- STL:
    - ❖ provides the ready-made implementation of some frequently used data structures and algorithms.
    - ❖ Vector
    - ❖ Iterators
    - ❖ Sort
    - ❖ Pair, Struct
    - ❖ Set
    - ❖ Map
    - ❖ Unordered-Set
    - ❖ Unordered-Map
    - ❖ Multiset
    - ❖ Multimap
    - ❖ Stack
    - ❖ Queue
    - ❖ Deque
    - ❖ Priority Queue
    - ❖ Comparators

      #include<vector>

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

- **Vector**:

  -> Dynamic Array
  -> **vector<data-type> name;**
  -> vector<int>arr; // empty
  -> vector<int>arr(n) // size n
  -> vector<int>arr(n,0)

// size n, initialized with a default value;

-> Accessing an element : arr[0]..... arr[n-1]

-> **Functions:**

- **Size : arr.size();** -> TC : O(1)
- **arr.push_back();** -> TC : O(1)
- **arr.resize(n);** -> TC : O(n)
- **arr.pop_back();** -> TC : O(1)
- **arr.erase(pos); arr.erase(start_pos, end_pos);** -> TC : O(n)
- **arr.clear();** -> TC : O(n)
- **arr.empty()**; -> TC : O(1)
- **arr.front(); arr.back();**(reference to 1st and last elements respectively) O(1)

- **2D vector** :

```
->vector<vector<int>> vect
    {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
-> vector<vector<int>> vect
    {  {1},
       {4, 5, 6},
       {7, 8}
     };
-> vector<vector<int>> vec( n , vector<int> (m));
```

-> vector<vector<int>> vec( n , vector<int> (m,0));
-> vector<vector<int>>arr[n];

- **Iterators :**

->like a pointer that points to an element inside the container.
-> **vector<int> :: iterator it = v.begin();**
-> traversing vector using iterators
-> for(auto i : arr)cout<<i<< " ";

    //int can also be used
    // can change elements by using
    // for(auto &i : arr)i = 0;
    -> auto is used when we do not know the datatype

-> v.end() == v.begin()+n

- **Sort** :

-> TC : O(n log n)
-> default : ascending order
-> Syntax :

**sort(arr.begin() , arr.end());** // vector
.begin() -> starting iterator,  .end() -> ending iterator

**sort(arr,arr+n);** // array

- **Reverse:**

// reverse function logic
-> swap(a,b)

-> TC : O(n)
-> Syntax :

```
reverse(arr.begin() , arr.end());
```

- ## Pair :
  -> **pair<data_type, data_type> p;**
  -> accessing : p.first, p.second;
  -> ex : vector<pair<int,int>> arr;
  -> sorting vector/ array of pairs

- ## Struct :
  -> Custom / User defined Data types
  -> example :

```
  struct student {
        int rollNumber;
        string name;
        string branch;
  };

student  x ;
x.rollNumber =
.
.
```

vector<student> students(n); // vector

student students[n]; // array

Problem :
https://www.codechef.com/LRNDSA03/problems/DPAIRS

Code :

```
int n, m;
    cin>>n>>m;
    vector<pair<int,int>> a(n);
    for(int i=0;i<n;++i) {
        cin>>a[i].first;
        a[i].second=i;
    }

    vector<pair<int,int>> b(m);
    for(int j=0;j<m;++j) {
        cin>>b[j].first;
        b[j].second=j;
    }

    sort(a.begin(), a.end());
    sort(b.begin(), b.end());

    for(int j=0;j<m;++j) {
        cout<<a[0].second<<" "<<b[j].second<<"\n";
    }

    for(int i=1;i<n;++i) {
        cout<<a[i].second<<" "<<b[m-1].second<<"\n";
    }
```

- **Set :**

  -> stores unique elements in sorted order
  -> Based on red-black Tree.
  -> while adding elements in a set, duplicates are discarded
  -> **set<data_type> name; // Ex : set<int>s;**
  -> printing set elements
  -> **Functions:**
    - **Size : s.size();** -> TC : O(1)
    - **s.push();** -> TC : O(log n)
    - **s.erase(element); or s.erase(iterator);** TC : O(log n)
    - **s.empty()**; -> TC : O(1)
    - **s.find(element);** -> TC : O(log n)

  //How to iterate in a set using auto iterator btado.

**Problem : [https://codeforces.com/contest/903/problem/C](https://codeforces.com/contest/903/problem/C)**
**Code:**

```
int n;
    cin>>n;
    vector<int>a(n);
    for(int i =0 ; i<n ; i++)cin>>a[i];
    sort(a.begin(),a.end());
    int ans = 0 , temp = 1;
```

```
for(int i = 1; i<n ; i++){
    if(a[i]==a[i-1])temp++;
    else{
        ans = max( ans, temp);
        temp = 1;
    }
}
ans = max(ans,temp);
cout<<ans<<"\n";
```

- **Map:**

  -> elements are stored as key value pair
  -> keys are stored in ascending order
  -> No two mapped values can have same key
  -> example : roll number of student, name
  -> **map<data_type, data_type>m;**
  -> Ex : map<int, string>students;
  m[021] = “student1”;
  -> printing map elements (.first, .second , auto)
  -> **Functions:**

    - **Size : m.size();** -> TC : O(1)
    - **m.count(element);** -> TC : O(log n)
    - **m.empty()**; -> TC : O(1)
    - **m.erase(key);** TC : O(log n)

- ## Unordered Set:

  -> Due to its different internal implementation, All operations on the **unordered_set** takes constant time O(1) on an average ( worst case can go upto O(n))
  -> Based on hashing
  -> **unordered_set <data_type> name;**
  -> **Ex: unordered_map <int,int> m;**

- ## Unordered Map:

  -> Again, due to different implementation, on an average, the cost of search, insert and delete from the unordered_map is O(1)( worst case can go upto O(n))
  -> **unordered_map <data_type,data_type> name;**

- ## Multiset :

  -> same as set, but can have repeated values
  -> difference in erase operation
  -> **s.erase(element); // erases all instances of the element**
  -> **s.erase(iterator); // erases only 1 instance**

- ## Multimap :

  -> same as map but multiple elements can have the same keys.
  -> it is NOT required that the key value and mapped value pair has to be unique in this case.

-> One important thing to note about multimap is that multimap keeps all the keys in sorted order.

Problem : https://codeforces.com/contest/4/problem/C
Code :

```
int n;
 cin>>n;
unordered_map<string,int>m;
for(int i =0 ; i<n ; i++){
    string s;
    cin>>s;

    if(m.find(s) == m.end()){
        m[s]++;
        cout<<"OK"<<"\n";
    }
    else{
        cout<<s<<m[s]<<"\n";
        m[s]++;
    }
}
```

- **Stack:**
  -> Stack is a container which follows the **LIFO (Last In First Out)** order
  -> the elements are inserted and deleted from one end of the container.
  **-> stack<data_type>name; Ex : stack<char>st;**
  -> **Functions:**

- **Size : st.size();** -> TC : O(1)
- **st.empty()**; -> TC : O(1)
- **st.push();** -> TC : O(1)
- **st.top();** -> TC : O(1)
- **st.pop();** -> TC : O(1)

Problem :
you have given a string , when two adjacent character are same they get blast and remove from string
if finally after all possible blast if string is empty than return 1 else return 0

Code :
```
string ss;
        cin>>ss;
        stack<char>s;
        int n=ss.length();
        for(int i=0;i<n;i++)
        {
            if(!s.empty() && s.top()==ss[i])
            {
                s.pop();
            }
            else
            {
                s.push(ss[i]);
            }
        }
        if(s.empty()) cout<<0;
        else cout<<1;
```

- ## Queue :

  -> Queue is a container which follows **FIFO order (First In First Out)** .
  -> Here elements are inserted at one end (rear ) and extracted from another end(front)
  -> **queue<data_type> name; // Ex : queue<int>qu;**
  -> **Functions:**
    - **Size : qu.size();** -> TC : O(1)
    - **qu.empty()**; -> TC : O(1)
    - **qu.push();** -> TC : O(1)
    - **qu.front();** -> TC : O(1)
    - **qu.pop();** -> TC : O(1)

- ## Deque:

  -> Insertion and deletion happens on both ends.
  -> **deque<data_type>name; // Ex: deque<int>dq;**
  -> **Functions:**
    - **Size : dq.size();** -> TC : O(1)
    - **dq.empty()**; -> TC : O(1)
    - **dq.push_back();** -> TC : O(1)
    - **dq.pop_back();** -> TC : O(1)
    - **dq.push_front();** -> TC : O(1)
    - **dq.pop_front();** -> TC : O(1)
    - **dq.front();** -> TC : O(1)
    - **dq.back();** -> TC : O(1)

https://codeforces.com/contest/1579/problem/E1

- **Priority Queue :**

  -> A priority queue is a container that provides constant time extraction of the (largest or smallest) element, at the expense of logarithmic insertion
  -> **priority_queue<int> pq; // max priority queue**
  -> **priority_queue <int,vector<int>,greater<int>> pq // min**
  -> **Functions:**
    - **Size : pq.size();** -> TC : O(1)
    - **pq.empty()**; -> TC : O(1)
    - **pq.push();** -> TC : O(log n)
    - **pq.top();** -> TC : O(1)
    - **pq.pop();** -> TC : O(log n)

- **Comparators :**

  -> for sort

```
//descending
bool comp(int a, int b){
        return a>b;
}

sort(arr, arr+n, comp);
```

Or

```
sort(a.begin(),a.end(),[](int a, int b){ return a > b; });
```

-> for STL containers :

```
struct compare {
    bool operator() (const pair<int,int>& a, const pair<int,int>& b)
{
         if (a.first > b.first)
            return true;
        else
            return false;

    }
};



 set<pair<int, int>, compare> s;

//priority_queue<int,vector<int>, comp>pq;
```

Additional Resources :

https://www.topcoder.com/thrive/articles/Power%20up%20C++%20with%20the%20Standard%20Template%20Library%20Part%20One

https://www.topcoder.com/thrive/articles/Power%20up%20C++%20with%20the%20Standard%20Template%20Library%20Part%20Two:%20Advanced%20Uses

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