Q 1. 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;

void reverseQueue(queue<int>& q) {

stack<int> s;

while (!q.empty()) {

s.push(q.front());

q.pop();

}

while (!s.empty()) {

q.push(s.top());

s.pop();

}

}

int main() {

queue<int> myqueue;

myqueue.push(1);

myqueue.push(2);

myqueue.push(3);

myqueue.push(4);

myqueue.push(5);

cout << "Original queue: ";

queue<int> temp = myqueue;

while (!temp.empty()) {

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

temp.pop();

}

cout << endl;

reverseQueue(myqueue);

cout << "Reversed queue: ";

while (!myqueue.empty()) {

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

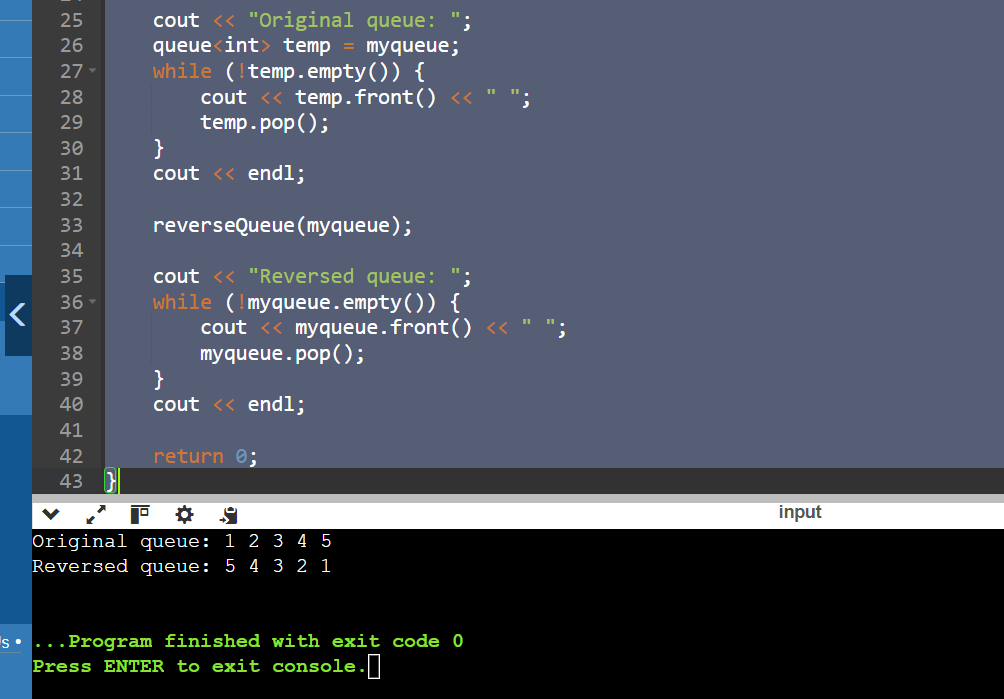
myqueue.pop();

}

cout << endl;

return 0;

}



Q 2 .Implement Queue Using Stacks.

#include <bits/stdc++.h>

using namespace std;

struct Queue {

stack<int> stack1, stack2;

void enqueue(int x)

{

while (!stack1.empty()) {

stack2.push(stack1.top());

stack1.pop();

}

stack1.push(x);

while (!stack2.empty()) {

stack1.push(stack2.top());

stack2.pop();

}

}

int dequeue()

{

if (stack1.empty()) {

cout << "queue is Empty";

exit(0);

}

int x = stack1.top();

stack1.pop();

return x;

}

};

int main()

{

Queue q;

q.enqueue(3);

q.enqueue(4);

q.enqueue(5);

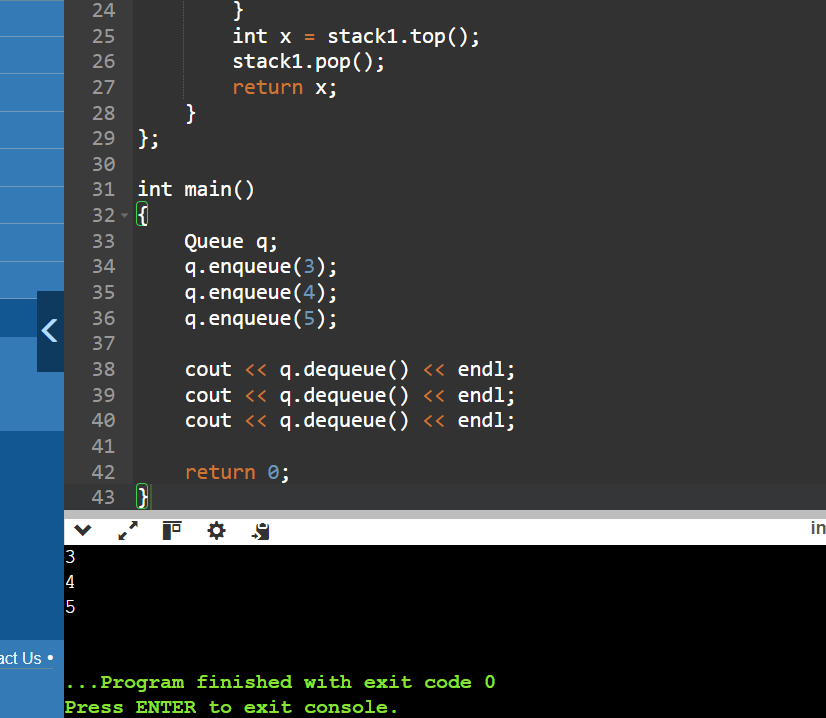
cout << q.dequeue() << endl;

cout << q.dequeue() << endl;

cout << q.dequeue() << endl;

return 0;

}



Q 3. Maximum Element in Stack

Description:

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

#include <iostream>

#include <stack>

#include <stdexcept>

using namespace std;

class MaxStack {

private:

stack<int> mainStack;

stack<int> maxStack;

public:

void push(int x) {

mainStack.push(x);

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

maxStack.push(x);

}

}

void pop() {

if (mainStack.empty()) {

throw out\_of\_range("Stack is empty");

}

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

maxStack.pop();

}

mainStack.pop();

}

int top() {

if (mainStack.empty()) {

throw out\_of\_range("Stack is empty");

}

return mainStack.top();

}

int getMax() {

if (maxStack.empty()) {

throw out\_of\_range("Stack is empty");

}

return maxStack.top();

}

};

int main() {

MaxStack s;

s.push(3);

s.push(15);

cout << "Max: " << s.getMax() << endl;

s.push(17);

s.push(69);

cout << "Max: " << s.getMax() << endl;

s.pop();

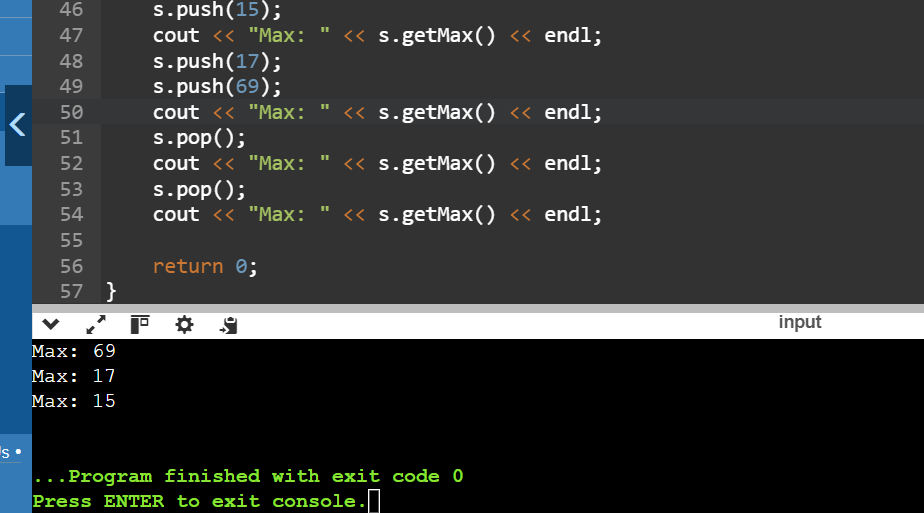
cout << "Max: " << s.getMax() << endl;

s.pop();

cout << "Max: " << s.getMax() << endl;

return 0;

}



Q 4. Circular Queue Implementation

Description:

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

#include <iostream>

#include <vector>

using namespace std;

class CircularQueue {

private:

std::vector<int> queue;

int head;

int tail;

int max\_size;

public:

CircularQueue(int k) : queue(k), head(-1), tail(-1), max\_size(k) {}

bool enqueue(int value) {

if ((tail + 1) % max\_size == head) {

return false;

}

if (head == -1) {

head = 0;

}

tail = (tail + 1) % max\_size;

queue[tail] = value;

return true;

}

bool dequeue() {

if (head == -1) {

return false;

}

if (head == tail) {

head = tail = -1;

} else {

head = (head + 1) % max\_size;

}

return true;

}

int front() {

if (head == -1) {

return -1;

}

return queue[head];

}

bool isEmpty() {

return head == -1;

}

bool isFull() {

return (tail + 1) % max\_size == head;

}

};

int main() {

CircularQueue q(5);

q.enqueue(11);

q.enqueue(22);

q.enqueue(31);

q.enqueue(45);

q.enqueue(51);

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

q.dequeue();

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

q.enqueue(6);

std::cout << "Front element after enqueue 6: " << q.front() << std::endl;

while (!q.isEmpty()) {

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

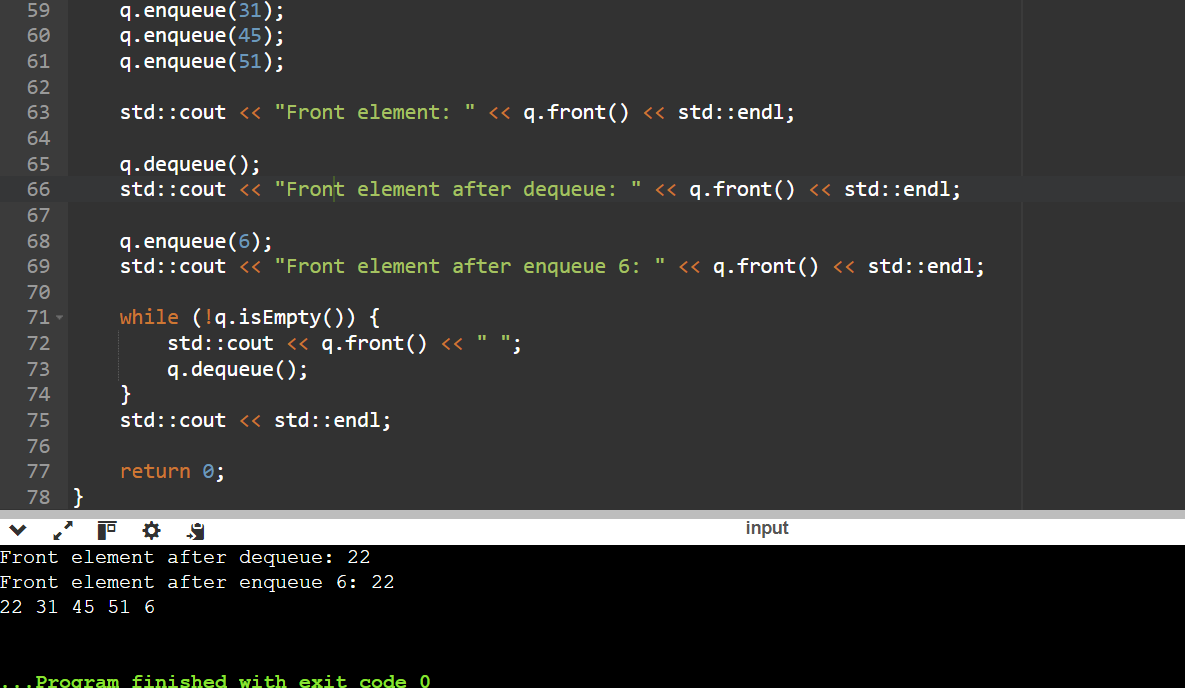
q.dequeue();

}

std::cout << std::endl;

return 0;

}



Q 5. Sort a Stack

Description:

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

#include <stack>

#include <iostream>

void sortStack(std::stack<int>& stack) {

std::stack<int> tempStack;

while (!stack.empty()) {

int temp = stack.top();

stack.pop();

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

stack.push(tempStack.top());

tempStack.pop();

}

tempStack.push(temp);

}

while (!tempStack.empty()) {

stack.push(tempStack.top());

tempStack.pop();

}

}

void printStack(std::stack<int> stack) {

while (!stack.empty()) {

std::cout << stack.top() << " ";

stack.pop();

}

std::cout << std::endl;

}

int main() {

std::stack<int> stack;

stack.push(134);

stack.push(32);

stack.push(331);

stack.push(918);

stack.push(192);

stack.push(123);

std::cout << "Original Stack: ";

printStack(stack);

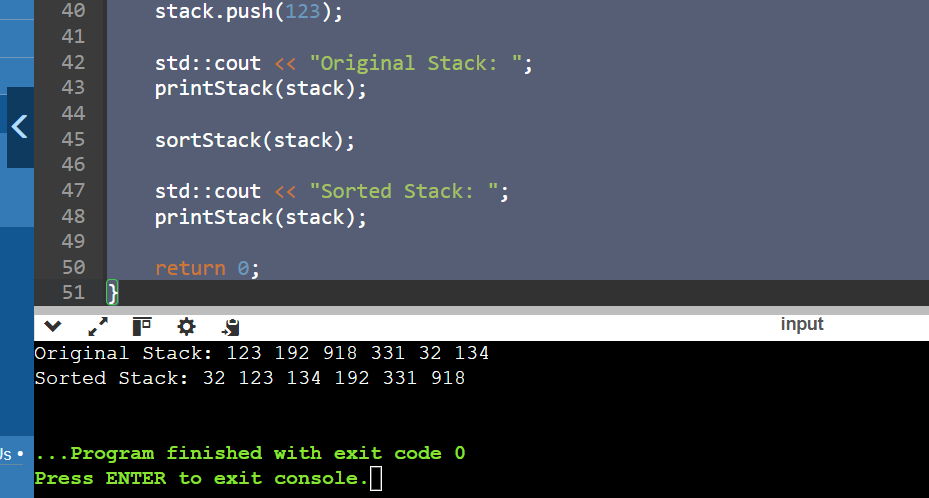
sortStack(stack);

std::cout << "Sorted Stack: ";

printStack(stack);

return 0;

}



Code: #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;

}

