Chapter 6

Functions

6.1

Modular Programming

Modular Programming

- Modular programming: breaking a program up into smaller, manageable functions or modules

- Function: a collection of statements to perform a task
- Motivation for modular programming:
  - Improves maintainability of programs
  - Simplifies the process of writing programs

This program has one long, complex function containing all of the statements necessary to solve a problem.

```
int main()
{
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
}
```

In this program the problem has been divided into smaller problems, each of which is handled by a separate function.

```
int main()
{
   statement;
   statement;
   statement;
}
```

main function

```
void function2()
{
   statement;
   statement;
   statement;
}
```

function 2

```
void function3()
{
   statement;
   statement;
   statement;
}
```

function 3

```
void function4()
{
   statement;
   statement;
   statement;
}
```

function 4

6.2

Defining and Calling Functions

Defining and Calling Functions

- Function call: statement causes a function to execute
- Function definition: statements that make up a function

Function Definition

- Definition includes:
  - return type: data type of the value that function returns to the part of the program that called it
  - name: name of the function. Function names follow same rules as variables
  - parameter list: variables containing values passed to the function
  - body: statements that perform the function's task, enclosed in { }

Function Definition

Return type

Function name

Parameter list (This one is empty)

Function body

```
int main ()
{
    cout << "Hello World\n";
    return 0;
}
```

Note: The line that reads `int main()` is the function header.

Function Return Type

- If a function returns a value, the type of the value must be indicated:

```
int main()
```

- If a function does not return a value, its return type is `void`:

```
void printHeading()
{
        cout << "Monthly Sales\n";
}
```

Calling a Function

- To call a function, use the function name followed by `()` and `;`

```
printHeading();
```

- When called, program executes the body of the called function

- After the function terminates, execution resumes in the calling function at point of call.

Program 6-1

```
// This program has two functions: main and displayMessage
#include <iostream>
using namespace std;

//****************************************
// Definition of function displayMessage  *
// This function displays a greeting.     *
//****************************************

void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}

//****************************************
// Function main                          *
//****************************************

int main()
{
   cout << "Hello from main.\n";
   displayMessage();
   cout << "Back in function main again.\n";
   return 0;
}
```

Program Output

```
Hello from main.
Hello from the function displayMessage.
Back in function main again.
```

Flow of Control in Program 6-1

```
void displayMessage()
{
    cout << "Hello from the function displayMessage.\n";
}
```

```
int main()
{
    cout << "Hello from main.\n"
    displayMessage();
    cout << "Back in function main again.\n";
    return;
}
```

Calling Functions

- `main` can call any number of functions
- Functions can call other functions
- Compiler must know the following about a function before it is called:
  - name
  - return type
  - number of parameters
  - data type of each parameter

Program 6-4

```
int main()
{
   cout << "I am starting in function main.\n";
   deep();     // Call function deep
   cout << "Back in function main again.\n";
   return 0;
}
```

```
// This program has three functions: main, deep,
//and deeper
#include <iostream>
using namespace std;

void deeper()
{
   cout << "I am now inside the function deeper.\n";
}
void deep()
{
   cout << "I am now inside the function deep.\n";
   deeper();   // Call function deeper
   cout << "Now I am back in deep.\n";
}
```

6.3

Function Prototypes

Function Prototypes

- Ways to notify the compiler about a function before a call to the function:
  - Place function definition before calling function's definition
  - Use a <u>function prototype</u> (<u>function declaration</u>) – like the function definition without the body
    - Header: `void printHeading()`
    - Prototype: `void printHeading();`

Program 6-5

```
1   // This program has three functions: main, First, and Second.
2   #include <iostream>
3   using namespace std;
4
5   // Function Prototypes
6   void first();
7   void second();
8
9   int main()
10  {
11     cout << "I am starting in function main.\n";
12     first();     // Call function first
13     second();    // Call function second
14     cout << "Back in function main again.\n";
15     return 0;
16  }
17
```

(Program Continues)

Program 6-5 (Continued)

```
18  //*************************************
19  // Definition of function first.      *
20  // This function displays a message.  *
21  //*************************************
22
23  void first()
24  {
25     cout << "I am now inside the function first.\n";
26  }
27
28  //*************************************
29  // Definition of function second.     *
30  // This function displays a message.  *
31  //*************************************
32
33  void second()
34  {
35     cout << "I am now inside the function second.\n";
36  }
```

Prototype Notes

- Place prototypes near top of program
- Program must include either prototype or full function definition before any call to the function – compiler error otherwise
- When using prototypes, can place function definitions in any order in source file

6.4

Sending Data into a Function

Sending Data into a Function

- Can pass values into a function at time of call:

```
c = pow(a, b);
```

- Values passed to function are <u>arguments</u>
- Variables in a function that hold the values passed as arguments are <u>parameters</u>

A Function with a Parameter Variable

```
void displayValue(int num)
{
    cout << "The value is " << num << endl;
}
```

The integer variable `num` is a parameter.
It accepts any integer value passed to the function.

Program 6-6

```
1  // This program demonstrates a function with a parameter.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototype
6  void displayValue(int);
7
8  int main()
9  {
10    cout << "I am passing 5 to displayValue.\n";
11    displayValue(5);  // Call displayValue with argument 5
12    cout << "Now I am back in main.\n";
13    return 0;
14 }
15
```

(Program Continues)

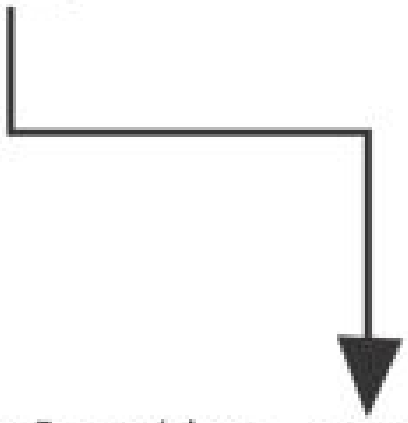
Program 6-6 *(continued)*

```
16  //*****************************************************
17  // Definition of function displayValue.               *
18  // It uses an integer parameter whose value is displayed. *
19  //*****************************************************
20
21  void displayValue(int num)
22  {
23     cout << "The value is " << num << endl;
24  }
```

Program Output

```
I am passing 5 to displayValue.
The value is 5
Now I am back in main.
```

```
displayValue(5);
```



```
void displayValue(int num)
{
  cout << "The value is " << num << endl;
}
```

The function call in line 11 passes the value 5 as an argument to the function.

Other Parameter Terminology

- A parameter can also be called a <u>formal parameter</u> or a <u>formal argument</u>
- An argument can also be called an <u>actual parameter</u> or an <u>actual argument</u>

Parameters, Prototypes, and Function Headers

- For each function argument,
  - the prototype must include the data type of each parameter inside its parentheses
  - the header must include a **declaration** for each parameter in its `()`

```
void evenOrOdd(int);        //prototype
void evenOrOdd(int num)  //header
evenOrOdd(val);             //call
```

Function Call Notes

- Value of argument is copied into parameter when the function is called
- A parameter's scope is the function which uses it
- There must be a data type listed in the prototype （）and an argument declaration in the function header （）for each parameter

Program 6-7

```
#include <iostream>
using namespace std
// Function Prototype

void displayValue(int);

int main()
{
   cout << "I am passing several values to displayValue.\n";
   displayValue(5);  // Call displayValue with argument 5
   displayValue(10); // Call displayValue with argument 10
   displayValue(2);  // Call displayValue with argument 2
   displayValue(16); // Call displayValue with argument 16
   cout << "Now I am back in main.\n";
   return 0;
}
```

```
void displayValue(int num)
{
   cout << "The value is " << num <<
endl;
}
```

Function Call Notes

- Arguments will be promoted/demoted as necessary to match parameters
- Function can have multiple parameters

Passing Multiple Arguments

When calling a function and passing multiple arguments:

- the number of arguments in the call must match the prototype and definition
- the first argument will be used to initialize the first parameter, the second argument to initialize the second parameter, etc.

Program 6-8

```
1  // This program demonstrates a function with three parameters.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototype
6  void showSum(int, int, int);
7
8  int main()
9  {
10    int value1, value2, value3;
11
12    // Get three integers.
13    cout << "Enter three integers and I will display ";
14    cout << "their sum: ";
15    cin >> value1 >> value2 >> value3;
16
17    // Call showSum passing three arguments.
18    showSum(value1, value2, value3);
19    return 0;
20 }
21
```

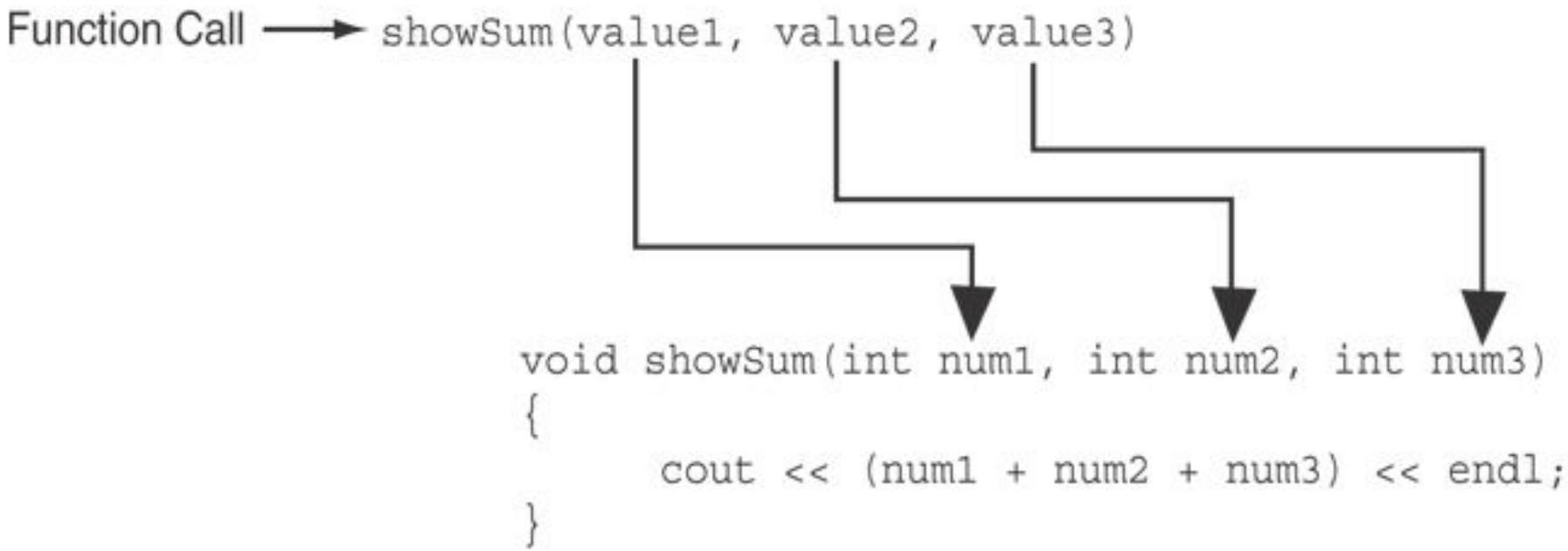
(Program Continues)

Program 6-8 (Continued)

```
22  //*************************************************************
23  // Definition of function showSum.                            *
24  // It uses three integer parameters. Their sum is displayed. *
25  //*************************************************************
26
27  void showSum(int num1, int num2, int num3)
28  {
29     cout << (num1 + num2 + num3) << endl;
30  }
```

Program Output with Example Input Shown in Bold

```
Enter three integers and I will display their sum: 4 8 7 [Enter]
19
```



The function call in line 18 passes value1, value2, and value3 as a arguments to the function.

6.5

Passing Data by Value

Passing Data by Value

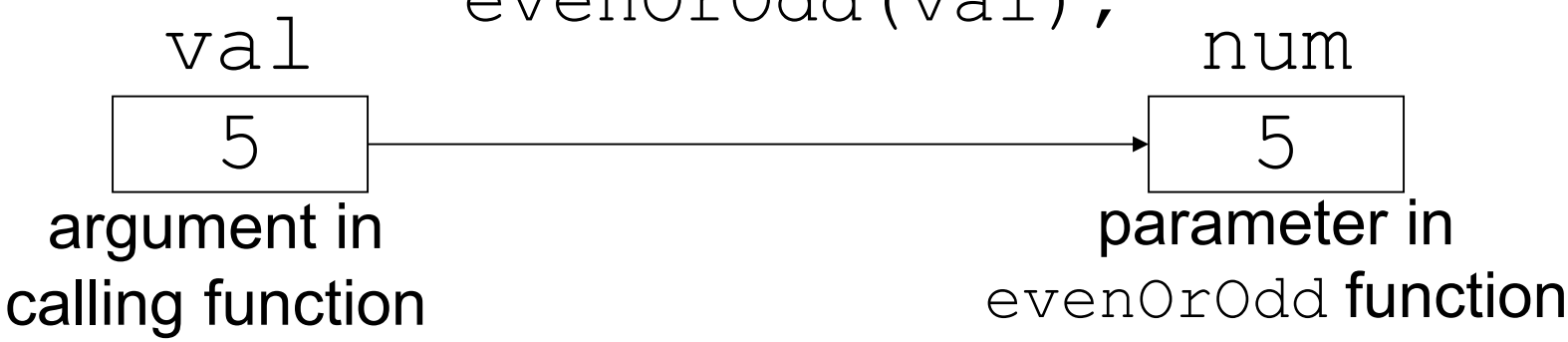
- Pass by value: when an argument is passed to a function, its value is copied into the parameter.

- Changes to the parameter in the function do not affect the value of the argument

Passing Information to Parameters by Value

- Example: `int val=5;`

  `evenOrOdd(val);`



- `evenOrOdd` can change variable `num`, but it will have no effect on variable `val`

```
// This program demonstrates that changes to a
function parameter
// have no affect on the original argument.
#include <iostream>
using namespace std;
// Function Prototype
void changeMe(int);

int main()
{
   int number = 12;
   // Display the value in number.
   cout << "number is " << number << endl;
   // Call changeMe, passing the value in number
   // as an argument.
   changeMe(number);
   // Display the value in number again.
   cout << "Now back in main again, the value of ";
   cout << "number is " << number << endl;
   return 0;
}
```

```
//*********************************************************
// Definition of function changeMe.
// This function changes the value of the parameter
// myValue.
//*********************************************************

void changeMe(int myValue)
{
   // Change the value of myValue to 0.
   myValue = 0;

   // Display the value in myValue.
   cout << "Now the value is " << myValue << endl;
}
```

OUTPUT:

```
number is 12
Now the value is 0
Now back in main again, the value of number is 12
```

6.6

Using Functions in Menu-Driven Programs

Using Functions in Menu-Driven Programs

- Functions can be used
  - to implement user choices from menu
  - to implement general-purpose tasks:
    - Higher-level functions can call general-purpose functions, minimizing the total number of functions and speeding program development time

Program 6-10

```
// This is a menu-driven program that makes a function call
// for each selection the user makes.
#include <iostream>
#include <iomanip>
using namespace std;

// Function prototypes
void showMenu();
void showFees(double, int);

int main()
{
   int choice;       // To hold a menu choice
   int months;       // To hold a number of months

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

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

Program 6-10 continued

```
do
 {
   // Display the menu and get the user's choice.
   showMenu();
   cin >> choice;

   // Validate the menu selection.
   while (choice < 1 || choice > 4)
   {
      cout << "Please enter 1, 2, 3, or 4: ";
      cin >> choice;
   }

   if (choice != 4)
   {
      // Get the number of months.
      cout << "For how many months? ";
      cin >> months;
```

Program 6-10 continued

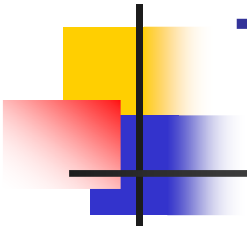
```
// Display the membership fees.
        switch (choice)
        {
          case 1:  showFees(ADULT, months);
                   break;
          case 2:  showFees(CHILD, months);
                   break;
          case 3:  showFees(SENIOR, months);
        }
      }
   } while (choice != 4);
   return 0;
}
```

Program 6-10

```
// Definition of function showMenu which displays the menu.

void showMenu()
{
   cout << "\n\t\tHealth Club Membership Menu\n\n";
   cout << "1. Standard Adult Membership\n";
   cout << "2. Child Membership\n";
   cout << "3. Senior Citizen Membership\n";
   cout << "4. Quit the Program\n\n";
   cout << "Enter your choice: ";
}

//*****************************************************************
// Definition of function showFees. The memberRate parameter      *
// the monthly membership rate and the months parameter holds the *
// number of months. The function displays the total charges.     *
//*****************************************************************

void showFees(double memberRate, int months)
{
    cout << "The total charges are $"
         << (memberRate * months) << endl;
}
```

6.7

The `return` Statement



The `return` Statement

- Used to end execution of a function
- Can be placed anywhere in a function
  - Statements that follow the `return` statement will not be executed
- Can be used to prevent abnormal termination of program
- In a `void` function without a `return` statement, the function ends at its last `}`

Program 6-11

```
1  // This program uses a function to perform division. If division
2  // by zero is detected, the function returns.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype.
7  void divide(double, double);
8
9  int main()
10 {
11    double num1, num2;
12
13    cout << "Enter two numbers and I will divide the first\n";
14    cout << "number by the second number: ";
15    cin >> num1 >> num2;
16    divide(num1, num2);
17    return 0;
18 }
```

(Program Continues)

Program 6-11(Continued)

```
20  //*************************************************************
21  // Definition of function divide.                            *
22  // Uses two parameters: arg1 and arg2. The function divides arg1*
23  // by arg2 and shows the result. If arg2 is zero, however, the  *
24  // function returns.                                         *
25  //*************************************************************
26
27  void divide(double arg1, double arg2)
28  {
29     if (arg2 == 0.0)
30     {
31        cout << "Sorry, I cannot divide by zero.\n";
32        return;
33     }
34     cout << "The quotient is " << (arg1 / arg2) << endl;
35  }
```

Program Output with Example Input Shown in Bold

```
Enter two numbers and I will divide the first
number by the second number: 12 0 [Enter]
Sorry, I cannot divide by zero.
```

6.8

Returning a Value From a Function

Returning a Value From a Function

- A function can return a value back to the statement that called the function.
- You've already seen the `pow` function, which returns a value:

```
double x;
x = pow(2.0, 10.0);
```

Returning a Value From a Function

- In a value-returning function, the `return` statement can be used to return a value from function to the point of call. Example:

```
int sum(int num1, int num2)
{
  double result;
  result = num1 + num2;
  return result;
}
```

A Value-Returning Function

Return Type

```
int sum(int num1, int num2)
{
    double result;
    result = num1 + num2;
    return result;
}
```

Value Being Returned

A Value-Returning Function

```
int sum(int num1, int num2)
{
    return num1 + num2;
}
```

Functions can return the values of expressions, such as `num1 + num2`

Program 6-12

```
1  // This program uses a function that returns a value.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  int sum(int, int);
7
8  int main()
9  {
10    int value1 = 20,   // The first value
11        value2 = 40,   // The second value
12        total;         // To hold the total
13
14    // Call the sum function, passing the contents of
15    // value1 and value2 as arguments. Assign the return
16    // value to the total variable.
17    total = sum(value1, value2);
18
19    // Display the sum of the values.
20    cout << "The sum of " << value1 << " and "
21         << value2 << " is " << total << endl;
22    return 0;
23 }
```

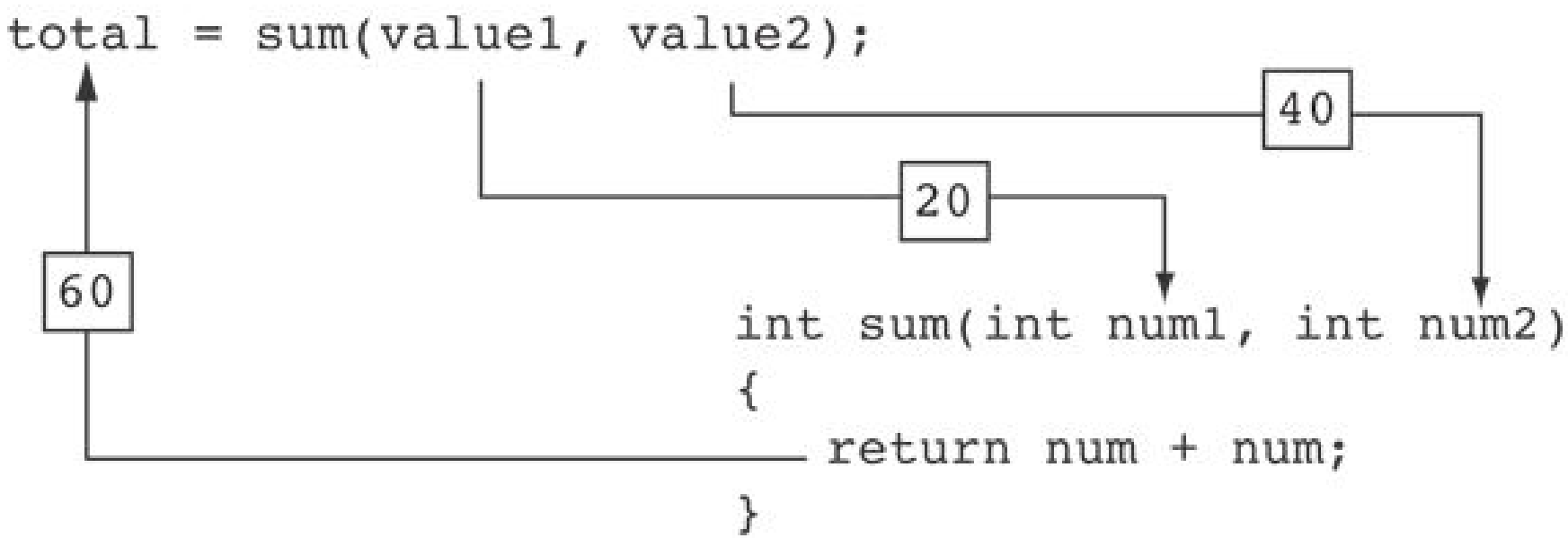
(Program Continues)

Program 6-12 (Continued)

```
24
25  //*****************************************************
26  // Definition of function sum. This function returns  *
27  // the sum of its two parameters.                     *
28  //*****************************************************
29
30  int sum(int num1, int num2)
31  {
32     return num1 + num2;
33  }
```

Program Output

```
The sum of 20 and 40 is 60
```



The statement in line 17 calls the sum function, passing `value1` and `value2` as arguments.
The return value is assigned to the `total` variable.

Another Example, from Program 6-13

```
area = PI * square(radius);
```

10

100

```
double square(double number)
{
    return number * number;
}
```

Returning a Value From a Function

- The prototype and the definition must indicate the data type of return value (not `void`)
- Calling function should use return value:
  - assign it to a variable
  - send it to `cout`
  - use it in an expression

6.9

Returning a Boolean Value

Returning a Boolean Value

- Function can return `true` or `false`
- Declare return type in function prototype and heading as `bool`
- Function body must contain `return` statement(s) that return `true` or `false`
- Calling function can use return value in a relational expression

Program 6-14

```
1  // This program uses a function that returns true or false.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  bool isEven(int);
7
8  int main()
9  {
10    int val;
11
12    // Get a number from the user.
13    cout << "Enter an integer and I will tell you ";
14    cout << "if it is even or odd: ";
15    cin >> val;
16
```

(Program Continues)

Program 6-14 *(continued)*

```
17      // Indicate whether it is even or odd.
18      if (isEven(val))
19         cout << val << " is even.\n";
20      else
21         cout << val << " is odd.\n";
22      return 0;
23   }
24
25   //*****************************************************************
26   // Definition of function isEven. This function accepts an        *
27   // integer argument and tests it to be even or odd. The function  *
28   // returns true if the argument is even or false if the argument  *
29   // is odd. The return value is an bool.                           *
30   //*****************************************************************
31
32   bool isEven(int number)
33   {
34      bool status;
35
36      if (number % 2)
37         status = false;   // number is odd if there's a remainder.
38      else
39         status = true;    // Otherwise, the number is even.
40      return status;
41   }
```

Program Output with Example Input Shown in Bold

```
Enter an integer and I will tell you if it is even or odd: 5 [Enter]
5 is odd.
```