OOPS Lab

Programming Assignment № 4

Polymorphism: Operator Overloading

CSE Department, JMI 5/09/2023

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.
- Note: Your Point class must have all of the previous functionality from earlier lab assignments. If it does not, please spend time after today's lab to catch up. You need to have distFrom, midPoint, all constructors, the static count, the quadrant-wise counts and the k-Nearest neighbours functionality. If you do not have the optional physics simulation, it is all right but I encourage you to try it out.

Problem 1 : Arithmetic Operator Overloading (10 marks)

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

1. Identify the overloaded operator declaration inside the class definition.
2. Identify the return type and argument type.
3. What do the two **const** keywords mean? What restrictions do they impose on the functionality of the operator?
4. Implement the operator by defining an appropriate function outside the class definition.
5. Test your implementation by adding 2 points. Which one is that "this" object and which one is "that" object?
6. Chain the operator and try to add more than two points. How many points can you chain?

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

```cpp
#include <iostream>

using namespace std;

class Point {
   static int count;
   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
      Point operator + (const Point & p) const;
      static int getCount();
};
```

Problem 2: Relational and assignment operators (10 marks)

1. Modify the above class to include a the "less than" (<) operator and implement it. Use the Euclidean norm as your distance measure as follows:

   Point $p_1 < p_2$ if and only if $|p_1| < |p_2|$ where if the coordinates of p_1 are (x_1, y_1) then $|p_1| = x_1^2 + y_1^2$.

2. Similarly add a "greater than" (>) operator.

3. Modify the above class to add an "equality" (==) operator that returns TRUE if $|p_1| - |p_2| < \epsilon$ and FALSE otherwise where ϵ is a user defined tolerance.

Problem 3: User defined functionality through operator overloading. (10 marks)

1. Link the "TinyPNG" library into your C++ program. See `https://github.com/rahulg/tinypng`.

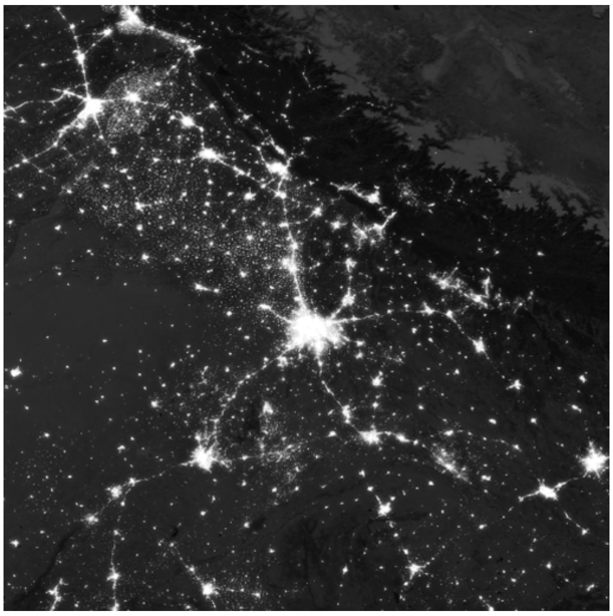

Figure 1: Nightlights over a part of the Indo-Gangetic Basin. Source: NASA. See https://blackmarble.gsfc.nasa.gov/.

2. Load the image called "nightlights-igp.png" into your program by instantiating an appropriate class from the library. This image represents the "nightlight intensity" over a part of the Indo-Gangetic basin.

3. Let's say we define the "less than" (<) operator as follows:

   Point $p_1 < p_2$ if and only if $I(p_1) < I(p_2)$ where $I(p_i)$ is the nighlight intensity at point p_i.

   The value of $I(p_i)$ will come from the image.

4. Test this new functionality using two points. What does it mean for a point to be "less than" another point under this new definition? Does it have a real-world interpretation?

[OPTIONAL] Problem 4: Scale Operator

1. Implement a scaling operator (*) which scales a point by a decimal number s s.t $p_1 * s$ will return a point in which both coordinates will be scaled by the factor s.

2. Using the scale operator, implement a function called **linear_combine** that takes two arguments; a point p and a scale factor $s \in [0, 1]$ and returns another point q such that q is the linear combination:

   $q = (this) * s + p * (1 - s)$