July 05, 2025

STL APPLICATION 1

* Limitations of array
  - → size is fixed.
  - → Non-resizable.

* STL : Standard Template Library

1. Vector
   - → dynamic array
   - → size can grow
   - → indexing is available.
   - → acts as container

// Vector in programs :

- #include <vector> : header required to use vectors in program.

  Or,

  If you are using #include <bits/stdc++.h> then it already contains <vector> and isn't needed to define separately.

- Declaration of vector

  vector < type > v (uniformity of homogenous elements is maintained)

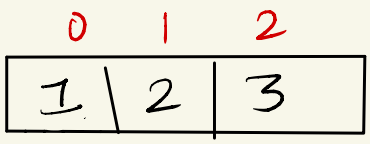
  ↳ This declares an empty vector.

  Ex:

  vector <int> v, vector <string> v

- Insertion : v.pushback(1) //append 1 at the end of vector

Ex: v.pushback(1); //O(1)
v.pushback(2);
v.pushback(3);



- Output the vector using indexing

```
for(int i=0; i<v.size(); i++)
    cout << v[i];
```

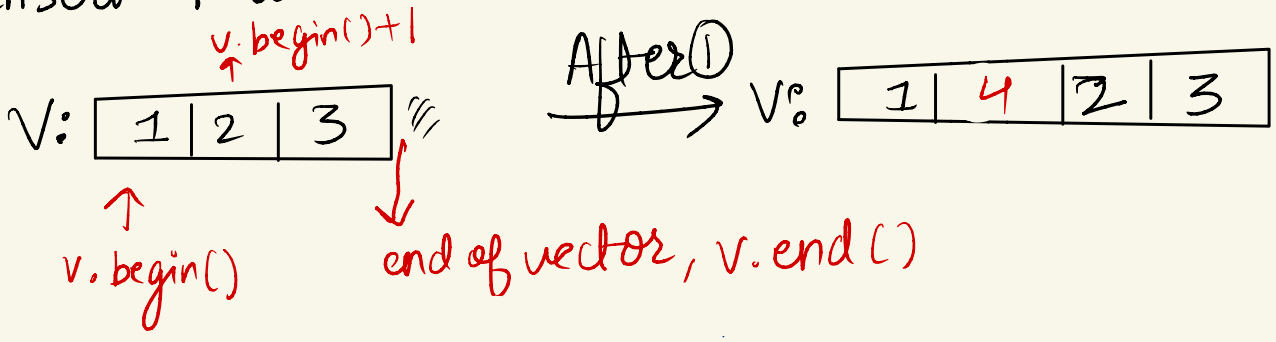
↳ gives the size of vector.

Another way : carries copy

```
for(int num : v)
    cout << num;
```

Both gives similar output.

- Insert 4 at 1 index. // O(N)

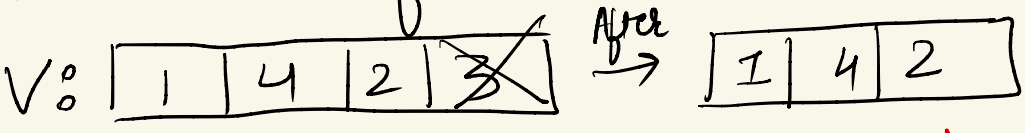


① ⟹ v.insert(v.begin()+1, 4);

↳ element that will be inserted at v.begin()+1

↑ address of the index where 4 has to be inserted

- Delete last element from the v. // O(1)



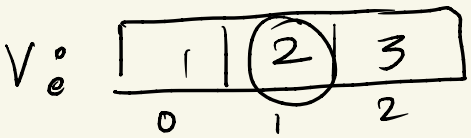
⟹ v.popback(); // remove the last element of the vector.

- v.insert( ) will always increase the size of vector by 1 even if you try to access an address that is outside v.size(). It will simply add a garbage value in the end. Better to make sure the address is valid.

- Remove an element from v. // O(N) → v.size()

v.erase(v.begin()+1); // erases v[1] i.e 4.

- Find element in vector // O(N) → v.size()
  ↳ returns an iterator address, so have to dereference it.



Find 2 in the range of v.begin() (start) to v.end() (end)

⇒ auto it = find(v.begin(), v.end(), 2)

auto: automatically assigns the data type

cout << *it; // returns the value 2

cout << it - v.begin(); // returns the index 1.

→ If we are looking for an element that is outside the vector then the above output will result in v.end()

- Search in vector. 1 4 3 2
  ↳ vector should be sorted first with sort()
  ⇒ sort(v.begin(), v.end())

After sort()

	0	1	2	3
v:	1	2	3	4

Now, we can use search fxn.

⇒ auto it = binary_search(v.begin(), v.end(), 4);

↳ will hold a boolean value here.

cout << it // will result 1, i.e the value 4 is present in the vector.

- Occurence of Duplicates in sorted vectors

	0	1	2	3	4	5	6	7	8	9	10
	1	1	1	2	2	2	3	3	3	4	4

# of 1's = 3 # of 2's = 3 # of 3's = 3

# of 4's = 2

use of lowerbound & upperbound

↓

will return iterator address of given element (first address in case of duplicates)

⇒ int idx = lower_bound(v.begin(), v.end(), 2) - v.begin();

cout << idx; // 3 (idx) → first occurrence of 2

⇒ int idx2 = upper_bound(v.begin(), v.end(), 2) - v.begin();

cout << idx2; // 6 (idx) → first occurrence of x > 2.

↓

element 3 at idx 6.

⇒ Total number of occurrences of element x in a sorted vector.

upper_bound - lower_bound

⇒ // find occurence of 2.

cout << upper_bound(v.begin(), v.end(), 2) - v.lower_bound(v.begin, v.end, 2)

↑ substract

// will count the occurrence of 2 and the output will be 3.

- Print sum of all elements in a veetor.

⇒ accumulate(v.begin(), v.end(), 0)

↳ will add here

- Max ele of vector

⇒ max_element : returns an address.

* max_element(v.begin(), v.end())

↳ using dereferencing operator, we can access the max element.

- Declaration of vector of a size

⇒ vector<int> v(10, 1)

default size — it can still grow or shrink.

↳ fill the vector with value 1

1	1	1	1	1	1	1	1	1	1

2 dimension vector with value 1.

⇒ `vector<vector<int>> mat(3, vector<int>(3,1));`

declaration of vector of vector

mat =

1	1	1
1	1	1
1	1	1