National Institute of Technology Warangal

Center for Career Planning and Development (CCPD)

Training Program

Data Structures & Algorithms
Object Oriented Programming
Database Management Systems

- By Sai Chebrolu and Rithin Pullela in association with CCPD

INTRO:

Hello people, this is Sai Chebrolu and Rithin Pullela final year students from EEE branch NITW. This is a software preparation guide for students of 3rd and 2nd year or even the 4th years students!

There are a lot of resources on the internet to prepare for software, but we wanted to filter out the best sources and provide them to you for better understanding.

We have clubbed together the video lectures on YouTube, problems from Leetcode and some concepts from Geeks for Geeks.

Even though there are a profusion of websites available to practice problems, we strongly suggest you guys to use leetcode.

Please note the videos we suggested are in Hindi, so you at least need to understand Hindi (very basic is also fine).

The best way to learn coding is by practicing a lot of sums. So, if you want to learn fast don't skip concepts or problems work a greater number of hours a day. That way you will surely get a good knowledge. But if you are in some real hurry then at least do 50% sums in each topic.

Don't directly jump into topics like DP and Graphs, that is a grave mistake lot of people do. Try to go in the specified order and you'll finish it faster.

While Solving Sums on Leetcode, just make sure you are going into the discussions section for every sum and no matter if you solve it or not. You will have an exponential learning there; you can find different approaches to the same sum.

We don't say this is a panacea but this will really be a good place to start at. Do refer other sources also, don't limit yourselves to this. We really hope this would be of some help to you. Happy coding!

PS: Dark Reader extension on your browser works well with Leetcode, enjoy the dark mode 

Data Structures and Algorithms:

We assume you are proficient enough in any one of the coding languages, if not we suggest you to try C++.

1. For basics of C++:
   https://www.youtube.com/watch?v=GMx_G05cqYI&list=PLF541C2C1F671AEF6

   Sums on basics:
   1) Check if a given number is prime.
   2) Check if a given string is palindrome.
   3) Find sum of squares of numbers in an array. Ex for {1,2,3} return $1^2+2^2+3^2=14$
2. For basics of pointers:
   https://www.youtube.com/watch?v=h-HBipu_1P0&list=PL2_aWCzGMAwLZp6LMUKI3cc7pgGsasm2_
3. For Basics of recursion:
   https://www.youtube.com/watch?v=_OmRGjbyzno&list=PL2_aWCzGMAwLz3g66WrxFGSXvSsvyfzCO
4. If you want to learn or revise basics of Data Structures, try this playlist:
   https://www.youtube.com/watch?v=92S4zgXN17o&list=PL2_aWCzGMAwI3W_JlcBbtYTwiQSsOTa6P

Hoping that you know basics of Data Structures, this would be our itinerary:

1) Intro to STL in C++
2) Binary Search
3) Stacks
4) Hash Tables (Map and Set)

5) Heaps
6) Arrays and Strings
7) Recursion
8) Back Tracking
9) Trees
10) Linked Lists
11) Dynamic Programming
12) Graphs and grid problems
13) Extras

1) C++ STL

The C++ STL, is one of the most used during coding, it reduces a lot of effort while coding and gives a shorter and a simple code, it is very essential while programming.

(There will be similar functions/ data structures in other languages too, have a look)

We are listing out the most important and the most used ones below.

1) Vectors
2) Map and Unordered map
3) Set and Unordered set

4) Priority Queue (heap)
5) Stack and queue
6) pair

Although there are many more like list, dequeue etc., the ones listed above are the most frequently used.

Watch this video to get a basic idea of how STL works **(MUST)**: https://www.youtube.com/watch?v=zBhVZzi5RdU

There are algorithms also which we use frequently which include

1) sort
2) upper_bound and lower_bound
3) find
4) reverse.

SUM on lower_bound:
https://www.hackerrank.com/challenges/cpp-lower-bound

Refer the below link for the algorithms library.

https://www.geeksforgeeks.org/c-magicians-stl-algorithms/

Vector:

Vector is basically an array which can do a lot more than a normal array! (Assume it like an array)

one way to define a vector in c++ is vector<int> v ;

v is the name of the vector.

We can also create a vector of a defined size.

Vector<int>v(10);

We can also initialize it:

vector<int>v(10,0);

So over here a vector of size 10 is created initialized to zero.

A vector simply functions like an array only, but it is much simpler to use due to how the memory is handled internally.

With vector we use the push_back(something), and pop_back() to insert at the end and remove from the end respectively.

V.size() will tell you the size of the vector currently.

https://www.geeksforgeeks.org/vector-in-cpp-stl/

Couple of sums to make you feel comfortable with vectors:

1) https://www.hackerrank.com/challenges/vector-sort
2) https://www.hackerrank.com/challenges/vector-erase

Map and unordered-map

Basically, maps and unordered maps are the same there is only small difference.

Maps have a key value pair, please watch any video on YouTube to have an idea how maps work.

Maps are very great to remember things, like it can be the occurrences or having to hold the count of something or searching for something, maps tend to do it very effectively.

```cpp
int main()
{
    map<int,int>m1 ;

    vector<int>v={2,1,2};

    for(int i=0;i<v.size();i++)
    {
        m1[v[i]]++;
        // m1 [  key  ] = value , so here for i = 0 , a new key will be created 2 -> 1 .
        // for i = 1 , a new key is created at i = 1 , 1 -> 1
        // for i = 2 , already exisitng 2 -> 1 , is updated to 2 -> 2.
    }
    for(auto it:m1)
    {
        cout<<it.first<<"->"<<it.second<<endl;
        // .first will access the key and .second will access its value.
    }
    // .find is used to search for a value.
    if(m1.find(2)==m1.end()) // this is a syntax i use for searching
    {
        cout<<" KEY NOT FOUND IN THE MAP ";
    }
    else cout<<" KEY FOUND IN THE MAP "<<endl; // if found , you can display value for the current key as
        well if you want using cout<<map[2] or map[1] respectively.
}
```

Similarly we can do multiple operations if certain key is present in our map, like m1.erase(2) will erase the key 2 if present.

The implementation for unordered_map is also similar. below is the link for unordered map.

https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/

A small sum to make you feel comfortable with maps:

https://www.hackerrank.com/challenges/cpp-maps

Set and unordered set

Efficient insertion, deletion and searching takes place just like maps.

Just like maps, we can use set to order or sort our data in increasing or decreasing manner. by default, it is increasing manner. the below link will contain the syntax for decreasing order. Also, in set random access is not possible so we have to iterate if we want each element.

https://www.geeksforgeeks.org/set-in-cpp-stl/

SUM: https://www.hackerrank.com/challenges/cpp-sets

Priority queue(heap) , stack , queue:

https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/

https://www.geeksforgeeks.org/stack-in-cpp-stl/

https://www.geeksforgeeks.org/queue-cpp-stl/

these are fairly simple to catch up with, so have a look.

Pair:

Pair ds is also pretty important, we personally had to use pair< > to solve a lot of questions.

Again, there are various ways to use it like pair<int,int> p;

pair<string,int>p;

pair<char,int>p;

etc depending on your need.

Depending on the question if you think in your data structure you need some more data you could always use pair as well.

Let's say you are trying to solve something and for that you need to take account for the element and the index.

So here we can use pair.

vector<pair<int,int>>v;

Let’s say arr is the name of the vector.

```
for(int i=0;i<arr.size();i++)
{
      V.push_back( {arr[i],i} );
}
```

This is the syntax we use while inserting and if we want to access, we access using .first and .second

Like v[i].first; v[i].second;

https://www.geeksforgeeks.org/pair-in-cpp-stl/

so here are the most important topics to cover in C++ STL, please do have a look in geeksforgeeks in case we missed out anything.

The ones we listed are the syntax that we prefer, you can use this or the one that is used in geeksforgeeks, whatever you are comfortable with.

Again, STL is very important, and you need to code all of what we mentioned if you want a good grip and use

effectively, please refer the links carefully and make sure to visit other websites if geeksforgeeks is not very clear.

Before you start you code in leetcode, know why leetcode is the best for preparation. It is mainly due to the discussions page. Leetcode has one of the best communities for all languages. So, even if your code runs or not always see 3-4 solutions in the discussions page, that is where you will come up with new ideas to approach and solve the problems. There is always a simple solution someone might have written. So please, **refer the discussions in leetcode for each problem.**

2) Binary search

Binary Search is quite an underrated concept but it is actually a very good concept to learn and start coding with.

Keeping it short, this is the playlist to watch:

https://www.youtube.com/watch?v=j7NodO9HIbk&list=PL_z_8CaSLPWeYfhtuKHj-9MpYb6XQJ_f2

This playlist is very important, please watch all the videos, and the last video is THE MOST IMPORTANT

video in binary search. Till now for final year placements 5 top tech companies have asked based on the concept of the last video in the playlist

Sums to Practice:

Short note before you start, no matter if you get 100% to your solution or you fail to solve it.

DO GO THE DISCUSSIONS SECTION and have a look at other solutions posted, this is where the most of the learning takes place.

1) https://leetcode.com/problems/count-negative-numbers-in-a-sorted-matrix
2) https://leetcode.com/problems/find-smallest-letter-greater-than-target
3) https://leetcode.com/problems/search-insert-position
4) https://leetcode.com/problems/valid-perfect-square
5) https://leetcode.com/problems/first-bad-version
6) https://leetcode.com/problems/sqrtx

Sums based on last video: (**very important)**

7) https://leetcode.com/problems/capacity-to-ship-packages-within-d-days/
8) https://leetcode.com/problems/koko-eating-bananas/

9) https://leetcode.com/problems/sum-of-mutated-array-closest-to-target/
10) https://leetcode.com/problems/magnetic-force-between-two-balls/

3) Stack:

Stack is a very versatile data structure which can be used in substantially different situations:

This is the playlist: https://www.youtube.com/watch?v=P1bAPZg5uaE&list=PL_z_8CaSLPWdeOezg68SKkeLN4-T_jNHd

SUMS:

1) https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string
2) https://leetcode.com/problems/next-greater-element-i
3) https://leetcode.com/problems/min-stack
4) https://leetcode.com/problems/valid-parentheses
5) https://leetcode.com/problems/minimum-add-to-make-parentheses-valid
6) https://leetcode.com/problems/daily-temperatures
7) https://leetcode.com/problems/next-greater-node-in-linked-list

8) https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string-ii
9) https://leetcode.com/problems/asteroid-collision
10) https://leetcode.com/problems/longest-well-performing-interval
11) https://leetcode.com/problems/trapping-rain-water
12) https://leetcode.com/problems/largest-rectangle-in-histogram

stack can be tricky because, you might not be able to just read the questions and decide that you have to use a stack, you will get that only by practice. So, keep practicing. watch the videos clearly and understand the topic to the core.

Stack questions were asked in Amazon, Oracle etc this year for placements.

4,5,6) Heaps, Arrays, Strings, Hash tables (Maps, sets):

These topics are very closely related to each other. And for most of the cases they can be considered as Arrays and Strings only.

Most people take this concept for granted, but don't forget this is the most important topic of all. At least 1 question can be expected in any online tests.

First step: Heaps playlist
https://www.youtube.com/watch?v=hW8PrQrvMNc&list=PL_z_8CaSLPWdtY9W22VjnPxG30CXNZpI9

Second Step: Brush up Maps and Sets working. You can also google them while solving sums and slowly you'll get a good grip on them.

Third step: String STL (VV Imp)

Try this video: https://youtu.be/NHsqNhWE1yE

Have a look in gfg:
https://www.geeksforgeeks.org/stdstring-class-in-c/

Some important functions:

Note that string is very similar to vector, you can use things like, s.begin(); s.end(); s[i]; s.size(); s.push_back(); s.pop_back(); s.clear() etc.

1) **String.erase(idx, len):** There are many ways of using this but I personally prefer the following to avoid confusion.

   To erase whole string:

   str.clear(); (can use str.erase() too)

   To erase part of string:

   Str.erase(index,length);

This deletes the sub string of length (length) starting from index (index).

Example: let string str = “HakunaMatata”;

str.erase(2,3) will modify it as “HaaMatata”

2) **string.substr(index,length):**

   Similar to erase function right! In terms of arguments of course. This is used to get a Sub-string from a string. Let me directly jump into the example.

   Example: let string str = “HakunaMatata”;

   str.substr(6,6) return a string as “Matata”

3)**to_string(any_type_num):** This converts any type of number be it integer type, float type, long, double etc. into a string.
For ex: if double x=12.06;
to_string(x) returns a string "12.06";

4) <u>stoi(string):</u>

This converts a string into integer (digits in string).
PS: It should be a valid case.
This can come in very handy when we are dealing with big numbers > INT_MAX or when we want to play around with digits. You can easily count number of digits without much code.

Ex: let string str= "1206"
stoi(str) gives the integer 1206.
If str= "12.06", stoi(str) gives integer 12.
If str = "12.06 hey" stoi(str) gives integer 12.

Feeling overwhelmed? No worries we can always do trial and error and there is no chance of forgetting these after the rigorous practice coming up.

Running sum/ prefix sum concept:

It is what the name suggests it to be, let us see it in an example.

Let array a= {1,2,3,4};

We create a running sum array which is supposed to be

Array running sum = {1,3,6,10}

This is the sum of all previous elements including itself.

For ex, at index 2 running sum= 3+2+1= 6.

Code of how to get running sum:

Execute | Share main.cpp STDIN Login

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
    vector<int> array={1,2,3,4};
    int n=array.size();
    vector<int> runningSum(n);

    runningSum[0]=array[0];

    for(int i=1;i<n;i++)
    runningSum[i]=array[i]+runningSum[i-1];

    for(auto i:runningSum)
    cout<<i<<" ";
   // this outputs 1, 3, 6, 10

   return 0;
}
```

Let’s dive into sums now, we took a lot of time organizing these sums, so yeah  and note that there might be surprise sums the previous concepts which we have discussed!

Quick Note: Please Refer the discussions page for every sum, make it a habit.

1) https://leetcode.com/problems/move-zeroes/
2) https://leetcode.com/problems/kids-with-the-greatest-number-of-candies/
3) https://leetcode.com/problems/number-of-good-pairs/
4) https://leetcode.com/problems/how-many-numbers-are-smaller-than-the-current-number/
5) https://leetcode.com/problems/split-a-string-in-balanced-strings/
6) https://leetcode.com/problems/destination-city/
7) https://leetcode.com/problems/reverse-string/
8) https://leetcode.com/problems/unique-email-addresses/
9) https://leetcode.com/problems/consecutive-characters/
10) https://leetcode.com/problems/find-numbers-with-even-number-of-digits/
11) https://leetcode.com/problems/count-negative-numbers-in-a-sorted-matrix/
12) https://leetcode.com/problems/replace-elements-with-greatest-element-on-right-side/
13) https://leetcode.com/problems/can-make-arithmetic-progression-from-sequence/

14) https://leetcode.com/problems/squares-of-a-sorted-array/
15) https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string-ii/description/
16) https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/
17) https://leetcode.com/problems/longest-well-performing-interval/description/
18) https://leetcode.com/problems/subarray-sum-equals-k/
19) https://leetcode.com/problems/check-if-array-pairs-are-divisible-by-k/
20) https://leetcode.com/problems/avoid-flood-in-the-city/
21) https://leetcode.com/problems/max-consecutive-ones/
22) https://leetcode.com/problems/kth-missing-positive-number/
23) https://leetcode.com/problems/search-suggestions-system/
24) https://leetcode.com/problems/destination-city/

Sliding Window Concept:

The next set of problems are based on this concept.

Here's a video explaining this:
https://www.youtube.com/watch?v=MK-NZ4hN7rs

Sums Continued:

Quick Note: Please Refer the discussions page for every sum, make it a habit.

25) https://leetcode.com/problems/max-consecutive-ones-iii/
26) https://leetcode.com/problems/maximum-number-of-vowels-in-a-substring-of-given-length/
27) https://leetcode.com/problems/permutation-in-string/
28) https://leetcode.com/problems/longest-substring-without-repeating-characters/

If you have followed exactly what's mentioned here, Congratulations you've finished 50 sums on leetcode.

29) https://leetcode.com/problems/maximize-distance-to-closest-person/
30) https://leetcode.com/problems/maximize-distance-to-closest-person/
31) https://leetcode.com/problems/best-time-to-buy-and-sell-stock/
32) https://leetcode.com/problems/partition-array-into-three-parts-with-equal-sum/
33) https://leetcode.com/problems/find-and-replace-pattern/
34) https://leetcode.com/problems/find-pivot-index/
35) https://leetcode.com/problems/non-decreasing-array/

36) https://leetcode.com/problems/number-of-good-ways-to-split-a-string/
37) https://leetcode.com/problems/minimum-number-of-steps-to-make-two-strings-anagram/
38) https://leetcode.com/problems/palindromic-substrings/
39) https://leetcode.com/problems/valid-triangle-number/
40) https://leetcode.com/problems/number-of-matching-subsequences/
41) https://leetcode.com/problems/minimum-increment-to-make-array-unique/
42) https://leetcode.com/problems/3sum-closest/
43) https://leetcode.com/problems/partition-array-into-disjoint-intervals/
44) https://leetcode.com/problems/maximum-swap/
45) https://leetcode.com/problems/subarray-sum-equals-k/
46) https://leetcode.com/problems/merge-intervals/
47) https://leetcode.com/problems/minimum-size-subarray-sum/
48) https://leetcode.com/problems/can-make-palindrome-from-substring/
49) https://leetcode.com/problems/spiral-matrix/
50) https://leetcode.com/problems/search-in-rotated-sorted-array-ii/

The sums below are important, but we suggest you to do it after finishing other topics.

You can also try sums on Arrays and Strings from interview bit, which was considered helpful by some of our friends.

51) https://leetcode.com/problems/grumpy-bookstore-owner/
52) https://leetcode.com/problems/divide-array-in-sets-of-k-consecutive-numbers/
53) https://leetcode.com/problems/rank-teams-by-votes/
54) https://leetcode.com/problems/container-with-most-water/
55) https://leetcode.com/problems/task-scheduler/
56) https://leetcode.com/problems/reorganize-string/
57) https://leetcode.com/problems/word-subsets/
58) https://leetcode.com/problems/minimum-number-of-frogs-croaking/
59) https://leetcode.com/problems/basic-calculator-ii/
60) https://leetcode.com/problems/replace-the-substring-for-balanced-string/
61) https://leetcode.com/problems/valid-parenthesis-string/
62) https://leetcode.com/problems/longest-substring-without-repeating-characters/
63) https://leetcode.com/problems/can-convert-string-in-k-moves/
64) https://leetcode.com/problems/shortest-unsorted-continuous-subarray/

65) https://leetcode.com/problems/lucky-numbers-in-a-matrix/
66) https://leetcode.com/problems/sort-array-by-parity-ii/
67) https://leetcode.com/problems/find-common-characters/
68) https://leetcode.com/problems/top-k-frequent-elements/
69) https://leetcode.com/problems/relative-sort-array/
70) https://leetcode.com/problems/find-words-that-can-be-formed-by-characters/
71) https://leetcode.com/problems/maximum-score-after-splitting-a-string/
72) https://leetcode.com/problems/first-unique-character-in-a-string/
73) https://leetcode.com/problems/repeated-substring-pattern/
74) https://leetcode.com/problems/valid-palindrome-ii/
75) https://leetcode.com/problems/minimum-absolute-difference/
76) https://leetcode.com/problems/toeplitz-matrix/
77) https://leetcode.com/problems/fair-candy-swap/
78) https://leetcode.com/problems/monotonic-array/
79) https://leetcode.com/problems/find-the-minimum-number-of-fibonacci-numbers-whose-sum-is-k/
80) https://leetcode.com/problems/product-of-array-except-self/

81) https://leetcode.com/problems/longest-subarray-of-1s-after-deleting-one-element/
82) https://leetcode.com/problems/rotate-image/
83) https://leetcode.com/problems/number-of-substrings-containing-all-three-characters/
84) https://leetcode.com/problems/group-anagrams/
85) https://leetcode.com/problems/maximum-number-of-vowels-in-a-substring-of-given-length/
86) https://leetcode.com/problems/smallest-subsequence-of-distinct-characters/

Recursion:

Quick Note: As you have come this far, it would be a good time to start preparing OOP and DBMS. Start with OOPS, study it parallelly while practicing coding.

Alright, so, Recursion might be very scary to a lot of people, but trust me once you get used to it, you'll understand how powerful recursion is. It works like magic!

Here, take your time to understand recursion properly even if you take 2-3 days to get the hang of it, it isn't a

problem because all the topics further below would require a good understanding on recursion.

https://www.youtube.com/watch?v=kHi1DUhp9kM&list=PL_z_8CaSLPWeT1ffjiImo0sYTcnLzo-wY

watch this playlist to get a good understanding, if you still aren't able to understand, please refer other videos as well on YouTube!

please see the discussions page on leetcode for every recursive problem we do from now on. It might be annoying we are repeating it so many times, but trust me your learning will be exponential if you refer the discussions page

Backtracking:

Backtracking is the best way to get a hang of recursion, it is a very good way to start off.

Always remember backtracking is very costly so only if the input is small, then we apply backtracking or else we try to approach using different concept.

Watch the below video to get the best understanding of backtracking.

https://www.youtube.com/watch?v=jFwREev_yvU&t=1075s

below are the few problems to practice on backtracking.

1) https://leetcode.com/problems/letter-tile-possibilities/
2) https://leetcode.com/problems/letter-case-permutation/
3) https://leetcode.com/problems/permutations/
4) https://leetcode.com/problems/generate-parentheses/
5) https://leetcode.com/problems/combination-sum-iii/
6) https://leetcode.com/problems/combination-sum/
7) https://leetcode.com/problems/combinations/
8) https://leetcode.com/problems/permutations-ii/
9) https://leetcode.com/problems/split-array-into-fibonacci-sequence/
10) https://leetcode.com/problems/restore-ip-addresses/
11) https://leetcode.com/problems/path-with-maximum-gold/

refer the solutions in the discussions page even if you are able to solve it, please, make it a habit to refer the discussions page after you give a try!

Trees:

A lot of people are very afraid of trees, but trees are very easy, the point being all tree questions are somewhat similar except a few changes here and there. The first and the most important thing is to understand tree traversals (preorder , inorder , postorder).The most important types of trees here are the binary and the binary search tree.

You will easily understand trees if you are good at recursion.

Trees we either do DFS (recursion manner) or a BFS (iterative using a queue) depending on the question.

Example a level order traversal would be easy if we use queue and BFS, although we can do it using DFS using recursion as well.

For tree traversals watch this video

https://www.youtube.com/watch?v=gm8DUJJhmY4

also watch these problems to get a basic understanding

https://www.youtube.com/watch?v=aqLTbtWh40E

https://www.youtube.com/watch?v=zmPj_Ee3B8c

https://www.youtube.com/watch?v=bD9oYzwfZ1w

some problems to do

1) https://leetcode.com/problems/merge-two-binary-trees
2) https://leetcode.com/problems/sum-of-root-to-leaf-binary-numbers
3) https://leetcode.com/problems/maximum-depth-of-binary-tree
4) https://leetcode.com/problems/invert-binary-tree
5) https://leetcode.com/problems/average-of-levels-in-binary-tree
6) https://leetcode.com/problems/binary-tree-level-order-traversal-ii
7) https://leetcode.com/problems/same-tree

8) https://leetcode.com/problems/deepest-leaves-sum/
9) https://leetcode.com/problems/sum-of-nodes-with-even-valued-grandparent/
10) https://leetcode.com/problems/insert-into-a-binary-search-tree/
11) https://leetcode.com/problems/lowest-common-ancestor-of-deepest-leaves/
12) https://leetcode.com/problems/maximum-difference-between-node-and-ancestor/
13) https://leetcode.com/problems/find-bottom-left-tree-value/
14) https://leetcode.com/problems/find-largest-value-in-each-tree-row/
15) https://leetcode.com/problems/smallest-subtree-with-all-the-deepest-nodes/
16) https://leetcode.com/problems/kth-smallest-element-in-a-bst/
17) https://leetcode.com/problems/most-frequent-subtree-sum/
18) https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal

trees might not be asked in an OT usually, but there is a high probability trees might be asked in an interview.

Please refer the discussions page for each tree problem.

There are many more problems on trees, we are only giving some of the good conceptual questions on trees, you can do other problems on leetcode if you wish.

Linked Lists:

Watch this video to get an idea about linked lists.

https://www.youtube.com/watch?v=dmb1i4oN5oE

Linked lists are also pretty simple, why because there are limited questions on linked list and I personally feel if you solve all the questions on leetcode (around 30-35) you will be able to solve any linked list question in the interview.

Similar to trees, linked lists do not have much probability of being asked in OT, but have a high probability in interviews. companies like Oracle, often ask questions based on linked lists.

Unfortunately, we cannot pick up specific problems and ask to solve because each problem has an equal probability to being asked in linked lists, so its best if you take some time and solve as many as you can to master linked lists.

https://leetcode.com/tag/linked-list/

sort it according to the difficulty, start with the easy problems and keep moving on, if you are struck then watch a video on YouTube or refer the discussions page.

Dynamic programming:

Yes, we are finally here. One of the most important topics and companies just LOVE to ask questions based on Dynamic programming. All companies with a high CTC always try to ask based on dynamic programming

The following playlist of Aditya verma is all about DP, a lot of good questions he picked up. Believe me, no one can explain better than him.

If you want to get GOOD at DP, watch all his videos in the playlist and try to code it one by one. It will give you a very good head start.

The following videos are **very important.**

https://www.youtube.com/watch?v=nqowUJzG-iM&list=PL_z_8CaSLPWekqhdCPmFohncHwz8TY2Go

if you want to nail dynamic programming, you **HAVE TO PRACTICE A LOT.**

So here are some good set of problems based on dynamic programming

1) https://leetcode.com/problems/min-cost-climbing-stairs
2) https://leetcode.com/problems/climbing-stairs
3) https://leetcode.com/problems/maximum-subarray
4) https://leetcode.com/problems/house-robber
5) https://leetcode.com/problems/count-square-submatrices-with-all-ones
6) https://leetcode.com/problems/partition-array-for-maximum-sum
7) https://leetcode.com/problems/minimum-falling-path-sum
8) https://leetcode.com/problems/palindromic-substrings
9) https://leetcode.com/problems/longest-common-subsequence
10) https://leetcode.com/problems/arithmetic-slices
11) https://leetcode.com/problems/minimum-path-sum
12) https://leetcode.com/problems/unique-paths

13) https://leetcode.com/problems/longest-arithmetic-subsequence
14) https://leetcode.com/problems/integer-break
15) https://leetcode.com/problems/largest-sum-of-averages
16) https://leetcode.com/problems/maximum-length-of-repeated-subarray
17) https://leetcode.com/problems/delete-and-earn
18) https://leetcode.com/problems/number-of-dice-rolls-with-target-sum
19) https://leetcode.com/problems/perfect-squares
20) https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown
21) https://leetcode.com/problems/greatest-sum-divisible-by-three
22) https://leetcode.com/problems/target-sum
23) https://leetcode.com/problems/partition-to-k-equal-sum-subsets
24) https://leetcode.com/problems/triangle
25) https://leetcode.com/problems/partition-equal-subset-sum
26) https://leetcode.com/problems/ones-and-zeroes

27) https://leetcode.com/problems/longest-increasing-subsequence
28) https://leetcode.com/problems/word-break
29) https://leetcode.com/problems/largest-divisible-subset
30) https://leetcode.com/problems/maximal-square
31) https://leetcode.com/problems/coin-change
32) https://leetcode.com/problems/longest-palindromic-substring
33) https://leetcode.com/problems/decode-ways
34) https://leetcode.com/problems/palindrome-partitioning-iii
35) https://leetcode.com/problems/burst-balloons
36) https://leetcode.com/problems/shortest-common-supersequence
37) https://leetcode.com/problems/minimum-cost-to-cut-a-stick
38) https://leetcode.com/problems/edit-distance
39) https://leetcode.com/problems/remove-boxes
40) https://leetcode.com/problems/maximum-sum-bst-in-binary-tree
41) https://leetcode.com/problems/interleaving-string

42) https://leetcode.com/problems/minimum-number-of-refueling-stops
43) https://leetcode.com/problems/longest-valid-parentheses
44) https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv
45) https://leetcode.com/problems/regular-expression-matching

phew !! these are a very big set of problems but they will strengthen your fundamentals to a large extent, don't worry if you cannot solve them, DP is a very complex topic and you need a lot and a lot of practice to solve them.

please refer discussions in DP for each problem, it will help a lot.

Graphs:

Graphs is a complex data structure, it has been asked recently In a good number of companies. it is good to know the basics of graphs and solve a few questions to be prepared to tackle graph problems.

Here are the few topics we will cover in graphs

1) Building a graph
2) DFS
3) BFS
4) Cycle in graph
5) Toplogical sort
6) Union find
7) Connected components
8) Graph colouring
9) Graph algorithms like prim, dijkshta (idk the spelling) and so on!

Building a graph

There are various methods to build a graph, usually your input is a vector<vector<int,int>>v , which is a nx2 matrix representing the edges that are connecting 2 nodes.

```cpp
void build_a_graph(int n ,vector<vector<int>>&edges)
{
    // n represents the number of nodes in the graph
    vector<vector<int>>graph(n);
    for(int i=0;i<edges.size();i++)
    {
        graph[edges[i][0]].push_back(edges[i][1]);
     //   graph[edges[i][1]].push_back(edges[i][0]); ( if undirected graph add this
        line)
    }
// basically lets say from node 0 we have edges to 1 and 3 , and from node 1 we have
    edges to 4 and 2 ,and from node 3 to node 1 ,this is how our vector<vector<int>>
    might look
    // 0th row 1 3
    // 1st row 4 2
    // 2nd row
    // 3rd row 1
    // 4th row
}
```

DFS and BFS

This is the basic types of problems that we might expect from a graph type question.

To be able to do any DFS or BFS problem, you will need to understand the concept of visited array.

Meaning in a normal DFS or a BFS we do not want to visit the same node again, so we usually keep a visited array and mark the node as visited as soon as we reach it and then we call another dfs to a node provided it is not visited already.

https://www.youtube.com/watch?v=pcKY4hjDrxk&t=4s

watch the above video to cover the DFS and BFS concepts.

```cpp
void dfs(int i,vector<vector<int>>&graph,vector<bool>&visited)
{
    visited[i]=true;
    cout<<i<<endl;
    for(auto it:graph[i])
    {
        if(visited[it]==false)
            dfs(it,graph,visited);
    }
    return ;
}
void dfstraversal(int n ,vector<vector<int>>&graph)
{
    vector<bool>visited(n,false);
    for(int i=0;i<n;i++)
    {
        if(visited[i]==false)
            dfs(i,graph,visited);
    }
}
```

```cpp
void bfs(int i,vector<vector<int>>&graph,vector<bool>&visited)
{
    visited[i]=true;
    cout<<i<<endl;
    queue<int>q;
    q.push(i);
    while(!q.empty())
    {
        int n = q.size();
        for(int k=0;k<n;k++)
        {
            int index = q.front(); q.pop();
            for(auto it:graph[index])
            {
                if(visited[it]==false)
                {
                    q.push(it); visited[it]=true;
                }
            }
        }
    }
}
void bfstraversal(int n ,vector<vector<int>>&graph)
{
    vector<bool>visited(n,false);
    for(int i=0;i<n;i++)
    {
        if(visited[i]==false)
            bfs(i,graph,visited);
    }
```

This is my syntax, how I usually do a DFS or a BFS, feel free to check out other methods if you aren't able to follow mine!

Cycle in a graph

This is an important concept, there can be a lot of questions which are coming under this topic. there is a cycle for directed graph and for undirected graph.

Follow the videos to understand better

https://www.youtube.com/watch?v=0dJmTuMrUZM

https://www.youtube.com/watch?v=6ZRhq2oFCuo

Topological sort

This is important when the order of something matters, meaning if there is an edge from u->v topological sort always tells that u must be visited before v. it will be easier if you watch a video than me explaining loool.

https://www.youtube.com/watch?v=qe_pQCh09yU&t=674s

union find

union find problems are a bit hard to figure out that we have to actually use union find here, but once we know they are pretty easy to solve.

https://www.youtube.com/watch?v=eTaWFhPXPz4

union find further helps in finding the number of connected components in an undirected graph and also cycle detection in an undirected graph.

Connected components

Given an undirected graph, if you understood the concept of union find, then after we perform union find. we can run a for loop to check if par[i]== i, and we do a count++, and we can get the number of connected components

https://www.youtube.com/watch?v=ymxPZk7TesQ

this is another simple way to find the connected components in an undirected graph, which is to count the number of DFS calls made, this is simpler compared to the above method.

https://www.youtube.com/watch?v=9esCn0awd5k&t=366s

graph colouring

graph colouring is used in various ways, to detect cycles or to separate nodes from each other or something.

https://www.youtube.com/watch?v=_sdVx_dWnlk

https://www.youtube.com/watch?v=0ACfAqs8mm0&t=1168s

https://www.youtube.com/watch?v=L0DcePeWHnM&t=92s

graph algorithms

there are N number of sources available for this, usually when we say graphs everyone suggests to learn these like prims, kruskals, Dijkstra, Floyd and so on... so have a look on YouTube and geeksforgeeks.

Dijkstra is the MOST important out of all of these.

https://www.youtube.com/watch?v=XB4MIexjvY0&t=572s

https://www.youtube.com/watch?v=4ZlRH0eK-qQ&t=507s

https://www.youtube.com/watch?v=FtN3BYH2Zes&t=402s

here are some graph problems

1) https://leetcode.com/problems/keys-and-rooms
2) https://leetcode.com/problems/redundant-connection (union find , cycle detection)
3) https://leetcode.com/problems/find-eventual-safe-states
4) https://leetcode.com/problems/is-graph-bipartite
5) https://leetcode.com/problems/network-delay-time (Dijkstra)

6) https://leetcode.com/problems/find-the-city-with-the-smallest-number-of-neighbors-at-a-threshold-distance (floyd)
7) https://leetcode.com/problems/possible-bipartition
8) https://leetcode.com/problems/course-schedule
9) https://leetcode.com/problems/course-schedule-iv
10) https://leetcode.com/problems/course-schedule-ii (topological)
11) https://leetcode.com/problems/most-stones-removed-with-same-row-or-column (union find)
12) https://leetcode.com/problems/number-of-operations-to-make-network-connected (connected components)

Grid problems:

Here you have to apply a DFS or a BFS , you will be given the directions you are allowed to travel and based on that you will have to make recursive calls , do check for boundary conditions in DFS meaning , you should always stay in the grid , while doing recursion take care of that , meaning if you are in the 0^{th} row during some recursive call you cannot do a dfs(i-1,j,grid) which will make you go out of bounds.

https://www.youtube.com/watch?v=__98uL6wst8

watch the above video to get an idea.

Here are some problems

1) https://leetcode.com/problems/max-area-of-island
2) https://leetcode.com/problems/number-of-closed-islands
3) https://leetcode.com/problems/number-of-enclaves

These are most frequently asked interview questions, you must solve each of these to get a good grip: https://leetcode.com/problemset/top-interview-questions/

Extras

Greedy Algo:

Greedy problems are hard to differentiate from DP ones, they have very similar problem description. So, basically by practice you will be able to understand greedy, do not ignore greedy! a problem from greedy was asked in BlackBuck recently.

https://leetcode.com/tag/greedy/

try to solve the easy and the medium problems here to get a good grip.

Bit manipulation:

Recently some questions on Bit manipulation were also asked in OTs.

Basically, you deal with AND, OR, XOR. mostly XOR is used.

here is a playlist for you to understand the basics

https://www.youtube.com/watch?v=efL86JCONH0&list=PL2q4fbVm1Ik7ip1VkWwe5U_CEb93vw6Iu&index=2&t=0s

https://www.youtube.com/playlist?list=PL-Jc9J83PIiFJRioti3ZV7QabwoJK6eKe

be sure to do ALL the easy problems in leetcode, it will strengthen your understanding.

https://leetcode.com/tag/bit-manipulation/

Some hard questions: Do it after you are complete all the things mentioned above.

1. https://leetcode.com/problems/first-missing-positive
2. https://leetcode.com/problems/median-of-two-sorted-arrays
3. https://leetcode.com/problems/word-ladder-ii
4. https://leetcode.com/problems/minimum-window-substring
5. https://leetcode.com/problems/shortest-palindrome
6. https://leetcode.com/problems/number-of-squareful-arrays
7. https://leetcode.com/problems/sudoku-solver
8. https://leetcode.com/problems/merge-k-sorted-lists
9. https://leetcode.com/problems/cheapest-flights-within-k-stops
10. https://leetcode.com/problems/design-twitter
11. https://leetcode.com/problems/distant-barcodes
12. https://leetcode.com/problems/sliding-window-maximum

Well if you've finished this congratulations! But this not the end, you can still learn a lot. Thank Leetcode and Aditya Verma and keep coding! All the Best! Do a lot of sums!!!

OBJECT ORIENTED PROGRAMMING CONCEPTS

This is the most important theory concept during interviews and also during OTs, you are expected to know OOPS concepts for sure.

Since Rithin and I are familiar with C++, we are explaining in C++, so very sorry if there is any inconvenience caused for java, python etc programmers, but we will be giving good examples which you might use.

Quick Note: There might be other concepts which we missed out so check other sources as well! (Java Point)

Some time back, knowing the definition for topics in OOPS concepts would be enough, but that is not the case now, companies expect you to relate real life examples with OOPS along with some software examples and some code as well. We will try our best to give you good examples which might help you understand better and which you can give in your technical interviews.

So here is the list of some of the most frequently asked and some of the most important topics in OOPS

1) Abstraction
2) Access specifiers
3) Data hiding
4) Encapsulation
5) Inheritance
6) Constructors
7) Destructors
8) Polymorphism
9) Virtual functions
10) Pure virtual functions
11) Abstract classes
12) Friend functions.

Before all of this first try to understand what classes and objects are

Watch these to get an idea.

https://www.youtube.com/watch?v=6Q0Cff29YwU

https://www.youtube.com/watch?v=8yjkWGRlUmY&t=227s

Abstraction

Starting off with a simple example, take a simple car for an example, when you sit in the car what exactly do you see while driving? mainly the steering wheel, and brakes and etc, but when we push the throttle in the car moves forward…. Now to a person driving it is not important to know how the engine actually responds or works when we push the throttle. So, in a way the driver is shown whatever is essential and the rest of the working is not known to him, this is an example for abstraction, we are hiding the internal processes going on. Coming to a software example, in C++ STL, when we use the sort function, our vector is sorted, but do we know what sorting algorithm is used? is it bubble? merge? quick? no we don't, because that information is hidden to us, meaning the implementation of the sort function internally is not known to us. Another example is the cout statement in C++, when we do a cout all we know is that something is displayed on the monitor, but internally there are some complex operations taking place like a system call is made and the kernel takes over and does an IO operation in some Operating systems. But the point is, we know that if we do a cout<<, we get a display, how this function is being done internally is in-fact kept away from us, the

internal implementation is hidden, this is again abstraction.

Hopefully now you have an idea what abstraction is, to define It now, abstraction shows only the essential details and hides the unnecessary details from the user, in simple terms.

Refer any simple code for abstraction, there are many websites with many examples, choose what is simple for you!

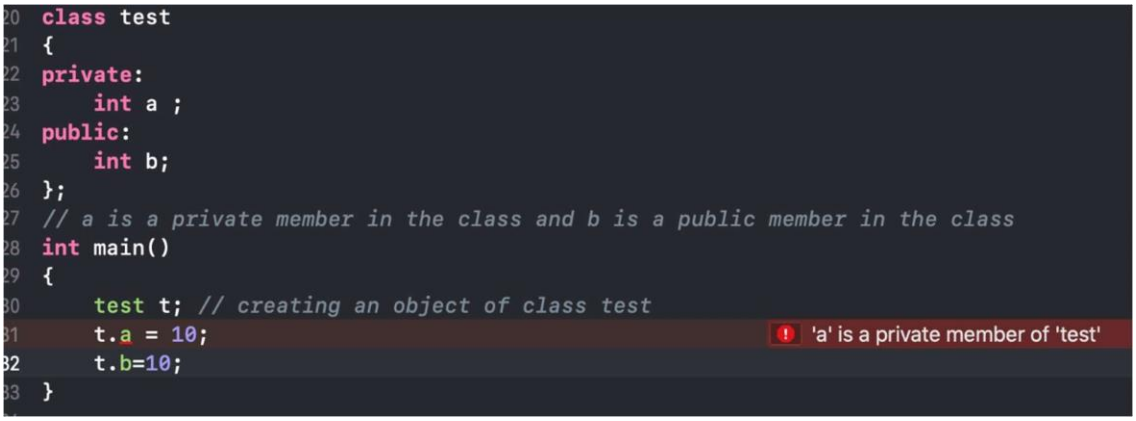
If you want a suggestion JavaPoint is a good website for OOP.

Access specifiers

there are 3 access specifiers in C++, public, private and protected.

The public members can be used inside and outside the class, the private members can be used only inside the class and cannot be used outside the class.

Protected members can be used inside the class and inherited classes and cannot be used outside classes.



Hope you have understood the error.

Access specifiers are important as they are used to implement abstraction and data hiding and types of inheritance and etc.

Data hiding

In our Kart of Team Tejas(❤❤), we have certain critical wire connections going to the motor which can be altered to change the working of the motor, basically any changes in wire might affect its operation, so what we did was we completely enclosed the motor such that the wires are completely hidden and no accidental changes can be made which might affect its operation.

Basically, as a manufacturer it is your responsibility to keep the critical data safe for proper functioning.

Let’s take another software example now, let’s say we have a class student and we have int age; defined as public, so it can be used outside the class. Now let’s say a person comes and makes the age of some student while creating the object as negative, as we know the age cannot be negative. But since we have made it public, it is accessible and it is prone to unwanted or bad changes which might affect the operations. So what is the solution is, we make age as private instead, and create a public function, the public function inside the class can access the private data and update the value, before updating we can check if the age Is negative and if negative we can return a prompt saying it is invalid or something. So hopefully you understood what exactly data hiding is and why we need it.

```
class student
{
private:
    int age ;
public:
    void set(int x)
    {
        if(x<0)
            cout<<"ENTER POSITIVE AGE"<<endl;
        else age = x;
    }
};

int main()
{
    student s;
    s.set(20);
}
```

In a way if we look at it, data hiding allows us to have better control over our software, if it were public we would have no control and the user could do anything, but using the access specifiers and then making age private and then doing some checking, we are having better control over our software.

Encapsulation

Encapsulation essentially wraps up or combines the data members and member functions into a single unit. Usually the data is kept private and the functions which are used to manipulate the data are kept as

public members. In a way we can say that encapsulation helps in achieving the data hiding concept.

So, you can also explain the data hiding examples saying they can be achieved using the encapsulation.

So, encapsulation is basically some data hiding and abstraction.

Classic example is a medicine or a capsule, the medicine has a lot of combinations and all, and it is all into a single unit. Like those 2 coloured medicine, one colour we can say they are data members and the other half colour is the member functions but in a whole, both form as one single unit.

If you are asked for a code, take any simple example and explain how we can manipulate the data members.

```
class student
{
private:
    int roll ;
public:
    void set(int x)
    {
        roll = x;
    }
    void show()
    {
        cout<<" The Roll Number is "<<roll<<endl;
    }
};

int main()
{
    student s;
    s.set(12312);
}
```

Here the private data is roll and set and show are the public member functions used to manipulate the private data member and roll, set (), show() are wrapped into a single unit which is our class itself.

Inheritance

This concept is mainly used to mainly introduce reusability. A good example can be , let's say we release a product this year with some features and then suppose next year, we want to make product with new features but we don't essentially build it up from scratch implementing everything from the start, what we can do is make our new product have the features

of all the previous year product along with that now we can introduce new features to our product, so in a way we have reused our previous product and also made additional changes which were implemented new. So, essentially, we borrowed from something already existing and add additional changes on our own to our new product. Here our product can be some car, or some mobile phone. Basically, make use of something already existing

Now to define inheritance, inheritance can be told as deriving a new class from an existing base class.

There are different types of inheritance, go through the below link, it is easy to follow.

https://www.javatpoint.com/cpp-inheritance

Constructors

Constructor simply initialises an object. Basically, the object gets all the resources allocated.

So, when exactly is a constructor called, essentially we can manually create a constructor, or if not a constructor is automatically called by the compiler when we create an object.

Student s ;

This statement automatically calls a constructor, the compiler takes care of it and calls the constructor. It allocates all the memory needed for the object, for example when we create object s of student type, all the members of student class like roll.no, name, age etc will be automatically created.

We can also define our own constructor, there are 3 ways to do it

1) Default constructor
2) Parameterised constructor
3) Copy constructor.

To understand this concept, let us take an example and learn, Let's say you go to a shop to buy a pen, if you just ask for a pen without mentioning any specifications, he might give you a normal pen of say blue colour. This is an example of default constructor, you don't specify anything particularly and complier gives an object with random garbage values in data members.

Next, Parameterised constructor, now let's say you go to the pen shop again but this time you specify the colour of the pen, brand of the pen etc. this time he gives you that particular pen. So parameterised constructor takes all the parameters, creates an object with those parameters.

Finally, Copy constructor, now you go to the pen shop again and this time you take a old pen, show it to the shopkeeper and ask him to give a similar pen. Then he gives you that particular type of pen. So, a copy constructor takes an object as an example and creates an object with same parameters as the example and gives you the new object.

Quick note: Constructor do not have a return type! It is ***not even void*** it has no return type. You will understand better after seeing the syntax here:

https://www.geeksforgeeks.org/constructors-c/

go through the above link to get a clear idea.

Problem with copy constructor

There are 2 types of copy constructors, shallow and deep copy constructors.

A shallow constructor fails when there are pointers involved.

I have taken a simple example to explain my point.

```
class student
{
private:
    int *p = new int[3];
public:
    student()
    {
        p[0]=0; p[1]=0; p[2]=0;
    }
    void modify(int x)
    {
        p[0]=x; p[1]=x; p[2]=x;
    }
    void display()
    {
        cout<<p[0]<<" "<<p[1]<<" "<<p[2]<<" "<<endl;
    }
};

int main()
{
    student s1;
    s1.display();
    student s2=s1; // copy
    s2.display();
    s1.modify(10); // we have modified S1 here but not S2.
    s2.display();  // when we display object S2 , it is supposed to be 0 0 0 only , but 10 10 10 is done
// this is because both the pointers or s1 and s2 are pointing to the same array of s2 , a new array was
    not created for S2 specifically , that was the problem.
}
```

```
0 0 0
0 0 0
10 10 10
Program ended with exit code: 0
```

Observe the display in the bottom right corner.

As we see a new array was not being created for S2 object, and both the pointers of S1 and S2 object were pointed to the same array.

This is the problem with shallow copy constructor.

```
20 class student
21 {
22 private:
23     int *p = new int[3];
24 public:
25     student()
26     {
27         p[0]=0; p[1]=0; p[2]=0;
28     }
29     student (student &s)
30     {
31         p = new int[3];
32         p[0] = s.p[0];
33         p[1] = s.p[1];
34         p[2] = s.p[2];
35     }
36     void modify(int x)
37     {
38         p[0]=x; p[1]=x; p[2]=x;
39     }
40     void display()
41     {
42         cout<<p[0]<<" "<<p[1]<<" "<<p[2]<<" "<<endl;
43     }
44 };
45
46 int main()
47 {
48     student s1;
49     student s2=s1;
50     s2.display();
51     s1.modify(5);
52     s2.display();
53 }
54
55
```

```
0 0 0
0 0 0
Program ended with exit code: 0
```

Here I used a deep copy constructor in line 29 and the problem was avoided, S2 has its own array to point to, that is why this time after we call the modify function to s1, s2 display still showed 0 0 0, unlike the case with the copy constructor.

I have given a simple example, hope you understood.

https://www.geeksforgeeks.org/copy-constructor-in-cpp/

refer the link if you have not.

Destructors

They delete an object when the object goes out of scope.

https://www.geeksforgeeks.org/destructors-c/

Polymorphism

Polymorphism means many forms. The classic examples are humans! We show different forms according to the situation and the people we are around!

we are the same person, but we tend to show a different form at different times.

Under polymorphism there are 3 things to pay attention to

1)function overloading (compile time)

2) function overriding (run time)

3) operator overloading

Function overloading, essentially the functions have the same name but have different parameters.

```
class student
{
public:
    int sum(int x , int y)
    {
        return x + y;
    }
    int sum(int x,int y,int z)
    {
        return x + y + z;
    }
    int sum(double x,double y)
    {
        return x+y;
    }
};

int main()
{
    student s;
    s.sum(1,2);
    s.sum(1,2,3);
    s.sum(2.45,2.3);
}
```

This is a classic example of function overloading, YES, we can do the following without classes also, but I wanted to show this is how it is done with classes.

Function overriding (runtime polymorphism)

Functions must have the same name, return type and parameters and parameters as well for function overriding. this concept comes under inheritance.

The point being let's say the base or parent class has a. function display() implemented in it , at the same time even the derived class has a function display() Implemented in it. Then which display will be executed when we call the display function in the derived class?

Keep in mind that when we inherit, the derived class can also use the functions which are there in the base class also, that is the whole point of inheritance.

So, what happens in the question asked is, if the function display was there only in base class and not derived class, and we call display with object of derived class then the display function of the base class will be called (inheritance, reusability). But see the code below.

```
class A
{
public:
    void display()
    {
        cout<<" THIS IS CLASS A "<<endl;
    }
};
class B:public A
{
public:
    void display()
    {
        cout<<" THIS IS CLASS B "<<endl;
    }
};
int main()
{
    A obj1;
    obj1.display();
    B obj2;
    obj2.display();
}
```

```
THIS IS CLASS A
THIS IS CLASS B
```

Since there is a display function present and implemented in the class B, this time when we call display, the display function of the base class will be overridden by the display function of the derived class.

Hope it’s clear now!

Quick note: Refer to Java Point if you have difficulty in understanding anything we discussed.

Virtual functions

Virtual functions are usually declared in the base class and they are overridden in the derived class.

https://www.geeksforgeeks.org/virtual-function-cpp/

refer the link above, try to pay attention to the concept of base class pointer and derived class object.

```
class A
{
public:
    void display()
    {
        cout<<" THIS IS CLASS A "<<endl;
    }
};
class B:public A
{
public:
    void display()
    {
        cout<<" THIS IS CLASS B "<<endl;
    }
};
int main()
{
    A *p; // base class pointer.
    B obj; // derived class object.
    p->display();                                   Variable 'p' is uninitialized when used here
// basically using pointer it is basically same like object using only , instead of ' . ' use ' -> ' but
    it has an advantage here.
    p = &obj; // p is pointing to  the object of B class.
    p->display(); // still the display of class A is called here.

}
```

```
THIS IS CLASS A
THIS IS CLASS A
```

But now see what happens when I add a virtual key word before display in the base class.

```
18
19
20  class A
21  {
22  public:
23      virtual void display()
24      {
25          cout<<" THIS IS CLASS A "<<endl;
26      }
27  };
28  class B:public A
29  {
30  public:
31      void display()
32      {
33          cout<<" THIS IS CLASS B "<<endl;
34      }
35  };
36  int main()
37  {
38      A *p; // base class pointer.
39      B obj; // derived class object.
40      p = &obj; // p is pointing to  the object of B class.
41      p->display(); // now the display funtion of B is called.
42
43  }
44
45
```

So that is how virtual function, the main advantage is that, instead of B class, assume we have a lot of derived class implemented as B class. Then just using

one point of A (base class) we can call any derived class display() function using this virtual function concept.

Pure virtual functions and abstract classes

In pure virtual function, it does not have any implementation, we just define it and keep it and assign zero.

A class with at least 1 pure virtual function is an abstract class.

Also remember we cannot create an object of an abstract class.

Also, if a pure virtual function is present in the base class, it MUST be overridden in the derived class or else even the derived class will be an abstract class.

So, in a way we can group out all the similar classes using the abstract classes, as all those classes MUST implement the pure virtual functions in the base class.

https://www.geeksforgeeks.org/pure-virtual-functions-and-abstract-classes/

Interview Question asked to me:

He just asked what is the difference between C and C++. Simple right?

I said, C++ has OOP and STL so things become easier but the major advantage is OOP.

He asked what advantage does OOP offer.

I said, sir, it offers things like Abstraction, Inheritance, blah blah blah and things get easier etc.

He asked me to explain with an example.

I tried to give an example of function overloading in polymorphism but he stopped me and said, don't explain individual concepts, tell me an example so that I can get the crux of it. Many people faced the same question and even though it is very simple and everyone knows the answer, it came down to presentation skill and giving an example.

I took a moment and answered it as follows:

Remember the negative age example? That is what I used.

Sir, let's consider a hypothetical scenario where I am a service provider writing a class and whoever is creating an object of my class in main is my consumer. Let us say I have made a class of student with name, roll.no

and age as data members. So, if I use a structure as in C, I will have no control over the data members. The user can enter age as -1, there is nothing stopping him. Instead if I use a class and make those members private, he must use my member function to update the age. And when he enters a negative age, we can give out an output saying age cannot be negative. Thus, the data entered is more realistic. Here users experience is also improved. Here we are binding together the data members and data functions, this is called encapsulation. We are hiding what process is going on when user uses a member function, this is called abstraction. Similarly, we can also use Polymorphism and Inheritance in more complex situations. And also, we can write a cout statement in constructor showing what all services my class can offer whenever an object is created. So, user will know what all he can do with my class and thus user interface is improved.

After this example the interviewer was thoroughly impressed and did not ask any questions on OOP anymore. So, yeah try to explain with an example.

DBMS

It is a very important concept and you will better understand in the sources mentioned by us rather than us trying to explain.

There are a lot of video lectures on YouTube. We followed the one by gate smashers:

Source1:

https://www.youtube.com/watch?v=kBdlM6hNDAE&list=PLxCzCOWd7aiFAN6I8CuViBuCdJgiOkT2Y

I have watched up to 93rd video and then watched this one for b+ trees:

https://www.youtube.com/watch?v=aZjYr87r1b8

There is one more very famous source, by Sanchit Jain:

Source 2:

https://www.youtube.com/watch?v=eTiP-H9GQ30&list=PLmXKhU9FNesR1rSES7oLdJaNFgmuj0SYV

After this you need to practice SQL queries, for that w3schools is a good platform:
https://www.w3schools.com/sql/

Go through the exercise section for every command!

Google for most frequent DBMS questions and try to answer them! And, that is it you are good to go!

We really hope we were of some help to you!

All the very best guys, Thank you!!