Standard Template Library

The STL is an advanced application of templates. It Contains several in-built functions and operators that Help the Programmer to develop Complex Programs.

→ The STL Programs Helps to write a bug-free Program. In Case any bug presents. it is easy to debug it. Through STL we Can reduce non of lines (Code) to write Programs with different types of data structures.

→ The Heng Lee and Alexander Stepanov of H·P introduced 'STL'.

→ STL Provides well-Coded and Compiled data structures and functions that are helpful in generic Programming and it is reusable.

→ STL Contents are defined in the namespace 'std'

18/10 – 68, 82, 86, 87, 92, 93, A3
A8, B1, 14-96

→ STL Programming Model :-

STL is divided into 3 parts

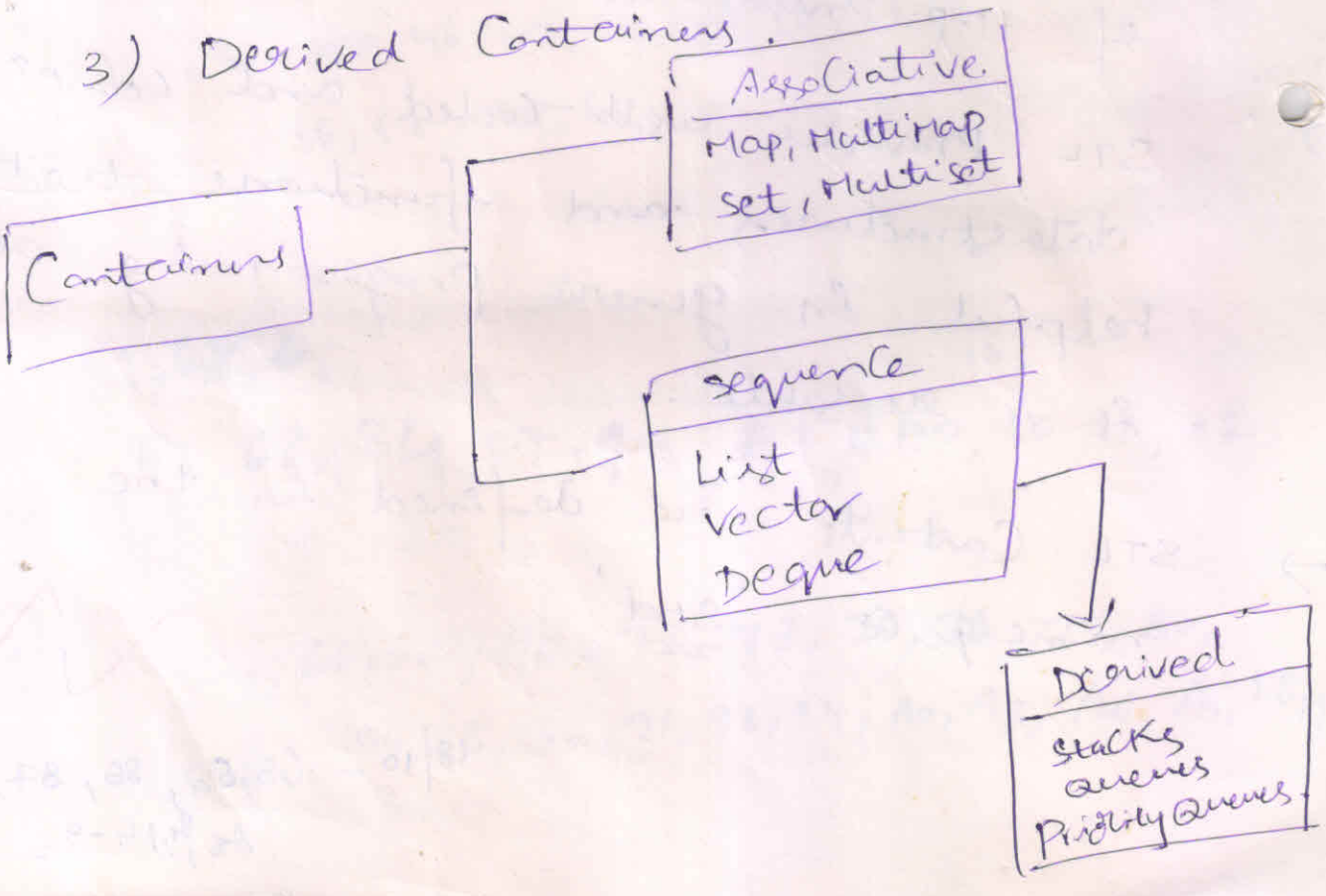
i) Containers

ii) Algorithms

iii) Iterators.

Containers :- STL Library has several Container classes to control Common Programming operations. There are 3 types of Containers.

1) Sequence Containers
2) Associative Containers
3) Derived Containers.



Traverse list using iterator

Using

```
typedef list<int> num_list;

int main()
{
  num_list num;
  for(int j=0; j<=5; ++j)
  num.push_back(j);
  for( num_list::const_iterator ls=num.begin();
       ls!=num.end(); ++ls)
    cout<<*ls<<" ";
    return 0;
}
```

4/1/22

Ab:- 75, A4, B8,

(3) 4

Iterators :-

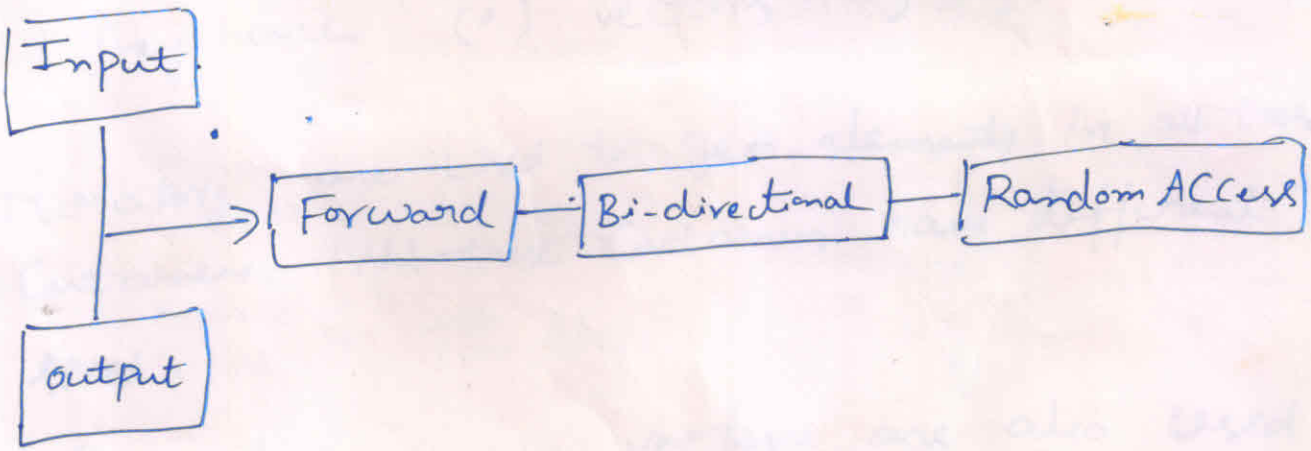
① Iterator is an object. It behaves exactly similar to pointer. It indicates or points to the dataelement in a Container.

2) It is utilized to move between the data items of Containers. Iterators can be incremented or decremented similar to Pointers. They link algorithms with Containers and handle the data stored in the Containers.

3) As iterators allow movement from one element to another element, it is called iterating.

4) Each Container type supports one kind of iterator according to the Container's necessity.

5)



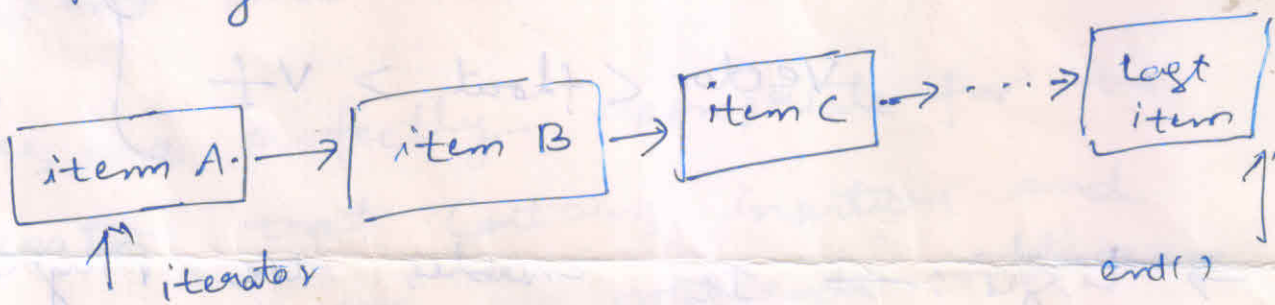
Iterator hierarchy

(6)

Operation of Container classes :

Iterator	Access Mode	way of Action	I/O ability	Comment
Forward	Linear	Can move forward only	write & read	Storable
Bi-directional	Linear	move front & back	write & read	"
Random Access	Random	move front & back	write & read	"
Input	Linear	move forward only	Read only	Non Storable
output	Linear	move forward only	write only	Non Storable

Sequence Containers :- STL sequence Containers allow controlled sequential access to elements. Sequence Containers hold data elements in a linear series. Every element is associated by it's location within the series. These elements also allow various operations such as copying and finding



Data elements in Sequence Container.

→ The STL has 3 types of Sequence Containers. They are (i) vectors (ii) Lists (iii) Deques

→ Iterators are used to get elements in all these Containers. All these Containers have different speed.

Vectors :- Like arrays, vectors are also used to store data elements and used to

insert or delete an element. All the ele
are accessed randomly i.e; They suppor
random access to individual elements.

→ Unlike arrays, the vectors can grow dynamically
i.e; they enlarge their size whenever
required.

Eg :- Vector<int> Vi
Vector<float> Vf
} declaration of vector.

→ Lists :— It enables the programmer
to perform the usual deletion and
insertion of items. It is also a
sequence but items can be accessed
bi-directionally. It is defined in the
headerfile <list>

→ here iterator is used to traverse the list.
The list supports all the member functions
of vector class.

→ Here iterator is similar to a pointer

iterator can be referenced similar to pointers to access elements.

→ Deques :— A deque is similar multiple ended vector. It has the ability to perform sequential read and write operations. The deque class allows improved frontend and back-end operations.

→ deques are perfectly appropriate for the operation that contains insertion and deletion at one or both ends where sequential access is essential.

→ Associative Containers :—

These allows fast access to the object in the Containers. These Containers are useful for huge dynamic tables in which we can search for an element randomly and sequentially.

→ These Containers use tree-like structures of elements instead of linked-lists.

(10)

→ Associative Containers are non-sequential and allow direct access to elements. They are (i) sets (ii) Multisets (iii) Maps (iv) Multimaps.

All these Containers hold data elements in a structure called a "tree" - The Tree data structure provides quick finding, deletions and insertions

→ The Containers sets & Multisets stores data elements Here Variable name is used as Key name.

→ The Multiset may have multiple sets of elements ie; it permits duplicate data elements.

→ Containers maps and multimaps stores both Key names and values. Usually values are associated with Keynames. These values are also called mapped values.

→ Multimaps allow various Keys, Map Containers permits Single Keys.

Algorithms :- Algorithms are independent template functions. They are not members or friend functions. They enable the Programmer to manipulate the data stored in various Containers.

→ Algorithms Carryout operations on Containers by dereferencing iterators.

→ an algorithm is nothing but a function template with arguments of iterative type. ie; algorithms receive iterators as arguments.

→ STL Containe approximately 60 standard algorithm

→ To use these algorithms include <algorithm> header file.

→ In general we have the following type of algorithms :

i) Non-mutating sequence operation
[search_n(), find_if(), search(), find_first_of(), find(), find_end() etc]

ii) sorting operators
[binary_search(), equal(), includes(), set_union(), sort(), sort_heap() etc]

(12)

iii) Mutating sequence operations

[swap_ranges( ), generate( ), fill( ),
reverse( ), copy( ), unique( ),
remove( ), replace( ) .. etc]

iv) Numeric operating

[Inner_Product( ), Adjacent_difference( ),
Accumalate( ), Partial_Sum( )]

11/10 A6:- 5, 9, 10, 11, 14, ~~15~~, 17, 19, 23, 24, 26, 29, 30,
35, ~~40~~ 46, 47, 48, ~~49~~, 50, 51, 52, 53, 54, 55,
56, 57, ~~58~~, 60

A5:- ~~61, 62, 6~~

19-

4/10 63, 68, ~~84~~ 87, 93, A1, ~~A2~~ 00, 10, 11, 02.

16/10 64, 66, 68, 70, 72, 73, 75, ~~76~~, 79, 80,
81, 84, 86, 87, 91, 93, 94, A0, A5, A6, A9, LE11

→ write a program to add and display elements in vector object

```
#include <vector>
int main()
{
 vector <int> e;
  e.push_back (5);
  e.push_back (8);
  e.push_back (9);
 Cout <<" In elements in the vector are: "<<"\n";
  Cout << e[0] << " " endl;
  Cout << e[1] << endl;
  Cout << e[2] << endl
```

→ Vector and iterator

```
#include <vector>
int main()
{
  vector <int> e;
for(int x =0; x<3; x++)
  e.Push_back (x+1);
Cout <<" Elements from vector are:";
  for (int x =0; x<3; x++)
   Cout << e[x] << " ";
```

```
vector<int>:: iterator it = e.begin();
    it = it+1;
    e.insert(it,7);
    cout "\n The elements after insertion :"
    for(x=0; x<4; x++)
      cout << e[x] << " ";
      return 0;
}
```

→ Display elements of vector in ascending & descending order

```
int main()
{
vector<int> e;
cout<<"Elements in ascending order: ";
for (int x=0; x<8; x++)
{ e.push_back(x+1);
  cout << x+1 <<" ";
}
    int s = e.size()-1;
    cout<<"\n elements in descending order: ";
```

```
for (int j=s; j>=0; j--)
cout << e[j] << " ";
cout << "\n";
return 0;
}
```

→ Program to remove elements from vector

vector
math.h

```
int main()
{
vector <float> e;
cout.precision(2);
cout << "\n original elements :";
for(int k=5; k<12; k++)
{
e.push_back(sqrt(k));
cout << e[k] << " ";
}
vector <float>::iterator ir = e.begin();
e.erase(ir+3, ir+5);
int s = e.size();
cout << "\n The elements after erase() :";
```

```
    cout << e[k] << " ";
    cout << "\n";
    return 0;
}
```

→ List

Creating:

```
#include <list>

void show (list <int> &num)
{
    list<int> :: iterator n;
    for (n = num.begin(); n != num.end(); ++n)
    cout << *n << " ";
}
int main()
{
    list <int> list;
    list.push_back (5);
    list.push_back (10);
    list.push_back (20);
    cout << " List numbers are: ";
    show (list);
    return 0;
```

→ Add numbers at both ends of list

```
Void show (list<int> &num)
{
 list<int> :: iterator n;
 for (n = num.begin (); n != num.end (); ++n)
 Cout << *n << " ";
}
int main ()
{
 list <int> list;
 list.push_back (5);
 list.push_back (10);
 list.push_back (20);
 Cout << "in Numbers are:";
  show (list);
  list.push_front (1);
  list.push_back (25);
 Cout << "in After Adding, the list elements:";
    show(list);
}     return 0;
```

5/1
Ab:—
72, 99, B4,
co,

(18)

→ To delete elements from list

```
Void show (list<int> &num)
{
   list<int> :: iterator n;
   for(n=num.begin(); n!=num.end(); ++n).
   Cout << *n << " ".
}
int main()
{
   list<int> list;
   list.push_back(5);
   list.push_back(10);
   list.push_back(15);
   list.push_back(20);
  Cout << " The list elements are: ";
   show (list);
   list.pop_front();
   list.pop_back();
  Cout << "After deletion, the list is: ";
    show(list);
    return 0.
}
```

→ To sort

```
void show (list <int> &num)
{
  list <int> :: iterator n;
  for(n=num.begin (); n!=num.end ();
                          ++n)

  Cout << *n << " ";
}
int main()
{
  list <int> list;
  list.push_back (25);
  list.push_back (19);
  list.push_back (5);
  list.push_back (15);
  list.push_back (25);
  list.push_back (20);
  Cout <<"in Unsorted list:";
    show (list);
  Cout <<"in sorted list:";
    list.sort();
    show (list);
  return 0;
}
```

→ To reverse a list use

```
list.reverse();
show(list);
```

→ Merging

```
void show(list <int> &num)
{
  list<int>::iterator n;
  for(n=num.begin(); n!=num.end(); ++n)
  cout << *n <<" ";
}
int main()
{
  list<int> listX, listY;
  listX.push_back(23);
                 (19);
                 (5);

  listY.push_back(15);
                 (25);
                 (20);
cout<<"Elements of listx:";
  show(listx);
  cout<<"in listy:";
  show(listy);
}
```

```
listx.merge(listy);
cout<<"in Merged list:";
show(listX);
return 0;
```

→

```
#include <map>
using namespace std;
typedef map<string, int> item_map;
int main()
{
  int sz;
 string item_name;
  int code_no;
  item_map item;
 cout << "Enter item name and code_no number
    for 2 items : \n";
for(int i=0; i<2; i++)
  {
     cin >> item_name;
     cin >> code_no;
  }
    item["pc"] = 2510;
 item.insert(pair<string, int>
              ("printer", 2211));
     sz. item.size();
```

```
cout << "\n size of map: " << sz << "\n\n";
cout << "List of item name and code number\n";
item_map :: iterator t;
for( t=item.begin(); t!=item.end();
                                t++)

cout << (*t).first << "  " << (*t).second
     << "\n";
cout << "\n";
cout << "Enter item name: ";
cin >> item_name;
codeno = item[item_name];
cout << " code number: " << codeno << "\n";
return 0;
}
```

→ clear – removes all elements from map
begin – provides reference of first element
erase – Removes the given elements
insert – Inserts the elements as given
empty() – Determines whether the map is vacant or not
end() – Provides references to the end of the map
size() – Provides size of the map

find ( ) — Provides the location of the given elements

swap ( ) — swaps the elements of the given map with that of the calling map.

Function of vector class :

swap ( ) — swap the elements in the given two vector

size — Provides the number of elements

resize ( ) — changes size of V

push_back ( ) — Appends element on the end

pop_back ( ) — Erases last element

insert ( ) — Inserts items in the vector

erase ( ) — Erases given elements

end ( ) — Provides reference to the end of vector

empty ( ) — Determine whether the vector is empty

clear ( ) — Erases entire elements from the vector

capacity ( ) — Provides the present capacity of vector

(24)

11/1/22 Ab:- 71, 79, 80, 97, 43;

12/1/22 65, 71, 72, 77, 78, 79, 81, 83, 84, 88, 89, Ab:- 91, 94, 98, A2, A3, A4, A5, A7, A8, B1, B2, B5, B6, B7, B8, C7 → (27) les, le6

- begain ( ) —

back ( ) — provides a ref to last element

at ( ) — Provides a ref to an ele

List functions:

swap ( ) — swaps the elements of list with those in the calling list

sort ( ) — Sorts the elements

clear ( ) — erases entire elements

back ( ) — Provides ref to the end element

empty ( ) — Determine if the list is vacant or not

begin ( ) — Provides ref to first element

erase ( ) — Erases given element

end ( ) — Provides reference of the end element of the list

merge ( ) — Combines 2 sorted lists

POP_front ( ) — Erases first elem

POP_back ( ) — " last elem

remove ( ) — Erases elements as specified

insert ( ) — Adds given element

Push-back ( ) — Appends an element to the end

Push-front ( ) — " " front

size ( ) — Provides size of list

unique ( ) — Erases identical elements in the list

99, 82, A3

Algorithms :

→ Algorithms are Independent template functions They are not members or friend functions.

→ They enable the programmer to manipulate the data stored in various Containers. Algorithms Carryout operations on Containers by deferencing iterators.

→ Each Container has it's functions for performing Common operations.

→ An algorithm is nothing but a function Template with arguments of iterator type.

→ Algorithm receives iterators as arguments.

→ The iterator informs the algorithm regarding the object of the Container on which the operation is to be Carried out.

→ STL Contains nearly 60 standard algorithms.

→ we should include <algorithm> heard file. to use these functions.

→ Algorithms Provides 3 types of operations

i) Non-Mutating sequence operations.

ii) Sorting operations

iii) Mutating sequence operations.

Non-Mutating sequence operations :-

1) search_n( ) — searches a sequence of a given number of same elements

2) find_if( ) — searches first equivalent of a predicate in a sequence

3) search( ) — searches sub-sequence in other sequence

4) find_first_of( ) — searches a value from one sequence in another

5) find( ) — searches first presence of a value in a sequence

6) find_end( ) — searches last occurence of a value in a sequence

7) adjacent_find( ) — searches contiguous pair of objects that are identical

pop_heap( ) — erases uppermost elements

push_heap( ) — Appends or adds an element to heap

merge( ) — Combines two two sorted sequences

⋮

Mutating sequence operations :—

fill( ) — fill up a series with a given values

copy( ) — Duplicates (copies) a sequence.

remove( ) — Erases elements of a given value

replace( ) — substitutes elements with a specified value

swap( ) — Exchange two elements

unique( ) — Erases similare Contiguous elements

⋮

mismatch( ) — searches element for which two sequences vary

Count( ) — Calculate presence of a value in a sequence

Count_if( ) — Calculate elements that Similar a predicate

equal( ) — True if two series are identical

for_each( ) — Performs a operation with each element.

Sorting operations:

sort( ) — Sorts the sequence Contains

equal( ) — searches whether the two sequences are equal or not

binary_search( ) — performs a binary search on an indexed sequence

max( ) — returns greater of two values

min( ) — returns smaller of two values

P

Algorithms in STL (sort, find ,search)

sort() Algorithm :

Sorting is one of the most basic functions applied to data. It means arranging the data in a particular fashion, which can be increasing or decreasing. There is a builtin function in C++ STL by the name of sort().
This function internally uses IntroSort. In more details it is implemented using hybrid of QuickSort, HeapSort and InsertionSort. By default, it uses QuickSort but if QuickSort is doing unfair partitioning and taking more than N*logN time, it switches to HeapSort and when the array size becomes really small, it switches to InsertionSort.

The prototype for sort is : sort (startaddress, endaddress)

startaddress: the address of the first
element of the array
endaddress: the address of the next
contiguous location of the
last element of the array.
So actually sort() sorts in the range of [startaddress,endaddress)

Program using sort() :

```
// C++ program to sort an array
#include <algorithm>
#include <iostream>

using namespace std;

void show(int a[], int array_size)
{
   for (int i = 0; i < array_size; ++i)
      cout << a[i] << " ";
```

```
}

// Driver code
int main()
{
   int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };

   // size of the array
   int asize = sizeof(a) / sizeof(a[0]);
   cout << "The array before sorting is : \n";

   // print the array
   show(a, asize);

    // sort the array
   sort(a, a + asize);

   cout << "\n\nThe array after sorting is :\n";

   // print the array after sorting
   show(a, asize);

   return 0;
}
```

find() Algorithm :

Finds the element in the given range of numbers. Returns an iterator to the first element in the range [first,last) that compares equal to val. If no such element is found, the function returns last.

Function Template :

InputIterator find (InputIterator first, InputIterator last, const T& val)

first,last :
Input iterators to the initial and final positions in a sequence. The range searched is [first,last), which contains all the elements between first and last, including the element pointed by first but not the element pointed by last.

val :
Value to be search in the range
Return Value :
An iterator to the first element in the range that compares equal to val.
If no elements match, the function returns last.

Program using find() Algorithm

```
#include<bits/stdc++.h>
 int main ()
{
   std::vector<int> vec { 10, 20, 30, 40 };

   // Iterator used to store the position
   // of searched element
   std::vector<int>::iterator it;

   // Print Original Vector
   std::cout << "Original vector :";
   for (int i=0; i<vec.size(); i++)
      std::cout << " " << vec[i];

   std::cout << "\n";

   // Element to be searched
   int ser = 30;

   // std::find function call
   it = std::find (vec.begin(), vec.end(), ser);
   if (it != vec.end())
   {
      std::cout << "Element " << ser <<" found at position : " ;
      std::cout << it - vec.begin() << " (counting from zero) \n" ;
   }
   else
      std::cout << "Element not found.\n\n";

   return 0; }
```

Search () Algorithm :

Search () Algorithm :

std::search is defined in the header file <algorithm> and used to find out the presence of a subsequence satisfying a condition (equality if no such predicate is defined) with respect to another sequence.

- It searches the sequence [first1, last1) for the first occurrence of the subsequence defined by [first2, last2), and returns an iterator to its first element of the occurrence, or last1 if no occurrences are found.
- It compares the elements in both ranges sequentially using operator== (version 1) or based on any given predicate (version 2). A subsequence of [first1, last1) is considered a match only when this is true for all the elements of [first2, last2). Finally, std::search returns the first of such occurrences.

It can be used in either of the two versions, as depicted below :

1. **For comparing elements using == :**

ForwardIterator1 search (ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2, ForwardIterator2 last2);

first1:
Forward iterator to beginning of first container to be searched into.

last1:
Forward iterator to end of first container to be searched into.

first2:
Forward iterator to the beginning of the subsequence of second container to be searched for.

last2:
Forward iterator to the ending of the subsequence of second container to be searched for.

Returns: *an iterator to the first element of the f*
irst occurrence of [first2, last2) in [first1, last1), or last1
if no occurrences are found.

val :
Value to be search in the range
Return Value :
An iterator to the first element in the range that compares equal to val.
If no elements match, the function returns last.

Program using find() Algorithm

```
#include<bits/stdc++.h>
 int main ()
{
   std::vector<int> vec { 10, 20, 30, 40 };

   // Iterator used to store the position
   // of searched element
   std::vector<int>::iterator it;

   // Print Original Vector
   std::cout << "Original vector :";
   for (int i=0; i<vec.size(); i++)
      std::cout << " " << vec[i];

   std::cout << "\n";

   // Element to be searched
   int ser = 30;

   // std::find function call
   it = std::find (vec.begin(), vec.end(), ser);
   if (it != vec.end())
   {
      std::cout << "Element " << ser <<" found at position : " ;
      std::cout << it - vec.begin() << " (counting from zero) \n" ;
   }
   else
      std::cout << "Element not found.\n\n";

   return 0; }
```

Search () Algorithm :

Program using search() Algorithm

Version 1:

```
// C++ program to demonstrate the use of std::search

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
   int i, j;

   // Declaring the sequence to be searched into
   vector<int> v1 = { 1, 2, 3, 4, 5, 6, 7 };

   // Declaring the subsequence to be searched for
   vector<int> v2 = { 3, 4, 5 };

   // Declaring an iterator for storing the returning pointer
   vector<int>::iterator i1;

   // Using std::search and storing the result in
   // iterator i1
   i1 = std::search(v1.begin(), v1.end(), v2.begin(), v2.end());

   // checking if iterator i1 contains end pointer of v1 or not
   if (i1 != v1.end()) {
      cout << "vector2 is present at index " << (i1 - v1.begin());
   } else {
      cout << "vector2 is not present in vector1";
   }

   return 0;
}
```

Version2 :

For comparison based on a predicate (or condition) :

ForwardIterator1 search (ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2, ForwardIterator2 last2, BinaryPredicate pred);

All the arguments are same as previous template, just one more argument is added

pred: *Binary function that accepts two elements as arguments (one of each of the two containers, in the same order), and returns a value convertible to bool. The returned value indicates whether the elements are considered to match in the context of this function. The function shall not modify any of its arguments. This can either be a function pointer or a function object.*

Returns: *an iterator, to the first element of the first occurrence of [first2, last2) satisfying a predicate, in [first1, last1), or last1 if no occurrences are found.*

```
// C++ program to demonstrate the use of std::search
// with binary predicate
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Defining the BinaryPredicate function
bool pred(int i, int j)
{
   if (i > j) {
      return 1;
   } else {
      return 0;
   }
}

int main()
{
   int i, j;

   // Declaring the sequence to be searched into
   vector<int> v1 = { 1, 2, 3, 4, 5, 6, 7 };
```

```
    // Declaring the subsequence to be compared to based
    // on predicate
    vector<int> v2 = { 3, 4, 5 };

    // Declaring an iterator for storing the returning pointer
    vector<int>::iterator i1;

    // Using std::search and storing the result in
    // iterator i1 based on predicate pred
    i1 = std::search(v1.begin(), v1.end(), v2.begin(), v2.end(), pred);

    // checking if iterator i1 contains end pointer of v1 or not
    if (i1 != v1.end()) {
        cout << "vector1 elements are greater than vector2 starting "
            << "from position " << (i1 - v1.begin());
    } else {
        cout << "vector1 elements are not greater than vector2 "
            << "elements consecutively.";
    }

    return 0;
}
```

Link : https://www.geeksforgeeks.org/sort-algorithms-the-c-standard-template-library-stl/