**DSF**

**ASSIGNMENT : 1**

**( Expression Tree )**

**NAME : KUNAL GUPTA**

**ROLL NO :23 | GR NO : 17U427 | BATCH : A1**

**CODE :**

**#include<iostream>**

**#include<string.h>**

**using namespace std;**

**class node //class node**

**{ public:**

**node \*left,\*right; //pointers**

**char data;**

**node(char x) //constructor**

**{**

**left=right=NULL;**

**data=x;**

**}**

**};**

**class tree //class tree**

**{ node \*root;**

**node \*create();**

**void inorder(node \*);**

**void preorder(node \*);**

**void postorder(node \*);**

**public:**

**tree()**

**{**

**root=NULL; //NULL root constructor**

**}**

**void create1() { root=create(); } //create to call from main function**

**void in() { inorder(root); } //inorder to call from main function**

**void pre() { preorder(root);} //preorder to call from main function**

**void post() { postorder(root); } //postorder to call from main function**

**void nonrec\_inorder();**

**void nonrec\_postorder();**

**void nonrec\_preorder();**

**};**

**void tree::inorder(node \*T) //rec inorder defination**

**{**

**if(T!=NULL)**

**{**

**inorder(T->left);**

**cout<<" "<<T->data;**

**inorder(T->right);**

**}**

**}**

**void tree::preorder(node \*T) //rec preorder**

**{**

**if(T!=NULL)**

**{ cout<<" "<<T->data;**

**preorder(T->left);**

**preorder(T->right);**

**}**

**}**

**void tree::postorder(node \*T) //rec postorder**

**{**

**if(T!=NULL)**

**{**

**postorder(T->left);**

**postorder(T->right);**

**cout<<" "<<T->data;**

**}**

**}**

**class stack //class stack for non rec traversals**

**{**

**node \*data[30];**

**int top;**

**public:**

**stack() //constructor**

**{**

**top=-1;**

**}**

**int isempty() //check for empty stack**

**{**

**if (top==-1)**

**return 1;**

**else**

**return 0;**

**}**

**void push(node \*p) //push value**

**{**

**data[++top]=p;**

**}**

**node \* pop() //pop value**

**{**

**return (data[top--]);**

**}**

**};**

**void tree::nonrec\_inorder() //non rec inorder traversal**

**{**

**stack s;**

**node \*p;**

**p=root;**

**while(p!=NULL)**

**{**

**s.push(p);**

**p=p->left;**

**}**

**while(!s.isempty())**

**{**

**p=s.pop();**

**cout<<" "<<p->data;**

**p=p->right;**

**while(p!=NULL)**

**{**

**s.push(p);**

**p=p->left;**

**}**

**}**

**}**

**void tree :: nonrec\_preorder() //non rec preorder traversal**

**{**

**stack s;**

**node \*p;**

**p=root;**

**while(p!=NULL)**

**{**

**cout<<" "<<p->data;**

**s.push(p);**

**p=p->left;**

**}**

**while(!s.isempty())**

**{**

**p=s.pop();**

**p=p->right;**

**while(p!=NULL)**

**{**

**cout<<" "<<p->data;**

**s.push(p);**

**p=p->left;**

**}**

**}**

**}**

**void tree :: nonrec\_postorder() //non rec postorder traversal**

**{**

**stack s,s1;**

**node \*p;**

**p=root;**

**while(p!=NULL)**

**{**

**s.push(p);**

**s1.push(NULL);**

**p=p->left;**

**}**

**while(!s.isempty())**

**{**

**p=s.pop();**

**if(s1.pop()==NULL)**

**{**

**s.push(p);**

**s1.push((node \*)1);**

**p=p->right;**

**while(p!=NULL)**

**{**

**s.push(p);**

**s1.push(NULL);**

**p=p->left;**

**}**

**}**

**else**

**{**

**cout<<" "<<p->data;**

**}**

**}**

**}**

**node \* tree::create() //create function**

**{**

**char exp[20];**

**int ch;**

**stack s; //stack to store the string**

**node \*temp,\*temp1,\*temp2;**

**cout<<"\nEnter choice \n1.Prefix Exp \n2.Postfix Exp and expression\n";**

**cin>>ch>>exp; //take exp post or pre fix**

**int i=0;**

**if(ch==2)**

**{**

**while(exp[i]!='\0')**

**{**

**if(isalnum(exp[i])) //check if operand or operator**

**{**

**temp=new node(exp[i]); //create new node**

**s.push(temp); //push into stack**

**}**

**else**

**{**

**temp2=s.pop(); //pop 2 nodes if operator is present**

**temp1=s.pop();**

**temp=new node(exp[i]); //create new operator node for operator**

**temp->right=temp2; //link the left and right part to operator**

**temp->left=temp1;**

**s.push(temp); //push the operaotor node**

**}**

**i++; //increment counter**

**}**

**root=s.pop(); //pop root for returning**

**return root;**

**}**

**else**

**{**

**i=strlen(exp)-1;**

**while(i>=0)**

**{**

**if(isalnum(exp[i])) //check if operand or operator**

**{**

**temp=new node(exp[i]); //create new node**

**s.push(temp); //push into stack**

**}**

**else**

**{**

**temp1=s.pop(); //pop 2 nodes if operator is present**

**temp2=s.pop();**

**temp=new node(exp[i]); //create new operator node Q for operator**

**temp->right=temp2;**

**temp->left=temp1;**

**s.push(temp); //push the operaotor node**

**}**

**i--;**

**}**

**root=s.pop(); //pop root for returning**

**return root;**

**}**

**}**

**int main() //main function**

**{**

**tree t; //tree object**

**int ch;**

**do**

**{**

**cout<<"\nEnter choice \n1.Accept \n2.Rec Inorder \n3.Rec Preorder \n4.Rec Postorder \n5.Non Rec Inorder \n6.Non Rec Pre \n7.Non Rec Post \n8.Exit\n";**

**cin>>ch;**

**cout<<"--------------------------------------------------------\n";**

**switch(ch)**

**{**

**case 1: t.create1(); //create call**

**break;**

**case 2: t.in(); //rec inorder traversal**

**break;**

**case 3: t.pre(); //rec pre**

**break;**

**case 4: t.post(); //rec post**

**break;**

**case 5: t.nonrec\_inorder(); //non rec inorder**

**break;**

**case 6: t.nonrec\_preorder(); //non rec preorder**

**break;**

**case 7: t.nonrec\_postorder(); //non rec postorders**

**break;**

**}**

**cout<<"\n------------------------------------------------------\n";**

**}while(ch!=8);**

**cout<<"\n------------------------------------------------------\n";**

**return 0;**

**}**

**OUTPUT :**

**Enter choice**

**1.Accept**

**2.Rec Inorder**

**3.Rec Preorder**

**4.Rec Postorder**

**5.Non Rec Inorder**

**6.Non Rec Pre**

**7.Non Rec Post**

**8.Exit**

**1**

**--------------------------------------------------------**

**Enter choice**

**1.Prefix Exp**

**2.Postfix Exp and expression**

**2**

**AB-CD+\***

**------------------------------------------------------**

**Enter choice**

**1.Accept**

**2.Rec Inorder**

**3.Rec Preorder**

**4.Rec Postorder**

**5.Non Rec Inorder**

**6.Non Rec Pre**

**7.Non Rec Post**

**8.Exit**

**2**

**--------------------------------------------------------**

**A - B \* C + D**

**------------------------------------------------------**

**Enter choice**

**1.Accept**

**2.Rec Inorder**

**3.Rec Preorder**

**4.Rec Postorder**

**5.Non Rec Inorder**

**6.Non Rec Pre**

**7.Non Rec Post**

**8.Exit**

**3**

**--------------------------------------------------------**

**\* - A B + C D**

**------------------------------------------------------**

**Enter choice**

**1.Accept**

**2.Rec Inorder**

**3.Rec Preorder**

**4.Rec Postorder**

**5.Non Rec Inorder**

**6.Non Rec Pre**

**7.Non Rec Post**

**8.Exit**

**4**

**--------------------------------------------------------**

**A B - C D + \***

**------------------------------------------------------**

**Enter choice**

**1.Accept**

**2.Rec Inorder**

**3.Rec Preorder**

**4.Rec Postorder**

**5.Non Rec Inorder**

**6.Non Rec Pre**

**7.Non Rec Post**

**8.Exit**

**5**

**--------------------------------------------------------**

**A - B \* C + D**

**------------------------------------------------------**

**Enter choice**

**1.Accept**

**2.Rec Inorder**

**3.Rec Preorder**

**4.Rec Postorder**

**5.Non Rec Inorder**

**6.Non Rec Pre**

**7.Non Rec Post**

**8.Exit**

**6**

**--------------------------------------------------------**

**\* - A B + C D**

**------------------------------------------------------**

**Enter choice**

**1.Accept**

**2.Rec Inorder**

**3.Rec Preorder**

**4.Rec Postorder**

**5.Non Rec Inorder**

**6.Non Rec Pre**

**7.Non Rec Post**

**8.Exit**

**7**

**--------------------------------------------------------**

**A B - C D + \***

**------------------------------------------------------**