Programming in C++

Functions in C++

Dr. Zeyad Safaa Younus Saffawi

Functions in C++

- A **function** is a group of statements that work together to perform a specific task.
- It is like a small program inside the main program.
- Every C++ program has at **least one function**, which is **main()**, and all the simple programs can define additional functions.
- Functions help make the code modular, organized, and easier to understand.

Functions

Structure of a Function:

```
return_type   function_name (parameter_list)  // Function Declarations
              {
              // Function body (statements)
               }
```

Part	Meaning
return_type	The type of value that the function returns (e.g., int, float, char). Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void
function_name	The actual name of the function. The function name and the parameter list together constitute the function signature.
parameter_list	Variables passed to the function for use inside it. It refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
function body	The set of statements (instructions) that define what the function does.

Functions

- A function **declaration** tells the compiler about a function name and how to call the function.

- The actual body of the function can be defined separately.

syntax

	return_type	**function_name**	**(parameter_list)**
example:	int	Sum	(int a, int b)

Functions

- **Calling a Function**
  - To use a function, you will have to call or invoke that function.
  - When a program calls a function, program control is transferred to the called function.
  - A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
  - To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

syntax

function_name **(parameter_list)**

Sum (num1, num2)

Functions

- **Example: Sum of Two Numbers**

```
#include <iostream>
using namespace std;
// Function definition
int Sum (int a, int b)
  {
   int result = a + b;
   return result;
  }
// Main function
int main ()
  {
   int num1 = 8, num2 = 12;
   int total = Sum (num1, num2);   // call the function Sum()
   cout << "The sum is: " << total;
   return 0;
   }
```

Explanation:

The function Sum () takes two numbers (a, b) and returns their sum.

The main program calls the function and prints the result.

Functions

- **Example: Sum of Two Numbers**

```
#include <iostream>
using namespace std;
// Function definition
int Sum (int a, int b)
  {
   int result = a + b;
   return result;
  }
// Main function
int main ()
  {
   int num1, num2;
   cout<< “enter the two numbers: ”;
   cin >> num1 >> num2;
   int total = Sum (num1, num2);   // call the function Sum()
   cout << "The sum is: " << total;
   return 0;
   }
```

Functions

Example: Function to find the maximum of two integers.

```
# include <iostream>
using namespace std;
int max (int num1, int num2)    // function declaration
 {
   // local variable declaration
  int result;
  if (num1 > num2)
     result = num1;
  else
     result = num2;
  return result;
 }
int main ()
 {
  int a = 100;
  int b = 200;
  int ret;
  ret = max (a, b);                  // calling a function max to get max value.
  cout << "Max value is : " << ret << endl;
  return 0;
 }
```

Functions

Example:

```
# include <iostream>
using namespace std;
int max (int num1, int num2)    // function declaration
 {
   // local variable declaration
  int result;
  if (num1 > num2)
     result = num1;
  else
     result = num2;
  return result;
 }
int main ()
 {
  int a,b;
  cout << "Enter the two numbers : ";
  cin >> a>>b;
  int ret;
  ret = max (a, b);                  // calling a function max to get max value.
  cout << "Max value is : " << ret << endl;
  return 0;
 }
```

Functions

- **Function Arguments**
- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.
- The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

1. **Call by reference**
2. **Call by Value**

Function Arguments (Call by reference)

- **Call by reference** means passing the *address* of a variable to a function, not just its value.
- When a variable is passed **by reference**, any changes made inside the function **directly affect** the original variable in the main program.
- **Why Use Call by Reference?**
  - To **modify** variables inside a function and have the changes reflected in the **main program**.
  - To **avoid using return values** when multiple variables need to be updated.
  - To **save memory and time**, because no copies of variables are made.

Syntax:

- It uses the **ampersand (&)** sign in the function parameter list to indicate *reference passing*.

Syntax return type **Function-Name (int &a, int &b)**

- Here, a and b are references — meaning they point to the original variables in memory.

Function Arguments

- **Example 1: Swap Two Numbers**

```
#include <iostream>
using namespace std;
void swap (int &x, int &y)
 {
   int temp = x;
   x = y;
   y = temp;
  }
 int main()
  {
   int a = 5, b = 10;
   cout << "Before swapping: a = " << a << ", b = " << b << endl;
    swap (a, b);       // call by reference
   cout << "After swapping: a = " << a << ", b = " << b << endl;
  return 0;
 }
```

Output:

Before swapping: a = 5, b = 10

After swapping: a = 10, b = 5

The values of a and b actually changed in main(), because we passed them by reference.

Function Arguments (call by value)

- The **call by value** method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.
- In this case, **changes** made to the parameter inside the function have **no effect** on the **argument**.

- **Syntax** return type **Function-Name (int a, int b)**

- By default, C++ uses **call by value** to pass arguments.

Function Arguments

Example:

```
#include <iostream>
using namespace std;
void swap (int a, int b)     // Function definition for swap using Call by Value
{
   int temp = a;
   a = b;
   b = temp;
   cout << "Inside the function (after swap): a = " << a << ", b = " << b << endl;
}
int main()
{
   int x = 10;
   int y = 20;
   cout << "Before function call: x = " << x << ", y = " << y << endl;
   swap (x, y);                  // Calling the function
   cout << "After function call:  x = " << x << ", y = " << y << endl;
   return 0;
}
```

Output:

```
Before function call: x = 10, y = 20
Inside the function (after swap): a = 20, b = 10
After function call:  x = 10, y = 20
```

Default Values for Parameters C++

- A *default argument* means giving a variable in a function a predefined value during defining a function that will be used if the **corresponding argument** is left **blank** when the function is **called**.
- When the function is called, if the caller does **not** provide a value for that parameter, the **default value** is used automatically. You can still change or override the default values when calling the function, but if a **value** is **specified**, this **default value** is **ignored**, and the **passed value** is used instead.

Default Values for Parameters

Function to calculate the sum of two integers

```
# include <iostream>
using namespace std;
int sum(int a, int b=20)
{
int result;
result = a + b;
return (result);
}
int main ()
{
int a = 100;
int b = 200;
int result;
result = sum(a, b);                // calling a function to add the values.
cout << "Total value of sum is :" << result << endl;
result = sum(a);                   // calling a function to add the values again.
cout << "Total value of sum is :" << result << endl;
return 0;
}
```

Output:

Total value is : 300

Total value is : 120

Default Values for Parameters

Example: Write a program to call a function that adds three numbers with default values (10, 20, 30).

```
#include <iostream>
using namespace std;
// Function with default arguments
int add(int a = 10, int b = 20, int c = 30)
{
   int d;
   d = a + b + c;
   return d;
}
// Main function
int main()
{
   int x, y, z, m;
   cout << "Enter three numbers: ";
   cin >> x >> y >> z;                    // Example input: x=5, y=6, z=1
   m = add();
   cout << "add() = " << m << endl;              // 10 + 20 + 30 = 60
   m = add(x);
   cout << "add(x) = " << m << endl;             // 5 + 20 + 30 = 55
   m = add (x, y);
   cout << "add (x, y) = " << m << endl;         // 5 + 6 + 30 = 41
   m = add(x, y, z);
   cout << "add(x, y, z) = " << m << endl;       // 5 + 6 + 1 = 12
   m = add (2, 3, 4);
   cout << "add(2, 3, 4) = " << m << endl;       // 2 + 3 + 4 = 9
   return 0;
}
```

Recursive Function

- A **recursive function** is a function that **calls itself** during its execution.
- It is used to solve problems by **breaking them into smaller subproblems** of the same type. Every recursive function has two main parts:
  - **Base Case:**
    The stopping condition that tells the function when to stop calling itself.
  - **Recursive Case:**
    The part where the function **calls itself** with a smaller or simpler version of the problem.

Syntax:

```
return_type  function_name (parameters)
{
   if (condition)        // Base case
      return value;
   else
      return function_name (modified_parameters);     // Recursive call
}
```

Recursive Function

- factorial function: The factorial of a number is defined as:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 1$$

Using function:

```
int fact (int n)
    {
     int i, f=1;
    for(i=1; i <= n; i++)
    f *= i;
    return f;
   }
```

Using Recursive:

```
 int fact (int n)
{
If (n = = 1)
return 1;
else
return n * fact(n-1);
}
```

Recursive Function

```
#include <iostream>
using namespace std;
int fact (int n)
{
   if (n == 1)                    // Base case
      return 1;
   else
      return n * fact(n - 1);   // Recursive call
}

int main ()
{
   int num;
   cout << "Enter a number: ";
   cin >> num;
   cout << "Factorial of " << num << " = " << fact(num);
   return 0;
}
```

Write a C++ **recursive function** to calculate the **factorial of a number**. Write a **main** program to **call the function** and **print the result**.

How It Works:
If the user enters num = 4, the calls happen like this:
fact (4)
= 4 * fact(3)
= 4 * (3 * fact(2))
= 4 * (3 * (2 * fact(1)))
= 4 * 3 * 2 * 1
= 24

References

- Gaddis, T. (2014). *Starting out with C++: From control structures through objects* (8th ed.). Pearson.
- Soulié, J. (2007, April 24). C++ language tutorial. cplusplus.com.
- Tutorials Point. (n.d.). *Learn C++ programming language*. Tutorials Point.