Unit:3

C++ Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters/arguments, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

## Create a Function

C++ provides some pre-defined functions, such as main ( ), which is used to execute code. But you can also create your own functions to perform certain actions.

To create (often referred to **as declare**) a function, specify the name of the function, followed by parentheses **()**:

void myFunction() {  
  // code to be executed  
}

## Call a Function

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called.

To call a function, write the function's name followed by two parentheses () and a semicolon ;

In the following example, myFunction() is used to print a text (the action), when it is called:

void myFunction()

{  
  cout << "I just got executed!";  
}  
  
int main()

{  
  **myFunction();** // call the function  
  return 0;  
}  
  
// Outputs "I just got executed!"

A function can be called multiple times:

void myFunction()

{  
  cout << "I just got executed!\n";  
}  
  
int main() {  
  **myFunction();**  
  **myFunction();**  
  **myFunction();**  
  return 0;  
}  
  
// I just got executed!  
// I just got executed!  
// I just got executed!

## Function Declaration and Definition

A C++ function consist of two parts:

* **Declaration:** the function's name, return type, and parameters (if any)
* **Definition:** the body of the function (code to be executed)

void **myFunction()** { // **declaration**  
  // the body of the function (**definition**)  
}

**Note:** If a user-defined function, such as myFunction() is declared after the main() function, **an error will occur**. It is because C++ works from top to bottom; which means that if the function is not declared above main().

You will often see C++ programs that have function declaration above main(), and function definition below main(). This will make the code better organized and easier to read:

// **1.** **Function declaration**  
void myFunction();  
  
**// The main method**int main() {  
  myFunction();  // **2. call the function**  
  return 0;  
}  
  
// **3.Function definition**  
void myFunction()

{  
  cout << "I just got executed!";  
}

## Parameters and Arguments

Information/data can be passed to functions as a parameter. Parameters act as variables inside the function.

**Terminology**

* **Formal Parameter :** A variable and its type as they appear in the prototype/declare of the function or method.
* **Actual Parameter :** The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:

void functionName(parameter1, parameter2, parameter3)

{  
  // code to be executed  
}

C++ Functions - Pass By Value

The following example has a function that takes a string called **fname** as parameter. When the function is called, we pass along a first name, which is used inside the function to print the full name:

void myFunction(**string fname**)

{  
  cout << fname << " Refsnes\n";  
}  
  
int main() {  
  myFunction(**"Liam"**);  
  myFunction(**"Jenny"**);  
  myFunction(**"Anja"**);

  myFunction(**"Lalit"**);  
  
  return 0;  
}  
  
// Liam Refsnes  
// Jenny Refsnes  
// Anja Refsnes

//Lalit Refsnes

## Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

If we call the function without an argument, it uses the default value ("Norway"):

void myFunction(**string country = "Norway"**)

{  
  cout << country << "\n";  
}  
  
int main() {  
  myFunction("Sweden");  
  myFunction("India");  
  **myFunction();**  
  myFunction("USA");  
  return 0;  
}  
  
// Sweden  
// India  
// Norway  
// USA

## Multiple Parameters

Inside the function, you can add as many parameters as you want:

void myFunction(**string fname, int age**) {  
  cout << fname << " Refsnes. " << age << " years old. \n";  
}  
  
int main() {  
  myFunction(**"Liam", 3**);  
  myFunction(**"Jenny", 14**);  
  myFunction(**"Anja", 30**);  
  return 0;  
}  
  
// Liam Refsnes. 3 years old.  
// Jenny Refsnes. 14 years old.  
// Anja Refsnes. 30 years old.

## Return Values

The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, string, etc.) instead of void, and use the return keyword inside the function:

**int** myFunction(int x) {  
  **return** 5 + x;  
}

int main() {  
  cout << myFunction(3);  
  return 0;  
}  
  
// Outputs 8 (5 + 3)

You can also store the result in a variable:

int myFunction(int x, int y) {  
  return x + y;  
}  
  
int main() {  
  int z = myFunction(5, 3);  
  cout << z;  
  return 0;  
}  
// Outputs 8 (5 + 3)

# C++ Functions - Pass By Reference

In the examples from the previous page, we used normal variables when we passed parameters to a function. You can also pass a [reference](https://www.w3schools.com/cpp/cpp_references.asp) to the function. This can be useful when you need to change the value of the arguments:

void swapNums(int &x, int &y) {  
  int z = x;  
  x = y;  
  y = z;  
}  
  
int main() {  
  int firstNum = 10;  
  int secondNum = 20;  
  
  cout << "Before swap: " << "\n";  
  cout << firstNum << secondNum << "\n";  
  
  // Call the function, which will change the values of firstNum and secondNum  
  swapNums(firstNum, secondNum);  
  
  cout << "After swap: " << "\n";  
  cout << firstNum << secondNum << "\n";  
  
  return 0;  
}

## Function Overloading

With**function overloading**, multiple functions can have the same name with different parameters:

int myFunction(int x)  
float myFunction(float x)  
double myFunction(double x, double y)

Consider the following example, which have two functions that add numbers of different type:

int plusFuncInt(int x, int y) {  
  return x + y;  
}  
double plusFuncDouble(double x, double y) {  
  return x + y;  
}  
int main() {  
  int myNum1 = plusFuncInt(8, 5);  
  double myNum2 = plusFuncDouble(4.3, 6.26);  
  cout << "Int: " << myNum1 << "\n";  
  cout << "Double: " << myNum2;  
  return 0;

}

int plusFunc(int x, int y)

{  
  return x + y;=13  
}

double plusFunc(double x, double y)

{  
  return x + y;=10.56  
}

int main()

{  
  int myNum1 = plusFunc(8, 5);=13  
  double myNum2 = plusFunc (4.3, 6.26);=10.56

  cout << "Int: " << myNum1 << "\n";  
  cout << "Double: " << myNum2;  
  return 0;

}

Int:13

Double:10.56

[Run example »](https://www.w3schools.com/cpp/showcpp.asp?filename=demo_functions)

1001 🡪memory address of x(&x)

10.1

* Value of x

X 🡪 identifier/variable

\*(&x) 🡪 value at address 1001 🡪 10.1

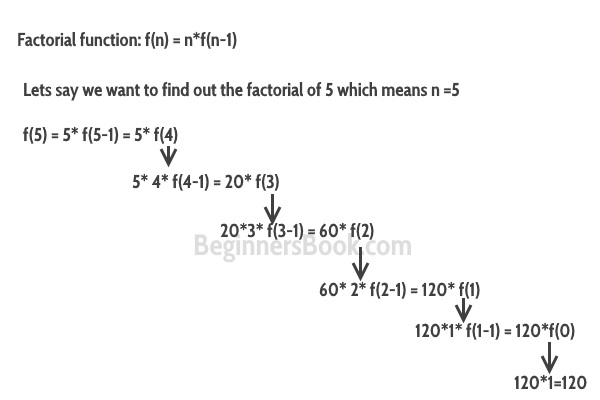
**Recursion**

The process in which a function calls itself is known as recursion and the corresponding function is called the **recursive function**.

The popular example to understand the recursion is factorial function.

**Factorial function:** f(n) = n\*f(n-1), base condition: if n<=1 then f(n) = 1. Don’t worry we will discuss what is **base condition** and why it is important.

In the following diagram. I have shown that how the factorial function is calling itself until the function reaches to the base condition.



#include <iostream.h>

//Factorial function

int f(int n){

/\* This is called the base condition, it is

\* very important to specify the base condition

\* in recursion, otherwise your program will throw

\* stack overflow error.

\*/

   if (n <= 1)

        return 1;

   else

      return n\*f(n-1);

}

int main(){

int num;

  cout<<"Enter a number: ";

   cin>>num;->5

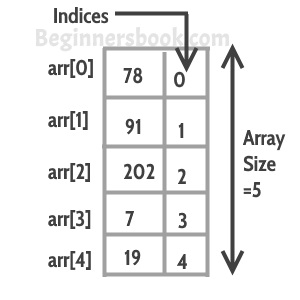
   cout<<"Factorial of entered number: "<<f(num);f(5)

return 0;

}

**Arrays in C++**

An array is a collection of similar items stored in contiguous memory locations. In programming, sometimes a simple variable is not enough to hold all the data.

For example, lets say we want to store the marks of 500 students, having 500 different variables for this task is not feasible, we can define an array with size 500 that can hold the marks of all students.  


Declaring an array in C++

int arr[] = {10, 20, 30, 40, 50};

OR

int arr[5];

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

arr[3] = 40;

arr[4] = 50;

OR

int arr[5] = {10, 20, 30, 40, 50};

## Accessing Array Elements

Array index starts with 0, which means the first array element is at index 0, second is at index 1 and so on. We can use this information to display the array elements. See the code below:

#include <iostream>

int main(){

int arr[] = {11, 22, 33, 44, 55};

cout<<arr[0]<<endl;

cout<<arr[1]<<endl;

cout<<arr[2]<<endl;

cout<<arr[3]<<endl;

cout<<arr[4]<<endl;

return 0;

}

Although this code worked fine, displaying all the elements of array like this is not recommended. When you want to access a particular array element then this is fine but if you want to display all the elements then you should use a loop like this:

#include <iostream>

using namespace std;

int main(){

int arr[] = {11, 22, 33, 44, 55};

int n=0;

while(n<=4){

cout<<arr[n]<<endl;

n++;

}

return 0;

}

# Passing Array to Function in C++

You can pass [array](https://beginnersbook.com/2017/08/cpp-arrays/) as an argument to a function just like you pass variables as arguments. In order to pass array to the function you just need to **mention the array name during function call** like this:

function\_name(array\_name);

## Example: Passing arrays to a function

In this example, we are passing two arrays a & b to the function sum(). This function adds the corresponding elements of both the arrays and display them.

#include <iostream>

/\* This function adds the corresponding

 \* elements of both the arrays and

 \* displays it.

 \*/

**void sum(int arr1[], int arr2[])**

{

int temp[5];

for(int i=0; i<5; i++)

{

temp[i] = arr1[i]+arr2[i];

cout<<temp[i]<<endl;

}

}

int main(){

int a[5] = {10, 20, 30, 40 ,50};

int b[5] = {1, 2, 3, 4, 5};

//Passing arrays to function

sum(a, b); // call

return 0;

}

**Output:**

11

22

33

44

55

## Pointer and arrays

While handling [arrays](https://beginnersbook.com/2017/08/cpp-arrays/) with pointers you need to take care few things. First and very important point to note regarding arrays is that the array name alone represents the base address of array so while assigning the address of array to pointer don’t use ampersand sign(&). Do it like this:  
**Correct:** Because arr represent the address of array.

p = arr;

**Incorrect:**

p = &arr;

### Example: Traversing the array using Pointers

#include <iostream>

int main(){

//Pointer declaration

int \*p;

//Array declaration

int arr[]={1, 2, 3, 4, 5, 6};

//Assignment

p = arr;

for(int i=0; i<6;i++)

{

    cout<<\*p<<endl;

//++ moves the pointer to next int position

    p++;

   }

return 0;

}

**Output:**

1

2

3

4

5

6

**C++ Program to print an Array using Recursion**

Write a program in C++ to print an [Array](https://www.geeksforgeeks.org/arrays-in-c-cpp/) using [Recursion](https://www.geeksforgeeks.org/recursion/)

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2015/05/Arrays.png)

1. **Using**[**static variable**](https://www.geeksforgeeks.org/static-variables-in-c/)**:**Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve their previous value in their previous scope and are not initialized again in the new scope.

**Syntax:**

static data\_type var\_name = var\_value;

|  |
| --- |
| // C++ Program to print  // an Array using Recursion  #include<stdio.h>  #include<conio.h>  // Recursive function to print the array  void **print\_array(**int arr[], int size)  {        // using the static variable      Static int i;0 1 2 3 4 5    // base case      if (i == size) {          i = 0;          cout << endl;          return;      }    // print the ith element      cout << arr[i] << " ";      i++;    // recursive call  **print\_array(arr, size);**  }    int main()  {       int arr[] = { 3, 5, 6, 8, 1 };      int n = sizeof(arr) / sizeof(arr[0]);        print\_array(arr, n);        return 0;  }  Output  3 5 6 8 1 |

**Returning by reference**

[Pointers](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/) and [References](https://www.geeksforgeeks.org/references-in-c/) in C++ held close relation with one another. The major difference is that the pointers can be operated on like adding values whereas references are just an alias for another variable.

* [Functions in C++](https://www.geeksforgeeks.org/functions-in-c/) can return a reference as it’s returns a pointer.
* When function returns a **reference** it means it returns an **implicit** pointer.

Return by reference is very different from[Call by reference](https://www.geeksforgeeks.org/difference-between-call-by-value-and-call-by-reference/). Functions behaves a very important role when variable or pointers are returned as reference.

***dataType& functionName(parameters);*** *where,****dataType****is the****return type****of the****function****,  
and****parameters****are the passed****arguments****to it.*

|  |
| --- |
| // C++ program to illustrate return by reference  #include <iostream>    // Function to return as return by reference  int& returnValue(int& x)  {        // Print the address      cout << "x = " << x           << " The address of x is "           << &x << endl;        // Return reference      return x;  }    int main()  {      int a = 20;      int& b = returnValue(a);        // Print a and its address      cout << "a = " << a           << " The address of a is "           << &a << endl;        // Print b and its address      cout << "b = " << b           << " The address of b is "           << &b << endl;        // We can also change the value of      // 'a' by using the address returned      // by returnValue function        // Since the function returns an alias      // of x, which is itself an alias of a,      // we can update the value of a      returnValue(a) = 13;        // The above expression assigns the      // value to the returned alias as 3.      cout << "a = " << a           << " The address of a is "           << &a << endl;      return 0;  } |

**Output:**

x = 20 The address of x is 0x7fff3025711c

a = 20 The address of a is 0x7fff3025711c

b = 20 The address of b is 0x7fff3025711c

x = 20 The address of x is 0x7fff3025711c

a = 13 The address of a is 0x7fff3025711c

**\*\*\*\*\*\*\*\*\*\*\*\*End #3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***