Lab Assignment 4

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

Name: Prathamesh Sadashiv Gadekar

Roll no.:14

Batch: S1

Class :SE-IT

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

Construct an Expression Tree from postfix and prefix expression. Perform recursive and non- recursive In-order, pre-order and post-order traversals.

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

#include <iostream>

#include <stack>

#include <queue>

using namespace std;

struct Node

{

char data;

Node \*left, \*right;

};

class Expression

{

char prefix[20], postfix[20], infix[20];

Node\* root;

int top;

public:

void input(int);

Node\* getroot()

{

return root;

}

int validate(int);

Node\* create\_Post();

void inorder\_withRec(Node\*);

void preorder\_withRec(Node\*);

void postorder\_withRec(Node\*);

void inorder\_withoutRec(Node\*);

void preorder\_withoutRec(Node\*);

void postorder\_withoutRec(Node\*);

};

int Expression::validate(int x)

{

int i = 0, opn = 0, opt = 0;

if(x == 0)

{

for(;i<prefix[i]!='\0';i++)

{

if(prefix[i] == '+' || prefix[i] == '-' ||prefix[i] == '\*' || prefix[i] == '/' || prefix[i] == '%' || prefix[i] == '^' || prefix[i] == '$')

opt++;

else

opn++;

}

if(opt != (opn-1))

return 0;

return 1;

}

else

{

for(;i<postfix[i]!='\0';i++)

{

if(postfix[i] == '+' || postfix[i] == '-' ||postfix[i] == '\*' || postfix[i] == '/' || postfix[i] == '%' || postfix[i] == '^' || postfix[i] == '$')

opt++;

else

opn++;

}

if(opt != (opn-1))

return 0;

return 1;

}

}

void Expression::input(int x)

{

if(x == 0)

{

while(1)

{

cout<<"\nEnter prefix Expression: ";

cin>>prefix;

if(validate(x))

break;

else

cout<<"\nWrong Expression. Please enter again.";

}

}

else

{

while(1)

{

cout<<"\nEnter postfix Expression: ";

cin>>postfix;

if(validate(x))

break;

else

cout<<"\nWrong Expression. Please enter again.";

}

}

}

Node\* Expression::create\_Post()

{

int i=0;

Node\* temp, \*t1, \*t2;

stack<Node\*> s;

for(;postfix[i]!='\0';i++)

{

if(isalnum(postfix[i]))

{

temp = new Node;

temp->data = postfix[i];

temp->left = temp->right = NULL;

s.push(temp);

}

else

{

t2 = s.top();

s.pop();

t1 = s.top();

s.pop();

temp = new Node;

temp->data = postfix[i];

temp->left = t1;

temp->right = t2;

s.push(temp);

}

}

temp = s.top();

s.pop();

return temp;

}

void Expression::inorder\_withRec(Node\* t)

{

if(t!=NULL)

{

inorder\_withRec(t->left);

cout<<" "<<t->data;

inorder\_withRec(t->right);

}

}

void Expression::preorder\_withRec(Node\* t)

{

if(t!=NULL)

{

cout<<" "<<t->data;

preorder\_withRec(t->left);

preorder\_withRec(t->right);

}

}

void Expression::postorder\_withRec(Node\* t)

{

if(t!=NULL)

{

postorder\_withRec(t->left);

postorder\_withRec(t->right);

cout<<" "<<t->data;

}

}

void Expression::inorder\_withoutRec(Node\* t)

{

Node\* temp = t;

stack<Node\*> s;

while(temp!=NULL)

{

s.push(temp);

temp = temp->left;

}

while(!s.empty())

{

temp = s.top();

s.pop();

cout<<" "<<temp->data;

temp = temp->right;

while(temp!=NULL)

{

s.push(temp);

temp = temp->left;

}

}

}

void Expression::preorder\_withoutRec(Node\* t)

{

Node\* temp = t;

stack<Node\*> s;

while(temp != NULL)

{

cout<<" "<<temp->data;

s.push(temp);

temp = temp->left;

}

while(!s.empty())

{

temp = s.top();

s.pop();

temp = temp->right;

while(temp != NULL)

{

cout<<" "<<temp->data;

s.push(temp);

temp = temp->left;

}

}

}

void Expression::postorder\_withoutRec(Node\* t)

{

Node \*t1, \*temp = t;

stack<Node\*> s, s1;

while(temp!=NULL)

{

s.push(temp);

s1.push(NULL);

temp = temp->left;

}

while(!s.empty())

{

temp = s.top();

s.pop();

t1 = s1.top();

s1.pop();

if(t1 == NULL)

{

s.push(temp);

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

temp = temp->right;

while(temp!=NULL)

{

s.push(temp);

s1.push(NULL);

temp = temp->left;

}

}

else

cout<<" "<<temp->data;

}

}

int main()

{

int choice;

Expression et;

Node\* t;

cout<<"\nPlease enter the postfix string: ";

et.input(1);

t = et.create\_Post();

while(1)

{

cout<<"\n\nExpression Tree";

cout<<"\n Recursive Traversals: ";

cout<<"\n1. Inorder Traversal";

cout<<"\n2. Preorder Traversal";

cout<<"\n3. Postorder Traversal";

cout<<"\nNon-Recursive Traversals: ";

cout<<"\n4. Inorder Traversal";

cout<<"\n5. Preorder Traversal";

cout<<"\n6. Postorder Traversal";

cout<<"\n7. Exit the program";

cout<<"\nEnter your choice:- ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\nInorder traversal with recursion:";

et.inorder\_withRec(t);

break;

case 2:

cout<<"\nPreorder traversal with recursion:";

et.preorder\_withRec(t);

break;

case 3:

cout<<"\nPostorder traversal with recursion:";

et.postorder\_withRec(t);

break;

case 4:

cout<<"\nInorder traversal without recursion:";

et.inorder\_withoutRec(t);

break;

case 5:

cout<<"\nPreorder traversal without recursion:";

et.preorder\_withoutRec(t);

break;

case 6:

cout<<"\nPostorder traversal without recursion:";

et.postorder\_withoutRec(t);

break;

case 7:

return 0;

default:

cout<<"\nWrong choice. Please enter again.";

}

}

return 0;

}

**OUTPUT**

Please enter the postfix string:

Enter postfix Expression: xy-z\*

Expression Tree

Recursive Traversals:

1. Inorder Traversal

2. Preorder Traversal

3. Postorder Traversal

Non-Recursive Traversals:

4. Inorder Traversal

5. Preorder Traversal

6. Postorder Traversal

7. Exit the program

Enter your choice:- 1

Inorder traversal with recursion: x - y \* z

Expression Tree

Recursive Traversals:

1. Inorder Traversal

2. Preorder Traversal

3. Postorder Traversal

Non-Recursive Traversals:

4. Inorder Traversal

5. Preorder Traversal

6. Postorder Traversal

7. Exit the program

Enter your choice:- 2

Preorder traversal with recursion: \* - x y z

Expression Tree

Recursive Traversals:

1. Inorder Traversal

2. Preorder Traversal

3. Postorder Traversal

Non-Recursive Traversals:

4. Inorder Traversal

5. Preorder Traversal

6. Postorder Traversal

7. Exit the program

Enter your choice:- 3

Postorder traversal with recursion: x y - z \*

Expression Tree

Recursive Traversals:

1. Inorder Traversal

2. Preorder Traversal

3. Postorder Traversal

Non-Recursive Traversals:

4. Inorder Traversal

5. Preorder Traversal

6. Postorder Traversal

7. Exit the program

Enter your choice:- 4

Inorder traversal without recursion: x - y \* z

Expression Tree

Recursive Traversals:

1. Inorder Traversal

2. Preorder Traversal

3. Postorder Traversal

Non-Recursive Traversals:

4. Inorder Traversal

5. Preorder Traversal

6. Postorder Traversal

7. Exit the program

Enter your choice:- 5

Preorder traversal without recursion: \* - x y z

Expression Tree

Recursive Traversals:

1. Inorder Traversal

2. Preorder Traversal

3. Postorder Traversal

Non-Recursive Traversals:

4. Inorder Traversal

5. Preorder Traversal

6. Postorder Traversal

7. Exit the program

Enter your choice:- 6

Postorder traversal without recursion: x y - z \*

Expression Tree

Recursive Traversals:

1. Inorder Traversal

2. Preorder Traversal

3. Postorder Traversal

Non-Recursive Traversals:

4. Inorder Traversal

5. Preorder Traversal

6. Postorder Traversal

7. Exit the program

Enter your choice:- 7