**Practical No.4**

**Expression Tree**

**Program Code:-**

#include<iostream>

using namespace std;

typedef struct node

{

char data;

struct node \*left;

struct node \*right;

}node;

typedef struct stacknode

{

node\* data;

struct stacknode \*next;

}stacknode;

class stack

{

stacknode \*top;

public:

stack()

{

top=NULL;

}

node\* topp()

{

return (top->data);

}

int isempty()

{

if(top==NULL)

return 1;

return 0;

}

void push(node\* a)

{

stacknode \*p;

p=new stacknode();

p->data=a;

p->next=top;

top=p;

}

node\* pop()

{ 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\_non\_recursive(node \*t);

void inorder(node \*p);

void preorder(node \*p);

void postorder(node \*p);

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

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

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

{node \*p;

stack s;

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

{

char token=postfix[i];

if(isalnum(token))

{

p=new node();

p->data=token;

p->left=NULL;

p->right=NULL;

s.push(p);

}

else

{

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];

if(isalnum(token))

{

p=new node();

p->data=token;

p->left=NULL;

p->right=NULL;

s.push(p);

}

else

{

p=new node();

p->data=token;

p->left=s.pop();

p->right=s.pop();

s.push(p);

}

}

return s.pop();

}

int main()

{

node \*r=NULL,\*r1;

char postfix[10],prefix[10];

int x;

int ch,choice;

do

{

cout<<"\n\t\*\*\*\*TREE OPERATIONS\*\*\*\*\n1.Construct tree from postfix expression/ prefix expression\n2.Inorder traversal\n3.Preorder traversal\n4.Postorder traversal\n5.Exit\nEnter your choice=";

cin>>ch;

switch(ch)

{

case 1:cout<<"ENTER CHOICE:\n1.Postfix expression\n2.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\nTree created successfully";

break;

case 2:cout<<"\nInorder Traversal of tree:\n";

inorder(r);

cout<<"\n Without recursion:\t";

inorder\_non\_recursive(r);

break;

case 3:cout<<"\nPreorder Traversal of tree:\n";

preorder(r);

cout<<"\npreorder traversal without recursion:\t";

preorder\_non\_recursive(r);

break;

case 4:cout<<"\nPostorder Traversal of tree:\n";

postorder(r);

cout<<"\npostorder traversal without recursion";

postorder\_non\_recursion(r);

break;

}

}while(ch!=5);

return 0;

}

void inorder(node \*p)

{

if(p!=NULL)

{

inorder(p->left);

cout<<p->data;

inorder(p->right);

}

}

void preorder(node \*p)

{

if(p!=NULL)

{

cout<<p->data;

preorder(p->left);

preorder(p->right);

}

}

void postorder(node \*p)

{

if(p!=NULL)

{

postorder(p->left);

postorder(p->right);

cout<<p->data;

}

}

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

{

stack s;

while(t!=NULL)

{

s.push(t);

t=t->left;

}

while(s.isempty()!=1)

{

t=s.pop();

cout<<t->data;

t=t->right;

while(t!=NULL)

{

s.push(t);

t=t->left;

}

}

}

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

{

stack s;

while(t!=NULL)

{

cout<<t->data;

s.push(t);

t=t->left;

}

while(s.isempty()!=1)

{

t=s.pop();

t=t->right;

while(t!=NULL)

{

cout<<t->data;

s.push(t);

t=t->left;

}

}

}

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

{stack s,s1;

node \*t1;

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)

{

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;

}

}