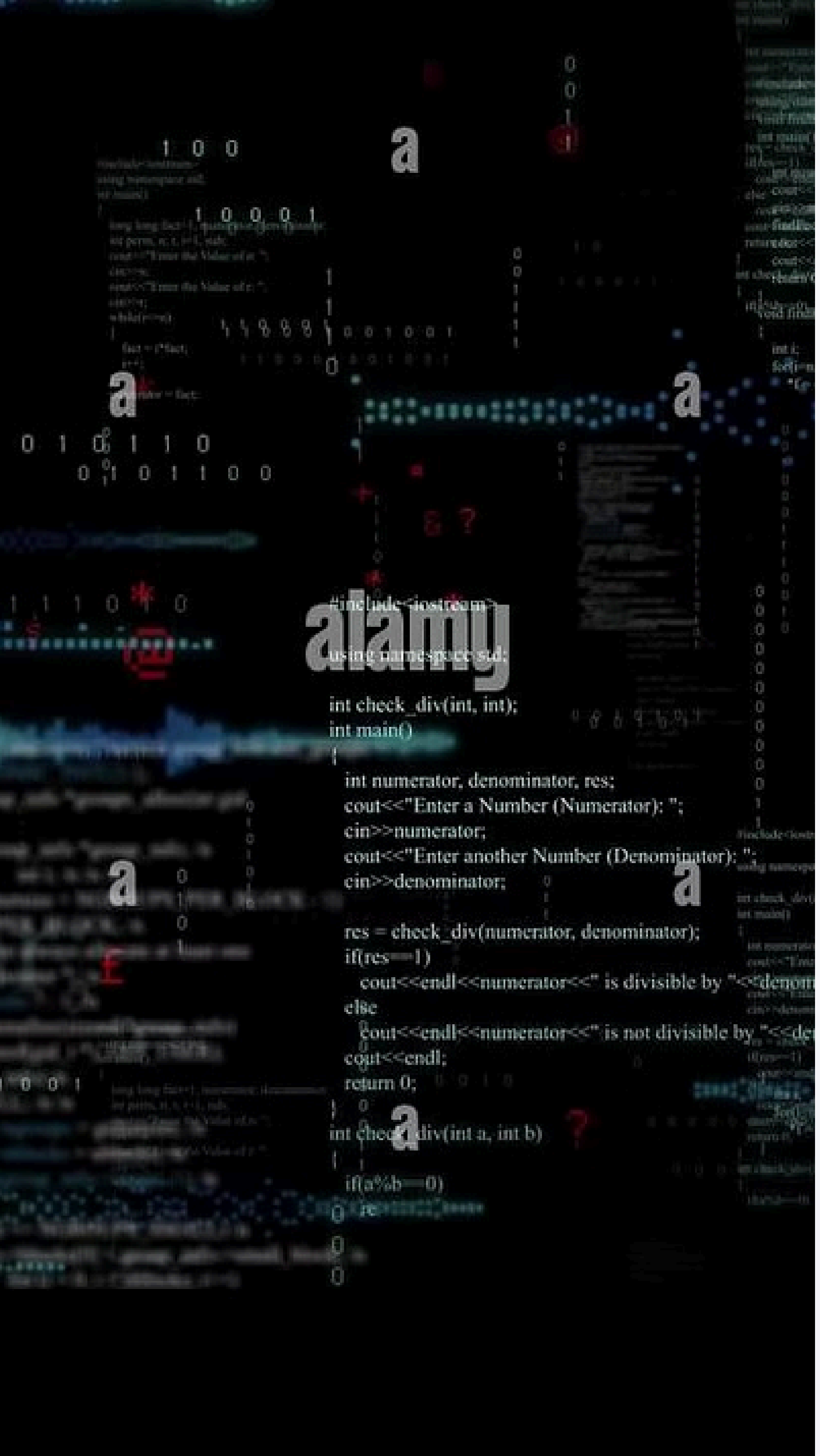
University Management System

OOP Project in C++

Ahmad Jamal Almashharawi

Under the supervision of: **Dr. Samer Hanna**

[redacted]



```
#include<iostream>
using namespace std;

int check_div(int, int);
int main()
{
  int numerator, denominator, res;
  cout<<"Enter a Number (Numerator): ";
  cin>>numerator;
  cout<<"Enter another Number (Denominator): ";
  cin>>denominator;

  res = check_div(numerator, denominator);
  if(res==1)
    cout<<endl<<numerator<<" is divisible by "<<denom
  else
    cout<<endl<<numerator<<" is not divisible by "<<de
  cout<<endl;
  return 0;
}
int check_div(int a, int b)
{
  if(a%b==0)
    re
```

Project Overview

A comprehensive management system built using Object-Oriented Programming principles in C++ to manage university entities including students and professors, demonstrating core OOP concepts and advanced practices.

Record Management

Efficiently manage student and professor data.

Data Persistence

file-based storage for all records.

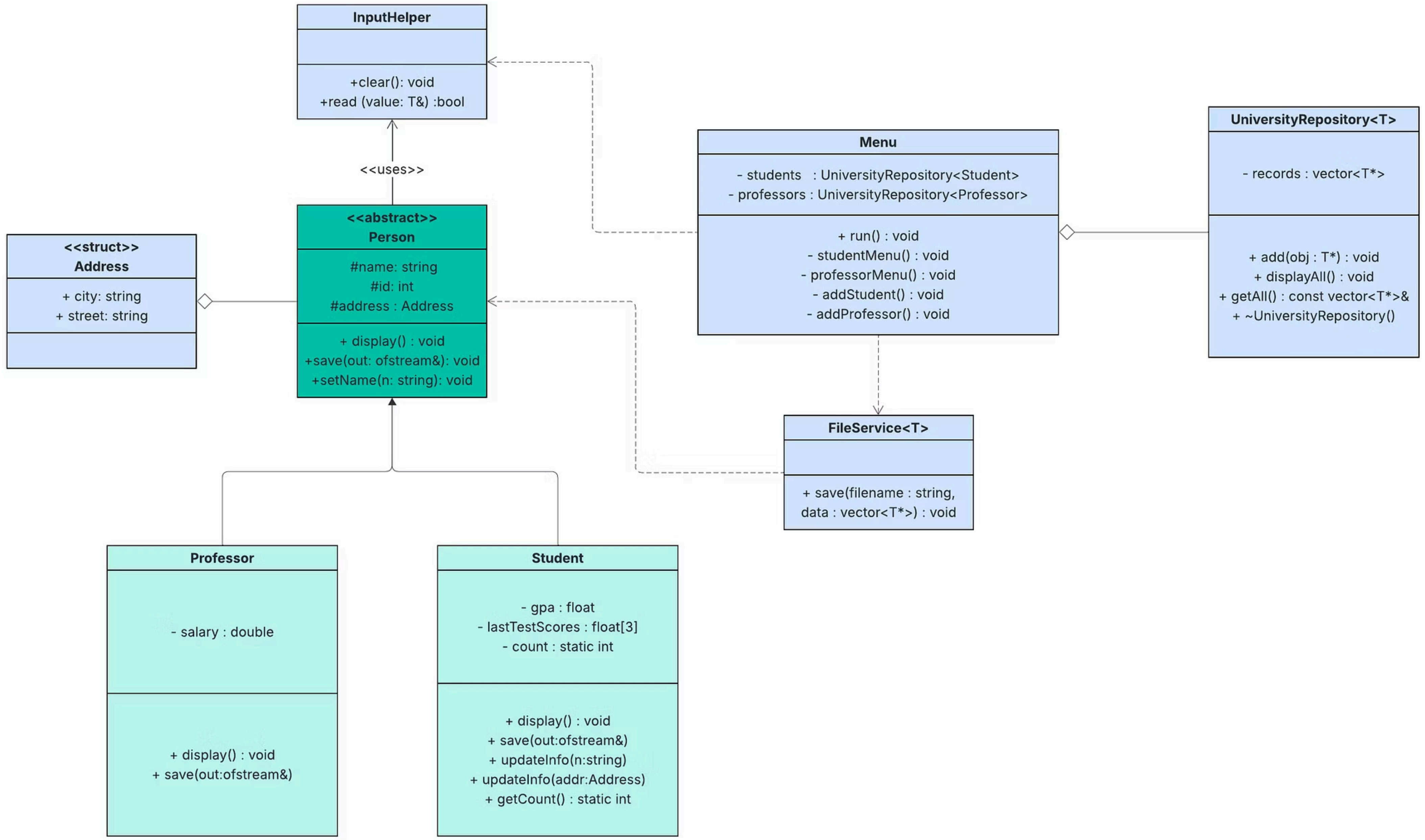
Robust Handling

Integrated exception handling and input validation.

Generic Repository

Template-based pattern for flexible data operations.

[redacted]

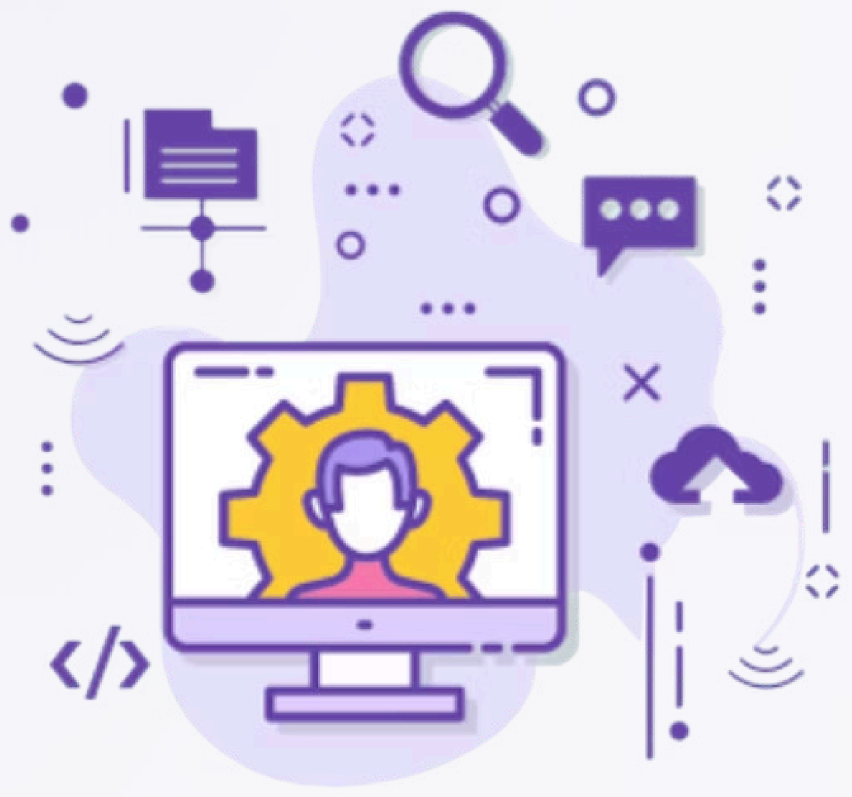

InputHelper
+clear(): void
+read (value: T&) :bool
<<uses>>
<<abstract>>
Person
#name: string
#id: int
#address : Address
+ display() : void
+save(out: ofstream&): void
+setName(n: string): void
<<struct>>
Address
+ city: string
+ street: string
Menu
- students : UniversityRepository<Student>
- professors : UniversityRepository<Professor>
+ run() : void
- studentMenu() : void
- professorMenu() : void
- addStudent() : void
- addProfessor() : void
UniversityRepository<T>
- records : vector<T*>
+ add(obj : T*) : void
+ displayAll() : void
+ getAll() : const vector<T*>&
+ ~UniversityRepository()
FileService<T>
+ save(filename : string,
data : vector<T*>) : void
Professor
- salary : double
+ display() : void
+ save(out:ofstream&)
Student
- gpa : float
- lastTestScores : float[3]
- count : static int
+ display() : void
+ save(out:ofstream&)
+ updateInfo(n:string)
+ updateInfo(addr:Address)
+ getCount() : static int

[redacted]

System Architecture - Core Classes

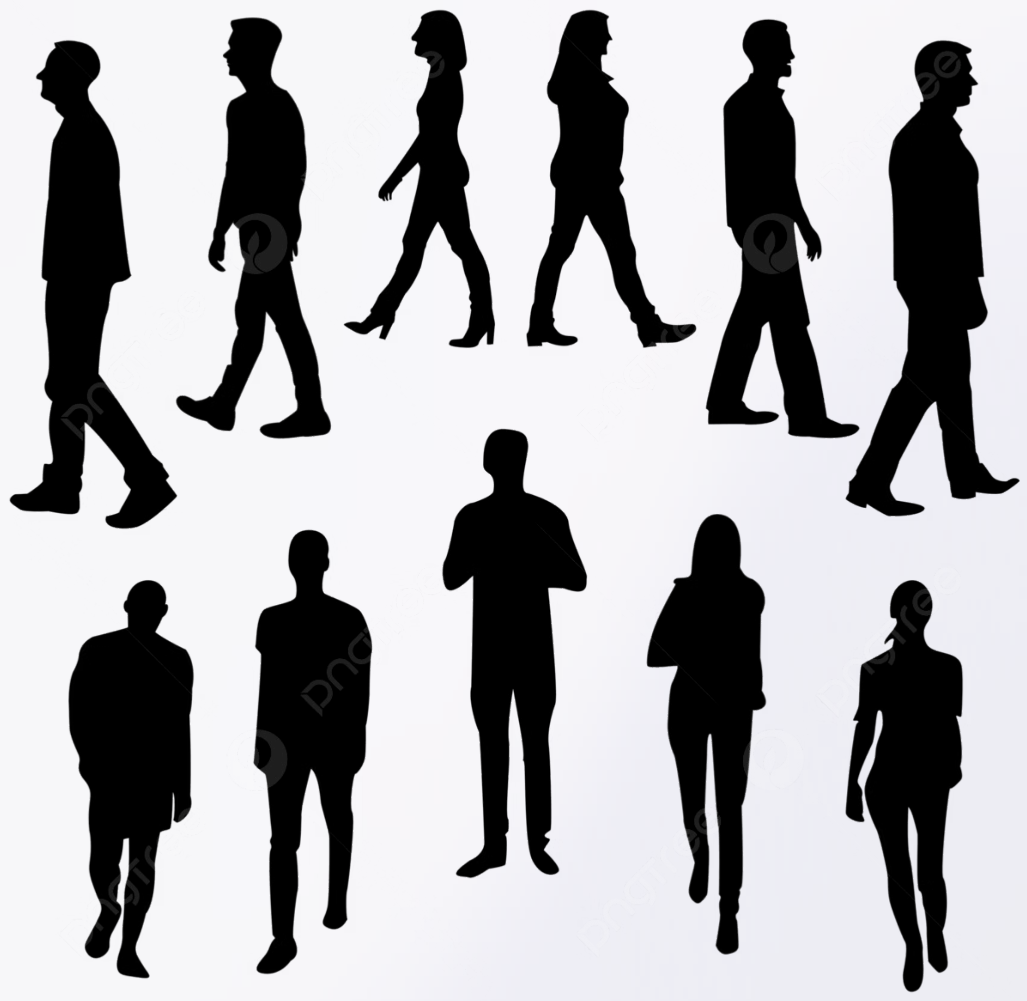
Utility Layer

- **InputHelper:** Safe and templated input handling.
- **Address (Struct):** Data structure for location information.



Domain Model

- **Person (Abstract):** Base class with pure virtual methods.
- **Student:** Inherits Person, manages GPA and quiz scores.
- **Professor:** Inherits Person, manages salary details.

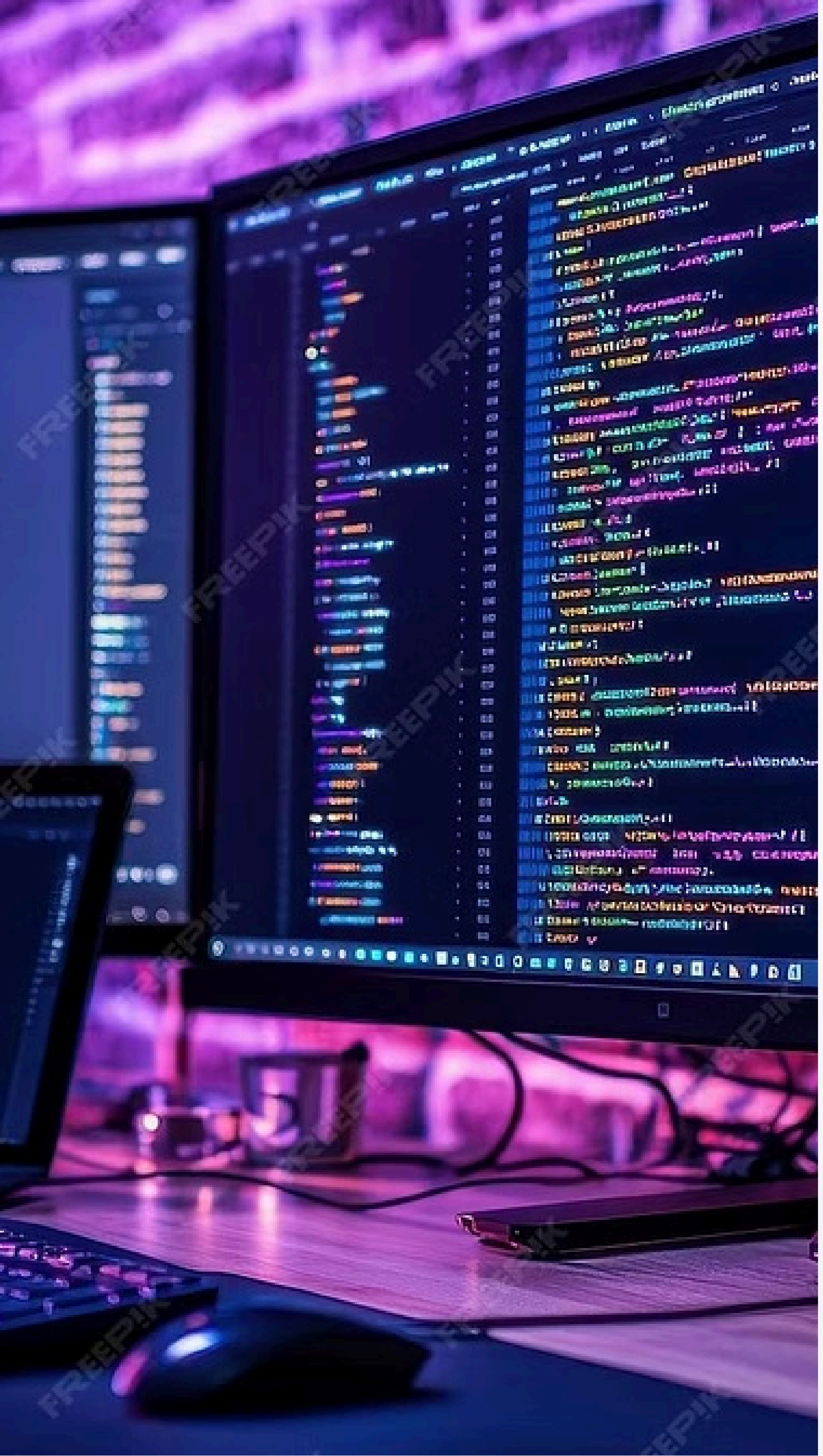


Repository & Services

- **UniversityRepository:** Template-based data storage.
- **FileService:** Handles all file I/O operations.

These core components form the backbone of the University Management System, ensuring modularity, scalability, and maintainability.

[redacted]

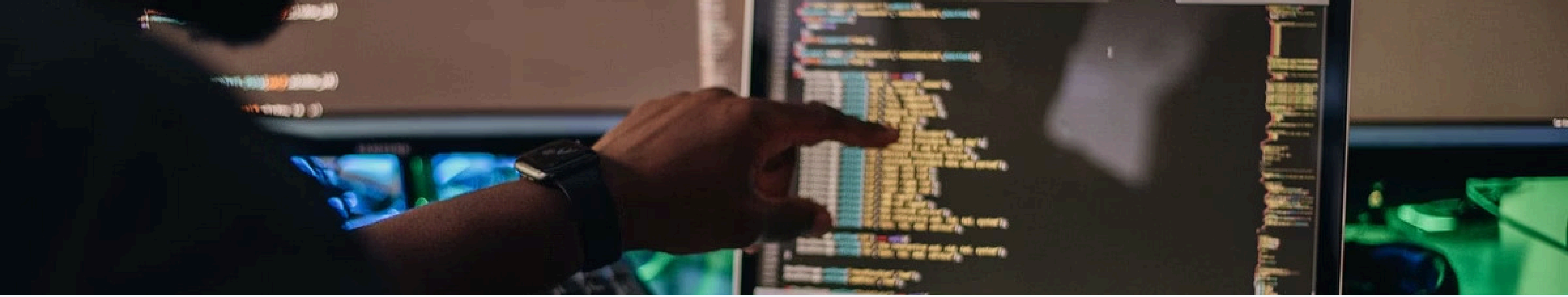


OOP Concepts Demonstrated

- **Structs:** Used for Address to encapsulate location data.
- **Arrays:** Employed for managing lastQuizScores[3].
- **Pointers:** Utilized for dynamic memory allocation.
- **Classes:** Person, Student, Professor as core entities.
- **Constructors:** Default and parameterized for object initialization.
- **Destructors:** Essential for proper memory cleanup.
- **Static Members:** e.g., Student::count for tracking instances.
- **Inheritance:** Student and Professor extending Person.
- **Composition:** Person class owning an Address object.
- **Method Overriding:** Polymorphic display and save methods.
- **Method Overloading:** e.g., updateInfo with varying parameters.
- **Templates:** Generic Repository and FileService for type safety.
- **Exception Handling:** Robust validation and error management.

And more…

[redacted]



Advanced Concepts - Self-Studied

Version Control (GitHub)

Managed source code, tracked changes, and maintained a structured workflow.

SOLID Principles Applied

Implemented Single Responsibility, Open/Closed, and Liskov Substitution principles.

Advanced Input Validation

Developed InputHelper for custom validation, preventing runtime errors and repetitive logic.

File Stream Operations

Utilized C++ fstream for external data persistence in .txt files, ensuring data integrity.

[redacted]

Key System Features

Safe Input Handling

Template-based validation with automatic error recovery.

Data Persistence

Records saved to text files in a structured format.

Memory Management

Automatic cleanup via destructors and RAII, preventing leaks.

Polymorphism

Abstract base class design enabling runtime polymorphic behavior.

Exception Handling

Robust validation for GPA range and file operations.

[redacted]

User Interface Flow

Main Menu

Choose between Student or Professor management.

Student Menu

Add, View, or Save student records.

Professor Menu

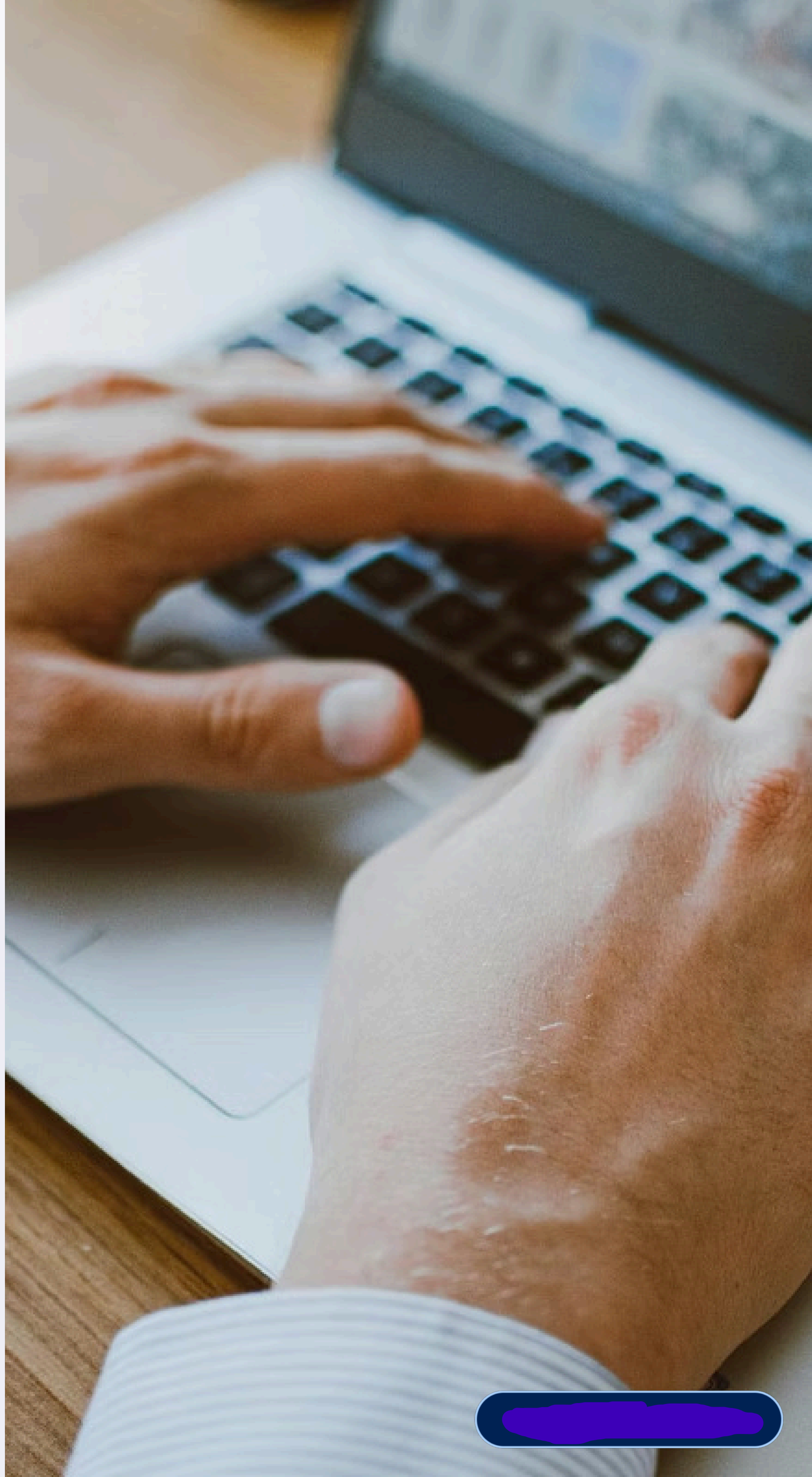
Add, View, or Save professor records.

Key Features

- Input validation at every step.
- Comprehensive exception handling.
- Easy navigation with numbered menus.
- Clear user prompts and feedback.

How to Use

1. Launch program → Main menu.
2. Select Student (1) or Professor (2).
3. Add records with validated input.
4. View all records on screen.
5. Save records to .txt files and exit safely.



[redacted]

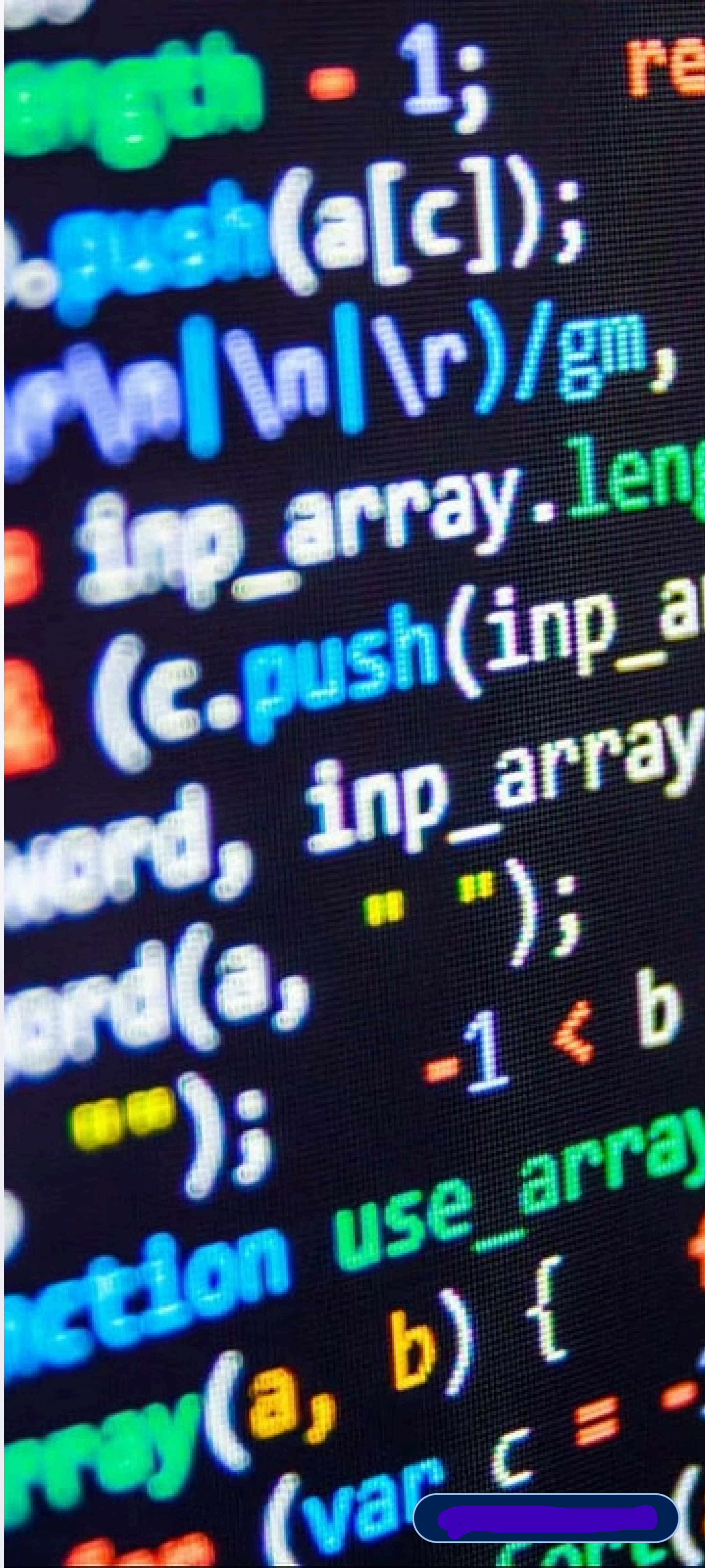
Code Quality Standards & Best Practices

Design Patterns Applied

- **Repository pattern** for abstracting data access logic, ensuring data persistence is independent of business logic.
- **Template method pattern** for generic operations like `CrudOperations` to define algorithms with deferrable steps to subclasses.
- **Abstract factory pattern** to create families of related or dependent objects without specifying their concrete classes (e.g., `Student` and `Professor`).
- **Strategy pattern** implemented for different validation rules, allowing easy extension of input validation.

Error Handling

- Robust try-catch blocks for all potential runtime exceptions, including file I/O and user input parsing.
- Comprehensive input validation at every entry point (e.g., type checking, range validation, format validation), ensuring data integrity.
- Graceful recovery mechanisms to prevent application crashes and maintain a stable state after an error.
- Error messages that guide the user on how to resolve issues, instead of technical jargon.



```
ength - 1;   re
.push(a[c]);
r\n|\n|\r)/gm,
inp_array.len
(c.push(inp_a
ord, inp_array
ord(a, "");
"");   -1 < b
ction use_array
rray(a, b) {
(var c = -
```

[redacted]

Conclusion & Demonstration

1. **Key Achievements**

   Successfully implemented all required OOP concepts and created a functional, error-free system.

2. **Advanced Application**

   Applied SOLID principles, design patterns, and comprehensive documentation.

3. **Extended Learning**

   Self-studied advanced concepts beyond course scope, demonstrating initiative.

Ready for Live Demonstration!

[redacted]