WATCH MY YOUTUBE VIDEO FOR DEEP EXPLANATION OF THESE SLIDES

C++ STL

(quickest way to learn, even for absolute beginners)

C++ STL is like a weapons-pack or “Toolkit” of C++.
It contains some very useful data structures and algorithms.
STL is one of the reasons why C++ is best for CP.

To start using:

```
#include<bits/stdc++.h>
using namespace std;
// Now all STL Containers and Functions are ready for use
```

About Me

Not Bragging, just telling it to learners so that they learn confidently with faith in the teacher.

Hi, I am Utkarsh Gupta.

Upcoming Google Software Engineer. (Offcampus, Google contacted me)

I am one of the best Competitive Programmers in India.

Subscribe to my YT Channel for more content

Achievements:

India Ranks 2, 2, 2, 3 in Google Kickstart Round A,B,C,D respectively.

Grandmaster on Codeforces (India Rank 2)

7 star coder on Codechef

Watch me do Leetcode Weekly Contest in less than half time

Benefits of STL

- Using STL, you can write shorter code that runs faster
- The prewritten codes in STL are extremely error-free and optimized.
- As you study advanced concepts - STL will be very important
  - Vector is used for graph adjacency list
  - pairs and sets are used for dijkstra algorithm in graph
  - And many more...

Vector

It is a dynamic sized array. Number of elements can be increased or decreased. (In Java same behaviour is shown by ArrayList).

vector<int> v; // empty vector of integers

vector<int> v(10); // vector of integers with 10 elements (all 0)

vector<char> v(10,'h'); // vector of chars with 10 elements (all 'h')

Important Functions:

v.push_back(x) - insert the value x to the end of the vector. O(1)

v.pop_back() - erase the last element. O(1)

v.clear() - erase all elements. O(n)

v.size() - returns the current size of the vector. O(1)

[] operator - can be used to access elements like an array. O(1)

cout << v[0]; // prints the first element in the vector

Importance of Vector

There are hundreds of use-cases but some of them might be too advanced to beginners, so here is an easier example.

Given N numbers in input, print 2 lines, in first line, all even integers in sorted order, in second line, all odd integers in sorted order.

Solution hint:
Make 2 vectors - one for even elements, other for odd elements, push_back() the elements into the correct vector accordingly. Then sort both vectors and print.

(Note: This problem can be done without vectors also, but it is easier with vectors)

sort()

This function can be used to sort an array or a vector or a string. The underlying sorting algorithm is called the gcc_sort which is a hybrid algorithm which is implemented in a very efficient way. O(NlogN)

If you manually write a sorting algorithm, it'll be slower than this.

Usage:

```
int a[n];
sort(a,a+n);
```

```
vector<int> v;
sort(v.begin(),v.end());
```

```
string s;
sort(s.begin(),s.end());
```

Note: begin() and end() are called iterators, we'll discuss them later.
In short, they're a little bit similar to pointers.

Pair

Advantages:

In case, you want to return multiple values from a function.

(see next slide for the main advantage)

Pair is a way of creating a ***Composite-Datatype*** composed of 2 different primitive/composite datatypes.

```
pair<int,int> p; // a pair of 2 ints
pair<int,string> p; // a pair of an int and a string
pair<int,pair<int,string>> p; // a pair of int and (pair of int and string)
pair<vector<int>,string> p; // a pair of a (vector of int) and a string
```

Access elements using .first and .second

```
pair<string,int> p = {“hello”,6};
cout << p.first << “ “ << p.second; // prints: hello 6
```

Sorting arrays/vectors of Pairs (Very Useful)

Say we have an array of pairs.

pair<int,int> p[5]; // an array of 5 pairs

p[0] = {1,2}; p[1] = {5,2}; p[2] = {8,1}; p[3] = {1,0}; p[4] = {3,4}

Let's sort this array:

sort(p,p+5);

Now the array looks like:

[{1,0}, {1,2}, {3,4}, {5,2}, {8,1}]

Sorting is done in a way that the ordering is done by the "first" element, but wherever the "first" is equal, the ties are broken by comparing second.

Try this question:

Given a list of names and scores of students, print the names of students in decreasing order of scores.

Iterators

NOTE: v.end() is the iterator to a non-existent element (after the last element)

These behave a lot like pointers.

```
vector<int> v = {10, 15, 12, 5, 20};
vector<int>::iterator it = v.begin();
// OR
auto it = v.begin();
cout << *it; // 10
it++;
cout << *it; // 15
it--;
cout << *it; // 10
cout << *(it + 3); // 5
```

"auto" keyword is used to deduce datatype automatically

```
int a[5] = {10, 15, 12, 5, 20};
int *p = a;

cout << *p; // 10
p++;
cout << *p; // 15
p--;
cout << *p; // 10
cout << *(p + 3); // 5
```

Set

Set is a container which keeps a **unique** copy of every element in **sorted order**.

(In Java same behaviour is shown by TreeSet).

set<int> s; // empty set of integers

set<string> s; // empty set of strings

Important Functions:

s.insert(x) - insert the value x into set, do nothing if already present. O(log N)

s.erase(x) - erase the value x from set if present. O(log N)

s.count(x) - returns 0 if x is not in set and 1 if x is in set. O(log N)

s.clear() - erase all elements. O(n)

s.size() - returns the current size of the set. O(1)

WRONG: cout << s[0]; // [] operator doesn't work with set

Set Iterators

NOTE: s.end() is the iterator to a non-existent element (after the last element)

Set iterators offer less features than vector iterators.

auto it = s.begin(); // it is the iterator to the first element

it++, it--, ++it, --it -> These are all valid and work in O(logN) time

NOTE: (it + 5) or it += 2 etc are INVALID. To advance multiple steps, do it++ multiple times.

Functions related to set iterators:

s.find(x): returns iterator to element with value x. Returns s.end() if not found. O(logN)

s.lower_bound(x): returns iterator to the first element which is >= x. Returns s.end() if not found. O(logN)

s.upper_bound(x): returns iterator to the first element which is > x. Returns s.end() if not found. O(logN)

s.erase(it): erases the element with iterator it. O(logN)

Both of the next 2 lines are exactly same.

```
if(s.find(10) == s.end()) cout << “Not Found”;
if(s.count(10) == 0) cout << “Not Found”;
```

Map

Very common use-case: Count frequency of various objects

You can think of these as special arrays in which the indices(keys) of elements can be negative or very big or even strings! These are like python-dictionaries. (In Java same behaviour is shown by TreeMap).

```
map<key_datatype, value_datatype> m;
```

```
map<string, int> m; // defines a map in which the keys of elements are strings
```

Now we can use it like:

```
m[“hello”] = 50;
m[“world”] = 12;
cout << m[“hello”] << “ “ << m[“world”]; // 50 12
```

NOTE: Maps are very similar to sets, in sets the values are unique and sorted, in maps, the keys are unique and sorted

```
map<int,int> m;
m[-234] = 49; // negative ints are also valid as keys
```

Map (Continued)

m.clear() - Clears a map

m[key] - value of element with key. O(logN)

m.count(key), m.find(key), m.erase(key),

m.lower_bound(key), m.upper_bound(key) - similar to set

Map Iterators behave similar to set iterators, but upon doing *it you instead of getting the value, you get a pair of {key, value}

BONUS:
(*it).first and (*it).second
Can instead be written as
it -> first
it -> second

Examples:

```
map<string, double> m;
// insert values in map
auto it = m.find("utkarsh");
pair<string, double> p = *it; // {"utkarsh", m["utkarsh] }
```

Iterating Containers

```
for(auto it = s.begin(); it != s.end(); it++){
    // *it
}
```

This works for all three: set, map and vector

Shorthand:

```
vector<int> v;
for(int x:v){
    // x
}
```

```
set<int> s;
for(int x:s){
    // x
}
```

```
map<int,int> m;
for(pair<int,int> x:v){
    // x.first, x.second
}
```

Try Out These Problems

https://codeforces.com/problemset/problem/22/A (SET)
https://codeforces.com/problemset/problem/782/A (SET)
https://codeforces.com/problemset/problem/4/C (MAP)

https://codeforces.com/contest/903/problem/C (MAP - medium level)

Do these also without knowing which containers to use:
https://www.spoj.com/problems/MINSTOCK/
https://codeforces.com/problemset/problem/799/B

Also, keep practicing problems from Codeforces, div2 B and C often require you to use some STL containers and functions.

References (Can be used like “Glossary”)

Any STL container or function, you want to learn about: Just google search “Cplusplus [container name]”

For example: “Cplusplus vector” gives the following result:
https://www.cplusplus.com/reference/vector/vector/

Similarly:
https://www.cplusplus.com/reference/set/set/
https://www.cplusplus.com/reference/map/map/

BEST OF LUCK!

Next Slides are not for Beginners, they have some intermediate level stuff, continue only if you have good grasp of STL and C++

Custom Comparators (Less Commonly Needed)

You can define your own rule for sorting!

For example:

```
bool decreasing_order(int x, int y){
   return x > y;
}
int a[10];
sort(a, a+10, decreasing_order); // sorts in descending order
```

NOTE: Using Comparator Classes, we can apply custom sorting rules to sets and maps also

The comparator with arguments (x,y) should return true IF AND ONLY IF, x is necessarily on the left of y in the sorted array. Read more here.

Exercise: Define Custom Comparator to sort pairs in increasing order of first and if there are ties, break those in decreasing order of second.

Further Study (Not Relevant for Beginners)

Read about these containers on your own, it should be easy because most of the important concepts are already covered. These are less commonly used so you don't need to worry about these for a long time.

- queue
- stack
- deque
- priority_queue
- multiset / multimap -> can store duplicates (too complex for beginners)
- unordered_set / unordered_map (like HashSet or HashMap in Java)

NOTE: unordered set and map are not-reliable and can perform bad in certain situations, beginners should always avoid them.