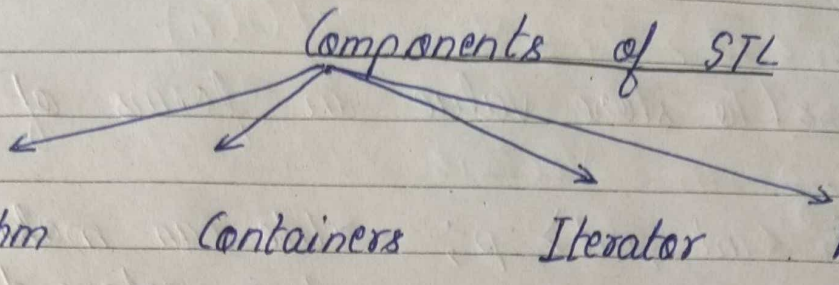
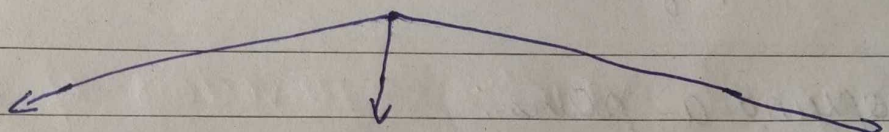
STL
(Standard Template Library)

Khushboo Shakya



1) Algorithms:- are the methods or functions that act on containers. ~~By~~ ~~us~~
Example Upper bound, Lower bound, max-element, min element etc

2) Containers:- collection of objects of a particul type of data structure.



Sequence Containers
- Vector
- Deque
- list

Associative Containers
- Set
- Multiset
- Map
- multimap

Derived Containe
- Stack
- Queue
- Priorit Queue

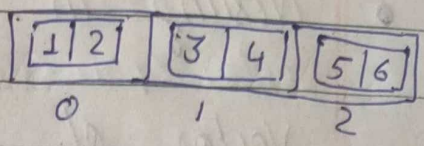
3) Iterators:-
- Point to memory address of containers
- begin(), end()
- vector<int> :: iterators it;

4) Functors:-
- Classes which can act as functions.

- pairs of arrays:

```
pair <int , int> arr[ ]={{1,2},{3,4},{5,6}};

cout << arr.[1]. second;
```



O/P → 4

2) Vectors:

Array doesn't provide dynamic allocation. After declaring the size, the size can't be increased or decreased but in vector we can change.

```
vector <int> v;
v. pop_back ();  → Removing element from the end of vector
v. push_back (1);  → Insert element at the end of vector
                  → O[1]
```

- Declaring a particular size of vector

```
• vector <int> v(5);   //O/P → 0 0 0 0 0
                              size → 5

• vector <int> v(5);
  v. push_back(7);     //O/P → 0 0 0 0 0 7
                              size → 6

• vector<int> v(10,3);   //O/P → 3 3 3 3 3 3 3 3 3 3 7
  v.push_back(7);              size → 11
```

- Copying a vector
  `vector<int> v2 = v;` //O(n)
- Nesting in Vectors

  Vector of pairs

  `vector <pair <int, int>> v;`
- Difference between array of vector and vector of vector:

  Array of vectors:-

  `vector<int> v[10];` → 10 vectors of zero size

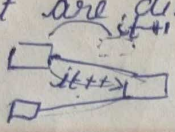
  Array of vectors is two dimensional array with fixed no. of rows where each row is vector of variable length.

Vectors of vectors:-
It is a 2D vector with a viriable no. of rows where each row is vector.

`vector<vector<data_type>> vec;` → Syntax of declaration

`vector_name.push_back(value);` → Insertion Syntax

map or set are discontinuate in nature that's it++ doesn't work

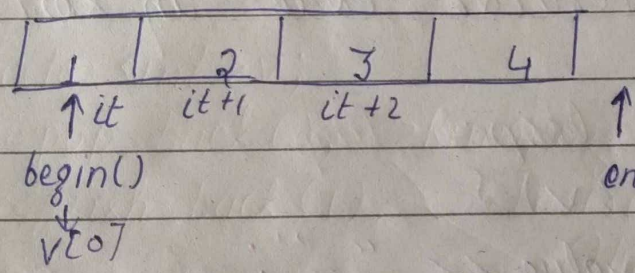


Iterators

Iterators are used to point at the memory addresses of STL containers.

vector <int> :: iterator it; ⟶ syntax for declaring iterators

- Basic member functions that allow to iterate
  - begin() = Return an iterator pointing to the first element
  - end() = ~~The~~ " " " pointing to the past the last element



- vector <int> :: iterator it = v.begin();
  cout << (*(it+1)) << endl;

  o/p → 2

- it++ → next iterator
  it+1 → next location

- Vector of pair in iterator:-

```
vector<pair<int, int>> v_p={{1,2},{2,3}};
                                  Declaration
vector <pair <int, int >> :: iterator it;

for (it = v_p.begin(); it != v_p.end(); ++it{
   cout <<(*it).first<< " "<< (*it).second <
   }

//(*it).first ⇔ (it→first)
```

same

- How to write iterator code in sort.

- Range based for loop:-

```
for(int value : v) {
    cout << value << " ";
    }
```

vector se jo value ayegi wo copy aayegi and if we try to change any value then we will use refrence value(&value)

- Auto:- Automatically converts the given datatype

```
for (auto &value: v_p) {
   cout <value.first << " " << value.second
   }
```

The map is implemented as a balanced tree structure

Date:
Page No.

Maps

Store elements in key & value pair.

key	value
{int	string}
1	abc
5	cde
3	acd

map<int, int> m; → declaration

m[1] = 2; → initialisie value
or
m.insert({4, "25"}); // O(log(n))

Note • Map store value in sorted order.

m[4] = 2
m[1] = 3

O/P :- 1 3
4 2

If it's string then it will store in a lexicographical ordering (decitionary order or ascii o

• Default value for doube, float, int is zero and for string, vector defaul value will be empty string.

m[1] = 2
m[3]

O/P → 2
0

- Key is unique.

```
m[2] = 7
m[3] = 9
m[2] = 8    // 8 will print correspond to 7
```

O/P
```
9
8
```

- m.find(value); ⟶ Finding // O(log(n))
- m.erase(value); ⟶ For removing the particular iterator // O(log(n))
- m.clear(); ⟶ For clearing a map

Unordered Map

Stored elements or given key value pair in unsorted manner. But collisions could take place.

unordered_map <int, string> m; ⟶ declaration

- It is implemented using hash Table.
- Time complexity = O(1) on average

Multimap :-

Stores a given key twice or more times (multiple key value pair)

SET, Unordered set, Multiset

Set

Stored element in a sorted order and unique elements only

syntax: set<datatype> setname;

- Set follows Binary Search implementation

Unordered Set

Same functionality as set

1) Stores unique element
2) doesn't store in sorted order
3) lower bound and upper bound doesn't applicable

Rest all function is same as set

Multiset

Same as set but allows to store duplicate elements (but in sorted order)

```
multiset < int > ms;
    ms.insert(1);    // {1}
    ms.insert(1);    // {1 1}
    ms.insert(1);    // {1 1 1}
```

```
ms.erase(1);   // All the 1's get erased
```

Only a single one erased

```
ms.erase(ms.find(1));
ms.erase(ms.find(1), ms.find(1)+1);
```

After 1's

Nesting in Maps and set

We use combined maps, set, pair and vector in nesting.

For example

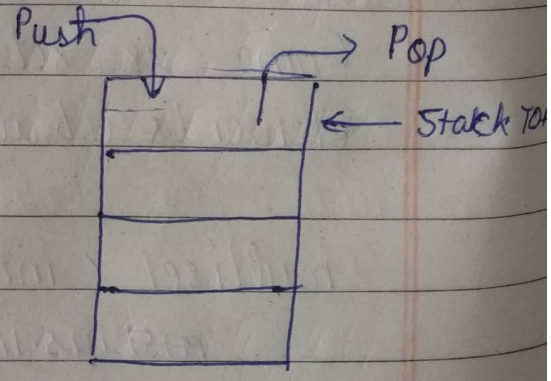
```
map<sets<int>, int> m;
set<pair<int, string>> s;
map<int, set<int>>m;
```

We don't use nesting in unordered map or unordered set because complex data structured as a key doesn't work on these.

Stacks and Queues

Stack :- Last in first Out (LIFO)

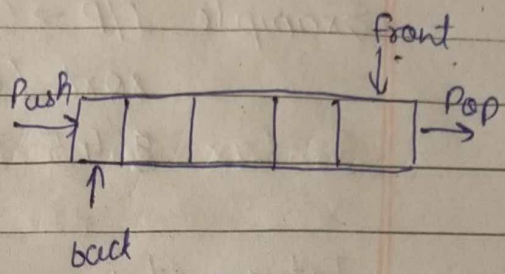
```
stack<int>st;
st.push(1);    // {1}
st.push(2);    // {2,1}
st.push(3);    // {3,2,1}
st.push(3)     // {3,3,2,1}
cout<<st.top();
//prints 3
```



```
cout << st.size(); //4
cout << st.empty(); // Stack is empty or not
```

It doesn't have iterators like begin, end, erase etc

Queue → FIFO (First in first out)



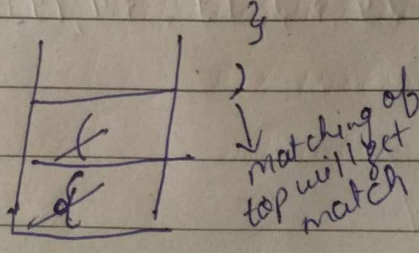
- Same as stack
- Limited functionalities
- Dynamic in size

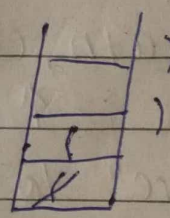
```
queue<int>q;
q.push(1);      // {1}
q.push(2);      // {1,2}
cout<<q.back(); // 2
cout<<q.front(); // 1
```
{q.top doesn't work}

Balanced Brackets Matching

A balanced bracket sequence is a string consisting of only brackets, such that this sequence when inserted certain number and mathematical operations, gives a valid mathmatics expression.

{()} ✓ ())(✗





Inbuilt Sort

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

- arr: address of begining of array
- arr+n: address where we want to end the sorting

Example I/P → 6, 4, 5, 2, 25, 7, 8

sort (a+2, a+n)

2 4 5 2 7 8 25

Introsort -

- best sorting algorithm
- Hybrid sorting algorithm (use more than one sorting algorithms) i.e. Quicksort, Heapsort and Insertion sort →

O(N logN)

- vector <int> a(n);
  sort (a.begin(), a.end()); → Syntax

Comparators

```
bool comp( pair<int, int>p1, pair<int, int>p2);
if (p1.second < p2.second) { return true; }
else if (p1.second == p2.second) { If (p1.first > p2.se
   return true} return false
```

```
pair <int, int> a[] = {{1,2}, {2,1}, {4,1}};
sort (a, a+n, comp);
```

↑

Sort it acc^n to 2^n element

U.B and L.B return location in array and return iterator in vector

{{4,1}, {2,1}, {1,2}}

1) If second element is same, then sort it according to the first element but in descending order.

Upper bound and Lower bound :-

Upperbound :-

Element present or element not present → return next bigger element

Example

4 5 5 7 8 25

U.B of 4 = 4 5 (5) 7 8 25

" " 6 = 4 5 5 (7) 8 25

Syntax is same as lower_bound (lower_bound, upper_bound)

Lower bound :-

- Arrays and vectors should be sorted If we are applying lower bound and upper bound. Other wise It will not work on log(n) complexity. It will work on O(n).

- Element present → return Same value
  Element not present → return next bigger element

Example

4 5 5 7 8 25

lower bound of 5 = 4 (5) 5 7 8 25

" " 6 = 4 5 5 (7) 8 25

" " 26 = 4 5 5 7 8 25 ( )

Syntax in array:

```
int *ptr = lower_bound(a, a+n, 5);
```

Starting address — address of last element — Element of which we have to find the lower bound

Time complexity of this function:
$O(\log N)$

Syntax in vector

```
auto it = upper_bound(a.begin(), a.end(), 5);
```

$O(\log N)$

Syntax in map and set:

```
auto it = upper_bound(a.begin(), a.end(), 5);
```

✗ preferred not to use this in set and map because it is $O(n)$ TC

```
auto it = s.lower_bound(rand());
```

✓ $O(\log N)$

Inbuilt Algorithms in STL

- min_element(v.begin(), v.end()) → return minimum iterator
- max_element(v.begin(), v.end()) → return max iterator
- accumulate(v.begin(), v.end(), 0) → return sum of vec

accumulate = Initial Sum + vector sum

- count(v.begin(), v.end(), 6) → return the count in the function

Jish value ka count nikalna ha

- find(v.begin(), v.end(), ?) → Finding the value
- reverse(v.begin(), v.end()) → reversing the string, vector, array.
- Lambda function :- Syntax of writing temporary funtion.

f (n)

Syntax `[](int x){return x+2;}(2)`

- `[]` : that we need to pass
- `(int x)` : variable
- `return x+2;` : what we need to return
- `(2)` : pass

It's same as

~~for(int x=2; i~~

```
int x = 2;
    sum = x + 2;
    cout(sum);
```

difference

All_of	Any_of	None_of
Sabhi ~~positive~~ element true ~~return~~ True	Ek bhi element true ~~return~~ True	Koi bhi element true nahi ~~return~~ true

Sum of two no using assigning :-

```
auto sum = [](int x, int y) {return x+y;}(4, 7)
           // 11
```

↑ we can reuse this function

- all_of Algorithms :- This function save time to ~~re~~ run a loop to check ~~el~~ each elements one by one

```
all_of(v.begin(), v.end(),
       [](int(x) {return x>0;});
```

lambda function

above all of function is same as this :

```
bool is_positive(int x) {
        return x>0;
    }
cout << all_of(v.begin(), v.end(),
        is_positive);
```

- any_of Algorithms :-

```
any_of(v.begin(), v.end(),
       [](int &x) {return x>0;});
```

- none_of Algorithms :-

```
none_of(v.begin(), v.end(),
        [](int x) {return x>0;});
```