* containers derived from sequence are the containers.

Ex: stack, queue, priority queue.

Vector:

* vectors are dynamic arrays which have the ability to resize itself dynamically when an element is inserted or deleted.
* The storage of vector is handled automatically by the container class.
* The general form of vector declaration is

```
vector <object_type> v1;
```

Ex:
```
vector <int> a;
vector <char> c;
vector <float> f;
vector <int> a = {1,2,3,4,5};
```

* some of the member functions of vectors are:

1. at(): Returns the element at the specified index
2. back(): Returns the reference to the last element.
3. begin(): Returns an iterator pointing to the first element of the vector.
4. capacity(): Returns the maximum number of elements that are allowed.

5. clear(): Deletes all the elements from the vector and assign an empty vector.
6. empty(): Returns a boolean value, true if the vector is empty and false if the vector is not empty.
7. end(): Returns an iterator pointing to the end of the vector.
8. erase(): Deletes a single element or a range of elements.
9. front(): Returns the reference to the first element.
10. insert(): Inserts new elements into the vector at a particular position.
11. pop_back(): Removes the last element from the vector.
12. push_back(): Inserts a new element at the end of the vector.
13. resize(): Resizes the vector to the new length which can be less than or greater than the current length.
14. size(): Returns the number of elements in the vector.

C++ program to implement vector and its functions:-

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

```
int main()
{
int i;
vector <int> v1;
vector <int>:: iterator itr;
cout<<"\n vector is empty:" << v1.empty();
for(i=0;i<5;i++)
v1.push_back(i+1);
cout <<"\n vector elements are:";
for(itr = v1.begin(); itr!=v1.end();itr++)
cout << *itr <<" ";
cout <<"\n vector elements are:";
for(i=0;i<5;i++)
cout << v1[i] <<" ";
itr = v1.begin();
cout <<"\n first element :" << *itr;
v1.push_back(6);
v1.pop_back();
cout <<"\n first element :" << v1.front();
cout <<" \n last element :" << v1.back();
itr = v1.end()-1;
cout <<"\n last element: " << *itr;
cout<<"\n Element at index 4:" << v1.at(4);
cout <<"\n capacity :" << v1.capacity();
cout <<"\n size :" << v1.size();
cout <<"\n vector is empty:" << v1.empty();
cout <<"\n vector is empty:";
return 0;
}
```

Output:

```
Vector is empty: 1
Vector elements are : 1 2 3 4 5
Vector elements are : 1 2 3 4 5
first element : 1
first element : 1
last element : 5
last element : 5
Element at index 4: 5
capacity: 8
size: 5
vector is empty: 0
```

Program:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int i;
    vector<int> v1;
    vector<int>::iterator itr;
    for(i=0; i<5; i++)
    v1.push_back(i+1);
    cout<<"\n vector elements are:";
    for(itr = v1.begin(); itr!=v1.end(); itr++)
    cout<<*itr<<" ";
    itr = v1.begin();
    cout<<"\n first element:"<<*itr;
```

```
vl.push_back(6);
vl.pop_back();
cout<<"\n first element:"<<vl.front();
cout<<"\n Last element:"<<vl.back();
itr = vl.end()-1;
cout<<"\n last element:"<<*itr;
cout<<"\n Element at index 4:"<<vl.at(4);
cout<<"\n capacity:"<<vl.capacity();
cout<<"\n size:"<<vl.size();
cout<<"\n vector is empty:"<<vl.empty();
vl.insert(vl.begin(),0);
vl.insert(vl.end(),6);
vl.erase(vl.begin());
vl.erase(vl.begin()+1,vl.end()-1);
cout<<"\n vector elements are:";
for(itr=vl.begin(); itr!=vl.end(); itr++)
cout<<*itr<<" ";
vl.clear()
cout<<"\n vector elements are:";
for(itr=vl.begin(); itr!=vl.end(); itr++)
cout<<*itr<<" ";
return 0;
}
```

Output:-

```
Vector elements are: 1 2 3 4 5
first element: 1
first element: 1
last element : 5
last element : 5
```

Element at index 4: 5
capacity: 8
size: 5
vector is empty: 0
Vector elements are : 1 6
Vector elements are :

Lists:-

* Lists are sequence containers that allow non-contiguous memory locations for elements.
* List containers are implemented as doubly linked list so it is possible to add and remove elements from both ends of the list.
* Lists are efficient to insert new elements and to sort and merge lists.
* Some of the member fonctions of list are :

1. push_back(): It adds a new element at the end of the list
2. push_front(): It adds a new element to the front.
3. pop_back(): It deletes the last element
4. pop_front(): It deletes the first element
5. empty(): It checks whether the list is empty or not
6. size(): It finds the number of elements present in the list.
7. max_size(): It finds the maximum size of the list.

8. front(): It returns the first element of the list
9. back(): It returns the last element of the list
10. swap(): It swaps two lists
11. reverse(): It reverses the elements of the list
12. sort(): It sorts the elements of the list in an increasing order.
13. merge(): It merges the two sorted lists.
14. begin(): Returns an iterator pointing to the first element of the list.
15. end(): Returns an iterator pointing to the end of the list.
16. clear(): Deletes all the elements from the list and assign an empty list.
17. erase(): Deletes a single element or a range of elements.

Program:-

```
#include <iostream>
#include <list>
using namespace std;
int main()
{ list <int> L1, L2;
  list <int> :: iterator itr;
  L1. push_back(5);
  L1. push_back(10);
  L1. push_back(15);
  L1. push_back(20);
  L2. push_back(70);
```

```
l2.push_back(80);
l2.push_back(90);
l2.push_back(100);
cout<<"\n List elements are: ";
for(itr=l1.begin(); itr!=l1.end(); itr++)
{
   cout<<*itr<<" ";
}
l1.push_front(25);
l1.pop_front();
l1.push_back(30);
l1.pop_back();
cout<<"\n Elements in reverse order: ";
l1.reverse();
for(itr=l1.begin(); itr!=l1.end(); ++itr)
{
   cout<<*itr<<" ";
}
```

I
```
cout<<"\n List1 elements after swapping: ";
l1.swap(l2);
for(itr=l1.begin(); itr!=l1.end(); ++itr)
{
   cout<<*itr<<" ";
}
```

II
```
cout<<"\n Elements after merging: ";
l1.merge(l2);
for(itr=l1.begin(); itr!=l1.end(); ++itr)
{
  cout<<*itr<<" ";
}
```

return

}

output:

I List el
Eleme
List1

II List
eleme
Eleme

Maps

* Maps
elemen
* Each
mappe
have
* The
can
*. Elem
by. th
direct
* Som
1. at()
2. beg
elemen
3. end
map
4. size

return 0;
}

output:

I List elements are : 5 10 15 20
Elements in reverse order : 20 15 10 5
List1 elements after swapping : 70 80 90 100
II List elements are : 5 10 15 20
elements in reverse order : 20 15 10 5
Elements after merging : 20 15 10 5 70 80 90 100

Maps :-

* Maps are associative containers that store elements as sequence of (key, value) pairs.
* Each element has a key value and a mapped value. No two mapped values can have same key values.
* The datatype of key value and mapped value can be different.
* Elements in map are always in sorted order by their corresponding key and can be accessed directly by their key.
* some basic functions associated with the map:

1. at(): returns value at the key.
2. begin(): returns an iterator to the first element in the map
3. end(): returns an iterator to the end of the map
4. size(): Returns the no. of elements in the map.

5. max_size(): Returns the maximum number of elements that the map can hold.

6. empty(): Returns whether the map is empty

7. insert(key, value): Adds a new element to the map.

8. clear() - Removes all the elements from the map

9. erase(): Removes the element at the position pointed by the iterator

10. swap(): Exchange the contents of the maps.

Program:

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
  map<int,string> student;
  map<int,string>:: iterator, itr;
  student.insert(pair<int,string>(1201," bhavani"));
  student.insert(pair<int,string>(1202," kavya"));
  student.insert(pair<int,string>(1203," satya"));
  student.insert(pair<int,string>(1205," sunita"));
  student.insert(pair<int,string>(1204," renuka"));
  cout<<"\n Map Elements are: \n";
  cout<<"\n RNO \t NAME \n";
  for(itr=student.begin(); itr!=student.end(); itr
  cout<<itr->first<<"\t"<<itr->second<<endl;
  cout<<"\n Map size is:"<<student.size();
```

```
cout<<"\n
itr=student
cout<<"\n
itr->seco
itr=stude
itr--;
cout<<"\n
itr->seco
return 0;
}
```

output:

```
Map Elem
RNO
1201
1202
1203
1204
1205
Map size
student
student
student
```

```
cout << "\n student with rno 1203 :" << student.at(1203);
itr = student.begin();
cout << "\n student with rno:" << itr -> first << "is" <<
itr -> second;
itr = student.end();
itr--;
cout << "\n student with rno:" << itr -> first << "is" <<
itr -> second;
return 0;
}
```

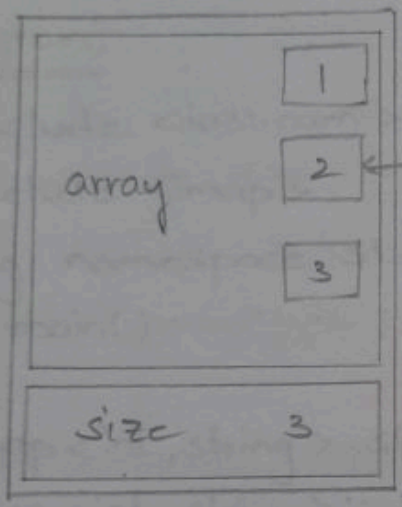
output:

```
Map Elements are :
RNO        NAME
1201       bhavani
1202       kavya
1203       satya
1204       renuka
1205       sunita

Map size is : 3
student with rno 1203 : satya
student with rno: 1201 is bhavani
student with rno: 1205 is sunita
```

Iterators :-

* Iterators are pointer-like entities that are used to access individual elements in a container.
* Iterators are moved sequentially from one element to another element. This process is known as iterating through a container.
* General form of iterator for a container:
container < object_type > :: iterator itr1;



Algorithms :-

* STL provide different types of algorithms that can be implemented upon any of the container with the help of iterators.
* Although each container provides its own basic operations, the standard algorithms provide more extended or complex actions.
* Algorithm functions can be used on any type of container.
* They allow us to work with two different types of containers at the same time.

Non-modifyin
do not cha
in the co
Modifying
designed t
a contain
Removal a
remove
Mutating
modifying
specifically
sorting a
modifying
efficient
sorted r
are spe
function
accordin
This all
Numeric
designed
Non-mo
countl)
min_el
max_e
search

Non-modifying algorithms:- Non modifying algorithms do not change the value/order of any element in the container.

Modifying algorithms:- Modifying algorithms are designed to alter the value of elements within a container.

Removal algorithms:- These are designed to remove elements in a container.

Mutating algorithms:- mutating algorithms are modifying algorithms, but they are designed specifically to modify the order of elements.

Sorting algorithms:- sorting algorithms are modifying algorithms specifically designed for efficient sorting of elements in a container.

Sorted range algorithms:- Sorted range algorithms are special sorting algorithms designed to function on a container which is already sorted according to a particular sorting criterion. This allows for greater efficiency.

Numeric algorithms:- Numeric algorithms are designed to work on numerical data.

Non-modifying algorithms:-

count()
min_element()
max_element()
search()

Modifying algorithms:-

copy()
merge()
fill()
replace()

Removal algorithms:-

remove()

Mutating algorithms:-

reverse()
rotate()

Numeric algorithms:-

iota()
accumulate()
partial_sum()

sorting algorithms:-

sort()
stable_sort()
make_heap()
push_heap()
pop_heap()
sort_heap()

sorted Range algorithms:-

binary_search()
lower_bound()
upper_bound()

program:-

```
#include <iostream>
#include <vector>
#include <algorithms>
#include, <numeric>
using namespace std;
int main()
{
   vector <int>. v1;
   vector <int> :: iterator itr;
   v1. push_back(10);
   v1. push_back(40);
   v1. push_back(20);
   v1. push_back(30);
   cout <<"\n vector elements are:";
   for(itr = v1. begin(); itr! = v1. end(); itr++)
   cout << * itr <<" ";
```

```
sort(v1. b
// reverse(
cout <<"
for(itr = v
cout <<
cout <<"
cout <<
cout <<"
cout <<
cout <<
if(bina
cout <<
else
cout <
return
}
```

output:-

vector
after s
10 20
Maximum
Minimum
The s
Elemen

```
sort(v1.begin(), v1.end());
// reverse(v1.begin(), v1.end());
cout << "\n after sorting: \n";
for (itr = v1.begin(); itr != v1.end(); itr++)
cout << *itr << " ";
cout << "\n Maximum element of vector is: ";
cout << *max_element (v1.begin(), v1.end());
cout << "\n Minimum element of vector is :";
cout << *min_element(v1.begin(), v1.end());
cout << "\n The sum of vector elements is :";
cout << accumulate(v1.begin(), v1.end(), 0);
if(binary_search(v1.begin(), v1.end(), 10))
cout << "\n Element found in the vector";
else
cout << "\n Element not found in the vector";
return 0;
}
```

Output:-

```
vector elements are : 10 40 20 30
after sorting:
10 20 30 40
Maximum element of vector is: 40
Minimum element of vector is: 10
The sum of vector elements is : 100
Element found in the vector.
```