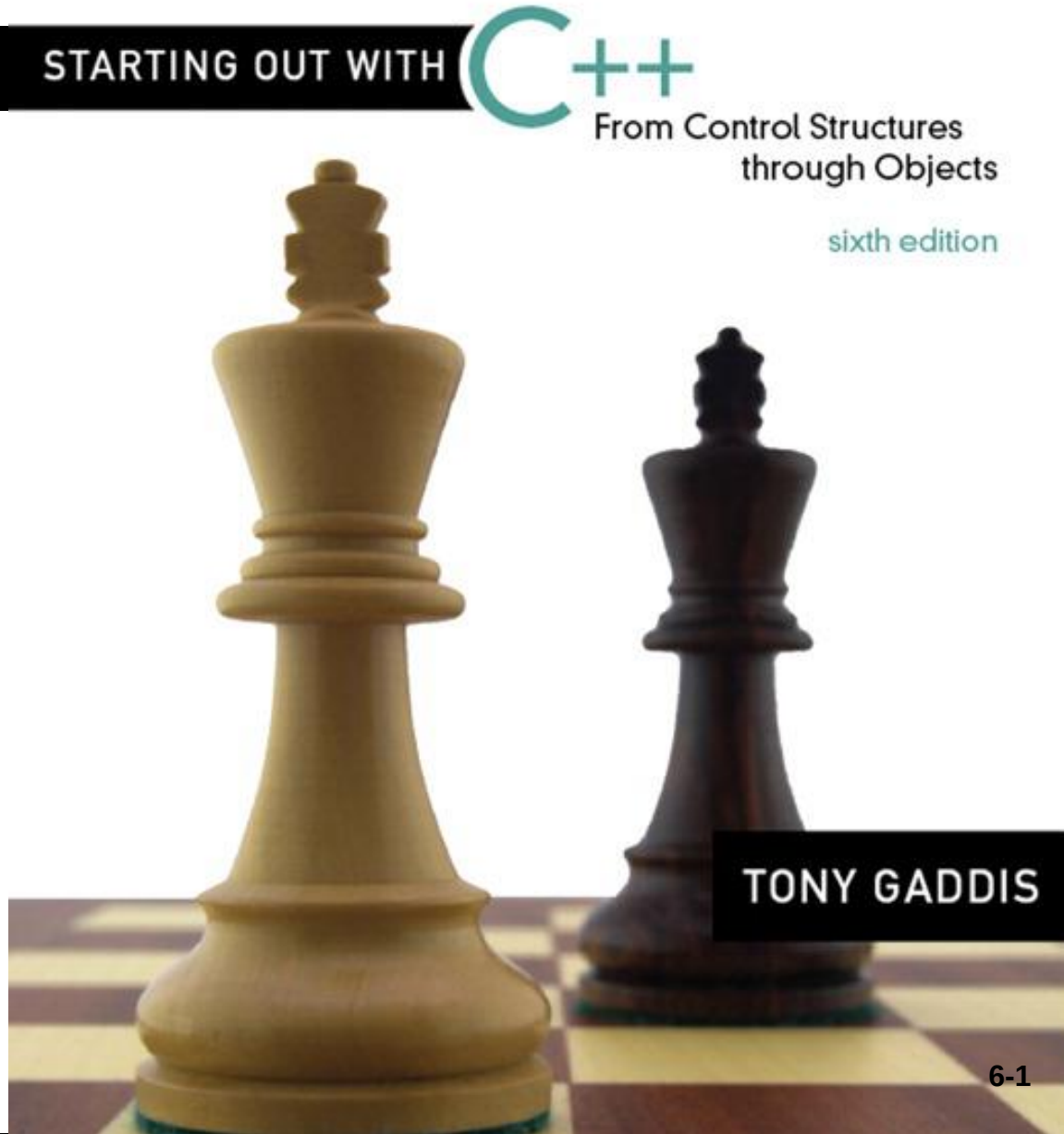
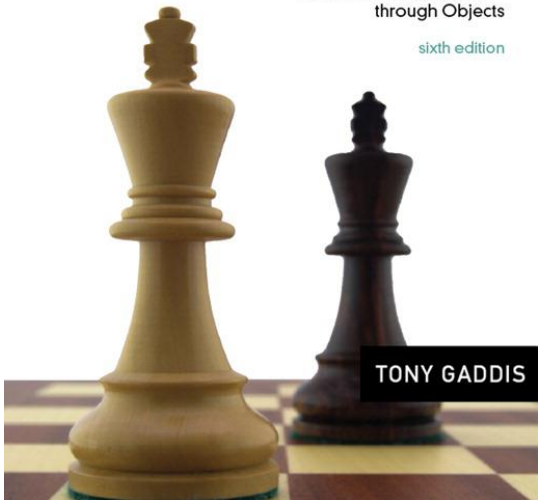
Chapter 6:

Functions





6.1

Introduction to Functions

What Is A Function?
Why Functions?

- We've been using functions (e.g. `main()`). C++ program consists of one or more functions
- Function: a collection of statements that performs a specific task and are grouped together under a certain name
- Modular programming: breaking a program up into smaller, manageable functions or modules
- Motivation for modular programming:
  - Improves maintainability and readability of programs
  - Simplifies the process of writing programs

```
// This program has one long and complex function (main)
// that contains all the necessary statements to solve
// a problem.
void main()
{
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    statement;
    ……
    statement;
}
```

Task1: read and validate the data

Task2: process the data

Task3: print the data with certain format

In this program the problem has been divided into 3 smaller modules, each of which is handled by a separate function.

```
void ReadData()
{
     statement;
     ……
}

void ProcessData()
{
     statement;
     ……
}

void PrintData()
{
     statement;
     ……
}

void main()
{
     ReadData();
     ProcessData();
     PrintData();
}
```

ReadData function contains statements to read and validate input data

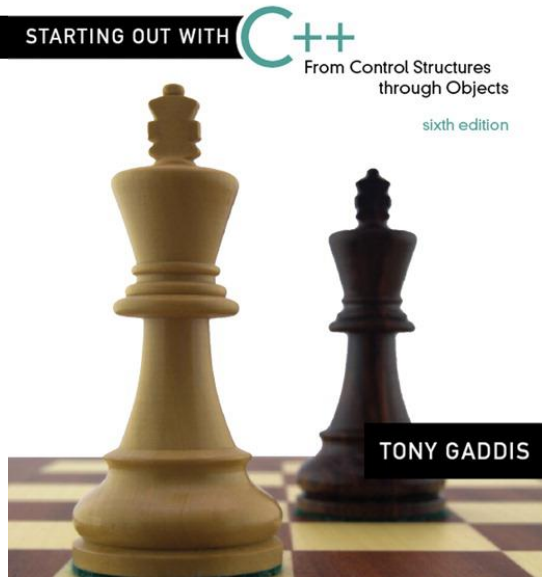
ProcessData function contains statements to process the data

PrintData function contains statements to print the data in required format

Main function calls three other functions to handle separate problems

What Is A Function?
Why Functions?

- The whole program is divided into smaller modules, each performing a specific task
- `main()` function is the control module
- It accomplishes the overall task by calling other functions
- To `main()` all other functions are like black boxes whose details are hidden



6.2

Defining and Calling Functions

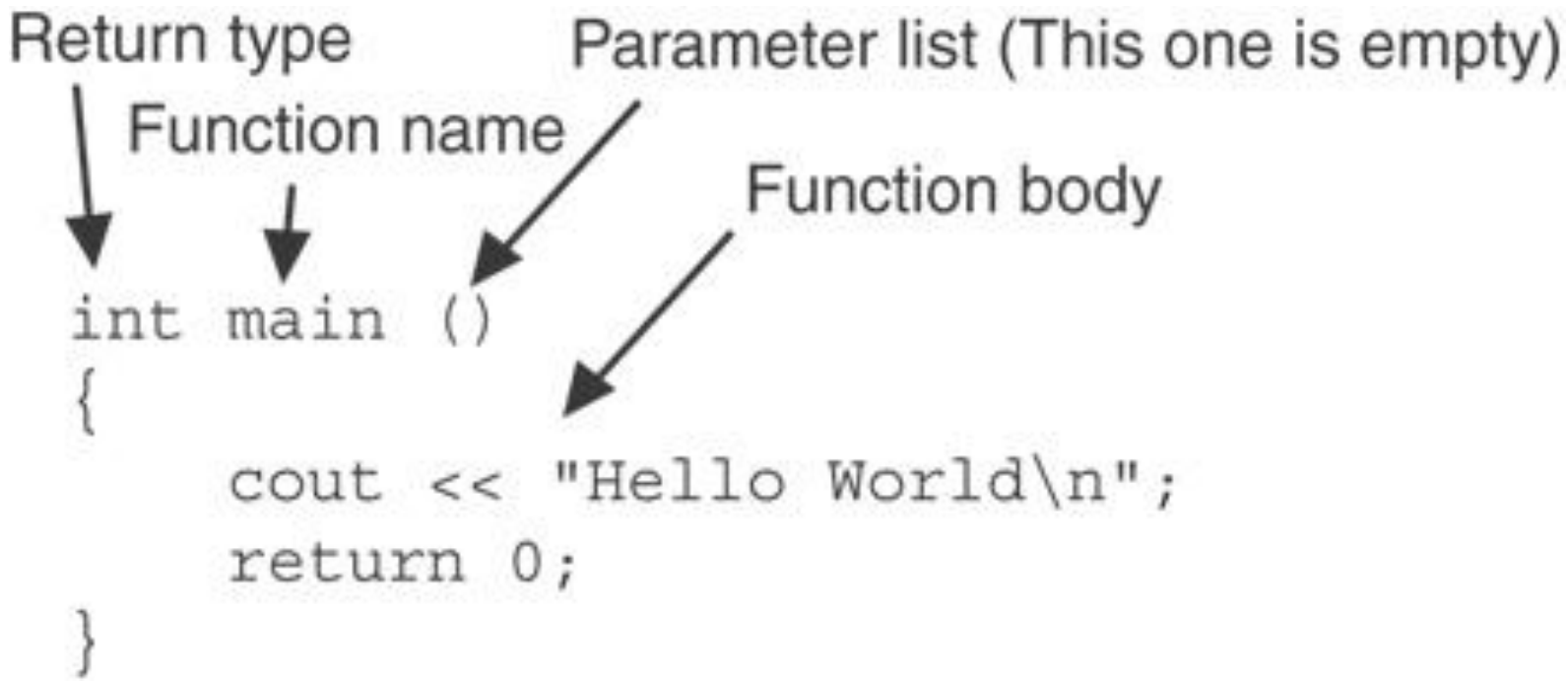
Defining and Calling Functions

- How to use the function: first define the function, then call the function
- Function definition: statements that make up a function
- Function call: statement causes a function to execute

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();
```

- The main function is called automatically when the program is executed
- When called, program executes the body of the called function
- After the function terminates, execution resumes in the calling function at point of call

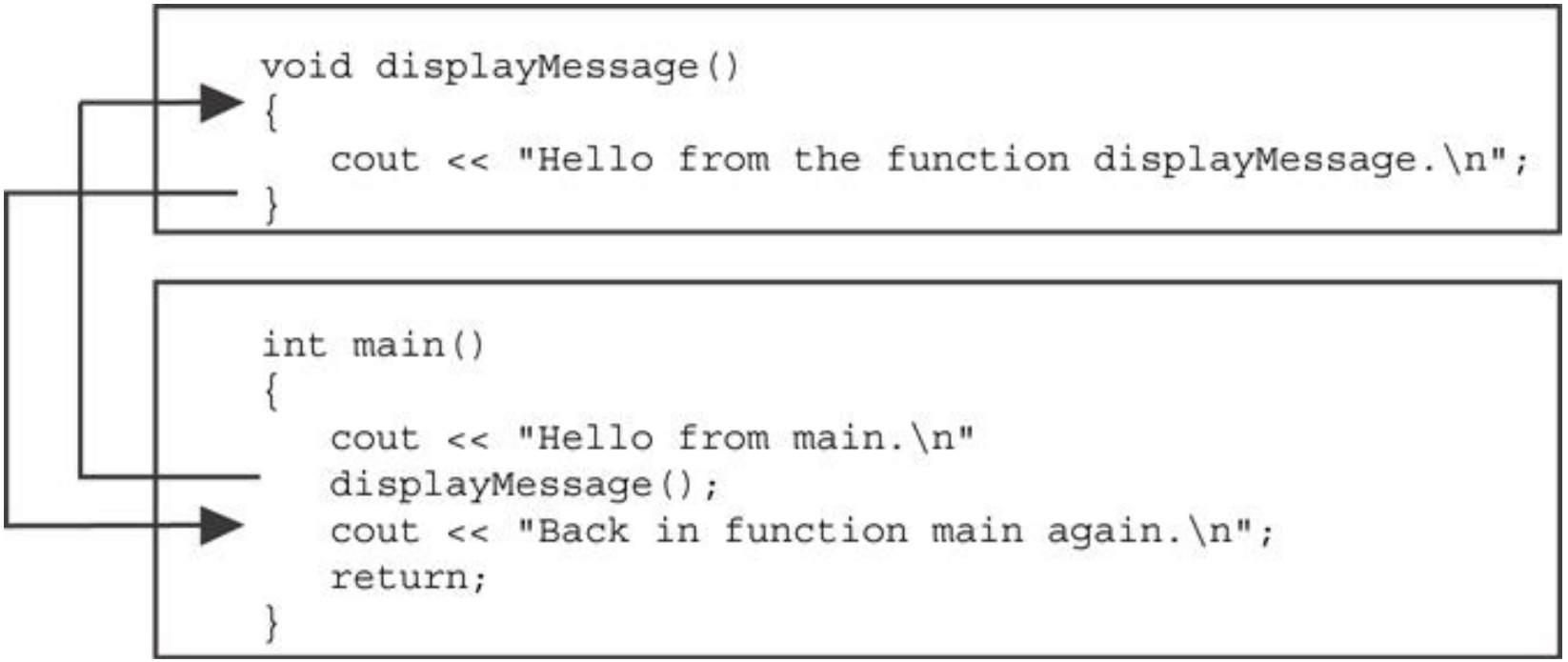
Program 6-1

```
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.
```

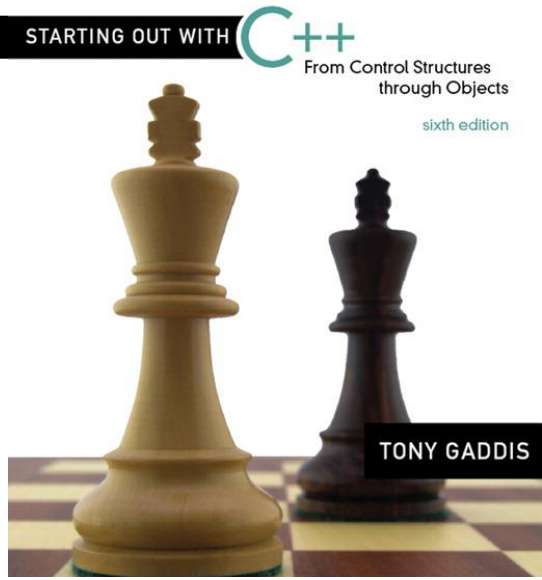
Flow of Control in Program 6-1



```
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
  - Program 6-3
- Functions can call other functions
  - Program 6-4
- Compiler must know the following about a function before it is called:
  - name
  - return type
  - number of parameters
  - data type of each parameter



6.3

Function Prototypes

Function Prototypes

- Ways to notify the compiler about a function before a call to the function:
  1) Place function definition before all calls to that function (e.g. Programs 6-1, 6-3, 6-4)
  2) Place a function prototype (function declaration) ahead of all calls to the function
     - It is like the function header but with a semicolon
       - Function header: `void printHeading()`
       - Prototype: `void printHeading();`
     - The function definition follows later

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)

```
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

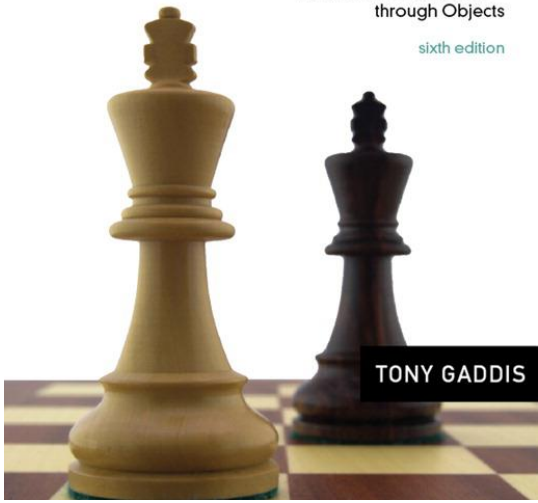
- Place prototypes near top of program
- Program must include either prototype or full function definition before any call to the function – compiler error otherwise
- Most programs use function prototypes
- When using prototypes, the function definitions are usually placed after the main function in any order

Program Structure with Functions

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

void myfunc();          // function prototypes go here

int main()
{
    …
    myfunc();           // function calls go here
    …
    return 0;
}

void myfunc()           // function definitions go here
{
    …
}
```

Using Functions in Menu-Driven Programs

- Functions can be used
  - to implement user choices from menu
  - to modularize the program to make it easier to understand
- *Change Program 5-8 using showMenu() function (refer to Program 6-10 in the book)*



6.4

Sending Data into a Function

Sending Data into a Function

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

```
double a=27, b=5, c;
c = pow(a, b);
```

{ $c = a^b$ }

- Values passed to a function call are called <u>arguments</u>
- Variables in a function definition that hold the values passed as arguments are called <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 (argument) 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)

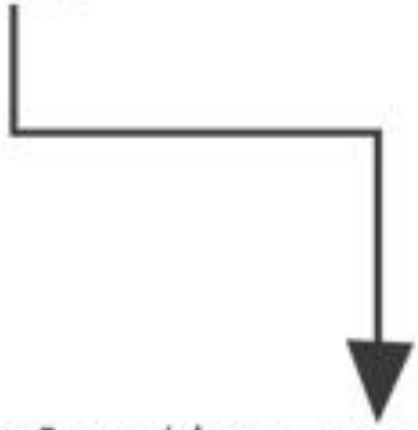
Program 6-6 *(continued)*

```
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 parameter num.

Other Parameter Terminology

- A parameter can also be called a formal parameter or a formal argument

- An argument can also be called an actual parameter or an actual argument

Parameters, Prototypes, and Function Headers

- The prototype must include the data type of each parameter inside its parentheses
- The header must include a declaration for each parameter in its `()`
- The call takes a value, an expression or a variable (without data type) for each argument

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

Function Call Notes

- Function can have multiple parameters
- There must be a data type listed in the prototype （） for each parameter and a declaration in the function header （） for each parameter
- Arguments will be promoted / demoted as necessary to match 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
```

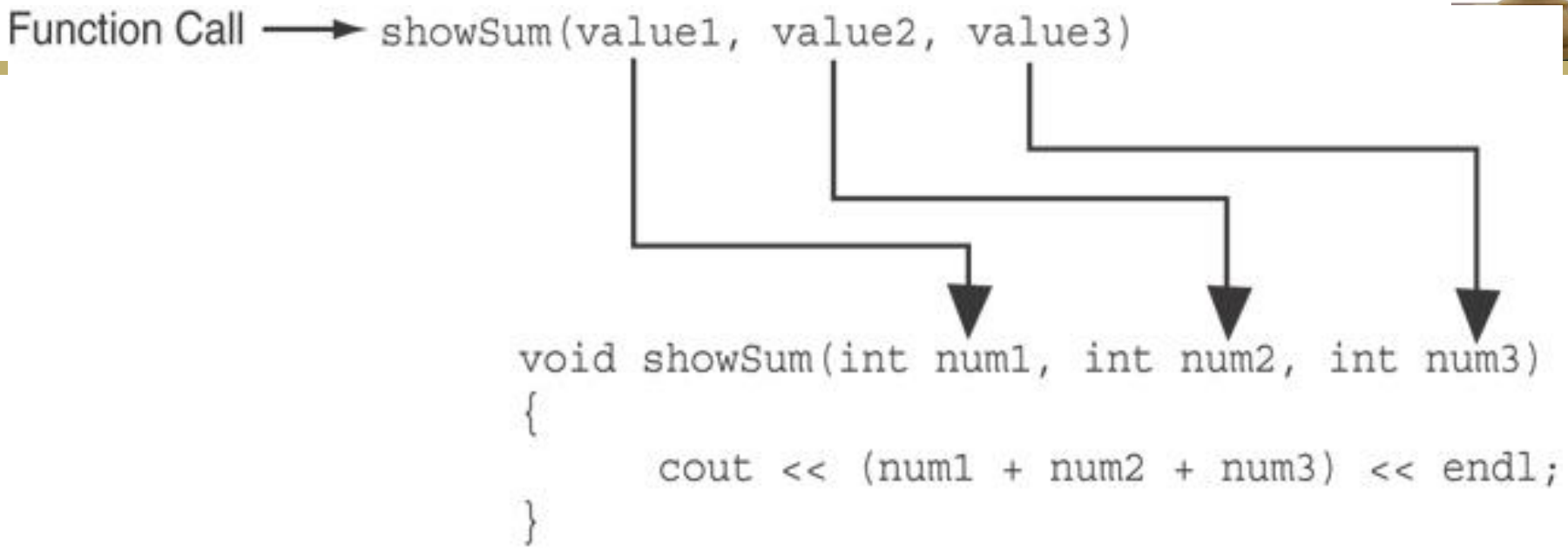
(Program Continues)

Program 6-8 (Continued)

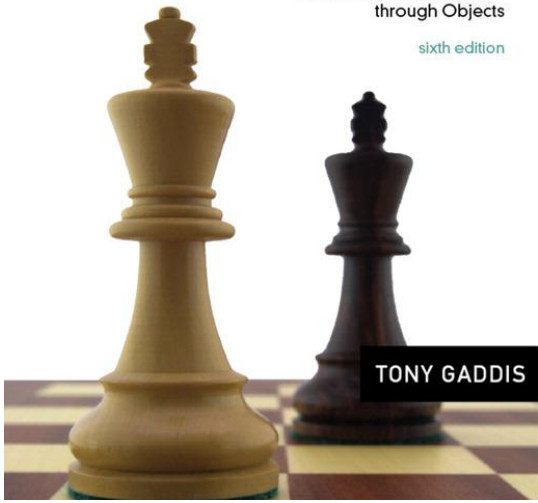
```
22  //*************************************************
23  // Definition of function showSum.                *
24  // It uses three integer parameters. Their sum is displayed. *
25  //*************************************************
26
27  void showSum(int num1, int num2, int num3)
28  {
29   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

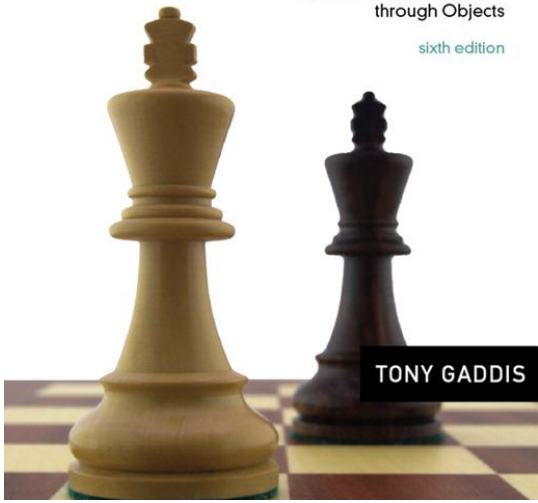
- `evenOrOdd` can change variable `num`, but it will have no effect on variable `val`

```
// This program demonstrates that changes to a
// function parameter have no effect on the
// original argument.
#include <iostream>
using namespace std;

// Function prototype
void increase(int);

void main()
{
    int number=10;

    cout << "Originally number = " << number << endl;

    increase(number);

    cout << "Finally number = " << number << endl;
}
```

```
// ***************************************
// Function definition for increase().   *
// This function increases the value of  *
// parameter by one.                     *
// ***************************************

void increase(int value)
{
    value ++;

    cout << "Inside function, number = "
         << value << endl;
}
```

Program Output:

```
Originally number = 10
Inside function, number = 11
Finally number = 10
```



6.13

Passing Data by Reference—Using Reference Variables as Parameters

Using Reference Variables as Parameters

- A mechanism that allows a function to access the original parameter's argument from the function call, not a copy of the argument
- Allows the function to modify variables defined in another function
- Provides a way for the function to "return" more than one value

Passing by Reference

- A reference variable is an alias for another variable
- Defined with an ampersand (`&`)

```
void getDimensions(int&, int&);
```

- Changes to a reference variable are made to the variable it refers to
- Use reference variables to implement passing data *by reference*

Program 6-25

The & here in the prototype indicates that the parameter is a reference variable.

```
1  // This program uses a reference variable as a function
2  // parameter.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype. The parameter is a reference variable.
7  void doubleNum(int &);
8
9  int main()
10 {
11    int value = 4;
12
13    cout << "In main, value is " << value << endl;
14    cout << "Now calling doubleNum..." << endl;
15    doubleNum(value);
16    cout << "Now back in main. value is " << value << endl;
17    return 0;
18 }
19
```

Here we are passing data by reference.

(Program Continues)

Program 6-25 *(Continued)*

The & also appears here in the function header.

```
20  //*************************************************************
21  // Definition of doubleNum.                                   *
22  // The parameter refVar is a reference variable. The value    *
23  // in refVar is doubled.                                      *
24  //*************************************************************
25
26  void doubleNum (int &refVar)
27  {
28     refVar *= 2;
29  }
```

Program Output

```
In main, value is 4
Now calling doubleNum...
Now back in main. value is 8
```

```
// This program demonstrates that using a
// reference variable in a function call can
// change the original argument.
#include <iostream>
using namespace std;

// Function prototype
void increase(int &);

void main()
{
    int number=10;

    cout << "Originally number = " << number << endl;

    increase(number);

    cout << "Finally number = " << number << endl;
}
```

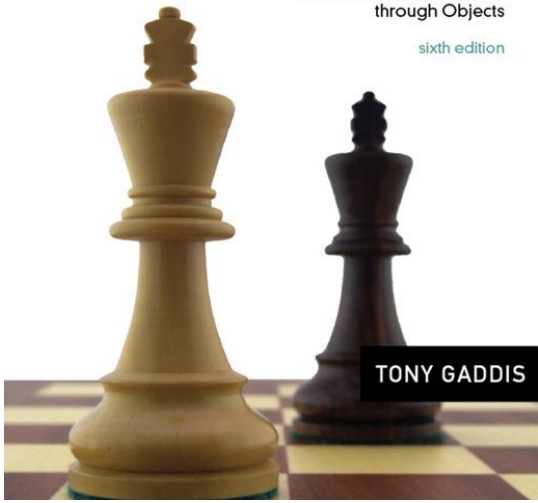
```
// ***************************************
// Function definition for increase().   *
// This function increases the value of  *
// parameter by one.                     *
// ***************************************

void increase(int &value)
{
    value ++;

    cout << "Inside function, number = "
         << value << endl;
}
```

Program Output:

```
Originally number = 10
Inside function, number = 11
Finally number = 11
```

Reference Variable Notes

- Reference variables must be defined with & in front of the name (but used without &)
- Space between type and & is unimportant
- Must use & in both prototype and header, but not needed in the function call
- Argument passed to the reference parameter must be a variable – cannot be an expression or constant
- Their data types must match each other
- Use when appropriate – don't use when argument should not be changed by function, or if function needs to return only 1 value

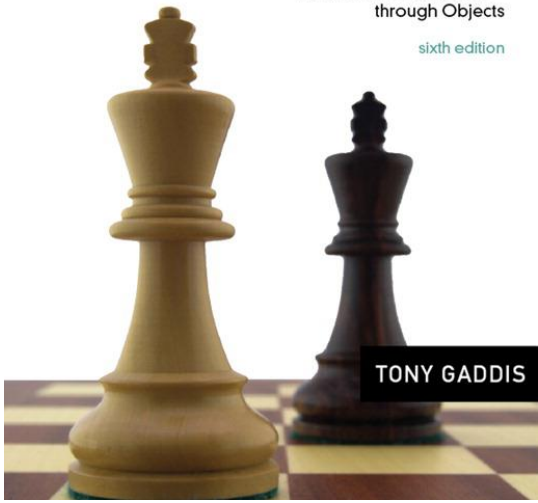


6.6

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 statements and speeding program development time
- *See Program 6-10 in the book (compare it with Program 5-8)*



6.7

The `return` Statement

The `return` Statement

- Used to end execution of a function and return the control back to the statement that called this function
- Can be placed anywhere in a function
  - Statements that follow the `return` statement will not be executed
- Used to return a value to the calling function
- 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 }
```

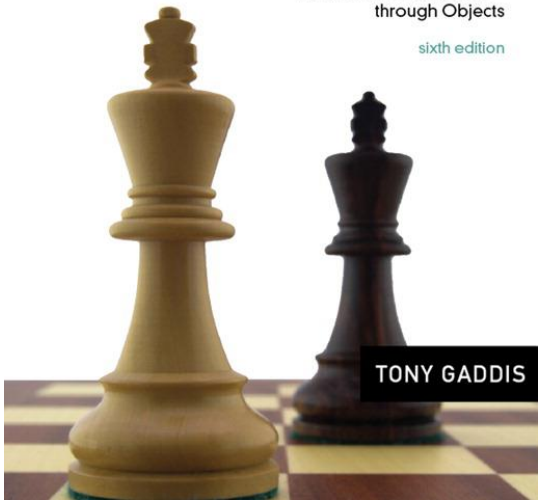
(Program Continues)

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

- Data can be passed to functions through parameter variables
- A function can also return a value back to the statement that called the function
- In the following, the `pow` function returns the value to `x` through an assignment statement:

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

Value-Returning Functions

- Functions that return a value are known as **value-returning** functions
- 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)
{
   int result;
   result = num1 + num2;
   return result;
}
```

Defining a Value-Returning Function

Return Type

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

Value Being Returned

Defining a Value-Returning Function

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

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

The next few slides show how to call a Value-Returning 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 }
```

(Program Continues)

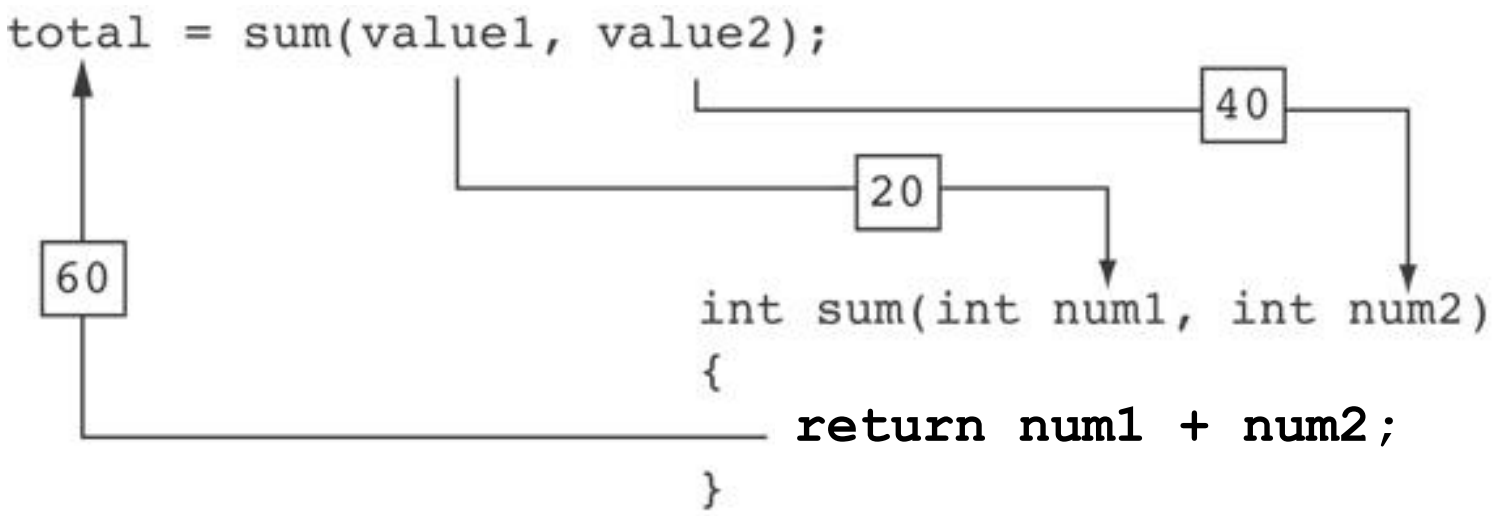
Program 6-12 (Continued)

```
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

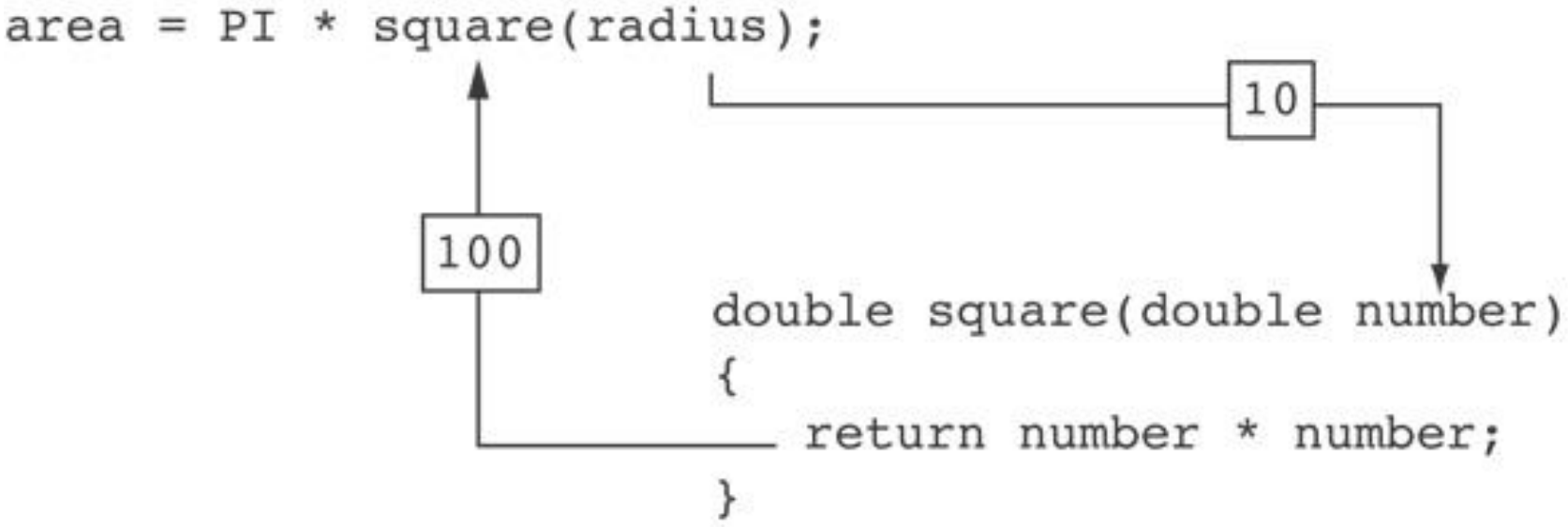
```
The sum of 20 and 40 is 60
```

Calling a Value-Returning 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.

Another Example, from Program 6-13



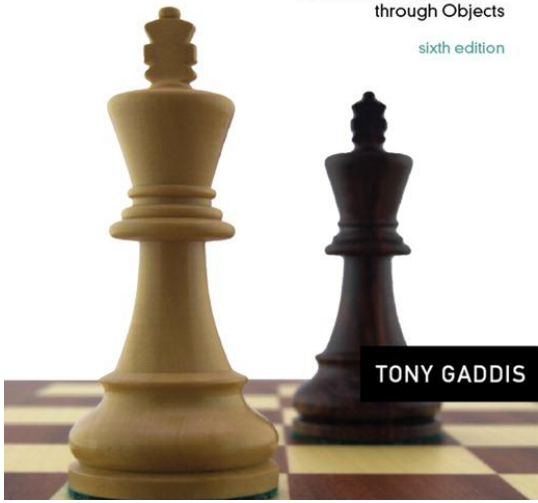
Returning a Value From a Function

- 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

```
int x=10, y=5;
double average;
cout << "The sum is " << sum(x, y);
average = sum(x, y)/2.0;
```

Converting Fahrenheit to Celcius

```
#include <iostream>
Using namespace std;
double Fahr2Cel(double);

void main()
{
    double degF, degC;

    cout << "Please enter temperature in Fahrenheit: ";
    cin >> degF;
    degC = Fahr2Cel( degF );
    cout >> "The temperature in Celcius is " >> degC >> endl;
}

double Fahr2Cel(double fahr)
{
    return (fahr-32)*5/9;
}
```



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 6-15

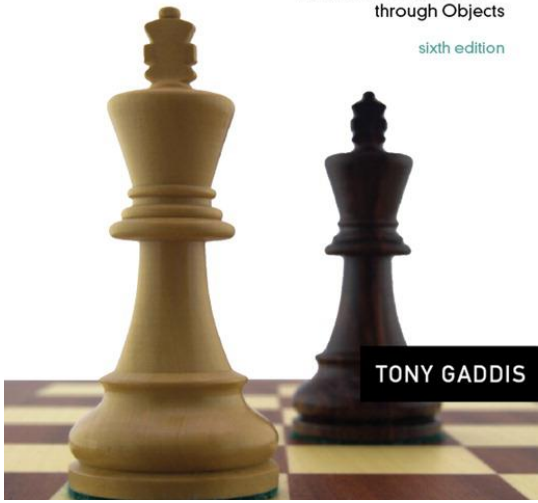
```
1  // This program uses a function that returns true or false.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  bool isEven(int);
7
8  int main()
9  {
10    int val;
11
12    // Get a number from the user.
13    cout << "Enter an integer and I will tell you ";
14    cout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

(Program Continues)

```
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.
```



6.10

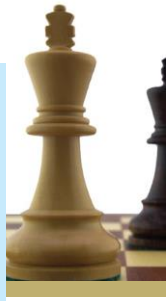
Local and Global Variables

Local and Global Variables

- Variables defined inside a function are *local* to that function. They are hidden from the statements in other functions, which normally cannot access them.
- Because the variables defined in a function are hidden, other functions may have separate, distinct variables with the same name.

Program 6-16

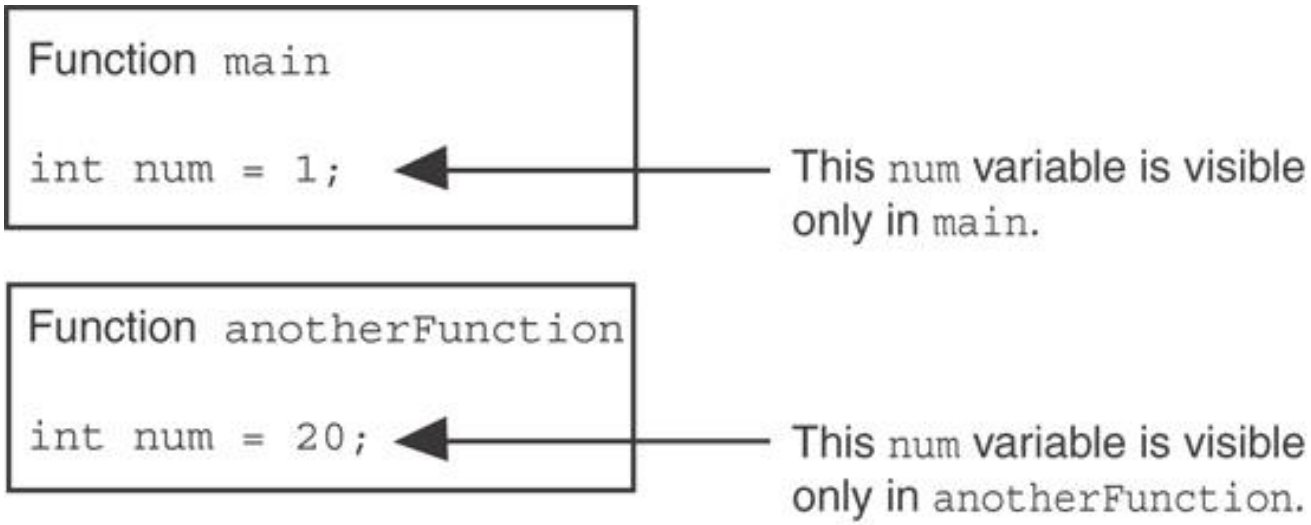
```
1  // This program shows that variables defined in a function
2  // are hidden from other functions.
3  #include <iostream>
4  using namespace std;
5
6  void anotherFunction(); // Function prototype
7
8  int main()
9  {
10    int num = 1;   // Local variable
11
12    cout << "In main, num is " << num << endl;
13    anotherFunction();
14    cout << "Back in main, num is " << num << endl;
15    return 0;
16 }
17
18 //*********************************************************
19 // Definition of anotherFunction                          *
20 // It has a local variable, num, whose initial value      *
21 // is displayed.                                          *
22 //*********************************************************
23
24 void anotherFunction()
25 {
26    int num = 20;  // Local variable
27
28    cout << "In anotherFunction, num is " << num << endl;
29 }
```



Program Output

```
In main, num is 1
In anotherFunction, num is 20
Back in main, num is 1
```

When the program is executing in `main`, the `num` variable defined in `main` is visible. When `anotherFunction` is called, however, only variables defined inside it are visible, so the `num` variable in `main` is hidden.



Local Variable Lifetime

- A function's local variables exist only while the function is executing. This is known as the *lifetime* of a local variable
- When the function begins, its local variables and its parameter variables are created in memory, and when the function ends, the local variables and parameter variables are destroyed
- This means that any value stored in a local variable is lost between calls to the function in which the variable is declared

Global Variables and Global Constants

- A global variable is any variable defined outside all the functions in a program.
- The scope of a global variable is the portion of the program from the variable definition to the end.
- This means that a global variable can be accessed by *all* functions that are defined after the global variable is defined.

Global Variables and Global Constants

- You should avoid using global variables for conventional purposes of storing, manipulating and retrieving data, because they make programs difficult to manage
- Global variables are commonly used to create *global constants*

Program 6-19

```
1  // This program calculates gross pay.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Global constants
7  const double PAY_RATE = 22.55;     // Hourly pay rate
8  const double BASE_HOURS = 40.0;    // Max non-overtime hours
9  const double OT_MULTIPLIER = 1.5; // Overtime multiplier
10
11 // Function prototypes
12 double getBasePay(double);
13 double getOvertimePay(double);
14
15 int main()
16 {
17    double hours,              // Hours worked
18           basePay,            // Base pay
19           overtime = 0.0,     // Overtime pay
20           totalPay;           // Total pay
```

Global constants defined for values that do not change throughout the program's execution.

The constants are then used for those values throughout the program.

```
29      // Get overtime pay, if any.
30      if (hours > BASE_HOURS)
31         overtime = getOvertimePay(hours);
```

```
56    // Determine base pay.
57    if (hoursWorked > BASE_HOURS)
58       basePay = BASE_HOURS * PAY_RATE;
59    else
60       basePay = hoursWorked * PAY_RATE;
```

```
75      // Determine overtime pay.
76      if (hoursWorked > BASE_HOURS)
77      {
78         overtimePay = (hoursWorked - BASE_HOURS) *
79                  PAY_RATE * OT_MULTIPLIER;
```

Initializing Local and Global Variables

- Local variables are not automatically initialized. They must be initialized by programmer.

- Global variables (not constants) are automatically initialized to `0` (numeric) or `NULL` (character) when the variable is defined.

Temperature Conversion Problem

- Write a program to convert temperature either from Celsius to Fahrenheit or vice versa. The main function will repeatedly prompt the user to input both a temperature and a choice of whether that number is in Celsius to be converted to Fahrenheit or vice versa. The main function **MUST** call one **value returning function** to do both conversion. The function should take two parameters: 1. Temperature, 2. Choice. The function should return the converted temperature to the calling function.

Algorithm (Main Function)

1) Prepare for data:

   inputTemp, outputTemp, choice

2) Repeat the following until user chooses to stop:

   i. Prompt the user to enter a choice:

      1: convert Cel to Fah

      2: convert Fah to Cel

      3: stop

   ii. If choice == 1 or 2

   prompt the user to enter the temperature in Cels or Fahr

   call function “convTemp” to do the selected conversion

   print the converted temperature

   iii. If choice == 3

   print a terminating message and stop the repetition

Algorithm (ConvTemp Function)

1) Prepare for data

   inputTemp, choice, outputTemp

2) If choice==1

   convert from Celsius to Fahrenheit using:

   outputTemp = inputTemp * 9 / 5 + 32;

3) If choice==2

   convert from Fahrenheit to Celsius using:

   outputTemp = (inputTemp – 32) *5 / 9;

4) Return the converted temperature (outputTemp) to the calling function

The Complete Program (1)

```
#include <iostream>
using namespace std;
double convTemp(int, double);

void main()
{
    double inputTemp, outputTemp;
    int choice;

    do {
        cout << "1: Convert Celsius to Fahrenheit\n";
        cout << "2: Convert Fahrenheit to Celsius\n";
        cout << "3: Quit\n\n";
        do {
            cout << "Your choice? ";
            cin >> choice;
        } while ( choice!=1 && choice!=2 && choice!=3);
```

The Complete Program (2)

```
if ( choice == 1 || choice == 2 )
{
    cout << "Enter temperature: ";
    cin >> inputTemp;
    outputTemp = convTemp(choice, inputTemp);
}

switch ( choice )
{
  case 1:
      cout << "\nThe temperature in Fahr is "
           <<  outputTemp << endl << endl;
      break;
  case 2:
      cout << "\nThe temperature in Cels is "
           << outputTemp << endl << endl;
      break;
```

The Complete Program (3)

```
            case 3:
                cout << "\nThe program is exiting.\n\n";
        }
    } while ( choice != 3 );
}

double convTemp(int choice, double temp)
{
    if ( choice == 1 )
        return temp*9/5+32;
    else
        return (temp-32)*5/9;
}
```