MOUNIKA C++ TRAINIG

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:

a e i o u

RBEGIN():

#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:

u o I e a

**QUEUE::PUSH()**

#include <iostream>

#include <queue>

using namespace std;

int main()

{

// Empty Queue

queue<int> myqueue;

myqueue.push(0);

myqueue.push(1);

myqueue.push(2);

while (!myqueue.empty()) {

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

myqueue.pop();

}

}

OUTPUT:

0 1 2

**STACK**

Stacks are a type of container adaptors with LIFO (Last In First Out) type of working, where a new elemen is added at one end and (top) an element is removed from that end only. Stack uses an encapsulated object of either vector or deque (by default) or list (sequential container class) as its underlying container, providing a specific set of member functions to access its elements.

The functions associated with stack are:

empty() - Returns whether the stack is empty - Time Complexity: O(1)

-

size() Returns the size of the stack - Time Complexity: O(1)

top() - Returns a reference to the top most element of the stack - Time Complexity: O(1)

push(g) - Adds the element 'g' at the top of the stack - Time Complexity: O(1)

pop() - Deletes the top most element of the stack - Time Complexity: O(1)

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;

}

OUTPUT:

The stack is: 1 5 20 30 10

s.size():5

s.top():1

s.pop(): 5 20 30 10

**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();

}

}

void printQueue(queue<int> q) {

while (!q.empty()) {

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

q.pop();

}

cout << endl;

}

int main() {

queue<int> myqueue;

myqueue.push(1);

myqueue.push(2);

myqueue.push(3);

myqueue.push(4);

myqueue.push(5);

cout << "Original Queue: ";

printQueue(myqueue);

reverseQueue(myqueue);

cout << "Reversed Queue: ";

printQueue(myqueue);

return 0;

}

OUTPUT:

Original Queue: 1 2 3 4 5

Reversed Queue: 5 4 3 2 1

**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;

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 runtime\_error("Stack is empty");

}

int popped = mainStack.top();

mainStack.pop();

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

maxStack.pop();

}

}

int getMax() {

if (maxStack.empty()) {

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

}

return maxStack.top(); }

};

int main() {

MaxStack stack;

stack.push(44);

stack.push(47);

stack.push(23);

stack.push(43);

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

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

cout << "Max element after pop: " << stack.getMax() << endl; return 0;

}

OUTPUT:

Max element: 47

Max element after pop: 47

Max element after pop: 47

**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(5);

cq.enqueue(1);

cq.enqueue(2);

cq.enqueue(3);

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

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

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

cq.enqueue(4);

cq.enqueue(5);

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

return 0;

}

OUTPUT:

Front element: 1

Dequeue: 1

Front element after dequeue: 2

Is queue full? No

**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();

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

s.push(tempStack.top());

tempStack.pop();

}

tempStack.push(current);

}

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:

Original Stack: 3 1 8 2 5

Sorted Stack: 1 2 3 5 8