Q1

#include<iostream>

using namespace std;

class Stack {

int capacity;

char\* arr;

int top;

public:

Stack(int c){

this->capacity =c;

this->arr = new char[c];

this->top =-1;

}

bool isEmpty(){

return this->top==-1;

}

bool isFull(){

return this->top == this->capacity-1;

}

void push(char data ){

if(isFull()){

cout<<"OverFlow";

return;

}

this->top++;

this->arr[this->top] = data;

}

void pop(){

if(isEmpty()){

cout<<"UnderFlow";

return ;

}

this->top--;

}

char gettop(){

if(isEmpty()){

cout<<"UnderFlow";

return ' ';

}

return this->arr[this->top];

}

int getsize(){

return this->top+1;

}

void print(){

if(isEmpty()){

cout<<"UnderFlow";

return ;

}

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

cout<<arr[i]<<" ";

}

cout<<endl;

}

void ispalindrome(Stack s){

Stack temp(s.capacity);

temp=s;

Stack s2(s.capacity);

bool check;

int size = s.getsize();

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

s2.push(s.gettop());

s.pop();

}

for(int i=0;i<size;i++)

{

if(s2.gettop()==temp.gettop())

{

s2.pop();

temp.pop();

check=true;

}

else

{

check=false;

break;

}

}

if(check)

{

cout<<"This is palindrome\n";

}else{

cout<<"This is Not palindrome";

}

}

};

int main(){

Stack s1(9);

s1.push('B');

s1.push('O');

s1.push('R');

s1.push('R');

s1.push('O');

s1.push('W');

s1.push('R');

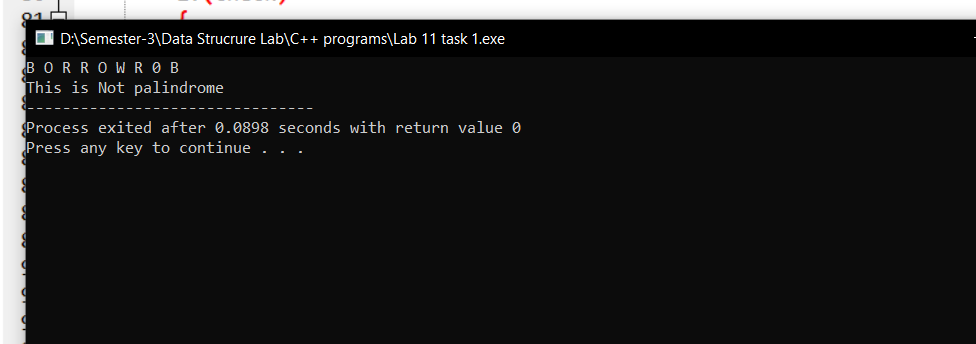
s1.push('0');

s1.push('B');

s1.print();

s1.ispalindrome(s1);

}



Q2

//osmn

#include<iostream>

using namespace std;

const int MAX\_SIZE = 10;

class Stack{

int top;

int front;

int arr[MAX\_SIZE];

public:

Stack(){

top = -1;

front =0;

}

void enqueue(int x) {

if (top >= MAX\_SIZE - 1) {

cout<<"queue is Full\n";

}else{

arr[++top]= x;

cout<<x<< " Enqueue into queue\n";

}

}

void dequeue(){

if (front>top){

cout<<"Queue is empty\n";

}else{

cout<<arr[front++]<<" Dequeued\n";

arr[front-1] =0;

}

}

};

int main(){

Stack s1;

s1.enqueue(10);

s1.enqueue(8);

s1.enqueue(6);

s1.enqueue(1);

s1.enqueue(4);

s1.enqueue(7);

s1.enqueue(13);

// s1.print();

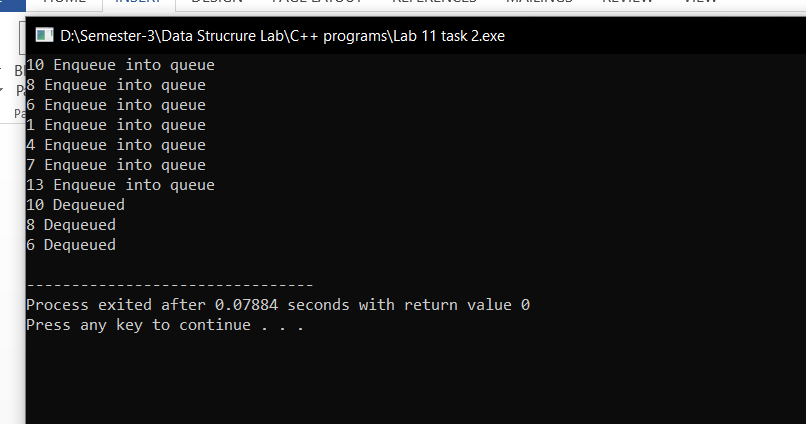
s1.dequeue();

s1.dequeue();

s1.dequeue();

// s1.print();

return 0;

}

Q3

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

class Node {

public:

char data;

Node\* next;

Node(char c) : data(c), next(NULL) {}

};

class LinkedListStack {

private:

Node\* top;

public:

LinkedListStack() {

top = NULL;

}

void push(char data) {

Node\* newNode = new Node(data);

newNode->next = top;

top = newNode;

}

char pop() {

if (is\_empty()) {

cerr << "Stack underflow\n";

return '\0';

}

char data = top->data;

Node\* temp = top;

top = top->next;

delete temp;

return data;

}

bool is\_empty() {

return top == NULL;

}

char peek() {

if (is\_empty()) {

cerr << "Stack is empty\n";

return '\0';

}

return top->data;

}

};

bool isOperator(char c) {

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

return true;

}

return false;

}

int precedence(char c) {

if (c == '^') return 3;

if (c == '\*' || c == '/') return 2;

if (c == '+' || c == '-') return 1;

return -1;

}

string infixToPrefix(LinkedListStack& s, const string& infix) {

string prefix;

string reversedInfix = infix;

reverse(reversedInfix.begin(), reversedInfix.end());

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

char& c = reversedInfix[i];

if (c == '(') {

c = ')';

}

else if (c == ')') {

c = '(';

}

}

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

char c = reversedInfix[i];

if (isalnum(c)) {

prefix += c;

}

else if (c == '(') {

s.push(c);

}

else if (c == ')') {

while (!s.is\_empty() && s.peek() != '(') {

prefix += s.pop();

}

if (!s.is\_empty() && s.peek() == '(') {

s.pop();

}

else {

cerr << "Mismatched parentheses\n";

exit(1);

}

}

else if (isOperator(c)) {

if (s.is\_empty()) {

s.push(c);

}

else {

if (precedence(c) > precedence(s.peek())) {

s.push(c);

}

else if (precedence(c) == precedence(s.peek()) && c == '^') {

while (precedence(c) == precedence(s.peek()) && c == '^') {

prefix += s.pop();

}

s.push(c);

}

else if (precedence(c) == precedence(s.peek())) {

s.push(c);

}

else {

while (!s.is\_empty() && precedence(c) < precedence(s.peek())) {

prefix += s.pop();

}

s.push(c);

}

}

}

else {

cerr << "Invalid character in the expression\n";

exit(1);

}

}

while (!s.is\_empty()) {

prefix += s.pop();

}

reverse(prefix.begin(), prefix.end());

return prefix;

}

int main() {

string infix, prefix;

cout << "Enter an Infix Expression: " << endl;

cin >> infix;

LinkedListStack stack;

cout << "INFIX EXPRESSION: " << infix << endl;

prefix = infixToPrefix(stack, infix);

cout << "PREFIX EXPRESSION: " << prefix << endl;

return 0;

}