OOPS Lab

Programming Assignment № 8

Run-time Polymorphism: Pointers, Abstract Classes, Virtual Functions

---

CSE Department, JMI 10/10/2023

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.
- TIMED LAB: This lab is timed. Each question has an indication of the amount of time you must spend in solving this question. When such time elapses, please move on to the next question. Failure to do so will impact grades.

Problem 1 : Abstract Class and Pure Virtual Functions (10 marks, 30 minutes)

1. Modify the **Element** class from your previous lab to have the following pure virtual functions:

   (a) **norm**: Returns a **double** number indicating the Euclidean distance or length of the element. For Points and Vectors, it is the sum of squared coordinates. For Lines, it is the square of the length.

   (b) **print**: Prints the details (element type name, data members and norm) of the particular derived class of **Element**.

2. Define and implement a class called **PointN** that is derived publicly from **Element**. It should represent an n-dimensional point where n is specified by the user. It should contain at least one default constructor and one parameterized constructor along with a destructor. It should naturally override the **print** and **norm** functions.

Problem 2: Insertion Sort (10 marks, 60-90 minutes)

1. Instantiate k objects according to the following rules:

   Start a counter j that goes from 0 to 9999. If $j\%3 == 0$, instantiate a **Point** and set it to random coordinates. If $j\%3 == 1$, instantiate a **Vector** and set it to random coordinates. If $j\%3 == 2$, instantiate a **PointN**, set its length to a random number between 2 and 10. Initialize it to random numbers.

2. Write a function called **slowSort** that sorts these objects using insertion sort by the magnitude of their norms.

3. Measure the performance (execution time in micro seconds) of **slowSort** algorithm for k = 100, 1000, 10000, 100000.

Problem 3: Performance Optimization (10 marks, 60 minutes)

1. Write a function called **fastSort** that sorts these objects using insertion sort by the magnitude of their norms. Initially, **fastSort** should be simply a copy of **slowSort**. Modify it to improve the execution time and report the speedup. Note that:

$$speedup = \frac{exec\ time\ of\ slowsort}{exec\ time\ of\ fastsort} \tag{1}$$

   Demonstrate a speed up of at least 1.2