STL

vector

Saurabh Shukla (MySirG)

Agenda

1. Vector
2. Creating vector object
3. Accessing vector elements
4. Methods of vector

vector

- vector is a sequential container
- vector allows random access via []
- vector is implemented using dynamic array data structure
- The header for the vector is <vector>

How to create vector object?

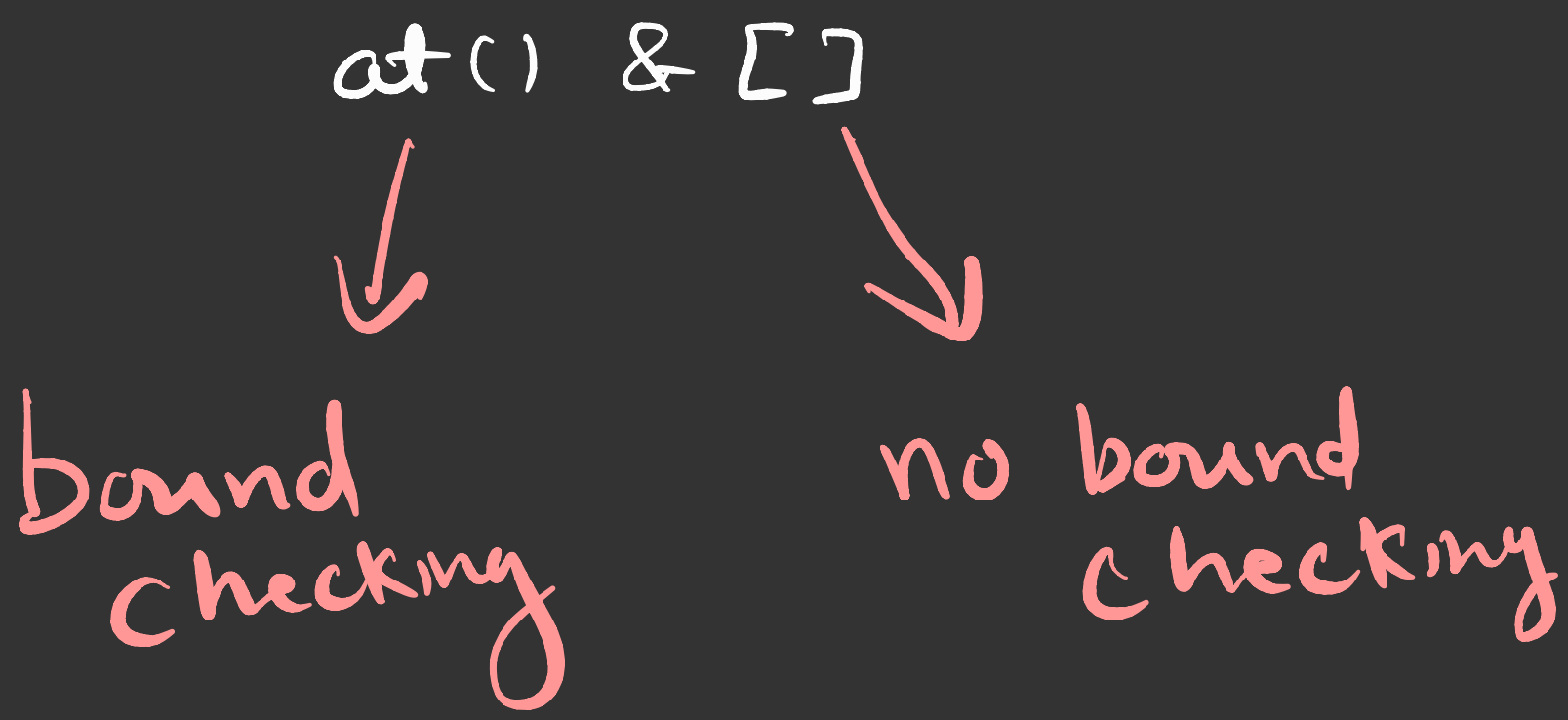
```
vector <int> v1;
vector <int> v2 = {10, 30, 50, 40};
```

capacity is 0

Accessing vector elements

You can access vector in the variety of ways.

1. [] without bound checking
2. at with bound checking
3. implicit iterator
4. explicit iterator


at() & []
bound checking
no bound checking

Implicit Iterator

```
vector <int> v1 = {10, 20, 30, 40};

for ( int x : v1 )
    cout<< x <<" ";
```

or

```
for (auto x :v1)
    cout << x <<" ";
```

Explicit iterator

You can get an iterator object from the following members of array.

①	begin()	end()	iterator
②	cbegin()	cend()	const_iterator
③	rbegin()	rend()	reverse_iterator
④	crbegin()	crend()	const_reverse_iterator

```
vector<int>::iterator it;
it = v1.begin();
```

0	1	2	3	4
10	50	20	40	30

```
it = it + 1
it++
it--
it = it - 2
```

Random access iterator

bidirectional

forward iterator

it ± i

Random access

Explicit Iterator

```
vector <int> v1 = { 50, 40, 10, 70, 60 };
vector<int> :: iterator it;

for (it = v1.begin() ; it != v1.end() ; it++)
        cout << *it << " ";

vector<int> :: const_iterator it;

for (it = v1.cbegin() ; it != .cend() ; it++)
        cout << *it << " ";
```

Explicit Iterator

```
vector <int > v1 = { 50, 40, 10, 70, 60 };
vector <int > :: reverse_iterator it ;
for (it = v1.rbegin() ; it != v1.rend() ; it++)
        cout << * it << " " ;

vector<int > :: const_reverse_iterator it ;
for (it = v1.crbegin() ; it != v1.crend() ; it++)
        cout << * it << " " ;
```

Methods of vector class

at()	
[]	
back()	returns last element
front()	returns first element
empty()	returns true or false
size()	returns the number of elements
swap()	swap elements of two vectors
clear()	erases all the elements
capacity()	returns the capacity of vector

push_back()

pop_back()

erase()

emplace() ⟶ better for non-primitive

emplace_back()

insert() insert one or many elements

shrink_to_fit()

insert single element

```
v1.insert(it, element);
```

insert same element multiple times

```
v1.insert(it, frequency, element);
```

insert few elements

```
v1.insert(it, {e1, e2, e3, ...});
```

Copy first 5 element of v1 into v2

```
v2.insert(v2.begin(), v1.begin(), v1.begin()+5)
```

inclusive

exclusive

add value at last.

```
v1.push_back(element);
```

emplace vs insert

Both are used to add elements in vector.

emplace avoids unnecessary copy of object, thus it is efficient

For primitives it doesn't matter which function you use to add elements in vector but for objects, use emplace() for efficiency.

erase

Erase an element of vector

```
v1.erase(it);
```

Erase range of elements of vector

```
v1.erase(it1, it2)
```

it1 ↑ inclusive

it2 ↑ exclusive