1. Write a program to convert an Infix expression into a Postfix expression.

#include<iostream>

#include<cstring>

using namespace std;

#define MAX 100

class Stack {

private:

char arr[MAX];

int top;

public:

Stack() {

top = -1;

}

bool isEmpty() {

return top == -1;

}

bool isFull() {

return top == MAX - 1;

}

void push(char x) {

if (isFull()) {

cout << "Stack Overflow!" << endl;

return;

}

arr[++top] = x;

}

char pop() {

if (isEmpty()) {

cout << "Stack Underflow!" << endl;

return '\0';

}

return arr[top--];

}

char peek() {

if (isEmpty()) {

return '\0';

}

return arr[top];

}

};

int precedence(char op) {

switch (op) {

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

return 3;

default:

return 0;

}

}

bool isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '%' || c == '^');

}

bool isOperand(char c) {

return ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9'));

}

string infixToPostfix(string infix) {

Stack stack;

string postfix = "";

for (int i = 0; i < infix.length(); i++) {

char c = infix[i];

if (isOperand(c)) {

postfix += c;

}

else if (c == '(') {

stack.push(c);

}

else if (c == ')') {

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

postfix += stack.pop();

}

if (!stack.isEmpty()) {

stack.pop();

}

}

else if (isOperator(c)) {

while (!stack.isEmpty() &&

precedence(stack.peek()) >= precedence(c) &&

stack.peek() != '(') {

postfix += stack.pop();

}

stack.push(c);

}

else if (c == ' ') {

continue;

}

}

while (!stack.isEmpty()) {

postfix += stack.pop();

}

return postfix;

}

int main() {

string infix;

cout << "Enter infix expression: ";

getline(cin, infix);

string postfix = infixToPostfix(infix);

cout << "Infix expression: " << infix << endl;

cout << "Postfix expression: " << postfix << endl;

return 0;

}

OUTPUT :

