OOP Assignment 03

Task 01:

q1_anotherApproach.cpp > main()

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

int get_int() {
    int n;
    for (;;) {
        if (cin >> n) {
            return n;
        }
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid entry. Please re-enter: ";
    }
}

class Medicine {
    protected:
        vector <string> name;
        vector <string> formula;
        vector <string> manufactureDate;
        vector <string> expiryDate;
        vector <int> retailPrice;

    public:
        // Constructor
        Medicine () {
            name = {"Aspirin","Panadol","Zyrtec"};
            formula = {"C9H8O4","C8H9NO2","C21H25CIN2O3"};
            manufactureDate = {"30June2023","30July2023","30Aug2023"};
            expiryDate = {"30June2024","30July2024","30Aug2024"};
            retailPrice = {200, 350, 400};
        }

        // Setters
        bool isInIndex(int index) {
            if (index >=0 && index<name.size()) {
                return true;
            }
            return false;
        }
```

```
    void setName (int index) {
        if (isInIndex(index)) {
            string newName;
            cout<<"Enter new Medicine Name for index "<<index<<": ";
            getline(cin, newName);
            name[index] = newName;
        }
        else {
            cout<<"Invalid index number." << endl;
        }
    }
    void setFormula (int index) {
        if (isInIndex(index)) {
            string newFormula;
            cout<<"Enter new Medicine Formula for index "<<index<<": ";
            getline(cin, newFormula);
            formula[index] = newFormula;
        }
        else {
            cout<<"Invalid index number." << endl;
        }
    }
    void setManuFactureDate (int index) {
        if (isInIndex(index)) {
            string newManufactureDate;
            cout<<"Enter new Medicine ManuFacture Date for index "<<index<<": ";
            getline(cin, newManufactureDate);
            manufactureDate[index] = newManufactureDate;
        }
        else {
            cout<<"Invalid index number." << endl;
        }
    }
    void setExpiryDate (int index) {
        if (isInIndex(index)) {
            string newExpiryDate;
            cout<<"Enter new Medicine Expiry Date for index "<<index<<": ";
            getline(cin, newExpiryDate);
            expiryDate[index] = newExpiryDate;
        }
        else {
            cout<<"Invalid index number." << endl;
        }
    }
```

```cpp
class Medicine {
        void setRetailPrice (int index) {
            if (isInIndex(index)) {
                int newRetailPrice;
                cout<<"Enter new Medicine Retail Price for index "<<index<<": ";
                cin>>newRetailPrice;
                retailPrice[index] = newRetailPrice;
            }
            else {
                cout<<"Invalid index number." << endl;
            }
        }

        // Getters
        const string getName (int index) {
            return name[index];
        }
        const int getNameSize () {
            return name.size();
        }
        const string getFormula (int index) {
            return formula[index];
        }
        const string getManufactureDate (int index) {
            return manufactureDate[index];
        }
        const string getExpiryDate (int index) {
            return expiryDate[index];
        }
        const int getRetailPrice (int index) {
            return retailPrice[index];
        }

        // Display Function for All Medicines
        virtual void displayInfo () {
            for (int i=0; i<name.size(); i++) {
                cout<<"Details for Medicine #" << i+1 << endl;
                cout<<"Name: " << name[i] << endl;
                cout<<"Formula: " << formula[i] << endl;
                cout<<"Manufature Date: " << manufactureDate[i] << endl;
                cout<<"Expiry Date: " << expiryDate[i] << endl;
                cout<<endl;
            }
        }

```

```cpp
class Medicine {
        // Display Function for a Specified Medicine
        virtual void displayInfo (int index) {
            if (index >=0 && index<name.size()) {
                cout<<"Details for Medicine #" << index+1 << endl;
                cout<<"Name: " << name[index] << endl;
                cout<<"Formula: " << formula[index] << endl;
                cout<<"Manufature Date: " << manufactureDate[index] << endl;
                cout<<"Expiry Date: " << expiryDate[index] << endl;
            }
            else {
                cout<<"Invalid index number. Medicine not found." << endl;
            }
        }

        // Operator Overloading
        bool operator==(Medicine &other) {
            return expiryDate == other.expiryDate;
        }

        // Friend Classes
        friend class Pharmacist;
        friend class Counter;

};

class Tablet : public Medicine {
    protected:
        vector <int> sucroseLevel; // 0 to 1

    public:
        Tablet () : Medicine () {
            sucroseLevel = {5, 6, 8};
        }

        // Display Function for All Medicines
        void displayInfo () {
            for (int i=0; i<name.size(); i++) {
                cout<<"Details for Tablet #" << i+1 << endl;
                cout<<"Name: " << name[i] << endl;
                cout<<"Formula: " << formula[i] << endl;
                cout<<"Sucrose Level: " << sucroseLevel[i] << endl;
                cout<<"Manufature Date: " << manufactureDate[i] << endl;
                cout<<"Expiry Date: " << expiryDate[i] << endl;
            }
        }
```

```
class Tablet : public Medicine {

        // Display Function for a specified medicine
        void displayInfo (int index) {
            if (index >=0 && index<name.size()) {
                cout<<"Details for Tablet #" << index+1 << endl;
                cout<<"Name: " << name[index] << endl;
                cout<<"Formula: " << formula[index] << endl;
                cout<<"Sucrose Level: " << sucroseLevel[index] << endl;
                cout<<"Manufature Date: " << manufactureDate[index] << endl;
                cout<<"Expiry Date: " << expiryDate[index] << endl;
            }
            else {
                cout<<"Invalid index number. Tablet not found." << endl;
            }
        }

};

class Capsule : public Medicine {
    protected:
        vector <float> absorptionPercentage; // 1 to 100

    public:
        Capsule () : Medicine () {
            absorptionPercentage = {65.3, 84.4, 87.5};
        }

        // Display Function for a All Medicine
        void displayInfo () {
            for (int i=0; i<name.size(); i++) {
                cout<<"Details for Tablet #" << i+1 << endl;
                cout<<"Name: " << name[i] << endl;
                cout<<"Formula: " << formula[i] << endl;
                cout<<"Absorption Percentage: " << absorptionPercentage[i] << endl;
                cout<<"Manufature Date: " << manufactureDate[i] << endl;
                cout<<"Expiry Date: " << expiryDate[i] << endl;
            }
            cout<<endl;
        }
```

```
class Capsule : public Medicine {
        }

        // Display Function for a Specific Medicine
        void displayInfo (int index) {
            if (index >=0 && index<name.size()) {
                cout<<"Details for Tablet #" << index+1 << endl;
                cout<<"Name: " << name[index] << endl;
                cout<<"Formula: " << formula[index] << endl;
                cout<<"Absorption Percentage: " << absorptionPercentage[index] << endl;
                cout<<"Manufature Date: " << manufactureDate[index] << endl;
                cout<<"Expiry Date: " << expiryDate[index] << endl;
            }
            else {
                cout<<"Invalid index number. Tablet not found." << endl;
            }
        }
};

class Syrup : public Medicine {
    protected:
        vector <int> quantityML;

    public:
        Syrup () : Medicine () {
            quantityML = {250, 120, 500};
        }

        // Display Function for All Medicines
        void displayInfo () {
            for (int i=0; i<name.size(); i++) {
                cout<<"Details for Tablet #" << i+1 << endl;
                cout<<"Name: " << name[i] << endl;
                cout<<"Formula: " << formula[i] << endl;
                cout<<"Quantity (ML): " << quantityML[i] << endl;
                cout<<"Manufature Date: " << manufactureDate[i] << endl;
                cout<<"Expiry Date: " << expiryDate[i] << endl;
            }
        }
```

```
class Syrup : public Medicine {
        // Display Function for a Specific Medicine
        void displayInfo (int index) {
            if (index >=0 && index<name.size()) {
                cout<<"Details for Tablet #" << index+1 << endl;
                cout<<"Name: " << name[index] << endl;
                cout<<"Formula: " << formula[index] << endl;
                cout<<"Quantity (ML): " << quantityML[index] << endl;
                cout<<"Manufature Date: " << manufactureDate[index] << endl;
                cout<<"Expiry Date: " << expiryDate[index] << endl;
            }
            else {
                cout<<"Invalid index number. Tablet not found." << endl;
            }
        }
};

class Pharmacist {
    protected:
        Medicine medicine;

    public:
        void searchMedicine () {
            bool found=false;
            string formulaToFind;
            cout<<"Enter the formula of medicine you want to find: ";
            getline(cin, formulaToFind);

            for (int i=0; i<medicine.getNameSize(); i++) {
                if (formulaToFind == medicine.getFormula(i)) {
                    found=true;
                    cout<<"Medicine with formula '" << formulaToFind << "' is found." << endl;
                    cout<<"Medicine Name: " << medicine.getName(i) << endl;
                    cout<<"Medicine Formula: " << medicine.getFormula(i) << endl;
                    cout<<"Medicine Manufature Date: " << medicine.getManufactureDate(i) << endl;
                    cout<<"Medicine Expiry Date: " << medicine.getExpiryDate(i) << endl;
                }
            }
            if (!found) {
                cout<<"Medicine with formula '" << formulaToFind << "' not found." << endl;
            }
        }
```

```
class Pharmacist {

        int recommendMedicine () {
            int choice;
            cout<<"1. Fever\n2. Headache\n3. Flu\nChoose one from the above list: ";
            choice=get_int();
            cout<<"The counter has noted the syptoms/prescription and it has been forwarded to the pharmacist." << endl;

            switch (choice) {
                case 1:
                    cout<<"The pharmacist has recommended Aspirin for fever to the counter." << endl;
                    return 0;
                    break;
                case 2:
                    cout<<"The pharmacist has recommended Panadol for headache to the counter." << endl;
                    return 1;
                    break;
                case 3:
                    cout<<"The pharmacist has recommended Zyrtec for flu to the counter." << endl;
                    break;
                    return 2;
                default:
                    cout<<"Sorry, we don't have medicines other than for these symptoms." << endl;
                    return 5;
                    break;
            }
            return 5;
        }
};
```

```
class Counter {
    protected:
        Medicine medicine;
        Pharmacist pharma;
        vector <string> soldMedicineName;
        int totalRevenue=0;
    public:

        void searchMedicine () {
            bool found=false;
            string medicineToFind;
            cout<<"Enter the name of medicine you want to find: ";
            getline(cin, medicineToFind);

            for (int i=0; i<medicine.getNameSize(); i++) {
                if (medicineToFind == medicine.getName(i)) {
                    found=true;
                    cout<<"Medicine with name '" << medicineToFind << "' is found." << endl;
                    cout<<"Medicine Name: " << medicine.getName(i) << endl;
                    cout<<"Medicine Formula: " << medicine.getFormula(i) << endl;
                    cout<<"Medicine Manufature Date: " << medicine.getManufactureDate(i) << endl;
                    cout<<"Medicine Expiry Date: " << medicine.getExpiryDate(i) << endl;
                }
            }
            if (!found) {
                cout<<"Medicine with name '" << medicineToFind << "' not found." << endl;
            }
        }

        void updateRevenue (int indexToBuy) {
            totalRevenue += medicine.getRetailPrice(indexToBuy);
        }
```

```
class Counter {

        void sellMedicine () {
            // medicine.displayInfo();
            bool flag = true;
            int numToBuy, indexToBuy;
            indexToBuy = pharma.recommendMedicine();
            cin.ignore();

            if (indexToBuy==5) {
                flag = false;
            }
            // cout<<"Enter the number of medicine from the list that you want to buy: ";
            // cin>>numToBuy;

            // indexToBuy = numToBuy-1;
            if (flag) {
                if (medicine.isInIndex(indexToBuy)) {
                    updateRevenue(indexToBuy);
                    cout<<"The counter has sold '" << medicine.getName(indexToBuy) << "' for Rs "<<medicine.getRetailPrice(indexToBuy)<<" to the customer." << endl;
                    soldMedicineName.push_back(medicine.getName(indexToBuy));
                    cout<<"The total revenue of today is now Rs " << totalRevenue <<"." << endl;
                }
                else {
                    cout<<"Invalid index number." << endl;
                }
            }
        }
};
// int Counter::totalRevenue=0;
```

```cpp
int main() {

    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
    cout<<"Name: Muhammad Hammad" << endl;
    cout<<"Roll No: 23K-2005" << endl;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

    Medicine m1;
    Medicine m2;
    Pharmacist p1;
    Counter c1;

    c1.sellMedicine();

    cout<<endl<<"------- Operator Overloading -------" << endl;
    if (m1 == m2) {
        cout<<"The two medicines have the same expiry date." << endl << endl;
    } else {
        cout<<"The two medicines do not have the same expiry date." << endl<< endl;
    }

    /*
    As required in the question, the counter collects the prescription from the customer.
    Counter then forwards the prescription to Pharmacist and Pharmacist suggests a medicine based on the prescription.
    The Counter then sells the medicine recommended by the Pharmacist and collects the money.
    Once the medicine is sold and payment is received, the total Revenue is updated.
    */

    // p1.searchMedicine(); -- Working fine
    // c1.searchMedicine(); -- Working fine

    return 0;
}
```

```
PS C:\Users\3TEE\Desktop\OOP Assignment 03> cd "c:\Users\3TEE\Desktop\OOP Assignment 03\" ; if ($?) { g++ q1_anotherApproach.cpp -o q1_anotherApproach } ; if ($?) { .\q1_anotherApproach }
~~~~~~~~~~~~~~~~~~~~~~~~
Name: Muhammad Hammad
Roll No: 23K-2005
~~~~~~~~~~~~~~~~~~~~~~~~
1. Fever
2. Headache
3. Flu
Choose one from the above list: 2
The counter has noted the syptoms/prescription and it has been forwarded to the pharmacist.
The pharmacist has recommended Panadol for headache to the counter.
The counter has sold 'Panadol' for Rs 350 to the customer.
The total revenue of today is now Rs 350.

------- Operator Overloading -------
The two medicines have the same expiry date.

PS C:\Users\3TEE\Desktop\OOP Assignment 03>
```

Task 02: (I have solved Task 2 with two approaches, both attached)

Task 02 (Approach #1)

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

template <typename T>
class Pet {
    protected:
        string name;
        int age;

    public:
        Pet (string name, int age) {
            this->name = name;
            this->age = age;
        }

        // pure virtual func
        virtual void makeSound() const = 0;

        void displayPetInfo () {
            cout<<"Name: " << name << endl;
            cout<<"Age: " << age << " year(s)" << endl;
        }
};

class Cat : public Pet <Cat> {
    public:
        Cat (string name, int age) : Pet (name, age) {}

        void makeSound () const {
            cout<<"The cat says Moew!" << endl;
        }
};

class Dog : public Pet <Dog> {
    public:
        Dog (string name, int age) : Pet (name, age) {}

        void makeSound () const {
            cout<<"The dog says Woof!" << endl;
        }
};
```

```cpp
class Bird : public Pet <Bird> {
    public:
        Bird (string name, int age) : Pet (name, age) {}

        void makeSound () const {
            cout<<"The bird chirps!" << endl;
        }
};

int main() {

    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
    cout<<"Name: Muhammad Hammad" << endl;
    cout<<"Roll No: 23K-2005" << endl;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

    Pet<Cat>* cat1 = new Cat("Billi", 2);
    Pet<Dog>* dog1 = new Dog("Kutta", 5);
    Pet<Bird>* bird1 = new Bird("Chirya", 1);

    cout<<"---------------------------------------" << endl;
    cout<<"\tInstances of Cat Class" << endl;
    cat1->displayPetInfo();
    cat1->makeSound();

    cout<< endl << "---------------------------------------" << endl;
    cout<<"\tInstances of Dog Class" << endl;
    dog1->displayPetInfo();
    dog1->makeSound();

    cout<< endl << "---------------------------------------" << endl;
    cout<<"\tInstances of Bird Class" << endl;
    bird1->displayPetInfo();
    bird1->makeSound();

    return 0;
}
```

```
PS C:\Users\3TEE\Desktop\OOP Assignment 03> cd "c:\Users\3TEE\Desktop\OOP Assignment 03\" ; if ($?) { g++ Task2A.cpp -o Task2A } ; if ($?) { .\Task2A }
~~~~~~~~~~~~~~~~~~~~~~~~~
Name: Muhammad Hammad
Roll No: 23K-2005
~~~~~~~~~~~~~~~~~~~~~~~~~
---------------------------------------
        Instances of Cat Class
Name: Billi
Age: 2 year(s)
The cat says Moew!

---------------------------------------
        Instances of Dog Class
Name: Kutta
Age: 5 year(s)
The dog says Woof!

---------------------------------------
        Instances of Bird Class
Name: Chirya
Age: 1 year(s)
The bird chirps!
PS C:\Users\3TEE\Desktop\OOP Assignment 03>
```

Task 02: (Approach #2)

Task2B.cpp > Pet<T1, T2> > Pet(T1, T2)

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

template <typename T1, typename T2>
class Pet {
    protected:
        T1 name;
        T2 age;

    public:
        Pet (T1 name, T2 age) {
            this->name = name;
            this->age = age;
        }

        // pure virtual func
        virtual void makeSound() const = 0;

        void displayPetInfo () {
            cout<<"Name: " << name << endl;
            cout<<"Age: " << age << " year(s)" << endl;
        }
};

template <typename T1, typename T2>
class Cat : public Pet<T1, T2> {
    public:
        Cat (T1 name, T2 age) : Pet<T1, T2> (name, age) {}

        void makeSound () const {
            cout<<"The cat says Moew!" << endl;
        }
};

template <typename T1, typename T2>
class Dog : public Pet<T1, T2> {
    public:
        Dog (T1 name, T2 age) : Pet<T1, T2> (name, age) {}

        void makeSound () const {
            cout<<"The dog says Woof!" << endl;
        }
};
```

```cpp
template <typename T1, typename T2>
class Bird : public Pet<T1, T2> {
    public:
        Bird (T1 name, T2 age) : Pet<T1, T2> (name, age) {}

        void makeSound () const {
            cout<<"The bird chirps!" << endl;
        }
};

int main() {

    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
    cout<<"Name: Muhammad Hammad" << endl;
    cout<<"Roll No: 23K-2005" << endl;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

    Pet<string, int>* cat1 = new Cat<string, int>("Billi", 2);
    Pet<int, float>* dog1 = new Dog<int, float>(27, 5.5);
    Pet<char, string>* bird1 = new Bird<char, string>('C', "1");

    cout<<"---------------------------------------" << endl;
    cout<<"\tInstances of Cat Class" << endl;
    cat1->displayPetInfo();
    cat1->makeSound();

    cout<< endl << "---------------------------------------" << endl;
    cout<<"\tInstances of Dog Class" << endl;
    dog1->displayPetInfo();
    dog1->makeSound();

    cout<< endl << "---------------------------------------" << endl;
    cout<<"\tInstances of Bird Class" << endl;
    bird1->displayPetInfo();
    bird1->makeSound();

    return 0;
}
```

```
PS C:\Users\3TEE\Desktop\OOP Assignment 03> cd "c:\Users\3TEE\Desktop\OOP Assignment 03\" ; if ($?) { g++ Task2B.cpp -o Task2B } ; if ($?) { .\Task2B }
~~~~~~~~~~~~~~~~~~~~~~~~~
Name: Muhammad Hammad
Roll No: 23K-2005
~~~~~~~~~~~~~~~~~~~~~~~~~
---------------------------------------
        Instances of Cat Class
Name: Billi
Age: 2 year(s)
The cat says Moew!

---------------------------------------
        Instances of Dog Class
Name: 27
Age: 5.5 year(s)
The dog says Woof!

---------------------------------------
        Instances of Bird Class
Name: C
Age: 1 year(s)
The bird chirps!
PS C:\Users\3TEE\Desktop\OOP Assignment 03>
```

Task 03:

Task3.cpp > Matrix<T> > displayMatrix()

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

template<typename T>
class Matrix {
private:
    T** data;
    int rows, cols;

public:
    Matrix(int rows, int cols) : rows(rows), cols(cols) {
        data = new T*[rows];
        for (int i=0; i<rows; i++) {
            data[i] = new T[cols];
        }
    }

    // ~Matrix() {
    //     for (int i=0; i<rows; i++) {
    //         delete[] data[i];
    //     }
    //     delete[] data;
    // }

    T& getMatrix(int row, int col) {
        return data[row][col];
    }

    void setMatrix(int row, int col, T value) {
        data[row][col] = value;
    }

    void displayMatrix() {
        for (int i=0; i<rows; i++) {
            for (int j=0; j<cols; j++) {
                cout << data[i][j] << "\t";
            }
            cout << endl;
        }
    }
```

```
class Matrix {

    Matrix<T> operator+(const  Matrix<T> &matrix2) const  {
        if (rows != matrix2.rows || cols != matrix2.cols) {
            cout<<"Matrices must have the same dimensions for addition.";
        }
        else {
            Matrix<T> resultantMatrix(rows, cols);
            for (int i=0; i<rows; i++) {
                for (int j=0; j<cols; j++) {
                    resultantMatrix.data[i][j] = data[i][j] + matrix2.data[i][j];
                }
            }
            // resultantMatrix.displayMatrix();
            return resultantMatrix;
        }
    }

    Matrix<T> operator-(Matrix<T> &matrix2) {
        if (rows != matrix2.rows || cols != matrix2.cols) {
            cout<<"Matrices must have the same dimensions for subtraction.";
        }
        else {
            Matrix<T> resultantMatrix(rows, cols);
            for (int i=0; i<rows; i++) {
                for (int j=0; j<cols; j++) {
                    resultantMatrix.data[i][j] = data[i][j] - matrix2.data[i][j];
                }
            }
            return resultantMatrix;
        }
    }

    template<typename U>
    Matrix<U> operator*(Matrix<U> &matrix2) {
        if (cols != matrix2.rows) {
            cout<<"Matrices cannot be multiplied because they don't have the same number of rows and columns.";
        }
        else {
            Matrix<U> resultantMatrix(rows, matrix2.cols);
            for (int i=0; i<rows; i++) {
                for (int j=0; j<matrix2.cols; j++) {
                    U sum = 0;
                    for (int k=0; k<cols; k++) {
                        sum += static_cast<U>(data[i][k]) * matrix2.data[k][j];
                    }
                    resultantMatrix.data[i][j] = sum;
```

```
class Matrix {
    Matrix<U> operator*(Matrix<U> &matrix2) {
                    }
                    resultantMatrix.data[i][j] = sum;
                }
            }
            return resultantMatrix;
        }
    }
};

int main() {

    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
    cout<<"Name: Muhammad Hammad" << endl;
    cout<<"Roll No: 23K-2005" << endl;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Int Matrices Declaration  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Matrix<int> intMatrix1(2, 2);
    for (int i=0; i<2; i++) {
        for (int j=0; j<2; j++) {
            intMatrix1.setMatrix(i,j,(rand() % 100 + 1));
        }
    }

    Matrix<int> intMatrix2(2, 2);
    for (int i=0; i<2; i++) {
        for (int j=0; j<2; j++) {
            intMatrix2.setMatrix(i,j,(rand() % 100 + 1));
        }
    }

    cout<< endl <<"============================"  << endl;
    cout<<"IntMatrix 1:" << endl;
    intMatrix1.displayMatrix();
    cout<<endl;

    cout<<"IntMatrix 2:" << endl;
    intMatrix2.displayMatrix();
    cout<<endl;
```

```
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Int Matrices Operations  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Matrix<int> intMatrixSum = intMatrix1 + intMatrix2;
    cout<<"IntMatrix Sum:" << endl;
    intMatrixSum.displayMatrix();
    cout<<endl;

    Matrix<int> intMatrixDifference = intMatrix1 - intMatrix2;
    cout<<"IntMatrix Difference:" << endl;
    intMatrixDifference.displayMatrix();
    cout<<endl;

    Matrix<int> intMatrixProduct = intMatrix1 * intMatrix2;
    cout<<"IntMatrix Product:" << endl;
    intMatrixProduct.displayMatrix();

    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Double Matrices Declaration  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Matrix<double> doubleMatrix1(2, 2);
    for (int i=0; i<2; i++) {
        for (int j=0; j<2; j++) {
            doubleMatrix1.setMatrix(i,j,(rand() % 100 + 1));
        }
    }

    Matrix<double> doubleMatrix2(2, 2);
    for (int i=0; i<2; i++) {
        for (int j=0; j<2; j++) {
            doubleMatrix2.setMatrix(i,j,(rand() % 100 + 1));
        }
    }

    cout<<"============================"  << endl;
    cout << "DoubleMatrix 1:" << endl;
    doubleMatrix1.displayMatrix();
    cout << endl;

    cout << "DoubleMatrix 2:" << endl;
    doubleMatrix2.displayMatrix();
    cout << endl;
```

```cpp
int main() {

    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Double Matrices Operations  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Matrix<double> doubleMatrixSum = doubleMatrix1 + doubleMatrix2;
    cout << "DoubleMatrix Sum:" << endl;
    doubleMatrixSum.displayMatrix();
    cout<<endl;

    Matrix<double> doubleMatrixDifference = doubleMatrix1 - doubleMatrix2;
    cout << "DoubleMatrix Difference:" << endl;
    doubleMatrixDifference.displayMatrix();
    cout<<endl;

    Matrix<double> doubleMatrixProduct = doubleMatrix1 * doubleMatrix2;
    cout << "DoubleMatrix Product:" << endl;
    doubleMatrixProduct.displayMatrix();
    cout<<"============================"  << endl;

    return 0;
}
```

```
PS C:\Users\3TEE\Desktop\OOP Assignment 03> cd "c:\Users\3TEE\Desktop\OOP Assignment 03\" ; if ($?) { g++ Task3.cpp -o Task3 } ; if ($?) { .\Task3 }
~~~~~~~~~~~~~~~~~~~~~~~~
Name: Muhammad Hammad
Roll No: 23K-2005
~~~~~~~~~~~~~~~~~~~~~~~~

=========================
IntMatrix 1:
42      68
35      1

IntMatrix 2:
70      25
79      59

IntMatrix Sum:
112     93
114     60

IntMatrix Difference:
-28     43
-44     -58

IntMatrix Product:
8312    5062
2529    934
=========================
DoubleMatrix 1:
63      65
6       46

DoubleMatrix 2:
82      28
62      92

DoubleMatrix Sum:
145     93
68      138

DoubleMatrix Difference:
-19     37
-56     -46

DoubleMatrix Product:
9196    7744
3344    4400
=========================
PS C:\Users\3TEE\Desktop\OOP Assignment 03>
```

Task 04:

Task4.cpp > ...

```cpp
#include <iostream>
#include <limits>
#include <cmath>

using namespace std;

float get_float() {
    float n;
    for (;;) {
        if (cin >> n) {
            return n;
        }
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid entry. Please re-enter: ";
    }
}

class Drone {
    protected:
        float latitude;
        float longitude;
        float altitude;
        float speed;
        bool flightStatus;

    public:
        Drone () {
            latitude=500;
            longitude=500;
            altitude=50;
            flightStatus=false;
        }

        Drone (float latitude, float longitude, float altitude, float speed) {
            this->latitude= latitude;
            this->longitude = longitude;
            this->altitude = altitude;
            this->speed = speed;
            flightStatus = true;
        }

        void adjustAltitude () {
            cout<<"Enter new altitude: " << endl;
            float newAltitude;
            newAltitude=get_float();
            altitude = newAltitude;
```

```
class Drone {
        }

        void setSpeed () {
            cout<<"Enter new speed: " << endl;
            float newSpeed;
            newSpeed=get_float();
            speed = newSpeed;
        }

        float calculateDistance(float destLatitude, float destLongitude, float destAltitude) {
            return sqrt(pow(destLatitude - latitude, 2) + pow(destLongitude - longitude, 2) + pow(destAltitude - altitude, 2));
        }

        // -------- FlyAble --------
        void takeOff () {
            cout<<"The drone has taken off." << endl;
            flightStatus=true;
        }

        void land () {
            if (flightStatus) {
                cout<<"The drone is being landed." << endl;
                flightStatus = false;
            }
            if (!flightStatus) {
                cout<<"The drone is already at ground." << endl;
            }
        }

         // -------- FlyAble --------
        virtual void navigateTo () {

        }

        // -------- ScanAble --------
        virtual void scanArea(float radius) {

        }
};
```

```cpp
class ReconDrone : public Drone {
    protected:
        int cameraResolution;
        int maxFlightTime;

    public:
        ReconDrone () : Drone () {
            cameraResolution = 0;
            maxFlightTime = 0;
        }

        ReconDrone (float latitude, float longitude, float altitude, float speed, int cameraResolution, int maxFlightTime)
        : Drone (latitude, longitude, altitude, speed) {
            this->cameraResolution = cameraResolution;
            this->maxFlightTime = maxFlightTime;
        }

        void setCameraResolution () {
            cout<<"Enter new camera resolution: " << endl;
            int newCameraResolution;
            cin>>newCameraResolution;
            cameraResolution = newCameraResolution;

        }

        void setMaxFlightTime () {
            cout<<"Enter new max flight time: " << endl;
            int newMaxFlightTime;
            cin>>newMaxFlightTime;
            maxFlightTime = newMaxFlightTime;
        }

        void navigateTo () {
            float newLatitude,  newLongitude,  newAltitude;
            cout<<"Enter the latitude to navigate the drone: ";
            newLatitude=get_float();

            cout<<"Enter the longitude to navigate the drone: ";
            newLongitude=get_float();

            cout<<"Enter the altitude to navigate the drone: ";
            newAltitude=get_float();

            double distance = calculateDistance(newLatitude, newLongitude, newAltitude);
            double estimatedTime = distance / speed;
```

```
class ReconDrone : public Drone {
        void navigateTo () {
            if (flightStatus) {
                cout<<"The drone is currently at position ("<<latitude<<","<<longitude<<") & "<<altitude <<" meters high and is navigated towards ("<<newLatitude<<","<<newLongitude<<") a
                cout<<"Estimated time to reach the destination: " << estimatedTime << " minutes." << endl;
            }
            if (!flightStatus) {
                flightStatus=true;
                cout<<"The drone is launched towards position ("<<newLatitude<<","<<newLongitude<<") and is projected to reach the height of "<<newAltitude<<" meters."<<endl;
                cout<<"Estimated time to reach the destination: " << estimatedTime << " minutes." << endl;
            }
            this->latitude=newLatitude;
            this->longitude=newLongitude;
            this->altitude=newAltitude;
        }

        // -------- ScanAble --------
        void scanArea(float radius) {
            bool found=false;
            int totalObjects=5;
            float objectLatitude[totalObjects];
            float objectLongitude[totalObjects];
            float objectAltitude[totalObjects];
            float objectDistanceFromDrone[totalObjects];

            for (int i=0; i<totalObjects; i++) {
                objectLatitude[i] = latitude + (i+1)*20;
                objectLongitude[i] = longitude + (i+1)*20;
                objectAltitude[i] = altitude + (i+1)*20;
                objectDistanceFromDrone[i] = calculateDistance(objectLatitude[i], objectLongitude[i], objectAltitude[i]);

                if (objectDistanceFromDrone[i] <= radius) {
                    found = true;
                    cout<<"Object Detected In Drone's Radius!" << endl;
                    cout<<"Object Distance from Drone: " << objectDistanceFromDrone[i] << " meters." << endl;
                    cout<<"Object Latitude: " << objectLatitude[i] << " meters." << endl;
                    cout<<"Object Longitude: " << objectLongitude[i] << " meters." << endl;
                    cout<<"Object Altitude: " << objectAltitude[i] << " meters." << endl << endl;
                }
            }

            if (!found) {
                cout<<"No object detected in the radius!" << endl;
            }
        }
};
```

Task4.cpp > Drone > setSpeed()

```
int main() {

    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
    cout<<"Name: Muhammad Hammad" << endl;
    cout<<"Roll No: 23K-2005" << endl;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

    ReconDrone rd(100, 100, 30, 50, 1080, 10);

    cout<<endl<<"----------- Navigating Recon Drone -----------" << endl;
    rd.navigateTo();

    cout<<endl<<"----------- Scanning Area -----------" << endl;
    rd.scanArea(120);

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
~~~~~~~~~~~~~~~~~~~~~~~~
Name: Muhammad Hammad
Roll No: 23K-2005
~~~~~~~~~~~~~~~~~~~~~~~~

----------- Navigating Recon Drone -----------
Enter the latitude to navigate the drone: 250
Enter the longitude to navigate the drone: 250
Enter the altitude to navigate the drone: 200
The drone is currently at position (100,100) & 30 meters high and is navigated towards (250,250) and is projected to reach the height of 30 meters.
Estimated time to reach the destination: 5.43691 minutes.

----------- Scanning Area -----------
Object Detected In Drone's Radius!
Object Distance from Drone: 34.641 meters.
Object Latitude: 270 meters.
Object Longitude: 270 meters.
Object Altitude: 220 meters.

Object Detected In Drone's Radius!
Object Distance from Drone: 69.282 meters.
Object Latitude: 290 meters.
Object Longitude: 290 meters.
Object Altitude: 240 meters.

Object Detected In Drone's Radius!
Object Distance from Drone: 103.923 meters.
Object Latitude: 310 meters.
Object Longitude: 310 meters.
Object Altitude: 260 meters.
```