Data Structures and Algorithms. **Degree in Computer Engineering** **Double Degree in Computer Engineering and Business Administration** **Degree in Applied Mathematics and Computing** **University Carlos III of Madrid** **Lab case. COURSE 2020-2021**	

Covid-19 Vaccination Campaign

The Lab case consists of three phases. This document explains the description of the phase 2. Phase 3 will be published in the upcoming weeks.

Phase 2 – Covid-19 Vaccination Campaign appointments management (10 puntos)

The objective of this phase is to develop an application which allows to manage appointments for patients who are going to be vaccinated in the upcoming weeks.

In this phase, we will continue leveraging *Patient* class (defined in phase 1), which allows to represent a patient and its relevant COVID-related information. We are assuming that the name (surname, name) is unique for each patient. Instead of relying a linked list to represent a Health Center, we will use a data structure, *HealthCenter2*, where the patients will be ordered alphabetically (unless the opposite is said) and the search, insertion and deletion operations have logarithmic complexity.

fase2.py file contains *Patient* class implementation as well as a skeleton for *HealthCenter2* class, which allows to represent the patients of a health center. Within this class, you need to implement the following features:

- Function ***searchPatients***, which receives the following parameters:
    - *year*: The function will look for all patients born in that year and the previous ones. Thus, if the parameter contains current year (2021), the function will retrieve all patients.

- *covid*: Boolean whose default value is going to be None, indicating that the function needs to look for all patients regardless of whether they suffered covid or not. If it is set to True, the function will look for patients who have suffered covid, or those who have not otherwise.

- *vaccine*: Its default value will be None, meaning that the function will search all patients, regardless of whether they have been vaccinated or not. If its value is 0, the function will look for patients who have received 0 doses. With a value of 1, it will bring patients vaccinated only once, and with a value of 2, patients who have received both vaccine doses.

The function must visit all patients to check whether they meet the conditions specified by the function parameters or not. To access the patients, this function must apply a **level order traversal**. It is not allowed to use other types of traversals. The function will return a new object of type *HealthCenter2*, which only contains the patients who meet the specified input criteria, and that must be ordered alphabetically.

- Function ***vaccine***, which takes the following parameters:

  - *name*: name (surname, name) of a patient

  - *vaccinated*: object of type *HealthCenter2*, where the patients are stored alphabetically.

  To emulate the vaccination process, the function needs to consider the following use cases:

  - If the patient does not exist in the invoking health center, the function shows a message saying that the patient does not exist. It also returns False.

  - If the patient exists in the invoking health centre, and had already received the two corresponding doses, the function displays a message informing that this patient had already been vaccinated previously. In addition, it removes the patient from the invoking health centre and stores him/her in the vaccinated health centre. The function returns False.

  - If the patient exists in the invoking health centre, and had only received one dose of vaccine, the function updates its number of doses to two, removing the patient from the invoking centre. Finally, the patient must be registered at the vaccinated health centre. The function returns True.

- If the patient exists in the invoking health centre, and has not received any doses, the function simply updates the number of doses administered to the patient, if it existed. The function returns True.

- Function ***makeAppointment*** will setup an appointment for a patient. This function receives the following arguments:
  - *name*: The name (surname, name) of a patient.
  - *appointment*: a string with forma “hh:mm” (for example, 08:00, 08:05, 09:55, 14:00, 18:30, 19:55, etc). To simplify the problem, we assume that the minutes of the appointment must be 00 or a multiple of 5. The vaccination times are from 08:00 to 19:55. To simplify the problem, we assume that all appointments are for the same day.
  - schedule: an object instante of *HealthCenter2*, containing all patients who have already an appointment. In this center, the patients should be sorted based on its appointment (string in format h:mm). As it was said above, all the appointments are defined for the same day.
- To emulate the appointment creation for a patient, you need to consider the following use cases:
  - If the patient does not exist in the invoking health center, the appointment will not be created. The function shows a message saying that the patient does not exist and returns False.
  - If the patient exists and has already received both doses, the appointment will not be created. The function will show a message saying that the patient had already been vaccinated and returns False.
  - If the patient exists and has not received both doses, the appointment should be added into *schedule*. We should consider the following cases:
    - If the time is free, the patient (and its appointment) should be added to the *schedule*. The function returns True.
    - If the time is already filled, the function will look for the best time slot instead, that is, the most time-close slot. If there are two slots at the same distance (for example, five minutes before and five minutes later to the original appointment), the function will assign the earlier one (that is, five minutes before). Then, the patient and its appointment should be added to schedule. The function will also return True.

- If there are no available slots, the function will print a message informing of it and will return False.

Note: The same patient is allowed to have several appointments in one day.

Algorithms analysis:

What is the time complexity of each function? Explain best and worst cases. You can include a comment within your code as an answer

Rules:

1. The Lab case must be carried out by a group of two members (both must belong to the same reduced group). Under no circumstances will be allowed groups with more than two members. In addition, the individual group is not allowed since one of the competences to be evaluated will be teamwork. If you don't have a partner, please email your lab teacher.
2. With the description of this phase, the following files are delivered to the student:
   a) The data folder that contains a series of *tsv* files with data to be uploaded the data structures. These files cannot be modified in any case. If you detect an error, please report it to your teacher.
   b) Files BTree.py y BSTree.py which contain binary tree and binary search tree implementations. These files cannot be modified in any case. If you detect an error, please report it to your teacher.
   c) The file *phase2.py* that you shall include your solution of this phase. The file already contains some code to help you get started.
   d) The file *unittest-fase2.py* that contains the necessary test cases to evaluate each one of the functions proposed in the Lab case. This file cannot be modified in any case. If you detect an error, please report it to your teacher. We recommend you do not initially run the whole test cases, but instead you can focus on one function (for example, *vaccine*). It will make easier for you to comment on the rest of the test cases. As you progress through the implementation, you will be able to execute all the test methods in the same execution. This file will give you a provisional grade for phase 1. The final grade will depend on this provisional grade, your grade in the defence and the efficiency of the functions.
3. In the implementation of the Lab case, it is not allowed to use Python structures such as Lists, Queues or Dictionaries, etc. However, you can use the SList or DList implementations, which we have studied in class. If you want to use a stack or queue, its implementation must be based on a linked list (that is, internally the queue or stack cannot be implemented with a Python list).
4. Delivery method: On Aula Global, a task entitled 'Delivery Phase 2' will be posted. The deadline for this first phase will be May 12, 23.59.

5. Delivery format: a zip file whose name is the two NIAs of the students that make up the group, separated by a hyphen. For example, 10001234-10005678.zip. Only one of the two members will be responsible for uploading the solution. The zip shall contain .py files with your solution.
6. Defense: During the face-to-face class on Thursday, May 13. The defense of the Lab case is an oral exam. Attendance is mandatory. If a student does not attend, his/her grade in the Lab case will be NP. During this defense, the teacher will ask each member of the team a series of questions that must be answered individually. Each team member shall be able to discuss any of the decisions made during the design or implementation of any of the functionalities described in the case study. The final grade of the Lab case will be conditioned by the grade obtained in the defense. If a student is not able to discuss and defend certain decisions, he/she will not be qualified with respect to them, even if they are correctly implemented.
7. It is recommended to follow the recommendations described in Zen of Python (https://www.python.org/dev/peps/pep-0020/) and the style guide (https://www.python.org/dev/ peps / pep-0008 /) published on the official Python website.