Task-1

#include <iostream>

#include<string>

using namespace std;

//Node for linked list

struct Node {

int data;

Node\* next;

};

//class stack is created for ADT

class stack{

Node\* top;

public:

//Push method

void push(int data)

{

Node\* temp = new Node();

if (!temp) {

cout << "\nStack Overflow";

exit(1);

}

temp->data = data;

temp->next = top;

top = temp;

}

//Method to check empty

int isEmpty()

{

if (top == NULL)

return false;

else {

return true;

}

}

//method to get top value

int topp() {

if (isEmpty())

return top->data;

else

exit(1);

}

//method to pop value from top and delete that

void pop() {

Node\* temp;

if (top == NULL) {

cout << "\nStack Underflow" << endl;

exit(1);

}

else {

temp = top;

top = top->next;

delete temp;

temp = NULL;

}

}

//method to check precedenc

int precedence(char oper) {

if (oper == '^')

return 3;

else if (oper == '/' || oper == '\*')

return 2;

else if (oper == '+' || oper == '-')

return 1;

else

return -1;

}

};

//Function to convert

void infixToPostfix(string s)

{

stack st;

string result;//for storing result

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

char c = s[i];

//Checking the operand and storing in result

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

result += c;

//if operator then push to stack

else if (c == '(')

st.push('(');

else if (c == ')') {

while (st.topp() != '(') {

result += st.topp();

st.pop();

}

st.pop();

}

else {

while (!st.isEmpty() && st.precedence(s[i]) <= st.precedence(st.topp())) {

result += st.topp();

st.pop();

}

st.push(c);

}

}

while (!st.isEmpty()) {

result += st.topp();

st.pop();

}

cout << result << endl;

}

int main() {

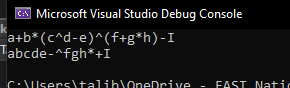
string str;

str = "a+b\*(c^d-e)^(f+g\*h)-I";

cout << str << endl;

infixToPostfix(str);

}



Task-2

#include <iostream>

#include<string>

using namespace std;

//Node for linked list

struct Node {

string data;

Node\* next;

};

//class stack is created for ADT

class stack {

Node\* top;

public:

//Push method

void push(string data)

{

Node\* temp = new Node();

if (!temp) {

cout << "\nStack Overflow";

exit(1);

}

temp->data = data;

temp->next = top;

top = temp;

}

//Method to check empty

int isEmpty()

{

if (top == NULL)

return false;

else {

return true;

}

}

//method to get top value

string topp() {

if (isEmpty())

return top->data;

else

exit(1);

}

//method to pop value from top and delete that

void pop() {

Node\* temp;

if (top == NULL) {

cout << "\nStack Underflow" << endl;

exit(1);

}

else {

temp = top;

top = top->next;

delete temp;

temp = NULL;

}

}

};

bool operand(char x)

{

return (x >= 'a' && x <= 'z') ||

(x >= 'A' && x <= 'Z');

}

string PostfixToInfix(string exp)

{

stack st;

//Pushing operands until no operator comes

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

{

if (operand(exp[i]))

{

string op(1, exp[i]);

st.push(op);

}

else

{

string op1 = st.topp();

st.pop();

string op2 = st.topp();

st.pop();

st.push("(" + op2 + exp[i] +

op1 + ")");

}

}

//Returnong top value because it has all the infix fully notation

return st.topp();

}

int main() {

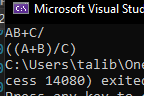
string str;

str = "AB+C/";

cout << str << endl;

cout<<PostfixToInfix(str);

}



Task-3

#include <iostream>

#include<string>

using namespace std;

//Node for linked list

struct Node {

string data;

Node\* next;

};

//class stack is created for ADT

class stack {

Node\* top;

public:

//Push method

void push(string data)

{

Node\* temp = new Node();

if (!temp) {

cout << "\nStack Overflow";

exit(1);

}

temp->data = data;

temp->next = top;

top = temp;

}

//Method to check empty

int isEmpty()

{

if (top == NULL)

return false;

else {

return true;

}

}

//method to get top value

string topp() {

if (isEmpty())

return top->data;

else

exit(1);

}

//method to pop value from top and delete that

void pop() {

Node\* temp;

if (top == NULL) {

cout << "\nStack Underflow" << endl;

exit(1);

}

else {

temp = top;

top = top->next;

delete temp;

temp = NULL;

}

}

};

bool isOperator(char x) {

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

return true;

}

else

return false;

}

string PreToPost(string pre) {

stack st;

//Pushing expression from right side

for (int i = pre.size() - 1; i >= 0; i--) {

if (isOperator(pre[i])) {

string op1 = st.topp();

st.pop();

string op2 = st.topp();

st.pop();

string temp = op1 + op2 + pre[i];

st.push(temp);

}

else

st.push(string(1, pre[i]));

}

//Returning top because it has final expression

return st.topp();

}

int main() {

string str;

str = "+-ABC";

cout << str << endl;

cout << PreToPost(str);

}

