Project 1

L.EIC Schedules

Elaborating schedules for the L.EIC classes is a complex task. The purpose of this project is not the creation of the schedules, but rather the development of a system to manage schedules after they have been elaborated. The system must include various functionalities related to schedules, such as modifying, searching, viewing, sorting, listing, among others.

The file **schedule.zip** contains real information about L.EIC's schedules for the 1st semester of the academic year of 2022/2023 with anonymized student data. A detailed description of the provided dataset is shown below. Note that a student's schedule may overlap classes if they are neither TP nor PL. That is, there may be overlapping classes between T and TP, between T and T, and between T and PL. A specific example of a schedule is shown below.

2022 - A 1S 1T 2T SP

Horas	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
08:00 - 08:30					MD (T) 1LEIC01 B001	
08:30 - 09:00	AM I (T) 1LEIC01 B001	AM I (TP) 1LEIC01 B329				
09:00 - 09:30				FP (TP) 1LEIC01 B301		
09:30 - 10:00			FP (T) 1LEIC01 B003			
10:00 - 10:30					FP (T) 1LEIC01 B001	
10:30 - 11:00	ALGA (TP) 1LEIC01 B339	ALGA (T) 1LEIC01 B001	FSC (TP) 1LEIC01 B335			
11:00 - 11:30				FSC (T) 1LEIC01 B002	MD (TP) 1LEIC01 B327	
11:30 - 12:00						
12:00 - 12:30						
12:30 - 13:00						
13:00 - 13:30						
13:30 - 14:00						
14:00 - 14:30			PUP (TP) 1LEIC01 B102			
14:30 - 15:00						
15:00 - 15:30						
15:30 - 16:00						
16:00 - 16:30						
16:30 - 17:00						
17:00 - 17:30						
17:30 - 18:00						
18:00 - 18:30						
18:30 - 19:00						
19:00 - 19:30						
19:30 - 20:00						

Dataset

The provided dataset is available in **schedule.zip** in the form of three csv (*comma separated values*) files, as described below.

- File **classes_per_uc.csv**: contains the existing classes in each course unit (UC).
  The first line includes the headers: *UcCode* (code of course unit), *ClassCode* (code of class).

```
UcCode,ClassCode
L.EIC001,1LEIC01
L.EIC001,1LEIC02
```

- File **classes.csv**: contains the schedules of classes.
  The first line includes the headers: *ClassCode* (code of class), *UcCode* (code of course unit), *Weekday* (day of the week), *StartHour* (start time of the class), *Duration* (duration of the class in hours), *Type* (type of class: T, TP, PL).

```
ClassCode,UcCode,Weekday,StartHour,Duration,Type
1LEIC01,L.EIC001,Monday,10.5,1.5,TP
1LEIC02,L.EIC001,Thursday,9.5,1.5,TP
1LEIC03,L.EIC001,Tuesday,9,1.5,TP
```

- File **students_classes.csv**: contains the classes of the students in each UC.
  The first line includes the headers: *StudentCode* (code of student), *StudentName* (name of student), *UcCode* (code of course unit), *ClassCode* (code of class).

```
StudentCode,StudentName,UcCode,ClassCode
202025232,Iara,L.EIC002,1LEIC05
202031607,Gisela,L.EIC004,1LEIC08
202031607,Gisela,L.EIC005,1LEIC08
202079037,Jose Jesualdo,L.EIC023,3LEIC08
```

Conceptually, a *class* is identified by the code of the UC, weekday, start time, duration, and type (T/TP/PL). A *schedule* is composed by several classes. A *student* is identified by a name, and has an associated schedule. A *class* is identified by a code and has a given schedule.

Statement of Work (SoW)

In this project you are invited to use the vector, list, stack, queue, and binary search tree data structures (you can use other structures when relevant). These structures will serve as the basis for the following tasks and are to be employed in the following context:

1. Read and parse the given data. This includes loading the parsed data into the data structures that are most appropriate for the required functionalities. Pay close attention to the linear and hierarchical data structures that you select for each scenario so that you can leverage each data structure's advantages and drawbacks appropriately.

2. Develop a schedule management system. The system must incorporate a comprehensive set of functionalities of schedule management, including those specified for this project and any others deemed relevant. It must have a user-friendly menu enabling users to define their desired criteria. The menu must show available functionalities and their corresponding outputs in a clear, organized, and logical manner, facilitating seamless program utilization and straightforward result interpretation.

3. Perform several listings of the data, both in a total or partial manner. Take into consideration that the user may wish to obtain filtered information from the system. In this sense, it is also important that the listed information be sorted in a logical manner, so as to further facilitate the use of the program. As a source of inspiration, you may consider the following listings:
   i. Consult the schedule of a given student or class;
   ii. Consult the students within a given class, course or year;
   iii. Consult the number of students registered in at least *n* UCs;
   iv. Consult the class/year/UC occupation (sorting by UC, ascending or descending order, …);
   v. Consult the UCs with the greatest number of students;
   vi. …

4. Process requests for new registrations or updates to existing registrations. Consider that students may need to register for a UC or class change. These requests must be analyzed before acceptance and therefore, they must be stored in appropriate data structures to process either one request at a time or all at once.
   Requests can have three different types: *add*, *remove* or *switch*. An *add*/*remove* request occurs when a student wants to join/leave either an UC or a class. Finally, a *switch* request enables students to trade an UC or a class where they were previously registered by a new UC or class.

   i. For UC changes, consider the following rules:
      1. A student cannot be registered in more than 7 UCs at any given time;
      2. There must be at least one class with a vacancy in the new UC;
      3. The resulting schedule will not conflict with the student's original schedule.

   ii. For class changes, consider the following rules:
      1. A student cannot be in more than one class at once for a given UC;
      2. A class can only accept a new student if its capacity has not been exceeded. Consider that there is a maximum capacity **Cap** for classes;

3. A class can only accept a new student if the balance between class occupation is not disturbed. The balance of class occupation is maintained when the difference between the number of students in any class is **less than or equal to 4**;
4. There is no conflict between the student's schedule and the new class's schedule.

5. Maintain a chronological record of all changes made to the system. The system must store accepted requests in the order they were processed, enabling users and administrators to easily undo the most recent actions. Take into consideration that undoing some requests may violate the aforementioned rules, making the undo operation impossible. In these cases, the undo request must be denied and its record removed from the system.

6. Include documentation for the most relevant functions that you implemented, generated using Doxygen. Indicate the time complexity of the most relevant functions or algorithms of your program.

Further than the aforementioned functionalities, you may also implement other functionalities if you consider them relevant.

Expected Results

The program you develop should allow for the registration and management of several different entities and their relationships, making use of both linear (vector, list, stack, queue) and hierarchical data structures (binary search tree).

In this sense, you must pay close attention to the following items:

- Model the relevant entities in a **class**-based system with appropriate interactions between the defined classes;
- Make appropriate use of the **linear** (vector, list, stack, queue) and **hierarchical data structures**;
- Save important information in **files** to allow future use;
- Include **documentation** of the implemented code (generated using Doxygen) and indicate the **time complexity** of the most relevant functions or algorithms of your program.

Your program should also allow for several different listings with the following conditions:

- Listings should be both **total** and **partial** with criteria defined by the user;
- Different **sorting** options should be provided to the user;
- Listing functionalities should make use of both **search** and **sorting** algorithms.

Demonstration and Presentation

Preparing a concise and focused presentation is a very important skill. Therefore, you should structure a short 15-minute presentation of your work, focusing on the following points:

- Present your schedule management system, using illustrative examples to demonstrate the implemented functionalities;
- Highlight and justify the data structures used to represent the provided data set, as well as the data structures used to implement the requested functionalities (modifying, searching, viewing, sorting, listing, …);
- Highlight the most important aspects of your implementation.

You should also elaborate a **PowerPoint presentation** to support your demonstration.

Turn-In Instructions and Deadline

Submit a zip file named **AED2324_PRJ1_G<TN>.zip** on Moodle, where **TN** stands for your class and group numbers (e.g., G**10** represents group **0** of class 2LEIC**01**), with the following content:

- Code folder, containing the program source code.
- Documentation folder, containing HTML documentation, generated using Doxygen.
- Presentation file (**PDF format**) that will serve as a basis for the demonstration.

No submissions will be accepted 24 hours after the deadline. Exceptions apply for justified and documented technical submissions issues.

The deadline is the 3rd of November of 2023 at midnight.