Data Structures and Algorithms

Computer Science Degree

Second Semester Exam. June 27, 2017.

Name: ____ **Group:** __________

Laboratory: ______________ **Desk:** ______________ **DOMjudge User:** ______________

1. **(2.5 puntos)** Extend the linear ADT `Queue` implemented as a single pointer list (as seen in class) adding a new operation with the following `C++` header

```
void shift(unsigned int pos, unsigned int dist);
```

This operation shifts the element in position `pos` of the queue `dist` positions to the left. Take int account that $pos = 1$ is the first element of the queue; $pos = 2$ is the second; and so on.

If the value of `dist` is too high and there are not enough positions in the queue to shift the element, it will be situated in the first position of the queue.

If `pos` is not a valid position in the queue, an error message must be written ("Posicion inexistente").

Apart from coding the operation, you have to justify its complexity.

In order to solve this exercise you cannot allocate or release dynamic memory; modifying the values stored in the queue is not allowed, either.

Entrada										**Salida**						
n	v							pos	dist							
7	1	2	3	4	5	6	7	4	1	1	2	4	3	5	6	7
7	1	2	3	4	5	6	7	4	2	1	4	2	3	5	6	7
7	1	2	3	4	5	6	7	4	3	4	1	2	3	5	6	7
7	1	2	3	4	5	6	7	4	4	4	1	2	3	5	6	7
7	1	2	3	4	5	6	7	4	0	1	2	3	4	5	6	7
7	1	2	3	4	5	6	7	0	3	1	2	3	4	5	6	7
7	1	2	3	4	5	6	7	8	5	1	2	3	4	5	6	7

2. **(3 puntos)** We say a node in a binary tree is *singular* if the sum of the values of its ancestors (including the root) equals the sum of the values of its descendants. Code an algorithm that, provided a binary tree of integer numbers, returns the number of singular nodes that it contains.

Apart from coding the algorithm, justify its complexity.

3. (4.5 puntos) We have been asked to implement a system to manage the admission of patients in the Emergency Service of a Hospital. When a patient arrives to the service, he provides his contact information and he is assigned a unique ID code (a non-negative number) and a seriousness level, depending on its severity (minor, moderate or severe). After that, the patient waits to be seen. Patients are seen taking into account their seriousness level, first, and their order of arrival. Once the patient has been seen, his data are erased from the system. At any moment, a patient can decide to leave the Emergency Service. In this case, there is an exit control where he is asked for his ID and all his data are erased from the system.

The system must be implemented using the ADT `AdmissionManagement` with the following operations:

- `create()`: constructor that creates an empty system.
- `addPatient(id, name, age, symptoms, severity)`: it adds a new patient to the system, with identification code `id`, name `name`, age `age`, description of the symptoms `symptoms` and seriousness level `severity`. In case the code is duplicated, the operation shows an error *"Paciente duplicado"*.
- `patientInfo(id, name, age, symptoms)`: `name`, `age` and `symptoms` return the information associated to the patient with code `id`. In case the code does not exist, the operation shows an error *"Paciente inexistente"*.
- `next(code, severity)`: it stores in `code` and `severity` the code and the seriousness level of the next patient to be seen. As stated previously, the first patients to be seen are the severe ones, then the moderate and, finally, those qualified as minor. Within each level, patients are seen in order of arrival. In case there are no more patients, the operation shows an error *"No hay pacientes"*.
- `morePatients()`: returns `true` if there are more patients waiting, and `false` otherwise.
- `erase(code)`: it erases from the system all the data corresponding to the patient with code `id`. If the patient does not exist, the operation has no effect.

Since this is a critical system, the implementation of the operations must be as efficient as possible. Therefore, you must choose an appropriate representation for the ADT, code the operations and justify the resulting complexity.