Day-13:

Begin():

#include <vector>

using namespace std;

int main() {

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

vector<char>::iterator itr;

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

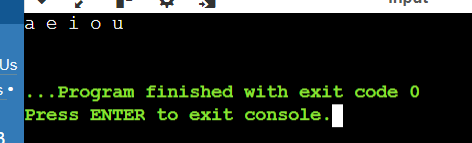
cout << \*itr << " ";

}

cout << endl;

return 0;

}  
output:



Rbegin():

#include <iostream>

#include <vector>

using namespace std;

int main() {

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

vector<char>::reverse\_iterator rit;

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

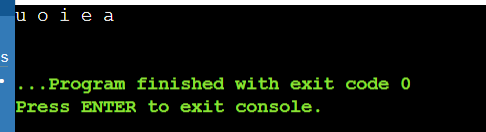
cout << \*rit << " ";

}

cout << endl;

return 0;

}



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())

{

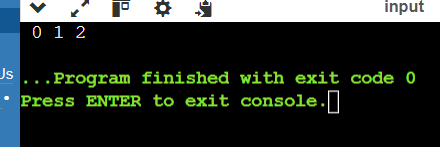
cout<<' '<<myqueue.front();

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())

{

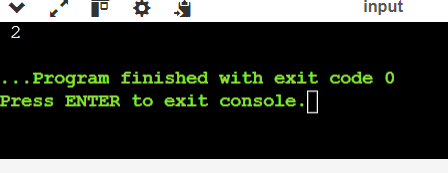
cout<<' '<<myqueue.front();

myqueue.pop();

}

}

Output:



STACK CONTINUED

#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();

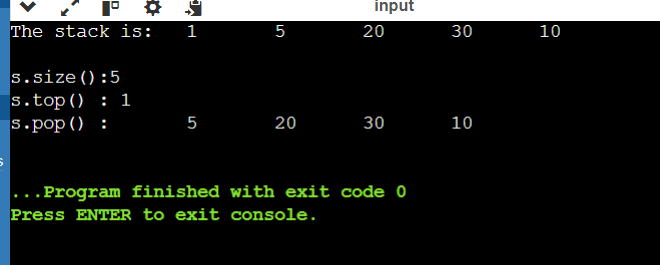
cout<<"\ns.pop() :";

s.pop();

showstack(s);

return 0;

}



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>

void reverseQueue(std::queue<int>& q) {

std::stack<int> s;

while (!q.empty()) {

s.push(q.front());

q.pop();

}

while (!s.empty()) {

q.push(s.top());

s.pop();

}

}

int main() {

std::queue<int> q;

q.push(1);

q.push(2);

q.push(3);

q.push(4);

q.push(5);

std::cout << "Original Queue: ";

std::queue<int> tmp = q;

while (!tmp.empty()) {

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

tmp.pop();

}

std::cout << std::endl;

reverseQueue(q);

std::cout << "Reversed Queue: ";

while (!q.empty()) {

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

q.pop();

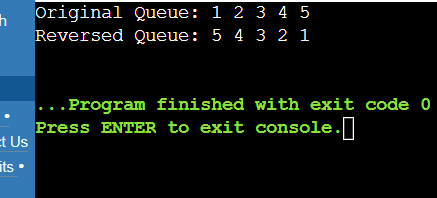
}

std::cout << std::endl;

return 0;

}

Output:



Maximum Element in Stack

Description:

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

#include <stack>

#include <iostream>

#include <cassert>

template<typename T>

class MaxStack {

private:

std::stack<T> data\_stack;

std::stack<T> max\_stack;

public:

void push(const T& value) {

data\_stack.push(value);

if (max\_stack.empty() || value >= max\_stack.top()) {

max\_stack.push(value);

}

}

void pop() {

assert(!data\_stack.empty());

if (data\_stack.top() == max\_stack.top()) {

max\_stack.pop();

}

data\_stack.pop();

}

T getMax() const {

assert(!max\_stack.empty());

return max\_stack.top();

}

bool empty() const {

return data\_stack.empty();

}

};

int main() {

MaxStack<int> stack;

stack.push(5);

stack.push(3);

stack.push(9);

stack.push(7);

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

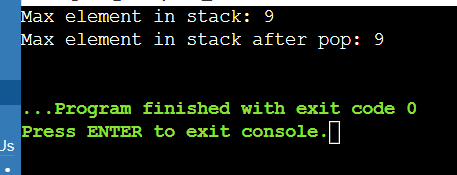
stack.pop();

std::cout << "Max element in stack after pop: " << stack.getMax() << std::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>

#include <vector>

#include <stdexcept>

template<typename T>

class CircularQueue {

private:

std::vector<T> data;

int front;

int rear;

int capacity;

int currentSize;

public:

CircularQueue(int size) : data(size), front(-1), rear(-1), capacity(size), currentSize(0) {}

bool isEmpty() {

return currentSize == 0;

}

bool isFull() {

return currentSize == capacity;

}

int size() {

return currentSize;

}

void enqueue(const T& value) {

if (isFull()) {

throw std::runtime\_error("Queue overflow");

}

rear = (rear + 1) % capacity;

data[rear] = value;

if (front == -1) {

front = rear;

}

currentSize++;

}

T dequeue() {

if (isEmpty()) {

throw std::runtime\_error("Queue underflow");

}

T frontElement = data[front];

if (front == rear) {

front = -1;

rear = -1;

} else {

front = (front + 1) % capacity;

}

currentSize--;

return frontElement;

}

T frontElement() {

if (isEmpty()) {

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

}

return data[front];

}

};

int main() {

CircularQueue<int> queue(5);

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

std::cout << "Front element: " << queue.frontElement() << std::endl;

queue.dequeue();

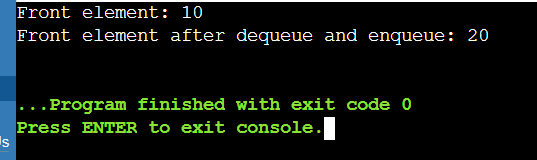
queue.enqueue(40);

std::cout << "Front element after dequeue and enqueue: " << queue.frontElement() << std::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 <stack>

#include <iostream>

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

std::stack<int> temp\_stack;

while (!original\_stack.empty()) {

int current = original\_stack.top();

original\_stack.pop();

while (!temp\_stack.empty() && temp\_stack.top() > current) {

original\_stack.push(temp\_stack.top());

temp\_stack.pop();

}

temp\_stack.push(current);

}

while (!temp\_stack.empty()) {

original\_stack.push(temp\_stack.top());

temp\_stack.pop();

}

}

int main() {

std::stack<int> s;

s.push(5);

s.push(2);

s.push(8);

s.push(1);

s.push(3);

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

std::stack<int> tmp = s;

while (!tmp.empty()) {

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

tmp.pop();

}

std::cout << std::endl;

sortStack(s);

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

while (!s.empty()) {

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

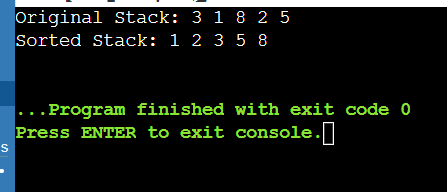
s.pop();

}

std::cout << std::endl;

return 0;

}



#include <iostream>

#include <list>

int main() {

std::list<int> myList;

myList.push\_back(10);

myList.push\_back(20);

myList.push\_back(30);

myList.push\_front(5);

myList.push\_front(1);

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

for (int val : myList) {

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

}

std::cout << std::endl;

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;

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;

myList.remove(10);

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

for (int val : myList) {

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

}

std::cout << std::endl;

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;

myList.sort();

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

for (int val : myList) {

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

}

std::cout << std::endl;

myList.reverse();

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

for (int val : myList) {

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

}

std::cout << std::endl;

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;

myList.clear();

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

for (int val : myList) {

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

}

std::cout << std::endl;

if (myList.empty()) {

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

}

myList.push\_back(100);

myList.push\_back(200);

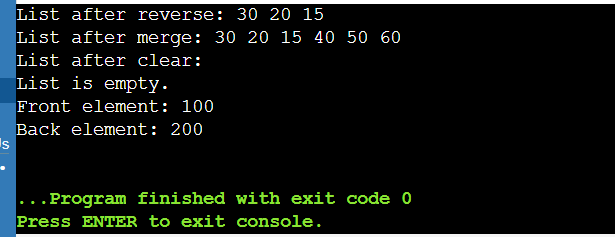
std::cout << "Front element: " << myList.front() << std::endl;

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

return 0;

}

Output:

output