OOPS Lab

Programming Assignment № 3

Objects, Attributes and Behaviour

CSE Department, JMI 29/08/2023

Read carefully before you begin:

- Total Marks: 30. Each question carries 10 marks.
- You have 2 hours to complete the assignment. Failure to have your program evaluated before you leave the lab will cause forfeiture of the grade for that lab.
- In order to receive full marks, you must demo the full working code and show the output and given an explanation of your approach where applicable.
- Please **save your code** throughout the semester in a place where you do not lose it. You will be required to submit it at the end.
- Use proper filenaming conventions and commenting. Code that is hard to read or understand will incur a penalty.
- Collaboration must kept to general discussions only. Please do NOT share code or directly share answers with each other. Plagiarism is unacceptable.

Problem 1 : Access Modifiers (10 marks)

Examine the following code listing carefully and then answer/do the following:

1. Identify the access modifiers in the class declaration.
2. What is the scope of the data attributes?
3. What is the scope of the member functions?
4. Implement the class defining each of the functions, constructors and destructor.
5. Assess the truth or falsehood of the following statement by experimentation. Demonstrate using code.

   > The member functions of a class can access the private members of *all* objects of that class.

Listing 1: Sample C++ code – a 2D Point C++. (Source: Saif Ali)

```cpp
#include <iostream>

using namespace std;

class Point {
   double _x;
   double _y;
   public:
      void setCoords( double x, double y );
      void printCoords( );
      double distFrom(Point &p);
      Point midPoint(Point &p);
      Point();  // This is the DEFAULT constructor
      Point(double x, double y);  // This is the PARAMETERIZED constructor
      Point(Point &p);  // This is the COPY constructor
      ~Point();  // This is the destructor
};
```

Problem 2: static Variables and const Members and Variables (10 marks)

1. Modify the above class to include a **static** variable called **count** which keeps track of how many objects of the type **Point** have been created in the program. Declare this variable and define it and modify the code to accomplish the required functionality.
2. Modify the above class to add four more count variables for each quadrant of the 2D plane. **countTL**, **countTR**, **countBR** and **countBL** (top-left, top-right, bottom-right, bottom-left). Modify the code to achieve the functionality.
3. Identify the member functions in the **Point** class that should be declared as **const**. Modify them to make them **const**. What effect will this have on access to data members?

Write tests to verify or falsify the following claims:

1. Static data members must be defined and initialized outside the class definition.
2. Memory is allocated for static data members once they are defined even though no objects of that class have been instantiated.
3. Static member functions cannot access non-static data members.
4. static const data members can be initialized within the class definition but not outside it.
5. const member functions cannot change the state of an object, i.e, they cannot change the values of any of the data members.

6.

Problem 3: Arrays of Objects (10 marks)

1. Make a 2D array of Point objects. Initialize them to be placed on the points of a 100 $\times$ 100 grid.
2. Visualize the points on a 2D plane (*Hint:* Use **graphics.h**).
3. Implement a function called **getKNN** that finds the k nearest neighbors of a given point where k is provided by the user s.t $2 \leq k \leq 10$.

OPTIONAL Add an attribute called **mass**. Add two more attributes called **forceX** and **forceY** that store the gravitational force experienced by this point in each direction. Assign a random mass to each point between 1 and 10. Implement a function called **pushNeighbours** which causes each point to exert a gravitational push on it's k nearest neighbors in accordance with Newton's laws. Then in the main function, perturb the position of the points in proportion to the gravitational force experienced by them. Simulate the movement of points and visualize it on the screen using an animation that goes for 10000 time steps. Use a time step of 1 second.