Standard Libraries

Standard Libraries

The C++ Standard Library provides several generic containers and functions to use and manipulate these containers.

Standard Libraries

- **The Standard Function Library:** It consists of general-purpose, stand-alone functions that are not part of any class.
- **The Object Oriented Class Library:** This is a collection of classes and associated functions.

Standard Libraries

- **The Standard Function Library:** It consists of general-purpose, stand-alone functions that are not part of any class.
- **The Object Oriented Class Library:** This is a collection of classes and associated functions.

Standard Function Library:

- I/O and string/character Haindling
- Mathematical,
- Time, date, and localization,
- Dynamic allocation
- Wide-character functions

Standard Libraries

- **The Standard Function Library:** It consists of general-purpose, stand-alone functions that are not part of any class.
- **The Object Oriented Class Library:** This is a collection of classes and associated functions.

Standard Function Library:

- I/O and string/character Haindling
- Mathematical,
- Time, date, and localization,
- Dynamic allocation
- Wide-character functions

Object Oriented Class Library:

- The Standard C++ I/O Classes
- The String Class
- The Numeric Classes
- The STL Container Classes
- The STL Algorithms
- The STL Function Objects
- The STL Iterators
- The STL Allocators
- The Localization library
- Exception Handling Classes
- Miscellaneous Support Library

Standard Libraries

- **The Standard Function Library:** It consists of general-purpose, stand-alone functions that are not part of any class.
- **The Object Oriented Class Library:** This is a collection of classes and associated functions.

Standard Function Library:

- I/O and string/character Haindling
- Mathematical,
- Time, date, and localization,
- Dynamic allocation
- Wide-character functions

Object Oriented Class Library:

- The Standard C++ I/O Classes
- The String Class
- The Numeric Classes
- The STL Container Classes
- The STL Algorithms
- The STL Function Objects
- The STL Iterators
- The STL Allocators
- The Localization library
- Exception Handling Classes
- Miscellaneous Support Library

Two types of template Libraries

- **User Defined Templates**
- **Standard Template Library(STL)**

Templates

Templates

- It enable us to define generic classes and functions, that provides support for Generic Programming
- Generics is the idea to allow type (Integer, String, … etc and user-defined types) to be a parameter to methods and classes.
- Template can be used to create a family of classes or functions
- **Example-** A class template for an array class would enable us to create arrays of various data types such as int array, float array etc. similarly, we can define a template for function say mult(), that would helps us to create various versions of mult()for multiplying int, float and double type values.

Templates

- Two Types of templates
  - Class Templates
  - Function Templates

Class Templates

```
class stack
{
   private:
        int arr[5];
   public:
        push()
        pop()
};
```

```
class stack
{
   private:
        char arr[5];
   public:
        push()
        pop()
};
```

Class Templates

```
class stack
{
   private:
        int arr[5];
   public:
        push()
        pop()
};
```

```
class stack
{
   private:
        char arr[5];
   public:
        push()
        pop()
};
```

Class Templates

```
class stack
{
   private:
        int arr[5];
   public:
        push()
        pop()
};
```

```
class stack
{
   private:
        char arr[5];
   public:
        push()
        pop()
};
```

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

template <class T>
class stack
{
    private:
        T arr[5],a;
    public:
        void pop();
        void push(T x);
};
```

Class Templates

```
class stack
{
   private:
        int arr[5];
   public:
        push()
        pop()
};
```

```
class stack
{
   private:
        char arr[5];
   public:
        push()
        pop()
};
```

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

template <class T>
class stack
{
    private:
        T arr[5],a;
    public:
        void pop();
        void push(T x);
};
```

```
int main()
{
    stack <int> s1;
    stack <float> s2;
    s1.push(10);
    s1.pop();
    s2.push(10.6);
    s2.pop();
}
```

Class Templates

Example: Write a class for addition of two numbers using class templates

Class Templates

Example: Write a class for addition of two numbers using class templates

```
template <class T>
class add
{   private:
        T a,b,sum;
    public:
        void getdata(T x,T y)
        {
            a=x; b=y;
        }
        void putdata()
        {   sum=a+b;
            cout<<"sum= "<<sum;
        }
};
```

```
int main()
{
    add <int> obj1;
    add <float> obj2;
    obj1.getdata(10,20);
    obj1.putdata();

    obj2.getdata(10.5,20.5);
    obj2.putdata();

}
```

Class Templates

Example: Write a class for addition of two numbers using class templates

```
template <class T>
class add
{    private:
         T a,b,sum;
     public:
         void getdata(T x,T y);
         void putdata();
};
```

```
int main()
{    add <int> obj1;
     add <float> obj2;
     obj1.getdata(10,20);
     obj1.putdata();
     obj2.getdata(10.5,20.5);
     obj2.putdata();
}
```

```
template<class T>
void add<T>::getdata(T x,T y)
{
     a=x; b=y;
}
```

```
template<class T>
void add<T>::putdata()
{    sum=a+b;
     cout<<"sum= "<<sum;
}
```

Class Templates with multiple parameters

Example: Write a class template with multiple parameters to find greater number among two given numbers

```
template <class T1,class T2>
class number
{       T1 x;
        T2 y;
    public:
        void getdata(T1 a, T2 b)
        {  x=a;  y=b;  }
        void cmp()
        {   if(x>y)
                cout<<x<<" is greater than "<<y;
            else
                cout<<y<<" is greater than "<<x;
        }
};
```

```
int main()
{
    number <int, float> n;
    n.getdata(3,2.6);
    n.cmp();
    return(0);
}
```

Function Templates

Function Templates

Function Overloading

```
int add(int x, int t){ }
float add(float x, float y {  }
double add(double x,double y) { }

int main()
{
   add(10,20);
   add(10.3f, 20.5f);
   add(5.3232, 23423.345);
}
```

Function Template:

```
template<typename T>
T add(T x, T y)
{ }


int main()
{
   add<int>(3,7);
   add<float>(3.3,3.7);
   add(double(3.55,7.66);
}
```

Function Templates

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

template<typename T>

T add(T x, T y)
{
    return(x+y);
}

int main()
{
    cout<<"Addition is:"<<add<int>(2,3);
    return(0);
}
```