**Date:10/07/2024**

**Day\_13\_Assignments**

**1.**

#include <iostream>

#include <vector>

using namespace std;

int main() {

std::vector<int> numbers = {1, 2, 3, 4, 5};

cout << " ";

for (auto it = numbers.begin(); it != numbers.end(); ++it) {

cout << \*it << " ";

}

cout<<endl;

cout << " ";

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

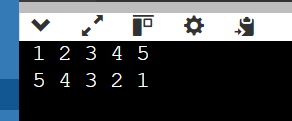
cout << \*rit << " ";

}

cout<<endl

return 0;

}



2.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct student {

std::string name;

int marks;

student(std::string n, int a) : name(n), marks(a) {}

};

class studentdetails {

private:

std::vector<student>people;

public:

void addstudent(const std::string &name, int marks) {

people.emplace\_back(name, marks);

}

void sortBymarks() {

std::sort(people.begin(), people.end(), [](const student &a, const student &b) {

return a.marks < b.marks;

});

}

void display() const {

for (const auto &student : people) {

cout << student.name << ": " << student.marks <<endl;

}

}

};

int main() {

studentdetails sd;

sd.addstudent("ram", 30);

sd.addstudent("dev", 68);

sd.addstudent("rama", 56);

sd.addstudent("seema", 78);

cout << "students before sorting:" <<endl;

sd.display();

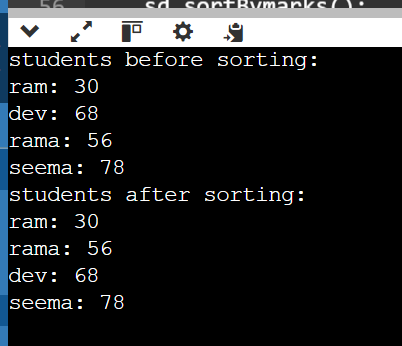
sd.sortBymarks();

cout << "students after sorting:" <<endl;

sd.display();

return 0;

}



3.

#include<iostream>

#include<queue>

using namespace std;

int main()

{

queue<int>myqueue;

myqueue.push(0);

myqueue.push(1);

myqueue.push(2);

while(!myqueue.empty())

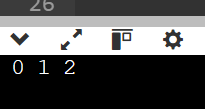
{

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

myqueue.pop();

}

}



4.

#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();

}

}

5.

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

void showstack(stack <int> s)

{

while(!s.empty())

{

cout<<' '<<s.top();

s.pop();

}

cout<<' ';

}

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();

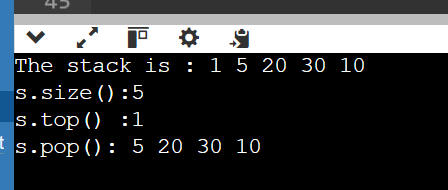
cout<<"\ns.pop():";

s.pop();

showstack(s);

return 0;

}



Q.6

#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> q;

for (int i = 1; i <= 5; ++i) {

q.push(i);

}

cout << "Original queue: ";

queue<int> temp = q;

while (!temp.empty()) {

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

temp.pop();

}

cout <<endl;

reverseQueue(q);

cout << "Reversed queue: ";

while (!q.empty()) {

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

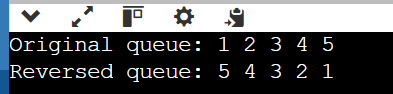
q.pop();

}

cout <<endl;

return 0;

}



7.

#include <iostream>

#include <stack>

using namespace std;

class MaxStack {

private:

stack<int> mainStack;

stack<int> maxStack;

public:

void push(int value) {

mainStack.push(value);

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

maxStack.push(value);

}

}

void pop() {

if (mainStack.empty()) {

std::cout << "Stack is empty" << std::endl;

return;

}

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

maxStack.pop();

}

mainStack.pop();

}

int top() {

if (mainStack.empty()) {

std::cout << "Stack is empty" << std::endl;

return -1;

}

return mainStack.top();

}

int getMax() {

if (maxStack.empty()) {

cout << "Stack is empty" <<endl;

return -1;

}

return maxStack.top();

}

};

int main() {

MaxStack s;

s.push(3);

s.push(5);

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

s.push(7);

s.push(19);

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

s.pop();

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

s.pop();

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

s.pop();

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

s.pop();

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

return 0;

}

8.

#include <iostream>

using namespace std;

class CircularQueue {

private:

int \*queue;

int front, rear, size, capacity;

public:

CircularQueue(int cap) {

capacity = cap;

queue = new int[capacity];

front = -1;

rear = -1;

size = 0;

}

~CircularQueue() {

delete[] queue;

}

bool enqueue(int value) {

if (isFull()) {

std::cout << "Queue is full" << std::endl;

return false;

}

if (isEmpty()) {

front = 0;

}

rear = (rear + 1) % capacity;

queue[rear] = value;

size++;

return true;

}

bool dequeue() {

if (isEmpty()) {

cout << "Queue is empty" <<endl;

return false;

}

if (front == rear) {

front = -1;

rear = -1;

} else {

front = (front + 1) % capacity;

}

size--;

return true;

}

int getFront() {

if (isEmpty()) {

std::cout << "Queue is empty" << std::endl;

return -1;

}

return queue[front];

}

bool isEmpty() {

return size == 0;

}

bool isFull() {

return size == capacity;

}

int getSize() {

return size;

}

};

int main() {

CircularQueue q(3);

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

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

q.dequeue();

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

q.enqueue(4);

cout << "Front element after enqueue 4: " << q.getFront() << endl;

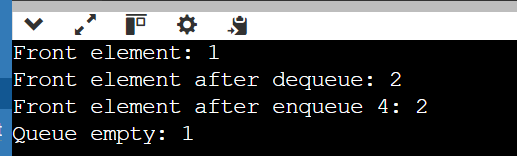
q.dequeue();

q.dequeue();

q.dequeue();

cout << "Queue empty: " << q.isEmpty() << endl;

return 0;}



9.

#include <iostream>

#include <stack>

using namespace std;

void sortStack(stack<int>& s) {

std::stack<int> tempStack;

while (!s.empty()) {

int tmp = s.top();

s.pop();

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

s.push(tempStack.top());

tempStack.pop();

}

tempStack.push(tmp);

}

while (!tempStack.empty()) {

s.push(tempStack.top());

tempStack.pop();

}

}

int main() {

stack<int> s;

s.push(5);

s.push(3);

s.push(8);

s.push(1);

s.push(2);

sortStack(s);

cout << "Sorted Stack: ";

while (!s.empty()) {

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

s.pop();

}

cout <<endl;

return 0;

}



10.

#include <iostream>

#include <list>

int main() {

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;

}

