**DS Lab 2 Tasks:**

**By: Shozab Mehdi (22k-4522)**

**Q.1**

#include <iostream>

#include <cstring>

using namespace std;

class Memory\_Array {

private:

    int size;

    int capacity; // Track the capacity of the array

    int \*vals;

public:

    Memory\_Array() {

        size = 0;

        capacity = 0;

        vals = NULL;

    }

    Memory\_Array(int s) {

        size = 0;

        capacity = s;

        vals = new int[capacity];

    }

    Memory\_Array(const Memory\_Array& other) {

        size = other.size;

        capacity = other.capacity;

        vals = new int[capacity];

        memcpy(vals, other.vals, sizeof(int) \* size);

    }

    Memory\_Array& operator=(const Memory\_Array& other) {

        if (this == &other)

            return \*this;

        delete[] vals;

        vals = NULL;

        size = other.size;

        capacity = other.capacity;

        vals = new int[capacity];

        memcpy(vals, other.vals, sizeof(int) \* size);

        return \*this;

    }

    ~Memory\_Array() {

        delete[] vals;

        vals = NULL;

    }

    void fillArray() {

        cout << "Enter " << capacity << " values: ";

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

            cin >> vals[i];

        }

        size = capacity;

    }

    void insert(int element) {

    // if (size == capacity) {

    //     // If the array is full, double the capacity

    //     capacity \*= 2;

    //     int \*temp = new int[capacity];

    //     memcpy(temp, vals, sizeof(int) \* size);

    //     delete[] vals;

    //     vals = temp;

    // }

    // vals[size] = element;

    // size++;

    // vals = new int[capacity+1];

    // vals[capacity+1]=element;

    int \*temp = new int[capacity + 1];

    memcpy(temp, vals, sizeof(int) \* size);

    temp[size] = element;

    delete[] vals;

    vals = temp;

    size++;

    capacity++;

}

    void deleteLast() {

        if (size == 0) {

            cout << "Array is empty" << endl;

            return;

        }

        size--;

    }

    int getElementAt(int index) const {

        if (index < 0 || index >= size) {

            cout << "Invalid index" << endl;

            return -1;

        }

        return vals[index];

    }

    int printarray(){

    for (int i=0;i<size;i++){

        cout << vals[i];

    }

    }

    int getsize() const{

        cout << sizeof(vals);

    }

};

int main() {

    int size;

    cout << "Enter the size of the array: ";

    cin >> size;

    Memory\_Array arr(size);

    cout << "Filling the array:" << endl;

    arr.fillArray();

    cout << "Deleted last element" << endl;

    arr.deleteLast();

    // cout << "Updated array" << endl;

    // for (int i=0;i<size;i++){

    //     cout << \*(arr+i);

    // }

    cout << "Inserted element: 100" << endl;

    arr.insert(100);

    cout << "Elements at 1st index in the array:" << endl;

        cout << arr.getElementAt(1) << " ";

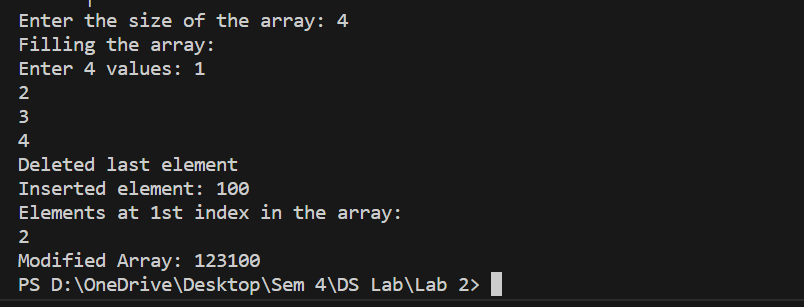
    cout << endl;

    cout << "Modified Array: " << arr.printarray() << " ";

    return 0;

}

**Output:**

****

**Q.2**

#include<iostream>

using namespace std;

int main() {

    int rows, cols;

    cout << "Input rows" << endl;

    cin >> rows;

    cout << "Input cols" << endl;

    cin >> cols;

    int \*\*arr=new int\* [rows];

    int \*avg= new int [rows];

    // int \*\*avg=new int\* [rows];

    for (int i=0;i<rows;++i) {

        arr[i]=new int[cols];

        // avg[i]=new int[cols];

    }

    for(int i=0;i<rows;i++){

        for (int j=0;j<cols;j++){

            cout << "Input the element at  " << i+1 << " row and " << j+1 << " column of the array" << endl;

            cin >> arr[i][j];

        }

    }

    for (int i=0;i<rows;i++){

        avg[i]=0;

        for (int j=0;j<cols;j++){

            avg[i]+=arr[i][j];

        }

        avg[i]=avg[i]/cols;

    }

    for(int i=0;i<rows;i++){

        cout << avg[i] << endl;

    }

    for (int i = 0; i < rows; ++i) {

        delete[] arr[i];

    }

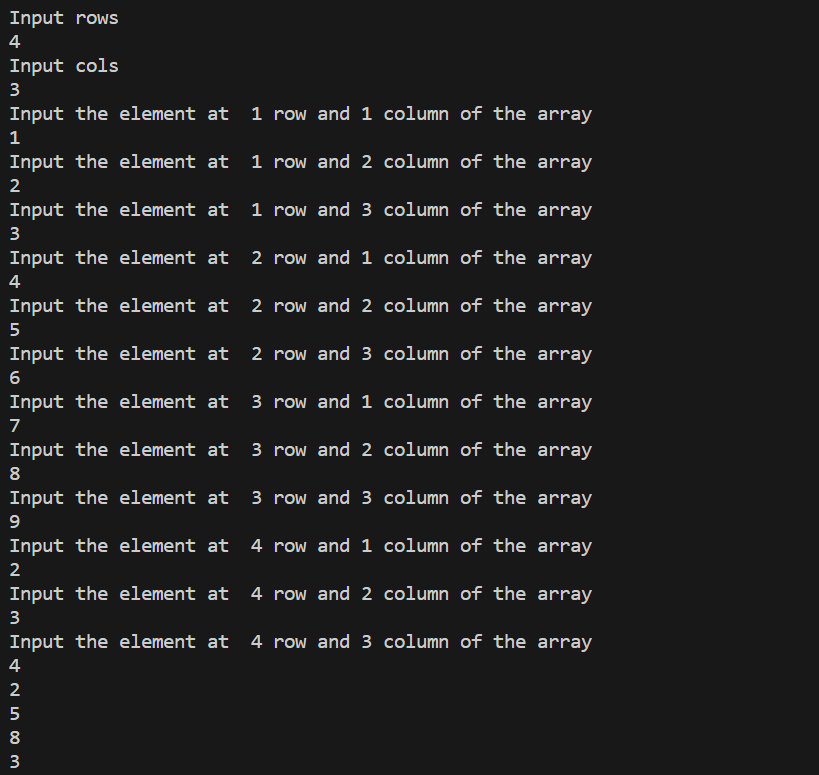
    delete[] arr;

    delete[] avg;

    return 0;

}

**Output:**

****

**Q.3**

#include<iostream>

#include<algorithm>

using namespace std;

class Array2D {

private:

    int rows;

    int cols;

    int \*\*arr;

public:

    Array2D(int r, int c) : rows(r), cols(c) {

        arr = new int\*[rows];

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

            arr[i] = new int[cols];

        }

    }

    ~Array2D() {

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

            delete[] arr[i];

        }

        delete[] arr;

    }

    int& operator()(int i, int j) {

        if (i < 0 || i >= rows || j < 0 || j >= cols) {

            throw out\_of\_range("Boundary Error");

        }

        return arr[i][j];

    }

    void transpose() {

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

            for (int j = i + 1; j < cols; ++j) {

                swap(arr[i][j], arr[j][i]);

            }

        }

    }

    void print() {

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

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

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

            }

            cout << endl;

        }

    }

};

int main() {

    int rows, cols;

    cout << "Input rows" << endl;

    cin >> rows;

    cout << "Input cols" << endl;

    cin >> cols;

    Array2D arr2D(rows, cols);

    for(int i = 0; i < rows; i++) {

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

            cout << "Input the element at " << i + 1 << " row and " << j + 1 << " column of the array" << endl;

            cin >> arr2D(i, j);

        }

    }

    cout << "Original Array" << endl;

    arr2D.print();

    arr2D.transpose();

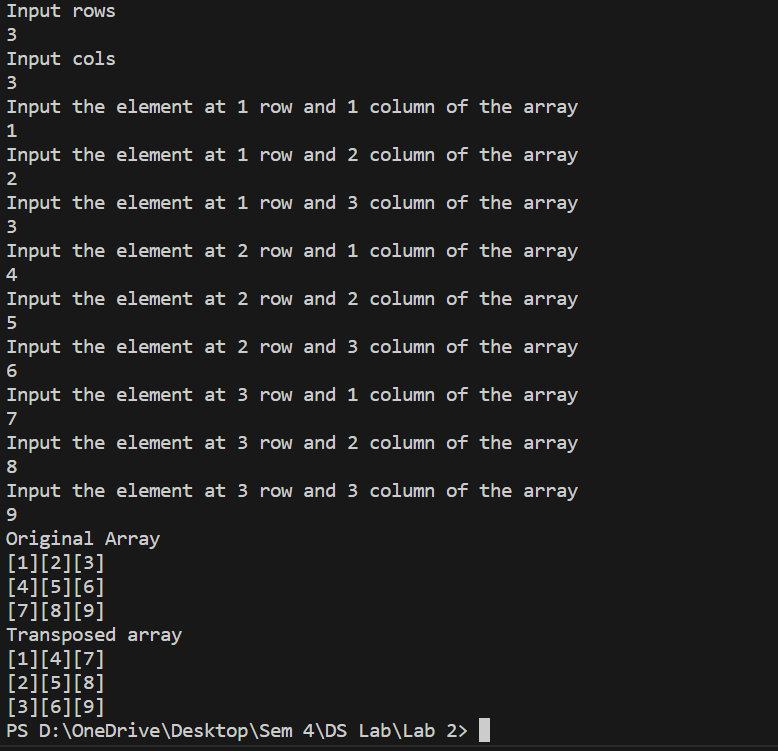
    cout << "Transposed array" << endl;

    arr2D.print();

    return 0;

}

**Output:**

****

**Q.4**

#include<iostream>

using namespace std;

int main(){

    int n;

    cout << "Enter n" << endl;

    cin >> n;

    int \*\*arr = new int\*[n];

    float \*avg = new float[n];

    int \*size = new int[n];

    // int size[10];

    // float avg[n];

    for (int i=0;i<n;i++){

        cout<<"Input row "<<i+1<< " size: ";

        cin>>size[i];

        arr[i] =new int[size[i]];

    }

    for(int i=0;i<n;i++){

        for(int j=0;j<size[i];j++){

            cout<<"Enter row " <<i+1<<" elements: ";

            cin>>arr[i][j];

    }

    }

    for (int i=0;i<n;i++){

        avg[i]=0;

        for (int j=0;j<size[i];j++){

            avg[i]+=arr[i][j];

        }

        avg[i]=avg[i]/size[i];

    }

    for(int i=0;i<n;i++){

        cout << avg[i] << " ";

    }

    for (int i = 0; i < n; ++i) {

        delete[] arr[i];

    }

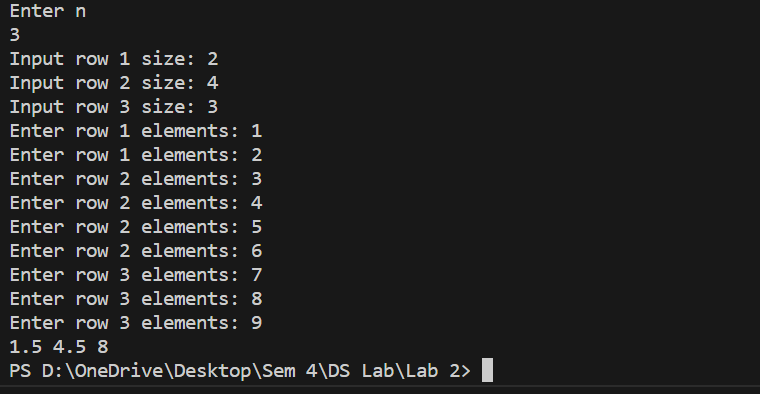
    delete[] arr;

    delete[] avg;

    return 0;

}

**Output:**

****

**Q.5**

#include<iostream>

using namespace std;

class friends{

private:

    bool \*\*arr;

    int row=5,col=5,i,j;

public:

    friends(){

        arr = new bool\*[row];

        for(int i = 0; i < row; ++i)

            arr[i] = new bool[col];

    }

    bool& operator()(int i, int j) {

        if (i < 0 || i >= row || j < 0 || j >= col) {

            throw out\_of\_range("Boundary Error");

        }

        return arr[i][j];

    }

    ~friends() {

        for(int i = 0; i < row; ++i)

            delete[] arr[i];

        delete[] arr;

    }

};

int main(){

    int key;

    friends f;

    for (int i=0;i<5;i++){

        for (int j=0;j<5;j++){

            cout << "Enter 1 if row " << i << " and col " << j << " are friends" << endl;

            cin >> key;

            if (key==1){

                f(i,j)=true;

            }

            else{

                f(i,j)=NULL;

            }

            if (f(i,j)==true){

                f(i,j)="\*";

            }

        }

    }

    for(int i = 0; i < 5; ++i) {

        for(int j = 0; j < 5; ++j) {

            cout << f(i, j) << " ";

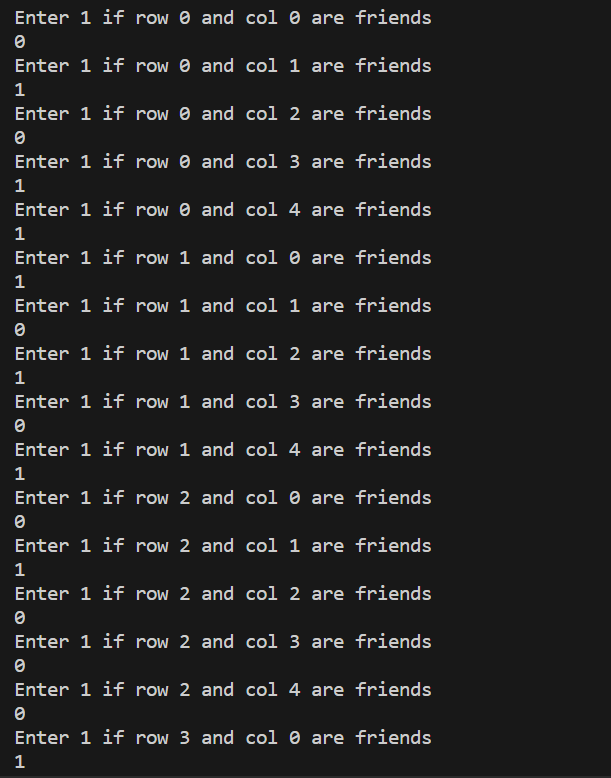
        }

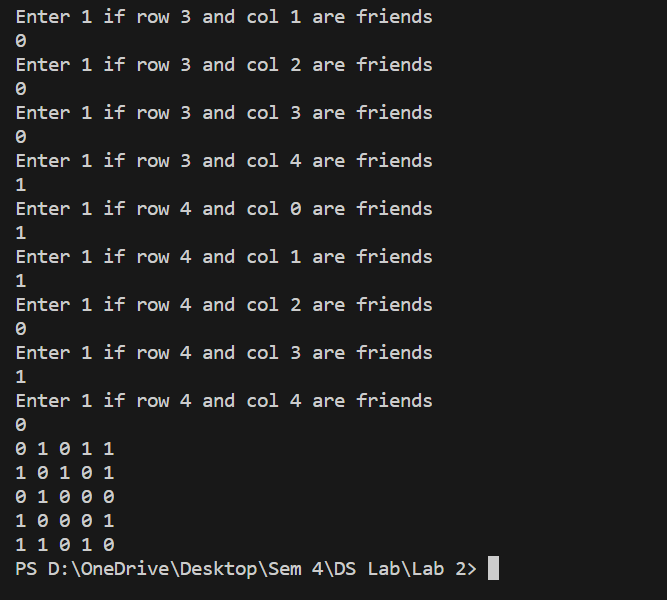
        cout << endl;

    }

}

**Output:**

****

****

**Q.6**

These are the potential pitfalls if the Rule of Three is not adhered to:

**Memory Leaks**: If dynamic memory allocation is used internally within the class, and objects are copied or assigned without properly managing this memory, memory leaks can occur.

**Resource Leaks**: For resources like database connections, if the connection is not properly closed or released when objects are copied or assigned, it can lead to resource leaks, exhausting system resources over time.

**Undefined Behavior**: Without proper copy constructors and assignment operators, shallow copies may be made, leading to multiple objects managing the same resource. This can result in undefined behavior when one object is destroyed or when modifications are made to the resource.

#include <iostream>

#include <string>

using namespace std;

class DatabaseConnection {

private:

    int\* Connection;

    void closeConnection() {

        if (Connection != nullptr) {

            cout << "Closing connection" << endl;

            delete Connection;

            Connection = nullptr;

        }

    }

public:

    DatabaseConnection() : Connection(new int(0)) {

        cout << "Connection established.\n";

        }

    DatabaseConnection(const DatabaseConnection& other) : Connection(new int(\*other.Connection)) {

        cout << "Connection copied.\n";

    }

    DatabaseConnection& operator=(const DatabaseConnection& other) {

        cout << "Connection assigned.\n";

        if (this != &other) {

            closeConnection();

            Connection = new int(\*other.Connection);

        }

        return \*this;

    }

    ~DatabaseConnection() {

        closeConnection();

        cout << "Connection destroyed.\n";

    }

};

int main() {

    DatabaseConnection c1;

    DatabaseConnection c2 = c1;

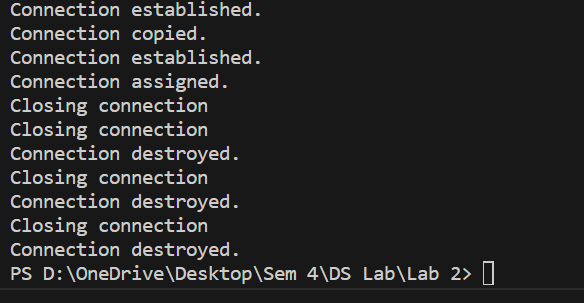
    DatabaseConnection c3;

    c3 = c2;

    return 0;

}

**Output:**



**Q.7**

#include <iostream>

#include <cstring>

using namespace std;

class Book {

private:

    char\* title\_;

    char\* author\_;

public:

    Book(const char\* title, const char\* author) : title\_(nullptr), author\_(nullptr) {

        title\_ = new char[strlen(title) + 1];

        strcpy(title\_, title);

        author\_ = new char[strlen(author) + 1];

        strcpy(author\_, author);

        cout << "Constructor called: " << title\_ << " by " << author\_ << endl;

    }

    Book(const Book& other) : title\_(nullptr), author\_(nullptr) {

        title\_ = new char[strlen(other.title\_) + 1];

        strcpy(title\_, other.title\_);

        author\_ = new char[strlen(other.author\_) + 1];

        strcpy(author\_, other.author\_);

        cout << "Copy constructor called: " << title\_ << " by " << author\_ << endl;

    }

    Book& operator=(const Book& other) {

        if (this != &other) {

            delete[] title\_;

            delete[] author\_;

            title\_ = new char[strlen(other.title\_) + 1];

            strcpy(title\_, other.title\_);

            author\_ = new char[strlen(other.author\_) + 1];

            strcpy(author\_, other.author\_);

            cout << "Copy assignment operator called: " << title\_ << " by " << author\_ << endl;

        }

        return \*this;

    }

    ~Book() {

        delete[] title\_;

        delete[] author\_;

        cout << "Destructor called" << endl;

    }

};

int main() {

    Book book1("Career Elevator", "Shozab Mehdi");

    Book book2 = book1;

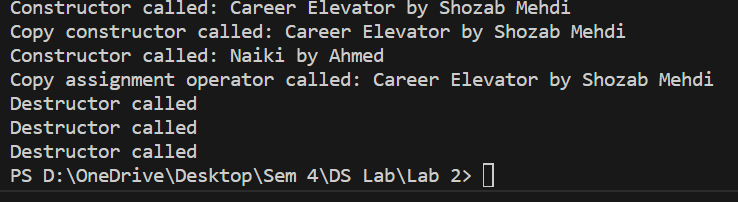
    Book book3("Naiki", "Ahmed");

    book3 = book2;

    return 0;

}

**Output:**

****