CHAPTER 5
TEMPLATES

Er. Ganga Gautam

OUTLINES

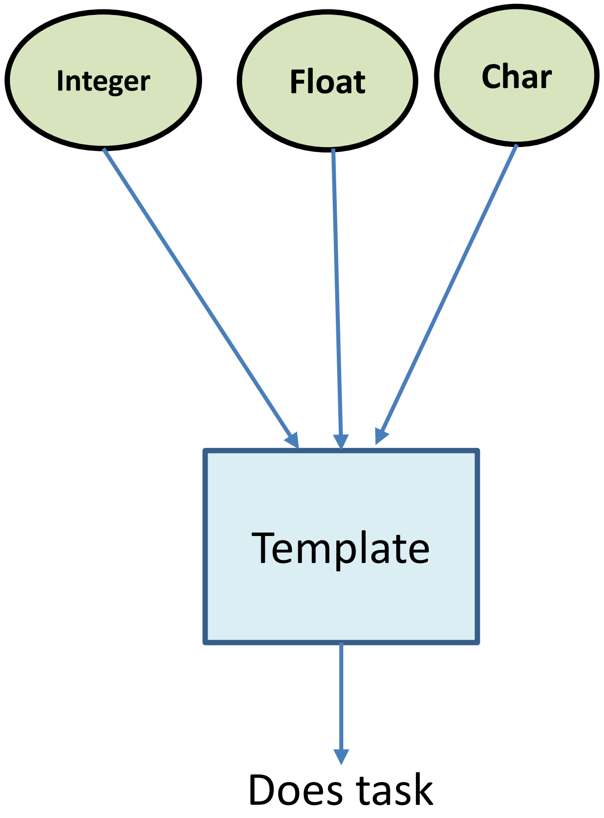
1. Generic Programming
2. Class Template and Function Template
3. Standard Template Library (STL): Container, Algorithm, Iterator

Generic programming

- Generic Programming enables the programmer to write a general algorithm which will work with all data types.
- It eliminates the need to create different algorithms for each and every data types (integer, string or a character).
- Generics can be implemented in C++ using **Templates.**
- The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types.
- For example, we need a ***sort()*** function for different data types.
- Rather than writing many sort() function for each and every datatypes, we can write one ***sort()*** and pass data type as a argument.

Templates

- Templates are powerful features of C++ which allows us to write generic programs.
- Templates allow programmer to create a common class or function that can be used for a variety of data types.
- The parameters used during its definition are of generic type and can be replaced later by actual parameters.
- Template is the method for writing a single function or single class for a family of similar functions or classes in generic manner.
- The generic manner means it is a method of parameterized declaration in such a way that the argument can accept all data types according to the passed value.



Advantages of templates

1. **Type safety**

   The templates are more type-safe because their parameters are known at compile time and strict type checking is done at compile time itself.

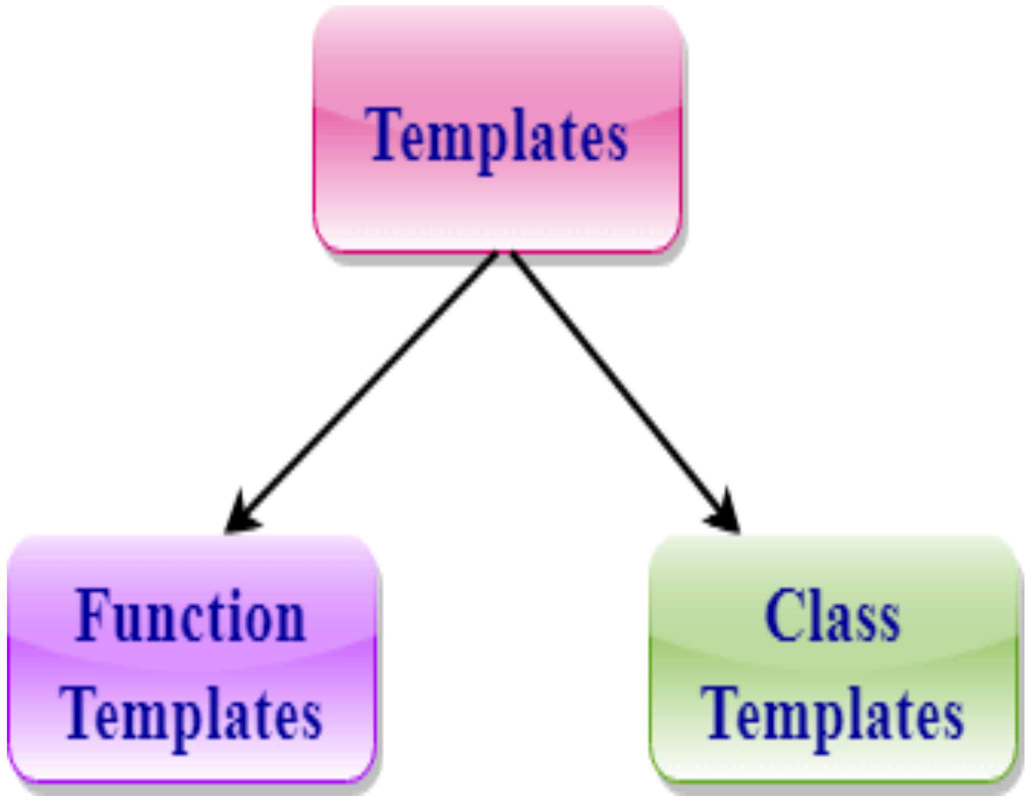
2. **Reusability**

   We use templates in situation that results in duplication of same code for multiple types. For eg: we can use function templates to create a set of functions that apply the same algorithm to different data types.

3. **Performance**

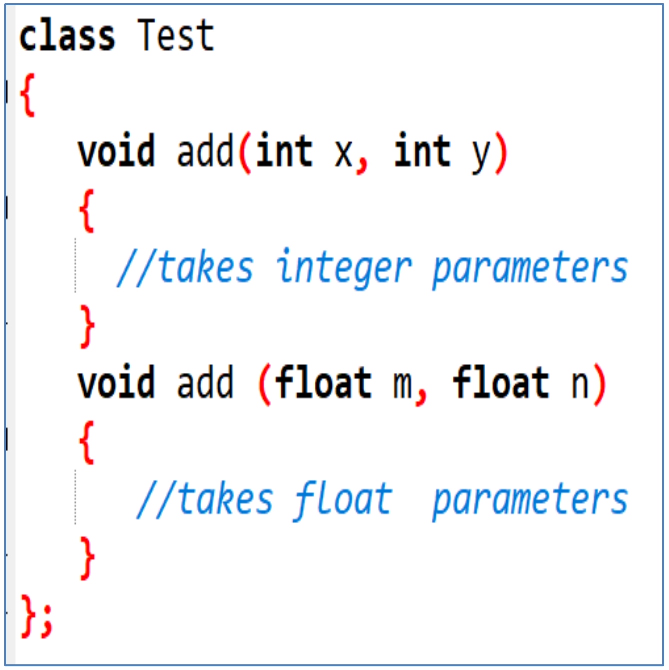
   Defining and using templates are a compile time mechanism. Because of this, there is no runtime overhead associated with using them.

Classification of Templates



Function Template (Generic function)

- Normally, if we need to perform identical operations on two or more types of data, we use function overloading to create two functions with the required function declaration.
- However, a better approach would be to use function templates



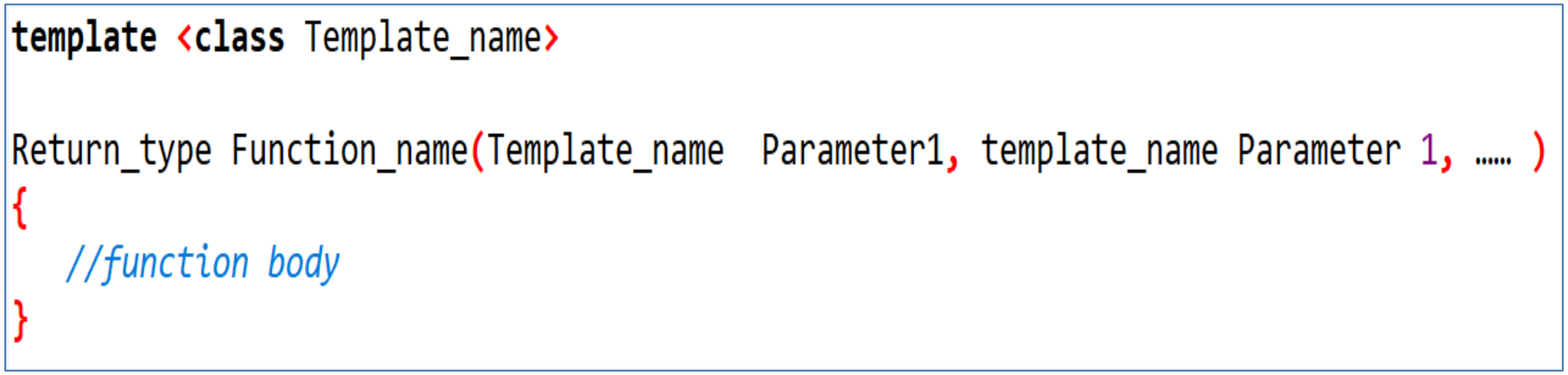
```
class Test
{
   void add(int x, int y)
   {
     //takes integer parameters
   }
   void add (float m, float n)
   {
      //takes float  parameters
   }
};
```

Function Template (Contd.)

- When a single function is written for a family of similar functions, then it is called a Function template.
- The function template does not specify the actual data types of the argument that a function accepts but it uses a generic or parameterized data types

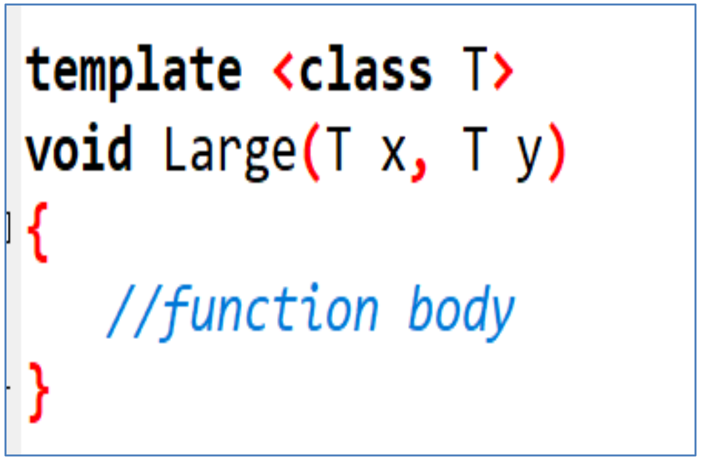
Function Template (Generic function)

- Syntax:



```
template <class Template_name>

Return_type Function_name(Template_name  Parameter1, template_name Parameter 1, ..... )
{
   //function body
}
```

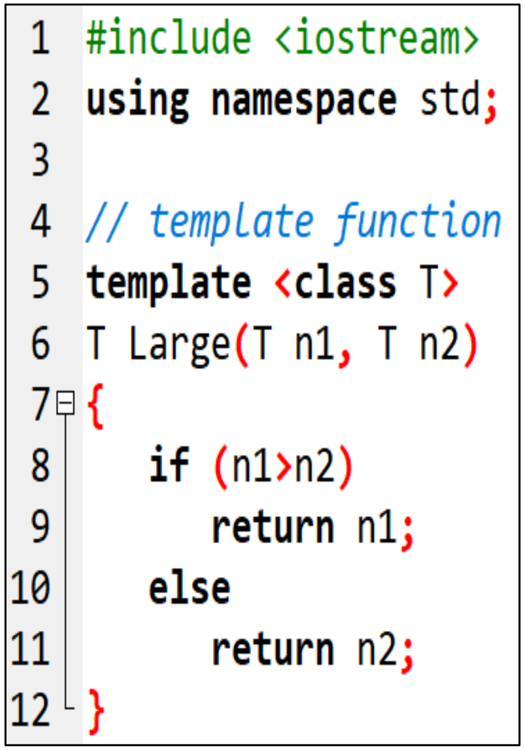
- Example:



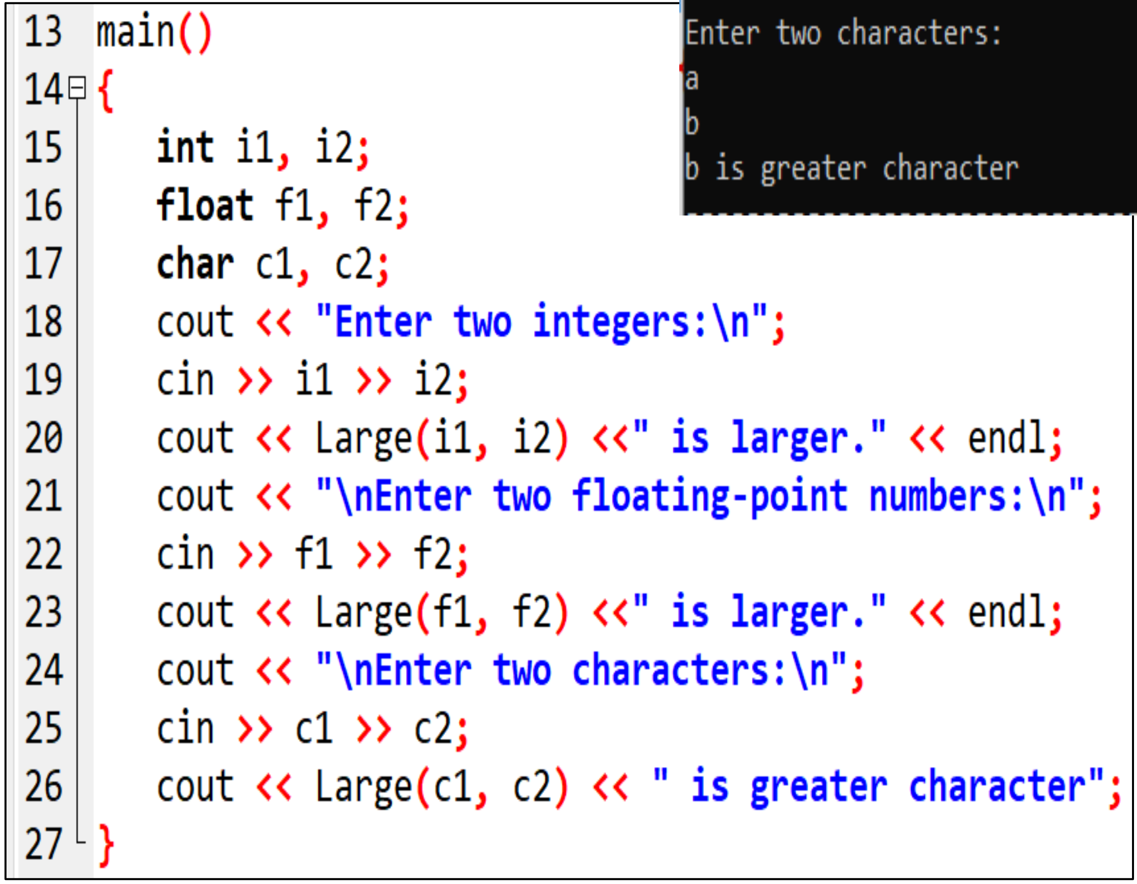
```
template <class T>
void Large(T x, T y)
{
    //function body
}
```

Sample program 6.1

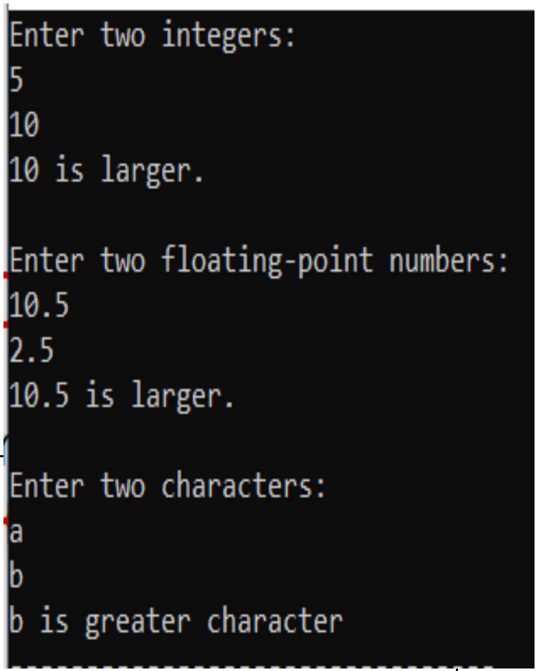
WAP using function template to accept two integers, two floating numbers and two characters, and then display the largest among them



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

// template function
template <class T>
T Large(T n1, T n2)
{
    if (n1>n2)
        return n1;
    else
        return n2;
}
```



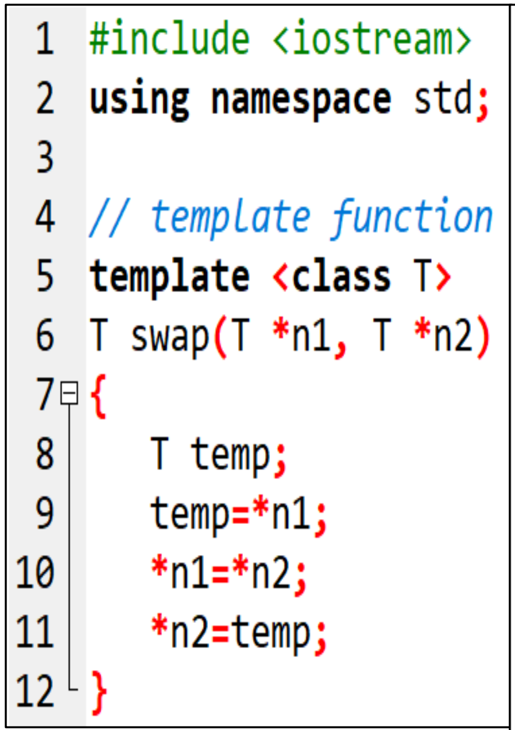
```cpp
main()
{
    int i1, i2;
    float f1, f2;
    char c1, c2;
    cout << "Enter two integers:\n";
    cin >> i1 >> i2;
    cout << Large(i1, i2) <<" is larger." << endl;
    cout << "\nEnter two floating-point numbers:\n";
    cin >> f1 >> f2;
    cout << Large(f1, f2) <<" is larger." << endl;
    cout << "\nEnter two characters:\n";
    cin >> c1 >> c2;
    cout << Large(c1, c2) << " is greater character";
}
```



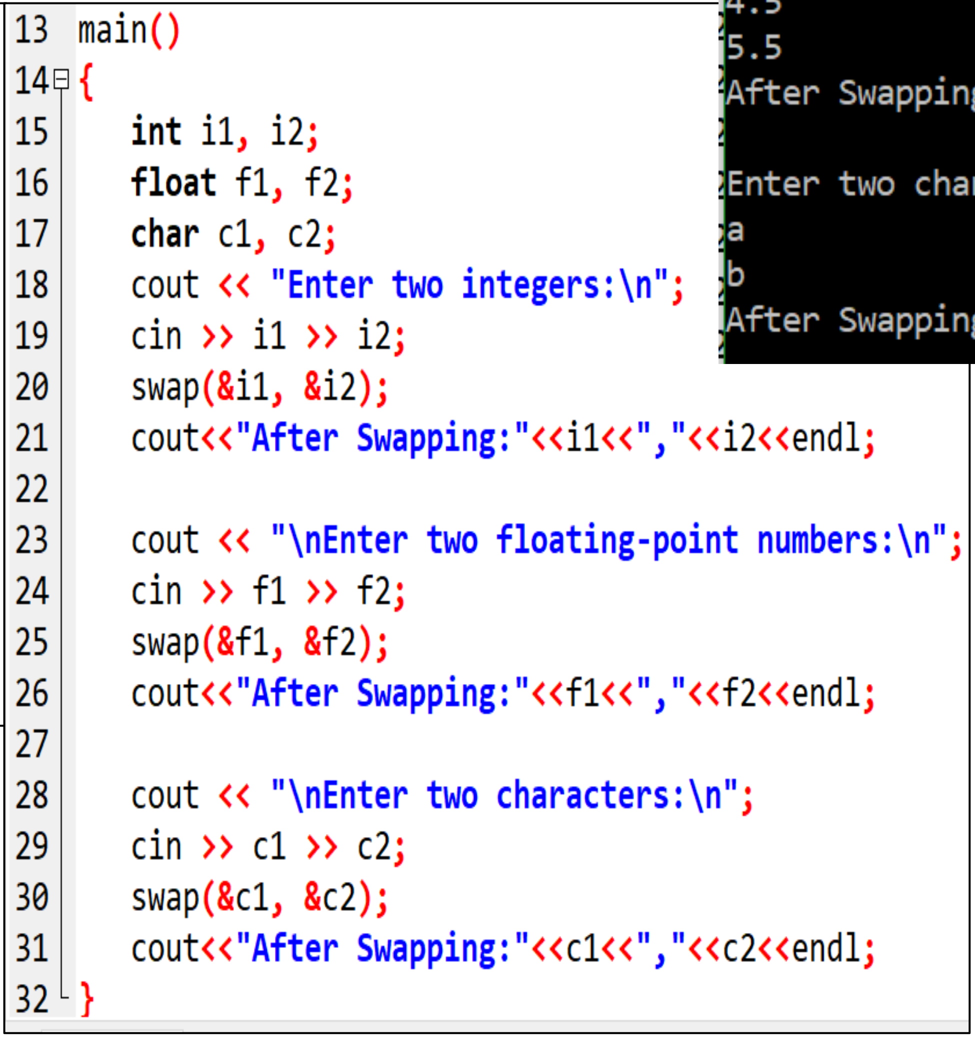
```
Enter two integers:
5
10
10 is larger.

Enter two floating-point numbers:
10.5
2.5
10.5 is larger.

Enter two characters:
a
b
b is greater character
```

Sample program 6.2

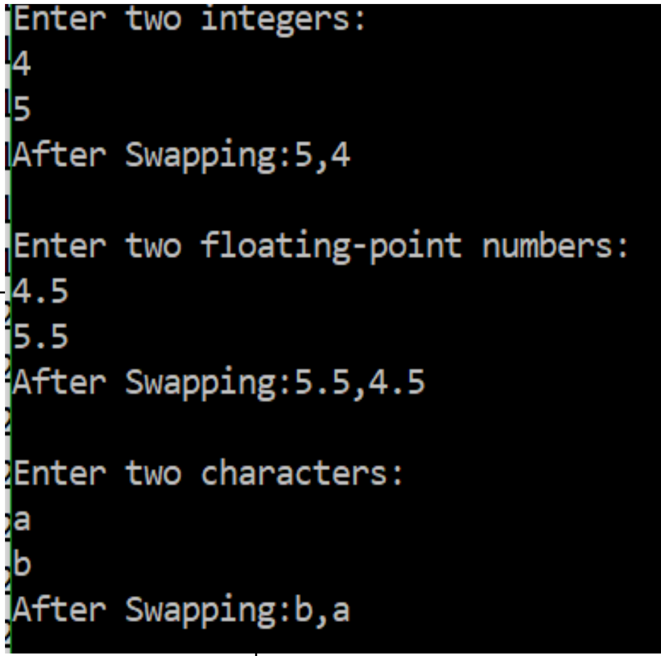
WAP using function template to accept two integers, two floating numbers and two characters, and then swap them.



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

// template function
template <class T>
T swap(T *n1, T *n2)
{
    T temp;
    temp=*n1;
    *n1=*n2;
    *n2=temp;
}
```



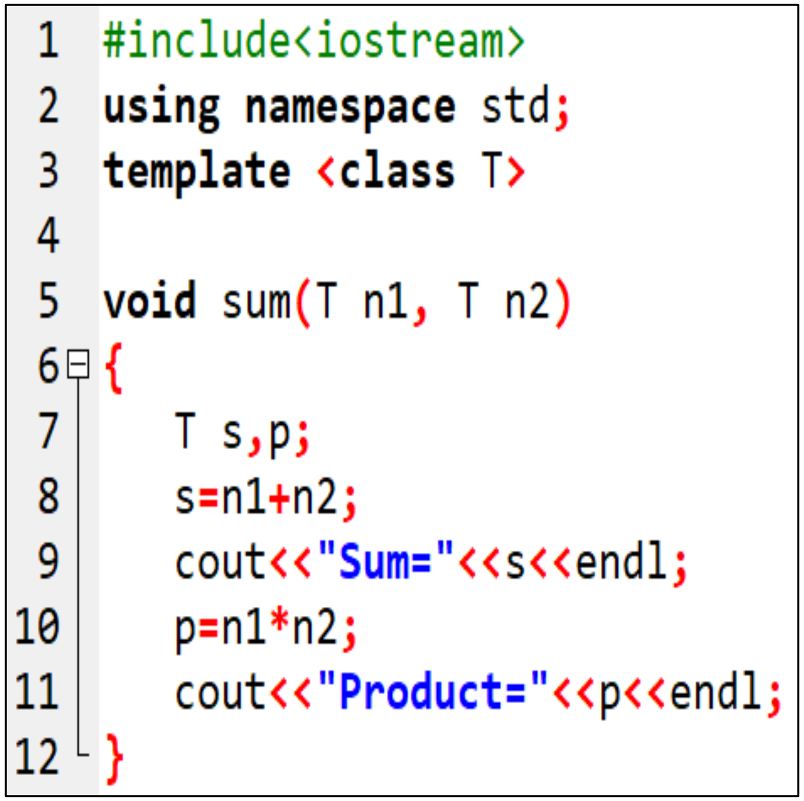
```cpp
main()
{
    int i1, i2;
    float f1, f2;
    char c1, c2;
    cout << "Enter two integers:\n";
    cin >> i1 >> i2;
    swap(&i1, &i2);
    cout<<"After Swapping:"<<i1<<","<<i2<<endl;

    cout << "\nEnter two floating-point numbers:\n";
    cin >> f1 >> f2;
    swap(&f1, &f2);
    cout<<"After Swapping:"<<f1<<","<<f2<<endl;

    cout << "\nEnter two characters:\n";
    cin >> c1 >> c2;
    swap(&c1, &c2);
    cout<<"After Swapping:"<<c1<<","<<c2<<endl;
}
```



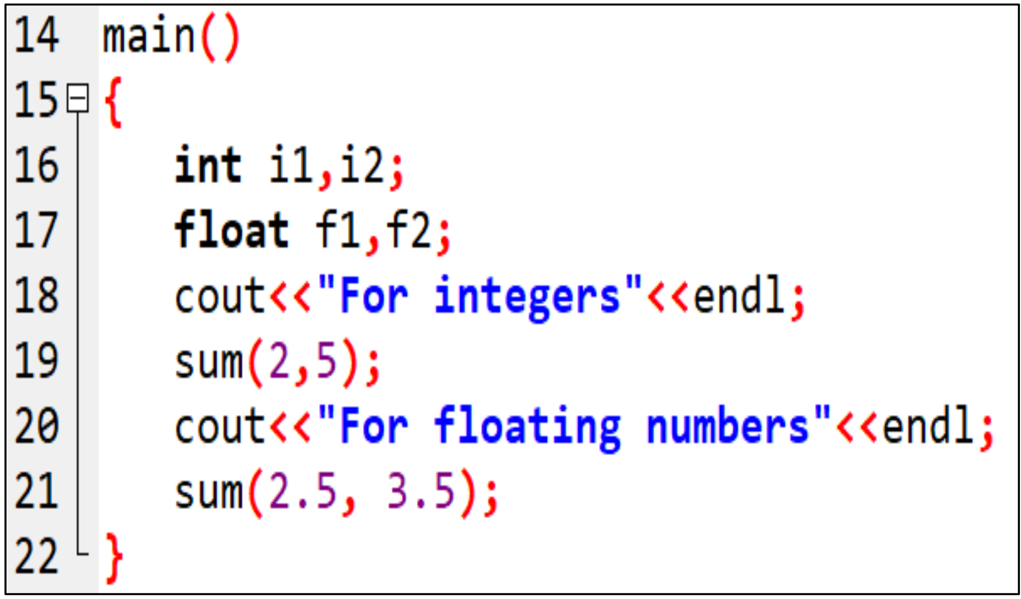
```
Enter two integers:
4
5
After Swapping:5,4

Enter two floating-point numbers:
4.5
5.5
After Swapping:5.5,4.5

Enter two characters:
a
b
After Swapping:b,a
```

Sample program 6.3

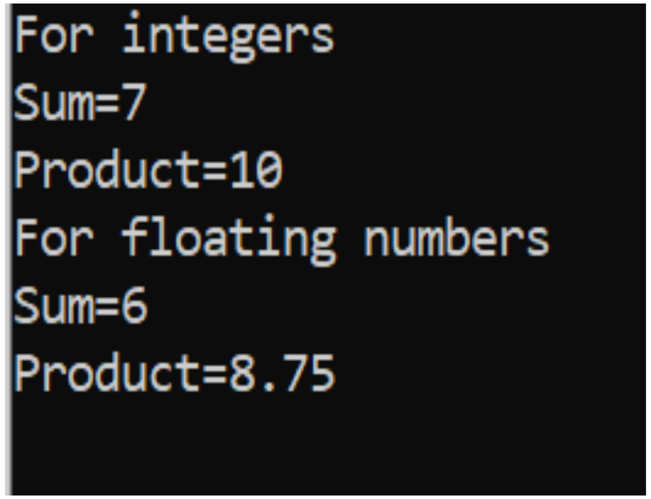
Write a function template to calculate the sum and product of two numbers (either integer or floating).



```cpp
#include<iostream>
using namespace std;
template <class T>

void sum(T n1, T n2)
{
    T s,p;
    s=n1+n2;
    cout<<"Sum="<<s<<endl;
    p=n1*n2;
    cout<<"Product="<<p<<endl;
}
```



```cpp
main()
{
    int i1,i2;
    float f1,f2;
    cout<<"For integers"<<endl;
    sum(2,5);
    cout<<"For floating numbers"<<endl;
    sum(2.5, 3.5);
}
```



```
For integers
Sum=7
Product=10
For floating numbers
Sum=6
Product=8.75
```

Practice question

1. WAP using function template, to accept two integers and display the sum and average, and also two floating numbers and display the sum and average,
2. WAP using function template, to accept five numbers and display the largest among them.
3. Compute the roots of quadratic equation by using function template.

b) Class Template (Generic class)

- When a single class is written for a family of similar class then it is called class template.
- C++ offers us the ability to create a class that contains one or more data types that are generic or parameterized.
- The process of declaring the class template is same as that of function template.

b) Class Template (Contd.)

- Syntax for defining Template class:

```
template <class Template_name>
class ClassName
{
   TemplateName variableName;
   public:
      Constructor (TemplateName   parameters)
      ... .. ...
};
```

- E

```
template <class T>
class Demo
{
   T n;
   public:
      Demo (T x)
         ... .. ...
};
```

- Syntax for defining object

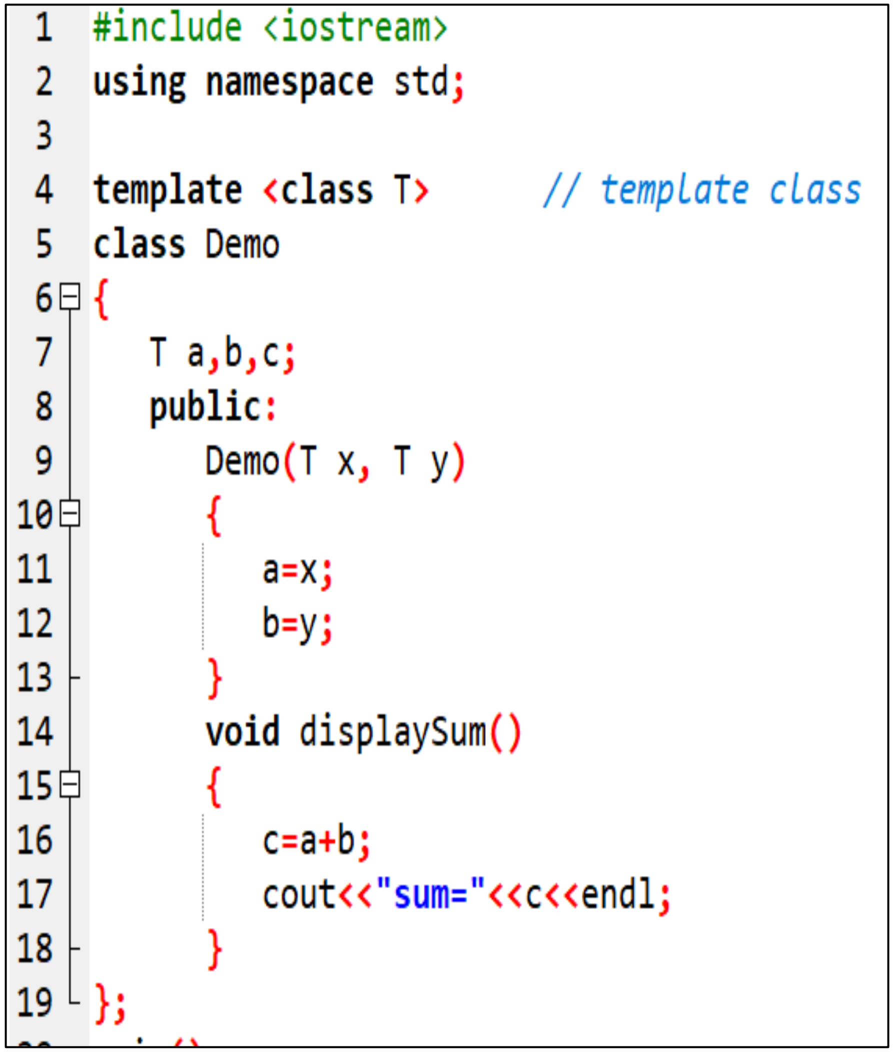
```
className <datatype1> object1;
className <datatype2> object2;
```

- Example:

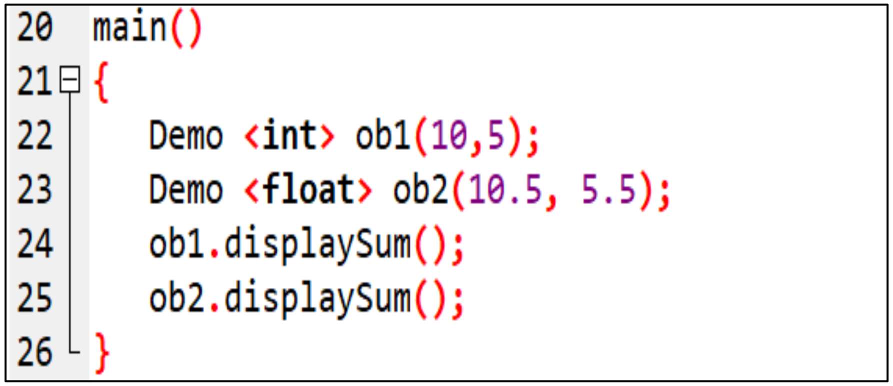
```
Demo <int> ob1;
Demo <float> ob2;
```

Sample program 6.4

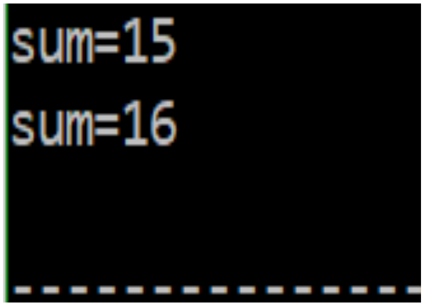
WAP using class template to initialize two integers and two floating numbers at compile time via object creation and then display the sum.



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

template <class T>       // template class
class Demo
{
    T a,b,c;
    public:
        Demo(T x, T y)
        {
            a=x;
            b=y;
        }
        void displaySum()
        {
            c=a+b;
            cout<<"sum="<<c<<endl;
        }
};
```



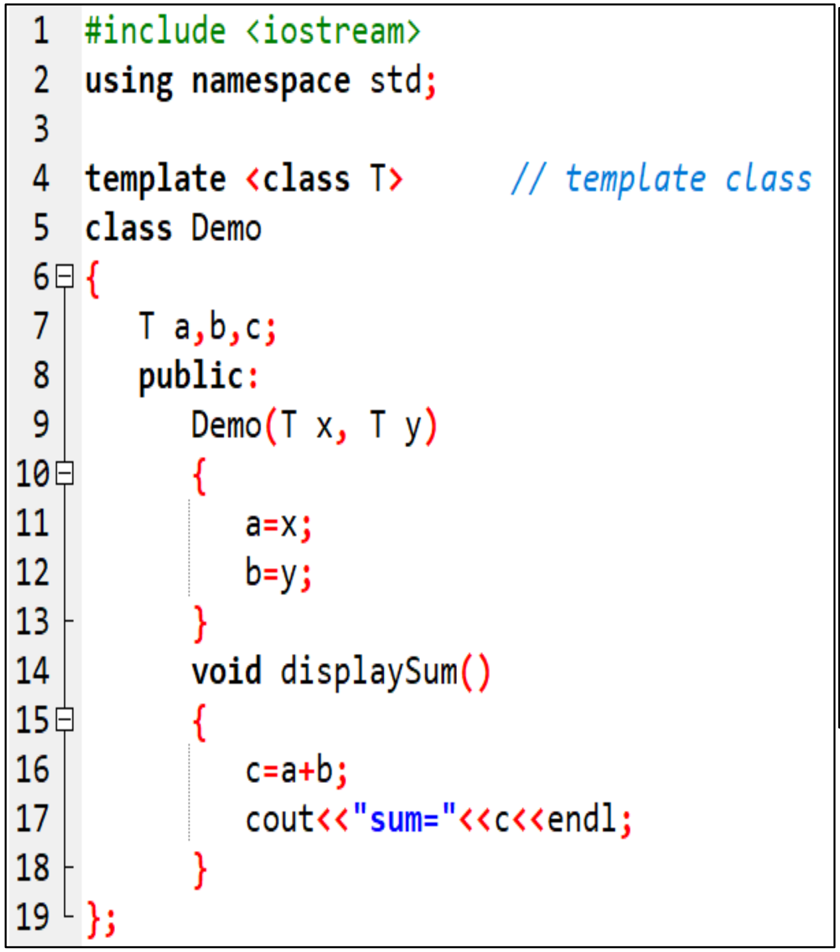
```cpp
main()
{
    Demo <int> ob1(10,5);
    Demo <float> ob2(10.5, 5.5);
    ob1.displaySum();
    ob2.displaySum();
}
```



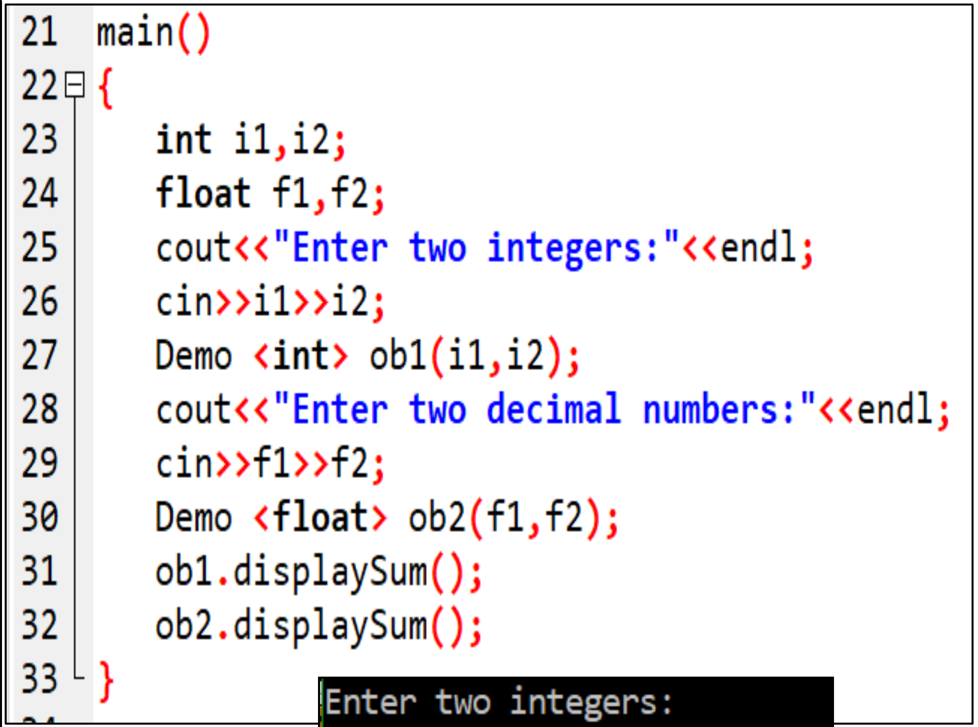
```
sum=15
sum=16

---------------
```

Sample program 6.5

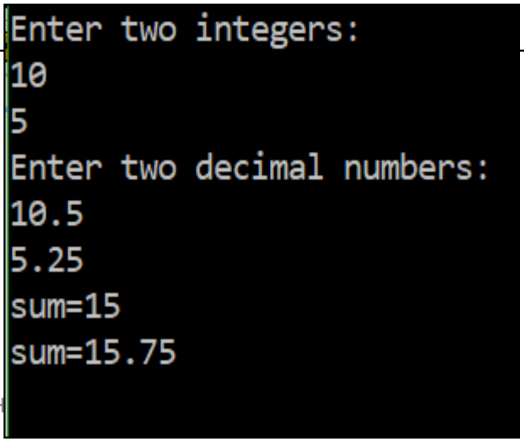
WAP using class template to accept two integers and two floating numbers from user and then display the sum.



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

template <class T>       // template class
class Demo
{
    T a,b,c;
    public:
       Demo(T x, T y)
       {
          a=x;
          b=y;
       }
       void displaySum()
       {
          c=a+b;
          cout<<"sum="<<c<<endl;
       }
};
```



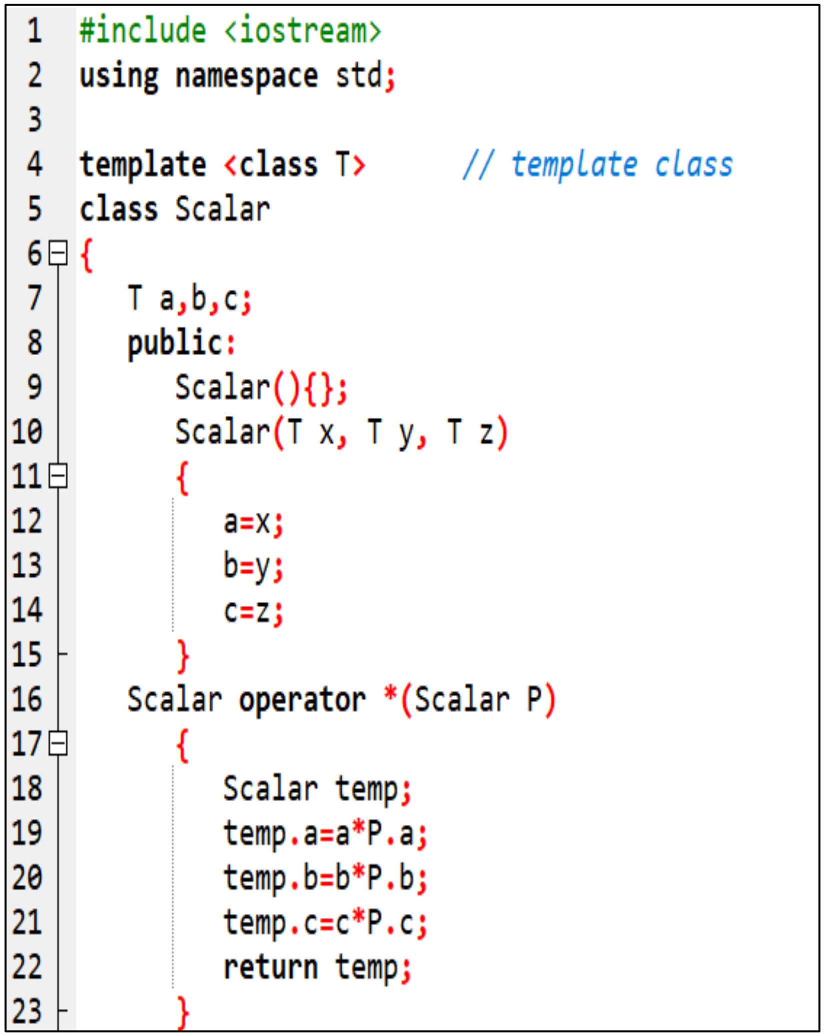
```cpp
main()
{
    int i1,i2;
    float f1,f2;
    cout<<"Enter two integers:"<<endl;
    cin>>i1>>i2;
    Demo <int> ob1(i1,i2);
    cout<<"Enter two decimal numbers:"<<endl;
    cin>>f1>>f2;
    Demo <float> ob2(f1,f2);
    ob1.displaySum();
    ob2.displaySum();
}
```



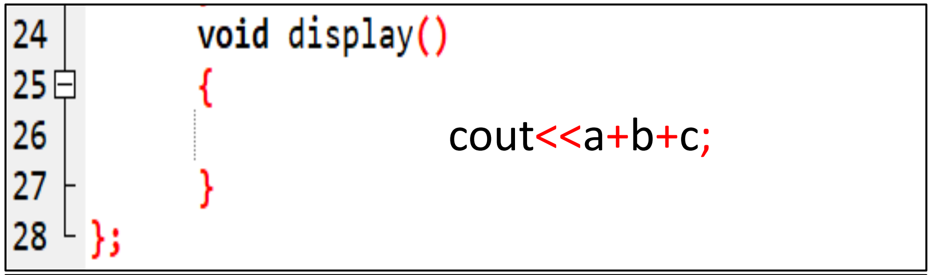
```
Enter two integers:
10
5
Enter two decimal numbers:
10.5
5.25
sum=15
sum=15.75
```

Sample program 6.6

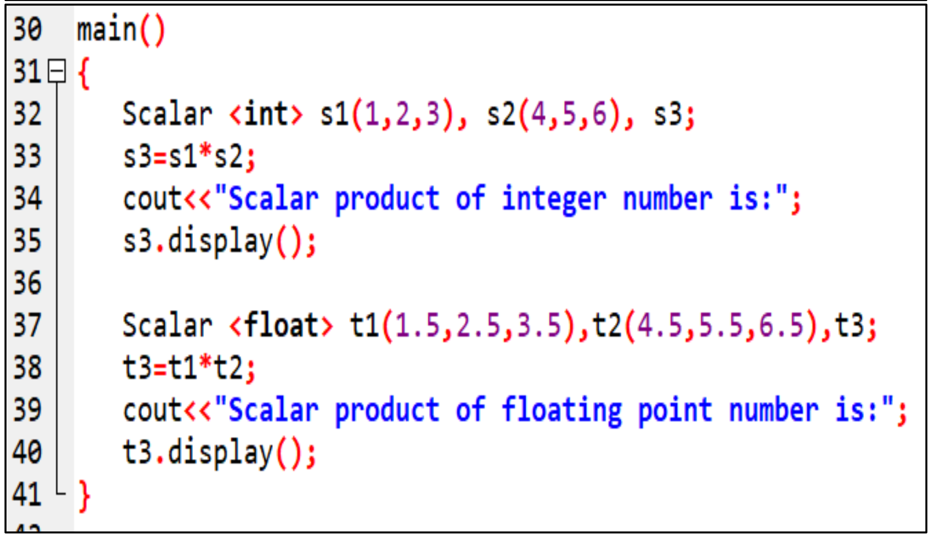
Create a template class to find the scalar product of vectors of integers and vectors of floating point numbers.



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

template <class T>       // template class
class Scalar
{
    T a,b,c;
    public:
        Scalar(){};
        Scalar(T x, T y, T z)
        {
            a=x;
            b=y;
            c=z;
        }
    Scalar operator *(Scalar P)
        {
            Scalar temp;
            temp.a=a*P.a;
            temp.b=b*P.b;
            temp.c=c*P.c;
            return temp;
        }
```



```cpp
        void display()
        {
                    cout<<a+b+c;
        }
};
```



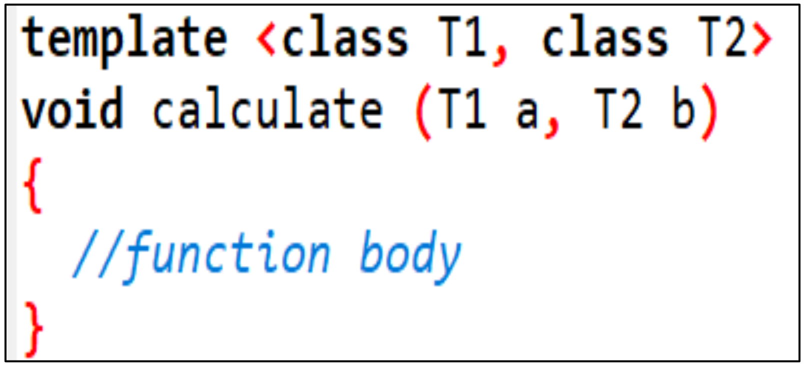
```cpp
main()
{
    Scalar <int> s1(1,2,3), s2(4,5,6), s3;
    s3=s1*s2;
    cout<<"Scalar product of integer number is:";
    s3.display();

    Scalar <float> t1(1.5,2.5,3.5),t2(4.5,5.5,6.5),t3;
    t3=t1*t2;
    cout<<"Scalar product of floating point number is:";
    t3.display();
}
```

Practice question

- Create a class template vector with a type parameter T which holds an array of type T. Also include constructor to initialize the objects, a function to calculate scalar product of two vectors by overloading ‘ * ’ operator so that the statements like v1 *v2 are permissible. Write a main function to test the above class template and find the sclar product of following pairs of vectors {1,2,3} , {4,5,6} & {2.5,3.5,4.5}, {1.5,2.5,3.5}.

Function Template with multi parameters

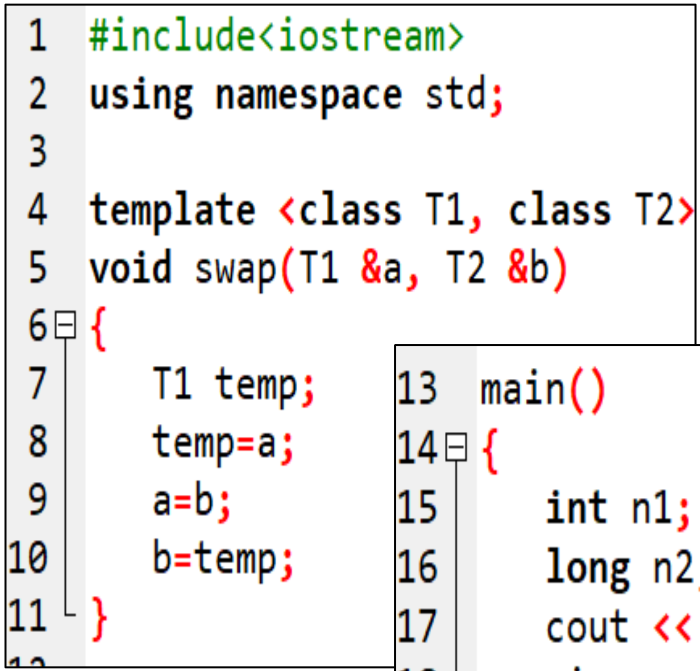
- We can also define function templates that accept more than one type parameter,
- simply by specifying more template parameters between the angle brackets.
- Syntax:



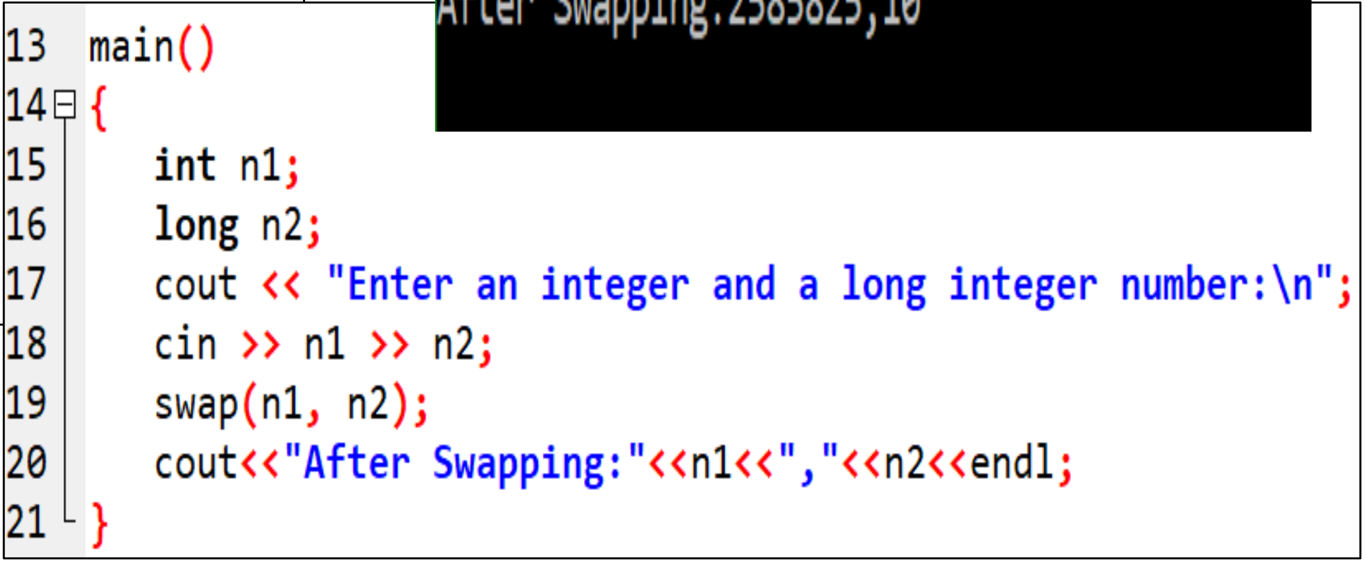
```
template <class T1, class T2>
void calculate (T1 a, T2 b)
{
  //function body
}
```

Sample program 6.7

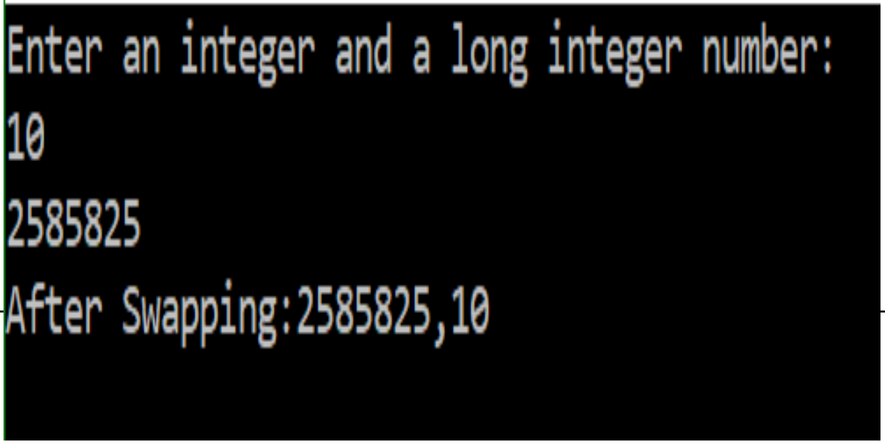
WAP to illustrate function template with multiple parameters.



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

template <class T1, class T2>
void swap(T1 &a, T2 &b)
{
    T1 temp;
    temp=a;
    a=b;
    b=temp;
}
```



```cpp
main()
{
    int n1;
    long n2;
    cout << "Enter an integer and a long integer number:\n";
    cin >> n1 >> n2;
    swap(n1, n2);
    cout<<"After Swapping:"<<n1<<","<<n2<<endl;
}
```



```
Enter an integer and a long integer number:
10
2585825
After Swapping:2585825,10
```

Overloading of function templates

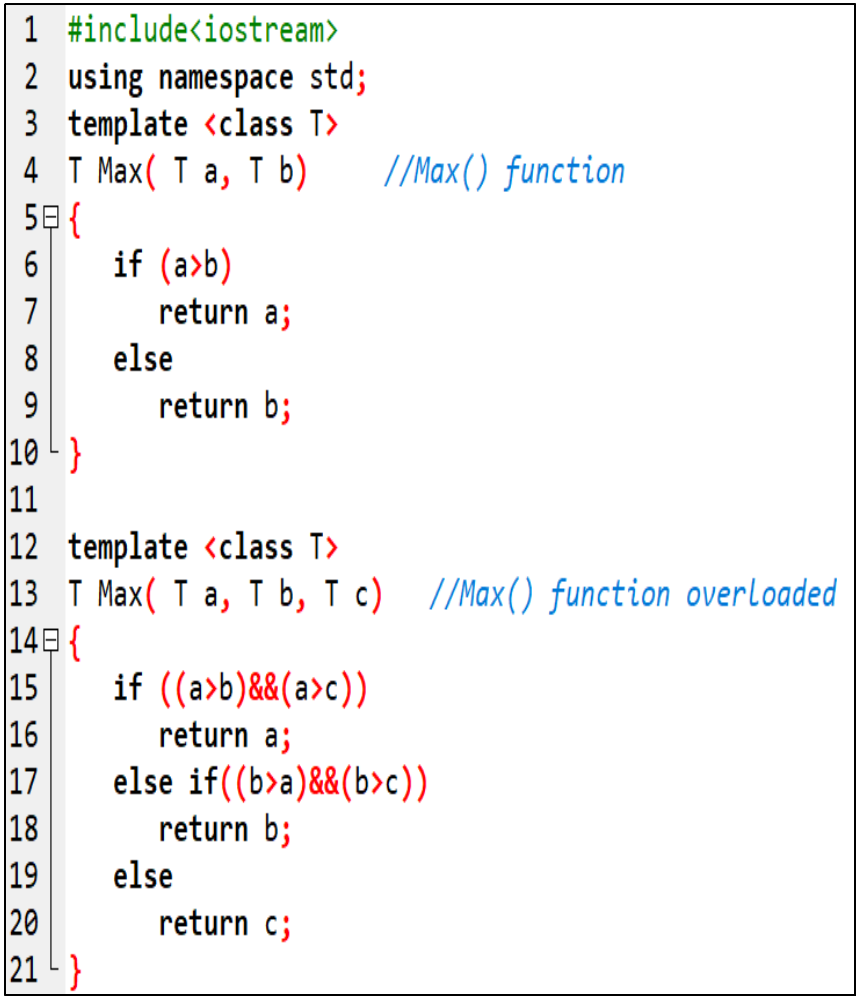
- A template function also supports the overloading mechanism like as the normal functions.
- While invoking these functions, an error occurs if no accurate match is met.
- No implicit conversion is carried out in the parameters of template functions.

Overloading of function templates (Contd.)

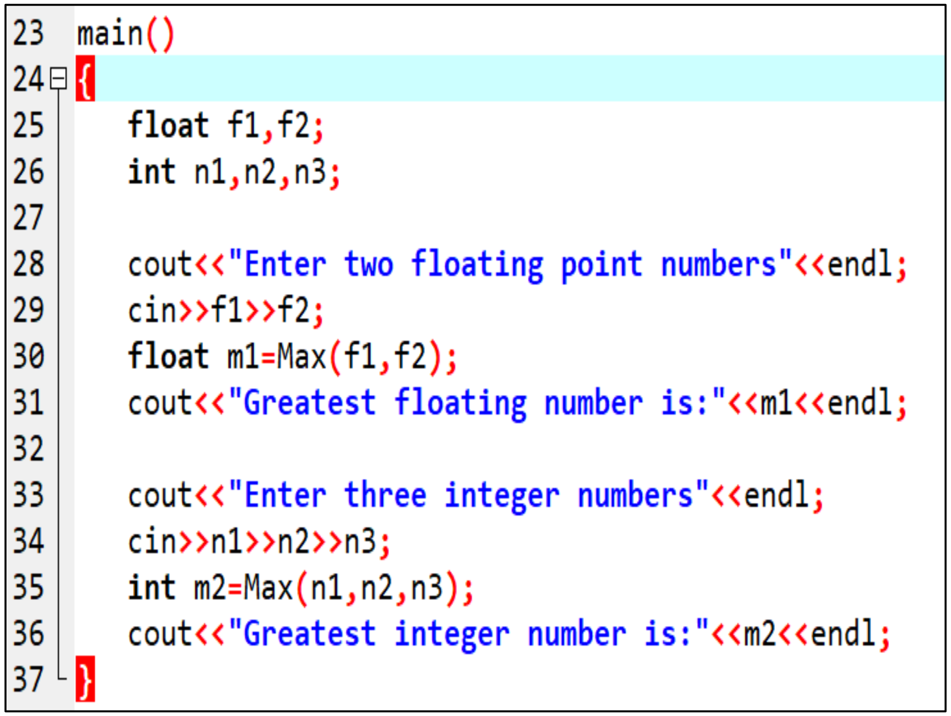
- The compiler observes the following rules for choosing an appropriate function when the programs contain overloaded functions:
  - Searches for an accurate match of functions; if found, it is invoked.
  - Searches for a template function which is a function that can be invoked with and accurate match can be generated; if found, it is invoked.
  - Attempts a normal overloading declaration for the function
  - In case no match is found, an error will be reported.

Sample program 6.8

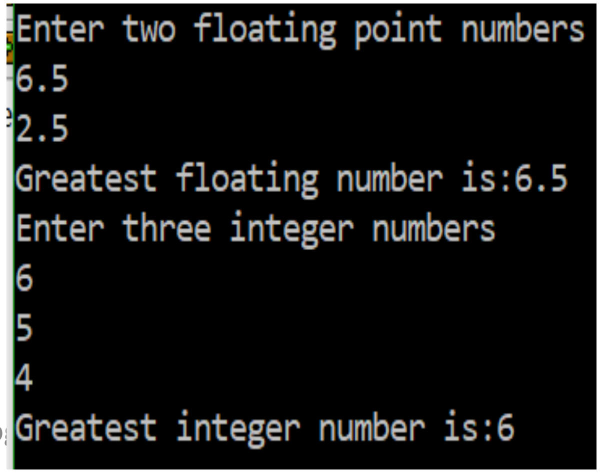
WAP to illustrate overloading of function templates.



```
#include<iostream>
using namespace std;
template <class T>
T Max( T a, T b)      //Max() function
{
    if (a>b)
        return a;
    else
        return b;
}

template <class T>
T Max( T a, T b, T c)   //Max() function overloaded
{
    if ((a>b)&&(a>c))
        return a;
    else if((b>a)&&(b>c))
        return b;
    else
        return c;
}
```



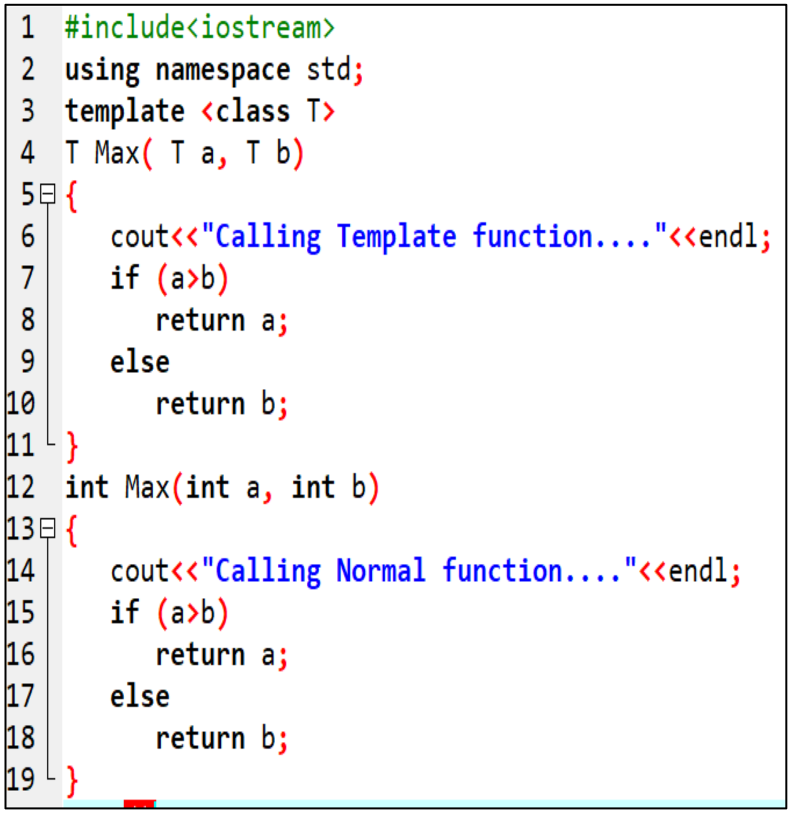
```
main()
{
    float f1,f2;
    int n1,n2,n3;

    cout<<"Enter two floating point numbers"<<endl;
    cin>>f1>>f2;
    float m1=Max(f1,f2);
    cout<<"Greatest floating number is:"<<m1<<endl;

    cout<<"Enter three integer numbers"<<endl;
    cin>>n1>>n2>>n3;
    int m2=Max(n1,n2,n3);
    cout<<"Greatest integer number is:"<<m2<<endl;
}
```



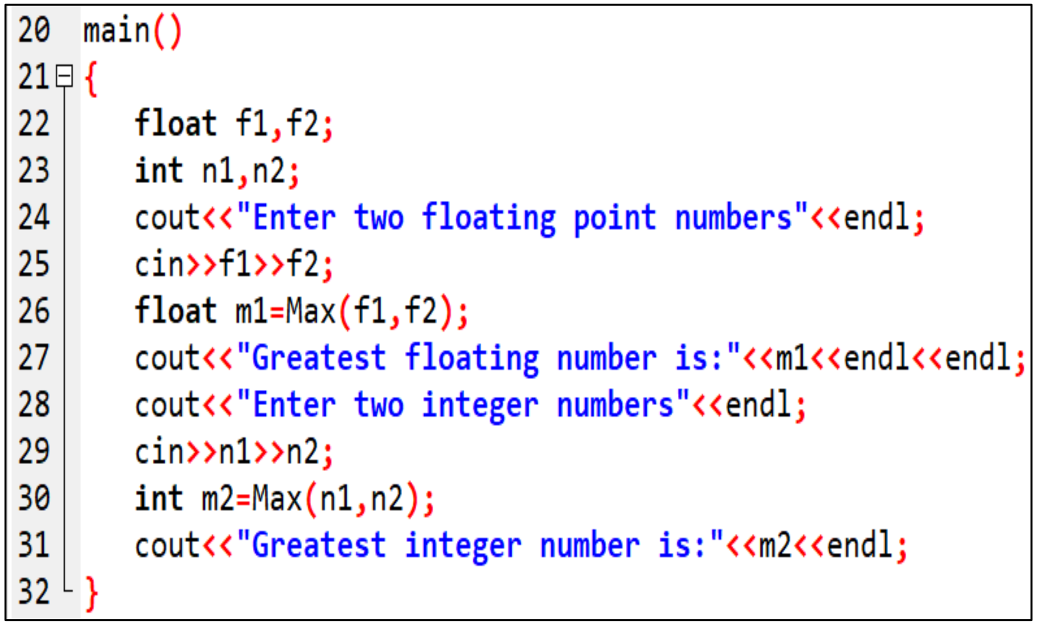
```
Enter two floating point numbers
6.5
2.5
Greatest floating number is:6.5
Enter three integer numbers
6
5
4
Greatest integer number is:6
```

Sample program 6.9

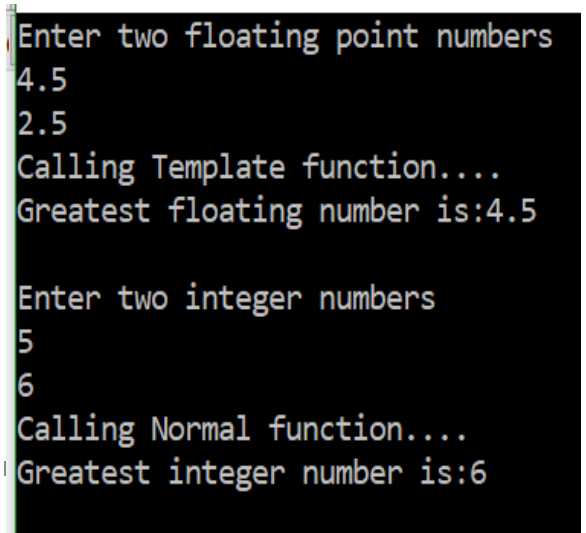
WAP to show that normal function is prioritized over function templates in case of overloading.



```
#include<iostream>
using namespace std;
template <class T>
T Max( T a, T b)
{
    cout<<"Calling Template function...."<<endl;
    if (a>b)
        return a;
    else
        return b;
}
int Max(int a, int b)
{
    cout<<"Calling Normal function...."<<endl;
    if (a>b)
        return a;
    else
        return b;
}
```



```
main()
{
    float f1,f2;
    int n1,n2;
    cout<<"Enter two floating point numbers"<<endl;
    cin>>f1>>f2;
    float m1=Max(f1,f2);
    cout<<"Greatest floating number is:"<<m1<<endl<<endl;
    cout<<"Enter two integer numbers"<<endl;
    cin>>n1>>n2;
    int m2=Max(n1,n2);
    cout<<"Greatest integer number is:"<<m2<<endl;
}
```

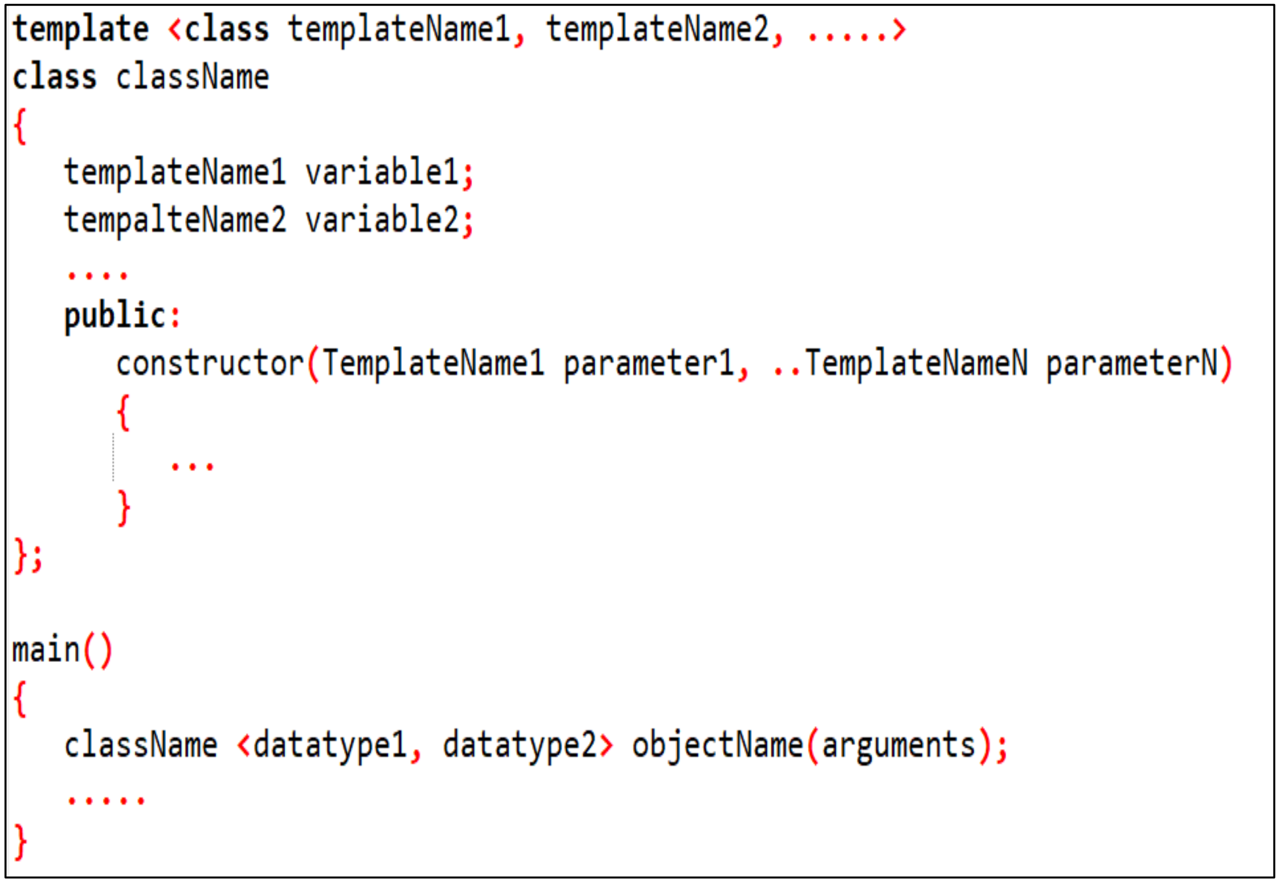


```
Enter two floating point numbers
4.5
2.5
Calling Template function....
Greatest floating number is:4.5

Enter two integer numbers
5
6
Calling Normal function....
Greatest integer number is:6
```

In the above program, the normal function *int Max(int a, int b)* is called for function call *Max(n1, n2)* instead of the Template function. This show the prioritization of normal function over template function.

6.1.5. Class Template with multi-parameters

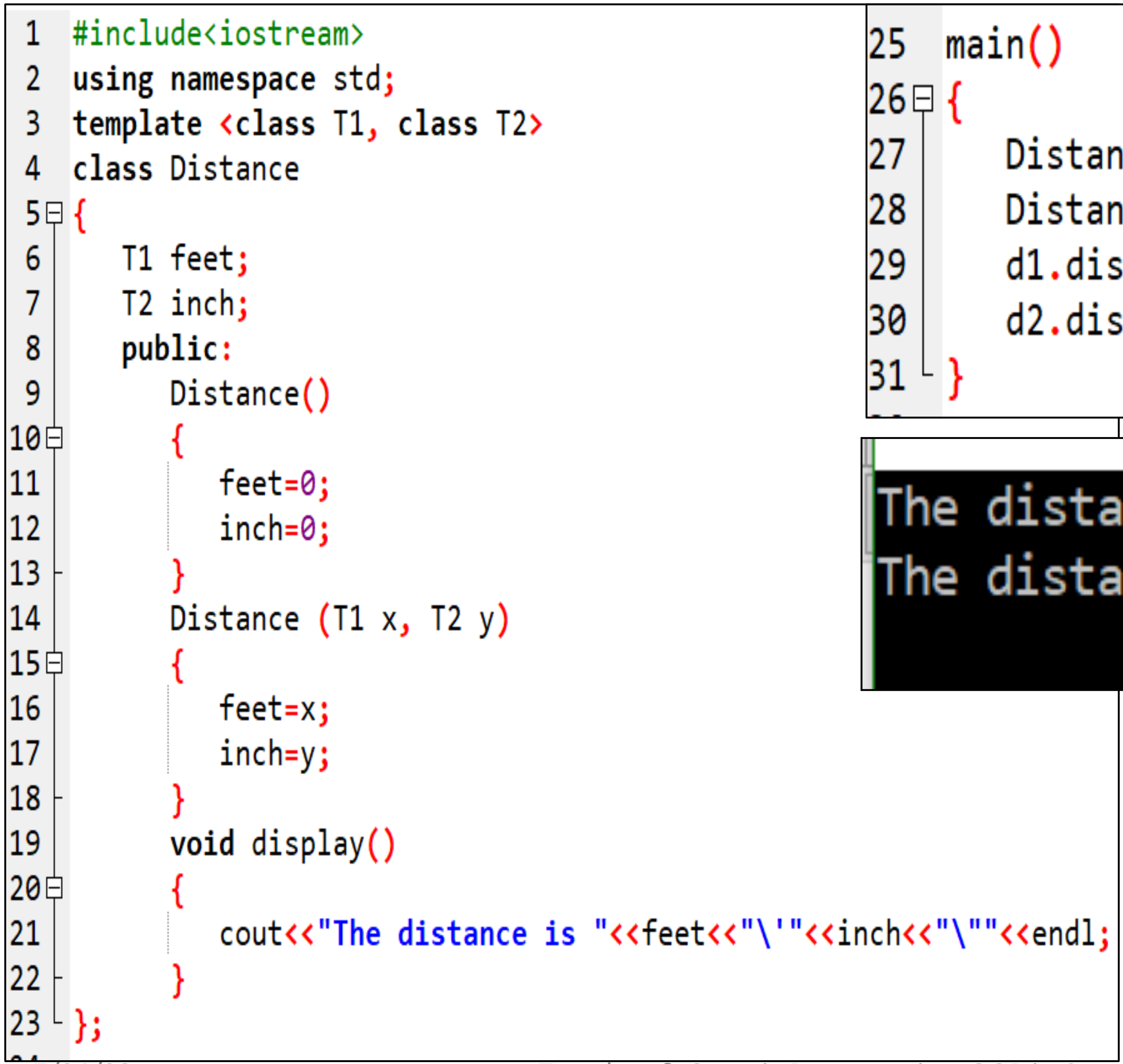
- We can define a class template with more than one generic data types. They are declared as comma separated list in the template specification.
- General format is:



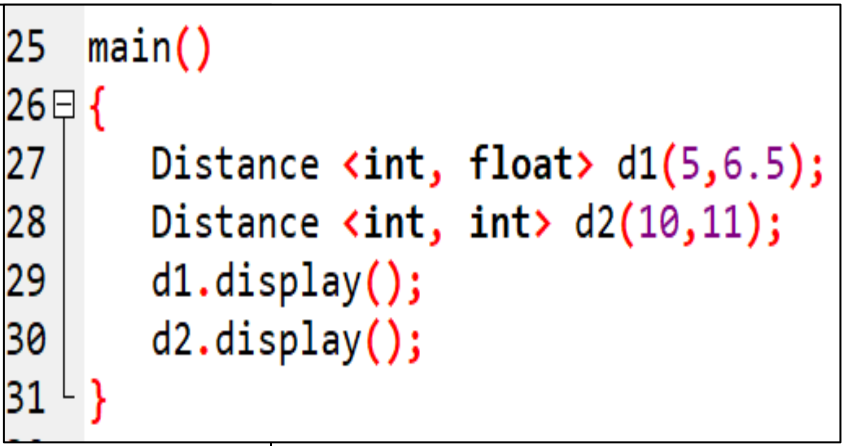
```
template <class templateName1, templateName2, .....>
class className
{
   templateName1 variable1;
   tempalteName2 variable2;
   ....
   public:
      constructor(TemplateName1 parameter1, ..TemplateNameN parameterN)
      {
          ...
      }
};

main()
{
   className <datatype1, datatype2> objectName(arguments);
   .....
}
```

Sample program 6.10

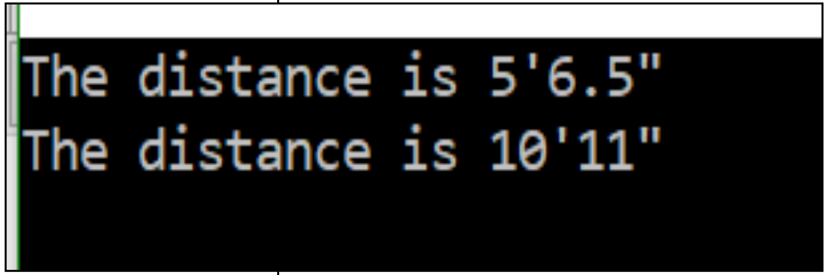
WAP to illustrate the case of multiple parameters (i.e. different datatypes) in class template.



```cpp
#include<iostream>
using namespace std;
template <class T1, class T2>
class Distance
{
    T1 feet;
    T2 inch;
    public:
        Distance()
        {
            feet=0;
            inch=0;
        }
        Distance (T1 x, T2 y)
        {
            feet=x;
            inch=y;
        }
        void display()
        {
            cout<<"The distance is "<<feet<<"\'"<<inch<<"\""<<endl;
        }
};
```



```cpp
main()
{
    Distance <int, float> d1(5,6.5);
    Distance <int, int> d2(10,11);
    d1.display();
    d2.display();
}
```



```
The distance is 5'6.5"
The distance is 10'11"
```

Standard Template Library (STL)

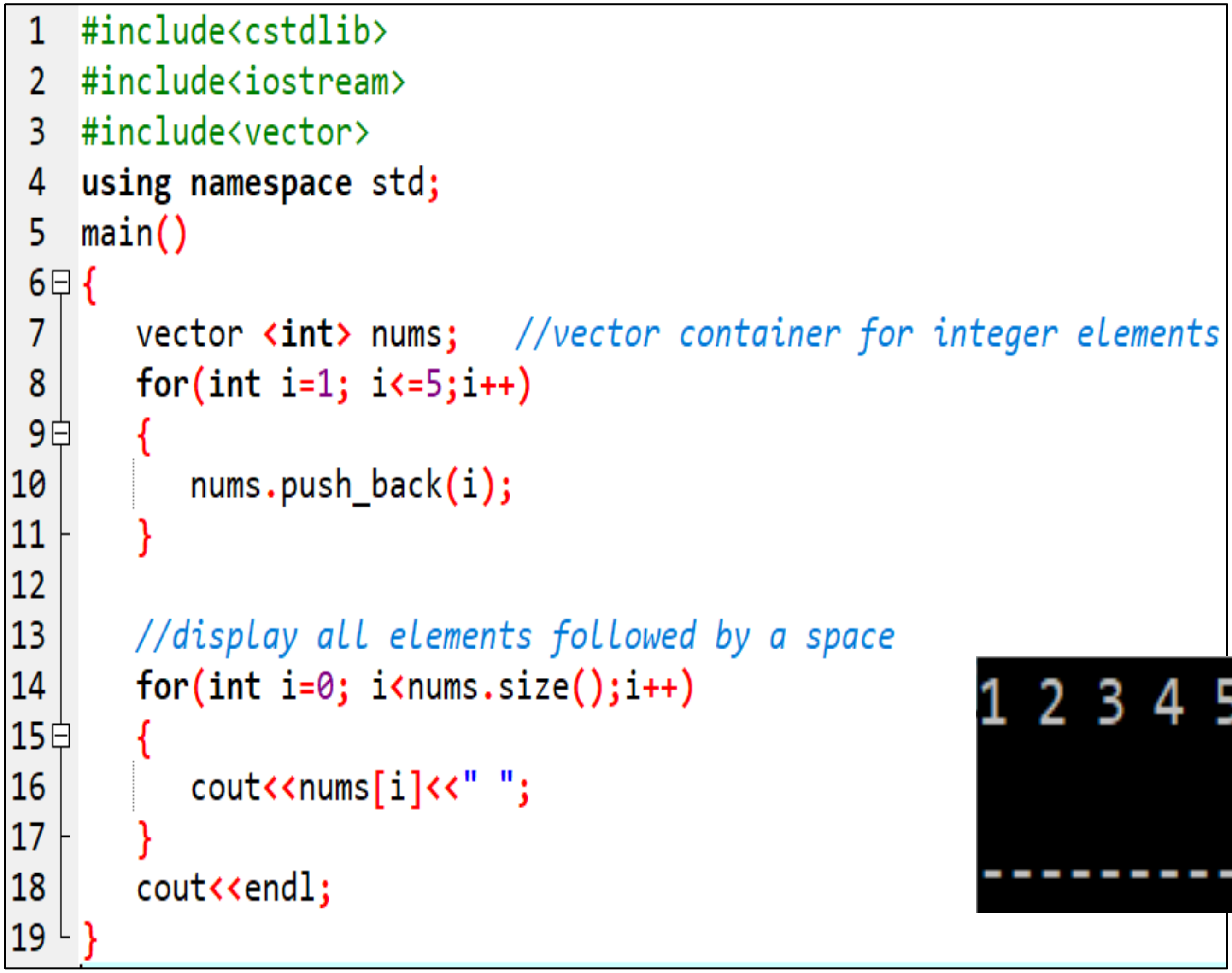
- STL is an acronym for standard template library.
- It is a set of C++ template classes that provide generic classes and function that can be used to implement data structures and algorithms.
- STL is a generic library, i.e a same container or algorithm can be operated on any data types ,
- we don’t have to define the same algorithm for different type of elements.
- For example, **sort** algorithm will sort the elements in the given range irrespective of their data type,
- we don’t have to implement different sort algorithm for different datatypes.

Use and application of STL

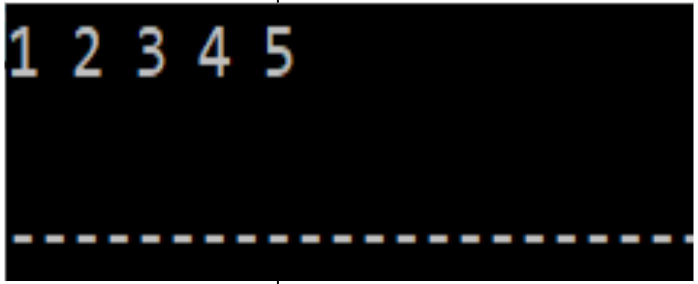
- STL saves us from defining data structures and algorithms from the scratch.
- Because of STL, now we do not have to define our *sort* function every time we make a new program or define same function twice for the different data types,
- instead we can just use the generic container and algorithms in STL.
- This saves a lot of time, code and effort during programming,
- thus STL is heavily used in the competitive programming, plus it is reliable and fast.

Sample program: 6.11

WAP to use vector template and its member functions to store integers.



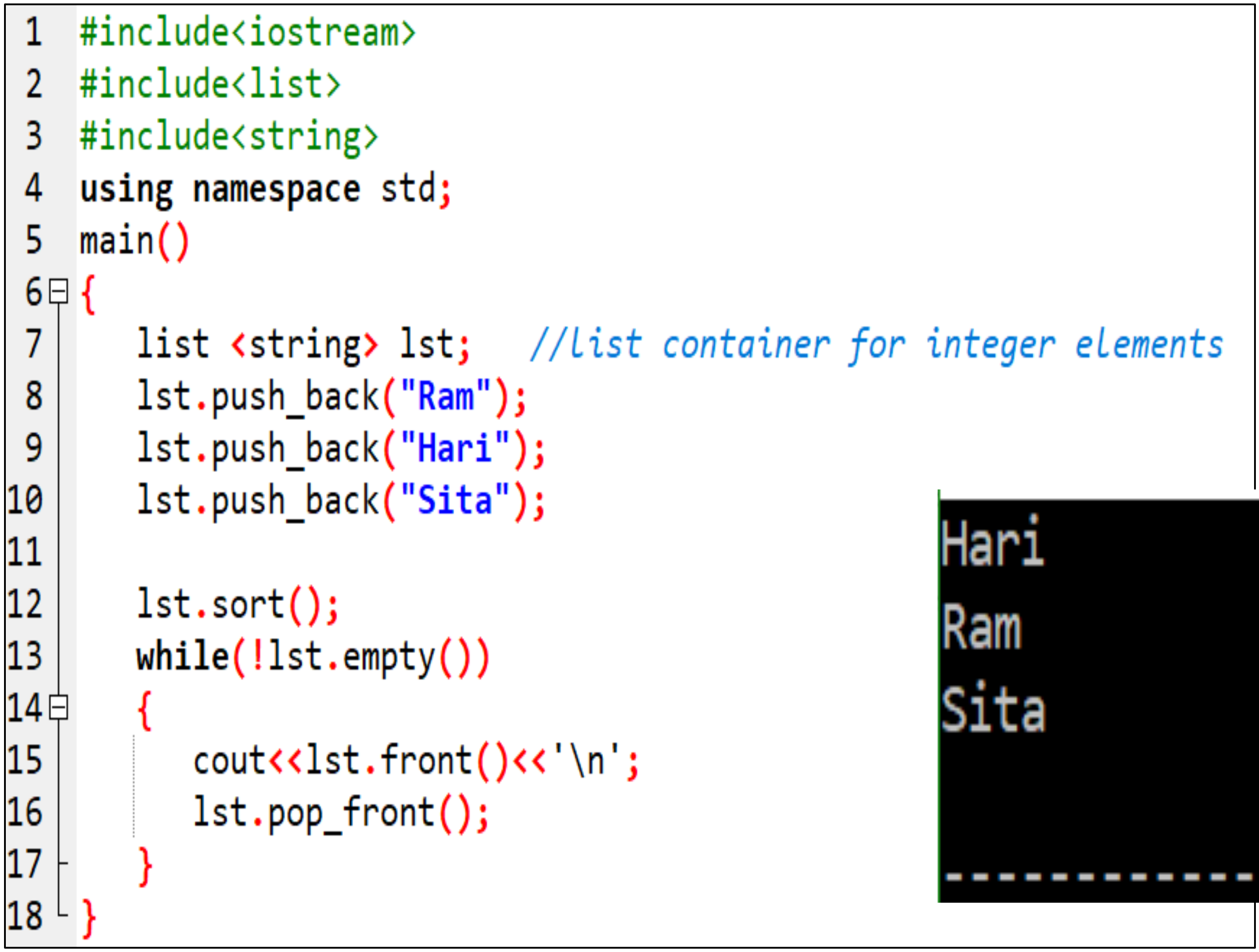
```cpp
#include<cstdlib>
#include<iostream>
#include<vector>
using namespace std;
main()
{
    vector <int> nums;    //vector container for integer elements
    for(int i=1; i<=5;i++)
    {
        nums.push_back(i);
    }

    //display all elements followed by a space
    for(int i=0; i<nums.size();i++)
    {
        cout<<nums[i]<<" ";
    }
    cout<<endl;
}
```



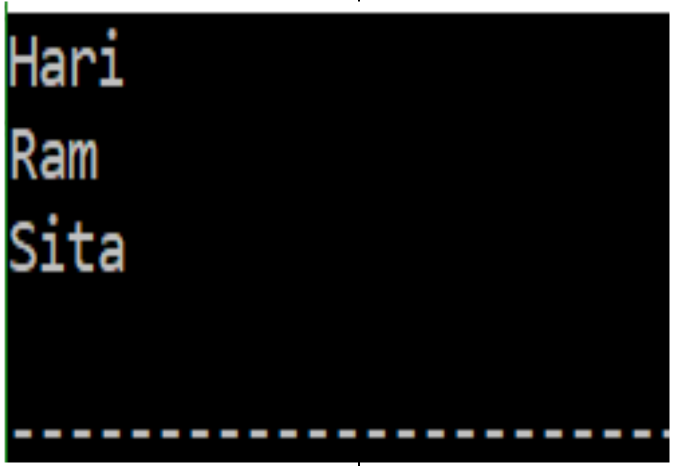
```
1 2 3 4 5

-----------------------
```

Sample program 6.12

WAP to illustrate the use of list template to insert data into list and display them in sorted order.



```cpp
#include<iostream>
#include<list>
#include<string>
using namespace std;
main()
{
    list <string> lst;   //list container for integer elements
    lst.push_back("Ram");
    lst.push_back("Hari");
    lst.push_back("Sita");

    lst.sort();
    while(!lst.empty())
    {
        cout<<lst.front()<<'\n';
        lst.pop_front();
    }
}
```



```
Hari
Ram
Sita

------------------------
```

Components of STL

1. Containers
2. Algorithms
3. Iterators

Components of STL

1. Containers

- Containers or container classes store objects and data.
- In STL, we have various types of container classes like Array, vector, queue, dequeue, list, map, set, etc.
- These containers are generic, they can hold elements of any data types.
- For example: **vector** can be used for creating dynamic arrays of char, integer, float and other types.

Components of STL (Contd.)

2. Algorithms

- STL provide number of algorithms that can be used of any container, irrespective of their type.
- Algorithms library contains built in functions that performs complex algorithms on the data structures.
- For example: one can reverse a range with *reverse()* function, sort a range with *sort()* function, search in a range with *binary_search()* and so on.
- Algorithm library provides abstraction, i.e we don't necessarily need to know how the algorithm works.

Components of STL (Contd.)

3. Iterators

- Iterators in STL are used to point to the containers.
- Iterators actually acts as a bridge between containers and algorithms.
- For example: *sort()* algorithm have two parameters, starting iterator and ending iterator,
- now *sort()* compare the elements pointed by each of these iterators and arrange them in sorted order,
- thus it does not matter what is the type of the container and same sort() can be used on different types of containers.

Classification of Container Class

S. No.	Container class	Details
1.	Sequence	• Containers the can be accessed in a sequential or linear manner are said to be sequential containers. • Arrays, Vectors, Lists, Deques are the STL containers that store data linearly and can be accessed in a sequential manner.
2.	Container adaptors	• Container adopters are sequential containers, • however, they are implemented by providing a different interface. • he underlying container is encapsulated in such a way that its elements are accessed by the members of the container adaptor independently of the underlying container class used. • Thus containers like a stack, queue and priority-queue are all classified as container adopters.

Classification of Container Class (Contd.)

S. No.	Container class	Details
3.	Associative	• Associative containers are containers that implement sorted data structures. • These containers are fast to search. • These containers are usually implemented in a key/value pair fashion. • Some of the examples of associative containers are Map, Set, MultiMap, Multiset, etc.
4.	Unordered associative	• Used to provide different interface to the sequence containers. • Examples: Unordered Set, Unordered Multiset, Unordered Map, Unordered Multimap

End of chapter 5