/\*

Construct a expression tree from postfix expression

===================================================

Functions:

1.Create

2.Inorder Traversal (Recursive)

3.Preorder Traversal (Recursive)

4.Postorder Traversal (Recursive)

5.Inorder Traversal(N.Recursive)

6.Preorder Traversal(N.Recursive)

7.Postorder Traversal(N.Recursive)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*/

#include<iostream>

using namespace std;

#include<stdlib.h>

#define MAX 30

//Check the Character is Operand or Not

int isoperand(char ch)

{

if((ch>='A' && ch<='Z')||(ch>='a' && ch<='z')||(ch>='0' && ch<='9'))

{

return 1;

}

else

{

return 0;

}

}

//Check the Character is Operator or Not

int isoperator(char ch)

{

if(ch=='$'||ch=='^'||ch=='+'||ch=='-'||ch=='\*'||ch=='/')

return 1;

else

return 0;

}

//Declare a self Referential Structure for Tree Nodes

struct Treenode

{

Treenode \*lchild;

char data;

Treenode \*rchild;

};

//Declare Class

class ET

{

Treenode \*root;

public:

ET();//Constructor

void create(char postfix[MAX]);

void inorder();//Wrapper Function as accessing private member root is not accessible in the main function

void inorder(Treenode \*);

void preorder();//Wrapper Function

void preorder(Treenode \*);

void postorder();//Wrapper Function

void postorder(Treenode \*);

void inorder\_nrc();

void preorder\_nrc();

void postorder\_nrc();

};

//Initialize root to NULL

ET::ET()

{

root=NULL;

}

//For Building Expression Tree from prefix Expression

void ET::create(char prefix[MAX])

{

//Declare Stack

Treenode \*stack[MAX];

int top=-1;

int i,len,val;

char ch;

Treenode \*temp;

//Calculate length of the prefix expression

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

len=i-1;

//Scan the prefix expression from right to left

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

{

ch=prefix[i];

//Create and Initialize a new node in the memory

temp=new Treenode;

temp->lchild=NULL;

temp->data=ch;

temp->rchild=NULL;

//If the character is operand push the address on stack

if(isoperand(ch))

{

stack[++top]=temp;

}

//If the character is operator pop two addresses from stack and built a sub tree,

//Push the parents address back on the stack

else if(isoperator(ch))

{

temp->lchild=stack[top--];

temp->rchild=stack[top--];

stack[++top]=temp;

}

else

{

cout<<"\nWrong expression tree";

cout<<"\nNode cannot be created";

exit(0);

}

}

//Once scanning the expression from right to left is over pop the address from stack and store it as root address

root=stack[top--];

}

void ET::inorder()

{

if(root)

inorder(root);

else

cout<<"\nEmpty expression tree";

}

void ET::inorder(Treenode \*root)

{

//LDR

if(root)

{

//Traverse Left

inorder(root->lchild);

//Print Data

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

//Traverse Right

inorder(root->rchild);

}

}

void ET::preorder()

{

if(root)

preorder(root);

else

cout<<"\nEmpty expression tree";

}

void ET::preorder(Treenode \*root)

{

//DLR

if(root)

{

//Print Data

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

//Traverse Left

preorder(root->lchild);

//Traverse Right

preorder(root->rchild);

}

}

void ET::postorder()

{

if(root)

postorder(root);

else

cout<<"\nEmpty expression tree";

}

void ET::postorder(Treenode \*root)

{

//DLR

if(root)

{

//Traverse Left

postorder(root->lchild);

//Traverse Right

postorder(root->rchild);

//Print Data

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

}

}

void ET::inorder\_nrc()

{

Treenode \*curr=root;

Treenode \*stack[MAX];

int top=-1;

while(1)

{

while(curr!=NULL)

{

stack[++top]=curr;

curr=curr->lchild;//Traverse Left

}

if(top!=-1)

{

curr=stack[top--];

cout<<curr->data<<" "; //Print Data

curr=curr->rchild; //Traverse Right

}

else

break;

}

}

void ET::preorder\_nrc()

{

Treenode \*curr=root;

Treenode \*stack[MAX];

int top=-1;

while(1)

{

while(curr!=NULL)

{

cout<<curr->data<<" "; //Print Data

stack[++top]=curr;

curr=curr->lchild; //Traverse Left

}

if(top!=-1)

{

curr=stack[top--];

curr=curr->rchild; //Traverse Right

}

else

break;

}

}

void ET::postorder\_nrc()

{

Treenode \*curr=root;

Treenode \*stack[MAX];

int top=-1,flag[MAX],f;

while(1)

{

if(curr!=NULL)

{

stack[++top]=curr;

flag[top]=0;

curr=curr->lchild; //Traverse Left

}

else

{

if(top!=-1)

{

f=flag[top];

curr=stack[top--];

if(f==0)

{

stack[++top]=curr;

flag[top]=1;

curr=curr->rchild; //Traverse Right

}

else if (f==1)

{

cout<<curr->data<<" "; //Print Data

curr=NULL;

}

}

else

break;

}

}

}

int main()

{

int ch;

char prefix[MAX];

ET e;

cout<<"\nEnter a prefix expression";

cin>>prefix;

while(1)

{

cout<<"\n\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*";

cout<<"\n1.Create a expression tree\n2.Inorder Traversal (Recursive)\n3.Preorder Traversal (Recursive)";

cout<<"\n4.Postorder (Recursive)\n5.Inorder Traversal(Non Recursive)\n6.Preorder Traversal(Non Recursive)";

cout<<"\n7.Post order Traversal(Non Recursive)\n8.Exit";

cout<<"\nEnter your choice";

cin>>ch;

switch(ch)

{

case 1: e.create(prefix);

cout<<"\nExpression Tree Created from Prefix Expression\n";

break;

case 2: e.inorder(); //Call the wrapper function

break;

case 3: e.preorder(); //Call the wrapper function

break;

case 4: e.postorder(); //Call the wrapper function

break;

case 5: e.inorder\_nrc();

break;

case 6: e.preorder\_nrc();

break;

case 7: e.postorder\_nrc();

break;

case 8:exit(0);

}

}

}