OVERLOADING: It refers to the use of same task for different purposes.

FUNCTION OVERLOADING

- **Functions having same name differ by number and type of arguments is called function overloading.**

- **Function overloading is using same function name multiple times with in the same program but with different parameters**. Complier differentiates which function is to be called depending upon:
  1. The number of arguments/parameters.
  2. Type of argument
  3. Return type of function.

Program to demonstrate function overloading:

```
#include<iostream>
using namespace std;
void area()
{
    cout<<"I will calculate\n ";
}

void  area(int x)
{
    cout<<”square area” <<x*x<<endl;
}

void area(int p,int q)
{
cout<<”rectangle area is”<<p*q<<endl;
}

void area(float a,float b)
{
    cout<<”float value is”<<a*b<<endl;
}

main()
{
    area();
```

```
  area(2,3);
  area(2);
   area(2.6f,3.6f);
}
```

ADVANTAGES OF FUNCTION OVERLOADING ARE:

- Different values can be passed to same function
- For a programmmer it will be easy to remember the same function name every time.

NOTE:

- Function overloading should be done with caution. That is we should not overload function that are not related to each other.
- Also if we are using classes and objects in function overloading then all the functions should be present in the same class.
- Functions of different classes cannot be overloaded.

TEMPLATES

- Template is a mechanism that makes one function or class to handle many kinds of data types.
- When templates are used with functions they are **function templates**
- When templates are used with classes they are **class templates or generic classes.**

Function Template:

If a program logic and operations are identical for each data type, this may be performed more conveniently using function templates. All function template definition begins with the keyword template followed by list of formal type parameters to the function template enclosed in **<>**. Every formal parameter is preceded by keyword **class** or **type name.**

Program to demonstrate function templates

```
#include <iostream.h>
template <class T>
T max(T  &a, T  &b)
```

```
{
if(a>b)
return a;
else
return b;
}

main()
{
cout << max(10, 20);
cout << max('v','m');
cout << max(3.5f,4.5f);
}
```

- After seeing a function template compiler remembers it for future use.
- Compiler will not generate any code using function template since it don't know that kind of data it is going to handle.
- When a function call is made depending on the type of values passed compiler substitutes that data type in place of 'T' in the function template.

Program to demonstrate class templates (bubble sort)

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

template<class t>
class bubble
{
t a[20];
public:
void get(int);
void sort(int);
void display(int);
};
template<class t>
void bubble<t>::get(int n)
{
```

```
cout<<"enter elements";
for(int i=0;i<n;i++)
cin>>a[i];
}
template<class t>
void bubble<t>::display(int n)
{
cout<<"sorted array is";
for(iint i=0;i<n;i++)
cout<<a[i];
}

template<class t>
void bubble<t>::sort(int n)
{
t temp;
for(int i=0; i<n; i++)
for(int j=0;j<n-1;j++)
if(a[j] > a[j+1])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}

main()
{
bubble<int> b1;
cout<<"integer sorting";
b1.get(5);
b1.sort(5);
b1.display(5);

bubble<char> b2;
```

```
cout<<"character sorting";
b2.get(5);
b2.sort(5);
b2.display(5);
}
```

- We can use multiple parameters in both class and function templates
- A specific class created from a class template is called as template class and the process of creating a template class is known as INSTANTIATION.
- Template functions can also be overloaded.