# Secure Card System

You are tasked with implementing the business logic for a card access system for a government building in C++. The system will allow users and administrators to interact with the system through a console-based interface and handle access based on the floors and employees clearance levels.

# System Requirements

All requirements for the system.

## 1. Functional Requirements (G)

All the business logic for the system.

### 1.1. Role based

When the program starts, the user should be prompted with choosing either to enter as a user, admin or exit the program.

```
----- SECURE CARD SYSTEM -------
[1] User
[2] Admin
[0] Exit
>
```

### 1.2. User login

The user should be prompted to enter their employee ID or their name.

```
----- SECURE CARD SYSTEM -------
* User login *
Enter employee id or name:
```

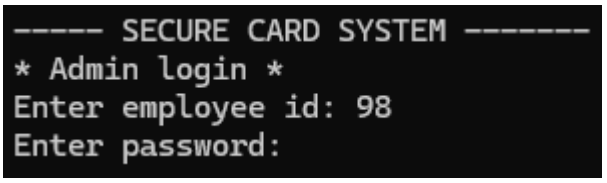
### 1.3. User Operations

When logged in as a user they should be allowed the following functions:

- List all available floors
    - Access floor (the program should both log and print out the result)
    - Back
- Show personal information about the logged in employee
  (name, email, phone, card id and card clearance level).
    - Change information (only name, phone number and email)
    - Back
- Log out (back to start of program)

## 1.4. Admin login

When logging in as admin, they need to enter their employee id and then their password.




## 1.5. Admin Operations

When logged in as an admin they should be allowed the following functions:

- List all floors
    - Choose floor
        - List access history of that floor
        - Change floor information (name and clearance level)
- List all users
    - Choose user (by id or name)
        - Change user information (name, email, phone)
        - Delete user - should also delete their card
- Create a new user - also needs to create a card for the user.
- Log out (back to start of program)

## 1.6. User

Contains:

- Id (must be unique)
- Name
- Email
- Phone number
- Card (not null)

## 1.7. Admin

Contains:

- Id (must be unique)
- Password
- Name
- Email
- Phone number
- Card

## 1.8. Floor

Contains:

- Id - must be unique
- Name - must be unique
- ClearanceLevel
- AccessHistory

### 1.9. Card

A card can only, and must, have one owner. There should be no owners sharing cards and no floating cards without owners.
Contains

- Id (must be unique)
- ClearanceLevel

## 2. Non-functional Requirements (G)

All non-business logic requirements for this system.

### 2.1. Persistence

All data shall be persistent, read and saved in CSV format. When the program starts - it should read from the data sources and not generate elements based on hardcoded values in the program.

### 2.2. Validation

The following data **must** be validated

- Password - minimum 8 characters long, 1 uppercase, 1 lowercase, 1 number and 1 special character.
- Email - local@domain.tld (at least one dot in the domain).
- Phone number - 07XXXXXXXX or +467XXXXXXXX

### 2.3. Exceptions

All exceptions for input operations should be handled and not crash the system.

### 2.4. Logging

Every floor should have an access-log, displaying which people have accessed the floor, at what time and if they were authorized to enter. The logs don’t need to be persistent.

## 3. Non-functional Requirements (VG)

### 3.1. Cache (Templated)

Small cache for the 10 latest searches in the system. The search functions should always look in the cache first.

### 3.2. System filler

Create a program or script that generates 1000 validated users (with cards) and 1 admin to handle the system.

### 3.3. Multithreading

The user shall at all times be able to put in the command “scs -save” which opens a thread that saves all of the current data in memory to our persistent memory.

# Grading

## 1. Deliverable

- The complete program must be submitted via a private GitHub repository and shared with the examiner (@lafftale1999).
- The student must also perform a live demonstration in a small group, where the program is run and key features are shown.
- During the demo, the student must be able to explain key design and implementation decisions, as well as answer questions regarding the codebase.

## 2. Grading

### 2.1. Pass (G)

- All functional and non-functional requirements marked as G are correctly implemented and working as intended.
- The system runs without crashes and fulfills the specified user and admin operations.
- The student can clearly explain how the program works.

### 2.2. Pass with distinction (VG)

- All requirements for Grade G are met in full.
- All VG-level requirements are fully implemented and functional.
- The student demonstrates:
    - Consistent naming conventions
    - Good folder structure
    - Efficient and non-repetitive code following DRY principles
    - Good use of STL
- The student can articulate and justify their design and implementation decisions, showing an understanding of advanced C++ concepts and system design principles.

# Learning objectives

- Describe the C++ language and its syntax, structure, functions, and relevant concepts
- Explain memory management, references, and pointers within C++
- Describe relevant advanced data structures and algorithms
- Describe libraries and templates in C++
- Describe object-oriented programming and classes, objects, inheritance, and polymorphism
- Explain multi-threaded software
- Develop applications in C++ at a specialized level while utilizing relevant functions, libraries, and templates
- Plan and develop multi-threaded software and being able to apply advanced data structures and algorithms
- Show testing, debugging and troubleshooting of applications in C++
- Apply and compare procedural and object-oriented programming techniques
- Manage and comprehend dynamic and static memory management
- Communicate commitments and solutions in the development of applications in C++ at a professional level
- Independently develop and debug software applications in C++ based on given requirements
- Independently identify and address a problem and be able to choose the appropriate tools and methods to solve the issue at hand
- Independently drive development in C++ by selecting suitable tools and methods like CMakeFiles.