#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<char> v{'a', 'e', 'i', 'o', 'u'};

vector<char>::iterator itr;

// Use a for loop to iterate through the vector

for (itr = v.begin(); itr != v.end(); ++itr) {

cout << \*itr << " "; // Print each element followed by a space

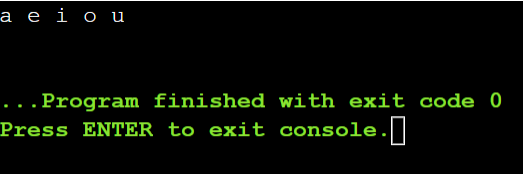
}

cout << endl; // Print a newline at the end

return 0;

}

**Output:**



#include <vector>

using namespace std;

int main() {

vector<char> v{'a', 'e', 'i', 'o', 'u'};

vector<char>::reverse\_iterator rit;

// Use a for loop to iterate through the vector in reverse

for (rit = v.rbegin(); rit != v.rend(); ++rit) {

cout << \*rit << " "; // Print each element followed by a space

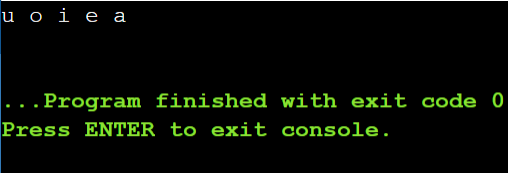
}

cout << endl; // Print a newline at the end

return 0;

}

**Output:**



**Vector:**

#include <iostream>

#include <vector>

#include <algorithm> // Include the algorithm header for std::remove

using namespace std;

template <typename T>

class CustomVector {

private:

vector<T> data;

public:

void add(const T& element) {

data.push\_back(element);

}

void remove(const T& element) {

auto it = std::remove(data.begin(), data.end(), element);

data.erase(it, data.end());

}

T get(size\_t index) const {

if (index >= data.size()) {

throw out\_of\_range("Index out of range");

}

return data[index];

}

size\_t size() const {

return data.size();

}

void print() const {

for (const auto& element : data) {

cout << element << " ";

}

cout << endl;

}

};

int main() {

CustomVector<int> myVector;

myVector.add(1);

myVector.add(2);

myVector.add(3);

cout << "Vector elements: ";

myVector.print();

cout << "Element at index 1: " << myVector.get(1) << endl;

myVector.remove(2);

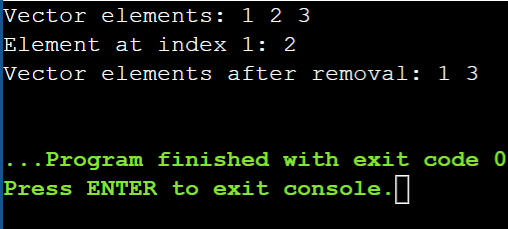
cout << "Vector elements after removal: ";

myVector.print();

return 0;

}

**Output:**



**Queue:push()**

#include<iostream>

#include<queue>

using namespace std;

int main()

{

queue<int>myqueue;

myqueue.push(0);

myqueue.push(1);

myqueue.push(2);

while(!myqueue.empty()){

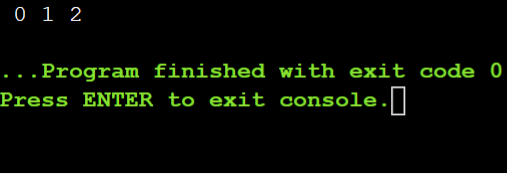
cout<<' '<<myqueue.front();

myqueue.pop();

}

}

**Output:**



**Queue:pop()**

#include <iostream>

#include<queue>

using namespace std;

int main()

{

queue<int> myqueue;

myqueue.push(0);

myqueue.push(1);

myqueue.push(2);

myqueue.pop();

myqueue.pop();

while(!myqueue.empty()){

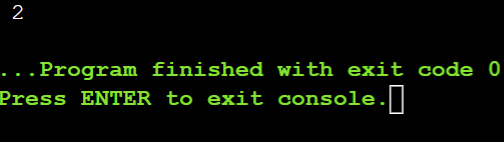
cout<<" "<<myqueue.front();

myqueue.pop();

}

}

**Output:**



**Stack:**

#include<bits/stdc++.h>

using namespace std;

void showstack(stack <int> s)

{

while(!s.empty())

{

cout<<'\t'<<s.top();

s.pop();

}

cout<<'\n';

}

int main()

{

stack<int> s;

s.push(10);

s.push(30);

s.push(20);

s.push(5);

s.push(1);

cout<<"The stack is:";

showstack(s);

cout<<"\ns.size():"<<s.size();

cout<<"\ns.stop():"<<s.top();

cout<<"\ns.pop():";

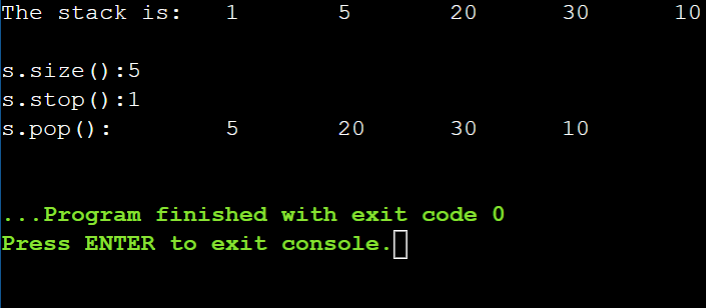
s.pop();

showstack(s);

return 0;

}

**Output:**



**Problem 4: Reverse a Queue**

**Description:**

**Implement a function to reverse the elements of a queue using a stack.**

#include <iostream>

#include <queue>

#include <stack>

using namespace std;

// Function to reverse a queue using a stack

void reverseQueue(queue<int>& q) {

stack<int> s;

// Transfer elements from queue to stack

while (!q.empty()) {

s.push(front());

q.pop();

}

// Transfer elements from stack to queue (reversed order)

while (!s.empty()) {

q.push(s.top());

s.pop();

}

}

// Function to print elements of a queue

void printQueue(queue<int> q) {

while (!q.empty()) {

cout << q.front() << " ";

q.pop();

}

cout << endl;

}

int main() {

queue<int> myqueue;

// Add elements to the queue

myqueue.push(1);

myqueue.push(2);

myqueue.push(3);

myqueue.push(4);

myqueue.push(5);

cout << "Original Queue: ";

printQueue(myqueue); // Output: Original Queue: 1 2 3 4 5

// Reverse the queue using the stack

reverseQueue(myqueue);

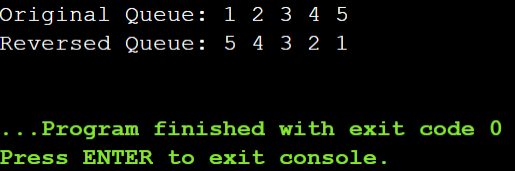
cout << "Reversed Queue: ";

printQueue(myqueue); // Output: Reversed Queue: 5 4 3 2 1

return 0;

}

**Output:**



**Implement Queue Using Stacks**

**Maximum Element in Stack**

**Description:**

**Design a stack that supports push, pop, and retrieving the maximum element in constant time.**

#include <iostream>

#include <stack>

using namespace std;

class MaxStack {

private:

stack<int> elements;

stack<int> maxElements;

public:

void push(int element) {

elements.push(element);

if (maxElements.empty() || element >= maxElements.top()) {

maxElements.push(element);

}

}

int pop() {

int element = elements.top();

elements.pop();

if (element == maxElements.top()) {

maxElements.pop();

}

return element;

}

int getMax() {

return maxElements.top();

}

};

int main() {

MaxStack stack;

stack.push(5);

stack.push(10);

stack.push(3);

stack.push(15);

stack.push(8);

cout<<"Max element: " << stack.getMax()<<endl;

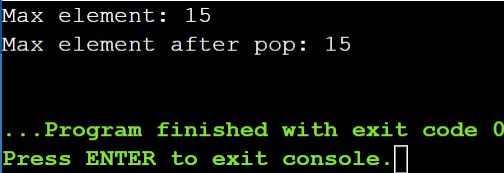
stack.pop();

cout<<"Max element after pop: "<<stack.getMax()<<endl;

return 0;

}

**Output:**



**Circular Queue Implementation**

**Description:**

**Implement a circular queue using an array. The queue should support enqueue, dequeue, and front operations.**

#include <iostream>

using namespace std;

class CircularQueue {

private:

int size;

int front;

int rear;

int \*array;

public:

CircularQueue(int size) {

this->size = size;

front = -1;

rear = -1;

array = new int[size];

}

bool isEmpty() {

return (front == -1);

}

bool isFull() {

return (front == 0 && rear == size - 1);

}

void enqueue(int element) {

if (isFull()) {

cout << "Queue is full. Can't enqueue element." << endl;

return;

}

if (isEmpty()) {

front = 0;

rear = 0;

array[rear] = element;

} else {

rear = (rear + 1) % size;

array[rear] = element;

}

}

int dequeue() {

if (isEmpty()) {

cout << "Queue is empty. Can't dequeue element." << endl;

return -1;

}

int element = array[front];

if (front == rear) {

front = -1;

rear = -1;

} else {

front = (front + 1) % size;

}

return element;

}

int Front() {

if (isEmpty()) {

cout << "Queue is empty. No front element." << endl;

return -1;

}

return array[front];

}

};

int main() {

CircularQueue q(5);

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

q.enqueue(4);

q.enqueue(5);

cout << "Front element: " << q.Front() << endl;

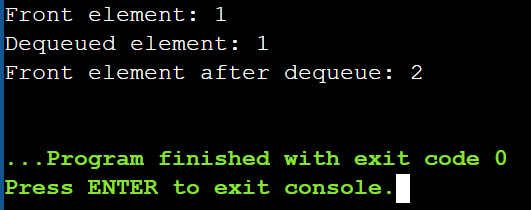
cout << "Dequeued element: " << q.dequeue() << endl;

cout << "Front element after dequeue: " << q.Front() << endl;

return 0;

}

**Output:**



**Sort a Stack**

**Description:**

**Write a function to sort a stack such that the smallest items are on the top.**

#include <iostream>

#include <stack>

using namespace std;

void sortStack(stack<int>& s) {

stack<int> tempStack;

while (!s.empty()) {

int current = s.top();

s.pop();

// Move elements from tempStack back to s until we find the correct position for current

while (!tempStack.empty() && tempStack.top() > current) {

s.push(tempStack.top());

tempStack.pop();

}

// Place current element in the correct sorted position in tempStack

tempStack.push(current);

}

// Move sorted elements from tempStack back to s

while (!tempStack.empty()) {

s.push(tempStack.top());

tempStack.pop();

}

}

int main() {

stack<int> s;

s.push(5);

s.push(2);

s.push(8);

s.push(1);

s.push(3);

cout << "Original Stack: ";

stack<int> original = s;

while (!original.empty()) {

cout << original.top() << " ";

original.pop();

}

cout << endl;

sortStack(s);

cout << "Sorted Stack: ";

while (!s.empty()) {

cout << s.top() << " ";

s.pop();

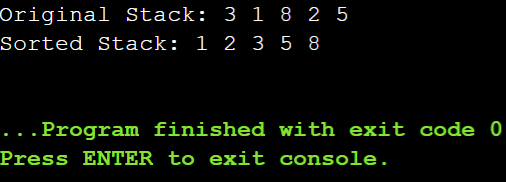
}

cout << endl;

return 0;

}

**Output:**



#include <iostream>

#include <list>

int main() {

// Create a list

std::list<int> myList;

// Insert elements at the end

myList.push\_back(10);

myList.push\_back(20);

myList.push\_back(30);

// Insert elements at the front

myList.push\_front(5);

myList.push\_front(1);

// Display elements

std::cout << "List after push\_back and push\_front: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Insert element at a specific position

auto it = myList.begin();

std::advance(it, 2);

myList.insert(it, 15);

std::cout << "List after insert: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Erase element at a specific position

it = myList.begin();

std::advance(it, 3);

myList.erase(it);

std::cout << "List after erase: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Remove elements by value

myList.remove(10);

std::cout << "List after remove: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Remove elements based on a condition

myList.remove\_if([](int n) { return n < 10; });

std::cout << "List after remove\_if: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Sorting the list

myList.sort();

std::cout << "List after sort: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Reversing the list

myList.reverse();

std::cout << "List after reverse: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Merging two lists

std::list<int> otherList = {40, 50, 60};

myList.merge(otherList);

std::cout << "List after merge: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Clearing the list

myList.clear();

std::cout << "List after clear: ";

for (int val : myList) {

std::cout << val << " ";

}

std::cout << std::endl;

// Checking if the list is empty

if (myList.empty()) {

std::cout << "List is empty." << std::endl;

}

// Adding elements again

myList.push\_back(100);

myList.push\_back(200);

// Accessing front and back elements

std::cout << "Front element: " << myList.front() << std::endl;

std::cout << "Back element: " << myList.back() << std::endl;

return 0;

}

**Output:**

