**Question#1,2,3**

**Code:**

#include<iostream>

#include<string>

using namespace std;

int precedence(char op) {

if (op == '+' || op == '-') {

return 1;

}

else if (op == '\*' || op == '/') {

return 2;

}

else if (op == '^') {

return 3;

}

return 0;

}

struct node {

char data;

node\* prev;

node(int val) {

this->data = val;

this->prev = nullptr;

}

};

class charstack {

private:

node\* top;

public:

charstack() {

top = nullptr;

}

void push(char val) {

node\* temp = new node(val);

if (top == nullptr) {

top = temp;

}

else {

temp->prev = top;

top = temp;

}

}

void pop(char& val) {

if (top == nullptr) {

cout << "Stack is Empty" << endl;

}

else {

node\* temp = top;

top = top->prev;

val = temp->data;

delete temp;

}

}

bool isEmpty() {

return top == nullptr;

}

char Top() {

return top->data;

}

string infix\_to\_postfix(string str) {

charstack obj;

string pos = "";

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

char char1 = str[i];

if (char1 >= 'A' && char1 <= 'Z' || char1 >= 'a' && char1 <= 'z') {

pos.push\_back(char1);

}

if (char1 == '(' || char1 == '{' || char1 == '[') {

obj.push(char1);

}

else if (char1 == ')' || char1 == '}' || char1 == ']') {

char operaterpop;

while (!obj.isEmpty() && obj.Top() != '(' && obj.Top() != '{' && obj.Top() != '[') {

obj.pop(operaterpop);

pos.push\_back(operaterpop);

}

obj.pop(operaterpop);

}

else if (char1 == '+' || char1 == '\*' || char1 == '-' || char1 == '/') {

if (char1 == '+' && str[i + 1] == '+' || char1 == '-' && str[i + 1] == '-') {

pos.push\_back(char1);

pos.push\_back(char1);

i++;

continue;

}

char operaterpop;

if (!obj.isEmpty() && precedence(obj.Top()) >= precedence(char1)) {

obj.pop(operaterpop);

pos.push\_back(operaterpop);

}

obj.push(char1);

}

}

while (!obj.isEmpty()) {

char operatorpop;

obj.pop(operatorpop);

pos.push\_back(operatorpop);

}

return pos;

}

void print() {

node\* temp = top;

while (temp != nullptr) {

cout << "[" << temp->data << "]" << endl;

temp = temp->prev;

}

cout << endl;

}

};

void replaceBrackets(string& str) {

int n = str.length();

for (int i = 0; i < n; i++) {

if (str[i] == '[')

str[i] = ']';

else if (str[i] == '{')

str[i] = '}';

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

str[i] = ')';

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

str[i] = '[';

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

str[i] = '(';

else if (str[i] == '}')

str[i] = '{';

}

}

string reverseString(string& str) {

int n = str.length();

string reversed = str;

for (int i = 0; i < n / 2; i++) {

swap(reversed[i], reversed[n - i - 1]);

}

replaceBrackets(reversed);

return reversed;

}

int main() {

charstack obj;

string str="((A + B) \* C - D) / (E \* F - G)";

cout << "Original String : " << str << endl;

string str1 = obj.infix\_to\_postfix(str);

cout << "Postfix String : " << str1 << endl;

string str2 = reverseString(str);

cout << "Reversed String : " << str2 << endl;

string str3 = obj.infix\_to\_postfix(str2);

string str4 = reverseString(str3);

cout << "Infix String : " << str4 << endl;

str = "((A + B++) \* C - D) / (E \* F - G)";

cout << "Original String : " << str << endl;

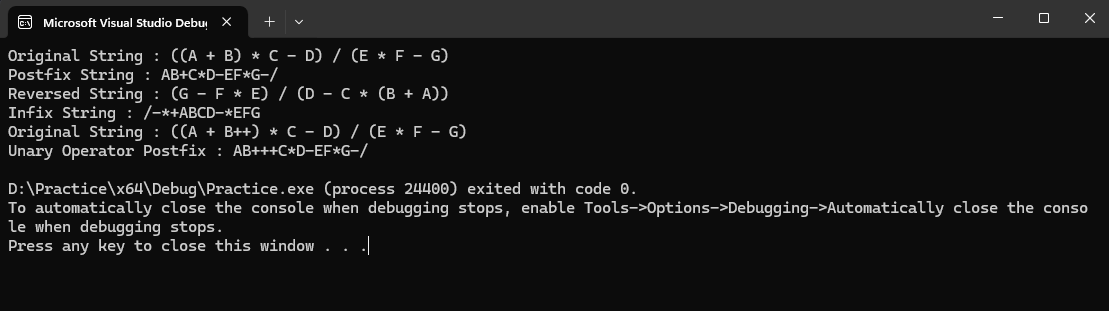
string str5 = obj.infix\_to\_postfix(str);

cout << "Unary Operator Postfix : " << str5 << endl;

return 0;

}

**Output:**

****