EXPRESSION_TREE.cpp

```
/*
 ROLL NUMBER : 2002
BATCH : E-10
*/

#include "tree.h"
#include<string>
#include<stack>
#include<iostream>
#include<malloc.h>
#include<stdlib.h>
using namespace std;

int main()
{

	tree t;
		string s;

		node *root;

	do
		{
			int ch,c;
			cout<<"----------------------MENU--------------------------\n";

			cout<<"\t\t1.Create a Tree\n\t\t"
				"2.Recursive Traversal\n\t\t"
				"3.Non-Recursive Traversal\n\t\t"
				"4.Exit\n\n";

			cout<<"Enter Choice : ";
			cin>>ch;
			cout<<endl;

			switch(ch)
			{
				case 1:
				{

					cout<<"\n\n\t1.Postfix Expression\n\t2.Prefix Expression\n\n";
					cout<<"Enter Choice : ";
					cin>>c;
					cout<<endl;

					switch(c)
					{
						case 1 : cout<<"Enter a postfix expression : ";
```

```
                                        cin>>s;
                                        cout<<endl;

                                        cout<<"Tree Created !!!\n";
                                        t.construct_tree_postorder(s);
                                        break;

                        case 2 : cout<<"Enter a prefix expression : ";
                                        cin>>s;
                                        cout<<endl;

                                        cout<<"Tree Created !!!\n";
                                        t.construct_tree_preorder(s);
                                        break;
                }

                break;

        }

        case 2:
        {
                cout<<"\n\n\t1.Recurssive Inorder\n\t\t2.Recurssive
Postorder\n\t\t"
                                "3.Recurssive Preorder\n\n";
                cout<<"Enter Choice : ";
                cin>>c;
                cout<<endl;

                switch(c)
                {
                        case 1 : cout<<"Inorder Traversal is : ";
                                        t.inorder(t.root);
                                        cout<<endl<<endl;
                                        break;

                        case 2 : cout<<"Postorder Traversal is : ";
                                        t.postorder(t.root);
                                        cout<<endl<<endl;
                                        break;

                        case 3 : cout<<"Preorder Traversal is : ";
                                        t.preorder(t.root);
                                        cout<<endl<<endl;
                                        break;
                }

                break;
        }

        case 3:
        {
```

```
                                        cout<<"\n\n\t1.Non-Recurssive Inorder\n\t\t2.Non-Recurssive
Postorder\n\t\t"
                                                "3.Non-Recurssive Preorder\n\n";
                                        cout<<"Enter Choice : ";
                                        cin>>c;
                                        cout<<endl;

                                        switch(c)
                                        {
                                                case 1 : cout<<"Inorder Traversal is : ";
                                                        t.inorder_non_recurs(t.root);
                                                        cout<<endl<<endl;
                                                        break;

                                                case 2 : cout<<"Postorder Traversal is : ";
                                                        t.postrder_non_recurs(t.root);
                                                        cout<<endl<<endl;
                                                        break;

                                                case 3 : cout<<"Preorder Traversal is : ";
                                                        t.preorder_non_recurs(t.root);
                                                        cout<<endl<<endl;
                                                        break;
                                        }

                                        break;
                                }

                                case 4:
                                {
                                        exit(0);
                                        break;
                                }

                        }

                }
                while(1);

        return 0;
}
```

TREE.h

```
/*
 * tree.h
 *
 *  Created on: 29-Jan-2018
 *      Author: e2002
 */

#include<string>
#include<stack>
using namespace std;

#ifndef TREE_H_
#define TREE_H_

struct node
{
	node *left;
	node *right;

	char data;
};

class tree
{

	public:
			node * root;
			node *  create(char x);

			void construct_tree_postorder(string str);
			void construct_tree_preorder(string str);

			void inorder(node* );
			void preorder(node* );
			void postorder(node* );

			void inorder_non_recurs(node *);
			void preorder_non_recurs(node *);
			void postrder_non_recurs(node *);
};

#endif /* TREE_H_ */
```

TREE.cpp

```
/*
 * tree.cpp
 *
 *  Created on: 29-Jan-2018
 *      Author: e2002
 */

#include "tree.h"
#include<string>
#include<stack>
#include<iostream>
#include<malloc.h>
using namespace std;


int isoperator(char x)
{
    if( ( (x== '+') || (x == '-') || (x == '*') || (x == '/') || (x == '^') ) )
        return 1;

    else
        return 0;
}

node * tree :: create(char x)
{
        node *p;
        if(x==-1)
                return NULL;

        p=new node;

        p->data=x;

        p->left=NULL;
        p->right=NULL;

        return p;

}

void tree :: construct_tree_postorder(string str)
{
        stack<node *>s;
        node *temp;

        int len=str.length();

        for(int i=0;i<len;i++)
```

```
	{
		char x=str[i];

		if(!isoperator(x))
		{
			node * temp;

			temp=create(x);
			s.push(temp);
		}

		else
		{
			temp=create(x);

			temp->right=s.top();
			s.pop();

			temp->left=s.top();
			s.pop();

			s.push(temp);

		}
	}

	root=temp;
}

void tree :: construct_tree_preorder(string str)

{
	stack<node *>s;
		node *temp;

		int len=str.length();

		for(int i=len-1;i>=0;i--)
		{
			char x=str[i];

			if(!isoperator(x))
			{
				node * temp;

				temp=create(x);
				s.push(temp);
			}

			else
			{
				temp=create(x);
```

```
                                temp->left=s.top();
                                s.pop();

                                temp->right=s.top();
                                s.pop();

                                s.push(temp);

                        }
                }

                root=temp;
}

void tree :: inorder(node * root)
{
        if(root == NULL)
                return;

        cout<<"(";
        inorder(root->left);
        cout<<root->data;
        inorder(root->right);
        cout<<")";
}

void tree :: preorder(node *root)
{
        if(root == NULL)
                return;

        cout<<root->data;
        preorder(root->left);
        preorder(root->right);
}

void tree :: postorder(node *root)
{
        if(root == NULL)
                return;

        postorder(root->left);
        postorder(root->right);
        cout<<root->data;
}

void tree :: inorder_non_recurs(node *root)

{
        stack<node *>s;
```

```
        while(1)
        {
                while(root != NULL)
                {
                        s.push(root);
                        root=root->left;
                }

                if(s.empty())
                        return;


                root=s.top();
                s.pop();

                cout<<root->data<<" ";
                root=root->right;


        }
}

void tree :: preorder_non_recurs(node *root)

{
        stack<node *>s;

                while(1)
                {
                        while(root != NULL)
                        {
                                cout<<root->data<<" ";
                                s.push(root);
                                root=root->left;
                        }

                        if(s.empty())
                                return;

                        root=s.top();
                        s.pop();

                        root=root->right;

                }
}

void tree :: postrder_non_recurs(node *root)

{
        stack<node *>s1,s2;
```

```
        s1.push(root);

        while(! s1.empty())
        {
                node * temp;
                temp=s1.top();
                s1.pop();

                s2.push(temp);

                if(temp->left != NULL)
                                s1.push(temp->left);

                if(temp->right != NULL)
                                s1.push(temp->right);
        }

        while(!s2.empty())
        {
                node *temp;
                temp=s2.top();
                cout<<temp->data<<" ";
                s2.pop();
        }
}
```