Chapter 6: Functions

Hanoi University of Science and Technology

December 3, 2025

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Six Edition, Pearson, 2009
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

Modular Programming

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;
   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;
}
```

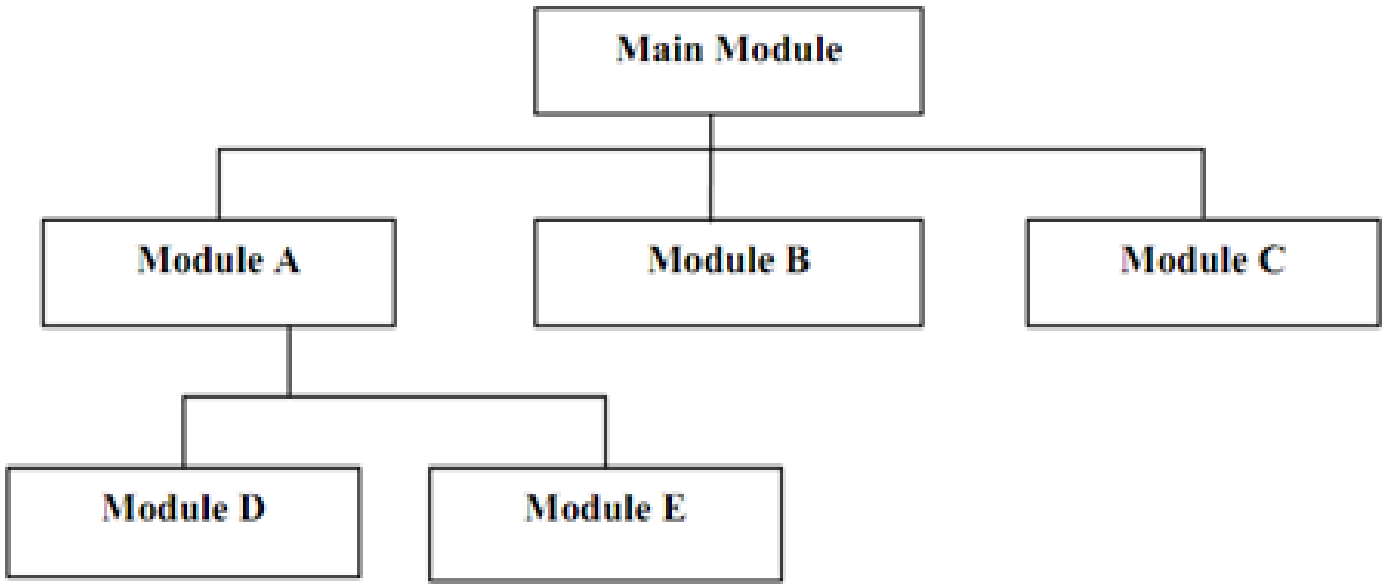
function 3

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

function 4

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



Defining and Calling Functions

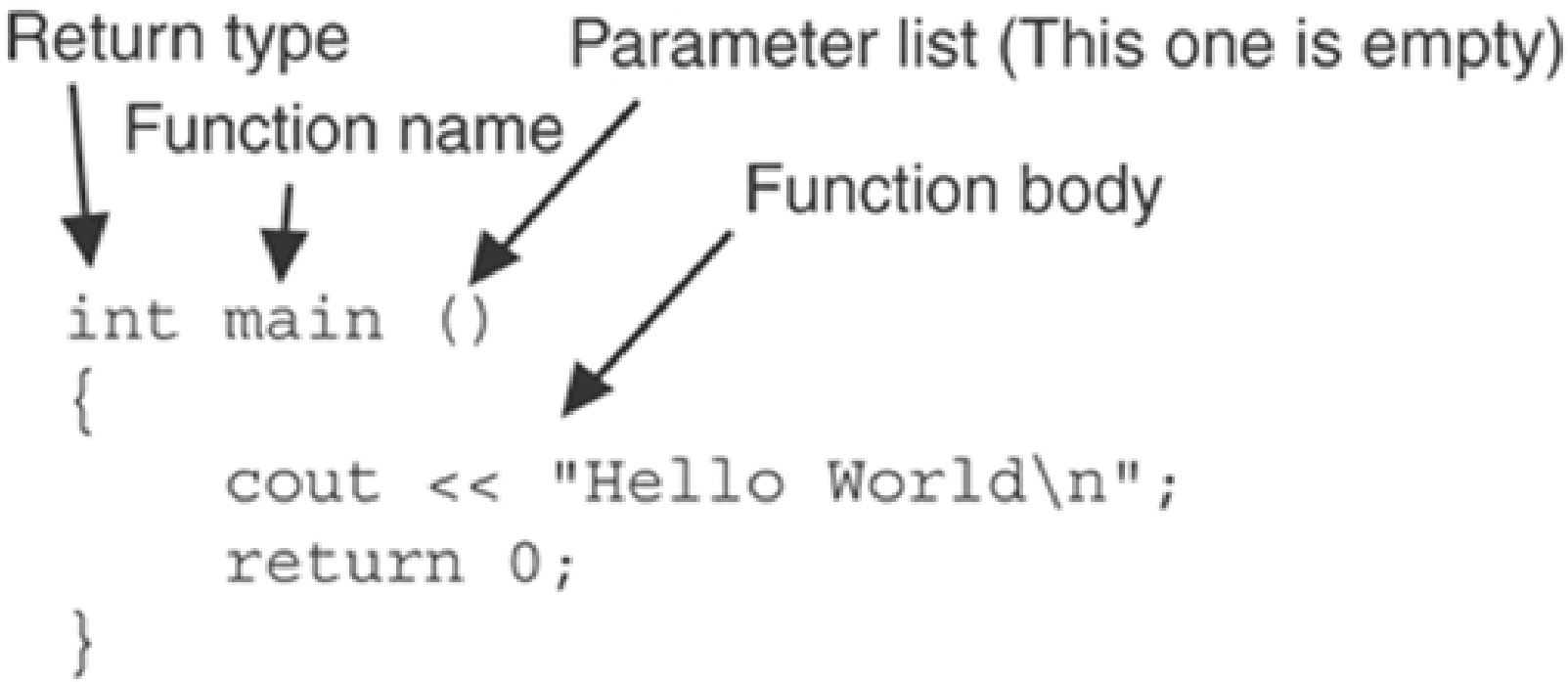
- Function call: statement causes a function to execute
- Function definition: statements that make up a function

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



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.

Functions in Program 6-1

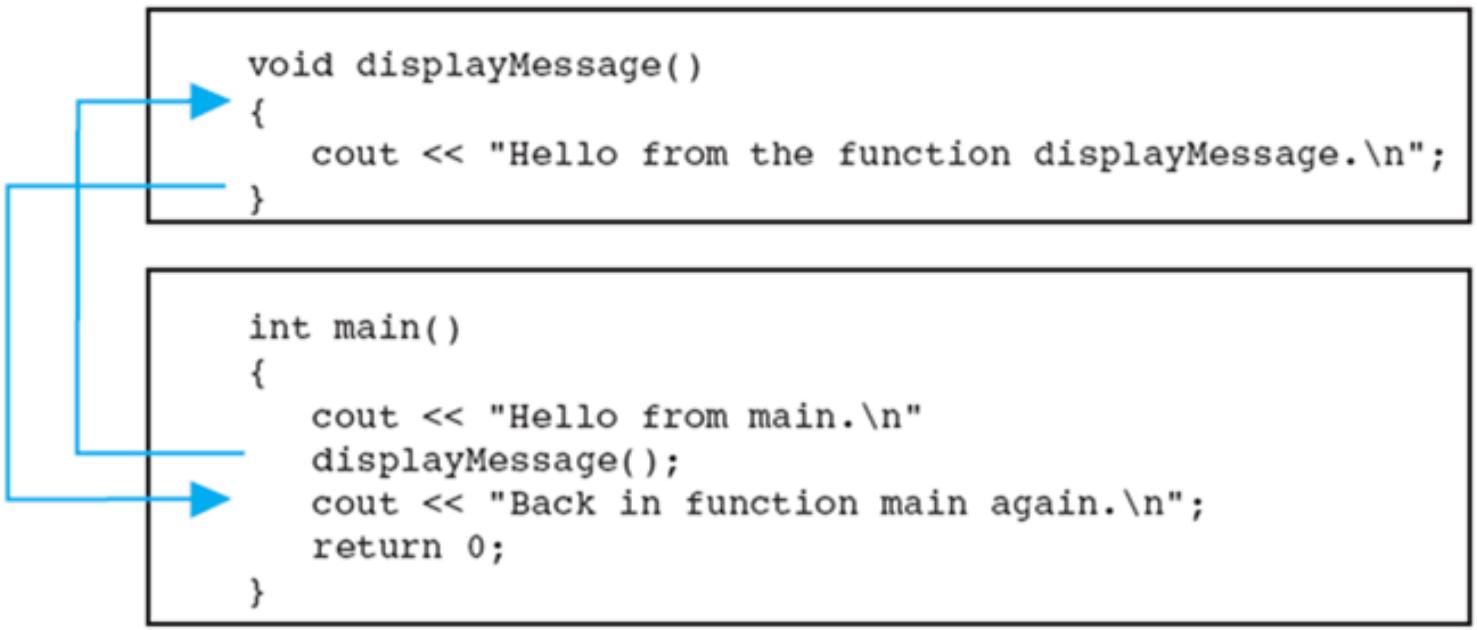
Program 6-1

```
1  // This program has two functions: main and displayMessage
2  #include <iostream>
3  using namespace std;
4
5  //*****************************************
6  // Definition of function displayMessage  *
7  // This function displays a greeting.     *
8  //*****************************************
9
10 void displayMessage()
11 {
12    cout << "Hello from the function displayMessage.\n";
13 }
14
15 //*****************************************
16 // Function main                          *
17 //*****************************************
18
19 int main()
20 {
21    cout << "Hello from main.\n";
22    displayMessage();
23    cout << "Back in function main again.\n";
24    return 0;
25 }
```

Program Output

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

Functions 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 0;
}
```

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

Function Prototypes

Ways to notify the compiler about a function before a call to the function:

- Place function definition before calling the function
- Use a function prototype (function declaration) – like the function definition without the body:
  - Header: `void printHeading()`
  - Prototype: `void printHeading();`

Function Prototypes

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
```

Function Prototypes

```
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

Sending Data into a Function

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

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

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

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.
  ```

A Function with a Parameter Variable

For each function argument:

- Prototype must include the data type of each parameter inside its parentheses
- Header must include a declaration for each parameter in its ()

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

A Function with a Parameter Variable

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
```

A Function with a Parameter Variable

Program 6-6 *(continued)*

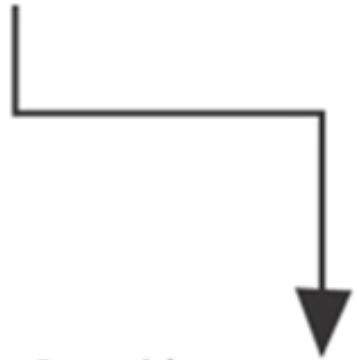
```
16  //*********************************************************
17  // Definition of function displayValue.                   *
18  // It uses an integer parameter whose value is displayed. *
19  //*********************************************************
20
21  void displayValue(int num)
22  {
23     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.
```

A Function with a Parameter Variable

```
displayValue(5);
```



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

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
- Function can have multiple parameters
- There must be a data type listed in the prototype () and an argument declaration in the function header () for each parameter

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.

Passing Multiple Arguments

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
```

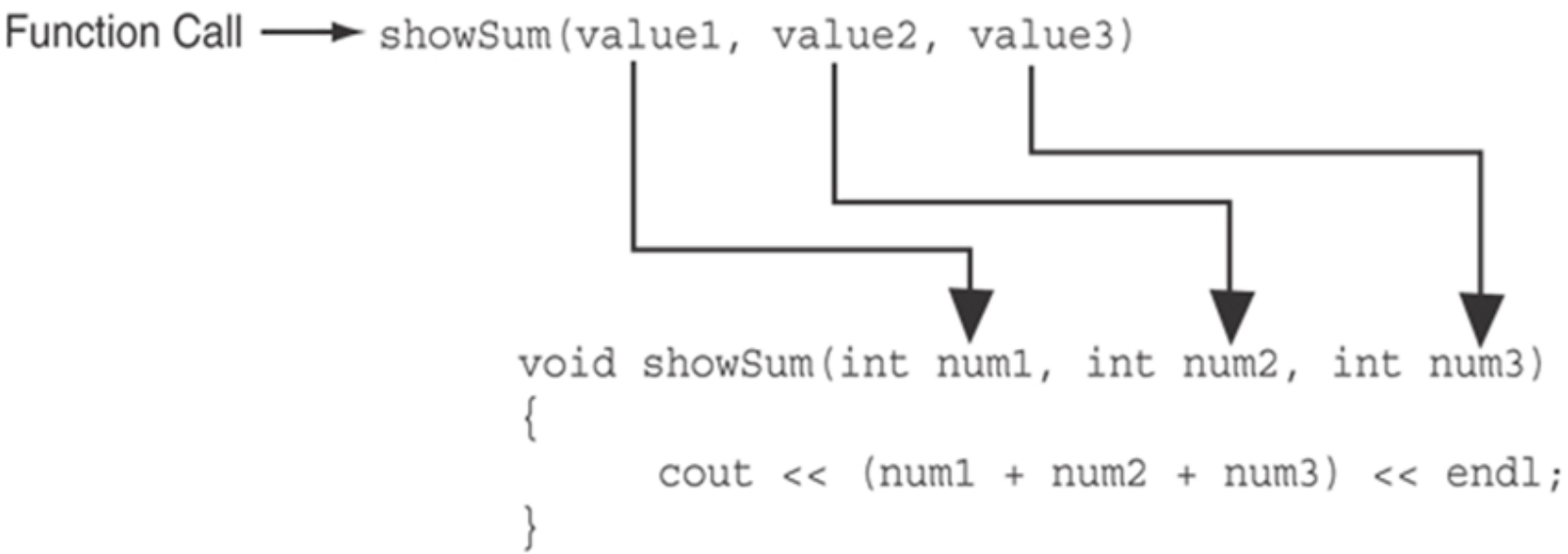
Passing Multiple Arguments

```
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
```

Passing Multiple Arguments



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

Program 6-9

```
1  // This program demonstrates that changes to a function paramete
2  // have no effect on the original argument.
3  #include <iostream>
4  using namespace std;
5
6  // Function Prototype
7  void changeMe(int);
8
9  int main()
10 {
11     int number = 12;
12
13     // Display the value in number.
14     cout << "number is " << number << endl;
15
16     // Call changeMe, passing the value in number
17     // as an argument.
18     changeMe(number);
19
20     // Display the value in number again.
21     cout << "Now back in main again, the value of ";
22     cout << "number is " << number << endl;
23     return 0;
24 }
25
26 //*************************************************************
27 // Definition of function changeMe.                           *
28 // This function changes the value of the parameter myValue.  *
29 //*************************************************************
```

Passing Information to Parameters by Value

```
30
31  void changeMe(int myValue)
32  {
33       // Change the value of myValue to 0.
34       myValue = 0;
35
36       // Display the value in myValue.
37       cout << "Now the value is " << myValue << endl;
38  }
```

Program Output

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

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
- In a `void` function without a `return` statement, the function ends at its last }

The return Statement

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 }
```

The return Statement

```
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.
```

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:

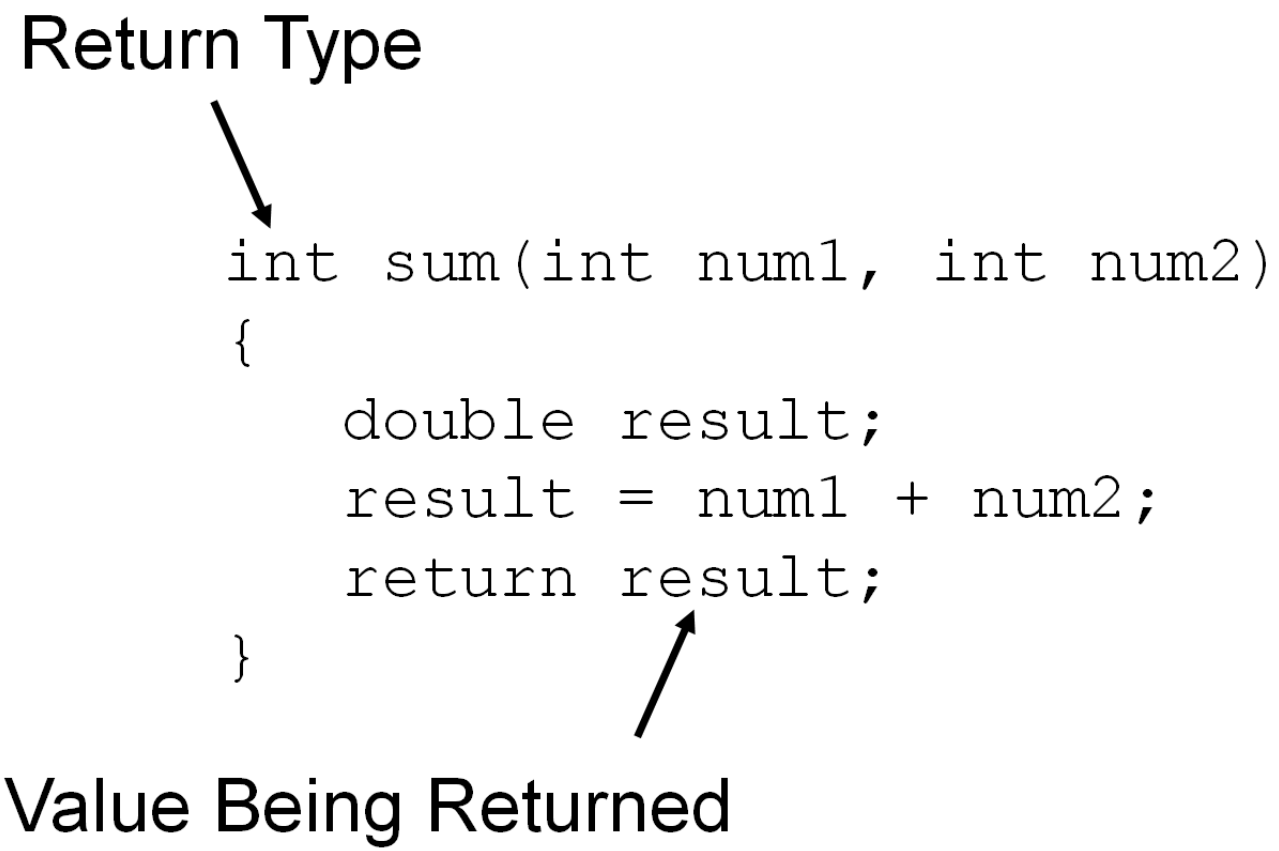
```
double x;
x = pow(2.0, 10.0);
```

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;
}
```

The return Statement



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

Returning a Value From a Function

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 }
```

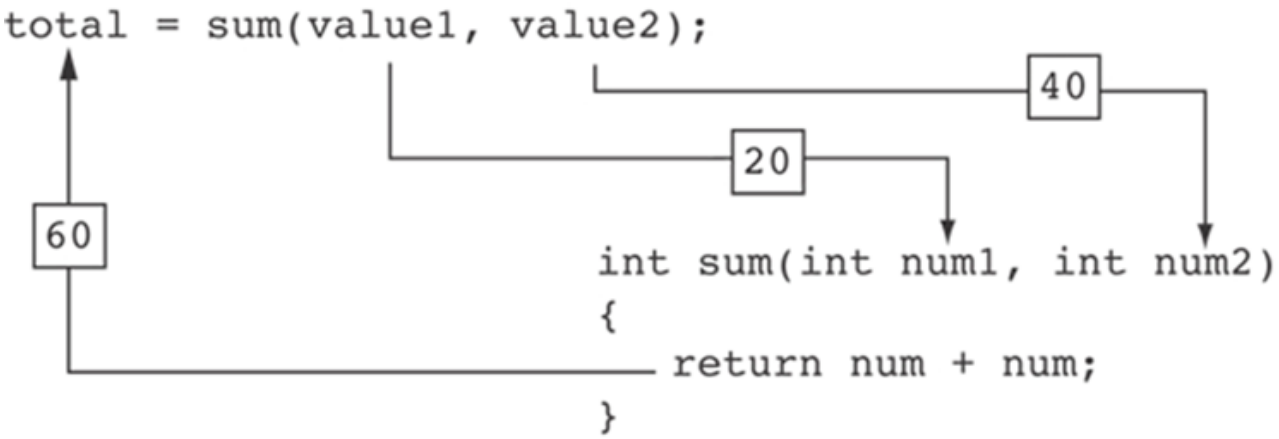
Returning a Value From a Function

```
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
```

Returning a Value From a Function



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

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

Returning a Boolean Value

Program 6-15

```
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
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
```

Returning a Boolean Value

```
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 a bool.                        *
30  //*************************************************************
31
32  bool isEven(int number)
33  {
34     bool status;
35
36     if (number % 2 == 0)
37        status = true;  // The number is even if there is no remainder.
38     else
39        status = false; // Otherwise, the number is odd.
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.
```

Exercises

- Challenging problems: 1, 2, 3, 4, 5, 6, 7, 8
- Write a program to calculate the n-th Fibonacci number. The recursive formula for the Fibonacci sequence is:

$$F_n := F(n) := \begin{cases} 0, & \text{when } n = 0; \\ 1, & \text{when } n = 1; \\ F(n-1) + F(n-2) & \text{when } n > 1. \end{cases}$$