Program:

#include <iostream>

using namespace std;

typedef struct node //structure defined for node

{

char data;

struct node \*left;

struct node \*right;

} node;

typedef struct stacknode //structure defined for stack

{

node \*data;

struct stacknode \*next;

} stacknode;

class stack

{

stacknode \*top; //top node is introduced

public:

stack()

{

top = NULL;

}

node \*topp() //it will return the top element

{

return (top->data);

}54

int isempty() //check if stack is empty(1)

{

if (top == NULL)

return 1;

return 0;

}

void push(node \*a) //push function

{

stacknode \*p;

p = new stacknode();

p->data = a;

p->next = top;

top = p;

}

node \*pop() //pop function

{

stacknode \*p;

node \*x;

x = top->data;

p = top;

top = top->next;

return x;

}

};

node \*create\_pre(char prefix[10]);

node \*create\_post(char postfix[10]);

void inorder(node \*p);

void preorder(node \*p);

void postorder(node \*p);

void inorder\_non\_recursive(node \*t);

void preorder\_non\_recursive(node \*t);

void postorder\_non\_recursive(node \*t);

node \*create\_post(char postfix[10])

{

node \*p;

stack s;

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

{

char token = postfix[i]; //token is the element in postfix55

if (isalnum(token)) //check if token is alphanumeric (operand)

{

p = new node(); //node creation

p->data = token;

p->left = NULL;

p->right = NULL;

s.push(p);

}

else //operator

{

p = new node();

p->data = token;

p->right = s.pop();

p->left = s.pop();

s.push(p);

}

}

return s.pop();

}

node \*create\_pre(char prefix[10])

{

node \*p;

stack s;

int i;

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

{

}

i = i - 1;

for (; i >= 0; i--)

{

char token = prefix[i]; // prefix element

if (isalnum(token)) // operand

{ //node creation

p = new node();

p->data = token;

p->left = NULL;

p->right = NULL;

s.push(p);

}

else //operator56

{

p = new node();

p->data = token;

p->left = s.pop();

p->right = s.pop();

s.push(p);

}

}

return s.pop();

}

void inorder(node \*p) //inorder traversal using recursion

{

if (p == NULL)

{

return;

}

inorder(p->left);

cout << p->data;

inorder(p->right);

}

void preorder(node \*p) //preorder traversal using recursion

{

if (p == NULL)

{

return;

}

cout << p->data;

preorder(p->left);

preorder(p->right);

}

void postorder(node \*p) //postorder traversal using recursion

{

if (p == NULL)

{

return;

}

postorder(p->left);

postorder(p->right);

cout << p->data;

}57

int main()

{

node \*r = NULL, \*r1;

char postfix[10], prefix[10];

int x;

int ch, choice;

do

{

cout << "\n\t\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\n\n1.Construct tree from postfix

Expression/prefix Expression.\n2.Inorder traversal.\n3.Preorder traversal.\n4.Postorder

Traversal.\n5.Exit\n\nEnter your choice: ";

cin >> ch;

switch (ch)

{

case 1:

cout << "\nENTER CHOICE:\n\t1.Postfix expression\n\t2.Prefix

expression\nChoice= ";

cin >> choice;

if (choice == 1)

{

cout << "\nEnter postfix expression= ";

cin >> postfix;

r = create\_post(postfix);

}

else

{

cout << "\nEnter prefix expression= ";

cin >> prefix;

r = create\_pre(prefix);

}

cout << "\n\*\* Tree created successfully \*\* \n";

break;

case 2:

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\nInorder Traversal of tree\n\n";

cout << "With recursion:\t";

inorder(r);

cout << "\n\nWithout recursion: ";

inorder\_non\_recursive(r);

cout << "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;58

case 3:

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\nPreorder Traversal of tree\n\n";

cout << "With recursion:\t";

preorder(r);

cout << "\n\nWithout recursion: ";

preorder\_non\_recursive(r);

cout << "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

case 4:

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\nPostorder Traversal of tree\n\n";

cout << "With recursion:\t";

postorder(r);

cout << "\n\nWithout recursion: ";

postorder\_non\_recursive(r);

cout << "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

}

} while (ch != 5);

return 0;

}

void inorder\_non\_recursive(node \*t)

{

stack s;

while (t != NULL)

{ //data pushed in stack and moved to left till null(last)

s.push(t);

t = t->left;

}

while (s.isempty() != 1)

{

t = s.pop(); // topmost data of stack is printed and then moved to the right

cout << t->data;

t = t->right;

while (t != NULL)

{ //if child is represent push it to the stack

s.push(t);

t = t->left;

}59

}

}

void preorder\_non\_recursive(node \*t)

{

stack s; //stack

while (t != NULL)

{ //it will start from the root and then move to left

cout << t->data;

s.push(t);

t = t->left;

} //once left side is traversed we will pop and move to right

while (s.isempty() != 1)

{

t = s.pop();

t = t->right;

while (t != NULL)

{ //if child is represent we will push in stack

cout << t->data;

s.push(t);

t = t->left;

}

}

}

void postorder\_non\_recursive(node \*t)

{

stack s, s1; //two stack maintained

node \*t1; //root

while (t != NULL)

{

s.push(t);

s1.push(NULL);

t = t->left;

}

while (s.isempty() != 1)

{

t = s.pop();

t1 = s1.pop();

if (t1 == NULL)

{60

s.push(t);

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

t = t->right;

while (t != NULL)

{

s.push(t);

s1.push(NULL);

t = t->left;

}

}

else

cout << t->data;

}

}

**OUTPUT:**

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1.Construct tree from postfix Expression/prefix Expression.

2.Inorder traversal.

3.Preorder traversal.

4.Postorder Traversal.

5.Exit

Enter your choice: 1

ENTER CHOICE:

1.Postfix expression

2.Prefix expression

Choice= 2

Enter prefix expression= \*+23+45

\*\* Tree created successfully \*\*

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*61

1.Construct tree from postfix Expression/prefix Expression.

2.Inorder traversal.

3.Preorder traversal.

4.Postorder Traversal.

5.Exit

Enter your choice: 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Inorder Traversal of tree

With recursion: 2+3\*4+5

Without recursion: 2+3\*4+5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1.Construct tree from postfix Expression/prefix Expression.

2.Inorder traversal.

3.Preorder traversal.

4.Postorder Traversal.

5.Exit

Enter your choice: 3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Preorder Traversal of tree

With recursion: \*+23+45

Without recursion: \*+23+45

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1.Construct tree from postfix Expression/prefix Expression.

2.Inorder traversal.62

3.Preorder traversal.

4.Postorder Traversal.

5.Exit

Enter your choice: 4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Postorder Traversal of tree

With recursion: 23+45+\*

Without recursion: 23+45+\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1.Construct tree from postfix Expression/prefix Expression.

2.Inorder traversal.

3.Preorder traversal.

4.Postorder Traversal.

5.Exit

Enter your choice: 5

PS D:\C++ GB\Assignments>