# Unit-I (c)

## Function Templates:

Function template can be used to create a family of functions with different argument types. The general format of a function temple is :

```
template<class T>
returntypefunctionname(arguments of type T)
{
   //……
   //body of function with type T
   //whenever appropriate
   //…….
}
```

- It is similar to class template except that a function is defined instead of a class.
- We must use the template parameter T as and when necessary in the type of data.

Example: The declaration for the function template that will swap two values of a given type of data is:

```
template<class T>
void  swap(T &x, T &y)
  {
     T temp = x;
     x = y;
     y = temp;
   }
```

- This declaration will create one function for each type of data. For example, in the following program, two swap function will be generated from the template function, one for each type.

```
#include<iostream>
using namespace std;
template<class T>
void swap()
{
   T temp = x;
    x = y;
    y=  temp;
}
```

```
int main()
{
int a=10, b=50;
float m=5.0, n=10.5;
cout<<"\n The values of a and b before swap are:  "<<a<<"  "<<b<<endl;
swap(a,b);
cout<<"\n The values of a and b after swap are:  "<<a<<"   "<<b<<endl;

cout<<"\n The values of m and n after swap are:  "<<m<<"  "<<n<<endl;
swap(m,n);
cout<<"\n The values of m and n after swap are:  "<<m<<"   "<<n<<endl;
 }
```

The output would be:

```
The values of a and b before swap are    :  10  50
The values of a and b after swap are     :  50  10
The values of m  and b before swap are  :  5.010.5
The values of a and b after swap are     :  10.5 5.0
```

We very often use the sort function for sorting arrays of various types such as int and double the template function for buble sort is:

```
Template <class T>
void buble( T1 v[], int n)
{
for(inti=0;i<n-1;++I )
for(j=0;j<n-i-1; ++j )
if(v[j]> v[j+1)
     {
         T1 temp = v[j];
v[j] = v[j+1];
v[j+1] = temp;
     }
}

template<class T2>
void swap( T2  &a, T2 &b)
{
  T2 temp = a;
```

```
  a=b;
  b=temp;
}
template<class T>
void display(T a[], int n)
{
for(inti=0; i<n; ++i)
cout<<a[i]<< "  ";
cout<<endl;
}
int main()
{
int X[10] = { 50, 20, 30, 45, 12, 34, 65, 33, 10, 27};
char Y[5] = { 'b', 'c', 'r', 'p','a'};
flat Z[5] = {35.5, 24.0, 41.2, 50.25, 10.25};

cout<<"\nSorted list of integers is: ";
sort(X,10);
display( X, 10 );

cout<<"\nSorted list of character is: ";
sort(Y, 5);
display( Y, 5 );

cout<<"\nSorted list of floats is: ";
sort( Z, 5 );
display( Z, 5 );
return 0;
}
```

OUTPUT:

```
Sorted list of integers is:
10 12 20  27 30 33 34 45  50 65
Sorted list of characters:
a b c p r
Sorted list of floats is:
10.24 24.0 35.5  41.2, 50.25
```

Q: Write a function template to find the minimum and maximum value from a given list of elements?

## Function templates with multiple parameters:

To declare function templates with multiple parameters use a comma separated list in the function template.

```
template<class T1, class T2, .....>
returntypefunctionname(arguments of type T1, T2, ........)
{
        .....
    .....
    .....    //function body
    .....
}
```

Example:

```
template<class T1, class T2>
void display(T1 x, T2 y)
{
cout<<x<<"  " <<y<<"\n";
}

int main()
{
cout<<"\nCalling function template with int and character string type parameters…\n";
display(2000, "CBIT");
cout<<"\nCalling function template with float and integer type parameters……..\n";
display(12.54, 1234);
return 0;
}
```

Output:

```
Calling function template with int and character string type parameters…
2000 CBIT
Calling function template with float and integers type parameters…
12.54 1234
```

## Overloading of Template Functions:

A template function may be overloaded either by template functions or ordinary functions of its name. In such cases, the overloading resolution is accomplished as follows:

1. Call an ordinary function that has an exact match.
2. Call a template function that could be created with an exact match.
3. Try normal overloading resolution to ordinary functions and call that matches.

An error is generated if no match is found.

Note: No automatic conversations are applied to arguments on the template functions.

Example: The following example shows how a template function is overloaded with an explicit function.

```
//Template function with explicit function
#include<iostream>
#include<cstring>
using namespace std;
template<class T>
void display(T x)                       //overloaded template function 'display()'
{
cout<<"\nOverloaded Template Display 1: "<<x<<endl;
}
template<class T1, class T2>
void display(T1 x, T2 x)                //Overloaded template function 'display()'.
{
cout<<"\nOverloaded Template Display 2:  "<<x<< ", "<<y<<endl;
}
void display(int x)                     //Overloaded generic display function
{
        cout<<"Explicit display:  "<<x<<endl;
}
int main()
{
display(100);
display(12.55);
display(100, 12.55);
display('C');
return 0;
}
```

The output of the above program is:

```
Explicit disl[play: 100
Overloaded Template Display 1: 12.55
Overloaded Template Display 2: 100, 12.55
Overloaded Template Display 1: C
```