#include <iostream>

#include <string.h>

using namespace std;

#define max 30

class node

{

public:

char data;

node \*left, \*right;

node()

{

left=right=NULL;

}

};

class expressiontree

{

public:

node \*root;

expressiontree()

{

root=NULL;

}

void create(string str);

void nonrec\_postorder(node \*temp);

int priority(char ch);

};

class stack

{

int top;

node \*stk[max];

public:

stack()

{

top=-1;

}

int empty()

{

if(top==-1)

return 1;

else

return 0;

}

void push(node \*temp)

{

stk[++top]= temp;

}

node \*pop()

{

return(stk[top--]);

}

node \*Top()

{

return(stk[top]);

}

};

int expressiontree :: priority(char ch)

{

switch(ch)

{

case '+':

case '-':

return 0;

case '\*':

case '/':

return 1;

case '^':

return 2;

}

}

void expressiontree :: create(string str)

{

stack operand\_st, operator\_st;

int i=0;

char ch;

while(str[i]!= '\0')

{

ch = str[i];

node \*temp = new node();

temp->data = ch;

if(isalpha(ch))

{

operand\_st.push(temp);

}

else

{

if(operator\_st.empty())

operator\_st.push(temp);

else

if(priority(ch)>priority(operator\_st.Top()->data))

operator\_st.push(temp);

else

{

while(!operator\_st.empty() && priority(ch)<= operator\_st.Top()->data)

{

node \*operat = operator\_st.pop();

operat->right= operand\_st.pop();

operat->left = operand\_st.pop();

operand\_st.push(operat);

}

operator\_st.push(temp);

}

}

i++;

}

while(!operator\_st.empty())

{

node \*operat = operator\_st.pop();

operat->right= operand\_st.pop();

operat->left = operand\_st.pop();

operand\_st.push(operat);

}

root= operand\_st.pop();

}

void expressiontree :: nonrec\_postorder(node \*temp)

{

if(root==NULL)

cout<<"\n Empty";

else

{

stack s1, s2;

s1.push(temp);

while(!s1.empty())

{

node \*temp= s1.pop();

s2.push(temp);

if(temp->left)

s1.push(temp->left);

if(temp->right)

s1.push(temp->right);

}

while(!s2.empty())

{

node \*temp= s2.pop();

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

}

}

}

int main()

{

expressiontree e;

string str;

int ch;

do

{

cout<<"\n Enter Choice: ";

cout<<"\n\t1.Create expression tree \n\t2.Non recursive Postorder Traversal \n\t3.Exit";

cout<<"\n Choice: ";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter Expression : ";

cin >> str;

e.create(str);

break;

case 2:cout<<"\n Non Recursive Postorder Traversal: ";

e.nonrec\_postorder(e.root);

break;

case 3: break;

}

}while(ch != 3);

return 0;

}