**OOP Assignment 3**

**Student ID:** 23k0703

**Section:** BCS-2D

**Question 1:**

**Code:**

#include <iostream>

#include <chrono>

using namespace std;

using namespace chrono;

class Medicine

{

protected:

string name, formula;

float retailPrice;

int manufactureDate, expirationDate;

public:

Medicine(

string name,

string formula,

float retailPrice,

int manufactureDate,

int expirationDate)

: name(name),

formula(formula),

retailPrice(retailPrice),

manufactureDate(manufactureDate),

expirationDate(expirationDate) {}

string getName()

{

*return* this->name;

}

void setName(string name)

{

this->name = name;

}

string getFormula()

{

*return* this->formula;

}

void setFormula(string formula)

{

this->formula = formula;

}

float getRetailPrice()

{

*return* this->retailPrice;

}

void setRetailPrice(float retailPrice)

{

this->retailPrice = retailPrice;

}

int getManufactureDate()

{

*return* this->manufactureDate;

}

void setManufactureDate(int manufactureDate)

{

this->manufactureDate = manufactureDate;

}

int getExpirationDate()

{

*return* this->expirationDate;

}

void setExpirationDate(int expirationDate)

{

this->expirationDate = expirationDate;

}

*virtual* void printMedicine() = 0;

};

class Tablet : public Medicine

{

float sucroseLevel;

public:

Tablet(

string name,

string formula,

float retailPrice,

int manufactureDate,

int expirationDate,

float sucroseLevel)

: Medicine(

name,

formula,

retailPrice,

manufactureDate,

expirationDate),

sucroseLevel(sucroseLevel) {}

void printMedicine()

{

cout << "=====Medicine Details=====" << endl;

cout << "name: " << name << endl;

cout << "formula: " << formula << endl;

cout << "retailPrice: " << retailPrice << endl;

cout << "manufactureDate: " << manufactureDate << endl;

cout << "expirationDate: " << expirationDate << endl;

cout << "sucroseLevel: " << sucroseLevel << endl;

}

};

class Capsule : public Medicine

{

float absorptionPercentage;

public:

Capsule(

string name,

string formula,

float retailPrice,

int manufactureDate,

int expirationDate,

float sucroseLevel)

: Medicine(

name,

formula,

retailPrice,

manufactureDate,

expirationDate),

absorptionPercentage(absorptionPercentage) {}

void printMedicine()

{

cout << "=====Medicine Details=====" << endl;

cout << "name: " << name << endl;

cout << "formula: " << formula << endl;

cout << "retailPrice: " << retailPrice << endl;

cout << "manufactureDate: " << manufactureDate << endl;

cout << "expirationDate: " << expirationDate << endl;

cout << "absorptionPercentage: " << absorptionPercentage << endl;

}

};

class Syrup : public Medicine

{

public:

Syrup(

string name,

string formula,

float retailPrice,

int manufactureDate,

int expirationDate,

float sucroseLevel)

: Medicine(

name,

formula,

retailPrice,

manufactureDate,

expirationDate) {}

};

class Pharmacist

{

public:

void searchMedicine(Medicine *\**medicineList);

};

class Counter

{

public:

void searchMedicine(Medicine *\**medicineList);

void updateRevenue(Medicine *&*medicine);

};

bool operator==(Medicine *&*med1, Medicine *&*med2)

{

*return* med1.getExpirationDate() == med1.getManufactureDate();

}

int main()

{

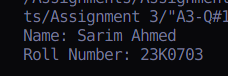
*// Header*

cout << "Name: Sarim Ahmed\nRoll Number: 23K0703\n\n";

}

**Output:**

**[No Output as per the question requirements of a skeleton class and function headers]**

****

**Question 2:**

**Code:**

#include <iostream>

using namespace std;

template <class T>

class Pet

{

public:

string name;

int age;

Pet(

string name,

int age)

: name(name),

age(age) {}

void *virtual* makeSound() = 0;

};

class Cat : public Pet<Cat>

{

public:

Cat(

string name,

int age)

: Pet(

name,

age) {}

void makeSound()

{

cout << "Meow" << endl;

}

};

class Dog : public Pet<Dog>

{

public:

Dog(

string name,

int age)

: Pet(

name,

age) {}

void makeSound()

{

cout << "Bark" << endl;

}

};

class Bird : public Pet<Bird>

{

public:

Bird(

string name,

int age)

: Pet(

name,

age) {}

void makeSound()

{

cout << "Chirp" << endl;

}

};

int main()

{

*// Header*

cout << "Name: Sarim Ahmed\nRoll Number: 23K0703\n\n";

Cat cat("Nimbo", 1);

Dog dog("Toto", 5);

Bird bird("F16", 10);

cout << "Name: " << cat.name << endl;

cout << "Age: " << cat.age << endl;

cat.makeSound();

cout << endl;

cout << "Name: " << dog.name << endl;

cout << "Age: " << dog.age << endl;

dog.makeSound();

cout << endl;

cout << "Name: " << bird.name << endl;

cout << "Age: " << bird.age << endl;

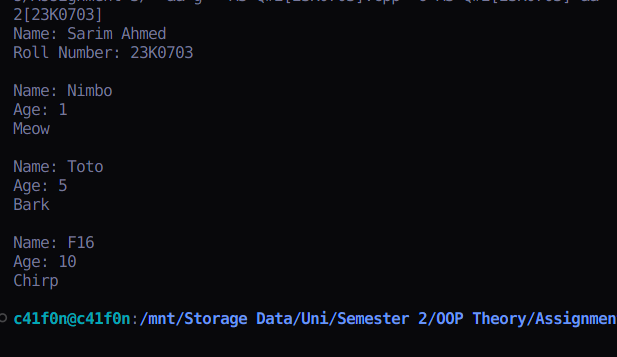
bird.makeSound();

cout << endl;

*return* 0;

}

**Output:**

****

**Question 3**

**Code:**

#include <iostream>

#include <vector>

using namespace std;

template <class T>

class Matrix

{

protected:

int rows, cols;

vector<vector<T>> data;

public:

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

{

}

void setElement(int row, int col, T value)

{

data[row][col] = value;

}

T getElement(int row, int col)

{

*return* data[row][col];

}

void *virtual* displayMatrix() = 0;

};

class IntMatrix : public Matrix<int>

{

public:

IntMatrix(

int rows,

int cols)

: Matrix(

rows,

cols)

{

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

{

vector<int> thisRow;

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

{

thisRow.push\_back(0);

}

data.push\_back(thisRow);

}

}

void displayMatrix()

{

cout << "Displaying Int Matrix" << endl;

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

{

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

{

cout << data[i][j] << ", ";

}

cout << "\b\b\n";

}

}

IntMatrix operator+(IntMatrix *&*mat)

{

int newRow, newCol;

newRow = min(rows, mat.rows);

newCol = min(cols, mat.cols);

IntMatrix temp(newRow, newCol);

for (int i = 0; i < newRow; i++)

{

for (int j = 0; j < newCol; j++)

{

temp.data[i][j] = data[i][j] + mat.getElement(i, j);

}

}

*return* temp;

}

IntMatrix operator-(IntMatrix *&*mat)

{

int newRow, newCol;

newRow = min(rows, mat.rows);

newCol = min(cols, mat.cols);

IntMatrix temp(newRow, newCol);

for (int i = 0; i < newRow; i++)

{

for (int j = 0; j < newCol; j++)

{

temp.data[i][j] = data[i][j] - mat.getElement(i, j);

}

}

*return* temp;

}

IntMatrix operator\*(int x)

{

IntMatrix temp(rows, cols);

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

{

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

{

temp.data[i][j] = data[i][j] \* x;

}

}

*return* temp;

}

};

class DoubleMatrix : public Matrix<double>

{

public:

DoubleMatrix(

int rows,

int cols)

: Matrix(

rows,

cols)

{

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

{

vector<double> thisRow;

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

{

thisRow.push\_back(0.0);

}

data.push\_back(thisRow);

}

}

void displayMatrix()

{

cout << "Displaying Double Matrix" << endl;

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

{

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

{

cout << data[i][j] << ", ";

}

cout << "\b\b\n";

}

}

DoubleMatrix operator+(DoubleMatrix *&*mat)

{

int newRow, newCol;

newRow = min(rows, mat.rows);

newCol = min(cols, mat.cols);

DoubleMatrix temp(newRow, newCol);

for (int i = 0; i < newRow; i++)

{

for (int j = 0; j < newCol; j++)

{

temp.data[i][j] = data[i][j] + mat.getElement(i, j);

}

}

*return* temp;

}

DoubleMatrix operator-(DoubleMatrix *&*mat)

{

int newRow, newCol;

newRow = min(rows, mat.rows);

newCol = min(cols, mat.cols);

DoubleMatrix temp(newRow, newCol);

for (int i = 0; i < newRow; i++)

{

for (int j = 0; j < newCol; j++)

{

temp.data[i][j] = data[i][j] - mat.getElement(i, j);

}

}

*return* temp;

}

DoubleMatrix operator\*(int x)

{

DoubleMatrix temp(rows, cols);

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

{

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

{

temp.data[i][j] = data[i][j] \* x;

}

}

*return* temp;

}

};

int main()

{

*// Header*

cout << "Name: Sarim Ahmed\nRoll Number: 23K0703\n\n";

IntMatrix mat1(2, 3);

IntMatrix mat2(2, 2);

mat1.setElement(0, 1, 50);

mat1.setElement(1, 1, 25);

mat2.setElement(0, 1, 50);

mat2.setElement(1, 1, 4);

IntMatrix mat3 = mat1 + mat2;

mat3.displayMatrix();

DoubleMatrix mat4(6, 6);

DoubleMatrix mat5(6, 6);

mat4.setElement(0, 5, 23);

mat4.setElement(2, 1, 25);

mat5.setElement(0, 5, 2);

mat5.setElement(2, 1, 2);

DoubleMatrix mat6 = mat5 + mat4;

mat6.displayMatrix();

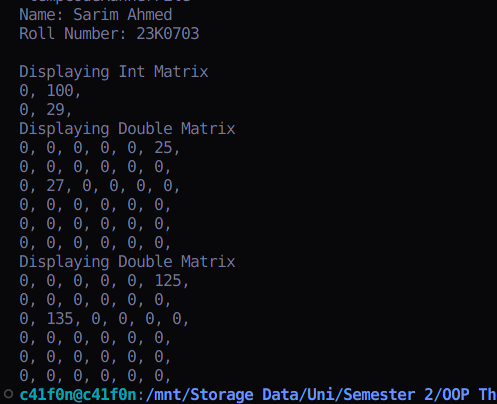
mat6 = mat6 \* 5;

mat6.displayMatrix();

*return* 0;

}

**Output:**

****

**Question 4**

**Code:**

#include <iostream>

#include <cmath>

using namespace std;

class Drone

{

protected:

float latitude, longitude, altitude, speed;

public:

Drone(

float latitude,

float longitude,

float altitude,

float speed)

: latitude(latitude),

longitude(longitude),

altitude(altitude),

speed(speed) {}

*virtual* void adjustAltitude(float altitude)

{

this->altitude = altitude;

}

*virtual* void setSpeed(float speed)

{

this->speed = speed;

}

};

class FlyableDrone : virtual public Drone

{

public:

FlyableDrone(

float latitude,

float longitude,

float altitude,

float speed)

: Drone(

latitude,

longitude,

altitude,

speed)

{

}

*virtual* void takeoff()

{

altitude += 50;

}

*virtual* void land()

{

altitude = 0;

}

*virtual* void navigateTo(float latitude, float longitude, float altitude)

{

this->latitude = latitude;

this->longitude = longitude;

this->altitude = altitude;

}

};

class ScannableDrone : virtual public Drone

{

public:

ScannableDrone(

float latitude,

float longitude,

float altitude,

float speed)

: Drone(

latitude,

longitude,

altitude,

speed)

{

}

*virtual* void scanArea(int radius)

{

cout << "Scanning everything in " << radius << "km radius." << endl;

}

};

class ReconDrone : public FlyableDrone, public ScannableDrone

{

int cameraResolution;

float maxFlightTime;

public:

ReconDrone(

float latitude,

float longitude,

float altitude,

float speed)

: FlyableDrone(

latitude,

longitude,

altitude,

speed),

ScannableDrone(

latitude,

longitude,

altitude,

speed),

Drone(

latitude,

longitude,

altitude,

speed)

{

}

void navigateTo(float latitude, float longitude, float altitude) *override*

{

float horizDistance = pow(pow(longitude - this->longitude, 2) + pow(latitude - this->latitude, 2), 0.5);

float verticalDistance = this->altitude - altitude;

float directDistance = pow(pow(verticalDistance, 2) + pow(horizDistance, 2), 0.5);

float estimatedTime = (float)(directDistance / speed);

cout << "Estimated time to reach (" << latitude << ", " << longitude << ", " << altitude << ") is " << estimatedTime << " seconds." << endl;

FlyableDrone::navigateTo(latitude, longitude, altitude);

}

void scanArea(int radius) *override*

{

try

{

if (radius <= 50)

{

ScannableDrone::scanArea(radius);

*// Rnadom objects being detected*

cout << "Bird detected at (13.222, 23.225, 2.99)" << endl;

cout << "Car detected at (14.242, 32.2222, 2.99)" << endl;

cout << "Human detected at (11.522, 82.2522, 2.99)" << endl;

}

else

{

throw radius;

}

}

catch (int e)

{

cout << "Requested radius is out of range" << endl;

}

}

};

int main()

{

*// Header*

cout << "Name: Sarim Ahmed\nRoll Number: 23K0703\n\n";

ReconDrone drone1(0, 0, 0, 50);

drone1.takeoff();

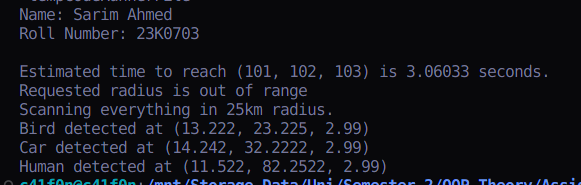
drone1.navigateTo(101, 102, 103);

drone1.scanArea(101);

drone1.scanArea(25);

}

**Output:**

****