PROLOG ACADEMY

- **DATA STRUCTURE**

By- MASEERA ALI

- Book followed - Data structures by Seymour Lipschutz (Schaum Series)

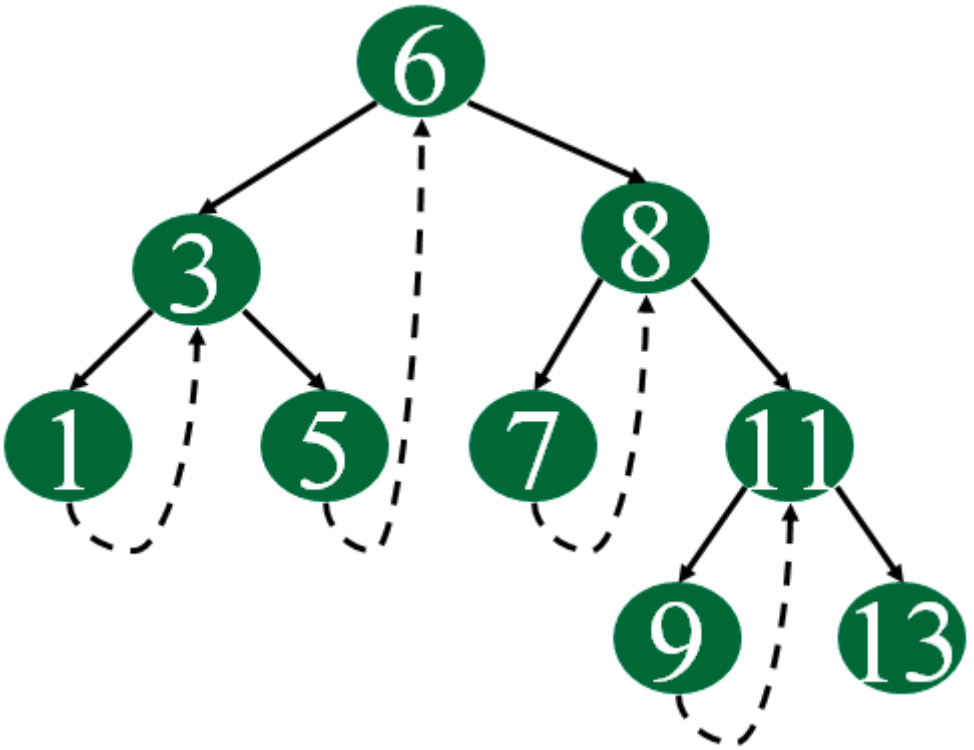
LET’S START!

Threaded Tree

- The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).
- There are two types of threaded binary trees.
- 1. Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)
- 2. Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
- The threads are also useful for fast accessing ancestors of a node.

Single Threaded Binary Tree

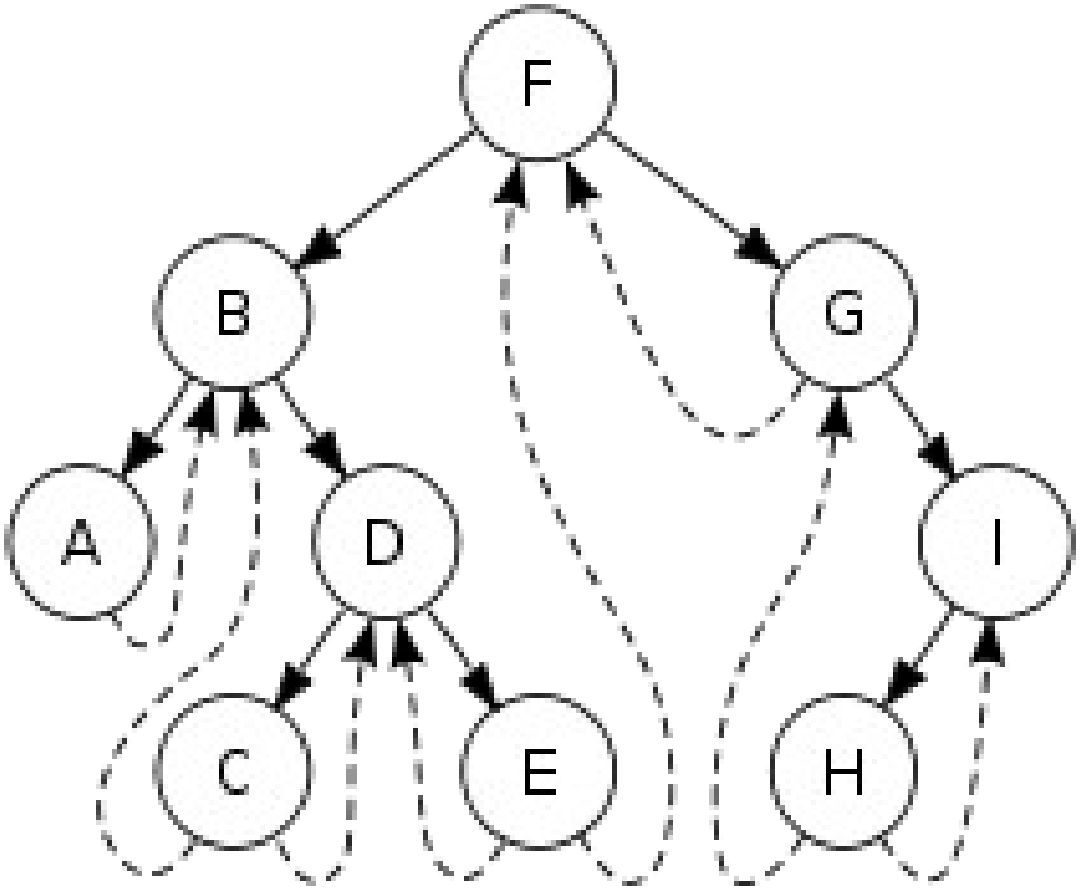
- The dotted lines represent threads
- Generally we differentiate a thread and a link by a negative sign. Links are simple pointers but threads are pointers with a negative sign.
- Below is the example of a one way inorder threading



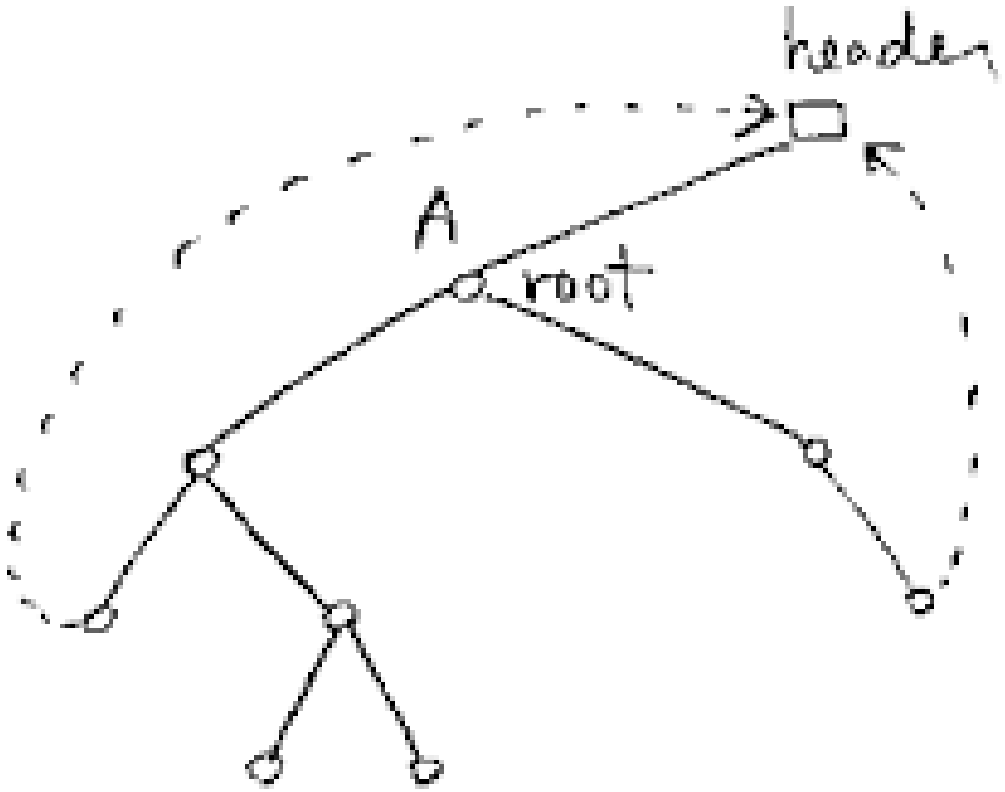
Double threaded Binary Tree

- In a Double threaded tree along with a right thread, another thread will appear in the LEFT field of the node and will point to the preceding node in the inorder traversal of tree
- Left pointer of the first node and right pointer of the last node will contain the NULL value when The tree does not have a header node.
- If the tree has a header node then these pointers will point to the header node.

Double threaded Binary Tree without header node

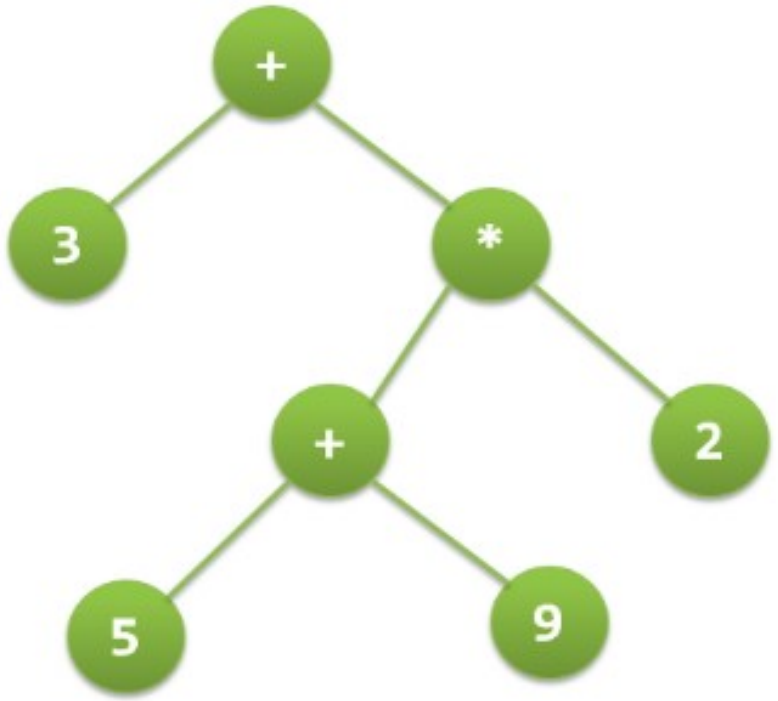


Double threaded Binary Tree with a header node



Expression Tree

- Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand
- Inorder traversal of expression tree produces infix version (same with preorder traversal and post order it gives prefix and postfix expressions respectively)
- expression tree for 3 + ((5+9)*2) would be:



Construction of Expression Tree:

- Now For constructing expression tree we use a stack. We loop through input expression and do following for every character.
- 1) If character is operand push that into stack
- 2) If character is operator pop two values from stack make them its child and push current node again.
- At the end only element of stack will be root of expression tree

Expression tree

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
struct tree
{
char data;
struct tree *left,*right;
};
int top=-1;
struct tree *stack[20];
struct tree *node;
void push(struct tree *node)
{
stack[++top]=node;
}
struct tree *pop()
{
return(stack[top--]);
}
int check(char c)
{
if(c=='+'||c=='-'||c=='/'||c=='*')
return 2;
else
return 1;
}
int cal(struct tree *node)
{
int ch;
ch=check(node->data);
if(ch==1)
return(node->data-48);
else if(ch==2)
{
if(node->data=='+')
return(cal(node->left)+cal(node->right));
else if(node->data=='-')
return(cal(node->right)-cal(node->left));
else if(node->data=='*')
return(cal(node->left)*cal(node->right));
else if(node->data=='/')
return(cal(node->right)/cal(node->left));
}
}
```

```
void operands(char b)
{
node=(struct tree*)malloc(sizeof(struct tree));
node->data=b;
node->left=NULL;
node->right=NULL;
push(node);
}
void operators(char a)
{
node=(struct tree*)malloc(sizeof(struct tree));
node->data=a;
node->left=pop();
node->right=pop();
push(node);}
void traverse(struct tree *node)
{if(node!=NULL)
{
traverse(node->right);
printf("%c",node->data);
traverse(node->left);
}
}
```

```
int main()
{
int i,p,ans;
char s[20];
printf("Enter the expression tree in postfix form: ");
fflush(stdin);
gets(s);
for(i=0;s[i]!='\0';i++)
{
p=check(s[i]);
if(p==1)
operands(s[i]);
else if(p==2)
operators(s[i]);
}
ans=cal(stack[top]);
printf("\nThe value of the postfix expression =
%d\n",ans);
printf("The actual traversal will be : ");
traverse(stack[top]);
printf("\n");
return 0;
}
```