Implementing a own Vector in C++

#include<vector>

#include<iostream>

using namespace std;

template <typename T> class vectorClass {

T\* arr;

int capacity;

int current;

public:

vectorClass()

{

arr = new T[1];

capacity = 1;

current = 0;

}

void push(T data)

{

if (current == capacity) {

T\* temp = new T[2 \* capacity];

for (int i = 0; i < capacity; i++) {

temp[i] = arr[i];

}

}

arr[current] = data;

current++;

}

void push(T data, int index)

{

if (index == capacity)

push(data);

else

arr[index] = data;

}

T get(int index)

{

if (index < current)

return arr[index];

return -1;

}

void pop() { current--; }

int size() { return current; }

int getcapacity() { return capacity; }

void print()

{

for (int i = 0; i < current; i++) {

cout << arr[i] << " ";

}

cout << endl;

}

};

int main()

{

vectorClass<int> v;

vectorClass<char> v1;

v.push(10);

v.push(20);

v.push(30);

v.push(40);

v.push(50);

v1.push(71);

v1.push(72);

v1.push(73);

v1.push(74);

cout << "Vector size : " << v.size() << endl;

cout << "Vector capacity : " << v.getcapacity() << endl;

cout << "Vector elements : ";

v.print();

v.push(100, 1);

cout << "\nAfter updating 1st index" << endl;

cout << "Vector elements of type int : " << endl;

v.print();

cout << "Vector elements of type char : " << endl;

v1.print();

cout << "Element at 1st index of type int: " << v.get(1)

<< endl;

cout << "Element at 1st index of type char: "

<< v1.get(1) << endl;

v.pop();

v1.pop();

cout << "\nAfter deleting last element" << endl;

cout << "Vector size of type int: " << v.size() << endl;

cout << "Vector size of type char: " << v1.size()

<< endl;

cout << "Vector capacity of type int : "

<< v.getcapacity() << endl;

cout << "Vector capacity of type char : "

<< v1.getcapacity() << endl;

cout << "Vector elements of type int: ";

v.print();

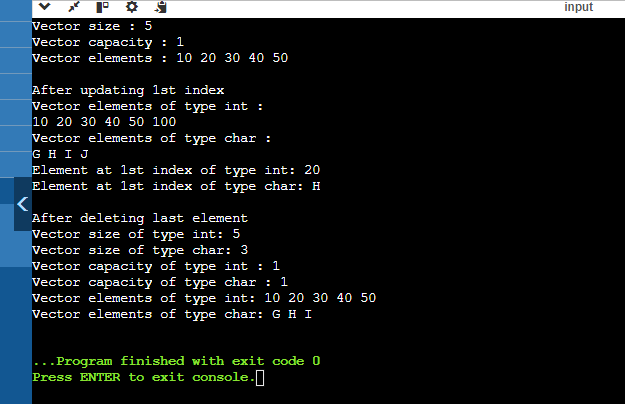
cout << "Vector elements of type char: ";

v1.print();

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;

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();

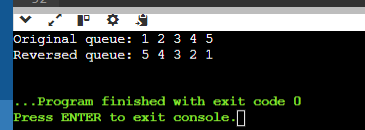
}

cout << endl;

return 0;

}

Output:



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;

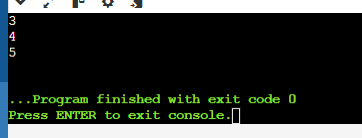
cout << q.dequeue() << endl;

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

return 0;

}

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>

#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(5);

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

s.push(7);

s.push(19);

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

s.pop();

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

s.pop();

cout << "Max: " << s.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>

#include <vector>

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(1);

q.enqueue(2);

q.enqueue(3);

q.enqueue(4);

q.enqueue(5);

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();

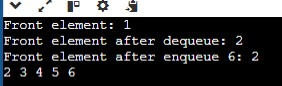
}

std::cout << std::endl;

return 0;

}

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>& 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(34);

stack.push(3);

stack.push(31);

stack.push(98);

stack.push(92);

stack.push(23);

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

printStack(stack);

sortStack(stack);

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

printStack(stack);

return 0;

}

Output:

