# Functions in C++

The ways of supporting functions have changed and improved in C++ as compared to C. These changes have been incorporated to C++ to make it *safer* and more *readable*.

**Function Prototype:**

- Function prototype is a declaration that defines- the arguments passed to the function and the type of value returned by the function.

  Example-

  *double fool (float, int);*

- In C++, through prototypes the compiler makes sure that the actual arguments (those used in calling function) and formal arguments (those used in called function) match in number, order and type.
- All functions in C++ must be prototyped.
- In C++, if one need to pass a long as an int, he/she can do it by type casting.

**Styles of defining functions in C++:**

```
double fool (a, b)
int a; float b;
{
// Some code
}
```

(Kernighan & Ritchie style)

```
double fool (int a, int b)
{
// Some code
}
```

(Prototype like style)

**Function Overloading:**

- With the facility of function overloading in C++, one can have multiple functions with the same name, unlike C, where all functions in a program must have unique names.
- There are three different functions in C, that return the absolute value of an argument-

```
int abs(int i);
long abs(long l);
double fabs(double d);
```

In C++, this can be done by using a single name-

```
#include <iostream>
using namespace std;
int abs(int );
long abs(long );
```

```
double abs(double );

int main()
{

    int i =-25,j;
    long l = -100000,m;
    double d = -12.34,e;
    j = abs(i);
    m = abs(l);
    e = abs(d);
    cout<<endl<<j<<endl<<m<<endl<<e<<endl;
    return 0;
}

int abs(int ii)
{
    return (ii>0?ii:ii*-1);
}
long  abs(long ll)
{
    return (ll>0?ll:ll*-1);
}
double abs(double dd)
{
    return (dd>0?dd:dd*-1);
}
```

- The function to be called by the compiler in the above case can be identified from the type of the argument being passed during the function call.
- If we make a call as bellow-

```
ch = abs('A');
```

It is not going to work as we have not declared an "*abs()*" to handle the character. Explicitly we can handle it as below-

```
ch = abs((int) 'A');
```

- Let us take one example-

```
typedef INT int;
void display(int);
void display(INT);
```

The above is not function overloading. (INT is just a name given to int)

- Just Check the error in the below code and try to understand-

```
#include<iostream>
using namespace std;
void display(char *);
void display(const char *);
int main()
{
	char *ch1 = "Hello";
	const char *ch2 = "Bye";
	display(ch1);
	display(ch2);
	return 0;

}

void display(char *p)
{
	cout<<p<<endl;
}
void display(const char *p)
{
	cout<<p<<endl;
}
```

**Default Values for Function Arguments:**

- In C, if a function is defined to get 2 arguments, one must have to send 2 values to this function while calling. If only one is passed, the second argument will be assigned some garbage value.
- C++ has a capability of defining default values for the arguments that are not passed when the function call is made.

  Example-

```
// CPP Program to demonstrate Default Arguments
```

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

// A function with default arguments,
// it can be called with
// 2 arguments or 3 arguments or 4 arguments.
int sum(int x, int y, int z = 0, int w = 0)
{
    return (x + y + z + w);
}

// Driver Code
int main()
{
    // Statement 1
    cout << sum(10, 15) << endl;

    // Statement 2
    cout << sum(10, 15, 25) << endl;

    // Statement 3
    cout << sum(10, 15, 25, 30) << endl;
    return 0;
}
```

- If one argument is missing when the function is called, it is assumed to be the last argument.
- Default arguments are given only in the function prototype and should not be repeated in the function definition.
- Default value for an argument can be a global constant, a global variable, or a global function call.

```
int myfunc(flag = display());
```

**Operator Overloading:**

- By overloading operators we ca give additional meaning to operators like +, *, -, <=, >= etc. We can overload the + operator to do the following task –

```
char str1[20] = "Nagpur";
char str2[] = "Bomabay";
char str3[20];
```

```
 strcpy (str3, str1);
 strcat(str3,str2);
```

After overloading the operator +, we can do the same task as below-

```
Str3=str1+str2;
```

- Tackling Complex number in C Style-

```
#include<iostream>
#include<conio.h>
using namespace std;
struct Complex
{
     double real, img;
};
Complex Complex_Set(double r, double i);
Complex Complex_Add(Complex, Complex);
void Complex_Print(Complex c);
int manin()
{
     Complex a ,b, c;
     a = Complex_Set(1.0,1.0);
     b = Complex_Set(2.0,2.0);
     c = Complex_Add(a,b);
     cout<<"c=";
     Complex_Print(c);
     getch();
     return 0;
}
 Complex Complex_Set(double r, double i)
 {
     Complex temp ;
     temp.real = r;
     temp.img  = i;
     return temp;
 }

 void Complex_Print(Complex t)
 {
     cout<<"("<<t.real<<','<<t.img<<")"<<endl;
 }

 Complex Complex_Add(Complex c1, Complex c2)
 {
     Complex temp;
     temp.real = c1.real+c2.real;
     temp.img  = c1.img+c2.img;
     return temp;
 }
```

- Tackling the complex number in C++

```
#include<iostream>
using namespace std;
struct Complex
{
     double real, img;
};

Complex Complex_Set(double r, double i);
void Complex_Print(Complex c);
Complex operator + (Complex c1, Complex c2);
Complex operator - (Complex c1, Complex c2);
int main()
{
     Complex a ,b, c, d;
     a = Complex_Set(1.0,1.0);
     b = Complex_Set(2.0,2.0);
     c = a+b;
     d = b+c-a;
     cout<<"c=";
     Complex_Print(c);
     cout<<"d=";
     Complex_Print(d);
     return 0;
}
 Complex Complex_Set(double r, double i)
 {
     Complex temp ;
     temp.real = r;
     temp.img  = i;
     return temp;
 }

 void Complex_Print(Complex t)
 {
     cout<<"("<<t.real<<','<<t.img<<")"<<endl;
 }

 Complex operator + (Complex c1, Complex c2)
 {
     Complex temp;
     temp.real = c1.real+c2.real;
     temp.img  = c1.img+c2.img;
     return temp;
 }
Complex operator - (Complex c1, Complex c2)
 {
     Complex temp;
     temp.real = c1.real-c2.real;
     temp.img  = c1.img-c2.img;
     return temp;
 }
```