OOPS-Assignment2

Name: Arpon Sadhukhan

Roll: 22CS8086

Section:X

Date Assigned:10.01.2024

Date Submitted:17.01.2024

**Program 1**

**Question:**

Implement a stack struct. You should have atleast these member functions:

initialize, push, pop, display, isempty, isfull, etc. Show the use of the stack

by creating structure elements and calling the member functions.

**Code:**

#include <iostream>

using namespace std;

class Stack{

private:

static const int MAX\_SIZE=100;

int top;

int arr[MAX\_SIZE];

public:

Stack() : top(-1) {}

void push(int value) {

if (top < MAX\_SIZE - 1) {

arr[++top] = value;

cout << "Pushed: " << value << endl;

} else {

cout << "Stack overflow! Cannot push more elements." << endl;

}

}

void pop() {

if (top >= 0) {

cout << "Popped: " << arr[top--] <<endl;

} else {

cout << "Stack underflow! Cannot pop from an empty stack." << endl;

}

}

void initialize() {

top = -1; }

bool isEmpty() const {

return top == -1;

}

bool isFull() const {

return top == MAX\_SIZE - 1;

}

void display() const {

if (isEmpty()) {

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

} else {

cout << "Stack elements: ";

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

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

}

cout << endl;

}

}

};

int main() {

Stack stack;

stack.push(10);

stack.push(20);

stack.push(30);

stack.display();

stack.pop();

stack.display();

cout << "Is stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

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

return 0;

}

**Output:**

Pushed: 10

Pushed: 20

Pushed: 30

Stack elements: 10 20 30

Popped: 30

Stack elements: 10 20

Is stack empty? No

Is stack full? No

**Program 2**

**Question:**

Implement a queue struct. You should have atleast these member

functions: initialize, push, pop, display, isempty, isfull, etc. Show the use of

the stack by creating structure elements and calling the member functions.

**Code:**

**#include <iostream>**

**using namespace std;**

**class Queue{**

**private:**

**static const int MAX\_SIZE=100;**

**int front;**

**int rear;**

**int arr[MAX\_SIZE];**

**public:**

**Queue() : front(-1), rear(-1) {}**

**void enqueue(int value) {**

**if (rear < MAX\_SIZE - 1) {**

**if (front == -1) {**

**front = 0;**

**}**

**arr[++rear] = value;**

**cout << "Enqueued: " << value << endl;**

**} else {**

**cout << "Queue overflow! Cannot enqueue more elements." << endl;**

**}**

**}**

**void dequeue() {**

**if (front <= rear && front != -1) {**

**cout << "Dequeued: " << arr[front++] << endl;**

**if (front > rear) {**

**front = rear = -1; // Reset queue when empty**

**}**

**} else {**

**cout << "Queue underflow! Cannot dequeue from an empty queue." << endl;**

**}**

**}**

**void initialize() {**

**front = rear = -1;**

**}**

**bool isEmpty() const {**

**return front == -1;**

**}**

**bool isFull() const {**

**return rear == MAX\_SIZE - 1;**

**}**

**void display(){**

**if(isEmpty()){**

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

**}**

**else{**

**cout<<"Queue Elements:";**

**for(int i=front;i<=rear;i++){**

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

**}**

**cout<<endl;**

**}**

**}**

**};**

**int main(){**

**Queue queue;**

**queue.enqueue(10);**

**queue.enqueue(20);**

**queue.enqueue(30);**

**queue.display();**

**queue.dequeue();**

**queue.dequeue();**

**queue.display();**

**cout << "Is queue empty? " << (queue.isEmpty() ? "Yes" : "No") << endl;**

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

**return 0;**

**}**

**Output:**

Enqueued: 10

Enqueued: 20

Enqueued: 30

Queue Elements:10

20

30

Dequeued: 10

Dequeued: 20

Queue Elements:30

Is queue empty? No

Is queue full? No

**Program 3**

**Question:**

Implement a matrix struct. You should have atleast these member

functions: allocatespace, readmatrix, displayMatrix, addMartix, multMatrix,

detMatrix.

**Code:**

**#include <iostream>**

**using namespace std;**

**class Matrix{**

**private:**

**int rows;**

**int cols;**

**int\*\* matrix;**

**public:**

**Matrix(int rows, int cols) : rows(rows), cols(cols) {**

**allocateSpace();**

**}**

**~Matrix() {**

**deallocateSpace();**

**}**

**void allocateSpace() {**

**matrix = new int\*[rows];**

**for (int i = 0; i < rows; ++i) {**

**matrix[i] = new int[cols];**

**}**

**}**

**void deallocateSpace() {**

**for (int i = 0; i < rows; ++i) {**

**delete[] matrix[i];**

**}**

**delete[] matrix;**

**}**

**void readMatrix() {**

**cout << "Enter matrix elements:" << endl;**

**for (int i = 0; i < rows; ++i) {**

**for (int j = 0; j < cols; ++j) {**

**cout << "Enter element at position (" << i + 1 << "," << j + 1 << "): ";**

**cin >> matrix[i][j];**

**}**

**}**

**}**

**void displayMatrix() const {**

**cout << "Matrix:" << endl;**

**for (int i = 0; i < rows; ++i) {**

**for (int j = 0; j < cols; ++j) {**

**std::cout << matrix[i][j] << " ";**

**}**

**cout << endl;**

**}**

**}**

**Matrix addMatrix(Matrix& other){**

**Matrix result(rows, cols);**

**for (int i = 0; i < rows; ++i) {**

**for (int j = 0; j < cols; ++j) {**

**result.matrix[i][j] = matrix[i][j] + other.matrix[i][j];**

**}**

**}**

**return result;**

**}**

**Matrix multiplyMatrix(Matrix& other){**

**Matrix result(rows, other.cols);**

**for (int i = 0; i < rows; ++i) {**

**for (int j = 0; j < other.cols; ++j) {**

**result.matrix[i][j] = 0;**

**for (int k = 0; k < cols; ++k) {**

**result.matrix[i][j] += matrix[i][k] \* other.matrix[k][j];**

**}**

**}**

**}**

**return result;**

**}**

**int determinant(){**

**// Assuming it's a 2x2 matrix for simplicity**

**if (rows == 2 && cols == 2) {**

**return (matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0]);**

**} else {**

**cerr << "Determinant calculation is supported only for 2x2 matrices." <<endl;**

**return 0;**

**}**

**/\***

**if (rows == 0 ||cols == 0|| rows!=cols) {**

**cerr << "Invalid matrix size." << endl;**

**return 0.0;**

**}**

**if (rows == 1) {**

**return matrix[0][0];**

**}**

**double det = 0.0;**

**for (int i = 0; i < rows; i++) {**

**Matrix submatrix(rows - 1, (rows - 1));**

**for (int j = 1; j < rows; j++) {**

**for (int k = 0, l = 0; k < rows; k++) {**

**if (k != i) {**

**submatrix.matrix[j - 1][l++] = matrix[j][k];**

**}**

**}**

**}**

**det += (i % 2 == 0 ? 1 : -1) \* matrix[0][i] \* determinant(submatrix);**

**}**

**return det;**

**\*/**

**}**

**};**

**int main() {**

**int rows, cols;**

**cout << "Enter the number of rows for the matrix: ";**

**cin >> rows;**

**cout << "Enter the number of columns for the matrix: ";**

**cin >> cols;**

**Matrix matrix1(rows, cols);**

**matrix1.readMatrix();**

**matrix1.displayMatrix();**

**Matrix matrix2(rows, cols);**

**matrix2.readMatrix();**

**matrix2.displayMatrix();**

**Matrix sumMatrix = matrix1.addMatrix(matrix2);**

**sumMatrix.displayMatrix();**

**Matrix productMatrix = matrix1.multiplyMatrix(matrix2);**

**productMatrix.displayMatrix();**

**cout << "Determinant of the first matrix: " << matrix1.determinant() << endl;**

**return 0;**

**}**

**Output:**

Enter the number of rows for the matrix: 2

Enter the number of columns for the matrix: 2

Enter matrix elements:

Enter element at position (1,1): 1

Enter element at position (1,2): 2

Enter element at position (2,1): 3

Enter element at position (2,2): 4

Matrix:

1 2

3 4

Enter matrix elements:

Enter element at position (1,1): 2

Enter element at position (1,2): 3

Enter element at position (2,1): 4

Enter element at position (2,2): 5

Matrix:

2 3

4 5

Matrix:

3 5

7 9

Matrix:

10 13

22 29

Determinant of the first matrix: -2