The C++ Standard Template Library

Srinidhi Ayyagari and Drishti Gupta

#HourOfCode   #CSforGood

Agenda:

- ➔ Intro
- ➔ Vectors
- ➔ Iterators
- ➔ Other interesting Operations
- ➔ Pairs
- ➔ Maps and Multi Maps
- ➔ Priority queues
- ➔ Practice!

Standard Template Library:

It’s Magic!

Intro

Containers or container classes store objects and data.

eg) #include<vector>, #include<queue>, #include<stack>, #include<utility>

The *header* <algorithm> defines a collection of functions/algos especially designed to be used on ranges of elements, and act on containers.

#include<algorithms>

Never Forget to #include your header files.

Or simply include bits/stdc++.h

Vectors

Vectors: Initializing 1d vectors

- Vectors = Dynamic Arrays.
- No need to allocate memory while declaration, it has the ability to resize itself automatically when an element is inserted or deleted.
- Initializing a 1D vector

  vector<*datatype*> vector_name

  or

  vector<*datatype*> vector_name(size, init value);   //works exactly like an array.
- Iterating on a Vector

  vector_name. ***begin()***: Returns an iterator pointing to the first element in the vector.

  vector_name. ***end()***: Returns an iterator pointing to the last element in the vector

- Adding, Removing Elements

  vector_name. ***push_back()***: Pushes or Adds elements to a vector.
  vector_name. ***pop_back()***: pops or removes elements from a vector from the back.

- Accessing Elements

  vector_name. ***front()***: Returns a reference to the first element in the vector
  vector_name. ***back()***: Returns a reference to the last element in the vector

- Size of a vector

  vector_name. ***size()***: Returns the number of elements in the vector.

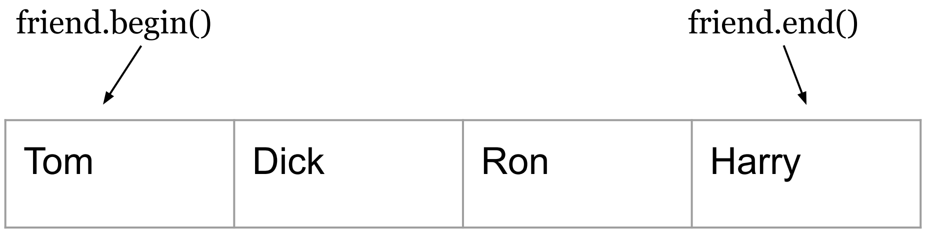
  vector_name. ***capacity()***: Returns the size of the space currently allocated to vector.

- Insert, erase elements from a particular index

  vector_name. ***insert(****position, element****)***: Inserts new elements before the specified position
  vector_name. ***erase(****position****)***: Removes element from the specified position.

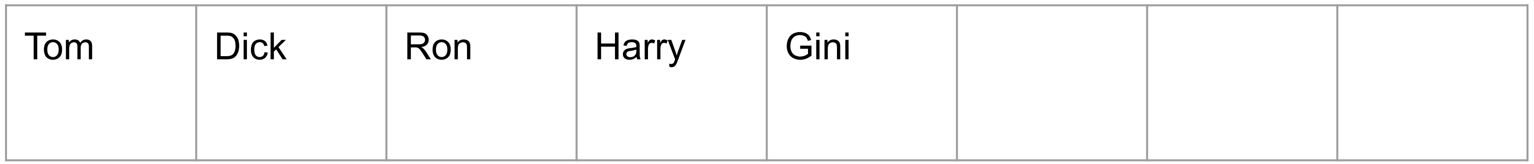
```
vector<string> friend;

friend.push_back("Tom");
friend.push_back("Dick");
friend.push_back("Ron");
friend.push_back("Harry");
```



Size = 4
Capacity = 4

```
friend.push_back("Gini");
```

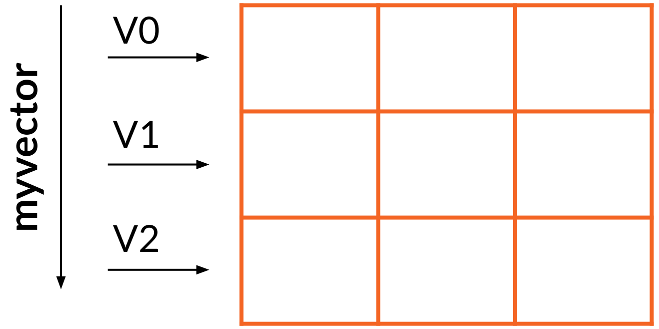


Size = 5
Capacity = 8

Vectors: Initializing 2d vectors

2D vector= 2D grid, or a *vector of vectors.*

$\texttt{vector<}\underbrace{\texttt{vector<}\underbrace{\textit{int}\texttt{>}}\texttt{>}}\texttt{ myvector;}$



- Operating on Elements of a Row

  myvector[i]. ***function()***: Operates on a Row as a Vector.
  Eg.
  myvector[i]. ***push_back()***: Adds element to ith Row of Vector

Vectors: Sorting

sort(*myvector*.begin( ), *myvector*.end( ));
// Sorts in Ascending Order by Default.

sort(*myvector*.begin( ), *myvector*.end( ), greater<*datatype_of_vector*>( ));
//Sorts in descending order

Sorting 2D vector:

sort(*myvector[i]*.begin( ), *myvector[i]*.end( ));
//sorts 1 row in ascending order

sort(*myvector*.begin( ), *myvector*.end( ), *sortbycol*);
//sorts the vector by columns, based on the *sortbycol* function defined by you.

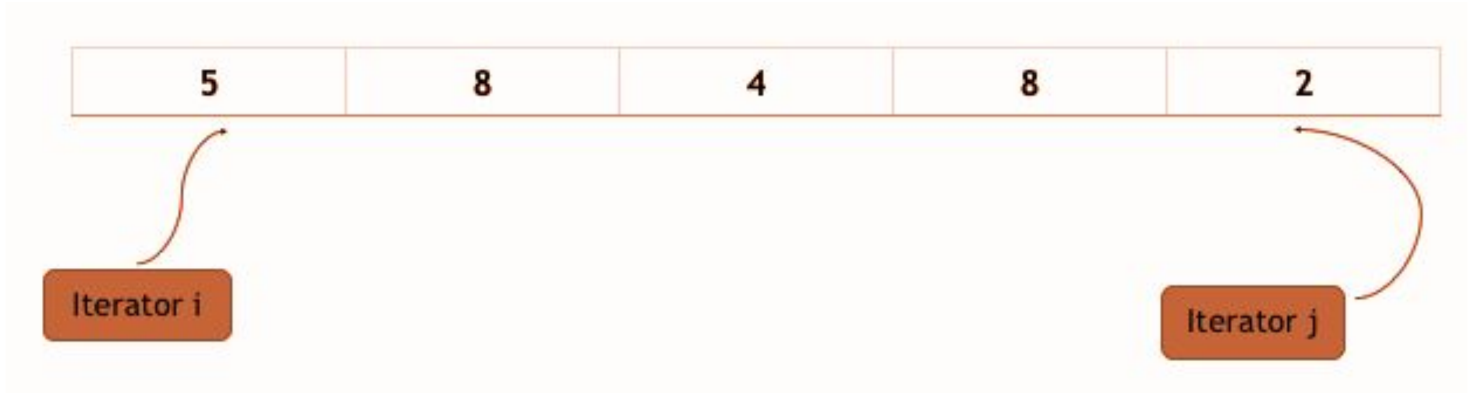
Let us try sorting a 2D vector by a Row now!

Iterators

Iterators: what are they?

- Iterators provide means for accessing data stored in container classes.
- Algorithms in STL work on iterators, not on the containers.

Example:
Here, there are two iterators, i and j. And i points to the beginning of the vector and j points to the end of the vector.



Syntax:

containerName **<templateParameters>** :: iterator iteratorName;

Example:

vector**<int>** :: iterator itr0;

map**<int,int>** :: iterator itr1;

Let declare a vector and an iterator,

vector**<int>** myvector;

vector**<int>** :: iterator i;

myvector**.begin()** returns an iterator to the first element of the vector myvector.

i=myvector**.begin();**

myvector.end() returns an iterator to the *past-the-end* element in the vector or the theoretical element that would follow the last element .

Dereferencing:

We can dereference the iterator to get the value of the element, it is pointing to.

```
vector<int>  myvector {9, 8, 7, 6, 10, 9};
vector<int> :: iterator itr;
for (itr=myvector.begin();   itr!=myvector.end();  ++myitr)
cout<<*itr<<" ";
```

What's ^ its output?

Tip

Read about front(), rend(), rbegin() and more !

Other Important/ Interesting operations

Other important Operations:

- Reverse a vector

  reverse (myvector.begin(), myvector.end());

- Find the maximum element for a vector

  cout << *__max_element__(myvector.begin(), myvector.end());

- Count the occurences of say x in a vector

  cout << **count**(myvector.begin(), myvector.end(), x);

Tip

Read about min_element(), distance(), lower_bound() !

Other important Operations:

- Finding an element, say x, in a vector

```
find(myvector.begin() , myvector.end(), x);
```

- Binary Search an element,say x, in a vector

```
if (binary_search(arr.begin(), arr.end(), x))
        cout << "x exists in vector";
```

- Upper Bound: This returns an iterator pointing to the first element in the range **[first,last)** which has a value greater than ‘x’

```
upper_bound(myvector.begin(), myvector.end(), x);
```

Other interesting Operations:

- Get the common elements of arr1 and arr2 of sizes n and m respectively,

```
itr = set_intersection( arr1, arr1+ n, arr2, arr2+ m, v.begin());
for (itr1 = v.begin(); itr1!= itr; ++itr1)
      cout <<" "<< *itr1; //prints the elements of v
```

- Let two character arrays, arr1 and arr2 of sizes n and m respectively, to know which character array occurs alphabetically first,

```
if( lexicographical_compare( arr1, arr1+n, arr2, arr2+m) )
      cout << "arr1 is lexicographically less than arr2";
```

- Get the summation of the vector elements, beginning the sum from x

```
cout << accumulate(myvector.begin(), myvector.end(), x);
```

Pairs

Pairs

- **“Pair”** is used to *combine* two values which may or may not be different in type.
- It is defined in **<utility>** header. // #include<utility>
- The first element is referenced as **‘first’** and the second element as **‘second’** and the order is fixed (first, second).
- The array of objects allocated in a map or hash_map are of type ‘pair’ by default in which all the ‘first’ elements are unique keys associated with their ‘second’ value objects.
- To access the elements, use variable name followed by dot operator followed by the keyword **first** or **second**.

Pairs Syntax

- **Declaration**
  pair <*data_type1*, *data_type2*> pair_name;

- **Accessing**
  mypair.first
  mypair.second

Pairs Example
Price and Fruit names

pair<*int*, *string*> fruits;

$$\underbrace{\begin{matrix}< 100\,, \\ < 80, \\ < 200,\end{matrix}}_{\text{fruits.first}}\ \underbrace{\begin{matrix}\text{Apple} > \\ \text{Banana} > \\ \text{Kiwi} >\end{matrix}}_{\text{fruits.second}}$$

Maps and Multi Maps

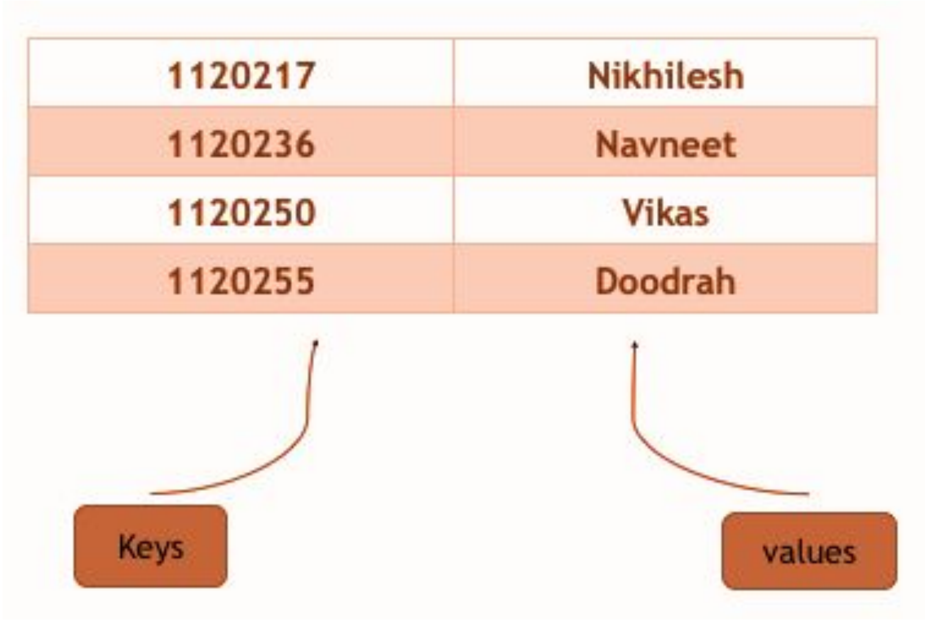
Maps

Maps are associative containers.

- Store elements as <u>sorted</u> *'key-value'* pairs
- The *key* is unique and cannot be altered
- The *values* associated with *keys* can be altered

Example:

A graphical representation of a map of students with
Key: Roll Number
Value: Student Name



1120217	Nikhilesh
1120236	Navneet
1120250	Vikas
1120255	Doodrah

Maps

Declaration

map **<keyType, valueType>** mapName;

Inserting

mapName[“**someKey**”]= **itsValue**;

Deleting

mapName.erase(**keyToBeDeleted**);

Accessing values through keys

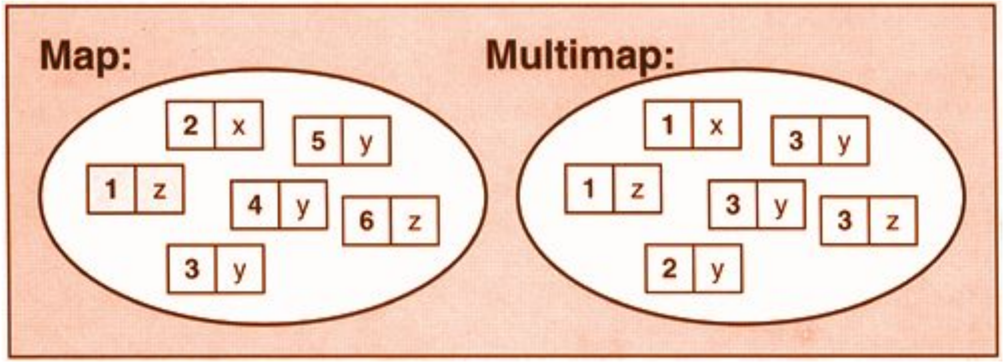
mapName[“**keyToAccess**”]; *or* mapName.at(**keyToAccess**);

Finding

auto itr = mapName.find(**keyToFind**);

Multi Maps

- Multimap is similar to map with an addition that multiple elements can have same keys.
- It is NOT required that the key value and mapped value pair has to be unique in this case.
- It keeps all the keys in sorted order always.



Tip

Read about unordered_map and unordered_multimap !

Priority Queues

Priority Queues

- Priority queues are data structures designed such that each element of the queue has a priority {fixed order}.
- Elements are arranged in either ascending or descending order, irrespective of the order they are pushed into.
- By default, **priority_queue** container is a ***max-heap***, i.e. elements are in non-increasing order.

Initialization

- **Priority_queue** <*data_type*> ***queue_name*** *//max-heap by default*
- **Priority_queue** <*data_type, vector<data_type>, greater<data_type>* > ***queue_name***
  To create min-heap or arranging elements in non-decreasing order

Operations in Priority Queues

- Adding, Removing Elements

  pq. ***push(****element****)***: Pushes or Adds elements to a priority queue and places it in order.

  pq. ***pop()***: pops or removes elements from the end of a queue.
  i.e. pop() will remove smallest element in a non-increasing queue.

- Accessing Elements

  pq. ***top()***: Returns a reference to the first element in the queue.
  i.e. top() will reference largest element in a non-increasing queue.

- Size of Queue

  pq. ***size()***: Returns the number of elements in the queue.

  pq. ***empty()***: Returns if queue is empty or not (Bool).

—

Some Practice Questions!

- https://leetcode.com/problems/k-closest-points-to-origin/
- https://codeforces.com/contest/855/problem/A

PRACTICE

- https://leetcode.com/problems/top-k-frequent-elements/
- https://www.hackerrank.com/domains/cpp?filters%5Bsubdomains%5D%5B%5D=stl&badge_type=cpp

All set?

Tip

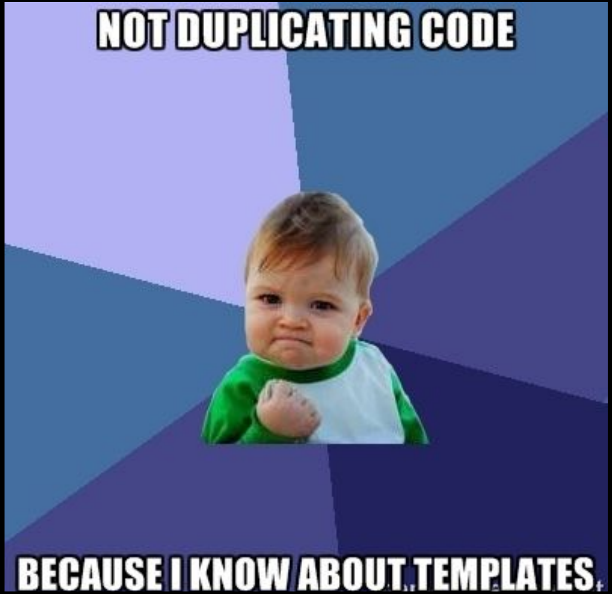
Practice!

Practice!

Practice!

Some other resources:

- https://drive.google.com/file/d/0B4AmxgIlrh_SUjN2VXE0NU5Benc/view
- https://drive.google.com/file/d/0B4AmxgIlrh_SS3ZLV1FubU5XR1U/view
- https://www.topcoder.com/community/competitive-programming/tutorials/power-up-c-with-the-standard-template-library-part-1/
- https://drive.google.com/file/d/0B4AmxgIlrh_SUmxtTW5qVUdvaUU/view
- https://www.youtube.com/watch?v=g-1Cn3ccwXY
- https://www.studytonight.com/cpp/stl/stl-introduction



Good luck!

We hope you'll use what you learnt today in any future coding competitions!

We are sure this was one memorable and interesting **#HourOfCode!**

Connect with us at:
everythinggeeky.101@gmail.com

Srinidhi Ayyagari
https://www.linkedin.com/in/ayyagari99/

Drishti Gupta
www.linkedin.com/in/drishti307