Programming In C++

Course 2: Lecture 4, Functions

Prepared By Dr. Ali Al-Sabaawi

Nineveh University- Faculty of IT- department of software

Functions

A function is a collection of statements that performs a specific task.

Functions are commonly used to break a problem down into small manageable pieces.

Instead of writing one long function that contains all of the statements necessary to solve a problem, several small functions that each solve a specific part of the problem can be written.

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

```
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

Defining and Calling Functions

When creating a function, you must write its definition. All function definitions have the following parts:

Return type: A function can send a value to the part of the program that executed it. The return type is the data type of the value that is sent from the function.

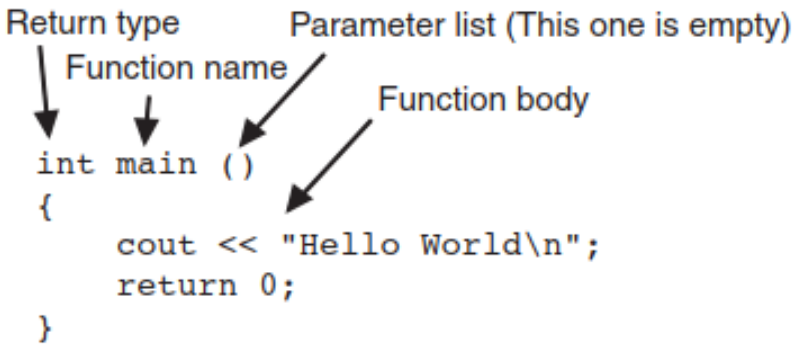
Name: You should give each function a descriptive name. In general, the same rules that apply to variable names also apply to function names.

Parameter list: The program can send data into a function. The parameter list is a list of variables that hold the values being passed to the function.

Body: The body of a function is the set of statements that perform the function's operation. They are enclosed in a set of braces.

Defining and Calling Functions

Program to access the elements above and below the main diagonal



The main function in all of the programs you have seen in this book is declared to return an int value to the operating system.

It isn't necessary for all functions to return a value. These are called void functions.

Name of the function

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

The function's name is displayMessage. This name gives an indication of what the function does: It displays a message. You should always give functions names that reflect their purpose.

Notice that the function's return type is void. This means the function does not return a value to the part of the program that executed it.

Calling a Function

A function is executed when it is called. Function main is called automatically when a program starts, but all other functions must be executed by function call statements.

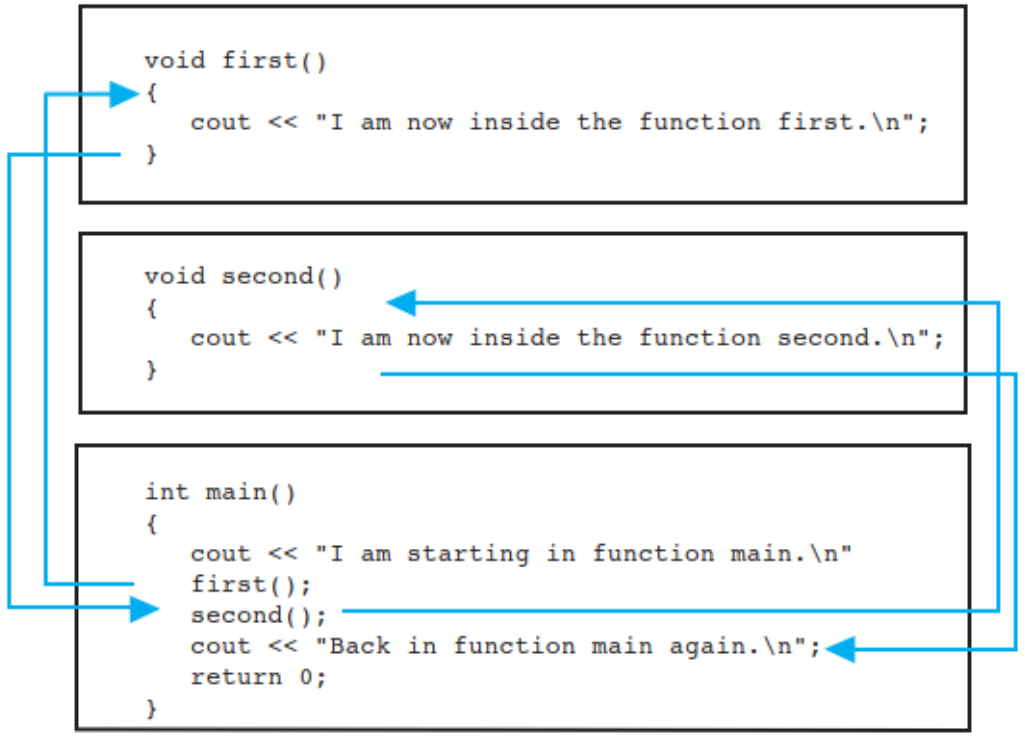
```
void displayMessage()
{
     cout << "Hello from the function displayMessage.\n";
}


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

Function Header ⟶ `void displayMessage()`
Function Call ⟶ `displayMessage();`

Calling a Function

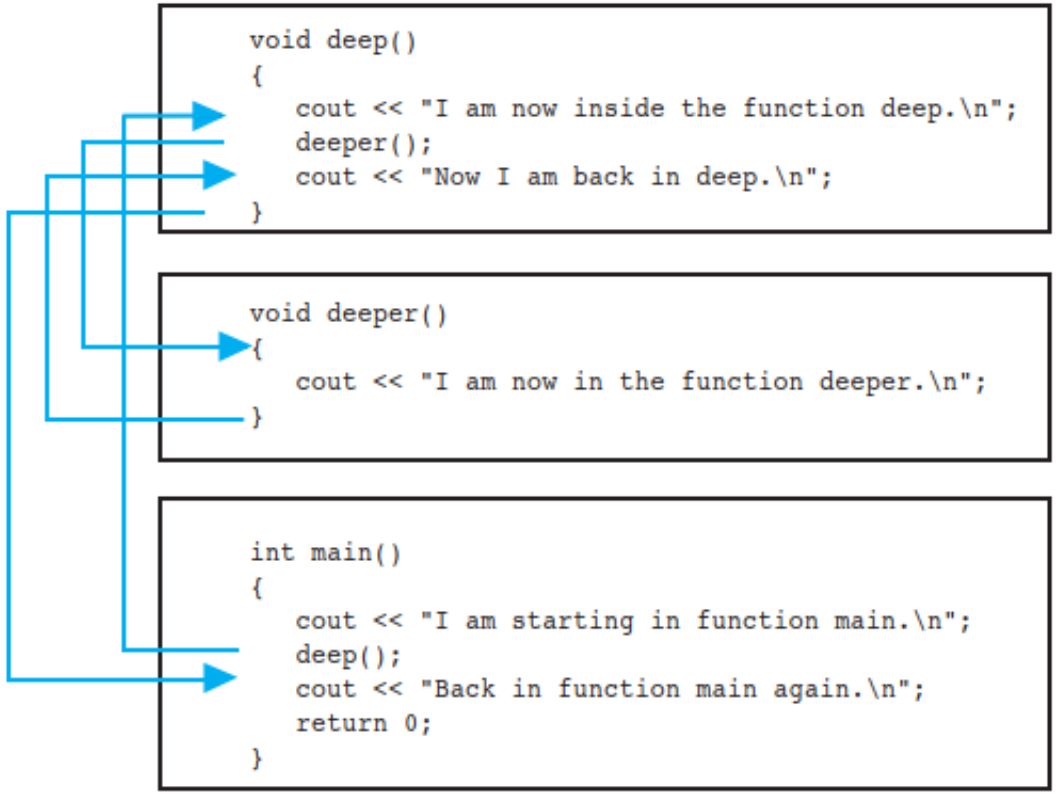
Even though the program starts executing at main, the function displayMessage is defined first.

This is because the compiler must know the function's return type, the number of parameters, and the type of each parameter before the function is called.



Calling a Function

Functions may also be called in a hierarchical, or layered, fashion.



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

void deeper()
{
   cout << "I am now in the function deeper.\n";
}

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

Sending Data into a Function

Values that are sent into a function are called arguments. In the following statement the function pow is being called and two arguments, 2.0 and 4.0, are passed to it:

result = pow(2.0, 4.0);

A parameter is a special variable that holds a value being passed into a function. Here is the definition of a function that uses a parameter:

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

We can call this as shown below

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

Sending Data into a Function


```
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
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 }
```

In the function header for showSum, the parameter list contains three variable definitions separated by commas:

void showSum(int num1, int num2, int num3)

In the function call in line 18, the variables value1, value2, and value3 are passed as

arguments:
showSum(value1, value2, value3);

The following function call will cause 5 to be passed into the num1 parameter, 10 to be passed into num2, and 15 to be passed into num3:
showSum(5, 10, 15);

1- Call by Value

When an argument is passed into a parameter, only a copy of the argument's value is passed. Changes to the parameter do not affect the original argument.

```
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
```

```
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  }
```

The return Statement

The return statement causes a function to end immediately.

It's possible to force a function to return before the last statement has been executed.

When the return statement is encountered, the function immediately terminates and control of the program returns to the statement that called the function.

```
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 }
```

```
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 may send a value back to the part of the program that called the function.

You've seen that data may be passed into a function by way of parameter variables. Data may also be returned from a function, back to the statement that called it.

double x;
x = pow(4.0, 2.0);

```
int sum(int num1, int num2)
{
   int result;

   result = num1 + num2;
   return result;
}
```

After the variable definition, the parameter variables num1 and num2 are added, and their sum is assigned to the result variable. The last statement in the function is

return result;

However, we could have eliminated the result variable and returned the expression num1 + num2, as shown in the following code:

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

2- Call by reference

Call by reference is the second way of passing arguments to a function. In this method, the address of an argument is copied into the parameter.

Inside the function, the address is used to access the actual argument used in the call.

This means that changes made to the parameter affect the argument.

2- Call by reference

```
2
3   using namespace std;
4
5   void mul(int x);
6
7   int main(void)
8   {
9   int t=10;
10  mul(t);
11  cout<<" "<<t;
12
13  return 0;
14
15  }
16
17  void mul(int x)
18  {
19  x = x*x;
20  cout<<x;
21  }
```

The output is: 100 10

```
2
3   using namespace std;
4
5   void mul(int &x);
6
7   int main(void)
8   {
9   int t=10;
10  mul(t);
11  cout<<" "<<t;
12
13  return 0;
14
15  }
16
17  void mul(int &x)
18  {
19  x = x*x;
20  cout<<x;
21  }
```

The output is: 100 100

2- Call by reference

```
3   using namespace std;
4
5   void swap(int &x,int &y);
6
7   int main(void)
8   {
9   int t1=10,t2=5;
10  swap(t1,t2);
11  cout<<"t1= "<<t1<<" "<<"t2= "<<t2;
12
13  return 0;
14
15  }
16
17  void swap(int &x,int &y)
18  {
19
20  int t=x;
21  x=y;
22  y=t;
23  }
```

Program to exchange between two numbers using function.

```
F:\software\C++\fuction\bin\Debug\fuction.exe
t1= 5 t2= 10
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

The End