Assignment 2

<u>Problem Statement :</u> Construct an expression tree from postfix/prefix expression and perform recursive and non recursive In-order, pre-order, post-order traversals. Input : postfix expression.

a. Create tree
b. All Recursive Traversals
c. All Non Recursive Traversals

<u>Code :</u>

```
#include<iostream>
using namespace std;

struct node{
  char data;
  node *left, *right;
};

class stack
{
  public:
    node *st[20];
    char st1[20];
    char data;
    int top,top1;
  public:
    stack()
    {
      top = -1;
      top1 = -1;
    }
    int isEmpty()
    {
      if(top==-1)
        return 1;
      else
        return 0;
    }
    int isEmpty1()
    {
      if(top1==-1)
        return 1;
      else
        return 0;
    }
    void push(node *nwnode)
    {
      top++;
      st[top] = nwnode;
    }
    node *pop()
    {
      if(isEmpty())
        return NULL;
      node *nwnode;
      nwnode = st[top];
      top--;
      return(nwnode);
    }
    void push1(char c)
    {
      top1++;
      st1[top1] = c;
    }
    char pop1()
    {
      if(!isEmpty1())
      {
        char c = st1[top1];
        top1--;
        return c;
      }
    }
};

class exptree
{
  private:
    char post[20];
  public:
  stack S;
  node *root;
  exptree()
  {
    root = NULL;
  }
  node* newnode(char);
  bool isOperator(char);
  void convert();
  void inorder(node *);
  void preorder(node *);
  void postorder(node *);
  void nr_inorder();
  void nr_preorder();
  void nr_postorder();
  void recursive_traversals();
  void non_recursive_traversals();
};

node* exptree::newnode(char c)
{
  node *tmp = new node;
  tmp->data = c;
  tmp->left = tmp->right = NULL;
  return tmp;
}

bool exptree::isOperator(char c)
{
  if(c=='+' || c=='-' || c=='*' || c=='/' || c=='^')
    return true;
  return false;
}

void exptree::convert()
{

  node *t, *t1, *t2;
  cout<<"\nEnter Postfix expression : ";
  cin>>post;
  for(int i=0;post[i]!='\0';i++)
  {
    if(isalpha(post[i]) || isdigit(post[i]))
    {
      t = newnode(post[i]);
      S.push(t);
    }
```

```
      else if(isOperator(post[i]))
      {
        t = newnode(post[i]);
        t2 = S.pop();
        t1 = S.pop();
        t->left = t1;
        t->right = t2;
        S.push(t);
      }
    }
    t = S.pop();
    root = t;
}

void exptree::inorder(node* temp)
{
    if(temp!=NULL)
    {
      inorder(temp->left);
      cout<<temp->data<<" ";
      inorder(temp->right);
    }
}

void exptree::preorder(node* temp)
{
    if(temp!=NULL)
    {
      cout<<temp->data<<" ";
      preorder(temp->left);
      preorder(temp->right);
    }
}

void exptree::postorder(node *temp)
{
    if(temp!=NULL)
    {
      postorder(temp->left);
      postorder(temp->right);
      cout<<temp->data<<" ";
    }
}

void exptree::nr_inorder()
{
    stack S;
    node *temp = root;
    while(1)
    {
      while(temp!=NULL)
      {
        S.push(temp);
        temp = temp->left;
      }
      if(S.isEmpty())
        return;
      temp = S.pop();
      cout<<temp->data<<" ";
      temp = temp->right;
    }
}

void exptree::nr_preorder()
```

```
{
  stack S;
  node *temp = root;
  while(1)
  {
    while(temp!=NULL)
    {
      S.push(temp);
      cout<<temp->data<<" ";
      temp = temp->left;
    }
    if(S.isEmpty())
      return;
    temp = S.pop();
    temp = temp->right;
  }
}

void exptree::nr_postorder()
{
  stack S;
  char flag;
  node *temp = root;
  while(1)
  {
    while(temp!=NULL)
    {
      S.push(temp);
      S.push1('L');
      temp = temp->left;
    }
    if(S.isEmpty())
      return;
    else{
      temp = S.pop();
      flag = S.pop1();
      if(flag=='R')
      {
        cout<<temp->data<<" ";
        temp = NULL;
      }
      else{
        S.push(temp);
        S.push1('R');
        temp = temp->right;
      }
    }
  }
}

void exptree::recursive_traversals()
{
  cout<<"\n-----------------------------------------------------------------";
  cout<<"\nRecursive Traversals!";
  cout<<"\nInorder Traversal : ";
  inorder(root);
  cout<<"\nPreorder Traversal : ";
  preorder(root);
  cout<<"\nPostorder Traversal : ";
  postorder(root);
}

void exptree::non_recursive_traversals()
{
  cout<<"\n--------------------------------------------------------------------";
  cout<<"\nNon Recursive Traversals!";
  cout<<"\nInorder Traversal : ";
  nr_inorder();
  cout<<"\nPreorder Traversal : ";
  nr_preorder();
  cout<<"\nPostorder Traversal : ";
  nr_postorder();
}

int main()
{
  exptree obj;
  int choice,ch;
  cout<<"\n1. Create an Expression Tree \n2. Exit"<<endl;
  cout<<"\nEnter your choice : ";
  cin>>choice;
  cout<<"\n--------------------------------------------------------------------";
  if(choice==1)
  {
    obj.convert();
    cout<<"\nExpression tree created Successfully!";
    while(1)
    {
      cout<<"\n--------------------------------------------------------------------";
      cout<<"\n1. Create new Tree \n2. Display Recursive Traversals \n3. Display Non Recursive Traversals \n4. Exit";
      cout<<"\nEnter your choice : ";
      cin>>ch;
      cout<<"\n--------------------------------------------------------------------";
      if(ch==1)
      {
        obj.convert();
        cout<<"\nExpression tree created Successfully!";
      }
      else if(ch==2)
      {
        obj.recursive_traversals();
      }
      else if(ch==3)
      {
        obj.non_recursive_traversals();
      }
      else{
        cout<<"\nProgram Exited!";
        exit(0);
      }
    }
  }
  else
  {
    cout<<"\nProgram Exited Successfully!";
  }
  return 0;
}
```

OUTPUT

```
C:\Users\Safir\Desktop\Sem 4\ADS\Code>g++ 2_expression_tree.cpp
C:\Users\Safir\Desktop\Sem 4\ADS\Code>a

1. Create an Expression Tree

2. Exit

Enter your choice : 1

 ----------------------------------------------------------------------

Enter Postfix expression : ab+cd-*

Expression tree created Successfully!

 ----------------------------------------------------------------------

 1. Create new Tree

2. Display Recursive Traversals

3. Display Non Recursive Traversals

4. Exit

Enter your choice : 2

----------------------------------------------------------------------

Recursive Traversals!

Inorder Traversal : a + b * c - d

Preorder Traversal : * + a b - c d

Postorder Traversal : a b + c d - *

 ----------------------------------------------------------------------

1. Create new Tree

2. Display Recursive Traversals

3. Display Non Recursive Traversals

4. Exit

Enter your choice : 3

----------------------------------------------------------------------

Non Recursive Traversals!

 Inorder Traversal : a + b * c - d

Preorder Traversal : * + a b - c d

Postorder Traversal : a b + c d - *

----------------------------------------------------------------------

1. Create new Tree

2. Display Recursive Traversals

3. Display Non Recursive Traversals

4. Exit

Enter your choice : 4

----------------------------------------------------------------------

Program Exited!
```