STUDY MATERIAL

Vivekananda College, Thakurpukur

NAAC Accredited Grade-A

Computer Science

(Honours and General)

Basic concepts of Data Structure

Mahuya Paul

(Teacher ,Dept. of Computer Science,Vivekananda college, Thakurpukur)

BASIC CONCEPT OF DATA STRUCTURE

(This material will cover the following definitions of Data Structure**)**

- **Array**
- **Stack**
- **Queue**
- **Linked List**
- **Binary Tree**
- **Binary Search Tree**

DATA STRUCTURE:

A **data structure** is a specialized format for organizing and storing **data** to suit a specific purpose so that it can be accessed and worked with in appropriate ways.

- According to the NIST (National Institute of Standard and Technology) **Data structure** is an organization of information, usually in memory, for better *algorithm efficiency*, such as *queue*, *stack*, *linked list*, *heap*, *dictionary*, and *tree*, or conceptual unity, such as the name and address of a person. It may include redundant information, such as length of the *list* or number of *nodes* in a *subtree*.

ABSTRACT DATA TYPE(ADT):

Abstract data types are mathematical model of a set of data values or information that share similar behavior or qualities and that can be specified and identified independent of specific implementations.

- According to the NIST (National Institute of Standard and Technology)definition for abstract data types, an abstract data structure or type "is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations."

ARRAY:

Array is a collection of variables belongings to the same data type storing in contiguous (adjacent) memory locations.

- An **array** object is a set of pairs, <index, value>, such that each index is unique and each index that is defined has a value associated with it (a mapping from indices to values).
- Position of any array element can be accessed from its index position with help of array's starting address.
- An element can be accessed, inserted or removed by specifying its position (number of elements preceding it)
- In Insertion shifting is to start from the last element of the array and in deletion shifting is to start from the next array index from where the element is to be deleted.

INDEX	A[0]	A[1]	A[2]	A[3]	A[4]
VALUE	10	20	30	40	50

There are two types of Array representation:

❖ **One dimensional array**

- **Array declaration syntax:** data_type arr_name [arr_size]; Exmp: int age [5];
- **Array initialization syntax:** data_type arr_name [arr_size]=(value1, value2, value3,….); Exmp: int age[5]={0, 1, 2, 3, 4}; char str[10]={'H','a','i'};
- **Array accessing syntax:** arr_name[index]; Exmp: age[0]; age[1];

❖ **Multi dimensional array(two dimensional, three dimensional, and so on)**

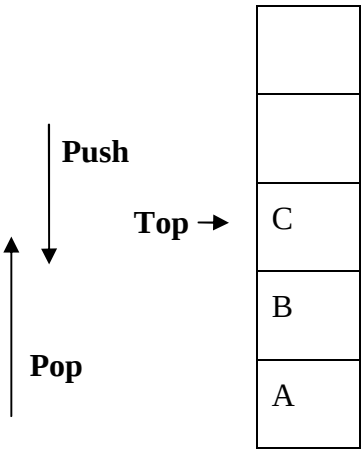
- **Array declaration syntax:** data_type arr_name [num_of_rows][num_of_column];

  int arr[2][2];
- **Array initialization syntax:** data_type arr_name[2][2] = {{0,0},{0,1},{1,0},{1,1}};
- **Array accessing syntax:** arr_name[index];

STACK:

A Stack is a linear data structure in which items may be added or removed only at one end called **top** of the stack. Stack is also called **Last in Fast Out(LIFO) lists** as the last item to be added to a stack is the first item to be removed.

Stack is also called **Push-Down** list as addition in stack is called **push** an element and deletion from a stack is called **pop an** element. Stack may be **underflow** when there is no element in the stack (empty stack) and may be **overflow** when the capacity of the stack is full.



Push an element in stack:

If(stack is full)

Return

Else

Stack[top]=stack[top+1]

Stack[top]=element

Pop an element from stack;

if(stack is empty)

return

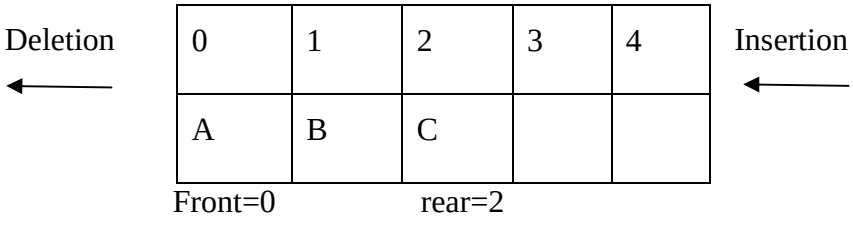
else

element=stack[top]

stack[top]=stack[top-1]

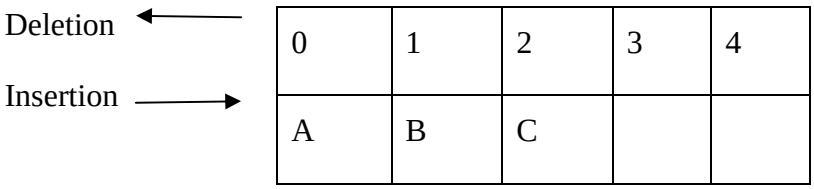
QUEUE:

A Queue is a linear data structure in which insertions can take place only at one end called **rear** and deletion can take place from the other end called **front**. A queue is called **First in first Out (FIFO)**list as the first element in a queue will be the first element out of the queues; i.e the order in which elements entered into a queue is the order in which they leave.



Dqueue:

A dequeue is a linear list in which elements can be added or removed at either end but not in the middle. The term dequeue is a contraction of the name double-ended dequeue



Front=0 rear=2

- **Input Restricted Dequeue**: Input restricted dequeue allows insertions at only one end of the list but allows deletions at both ends of the list.
- **Output Restricted Dequeue**: Output restricted dequeue allows deletions at only one end of the list but allows insertions at both ends of the list.

Circular queue: Circular queue is the same as ordinary queue only the difference is that it logically implies the first element array[0] comes after the last element array[n-1] or after last element first element appears.

The **drawback of singly queue** using array is that the insertion is denied even if the space is available at the front when the rear points to the end can be resolved by the **circular queue**.

Priority queue:

A priority queue is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed comes from the following rules:

i) An element of higher priority is processed before any element of lower priority.

ii) Two elements with the same priority are processed according to the order in which they were added to the queue.

There are two types of priority queues:

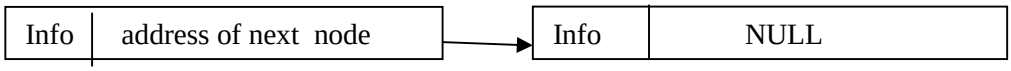
a) Ascending priority queue: Items can be inserted arbitrarily and from which only the smallest item can be removed**.**

b) Descending priority queue: Items can be inserted arbitrarily and from which only the largest item can be removed.

LINKED LIST:

Linked list is a linear dynamic data structure consisting of a sequence of nodes connected in a chain by links called pointer. Technically each such element of a linked list is called node

In a **singly linked list** each node containing two fields stores the contents of the node(info/data part) and a pointer or reference to the next node(address of the next node) in the list. It is called a **singly linked list** because each node only has a **single link** to another node.

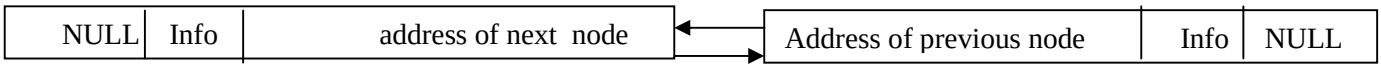


- There may be a header link and trailer link
- The operations we can perform on singly ***linked lists*** are insertion, deletion and traversal.

Different Types of Linked list:

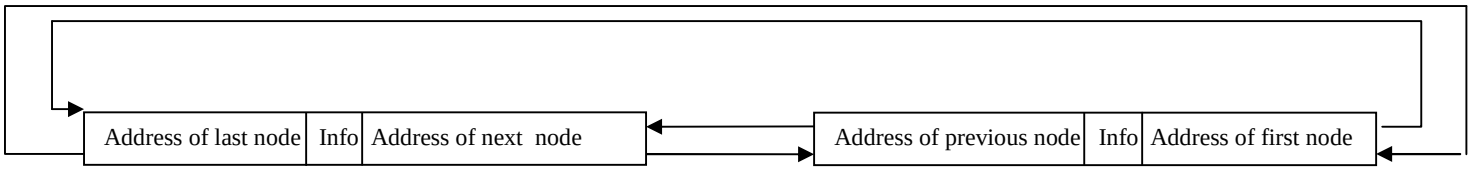
1) **Singly Linked list**: Each node contains data field and only one pointer field pointing to the adjacent node

2) **Doubly linked list**: Each node contains data field and one pointer field pointing to the location of next node and one pointer field pointing to the location of previous node of the list. This allows easily accessing list items backward as well as forward and deleting any item in constant time.



- In doubly linked list the first node always pointed by head
- The previous link field of the first node and always the next link field of the last node be NULL
- **Also known as** two-way linked list, symmetrically linked list.
- Deletion an Insertion becomes easy as there is no need to remember the address of the previous node.

3) **Circular singly linked list**:Last node contains the address of the first node in the list. In a **circularly linked list**, all nodes are **linked** in a continuous circle, without using null.



4) **Circular double linked list**:First node contains the address of last node as previous link part and last node contains the address of first node as next link part.

Compare linear and non linear Data Structure:

In **linear data structures** the data items are arranged in a linear sequence like in an array and linked list.

In **non-linear data structures** the data items are not in sequence like Tree and Graph

Compare Static and Dynamic Data Structure:

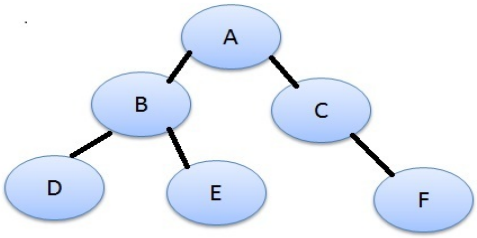
Static structures are ones whose sizes and structures associated memory locations are fixed at compile time.Example:Array.

Dynamic structures are ones which expand or shrink as required during program execution and their associated memory locations change. Example: Linked list.

TREE: A **tree** is an acyclic simple, connected graph, i.e a tree contains no loops and no cycles, there is no more than one edge between any pair of nodes.

A **binary tree** is defined as a finite set of elements called nodes, such that,

a) T is empty(called the null tree or empty tree)
b) T contains a distinguished node R, called the root of T and the remaining nodes of T form an ordered pair of disjoint binary trees T1 and T2 called respectively the left and right subtrees of R.

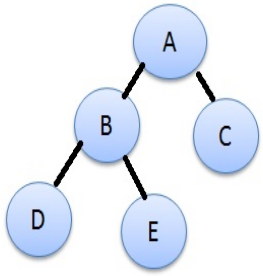


Binary tree

- A node has no child is called a **leaf.**
- The root of a binary tree has **level** '0' and the level of any other node of the tree is more than the level of its father. In general, if a node is at level n, then its children will be at level n+1.
- **Root** is the first node in the hierarchical arrangement of data items. A is the root.
- Each element of a tree is called a **node**. It specifies the information and links (branches) to other data items.
- **Parent** of a node is the immediate predecessor of a node. B is the parent of D and E
- Each immediate successor of a node is known as **child**. B and C are two children of A.
- The child nodes of a given parent node are called siblings. Here B and C are siblings.
- The number of subtrees of a node in a given tree is called **degree of that node**. The degree of node A is 3.
- The maximum degree of nodes in a given tree is called the **degree of the tree**. The degree of the tree is 3.
- **Edge** of a tree is connecting line of two nodes.
- **Path** is a sequence of consecutive edges from the source node to the destination node.
- **Depth** of node n_i is the length of the unique path from the root to n_i . Thus the root is at depth 0 and the depth of E is 2.
- The **height** of node n_i is the length of the longest path from n_i to a leaf.
- If there is a path from n_1 to n_2, then n_1 is called **ancestor** of n_2 and n_2 is called **a descendant** of n_1.
- If a binary tree contains m nodes at level l, it contains at most 2m nodes at level l+1
- Binary tree contain at most 2^l nodes at level l; at most one node at level 0(The root node)
- A binary tree of depth d contains exactly 2^d nodes at level d

Strictly binary tree:

If every non leaf node in a binary tree has non empty left and right subtrees,the tree is termed as a strictly binary tree. A binary tree with n leaves always contains 2n-1 nodes.



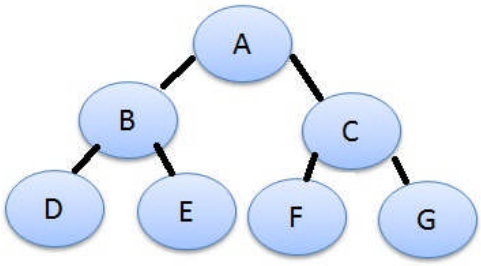
Complete binary tree:

The tree T is said to be complete if all its levels, except possibly the last have the maximum number of possible nodes and all of its leaf nodes are at same level.

A complete binary tree of depth d is the strictly binary tree all of whose leaves at level d(last level).

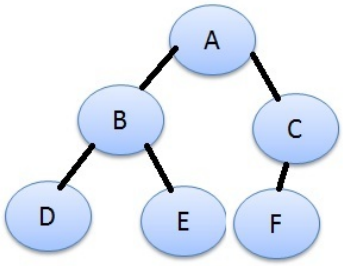
The total no. of nodes in a complete binary tree of depth d is t_n equals the sum of the no. of nodes at each level between 0 and d. Thus, $t_n=2^0+2^1+2^2+\ldots\ldots+2^d$

$= \sum_{j=0}^{d} 2^j \quad = 2^{d+1}-1$



Almost complete binary tree: A binary tree of depth d is an almost complete binary tree if:

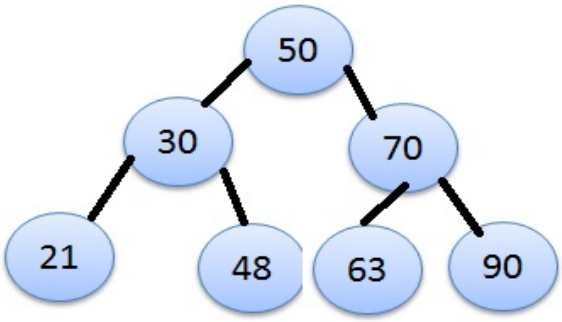
1. Each leaf in the tree is either at level d or at level d-1
2. For any node 'nd' in the tree with a right descendant at level d, all the left descendant of node nd that are leaves are also at level d



Binary Search Tree: A binary tree T is termed as binary search tree(BST)or Binary Sorted tree if each node n of T satisfies the following properties:

1. The value of n (Parent node) is greater than the values of all nodes in its left subtree and
2. The value of n (Parent node) is less than the values of all nodes in its right subtree and
3. The tree follows the properties of Almost Complete Binary tree

Example: nodes of a BST are 50, 30, 70, 21, 48, 90, 63



Further Reading:

1.Data Structures,Algorithms and Applications in C++,Satraj Sahani,2nd Edition

2.Data Structures Using C,Dr Amiya Kumar Rtah and Alok Kumar Jagdev,2nd Edition.

3.Data Structures ,Seymour Lipschutz

4.Data Structures and Algorithms,Alfred V Aho,John E Hopcroft,Jeffrey D. Ullman,1st Edition.

5.Classic Data Structure,Debadis Samanta