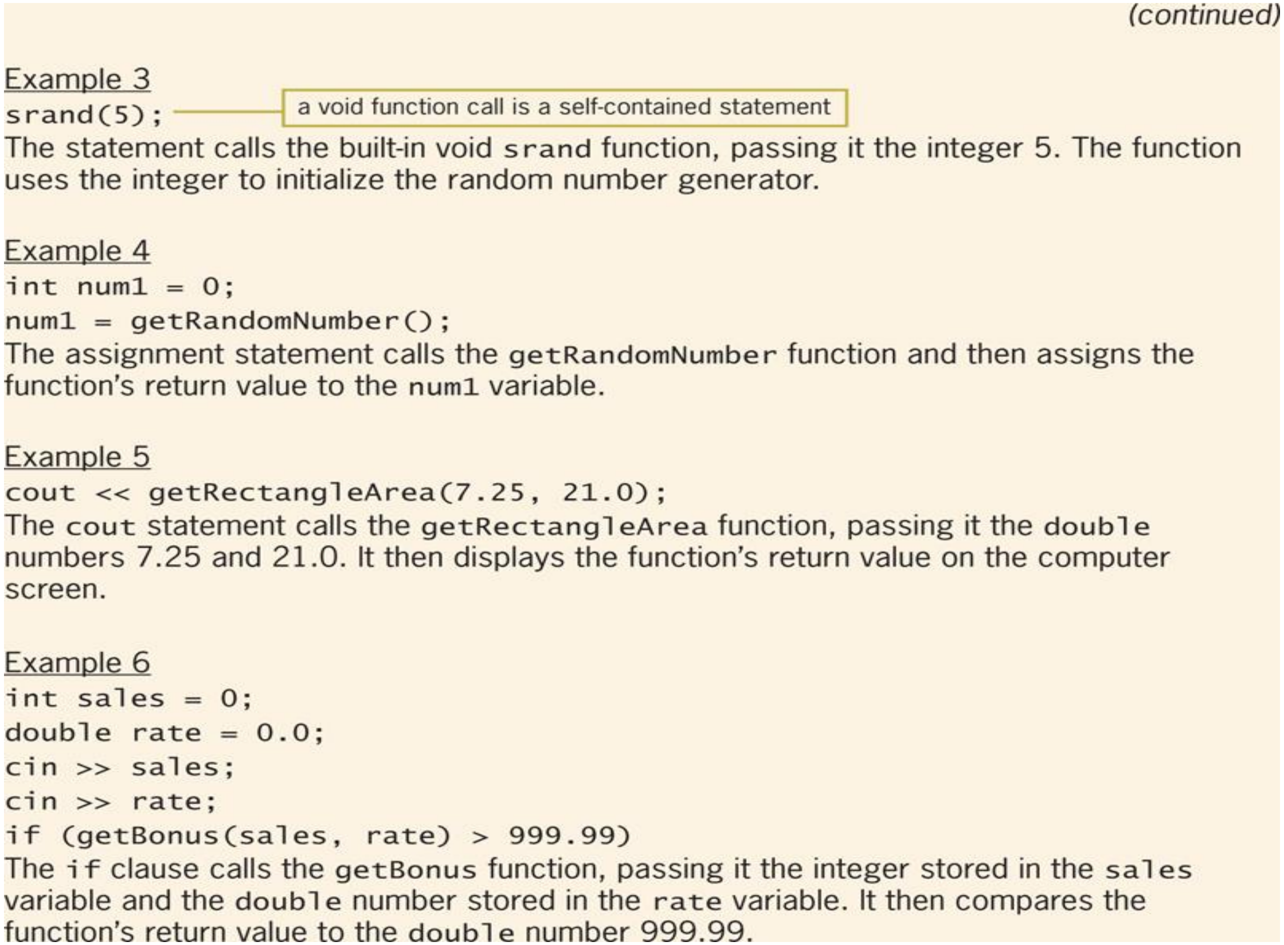
```
double squareRoot = 0.0;
squareRoot = sqrt(100.0);
```

The assignment statement calls the built-in value-returning `sqrt` function, passing it the `double` number 100.0. It then assigns the function's return value to the `squareRoot` variable.

(continues)

Figure 9-18 How to call a function

Calling a Function (cont'd.)



(continued)

Example 3

```
srand(5);
```

a void function call is a self-contained statement

The statement calls the built-in void `srand` function, passing it the integer 5. The function uses the integer to initialize the random number generator.

Example 4

```
int num1 = 0;
num1 = getRandomNumber();
```

The assignment statement calls the `getRandomNumber` function and then assigns the function's return value to the `num1` variable.

Example 5

```
cout << getRectangleArea(7.25, 21.0);
```

The `cout` statement calls the `getRectangleArea` function, passing it the `double` numbers 7.25 and 21.0. It then displays the function's return value on the computer screen.

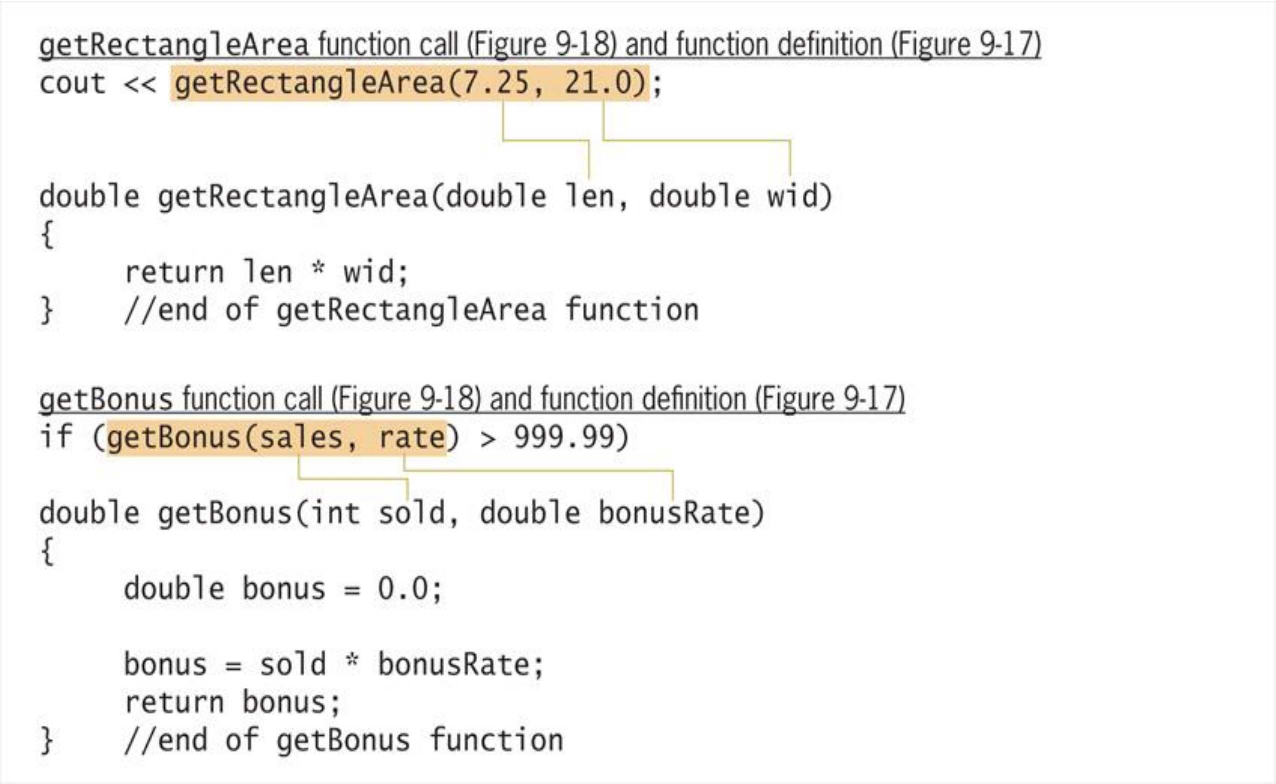
Example 6

```
int sales = 0;
double rate = 0.0;
cin >> sales;
cin >> rate;
if (getBonus(sales, rate) > 999.99)
```

The `if` clause calls the `getBonus` function, passing it the integer stored in the `sales` variable and the `double` number stored in the `rate` variable. It then compares the function's return value to the `double` number 999.99.

Figure 9-18 How to call a function (cont'd.)

Calling a Function (cont'd.)



getRectangleArea function call (Figure 9-18) and function definition (Figure 9-17)

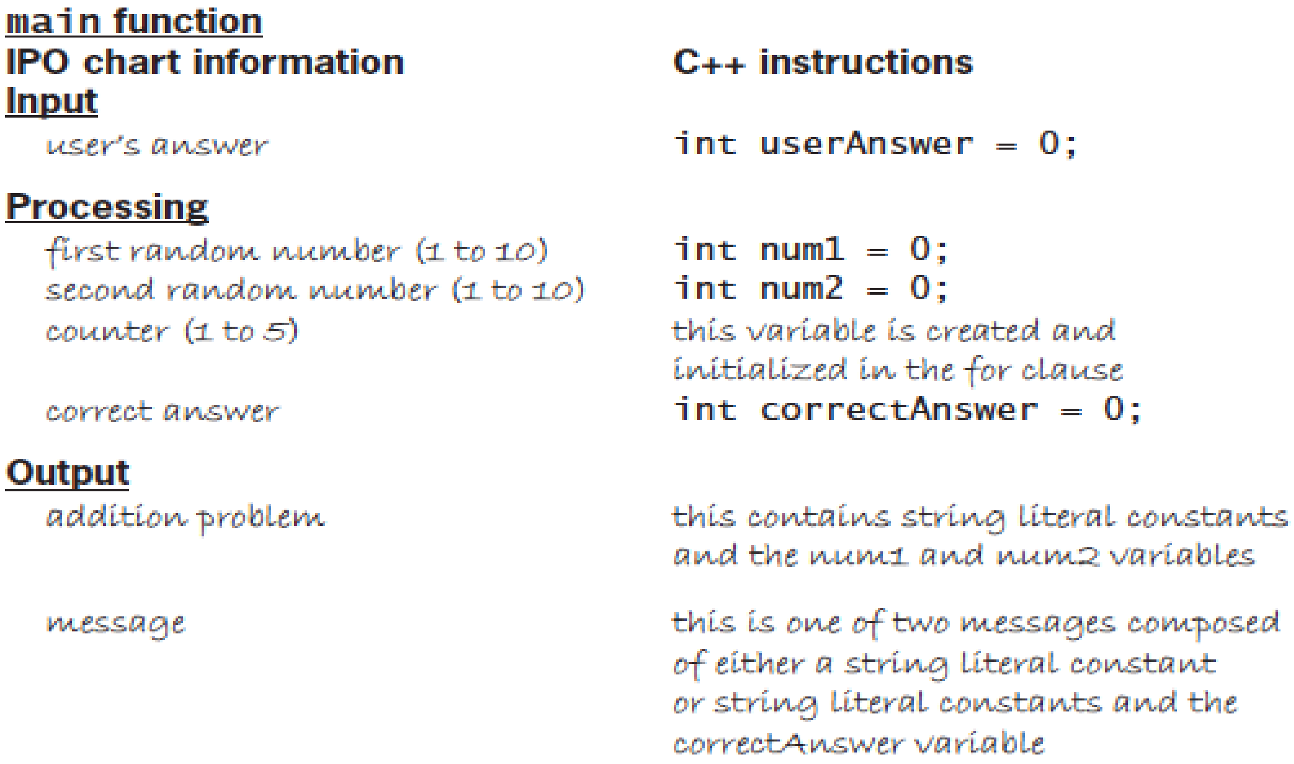
```
cout << getRectangleArea(7.25, 21.0);


double getRectangleArea(double len, double wid)
{
      return len * wid;
}     //end of getRectangleArea function
```

getBonus function call (Figure 9-18) and function definition (Figure 9-17)

```
if (getBonus(sales, rate) > 999.99)

double getBonus(int sold, double bonusRate)
{
      double bonus = 0.0;

      bonus = sold * bonusRate;
      return bonus;
}     //end of getBonus function
```

Figure 9-19 Function calls and function definitions

Calling a Function (cont'd.)

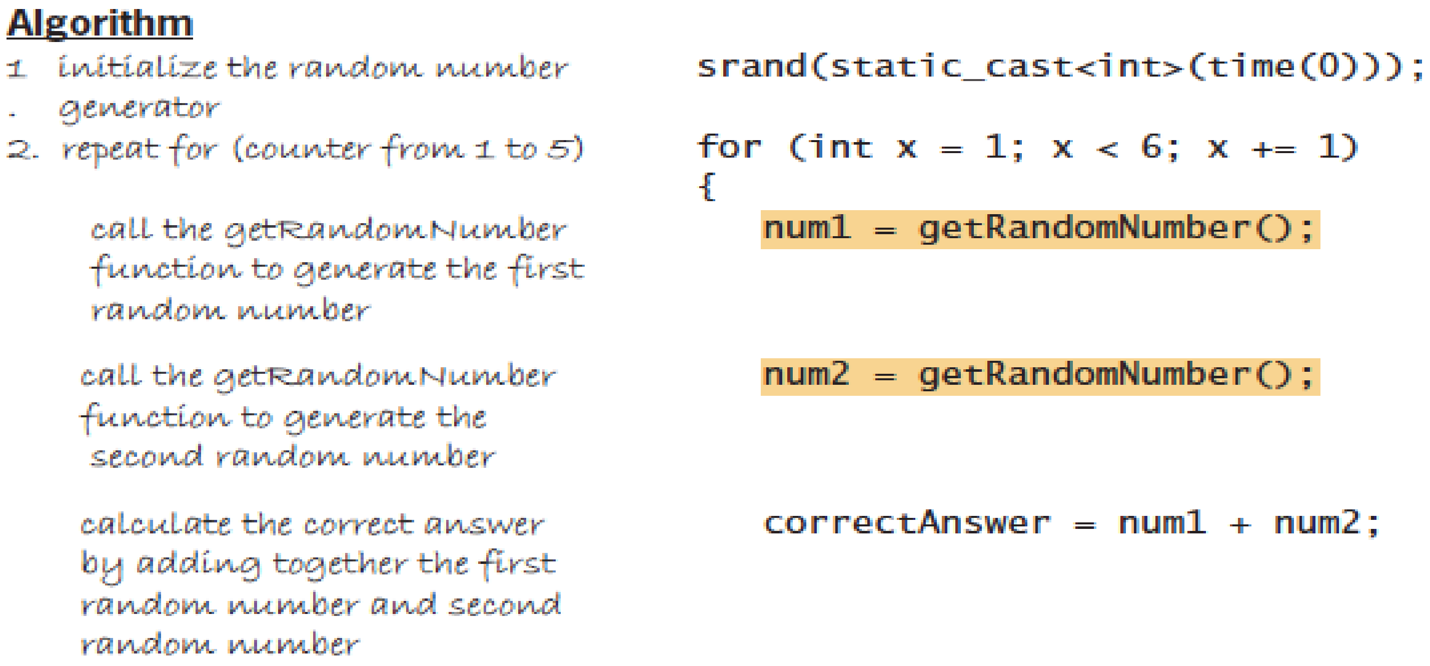


main function

IPO chart information	C++ instructions
Input	
user's answer	`int userAnswer = 0;`
Processing	
first random number (1 to 10)	`int num1 = 0;`
second random number (1 to 10)	`int num2 = 0;`
counter (1 to 5)	this variable is created and initialized in the for clause
correct answer	`int correctAnswer = 0;`
Output	
addition problem	this contains string literal constants and the num1 and num2 variables
message	this is one of two messages composed of either a string literal constant or string literal constants and the correctAnswer variable

Figure 9-20 IPO chart information and C++ instructions for the modified random addition problems program

Calling a Function (cont'd.)



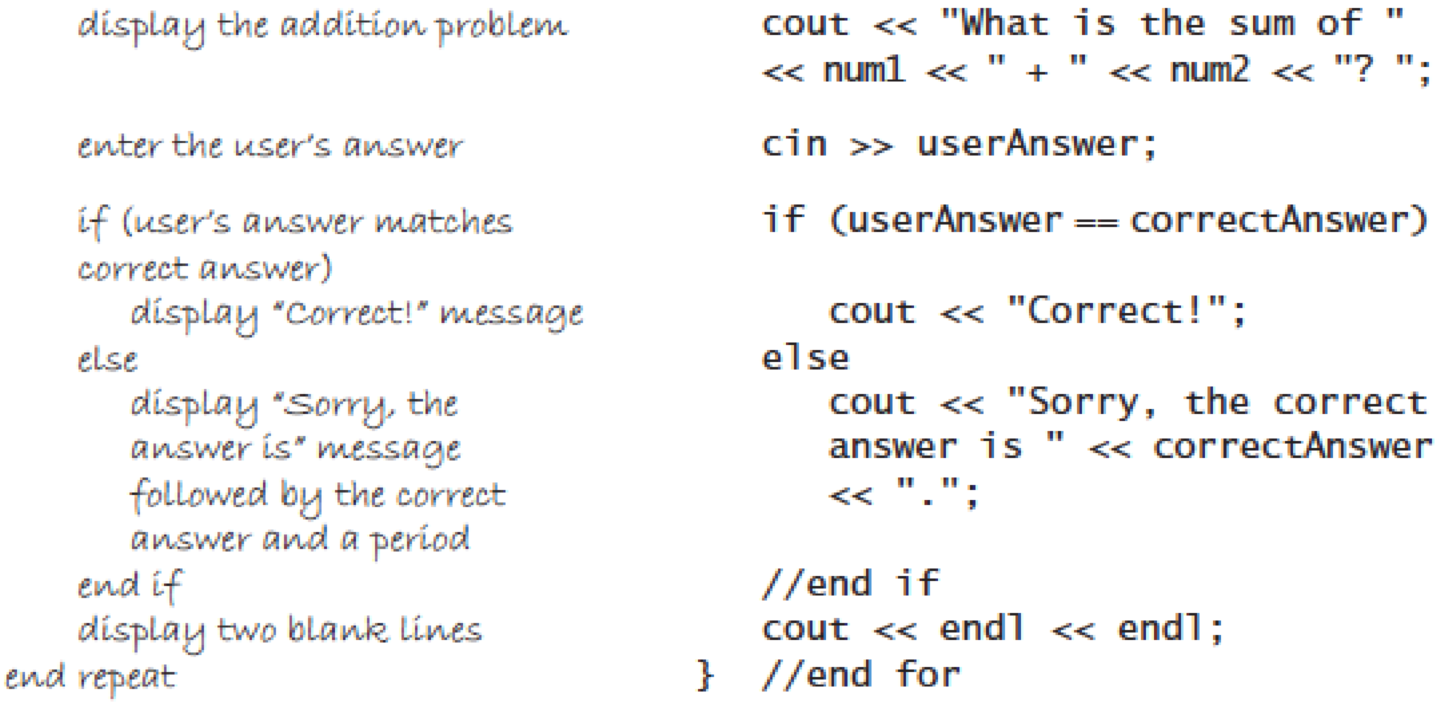
Algorithm	
1 initialize the random number . generator	`srand(static_cast<int>(time(0)));`

2. repeat for (counter from 1 to 5)	`for (int x = 1; x < 6; x += 1)`
	`{`

call the getRandomNumber function to generate the first random number	`num1 = getRandomNumber();`
call the getRandomNumber function to generate the second random number	`num2 = getRandomNumber();`
calculate the correct answer by adding together the first random number and second random number	`correctAnswer = num1 + num2;`

Figure 9-20 IPO chart information and C++ instructions for the modified random addition problems program (cont'd.)

Calling a Function (cont’d.)

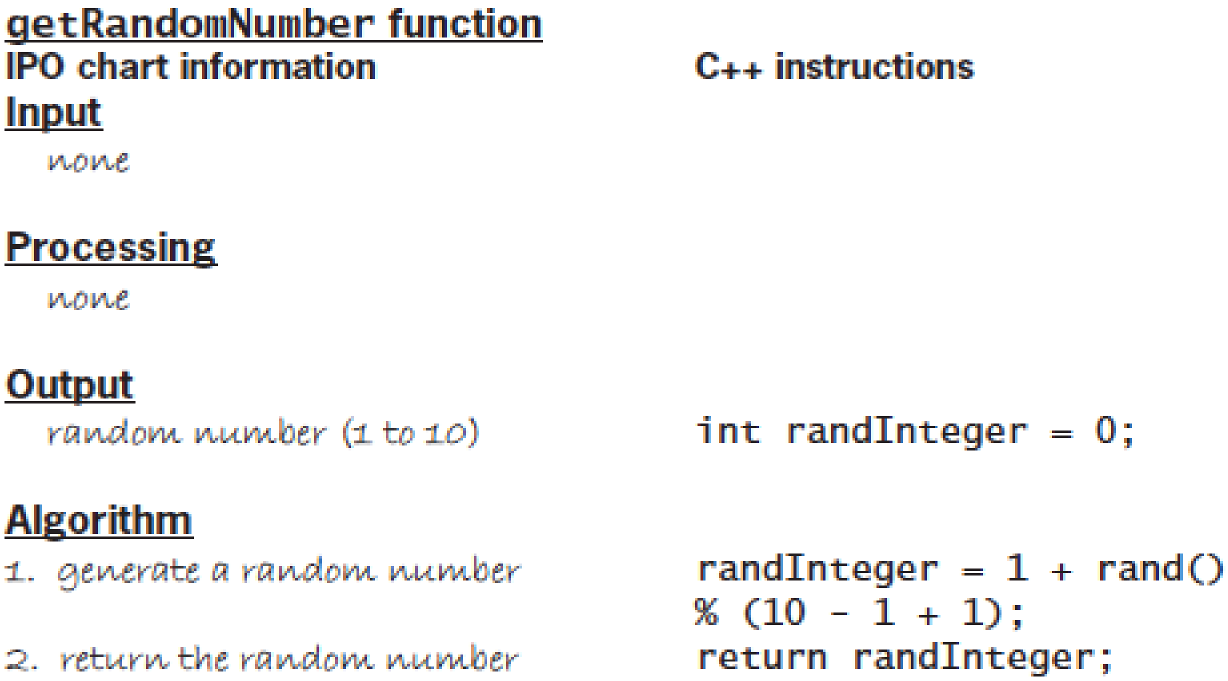


```
    display the addition problem

    enter the user's answer

    if (user's answer matches
    correct answer)
       display "Correct!" message
    else
       display "Sorry, the
       answer is" message
       followed by the correct
       answer and a period
    end if
    display two blank lines
end repeat
```

```
    cout << "What is the sum of "
    << num1 << " + " << num2 << "? ";

    cin >> userAnswer;

    if (userAnswer == correctAnswer)

       cout << "Correct!";
    else
       cout << "Sorry, the correct
       answer is " << correctAnswer
       << ".";

    //end if
    cout << endl << endl;
}   //end for
```

Figure 9-20 IPO chart information and C++ instructions
for the modified random addition problems program (cont’d.)

Calling a Function (cont’d.)



getRandomNumber function

IPO chart information	**C++ instructions**
Input	
none	
Processing	
none	
Output	
random number (1 to 10)	`int randInteger = 0;`
Algorithm	

1. generate a random number	`randInteger = 1 + rand()`
	`% (10 - 1 + 1);`

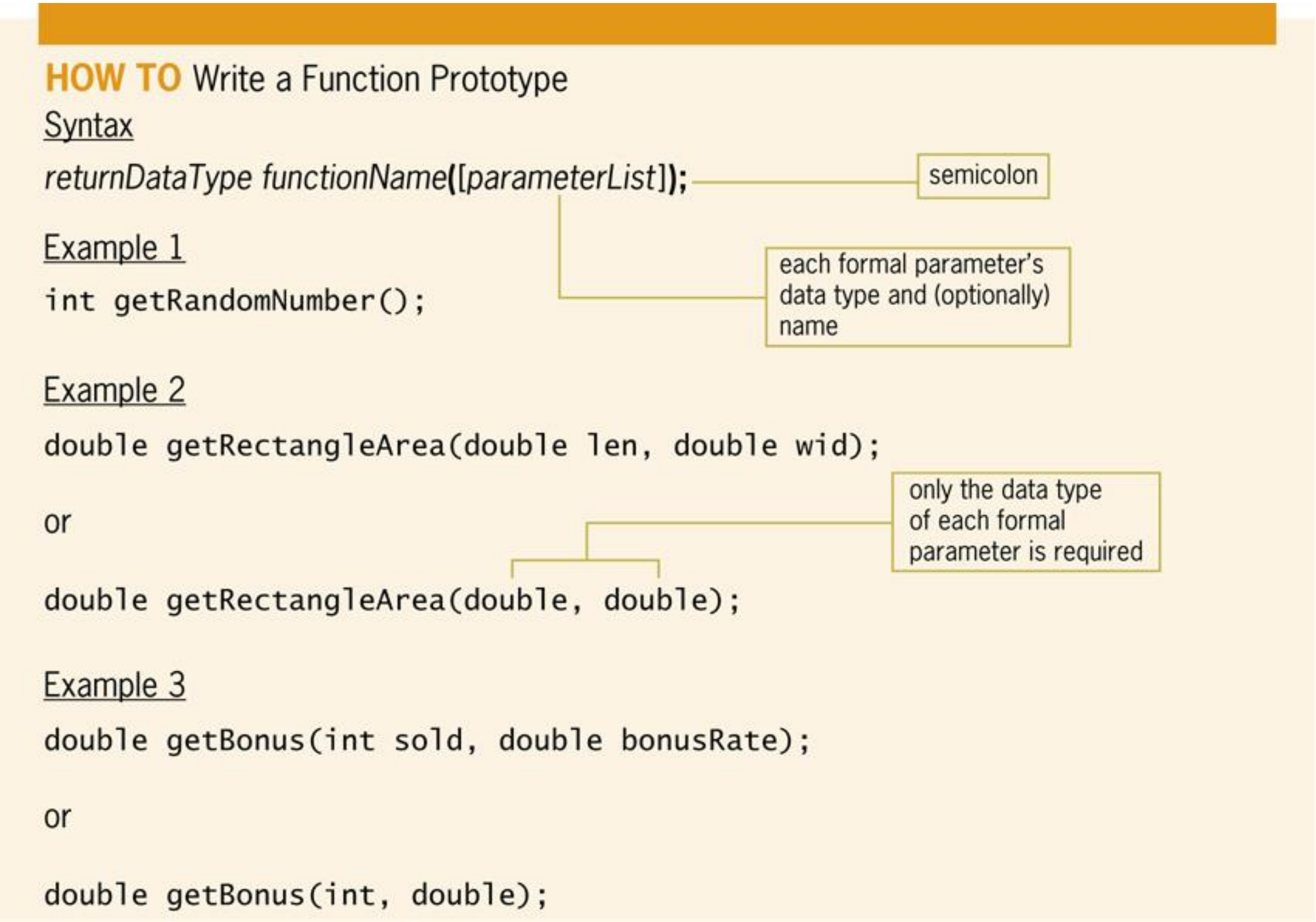
2. return the random number	`return randInteger;`

Figure 9-20 IPO chart information and C++ instructions for the modified random addition problems program (cont’d.)

Function Prototypes

- When a function definition appears below the `main` function, you must enter a function prototype above the `main` function
- A **function prototype** is a statement that specifies the function’s name, data type of its return value, and data type of each of its formal parameters (if any)
  - Names for the formal parameters are not required
- Programmers usually place function prototypes at beginning of program, after the `#include` directives

Function Prototypes (cont'd.)



HOW TO Write a Function Prototype

Syntax

returnDataType functionName(**[***parameterList***]**)**;**

semicolon

each formal parameter's data type and (optionally) name

Example 1

```
int getRandomNumber();
```

Example 2

```
double getRectangleArea(double len, double wid);
```

or

```
double getRectangleArea(double, double);
```

only the data type of each formal parameter is required

Example 3

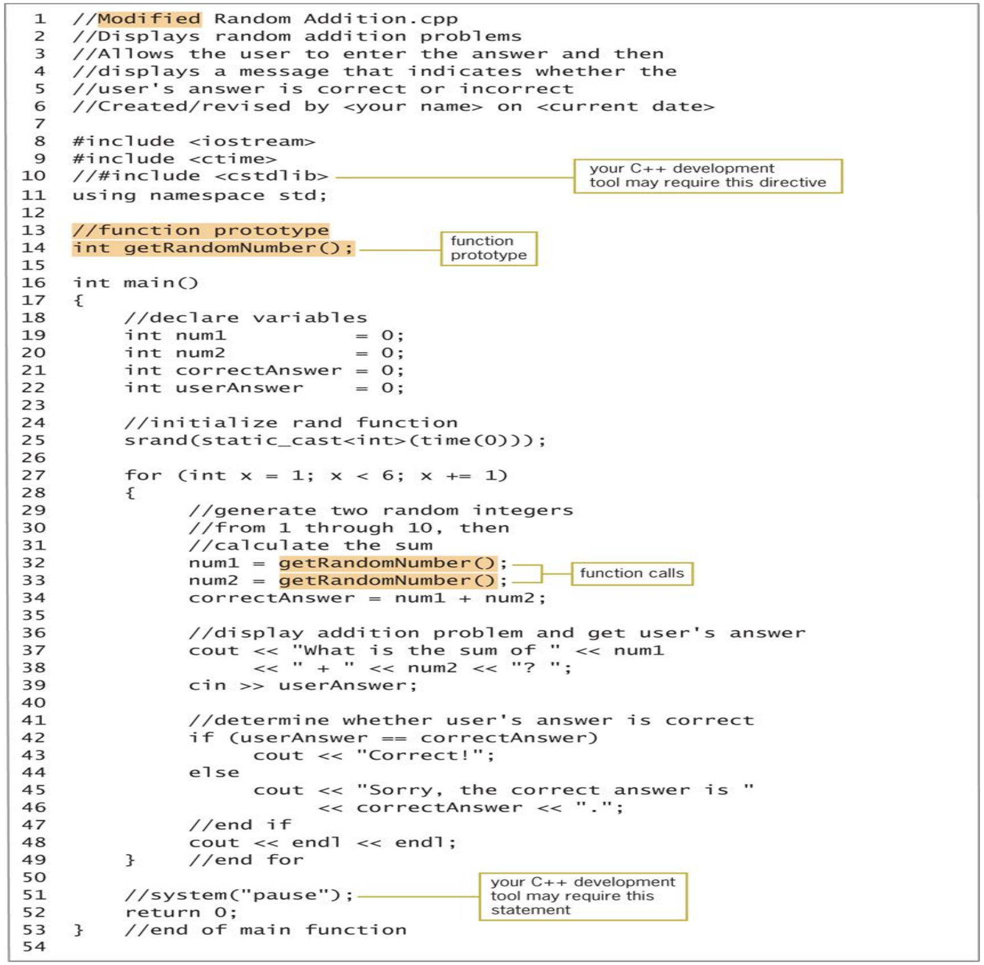
```
double getBonus(int sold, double bonusRate);
```

or

```
double getBonus(int, double);
```

Figure 9-21 How to write a function prototype

Function Prototypes (cont’d.)



```
//Modified Random Addition.cpp
//Displays random addition problems
//Allows the user to enter the answer and then
//displays a message that indicates whether the
//user's answer is correct or incorrect
//Created/revised by <your name> on <current date>

#include <iostream>
#include <ctime>
//#include <cstdlib>
using namespace std;

//function prototype
int getRandomNumber();

int main()
{
    //declare variables
    int num1          = 0;
    int num2          = 0;
    int correctAnswer = 0;
    int userAnswer    = 0;

    //initialize rand function
    srand(static_cast<int>(time(0)));

    for (int x = 1; x < 6; x += 1)
    {
         //generate two random integers
         //from 1 through 10, then
         //calculate the sum
         num1 = getRandomNumber();
         num2 = getRandomNumber();
         correctAnswer = num1 + num2;

         //display addition problem and get user's answer
         cout << "What is the sum of " << num1
              << " + " << num2 << "? ";
         cin >> userAnswer;

         //determine whether user's answer is correct
         if (userAnswer == correctAnswer)
              cout << "Correct!";
         else
              cout << "Sorry, the correct answer is "
                   << correctAnswer << ".";
         //end if
         cout << endl << endl;
    }    //end for

    //system("pause");
    return 0;
}   //end of main function

```

your C++ development tool may require this directive

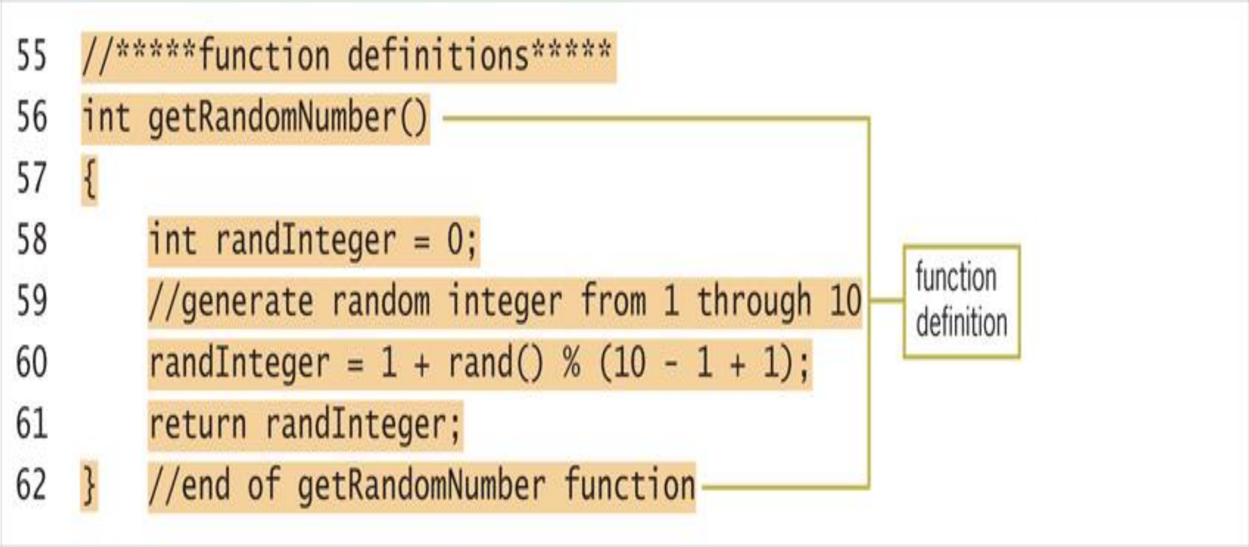
function prototype

function calls

your C++ development tool may require this statement

Figure 9-22 Modified random addition problems program

Function Prototypes (cont’d.)



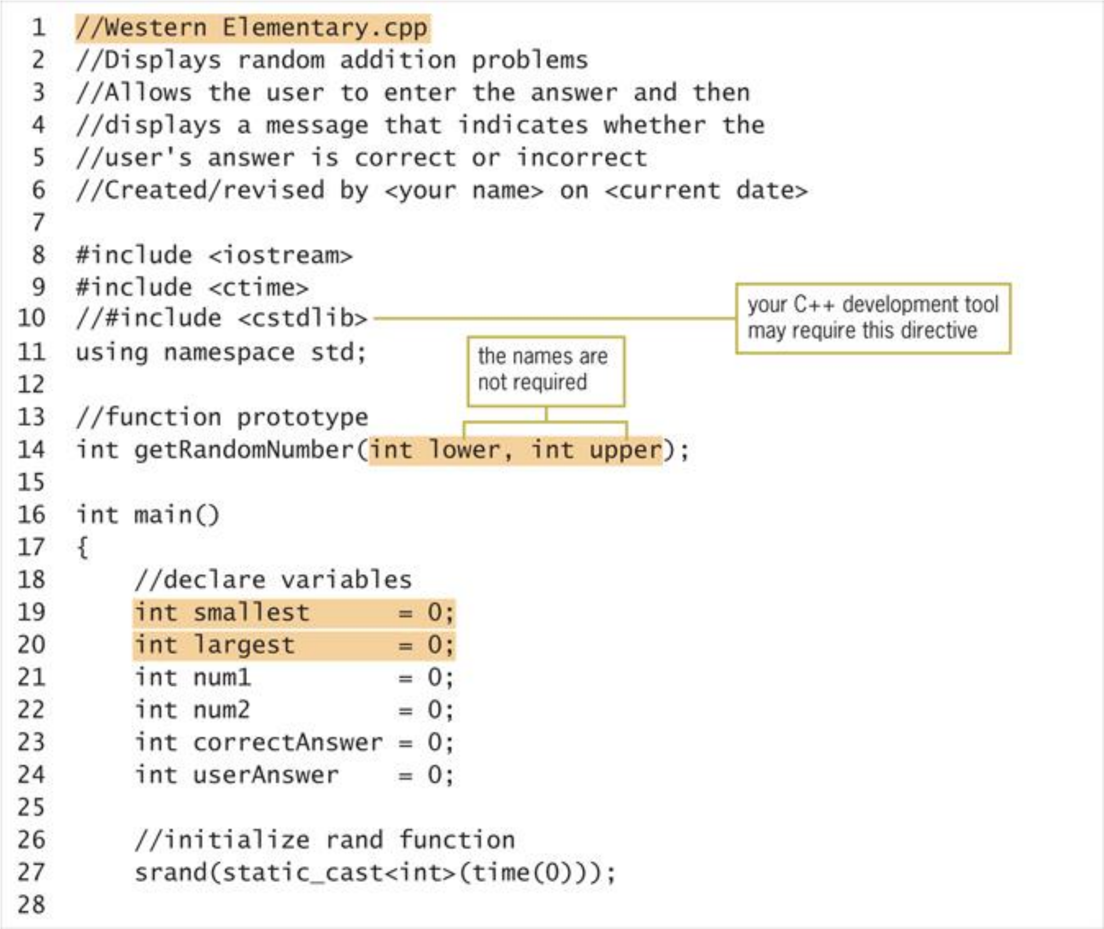
```
//*****function definitions*****
int getRandomNumber()
{
    int randInteger = 0;
    //generate random integer from 1 through 10
    randInteger = 1 + rand() % (10 - 1 + 1);
    return randInteger;
}   //end of getRandomNumber function
```

Figure 9-22 Modified random addition problems program (cont’d.)

The Western Elementary School Program

- Modification of the random addition problems program
- User should have the option to specify the range of random numbers that the program generates
- Program’s tasks are divided up into functions

The Western Elementary School Program (cont'd.)



```
//Western Elementary.cpp
//Displays random addition problems
//Allows the user to enter the answer and then
//displays a message that indicates whether the
//user's answer is correct or incorrect
//Created/revised by <your name> on <current date>

#include <iostream>
#include <ctime>
//#include <cstdlib>
using namespace std;

//function prototype
int getRandomNumber(int lower, int upper);

int main()
{
    //declare variables
    int smallest      = 0;
    int largest       = 0;
    int num1          = 0;
    int num2          = 0;
    int correctAnswer = 0;
    int userAnswer    = 0;

    //initialize rand function
    srand(static_cast<int>(time(0)));

```

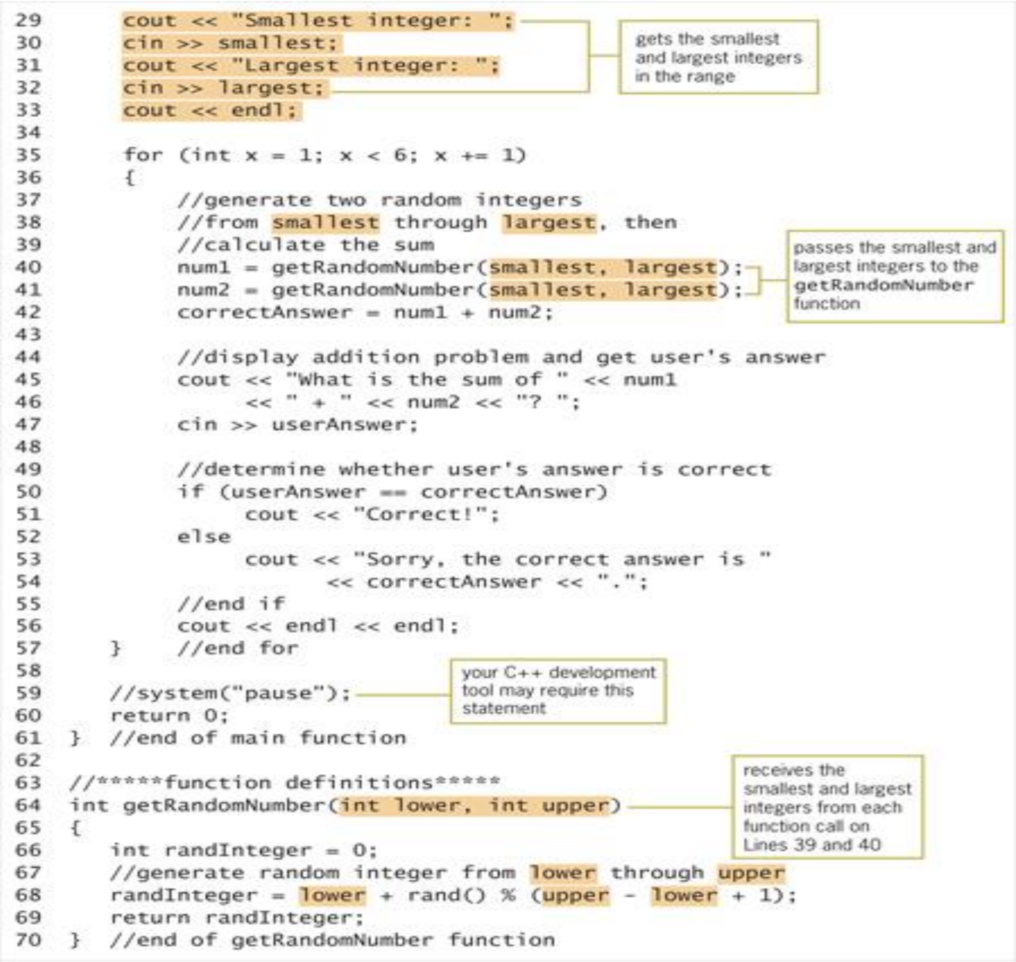
your C++ development tool may require this directive

the names are not required

Figure 9-23 Western Elementary School program

The Western Elementary School Program (cont'd.)

passes the smallest and largest integers to the getRandomNumber function



```
29          cout << "Smallest integer: ";
30          cin >> smallest;
31          cout << "Largest integer: ";
32          cin >> largest;
33          cout << endl;
34
35          for (int x = 1; x < 6; x += 1)
36          {
37              //generate two random integers
38              //from smallest through largest, then
39              //calculate the sum
40              num1 = getRandomNumber(smallest, largest);
41              num2 = getRandomNumber(smallest, largest);
42              correctAnswer = num1 + num2;
43
44              //display addition problem and get user's answer
45              cout << "What is the sum of " << num1
46                   << " + " << num2 << "? ";
47              cin >> userAnswer;
48
49              //determine whether user's answer is correct
50              if (userAnswer == correctAnswer)
51                   cout << "Correct!";
52              else
53                   cout << "Sorry, the correct answer is "
54                          << correctAnswer << ".";
55              //end if
56              cout << endl << endl;
57         }    //end for
58
59         //system("pause");
60         return 0;
61    }    //end of main function
62
63    //*****function definitions*****
64    int getRandomNumber(int lower, int upper)
65    {
66         int randInteger = 0;
67         //generate random integer from lower through upper
68         randInteger = lower + rand() % (upper - lower + 1);
69         return randInteger;
70    }    //end of getRandomNumber function
```

gets the smallest and largest integers in the range

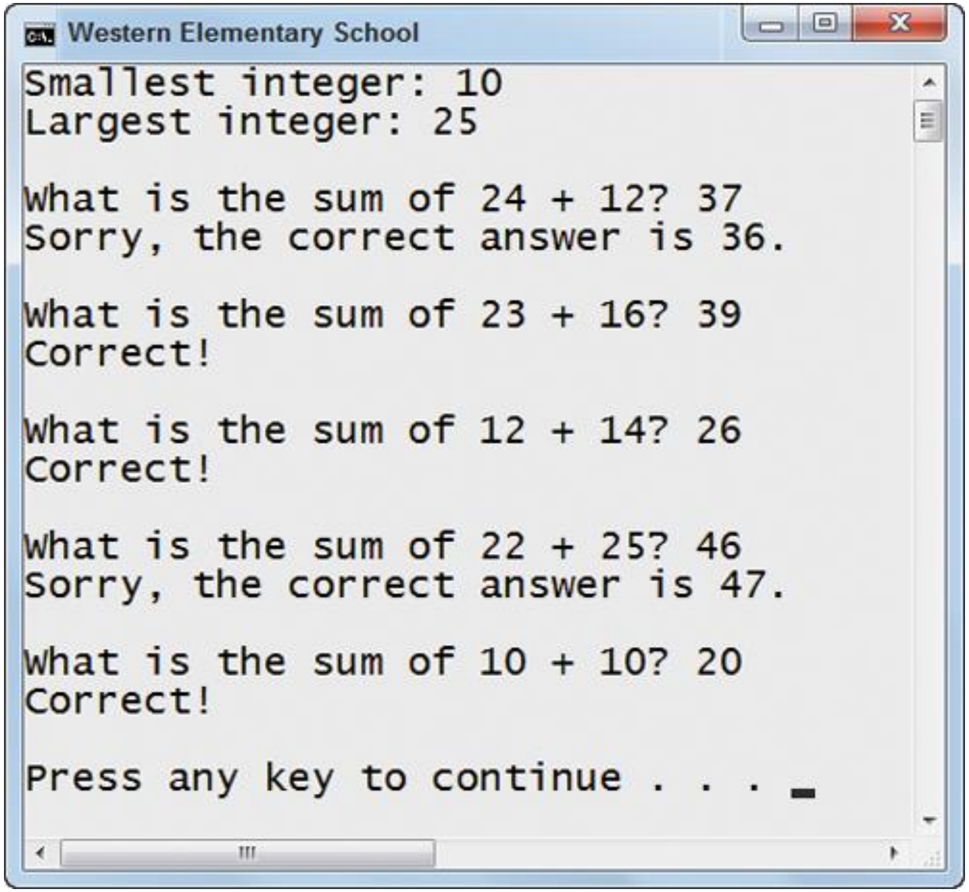
passes the smallest and largest integers to the getRandomNumber function

your C++ development tool may require this statement

receives the smallest and largest integers from each function call on Lines 39 and 40

Figure 9-23 Western Elementary School program (cont'd.)

The Western Elementary School Program (cont'd.)



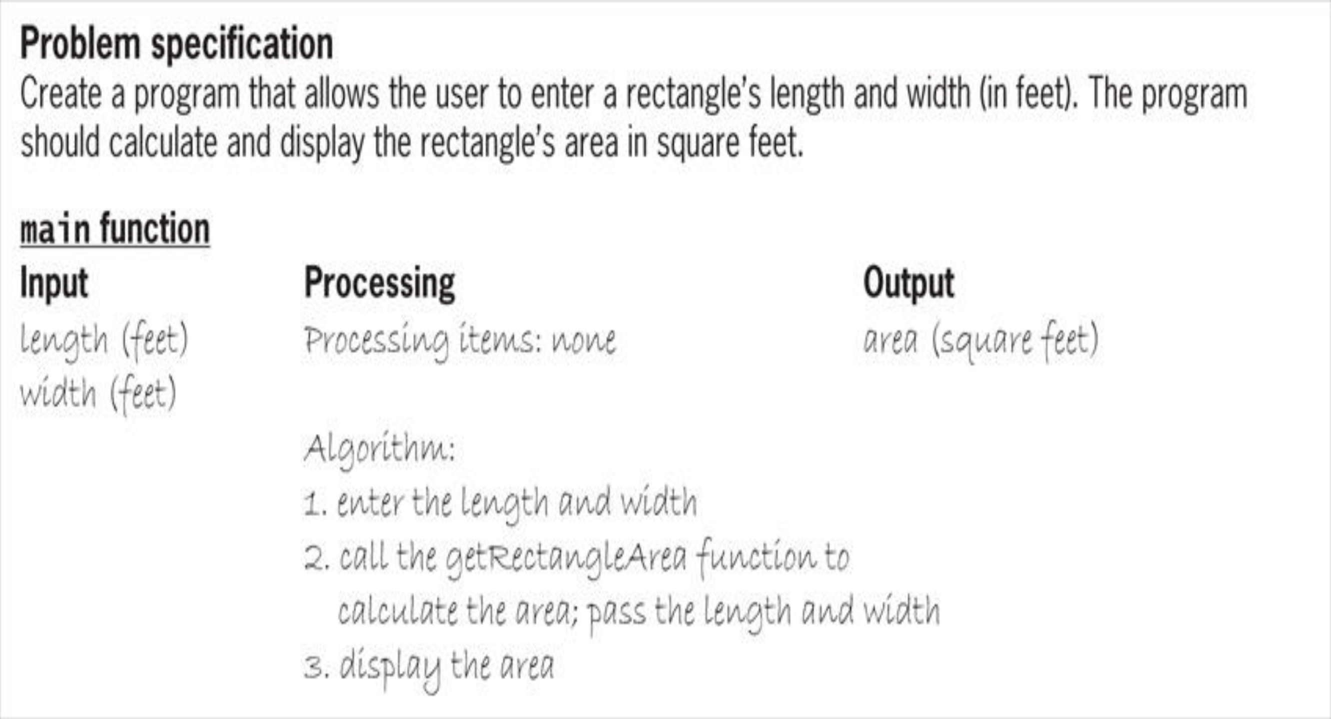
```
Smallest integer: 10
Largest integer: 25

What is the sum of 24 + 12? 37
Sorry, the correct answer is 36.

What is the sum of 23 + 16? 39
Correct!

What is the sum of 12 + 14? 26
Correct!

What is the sum of 22 + 25? 46
Sorry, the correct answer is 47.

What is the sum of 10 + 10? 20
Correct!

Press any key to continue . . .
```

Figure 9-24 Sample run of Western Elementary School program

The Area Calculator Program

- Program that uses a program-defined, value-returning function to calculate the area of a rectangle
- Input is rectangle's length and width measurements
- Program calculates area and then displays it on the screen

The Area Calculator Program (cont'd.)



Problem specification

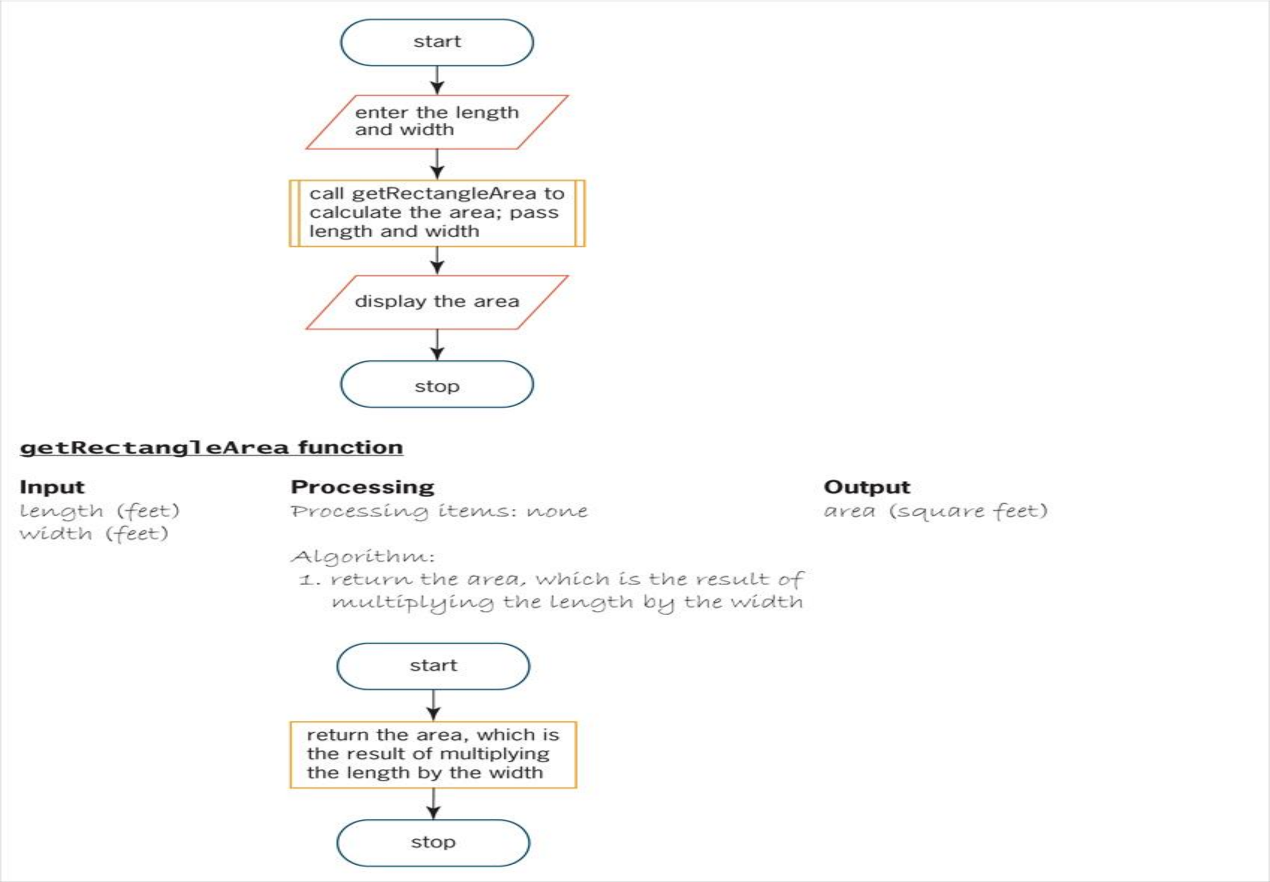
Create a program that allows the user to enter a rectangle's length and width (in feet). The program should calculate and display the rectangle's area in square feet.

main function

Input	Processing	Output
length (feet) width (feet)	Processing items: none Algorithm: 1. enter the length and width 2. call the getRectangleArea function to calculate the area; pass the length and width 3. display the area	area (square feet)

Figure 9-25 Problem specification and IPO charts for area calculator program

The Area Calculator Program (cont'd.)


start
enter the length and width
call getRectangleArea to calculate the area; pass length and width
display the area
stop

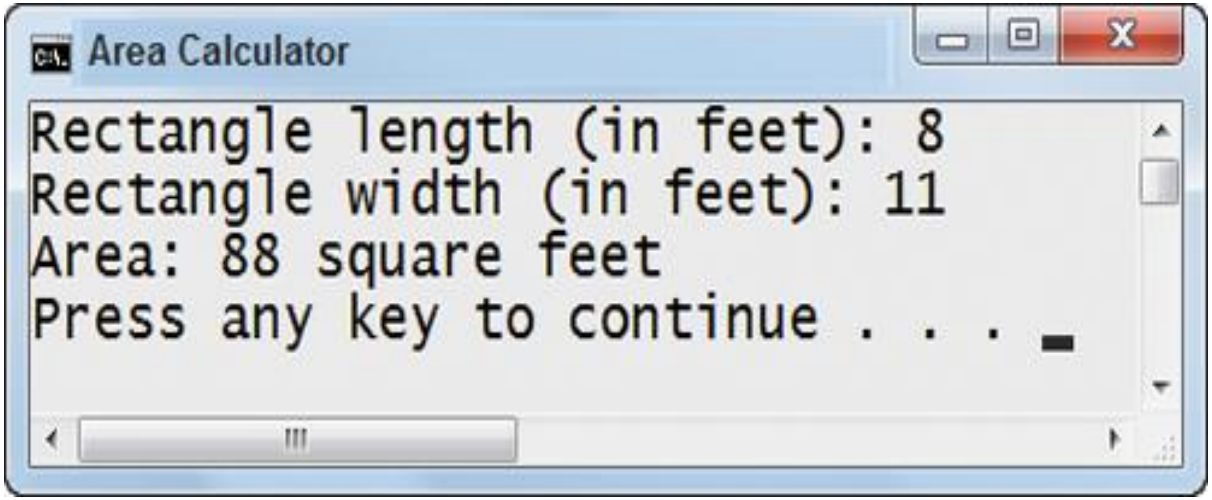
getRectangleArea function

Input	Processing	Output
length (feet) width (feet)	Processing items: none Algorithm: 1. return the area, which is the result of multiplying the length by the width	area (square feet)

start
return the area, which is the result of multiplying the length by the width
stop

Figure 9-25 Problem specification and IPO charts for the area calculator program (cont'd.)

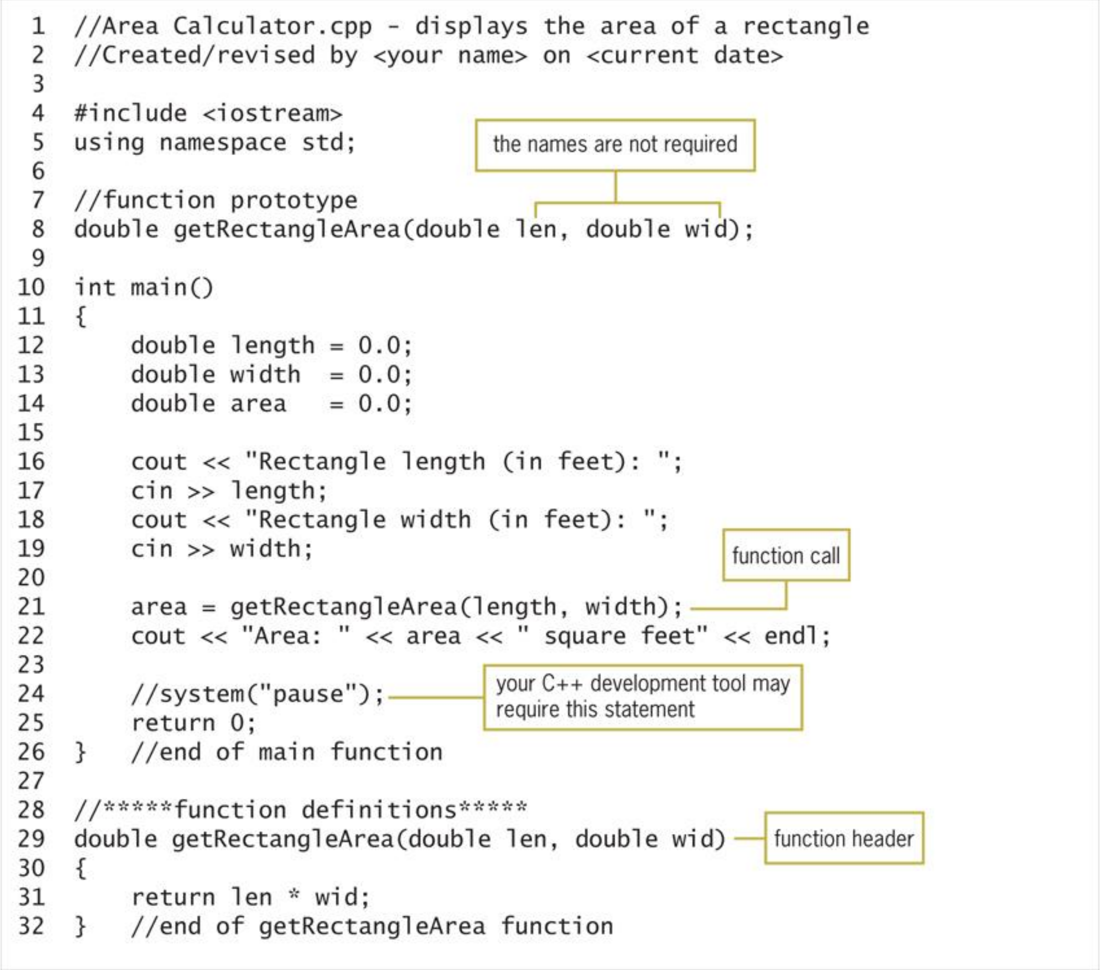
The Area Calculator Program (cont'd.)



```
Rectangle length (in feet): 8
Rectangle width (in feet): 11
Area: 88 square feet
Press any key to continue . . .
```

Figure 9-26 Sample run of the area calculator program

The Area Calculator Program (cont'd.)



```
//Area Calculator.cpp - displays the area of a rectangle
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

//function prototype
double getRectangleArea(double len, double wid);

int main()
{
    double length = 0.0;
    double width  = 0.0;
    double area   = 0.0;

    cout << "Rectangle length (in feet): ";
    cin >> length;
    cout << "Rectangle width (in feet): ";
    cin >> width;

    area = getRectangleArea(length, width);
    cout << "Area: " << area << " square feet" << endl;

    //system("pause");
    return 0;
}   //end of main function

//*****function definitions*****
double getRectangleArea(double len, double wid)
{
    return len * wid;
}   //end of getRectangleArea function
```

the names are not required

function call

your C++ development tool may require this statement

function header

Figure 9-27 Area calculator program

The Area Calculator Program (cont'd.)

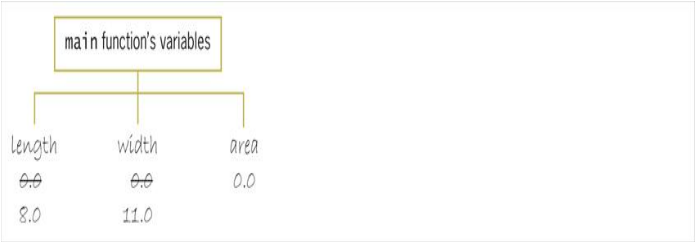


Figure 9-28 Desk-check table after statements on lines 12 through 19 are processed

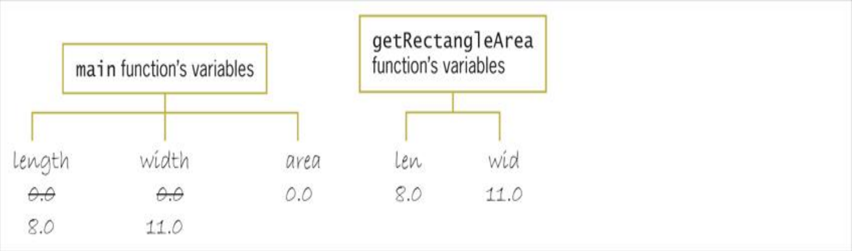
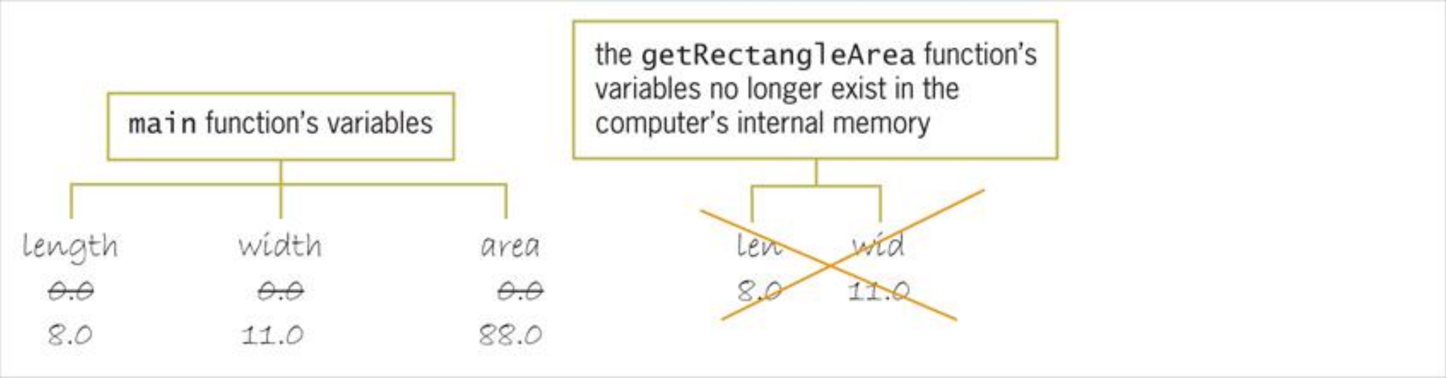

main function's variables
length
~~0.0~~
8.0
width
~~0.0~~
11.0
area
0.0
getRectangleArea function's variables
len
8.0
wid
11.0

Figure 9-29 Desk-check table after function header `getRectangleArea` is processed

The Area Calculator Program (cont'd.)



main function's variables

the getRectangleArea function's variables no longer exist in the computer's internal memory

length
~~0.0~~
8.0

width
~~0.0~~
11.0

area
~~0.0~~
88.0

~~len~~
~~8.0~~

~~wid~~
~~11.0~~

Figure 9-30 Desk-check table after `getRectangleArea` function ends

The Scope and Lifetime of a Variable

- A variable's **scope** indicates where in the program the variable can be used
- A variable's **lifetime** indicates how long the variable remains in the computer's internal memory
- Both scope and lifetime are determined by where you declare the variable in the program
- Variables declared within a function and those that appear in a function's *parameterList* have a local scope and are referred to as local variables

The Scope and Lifetime of a Variable (cont'd.)

- **Local variables** can be used only by the function in which they are declared or in whose *parameterList* they appear
  - Remain in internal memory until the function ends
- **Global variables** are declared outside of any function in the program
  - Remain in memory until the program ends
- Any statement can use a global variable

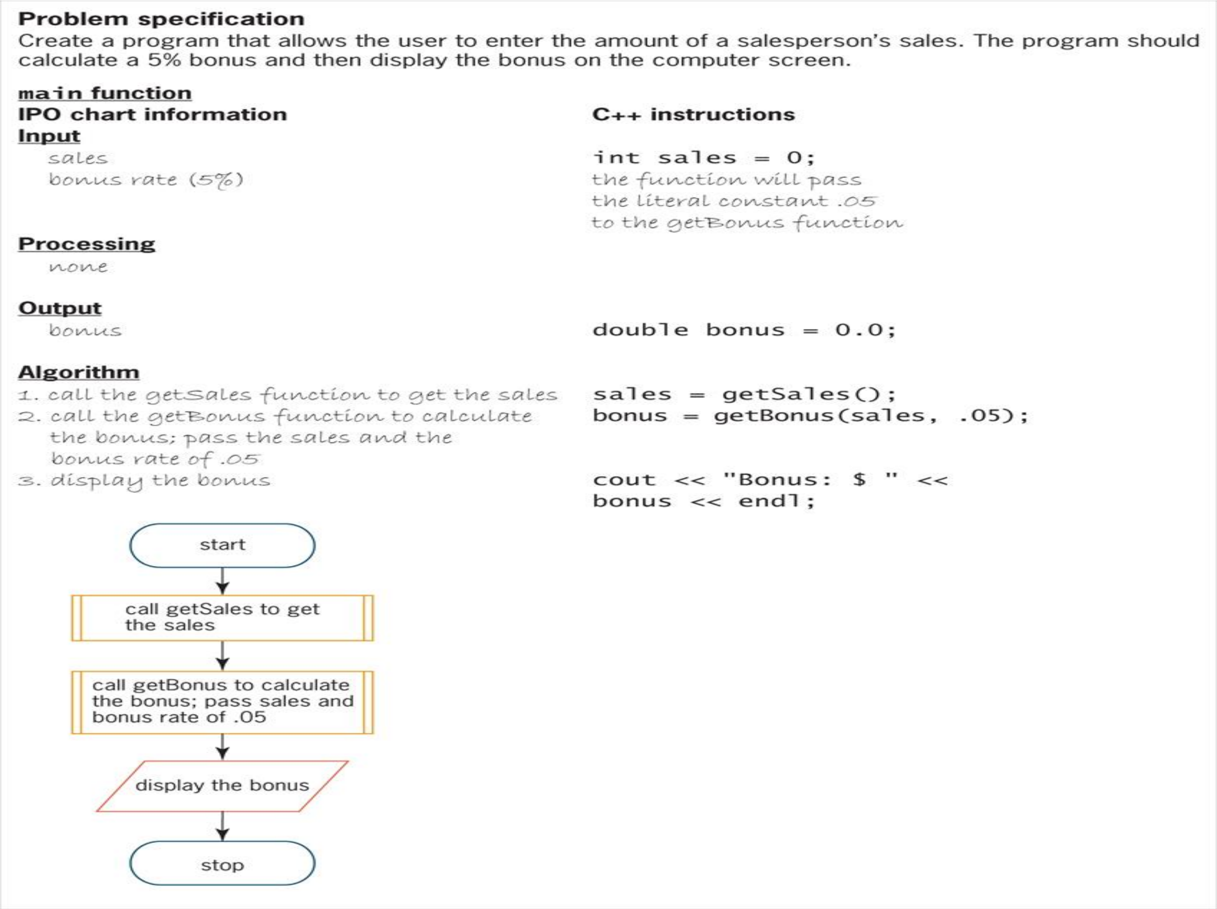
The Scope and Lifetime of a Variable (cont'd.)

- Declaring a variable as global can allow unintentional errors to occur
  - e.g., a function that should not have access to the variable inadvertently changes the variable's contents
- You should avoid using global variables unless necessary
- If more than one function needs to access the same variable, it is better to create a local variable in one function and pass it to other functions that need it

The Bonus Calculator Program

- Program that calculates a salesperson’s bonus (5% of his or her sales)
  - Uses two program-defined, value-returning functions
  - Illustrates how, when two memory locations have the same name, the position of a statement that uses the name determines which location is used (based on scope)

The Bonus Calculator Program (cont'd.)



Problem specification
Create a program that allows the user to enter the amount of a salesperson's sales. The program should calculate a 5% bonus and then display the bonus on the computer screen.

main function

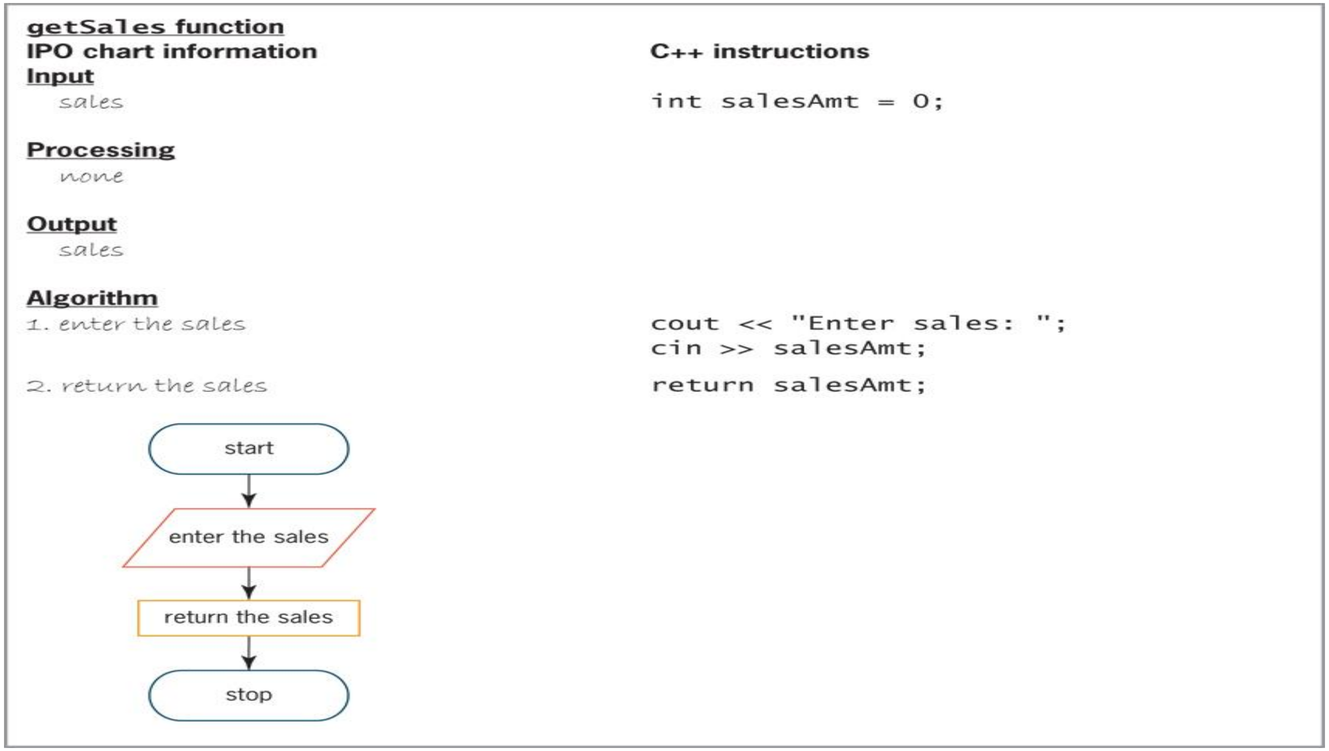
IPO chart information	**C++ instructions**
Input	
sales	`int sales = 0;`
bonus rate (5%)	the function will pass the literal constant .05 to the getBonus function
Processing	
none	
Output	
bonus	`double bonus = 0.0;`
Algorithm	
1. call the getSales function to get the sales	`sales = getSales();`
2. call the getBonus function to calculate the bonus; pass the sales and the bonus rate of .05	`bonus = getBonus(sales, .05);`

3. display the bonus	`cout << "Bonus: $ " <<`
	`bonus << endl;`

start
call getSales to get the sales
call getBonus to calculate the bonus; pass sales and bonus rate of .05
display the bonus
stop

Figure 9-31 Problem specification, IPO chart information, and C++ code for the `main` function

The Bonus Calculator Program (cont'd.)



getSales function

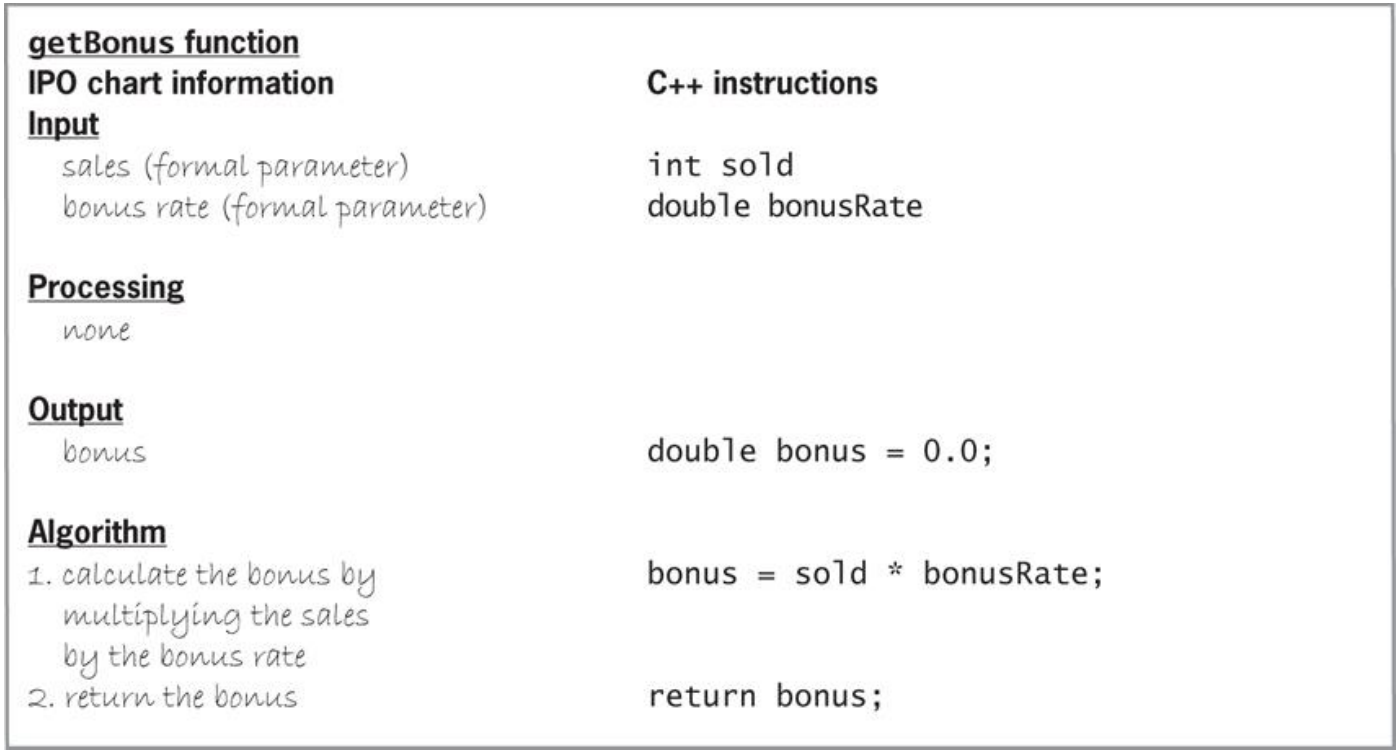
IPO chart information	**C++ instructions**
Input	
sales	`int salesAmt = 0;`
Processing	
none	
Output	
sales	
Algorithm	

1. enter the sales	`cout << "Enter sales: ";`
	`cin >> salesAmt;`

2. return the sales	`return salesAmt;`

Figure 9-32 IPO chart information and C++ code for the `getSales` function

The Bonus Calculator Program (cont'd.)

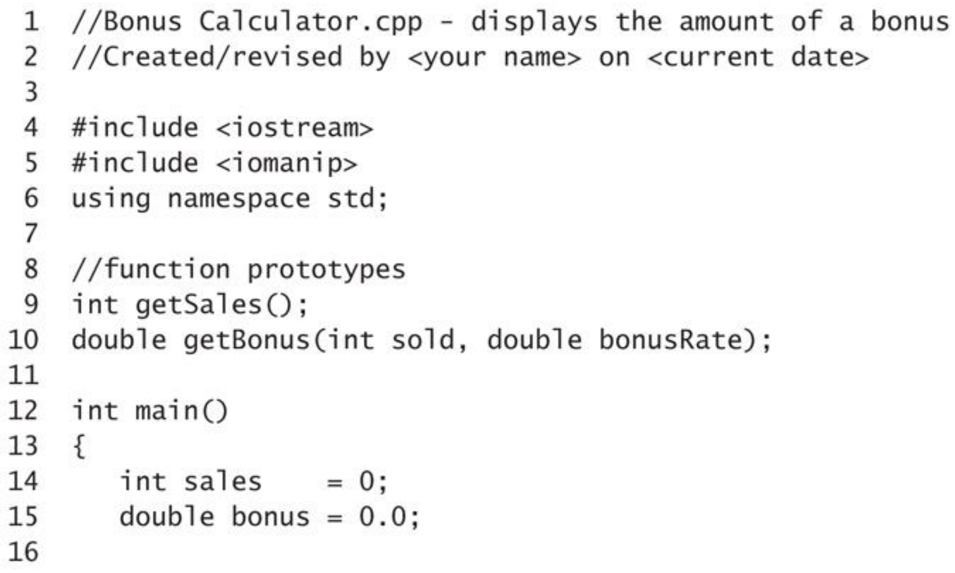


getBonus function

IPO chart information	C++ instructions
Input	
sales (formal parameter)	`int sold`
bonus rate (formal parameter)	`double bonusRate`
Processing	
none	
Output	
bonus	`double bonus = 0.0;`
Algorithm	
1. calculate the bonus by multiplying the sales by the bonus rate	`bonus = sold * bonusRate;`
2. return the bonus	`return bonus;`

Figure 9-33 IPO chart information and C++ code for the `getBonus` function

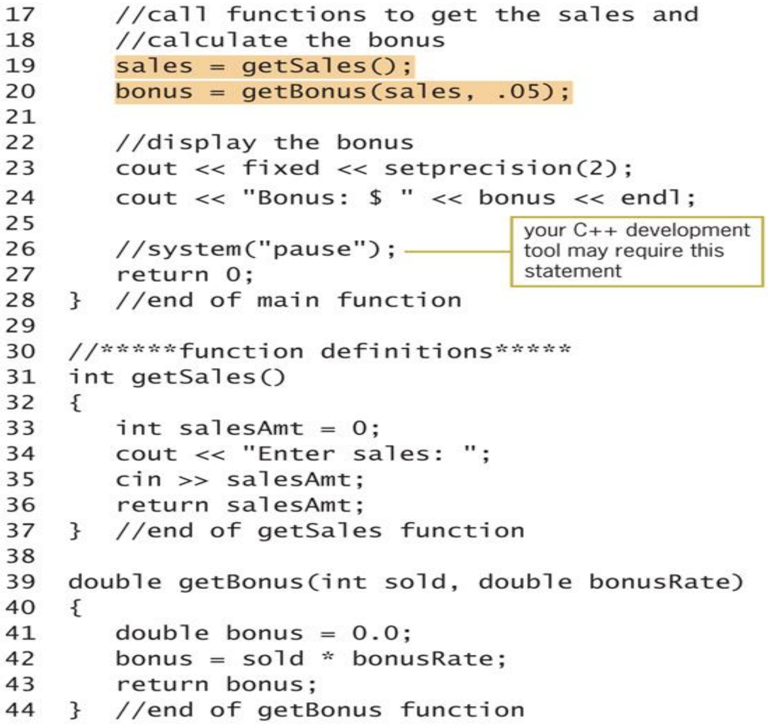
The Bonus Calculator Program (cont'd.)



```cpp
//Bonus Calculator.cpp - displays the amount of a bonus
//Created/revised by <your name> on <current date>

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

//function prototypes
int getSales();
double getBonus(int sold, double bonusRate);

int main()
{
   int sales    = 0;
   double bonus = 0.0;

```

Figure 9-35 Bonus calculator program

The Bonus Calculator Program (cont'd.)

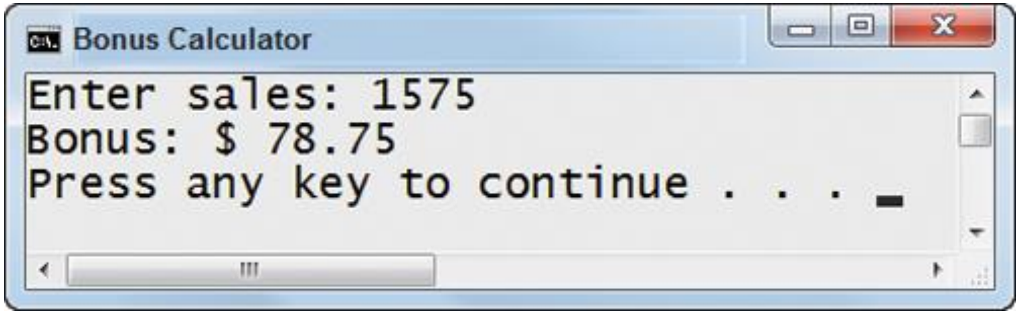


```
    //call functions to get the sales and
    //calculate the bonus
    sales = getSales();
    bonus = getBonus(sales, .05);

    //display the bonus
    cout << fixed << setprecision(2);
    cout << "Bonus: $ " << bonus << endl;

    //system("pause");
    return 0;
}   //end of main function

//*****function definitions*****
int getSales()
{
    int salesAmt = 0;
    cout << "Enter sales: ";
    cin >> salesAmt;
    return salesAmt;
}   //end of getSales function

double getBonus(int sold, double bonusRate)
{
    double bonus = 0.0;
    bonus = sold * bonusRate;
    return bonus;
}   //end of getBonus function
```

your C++ development tool may require this statement

Figure 9-35 Bonus calculator program (cont'd.)

The Bonus Calculator Program (cont'd.)



```
Enter sales: 1575
Bonus: $ 78.75
Press any key to continue . . . _
```

Figure 9-34 Sample run of bonus calculator program

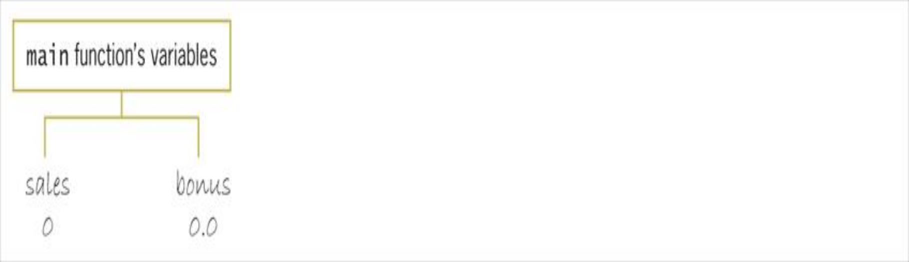
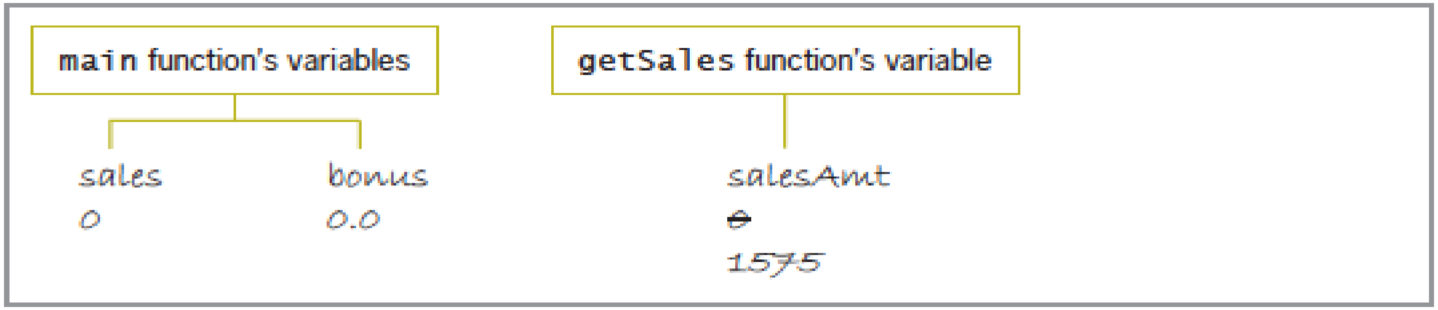


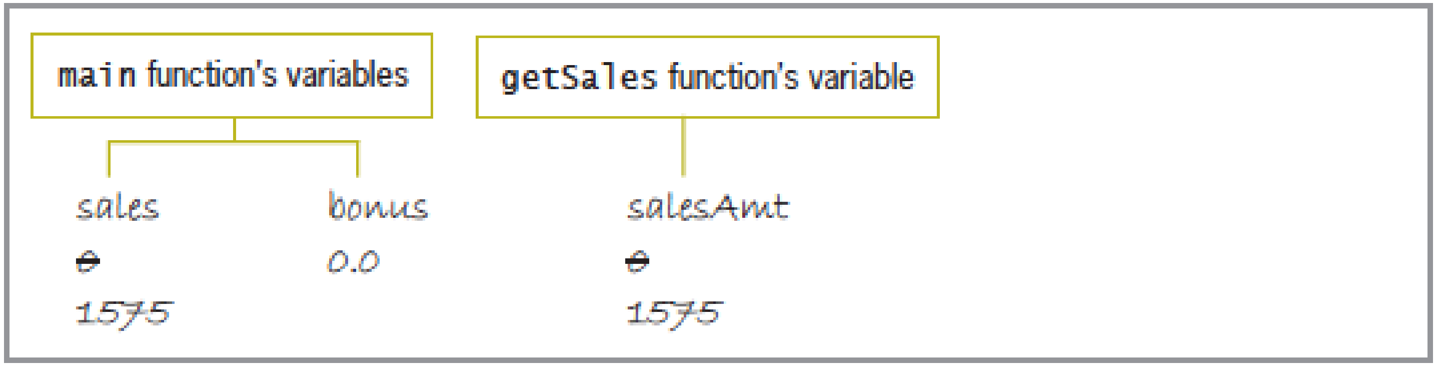
Figure 9-36 Desk-check table after variable declaration statements on lines 14 & 15 are processed

The Bonus Calculator Program (cont'd.)



main function's variables		getSales function's variable
sales	bonus	salesAmt
0	0.0	~~0~~
		1575

Figure 9-37 Desk-check table after the sales amount is entered



main function's variables		getSales function's variable
sales	bonus	salesAmt
~~0~~	0.0	~~0~~
1575		1575

Figure 9-38 Desk-check table after the sales amount is returned to the `main` function

The Bonus Calculator Program (cont’d.)

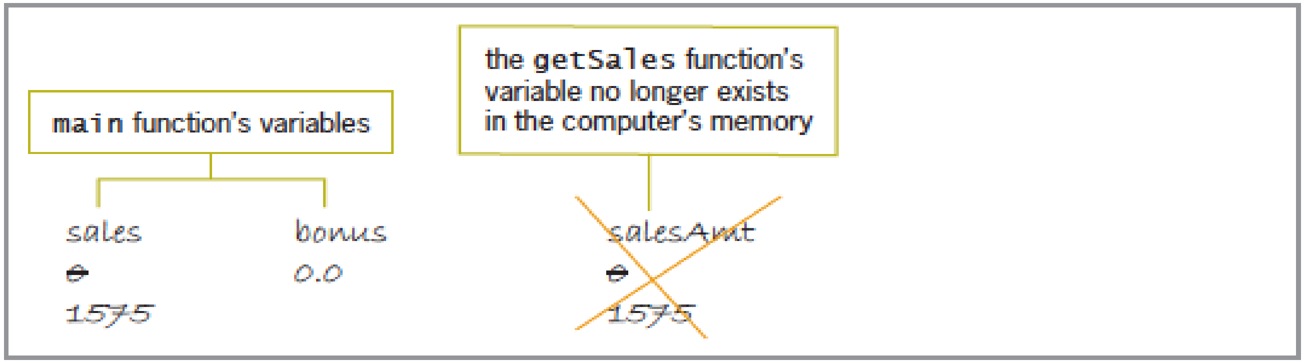

main function's variables
sales
~~0~~
1575
bonus
0.0
the getSales function's variable no longer exists in the computer's memory
~~salesAmt~~
~~0~~
~~1575~~

Figure 9-39 Desk-check table after `getSales` function ends

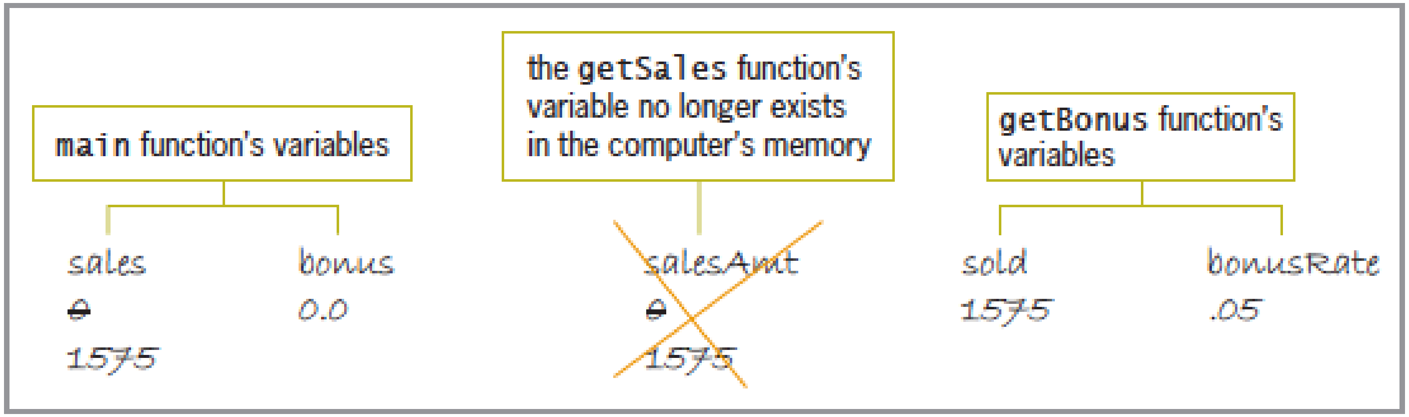
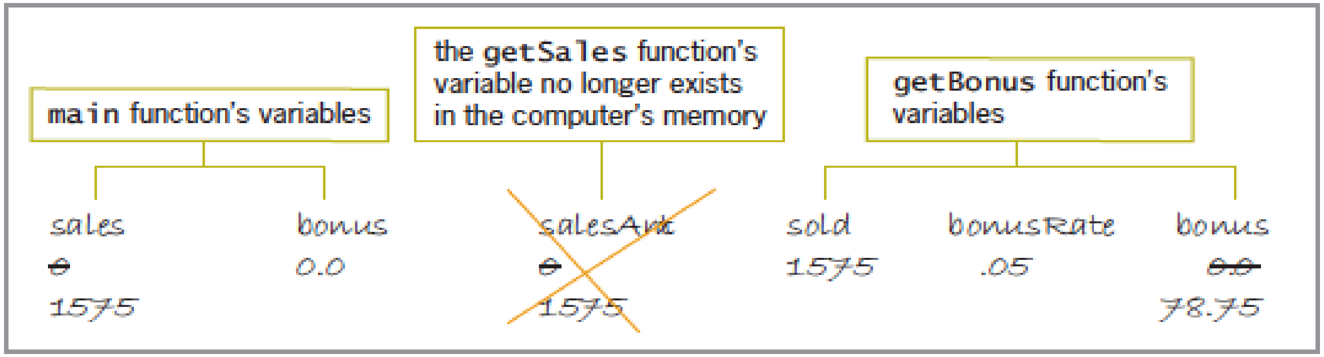

main function's variables
sales
~~0~~
1575
bonus
0.0
the getSales function's variable no longer exists in the computer's memory
~~salesAmt~~
~~0~~
~~1575~~
getBonus function's variables
sold
1575
bonusRate
.05

Figure 9-40 Desk-check table after `getBonus` function header is processed

The Bonus Calculator Program (cont’d.)

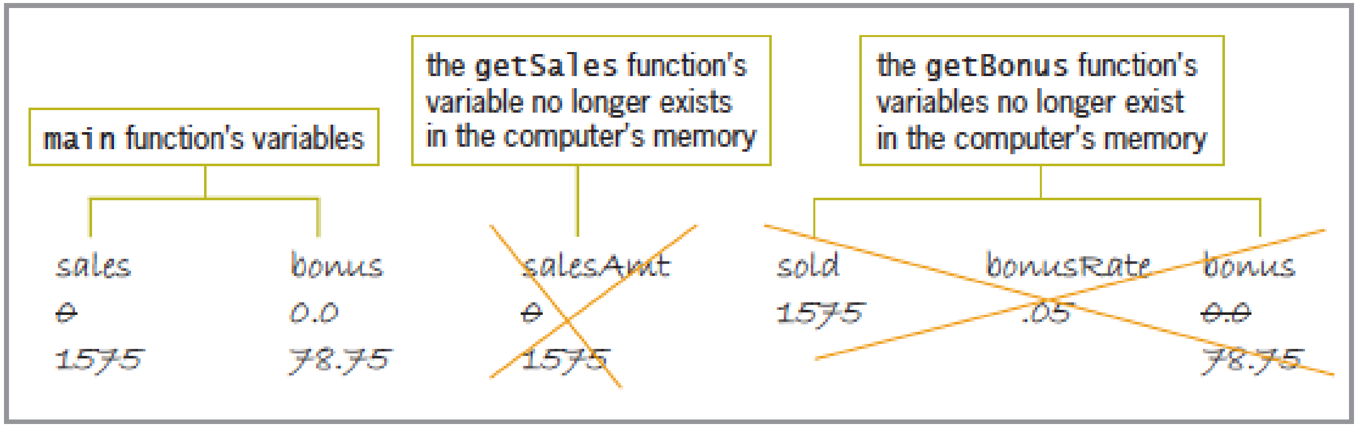


main function's variables

the getSales function's variable no longer exists in the computer's memory

getBonus function's variables

Figure 9-41 Desk-check table after bonus is calculated



main function's variables

the getSales function's variable no longer exists in the computer's memory

the getBonus function's variables no longer exist in the computer's memory

Figure 9-42 Desk-check table after `getBonus` function ends

Summary

- Functions
  - Allow programmers to avoid duplicating code
  - Allow for large, complex programs to be broken into small, manageable tasks
- Some functions are built into the language, and others are program-defined
- All functions are either value-returning or void
- A value-returning function returns one value
  - Value returned to statement that called the function
- A void function returns no value

Summary (cont'd.)

- Use the `sqrt` function to find the square root of a number
- Items in parentheses in a function call are called actual arguments
- The `rand` function is used to generate random numbers
  - Returns an integer between 0 and `RAND_MAX`
- `srand` function is used to initialize rand function
  - `time` function usually used as seed (starting point)

Summary (cont'd.)

- Function definition composed of header and body
- Header specifies function name, return data type, and formal parameter names and types (if any)
  - Data types and ordering of formal parameters must match data types and ordering of actual arguments
- Body contains instructions for performing the function's assigned task
  - Surrounded by braces (`{ }`)
- `return` statement returns the result of an expression to the calling function

Summary (cont'd.)

- You call a function by including its name and actual arguments (if any) in a statement
- Variables in C++ are passed *by value* by default
- A function prototype must be provided for each function defined below the `main` function
- Scope of a variable indicates where in the program it can be used
- Lifetime of a variable indicates how long it will stay in internal memory

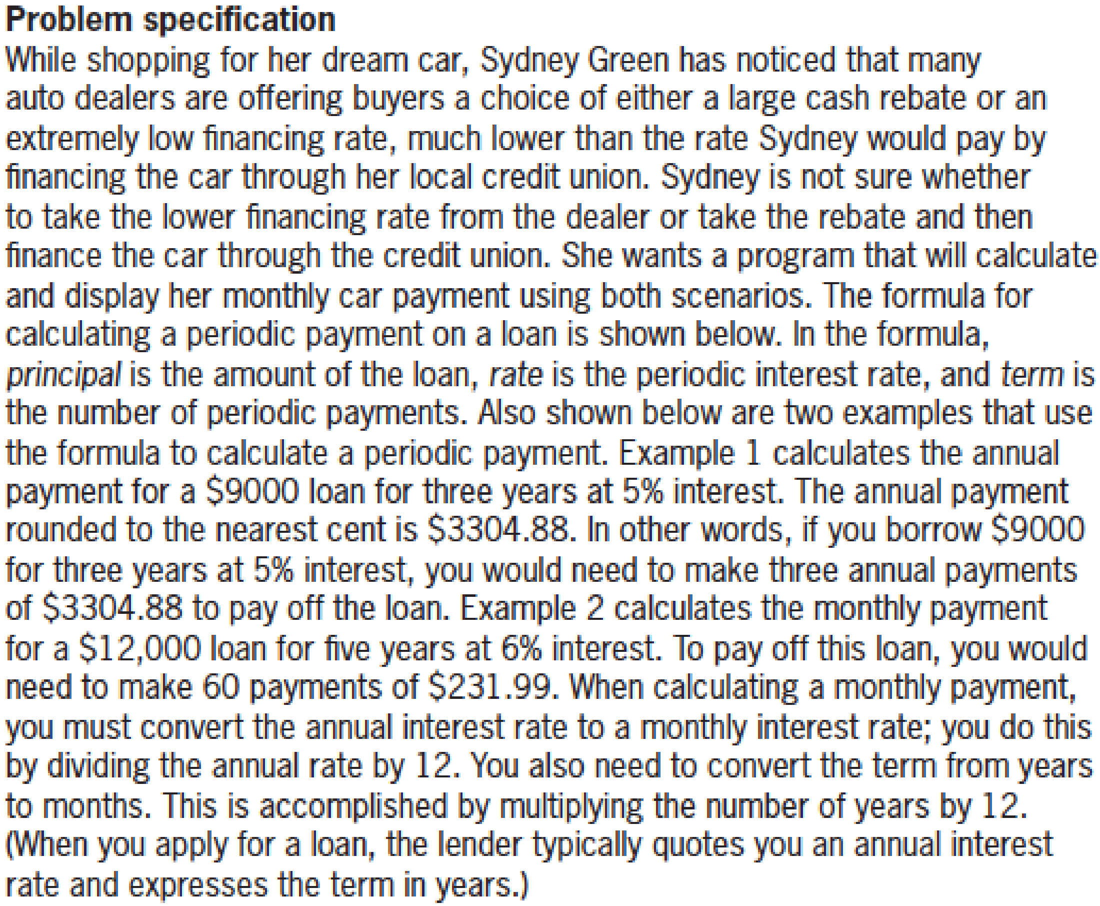
Summary (cont'd.)

- Local variables can be used only within the function in which they are declared or in whose *parameterList* they appear
  - Remain in memory until the function ends
- Global variables can be used anywhere
  - Remain in memory until the program ends
- If more than one memory location have the same name, position of the statement in which the name is used determines which location is used

Lab 9-1: Stop and Analyze

- Study the program in Figure 9-43, and then answer the questions

Lab 9-2: Plan and Create



Problem specification

While shopping for her dream car, Sydney Green has noticed that many auto dealers are offering buyers a choice of either a large cash rebate or an extremely low financing rate, much lower than the rate Sydney would pay by financing the car through her local credit union. Sydney is not sure whether to take the lower financing rate from the dealer or take the rebate and then finance the car through the credit union. She wants a program that will calculate and display her monthly car payment using both scenarios. The formula for calculating a periodic payment on a loan is shown below. In the formula, *principal* is the amount of the loan, *rate* is the periodic interest rate, and *term* is the number of periodic payments. Also shown below are two examples that use the formula to calculate a periodic payment. Example 1 calculates the annual payment for a $9000 loan for three years at 5% interest. The annual payment rounded to the nearest cent is $3304.88. In other words, if you borrow $9000 for three years at 5% interest, you would need to make three annual payments of $3304.88 to pay off the loan. Example 2 calculates the monthly payment for a $12,000 loan for five years at 6% interest. To pay off this loan, you would need to make 60 payments of $231.99. When calculating a monthly payment, you must convert the annual interest rate to a monthly interest rate; you do this by dividing the annual rate by 12. You also need to convert the term from years to months. This is accomplished by multiplying the number of years by 12. (When you apply for a loan, the lender typically quotes you an annual interest rate and expresses the term in years.)

Figure 9-44 Problem specification for Lab 9-2

Lab 9-3: Modify

- Modify the program from Lab 9-2 in three ways:
  - Allow user to enter an interest rate either as a whole number or a decimal
  - Program should compare both monthly payments and display one of three messages
  - The user should be able to calculate the monthly payments as many times as needed without having to run the program multiple times

Lab 9-4: Desk-Check

- Desk-check the code in Figure 9-51 using the data:

  Beginning balance: 2000

  w, 400, y

  D, 1200, y

  W, 45, y

  w, 55, y

  k, y

  w, 150, y

  d, 15, y

  W, 1050, n

- What current balance will the code display on the screen?

Lab 9-5: Debug

- Test the program in the Lab9-5.cpp file using the data 20500, 3500, and 10 as the asset cost, salvage value, and useful life
- The depreciation should be $1700.00
- Debug the program