University of Mila *2nd ^{year} LMD computer science*
Department of Informatics *Course :ASD3*
Teacher :F.Khennouf *Year 2025-2026*

Mini-Project
Patient Medical Records Management

Context

In a hospital, efficient management of patient medical records is essential to ensure proper medical follow-up. Each patient is identified by a **unique ID**, along with **their last name, first name, and age**.

To optimize search and management operations, all patients are stored in a **Binary Search Tree (BST),** ordered according to the **patient ID**.

Each patient also has a **medical record** containing the history of their medical consultations. Consultations are stored using a singly linked list, where each node represents a medical consultation containing the consultation date, the diagnosis, and the attending physician's name.

You are asked to design **a modular C++ program,** using a procedural approach that allows organizing, updating, and analyzing these data structures.

Tasks to do

1. Define the data structure **Consultation**, implemented as a singly linked list node, representing a medical consultation. The structure must contain the consultation date, the diagnosis, the doctor's name, and a pointer to the next consultation.
2. Define the data structure Patient, grouping the patient's personal information (ID, last name, first name, age) and a pointer to the linked list of consultations.
3. Define the data structure **PatientNode**, representing a node of the Binary Search Tree. Each node must contain a Patient structure and two pointers to the left and right subtrees.
4. Write a function InsertPatient that inserts a new patient into the Binary Search Tree according to their ID.
   Function signature: PatientNode* InsertPatient(PatientNode* root, Patient p);
5. Write a function SearchPatient that searches for a patient using their ID and displays the corresponding medical record if found.
   Function signature: PatientNode* SearchPatient(PatientNode* root, int id);
6. Write a procedure DeletePatient that removes a patient from the Binary Search Tree while respecting the deletion rules of a BST (leaf node, node with one child, node with two children).
   Function signature: PatientNode* DeletePatient(PatientNode* root, int id);
7. Write a function AddConsultation that searches for a patient in the Binary Search Tree and adds a new consultation to that patient's consultation linked list.
   Function signature: void AddConsultation(PatientNode* root, int patientId, Consultation c);
8. Write a function DisplayLastConsultation that displays the most recent consultation of a given patient.
   Function signature: void DisplayLastConsultation(PatientNode* root, int patientId);
9. Write a procedure DisplayStatistics that traverses the Binary Search Tree and displays the following information:

- the total number of registered patients,
- the number of patients aged over 65,
- the last name and first name of the patient who has the highest number of consultations.

The traversal method (in-order, pre-order, or post-order) is left to your choice.
Function signature: void DisplayStatistics(PatientNode* root);

10. A patient is considered inactive if they have not had any consultation for more than a given number of months (duration). Write a procedure RemoveInactivePatients that traverses the Binary Search Tree, identifies inactive patients, and removes them from the tree.
Function signature:PatientNode* RemoveInactivePatients(PatientNode* root, int duration);

11. Stack usage (mandatory). Write the necessary functions to manage a stack of consultations (LIFO order) in order to access the most recent consultation efficiently. Clearly explain how the stack is used to retrieve the last consultation.

Function signatures:

- ***void PushConsultation(Consultation*& stack, Consultation c);***
- ***Consultation PopConsultation(Consultation*& stack);***

Queue usage (mandatory). Write the necessary functions to manage a queue of consultations (FIFO order), representing consultations waiting to be processed (for example, daily validations). Explain the role of the queue in the consultation management process.
Function signatures:

- ***void EnqueueConsultation(Consultation*& front, Consultation*& rear, Consultation c);***
- ***Consultation DequeueConsultation(Consultation*& front, Consultation*& rear);***

Implementation Constraints

1. The program must be written in standard C++.
2. Dynamic memory allocation must be handled correctly using new and delete.
3. The code must be modular, well-structured, and properly commented.

Evaluation Guidelines

- The mini-project will be evaluated according to the following criteria:
- Correct definition of data structures: 20%
- Correct implementation of Binary Search Tree operations: 25%
- Consultation management using linked lists: 15%
- Proper use of stack and queue: 15%
- Tree traversal and statistics computation: 15%
- Code clarity, structure, and documentation: 10%

Deadline

The deadline for submission is 15 days after the mini-project is assigned. Late submissions may be penalized according to the course regulations.

Good Luck