

C, C++, DSA in depth

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;  capacity is 0

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

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 | <code>begin()</code> | <code>end()</code> | <code>iterator</code> |
| ② | <code>cbegin()</code> | <code>cend()</code> | <code>const_iterator</code> |
| ③ | <code>rbegin()</code> | <code>rend()</code> | <code>reverse_iterator</code> |
| ④ | <code>crbegin()</code> | <code>crend()</code> | <code>const_reverse_iterator</code> |

```
vector<int> :: iterator it;
```

```
it = vi.begin();
```

0	1	2	3	4
10	50	20	40	30

```
it=it+1
```

Random access iterator

```
it++
```

bidirectional

```
it--
```

forward iterator

```
it=it-2
```

it + i

Random access

Explicit Iterator

```
vector <int> vi = { 50, 40, 10, 70, 60 };
```

```
vector<int> :: iterator it;
```

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

```
vector<int> :: const_iterator it;
```

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

Explicit Iterator

```
vector <int> vi = { 50, 40, 10, 70, 60 };
```

```
vector <int> :: reverse_iterator it;
```

```
for (it = vi.rbegin(); it != vi.rend(); it++)
```

```
cout << *it << " ";
```

```
vector<int> :: const_reverse_iterator it;
```

```
for (it = vi.cbegin(); it != vi.cend(); 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()`

`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 elements 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

vi. `erase (it);`

Erase range of elements of vector

vi. `erase (it1 , it2)`

↑
inclusive ↑ exclusive