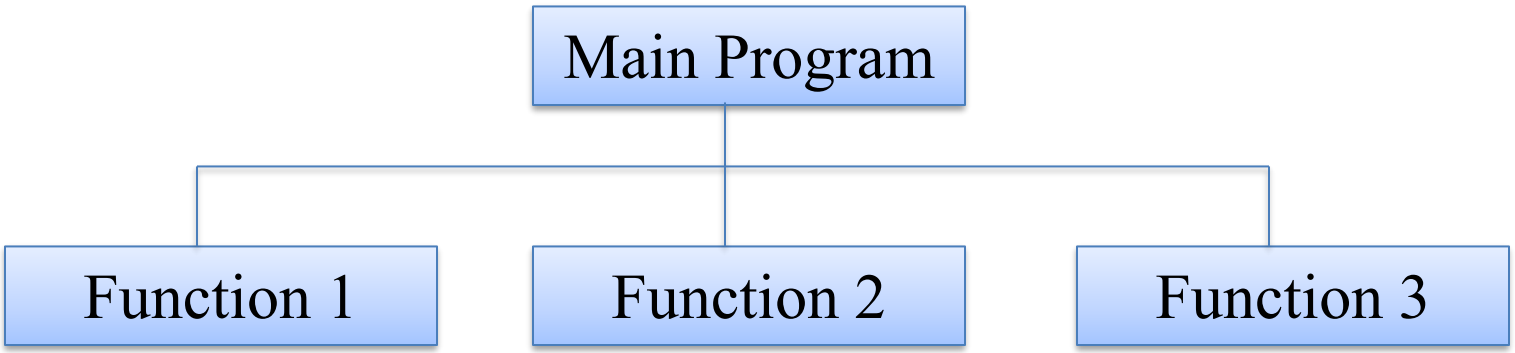
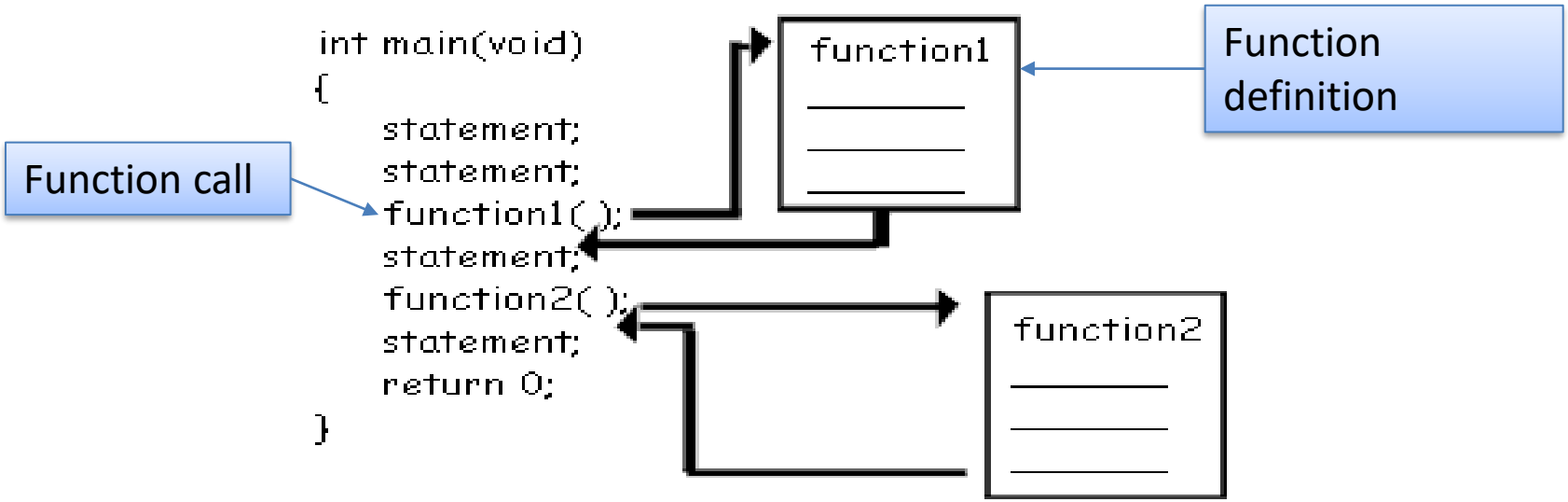
User-Defined Functions

- User-defined functions are created by you, the programmer.
- Commonly used to break a problem down into small **manageable** pieces.
- You are already familiar with the one function that every C++ program possesses: `int main()`
  - Ideally, your `main()` function should be very short and should consist primarily of function calls.



Defining and Calling Functions

- Every functions must have:
  - **Function call:** statement causes a function to execute
  - **Function definition:** statements that make up a function



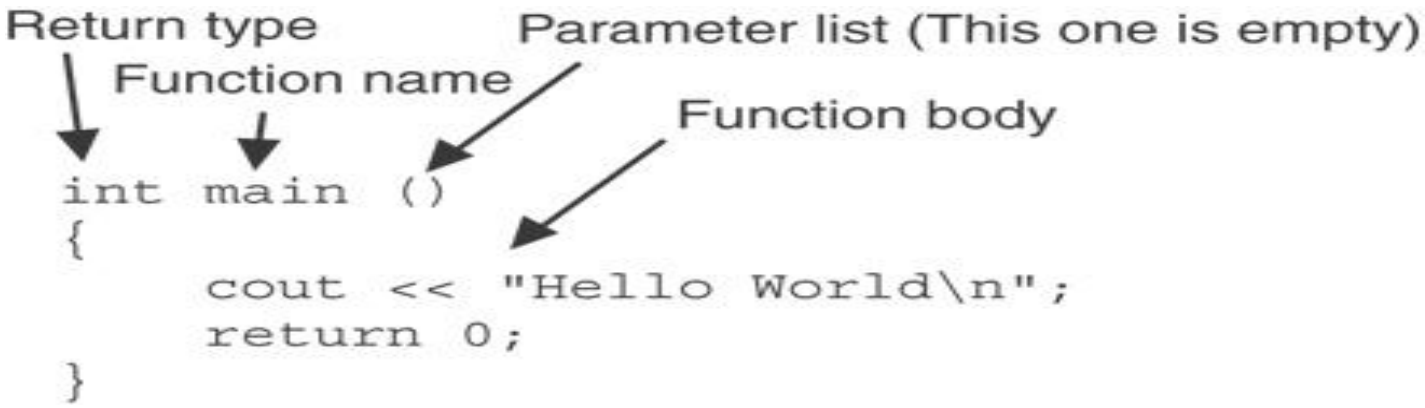
Function Definition

- Definition includes:
  - return type: data type of the value that function returns to the part of the program that calls 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 (cont.)

- The general form of a function definition in C++ is as follows:

```
function-returntype function-name( parameter-list )
{
        local-definitions;
        function-implementation;
}
```



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";
}
```

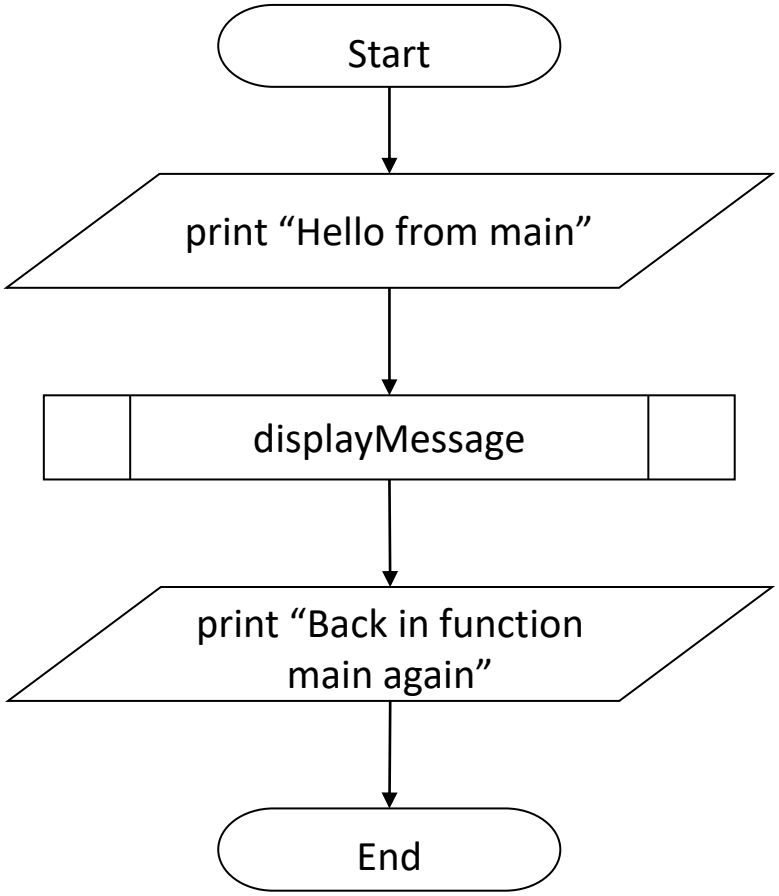
Calling a Function

- 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.

The Flowchart

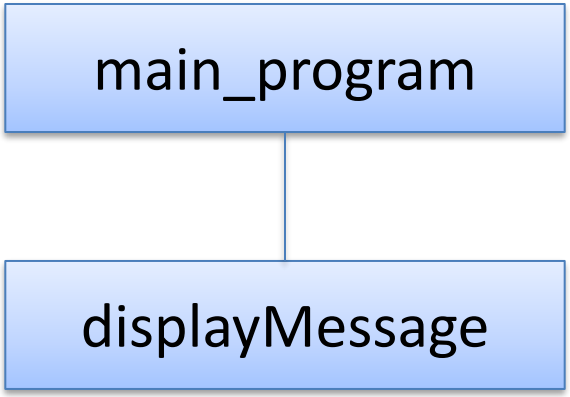


The Pseudo Code

- Start
- Print “Hello from main”
- call displayMessage
- Print “Back in function main again”
- End

- displayMessage:
  - Print “Hello from the function displayMessage”

The Structure Chart



Calling a Function - Example

Function definition

Function call

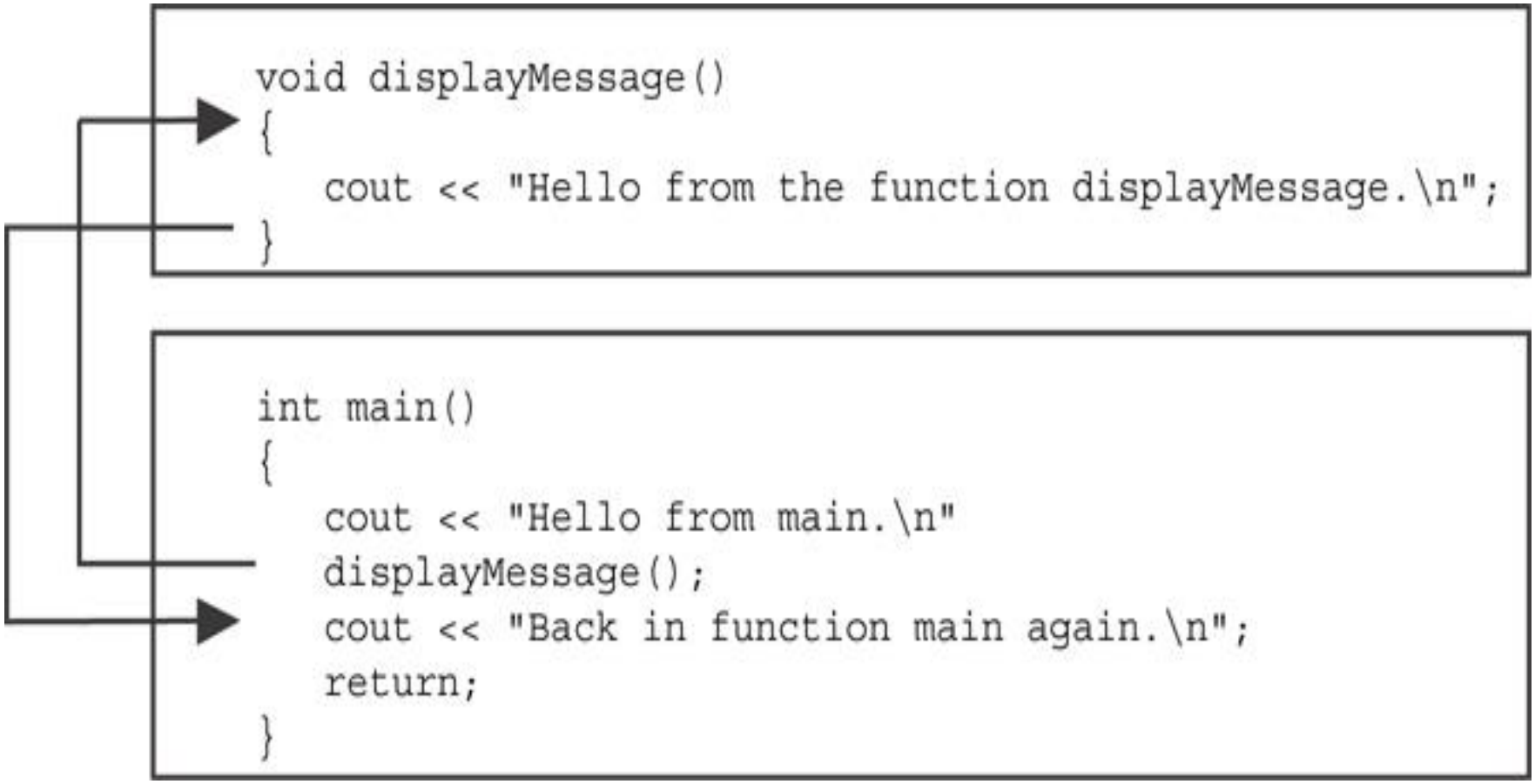
Program 6-1

```
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;
}
```

User-Defined Functions: Example 2

```
01  #include <iostream>
02  #include <cmath>
03  using namespace std;
04
05  float distance(float x, float y)
06  {    float dist;
07       dist = sqrt(x * x + y * y);
08       return dist;
09  }
10
11  void main()
12  {
13       float x,y,dist;
14       cout << "Testing function distance(x,y)"  << endl;
15       cout << "Enter values for x and y: ";
16       cin >> x >> y;
17       dist = distance(x,y);
18       cout << "Distance of (" << x << ',' << y << ") from origin is
         " << dist << endl << "Tested" << endl;
19  }
```

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

In-Class Exercise

- Do Lab 11, Exercise 1, No 1 (pg. 147-149)
- Which of the following function headers are valid? If they are invalid, explain why.
  - `one (int a, int b)`
  - `int thisone(char x)`
  - `char another (int a, b)`
  - `double yetanother`

Function Prototypes

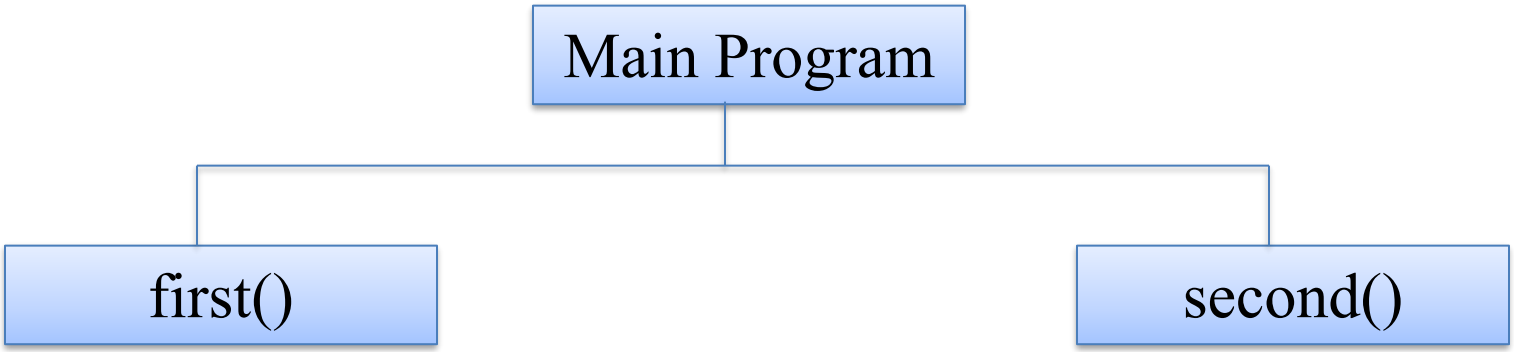
- Ways to **notify the compiler** about a function before a call to the function:
  - Place function definition before calling function's definition
  - Use a function prototype (function declaration) – like the function definition without the body
    - Header: `void printHeading()`
    - Prototype: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

User-Defined Functions: Function Prototypes

```
01  #include <iostream>
02
03  void first();
04  void second();
05
06  void main()
07  {
08     cout<< "Starting in main function \n";
09     first();
10     second();
11     cout<<"Control back to main\n";
12  }
13
14  void first()
15  {  cout<<"Inside the first function\n";}
16
17  void second()
18  {  cout<<"Inside the second function\n";}
```

Function prototypes

Structured Chart



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

User-Defined Functions: Functions with No Parameters

```
01  #include <iostream>
02  void printhi();
03
04  void main(){
05     cout << "Testing function printhi()" << endl;
06     printhi();
07     cout << "Tested" << endl;
08  } // End of main
09
10  // Function Definitions
11  void printhi()
12  {
13     cout << "Hi \n";
14  }
```

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

```
01  void displayValue(int num)
02  {
03      cout << "The value is " << num << endl;
04  }
```

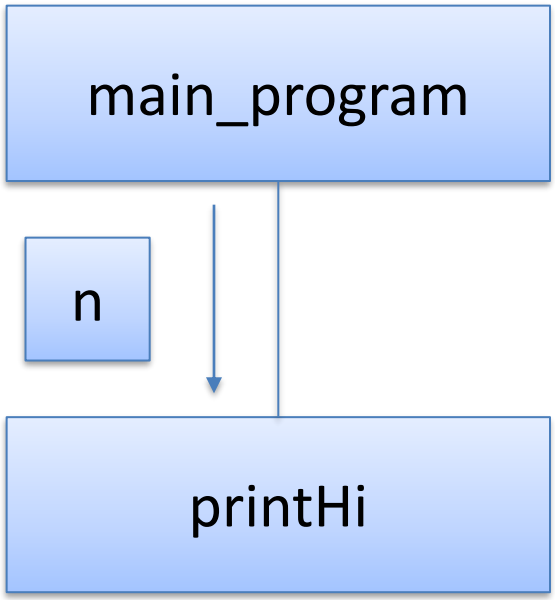
The integer variable `num` is a **parameter**.

It accepts any integer value passed to the function.

User-Defined Functions: Functions with Parameter and No Return Values (1)

```
01  #include <iostream>
02  using namespace std;
03  void printhi(int);
04
05  void main(){
06     int n;
07     cout <<"Enter a value for n: ";
08     cin >> n;
09     printhi(n);
10     cout << "Tested \n";}
11
12  void printhi(int n)
13  {
14     int i;
14     for (i = 0; i < n; i++)
15     cout << "Hi \n";
16  }
```

The Structure Chart



User-Defined Functions: Functions with Parameter and No Return Values (2)

```
01  #include <iostream>
02  using namespace std;
03
04  void displayValue(int);
05
06  void main()
07  {
08     cout<<"Passing number 5 to displayValue\n";
09      displayValue(5);
10      cout<<"Back in main\n";
11  }
12
13  void displayValue(int n)
14  {
15     cout<<"The value is " << n << endl;
16  }
```

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

- 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
- 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
- Arguments will be promoted/demoted as necessary to **match** parameters

In-Class Exercise

- What is the output of this program?

```
01   #include <iostream>
02
03   // Function prototype
04   void showDouble(int);
05
06   int main(){
07       int num;
08       for (num = 0; num < 10; num++)
09       showDouble(num);
10       system("pause");
11       return 0;
12   }
13
14   //Definition of function
15   void showDouble(int value) {
16       cout<<value <<"\t";
17       cout << (value * 2)<< endl;
18   }
```

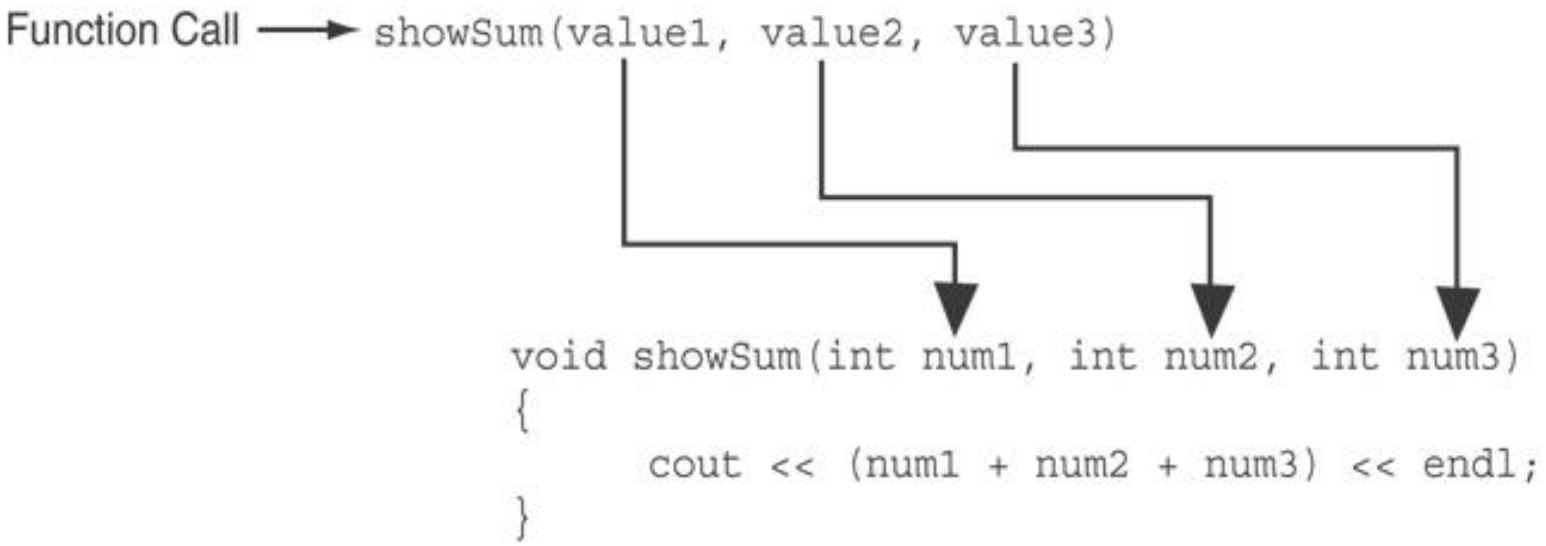
User Defined Functions: Passing Multiple Arguments

- 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.

User-Defined Functions: Passing Multiple Arguments (cont.)

```
01  #include <iostream>
02  using namespace std;
03  void showSum(int, int, int);
04
05  int main()
06  {
07     int value1, value2, value3;
08
09     cout<<"Enter 3 integers: ";
10     cin>> value1 >> value2 >> value3;
11     showSum(value1, value2, value3);
12
13     return 0;
14  }
15
16  void showSum(int a, int b, int c)
17  {
18     cout<<"The sum: "<<a+b+c;
19  }
```

Passing Multiple Arguments (cont.)



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

In-Class Exercise

- What is the output of this program?

```
01   #include <iostream>
02
03   // Function prototype
04   void func1(double, int);
05
06   int main(){
07       int x = 0; double y = 1.5;
08       cout << x << " " <<y<< endl;
09       func1 (y, x);
10       cout << x << " " <<y<< endl;
11       system ("pause");
12       return 0;
13   }
14
15   void func1(double a, int b){
16       cout << a << " " <<b<< endl;
17       a=0.0; b=10;
18       cout << a << " " <<b<< endl;
19   }
```