**Assignment No: 4**

**Problem Statement:**

Implement stack as an abstract data type using linked list and use this ADT for conversion of infix expression to postfix, prefix, and evaluation of postfix and prefix expression.

**Objectives:**

4. To implement abstract properties of various data structures such as stacks, queues.

**Outcomes:**

Students will be able to:

5. Make use of abstract properties of various data structures such as stacks, queues.

**Code:**

#include<iostream>

using namespace std;

class Node{

public:

char data;

Node \*prev;

};

class Stack{

private:

Node \*top;

public:

Stack(){

top = NULL;

}

void push(char data){

Node \*newNode = new Node;

newNode->data = data;

newNode->prev = top;

top = newNode;

}

char pop(){

if(top == NULL){

cout<<"\nSTACK UNDERFLOW !";

return '\0';

}

Node \*temp = top;

char data = temp->data;

top = top->prev;

delete temp;

return data;

}

char peek(){

return top->data;

}

void print(){

Node \*temp = top;

cout<<"\nTop->";

while(temp != NULL){

cout<<"\t"<<temp->data<<endl;

temp = temp->prev;

}

}

~Stack(){

Node \*temp;

temp = top;

while(temp!=NULL){

delete temp;

temp = temp->prev;

}

delete top;

}

};

string infixToPostfix(string);

string infixToPrefix(string);

string reverseStr(string);

int main(){

int choice;

do{

cout<<"\n#### MENU ####";

cout<<"\n1. Infix to Postfix\n2. Infix to Prefix\n3. Exit";

cout<<"\nChoice: ";

cin>>choice;

switch(choice){

case 1:{

string infix;

cout<<"\nEnter the Infix Expression: ";

cin>>infix;

string postfix = infixToPostfix(infix);

cout<<"\nThe Postfix Expression: "<<postfix;

break;

}

case 2:{

string infix;

cout<<"\nEnter the Infix Expression: ";

cin>>infix;

string prefix = infixToPrefix(infix);

cout<<"\nThe Prefix Expression: "<<prefix;

break;

}

case 3:cout<<"\nShutting Down...";break;

default: "\nInvalid Choice";

}

}while(choice != 3);

return 0;

}

int precedence(char ch){

if(ch == '^'){

return 10;

}

else if(ch == '\*' || ch == '/' || ch == '%'){

return 9;

}

else if(ch == '+' || ch == '-'){

return 8;

}

else if(ch == ')' || ch == '('){

return 7;

}

else{

return -1;

}

}

bool isOperator(char ch){

if(ch == '+'|| ch == '-' || ch == '\*' || ch == '/' || ch == '%' || ch == '^')

return true;

return false;

}

bool isNumber(char ch){

int num = (int)ch;

if(num>=48 && num<=57)

return true;

return false;

}

bool isChar(char ch){

int num = (int)ch;

if((num>=65 && num<=90) || (num>=97 && num<=122))

return true;

return false;

}

bool isLeftPar(char ch){

if(ch == '(')

return true;

return false;

}

bool isRightPar(char ch){

if(ch == ')')

return true;

return false;

}

string reverseStr(string myString){

string rev="";

for(int i=myString.length()-1;i>=0;i--){

rev = rev+myString[i];

}

return rev;

}

string infixToPostfix(string infix){

Stack stk;

string postfix="";

infix = '(' + infix + ')';

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

if(isLeftPar(infix[i]))

stk.push(infix[i]);

else if(isChar(infix[i]) || isNumber(infix[i])){

postfix = postfix + infix[i];

}

else if(isOperator(infix[i])){

while(precedence(stk.peek()) >= precedence(infix[i]) ){

postfix = postfix + stk.pop();

}

stk.push(infix[i]);

}

else if(isRightPar(infix[i])){

while(stk.peek() != '('){

postfix = postfix + stk.pop();

}

stk.pop();

}

else{

cout<<"\nInvalid Infix expression";

return "";

}

}

return postfix;

}

string infixToPrefix(string infix){

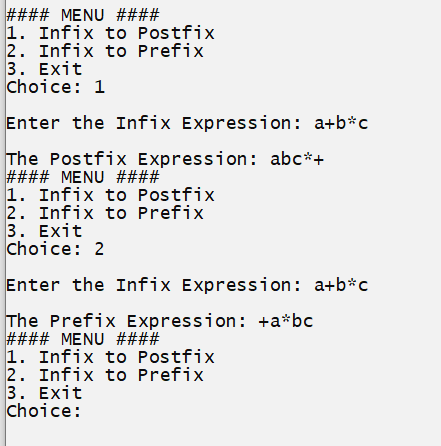
string rev = reverseStr(infix);

rev = infixToPostfix(rev);

return reverseStr(rev);

}

**Output:**

****