**Question#1**

**Code:**

#include<iostream>

using namespace std;

struct node {

char data;

node\* prev;

node() :data(' '), prev(nullptr) {}

};

class base {

protected:

node\* top;

public:

virtual bool isEmpty() = 0;

virtual void push(char val) = 0;

virtual void pop(char& val) = 0;

virtual char Top() = 0;

virtual bool isPalindrome() = 0;

virtual void print() = 0;

};

class Stack : public base {

public:

Stack() {

top = nullptr;

}

~Stack() {

char temp = ' ';

while (!isEmpty()) {

pop(temp);

}

}

bool isEmpty() {

if (top == nullptr) {

return true;

}

return false;

}

void push(char val) {

node\* temp = new node;

temp->data = val;

temp->prev = top;

top = temp;

}

void pop(char& val) {

if (isEmpty() == true) {

cout << "Empty" << endl;

}

else {

node\* temp = top->prev;

val = top->data;

delete top;

top = temp;

}

}

char Top() {

if (isEmpty() == true) {

cout << "Empty" << endl;

return ' ';

}

return (top->data);

}

bool isPalindrome() {

int count = 0;

Stack obj;

char temp = ' ';

while (!isEmpty()) {

pop(temp);

obj.push(temp);

count++;

}

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

obj.pop(temp);

push(temp);

}

if (count % 2 != 0) {

obj.pop(temp);

}

while (!isEmpty()) {

if (Top() != obj.Top()) {

return false;

}

pop(temp);

obj.pop(temp);

}

return true;

}

void print() {

char temp = ' ';

while (!isEmpty()) {

pop(temp);

cout << "[" << temp << "]";

}

cout << endl;

}

};

int main() {

Stack obj;

string str = "AbcDcbA";

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

cout << "Stack : ";

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

obj.push(str[i]);

}

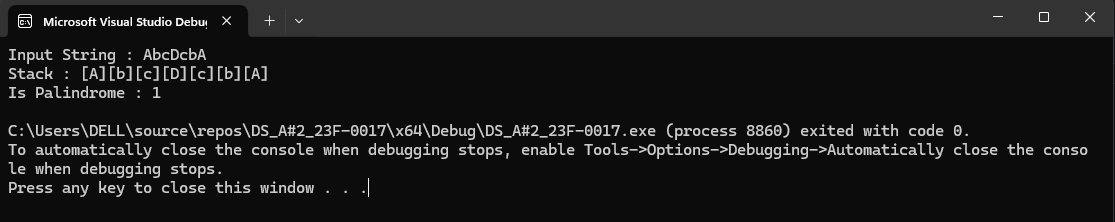
obj.print();

cout << "Is Palindrome : " << obj.isPalindrome() << endl;

return 0;

}

**Output:**

****

**Question#2**

**Code:**

#include<iostream>

#include<string>

using namespace std;

struct node {

char data;

node\* prev;

node() :data(' '), prev(nullptr) {}

};

class base {

protected:

node\* top;

public:

virtual bool isEmpty() = 0;

virtual void push(char val) = 0;

virtual void pop(char& val) = 0;

virtual char Top() = 0;

virtual void print() = 0;

virtual string infixToPostfix(string str) = 0;

virtual string infixToPrefix(string str) = 0;

virtual int evaluate(string str) = 0;

};

class Stack : public base {

public:

Stack() {

top = nullptr;

}

~Stack() {

char temp = ' ';

while (!isEmpty()) {

pop(temp);

}

}

bool isEmpty() {

if (top == nullptr) {

return true;

}

return false;

}

void push(char val) {

node\* temp = new node;

temp->data = val;

temp->prev = top;

top = temp;

}

void pop(char& val) {

if (isEmpty() == true) {

cout << "Empty" << endl;

}

else {

node\* temp = top->prev;

val = top->data;

delete top;

top = temp;

}

}

char Top() {

if (isEmpty() == true) {

cout << "Empty" << endl;

return ' ';

}

return (top->data);

}

void print() {

char temp = ' ';

while (!isEmpty()) {

pop(temp);

cout << "[" << temp << "]";

}

cout << endl;

}

int precedence(char temp) {

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

return 1;

}

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

return 2;

}

else if (temp == '(') {

return 0;

}

return -1;

}

string infixToPostfix(string str) {

string postfixString = "";

Stack opStk;

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

if ((str[i] >= 'A' && str[i] <= 'Z') || (str[i] >= 'a' && str[i] <= 'z') || (str[i] >= '0' && str[i] <= '9')) {

postfixString += str[i];

}

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

opStk.push(str[i]);

}

else if (str[i] == '+' || str[i] == '-' || str[i] == '\*' || str[i] == '/' || str[i] == '%') {

while (!opStk.isEmpty() && precedence(opStk.Top()) >= precedence(str[i])) {

char temp = ' ';

opStk.pop(temp);

postfixString += temp;

}

opStk.push(str[i]);

}

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

char temp = ' ';

opStk.pop(temp);

while (temp != '(') {

postfixString += temp;

opStk.pop(temp);

}

}

}

while (!opStk.isEmpty()) {

char temp = ' ';

opStk.pop(temp);

postfixString += temp;

}

return postfixString;

}

void reverseString(string& str) {

string temp = "";

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

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

str[i] = ')';

}

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

str[i] = '(';

}

temp += str[i];

}

str = temp;

}

string infixToPrefix(string str) {

reverseString(str);

string str1 = infixToPostfix(str);

reverseString(str1);

return str1;

}

int evaluate(string str) {

string postfixString = infixToPostfix(str);

Stack ans;

int op1 = 0;

int op2 = 0;

int result = 0;

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

if (postfixString[i] >= '0' && postfixString[i] <= '9') {

ans.push(postfixString[i] - '0');

}

else if (postfixString[i] == '+' || postfixString[i] == '-' || postfixString[i] == '\*' || postfixString[i] == '/' || postfixString[i] == '%') {

char opChar;

ans.pop(opChar);

op2 = opChar - '0';

ans.pop(opChar);

op1 = opChar - '0';

switch (postfixString[i]) {

case '+':

result = op1 + op2;

break;

case '-':

result = op1 - op2;

break;

case '\*':

result = op1 \* op2;

break;

case '/':

result = op1 / op2;

break;

case '%':

result = op1 % op2;

break;

}

ans.push(result + '0');

}

}

char finalResult;

ans.pop(finalResult);

return finalResult - '0';

}

};

int main() {

Stack obj;

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

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

cout << "Infix to Postfix : " << obj.infixToPostfix(str) << endl;

cout << "Infix to Prefix : " << obj.infixToPrefix(str) << endl;

cout << "Evaluation : " << endl;

str = "((1+2)\*3-4)/(5\*6-7)";

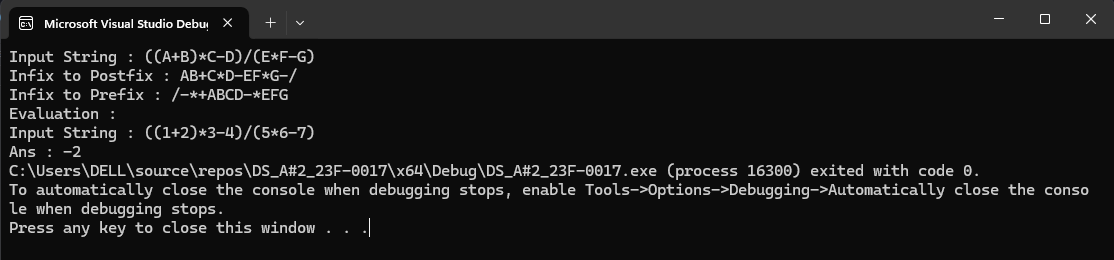
cout << "Input String : " << str << endl;

cout<<"Ans : "<< obj.evaluate(str);

return 0;

}

**Output:**

****

**Question#4**

**Code:**

#include<iostream>

using namespace std;

template<typename Q>

class node {

public:

node\* next;

Q data;

node() :data(0), next(nullptr) {}

};

template<typename T>

class Queue {

node<T>\* front;

node<T>\* rear;

public:

Queue() : front(nullptr), rear(nullptr) {}

void enqueue(T data) {

node<T>\* temp = new node<T>;

temp->data = data;

temp->next = nullptr;

if (isEmpty()) {

front = temp;

rear = temp;

}

else {

rear->next = temp;

rear = temp;

}

}

T dequeue() {

if (isEmpty()) {

cout << "Empty" << endl;

return T();

}

T data;

node<T>\* temp = front;

front = front->next;

data = temp->data;

if (front == nullptr) {

rear = nullptr;

}

delete temp;

return data;

}

bool isEmpty() const {

return front == nullptr;

}

void displayQueue() const {

if (isEmpty()) {

cout << "Empty" << endl;

return;

}

node<T>\* temp = front;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << endl;

}

};

int main() {

Queue<int> Q1;

Queue<int> Q2;

Queue<int> Q3;

for (int i = 1; i <= 10; ++i) {

Q1.enqueue(i);

Q2.enqueue(i + 10);

Q3.enqueue(i + 20);

}

cout << "Initial Q1 : ";

Q1.displayQueue();

cout << endl << "Initial Q2 : ";

Q2.displayQueue();

cout << endl << "Initial Q3 : ";

Q3.displayQueue();

cout << endl << "Dequeuing From Q1 And Performing Chained Operations : " << endl;

for (int i = 0; i < 3; ++i) {

Q1.dequeue();

Q1.enqueue(Q2.dequeue());

Q2.enqueue(Q3.dequeue());

}

cout << "After Dequeuing And Enqueuing:" << endl;

cout << "Q1:" << endl;

Q1.displayQueue();

cout << "Q2:" << endl;

Q2.displayQueue();

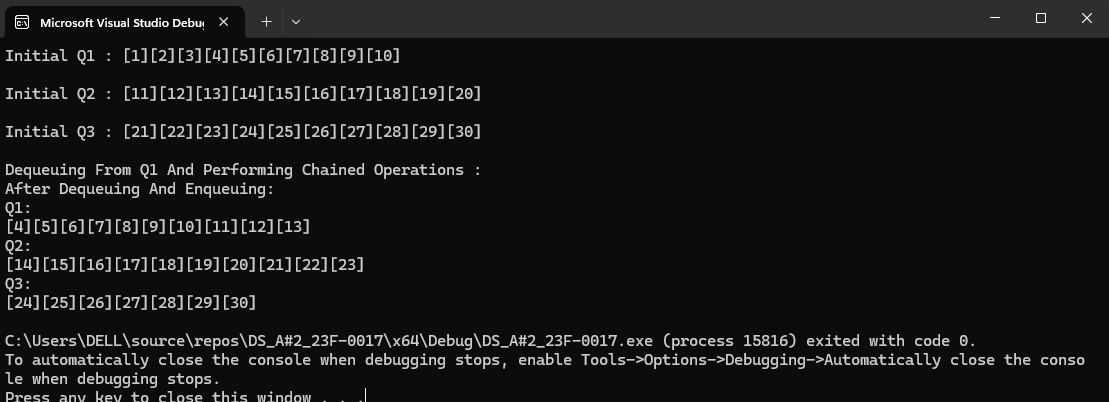
cout << "Q3:" << endl;

Q3.displayQueue();

return 0;

}

**Output:**

****

**Question#5**

**Code:**

#include <iostream>

using namespace std;

class DoubleEndedQueue {

int\* arr;

int maxSize, rear, front;

public:

DoubleEndedQueue(int size = 5) {

maxSize = size;

arr = new int[maxSize];

front = -1;

rear = -1;

}

~DoubleEndedQueue() {

delete[] arr;

}

bool isEmpty() {

return (front == -1);

}

bool isFull() {

return ((front == 0 && rear == maxSize - 1) || (front == rear + 1));

}

void insertFront(int value) {

if (isFull()) {

cout << "Full" << endl;

return;

}

if (front == -1) {

front = rear = 0;

}

else if (front == 0) {

front = maxSize - 1;

}

else {

front--;

}

arr[front] = value;

cout << "Inserted " << value << " at Front" << endl;

}

void insertBack(int value) {

if (isFull()) {

cout << "Full" << endl;

return;

}

if (rear == -1) {

front = rear = 0;

}

else if (rear == maxSize - 1) {

rear = 0;

}

else {

rear++;

}

arr[rear] = value;

cout << "Inserted " << value << " at Back" << endl;

}

int removeFront() {

if (isEmpty()) {

cout << "Empty" << endl;

return -1;

}

int value = arr[front];

if (front == rear) {

front = rear = -1;

}

else if (front == maxSize - 1) {

front = 0;

}

else {

front++;

}

cout << "Removed " << value << " From Front" << endl;

return value;

}

int removeBack() {

if (isEmpty()) {

cout << "Empty" << endl;

return -1;

}

int value = arr[rear];

if (front == rear) {

front = rear = -1;

}

else if (rear == 0) {

rear = maxSize - 1;

}

else {

rear--;

}

cout << "Removed " << value << " From Back" << endl;

return value;

}

};

int main() {

DoubleEndedQueue obj(5);

obj.insertFront(10);

obj.insertBack(20);

obj.insertFront(30);

obj.insertBack(40);

obj.removeFront();

obj.removeBack();

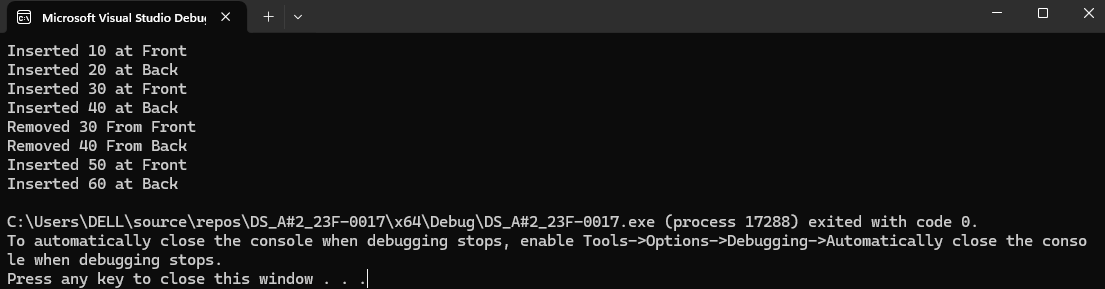
obj.insertFront(50);

obj.insertBack(60);

return 0;

}

**Output:**

****

**Question#6**

**Code:**

#include <iostream>

using namespace std;

class Node {

public:

char data;

Node\* next;

Node(char value):data(value),next(nullptr){}

};

class Queue {

private:

Node\* front;

Node\* rear;

public:

Queue() {

front = nullptr;

rear = nullptr;

}

void enqueue(char value) {

Node\* newNode = new Node(value);

if (rear == nullptr) {

front = rear = newNode;

}

else {

rear->next = newNode;

rear = newNode;

}

}

char dequeue() {

if (front == nullptr) {

cout << "Queue is Empty!" << endl;

return ' ';

}

Node\* temp = front;

char value = front->data;

front = front->next;

if (front == nullptr) {

rear = nullptr;

}

delete temp;

return value;

}

bool isEmpty() {

if (front == nullptr && rear == nullptr) {

return true;

}

return false;

}

Queue copyQueue() {

Queue newQueue;

Node\* current = front;

while (current != nullptr) {

newQueue.enqueue(current->data);

current = current->next;

}

return newQueue;

}

};

class Stack {

private:

Node\* top;

public:

Stack() {

top = nullptr;

}

void push(char value) {

Node\* newNode = new Node(value);

if (top == nullptr) {

top = newNode;

}

else {

newNode->next = top;

top = newNode;

}

}

char pop() {

if (top == nullptr) {

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

return ' ';

}

Node\* temp = top;

char value = top->data;

top = top->next;

delete temp;

return value;

}

bool isEmpty() {

if (top == nullptr) {

return true;

}

return false;

}

};

bool isPalindrome(Queue& q) {

Stack s;

Queue tempQueue = q.copyQueue();

while (!tempQueue.isEmpty()) {

char value = tempQueue.dequeue();

s.push(value);

}

tempQueue = q.copyQueue();

while (!tempQueue.isEmpty()) {

char queueFront = tempQueue.dequeue();

char stackTop = s.pop();

if (queueFront != stackTop) {

return false;

}

}

return true;

}

int main() {

Queue q;

string input = "";

cout << "Enter a String to Check Palindrome : ";cin >> input;

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

q.enqueue(input[i]);

}

if (isPalindrome(q)) {

cout << "The Sequence is a Palindrome" << endl;

}

else {

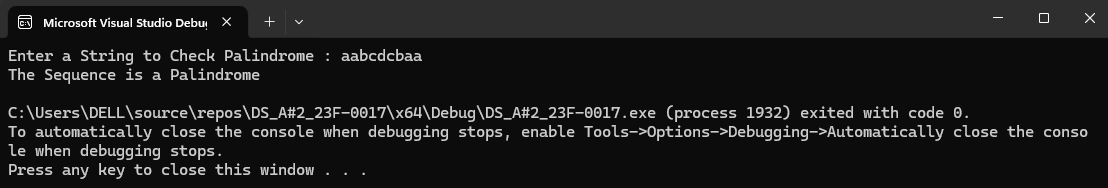
cout << "The Sequence is Not a Palindrome" << endl;

}

return 0;

}

**Output:**

****