#include<bits/stdc++.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include<iostream>

#define COUNT 10

using namespace std;

struct et

{

char value;

et\* left, \*right;

};

struct Stack

{

int top;

unsigned capacity;

int\* array;

};

Stack\* createStack( unsigned capacity )

{

struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));

if (!stack)

return NULL;

stack->top = -1;

stack->capacity = capacity;

stack->array = (int\*) malloc(stack->capacity \* sizeof(int));

if (!stack->array)

return NULL;

return stack;

}

int isEmpty(struct Stack\* stack)

{

return stack->top == -1 ;

}

char peek(struct Stack\* stack)

{

return stack->array[stack->top];

}

char pop(struct Stack\* stack)

{

if (!isEmpty(stack))

return stack->array[stack->top--] ;

return '$';

}

void push(struct Stack\* stack, char op)

{

stack->array[++stack->top] = op;

}

int isOperand(char ch)

{

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int Prec(char ch)

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

int infixToPostfix(char\* exp)

{

int i, k;

struct Stack\* stack = createStack(strlen(exp));

if(!stack)

return -1 ;

for (i = 0, k = -1; exp[i]; ++i)

{

if (isOperand(exp[i]))

exp[++k] = exp[i];

else if (exp[i] == '(')

push(stack, exp[i]);

else if (exp[i] == ')')

{

while (!isEmpty(stack) && peek(stack) != '(')

exp[++k] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(')

return -1;

else

pop(stack);

}

else

{

while (!isEmpty(stack) && Prec(exp[i]) <= Prec(peek(stack)))

exp[++k] = pop(stack);

push(stack, exp[i]);

}

}

while (!isEmpty(stack))

exp[++k] = pop(stack );

exp[++k] = '\0';

}

bool isOperator(char c)

{

if (c == '+' || c == '-' ||

c == '\*' || c == '/' ||

c == '^')

return true;

return false;

}

et\* newNode(int v)

{

et \*temp = new et;

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

temp->value = v;

return temp;

};

et\* constructTree(char postfix[])

{

stack<et \*> st;

et \*t, \*t1, \*t2;

for (int i=0; i<strlen(postfix); i++)

{

if (!isOperator(postfix[i]))

{

t = newNode(postfix[i]);

st.push(t);

}

else

{

t = newNode(postfix[i]);

t1 = st.top();

st.pop();

t2 = st.top();

st.pop();

t->right = t1;

t->left = t2;

st.push(t);

}

}

t = st.top();

st.pop();

return t;

}

void print2DUtil(et \*root, int space)

{

if (root == NULL)

return;

space += COUNT;

print2DUtil(root->right, space);

cout<<endl;

for (int i = COUNT; i < space; i++)

cout<<" ";

cout<<"( "<<root->value<<" )\n";

print2DUtil(root->left, space);

}

int main()

{

char postfix[100];

cout<<"Enter the infix expression to create the tree: ";

cin>>postfix;

infixToPostfix(postfix);

et\* r = constructTree(postfix);

print2DUtil(r,0);

return 0;

}