CS-15 C++ and Object Oriented Programming

Unit – 1 Part – 2

Presented By : Dhruvita Savaliya

Functions in C++

Topics :

- The main function
- Call by reference
- Return by reference
- Inline function
- Default arguments
- Const arguments
- Functions overloading

The Main Functions :

- The execution of each and every c++ program is start from the main() function. It is the entry point of a program execution.
- The general format of main() function is as follow.
- **Syntax:**

```
Return type main()
{
        Body of the main function
}
```

Or as a command line argument :

```
Return type main([int argc, char *argv[ ],[char ** envp]])
{
        Body of the main function
}
```

- According to the above format the **return type of the main() function must be either void or int.**
- Here, return type specifies the status of the program termination.
- The main() function can also takes arguments from command prompt to.
- It is known as command line arguments.
- argc specifies the total number at argument. (Argument Count)
- It is the argument counter.
- Its value is always positive.
- Argv represents the argument vector(array).
- It holds pointer to the argument passed from the command line.
- **argv[] is one kind of an array so it holds the data in following manner:**
- argv[0] = pointer to the name of the executable program.
- Argv[1], argv[2]..... argv[n] = pointers to argument strings
- envp represents an environment parameter. It is optional.

Example :

```
#include<iostream.h>
#include<conio.h>
int main(int argc, char *argv[])
{
   clrscr();
   int i;
   cout<<endl<<"Total arguments="<<argc;
   cout<<endl<<"Program name is="<<argv[0];
   cout<<endl<<"Other Arguments are\n\n";
   for(i=1;i<argc;i++)
   {
          cout<<endl<<argv[i];
   }
   cout<<endl<<"Total Number of Argument are : "<<argc;
   getch();
   return(0);
}
```

run the above program from **dos shell** and enter following arguments:

```
C:\TC\BIN\SOURCE> prog_name.exe hello
```

Call by Value :

- Whenever a function is called, its arguments are passed to the function definition.
- In C++ we can use value of variable as arguments.
- When we pass arguments as value, it passes to the function as give.
- **Example :**

```
    total=sum(10,20);
```

Call By Reference :

- Call by reference means we can call the function by its reference means address of variable.
- A reference as its name, is like alias.
- It refer to the same entity.
- A variable and its reference are tightly attached with each other.
- So, change in one it will also change in the other.
- When call any function by its reference any modifications made through the formal pointer parameter is also reflected in the actual parameter.
- It has functionality of pass-by-pointer and the syntax of call-by-value.
- In the function declaration parameter are to be received by reference must be preceded by the **& operator and arguments or parameters pass same as call by value.**
- However any modification in the variable in function body directly reflected to the actual parameter.
- **Example :**
- void fun(int &x, int &y);
- With the call by reference we can directly change the value of variable in the user define function because we use the reference of variable.
- In general case we can not change the value of variable permanently.

Example of Call By Refeerence :

```
#include <iostream.h>

void f(int  *x)
{
        cout<<endl<<x;
        *x--;
}

int main()
{
        int a = 5;
        cout << a << endl;
        f(&a);
        cout << a << endl;
        return 0;
}
```

Example for Function Argument as **REFERENCE Variable** :

```
#include <iostream.h>

void f(int & x)
{
        x--;
}

int main()
{
        int a = 5;
        cout << a << endl;
        f(a);
        cout << a << endl;
        return 0;
}
```

Difference Between call by value & call by reference

Call by Value	Call by Reference
Changes made on arguments will not affect to the original values.	Changes made on arguments will affect the original values.
Cannot return multiple values.	Multiple values can be returned.
Normal variables are used .	Reference variables are used.
Example : Any normal function.	**Example :** Swapping, Bubble sort.

Return by Reference :

- Pointers and References 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++ can return a reference as it's returns a pointer.
- When function returns a reference it means it returns a implicit pointer.
- Return by reference is very different from Call by reference.
- Functions behaves a very important role when variable or pointers are returned as reference.
- It will work with the GLOBAL VARIABLE.
- A function can be called on the receiving side of an assignment.

- **Syntax :**

  dataType & functionName(parameters);

where,
dataType is the return type of the function,
and parameters are the passed arguments to it

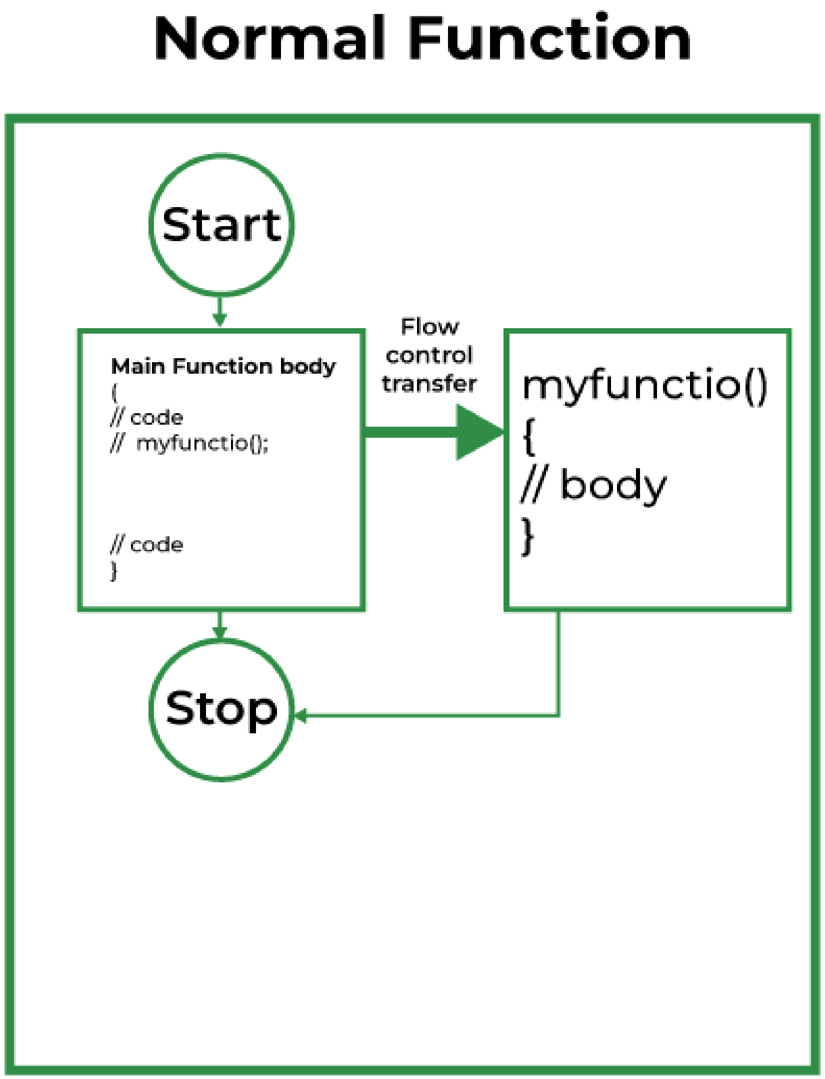
Example :

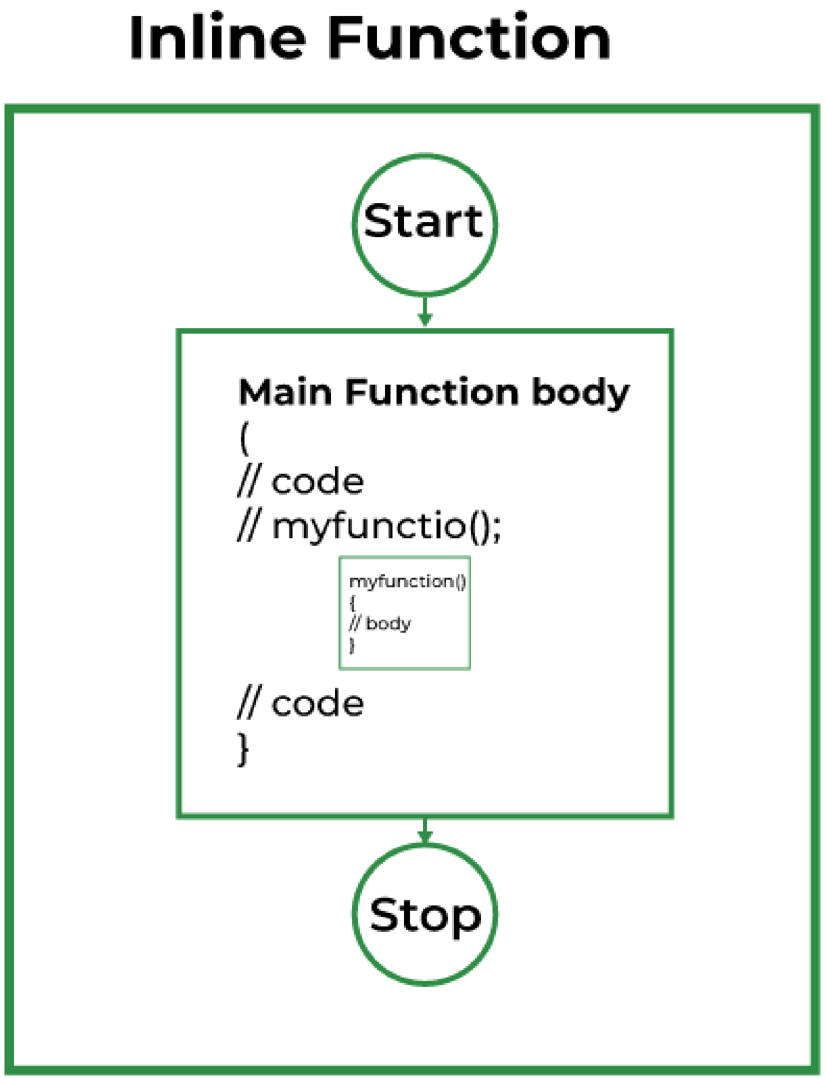
```
#include<iostream.h>
#include<conio.h>
int a;
int& fun( )
{
        return a;
}
void main( )
{
   clrscr();
   fun( ) = 10;
   cout<<a;
   getch( );
}
```

Inline Function :

- This function is expanded inline at a time of compilation that is a function body is inserted in place of function call and so run time overhead for function linkage is reduced, but executable file size is increase.
- inline function definition must be known to the compiler before function call occurs.
- If function body contains loops, goto, switch, or a static variables then such function can not expanded inline.
- A recursive function also can not be expanded inline.
- If inline expansion is not possible then compiler ignores the word "inline" and considers it has normal UDF.
- In short, inline function is very much similar to macro definition with #define.
- **Syntax :**

```
inline return_type fun_name ( arguments )
{
        // body of function
}
```


Normal Function
Start
Main Function body
(
// code
// myfunctio();
// code
}
Flow control transfer
myfunctio()
{
// body
}
Stop


Inline Function
Start
Main Function body
(
// code
// myfunctio();
myfunction()
{
// body
}
// code
}
Stop

Example :

```
inline int cube(int s)
{
    return s * s * s;
}
void main()
{
        cout << "The cube of 3 is: " << cube(3) << "\n";
}
```

- **The compiler may not perform inlining in such circumstances as:**
- If a function contains a loop. (*for, while and do-while*)
- If a function contains static variables.
- If a function is recursive.
- If a function return type is other than void, and the return statement doesn't exist in a function body.
- If a function contains a switch or goto statement.

Default Argument :

- We can provide default values for function arguments in function definition or in function declaration and so if one or more arguments are mission in a function call then it takes its default value.
- We can assign default values for argument in the order from right to left only.
- **Example :**

```
#include <iostream>
int sum(int x, int y, int z = 0, int w = 0)
{
   return (x + y + z + w);
}
void main()
{
   cout << sum(10, 15) << endl;
   cout << sum(10, 15, 25) << endl;
   cout << sum(10, 15, 25, 30) << endl;
}
```

Const arguments :

- The keyword const specifies that the value of variable will not change throughout the program.
- If anyone attempt to after the value of variable defined with this qualifier an error can be created.
- A function can also take an argument as a const. which is specifies no any modification on the value.
- Const Arguments means value of arguments can not change.
- **Example :**

```
void fun(const int n) //n value is constant now
{
        //n=5; not possible
        cout<<n;
}
main()
{
        fun(43);
}
```

Functions Overloading :

- Overloading means use of same name / symbol to perform different work.
- Function overloading means function overloading, multiple functions can have the same name with different parameters.
- Function overloading means the use of same function to perform different action.
- Function overloading is also called function polymorphism.
- Poly means many, and morph means form: a polymorphic function is many-formed.
- In other word, the function is known as overloaded function if any other function with the same name is defined.
- This overloaded functions must differ either in number of arguments or in there data types.
- At the time of function call depending upon the number of actual arguments & their data types, the appropriate function definition will be executed.

- **Rules of Function Overloading in C++ :**

1. The functions must have the same name
2. The functions must have different types of parameters.
3. The functions must have a different set of parameters.
4. The functions must have a different sequence of parameters.

- **Examples :**

```
void test(int x, int y);
void test(int x, int y, int z);
void test(int x, float y, char z);
void test(int a, float b);
```

- **Example :**

```
#include<iostream.h>
void fun(int a)
{
        cout<<a<<endl;
}
void fun(char f[ ])
{
        cout<<f<<endl;
}
void fun(char c)
{
        cout<<c<<endl;
}
void main()
{
        fun(12);
        fun("hello");
        fun('d');
}
```