**Vector:  
Begin:**

#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

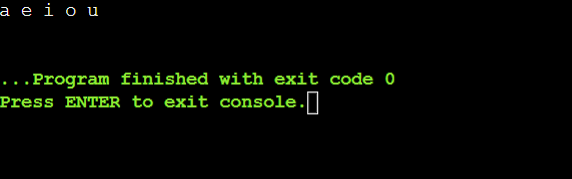
}

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

return 0;

}

**Output:**

****

**Reverse begin:**

#include <iostream>

#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

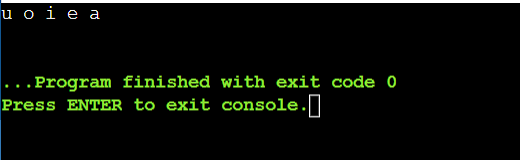
}

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

return 0;

}

**Output:**

****

**Custom vector in class**#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);

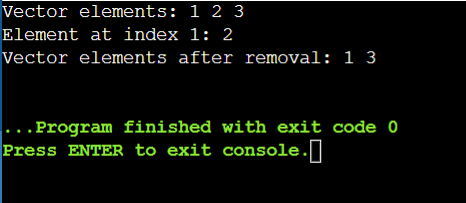
cout << "Vector elements after removal: ";

myVector.print();

return 0;

}

**Output:**

****

**Queue::Push()**

//CPP program to illustrate

//Implementation of push() function

#include<iostream>

#include<queue>

using namespace std;

int main()

{

//Empty Queue

queue<int> myqueue;

myqueue.push(0);

myqueue.push(1);

myqueue.push(2);

//Printing content of queue

while (!myqueue.empty()) {

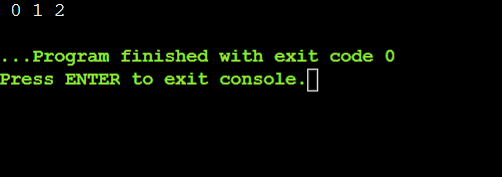
cout <<' '<< myqueue.front();

myqueue.pop();

}

}

**Output:**

****

**Queue::Pop()**

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> myqueue;

// Push elements 0, 1, and 2 into the queue

myqueue.push(0);

myqueue.push(1);

myqueue.push(2);

// Remove the first two elements from the queue

myqueue.pop();

myqueue.pop();

// Print remaining elements in the queue

while (!myqueue.empty()) {

cout << ' ' << myqueue.front(); // Corrected single quotes to double quotes for space

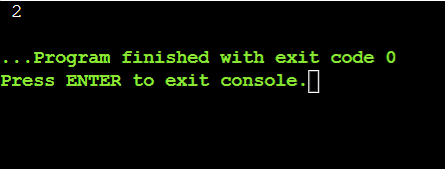
myqueue.pop();

}

return 0;

}

**Output:**

****

**Stack Continued:**

// Cpp program to demonstrate working of STL 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.top() : " << s.top();

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

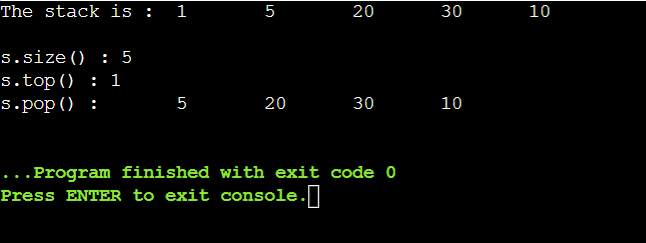
s.pop();

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(q.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);

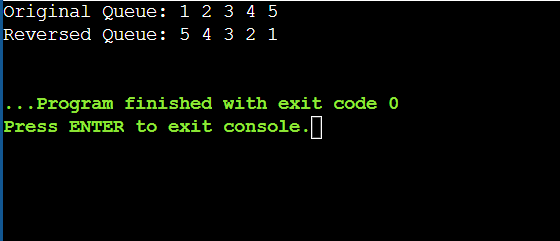
cout << "Reversed Queue: ";

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

return 0;

}

**Output:**

****

**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> mainStack; // Main stack to store elements

stack<int> maxStack; // Auxiliary stack to store maximum elements

public:

void push(int x) {

mainStack.push(x); // Push element onto main stack

// Update maxStack with current maximum

if (maxStack.empty() || x >= maxStack.top()) {

maxStack.push(x);

}

}

void pop() {

if (mainStack.empty()) {

throw runtime\_error("Stack is empty");

}

// Pop element from main stack

int popped = mainStack.top();

mainStack.pop();

// If popped element is the current maximum, pop from maxStack as well

if (popped == maxStack.top()) {

maxStack.pop();

}

}

int getMax() {

if (maxStack.empty()) {

throw runtime\_error("Stack is empty");

}

return maxStack.top(); // Return the maximum element from maxStack

}

};

int main() {

MaxStack stack;

stack.push(44);

stack.push(47);

stack.push(23);

stack.push(43);

cout << "Max element: " << stack.getMax() << endl; // Output: Max element: 47

stack.pop();

cout << "Max element after pop: " << stack.getMax() << endl; // Output: Max element after pop: 47

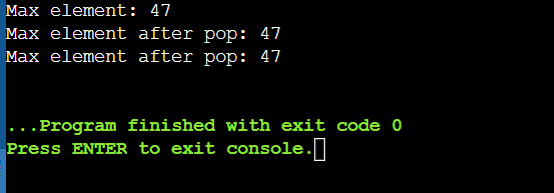
stack.pop();

cout << "Max element after pop: " << stack.getMax() << endl; // Output: Max element after pop: 47

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 \*arr;

int front, rear, capacity, size;

public:

CircularQueue(int cap) {

capacity = cap;

arr = new int[capacity];

front = 0;

rear = -1;

size = 0;

}

~CircularQueue() {

delete[] arr;

}

void enqueue(int value) {

if (isFull()) {

throw runtime\_error("Queue is full");

}

rear = (rear + 1) % capacity;

arr[rear] = value;

size++;

}

int dequeue() {

if (isEmpty()) {

throw runtime\_error("Queue is empty");

}

int frontValue = arr[front];

front = (front + 1) % capacity;

size--;

return frontValue;

}

int frontElement() {

if (isEmpty()) {

throw runtime\_error("Queue is empty");

}

return arr[front];

}

bool isEmpty() {

return size == 0;

}

bool isFull() {

return size == capacity;

}

};

int main() {

CircularQueue cq(44);

cq.enqueue(47);

cq.enqueue(23);

cq.enqueue(42);

cout << "Front element: " << cq.frontElement() << endl; // Output: Front element: 47

cout << "Dequeue: " << cq.dequeue() << endl; // Output: Dequeue: 47

cout << "Front element after dequeue: " << cq.frontElement() << endl; // Output: Front element after dequeue: 23

cq.enqueue(4);

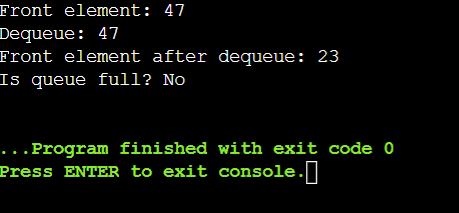
cq.enqueue(5);

cout << "Is queue full? " << (cq.isFull() ? "Yes" : "No") << endl; // Output: Is queue full? Yes

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(44);

s.push(47);

s.push(23);

s.push(43);

s.push(82);

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();

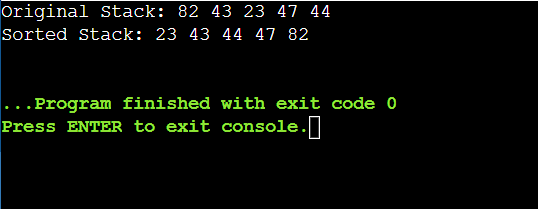
}

cout << endl;

return 0;

}

**Output:**

****

#include <iostream>

#include <list>

using namespace std;

int main() {

// Create a list

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

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

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Insert element at a specific position

auto it = myList.begin();

advance(it, 2);

myList.insert(it, 15);

cout << "List after insert: ";

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Erase element at a specific position

it = myList.begin();

advance(it, 3);

myList.erase(it);

cout << "List after erase: ";

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Remove elements by value

myList.remove(10);

cout << "List after remove: ";

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Remove elements based on a condition

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

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

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Sorting the list

myList.sort();

cout << "List after sort: ";

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Reversing the list

myList.reverse();

cout << "List after reverse: ";

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Merging two lists

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

myList.merge(otherList);

cout << "List after merge: ";

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Clearing the list

myList.clear();

cout << "List after clear: ";

for (int val : myList) {

cout << val << " ";

}

cout << endl;

// Checking if the list is empty

if (myList.empty()) {

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

}

// Adding elements again

myList.push\_back(100);

myList.push\_back(200);

// Accessing front and back elements

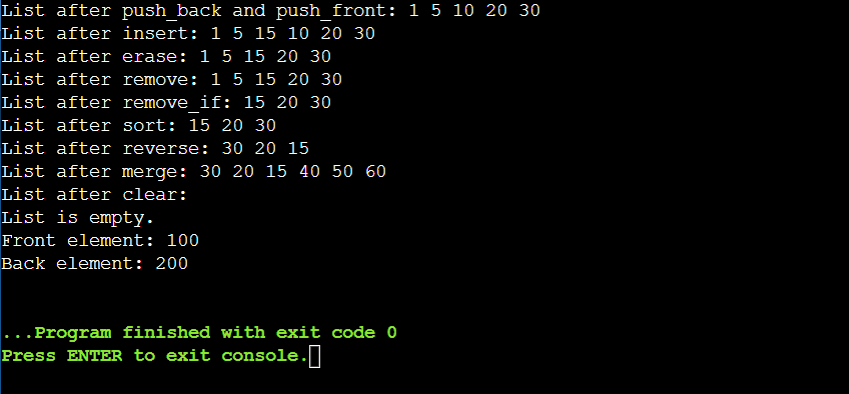
cout << "Front element: " << myList.front() << endl;

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

return 0;

}

**Output:**

****