Introduction to Programming in C++
Seventh Edition

Chapter 10: Void Functions

Objectives

- Create a void function
- Invoke a void function
- Pass information *by reference* to a function

Functions

- Recall that value-returning functions perform a task and then return a single value
- **Void functions** also perform tasks but do not return a value
- A void function may be used to do something like display information on the screen
  - Doesn't need to return a value

Functions (cont'd.)

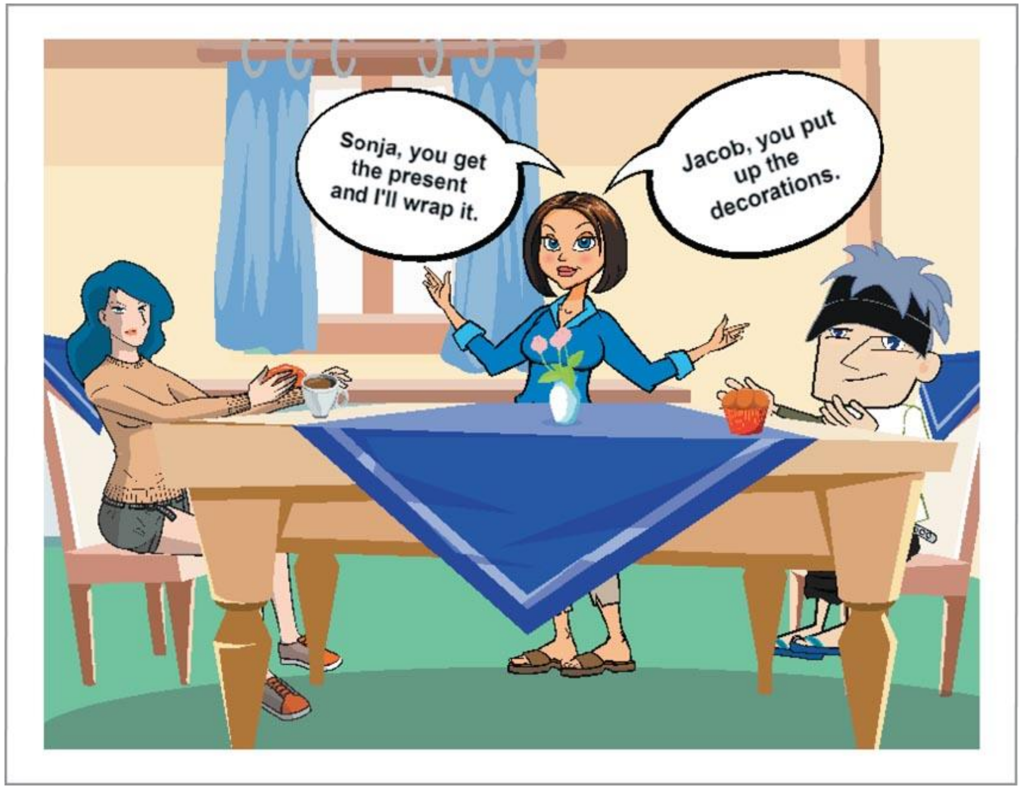
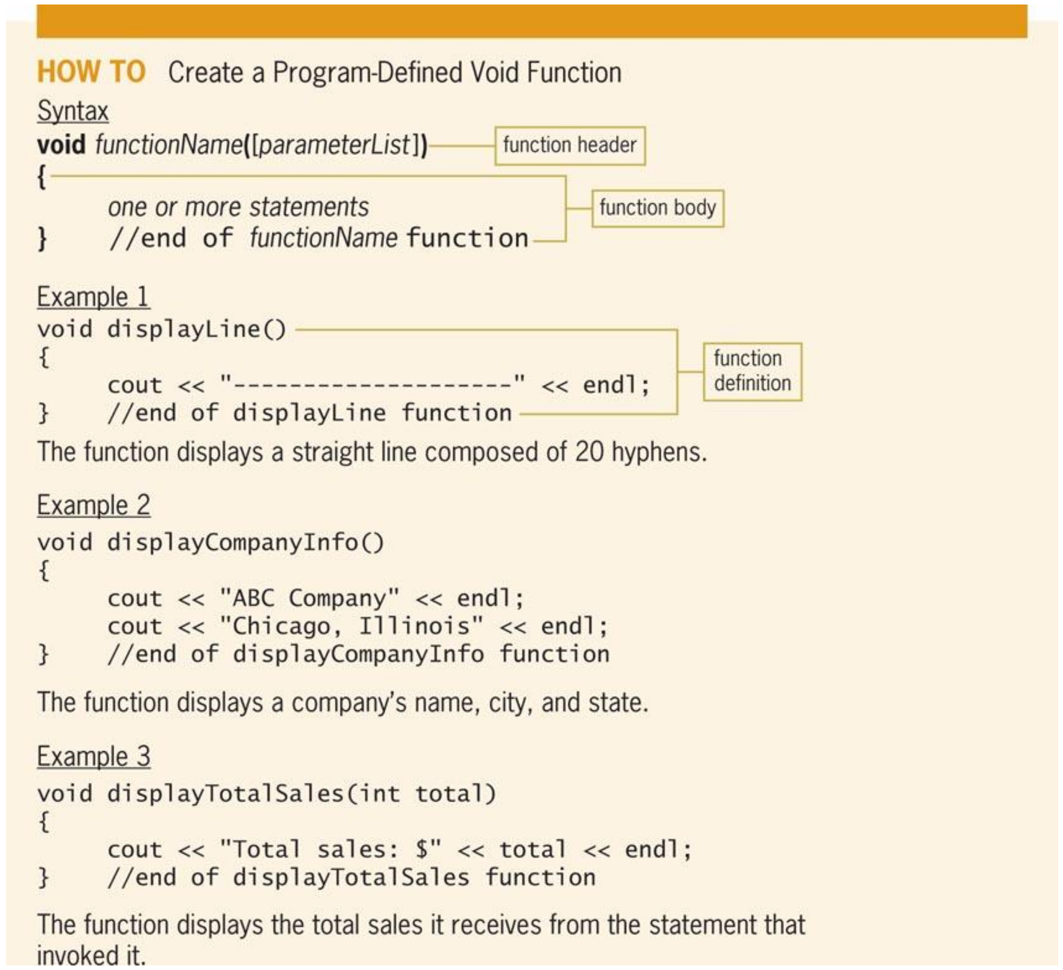

Sonja, you get the present and I'll wrap it.
Jacob, you put up the decorations.

Figure 10-1 Illustration of value-returning and void functions

Creating Program-Defined Void Functions


function header
function body
function definition

HOW TO Create a Program-Defined Void Function

Syntax

```
void functionName([parameterList])
{
      one or more statements
}     //end of functionName function
```

Example 1

```
void displayLine()
{
      cout << "--------------------" << endl;
}     //end of displayLine function
```

The function displays a straight line composed of 20 hyphens.

Example 2

```
void displayCompanyInfo()
{
      cout << "ABC Company" << endl;
      cout << "Chicago, Illinois" << endl;
}     //end of displayCompanyInfo function
```

The function displays a company's name, city, and state.

Example 3

```
void displayTotalSales(int total)
{
      cout << "Total sales: $" << total << endl;
}     //end of displayTotalSales function
```

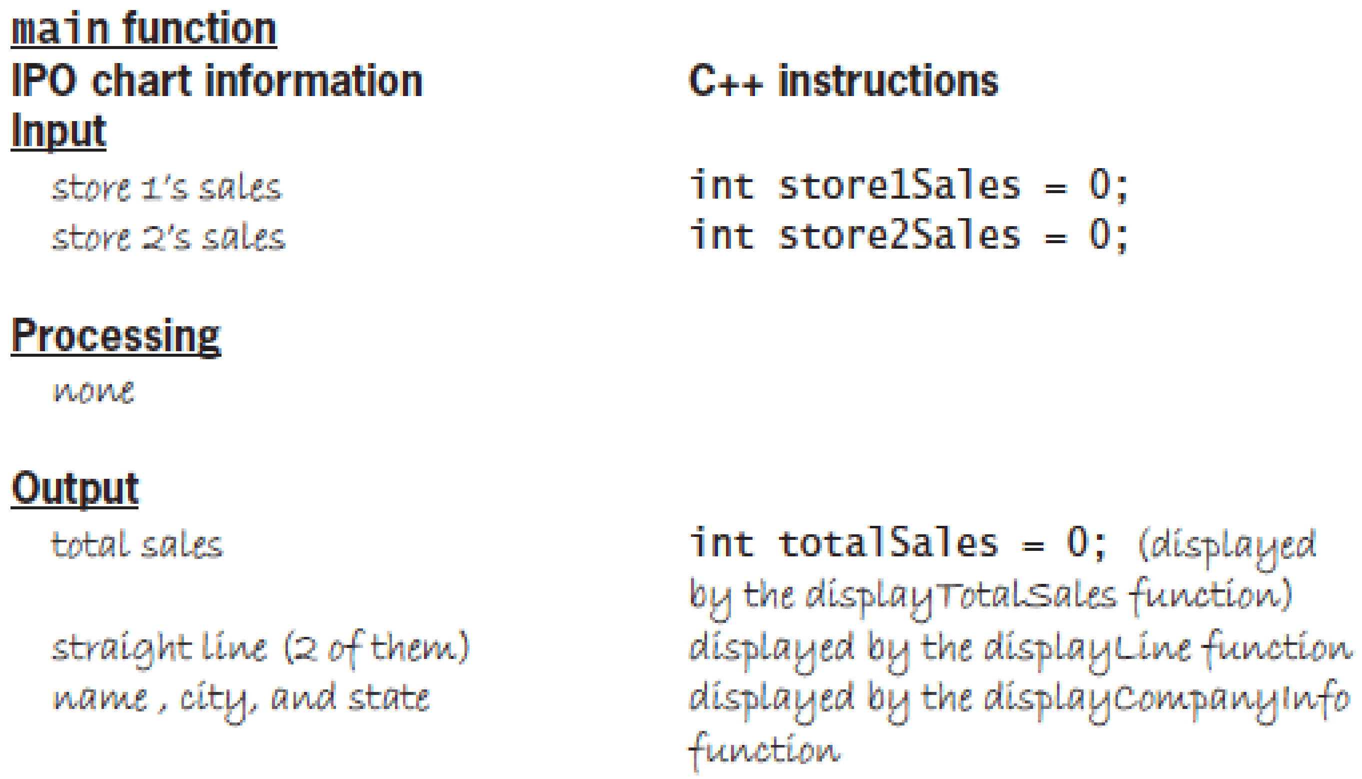
The function displays the total sales it receives from the statement that invoked it.

Figure 10-2 How to create a program-defined void function

Creating Program-Defined Void Functions (cont'd.)

- Note that header begins with keyword `void`, instead of a return data type
  - Indicates that the function does not return a value
- Function body does not contain a `return` statement
- Call a void function by including its name and actual arguments (if any) in a statement
- Call to a void function appears as a self-contained statement, not part of another statement
- Execution is same as for value-returning functions

Creating Program-Defined Void Functions (cont'd.)

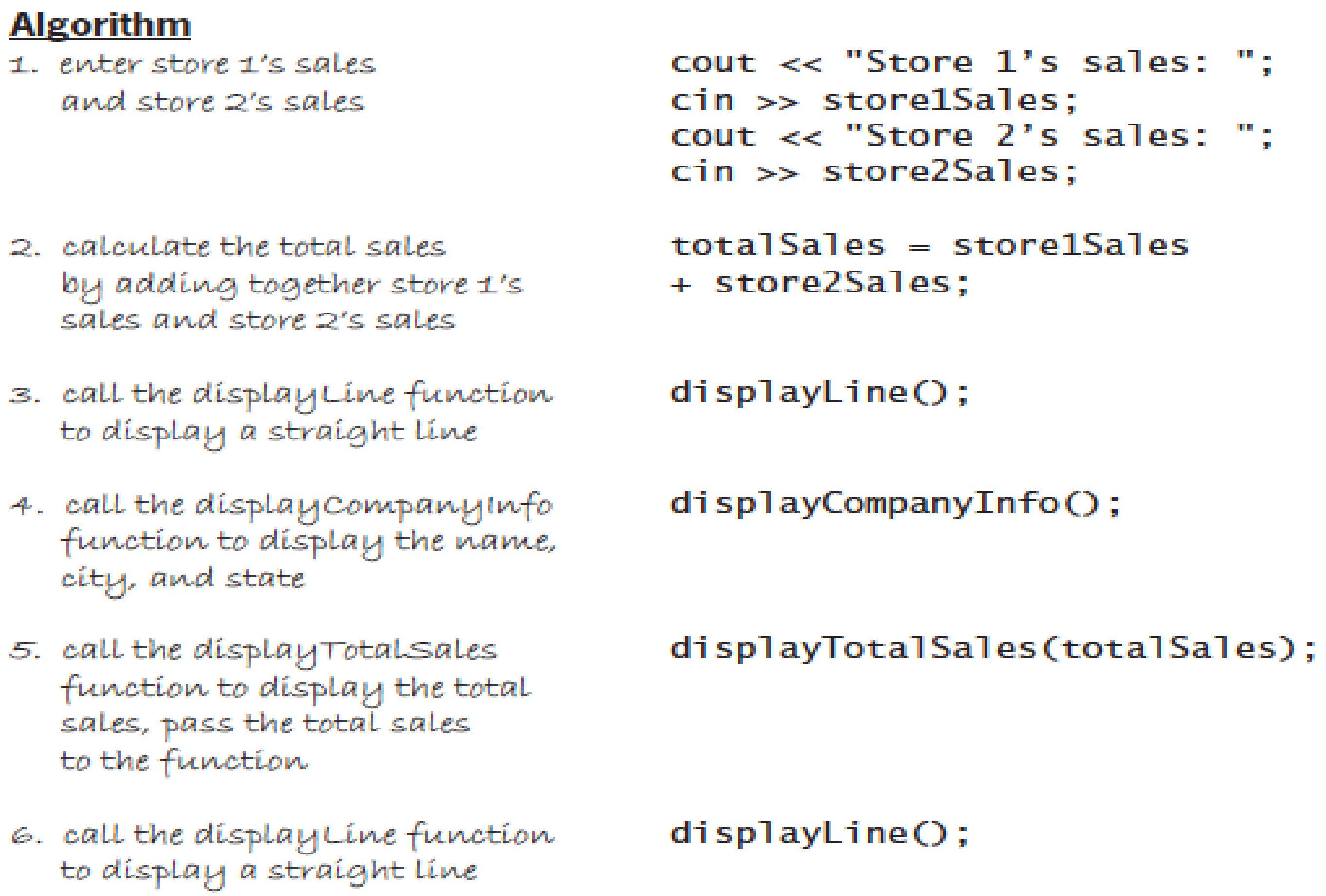


main function

IPO chart information	C++ instructions
Input	
store 1's sales	`int store1Sales = 0;`
store 2's sales	`int store2Sales = 0;`
Processing	
none	
Output	
total sales	`int totalSales = 0;` (displayed by the displayTotalSales function)
straight line (2 of them)	displayed by the displayLine function
name , city, and state	displayed by the displayCompanyInfo function

Figure 10-4 IPO chart information and C++ instructions for the ABC Company program

Creating Program-Defined Void Functions (cont'd.)



Algorithm	

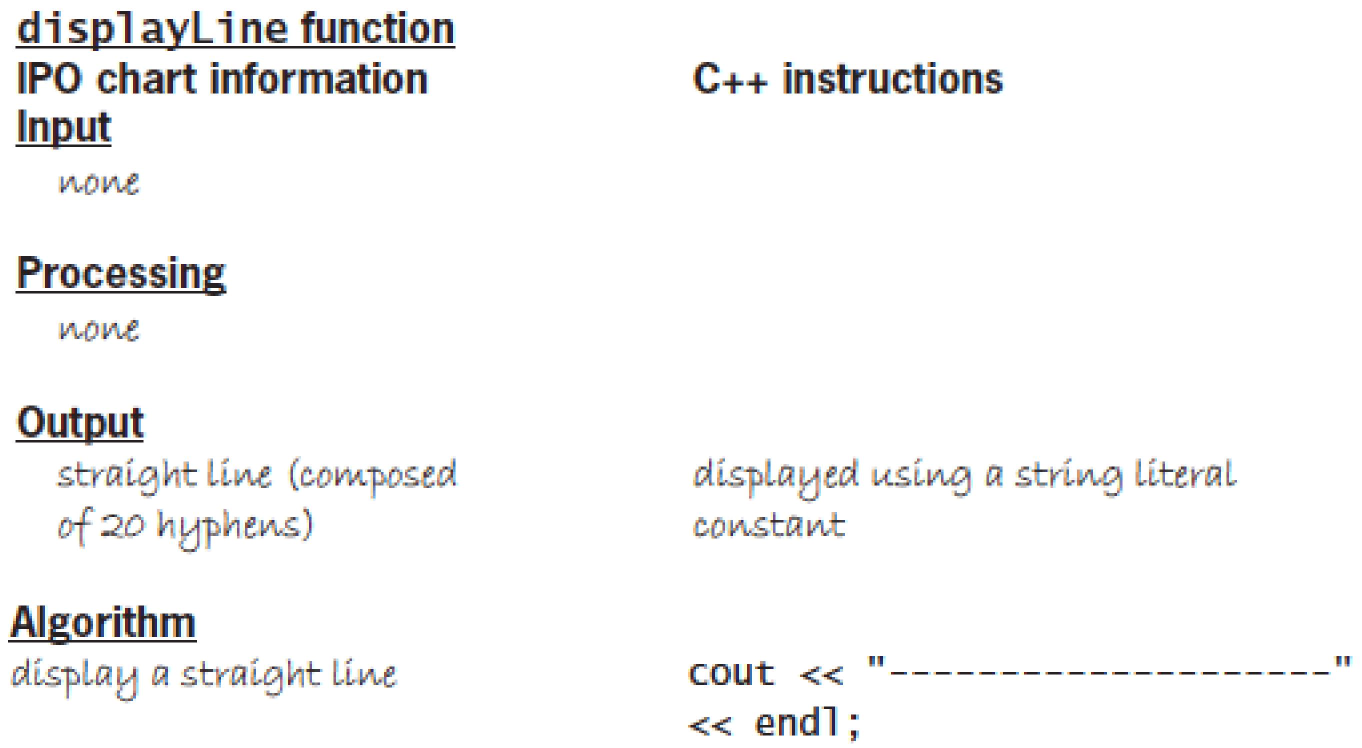
1. enter store 1's sales and store 2's sales	`cout << "Store 1's sales: ";`
	`cin >> store1Sales;`
	`cout << "Store 2's sales: ";`
	`cin >> store2Sales;`

2. calculate the total sales by adding together store 1's sales and store 2's sales	`totalSales = store1Sales`
	`+ store2Sales;`

3. call the displayLine function to display a straight line	`displayLine();`
4. call the displayCompanyInfo function to display the name, city, and state	`displayCompanyInfo();`
5. call the displayTotalSales function to display the total sales, pass the total sales to the function	`displayTotalSales(totalSales);`
6. call the displayLine function to display a straight line	`displayLine();`

Figure 10-4 IPO chart information and C++ instructions for the ABC Company program (cont'd.)

Creating Program-Defined Void Functions (cont'd.)

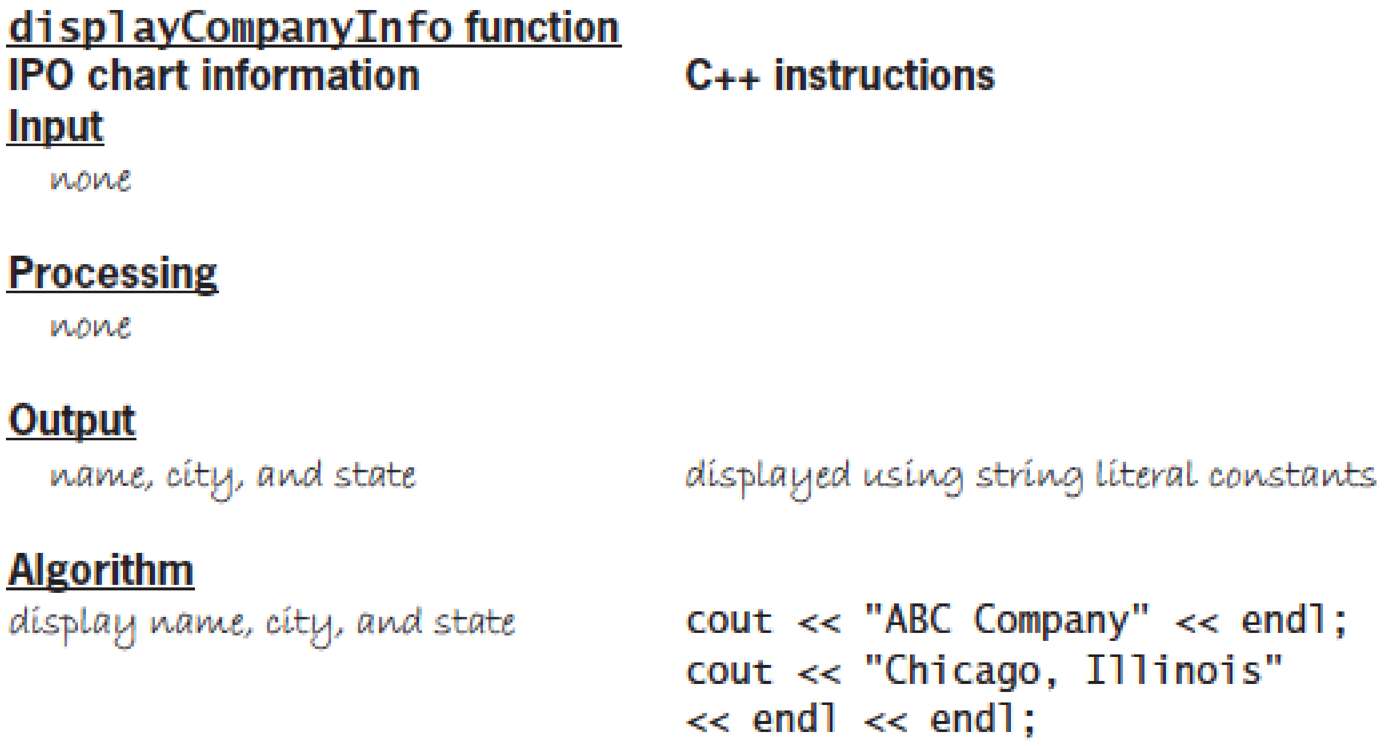


displayLine function

IPO chart information	C++ instructions
Input	
none	
Processing	
none	
Output	
straight line (composed of 20 hyphens)	displayed using a string literal constant
Algorithm	
display a straight line	`cout << "--------------------" << endl;`

Figure 10-4 IPO chart information and C++ instructions for the ABC Company program (cont'd.)

Creating Program-Defined Void Functions (cont'd.)

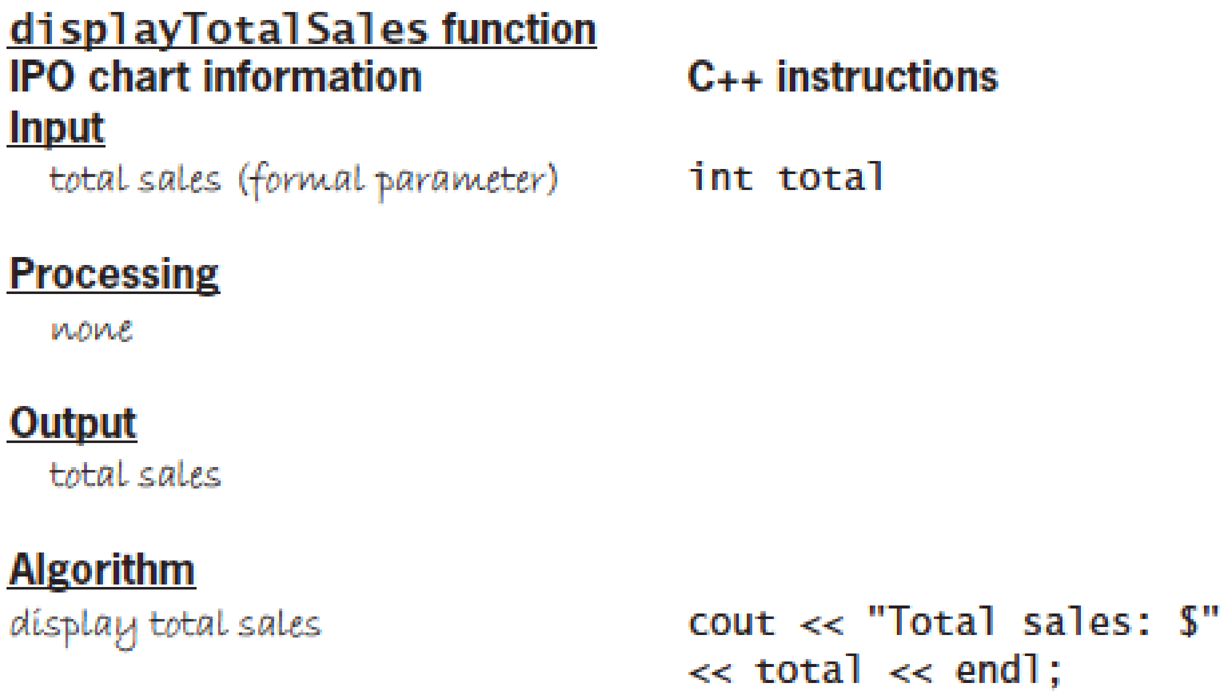


displayCompanyInfo function

IPO chart information	**C++ instructions**
Input	
none	
Processing	
none	
Output	
name, city, and state	*displayed using string literal constants*
Algorithm	
display name, city, and state	`cout << "ABC Company" << endl;` `cout << "Chicago, Illinois"` `<< endl << endl;`

Figure 10-4 IPO chart information and C++ instructions for the ABC Company program (cont'd.)

Creating Program-Defined Void Functions (cont'd.)



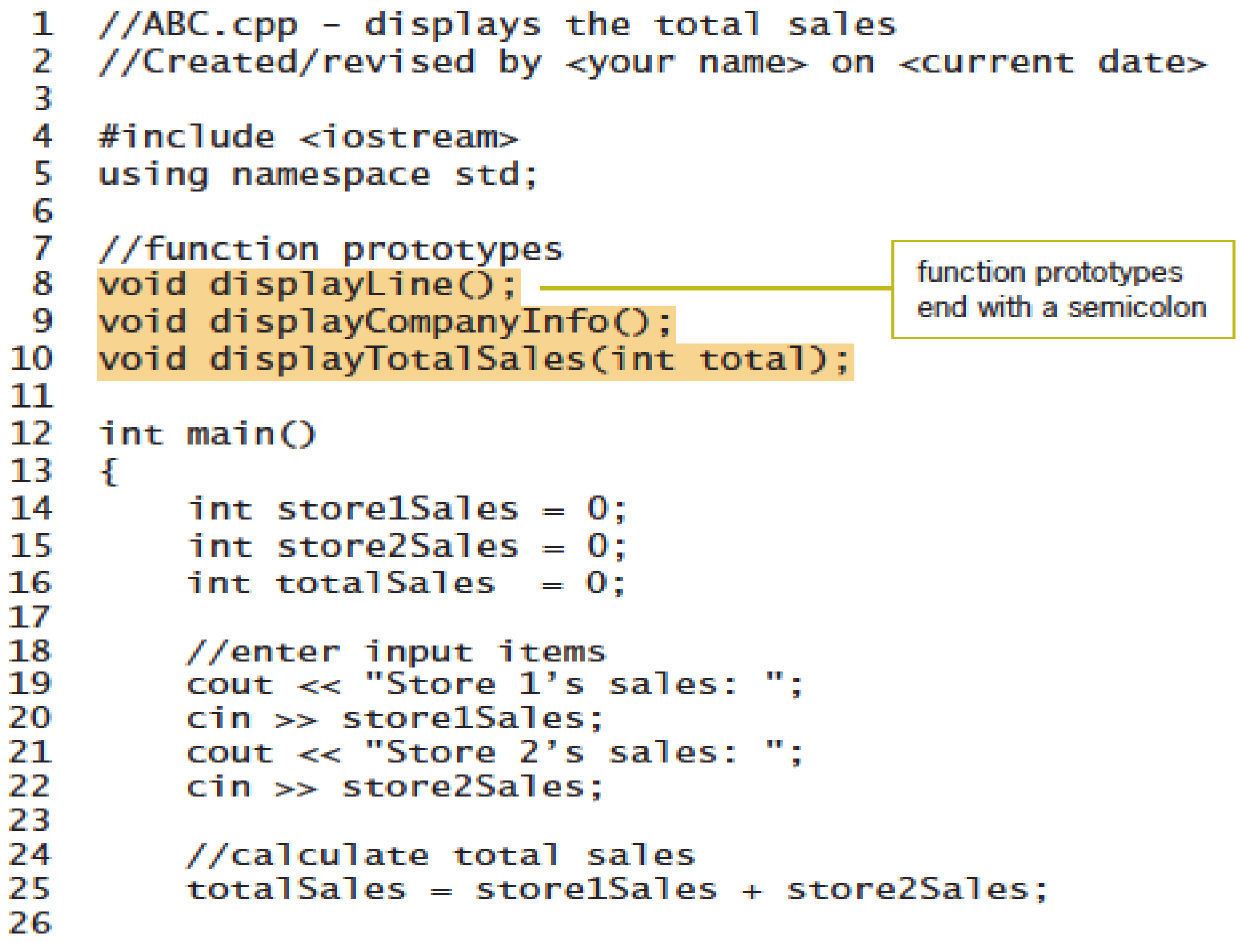
displayTotalSales function

IPO chart information	C++ instructions
Input	
total sales (formal parameter)	`int total`
Processing	
none	
Output	
total sales	
Algorithm	

display total sales	`cout << "Total sales: $"`
	`<< total << endl;`

Figure 10-4 IPO chart information and C++ instructions for the ABC Company program (cont'd.)

Creating Program-Defined Void Functions (cont'd.)



```
//ABC.cpp - displays the total sales
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

//function prototypes
void displayLine();
void displayCompanyInfo();
void displayTotalSales(int total);

int main()
{
    int store1Sales = 0;
    int store2Sales = 0;
    int totalSales  = 0;

    //enter input items
    cout << "Store 1's sales: ";
    cin >> store1Sales;
    cout << "Store 2's sales: ";
    cin >> store2Sales;

    //calculate total sales
    totalSales = store1Sales + store2Sales;

```

function prototypes end with a semicolon

Figure 10-5 ABC Company program

Creating Program-Defined Void Functions (cont'd.)

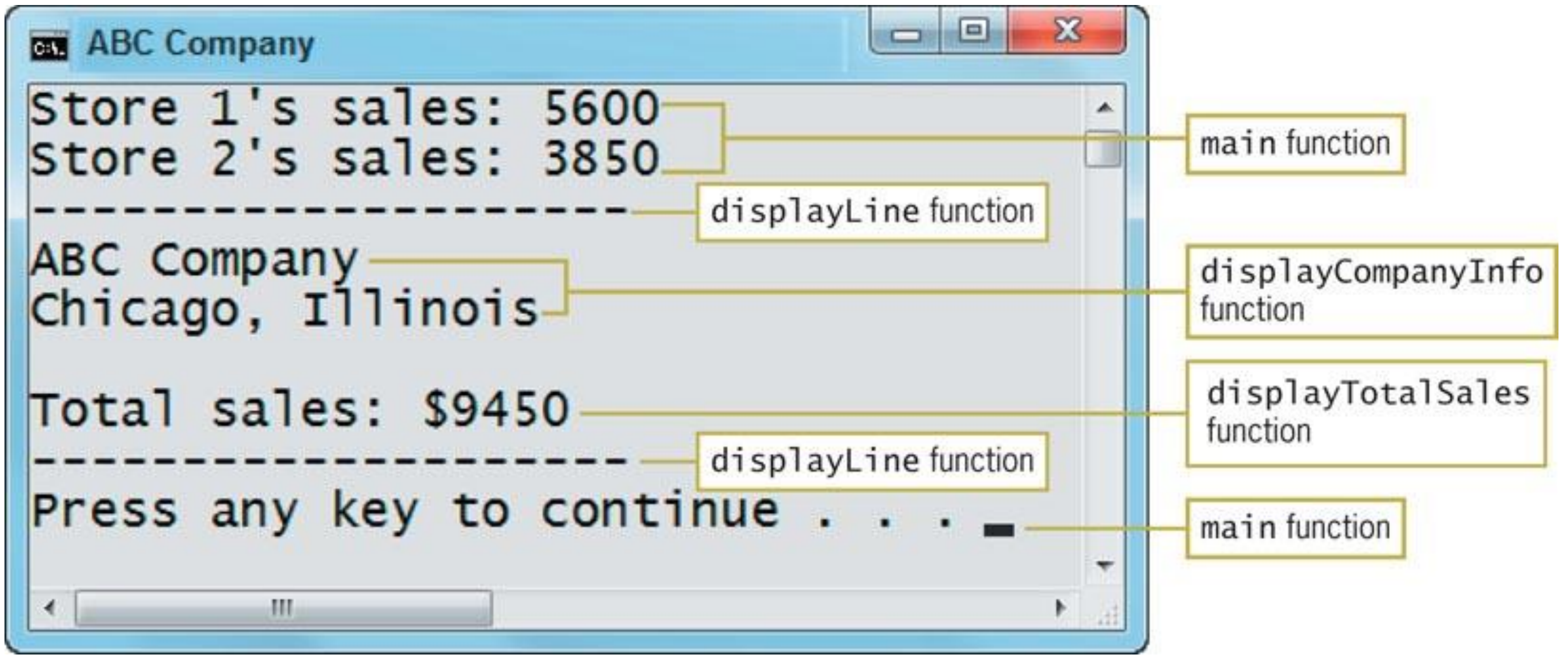
```
    //display output items
    displayLine();
    displayCompanyInfo();
    displayTotalSales(totalSales);
    displayLine();

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

//*****function definitions*****
void displayLine()
{
    cout << "---------------------" << endl;
}   //end of displayLine function

void displayCompanyInfo()
{
    cout << "ABC Company" << endl;
    cout << "Chicago, Illinois" << endl << endl;
}   //end of displayCompanyInfo function

void displayTotalSales(int total)
{
    cout << "Total sales: $" << total << endl;
}   //end of displayTotalSales function
```

your C++ development tool may not require this statement

function headers do not end with a semicolon

Figure 10-5 ABC Company program (cont'd.)

Creating Program-Defined Void Functions (cont'd.)



```
Store 1's sales: 5600
Store 2's sales: 3850
---------------------
ABC Company
Chicago, Illinois

Total sales: $9450
---------------------
Press any key to continue . . .
```

Figure 10-6 Sample run of the ABC Company program

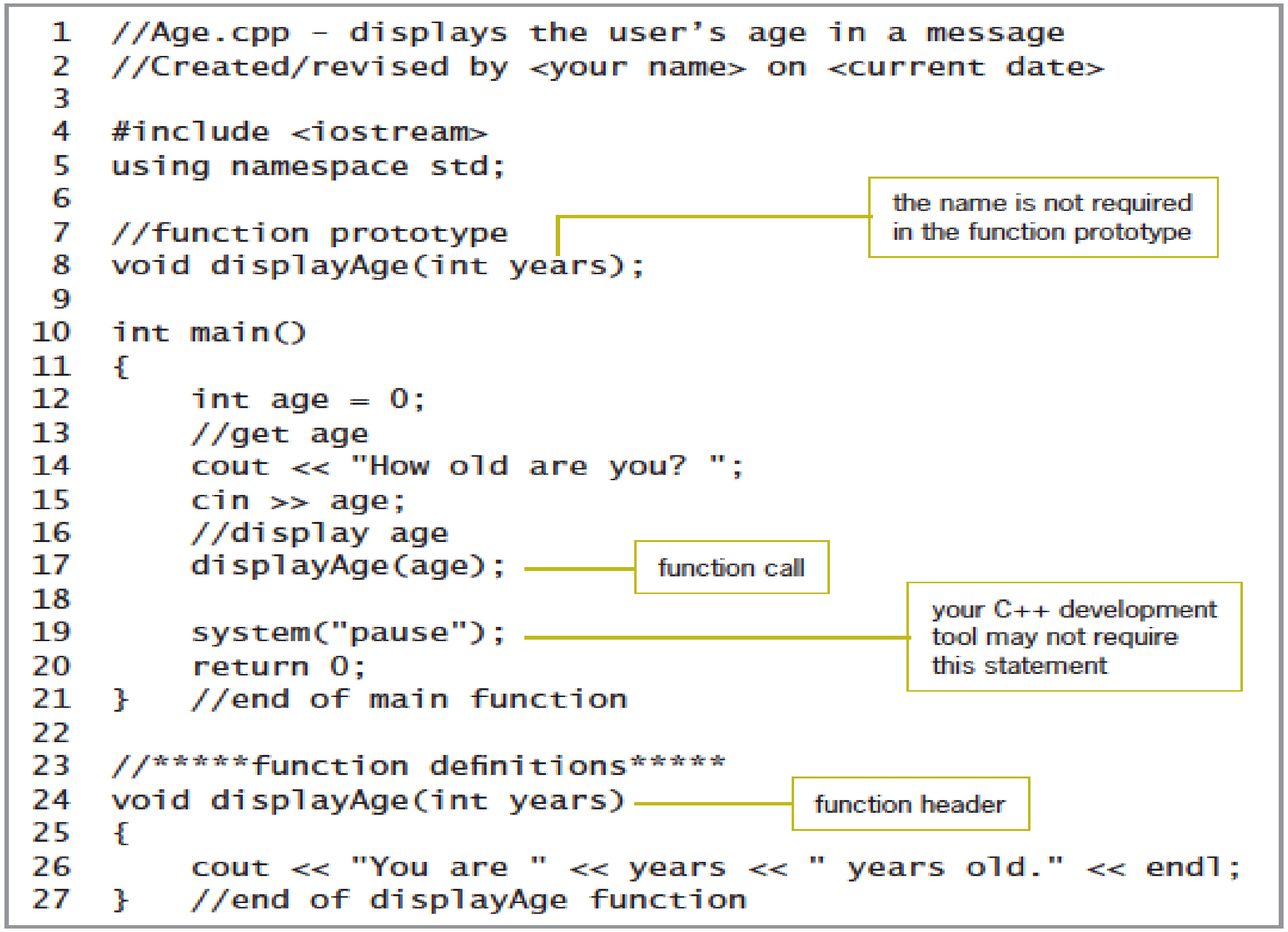
Passing Variables to a Function

- Recall you can pass a variable's value or its address
- Passing a variable's value is referred to as passing *by value*, while passing a variable's address is referred to as **passing *by reference***
- Which one you choose depends on whether the receiving function should have access to the variable in memory
- Passing *by value* will not permit the function to change the contents of the variable, but passing *by reference* will

Reviewing Passing Variables by Value

- Passing a variable *by value* means that only a copy of the variable's contents is passed, not the address of the variable
- This means that the receiving function cannot change the contents of the variable
- It is thus appropriate to pass *by value* when the receiving function needs to know the value of the variable but does not need to change it

Reviewing Passing Variables by Value (cont'd.)



```
//Age.cpp - displays the user's age in a message
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

//function prototype
void displayAge(int years);

int main()
{
    int age = 0;
    //get age
    cout << "How old are you? ";
    cin >> age;
    //display age
    displayAge(age);

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

//*****function definitions*****
void displayAge(int years)
{
    cout << "You are " << years << " years old." << endl;
}   //end of displayAge function
```

the name is not required in the function prototype
function call
your C++ development tool may not require this statement
function header

Figure 10-8 Age message program

Reviewing Passing Variables by Value (cont'd.)

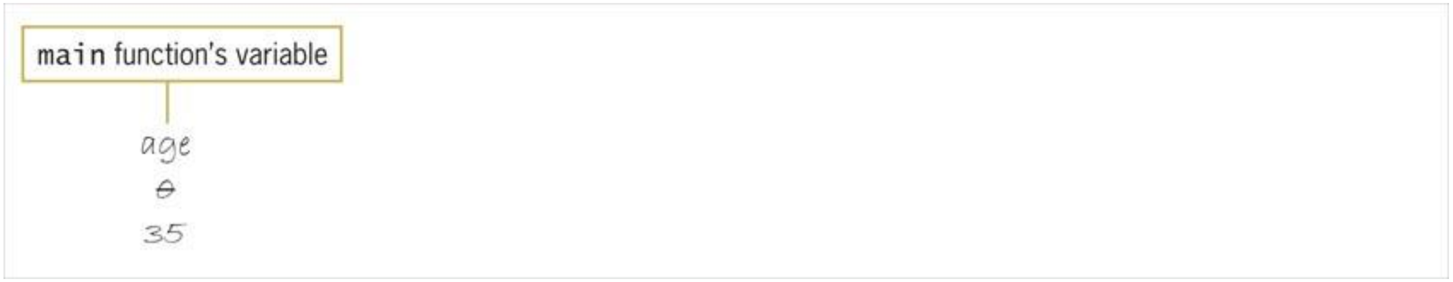


Figure 10-9 Desk-check table after the first three statements in the `main` function are processed

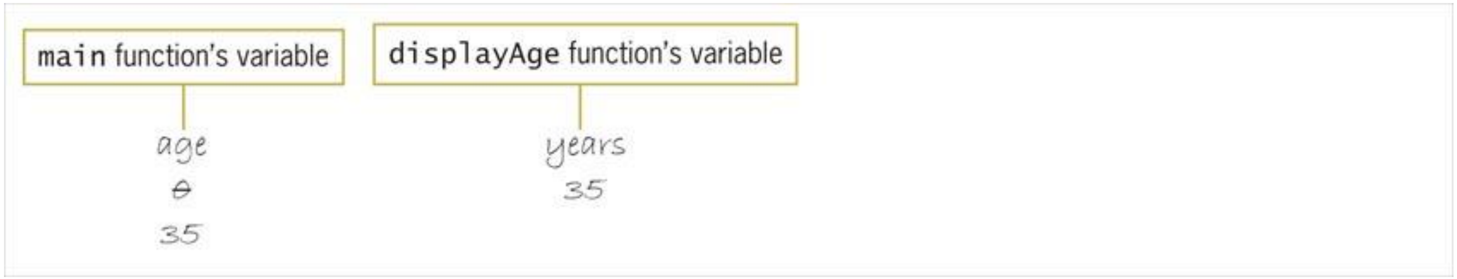


Figure 10-10 Desk-check table after the `displayAge` function header is processed

Reviewing Passing Variables by Value (cont'd.)

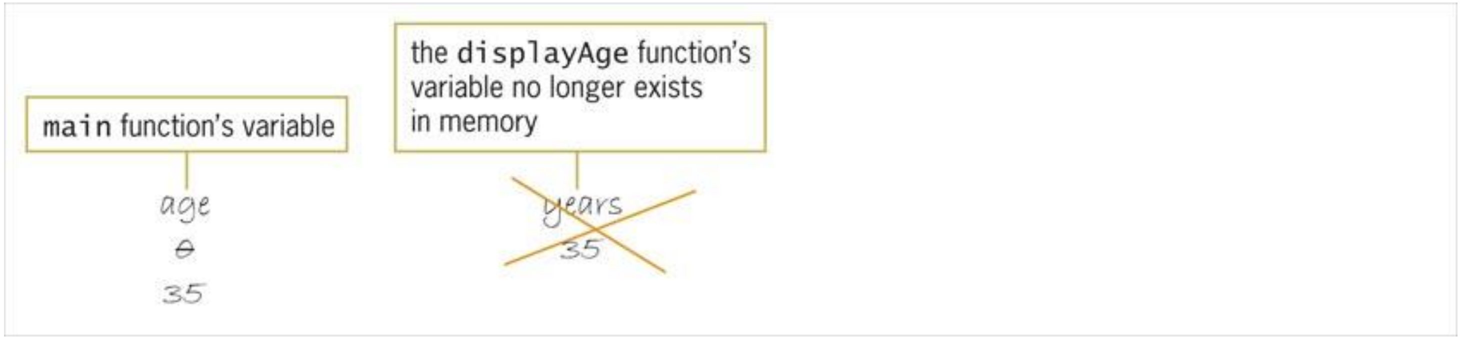
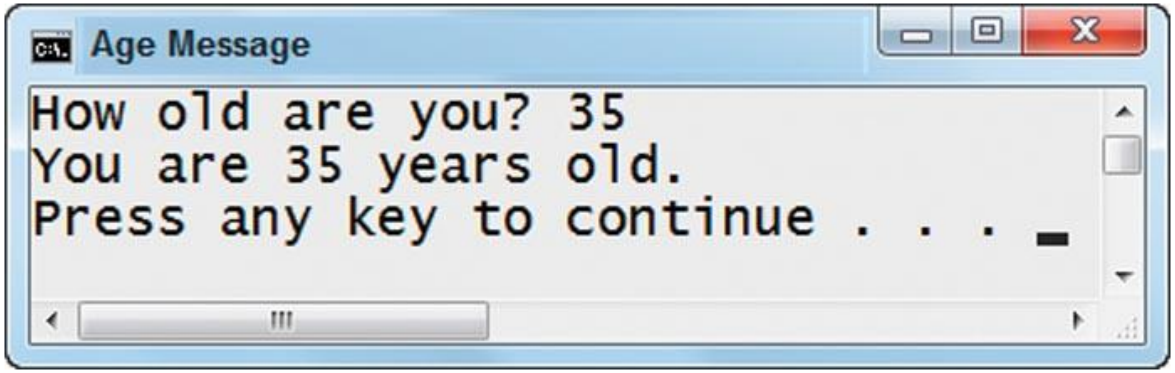

main function's variable
age
~~0~~
35
the displayAge function's variable no longer exists in memory
~~years~~
~~35~~

Figure 10-11 Desk-check table after the `displayAge` function ends



```
How old are you? 35
You are 35 years old.
Press any key to continue . . .
```

Figure 10-12 Sample run of the age message program

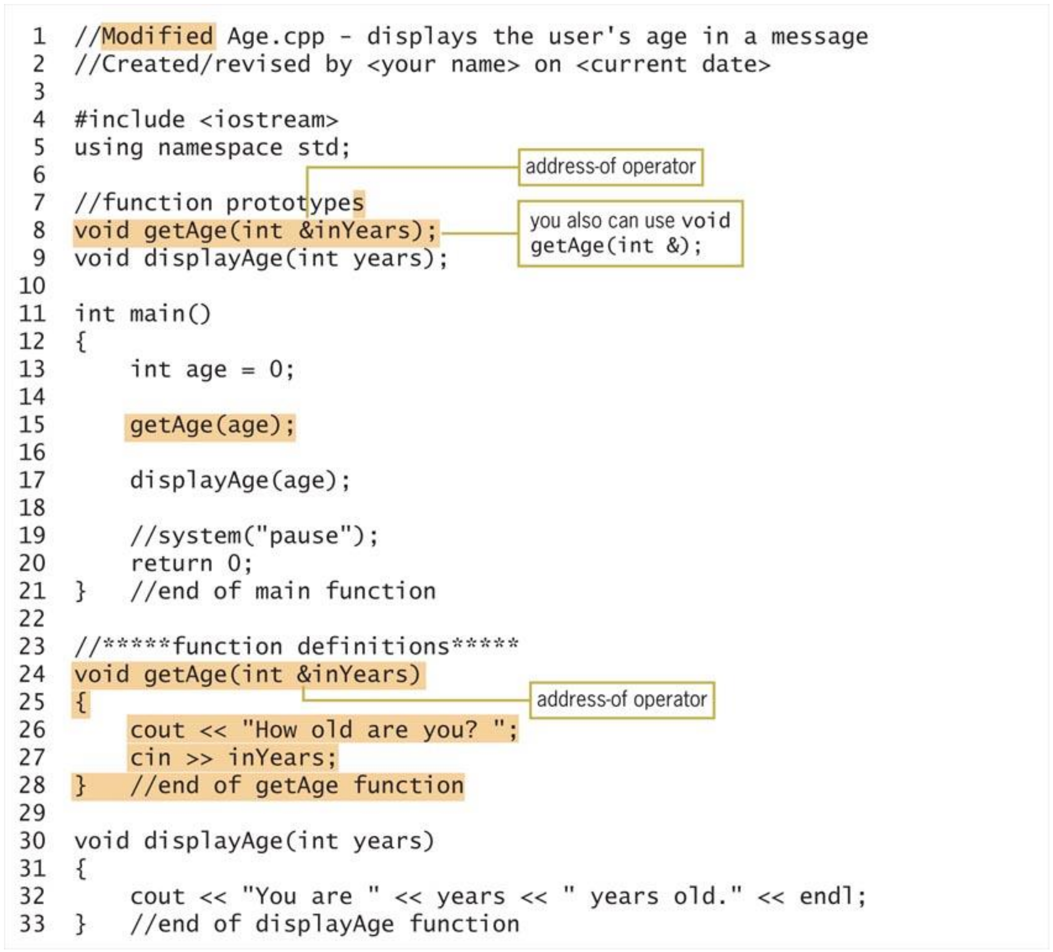
Passing Variables by Reference

- Passing a variable's address in internal memory to a function is referred to as passing *by reference*
- You pass *by reference* when you want the receiving function to change the contents of the variable
- To pass *by reference* in C++, you include an ampersand (&) before the name of the formal parameter in the receiving function's header
- Ampersand (&) is the **address-of operator**
  – Tells the computer to pass the variable's address rather than a copy of its contents

Passing Variables by Reference (cont'd.)

- If receiving function appears below `main`, you must also include the & in the receiving function's prototype
- You enter the & immediately before the name of the formal parameter in the prototype
  - If the prototype does not contain the formal parameter's name, you enter a space followed by & after the formal parameter's data type
- Void functions use variables passed *by reference* to send information back to the calling function, instead of a `return` value

Passing Variables by Reference (cont'd.)



```
//Modified Age.cpp - displays the user's age in a message
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

//function prototypes
void getAge(int &inYears);
void displayAge(int years);

int main()
{
    int age = 0;

    getAge(age);

    displayAge(age);

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

//*****function definitions*****
void getAge(int &inYears)
{
    cout << "How old are you? ";
    cin >> inYears;
}   //end of getAge function

void displayAge(int years)
{
    cout << "You are " << years << " years old." << endl;
}   //end of displayAge function
```

Figure 10-13 Modified age message program

Passing Variables by Reference (cont'd.)

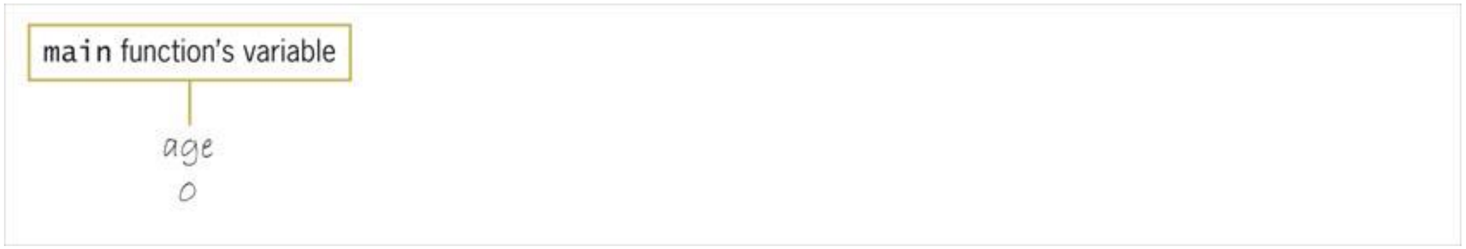


Figure 10-14 Desk-check table after the declaration statement in the `main` function is processed

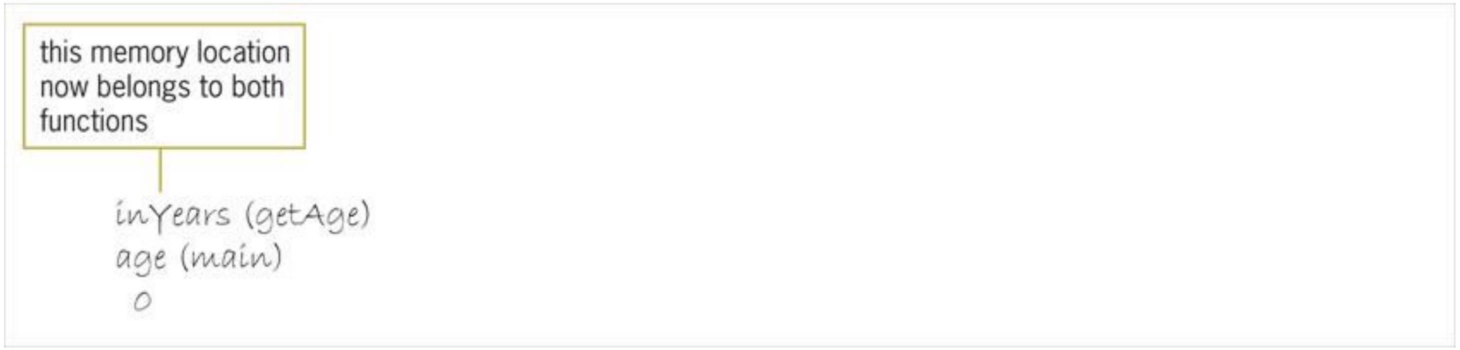


Figure 10-15 Desk-check table after the `getAge` function header is processed

Passing Variables by Reference (cont'd.)

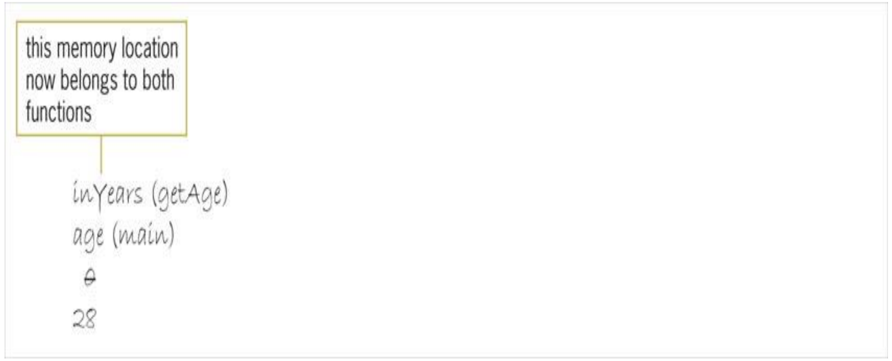


Figure 10-16 Desk-check table after the statements in the `getAge` function are processed

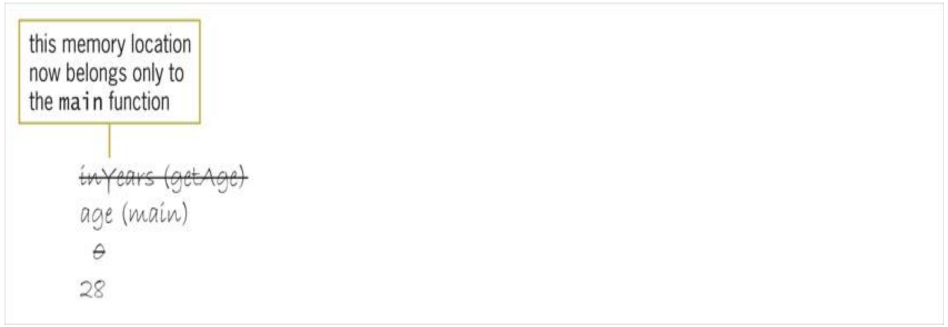


Figure 10-17 Desk-check table after the `getAge` function ends

Passing Variables by Reference (cont'd.)

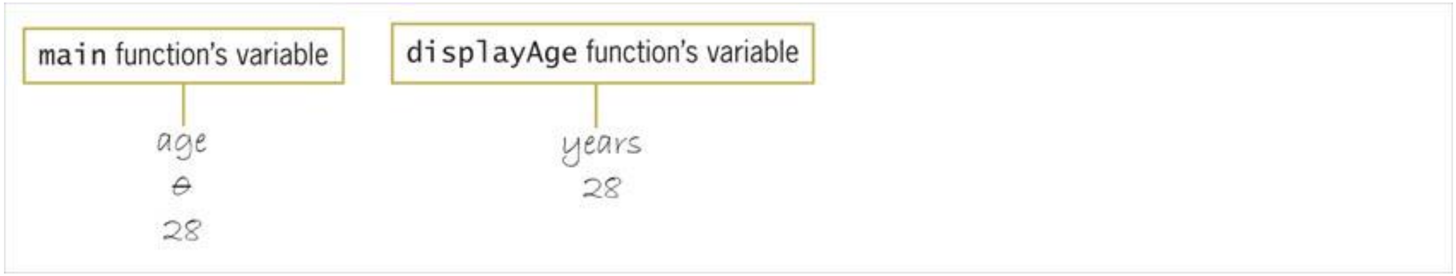


Figure 10-18 Desk-check table after the computer processes the `displayAge` function header

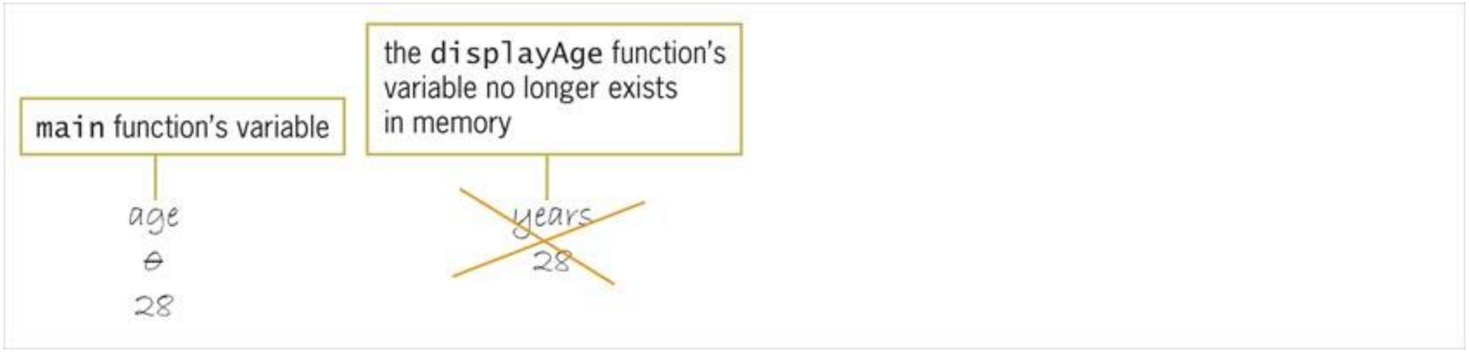
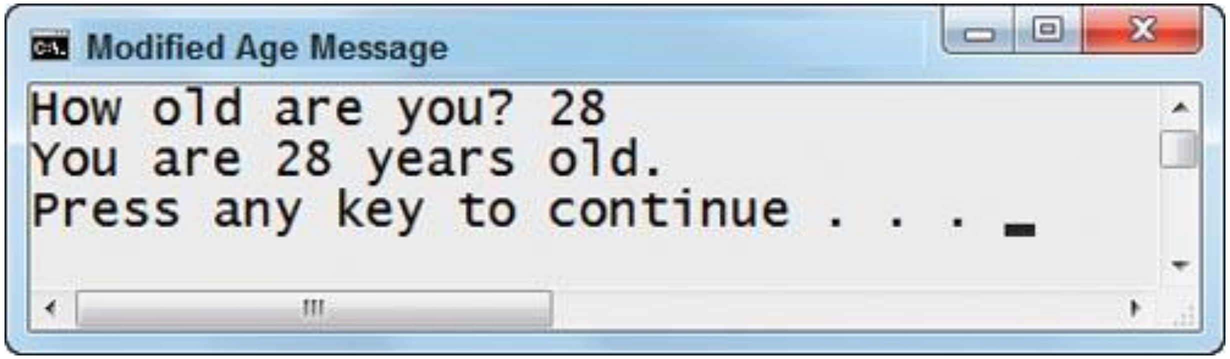

main function's variable
age
~~0~~
28
the displayAge function's variable no longer exists in memory
~~years~~
~~28~~

Figure 10-19 Desk-check table after the `displayAge` function ends

Passing Variables by Reference (cont'd.)



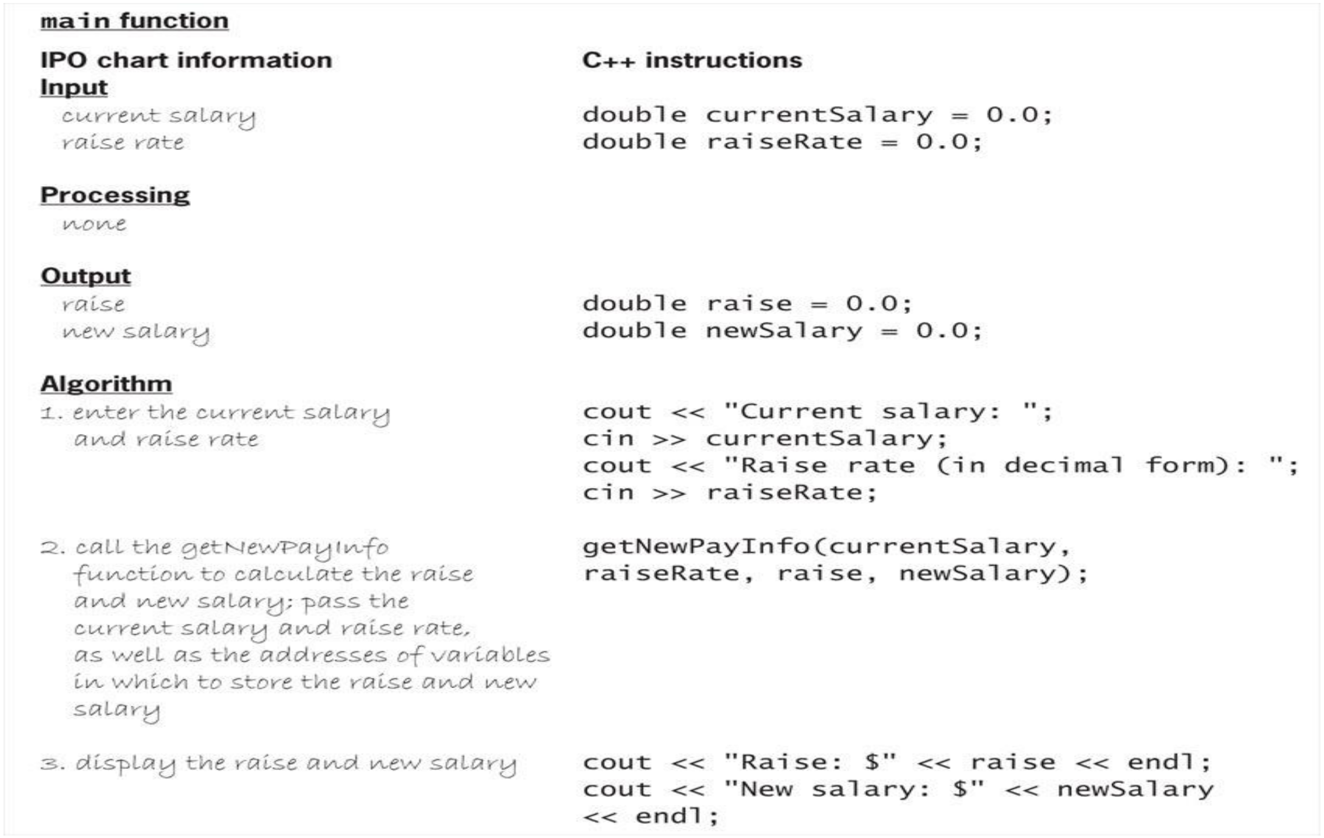
```
How old are you? 28
You are 28 years old.
Press any key to continue . . . _
```

Figure 10-20 Sample run of the modified age message program

The Salary Program

- Program that allows the user to enter an employee’s current salary and raise rate
- Computes the employee’s raise and new salary
- Program makes use of a void function that is passed two variables *by value* and two *by reference*

The Salary Program (cont'd.)

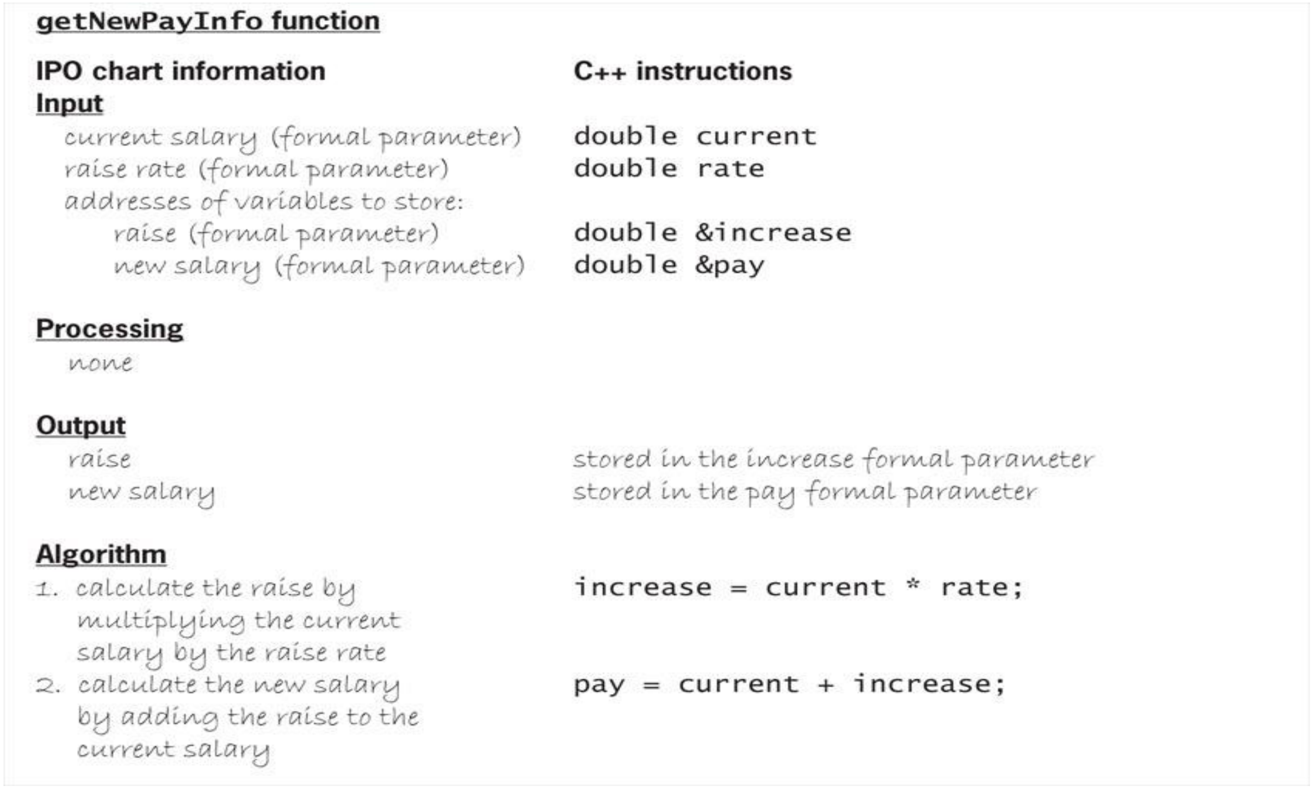


main function

IPO chart information	C++ instructions
Input	
current salary	`double currentSalary = 0.0;`
raise rate	`double raiseRate = 0.0;`
Processing	
none	
Output	
raise	`double raise = 0.0;`
new salary	`double newSalary = 0.0;`
Algorithm	
1. enter the current salary and raise rate	`cout << "Current salary: ";` `cin >> currentSalary;` `cout << "Raise rate (in decimal form): ";` `cin >> raiseRate;`
2. call the getNewPayInfo function to calculate the raise and new salary; pass the current salary and raise rate, as well as the addresses of variables in which to store the raise and new salary	`getNewPayInfo(currentSalary, raiseRate, raise, newSalary);`
3. display the raise and new salary	`cout << "Raise: $" << raise << endl;` `cout << "New salary: $" << newSalary << endl;`

Figure 10-22 IPO chart information and C++ instructions for the salary program

The Salary Program (cont'd.)

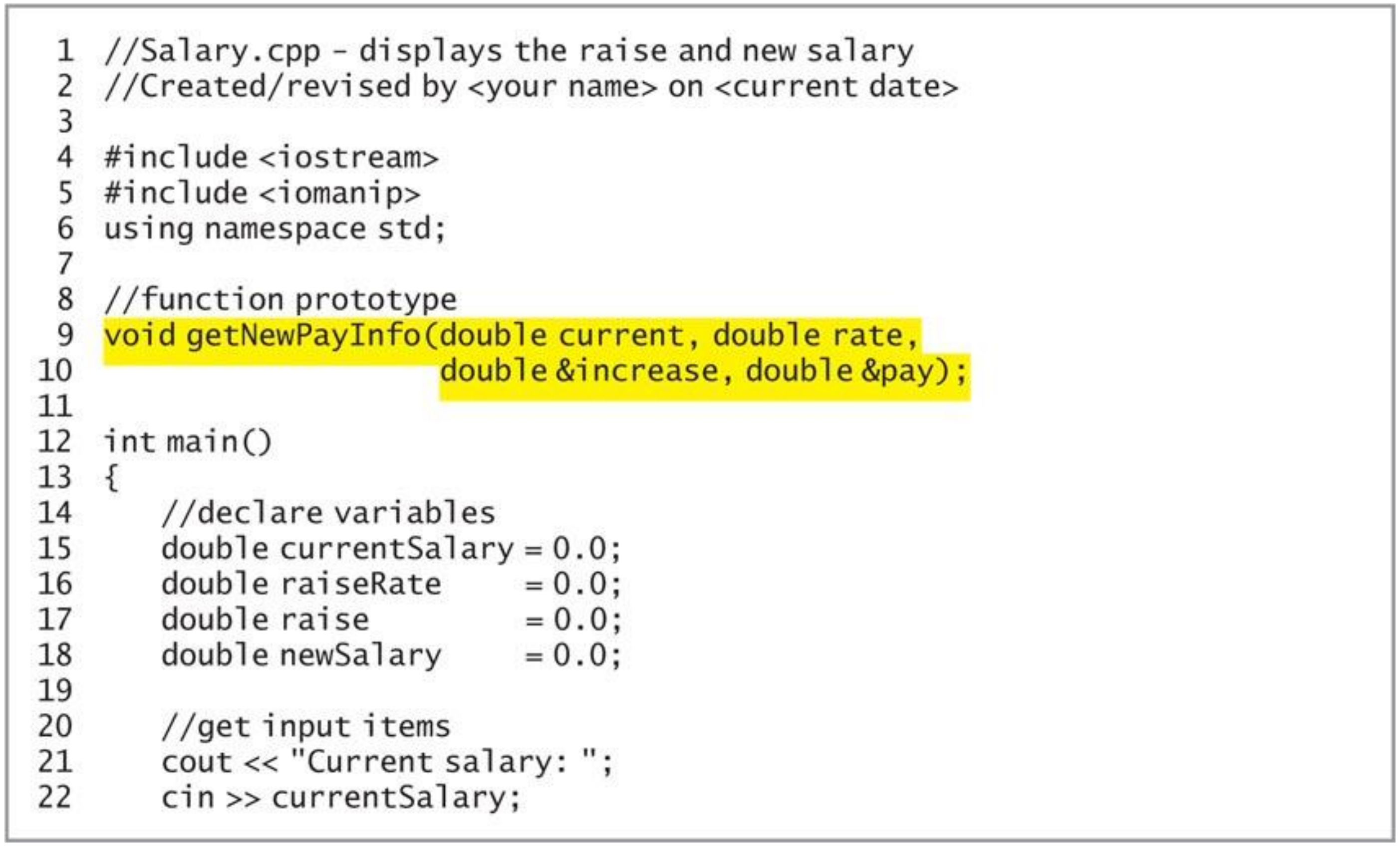


getNewPayInfo function

IPO chart information	**C++ instructions**
Input	
current salary (formal parameter)	`double current`
raise rate (formal parameter)	`double rate`
addresses of variables to store:	
raise (formal parameter)	`double &increase`
new salary (formal parameter)	`double &pay`
Processing	
none	
Output	
raise	stored in the increase formal parameter
new salary	stored in the pay formal parameter
Algorithm	
1. calculate the raise by multiplying the current salary by the raise rate	`increase = current * rate;`
2. calculate the new salary by adding the raise to the current salary	`pay = current + increase;`

Figure 10-22 IPO chart information and C++ instructions for the salary program (cont'd.)

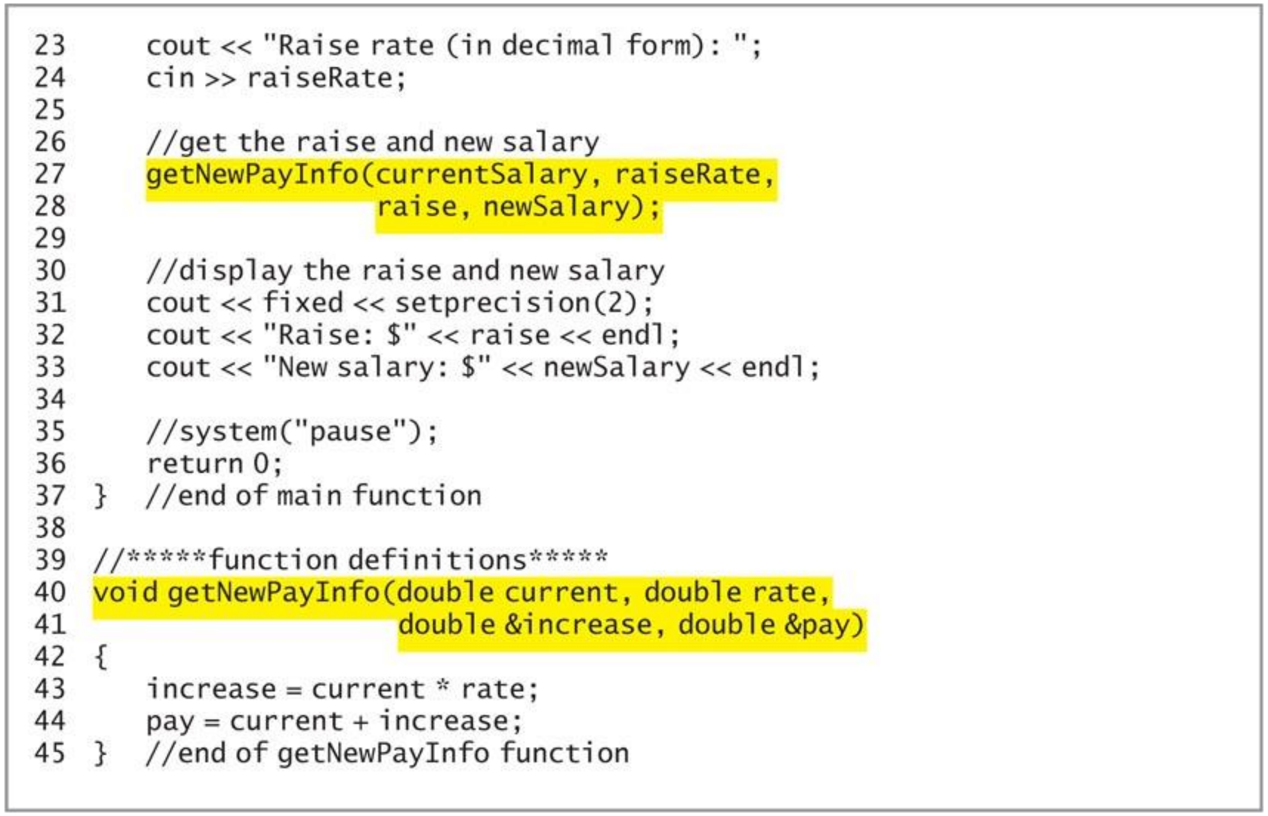
The Salary Program (cont'd.)



```
//Salary.cpp - displays the raise and new salary
//Created/revised by <your name> on <current date>

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

//function prototype
void getNewPayInfo(double current, double rate,
                   double &increase, double &pay);

int main()
{
    //declare variables
    double currentSalary = 0.0;
    double raiseRate     = 0.0;
    double raise         = 0.0;
    double newSalary     = 0.0;

    //get input items
    cout << "Current salary: ";
    cin >> currentSalary;
```

Figure 10-23 Salary program

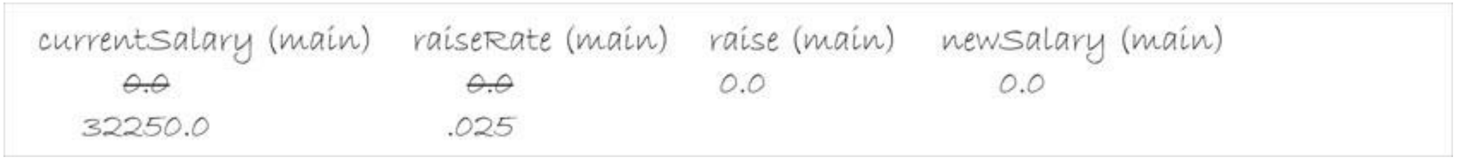
The Salary Program (cont'd.)



```
    cout << "Raise rate (in decimal form): ";
    cin >> raiseRate;

    //get the raise and new salary
    getNewPayInfo(currentSalary, raiseRate,
                  raise, newSalary);

    //display the raise and new salary
    cout << fixed << setprecision(2);
    cout << "Raise: $" << raise << endl;
    cout << "New salary: $" << newSalary << endl;

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

//*****function definitions*****
void getNewPayInfo(double current, double rate,
                   double &increase, double &pay)
{
    increase = current * rate;
    pay = current + increase;
}   //end of getNewPayInfo function
```

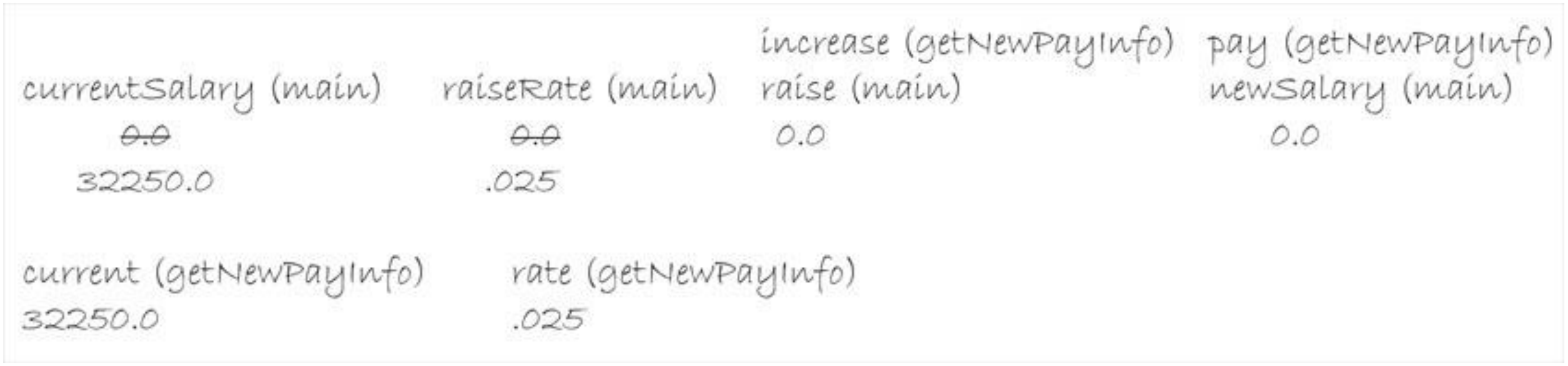
Figure 10-23 Salary program (cont'd.)

The Salary Program (cont'd.)



currentSalary (main)	raiseRate (main)	raise (main)	newSalary (main)
~~0.0~~	~~0.0~~	0.0	0.0
32250.0	.025		

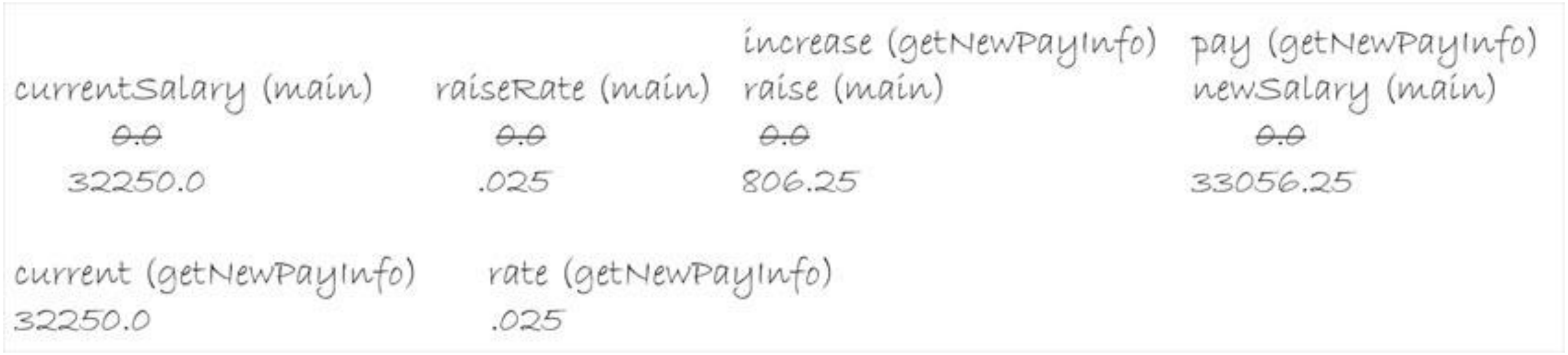
Figure 10-24 Desk-check table after the statements on lines 15 through 24 are processed



currentSalary (main)	raiseRate (main)	increase (getNewPayInfo) raise (main)	pay (getNewPayInfo) newSalary (main)
~~0.0~~	~~0.0~~	0.0	0.0
32250.0	.025		
current (getNewPayInfo)	rate (getNewPayInfo)		
32250.0	.025		

Figure 10-25 Desk-check table after the computer processes the `getNewPayInfo` function header

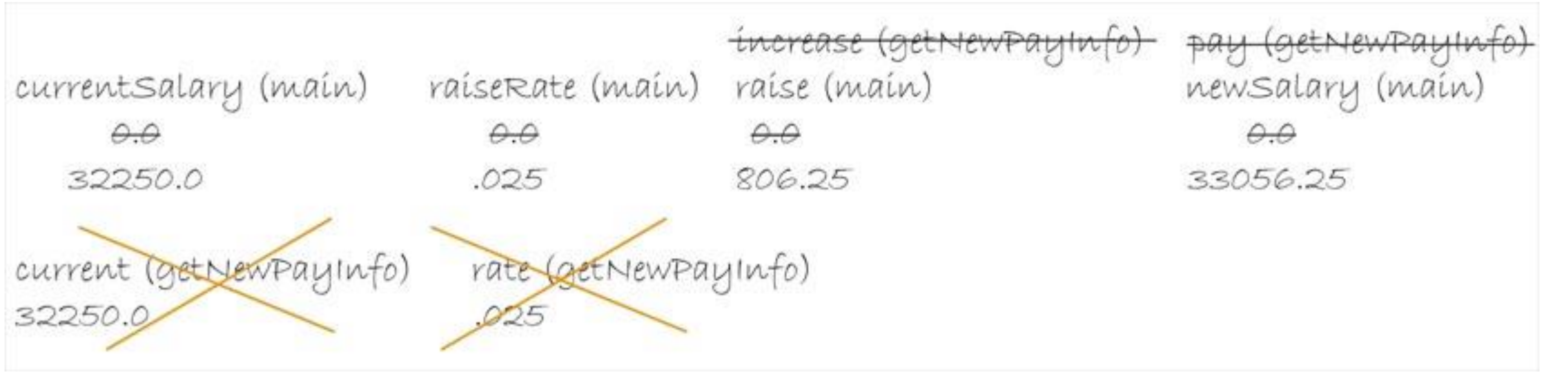
The Salary Program (cont'd.)



currentSalary (main)	raiseRate (main)	increase (getNewPayInfo) raise (main)	pay (getNewPayInfo) newSalary (main)
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
32250.0	.025	806.25	33056.25

current (getNewPayInfo)	rate (getNewPayInfo)
32250.0	.025

Figure 10-26 Desk-check table after the computer processes the statements in the `getNewPayInfo` function body

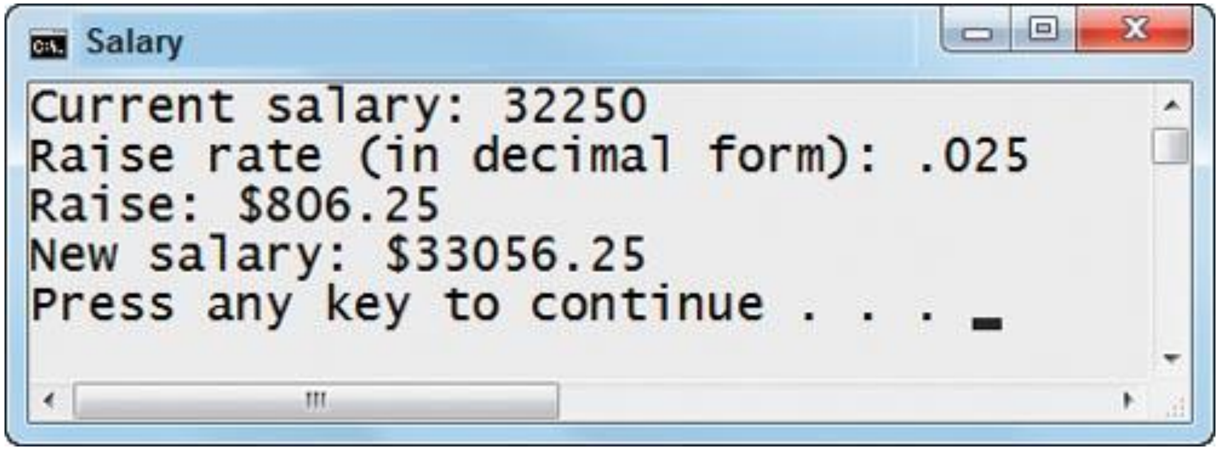


currentSalary (main)	raiseRate (main)	~~increase (getNewPayInfo)~~ raise (main)	~~pay (getNewPayInfo)~~ newSalary (main)
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
32250.0	.025	806.25	33056.25

current (getNewPayInfo)	rate (getNewPayInfo)
32250.0	.025

Figure 10-27 Desk-check table after the `getNewPayInfo` function ends

The Salary Program (cont’d.)



```
Current salary: 32250
Raise rate (in decimal form): .025
Raise: $806.25
New salary: $33056.25
Press any key to continue . . .
```

Figure 10-28 Sample run of the salary program

Summary

- All functions are either void or value-returning
- Value-returning functions return one value
- Void functions do not return a value
- Function header of a void function begins with the keyword `void` instead of a return data type
- Function body of a void function does not contain a `return` statement
- You call a void function by including its name and actual arguments in a statement

Summary (cont'd.)

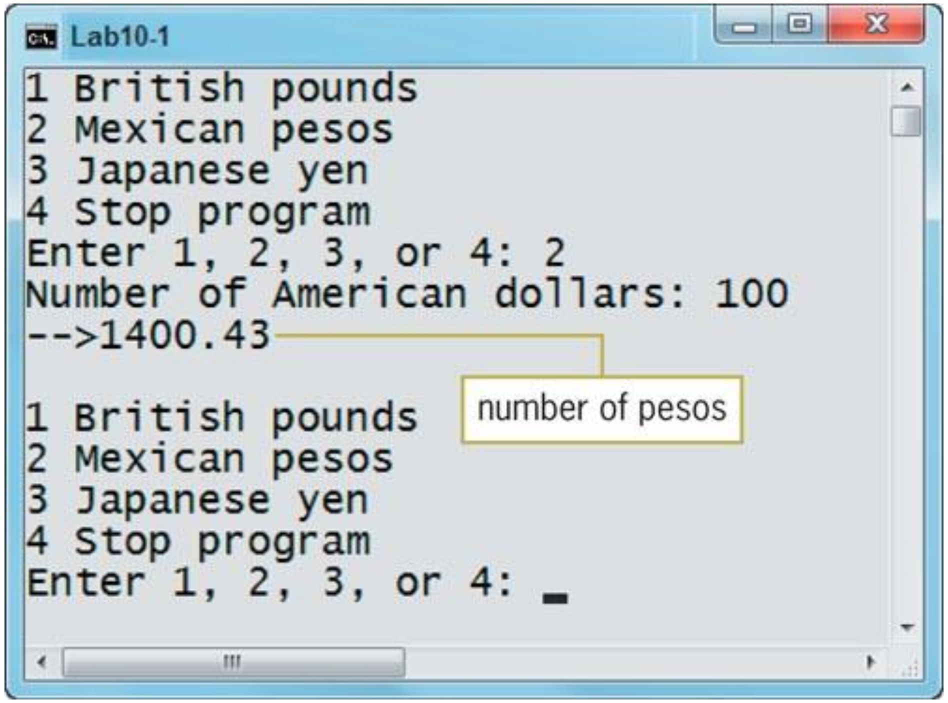
- A call to a void function appears as a statement by itself rather than as part of another statement
- Variables can be passed to functions either *by value* (the default) or *by reference*
- When a variable is passed *by value*, only a copy of the variable's value is passed
  - Receiving function is not given access to the variable, so it cannot change the variable's contents
  - Computer uses data type and name of formal parameter to store a copy of the value

Summary (cont'd.)

- When a variable is passed *by reference*, the variable's address in memory is passed
  - Receiving function can change variable's contents
  - Computer assigns name of formal parameter to memory location – variable then has two names
- To pass *by reference* you include the address-of operator (&) before the name of the formal parameter in function header
- If function appears below `main`, you must also include the & in the function's prototype

Lab 10-1: Stop and Analyze

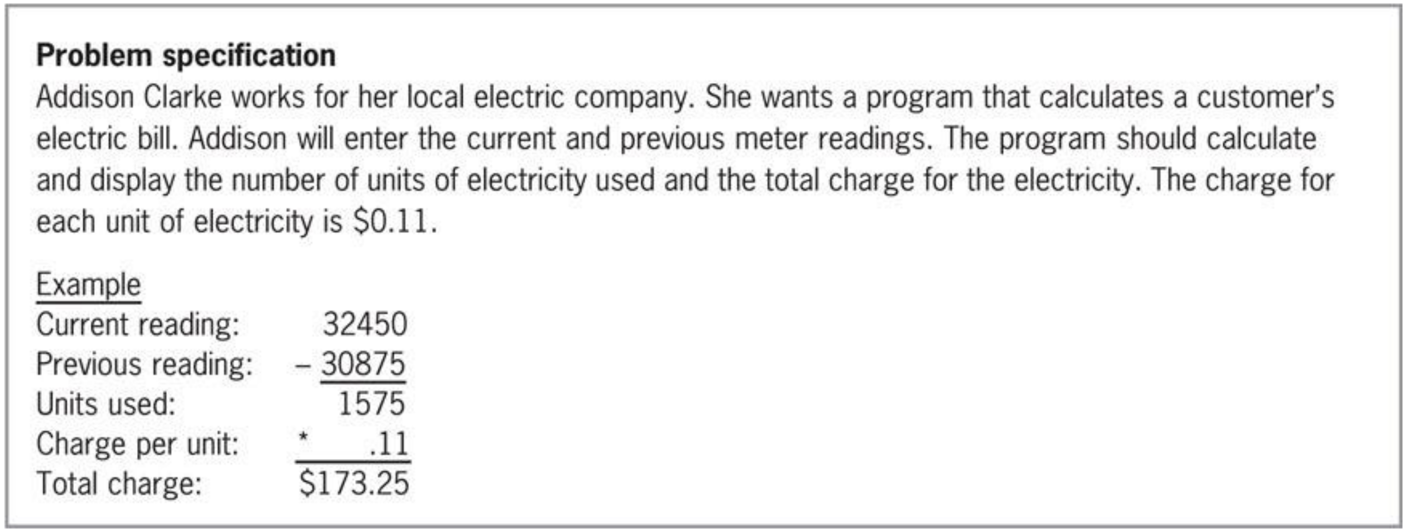
- Study the code in Figure 10-30 and then answer the questions (sample run below)



```
1 British pounds
2 Mexican pesos
3 Japanese yen
4 Stop program
Enter 1, 2, 3, or 4: 2
Number of American dollars: 100
-->1400.43

1 British pounds
2 Mexican pesos
3 Japanese yen
4 Stop program
Enter 1, 2, 3, or 4: _
```

Figure 10-29 Sample run of program for Lab 10-1

Lab 10-2: Plan and Create



Problem specification

Addison Clarke works for her local electric company. She wants a program that calculates a customer's electric bill. Addison will enter the current and previous meter readings. The program should calculate and display the number of units of electricity used and the total charge for the electricity. The charge for each unit of electricity is $0.11.

Example

Current reading:	32450
Previous reading:	– 30875
Units used:	1575
Charge per unit:	* .11
Total charge:	$173.25

Figure 10-31 Problem specification and a sample calculation for Lab 10-2

Lab 10-3: Modify

- Make a copy of Lab 10-2 to modify
- Current version uses one void function to calculate both the number of units used and the total charge
- Replace the `calcBill` functions with two functions:
  - A void function `getUnits` that calculates the total number of units used
  - A value-returning function `getTotal` that calculates and returns the total charge
- Test the program appropriately

Lab 10-4: Desk-Check

- Desk-check the code in Figure 10-37 using the following four sets of test scores:
  - 78 and 85
  - 45 and 93
  - 87 and 98
  - 54 and 32

Lab 10-5: Debug

- Run the program in the Lab10-5.cpp file
- Enter the following scores: 93, 90, 85, and 100
- The program should display 368 as the total points and A as the grade
- Debug the program