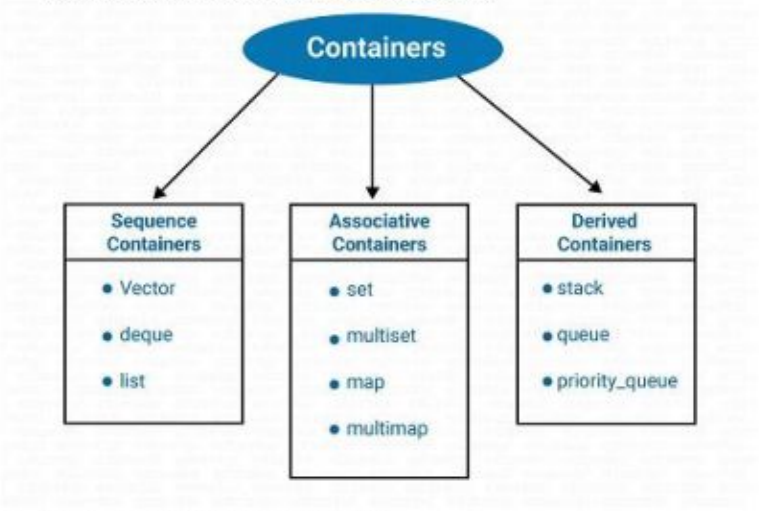
STL

- The **Standard Template Library** (STL) is a set of C++ template classes to provide common programming data structures and functions such as **lists, stacks, arrays, etc.**
- It is a library of **container classes, algorithms and iterators**.
- Its components are parameterized. A working knowledge of template classes is a prerequisite for working with STL.



- **STL has four components**
  - Containers
  - Algorithms
  - Functions
  - Iterators

- **Containers**
  - Container classes store objects and data.
  - There are in total seven standard ***"core-class"*** container classes and three container **adaptor classes** and only seven header files that provide access to these containers or container adaptors.
  - **Sequence Containers**
    - Implement data structures which can be accessed in a sequential manner.
      - Vector
      - List
      - Deque
      - Arrays

  - **Container Adaptors**
    - Provide a different interface for sequential containers.
      - Queue
      - priority_queue
      - Stack
  - **Associative Containers**
    - Limplement sorted data structures that can be quickly searched (O(log n) complexity).

- Set
- Multiset
- Map
- multimap

- **Algorithms**
  - The header algorithm defines a collection of functions especially designed to be used on elements.
  - **They act on containers** and provide means for various operations for the contents of the containers.
    - **Algorithm**
      - Sorting
      - Searching
      - Important STL Algorithms
      - Useful Array algorithms
    - Partition Operations
    - Numeric

- **Functions**
  - The STL includes classes that overload the function call operator.
  - Instances of such classes are called function objects .
  - function objects allow the working of the associated function to be customized with the help of parameters to be passed.
- Iterators
  - As the name suggests, iterators are used for working upon a sequence of values.

Vector

- A **dynamic array** and store elements in contiguous memory locations.
- Allow **insertion and deletion.**
- Permit **direct access**
- Header file **<vector>**
- **constructor**
  - vector <int> v; // zero length vector
  - vector <int> v(size); //initialize the length
- **Methods**

(1) at() gives the reference of an element
(2) back(): give the reference of the last element
(3) begin(): give the reference of the first element
(4) capacity(): give the current capacity of the vector.
(5) clear() : delete all the elements from the vector
(6) empty():determine if vector is empty or not
(7) end(): give the reference of the last element
(8) erase():delete specified element
(9) insert(): insert elelement into the vector
(10) pop_back():delete the last element
(11) push_back(): add the element at the end
(12) resize() : modify the size of the element of the rector to the specified values.
(13) size() : gives the number of element

(14) swap() : exchanges elements in the specified two vector
e.g.

```
#include<iostream>
#include<vector>
using namespace std;
main()
{
        vector <int> a;
        int x ;

        for ( int i = 0; i < 5 ; i++)
        {
                cin >> x;
                a.push_back(x);
        }
        cout << endl << a.size();

        for ( int i = 0; i < 5 ; i++)
        {
                cout <<endl<< a[i]; // a.operator[](i)
        }

        cout << endl ;
        vector<int> :: iterator itr = a.begin();
        itr = itr + 3 ; // 4th position
        a.insert(itr,1);
        cout << a.size();

        for ( int i = 0; i < a.size() ; i++)
        {
                cout << endl<< a[i];
        }
        cout << endl ;

        a.erase(itr, itr+2);
        for ( int i = 0; i < a.size() ; i++)
        {

                cout << endl<< a[i];
        }

}
```


- **List**
  - The list container supports a bidirectional, linear and provides an efficient implementation for deletion and is insertion operation.
  - unlike a vector which support a random access.
  - **include <list>**
  - **methods**
    (1) back(): give the reference of the last element
    (2) begin(): give the reference of the first element

(3) clear() : delete all the elements from the list
(4) empty():determine if list is empty or not
(5) end(): give the reference of the last element
(6) erase():delete specified element
(7) insert(): insert elelement into the list
(8) merge(): merget two ordered list
(9) pop_back():delete the last element
(10) pop_front():delete the first element
(11) push_back(): add the element at the end
(12) push_front(): add the element at the first
(13) resize() : modify the size of the element of the rector to the specified values
(14) size() : gives the number of element
(15) sort(): to sort the lsit
(16) slice(): insert a list into the invoking list
(17) swap() : exchanges eleemtns in the specified two list
(18) unique(): delete the duplicate element

e.g.

```
#include<iostream>
#include<list>
#include<cstdlib>
using namespace std;
void display(list<int> &l)
{
        list<int> :: iterator itr ;
        for (itr=l.begin() ; itr!= l.end(); ++itr)
        {
                cout << *itr << ",";
        }
}

main()
{
        list <int> list1;
        list <int> list2(5);

        for ( int i = 0; i < 5 ; i++)
        {
                list1.push_back(rand()%100);
        }
        cout << endl;


        cout << endl ;
        list<int> :: iterator itr ;

        for (itr=list2.begin() ; itr!= list2.end(); ++itr)
        {
                *itr = rand()%100;
```

```
}

cout << "\nlist 1";
display(list1);
cout <<"\nlist 2";
display(list2);

list1.sort();
list2.sort();
list1.merge(list2);
cout << endl ;
display(list1);
list1.reverse();
display(list1);
}
```

- **Map**
  - A map is sequence of (**key,value**) pair where value is associated with each unique key.
  - **Retrieval of values is based on the key.**
  - **it also known as associative array**

Methods

(1) begin(): give the reference of the first element
(2) clear() : delete all the elemenets from the map
(3) empty():determine if map is empty or not
(4) end(): give the reference of the last element
(5) erase():delete specified element
(6) insert(): insert elelement into the map
(7) merge(): merget two ordered list
(8) size() : gives the number of element
(9) sort(): to sort the map \
(10) swap() : exchanges eleemtns in the specified two map
(11) find (): gives the location of the specified location

e.g.

```
#include<iostream>
#include<map>
#include<string>
using namespace std;
main()
{
        string name;
        int number;
        map <string,int> phone;
        cout << "\nEnter three sts of name and number";
        for ( int i = 0 ; i< 3 ; i++)
        {
                cin >> name ;
                cin >> number;
                phone[name] = number;
        }
        phone["taregt"]= 9981315087;
        phone.insert(pair<string,int>("boss",6666));
```

```
        cout << "list of phone number";
        map<string,int>::iterator p ;
        for ( p = phone.begin(); p!= phone.end(); p++)
        {
                cout << (*p).first << "  " << (*p).second ;
        }

        number = phone["boss"];
        cout << endl << number;


}
```