2025

Assignment

WEEK 2

TARUN TOM – B230589EC

Q1

```cpp
#include <iostream>
#include <string>

class Peripheral
{
public:
    std::string name;
    double price;
    int sold;

    Peripheral(std::string n, double p) : name(n), price(p), sold(0) {}

    void sell(int quantity){
        sold += quantity;
    }

    double totalSales() const
    {
        return sold * price;
    }
};

class Warehouse
{
private:
    Peripheral keyboard;
    Peripheral mouse;
    Peripheral monitor;
    double salesTarget;

public:
    Warehouse(double target)
        : keyboard("Keyboard", 20.0),
          mouse("Mouse", 15.0),
          monitor("Monitor", 100.0),
          salesTarget(target) {}

    void sellPeripheral(const std::string &name, int quantity)
    {
        if (name == "Keyboard")
        {
            keyboard.sell(quantity);
        }
        else if (name == "Mouse")
        {
            mouse.sell(quantity);
```

```
        }
        else if (name == "Monitor")
        {
            monitor.sell(quantity);
        }
        else
        {
            std::cout << "Unknown peripheral: " << name << std::endl;
        }
    }

    void printSalesReport() const
    {
        double totalSales = keyboard.totalSales() + mouse.totalSales() +
monitor.totalSales();

        std::cout << "Sales Report:" << std::endl;
        std::cout << "Keyboards sold: " << keyboard.sold << ", Total: $" <<
keyboard.totalSales() << std::endl;
        std::cout << "Mice sold: " << mouse.sold << ", Total: $" <<
mouse.totalSales() << std::endl;
        std::cout << "Monitors sold: " << monitor.sold << ", Total: $" <<
monitor.totalSales() << std::endl;
        std::cout << "Total Sales: $" << totalSales << std::endl;

        if (totalSales >= salesTarget)
        {
            std::cout << "Target achieved!" << std::endl;
        }
        else
        {
            std::cout << "Target not achieved." << std::endl;
        }
    }
};

int main()
{
    // Set a sales target
    Warehouse warehouse(1000.0);

    // Simulate sales
    warehouse.sellPeripheral("Keyboard", 30);
    warehouse.sellPeripheral("Mouse", 50);
    warehouse.sellPeripheral("Monitor", 10);

    // Print the sales report
    warehouse.printSalesReport();
```

```
    int a;
    std::cin >> a;

    return 0;
}
```

Output

```
Sales Report:
Keyboards sold: 30, Total: Rs600
Mice sold: 50, Total: Rs750
Monitors sold: 10, Total: Rs1000
Total Sales: Rs2350
Target achieved!
```

Q2

```
#include <iostream>
#include <string>

class Employee
{
private:
    std::string name;
    std::string id_number;
    double salary;
    int total_leaves;
    int remaining_leaves;

public:
    Employee(const std::string &name, const std::string &id_number, double
salary, int total_leaves)
        : name(name), id_number(id_number), salary(salary),
total_leaves(total_leaves), remaining_leaves(total_leaves) {}

    void apply_leave(int leave_days)
    {
        if (leave_days <= remaining_leaves)
        {
            remaining_leaves -= leave_days;
            std::cout << leave_days << " days of leave approved for " << name
<< ". Remaining leaves: " << remaining_leaves << std::endl;
        }
        else
        {
```

```
            std::cout << "Leave request denied for " << name << ". Not enough
leave balance." << std::endl;
        }
    }

    void annual_increment(double increment_amount)
    {
        salary += increment_amount;
        std::cout << "Annual increment applied for " << name << ". New salary:
" << salary << std::endl;
    }

    void display_info() const
    {
        std::cout << "Employee Name: " << name << std::endl;
        std::cout << "ID Number: " << id_number << std::endl;
        std::cout << "Salary: " << salary << std::endl;
        std::cout << "Total Leaves: " << total_leaves << std::endl;
        std::cout << "Remaining Leaves: " << remaining_leaves << std::endl;
    }
};

int main()
{
    // Creating employee objects
    Employee emp1("John Doe", "E001", 50000, 20);
    Employee emp2("Jane Smith", "E002", 60000, 15);

    // Displaying initial employee information
    emp1.display_info();
    std::cout << "-" << std::endl;
    emp2.display_info();
    std::cout << std::string(20, '-') << std::endl;

    // Applying leaves
    emp1.apply_leave(5);
    emp2.apply_leave(16);
    std::cout << std::string(20, '-') << std::endl;

    // Displaying employee information after leave
    emp1.display_info();
    std::cout << "-" << std::endl;
    emp2.display_info();
    std::cout << std::string(20, '-') << std::endl;

    // Applying annual increment
    emp1.annual_increment(5000);
    emp2.annual_increment(4000);
```

```
    std::cout << std::string(20, '-') << std::endl;

    // Displaying final employee information
    emp1.display_info();
    std::cout << "-" << std::endl;
    emp2.display_info();

    return 0;
}
```

Output

```
Salary: 50000
Total Leaves: 20
Remaining Leaves: 15
-
Employee Name: Jane Smith
ID Number: E002
Salary: 60000
Total Leaves: 15
Remaining Leaves: 15
--------------------
Annual increment applied for John Doe. New salary: 55000
Annual increment applied for Jane Smith. New salary: 64000
--------------------
Employee Name: John Doe
ID Number: E001
Salary: 55000
Total Leaves: 20
Remaining Leaves: 15
-
Employee Name: Jane Smith
ID Number: E002
Salary: 64000
Total Leaves: 15
Remaining Leaves: 15
```

Q3_a

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

class PersonalInfo {
public:
    string name;
    string dob;
    string bloodGroup;
    float height;
    float weight;
    string address;
```

```
    string phoneNumber;

    PersonalInfo(string name, string dob, string bloodGroup, float height,
float weight, string address, string phoneNumber) {
        this->name = name;
        this->dob = dob;
        this->bloodGroup = bloodGroup;
        this->height = height;
        this->weight = weight;
        this->address = address;
        this->phoneNumber = phoneNumber;
    }

    void display() {
        cout << "Name: " << name << endl;
        cout << "DOB: " << dob << endl;
        cout << "Blood Group: " << bloodGroup << endl;
        cout << "Height: " << height << endl;
        cout << "Weight: " << weight << endl;
        cout << "Address: " << address << endl;
        cout << "Phone Number: " << phoneNumber << endl;
        cout << "-------------------------" << endl;
    }
};

class Database {
private:
    vector<PersonalInfo> records;

public:
    void buildMasterTable(vector<PersonalInfo> initialRecords) {
        records = initialRecords;
    }

    void listTable() {
        for (auto& record : records) {
            record.display();
        }
    }

    void insertEntry(PersonalInfo newRecord) {
        records.push_back(newRecord);
    }

    void editEntry(string nameToEdit, PersonalInfo updatedRecord) {
        for (auto& record : records) {
            if (record.name == nameToEdit) {
                record = updatedRecord;
```

```
                cout << "Record updated successfully!" << endl;
                return;
            }
        }
        cout << "Record not found!" << endl;
    }

    void searchRecord(string nameToSearch) {
        for (auto& record : records) {
            if (record.name == nameToSearch) {
                record.display();
                return;
            }
        }
        cout << "Record not found!" << endl;
    }

    void sortEntries() {
        sort(records.begin(), records.end(), [](PersonalInfo a, PersonalInfo
b) {
            return a.name < b.name;
            });
        cout << "Records sorted by name!" << endl;
    }
};

int main() {
    Database db;

    // Sample records
    vector<PersonalInfo> initialRecords = {
        PersonalInfo("Alice", "1995-06-15", "A+", 5.5, 55.0, "123 Street,
City", "1234567890"),
        PersonalInfo("Bob", "1992-03-22", "B+", 5.8, 70.0, "456 Avenue, Town",
"9876543210")
    };

    db.buildMasterTable(initialRecords);

    int choice;
    do {
        cout << "\n1. List Table" << endl;
        cout << "2. Insert New Entry" << endl;
        cout << "3. Edit Entry" << endl;
        cout << "4. Search Record" << endl;
        cout << "5. Sort Entries" << endl;
        cout << "6. Exit" << endl;
        cout << "Enter your choice: ";
```

```
        cin >> choice;

        switch (choice) {
        case 1:
            db.listTable();
            break;
        case 2: {
            string name, dob, bloodGroup, address, phoneNumber;
            float height, weight;
            cout << "Enter Name: "; cin.ignore(); getline(cin, name);
            cout << "Enter DOB: "; getline(cin, dob);
            cout << "Enter Blood Group: "; getline(cin, bloodGroup);
            cout << "Enter Height: "; cin >> height;
            cout << "Enter Weight: "; cin >> weight;
            cin.ignore();
            cout << "Enter Address: "; getline(cin, address);
            cout << "Enter Phone Number: "; getline(cin, phoneNumber);

            db.insertEntry(PersonalInfo(name, dob, bloodGroup, height, weight,
address, phoneNumber));
            break;
        }
        case 3: {
            string nameToEdit;
            cout << "Enter the name of the record to edit: ";
            cin.ignore();
            getline(cin, nameToEdit);

            string name, dob, bloodGroup, address, phoneNumber;
            float height, weight;
            cout << "Enter Updated Name: "; getline(cin, name);
            cout << "Enter Updated DOB: "; getline(cin, dob);
            cout << "Enter Updated Blood Group: "; getline(cin, bloodGroup);
            cout << "Enter Updated Height: "; cin >> height;
            cout << "Enter Updated Weight: "; cin >> weight;
            cin.ignore();
            cout << "Enter Updated Address: "; getline(cin, address);
            cout << "Enter Updated Phone Number: "; getline(cin, phoneNumber);

            db.editEntry(nameToEdit, PersonalInfo(name, dob, bloodGroup,
height, weight, address, phoneNumber));
            break;
        }
        case 4: {
            string nameToSearch;
            cout << "Enter the name of the record to search: ";
            cin.ignore();
            getline(cin, nameToSearch);
```

```
            db.searchRecord(nameToSearch);
            break;
        }
        case 5:
            db.sortEntries();
            break;
        case 6:
            cout << "Exiting program." << endl;
            break;
        default:
            cout << "Invalid choice! Try again." << endl;
        }
    } while (choice != 6);

    return 0;
}
```

Output

```
1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
Enter your choice: 1
Name: Alice
DOB: 1995-06-15
Blood Group: A+
Height: 5.5
Weight: 55
Address: 123 Street, City
Phone Number: 1234567890
------------------------
Name: Bob
DOB: 1992-03-22
Blood Group: B+
Height: 5.8
Weight: 70
Address: 456 Avenue, Town
Phone Number: 9876543210
------------------------

1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
Enter your choice: 2
Enter Name: Tarun
Enter DOB: 2004-10-04
Enter Blood Group: O+
Enter Height: 5.10
Enter Weight: 70
Enter Address: sajhdhlascap
Enter Phone Number: 9446126519

1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
Enter your choice: 5
Records sorted by name!

1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
Enter your choice: 4
Enter the name of the record to
search: Tarun
Name: Tarun
DOB: 2004-10-04
Blood Group: O+
Height: 5.1
Weight: 70
Address: sajhdhlascap
Phone Number: 9446126519
```

Q3_b

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

class Book {
public:
    int accessionNumber;
    string authorName;
    string title;
    int publicationYear;
    double cost;

    Book(int accessionNumber, string authorName, string title, int
publicationYear, double cost) {
        this->accessionNumber = accessionNumber;
        this->authorName = authorName;
        this->title = title;
        this->publicationYear = publicationYear;
        this->cost = cost;
    }

    void display() {
        cout << "Accession Number: " << accessionNumber << endl;
        cout << "Author Name: " << authorName << endl;
        cout << "Title: " << title << endl;
        cout << "Publication Year: " << publicationYear << endl;
        cout << "Cost: " << cost << endl;
        cout << "-------------------------" << endl;
    }
};

class LibraryDatabase {
private:
    vector<Book> records;

public:
    void buildMasterTable(vector<Book> initialRecords) {
        records = initialRecords;
    }

    void listTable() {
        for (auto& record : records) {
            record.display();
```

```
        }
    }

    void insertEntry(Book newBook) {
        records.push_back(newBook);
    }

    void editEntry(int accessionNumberToEdit, Book updatedBook) {
        for (auto& record : records) {
            if (record.accessionNumber == accessionNumberToEdit) {
                record = updatedBook;
                cout << "Record updated successfully!" << endl;
                return;
            }
        }
        cout << "Record not found!" << endl;
    }

    void searchRecord(int accessionNumberToSearch) {
        for (auto& record : records) {
            if (record.accessionNumber == accessionNumberToSearch) {
                record.display();
                return;
            }
        }
        cout << "Record not found!" << endl;
    }

    void sortEntries() {
        sort(records.begin(), records.end(), [](Book a, Book b) {
            return a.accessionNumber < b.accessionNumber;
            });
        cout << "Records sorted by accession number!" << endl;
    }
};

int main() {
    LibraryDatabase db;

    // Sample records
    vector<Book> initialRecords = {
        Book(1001, "Author A", "Title A", 2005, 250.50),
        Book(1002, "Author B", "Title B", 2010, 300.75)
    };

    db.buildMasterTable(initialRecords);

    int choice;
```

```
    do {
        cout << "\n1. List Table" << endl;
        cout << "2. Insert New Entry" << endl;
        cout << "3. Edit Entry" << endl;
        cout << "4. Search Record" << endl;
        cout << "5. Sort Entries" << endl;
        cout << "6. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            db.listTable();
            break;
        case 2: {
            int accessionNumber, publicationYear;
            string authorName, title;
            double cost;
            cout << "Enter Accession Number: "; cin >> accessionNumber;
            cin.ignore();
            cout << "Enter Author Name: "; getline(cin, authorName);
            cout << "Enter Title: "; getline(cin, title);
            cout << "Enter Publication Year: "; cin >> publicationYear;
            cout << "Enter Cost: "; cin >> cost;

            db.insertEntry(Book(accessionNumber, authorName, title,
publicationYear, cost));
            break;
        }
        case 3: {
            int accessionNumberToEdit;
            cout << "Enter the accession number of the record to edit: ";
            cin >> accessionNumberToEdit;

            int accessionNumber, publicationYear;
            string authorName, title;
            double cost;
            cout << "Enter Updated Accession Number: "; cin >>
accessionNumber;
            cin.ignore();
            cout << "Enter Updated Author Name: "; getline(cin, authorName);
            cout << "Enter Updated Title: "; getline(cin, title);
            cout << "Enter Updated Publication Year: "; cin >>
publicationYear;
            cout << "Enter Updated Cost: "; cin >> cost;

            db.editEntry(accessionNumberToEdit, Book(accessionNumber,
authorName, title, publicationYear, cost));
```

```
            break;
        }
        case 4: {
            int accessionNumberToSearch;
            cout << "Enter the accession number of the record to search: ";
            cin >> accessionNumberToSearch;

            db.searchRecord(accessionNumberToSearch);
            break;
        }
        case 5:
            db.sortEntries();
            break;
        case 6:
            cout << "Exiting program." << endl;
            break;
        default:
            cout << "Invalid choice! Try again." << endl;
        }
    } while (choice != 6);

    return 0;
}
```

Output

```
1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
Enter your choice: 1
Accession Number: 1001
Author Name: Author A
Title: Title A
Publication Year: 2005
Cost: 250.5
------------------------
Accession Number: 1002
Author Name: Author B
Title: Title B
Publication Year: 2010
Cost: 300.75
------------------------

1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
```

```
Enter your choice: 2
Enter Accession Number: 1234
Enter Author Name: Somebody
Enter Title: Something
Enter Publication Year: 2221
Enter Cost: 1000

1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
Enter your choice: 5
Records sorted by accession number!

1. List Table
2. Insert New Entry
3. Edit Entry
4. Search Record
5. Sort Entries
6. Exit
Enter your choice: 1
Accession Number: 1001
Author Name: Author A
Title: Title A
```

```
Publication Year: 2005
Cost: 250.5
------------------------
Accession Number: 1002
Author Name: Author B
Title: Title B
Publication Year: 2010
Cost: 300.75
```

```
------------------------
Accession Number: 1234
Author Name: Somebody
Title: Something
Publication Year: 2221
Cost: 1000
------------------------
```

Q4

```cpp
#include <iostream>
using namespace std;

// Base class for polygons
class Polygon {
protected:
    double length, breadth; // Dimensions of the polygon
public:
    // Constructor
    Polygon(double l = 0, double b = 0) : length(l), breadth(b) {}

    // Virtual functions for area and perimeter
    virtual double area() const = 0;
    virtual double perimeter() const = 0;

    // Virtual destructor
    virtual ~Polygon() {}
};

// Derived class for Rectangle
class Rectangle : public Polygon {
public:
    // Constructor
    Rectangle(double l, double b) : Polygon(l, b) {}

    // Overridden function to calculate area
    double area() const override {
        return length * breadth;
    }

    // Overridden function to calculate perimeter
    double perimeter() const override {
        return 2 * (length + breadth);
    }
};

// Function to calculate charges
```

```
void calculateCharges(Polygon* polygon, double fencingCostPerUnit, double
lawnCostPerUnit) {
    double perimeter = polygon->perimeter();
    double area = polygon->area();

    double fencingCost = perimeter * fencingCostPerUnit;
    double lawnCost = area * lawnCostPerUnit;

    cout << "Fencing Cost: $" << fencingCost << endl;
    cout << "Lawn Laying Cost: $" << lawnCost << endl;
}

int main() {
    // Input dimensions of the rectangle
    double length, breadth;
    cout << "Enter the length and breadth of the rectangle: ";
    cin >> length >> breadth;

    // Input costs
    double fencingCostPerUnit, lawnCostPerUnit;
    cout << "Enter the cost per unit for fencing: ";
    cin >> fencingCostPerUnit;
    cout << "Enter the cost per unit for laying lawn: ";
    cin >> lawnCostPerUnit;

    // Create a Rectangle object
    Rectangle rect(length, breadth);

    // Calculate and display charges
    calculateCharges(&rect, fencingCostPerUnit, lawnCostPerUnit);

    return 0;
}
```

Output

```
Enter the length and breadth of the rectangle: 100
200
Enter the cost per unit for fencing: 100
Enter the cost per unit for laying lawn: 200
Fencing Cost: Rs60000
Lawn Laying Cost: Rs4e+06
```

Set 2

Q1

```cpp
#include <iostream>
#include <string>

using namespace std;

// Class definition for Student
class Student
{
private:
    string name;      // Student name
    int rollNo;       // Roll number
    float totalMarks; // Total marks obtained

public:
    // Method to input student details
    void inputDetails()
    {
        cout << "Enter student name: ";
        getline(cin, name);
        cout << "Enter roll number: ";
        cin >> rollNo;
        cout << "Enter total marks obtained: ";
        cin >> totalMarks;
        cin.ignore(); // Clear input buffer for next getline
    }

    // Method to display student details
    void displayDetails() const
    {
        cout << "\nStudent Details:" << endl;
        cout << "Name: " << name << endl;
        cout << "Roll Number: " << rollNo << endl;
        cout << "Total Marks: " << totalMarks << endl;
    }
};

int main()
{
    // Create a Student object
    Student student;

    // Input and display details
    student.inputDetails();
    student.displayDetails();

    return 0;
```

```
}
```

Output

```
Enter student name: ajsh
Enter roll number: 5464
Enter total marks obtained: 20

Student Details:
Name: ajsh
Roll Number: 5464
Total Marks: 20
```

Q2

```
#include <iostream>
#include <cmath>

class Triangle
{
private:
    double side1, side2, side3;

public:
    // Constructor to initialize the sides of the triangle
    Triangle(double s1, double s2, double s3)
    {
        side1 = s1;
        side2 = s2;
        side3 = s3;
    }

    // Function to calculate and print the perimeter
    void printPerimeter()
    {
        double perimeter = side1 + side2 + side3;
        std::cout << "Perimeter: " << perimeter << " units" << std::endl;
    }

    // Function to calculate and print the area
    void printArea()
    {
        double s = (side1 + side2 + side3) /
2;                              // Semi-perimeter
```

```
        double area = std::sqrt(s * (s - side1) * (s - side2) * (s - side3));
// Heron's formula
        std::cout << "Area: " << area << " square units" << std::endl;
    }
};

int main()
{
    // Create a Triangle object with sides 3, 4, and 5
    Triangle triangle(3, 4, 5);

    // Print the perimeter and area
    triangle.printPerimeter();
    triangle.printArea();

    return 0;
}
```

Output

```
Perimeter: 12 units
Area: 6 square units
```

Q3

```
#include <iostream>
using namespace std;

// Class to represent a complex number
class Complex
{
private:
    double real;
    double imag;

public:
    // Constructor
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    // Overload the '+' operator for addition
    Complex operator+(const Complex &other)
    {
        return Complex(real + other.real, imag + other.imag);
    }

    // Overload the '-' operator for subtraction
    Complex operator-(const Complex &other)
```

```
    {
        return Complex(real - other.real, imag - other.imag);
    }

    // Overload the '*' operator for multiplication
    Complex operator*(const Complex &other)
    {
        return Complex(real * other.real - imag * other.imag,
                       real * other.imag + imag * other.real);
    }

    // Function to display the complex number
    void display() const
    {
        cout << real << (imag >= 0 ? " + " : " - ") << abs(imag) << "i" <<
endl;
    }
};

int main()
{
    double real1, imag1, real2, imag2;

    cout << "Enter the real and imaginary parts of the first complex number:
";
    cin >> real1 >> imag1;

    cout << "Enter the real and imaginary parts of the second complex number:
";
    cin >> real2 >> imag2;

    // Create two Complex objects
    Complex c1(real1, imag1);
    Complex c2(real2, imag2);

    // Perform operations
    Complex sum = c1 + c2;
    Complex difference = c1 - c2;
    Complex product = c1 * c2;

    cout << "Sum: ";
    sum.display();

    cout << "Difference: ";
    difference.display();

    cout << "Product: ";
    product.display();
```

```
    return 0;
}
```

Output

```
Enter the real and imaginary parts of the first complex number: 20 10
Enter the real and imaginary parts of the second complex number: 11 12
Sum: 31 + 22i
Difference: 9 - 2i
Product: 100 + 350i
```

Q5

```
#include <iostream>
#include <vector>
using namespace std;

class Matrix {
private:
    int rows;
    int cols;
    vector<vector<int>> elements;

public:
    // Constructor to initialize rows, columns, and allocate space for the
matrix
    Matrix(int r, int c) : rows(r), cols(c), elements(r, vector<int>(c, 0)) {}

    // Get the number of rows
    int getRows() const {
        return rows;
    }

    // Get the number of columns
    int getCols() const {
        return cols;
    }

    // Set the elements of the matrix at a given position (i, j)
    void setElement(int i, int j, int value) {
        if (i >= 0 && i < rows && j >= 0 && j < cols) {
            elements[i][j] = value;
        }
        else {
            cerr << "Index out of bounds" << endl;
        }
```

```
    }

    // Get the element at a given position (i, j)
    int getElement(int i, int j) const {
        if (i >= 0 && i < rows && j >= 0 && j < cols) {
            return elements[i][j];
        }
        else {
            cerr << "Index out of bounds" << endl;
            return -1;
        }
    }

    // Add two matrices
    Matrix add(const Matrix& other) const {
        if (rows != other.rows || cols != other.cols) {
            cerr << "Matrix dimensions do not match for addition" << endl;
            return Matrix(0, 0);
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                result.setElement(i, j, elements[i][j] +
other.elements[i][j]);
            }
        }

        return result;
    }

    // Multiply two matrices
    Matrix multiply(const Matrix& other) const {
        if (cols != other.rows) {
            cerr << "Matrix dimensions do not match for multiplication" <<
endl;
            return Matrix(0, 0);
        }

        Matrix result(rows, other.cols);
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < other.cols; ++j) {
                int sum = 0;
                for (int k = 0; k < cols; ++k) {
                    sum += elements[i][k] * other.elements[k][j];
                }
                result.setElement(i, j, sum);
            }
```

```
        }

        return result;
    }

    // Display the matrix
    void display() const {
        for (const auto& row : elements) {
            for (const auto& elem : row) {
                cout << elem << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    // Example usage of the Matrix class
    Matrix mat1(2, 3);
    Matrix mat2(2, 3);

    // Initialize mat1
    mat1.setElement(0, 0, 1);
    mat1.setElement(0, 1, 2);
    mat1.setElement(0, 2, 3);
    mat1.setElement(1, 0, 4);
    mat1.setElement(1, 1, 5);
    mat1.setElement(1, 2, 6);

    // Initialize mat2
    mat2.setElement(0, 0, 7);
    mat2.setElement(0, 1, 8);
    mat2.setElement(0, 2, 9);
    mat2.setElement(1, 0, 10);
    mat2.setElement(1, 1, 11);
    mat2.setElement(1, 2, 12);

    // Add matrices
    Matrix sum = mat1.add(mat2);

    // Display the result
    cout << "Matrix 1:" << endl;
    mat1.display();
    cout << "Matrix 2:" << endl;
    mat2.display();
    cout << "Sum of matrices:" << endl;
    sum.display();
```

```
    // Multiply matrices (example with compatible dimensions)

    Matrix mat3(3, 2);
    mat3.setElement(0, 0, 1);
    mat3.setElement(0, 1, 2);
    mat3.setElement(1, 0, 3);
    mat3.setElement(1, 1, 4);
    mat3.setElement(2, 0, 5);
    mat3.setElement(2, 1, 6);
    cout << "Matrix 3:" << endl;
    mat3.display();

    Matrix product = mat1.multiply(mat3);
    cout << "Product of matrices 1 and 3:" << endl;
    product.display();

    return 0;
}
```

Output

```
Matrix 1:
1 2 3
4 5 6
Matrix 2:
7 8 9
10 11 12
Sum of matrices:
8 10 12
14 16 18
Matrix 3:
1 2
3 4
5 6
Product of matrices 1 and 3:
22 28
49 64
```

Q6

```
#include <iostream>
#include <string>
using namespace std;

class REPORT {
private:
```

```
    int adno; //admission number
    char name[21]; // 20 characters + 1 for null terminator
    float marks[5]; // Array of 5 floating point values
    float average; // Average marks obtained

    // Private function to compute the average of marks
    void GETAVG() {
        float sum = 0;
        for (int i = 0; i < 5; ++i) {
            sum += marks[i];
        }
        average = sum / 5;
    }

public:
    // Function to accept values for adno, name, and marks
    void READINFO() {
        cout << "Enter 4-digit admission number: ";
        cin >> adno;
        cin.ignore(); // Clear the input buffer

        cout << "Enter name (max 20 characters): ";
        cin.getline(name, 21);

        cout << "Enter marks for 5 subjects: \n";
        for (int i = 0; i < 5; ++i) {
            cout << "Mark " << (i + 1) << ": ";
            cin >> marks[i];
        }

        // Compute the average marks
        GETAVG();
    }

    // Function to display all data members
    void DISPLAYINFO() {
        cout << "\nAdmission Number: " << adno;
        cout << "\nName: " << name;
        cout << "\nMarks: ";
        for (int i = 0; i < 5; ++i) {
            cout << marks[i] << " ";
        }
        cout << "\nAverage Marks: " << average << endl;
    }
};

int main() {
    REPORT student;
```

```
    // Read information for the student
    student.READINFO();

    // Display the information
    student.DISPLAYINFO();

    return 0;
}
```

Output

```
Enter 4-digit admission number: 1234
Enter name (max 20 characters): mnbvc
Enter marks for 5 subjects:
Mark 1: 20
Mark 2: 21
Mark 3: 22
Mark 4: 23
Mark 5: 24

Admission Number: 1234
Name: mnbvc
Marks: 20 21 22 23 24
Average Marks: 22
```

Q7

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Movie {
private:
    string title;
    string studio;
    string rating;

public:
    // Constructor that takes title, studio, and rating
    Movie(string t, string s, string r) : title(t), studio(s), rating(r) {}

    // Constructor that takes title and studio, sets rating to "PG"
    Movie(string t, string s) : title(t), studio(s), rating("PG") {}

    // Method to filter movies with "PG" rating
    static vector<Movie> getPG(const vector<Movie>& movies) {
```

```
        vector<Movie> pgMovies;
        for (const Movie& movie : movies) {
            if (movie.rating == "PG") {
                pgMovies.push_back(movie);
            }
        }
        return pgMovies;
    }

    // Method to display movie details (for testing purposes)
    void display() const {
        cout << "Title: " << title << ", Studio: " << studio << ", Rating: "
<< rating << endl;
    }
};

int main() {
    // Creating an instance of the class Movie
    Movie casinoRoyale("Casino Royale", "Eon Productions", "PG13");

    // Creating additional movie instances for demonstration
    Movie movie1("Movie A", "Studio A", "PG");
    Movie movie2("Movie B", "Studio B", "R");
    Movie movie3("Movie C", "Studio C", "PG");

    // Vector of movies
    vector<Movie> movies = { casinoRoyale, movie1, movie2, movie3 };

    // Filtering movies with "PG" rating
    vector<Movie> pgMovies = Movie::getPG(movies);

    // Displaying the PG-rated movies
    cout << "PG-rated movies:" << endl;
    for (const Movie& movie : pgMovies) {
        movie.display();
    }

    return 0;
}
```

Output

```
PG-rated movies:
Title: Movie A, Studio: Studio A, Rating: PG
Title: Movie C, Studio: Studio C, Rating: PG
```