DSA

2322

Page No. 1
Date.

Assignment No. 04

Title: Expression tree creation & transversal

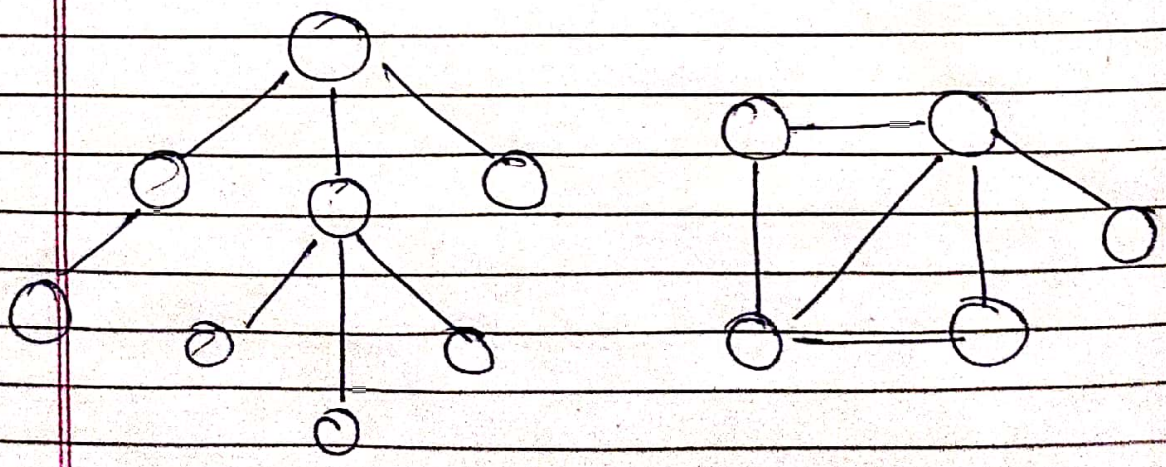
Aim: To implement a expression tree using stack data structure.

problem statement: Construct an expression tree for postfix expression ~~tree for postf~~ perform recursive and non-recursive inorder, preorder & postorder transversal

Theory:

concept of non-linear data structure with ex:

1. Data structure where data elements are not arranged linearly or sequentially are called non-linear data structures
2. In Non-linear data structure, singly level is not evolved, therefore we can't traverse all the elements in single runs only.
3. But it utilizes computer memory efficiently in comparsion to linear data structures



Tree

graph

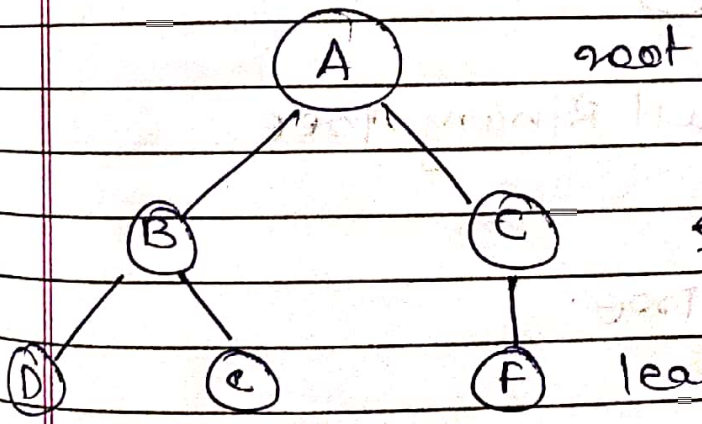
Page No.
Date.

Binary Tree :

Tree in which any node can have atmost two branches, i.e at most two children is a binary tree

Defination :

A binary tree is a finite set of nodes that is either empty or consist of root and two disjoint binary trees called 'left subtree' & 'right subtree'

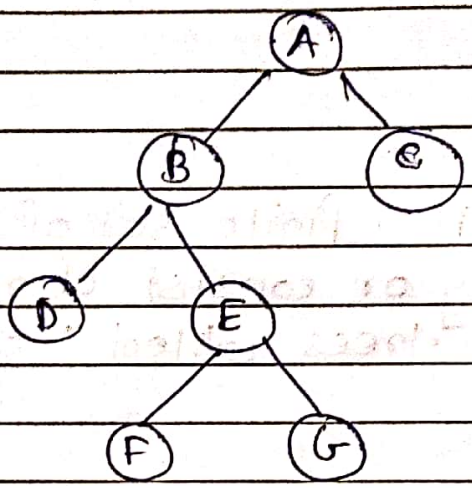


Terminologies :

1) Root : Node without parent
2) Sibling : Nodes share same parents
3) Internal Nodes : Nodes with atleast one child
4) External Nodes : Nodes without children
5) Ancestors of node : Parent, grandparent.
6) Depth of node : Number of edges from root node
7) Height of tree : Maximum depth of any node

Full Binary Tree:

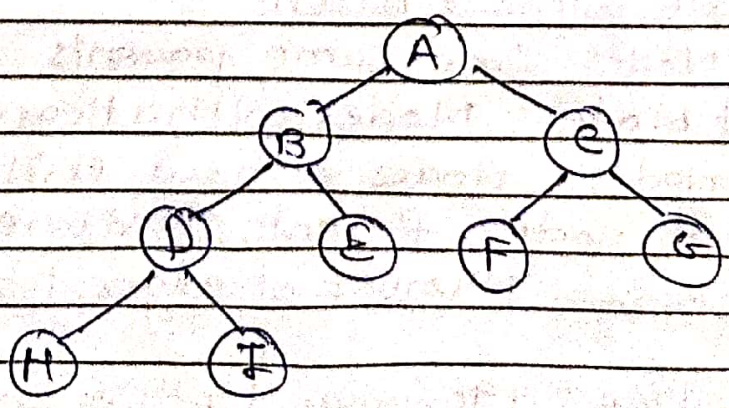
A binary tree is full binary tree if every node has zero or two children



Full Binary tree

complete Binary tree:

A complete binary tree is a binary tree which is completely filled with the possible exception of bottom level which is filled left to right



Complete Binary tree

Page No.
Date.

Binary Tree ADT:

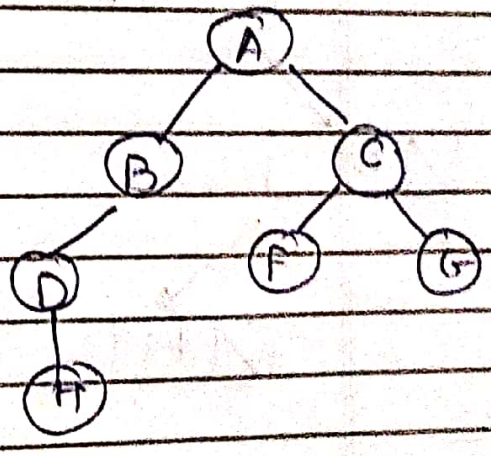
Structure Binary tree is ∞ set of node either or consisting of root, left binary tree & right binary tree.

~~Operations~~

Realization of ADT with binary:

If binary tree with 'n' nodes is represented sequentially, then for any node with index i, $1 \leq i \leq n$, we have:

parent is at $\lfloor i/2 \rfloor$
left child is at $2i$
right child is at $2i+1$



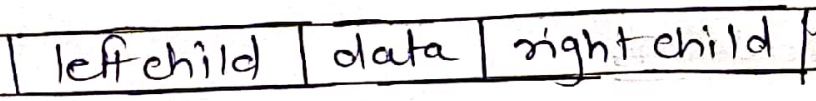
Array Representation:

0	1	2	3	4	5	6
A	B	C	D	E	F	G

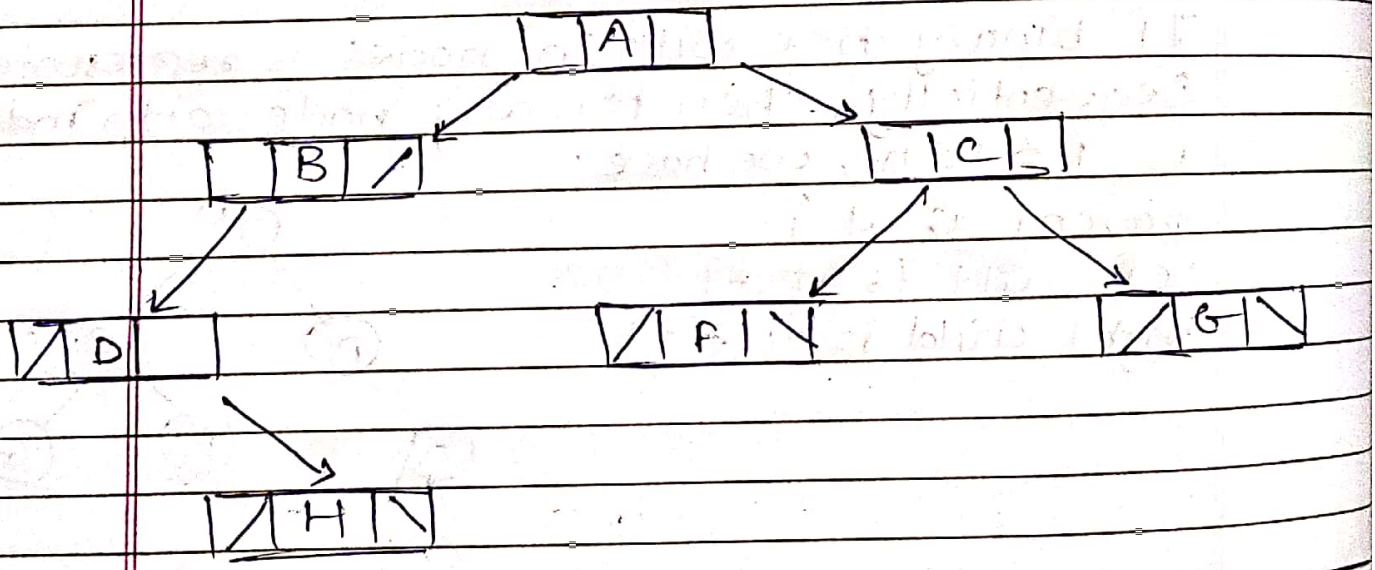
Realization of ADT with linked list:

Binary Tree is ~~called~~ linked list representation are stored in memory as linked list. These list are linked to each other through parent child relationship associate with trees.

Each node has ~~these~~ three path
1. Data element.
2. pointer that points towards left node.
3. pointer that points towards right node.



Linked list representation:



Binary tree Applications:

1) A binary tree is useful data structure when two way decisions must be made at each point in process.
Ex. Finding duplicates in list of members

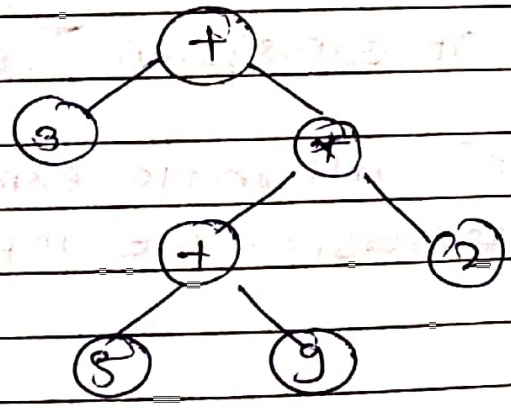
2) A binary tree can be used for representation expression containing operands (leaf) and operator (internal node)

Page No.
Date.

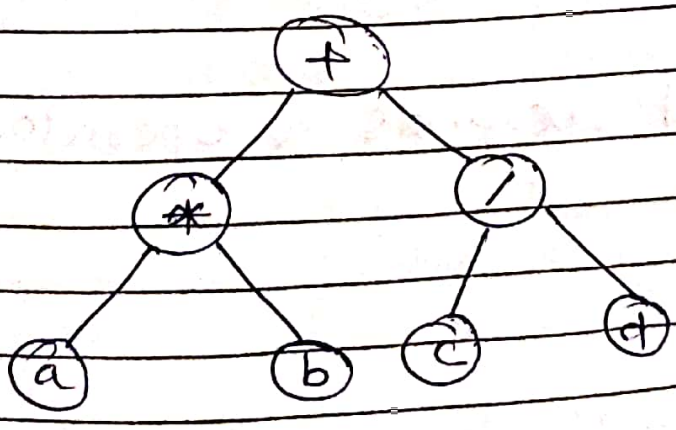
Expression Tree concepts:

An expression tree is a representation of expression arrange in a tree like data structure
It is binary tree in which internal nodes corresponding to the operator and each leaf node corresponds to the operators

eg Infix 3+((5+9)*2)

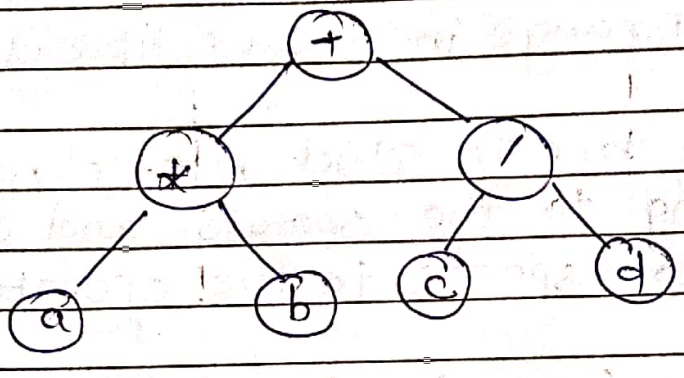


Example of prefix expression
+ * ab / cd



Example of postfix expression :

ab * cd / +



Applications of expression Tree :

1) Evaluation of arithmatic expression
2) Expression conversion i.e infix to postfix or prefix

Test cases / validations :

1) validation :

Number of operands & operators relationship

2) Test cases :

— Next page —

	Infix Expression	postfix Expression	Prefix Expression
1)	A+B*C	ABC*+	+A*CB
2)	A*B-C	AB*C-	-*ACB
3)	A^B-C	AB^C-	-^ABC
4)	A+B*C^E	ABCE^*+	+A*B^CE
5)	A-B*C+A	ABC*-A+	-+A*BCA
6)	(A+B)/(C+D)^E^F	AB+CD+EF^^/DF	/+AB+CD^EF^
7)	-D*F-D	-D-	DFD
8)	A+B+C	AB+C+	++ABC
9)	A*B/C	AB*C/	/*ACD
10)	A^B^C	ABC^^	^A^BC

Conclusion:

Using Binary tree it is possible to build an expression tree. Transversal of tree gives expression in various forms i.e inorder traversal for prefix expression preorder traversal for prefix expression, postorder transversal for