K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment No. 6

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Expression Trees using BST

Objective: Implementation of construction of expression tree using postfix expression.

Expected Outcome of Experiment:

CO	Outcome
CO1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein; (CLRS) "Introduction to Algorithms", Third Edition, The MIT Press.

Abstract:-

NODE STRUCTURE:

1. char op //to store the operands and operators
2. node* left //to point to the left node of parent
3. node* right //to point to the right node of parent
4. node* tree[SIZE] //to store the trees at each step
5. int top=-1 //to maintain the top of stack

OPERATIONS:

1. **node* pop()**

   This function is used to pop the top most node from the stack.

2. **void push(node* temp1)**

   This function is used to push an element of type node into the stack.

3. **void inorder(node* parent)**

   This function is used to display the elements of the binary search tree in inorder form.

4. **void preorder(node* parent)**

   This function is used to display the elements of the binary search tree in preorder form.

5. **void postorder(node* parent)**

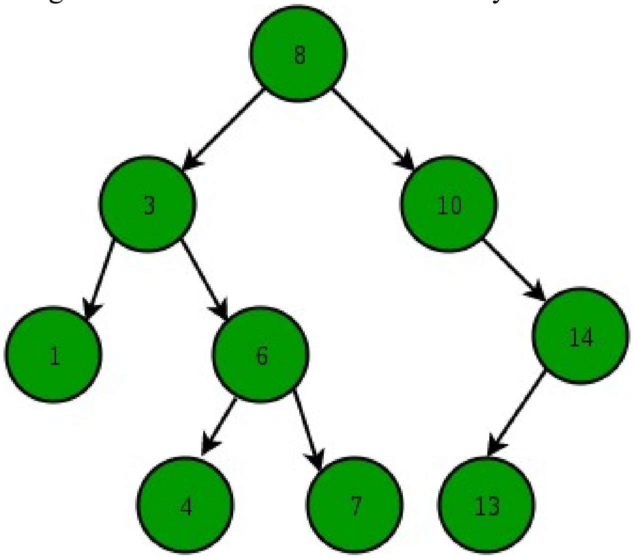
   This function is used to display the elements of the binary search tree in postorder form.

Related Theory: -

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than or equal to the node’s key.

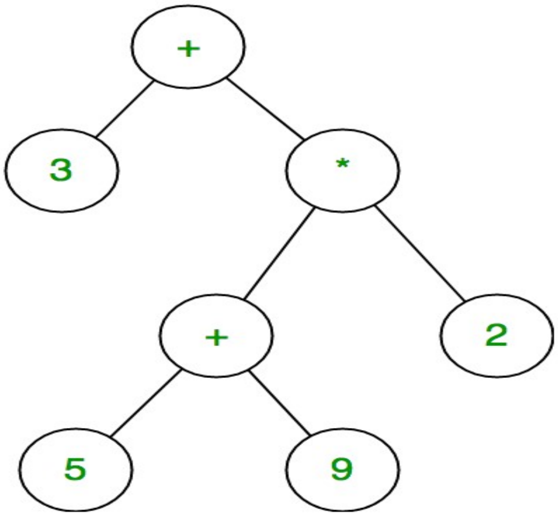
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.



The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast.

Expression Trees:

Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand so for example expression tree for 3 + ((5+9)*2) would be:



Inorder traversal of expression tree produces infix version of given postfix expression (same with preorder traversal it gives prefix expression),

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.

Implementation Details:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 50
typedef struct node{
  char op;
  struct node* left;
  struct node* right;
}node;
node* tree[SIZE];
node* temp;
int top=-1;

node* pop(){
  return tree[top--];
}
void push(node* temp1){
  tree[++top]=temp1;
}
void inorder(node *parent){
  if(parent==NULL)
    return;
  if(parent->left!=NULL)
  {
    printf("(");
    inorder(parent->left);
  }
  printf("%c ", parent->op);
  if(parent->right!=NULL){
    inorder(parent->right);
    printf(")");
  }
  else
  return;
```

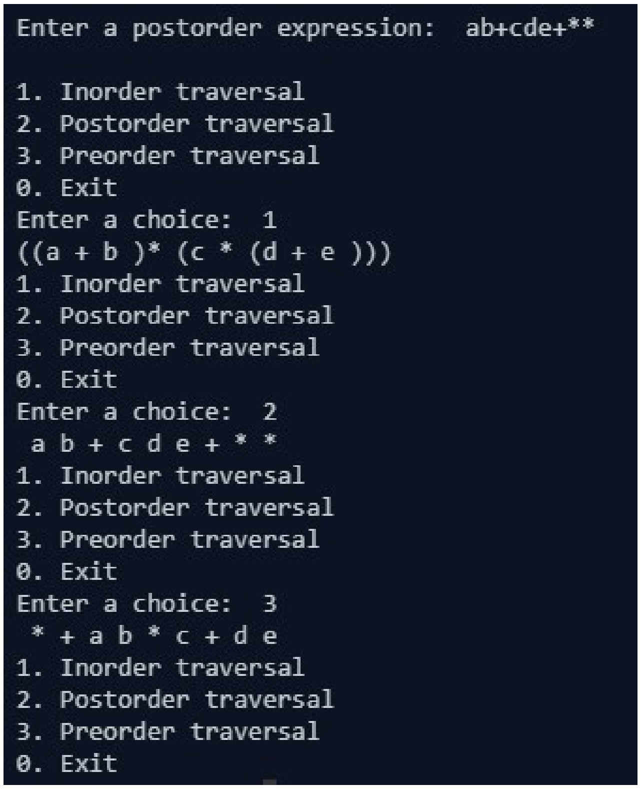
```
}

void preorder(node *parent){
  if(parent==NULL)
    return;
  else if(parent!=NULL){
    printf(" %c", parent->op);
    preorder(parent->left);
    preorder(parent->right);
  }
}
void postorder(node* parent){
  if (parent==NULL)
    return;
  else if(parent!=NULL){
    postorder(parent->left);
    postorder(parent->right);
    printf(" %c", parent->op);
    }
}

void main(){
  char postfix[SIZE];
  char e;
  int choice;
  node* a,*b;
  printf("Enter a postorder expression: ");
  scanf("%s", postfix);
  for(int i=0;i<strlen(postfix);i++){
    e=postfix[i];
    if(isalnum(e)){
      temp=(node*)malloc(sizeof(node*));
      temp->op=e;
      temp->left=temp->right=NULL;
      push(temp);
    }
    else if(e=='*' || e=='/' || e=='+' || e=='-' || e=='^' ||
e=='%'){
      b=pop();
      a=pop();
      temp=(node*)malloc(sizeof(node*));
      temp->right=b;
      temp->left=a;
      temp->op=e;
      push(temp);
    }
  }
  do{
    printf("\n1. Inorder traversal\n2. Postorder traversal\n3. Preorder traversal\n0. Exit\nEnter a choice: ");
    scanf("%d", &choice);
    switch(choice){
      case 1:
        inorder(temp);
        break;
      case 2:
        postorder(temp);
        break;
      case 3:
        preorder(temp);
        break;
      case 0:
        exit(1);
    }
  }while(choice!=0);
}
```

For verification, my code is available on:
https://repl.it/@ARGHYADEEPDAS/ExpressionTree

OUTPUT SCREEN:



```
Enter a postorder expression:  ab+cde+**

1. Inorder traversal
2. Postorder traversal
3. Preorder traversal
0. Exit
Enter a choice:  1
((a + b )* (c * (d + e )))
1. Inorder traversal
2. Postorder traversal
3. Preorder traversal
0. Exit
Enter a choice:  2
 a b + c d e + * *
1. Inorder traversal
2. Postorder traversal
3. Preorder traversal
0. Exit
Enter a choice:  3
 * + a b * c + d e
1. Inorder traversal
2. Postorder traversal
3. Preorder traversal
0. Exit
```

CONCLUSION:

The program ran successfully as we were able to implement construction of expression tree from postfix expression and we verified it by printing the tree in inorder and postorder form.