LOST AND FOUND INVENTORY TRACKER SYSTEM

Verano, James Daniel M.
Baula, Djordje Blythe M.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

In school organizations, when an Item is accidentally left or misplaced by owners, some people that may potentially find the item, can surrender it to the Lost & Found Office or known as the repository. Lost & Found offices are created for easier consultations for missing items that can be surrendered or potentially be found by maintenance staff and are stored to the repository. (Vocabulary, 2025) Although items in the repository do not last for too long, a probability of missing items due to unorganized and heavy load of items making it hard to keep track of every item in the repository, items getting mixed up, and/or it does not reach the correct owner which causes conflicts from false item claims. (Markham, 2024)

Due to the complexity of keeping track of all items that are stored and to those who are claiming items that are not theirs or items that are missing and getting mixed up in the repository, some of the people's item can no longer be retrieved due to the lack of tracking history of claimings and name of the receivers, and item log and tracking. (Castro et al., 2022) This causes possible irrelevance of the office, where its core purpose is no longer necessary or its function no longer seems to be helpful to some people / students. (Markham, 2024)

With the information above, Engineers came up with a new and innovative solution, a system that keeps track of all logs of all items surrendered and the history log of all claimed items and the claimer's information for people claiming the wrong items and to avoid people claiming the wrong items. (Markham, 2024) The claimer's information is used ONCE conflict may arise that the potential claimed items are not theirs. This may occur only when 2 individuals claim that the items are theirs. Thus, calling both individuals to have a consultation to undergo the correct procedure of identifying who is the owner of the items / belongings.

The Project

This project aims to address common challenges faced by different individuals and lost & found offices that provides services that handles and manages all missing items or/and misplaced items of individuals. Engineers have created a digital approach, for logging, categorizing, and tracking all surrendered items and their corresponding claim histories by individuals who are believed to be the owner of the claimed items. This system is called the **Lost & Found Inventory Tracker System (LoFITS).** Through the implementation of LoFITS, Engineers ensure that this project will be able to address the common challenges such as misplacement, duplication, or false claiming of items. With the System providing log history of every process of surrendering and claiming properties, every transaction of claimed properties is properly recorded and traceable. With the existence of the **LoFITS**, Engineers are confident that the system will be able to maintain a detailed log of all items currently in the repository and to also maintain and provide the complete history of all items that have been claimed by an individual, including their personal information to be used for tracking in cases of conflict.

Objectives

Engineers main objective is to implement the project **Lost & Found Inventory Tracker System (LoFITS)**, which intends to solve the problem of the teachers / students in correlation of recovering and keeping track of lost and claimed items that are in the lost & found. **LoFITS** aims to improve the efficiency, accuracy, and reliability of managing lost and found items within a school or organizational setting. Furthermore, this system provides transparency and accountability in terms of recording the claimer's identification details as a preventive measure in cases of dispute or conflict, the office can easily trace the involved individuals and initiate proper verification procedures. This general objective contains different features which implement different purposes:

1. **history log of their personal information and claimed items** from the Lost & Found Inventory for a secured preventive measure in case of conflict.
2. **Item log of what's currently in the Lost & Found System** to keep track of how many items are remaining in the system.
3. **Organized collection of collected items in the lost & found inventory system** to be able to follow through what types of items are in the system. (Accessories / clothes / electronics).

Flowchart of the System

The flowchart of the system presents how LoFITS functions and understands how the program works by showing the step-by-step process of logging, claiming, and displaying items, as well as managing user login and claim history in an organized manner.

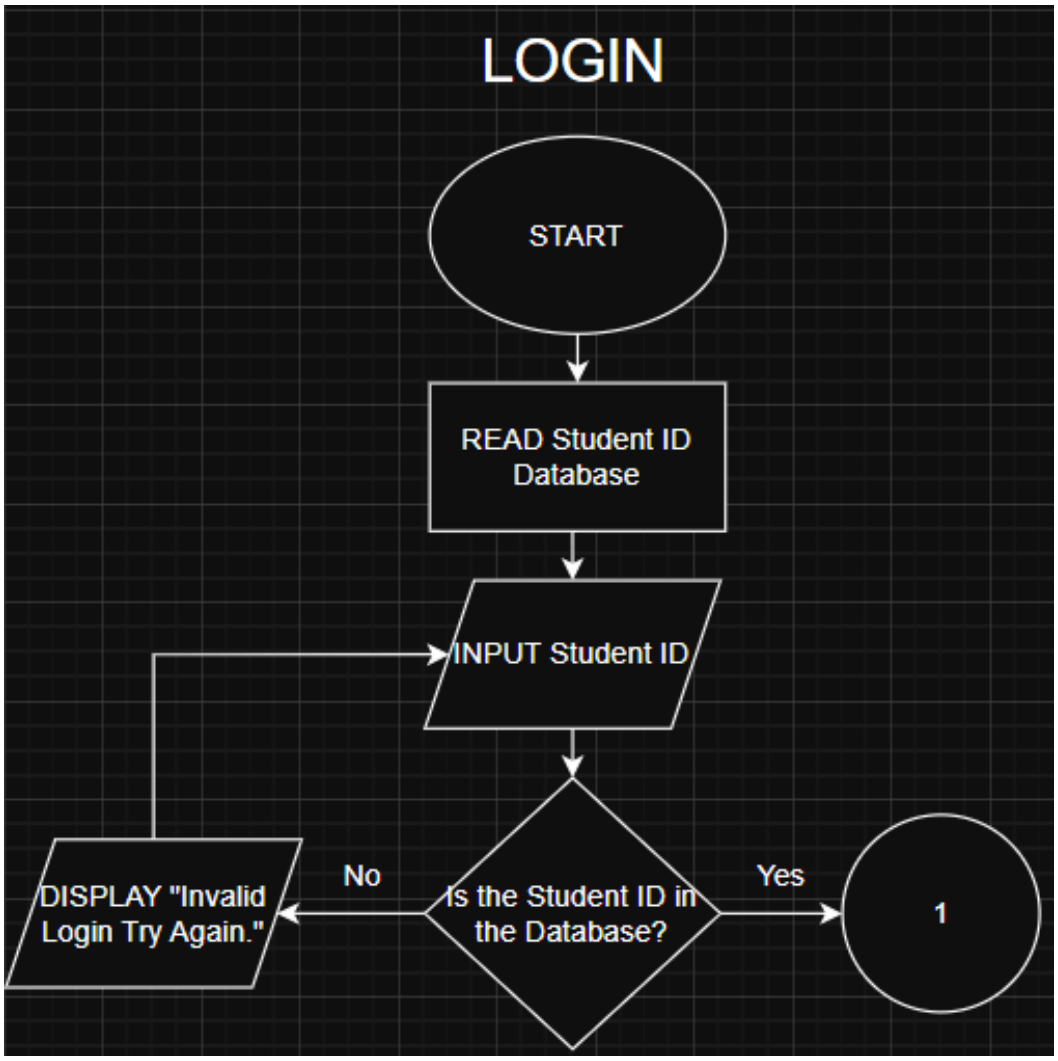

Figure 1. Login Flowchart

Figure 1 illustrates the Login Flowchart. In executing the file, the program first initiates by reading from the Student ID database to authenticate all validIDs and users and to authorize their access. After processing the initialization, the program will prompt a login input to enter their credentials to be able to grant access to the program. If the entered credentials are inside the student ID database, then the program will proceed to the main menu of the program (represented by connector 1), otherwise it will display an invalid message and prompt the user to input their credentials again.

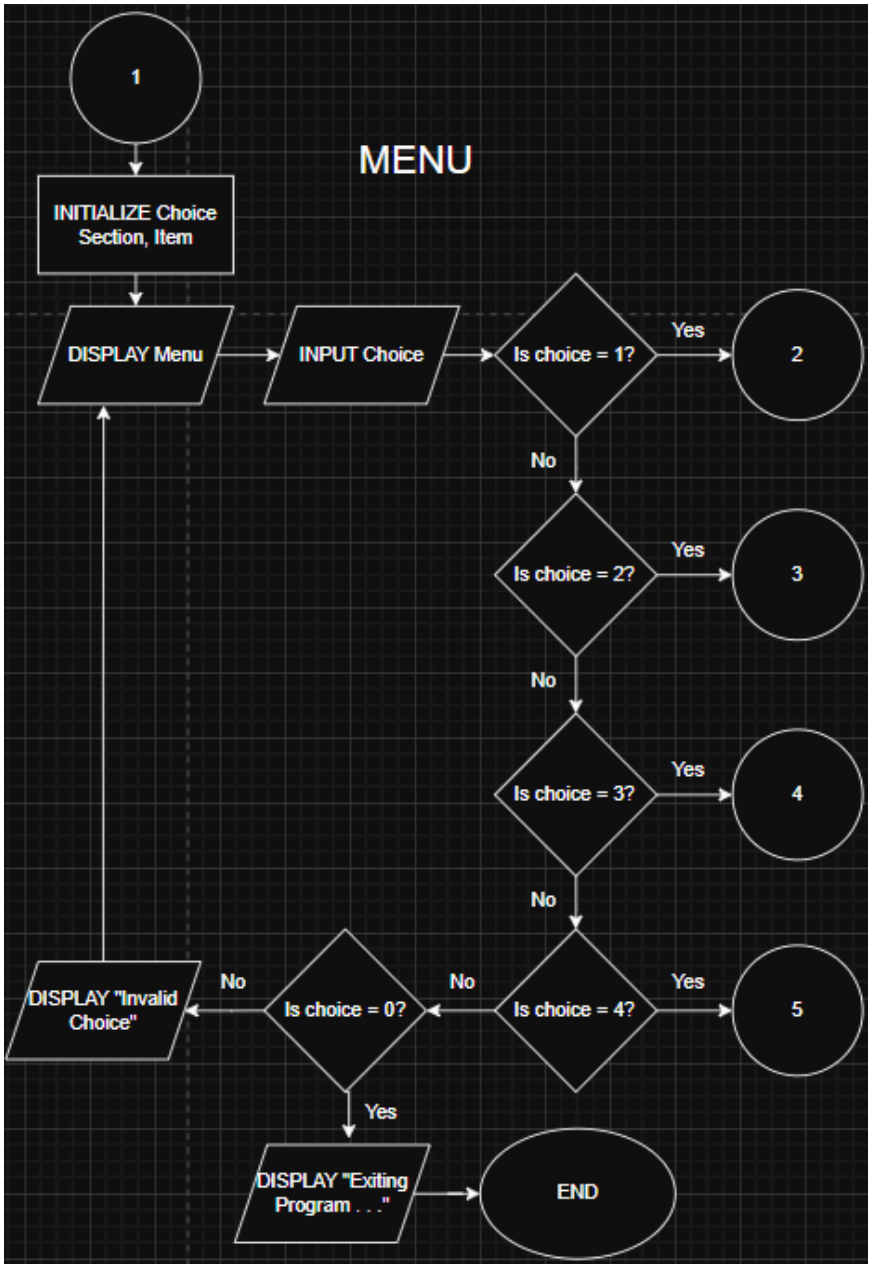

Figure 2. Main Menu Flowchart

Figure 2 illustrates the Main Menu Flowchart. In the main menu, the program first initializes the int choice, and string section and item variables. After initializing, The user is then prompted to input their decision into the choice variable to process and be evaluated through a series of conditional checks. If the choice is equal to 1, 2, 3, and 4, the corresponding processes (labeled 2 through) are executed. In expanded form, if the choice is equal to 1, it will call the function logItem(); if it's equal to 2, it will call the function claimItem(); if it's 3, it will call the function displayLog(); and lastly, if it's 4, it will call the function historyLog(); If the user inputs zero, an exit message is displayed and the program ends. If the inputted choices are not any of the valid choices, an invalid message will be displayed and loops around.

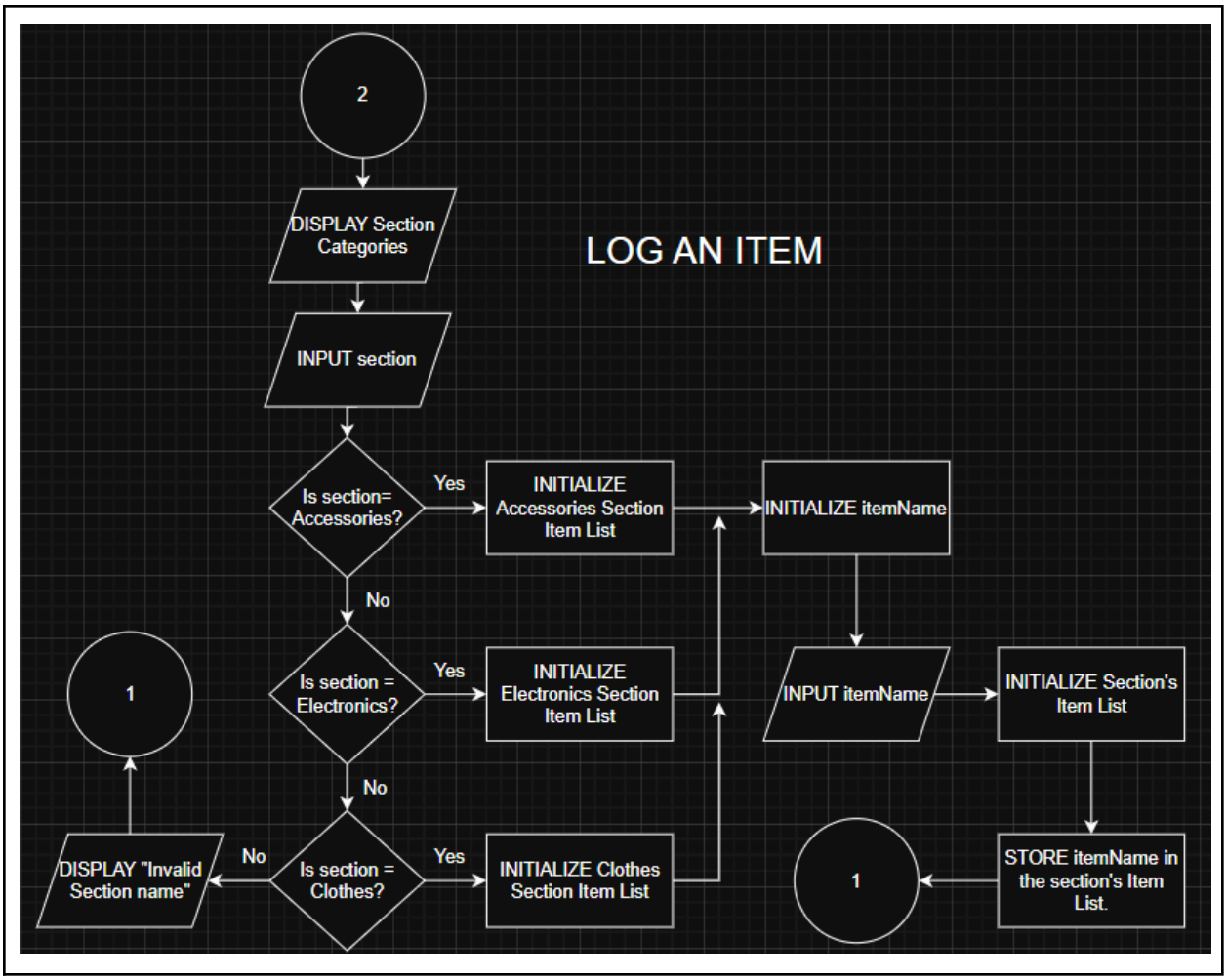

Figure 3. Log Item Flowchart

Figure 3 illustrates the Log an Item flowchart. Connector 2 represents the log an Item sub-program, it begins by displaying the section categories which are Accessories, Electronics, and Clothes. The user is then prompted to enter which section they would like to log an item in, if the inputted section is invalid, an invalid message will be displayed and the user is sent back to the main menu.

Otherwise, the corresponding section's item list will initialize along with the itemName variable before prompting the user to input the item name, short description about the item, Then the item will be stored in its chosen section. A confirmation message will be displayed along with the current date and time before going back to the main menu.

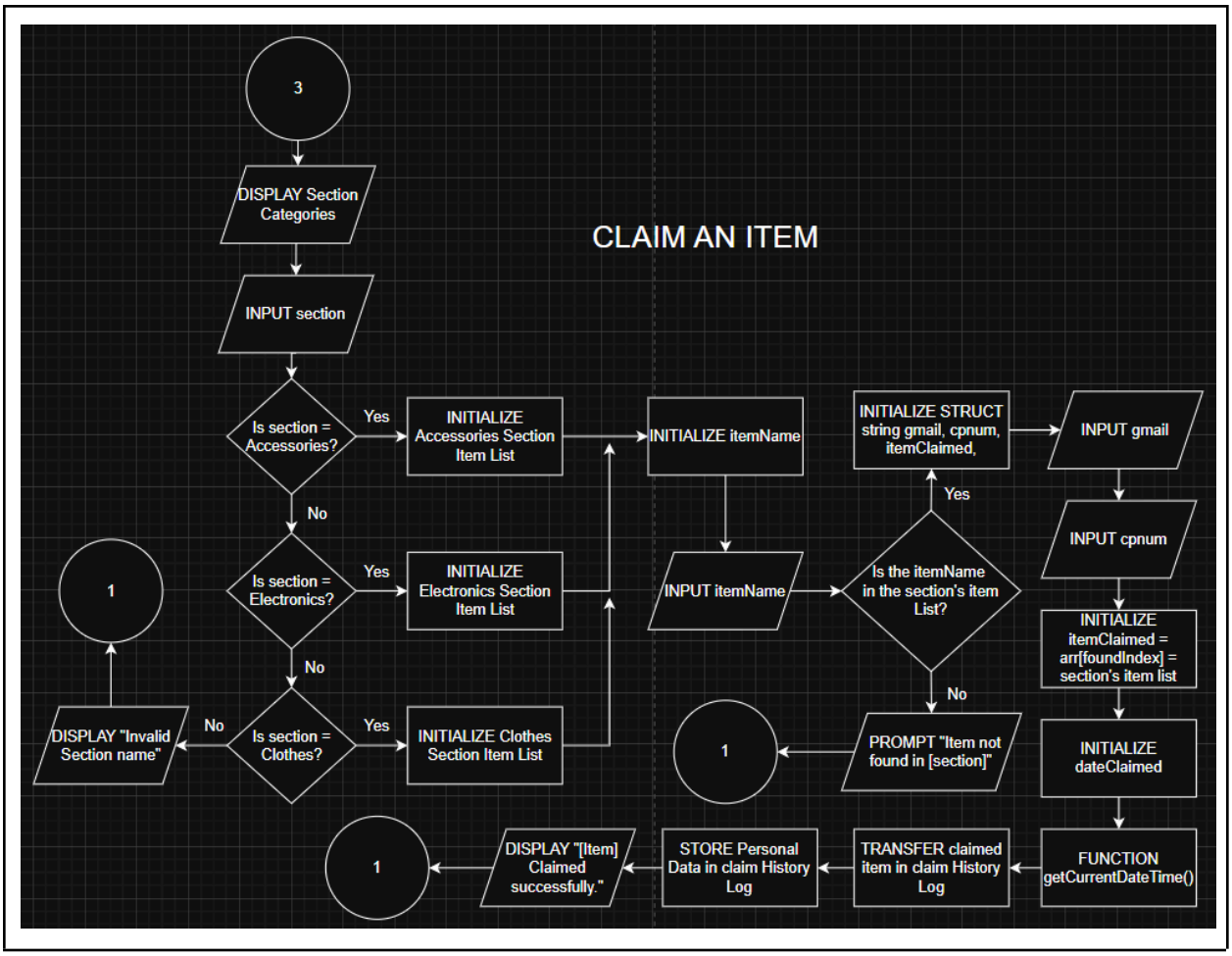

Figure 4. Item Claim Flowchart

Figure 4 illustrates the Item Claim flowchart. The Claim an Item sub-program is represented by connector 3, first it displays the section categories and prompts the user to input a section. If the inputted section is invalid, an invalid message will be displayed and the user is sent back to the main menu. Otherwise the corresponding section's item list will initialize along with the itemName variable and the user will be prompted to input the item name. If the item name is not on the selected section's item list then a "nor found" message will be displayed and the user is sent back to the main menu.

Otherwise, structures for gmail, phone number, and item claimed will initialize and the user will be prompted to enter their gmail address and phone number. After which other variables will be initialized along with the dateClaimed variable then the function getCurrentDateTime will activate before the claimed item will be transferred to the claim history log along with the claimant's personal information. Then a confirmation message will be displayed before the user is sent back to the main menu.

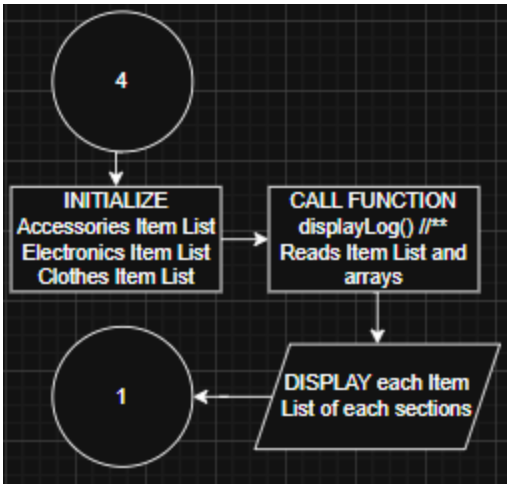

Figure 5. Display Item Log List Flowchart

Figure 5 illustrates the Display Item List Flowchart. The Display Item List sub-program is represented by connector 4, first it initiates the item lists for all categories before activating a function that reads the item lists, then it displays the items of each category before displaying the main menu.

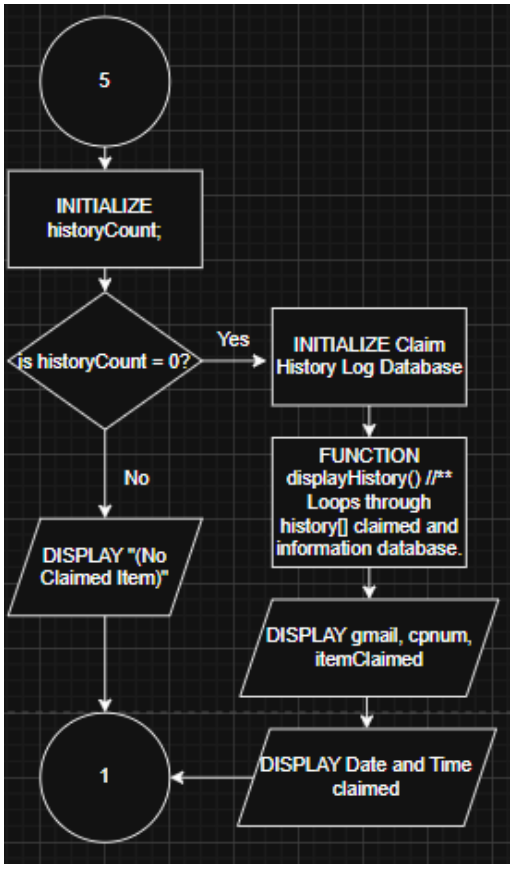

Figure 6. Display Claim History Flowchart

Figure 6 illustrates the Display Claim History Flowchart. The Display Claim History sub-program is represented by connector 5, first it initializes the historyCount variable then checks to see if the history count is zero, if it is not, then it will display a “No Claimed Item” message then display the main menu.

Otherwise, it will initialize the Claim History Log Database and activate a function that displays the information of claimants and the items they claimed along with the date and time they were claimed before displaying the main menu.

Pseudocode

The pseudocode presents organized modular functions and uses clear logic and appropriate control structures. It represents a Student Lost and Found Management System similar to the Engineers project. Pseudocode describes what the code does step by step and it focuses on explaining the flow, ideas, and logic on how the program will functionalize. It is a representation of a bridge between human and actual programming to help you understand the structure and purpose of the program you're making.

```
START
     FUNCTION studentLogin()
          DECLARE validIDs = {202501_CPE1, 202501_CP...}
          LOOP
               DISPLAY Student Login Menu
               PROMPT "Enter Student ID: " → inputID
               IF inputID == validIDs THEN
                    DISPLAY "Login successful! Welcome,
                    Student."
                    RETURN TRUE
               END IF
          END LOOP
     END FUNCTION

     FUNCTION logItem(section, itemName)
          IF section = "accessories" THEN
               IF accCount < MAX_ITEMS THEN
                    ADD itemName to accessories[]
                    INCREMENT accCount
               ELSE
                    DISPLAY "Accessories section full!"
               END IF

          ELSE IF section = "electronics" THEN
               IF elecCount< MAX_ITEMS THEN
                    ADD itemName to electronics[]
                    INCREMENT elecCount
               ELSE
                    DISPLAY "Electronics section full!"
               END IF

          ELSE IF section = "clothes" THEN
               IF clothCount< MAX_ITEMS THEN
                    ADD itemName to clothes[]
                    INCREMENT clothCount
               ELSE
```

```
                DISPLAY "clothes section full!"
            END IF

        DISPLAY "ITEM added successfully."
    END FUNCTION

    FUNCTION claimItem(section, itemName)
        IF section == "accessories" THEN
            CALL claimFromSection(accessories, accCount,
            itemName, "Accessories")
        ELSE IF section == "electronics" THEN
            CALL claimFromSection(electronics, elecCount,
            itemName, "Electronics")
        ELSE IF section == "clothes" THEN
            CALL claimFromSection(clothes, clothCount,
            itemName, "Clothes")
        ELSE
            DISPLAY "Invalid section name."
        END IF
    END FUNCTION

    FUNCTION displayLogs()
        DISPLAY "Accessories:"
            IF accCount == 0 THEN
                DISPLAY "(no items)"
            ELSE
                FOR i FROM 0 TO accCount - 1
                    DISPLAY accessories[i]
                END FOR
            END IF

        DISPLAY "Electronics:"
            IF elecCount== 0 THEN
                DISPLAY "(no items)"
            ELSE
                FOR i FROM 0 TO elecCount - 1
                    DISPLAY electronics[i]
                END FOR
            END IF

        DISPLAY "Clothes:"
            IF clothCount == 0 THEN
                DISPLAY "(no items)"
            ELSE
                FOR i FROM 0 TO clothCount - 1
                    DISPLAY clothes[i]
```

```
                END FOR
            END IF
END FUNCTION

FUNCTION displayHistory()
     IF historyCount == 0 THEN
          DISPLAY "(No claimed items)"
     ELSE
          FOR i FROM 0 TO historyCount - 1
             DISPLAY "Gmail: " << history[i].gmail
             DISPLAY "Mobile #: " << history[i].cpnum
             DISPLAY "Item Claimed: "
                      << history[i].itemClaimed
             DISPLAY "Date Claimed: "
                      << history[i].dateClaimed
          END FOR
     END IF
END FUNCTION

FUNCTION MAIN()
     CALL studentLogin()
     IF NOT studentLogin() THEN
     TERMINATE PROGRAM
     END IF

CREATE object logger
DECLARE choice, section, item;
DO
     DISPLAY Main Menu
     PROMPT "Choose an option" → choice
          SWITCH(choice)
               Case 1:
                    PROMPT "Enter Section: " → section
                    PROMPT "Enter Item Name: " → item
                    CALL logger.logItem(section, item)

               Case 2:
                    PROMPT "Enter Section: " → section
                    PROMPT "Enter Item to Claim: " → item
                    CALL logger.claimItem(section, item)

               Case 3:
                    CALL logger.displayLogs()

               Case 4:
                    CALL logger.displayHistory()
```

```
                    Case 0:
                         DISPLAY "Exiting Program..."

                    Default:
                         DISPLAY "Invalid choice!"
               END SWITCH
     WHILE choice is NOT 0
     RETURN 0;
END
```

Figure 7. Pseudo code of Lost & Found Inventory Tracker System (LoFITS)

Figure 7 presents the pseudocode of the LoFITS program system. The given pseudocode represents a Lost and Found Management System designed for organizations and school campuses. It allows students to log in, record found items, claim items, and view logs or claimed history. The program starts by calling the "studentLogin" function, which continuously prompts the user to enter a valid student ID from a predefined list in the system. Once a valid ID is inputted, access is granted and the main menu of the program LoFITS is displayed. It is a menu with options to log an item, claim an item, display current logs, and show the claim history. The program loops, unless the user chooses to exit the program.

The pseudocode demonstrates all the loops, conditional statements, do-while loop functions, in the program. This allows the user to continuously interact with the program. While, the IF and SWITCH statements handle the decision-making The pseudocode is organized into several functions that handle specific tasks. The "logItem()" functions record found items into their respective sections, the claimItem() functions enables students to claim an item from one of the sections by calling another helper function which was presented in the pseudocode "claimFromSection()." The "displayLogs()" functions lists all stored items by category, and lastly the "displayHistory()" which shows the details of claimed items such as the claimant's personal information and the date & time of the claimed item. These functions are managed using arrays and along with their own corresponding counters to keep track of the numbers of items in each section.

Overall, The pseudocode presents a well structured and functional outline of the program Lost and Found Inventory Tracker System (LoFITS). It effectively simulates item management and retrieval of items.

Data Dictionary

The Data Dictionary that is constructed in correlation with the engineers project, represents detailed information about all the variables, constants, arrays, and functions that are used inside the program. The Data Dictionary serves as a guide to understand the purpose and limitations of each element in the code. With the existence of the data dictionary, this ensures that in the procedure of debugging or extending the program, developers would be able to quickly identify how the program code functions in terms of how information is stored, processed, and manipulated.

As represented in the Data Dictionary, it presents the summary of the key components and variables in the program. As previously mentioned, it includes constants, arrays, multiple counters for each section of the program, and a user-defined function structure. As per table, each provides the element's name, range or size, data type, and a description of its functionality in the program system. For example, arrays such as the accessories, electronics, and clothes are introduced as a variable that stores items in their respective sections. On the other hand, in connection with the section, the history array tracks claimed items along with the claimant information. Every data that is included in the Data Dictionary, are all interconnected to one another. It furtherly describes the flow of data between storing, transferring, and retrieving of data with a user-interface compatible.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
1. gmail	Dynamic Size	String	Stores the Gmail address of the person who claimed the item
2. cpnum	Dynamic Size	String	Stores the cellphone / mobile number of the person claiming the item
3. itemClaimed	Dynamic Size	String	Stores the name of the item that a person claimed.
4. dateClaimed	Fixed (19 characters)	String	Stores the date and time when the item was claimed.
5. MAX_ITEMS	50	Const int	Maximum number of items that can be logged in each sections (accessories, electronics, clothes)

6. MAX_HISTORY	150	Const int	Maximum number of claim history that can be stored.
7. MAX_USERS	5	Const int	Maximum number of valid student users allowed to log in.
8. accessories[ ]	50	String	Array that stores names of accessories items.
9. electronics[ ]	50	String	Array that stores names of electronics items.
10. clothes[ ]	50	String	Array that stores names of clothes items.
11. accCount	1 to 50	Int	Count of items in the accessories section.
12. elecCount	1 to 50	Int	Count of items in the electronics section.
13. clothCount	1 to 50	Int	Count of items in the clothes section.
14. history[ ]	100	Data[ ]	Array of data structs to store the claim history
15. historyCount	0 - 100	Int	Count of claimed items in the history section.
16. section	Dynamic Size	String	Stores the section name input by the user, used in decisions whether which section to proceed.
17. choice	0 - 4	Int	Stores menu choice selected by user, used in decision whether which of the menu will proceed.

Code

This section represents the C++ code implementation of our program, "**Lost and Found Inventory Tracker System**", on how it is designed for teachers and students within the campus of the school perimeters. It presents how the system allows users to log found items, claim them with their personal details, and track all transactions of claimed items in a history record. With the visual representation of the code, engineers can furtherly analyze how the system flows our information and how it tracks with consistency and efficiency.

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <cctype>
#include <ctime>
#include <iomanip>
#include <sstream>
using namespace std;

#define MAX_ITEMS 50
#define MAX_HISTORY 150
#define MAX_USERS 5

struct Data {
    string gmail;
    string cpnum;
    string itemClaimed;
    string dateClaimed;
};

string toLowerStr(string s) {
    for (int i = 0; i < (int)s.length(); i++) {
        s[i] = tolower(s[i]);
    }
    return s;
}
```

Figure 8. DECLARATION, DATA STRUCTURES, AND HELPER FUNCTION

In this figure, it presents constant declarations that are used for the max capacity of the arrays of the items, history, and user database, a data structure for the claimant's personal information in claiming an item and for the claim history database, and a helper function, which is the string [toLowerStr()] to furtherly assess and lessen the malfunctions / errors of the program whenever the program are asking to type a string value specially a section. There are several new libraries used too in the program that are necessary for a more informative and responsive approach of the program, such as the library <ctime>, which displays and consists of date-time format for the claim history, <string> and <sstream>, input/output operations and string handling, <cstdlib> memory and console management and many more libraries capable of providing fundamental tools in creating the program responsive.

```
string getCurrentDateTime() {
    time_t now = time(0);
    tm *ltm = localtime(&now);

    char buffer[30]; // enough for "YYYY-MM-DD HH:MM:SS"
    snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d",
             1900 + ltm->tm_year,
             1 + ltm->tm_mon,
             ltm->tm_mday,
             ltm->tm_hour,
             ltm->tm_min,
             ltm->tm_sec);
    return string(buffer);
}
```

Figure 9. REAL-TIME DATE AND TIME TRACKER

This figure presents the date and time function and how the program tracks the date and time of the system for the claim history record for when the item was claimed by the claimant. This function needs the library that was introduced in figure 8, which was the <ctime> library. It consists of the format of the date and time in order for the function to be displayed in the executable file. The <ctime> library returns the timestamp as a human-readable string in the format "YYYY-MM-DD HH:MM:SS". This timestamp is later used in the claim history to record when an item was claimed. This function is represented as a string **getCurrentDateTime()** to display it inside the main block inside the claim history records.

```
bool studentLogin() {
    string validIDs[MAX_USERS] = {
        "202501_CPE1", "202502_CPE1", "202503_CPE1",
        "202504_CPE1", "202505_CPE1"
    };
    string inputID;

    while (true) {
        cout << "\t|=== STUDENT LOGIN ===|\n";
        cout << "\t| Enter Student ID: ";
        cin >> inputID;

        if (toLowerStr(inputID) == "exit") {
            cout << "\nExiting program...\n";
            return false;
        }

        for (int i = 0; i < MAX_USERS; i++) {
            if (inputID == validIDs[i]) {
                cout << "
-----------------------------------------\n";
                cout << "? Login successful! Welcome, Student " <<
inputID << ".\n";
                cout << "
-----------------------------------------\n";
                return true;
            }
        }
           system("CLS");
        cout << "? Invalid Student ID! Please try again.\n\n";
    }
}
```

Figure 10. LOGIN SYSTEM FUNCTION

In this figure, it presents one of the features that includes CRUD operations. This is the login system of the LoFITS Program. It is only made with constant database access where it is only accessible within an organization or school campuses. As you can see, the declared variables [ValidIDS] are the student IDS that are only capable of accessing the program. They are only the valid users that can log in to the system

and use the system. It's a bool function since the program only needs to verify if the validID is equal to the inputID that was inputted by the user. If any error may occur, the program has a hidden function that when engineers type exit in the studentID, It will terminate the program and restart the program.

```
class ItemLogger {
private:
    string accessories[MAX_ITEMS];
    string electronics[MAX_ITEMS];
    string clothes[MAX_ITEMS];
    int accCount, elecCount, clothCount;

    Data history[MAX_HISTORY];
    int historyCount;

public:
    ItemLogger() {
        accCount = elecCount = clothCount = 0;
        historyCount = 0;
    }
```

Figure 11. ITEM LOGGER CLASS SECTION DECLARATIONS

The Engineers program consists of a class named "ItemLogger". It consists of several public functions that handle various operations and declarations. It also consists of private functions that are only used inside the class. What are public and private sections? They are access specifiers that determine which parts of your program can access specific variables or functions of a class. As stated early, "**private:**" section of your class is hidden from the outside, which can only be accessed and used inside the class. They cannot be directly modified, and they are the ones who store the actual data and handle internal operations. As represented in the figure, the private section holds the array values of the sections and their counters of the program since they are the internal storage system of the program. The "**public:**" section are functions that are accessible from outside the class. These are the functions you can call directly. As for example, in terms of calling the **logItem()** to log an Item or **displayHistory()** to display the claim history records.

```
   void logItem(string section, string itemName) {
        section = toLowerStr(section);
        if (section == "accessories") {
            if (accCount < MAX_ITEMS) accessories[accCount++] =
itemName;
            else cout << "Accessories section full!\n";
        } else if (section == "electronics") {
            if (elecCount < MAX_ITEMS) electronics[elecCount++] =
itemName;
            else cout << "Electronics section full!\n";
        } else if (section == "clothes") {
            if (clothCount < MAX_ITEMS) clothes[clothCount++] =
itemName;
            else cout << "Clothes section full!\n";
        } else {
            cout << "Invalid section name.\n";
            return;
        }
        cout << "? \"" << itemName << "\" added under " << section <<
".\n";
    }
```

Figure 12. LOG AN ITEM FUNCTION

One of the key functions of the LoFITS program, Logging an Item into the system. In this figure, it is composed of else-if functions and conditions that depend on the string that are inside the section variable. This function is inside a switch-case function in the main block, where it has a decision to either choose between the 3 sections that are in the user-prompt function menu. There are 3 sections, [Accessories / Electronics / Clothes]. Each section has their own item database, in basic terms, an array list storage of logged items. Once you log an item, the declared counters in each section increase to verify if it has reached the maximum capacity of the storage. The items that were logged are stored inside the section's database. This function also has preventive measures in case that the section is not within the option, it returns to the main menu for retrial. If you have logged an Item successfully, it will display that the Item is/are added under which section you have chosen in the section category menu.

```
private:
    void claimFromSection(string arr[], int& count, const string&
itemName, const string& sectionName) {
        int foundIndex = -1;
        for (int i = 0; i < count; i++) {
            if (toLowerStr(arr[i]) == toLowerStr(itemName)) {
                foundIndex = i;
                break;
            }
        }
        if (foundIndex != -1) {
            if (historyCount < MAX_HISTORY) {
                // Ask claimant info
                Data logData;
                cout << "\n--- Claimant Information ---\n";
                cout << "Enter Gmail: ";
                cin >> logData.gmail;
                cout << "Enter Mobile #: ";
                cin >> logData.cpnum;
                logData.itemClaimed = arr[foundIndex];
                logData.dateClaimed = getCurrentDateTime();

                // Add to history log
                history[historyCount++] = logData;
                cout << "? \"" << arr[foundIndex] << "\" claimed
successfully on " << logData.dateClaimed << ".\n";
            }
            // Remove claimed item from section
            for (int j = foundIndex; j < count - 1; j++) {
                arr[j] = arr[j + 1];
            }
            count--;
        } else {
            cout << "? Item not found in " << sectionName << ".\n";
        }
    }
};
```

Figure 13. CLAIM AN ITEM FUNCTION

Another one of the key functions of the LoFITS, consisting of CRUD features. In this Figure, it represents the Item claim function, where you claim an item that was logged in the system, that is inside the sections database / array's item list. It has the same functionality as the log an item function, It is also inside the main block function of the switch-case of the user-prompt menu, the program takes which section will the user claim an item between the 3 sections. Once chosen, they shall write the EXACT name of the item. It supports and ensures case-insensitive string comparison, but it can't support incorrect item names. If the item name is incorrect, a single letter missing or space not included, or special characters, the system will not be able to identify it and will display that the item does not exist or cannot be found in the item list of the section database. If the name is correct, and it exists in the item list of the section's database, then the system will run another function, using an IF-ELSE function statement of IF the item exists, the system will declare logData. The system will prompt a personal information menu to fill up in order to successfully claim the item. Once the information is filled and submitted, the system will get the current date and time of when it was claimed, and the system will remove the claimed item from the section. The submitted information will be stored in the Struct logData, which will be used for the program function that records the history of items claimed.

```
   void displayLogs() const {
        cout << "\n--- Current Item Logs ---\n";
        cout << "Accessories:\n";
        if (accCount == 0) cout << "  (No items)\n";
        for (int i = 0; i < accCount; i++) cout << "  - " <<
accessories[i] << "\n";

        cout << "Electronics:\n";
        if (elecCount == 0) cout << "  (No items)\n";
        for (int i = 0; i < elecCount; i++) cout << "  - " <<
electronics[i] << "\n";

        cout << "Clothes:\n";
        if (clothCount == 0) cout << "  (No items)\n";
        for (int i = 0; i < clothCount; i++) cout << "  - " <<
clothes[i] << "\n";
    }
```

Figure 14. DISPLAY ITEM LIST FUNCTION

In this figure, it is the representation of a function of displaying the sections database / item list. It uses arrays in displaying all stored items inside the item list of each section. It separates the items in their own designated database in accordance with what the user chose to store the items to. It is one of the basic code functions compared to the rest of the key features that consist of multiple functions to operate. The

code is just repetitive since it is just about displaying the item logs that are stored inside the section's database.

```
   void displayHistory() const {
        cout << "\n--- Claimed Item History ---\n";
        if (historyCount == 0) {
            cout << "(No claimed items)\n";
            return;
        }
        for (int i = 0; i < historyCount; i++) {
            cout << "[" << i + 1 << "] Gmail: " << history[i].gmail
                 << "\nMobile #: " << history[i].cpnum
                 << "\nItem Claimed: " << history[i].itemClaimed
                 << "\nDate & Time Claimed: " <<
history[i].dateClaimed << "\n\n";
        }
    }
```

Figure 15. CLAIMED HISTORY FUNCTION

One of the most important functions and key features of the system, the claimed item history log function. All functions that were introduced earlier, are all just a combination of this function. In the claim of an item user-prompt, the system took the claimant's personal item and stored it inside the logData. The logData Structure is more often used in this function, since it will read all the information that is stored in the struct logData function and display it for conflict matters. The Claimed History Function is created in order for us to keep track of ingoing and outgoing items in the system to prevent issues between individuals claiming the wrong items and to display all the recorded claimed items history. Each entry displays the claimant's Gmail, contact number, the name of the item claimed, and the date and time of when the item was claimed.

```
int main() {
    if (!studentLogin()) {
        cout << "\nProgram terminated due to invalid login.\n";
        return 0;
    }
    ItemLogger logger;
    int choice;
    string section, item;

    do {
```

```
     cout << "\n -------------------------";
        cout << "\n ==== ITEM LOGGER MENU ====\n";
     cout << " -------------------------\n";
        cout << "[1.] |    Log New Item    |\n";
        cout << "[2.] |     Claim Item     |\n";
        cout << "[3.] | Show Current Logs  |\n";
        cout << "[4.] | Show Claim History |\n";
        cout << "[0.] |        Exit        |\n";
        cout << "--------------------------\n";
        cout << "Choose an option: ";
        cin >> choice;
        cout << "--------------------------\n";
        cin.ignore();

        system("CLS"); // clear screen (optional)

        switch (choice) {
        case 1:
            cout << "----------------------------------------\n";
            cout << " ===============| Section |===============\n"
                 << "----------------------------------------\n"
                      << " |( Accessories | Electronics | Clothes
)|\n";
                cout <<
"----------------------------------------\n Enter Section: ";
            getline(cin, section);
            cout << "----------------------------------------\n";
            cout << "Enter item name: ";
            getline(cin, item);
            cout << "----------------------------------------\n";
            logger.logItem(section, item);
            break;

        case 2:
            cout << "----------------------------------------\n";
            cout << " ===============| Section |===============\n"
                 << "----------------------------------------\n"
                      << " |( Accessories | Electronics | Clothes
)|\n";
```

```
                cout <<
"----------------------------------------\n Enter Section: ";
            getline(cin, section);
            cout << "Enter item name to claim: ";
            getline(cin, item);
            logger.claimItem(section, item);
            break;

        case 3:
            logger.displayLogs();
            break;

        case 4:
            logger.displayHistory();
            break;

        case 0:
            cout << "Exiting program...\n";
            break;

        default:
            cout << "Invalid choice!\n";
        }

    } while (choice != 0);
    return 0;
}
```

Figure 16. LoFITS MAIN MENU FUNCTION

This figure is the main menu of the LoFITS program, it is where the user decisions are mostly taken in order to transition between functions of the program. In the main block, it is where all the functions are declared and initialized. This serves as the control center of the entire program. It starts by calling the studentLogin() function, which is the login prompt to ensure that only authorized users can access the program. The menu provides options to log a new item, claim an existing item in the database, view the current item logs, and view the claim history. Depending on the user's input, the corresponding function within the itemLogger class that was mentioned above, is executed. The menu loop continues to run until the user chooses to exit the program.

Results and Discussion

This section presents the output and the functionality of the developed and implemented Item Logger System named Lost and Found Inventory Tracker System (LoFITS). The goal of LoFITS was to provide a simple yet efficient way to record, claim, and monitor items using a C++ console-based program with basic features of CRUD and structured data handling.

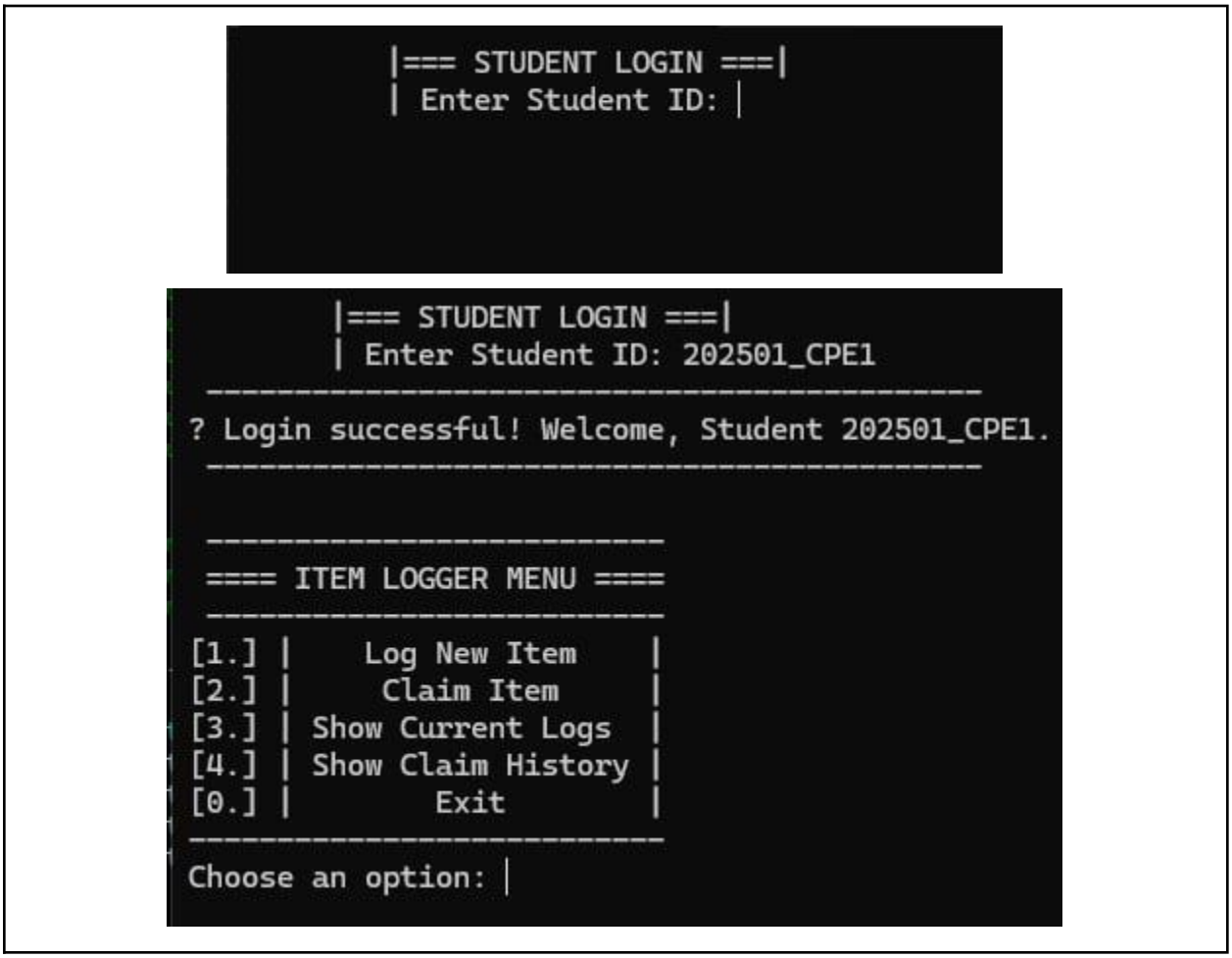

Figure 17. Login Prompt & Main Menu

Upon running the program, the user is greeted with a student login interface requiring a valid student ID in order to use the program and to be granted access to use the program. Only authorized users can proceed to access the main menu. Logins trigger a retry prompt until a correct ID is provided. Once logged in, the main menu displays several options for the user.

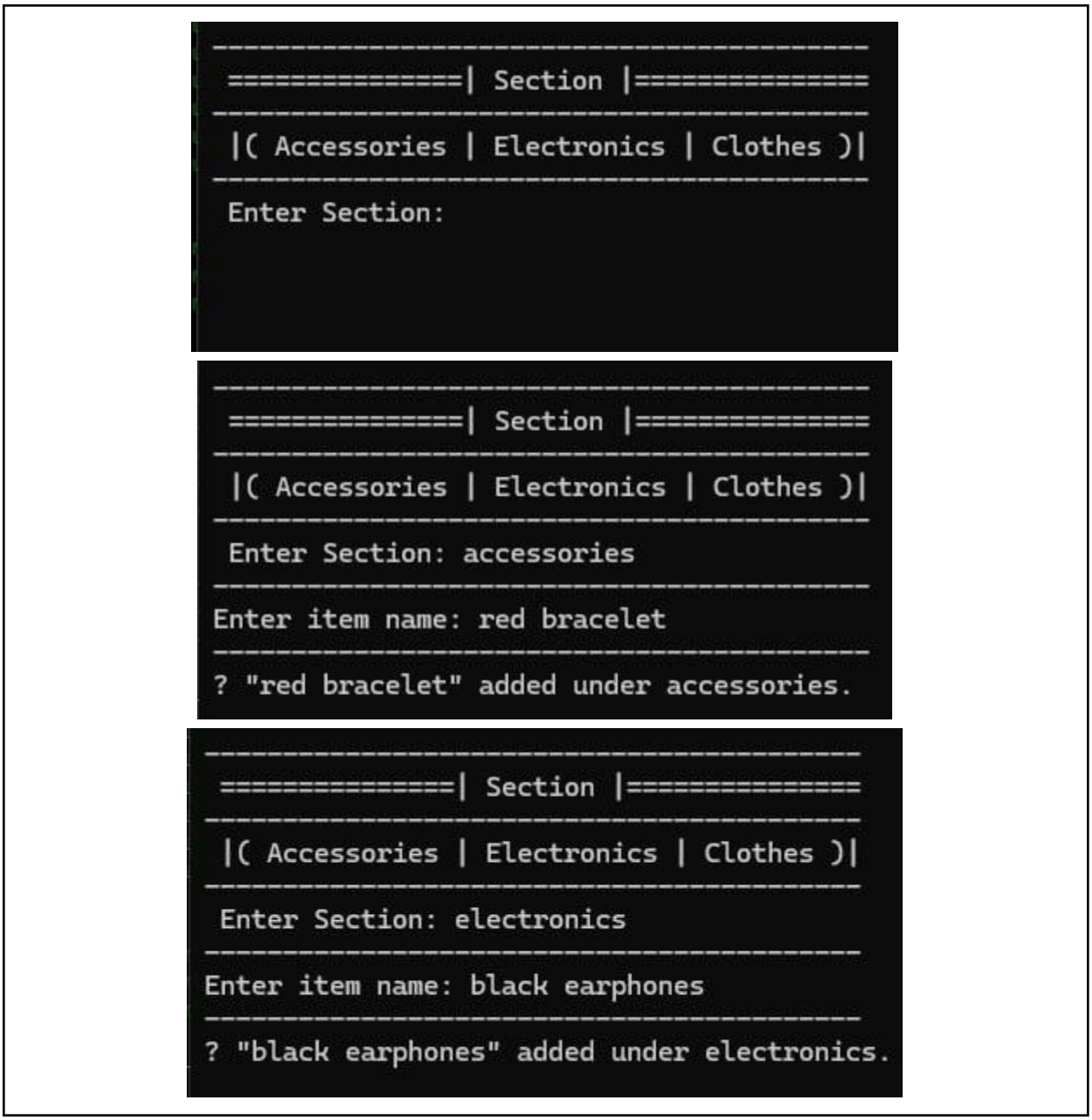

Figure 18. Logging an Item menu function

In the logging an Item menu, the program successfully adds the new item under the chosen section by the user. Logged items are stored temporarily in memory and displayed through “Show Current Logs” Option in the main menu.

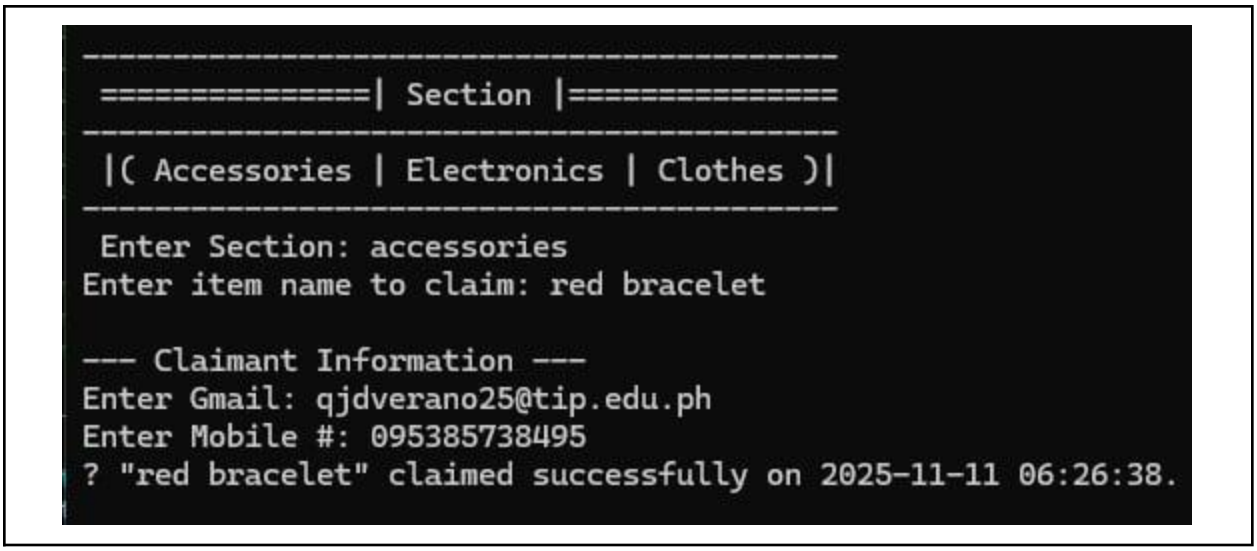

Figure 19. Claiming an Item menu function

In this figure, the system records the claimant's information upon claiming an item. The system keeps record of the item name, timestamp of the transaction, and the personal information that was filled by the claimant. It also automatically removes the claimed item from the current log list.

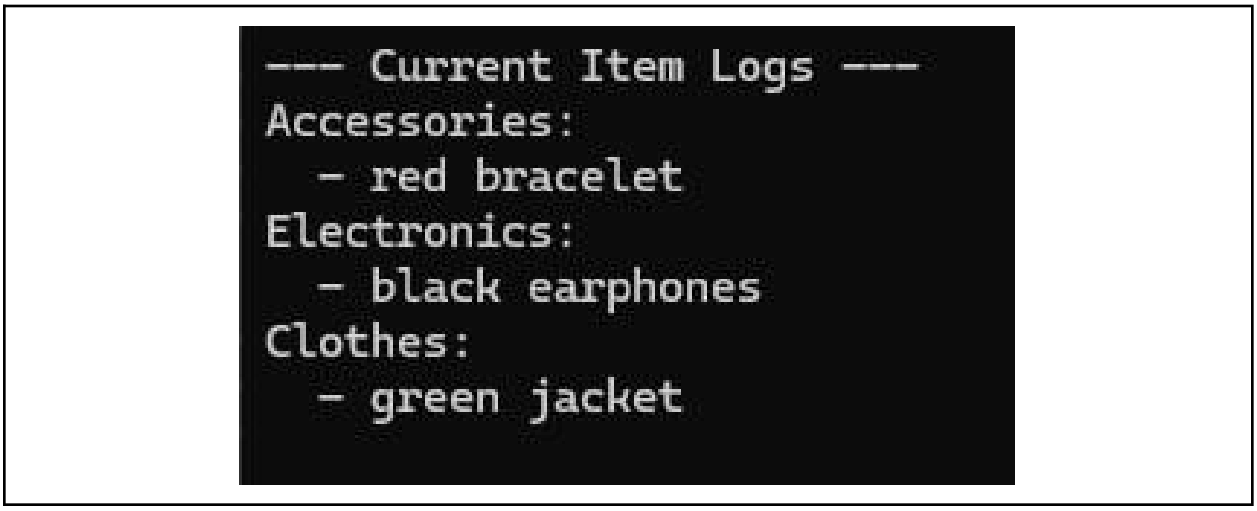

Figure 20. Current Item Logs Menu

This is the program that displays the new items that were logged by the user. It is temporarily stored here and is tracked in terms of what are the items in each section's item list. The output verifies that the system correctly updates and maintains separate lists for unclaimed and claimed items.

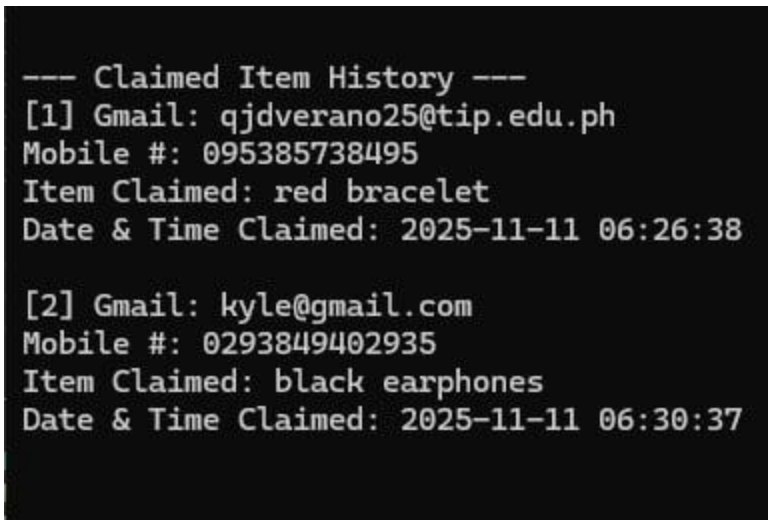

Figure 21. Claimed Item History Menu

This is the program's history log. This is where all claimed items are stored, the claimant's information, and when was the item claimed. This provides an organized and a preventive measure in terms of conflict, being able to contact the person who claimed an item that could possibly not be theirs.

The program demonstrates and represents functional data handling using structured arrays and functions. As the output presents a clear and easy to understand menu, users can easily log and claim an item. The use of time-stamping and personal information logging enhances the traceability and preventive measure for the claimed item. This tracks the claimant and is contacted to evaluate a correct analysis if the items are really theirs. This program also has student login validation to ensure that the user is within the organization to be able to use the program. The implementation of the LoFITS effectively showcases the basic concepts of programming such as structures, arrays, loops, conditionals, and user input management.

Conclusion

Overall, the program effectively implements the desired objectives in item management and claim-tracking systems. The LoFITS Successfully achieves all its specific objectives of the program. LoFITS was able to address the common challenges such as misplacement, duplication or false claiming of items. With LoFITS, engineers ensure that all problems regarding the challenges of students being unable to retrieve their items back, provides the precise and accurate solution. Especially with LoFITS system of maintaining a detailed log of all items in the repository, and to also provide the complete history of all ingoing and outgoing items in the Lost and Found Inventory, with provided personal informations for items that were claimed that can be used for tracking in case of conflict.

The login feature provides basic access control, the timestamped history adds accountability to the claim process. Although the program uses static arrays that limit the amount of data that can be stored, it is easier to manage and to follow. Recommendations such as enhancements of replacing arrays with advanced C++ language, adding input validation for emails and phone numbers for a more secure approach, or implementing a file storage for data persistence that could furtherly improve the functionality and real-world applicability of the system. With the recommendation, future engineers can potentially enhance this program and improvise this system to a more reliable and organized program for organizations.

References

Castro, E., et. al. (2022, March 10). AUFound: Retrieval of Misplaced Personal Belongings Through Mobile Application and Web-Based Management System Designed for Angeles University Foundation. https://ieomsociety.org/proceedings/2022istanbul/383.pdf

Markham, A. (2024, May 13). Navigating Security Challenges: Minimizing Loss and Liability with Pixit's Secure Lost and Found Platform. https://www.pixithq.com/blog/navigating-security-challenges-minimizing-loss-and-liability-with-pixits-secure-lost-and-found-platform

Markham, A. (2024, August 26). The Cost Of Chaos: How A Disorganized Lost And Found Hurts Your Customer Satisfaction https://www.pixithq.com/blog/the-cost-of-chaos-how-a-disorganized-lost-and-found-hurts-your-customer-satisfaction

Vocabulary.com. (n.d.). Lost-and-found. In Vocabulary.com Dictionary. https://www.vocabulary.com/dictionary/lost-and-found

Appendices

Appendix A: LoFITS.cpp

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <cctype>
#include <ctime>
#include <iomanip>
#include <sstream>
using namespace std;

#define MAX_ITEMS 50
#define MAX_HISTORY 150
#define MAX_USERS 5

//------------------------------------ DATA STRUCTURE FOR CLAIM
HISTORY -------------------------------------//

struct Data {
    string gmail;
    string cpnum;
    string itemClaimed;
    string dateClaimed;
};

//----------------------------------- HELPER FUNCTION TO LOWERCASE A
STRING ------------------------------------//
string toLowerStr(string s) {
    for (int i = 0; i < (int)s.length(); i++) {
        s[i] = tolower(s[i]);
    }
    return s;
}

string getCurrentDateTime() {
    time_t now = time(0);
    tm *ltm = localtime(&now);
```

```
    char buffer[30]; // enough for "YYYY-MM-DD HH:MM:SS"
    snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d",
             1900 + ltm->tm_year,
             1 + ltm->tm_mon,
             ltm->tm_mday,
             ltm->tm_hour,
             ltm->tm_min,
             ltm->tm_sec);
    return string(buffer);
}

//------------------------------------ ITEM LOGGER CLASS
------------------------------------//

class ItemLogger {
private:
    string accessories[MAX_ITEMS];
    string electronics[MAX_ITEMS];
    string clothes[MAX_ITEMS];
    int accCount, elecCount, clothCount;

    Data history[MAX_HISTORY];
    int historyCount;

public:
    ItemLogger() {
        accCount = elecCount = clothCount = 0;
        historyCount = 0;
    }

    void logItem(string section, string itemName) {
        section = toLowerStr(section);
        if (section == "accessories") {
            if (accCount < MAX_ITEMS) accessories[accCount++] =
itemName;
            else cout << "Accessories section full!\n";
        } else if (section == "electronics") {
            if (elecCount < MAX_ITEMS) electronics[elecCount++] =
itemName;
```

```
            else cout << "Electronics section full!\n";
        } else if (section == "clothes") {
            if (clothCount < MAX_ITEMS) clothes[clothCount++] =
itemName;
            else cout << "Clothes section full!\n";
        } else {
            cout << "Invalid section name.\n";
            return;
        }
        cout << "? \"" << itemName << "\" added under " << section <<
".\n";
    }

    void claimItem(string section, string itemName) {
        section = toLowerStr(section);
        if (section == "accessories") {
            claimFromSection(accessories, accCount, itemName,
"Accessories");
        } else if (section == "electronics") {
            claimFromSection(electronics, elecCount, itemName,
"Electronics");
        } else if (section == "clothes") {
            claimFromSection(clothes, clothCount, itemName,
"Clothes");
        } else {
            cout << "Invalid section name.\n";
        }
    }

    void displayLogs() const {
        cout << "\n--- Current Item Logs ---\n";
        cout << "Accessories:\n";
        if (accCount == 0) cout << "  (No items)\n";
        for (int i = 0; i < accCount; i++) cout << "  - " <<
accessories[i] << "\n";

        cout << "Electronics:\n";
        if (elecCount == 0) cout << "  (No items)\n";
        for (int i = 0; i < elecCount; i++) cout << "  - " <<
```

```
electronics[i] << "\n";

        cout << "Clothes:\n";
        if (clothCount == 0) cout << "  (No items)\n";
        for (int i = 0; i < clothCount; i++) cout << "  - " <<
clothes[i] << "\n";
    }

    void displayHistory() const {
        cout << "\n--- Claimed Item History ---\n";
        if (historyCount == 0) {
            cout << "(No claimed items)\n";
            return;
        }
        for (int i = 0; i < historyCount; i++) {
            cout << "[" << i + 1 << "] Gmail: " << history[i].gmail
                 << "\nMobile #: " << history[i].cpnum
                 << "\nItem Claimed: " << history[i].itemClaimed
                 << "\nDate & Time Claimed: " <<
history[i].dateClaimed << "\n\n";
        }
    }

private:
    void claimFromSection(string arr[], int& count, const string&
itemName, const string& sectionName) {
        int foundIndex = -1;
        for (int i = 0; i < count; i++) {
            if (toLowerStr(arr[i]) == toLowerStr(itemName)) {
                foundIndex = i;
                break;
            }
        }

        if (foundIndex != -1) {
            if (historyCount < MAX_HISTORY) {
                Data logData;
                cout << "\n--- Claimant Information ---\n";
                cout << "Enter Gmail: ";
```

```
                cin >> logData.gmail;
                cout << "Enter Mobile #: ";
                cin >> logData.cpnum;
                logData.itemClaimed = arr[foundIndex];
                logData.dateClaimed = getCurrentDateTime();

                history[historyCount++] = logData;
                cout << "? \"" << arr[foundIndex] << "\" claimed
successfully on " << logData.dateClaimed << ".\n";
            }

            // Remove claimed item from section
            for (int j = foundIndex; j < count - 1; j++) {
                arr[j] = arr[j + 1];
            }
            count--;
        } else {
            cout << "? Item not found in " << sectionName << ".\n";
        }
    }
};

//---------------------------------- STUDENT LOGIN FUNCTION
----------------------------------//

bool studentLogin() {
    string validIDs[MAX_USERS] = {
        "202501_CPE1", "202502_CPE1", "202503_CPE1",
        "202504_CPE1", "202505_CPE1"
    };
    string inputID;

    while (true) {
        cout << "\t|=== STUDENT LOGIN ===|\n";
        cout << "\t| Enter Student ID: ";
        cin >> inputID;

        if (toLowerStr(inputID) == "exit") {
            cout << "\nExiting program...\n";
```

```
            return false;
        }

        for (int i = 0; i < MAX_USERS; i++) {
            if (inputID == validIDs[i]) {
                cout << "
-------------------------------------------\n";
                cout << "? Login successful! Welcome, Student " <<
inputID << ".\n";
                cout << "
-------------------------------------------\n";
                return true;
            }
        }
           system("CLS");
        cout << "? Invalid Student ID! Please try again.\n\n";
    }
}

//----------------------------------- MAIN MENU
-----------------------------------//

int main() {
    if (!studentLogin()) {
        cout << "\nProgram terminated due to invalid login.\n";
        return 0;
    }

    ItemLogger logger;
    int choice;
    string section, item;

    do {
      cout << "\n -------------------------";
        cout << "\n ==== ITEM LOGGER MENU ====\n";
      cout << " -------------------------\n";
        cout << "[1.] |    Log New Item    |\n";
        cout << "[2.] |     Claim Item     |\n";
        cout << "[3.] | Show Current Logs  |\n";
```

```
        cout << "[4.] | Show Claim History |\n";
        cout << "[0.] |        Exit        |\n";
        cout << "--------------------------\n";
        cout << "Choose an option: ";
        cin >> choice;
        cout << "--------------------------\n";
        cin.ignore();

        system("CLS"); // clear screen (optional)

        switch (choice) {
        case 1:
            cout << "----------------------------------------\n";
            cout << " ===============| Section |===============\n"
                 << "----------------------------------------\n"
                      << " |( Accessories | Electronics | Clothes
)|\n";
                cout <<
"----------------------------------------\n Enter Section: ";
            getline(cin, section);
            cout << "----------------------------------------\n";
            cout << "Enter item name: ";
            getline(cin, item);
            cout << "----------------------------------------\n";
            logger.logItem(section, item);
            break;

        case 2:
            cout << "----------------------------------------\n";
            cout << " ===============| Section |===============\n"
                 << "----------------------------------------\n"
                      << " |( Accessories | Electronics | Clothes
)|\n";
                cout <<
"----------------------------------------\n Enter Section: ";
            getline(cin, section);
            cout << "Enter item name to claim: ";
            getline(cin, item);
            logger.claimItem(section, item);
```

```
            break;

        case 3:
            logger.displayLogs();
            break;

        case 4:
            logger.displayHistory();
            break;

        case 0:
            cout << "Exiting program...\n";
            break;

        default:
            cout << "Invalid choice!\n";
        }

    } while (choice != 0);

    return 0;
}
```