C++ Coding Question: University Management System

Your task is to implement a **University Management System** that models **departments, professors, and students** while demonstrating **OOP principles** like **inheritance, abstraction, encapsulation, polymorphism**, and **dynamic memory allocation**. You will also practice **destructors** to **ensure proper memory management**.

Class Design and Requirements

1. Base Class: `Person` (Abstract Class)

This is an abstract base class that represents a general **person** (either a professor or a student).

Attributes (Private)

- `string name` → The name of the person.
- `int age` → The age of the person.
- `int id` → Unique identifier for a person (auto-generated).
- **Static variable:** `static int idCounter` to auto-increment `id` for every new `Person`.

Methods

- **Constructor**: `Person(const string& name, int age);`
- **Virtual Destructor**: `virtual ~Person();`
- **Pure Virtual Function**: `virtual void displayInfo() const = 0;` (Must be overridden by derived classes)
- **Getter methods**:
  - `string getName() const;`
  - `int getAge() const;`
  - `int getID() const;`
- **Setters (optional)**: Can be added for modifying attributes if needed.

---

2. Derived Class: `Professor` (Inherits from `Person`)

Represents a **professor** with a list of **courses** they teach.

Additional Attributes

- `vector<string> courses;` → Stores courses assigned to the professor.

Methods

- **Constructor**: `Professor(const string& name, int age);`
- **Destructor**: `~Professor();` (Ensure proper memory management)
- **Override `displayInfo()`**:

```
void displayInfo() const override;
```

- **Course Management**
  - `void addCourse(const string& course);`
  - `void removeCourse(const string& course);`
  - `bool teachesCourse(const string& course) const;` → Returns `true` if the professor teaches a given course.
- **Getters**
  - `vector<string> getCourses() const;`

---

3. Derived Class: `Student` (Inherits from `Person`)

Represents a **student** with a list of **grades**.

Additional Attributes

- `vector<int> grades;` → Stores grades.

Methods

- **Constructor**: `Student(const string& name, int age);`
- **Destructor**: `~Student();`
- **Override `displayInfo()`**:

```
void displayInfo() const override;
```

- **Grade Management**
  - `void addGrade(int grade);`
  - `void removeGrade(int index);`
  - `double calculateGPA() const;`
- **Getters**
  - `vector<int> getGrades() const;`

---

4. Class: `Department`

A department manages **professors and students** dynamically.

Attributes

- `string departmentName;`
- `vector<Professor*> professors;`
- `vector<Student*> students;`

Methods

- **Constructor**: `Department(const string& name);`
- **Destructor**: `~Department();` (Must delete all dynamically allocated `Professor` and `Student` objects)
- **CRUD Operations**
  - `void addProfessor(Professor* professor);`
  - `void addStudent(Student* student);`
  - `bool removeProfessor(int professorID);`
  - `bool removeStudent(int studentID);`
- **Search Operations**
  - `Professor* findProfessorByID(int id) const;`
  - `Student* findStudentByID(int id) const;`
  - `vector<Professor*> findProfessorsByCourse(const string& course) const;`
- **Display**
  - `void listProfessors() const;`
  - `void listStudents() const;`
  - `void listMembers() const;` (Lists all members)

Implementation Details

Encapsulation

- All attributes are **private**.
- Public methods provide **controlled access**.

Inheritance

- `Professor` and `Student` **inherit** from `Person`.

Polymorphism

- `displayInfo()` is **virtual** and overridden in `Professor` and `Student`.

Dynamic Memory Management

- `Department` **stores pointers** to `Professor` and `Student`.
- `Department`'s **destructor** must **delete all allocated objects**.

Expected Output Example

```
int main() {
    Department* csDept = new Department("Computer Science");

    Professor* prof1 = new Professor("Dr. Smith", 45);
    prof1->addCourse("Data Structures");
    prof1->addCourse("Algorithms");

    Student* student1 = new Student("Alice", 20);
    student1->addGrade(85);
    student1->addGrade(90);

    csDept->addProfessor(prof1);
    csDept->addStudent(student1);

    cout << "=== Listing All Members ===" << endl;
    csDept->listMembers();

    cout << "=== Searching for Professors Teaching 'Algorithms' ===" << endl;
    vector<Professor*> algProfessors = csDept-
>findProfessorsByCourse("Algorithms");
    for (const auto& p : algProfessors) {
        cout << p->getName() << " teaches Algorithms." << endl;
    }

    cout << "=== Removing Student Alice ===" << endl;
    csDept->removeStudent(student1->getID());

    delete csDept; // Ensures all dynamic memory is freed
    return 0;
}
```

Your Task

1. **Implement all classes with correct attributes and methods.**
2. **Ensure proper memory management** with destructors.
3. **Use vectors for storage instead of raw arrays.**
4. **Demonstrate polymorphism with `displayInfo()`.**
5. **Implement search and CRUD operations.**