Object Oriented Programming 2019/2020

Assignment / Practical project

The assignment statement consists of three parts:

Part A – Rules of the assignment

The work consists of a C++ program, which must implement the proposed theme correctly using the principles of object oriented programming and the mechanisms and standard library components presented in the classes. The user interface is organized according to the concept of non-graphical console, and a library to assist in this task will be made available. The project statement leaves several aspects open that are less relevant to the work and must be analyzed and solved in accordance with basic logic and common sense, as long as they do not decrease the remove complexity, functionality nor OPP topics from the assignment.

The project can be done in groups of **two** students, even if the students are from different practical classes.

The work is delivered in two milestones. The deliveries are made via moodle, and indications will be given in due course. There are presentation/defenses that are mandatory and greatly influence the grade. Additional rules for the practical work are described in "Object Oriented Programming – information and evaluation rules", in moodle, whose consultation is required before asking questions on these rules.

Any relevant missing issues will be resolved and clarified by the teachers in moodle.

Excluding the prior knowledge of 1st year courses, this project only requires knowledge of C++ programming topics that are covered in this course.

Part B - Theme description

1. INTRODUCTION

We want to create a car racing simulator in C++. The simulator includes some obvious concepts such as **Race, Car, Driver, Race Track, Track and Garage** (there may be others). Each of these concepts is explained below. However, it is very important to consider the following:

- Only major entities are described. Other entities, with greater or lesser relevance may be necessary.

- A relationship of one entity to one class is not mandatory. It is very likely that this relationship exists, but it can happen that the representation of an entity/concept is spread over several classes, or is simply an attribute of another class.
- This description focuses mainly on the properties and behaviors of each concept/class. Specific details on methods and data, including those not mentioned explicitly, shall be inferred and implemented.
- The members of each class should be inferred given the desired behavior for each class, depending on the concept that this class represents. You must select appropriate data types and structures within each context. You should take particular care to avoid strategies that would be appropriate in C but not in C++.
- There is more than one possible strategy and implementation.

2. CARS

A car is a small device that simulates a real vehicle with the following behavior:

- It has brand and model. No car can be created without brand. The model should also be set at the beginning, but if omitted, the model shall be "base model".
- Each car has an identification (one letter from **a** to **z**) that is automatically assigned in the car creation. If there are more cars than letters, surpluses will be assigned the letter '?'. The letter assignment is automatic (without intervention from the rest of the program), with the first letter assigned being 'a'.
- It is electric and has a certain amount of energy (in mAh). It can be charged, but only when the car is stationary. The charging process involves a crank that provides **n** mAh, where **n** is greater than zero. The car will not exceed the maximum capacity of the battery. The maximum capacity and the initial capacity are characteristics that are specified when a car is manufactured. The value of **n** is given when the operation is performed.
- It may be moving or stationary. When it moves, it goes forward at a certain speed (meters per second).
- When moving, a car can keep its speed, accelerate or brake. The accelerator can be pressed or released. While the accelerator is pressed, its effect is to increase the car's speed by 1 m/s. This effect occurs only once per second and is maintained until the accelerator is released. This means that if you want to increase the speed by 2 m/s, the accelerator must be pressed and held for 2 seconds and then released, or pressed and released twice. The accelerator can be pressed and released immediately with the corresponding speed increase; however, it can only be pressed and released only once each second.
- The car also has a brake that works exactly the same way as the accelerator, with the only difference being that the speed is decreased by 1 m/s instead of increasing. Everything else works in the same way as the accelerator.
- There is a maximum speed, which is one of the car construction parameters. This maximum speed will never be exceeded. Similarly, the speed will never be less than 0. The acceleration and brake pedals cannot be simultaneously pressed.

- Movement consumes energy: 0.1 mAh per second per meter/s. If there is no energy, the car slows down as if it was braking (continuously losing 1 m/s per second). If the driver simultaneously brakes, this effect is added to the speed loss effect for lack of energy.
- It has an emergency signal that can be switched on or off, regardless of speed.
- A car does everything described above only if a driver is inside it (see below). It is the driver that operates the various mechanisms in the car where it is in. There can only be one driver in the car. The driver can leave or enter the car only when it is stationary.
- The car must have a way of knowing that time has passed. This will allow it to simulate its behavior (for example, spend energy while moving, accelerate, brake, etc.).
- A car can be irreparably damaged. If this happens, the car is immediately set to speed 0 and will no longer respond to any command except the following two: to be removed from a race track while continuing to exist (see below); and to be permanently removed (cease to exist). If the car suffers an irreparable damage, the driver loses its “life” (it’s a robot). Naturally, this only applies if the car has a driver at that moment.
- A driver cannot be assigned to an irreparably damaged car.

3. DRIVERS

A driver is a small robot that simulates the behavior of a real driver. It has the following characteristics:

- It has a name assigned when it is manufactured (such as real people, when they are born) and the name never changes.
- The name of each driver is unique. This aspect is automatically checked by the class, if it detects repeat, it will automatically modify the assigned name in order to be unique.
- It can be driving a car. Of course, at a given instant it can only drive one car. It can operate the mechanisms of the car it is driving. This implies that the driver can decide what it will do with the car while also having the ability to receive orders (coming from outside) to perform a certain action. For example, the driver can decide to brake, but it can also receive an order to do so (imagine a law enforcement officer who demands that a car stops for speeding). Basically, anything that can be done with the car can be decided by the driver or by means of an order coming from "outside".
- It may be ordered to enter a specific car or to get out of the car it is currently in.
- It can be competing in a race. When driving, at every second of the race the driver displays a certain behavior according to its personality, which defines its actions over the car. The specific personality of a driver depends on its type, as described below.

CRAZY DRIVER

This type of driver exhibits the following behavior when competing:

- It is inattentive and only remembers to start running at the X^{th} second of the race (where X is a random number between 1 and 5).
- At every new second it asks the track what its place in the race is (see below). If it is not in the first or last place, it accelerates. If it is in first place, it maintains the current speed. If it is in last place, it gets in a bad mood and brakes.
- If it notices that it lost places since the previous instant (second), it accelerates in order to increase its speed by 2 m/s (unless it is in the last place, as described above). If the car runs out of energy, it panics and turns on the emergency signal.
- There is a 5% probability of this driver doing something that irreparably damages the car. When this happens, the car in the place immediately behind also is irreparably damaged (the driver is scared and crashes) but no other car suffers from this.

4. FAST DRIVER

The fast driver displays the following behavior:

- It is extremely anxious and starts from the first second of the race.
- It continuously accelerates until the car's energy reaches half of its maximum capacity. As a precaution, from that moment on, it will only accelerate once (to + 1 m/s) every 3 seconds.
- Due to its anxiety, every 10 seconds there is a 10% probability of having a panic attack, and if this is the case it activates the emergency signal.

5. SURPRISE DRIVER

Define and implement other type of driver that behaves differently from the previous two. This type of driver should be implemented only by your group (it will be very "surprising" if he same surprise driver appears in more than one group).

6. GENERAL DEPARTMENT OF TRAFFIC - DGV

This entity has the record of all cars and drivers and is directly responsible for the existence of these objects. This entity can be backed up if the user so wishes (see command section).

7. RACE TRACK: TRACK AND GARAGE

The race track comprises the actual track where cars compete and a garage where cars are temporarily stored when not competing. Track and garage are self-contained concepts, but do not exist outside of the race track in which they are inserted. The race track has a name that allows it to be differentiated from other race tracks. The name of each race track is unique, and this is validated and corrected automatically by the class in question. The race track has a track certified for a maximum of N cars. The number N is one of the parameters of its construction. The track has a certain length in meters, corresponding to the distance between the start line and finish line. This distance is a feature of the race track and can never be changed, because the race track is already built and is not made of rubber.

The cars do not belong to the race track and can get out of a race track and go to another race track, to compete in other races. However, if a race track is destroyed (for example, a flood occurred), all the cars that are on it are irreparably damaged and are considered automatically off that race track. A car might not have a race track associated at a given moment. If this is the case, then it will be just under the purview of the DGV (**G**eneral **D**irectorate of **V**ehicles, in Portuguese), which is the entity that controls the existence of cars. This entity also controls the existence of the drivers.

The race takes place on the track, where cars are placed for this purpose. When a race is in progress, the track keeps track of the position and place of each of the cars that are competing. The car will be in a certain **position** at the beginning of the track (the site from which it starts) and it will be in one **place** during the race. The first place corresponds to the car that is further ahead. Vehicles that are in the garage are not counted for maximum N cars and do not compete in races. The track displays the following behavior:

- The track allows the addition of cars, as long as the race has not started yet. The cars are placed side by side. The track is like the races of the Olympic 100 meters: there are N lines, one for each car.

- The track allows the insertion of a driver in a car. Naturally, this insertion only takes place if the car is empty and stationary, as by the car rules.

- The track allows the start of the race. This only happens if there are at least two cars competing on the track and all the cars on the track have drivers. After the start of the race, no other car is accepted, and no driver change is allowed.

- The track allows to advance the time by one or more seconds during the race, if it has already started and not finished. Each time this mechanism is activated, the track indicates to each of the cars the time progressed. Note that in the case of moving more than a second at a time, cars will have to be explicitly informed of each of these seconds so that their behavior is correct (e.g., acceleration and braking).

- If any car turns on the emergency signal, it will be removed from the race and moved to the garage. The garage is a place to store cars. When a car is placed in the garage, the driver is signaled to leave the car as the race is over for him.

- It allows to finish the race. The cars are removed from the track and go to the garage, and the drivers get out of their vehicles.

- It has a mechanism to get ("get" ≠ "show") race-related information indicating whether it has already started or not and, if so, detailing the current classification. A classification example might be:

```
Information about the race in the racetrack Silverstone (5000 m):
1. B Ferrari / Vettel (fast) - 5mAh, 290mAh - 2410M - 55m / s
2. D Mercedes / Hamilton (crazy) - 5mAh, 350mAh - 2300m - 50m / s
3.
...
```

8. CHAMPIONSHIP

There is also a championship concept:

- A championship consists of a series of races, each being held in a different race track, one at a time. A championship adds a set of race tracks in which a race is held in each one of them, with a certain order.
- The association drivers/cars and participant race tracks are defined before the start of the championship and cannot be modified during it.
- Cars that become irreparably damaged in a race fail the rest of the championship.
- At the end of each race the following scores are attributed: 10 points for first place, five points for second and two points for third. These scores are only awarded if the drivers have finished the race.
- The championship must maintain a classification of drivers, allowing the presentation of the champion and the runner-up when the races are over.
- There are various aspects and features that should be analyzed and developed by the students:
    - You must define a set of cars, drivers and race tracks for a championship. This is the "population" for the various races.
    - There should be mechanisms to perform the next race, to see scores, etc. (and other obvious features).
    - It should be possible to propagate commands to the tracks, cars and drivers.

  All this functionality will be activated via user commands as described later.

Regarding the organization of the various entities: for most of the entities it was described who controls what/who has what. For cases not covered, you can assume that the simulator is, ultimately, who has everything.

VISUALIZATION

You should implement a functional user interface which allows to:

- See the scores of drivers, ordered in decreasing order.
- See the race in progress, using a helper library for console control, or any other library chosen by the students (e.g., Qt, ncurses, other). This interface does not have to be very elaborate – the purpose is functional and mainly to distinguish the cars, see where they are at their position on the track, etc.
    - Visually, the cars are identified by letters: this should be lower case whenever a car is not driven and upper case otherwise.
    - The track is made up of several "lines" as the tracks of the races of the Olympic 100 meters race: each car has its space (way) (its line on the screen). It's like having one road for each car. Assume it will not be too many lines to the screen.
    - The representation of the car in its position on the track can be made in a simple manner; through a simple rule of three, it is made to match the size of the track the number of available characters on the screen.
    - All the events that depend on a probability, should be recorded in a log in the simulator. This log can be displayed on the screen with a command (described later).
- View the cars in the garage, which in this case are without driver.

Direct interaction with the user must be placed exclusively on the part of the program where it makes more sense. Logic concepts should not interact directly with the user, e.g., cars, drivers, etc. Instead, there should exist classes to interact with the user, while others are indirectly called by these components.

9. COMMANDS AND USER INTERACTION

The user interacts with the simulator through written commands. There are no plans to use graphical paradigms and if you want to, you can, provided they do not leave C++ realm, but it is up to you.

Each command is a line of text formed by the command name and 0 or more parameters separated by spaces. The parameters specify which "object" to which it is giving the order (car, driver, track, race track, etc.) and other parameters depending on the given order (example: number of seconds to advance in the simulation).

The simulator has two operating modes:

- **Mode 1** - Sets what exists: cars, race tracks, drivers, with all its features. It should also be possible to add and remove any of these elements, as well as to enter and exit drivers in cars. You can make further backup/restore to/from the memory of the DGV.
- **Mode 2** - Simulation of a championship: the simulator lets you choose the race tracks and their order in the championship. All cars with associated drivers participate in all races as long as they fulfill the necessary conditions to do so. A score for each driver must be maintained.

The characters **<** and **>** are not written by the user and what is between these **<** and **>** are values or names according to their meaning in the command context.

The words for the commands are based on the Portuguese word for the action done. You can see what the command does by reading its description. We insist on using the same command words for everyone for obvious practical reasons. If you really feel the need to change the words for something based on English, ask first for permission.

Mode 1

In **mode 1**, the user has the following commands:

- **carregaP <filename>** - Obtain a set of drivers to be created from the data present in the file, whose lines have the following format:

type name

  The type can be: crazy, fast or surprise.

- **carregaC <filename>** - Obtain a set of cars to be created from the data present in the file whose lines to the following format:

initialCapacity maximumCapacity brand model maximumVelocity

- **carregaA <filename>** - Obtain a set of race tracks to include in this championship to be created from the data in the file, in the format shown below. For race tracks can assume that the name is just a word.

N length name

- **cria <type> <object data>** - Adds an object of a particular type (c = car, p = driver, a = race track) where object data are exactly the data that must exist to create an object of a given class (the same used in the appropriate file format to create multiple objects). Example: add p fast Jairton Senna creates a driver of this type with the given name.

- **apaga <type> <identifier>** - Eliminates a particular object of one type (c = car, p = driver, a = race track) where identifier is the name that identifies the objects, respectively: one letter to identify the car (if the car is already associated with a driver, the driver becomes available to be associated with another car); name is the identifier for driver and race track. Example: delete p Jairton Senna deletes the driver with the given name.

- **entranocarro <carId> <driverName>** - Places the driver driverName into the car carId. If the driver is already in another car, nothing happens, according to common sense.

- **saidocarro <carId>** - Removes the driver from the car carId.

- **lista -** Displays on screen the information on the cars, the drivers and race tracks as well as who is in that car and what else is important to know about these entities.

- **savedgv <name> -** Makes a copy of the object representing the DGV into memory (warning: it is not a copy to a file), associating it with the given name. The "original" DGV can still be used and manipulated by the commands.

- **loaddgv <name>** - Retrieves the DGV object previously stored in memory with the given name. All commands now affect this recovered copy instead of the previous object that was being manipulated. The information in this previous object is lost.

- **deldgv <name>** - deletes the memory copy of the object that represents the DGV with the given name.

Mode 2

Mode 2 begins with the command **campeonato** in which you define the order in which the race tracks participate in the championship:

- **campeonato <A1> <A2> ... <An>** - The first race takes place at the race track **A1**, the second race in **A2**, etc.,**An**, where **An** is the name of the race track **n**. The number of parameters is equivalent to the number of participating race tracks. The batteries of all cars are charged up to their maximum capacity.

- **listacarros** - It shows information about the cars.

- **carregabat <carId>** <Q> - supplies **Q** mAh of energy to the car identified by carId.

- **carregatudo** – charges all car batteries up to their maximum capacity.

- **corrida** - The next race starts. Only the cars that have drivers and are able to compete (for example, that are not in the garage) can participate in the race.

- **acidente <carId>** - causes irreparable damage to the car identified by carId.

- **stop <driverName>** - gives the order for the car driven by driver driverName to stop (brake until full stop). If the driver is in a race, it goes out of it. If the driver is in a championship, it goes out of the race but it remains in the championship and can participate in the remaining races of that championship.

- **destroi <carId>** - remove the car of existence (it stops existing). If a driver is inside, it is removed from the car.

- **passatempo <n>** - passes **n** seconds of simulation time.

- **log** - shows the log messages that the simulator has registered.

Part C - Considerations, milestones and deliveries

CONSIDERATIONS AND RULES

It is expected the following:

- The program should compile without any warning or error.
- The user (note: "user" ≠ "driver") may exhibit unpredictable behavior and the program should be fairly robust to this aspect (for example, tolerate incorrect data input). If the user enters incorrect values (for example, for the construction of objects), an exception should be generated and captured by the program.
- The simulation should not cause runtime errors.
- Classes shall implement the methods needed to make them robust and self-contained within the common C++ usage.

DELIVERIES AND IMPORTANT DATES

Milestone 1 - Deadline: **November 24, 2019**

Program that implements the following features:

- Command interpretation
- Mode 1:
  - Execution of all commands except savedgv, loaddgv and deldgv.
  - Creates only 1 race track.
  - Creates only one type of driver, the generic driver (described in section 3).
- Mode 2:
  - Command "campeonato <A1>" (there is only one race track).
  - Command "passatempo <N>"
  - In this milestone, the cars always move one position (meter/s) forward regardless of their driver. They are not charged nor use energy.
- Visualization of the movement of cars.
- Report - information will be given about the content of the report.

Milestone 2 - Deadline: **January 5, 2020**
Objectives: full simulator with all the functionality and a report.