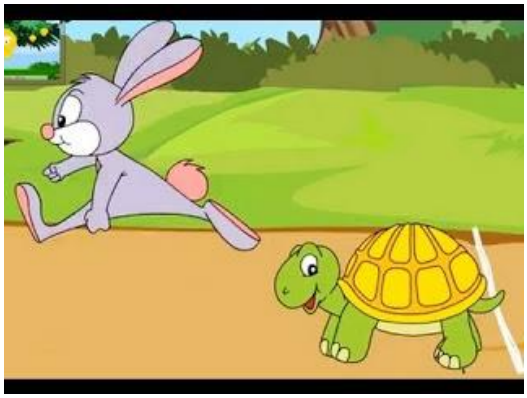
OBJECT ORIENTED PROGRAMMING

Practical test

Date: 08.04.2021

Time allotted: 1 hour 30 mins

Rabbit vs. Tortoise chocolate race for chocolate



For this test you will implement a simple a game where multiple characters (rabbits and tortoises) walk around an infinite 2D grid to gather chocolate bars.

You don't need to actually implement anything for this grid! You will just need to store the position of the characters on this grid. As the grid is infinite, you can also have negative positions for the characters, like (-100, 20), (-40, -30) etc.

First you need to implement a class hierarchy for the characters; you will have three classes: **Character**, **Rabbit** and **Tortoise**.

Character is the generic base class and it has (at least) the following attributes: the ***id*** of the character (you can store the id as an integer), a ***bag*** (an array) where the character can gather chocolate bars and the ***position*** of the character. This base class has a method ***void move(int direction)***, that will be overridden in all its subclasses. You can create the method as pure virtual in the Character class or you can just provide an empty implementation for it.

The bag where the chocolate bars are stored is represented as an array of integer which represents the weights of the chocolate bars the character gathered. This array should be allocated on the HEAP and it should have a maximum size of 100 bars. You **don't need to resize** the array if a character gathered 100 chocolate bars! Create a method: ***bool pickChocholateBar(int barWeight);*** that adds a chocolate bar of weight ***barWeight*** at the end of this array.

To store the position of a character you should write a class **Point.** The class point should have two integer attributes: the *x* and *y* coordinates of the point. Overload the stream operator (operator<< and operator>> for the point class). Also, provide a default constructor (which sets *x* and *y* to 0), and a parameterized constructor. Implement getters and setters for the *x* and *y* attributes.

Then create the two sub-classes for the Character base class: **Rabbit** and **Tortoise**. As mentioned above, each of these subclasses should override the ***void move(int direction)*** method. The parameter *direction* controls the direction in which the character should move (0 – left, 1 – right, 2 – up, 3 – down). If you prefer, you can change the parameter type to an **enum**.

- The Rabbit moves 4 cells in indicated direction. For example:
    - if a Rabbit position is Point(0, 10), after you call move(0), i.e. left, the position of the Rabbit will be (-4, 10);

    - if a Rabbit position is Point(0, 10), after you call move(1), i.e. right, the position of the Rabbit will be (4, 10);
    - if a Rabbit position is Point(0, 10), after you call move(2), i.e. up, the position of the Rabbit will be (0, 14);
    - if a Rabbit position is Point(0, 10), after you call move(3), i.e. down, the position of the Rabbit will be (0, 6).
- The Tortoise moves 3 cells in the indicated direction. For example:
    - if a Tortoise position is Point(0, 10), after you call move(0), i.e. left, the position of the Tortoise will be (-3, 10);
    - if a Tortoise position is Point(0, 10), after you call move(1), i.e. right, the position of the Tortoise will be (3, 10);
    - if a Tortoise position is Point(0, 10), after you call move(2), i.e. up, the position of the Tortoise will be (0, 13);
    - if a Tortoise position is Point(0, 10), after you call move(3), i.e. down, the position of the Tortoise will be (0, 7)

In the main function, create a **polymorphic** array (**in the main function, you don't need to create a class for this!**) that will store at most 10 characters (you can allocate this array however you prefer, on the heap or on the stack). Then, create and insert one Rabbit and two Tortoises in this array.

Finally, in the main function create a simulation of n steps of the game.

At the beginning all the characters are positioned randomly on the grid.

Then, in each of the n^{th} step, the all the characters move in the same direction.

If a Rabbit reaches a grid position where one of its coordinates are divisible by three, then it can pick up a chocolate bar of weight equal to the absolute value x coordinate of that grid position. For example, a Rabbit will gather a chocolate bar of weight 2 in the position (2, -3) and a chocolate bar of weight 12 in the position (-12, 2).

If a Tortoise reaches a grid position where one of its coordinates is even and the other one is odd, then it can pick up a chocolate equal to the sum of the absolute values of the x and y coordinates of that grid position. For example, a Rabbit will gather a chocolate bar of weight 5 in the position (2, -3) and a chocolate bar of weight 15 in the position (-13, 2).

At the end of the simulation, display, for each character, the name, the type, and the total weight of the chocolate bars it found. Then display the winner (the character that gathered most of the chocolate bars)

Rubric

Description	
Point class implementation: - Default constructor - Parameterized constructor - Getters and setters for the *x* and *y* coordinates - Overloading operator<< and operator>> for the point class	2 points
Class hierarchy, method overriding	2 points
Copy constructor, assignment operator and destructor for Character class	2 points
Polymorphism, game simulation	3 points
Default	1 point
Total	**10 points**

How to **lose** points?

- Write all the classes in the same file. Don't use headers. **You can lose up to 0.2 points.**
- Memory leaks. **You can lose up to 0.5 points.**
- Don't use meaningful names for your variables. **You can lose up to 0.1 points.**
- Use global variables. **You can lose 0.2 points.**
- **Plagiarism: you will definitely lose 10 points.**