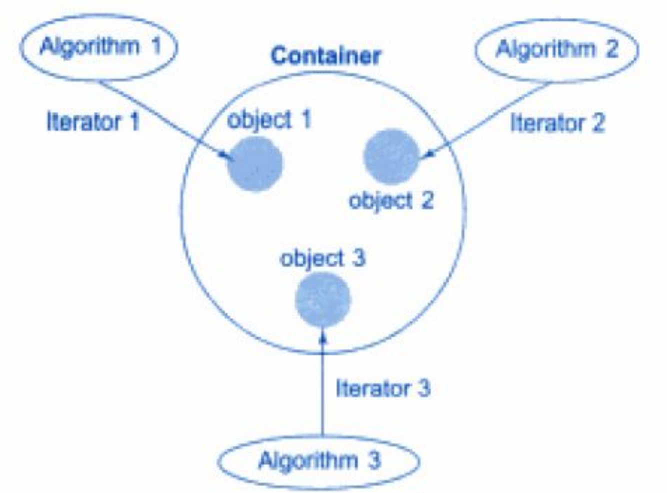
Standard Template Library(STL)

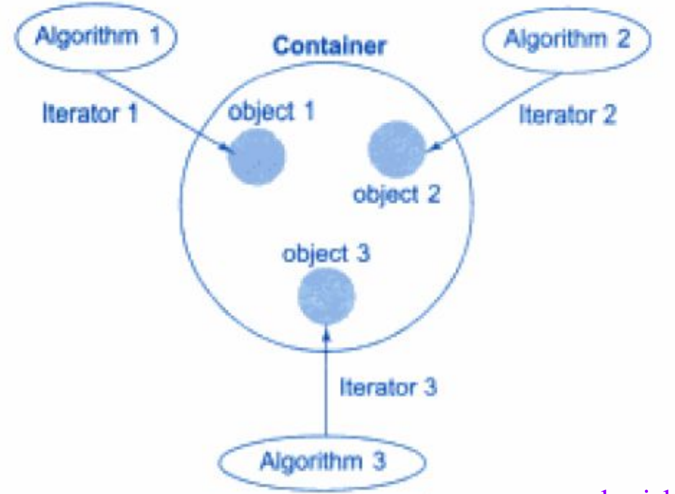
Standard Template Library(STL)

- Using STL, we can write shorter code that runs faster
- Pre written codes in STL are extremely error free
- It is a library of container classes, algorithms, and iterators.
- STL has three components
  - Containers
  - Algorithms
  - Iterators



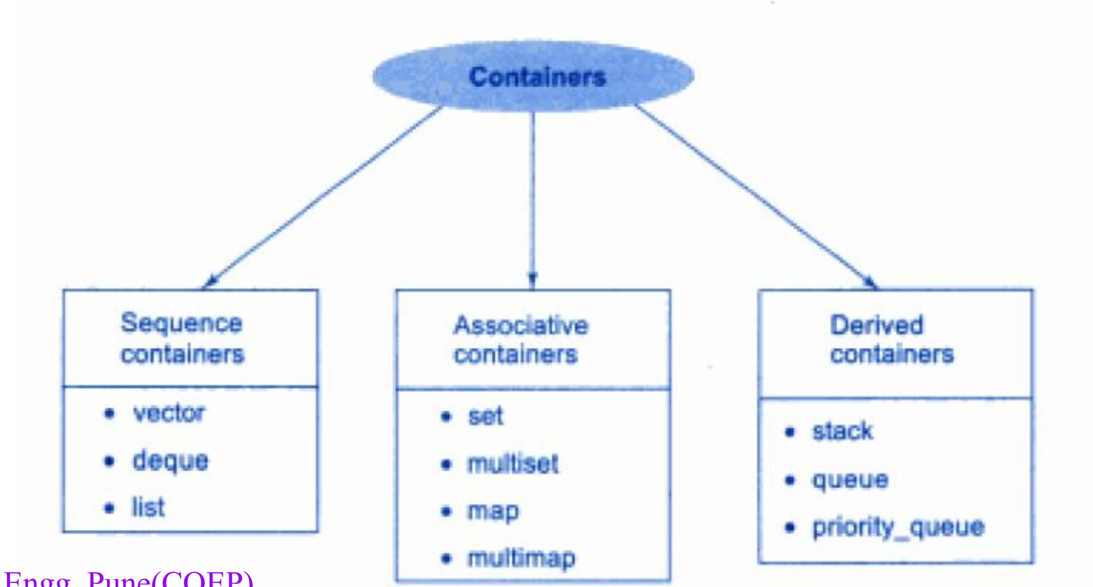
Standard Template Library(STL)

- A **containers** are the objects that holds the data of same kind
- An **algorithm** is a procedure that is used to process the data that contained in the container.
- **Iterator** is an object (like a pointer) that points to an element in a container.

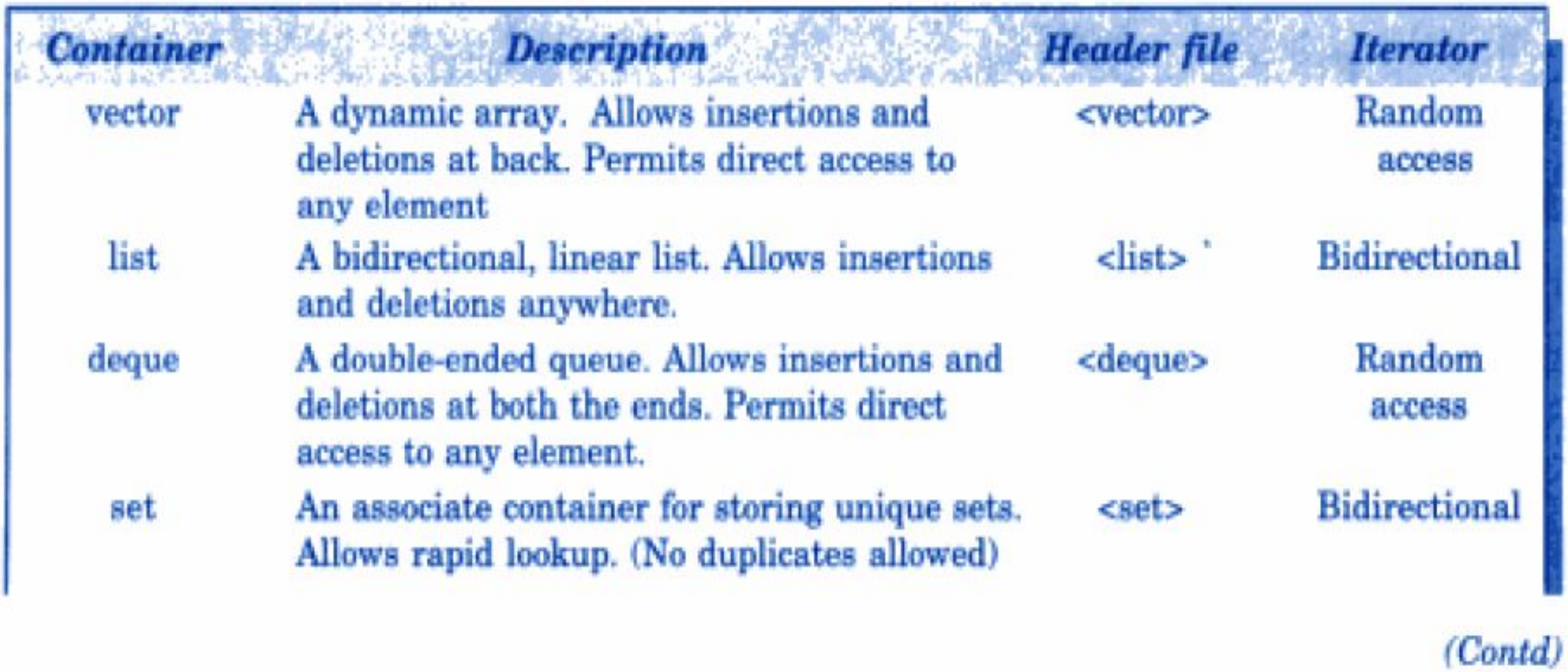


Containers

- A **containers** are the objects that holds the data of same kind
- STL defines 10 containers which are grouped into three categories



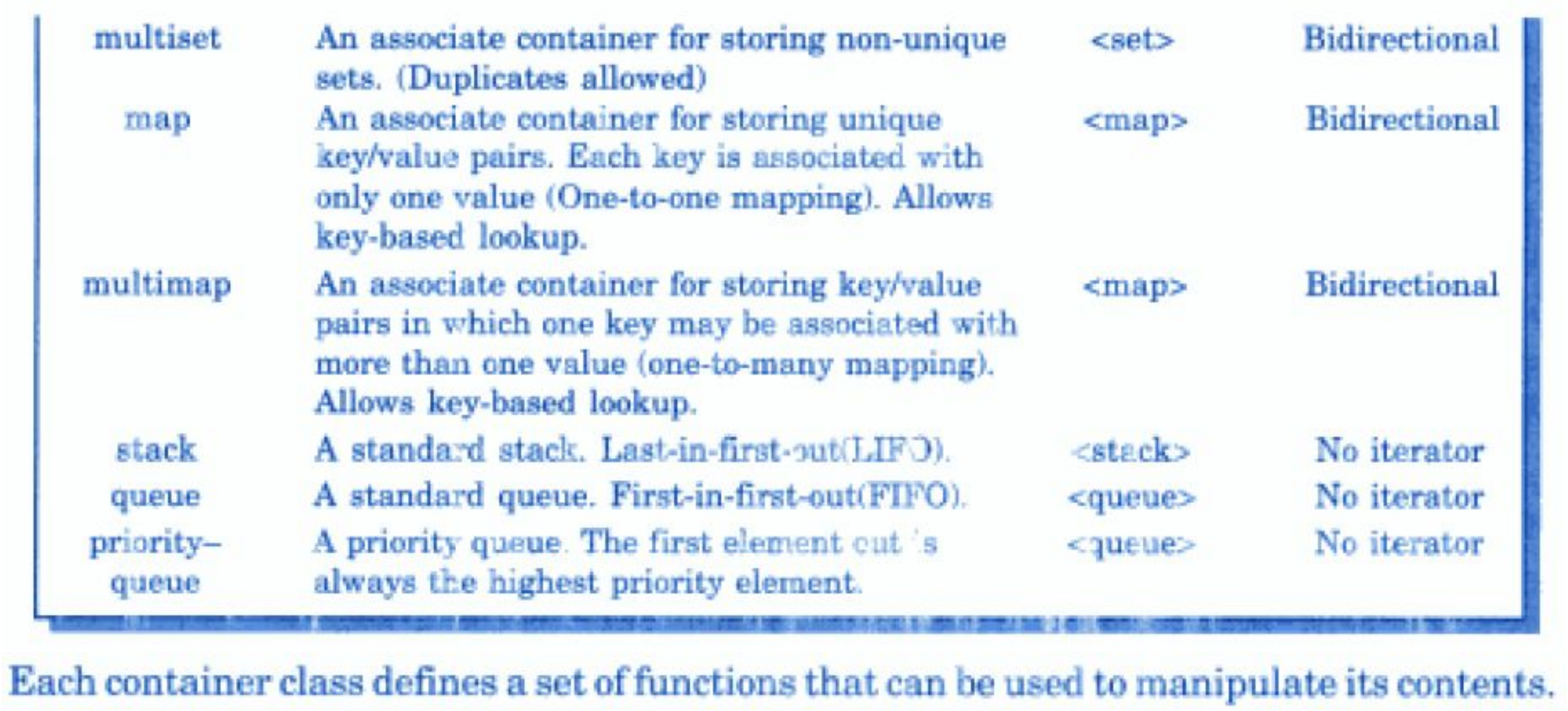
Containers



Container	Description	Header file	Iterator
vector	A dynamic array. Allows insertions and deletions at back. Permits direct access to any element	<vector>	Random access
list	A bidirectional, linear list. Allows insertions and deletions anywhere.	<list>	Bidirectional
deque	A double-ended queue. Allows insertions and deletions at both the ends. Permits direct access to any element.	<deque>	Random access
set	An associate container for storing unique sets. Allows rapid lookup. (No duplicates allowed)	<set>	Bidirectional

(Contd)

Containers



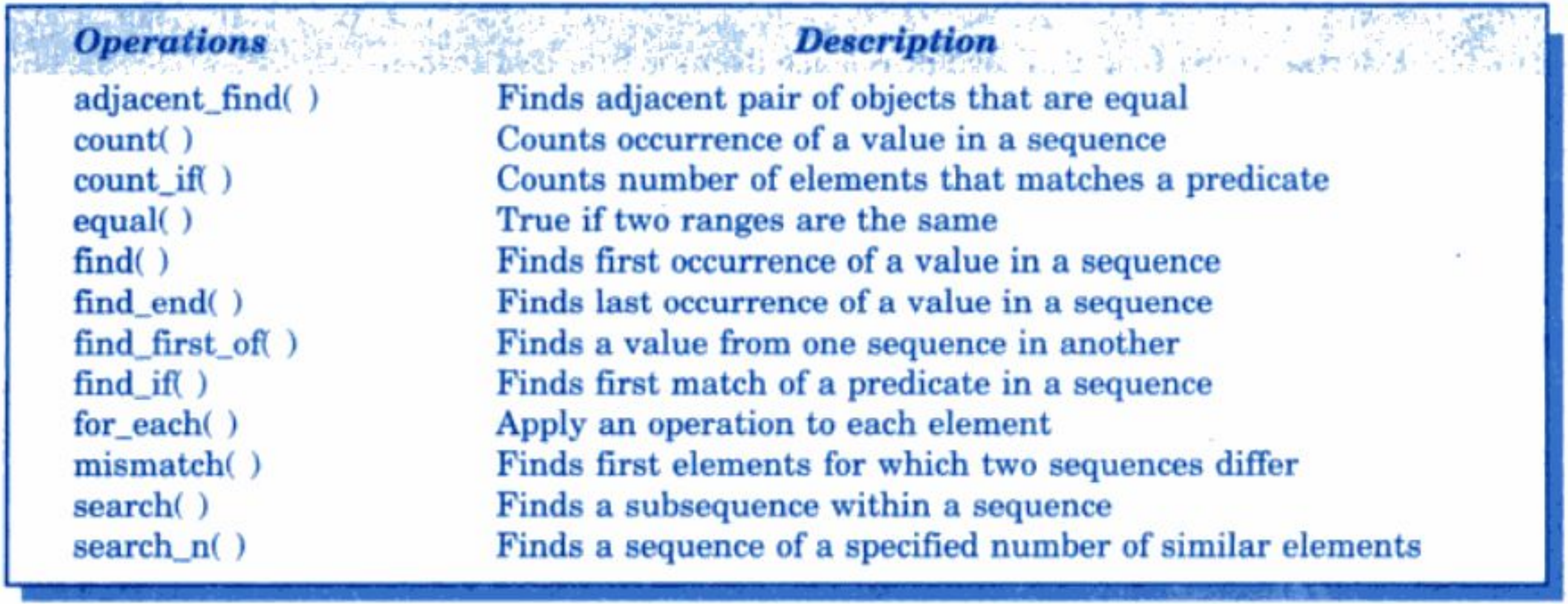
multiset	An associate container for storing non-unique sets. (Duplicates allowed)	<set>	Bidirectional
map	An associate container for storing unique key/value pairs. Each key is associated with only one value (One-to-one mapping). Allows key-based lookup.	<map>	Bidirectional
multimap	An associate container for storing key/value pairs in which one key may be associated with more than one value (one-to-many mapping). Allows key-based lookup.	<map>	Bidirectional
stack	A standard stack. Last-in-first-out(LIFO).	<stack>	No iterator
queue	A standard queue. First-in-first-out(FIFO).	<queue>	No iterator
priority–queue	A priority queue. The first element out is always the highest priority element.	<queue>	No iterator

Each container class defines a set of functions that can be used to manipulate its contents.

Algorithms

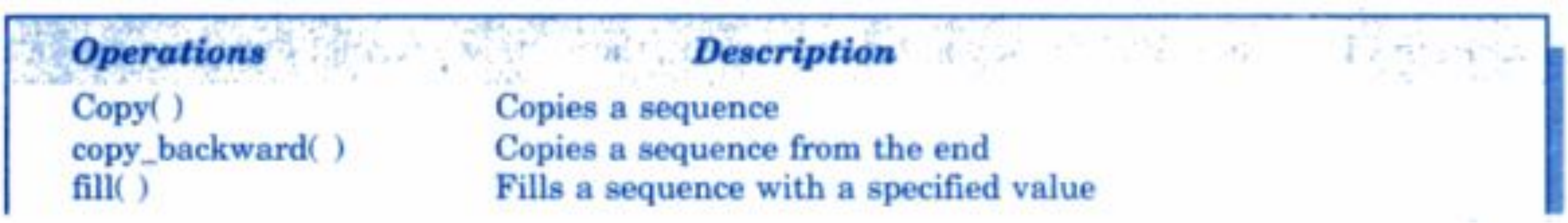
- Although containers provides functions for its basics operations, STL provides more than sixty standard algorithms.
- Standard algorithms also permits us to work two different types of containers at the same time.
- By using these algorithms, programmers can save lots of time and efforts.
- To have access to these STL algorithms, we must include <algorithm> in our program.
- STL algorithms can be categorized as
  - ○ Non-Mutating Algorithms
  - ○ Mutating Algorithms
  - ○ Sorting Algorithms

Non-Mutating Algorithms



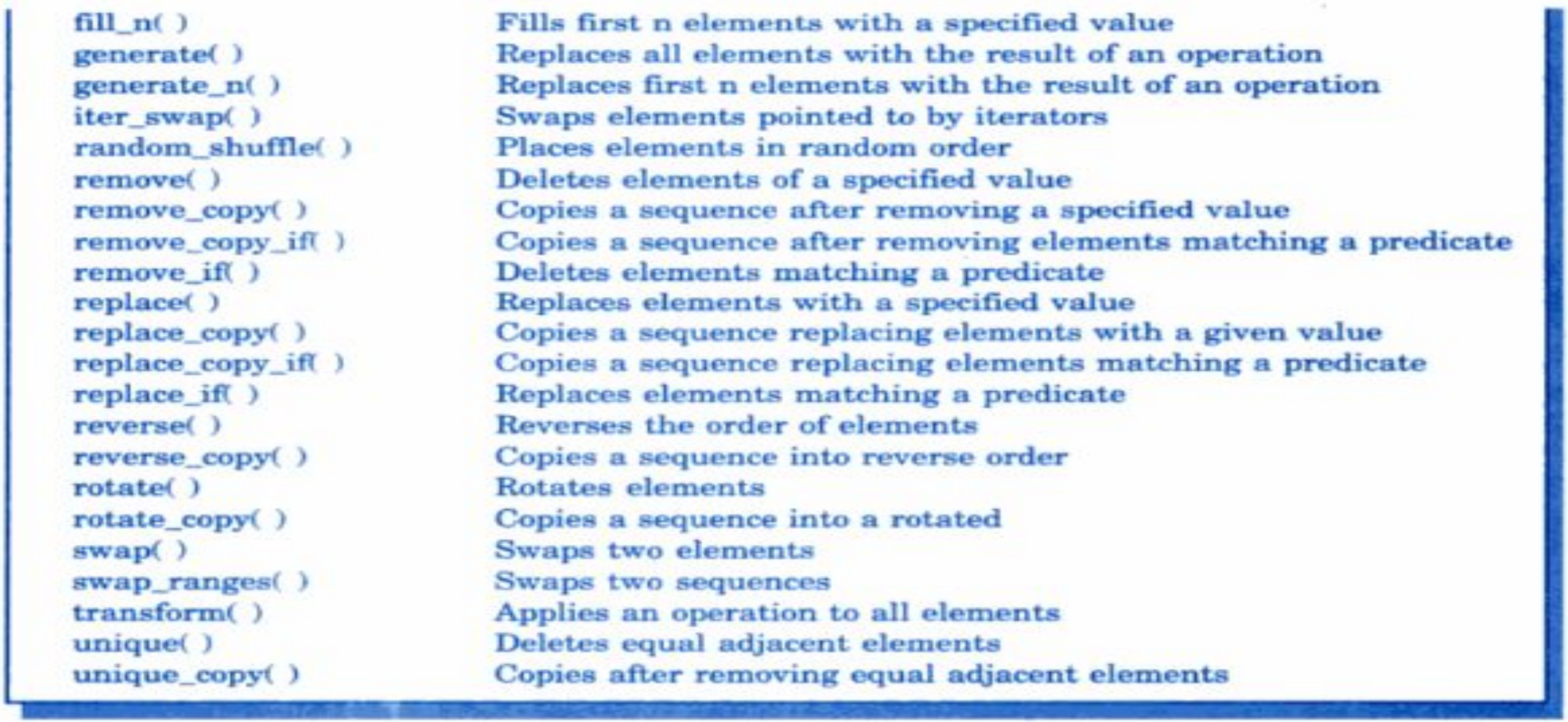
Operations	***Description***
adjacent_find( )	Finds adjacent pair of objects that are equal
count( )	Counts occurrence of a value in a sequence
count_if( )	Counts number of elements that matches a predicate
equal( )	True if two ranges are the same
find( )	Finds first occurrence of a value in a sequence
find_end( )	Finds last occurrence of a value in a sequence
find_first_of( )	Finds a value from one sequence in another
find_if( )	Finds first match of a predicate in a sequence
for_each( )	Apply an operation to each element
mismatch( )	Finds first elements for which two sequences differ
search( )	Finds a subsequence within a sequence
search_n( )	Finds a sequence of a specified number of similar elements

Mutating Algorithms



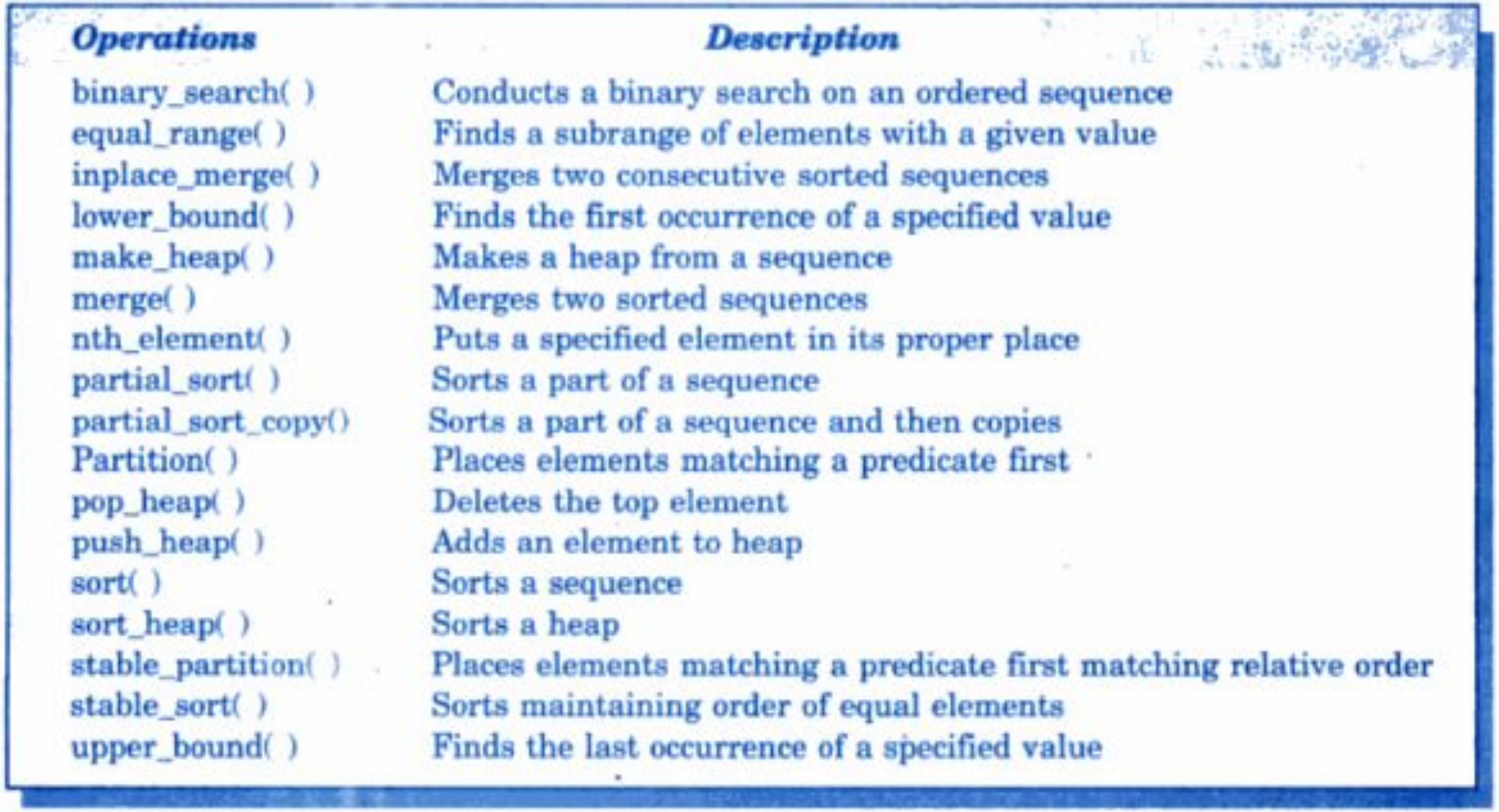
Operations	*Description*
Copy( )	Copies a sequence
copy_backward( )	Copies a sequence from the end
fill( )	Fills a sequence with a specified value

Mutating Algorithms



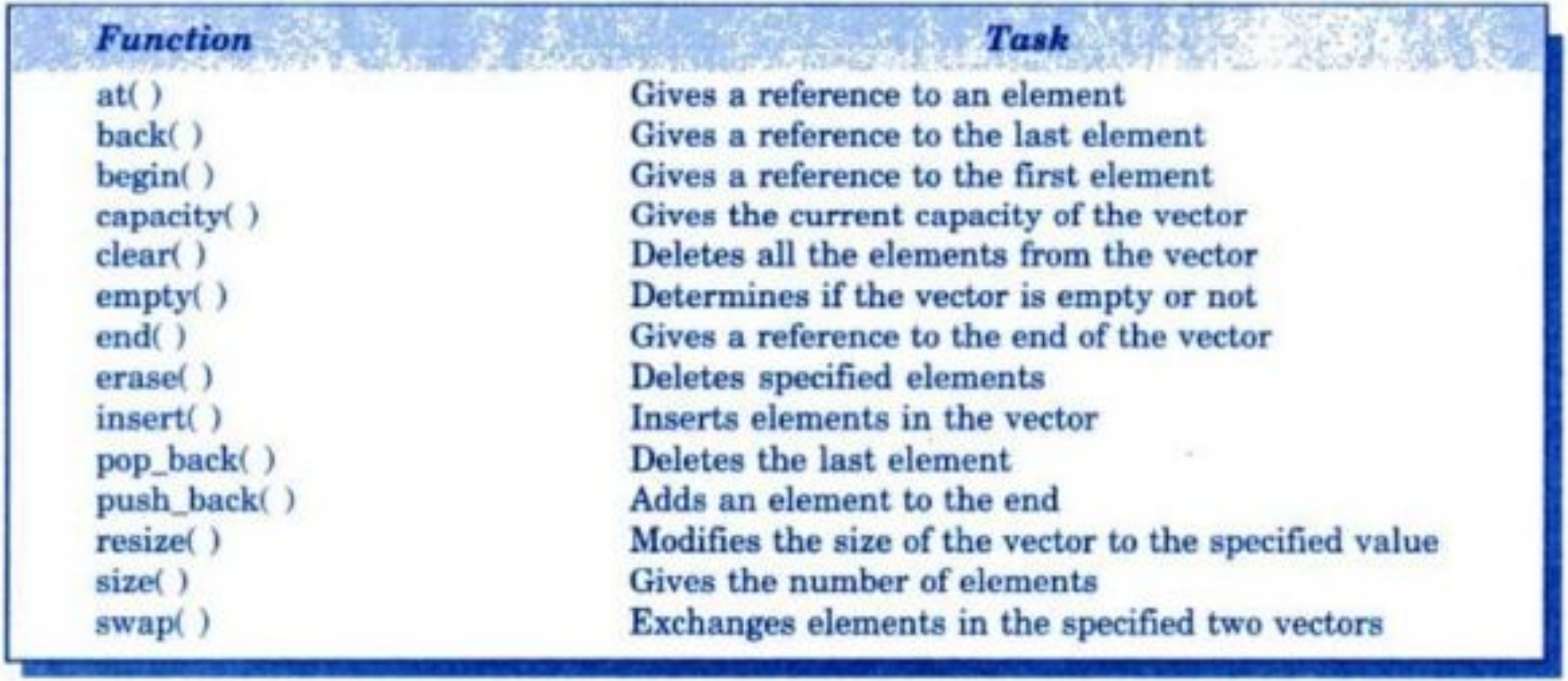
fill_n( )	Fills first n elements with a specified value
generate( )	Replaces all elements with the result of an operation
generate_n( )	Replaces first n elements with the result of an operation
iter_swap( )	Swaps elements pointed to by iterators
random_shuffle( )	Places elements in random order
remove( )	Deletes elements of a specified value
remove_copy( )	Copies a sequence after removing a specified value
remove_copy_if( )	Copies a sequence after removing elements matching a predicate
remove_if( )	Deletes elements matching a predicate
replace( )	Replaces elements with a specified value
replace_copy( )	Copies a sequence replacing elements with a given value
replace_copy_if( )	Copies a sequence replacing elements matching a predicate
replace_if( )	Replaces elements matching a predicate
reverse( )	Reverses the order of elements
reverse_copy( )	Copies a sequence into reverse order
rotate( )	Rotates elements
rotate_copy( )	Copies a sequence into a rotated
swap( )	Swaps two elements
swap_ranges( )	Swaps two sequences
transform( )	Applies an operation to all elements
unique( )	Deletes equal adjacent elements
unique_copy( )	Copies after removing equal adjacent elements

Sorting Algorithms



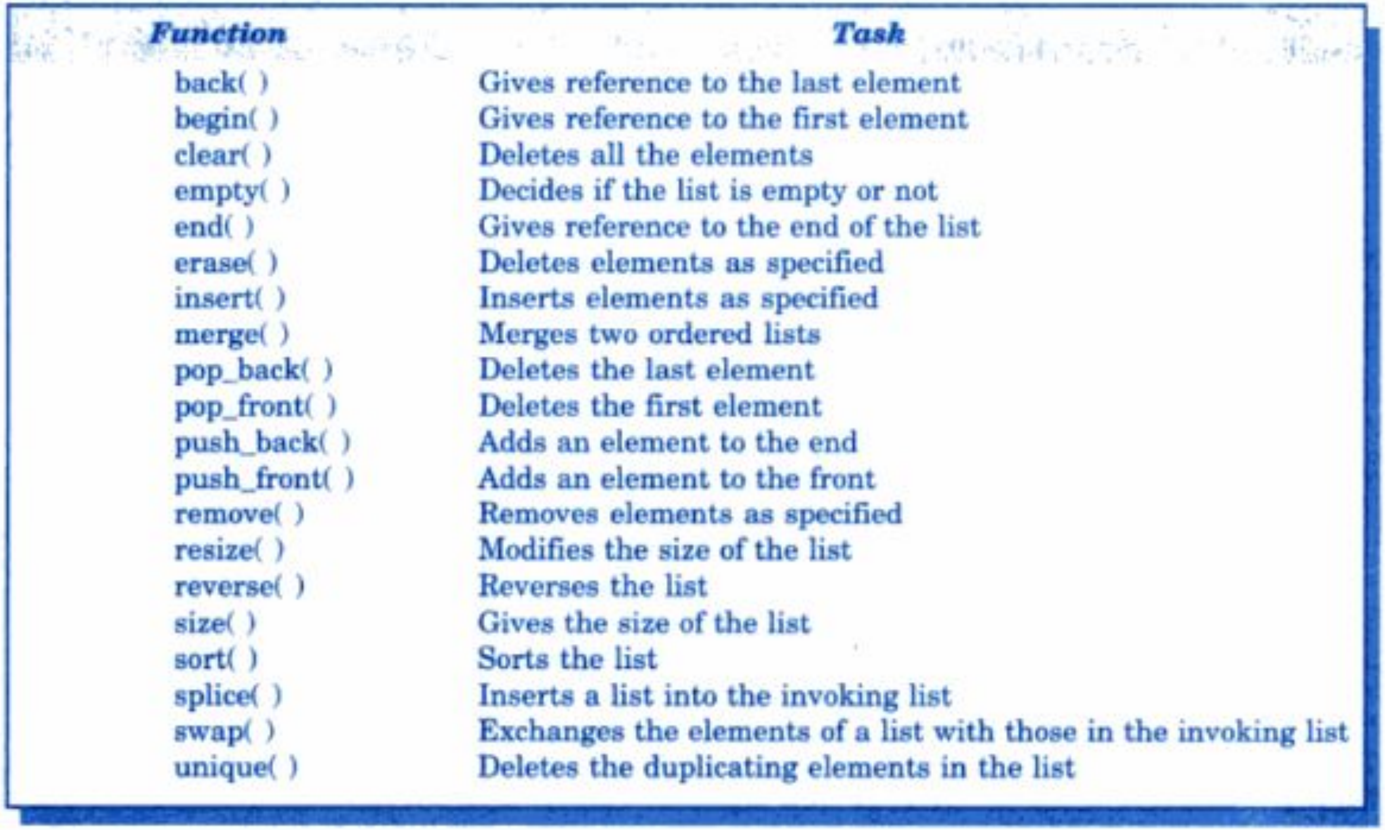
Operations	*Description*
binary_search( )	Conducts a binary search on an ordered sequence
equal_range( )	Finds a subrange of elements with a given value
inplace_merge( )	Merges two consecutive sorted sequences
lower_bound( )	Finds the first occurrence of a specified value
make_heap( )	Makes a heap from a sequence
merge( )	Merges two sorted sequences
nth_element( )	Puts a specified element in its proper place
partial_sort( )	Sorts a part of a sequence
partial_sort_copy()	Sorts a part of a sequence and then copies
Partition( )	Places elements matching a predicate first
pop_heap( )	Deletes the top element
push_heap( )	Adds an element to heap
sort( )	Sorts a sequence
sort_heap( )	Sorts a heap
stable_partition( )	Places elements matching a predicate first matching relative order
stable_sort( )	Sorts maintaining order of equal elements
upper_bound( )	Finds the last occurrence of a specified value

Member Function for Vector Class



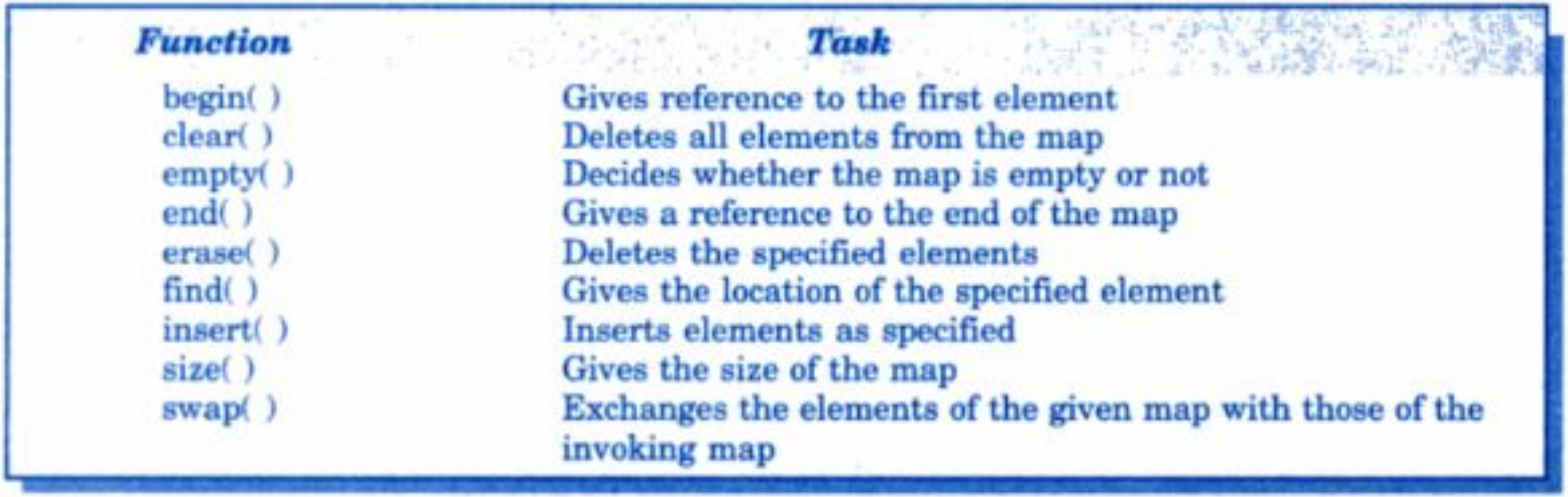
Function	*Task*
at( )	Gives a reference to an element
back( )	Gives a reference to the last element
begin( )	Gives a reference to the first element
capacity( )	Gives the current capacity of the vector
clear( )	Deletes all the elements from the vector
empty( )	Determines if the vector is empty or not
end( )	Gives a reference to the end of the vector
erase( )	Deletes specified elements
insert( )	Inserts elements in the vector
pop_back( )	Deletes the last element
push_back( )	Adds an element to the end
resize( )	Modifies the size of the vector to the specified value
size( )	Gives the number of elements
swap( )	Exchanges elements in the specified two vectors

Member Function for List Class



Function	*Task*
back( )	Gives reference to the last element
begin( )	Gives reference to the first element
clear( )	Deletes all the elements
empty( )	Decides if the list is empty or not
end( )	Gives reference to the end of the list
erase( )	Deletes elements as specified
insert( )	Inserts elements as specified
merge( )	Merges two ordered lists
pop_back( )	Deletes the last element
pop_front( )	Deletes the first element
push_back( )	Adds an element to the end
push_front( )	Adds an element to the front
remove( )	Removes elements as specified
resize( )	Modifies the size of the list
reverse( )	Reverses the list
size( )	Gives the size of the list
sort( )	Sorts the list
splice( )	Inserts a list into the invoking list
swap( )	Exchanges the elements of a list with those in the invoking list
unique( )	Deletes the duplicating elements in the list

Member Function for Map Class



Function	*Task*
begin( )	Gives reference to the first element
clear( )	Deletes all elements from the map
empty( )	Decides whether the map is empty or not
end( )	Gives a reference to the end of the map
erase( )	Deletes the specified elements
find( )	Gives the location of the specified element
insert( )	Inserts elements as specified
size( )	Gives the size of the map
swap( )	Exchanges the elements of the given map with those of the invoking map

Vector

Vector

It is dynamic sized array. Number of elements can be increased or decreased

```
    Vector <int> v;             // Empty vector of integers
    Vector <int v(10);          // Vector of integers with 10 elements(all 0)
    vector <char> v(10,’h’);  //Vector of strings with 10 elements(all h)
```

Important Functions:

```
  v.push_back(x);     // Insert the value of X at the end of the vector. O(1)
  v.pop_back()            //Erase the last element. O(1)
  v.clear()           // Erase all the elements. O(n)
  v.size()                //Returns current size of the vector. O(1)
```

[ ] operator is used to access the elements like an array.

```
     cout<<v[10];        prints first elements in vector.
```

Vector

```
#include<bits/stdc++.h>
using namespace std;
int main()
{   vector <int> v;
    cout<<"Size : "<<v.size()<<endl;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    cout<<v[0]<<"  "<<v[1]<<"  "<<v[2]<<endl;
    cout<<"Size : "<<v.size()<<endl;
    v.pop_back();
    cout<<"Size : "<<v.size()<<endl;
    v.push_back(90);
    cout<<"Size : "<<v.size()<<endl;
    cout<<"\n";
    cout<<v[0]<<"  "<<v[1]<<"  "<<v[2]<<endl;
}
```

Vector

```
#include<bits/stdc++.h>
using namespace std;
int main()
{   vector <int> v;
    cout<<"Size : "<<v.size()<<endl;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    cout<<v[0]<<"  "<<v[1]<<"  "<<v[2]<<endl;
    cout<<"Size : "<<v.size()<<endl;
    v.pop_back();
    cout<<"Size : "<<v.size()<<endl;
    v.push_back(90);
    cout<<"Size : "<<v.size()<<endl;
    cout<<"\n";
    cout<<v[0]<<"  "<<v[1]<<"  "<<v[2]<<endl;
}
```

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
    vector <int> v(10);
    cout<<v[3];
}
```

Vector

```
#include<bits/stdc++.h>
using namespace std;
int main()
{   vector <int> v;
    cout<<"Size : "<<v.size()<<endl;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    cout<<v[0]<<"  "<<v[1]<<"  "<<v[2]<<endl;
    cout<<"Size : "<<v.size()<<endl;
    v.pop_back();
    cout<<"Size : "<<v.size()<<endl;
    v.push_back(90);
    cout<<"Size : "<<v.size()<<endl;
    cout<<"\n";
    cout<<v[0]<<"  "<<v[1]<<"  "<<v[2]<<endl;
}
```

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
    vector <int> v(10);
    cout<<v[3];
}
```

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
    vector <int> v(10,2);
    cout<<v[3];
}
```

Vector

```
// Illustrate push_back(),
pop_back(), size(), empty(),
capacity()
#include<vector>
#include<iostream>
using namespace std;

int main()
{
    vector<int>v;
    int x;

    cout<<"\nEnter Elements : \n";
    for(int i=0;i<5;i++)
    {
        cin>>x;
        v.push_back(x);
    }
    cout<<"size is : "<<v.size()<<endl;

    cout<<"Vector Elements are  :";
    for(int i=0;i<v.size();i++)
    {
        cout<<v[i]<<"   ";
    }
    cout<<"\nBack element is popped :";
    v.pop_back();

    cout<<"\nSize is : "<<v.size();
    cout<<"\nCapacity : "<<v.capacity();

    cout<<"\nEmpty/Non-Empty : "<<v.empty();
    return(0);
}
```