- **TreeStack.h**

```
/*
 * TreeStack.h
 *
 *  Created on: Oct 27, 2020
 *      Author: Megha Sonavane
 */

#ifndef TREESTACK_H_
#define TREESTACK_H_
using namespace std;
struct TreeNode{
	char symbol;
	struct TreeNode*left,*right;
};
class TreeStack {
	int top;
	TreeNode*s[10];
public:
	TreeStack();
	bool isEmpty();
	int getTop();
	void Push(TreeNode*);
	TreeNode* pop();
	TreeNode* peek();
	virtual ~TreeStack();
};

#endif /* TREESTACK_H_ */
```

- **TreeStack.cpp**

```
/*
 * TreeStack.cpp
 *  Created on: Oct 27, 2020
 *      Author: Meghas Sonavane
 */
#include<iostream>
#include "TreeStack.h"

TreeStack::TreeStack() {
	top=-1;

}
void TreeStack::Push(TreeNode* T){
	top=top+1;
	s[top]=T;

}
bool TreeStack::isEmpty(){
	if(top==-1)
		return true;
	return false;
}
TreeNode* TreeStack::pop(){
	TreeNode* T=new TreeNode;
	T=s[top];
	top=top-1;
	return T;
```

```
}
TreeNode* TreeStack::peek(){
	return s[top];
}
int TreeStack::getTop(){
	return top;
}

TreeStack::~TreeStack() {
	// TODO Auto-generated destructor stub
}
```

- **Assignment4.cpp**

```
//============================================================================
// Name        : Assignment4.cpp

// Author      : Megha Sonavane
//
// Description : Expression tree
//============================================================================

#include <iostream>
#include "TreeStack.h"
using namespace std;

//class declaration
class ExpTree{
        TreeNode* root;
public:
        ExpTree(){
                root=NULL;
        }
        TreeNode* create_postfix(string);
        TreeNode* create_prefix(string);
        void inorder_Recursive(TreeNode*);
        void inorder_NonRecursive(TreeNode*);
        void preorder_Recursive(TreeNode*);
        void preorder_NonRecursive(TreeNode*);
        void postorder_Recursive(TreeNode*);
        void postorder_NonRecursive(TreeNode*);
```

```
};
//-------------------------definition of create from prefix-------------------------------
TreeNode* ExpTree::create_prefix(string prefix){
	TreeNode* newNode;
	TreeStack s;
	string reverse="";
	int len=prefix.length();
	for(int i=len-1;i>=0;i--){ //reversing the prefix expression
		reverse+=prefix[i];
	}
	len=reverse.length();
	//-----------------tree creation-----------------------------------
	for(int i=0;i<len;i++)
	{
		//------if it is operand-----------------
		if(isalpha(reverse[i])){
			//create new node and push into stack
				newNode=new TreeNode;
				newNode->symbol=reverse[i];
				newNode->left=NULL;
				newNode->right=NULL;
				s.Push(newNode);
			}
		//-----if it is operator-------
		else{
				//create new node and set left and right child
				newNode=new TreeNode;
				newNode->symbol=reverse[i];
				newNode->left=s.pop();
```

```
                        newNode->right=s.pop();
                        s.Push(newNode); //push into stack
                }
        }
        root=s.pop(); //root is at top of stack
        cout<<"***Tree Created***"<<endl;
        return root;
}
//-------------------------definition of create from postfix-----------------------------
TreeNode *ExpTree::create_postfix(string exp){

        TreeNode* newNode;
        TreeStack s;
        int len=exp.length();
        for(int i=0;i<len;i++)
        {
                //------if it is operand--------
                if(isalpha(exp[i])){
                        //create new node and push into stack
                        newNode=new TreeNode;
                        newNode->symbol=exp[i];
                        newNode->left=NULL;
                        newNode->right=NULL;
                        s.Push(newNode);
                }
                //-----if it is operator-------
                else{
                        //create new node and set left and right child
                        newNode=new TreeNode;
```

```
                newNode->symbol=exp[i];
                newNode->right=s.pop();
                newNode->left=s.pop();
                s.Push(newNode);
            }
        }
        root=s.pop(); //root is at top of stack
        cout<<"***Tree Created***"<<endl;
        return root;
}
//-------------------------------------recursive inorder ----------------------------------------------------------------
void ExpTree::inorder_Recursive(TreeNode* root){

        if(root==NULL)
            return;
        inorder_Recursive(root->left);
        cout<<root->symbol;
        inorder_Recursive(root->right);


}
//-----------------------------------------non-recursive inorder-------------------------------------------------------
void ExpTree::inorder_NonRecursive(TreeNode* T){
        TreeStack s;
        while((T!=NULL) || !(s.isEmpty()) ){
            while(T!=NULL)
            {
                s.Push(T);
                T=T->left;
```

```
                }
                if(!s.isEmpty()){
                        T=s.pop();
                        cout<<T->symbol;
                        T=T->right;
                }
        }
}
//---------------------------------------recursive preorder-----------------------------------------------------------
void ExpTree::preorder_Recursive(TreeNode*T){
        if(T==NULL)
                return;
        cout<<T->symbol;
        preorder_Recursive(T->left);
        preorder_Recursive(T->right);
}
//---------------------------------------non-recursive preorder-------------------------------------------------------
void ExpTree::preorder_NonRecursive(TreeNode*T){
        TreeStack s;
        while((T!=NULL)||!(s.isEmpty())){
                while(T!=NULL)
                {
                        cout<<T->symbol;
                        s.Push(T);
                        T=T->left;
                }
                if(!s.isEmpty()){
                        T=s.pop();
                        T=T->right;
```

```
                }
        }
}
//-----------------------------------------recursive postorder-----------------------------------------------
void ExpTree::postorder_Recursive(TreeNode*T){
        if(T==NULL)
                return;
        postorder_Recursive(T->left);
        postorder_Recursive(T->right);
        cout<<T->symbol;
}
//-----------------------------------------non-recursive postorder-----------------------------------------
void ExpTree::postorder_NonRecursive(TreeNode*T){
        int flag[10];
        TreeStack s;
        while(T!=NULL||!(s.isEmpty()))
        {
                while(T!=NULL)
                {
                        s.Push(T);
                        flag[s.getTop()]=1;
                        T=T->left;
                }
                T=s.peek();
                if(flag[s.getTop()]==2){
                        cout<<T->symbol;
                        s.pop();
                        T=NULL;
                }
```

```
                else{
                        flag[s.getTop()]=2;
                        T=T->right;
                }

        }

}
//-----------------------------------------driver function-----------------------------------------------------------
int main() {
        ExpTree e;
        TreeNode* root;
        int ch;
        string exp;
        cout<<"\t****Creation of tree****"<<endl;
        cout<<"\t1:From prefix expression"<<endl<<"\t2:From postfix expression"<<endl;
        cout<<"Enter choice:"; //enter choice for prefix or postfix expression
        cin>>ch;
        if(ch==1){
                //creation of tree from prefix expression
                cout<<"Enter prefix expression:";
                cin>>exp;
                root=e.create_prefix(exp);
        }
        else{
                //creation of tree from poostfix expression
                cout<<"Enter postfix expression:";
                cin>>exp;
                root=e.create_postfix(exp);
```

```
        }

        do{
                //display menus to user
                cout<<endl<<"-----------------------------------------------------------"<<endl;
                cout<<"\t1:Recursive inorder"<<endl<<"\t2:Non-recursive inorder"<<endl;
                cout<<"\t3:Recursive preorder"<<endl<<"\t4:Non-recursive preorder"<<endl;
                cout<<"\t5:Recursive postorder"<<endl<<"\t6:Non-recursive postorder"<<endl<<"\t7:Enter new
expression"<<endl<<"\t8:Exit"<<endl;
                cout<<"\tEnter choice:";
                cin>>ch;
                switch(ch)
                {
                case 1:
                        cout<<"\tResult==> ";
                        e.inorder_Recursive(root);
                        break;
                case 2:
                        cout<<"\tResult==> ";
                        e.inorder_NonRecursive(root);
                        break;
                case 3:
                        cout<<"\tResult==> ";
                        e.preorder_Recursive(root);
                        break;
                case 4:
                        cout<<"\tResult==> ";
                        e.preorder_NonRecursive(root);
                        break;
```

```
case 5:
        cout<<"\tResult==> ";
        e.postorder_Recursive(root);
        break;
case 6:
        cout<<"\tResult==> ";
        e.postorder_NonRecursive(root);
        break;
case 7:
        cout<<"\t****Tree creation****"<<endl;
        cout<<"\t1:From prefix expression"<<endl<<"\t2:From postfix expression"<<endl;
        cout<<"\tEnter choice:"; //enter choice for prefix or postfix expression
        cin>>ch;
        if(ch==1){
                //creation of tree from prefix expression
                cout<<"\tEnter prefix expression:";
                cin>>exp;
                root=e.create_prefix(exp);
        }
        else{
                //creation of tree from poostfix expression
                cout<<"\tEnter postfix expression:";
                cin>>exp;
                root=e.create_postfix(exp);
        }

        break;
case 8:
        cout<<"\tThank you..."<<endl;
```

```
                    break;
                default:
                    cout<<"\tInvalid choice..."<<endl;;
                }
        }while(ch!=8);
        return 0;
}
```

- **Output:**

****Creation of tree****
1:From prefix expression
2:From postfix expression
Enter choice:1
Enter prefix expression:*+ab+cd
Tree Created

1:Recursive inorder
2:Non-recursive inorder
3:Recursive preorder
4:Non-recursive preorder
5:Recursive postorder
6:Non-recursive postorder
7:Enter new expression
8:Exit
Enter choice:1
Result==> a+b*c+d

1:Recursive inorder
2:Non-recursive inorder
3:Recursive preorder
4:Non-recursive preorder
5:Recursive postorder
6:Non-recursive postorder
7:Enter new expression
8:Exit

```
        Enter choice:2
        Result==> a+b*c+d
-------------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:3
        Result==> *+ab+cd
-------------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:4
        Result==> *+ab+cd
-------------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
```

```
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:5
        Result==> ab+cd+*
-------------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:6
        Result==> ab+cd+*
-------------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:7
        ****Tree creation****
        1:From prefix expression
```

```
        2:From postfix expression
        Enter choice:2
        Enter postfix expression:ab+cd+*
***Tree Created***

-----------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:1
        Result==> a+b*c+d
-----------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:2
        Result==> a+b*c+d
-----------------------------------------------------
        1:Recursive inorder
```

2:Non-recursive inorder
3:Recursive preorder
4:Non-recursive preorder
5:Recursive postorder
6:Non-recursive postorder
7:Enter new expression
8:Exit
Enter choice:3
Result==> *+ab+cd

1:Recursive inorder
2:Non-recursive inorder
3:Recursive preorder
4:Non-recursive preorder
5:Recursive postorder
6:Non-recursive postorder
7:Enter new expression
8:Exit
Enter choice:4
Result==> *+ab+cd

1:Recursive inorder
2:Non-recursive inorder
3:Recursive preorder
4:Non-recursive preorder
5:Recursive postorder
6:Non-recursive postorder
7:Enter new expression
8:Exit

```
        Enter choice:5
        Result==> ab+cd+*
-------------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:6
        Result==> ab+cd+*
-------------------------------------------------------
        1:Recursive inorder
        2:Non-recursive inorder
        3:Recursive preorder
        4:Non-recursive preorder
        5:Recursive postorder
        6:Non-recursive postorder
        7:Enter new expression
        8:Exit
        Enter choice:8
        Thank you...
```