Department of Electronics and Communication Engineering

CSE115 – Programming Language I

A Project Report On

Lost and Found Item Management

Submitted by

Name: Md. Mahinur Hyder Antor.

ID: 2412755642

Name: Mst Khatune Jannat Tonni

ID: 2413336043

Date: 12/12/2025

Submitted to

Shahriyar Zaman Ridoy

Department of Electrical and Computer Engineering, North South University.

PROJECT REPORT: Lost and Found Item Management System

1. Introduction

In high-traffic environments such as university campuses, airports, railway stations, and corporate offices, the loss of personal belongings is a common occurrence. Managing these lost items manually through physical logbooks or designated "lost and found" boxes is often inefficient, insecure, and prone to human error. There is often no transparent way for a person who lost an item to know if it has been found without physically visiting a specific location.

The **Lost and Found Item Management System** is a software solution developed in the C programming language to address this issue. It serves as a digital bridge between individuals who find lost items ("Finders") and those who have lost them ("Owners"). By digitizing the record-keeping process, the system ensures data integrity, quick retrieval of information, and a secure verification process managed by an administrator.

2. Proposed Project and Objective

2.1 Proposed Project Scope

The proposed system is a console-based application that creates a centralized database of lost items. It replaces the traditional pen-and-paper method with a file-based storage system. The system supports two distinct user roles:

1. **General User:** Can report items they have found and search/claim items they have lost.
2. **Administrator:** Acts as the gatekeeper. They review claims made by users and approve the return of items to ensure they are going to the rightful owners.

2.2 Objectives

The primary objectives of this project are:

- **Automation:** To automate the recording and tracking of lost items, reducing manual effort.
- **Data Persistence:** To ensure that records of lost items and user accounts are saved permanently using file handling (`.txt` files), allowing data recovery even after the program is closed.
- **Security & Verification:** To implement a workflow where an item cannot be marked as "returned" without Administrator approval, thereby preventing theft or false claims.
- **Transparency:** To allow users to track the status of their claims (e.g., waiting for approval vs. returned).
- **User Management:** To implement a registration and login system that restricts access to authorized personnel only.

3. Programme Specifications

This section details the technical environment required to develop and execute the project.

3.1 Hardware Requirements

Since the project is a text-based console application, it is lightweight and requires minimal hardware:

- **Processor:** Intel Pentium 4 or higher / AMD equivalent.
- **RAM:** 512 MB or higher.

- **Storage:** Minimal space required (less than 5MB for code and text files).

3.2 Software Requirements

- **Operating System:** Windows (Required due to the use of `<conio.h>` for `getch()` and `system("cls")`).
- **Programming Language:** C Standard (C99 or C11).
- **Compiler:** GCC (MinGW) or any C-compatible compiler.
- **IDE/Editor:** VS Code, DevC++, Code::Blocks, or Turbo C.

3.3 Data Storage Specifications

The system avoids complex database engines (like SQL) in favor of lightweight text files:

- **`users.txt`:** Stores `Username`, `Password`, and `Role` (Integer).
- **`items.txt`:** Stores `ID`, `Name`, `Category`, `Description`, `FoundBy_User`, `ClaimedBy_User`, and `Status`.

4. Programme Topics

The development of this system integrates several core concepts of Computer Science and C Programming:

- **Structure (`struct`) Data Types:**
  - Used to create custom data models.
  - `struct User`: Encapsulates user credentials and access rights.
  - `struct Item`: Encapsulates all metadata regarding a lost object (Name, Category, Status codes).
- **File Handling (I/O):**
  - The program uses `fopen()`, `fscanf()`, `fprintf()`, and `fclose()` to read and write data. This ensures the system acts as a persistent database rather than just a runtime calculator.
- **Arrays and Strings:**
  - **Arrays of Structures:** `User users[MAX_USERS]` and `Item items[MAX_ITEMS]` are used to load the database into RAM for fast processing.
  - **String Manipulation:** Functions from `<string.h>` (such as `strcmp` for password comparison and `strcpy` for assigning data) are utilized extensively.
- **Preprocessor Directives:**
  - Macros `#define MAX_USERS 100` are used to define buffer sizes, making the code easier to maintain and modify.
- **Modular Programming:**
  - The code is divided into specific functions (`loadData`, `saveData`, `adminMenu`, `userMenu`) rather than writing everything in `main()`. This promotes code reusability and debugging ease.

5. Implementation

The system logic follows a specific lifecycle for every item reported.

5.1 Initialization and Authentication

1. **Loading Data:** Upon startup, the `loadData()` function opens the text files and populates the program's memory arrays.
2. **Login/Register:**
   a. New users register, and their details are appended to the user array and file.
   b. During login, the system checks credentials. Crucially, it checks the **Role ID**:
      i. If `Role == 1`: Redirect to User Menu.
      ii. If `Role == 2`: Redirect to Admin Menu.

5.2 The Item Lifecycle (Status Logic)

The core of the implementation relies on an integer variable `status` inside the `Item` structure:

- **Stage 1: Reporting (Status = 0)**
  - A user logs in and selects "Throw in lost item".
  - They enter details (Name, Category).
  - The system assigns `Status = 0` (Available) and records the finder's username.
- **Stage 2: Claiming (Status = 1)**
  - Another user (the owner) logs in and selects "Look for item".
  - They see a list of items where `Status == 0`.
  - They select an item by ID.
  - The system updates `Status = 1` and records the claimer's username. The item is now locked and "Pending Approval."
- **Stage 3: Approval/Return (Status = 2)**
  - The Administrator logs in.
  - They view a list of "Pending Requests" (Items where `Status == 1`).
  - The Admin verifies the claim and approves it.
  - The system updates `Status = 2`. The item now appears in the user's "Belonging Items" list as "Returned."

6. Pros and Cons

6.1 Pros (Advantages)

1. **Enhanced Organization:** Replaces messy physical logbooks with a structured digital list.
2. **Role-Based Access Control (RBAC):** By separating Admins from Users, the system prevents unauthorized users from falsely marking items as "returned" without verification.
3. **Cost-Effective:** The system requires no expensive server infrastructure or paid database licenses; it runs on standard text files.
4. **Portability:** The source code and data files can be copied to any Windows computer and run immediately without installation.
5. **User Accountability:** Every item tracks *who* found it and *who* claimed it, creating an audit trail.

6.2 Cons (Limitations)

1. **Static Memory Allocation:** The system uses fixed arrays (`MAX 100`). If the number of users or items exceeds 100, the program must be recompiled with a larger number, or it will crash.

2. **Single-User Environment:** As a console app handling local files, only one person can use the specific computer at a time. It is not a web-based application accessible via the internet.
3. **Platform Dependency:** The code relies on Windows-specific libraries (`conio.h`), making it incompatible with Linux or MacOS without code modification.
4. **Security:** Passwords are stored in plain text in `users.txt`. In a real-world commercial deployment, hashing encryption would be required.

7. Conclusion

The **Lost and Found Item Management System** project successfully demonstrates the capability of C programming to solve real-world logistical problems. By implementing file handling, data structures, and logical workflows, the system provides a functional prototype for managing lost property.

It fulfills the primary objective of creating a digital, secure, and persistent record of items. While the current version allows for 100 items and operates locally, the logical foundation is solid. Future iterations of this project could involve migrating to a SQL database for unlimited storage, creating a Graphical User Interface (GUI) for better aesthetics, and hosting the data on a cloud server to allow remote access via the internet.