# FUNCTIONS IN COMPUTER PROGRAMMING

## CSC 205 COMPUTER PROGRAMMING II
## LECTURE NOTE

**PREPARED BY**
**MR YAKUBU ERNEST NWUKU**

**USER DEFINED FUNCTIONS IN C++**

The function in C++ language is also known as procedure or subroutine in other programming languages. To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

In other words, Functions are used to provide modularity to a program. Creating an application using function makes it easier to understand, edit, check errors etc. Basically, A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

### Advantage of functions in C++

There are many advantages of functions.

**Code Reusability:**

By creating functions in C++, you can call it many times. So we don't need to write the same code again and again.

**Code optimization:**

It makes the code optimized, we don't need to write much code.

For Example, Suppose you have to check 3 numbers (531, 883 and 781) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code.

But if you use functions, you need to write the logic only once and you can reuse it several times.

**Syntax for using Functions in C++**

Here is how you define a function in C++,

```
return-type function-name(parameter1, parameter2, ...)
{
   // function-body
}
```

**Return Type** − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

**Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.

**Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list 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 function body contains a collection of statements that define what the function does.

```
// function for adding two values
void sum(int x, int y)
{
   int z;
   z = x + y;
   cout << z;
}

int main()
{
   int a = 10;
   int b = 20;
   // calling the function with name 'sum'
   sum (a, b);
}
```

Here, a and b are two variables which are sent as arguments to the function sum, and x and y are parameters which will hold values of a and b to perform the required operation inside the function.

## Declaring, Defining and Calling a Function

**1. Function declaration:**

It is done to tell the compiler about the existence of the function. That means, 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. A function declaration has the following parts −

```
return_type function_name( parameter list );
```

For the following defined function max(), following is the function declaration −

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so following is also valid declaration −

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function. Let's understand this with help of an example.

```
// function returning the max between two numbers

int max(int num1, int num2) {
  // local variable declaration
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

**2. Calling a Function:**

While creating a C++ function, you give a definition of what the function has to do. 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. For example −

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

// function declaration
int max(int num1, int num2);

int main () {
  // local variable declaration:
  int a = 100;
  int b = 200;
  int ret;

  // calling a function to get max value.
  ret = max(a, b);
  cout << "Max value is : " << ret << endl;
  return 0;
}
```

```
// function returning the max between two numbers
int max(int num1, int num2) {
  // local variable declaration
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Output:
Max value is: 200

Example:
Now, combining all the above concepts into one program :

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

//declaring the function
int sum (int x, int y);

int main()
{
   int a = 10;
   int b = 20;
   int c = sum (a, b);    //calling the function
   cout << c;
}

//defining the function
int sum (int x, int y)
{
   return (x + y);
}
```

Here, initially the function is declared, without body. Then inside main() function it is called, as the function returns sumation of two values, and variable c is there to store the result. Then, at last, function is defined, where the body of function is specified. We can also, declare & define the function together, but then it should be done before it is called.

**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.
While calling a function, there are three ways that arguments can be passed to a function −

**1. call by value :**
This 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.

**NOTE**: By default, C++ uses call by value to pass arguments.
In general, this means that code within a function cannot alter the arguments used to call the function. Consider the program of swapping numbers.

Example :

```
#include <iostream>
using namespace std;
// function definition to swap the values.
void swap(int x, int y) {
  int temp;

  temp = x; /* save the value of x */
  x = y;    /* put y into x */
  y = temp; /* put x into y */

  return;
}
int main () {
  // local variable declaration:
  int a = 100;
  int b = 200;

  cout << "Before swap, value of a :" << a << endl;
  cout << "Before swap, value of b :" << b << endl;

  // calling a function to swap the values.
  swap(a, b);

  cout << "After swap, value of a :" << a << endl;
  cout << "After swap, value of b :" << b << endl;

  return 0;
}
```

Output :

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
```

After swap, value of b :200
Which shows that there is no change in the values though they had been changed inside the function.

**2. call by pointer:**

This method of passing arguments to a function copies the address of an argument into the formal 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 passed argument.

To pass the value by pointer, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to by its arguments

Example :

```
#include <iostream>
using namespace std;
// function definition to swap the values.
void swap(int *x, int *y) {
  int temp;
  temp = *x; /* save the value at address x */
  *x = *y; /* put y into x */
  *y = temp; /* put x into y */

  return;
}
#include <iostream>
using namespace std;

// function declaration
void swap(int *x, int *y);

int main () {
  // local variable declaration:
  int a = 100;
  int b = 200;

  cout << "Before swap, value of a :" << a << endl;
  cout << "Before swap, value of b :" << b << endl;

  /* calling a function to swap the values.
     * &a indicates pointer to a ie. address of variable a and
     * &b indicates pointer to b ie. address of variable b.
  */
  swap(&a, &b);

  cout << "After swap, value of a :" << a << endl;
  cout << "After swap, value of b :" << b << endl;
```

```
   return 0;
}
```
Output :
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100

**3. call by reference**

This method of passing arguments to a function copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

To pass the value by reference, argument reference is passed to the functions just like any other value. So accordingly you need to declare the function parameters as reference types as in the following function swap(), which exchanges the values of the two integer variables pointed to by its arguments

Example :

```
#include <iostream>
using namespace std;
// function definition to swap the values.
// function definition to swap the values.
void swap(int &x, int &y) {
   int temp;
   temp = x; /* save the value at address x */
   x = y;    /* put y into x */
   y = temp; /* put x into y */

   return;
}
#include <iostream>
using namespace std;

// function declaration
void swap(int &x, int &y);

int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;

   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl;

   /* calling a function to swap the values using variable reference.*/
   swap(a, b);
```

```
  cout << "After swap, value of a :" << a << endl;
  cout << "After swap, value of b :" << b << endl;

  return 0;
}
```
Output :
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100

Note : Although we are getting same outputs of above programs for References and pointer but there is difference between them .
Pointers: A pointer is a variable that holds memory address of another variable. A pointer needs to be dereferenced with * operator to access the memory location it points to.
**References**: A reference variable is an alias, that is, another name for an already existing variable. A reference, like a pointer is also implemented by storing the address of an object.A reference can be thought of as a constant pointer (not to be confused with a pointer to a constant value!) with automatic indirection, i.e the compiler will apply the * operator for you.

## Types of Functions

There are two types of functions in C programming:

**Library Functions:** are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x) etc.

**User-defined functions:** are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code. Here our major area of study is User-defined functions. so, we will focus on this only.
User-defined functions C++ allows programmer to define their own function.
A user-defined function groups code to perform a specific task and that group of code is given a name(identifier). When the function is invoked from any part of program, it all executes the codes defined in the body of function.

Example of User-defined functions

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

// Function prototype (declaration)
int add(int, int);

int main()
{
   int num1, num2, sum;
   cout<<"Enters two numbers to add: ";
   cin >> num1 >> num2;
```

```
    // Function call
    sum = add(num1, num2);
    cout << "Sum = " << sum;
    return 0;
}

// Function definition
int add(int a, int b)
{
    int add;
    add = a + b;

    // Return statement
    return add;
}
```

Output

```
Enters two integers: 8
-4
Sum = 4
```

## Types of User-defined functions

There can be 4 different types of user-defined functions, they are:

Function with no arguments and no return value
Function with no arguments and a return value
Function with arguments and no return value
Function with arguments and a return value
Below, we will discuss about all these types, along with program examples.

**1. Function with no arguments and no return value :**

Such functions can either be used to display information or they are completely dependent on user inputs.

Example :
In the following program, prime() is called from the main() with 0no arguments.
prime() takes the positive number from the user and checks whether the number is a prime number or not.
Since, return type of prime() is void, no value is returned from the function.

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

void prime();

int main()
```

```
{
   // No argument is passed to prime()
   prime();
   return 0;
}

// Return type of function is void because value is not returned.
void prime()
{
   int num, i, flag = 0;
   cout << "Enter a positive integer enter to check: ";
   cin >> num;

   for(i = 2; i <= num/2; ++i)
   {
      if(num % i == 0)
      {
         flag = 1;
         break;
      }
   }
   if (flag == 1)
   {
      cout << num << " is not a prime number.";
   }
   else
   {
      cout << num << " is a prime number.";
   }
}
```

**2. Function with no arguments and a return value :**

In the following program, prime() function is called from the main() with no arguments. prime() takes a positive integer from the user. Since, return type of the function is an int, it returns the inputted number from the user back to the calling main() function. Then, whether the number is prime or not is checked in the main() itself and printed onto the screen.

Example

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

int prime();

int main()
{
   int num, i, flag = 0;
```

```
   // No argument is passed to prime()
   num = prime();
   for (i = 2; i <= num/2; ++i)
   {
      if (num%i == 0)
      {
         flag = 1;
         break;
      }
   }
   if (flag == 1)
   {
      cout<<num<<" is not a prime number.";
   }
   else
   {
      cout<<num<<" is a prime number.";
   }
   return 0;
}

// Return type of function is int
int prime()
{
   int n;

   printf("Enter a positive integer to check: ");
   cin >> n;

   return n;
}
```

**3. Function with arguments and no return value:**

We are using the same function as example again and again, to demonstrate that to solve a problem there can be many different ways. This time, we have modified the above example to make the function prime() take one int value as argument, but it will not be returning anything. In the program, positive number is first asked from the user which is stored in the variable num.

Then, num is passed to the prime() function where, whether the number is prime or not is checked and printed. Since, the return type of prime() is a void, no value is returned from the function.

Example

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

void prime(int n);
```

```
int main()
{
   int num;
   cout << "Enter a positive integer to check: ";
   cin >> num;

   // Argument num is passed to the function prime()
   prime(num);
   return 0;
}

// There is no return value to calling function. Hence, return type of function is void. */
void prime(int n)
{   int i, flag = 0;
   for (i = 2; i <= n/2; ++i)
   {
      if (n%i == 0)
      {
         flag = 1;
         break;
      }
   }
   if (flag == 1)
   {
      cout << n << " is not a prime number.";
   }
   else {
      cout << n << " is a prime number.";
   }
}
```

**4. Function with arguments and a return value :**

This is the best type, as this makes the function completely independent of inputs and outputs, and only the logic is defined inside the function body. In the following program, a positive integer is asked from the user and stored in the variable num. Then, num is passed to the function prime() where, whether the number is prime or not is checked.

Since, the return type of prime() is an int, 1 or 0 is returned to the main() calling function. If the number is a prime number, 1 is returned. If not, 0 is returned. Back in the main() function, the returned 1 or 0 is stored in the variable flag, and the corresponding text is printed onto the screen.

Example

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

int prime(int n);
```

```
int main()
{
   int num, flag = 0;
   cout << "Enter positive integer to check: ";
   cin >> num;

   // Argument num is passed to check() function
   flag = prime(num);

   if(flag == 1)
      cout << num << " is not a prime number.";
   else
      cout<< num << " is a prime number.";
   return 0;
}
/* This function returns integer value.  */
int prime(int n)
{
   int i;
   for(i = 2; i <= n/2; ++i)
   {
      if(n % i == 0)
         return 1;
   }

   return 0;
}
```

**Exercise**

1. Write a program that uses math library functions
2. Write a program that uses Character functions
3. Write a program that uses String functions