STL Containairs
(containers in STL)

containers are of 4 types

class1

1) Sequence containers - Stores data in a sequential manner
(Implement data structure which can be accessed in a sequential manner)

a) Vector

b) List (Doubly linked list)

c) deque (Doubly ended queue) (Insertion from both end)

d) array

e) Forward List (Singly linked list)

class2

2) Container adaptors :- These containers although have a underlined data structure that is sequential but you cannot access all the elements of a sequence.

Eg Stack → Stack is build using an array but the stack only gives the access to the top-most element.
(Provide a different interface for sequential containers)

a) queue

b) priority queue (heap (uses array))

c) stack

class3 Associative containers : that stores the data in a sorted manners.

Implement sorted data structure that can be quickly searched (O(log n)) complexity)

a) set

b) Multiset

c) map

d) multimap

class4 unordered Associative containers : where order doesn't matter when searching

(Implement unordered data structures that can be quickly searched).

a) Unordered set

b) unordered multiset

c) unordered map

d) unordered multimap

Algorithm (Inbuild Headerfile and function)

Stl for searching an element in an array (unsorted)

```
1) # include <algorithm>
   # include <iostream>
   using namespace std;
   int main()
   { int arr[] = {1, 10, 11, 9, 100}
     int n = sizeof(arr) / sizeof(int);
     // To search if an element is present or not in an array.

     int key;
     cin>> key;
     auto it = find (arr, arr+n, key);
     cout << it << endl; // return the address of the key
     int index = it - arr; // Key address - array's base add gives the pos of the key
     cout << index; // returns the position
   // If key is not present the index display the size of array.
        if (index == n)
        { cout << key <<"is not present"; }
        else
     return 0;
   }
```

here we can also write int* it in place of auto it

Stl for searching an element from sorted array

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

  int main ()
  { int a = {1, 2, 3};
  int n = 3;
     int k;
     cin >> k;

 bool present = binary_search (a, a+n, k);
 if (present) cout << "present" ;
 else   cout << "Absent";
```

To check if the element is present or not.

//2 more function.

// 1. lowerbound.

```
auto it = lower_bound (a, a+n, k);
```

// This function will return the first address of the first

// element that is $\boxed{>=}$ key.

// eg a = {1, 2, 2, 2, 2, 3, 4} if the key is 2 Lowerbound
(0 1 2 3 4 5 6)

// will return the address of the first 2.

```
cout << "\n" << " lower bound of key " << (it - arr);

auto it = upper_bound (a, a+n, k);
```

// This function returns strictly the no. which is strictly

// greater $\boxed{>}$ than key.

```
// Eg  a = { 1, 2, 2, 2, 2, 3 } , k = 2,  Ub = 4(index)
          0  1  2  3  4  5
// Lb = 1(st index).  Ub - Lb = no. of occurance of the element
   cout<<" upper bound of key is <<(it -a);
 cout<<" occurrence of key ="<<(Ub - lb);
```

String Class (C++ STL)

↳ One major advantage is it provides an alternative for character arrays

```
 # include <iostream>
 # include <string>
   Using namespace std;
   int main()
 {   // Initialisizing A string
 1>  string s0; // empty string
 2>  String s1("Hellow"); // O/P → Hello
 3>  string s2{ = "Hellow World!"; // Hello w world!
 4>  string s3(s2); // Hellow world!
 5>  string s4 = s3; // Hellow world!
 6>  char a[] = {'a', 'b', 'c', '\0'};
     string s5(a); // abc
// Empty functn
          if (s0.empty())
       { cout<<"s0 is empty string";
       }
```

// Append functh

```
1) S0. append (" I love C++");
   cout << S0 << endl; // I love C++
2) S0 + = ".and Python";
   cout << S0 << "\n"; // I love.C++.and.Python
```

// length functh and clear functh

```
Cout << S0.length() << "\n"; // 22
S0.clear(); // Erase the whole string.
cout << S0.length() << "\n"; // 0
```

// string compare function

S0 = ~~Mango~~ "Apple";
S1 = "Mango";

```
Cout << S1. compare (S2) << "\n"; // -12
cout << S2. compare (S1) << "\n"; // 12
cout << S1. compare (S1) << "\n"; // 0
```

// compare a functh returns an integer ==0 if both the strings are equal
// if 1st string is lexological smaller than the other string then return some (-ve) value
// if 1st string is larger then return (+ve) value

// Operator overloading

```
if (S1 > S0)
{ cout << "Mango is greater";
else
  cout << "Apple is greater";
```

// Accessing the ith character of a string

```
cout << s1[0] << "\n"; // A
```

// Find substring

```
string s = "I want to have apple juice";
            0 1 2 3 45 6 78 9 10 11 12 13 14 15|
int idx = s.find("apple");
cout << idx << "\n"; // O/P → 15 (returns starting
                        index of the substring in the
                        main string.
```

// Remove a word from the string

```
string word = "apple"
int len = word.length();
                len+1
s.erase(idx, ^~~length~~);
// I want to have juice
```

// Iterate over all the character in the string

```
for (i=0; i< s1.length(); i++)
{ cout << s1[i] << ":"; }

// O/P → M:a:n:g:o:
```

// Iterate with the iterator.

```
         string :: iterator it // auto can be replaced as.
for (auto it = s1.begin(); it != s1.end(); it++)
{ cout << (*it) << ","; }

// O/P → M,a,n,g,o,
```

→ as it prints the address thus, *it prints the value in that address.

// For Each loop

// auto itself detect the datatype of the variable

// char, string, int, float etc .

```
         char
for (auto c : s1)
{ cout << c << "." ; } // M.a.n.g.o
```

// This will itrate through loop ~~and~~ jst like for loop.

String Sorting

```
#include <iostream>
#include < algorithm>
#include <string>
using namespace std;
int main ()
{  int n;
   cin >> n;
    cin.get();
    string s[100];
   for (int i=0; i<n; i++)
   { getline(cin, s[i]); }
    for (int i=0; i<n; i++)
    { cout << s[i] << endl; }
      sort(s, s+n);
```