Cs301 LetureWise Question from 1 to 22 Lecture

Lecture No 1:

What are the constraints ?

Constraint defines a limitation or restriction on resources of a system. In the computer, we have hard disk, memory and other hardware. We also have time to consider as a resource.
Suppose you are writing an algorithm to solve a given problem and you have disc space of 400 Mega bytes. So, here you have space constraint that your algorithm should use only the available space in an efficient way. A solution is considered to be an efficient solution if it solves the problem within its resource constraints.

What is lvalue?

In an assignment statement, lvalue refers to some memory location where some value can be stored. Lvalue variable is written on left hand side of an assignment statement. For example, in the following statement, variable 'var' is lvalue variable and value 7 will be stored in it.
int var= 7;

What is "current" pointer ?

"Current" is a variable showing the status of currently focused element in array/list, for example if current = 5, it means that currently the focus is on element no. 5 in the array/list.

What is algorithem?

Algorithm is a finite sequence of steps for the solution of a problem.

WHAT IS ROLE OF ARRAYS IN DATA STRUCTURE?

Array is the most widely used data structure. It is also used as constituting integral part of other data structures.
Array is mainly used to store similar data. For storing elements of indexed data, the particular method is array. For example, if you were storing information about each day in August, you would create an

array with an index capable of addressing 31 values -- one for each day of the month. Think of a notebook with pages numbered 1 through 12. Each page may or may not contain information on it. The notebook is an array of pages. Each page is an element of the array 'notebook'.
Arrays can also be multidimensional - instead of accessing an element of a one-dimensional list, elements are accessed by two or more indices, as from a matrix or tensor.
Arrays guarantee constant time read and write access,O(1) , however many lookup operations (find_min, find_max, find_index) of an instance of an element are linear time, O(n). Arrays are very efficient in most languages, as operations compute the address of an element via a simple formula based on the base address element of the array.

HOW MANY TYPES AN DATA CAN BE ORGANIZE?

Data can be oranized in two main ways.
1- Linear as in arrays, lists
2-Non Linear or Hierarchical like Trees

What is important of data structure in our today's life?

Data structures importance/applications in Daily life :

In bank service rows of men follow the Queue data structure.
In online shopping, selection and cancelation in the list of items is done/maintained via data structures.
For representing a city region telephone network.
Used to store a set of fixed key words which are referenced very frequently.
DS are used to represent an image in the form of a bitmap.
To implement back functionality in the internet browser.
To implement printer spooler so that jobs can be printed in the order of their arrival.
To record the sequence of all the pages browsed in one session.
To implement the undo function in a text editor.
To store information about the directories and files in a system

What is chunk?

Chunk means "a piece of reasonable size". Computer memory is divide into small parts. Combination (allocation in sum) of these parts constitutes and can be called as chunk.

What is Basic purpose of data structure?

The main purpose of data structure is to organize and store the Data in Computer, in order to make the use of data efficient. Use of data efficiently means to use less space by the data and operations like searching, deletion, insertion etc on data took less time.

What are argc and argv ?

The variables argc and argv in the definition of main function, that is, main(int argc, char argv) are called command line arguments,**

What is the difference between the Data structure and Algorithms?
Data Structure is the organization of Data in computer in an efficient way, while Algorithm is a finite sequence of steps for the solution of a problem.

Example,
Consider there are a number of students whose marks are needed to be store in the computer, now we can stored them in computer in an efficient way by using data structure like array, Linked List etc..
Consider we need to find the maximum marks obtained by a student in this list, now for this we need to use an algorithm (Sequence of steps), for example,

What is efficiency?
A solution is said to be efficient if it solves the problem within its resource constraints.
Space
Time
The cost of a solution is the amount of resources that the solution consumes.
What is *x*?

X is a name of collection of items. Its individual items are numbered from zero to one less than array size. To access a cell, use the array name and an index as under:

x[0], x[1], x[2], x[3], x[4], x[5]

What is Array Name?
-'x' is an array name but there is no variable x.
'X' is not an 'l' value.
For example, if we have the code
Int a, b;
Then we can write
b=2;
a=b;
a=5;
But we cannot write 2=a;

List Data Structure:
What are lists?
A list is collection of items that are all of the same type (grocery items, integers, names)

Operation Name	**Description**
createList()	**Create a new list (presumably empty)**

copy()	**Set one list to be a copy of another**
clear();	**Clear a list (remove all elements)**
insert(X, ?)	**Insert element X at a particular position in the list**
remove(?)	**Remove element at some position in the list**
get(?)	**Get element at a given position**
update(X, ?)	**Replace the element at a given position with X**
find(X)	**Determine if the element X is in the list**
length()	**Returns the length of the list.**

If we use the "current" marker, the following four methods would be useful:

Functions	**Description**
start()	**Moves the "current" pointer to the very first element**
tail()	**Moves the "current" pointer to the very last element**
next()	**Move the current position forward one element**
back()	**Move the current position backward one element**

-

What are dynamic arrays?

In computer science, a dynamic array, grow able array, resizable array, dynamic table, or array list is a data structure that can be resized and allows elements to be added or removed. It is supplied with standard libraries in many modern mainstream programming languages.

A dynamic array is not the same thing as a dynamically-allocated array, which is a fixed-size array whose size is fixed when the array is allocated.

what is the majour difference between "Array" and "List"?

Array is a collection of memory cells of same type. In computer science, an array data structure or simply array is a data structure consisting of a collection of elements (values or variables), each identified by one or more integer indices, stored so that the address of each element can be computed from its index topple by a simple mathematical formula.For example, an array of 10 integer variables, with indices 0 through 9, may be stored as 10 words at memory addresses 2000, 2004, 2008, … 2036; so that the element with index i has address 2000 + 4 × i

In computer science, a linked list is a data structure that consists of a sequence of data records such that in each record there is a field that contains a reference (i.e., a link) to the next record in the sequence.

what is the basic concept of Data Structures?

This is very important subject as the topics covered in it will be encountered by you again and again in the future courses. Due to its great applicability, this is usually called as the foundation course. You have already studied Introduction to programming using C and C++ and used some data structures. The focus of that course was on how to carry out programming with the use of C and C++ languages besides the resolution of different problems. In this course, we will continue problem solving and see that the organization of data in some cases is of immense importance. Therefore, the data will be stored in a special way so that the required result should be calculated as fast as possible what is list data structure ?

In computer science, a list or sequence is an abstract data structure that implements an ordered collection of values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a finite sequence, that is, a tuple. Each instance of a value in the list is usually called an item, entry, or element of the list; if the same value occurs multiple times, each occurrence is considered a distinct item.

-

what parameters are considered to identify l value.?
If we can assign a value to a variable then we can the variable is lvalue type. Lets see with an examples

int x;
Here x is a integer type variable we can assign any integer value to x like x = 10 or x = 2

In the following code statement y is an array which can hold 10 values.
int y[10];

In this scenario y is not lvalue, the reason is we cannot assign a single integer value to y like y = 10

Lets we elaborate it with diagram.

Suppose the box given below is int variable x. we can easily assign (save) 10 or 2 to it.

.

Assume 10 boxes given below are representing an array y. We cannot assign single value 10 or 2 to y. The problem is, if we will assign 10 to y then which box (place) in array it will save. This is the reason array is not lvalue.

data structure concept in easy wording
A data structure is a group of data elements grouped together under one name. Data structure helps to store the data in computer in an organized and efficient way. Data structures are used to efficiently utilize the memory of computer. Data structures not only consider the items stored but also the relationship to each other.

-

what is Difference b/w arrays and linked list?

The difference between arrays and linked lists are:

- Arrays are linear data structures. Linked lists are linear and non-linear data structures.
- Linked lists are linear for accessing, and non-linear for storing in memory
- Array has homogenous values. And each element is independent of each other positions. Each node in the linked list is connected with its previous node which is a pointer to the node.

- Array elements can be modified easily by identifying the index value. It is a complex process for modifying the node in a linked list.
- Array elements can not be added, deleted once it is declared. The nodes in the linked list can be added and deleted from the list.

•

Lecture no 2:

What are Linked lists ?

Linked lists are a way to store data with structures so that the programmer can automatically create a new place to store data whenever necessary. Specifically, the programmer writes a struct definition that contains variables holding information about something and that has a pointer to a struct of its same type (it has to be a pointer--otherwise, every time an element was created, it would create a new element, infinitely). Each of these individual structs or classes in the list is commonly known as a node or element of the list.

Why we need array to implement this linked memory list?

Using linked memory method is the other way of implementing the list. We have to implement a list of elements, so this implementation can be done in 2 ways. First is array implementation in which the elements will be stored in contiguous memory locations. There are certain limitations of this type of implementation of list i.e. the size is fixed. So to avoid these limitations, we can implement list by other method. That method is implementing via linked memory locations which do not restrict the list items to be stored contiguously.

•

what is the difference b/w array that is dynamic allocated and that is simple ?

**Simple array (Static) is created in stack and dynamic is created in heap area of the memory.
In static array you have to tell the size before program runs but in dynamic you can take input from user and then make array of that size.**

Example static: int array[6];

**Example dynamic: int *array;
cout > size;
array = new int[size];**

What is the difference b/w array and list data structure?

Array is collection of homogeneous elements. List is collection of heterogeneous elements. For Array memory allocated is static and continuous while for List memory allocated is dynamic and Random.
In array, user need not have to keep in track of next memory allocation while in list user has to keep in Track of next location where memory is allocated.
Array size should be specified but the list size can be determined at run time also.

-

How can we set current position ?

"current" is a variable showing the status of currently focused element in array/list, for example if current = 5, it means that currently the focus is on element no. 5 in the array/list.

You can simply assign any array index to the variable "current" like, current = 3;

where 3 is an array index.

How can we shift element left or right?

This is a programming way that how can we shift element in an array.

For example, consider the following small program,

```
#include <iostream.h>

#include <conio.h>

main()

{

int a[5] = { 10, 20, 30, 40, 50};

for(int i =0;i<5;i++)

{

couta[i] '\t' ;

}
coutendl;

// after shifting

int first = a[0]; // Save 1st element or delete

for(int i =0;i<4;i++)

{

a[i] = a[i+1]; // Real shifting statement.

couta[i] '\t' ;

}
getche();

return 0;

}
```

-

Following is the list of operations and their description:

Operation Name	Description
createList()	Create a new list (presumably empty)
copy()	Set one list to be a copy of another
clear();	Clear a list (remove all elements)
insert(X, ?)	Insert element X at a particular position in the list
remove(?)	Remove element at some position in the list
get(?)	Get element at a given position
update(X, ?)	Replace the element at a given position with X
find(X)	Determine if the element X is in the list
length()	Returns the length of the list.

what is the main difference between array and list?

In computer science, an array data structure or simply array is a data structure consisting of a collection of elements (values or variables), each identified by one or more integer indices, stored so that the address of each element can be computed from its index tuple by a simple mathematical formula. For example, an array of 10 integer variables, with indices 0 through 9, may be stored as 10 words at memory addresses 2000, 2004, 2008, … 2036; so that the element with index i has address 2000 + 4 × i.
In computer science, a list or sequence is an abstract data structure that implements an ordered collection of values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a finite sequence, that is, a tuple. Each

instance of a value in the list is usually called an item, entry, or element of the list; if the same value occurs multiple times, each occurrence is considered a distinct item.

what is dynamic memory allocation ?

In computer science, dynamic memory allocation (also known as heap-based memory allocation) is the allocation of memory storage for use in a computer program during the runtime of that program. It can be seen also as a way of distributing ownership of limited memory resources among many pieces of data and code.Dynamically allocated memory exists until it is released either explicitly by the programmer, or by the garbage collector. This is in contrast to static memory allocation, which has a fixed duration. It is said that an object so allocated has a dynamic lifetime.

what is the main difference between list implementation and add method?

Suppose we want to create a list of integers. For this purpose, the methods of the list can be implemented with the use of an array inside. For example, the list of integers (2, 6, 8, 7, 1) can be represented in the following manner where the current position is 3.

A	2	6	8	7	1	**Current**	**Size**
	1	2	3	4	5	3	5

In this case, we start the index of the array from 1 just for simplification against the usual practice in which the index of an array starts from zero in C++. It is not necessary to always start the indexing from zero. Sometimes, it is required to start the indexing from 1.

<u>Add Method</u>
Suppose there is a call to add an element in the list i.e. *add(9)*. As we said earlier that the current position is 3, so by adding the element 9 to the list, the new list will be (2, 6, 8, 9, 7, 1).
To add the new element (9) to the list at the current position, at first, we have to make space for this element. For this purpose, we shift every element on the right of 8 (the current position) to one place on the right. Thus after creating the space for new element at position 4, the array can be represented as

A	2	6	8		7	1	**CurrentSize**	
	1	2	3	4	5	6	4	6

Now in the second step, we put the element 9 at the empty space i.e. position 4. Thus the array will attain the following shape. The figure shows the elements in the array in the same order as stored in the list.

A	2	6	8	9	7	1	**CurrentSize**	
	1	2	3	4	5	6	4	6

We have moved the current position to 4 while increasing the size to 6. The size shows that the elements in the list. Where as the size of the array is different that we have defined already to a fixed length, which may be 100, 200 or even greater.

•

WHat is a difference between List(sequence),Array and pointer?

A list or sequence is an abstract or conceptual data type that implements an ordered collection of values or items. In order to practically implement this abstract or conceptual data type we must use some strategy like an array or linked list. So we use either arrays or linked list for the implementation or manipulation of List.

Whereas a pointer is a special type of variable in which a memory address is stored. Pointer contains a memory address, not the value of the variable.

An ***array*** is a data structure in which identical data types are stored. Every array has a data type i.e. name and size. Data type can be any valid data type. The size of the array tells how many elements are there in the array. The arrays occupy the memory depending upon their size and have contiguous area of memory. We can access the arrays using the array index.

•

Difference between Programming and Data Structure?

Programming is very wide topic and covers lots of computer science concepts. In simple words programming is developing software to automate or solve a problem, with the help of different data structures and algorithms.

While data structure is a part of programming. Data structures are instructions to solve a problem by using computer resources efficiently.

•

What are LValues Variables?

Lvalue variables are those variables that can used on left side to save a value.

Variables are labels of memory locations that we want to use in our program. Assigning a value to variable is saving value on a memory location.

In following example

int x = 10;
Here x is Lvalue and holding a value 10.

int y[10];
Here y is not Lvalue and we cannot write y = 20;
Because y is an array and can be used to hold 10 values. In above line out program will not know on which location it should save value 20.

What is Index?

Index is location of element in an array. With the help of index we can reach to any element in the array. Here is example of using index

int grades[10]; // declaring an integer array to store 10 integer values
grades[4] = 3; // assigning integer 3 to 5th location of array. here 4 is index of element.

- What do u meant By Link Memory?

Linked memory means linked list. A linked list is a data structure consisting of a group of nodes which together represent a sequence or list. Linked list is a data structure in which each node contains data field and address of the next node i.e. a pointer to the next node. Linked list is also called linked memory because in linked list, different memory cells are linked together by use of pointers.

- How we can analyze Array List?

We analyze array list (i.e. a list implemented by using an array) in order to check the performance of array list for different cases of adding, removing and finding an element from array list.

<u>Add:</u>
Worst case: When we will add an element at the start of an array then we should have to move every element to the right to make space for the new element.

Best case: When we will add the element at the end of an array then we will not move or shift any other array element.

Average cases: Normally we may have to move half of the elements.

<u>Find:</u>
Worst case: Find method takes an element and traverses the list to find that element. The worst case of the find method is that it has to search the entire list from beginning to end. So, it finds the element at the end of the array

Best case: If the element that we want to find is at the start of an array then we should not have to search the remaining array to find the element.

Average cases: Normally we may have to search half of the elements.

<u>Remove:</u>
Worst case: When we will remove an element from the start of an array then we should have to move every element to the left to occupy blank space of deleted element

Best case: When we will remove the element from the end of an array then we will not move or shift any other array element.

Average case: Normally we may have to move half of the elements.

- What is Add method and Current Pointer?

Add method is used to add an element in the list. A list can be implemented either by using an array or linked list. When we implement a list by using an array then it is called array list. When we will add an element at the start of an array list then we should have to move every element to the right to make space for the new element. When we will add the element at the end of an array list then we will not move or shift any other array element. Normally we may have to move half of the elements in average cases.

Current marker or pointer refers to a particular position in the list where we are currently standing. If we do not use current pointer then we will not be able to keep track of our list i.e. we will be unable to identify our current position so it will become difficult for us to add or remove an element from list.

What do we meant by Link list inside a computer memory ?

It means how the memory will be allocated to the linked list and how memory is linked for your list elements.

Lecture no 3:

What is Node?

These are the elements that make it possible to have a linked list. Without them, there would be no listing.

Each node contains two parts; a part for the data and a part for the address of the next element. As you can see, this will use pointers.

- **What is state variables?**
The products that are made by the factory have their own characteristics. For example, a car made by an automobile factory has an engine, wheels, steering and seats etc. These variables inside a class are called state variables.

- **getNext()**
getNext() which returns a pointer to an object of type Node lying somewhere in the memory. It returns nextNode i.e. a pointer to an object of type Node. NextNode contains the address of next node in the linked list.

setNext()

setNext() that accepts a pointer of type Node, further assigned to nextNode data member of the object. This method is used to connect the next node of the linked list with the current object. It is passed an address of the next node in the linked list.

- **What are list interface, list implementation and analysis?**

List interface means that how list works internally and how we interact with the list. If write the C++ program to implement the list then we simply need to know that what are those methods, their names and what arguments that method takes etc.

Now we will see what the implementation of the list is and how one can create a list in C++. After designing the interface for the list, it is advisable to know how to implement that interface. Suppose we want to create a list of integers. For this purpose, the methods of the list can be implemented with the use of an array inside. For example, the list of integers (2, 6, 8, 7, 1) can be represented in the following

manner where the current position is 3.

A	2	6	8	7	1	**Current**	**Size**
	1	2	3	4	5	**3**	**5**

In this case, we start the index of the array from 1 just for simplification against the usual practice in which the index of an array starts from zero in C++. It is not necessary to always start the indexing from zero. Sometimes, it is required to start the indexing from 1. For this, we leave the zeroth position and start using the array from index 1 that is actually the second position. Suppose we have to store the numbers from 1 to 6 in the array. We take an array of 7 elements and put the numbers from the index 1. Thus there is a correspondence between index and the numbers stored in it. This is not very useful. So, it does not justify the non-use of zeroth position of the array out-rightly. However for simplification purposes, it is good to use the index from 1.

- **What is the main purpose of node?**

For the utilization of the concept of linked memory, we usually define a structure, called linked list. To form a linked list, at first, we define a node. A node comprises two fields. i.e. the object field that holds the actual list element and the next that holds the starting location of the next node.

What is the importance of link list in memory?

Linked lists are among the simplest and most common data structures, and are used to implement many important abstract data structures, such as stacks, queues, hash tables, symbolic expressions, skip lists, and many more.

The principal benefit of a linked list over a conventional array is that the order of the linked items may be different from the order that the data items are stored in memory or on disk. For that reason, linked lists allow insertion and removal of nodes at any point in the list, with a constant number of operations

what are the operations of the linked list?

The linked list data structure provides operations to work on the nodes inside the list.
The first operation we are going to discuss here is to create a new node in the memory. The Add(9) is used to create a new node in the memory at the current position to hold '9'. You must remember while working with arrays, to add an element at the current position that all the elements after the current position were shifted to the right and then the element was added to the empty slot.
Here, we are talking about the internal representation of the list using linked list. Its interface will remain the same as in case of arrays.
We can create a new node in the following manner in the add() operation of the linked list with code in C++:
Node * newNode = new Node(9);
The first part of the statement that is on the left of the assignment is declaring a variable pointer of type Node. It may also be written as Node * newNode. On the right of this statement, the new operator is used to create a new Node object as new Node(9). This is one way in C++ to create objects of classes. The name of the class is provided with the new operator that causes the constructor of the class to be called. The constructor of a class has the same name as the class and as this a function, parameters can also be passed to it. In this case, the constructor of the Node class is called and '9' is passed to it as an int parameter.
Hence, the whole statement means:
"Call the constructor of the Node class and pass it '9' as a parameter. After constructing the object in memory, give me the starting memory address of the object. That address will be stored in the pointer variable newNode."

- ***<u>Limitation of array and use of linked list</u>***

There are number of limitations of array data structure. Most common are as follows:-

1. **Once an array is created, its size cannot be altered**
2. **Array provides inadequate support for inserting and deleting operations.**

A linked list is a popular data structure to store data in sequential order. Meanwhile, linked list structure allow following operations which overcomes limitation of array: for example

1. **Retrieve an element from list.**
2. **Insert a new element to the list.**

3. Delete an element from the list.
4. Find how many elements are in the list.
5. Find a specific element is in the list.
6. Find if this list is empty.

what is constant pointer?

Let's look at the use of const. Consider the following line of declaration:

int *const myptr = &x ;

The right hand side of this assignment statement could be read as, myptr is a constant pointer to an integer. Whenever we use the keyword const with a variable, the value of that variable becomes constant and no other value can be assigned to it later on. We know that when we declare a constant variable like const int x ; it is necessary to assign a value to x and we write const int x = 10 . After this, we cannot assign some other value to x. The value of x can not be changed as it is declared as a constant. Now consider the previous statement

int *const myptr = &x ;

Here we declare a constant pointer to an integer. Being a constant pointer, it should immediately point to something. Therefore, we assign this pointer an address of a variable x at the time of declaration. Now this pointer cannot be changed i.e. now we cannot assign address of any other variable to pointer myptr at any stage. The pointer myptr will hold the address of variable x throughout the program.

- **What are constructors and Its use?**

Constructor is a special function of a class, which is called every time a new object of the class is created. Constructor has the same name as of the class.
It is basically to initialize the object of the class. This function does not have return type. If you do not include an explicit constructor, the compiler will call a default implicit constructor.

- **What is command line arguments?**
main() is function from which the execution of the program is started. Your program can not be executed if main is missing in it.

main(int argc, char argv)**
Both of arguments in main are used to manage command line arguments passed to our program. The first argument argc (arguments count) gives the number of command line arguments passed. While second argument argv (arguments vector) contain actual command line arguments passed.

Example:

For example

if we are executing our program Test.exe we will use this command in cmd.
Test.exe

If we want to pass arguments while executing our program, we can write this command.
Test.exe abc def gh i 1 5 xyz
Here Test.exe is our program executable file and others are command line arguments we are passing during execution.

C++ will consider Test.exe as command line argument too. Now argc variable will hold 7 as it is number of command line arguments and argv will hold all command line arguments we passed.

- **How we can add(5) in link-list but on ascending order ?**
First of all, you will search the correct position of the new node and then insert it into that position. To search the correct position, you will start form the first node. Compare the data of first node with the new node, if it is smaller than new data, move one node forward. Repeat the process until you find the node with the greater data than new data. At the end of search, current pointer will point to the desired node. Make a connection of previous node of current node with new node, and place current node reference in new node.

Lecture no 4:

What is the Josephus problem (or Josephus permutation) ?

In computer science and mathematics, the Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game.
There are people standing in a circle waiting to be executed. After the first man is executed, certain number of people are skipped and one man is executed. Then again, people are skipped and a man is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last man remains, who is given freedom.

- What is the difference between doubly link list and single link list?

If you look at single link list, the chain is seen formed in a way that every node has a field next that point to the next node. This continues till the last node where we set the next to NULL i.e. the end of the list. There is a headNode pointer that points to the start of the list. We have seen that moving forward is easy in single link list but going back is difficult. For moving backward, we have to go at the start of the list and begin from there. Do you need a list in which one has to move back or forward or at the start or in the end very often? If so, we have to use double link list.

In doubly-link list, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node. Now our node factory will create a node with three parts. First part is prev i.e. the pointer pointing to the previous node, second part is element, containing the data to be inserted in the list. The third part is next pointer that points to the next node of the list. The objective of prev is to store the address of the previous node.

How we come to know which one is current point?

current is a marker or pointer to refer to a particular position in the list.

We have a variable named current.
Now in the start, when our list is empty, the current has value 0 i.e; current is pointing to the 0th location of array which is empty.
When we want to insert first item into the list, we first increment value in the current, so now current has value 1 i.e; current is pointing to the 1st location of array. So here we insert the first item.
That is how we keep track of the current item in the array list.

Consider the following example to understand the current position.
Suppose you are reading a page and using your finger to point at the word you are reading. As you read the next word in the line, you move your finger towards next. Now at any time, if you or someone wants to check which word you are reading, simply check which word your finger is pointing at.

What are Arrays advantages over Linked list?

Arrays advantages over Linked list:
Arrays allow random access to its elements and thus the complexity is order of O(1).
Linked lists allow only sequential access to elements. Thus the algorithmic complexity is order of O(n).

Arrays do not need an extra storage to point to next data item. Each element can be accessed via indexes.
Linked lists require an extra storage for references. This makes them impractical for lists of small data items such as characters or Boolean values.

What are Linked list advantages over Arrays?
Linked list advantages over Arrays:
The size of array is restricted to declaration. If you want to increase array size, declare a new array of larger size and then copy old array elements to this new array.
The size of Linked lists is dynamic by nature.

If you want to delete something from the middle of array, you have to move half of the array to the left. But in case of Linked list, you only move two or three pointers. Same is the case with insertion.

<u>Circularly-linked lists</u>
The next field in the last node in a singly-linked list is set to NULL. The same is the case in the doubly-linked list. Moving along a singly-linked list has to be done in a watchful manner. Doubly-linked lists have two NULL pointers i.e. prev in the first node and next in the last node. A way around this potential hazard is to link the last node with the first node in the list to create a circularly-linked list.

What is Node?

We form a link list by joining nodes, a node consists of two fields first is used to store the actual object value and second is used to store a reference to the next node. infect a linked list is a chain of these nodes and each node have a pointer or reference to the next node while the last node have a null pointer.

difference between Find and back in linked list.?

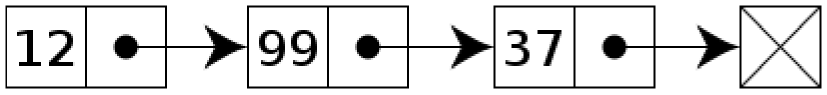
Find() method uses to find an element in a data structure whereas Back() function of the list will move the current pointer one item back, for example, if the current pointer point to item no 3 of the list and we call the back() function of this list, after this call the current pointer (means current item) will be item no. 2. Find() and Back are not one step operations, it is because sometimes during Find() operation we may have to search the entire list in order to find some particular element say x whereas in the back() method, we move the current pointer one position back. Moving the current pointer back, one requires traversing the list from the start until a node reach, whose next pointer points to current node.

how we know our current node in circulatatry- linked list?

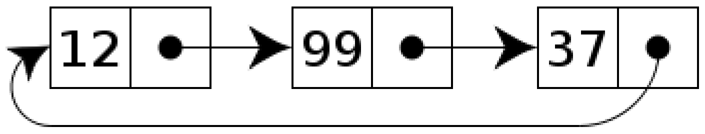
Suppose that you are given a linked list that is either circular or or not circular (another word for not circular is acyclic). Take a look at the figures below if you are not sure what a circular linked list looks like.

Write a function that takes as an input a pointer to the head of a linked list and determines whether the list is circular or if the list has an ending node. If the linked list is circular then your function should return true, otherwise your function should return false if the linked list is not circular. You can not modify the linked list in any way.

This is an acyclic (non-circular) linked list:



This is a circular linked list:



You should start out this problem by taking a close look at the pictures of a circular linked list and a singly linked list so that you can understand the difference between the 2 types of lists.

The difference between the 2 types of linked lists is at the very last node of the list. In the circular linked list, you can see that the very last node (37) links right back to the first node in the list – which makes it circular. However, in the acyclic or non-circular linked list you can see that the end node does not point to another node in the linked list and just ends.

It is easy enough to know when you are in an acyclic linked list – the pointer in the end node will just be pointing to NULL. However, knowing when you are in a circularly linked list is more difficult because there is no end node, so your function would wind up in an infinite loop if it just searches for an end node. So, there has to be a solution other than just looking for a node that points to NULL, since that clearly will not work.

-

Diffrence between josephus problem and circularly linked?
In circularly link list, the last node is connected to the head node. Joseph problem is an application of circularly link list. We use circular link list to solve the Joseph problem.

•

Analysis of Linked List
§add

•we simply insert the new node after the current node. So add is a one-step operation.

§remove

§remove is also a one-step operation

§find

§worst-case: may have to search the entire list

§back

§moving the current pointer back one node requires traversing the list from the start until the node

Doubly-linked List
§Moving forward in a singly-linked list is easy; moving backwards is not so easy.

§To move back one node, we have to start at the head of the singly-linked list and move forward until the node before the current.

§To avoid this we can use two pointers in a node: one to point to next node and another to point to the previous node:

Is link list and List same?if yes than explain how?

List and linked list is same thing. List is the most generic data structure. A list is the collection of the same type of items. List can be implemented through array and linked memory. When we implement list through linked memory, it becomes linked list.

•

Difference between Head node and Current Node?

headNode is a node, whose next pointer has a reference of the first node of the list. The data elements of this node do not have any values, except nextNode pointer.

Suppose, we have a list with elements 1, 2, 3, 4, 5,6. 1 is the data element of the first node of the list. headNode's next pointer will have reference of the node with the data element 1.
And current is a pointer variable in the linked list implementation which points to the node currently working upon.

•

Lecture no 5:

What is abstract data ?

There is no term as "abstract data" in data structure. What I perceive from your question is that you want to ask about Abstract Data Type (ADT). An abstract data type is a mathematical model of a certain type of data structure for example a Stack. It is defined in terms of operations on it; and implementation is hidden.

What are Object File and Binary File ?

Object File:
When Compiler compiles a source code file (for example abc.cpp), the resulting compiled file is known as object file (abc.obj). Object files end in ".o" by convention, although on some operating systems (e.g. Windows, MS-DOS), they often end in ".obj". An object file contains function definitions in binary form, but they are not executable by themselves.
Binary File:
The linker links together a number of object files and as a result of that process generates binary files that can be directly executed. Binary executables have no special suffix on Unix operating systems, although they generally end in ".exe" on Windows.

What is the difference between isEmpty() and isFull()?

Two of the methods used in the interface of stack are: isEmpty() and isFull().

isEmpty() is a Boolean function which will return true if stack is empty and return false if it is not empty; means there may be one or more elements in the stack.

isFull() is also a Boolean function which will return a Boolean value. If it returns true, it means that the stack is full and no more elements can be inserted and if it returns false, it means stack is not full and an element can be inserted into it.

What is the difference between an array pointer and an ordinary pointer?

The difference between an array pointer and an ordinary pointer is that the array name is a constant pointer. It always contains the memory address of the first element of the array. As value of constant can never be changed or reassigned any other address so you can not perform an increment operator on an array pointer. Hence, you can not use (*Y)++ and so, in case Y is the name of an array.

-

What are stacks and how are they affected in Data Structure?

A stack is a list with the restriction that inserts and deletes can be performed in only one position, namely the end of the list called the top.
Stack is used in infix to postfix conversion, function calls.

What is Abstract data type (ADT)?

Abstract data type (ADT): A data type whose properties (domain and operations) are specified independently of any particular implementation.
List, for example, is an ADT. The functionality of an ADT can be implemented in many ways. The programs that use these ADTs will not need to know which implementation was used as long as it performs the required functionality.

The List ADT can be implemented through Arrays (elementary data structure that exists as built-in in most programming languages) or linked-structure (list implemented through linked-structure is called Linked-list).

-

Stacks
Definition of stacks
"Stack is a collection of elements arranged in a linear order"
Suppose we have some video cassettes. We took one cassette and put it on the table. We get another cassette and put it on the top of first cassette. Now there are two cassettes on the table- one at the top of other. Now we take the third cassette and stack it on the two. Take the fourth cassette and stack it on the three cassettes.
Now if we want to take the cassette, we can get the fourth cassette which is at the top and remove it from the stack. Now we can remove the third cassette from the stack and so on. Suppose that we have fifty cassettes stacked on each other and want to

access the first cassette that is at the bottom of the stack. What will happen? All the cassettes will fell down. It will not happen exactly the same in the computer. There may be some problem. It does not mean that our data structure is incorrect. As we see in the above example that the top most cassette will be removed first and the new cassette will be stacked at the top. The same example can be repeated with the books. In the daily life, we deal with the stacked goods very carefully.

<u>Abstract, dynamic data type and stack.</u>

Abstract data type:
If for a particular collection of data only the structure of data and the functions to be performed on the data is defined but the implementation is not defined,then such a collection of data is called Abstrct data type. It means "Only decleration is there but implementation is not i.e it never uses further but it can be inherited ".

Dynamic data:

Dynamic data or transactional data denotes information that is asynchronously changed as further updates to the information become available.The opposite of this is persistent data, which is data that is infrequently accessed and not likely to be modified. Dynamic data is also different from streaming data, in that there is no constant flow of information. Rather, updates may come at any time, with periods of inactivity in between.

Stack:
in computer science, a stack is a last in, first out (LIFO) abstract data type and data structure. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: push and pop. The push operation adds to the top of the list, hiding any items already on the stack, or initializing the stack if it is empty. The pop operation removes an item from the top of the list, and returns this value to the caller. A pop either reveals previously concealed items, or results in an empty list. Stacks have many applications. For example, as processor executes a program, when a function call is made, the called function must know how to return back to the program, so the current address of program execution is pushed onto a stack. Once the function is finished, the address that was saved is removed from the stack, and execution of the program resumes. If a series of function calls occur, the successive return values are pushed onto the stack in LIFO order so that each function can return back to calling program. Stacks support recursive function calls in the same manner as conventional non recursive calls.

<u>Difference between stack and Linked list:?</u>
A stack is an abstract or conceptual data type that implements a collection of values or items in which the principal (or only) operations on the collection are the addition of an entity/element to the collection, known as push and removal of an entity/element, known as pop. The relation between the push and pop operations is such that the stack is a Last-In-First-Out (LIFO) data structure. In order to practically implement this abstract or conceptual data type we must use some strategy like an array or linked list. A linked list is a data structure consisting of a group of nodes which together represent a stack. So in simple words, stack is an abstract or conceptual data type that linked list is used to implement that abstract data type.

It depends on our requirement that out of different data structures which data structure is best suited for our requirements and project e.g. for some projects linked list is useful whereas for some projects arrays may be your preference. For example, let us consider that we want to develop a real world project in which each task should be completed as early a possible. In this case we should use linked list as compare to arrays because adding or removing an element/node from the middle of a linked list takes less time as compare to arrays. Similarly if we develop a project in which our main concern is to save the memory space as much as possible then we could use arrays instead of linked list.

Difference among ADT and class:?

In simple words, ADT (abstract data type) is a user defined conceptual data types and a class is used to implement this ADT/user defined data type. For example, a stack is an abstract data type and we use to implement this abstract data type by using a stack class

- **Abstract data type:**

it is the collection of values and a set of operation on those values. ADT refers to the basic mathematical concepts. that defines the data type.

- **Abstract Data Types and Object-Orientation**

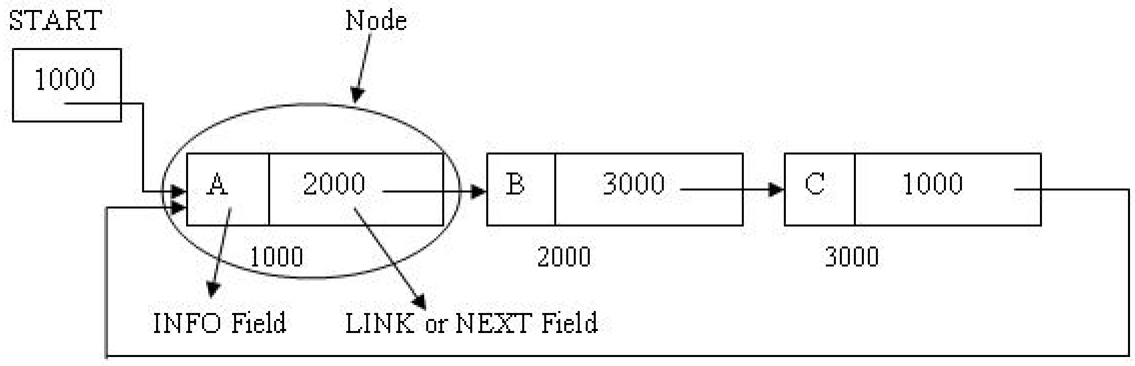
ADTs allows the creation of instances with well-defined properties and behaviour. In object-orientation ADTs are referred to as *classes*. Therefore a class defines properties of *objects* which are the instances in an object-oriented environment.
ADTs define functionality by putting main emphasis on the involved data, their structure, operations as well as axioms and preconditions. Consequently, object-oriented programming is ``programming with ADTs'': combining functionality of different ADTs to solve a problem. Therefore instances (objects) of ADTs (classes) are dynamically created, destroyed and used.

- **What is Linked list or Circular linked list ?**

Linked list is a dynamic data structure that contains a "link" to the structure containing the next item. It is a collection of structures ordered not by their physical placement in memory (like array) but by logical links that are stored as part of the data in the structure itself

In it the last node does not contain NULL pointer. Instead the last node contains a pointer that has the address of first node and thus points back to the first node.

It is shown below:



What are Advantages and Disadvantages of Link list ?

<u>Advantages:</u>

-Dynamic structure (Mem. Allocated at run-time).

- We can have more than one datatype.

- Re-arrange of linked list is easy (Insertion-Deletion).

- It doesn't waste memory.

<u>Disadvantages:</u>

1. It is not easy to reverse the linked list.
2. If proper care is not taken, then the problem of infinite loop can occur.
3. If we at a node and go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node.

Define Types of Linked lists?

Types of Linked List

- Singly or Chain Linked List.

- Doubly or Two Way Linked List.

- Circular Linked List.

- Circular Doubly Linked List.

<u>Singly or Chain Linked List</u>

The way to represent a linear list is to expand each node to contain a link or pointer to the next node. This representation is called a one-way chain or singly linked list.

<u>Circular Linked List</u>

A linked list in which the pointer of the last node points to the first node of the list is called circular linked list.

<u>Doubly or Two-Way Linked List</u>

A linked list which can be traversed both in backward as well as forward direction is called doubly linked list. It uses double set of pointers.

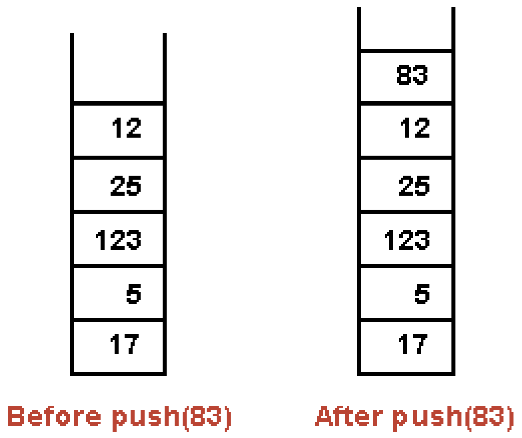
<u>Circular Doubly Linked List</u>

It employs both the forward pointer and backward pointer in circular form.

•

Important thing in Stack ....

In a stack, all operations take place at the "top" of the stack. The "push" operation adds an item to the top of the stack.
The "pop" operation removes the item on the top of the stack and returns it.



What is the difference between A[current++] and A[++current] please Explain because they are totally different when implemented it in the stack?

To understand the difference between above mentioned statements, you have to understand the difference between pre and post increment operator. Let me explain you with an example.

Suppose we have a variable x

int x=10;

Now we will apply pre and post increment operator over it.

cout"x= "x++;

Now in the above statement, I have used post increment operator with x. The result of the above statement will be 10. As in post increment operator, the value is used first and then the increment is applied. So firstly the value is printed and then it is incremented.

cout"x= "++x;

Now in the above statement, I have used pre increment operator with x. The result of the above statement will be 11. As in pre increment operator, the value is first incremented and then used. So firstly the value is printed and then it is incremented.

Now in the statement A[current++], the compiler will assign the value at the current index of array and then the current is incremented. And in A[++current], the value of current id first incremented and then the value is assigned to the new incremented index

.

int isEmpty(){return (current == -1);} // Will return true when stack is empty

int isFull(){ return (current == size-1);}

In the constructor of the class stack, the member variable current is initialized to -1. When an element is inserted in the stack, current is incremented by 1. Now in isEmpty method we check whether the value of current is -1 or not. If the value is -1 the return statement will return true. If stack even has one element, the function will return false.

Stack can only hold elements equal to the size of stack. If the size is 10 then stack can only have 10 elements in it. As we are incrementing the value of current during insertion, so when the value of current reaches the value of size, it means that now there is no more capacity in the stack. isFull method compare the value of current with the value of size. If both are equal it returns true otherwise false.

int pop(){ return A[current--];} // The pop function

void push(int x){A[++current] = x;}

will pop and push?

During the push function, if the stack is not full yet than the value is inserted in the stack. Before the value is inserted the value of current is incremented first because its value is one less than the current empty space.

As stack follow LIFO method to pop the value. So the pop will return the value at the index current.

-

---

when we write the method: int *y = new int[10]

Does the new method in this command returns the address of the 1st location to a pointer *y?
int *y = new int[10] line is reserving the memory for 10 integer elements and giving address of first element to integer point y.

Let us consider that you declare an array size 3 to store 3 elements of stack in the following way

int a[2];

In the above case, size 3 refers to the numbers of elements of stack in the array. But the array starts from index 0, so 2 will be the last index of array which is one less than size. So that's why we use

```
int isFull()
{ return ( current==size-1); }
```

- What is Use of This pointer?

this pointer is used to access data member and function of current object. It is not necessary to use this pointer. Compiler uses it implicitly. Here we are using this pointer to differentiate between local variable Object and Node class data member Object. When we want to access any data member or function using pointer then we use arrow sign. When we want to access data member or function using object of class then we use dot sign.

-

LECTURE NO 6:

what is the difference between stack, array and linked list?

<u>Array:</u>
It stores a collection of individual values that are of the same data type.
Arrays allow random access to its elements and thus the complexity is order of O(1).
Arrays do not need an extra storage to point to next data item. Each element can be accessed via indexes.
The size of array is restricted to declaration.
Insertion/Deletion of values in arrays is very expensive. It requires memory re-allocation.

<u>Linked list:</u>
In linked list, we have the objects of Node class. Each node has two parts; data and next pointer. The next pointer points to the next node in the list. A chain of these nodes forms a linked list.
Linked list allows only sequential access to elements. Thus the complexity is order of O(n).
Linked list requires an extra storage for references. This makes them impractical for lists of small data items such as characters or Boolean values.
The size of Linked list is dynamic by nature.
Elements can be easily inserted or deleted from the linked list.

Stack:
Stack in know as LIFO (Last In First Out) structure.
A stack is a list with the restriction that insertion and deletion can be performed in only one position, namely the end of the list called the top. The fundamental operations on a

stack are push, which is equivalent to an insert, and pop, which deletes the most recently inserted element. The most recently inserted element can be examined prior to performing a pop by use of the top method.

what is the difference between Infix , Postfix and Prefix?

here are three ways to write an expression: Infix, Postfix and Prefix.
Infix: the operator is in center. For example, A + B
Postfix: the operator comes after the operands. For example, A B +
Prefix: the operator comes before the operands. For example, + A B

•

What is the back function of lists?

back() function of the list will move the current pointer one item back, for example, if the current pointer point to item no 3 of the list and we call the back() function of this list, after this call the current pointer (means current item) will be item no. 2.

What is a Precedence of Operators?

In programming languages, common operator notation is just one way of notating mathematical expressions as a linear sequence of tokens, or operators, but this is not the only way. The use of operator precedence classes and associativities is just one way. However, it is not the most general way: this model cannot give an operator more precedence when competing with '−' than it can when competing with '+', while still giving '+' and '−' equivalent precedences and associativities.

What are puch(), pop(), top()?

The push(x) method will take an element and insert it at the top of the stack. This element will become top element. The pop() method will remove the top element of the stock and return it to the calling program. The top() method returns the top-most stack element but does not remove it from the stack. The interface method names that we choose has special objective. In case of list, we have used add, remove, get, set as the suitable names. However, for stack, we are using push, pop and top. We can depict the activity from the method name like push means that we are placing an element on the top of the stack and pushing the other elements down.

Precedence of Operators
There are five binary operators, called addition, subtraction, multiplication, division and exponentiation. We are aware of some other binary operators. For example, all relational operators are binary ones. There are some unary operators as well. These require only one operand e.g. – and +. There are rules or order of execution of operators in Mathematics called precedence. Firstly, the exponentiation operation is

executed, followed by multiplication/division and at the end addition/subtraction is done. The order of precedence is (highest to lowest):

Binary and Unary Operator?

Unary Operators has only one operand. The operation takes place using a single operand. The Unary Operators are ++,--,&,*(indirection),-(positive),-(neg… etc.
Unary operators have precedence over binary operators
Binary operators (”bi” as in “two”) have two operands. In “A*B” the * operator has two operands: A and B. In “!B” the “!” operator (meaning boolean NOT) has only one operand, and is therefore a unary operator. The “-” and “+” operators can be both binary and unary, in “-4″ or “+4″ it denotes a negative or a positive number, in “0-4″ it acts as the subtraction operator.

-

INFIX,POSTFIX,PREFIX?

A major application that illustrates the different types of stacks and the various operations and functions defined upon them.

A + B à Infix
+ AB à prefix
AB+ à postfix

A + (B * C)
A + (BC*)
A(BC*)+
ABC * +

The following is the order of precedence (highest to lowest)

Exponentiation

Multiplication / division
Addition / subtraction

Infix

A+B
A + B – C

(A+B) * (C – D)
A $ B * C – D + E / F / (G + H)

postfix

AB+
AB + C –
AB+ CD- *
AB$C*D-EF / GH + / +

Prefix

+AB
- + ABC
* + AB – CD
+ - * $ ABCD / / EF + GH

Define Pop() method?

pop() is one of the methods of the stack class. It is used to retrieve elements from stack one by one. As stack follow LIFO method, this method is applied in pop function. The element which is inserted last during push method is retrieved first in pop method.

For example you have inserted 5 elements in the stack i.e. 2, 13, 8, 9, 5(order is same, 2 is inserted first and 5 at last). Now we will cal pop method once, 5 will be printed on screen. We again cal pop method, 9 will be printed on screen and so on.

What is the benefit writing implementation separately ,if we can write it in same program.?

Public member functions exposed by a class are called interface. Purpose of writing interface in a different file with .h extension and writing implementation in different file with .cpp extension is to separate interface and implementation details of your program. Separation of implementation from the interface is good software engineering.

Separating interface and implementation allows you to change the implementation independently of the interface. This helps to deal with changing requirements.

For example, let us consider that one year ago, you developed an application by separating interface and implementation. You developed that application in which data processing capability was not so fast. But with the passage of time you come to know that you should change the implementation of your application so that data processing capability may become fast. For this purpose you replaced old implementation with the some new better and faster implementation without having to change the interface of the system.

Advantages

1. User is only concerned about ways of accessing data (interface)
2. User has no concern about the internal representation and implementation of the class

Lecture no 7

What Is Constant time?

“Constant time" means that no matter what the input is, the program will not run longer than a known amount of time.

OR

"Constant time" means that the operation will execute in an amount of time independent of the input size

For example, when we say that insertion in a linked list is a constant time operation, then it means that process of inserting a node in a linked list will take a particular amount of time i.e. 5 milliseconds. And if we change the value to be inserted then the time will not exceed more than 5 milliseconds.

What is Traversing?

Traversing means visiting each and every element in a data structure, at least once. Traversing is used in searching and printing elements from a data structure.

What is the postfix expression.

Postfix expression is in the form of ABO where, A and B are numbers or also postfix expression and O is operator (+, -, *, /).e.g. AB+.

The reason to convert infix to postfix expression is that postfix expression evaluation is easier by using a stack. In postfix expression evaluation, maintain a stack and scan the postfix expression from left to right. If the element is a number, push it into the stack and if the element is an operator O, pop twice and get A and B respectively. Calculate BOA and push it back to the stack. When the expression is ended, the number in the stack is the final answer.

For example, for given 3 6 9*+ 3- expression, start scanning from left. Follow the following given steps;

- First, push(3) into the Stack
- Then, push(6) into the Stack
- Push(9) into the stack

- Now we see an operator *, that means we can get an new number by calculation
- Pop the first two numbers and perform calculations on these numbersi.e.6*9=54
- Push the new number back into the stack
- Then we see the next operator +, pop the top two values and perform the calculation 3+54=57
- Push the new number back
- We see the next number 3, Push (3) into the stack
- Then we see the next operator -, pop the top two values and perform the calculation 57-3= 54

54 is the required result.

What are Infix notation and Postfix notation ?

<u>Infix notation:</u>
Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A * (B + C) / D is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer."

<u>Postfix notation:</u>
Operators are written after their operands. The infix expression given above is equivalent to A B C + * D /

Examples:

Infix	Postfix	Notes
A * B + C / D	A B * C D / +	multiply A and B, divide C by D, add the results
A * (B + C) / D	A B C + * D /	add B and C, multiply by A, divide by D
A * (B + C / D)	A B C D / + *	divide C by D, add B, multiply by A

What is the precedence rule before evaluating the expression?

The arithmetic operators in an expression are evaluated according to their precedence. The precedence means which operator will be evaluated first and which will be evaluated after that and so on. In an expression, the parentheses ( ) are used to force the evaluation order. The operators in the parentheses ( ) are evaluated first. If there are nested parentheses then the inner most is evaluated first.
The expressions are always evaluated from left to right. The operators *, / and % have the highest precedence after parentheses. These operators are evaluated before + and – operators. Thus + and – **operators has the** lowest precedence. It means that if there are * and + operators in an expression then first the * will be

evaluated and then its result will be added to other operand. If there are * and / operators in an expression (both have the same precedence) then the operator which occurs first from left will be evaluated first and then the next, except you force any operator to evaluate by putting parentheses around it.

The following table explains the precedence of the arithmetic operators:

OPERATORS	OPERATIONS	PRECEDENCE (ORDER OF EVALUATION)
( )	Parentheses	Evaluated first
*, /, or %	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated from left to right
+ or -	Addition, Subtraction	Evaluated last. If there are several, they are evaluated from left to right

in the postfix form, parentheses are not used. Consider the infix expressions as '4+3*5' and '(4+3)*5'. The parentheses are not needed in the first but ***are necessary in the second expression. The position of operators and operands in the expression makes it clear in which order we have to do the multiplication and addition.***

How can we evaluate it?

Each operator in a postfix expression refers to the previous two operands. As the operators are binary (we are not talking about unary operators here), so two operands are needed for each operator. The nature of these operators is not affected in the postfix form i.e. the plus operator (+) will apply on two operands. Each time we read an operand, we will push it on the stack. We are going to evaluate the postfix expression with the help of stack. After reaching an operator, we pop the two operands from the top of the stack, apply the operator and push the result back on the stack

Using Parentheses to Specify the Order of Operators

You can enclose search terms and their operators in parentheses to specify the order that they are interpreted.

- **Information within parentheses is read first, then information outside parentheses is read next.**
- **If there are nested parentheses, the search engine processes the innermost parenthetical expression first.**

Nested Parentheses

If there are nested parentheses, the search engine processes the innermost parenthetical expression first, then the next, and so on until the entire query is interpreted.

Infix to postfix Conversion

We have seen how to evaluate the postfix expressions while using the stack. How can we convert the infix expression into postfix form? Consider the example of a spreadsheet. We have to evaluate expressions. The users of this spreadsheet will employ the infix form of expressions. Consider the infix expressions 'A+B*C' and '(A+B)*C'. The postfix versions are 'ABC*+' and 'AB+C*' respectively. The order of operands in postfix is the same as that in the infix. In both the infix expressions, we have the order of operands as A, B and then C. In the postfix expressions too, the order is the same i.e. A, B, followed by C. The order of operands is not changed in postfix form. However, the order of operators may be changed. In the first expression 'A+B*C', the postfix expression is 'ABC*+'. In the postfix form multiplication comes before the plus operator. In scanning from left to right, the operand 'A' can be inserted into postfix expression.

Lecture no 8

What is STL of data structures?

STL stands for Standard Template Library. It is a software library partially included in the C++ Standard Library. The STL provides a ready-made set of common classes for C++, such as containers and associative arrays, that can be used with any built-in type and with any user-defined type that supports some elementary operations (such as copying and assignment). STL algorithms are independent of containers, which significantly reduces the complexity of the library.

what is fatal error ?

A fatal error is an error that causes a program to abort and may return the user to the operating system. Due to this error, data that the program was processing may be lost.

A fatal error usually occurs due to many reasons; some are given below:

1. User may have tried to access an illegal memory location
2. Invalid data or code has been accessed
3. A Program attempts to divide an integer by zero.

What do u mean by calling template function?

Usually function perform on the type of data for which it has been defined, for example, consider the following function for calculating the square of an integer number,

```
int squar(int sq)
{
return sq*sq;
}
```

This function will work only for integer data, and we can only call it in main function like,

```
int main()
```

```
{
int ivar = 10;
coutsquar(ivar) // Can be called only with int values/variable

getche();
return 0;
}
```

However if we want to make this function as general, means it work for int, float etc as well, the only method is to make the template of this function, the templatize version of this function will be as,

```
template <class T>

T squar(T sq)
{
return sq*sq;
}
```

In main function it can be called with different type of data as,

```
int main()
{
int ivar = 10;
float fvar = 2.5;
coutsquar(ivar) endl squar(fvar); // Calling the same function with different type of data

getche();
return 0;
}
```

What is the purpose of using(*)?

With reference to pointers, * can be used to declare a pointer, like char * aName; (aName is a pointer of type char)
Pointers are special type of variables in which **a memory address is stored. They contain a memory address, not value of the variable.**

What is the Thing Queue?

A queue is a linear data structure into which items can only be inserted at one end and removed from the other. In contrast to the stack, which is a LIFO (Last In First Out) structure, a queue is a FIFO (First In First Out) structure.
The usage of queue in daily life is pretty common. For example, we queue up while depositing a utility bill or purchasing a ticket. The objective of that queue is to serve persons in their arrival order; the first coming person is served first. The person, who comes first, stands at the start followed by the person coming after him and so on. At the serving side, the person who has joined the queue first is served first.

what are Templates?

Through templates we can create generic functions and classes. In a generic function or class, the type of data upon which the function or class operates is specified as a parameter.
Thus, you can use one function or class with several different types of data without having to explicitly re-code specific versions for each data type.

The syntax of the template class is:

```
template <class T>
class class-name()
{
//definition of class
};
```

In the definition of the class where the generic data type is required, we write T. For example, there is a class in which we want to write int data type. The int is the data type that may be a float or double at different times. For this, T is written wherever we are using int in the class definition.

What is the difference between stack and heap?

There are two places that stores items in memory as your code executes. These are Stack and the Heap. Both the stack and heap help us run our code. They reside in the operating memory on our machine and contain the pieces of information we need to make it all happen. Stack memory stores variable types in memory, these variables in programming are called local variables and are often stored for short amounts of time while a function/method block uses them to compute a task. Once a function/method has completed its cycle the reference to the variable in the stack is removed.
Heap is similar except that its purpose is to hold information (not keep track of execution most of the time) so anything in our Heap can be accessed at any time. With the Heap, there are no constraints as to what can be accessed like in the stack Heap memory stores all instances or attributes, constructors and methods of a class/object. A Heap reference is also stored in Stack memory until the life cycle of the object has completed. Inside the Heap reference all the contents of the object are stored whereas with a local variable only the variable contents are stored in the stack.

What are the local, global variables in the function and the stack in the function?

These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program. As such, they are normally implemented using a stack. Local variables cease to exist once the function that created them is completed. They are recreated each time a function is executed or called.

Global:
global variables are declared outside the boundary of the function. These variables can be accessed by any function comprising the program. They are implemented by associating memory locations with

variable names. They do not get recreated if the function is recalled.

stack in function:
Stack is a data structure that stores information about the active subroutines of a computer program. This kind of stack is also known as an execution stack, control stack, function stack, or run-time stack, and is often shortened to just "the stack". Although maintenance of the call stack is important for the proper functioning of most software. Whenever we call a function then compiler will create a stack for the function and first of all it places the return address of the function where the control will go back after executing the function. After this it place the arguments of the function on the stack. After the execution of the function called the programs start its execution from the next line after the function call by taking return address from the stack.

-

What is difference between stack and heap?

There are two places that stores items in memory as your code executes. These are Stack and the Heap. Both the stack and heap help us run our code. They reside in the operating memory on our machine and contain the pieces of information we need to make it all happen. Stack memory stores variable types in memory, these variables in programming are called local variables and are often stored for short amounts of time while a function/method block uses them to compute a task. Once a function/method has completed its cycle the reference to the variable in the stack is removed.
Heap is similar except that its purpose is to hold information (not keep track of execution most of the time) so anything in our Heap can be accessed at any time. With the Heap, there are no constraints as to what can be accessed like in the stack Heap memory stores all instances or attributes, constructors and methods of a class/object. A Heap reference is also stored in Stack memory until the life cycle of the object has completed. Inside the Heap reference all the contents of the object are stored whereas with a local variable only the variable contents are stored in the stack.

What is link list and how we use it in our daily life?

List is a sequential data structure, i.e. a collection of items accessible one after another Beginning at the head and ending at the tail. It is a widely used data structure for Applications which do not need random access. It differs from the stack and queue data structures in that additions and removals can be made at any position in the list. Link list is a type of list data structure in which we implement the concept of dynamic memory The Linked List is stored as a sequence of linked nodes. As in the case of the stack, each node in a linked list contains data AND a reference to the next node.

-

Stack Abstract Data Type
The stack abstract data type is defined by structure and operations. A stack is structured,

"as an ordered collection of items where items are added to and removed from the end called the “top.” Stacks are ordered LIFO.

<u>What are Stack operators?</u>

<u>The stack operations are :</u>

Stack() creates a new stack that is empty. It needs no parameters and returns an empty stack.

push(item) adds a new item to the top of the stack. It needs the item and returns nothing.

pop() removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.

peek() returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.

isEmpty() tests to see whether the stack is empty. It needs no parameters and returns a boolean value.

size() returns the number of items on the stack. It needs no parameters and returns an integer.

C++ Templates

§We need a stack of operands and a stack of operators.
§Operands can be integers and floating point numbers, even variables.
§Operators are single characters.
§We would have to create classes FloatStack and CharStack.
§Yet the internal workings of both classes is the same.
§We can use C++ Templates to create a “template” of a stack class.
§Instantiate float stack, char stack, or stack for any type of element we want.

Function Call Stack

§ Stacks play a key role in implementation of function calls in programming languages.
§ In C++, for example, the “call stack” is used to pass function arguments and receive return values.
§The call stack is also used for “local variables”

Why the return type of push() is void ?

The return type of push method is void because this method returns nothing. This method is used to insert a new element in the stack. If you want to return something(any value) from push method then you can change its return type according to the value returned. In case of stack and the functionality explained here in this course, there is no need of changing return type.

What is psedo code?

It is an outline of a program, written in a form that can easily be converted into real programming statements. It uses a combination of informal programming structures and verbal descriptions of code. Emphasis is placed on expressing the behavior or outcome of each portion of code rather than on strictly correct syntax.

Lecture no 9

Use of Queues??

§Out of the numerous uses of the queues, one of the most useful is simulation.
§A simulation program attempts to model a real-world phenomenon.
§Many popular video games are simulations, e.g., SimCity, FlightSimulator
§Each object and action in the simulation has a counterpart in real world.
§If the simulation is accurate, the result of the program should mirror the results of the real-world event.
§Thus it is possible to understand what occurs in the real-world without actually observing its occurrence.
§Let us look at an example. Suppose there is a bank with four tellers.

AVL and Binary tree node insertion?

After inserting an element into an AVL tree, if the tree becomes unbalanced then you should have to balance the tree by performing either single or double rotation on the tree. Whereas in BST, you don’t balance the tree after inserting any data element in it.

Liner and Nonlinear data structure?

A data structure is classified into two categories: Linear and Non-Linear data structures. A data structure is said to be linear if the elements form a sequence, for example Array, Linked list, **queue** etc. Elements in a nonlinear data structure do not form a sequence, for example AVL Tree, Binary tree, etc.
Ways of representation:

There are two ways of representing linear data structures in memory. One way is to have the linear relationship between the elements by means of sequential memory locations. Such linear structures are called arrays. The other way is to have the linear relationship between the elements represented by means of links. Such linear data structures are called linked list.

•

what is wrap around?

In simple words, wrap-around means "connect the end to the start". Circular list is an example of wrap-around in which we connect end-of-list to the start-of-list.

How queue are effective in Data Structures?

A queue is a linear data structure into which items can only be inserted at one end and removed from the other. In contrast to the stack, which is a LIFO (Last In First Out) structure, a queue is a FIFO (First In First Out) structure.
The usage of queue in daily life is pretty common. For example, we queue up while depositing a utility bill or purchasing a ticket. The objective of that queue is to serve persons in their arrival order; the first coming person is served first. The person, who comes first, stands at the start followed by the person coming after him and so on. At the serving side, the person who has joined the queue first is served first.

Out of the numerous uses of the queues, one of the most useful is simulation. A simulation program attempts to model a real-world phenomenon. Many popular video games are simulations, e.g., SimCity, Flight Simulator etc.

What is meant by "Parameter pass by value and pass by reference" in c++?

Parameter passed by value:

When we call a function with parameter we pass the copies of their values.
For example

Int x=2, y= 3, z;
z= addition (x,y);
What we did that we pass the value of x and y (i.e. 2 and 3)to our desired function.

Parameter passed by reference:

In parameter passed by reference, the first thing is that in the declaration of our function the type of each parameter was followed by an ampersand sign (&). This ampersand is what specifies that their corresponding arguments are to be passed by reference instead of by value. When a variable is passed by reference we are not passing a copy of its value, but we are somehow passing the variable itself to the function and any modification that we do to the local variables will have an effect in their counterpart variables passed as arguments in the call to the function.
Example

```
Void addition (int & a, int & b, int & c)
{
a * = 1;
b* = 2;
c* = 3;
}
```

Now when we call the function

```
int x=5, y=3, z=2;
addition (x, y, z);
```

Here x, y, z will be passed by reference to our original addition function.

What is a Dangling pointer?

A dangling pointer is a pointer which has been allocated but does not point to any entity. For example un-initialized pointer.

```
int * ptr;
*ptr = 10;
```

The correct way of doing so is:

```
int * ptr;
int a;
ptr = &a;
*ptr = 10;
```

Another example can be a function returning reference of a local variable. The local variables are destroyed when function call ends, so if you try to access that location which has been deleted, it may lead you to unpredictable results.

```
int * function()
{
int a = 10;
return &a;
}

int* ptr = function();
*ptr = 5;
```

•

<u>What is Queue??</u>

A queue is a linear data structure into which items can only be inserted at one end and removed from the other. In contrast to the stack, which is a LIFO (Last In First Out) structure, a queue is a FIFO (First In First Out) structure.

The usage of queue in daily life is pretty common. For example, we queue up while depositing a utility bill or purchasing a ticket. The objective of that queue is to serve persons in their arrival order; the first coming person is served first. The person, who comes first, stands at the start followed by the person coming after him and so on. At the serving side, the person who has joined the queue first is served first. If the requirement is to serve the people in some sort of priority order, there is a separate data structure that supports priorities.

WHAt is Priority Queue?

A queue in which the items are sorted so that the highest priority item is always the next one to be extracted.

What is Life critical systems?

Systems on which we depend for safety and which may result in death or injury if they fail: medical monitoring, industrial plant monitoring and control and aircraft control systems are examples of life critical systems.

What is Real time systems?

Systems in which time is a constraint. A system which must respond to some event (eg the change in attitude of an aircraft caused by some atmospheric event like wind-shear) within a fixed time to maintain stability or continue correct operation (eg the aircraft systems must make the necessary adjustments to the control surfaces before the aircraft falls out of the sky!).

•

Queues

§A stack is LIFO (Last-In First Out) structure.
§In contrast, a queue is a FIFO (First-In First-Out) structure.
§A queue is a linear structure for which items can be only inserted at one end and removed at another end.

Queue Operations

Enqueue(X) – place X at the rear of the queue.

Dequeue() -- remove the front element and return it.

Front() -- return front element without removing it.

IsEmpty() -- return TRUE if queue is empty, FALSE otherwise

•

Implementing Queue

§Using linked List: Recall

§Insert works in constant time for either end of a linked list.
§Remove works in constant time only.
§Seems best that head of the linked list be the front of the queue so that all removes will be from the front.
§Inserts will be at the end of the list.

Queue using Array

§If we use an array to hold queue elements, both insertions and removal at the front (start) of the array are expensive.
§This is because we may have to shift up to "n" elements.
§For the stack, we needed only one end; for queue we need both.
§To get around this, we will not shift upon removal of an element.

-
- **§We have inserts and removal running in constant time but we created a new problem.**
- **§Cannot insert new elements even though there are two places available at the start of the array.**
- **§Solution: allow the queue to "wrap around".**

LECTURE NO 10

What are Simulation Models?

Simulation model is the representation of real world problem through computer software. It is very near to original model.

-

AVL and Binary tree node insertion?

After inserting an element into an AVL tree, if the tree becomes unbalanced then you should have to balance the tree by performing either single or double rotation on the tree. Whereas in BST, you don't balance the tree after inserting any data element in it.

-

Lecture no 10

---

what is the real data structure ?

Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently, by efficiency here means that it executes in smaller time ,take less space while storing and take less time while retrieval. Data structure is not only concerned with C++ or other specific programming language but it plays a central role in modern computer Science. As computer applications are becoming complex, so there is need for more resources. This does not mean that we should buy a new computer to make the application execute faster. Our effort should be to ensure that the solution is achieved with the help of programming, data structures and algorithm. Data structure enables an efficient storage of data for an easy access. It enables to represent the inherent relationship of the data in the real world. It enables an efficient processing of data. It helps in data protection and management.

-

WHat is FIFO and LIFO queue?

FIFO queue:
A queue in which the first item added is always the first one out.

LIFO queue:
A queue in which the item most recently added is always the first one out.

-

FIFO means first in first out and LIFO means last in first out.......

In case of FIFO the items that were added firstly, they will remove firstly to complete a transaction.

While in LIFO the items that were added in the last will be removed firstly from the queue.

-

Event based Simulation

§Don't wait for the clock to tic until the next event.
§Compute the time of next event and maintain a list of events in increasing order of time.
§Remove a event from the list in a loop and process it.

-

Lecture no 11

Strict Binary Tree and Complete Binary Tree

Strict Binary Tree: If every non-leaf node has non-empty left and right subtrees.

Complete Binary Tree: If every non-leaf node has non-empty left and right subtrees and all leaf nodes are at the same level.

So, complete binary tree is a strict binary tree with the restriction that all leaf nodes must be at the same level.

What is Binary Tree?

Binary Tree The mathematical definition of a binary tree is "A binary tree is a finite set of elements that is either empty or is partitioned into three disjoint subsets. The first subset contains a single element called the root of the tree. The other two subsets are themselves binary trees called the left and right sub-trees”. Each element of a binary tree is called a node of the tree.

Depth of tree Explain?

There are different levels in tree. Each level has different number of nodes. Root node lies at level 0, Childs of root node lie at level 1, similarly nodes which lie at level 1 may also have different children’s which lie at level 2 and so on.
In formula, d stands for depth of tree. The depth of a tree is the maximum level of any node in the tree

---

<u>Implementation of Priority Queues?</u>

There are a variety of simple, usually inefficient, ways to implement a priority queue. They provide an analogy to help one understand what a priority queue is. For instance, one can keep all the elements in an unsorted list.

Whenever the highest-priority element is requested, search through all elements for the one with the highest priority. (In big O notation: O(1) insertion time, O(n) pull time due to search.)

<u>What is full binary tree/Strictly?</u>

A full binary tree (sometimes proper binary tree or 2-tree or strictly binary tree) is a tree in which every node other than the leaves has two children.

so you have no leaves with only 1 child. Appears to be the same as strict binary tree.

Here is an image of a full/strict binary tree

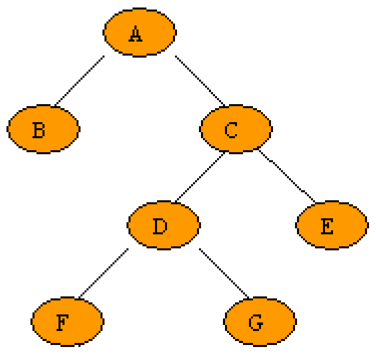
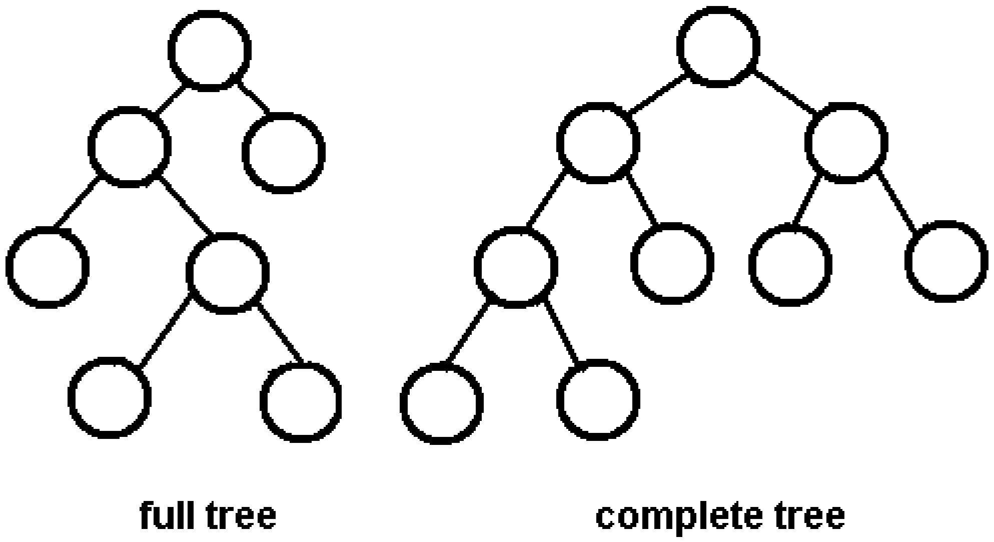


Fig .5.1.4 Strictly binary tree

Full binary tree are a complete binary tree but reverse is not possible, and if the depth of the binary is n the no. of nodes in the full binary tree is (2^n-1). It is not necessary in the binary tree that it have two child but in the full binary it every node have no or two child.

COmplete Binary Tree?

A binary tree T with n levels is complete if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.



A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

A complete binary tree can be used to represent a heap. It can be easily represented in contiguous memory with no gaps (i.e. all array elements are used save for any space that may exist at the end).

What is Level?

Level
The level of a node in a binary tree is defined as follows:
• Root has level 0,
• Level of any other node is one more than the level its parent (father).
• The depth of a binary tree is the maximum level of any leaf in the tree.
•

LECTURE NO 12

What are Trace of Insert?

"Trace of Insert" means that we insert a new node into a tree and will observe step-by-step how it reaches to its proper location in the tree.

For example, in handouts, 17 is inserted into the tree.
First of all, we will compare it with the root (14) of the tree.
As 17 is greater than the root(14), so it will be moved to the right sub-tree.
Then, 17 will be compared with 15, still greater, move to the right sub-tree.
Then , 17 will be compared with 18, now will move to the left sub-tree.
Then, 17 will be compared with 16, it will be moved to the right sub-tree.
There is no node on the right of 16, so 17 will be inserted here.

This was the trace of inserting 17 into the tree.

Level of a Complete Binary Tree
We can find the depth of a complete binary tree if we know the total number of nodes. If we have a complete binary tree with n total nodes then by the equation to total number of nodes we can write..

Total number of nodes=2d+1-1 = n
To find the value of d, we solve the above equation as under

2d+1-1 = n
2d+1 = n + 1
d + 1 = log2 (n + 1)
d = log2 (n + 1) – 1

After having n total nodes, we can find the depth d of the tree by the above equation. Suppose we have 100,000 nodes. It means that the value of n is 100,000, reflecting a depth i.e. d of the tree will be log2 (100000 + 1) – 1, which evaluates to 20. So the the tree will be 20. In other words, the tree will be 20 levels deep.

What is Binary Heap ?

The binary heap is a data structure that can efficiently support the basic priority-queue operations. In a binary heap, the items are stored in an array such that each key is guaranteed to be larger than (or equal to) the keys at two other specific positions.

In turn, each of those keys must be larger than two more keys, and so forth. This ordering is easy to see if we view the keys as being in a binary tree structure with edges from each key to the two keys known to be smaller.

Definition. *A binary heap is a set of nodes with keys arranged in a complete heap-ordered binary tree, represented in level order in an array (not using the first entry).*

A binary tree is heap-ordered if the key in each node is larger than (or equal to) the keys in that nodes two children (if any)

Uses and Properties of Binary tree ?

Uses for Binary Trees…

- **Use for storing and retrieving information**
- **Insert, delete, and search faster than with a linked list**
- **Take advantage of logn height**
- **Idea: Store information in an ordered way (keys)**

A Property of Binary Search Trees

- **The key of the root is larger than any key in the left subtree**
- **The key of the root is smaller than any key in the right subtree**

Note: Duplicated keys are not allowed

How to insert some new numbers in the tree?

Binary tree is a tree like (means hierarchical) data structure in which each node has at most two child nodes, these child nodes are called left child (or left subtree) and right child (or right subtree), while binary search tree (BST) is a special binary tree with the following properties,

1. The left child (left subtree) of a node contains only those nodes which have values less than the value of this node.
2. The right child (right subtree) of a node contains only those nodes which have values greater than the value of this node.
3. Both the left and right subtrees must also be binary search trees.

So inorder to insert elements in binary search tree, you should have to keep in mind the properties of BST that has been discussed above. Now let us consider a string

14, 15, 4

Now 14 i.e. first elements will be root node of BST. As 15 is greater than 14 so 15 will become right child of 14. Similarly as 4 is less than 14 so 4 will become left child of node 14.

What is Strictly Binary tree?

A strictly binary tree is a tree in which every node other than the leaves has two children. Or, perhaps more clearly, every node in a binary tree has exactly (strictly) 0 or 2 children.

Lecture no 13

What is the purpose of traversing a Tree data data structure?

Accessing and processing each data item of a data structure is called traversing.
Data is stored in computers using data structures in order to perform different operations on the data, for which each data item is needed to access and process.

What is Binary Tree Traversal?

Many binary tree operations are done by performing a traversal of the binary tree .In a traversal, each element of the binary tree is visited exactly once . During the visit of an element, all action (make a clone, display, evaluate the
operator, etc.) with respect to this element is taken.

Binary Tree - Traversal
Traversal of a binary tree is defined recursively. For example,

1. Visit the node (root or subroot)
2. Traverse the left subtree
3. Traverse the right subtree

Traversal Applications
§ Make a clone
§ Determine height
§ Determine number of nodes

Uses for Binary Trees…
-- Binary Expression Trees
Binary trees are a good way to express arithmetic expressions.
– The leaves are operands and the other nodes are operators.
– The left and right subtrees of an operator node represent sub expressions that must be evaluated before applying the operator at the root of the subtree.

-

What are methods of traversing a tree?

<u>Depth-first traversal:</u> This strategy consists of searching deeper in the tree whenever possible. Certain depth-first traversals occurs frequently enough that they are given names of their own

<u>Breadth-first traversal:</u> This is a very simple idea which consists of visiting the nodes based on their level in the tree. It visits all nodes with depth 0, then depth 1, then depth 2, and so on

<u>Special cases of depth-first traversal for binary trees</u>
In binary trees there are three basic ways to traverse a tree using the a depth-first search idea

<u>– Preorder:</u> We visit a node, then visit the left and the right subtrees
```
void preOrder(BinaryTreeNode t)
{
if (t != null)
{
visit(t);
preOrder(t.leftChild);
preOrder(t.rightChild);
}
}
```

<u>– Inorder</u>: We visit the left subtree then we visit the node, then we visit the right subtree
```
void inOrder(BinaryTreeNode t)
{
if (t != null)
{
inOrder(t.leftChild);
visit(t);
inOrder(t.rightChild);

}
}
```

<u>– Postorder:</u> We visit the left and right subtree and then we visit the node. This is what normally authors mean if they mention just depth -first traversal
```
void postOrder(BinaryTreeNode t)
{
if (t != null)
{
postOrder(t.leftChild);
postOrder(t.rightChild);
visit(t);

}
}
```

•

Lecture no 14

How can we determine, whether data structure is recursive or not?

Recursive data structures are used for recursive problems, where Recursive problems are those problems in which the solution depends on the solution to smaller instances (occurrences) of the same problem.

For example, we need to find the sum of first 100 numbers (i.e. sum of 1-100).
Now,
Sum of 1-100 = 1+ (Sum of 2-100)
Sum of 2-100 = 2+ (Sum of 3-100)
Sum of 3-100 = 3+ (Sum of 4-100)
Sum of 4-100 = 4+ (Sum of 5-100)

The (Sum of 2-100), (Sum of 3-100), (Sum of 4-100) etc are the smaller instances of the main problem.

When a data structure solves a problem which is recursive in nature, then the data structure may be called recursive data structure.

Preorder Traversal (cont.)
Visit the root of the tree first, then visit the nodes in the left subtree, then visit the nodes in the right subtree

Preorder(tree)

```
If tree is not NULL {
Visit Info(tree)
Preorder(Left(tree))
Pre Order(Right(tree)


}
```

- ***Inorder Traversal (cont.)***
Visit the nodes in the left subtree, then visit the root of the tree, then visit the nodes in the right subtree

Inorder(tree)
```
If tree is not NULL {
Inorder(Left(tree))
Visit Info(tree)
Inorder(Right(tree))
}
```

- ***Postorder Traversal***
Visit the nodes in the left subtree first, then visit the nodes in the right subtree, then visit the root of the tree

Postorder(tree)

```
If tree is not NULL {
Postorder(Left(tree))
Postorder(Right(tree))
Visit Info(tree)
}
```

WHat is Breadth First traversal of a tree?

Breadth-first traversal of a tree

A breadth-first traversal consists of visiting the nodes based on their level in the tree. It visits all nodes with level
depth 0, then depth 1, then depth 2, and so on.
Use a queue to implement breadth first traversal

Lecture no 15

Deleting a Node from the BST

we may also require to delete some data (nodes) from a binary tree. Consider the case where we used binary tree to implement the telephone directory, when a person leaves a city, its telephone number from the directory is deleted.
It is common with many data structures that the hardest operation is deletion. Once we have found the node to be deleted, we need to consider several possibilities.
For case 1, If the node is a *leaf*, it can be deleted quite easily.
See the tree figure below.

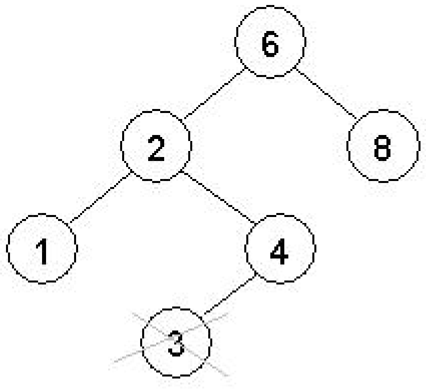


Fig 15.9: BST

What is COst of Search?

Cost of searching an element in a data structure means the time it takes to search an element or numbers of comparisons performed to search an element in a particular data structure**.**

What is Binary Search Tree?

Binary tree is a tree like (means hierarchical) data structure in which each node has at most two child nodes, these child nodes are called left child (or left subtree) and right child (or right subtree), while binary search tree (BST) is a special binary tree with the following properties,

a. The left child (left subtree) of a node contains only those nodes which have values less than the value of this node.

b. The right child (right subtree) of a node contains only those nodes which have values greater than the value of this node.

What is recursive Call?

Both the left and right subtrees must also be binary search trees.

3) When a function calls itself then it is called recursion or recursive call.

What is Pre order?

Pre-order traversal carried out by using recursion technique is called pre-order recursion

What is the purpose of level order traversal of binary tree?

It has observed that the other traversal method that is, preorder, inorder and postorder use the recursive method while the level order traversal use a non-recursive method.
Recursive solutions may involve extensive overhead because they use function calls.

When a function call is made, it takes time to build a stack frame and push it into the stack. Conversely, when a return is executed, the stack frame must be popped from the stack and the local variables reset to their previous values.
Further, this takes time. A recursive procedure therefore generally runs slower than non recursive procedure.
In addition, each time when we make a function call we use some of our memory allocation. If the recursion is deep, that is, if there are many recursive calls, then we may run out of memory.

Lecture no 16:

void remove(const int& x);
is a user define function which take a constant integer reference as an argument.

void insert(const EType& x);
is a user define function which take a constant Template reference as an argument.

void remove(const EType& x);

is a user define function which take a constant Template reference as an argument.

Explain

const EType& find(const EType & x) const;?

This method will takes an argument *x* of *EType*. It will search the tree whether *x* exists in the tree or not. Then we have *isEmpty* method that will ascertain if the tree is empty or not. There is also the *printInorder* method, which will print the tree in inorder traversal. If the tree has integers, we have sorted integers as a result of inorder traversal. Next thing we have is the *insert (const EType& x)* method, which inserts the*x* as a new node in the tree. After this, there is the *remove* method i.e.delete method. We have renamed *delete* as *remove* because *delete* is a keyword of C++. We can also name it as *deleteNode* or some other name which you think is suitable. The interface of the class is almost complete. Now think as a user of *BinarySearchTree* and decide if you need more methods. With the help of methods defined above, we can do a lot of work on *BinarySearchTree*. If you feel to add more methods due to the usage need, these can be added later.

Q1. Why we should avoid by using the references of transient object?

Storing of references of transient objects can create problems because the transient object (object created on stack) is destroyed when the function execution finishes. The question arises, what can we do, if we do not want the objects created in a function to be destroyed. The answer to this is dynamic memory allocation. All the variables or objects created in a function that we want to access later are created on memory heap (sometimes called free store) using the dynamic memory allocation functions or operators like new. Heap is an area in computer memory that is allocated dynamically. You should remember that all the objects created using new operator have to be explicitly destroyed using the delete operator.

So now, we understand that we cannot pass references to transient objects. If we want to use the objects later we create them on heap and keep the address

Lecture no 17

how can we ensure the provision of a terminating condition in recursive calls ?

It is necessary to provide a terminating condition for the recursion in a recursive function,
For example, consider the inorder method of binary tree,

```
void inorder(TreeNode<char> * treeNode)
{
if( treeNode != NULL ) // Terminating condition for recursion.
{
inorder(treeNode->getLeft());
cout (treeNode->getInfo())" ";
inorder(treeNode->getRight());
}
}
```

Similarly, there is always a condition for the termination of recursion depending upon the problem.

List::list()
Can we only use this method in the case of template?
1) Scope resolution operator i.e. '::' has no relation with templates. You can use Scope resolution operator i.e. '::' either you use templates or you don't use templates. Scope resolution operator(::) is used to define a function outside a class or when we want to use a global variable but also has a local variable with same name.

C++ programming code

```
#include <iostream>

using namespace std;



char c = 'a';     // global variable



int main() {

  char c = 'b';   //local variable



  cout  "Local c: "  c  "\n";
  cout  "Global c: "  ::c  "\n";  //using scope resolution operator


  return 0;

}
```

Scope resolution operator in class

```
#include <iostream>

using namespace std;
```

```
class programming {

public:

  void output();  //function declaration

};



// function definition outside the class



void programming::output() {

  cout  "Function defined outside the class.\n";
}



int main() {

  programming x;

  x.output();



  return 0;

}
```

If make a class for Node and have a method as void set(int object);

what does the int object here means?
In method
void set(int object);
A variable of type int is passed by value. Here object is simply a variable name of type int. You can also use any other variable name instead of object like data e.g.
void set(int data);

Use of reference Pointer?

In case of pass by value, when a variable/argument will be passed from calling function to a called function then a new copy of variable will be created of this passed argument/variable and in called function, changes will be done on new variable instead of carrying out changes in original variable of calling function. Where as in case of pass by reference, no new variable will be created and changes that will be carried out by called function in passed argument/variable will actually be the changes which will be carried out in original variable of calling function. So by using call reference we saved our self from copying of an argument/variable/object, that is why we passed by reference instead of pass by value.

Lecture no 18

why we use reference variable in a program?

Passing data to functions through reference variables is used to avoid putting data on the stack during function call.

The reference variables are mostly used with user define data type (objects), as the size of an object may be enough large and may cause memory problem (stack problem) during the function call if it is pass by value, therefore objects are usually pass by reference in which the object copy does not store on stack (memory), and the calling function access the object within the caller function directly.

•

Ampersand? and its use&?

&(ampersand) operator is used to get the address of any variable. The address of a variable can be obtained by preceding the name of a variable with an ampersand sign.

int a;

int *b;

b=a;

After these statement b will contain the address of a.

•

Lecture no 19

What is Const explain?

When "const" keyword is used before a variable declaration in the parameter list of a function, it means, the function can use the variable within its body but cannot changes/update its value.
When "const" keyword is used before the return type of a function, it means, the function will return constant value.
When "const" keyword is used after a function name in a class, it means, this member function cannot change/update the object status (that is, cannot change the data members).

What are ETYPE and findMin()?

"EType" is a class (Template class) name while findMin() is a function which finds the minimum number (node) in a binary search tree (BST).

- **What is a reference variable?**

In fact reference variable is a synonym (second name) for a pre define variable.
Consider the following program,

```
#include <iostream.h>
#include <conio.h>

main()
{
int var = 10;

int &refvar = var; // refvar is a reference variable and is a synonym for variable var;

coutvar;

refvar = refvar + 20; // "refvar" variable used for var

cout endl var; // value of "var" has changed by "refvar" variable.


getche();
return 0;
}
```

How we can find left and right node in traversal process?

The left, right and root node can be identified if we know by which traversal these nodes are taken from the tree. Suppose tree is traversed by inorder traversal, then in the first case 4 is left child, 5 is root and 6 is right child. And if tree is traversed by postorder traversal than 4 is left child, 5 is right child and 6 is the root node.

In How many Possible ways a tree can be transverse?

1: (4, 14, 15)

2: (14, 4, 15)

3: (15, 4, 14)

4: (4, 15, 14)

5: (14, 15, 4)

6: (15, 14, 4)

When we apply permutation then these 6 are the possible ways in which the above mentioned tree can be traversed. Root, left and right child will remain the same only the sequence of their traversal from the tree will be change.

Difference between BST and AVL tree?

Binary tree is a tree like (means hierarchical) data structure in which each node has at most two child nodes, these child nodes are called left child (or left subtree) and right child (or right subtree), while binary search tree (BST) is a special binary tree with the following properties,

a. The left child (left subtree) of a node contains only those nodes which have values less than the value of this node.

b. The right child (right subtree) of a node contains only those nodes which have values greater than the value of this node.

c. Both the left and right subtrees must also be binary search trees.

AVL tree is just like binary search tree. But difference between BST and AVL tree is that, during insertion of nodes, AVL tree re-balances itself when the balance factor of any node becomes more than -1 or 0 or 1. AVL tree re-balances itself by applying either single or double rotation on unbalanced node in such away that balance factor again becomes either -1 or 0 or 1.

Lecture no 20

What is balance AVL Tree?

The balance of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

On the basis of this definition, a Balanced AVL tree will be a an AVL tree whose each node has a balance of either 1, 0 or -1, while the balance of an empty tree is -1.

.How can we evaluate $\log_2$ (1000000)=20?

The log2 (1000000) is 19.93 which is about 20.

You can calculate it by yourself by using excel sheet to find the log base 2 of any value. Formula or method to find out log base 2 of any value by using excel sheet is given below.

=Log(value, base)

Let us suppose that you want to find log base 2 of value 1000000. For this purpose, you will put the value in above formula in the following way

=Log(1000000,2)

In order to calculate log base 2 of value 1000000, you will write =Log(1000000,2) in any cell of excel sheet and then you will press enter button of your keyboard to find out the log base 2 of value 1000000. Similarly you can use the formula =Log(value, base) to find out log of any value for any base by using excel sheet. E.g. Let us suppose that you want to find log base 10 of value 15. In order to calculate log base 10 of value 15, you will write =Log(15,10) in any cell of excel sheet and then you will press enter button of your keyboard to find out the log base 10 of value 15.

what is Degenerate BST?
When we use sorted data to generate binary search tree then shape of binary search tree will become like linked list. This is known as degeneration of binary search tree.

As we have observed that when a BST is made from sorted data then the degeneration problem occur, means the cost of searches increases, so in order to minimize this, we balance the tree i.e. we use AVL tree In order to avoid degeneration of binary search tree in case of sorted data.

What does this method do Event* remove()?

Method Event* remove() will return a pointer of type Event where Event is a user defined data type.
Event* e=node[0];
Statement Event* e=node[0]; is incorrect and it will given an error. It is because in order to save the first address of array in pointer of type Event, you should use new operator for dynamic memory allocation in the following way

Event* e=new Event[10];

Or in case of primitive data type like int, you should write the statement in the following way

int* e= new int[10];

Explain the difference of a binary tree and binary search tree?

Binary tree is a tree in which a node can only have two children at most. The order of insertion of nodes in the tree does not matter in case of binary tree.

Binary search tree is a type of binary tree. It is like binary tree except, the left child contains *only* nodes with values less than the parent node and the right child *only* contains nodes with values greater than the parent.

-

AVL and Shallow Tree?

An AVL tree is said to be balanced, if each node in it has the balance 1, 0, -1. Balance of each node is determined by the difference of its left and right sub trees.

Shallow tree is a tree in which any node either do not have right or left child. The height of the left and right sub trees should be same to not be shallow tree.

-

Lecture no 21

Cases of Rotation

The single rotation does not seem to restore the balance. We will re-visit the tree and rotations to identify the problem area. We will call the node that is to be rotated as a (node requires to be re-balanced). Since any node has at the most two children, and a height imbalance requires that a 's two sub-trees differ by two (or –2), the violation will occur in four cases:

1. An insertion into left subtree of the left child of a .
2. An insertion into right subtree of the left child of a .
3. An insertion into left subtree of the right child of a .
4. An insertion into right subtree of the right child of a .

The insertion occurs on the *outside* (i.e., left-left or right-right) in *cases 1* and *4.* Single rotation can fix the balance in *cases 1* and*4.*

Insertion occurs on the *inside* in *cases 2* and *3* which a single rotation cannot fix.

•

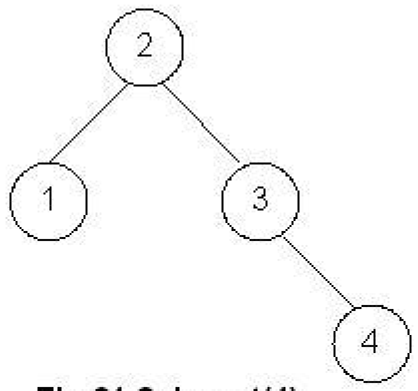


Fig 21.3: insert(4)

Once we insert a node in the tree, it is necessary to check its balance to see whether it is within AVL defined balance. If it is not so, then we have to rotate a node. The balance factor of the node containing number *4* is zero due to the absence of any left or right subtrees. Now, we see the *balance factor* of the node containing number *3*. As it has no left child, but only right subtree, the*balance factor* is *–1*. The *balance factor* of the node containing number *1* is *0*. For the node containing number *2*, the height of the left subtree is *1* while that of the right subtree is *2*. Therefore, the *balance factor* of the node containing number *2* is *1 – 2 = -1*. So every node in the tree in *fig. 21.3* has *balance factor* either *1* or less than that. You must be remembering that the condition for a tree to be an AVL tree, every node's balance needs not to be zero necessarily. Rather, the tree will be called AVL tree, if the *balance factor* of each node in a tree is *0*, *1* or *–1*. By the way, if the *balance factor* of each node inside the tree is *0*, it will be a perfectly balanced tree.

What is the depth of a binary tree and level of a binary tree?

Depth of a binary tree means the number of edges along the path from the root node to the deepest leaf node.

level of a binary tree:

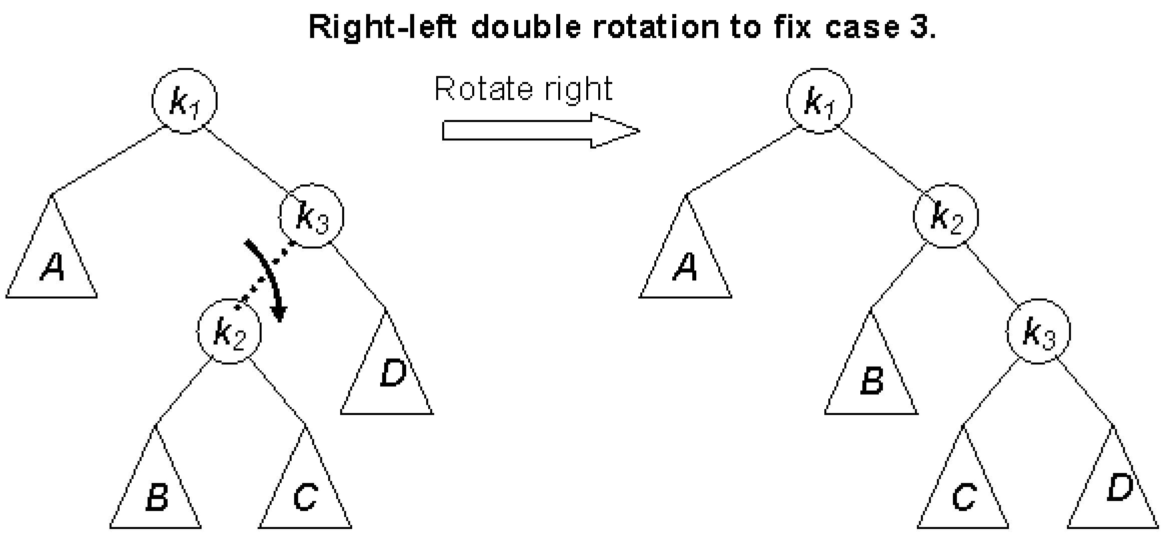
There are different levels in tree. Each level has different number of nodes. Root node lies at level 0, child of root nodes lie at level 1, similarly nodes which lie at level 1 may also have different children which lie at level 2 and so on.

-

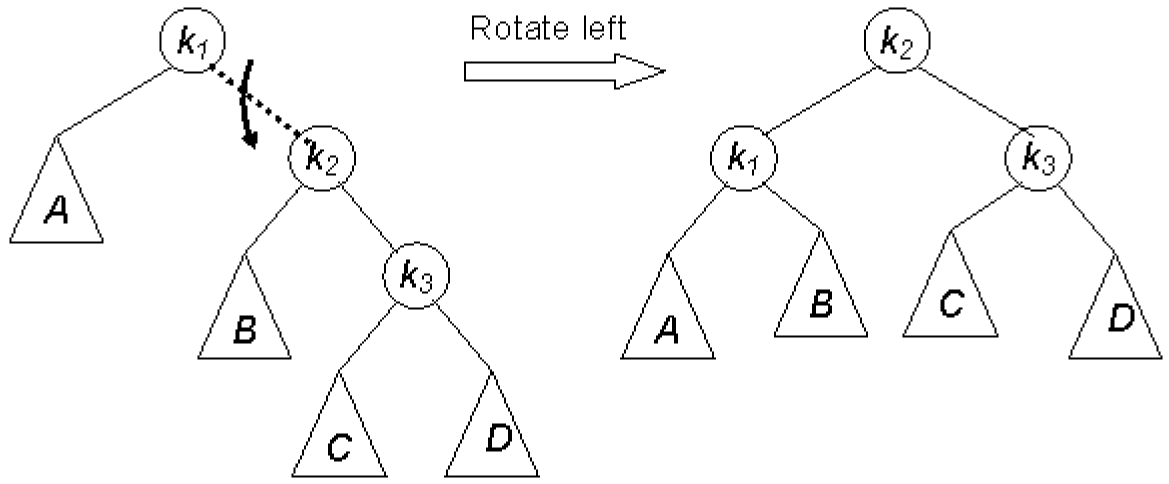
Lecture no 22

<u>How perform the Right-left double rotation to fix case 3??</u>

In case, the node is inserted in left sub tree of the right child, we encounter the same situation as discussed above. But here, we will perform right rotation at first before going for a left rotation. Let's discuss this symmetric case and see how we can apply double rotation here. First we perform the right rotation.



Here *k1* is the root of the tree while *k3* is the right child of the *k1*. *k2* is the inner child. It is the *Y* tree expanded again here and the new node will be inserted in the *k2*'s right subtree *C* or left subtree *B*. As we have to transform the *k2* into the root of the tree, so the right rotation between the link *k2* and *k3* will be carried out. As a result of this rotation, *k2* will come up and *k3* will go down. The subtree *B* has gone up with the *k2* while subtree *C* is now attached with the *k3*. To make the *k2* root of the tree, we will perform the left rotation between then *k1* and *k2*.



At the right side, we have the final shape of the tree. You can see that *k2* has become the root of the tree. *k1* and *k3* are its left and right children respectively. While performing the inorder traversal, you will see that we have preserved our inorder traversal.

•

<u>Concepts of Single rotation and Double rotation?</u>

Single rotation can restore the balance of the tree if the new inserted node has inserted as per the following conditions,

1. An insertion into left subtree of the left child of α.
2. An insertion into right subtree of the right child of α.

Double rotations can restore the balance of the tree if the new inserted node has inserted as per the following conditions,

1. An insertion into right subtree of the left child of α.
2. An insertion into left subtree of the right child of α.

α is a node that requires to be re-balanced. In order to know that which node should rotate left or right in balancing the AVL tree, you need to study the topic "Cases of Rotation" in lecture no. 21 of handouts. This topic has already been discussed in very deep detail by using graphical trees that which node (or sub tree) should go left or right during rotations.

•

what r the conditions of AVL tree?

A tree is said to be an AVL tree, if it fulfills the AVL condition. An AVL tree is said to be balanced, if each node in it has the balance 1, 0, -1. Balance of each node is determined by the difference of its left and right sub trees. To check whether a given tree is AVL or not, check the balance of each and every node of the tree. If the balance of each node in the tree is 1,0 or -1, it is said AVL tree.

•