Attendance Report System

Cenar, Marqui Joshua
Formalejo, Francis Elijah J.
Ramirez, Angel Mae C.
Silao, Johrel Louie T.
Tobias, Lawrence C.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

In educational institutions today, attendance is important for assessing student involvement and behavior. Numerous educational organizations continue to depend on manual methods for taking attendance. The majority of Philippine schools use laborious and time-consuming attendance tracking. Some employ paper-based methods, requiring students to turn them in along with their section and name. (Santos et al.,2021) This conventional approach can consume a significant amount of time during class sessions and may easily result in errors, including overlooked names, incorrect attendance, or misplaced records.(Lenuf, 2025). In a generation where technology continues to evolve rapidly, it seems outdated and inefficient for schools to still depend on manual systems when there are faster and more accurate alternatives available. This has become a general problem in many academic institutions, especially in large classes where monitoring attendance manually becomes more difficult and disorganized.

The specific problem arises when attendance data is not properly recorded or updated. Teachers may face challenges in tracking who is present, late, or absent, especially if there are multiple classes in a day. According to (Nwabuwe, et al.) “The most common act of misconduct was to sign a proxy for an absent friend.” This directly states the problem of false attendance being common in student settings. Over time, these small inaccuracies can affect grading and performance evaluation. Furthermore, the process of computing attendance percentages manually can take too much time leaving teachers less time to focus on their lessons and student engagement. In this sense, the problem is not just about recording attendance, but also about maintaining accuracy.

To address these problems, our project aims to create a code-based attendance checking system that automates the entire process. The program will be designed to record, track, and report student attendance efficiently. Using the C++ programming language, attendance can be monitored digitally reducing errors and saving time for both teachers and students. Using online attendance management automates the labour-intensive process of manual attendance tracking, this significantly reduces the time and effort required by educators and administrators. (Rahaman, et al. 2025). By implementing the Attendance Report System, educators can take one step forward towards a more effective classroom management.

The Project

The project aims to improve the monitoring and management of student attendance by providing a digital system to record, track, and view attendance reports. The program allows users to register by providing a username and password. Once registered, users can log in using their credentials, which are securely stored in the system's internal data structure, and are then presented with a menu of options.

Register and Login: Users can create an account by registering with a username and password. Once registered, they can log in using their credentials, which are stored securely in the system. This allows the program to identify each user and maintain individual attendance records.

Mark Attendance: After logging in, the user can input the student's name, date, and time in. The system automatically calculates the attendance status as Present, Late, or Absent based on the time entered, and the entry is saved under the logged-in user's records.

View Full Attendance Report: Users can view all attendance entries in a formatted table that includes the student's name, date, time in, and attendance status. This gives a clear overview of all recorded attendance data.

View Attendance by Date: Users can filter attendance records for a specific date, allowing them to quickly check attendance for a particular day.

Logout: Users can securely log out from the system, ending their session and protecting access to their attendance records.

Objectives

The project aims to develop an Attendance Report System that automates the process of recording and managing student attendance. Specifically:

- Enable users to register and log in securely using a unique username and password.
- Allow users to mark attendance by inputting the student's name, date, and time in.
- Automatically determine and record the attendance status as Present, Late, or Absent based on the time entered.
- Display attendance reports in a clear and organized format, including options to view all records or filter by a specific date.

Flowchart of the System

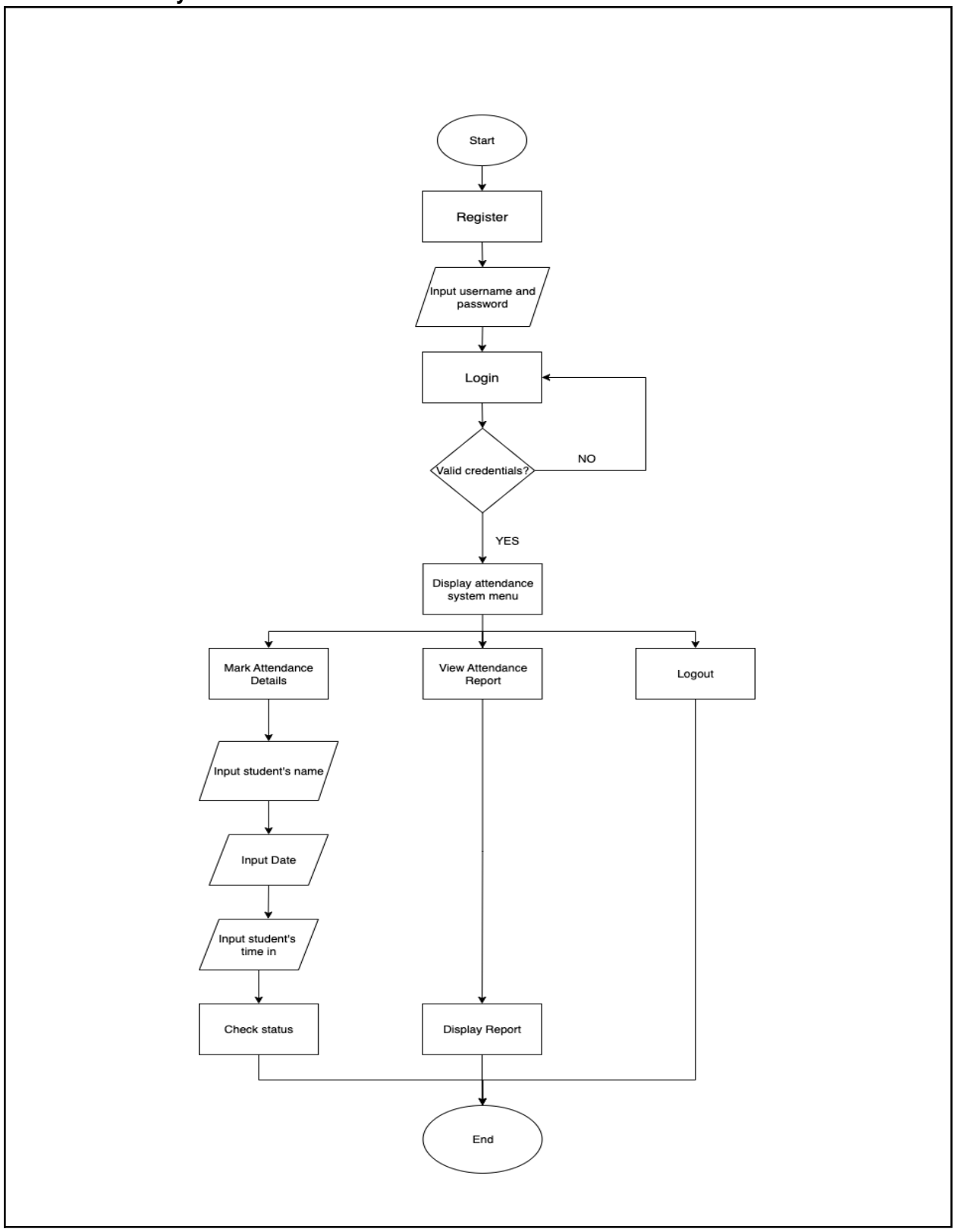


Figure 1: Flowchart of the Attendance Report System

Flowchart Description

The flowchart illustrates the step-by-step process of the Attendance System Program. It begins with the registration phase, where the user is required to create an account by entering a username and password. After successfully registering, the user proceeds to the login stage, where the system verifies the entered credentials. If the credentials are invalid, the user is prompted to try again; however, if they are correct, the system grants access to the main attendance system menu. Once inside the menu, the user is presented with three main options: Mark Attendance Details, View Attendance Report, and Logout. When selecting Mark Attendance Details, the system allows the user to input the student's name, date, and time-in, which are then used to determine and record the student's attendance status. If the user chooses View Attendance Report, the system retrieves and displays the recorded attendance data for review and monitoring. Meanwhile, selecting Logout securely ends the user session, ensuring that no unauthorized access occurs. The program then proceeds to the end of the process. Overall, the flowchart demonstrates a simple yet organized structure that ensures accurate attendance tracking, user authentication, and secure system management.

Pseudocode

This program uses a user login system to monitor and record the student's attendance. Users can log out, manage attendance reports, and mark their attendance. Based on the time entered, the system checks the attendance status (Present, Late or Absent).

main.cpp
START

```
users = empty list
currentUser = NULL

LOOP forever

   IF currentUser is NULL THEN
      DISPLAY "1. Login  2. Register  3. Exit"
      READ choice

      IF choice == 1 THEN
         IF users is empty THEN
            PRINT "Register first"
         ELSE
            currentUser = login(users)
            IF currentUser is NULL THEN
               PRINT "Invalid login"
            ENDIF
         ENDIF

      ELSE IF choice == 2 THEN
         registerUser(users)

      ELSE IF choice == 3 THEN
         PRINT "Goodbye"
         BREAK

      ELSE
         PRINT "Invalid choice"
      ENDIF

   ELSE
      displayMenu()
      READ choice

      SWITCH choice
         CASE 1: markAttendance(currentUser)
         CASE 2: viewReport(currentUser)
         CASE 3: viewReportByDate(currentUser)
         CASE 4: currentUser = NULL   // logout
         DEFAULT: PRINT "Invalid choice"
      END SWITCH

   ENDIF

END LOOP
```

END

Attendance_system.h

```
STRUCT Attendance:
   student_name
   date
   time_in
   status

STRUCT User:
   username
   password
   attendances (list of Attendance)

FUNCTION getCurrentDate()
FUNCTION trim(text)
FUNCTION lowerCopy(text)
FUNCTION parseTimeToMinutes(time_in)
FUNCTION calculateStatus(time_in)

FUNCTION displayMenu()
FUNCTION login(users)
FUNCTION registerUser(users)
FUNCTION markAttendance(user)
FUNCTION viewReport(user)
FUNCTION viewReportByDate(user)
```

Attendance_system.cpp

```
STRUCT Attendance:
   student_name, date, time_in, status

STRUCT User:
   username, password, attendances

FUNCTION getCurrentDate()
   RETURN system date in YYYY-MM-DD

FUNCTION trim(text)
   RETURN text without leading/trailing spaces
```

```
FUNCTION lowerCopy(text)
   RETURN text in lowercase

FUNCTION parseTimeToMinutes(time_in)
   CLEAN time_in
   CONVERT to hours and minutes
   HANDLE am/pm if present
   IF invalid -> RETURN -1
   RETURN total_minutes

FUNCTION calculateStatus(time_in)
   minutes = parseTimeToMinutes(time_in)
   IF minutes == -1 -> RETURN "Absent"
   IF minutes <= 7:30 AM -> "Present"
   ELSE IF minutes <= 4:00 PM -> "Late"
   ELSE -> "Absent"

FUNCTION displayMenu()
   PRINT "1. Mark Attendance 2. View Full Report 3. View by Date 4. Logout"

FUNCTION login(users)
   INPUT username, password
   FOR each user:
      IF username AND password match -> RETURN user
   RETURN NULL

FUNCTION registerUser(users)
   INPUT username
   IF username already exists -> PRINT "Taken" and RETURN
   INPUT password
   CREATE new User and add to users

FUNCTION markAttendance(user)
   INPUT student_name, date (default today), time_in
   status = calculateStatus(time_in)
   ADD Attendance to user.attendances
   PRINT summary

FUNCTION viewReport(user)
   IF no attendances -> PRINT "No records"
   ELSE PRINT all attendance records (name, date, time, status)

FUNCTION viewReportByDate(user)
   INPUT date (default today)
   FILTER attendances by date
```

```
IF none -> PRINT "No records"
ELSE PRINT filtered records (name, time, status)
```

Figure 2: Pseudocode of attendance report system

Pseudocode Description

main.cpp

This pseudocode shows how the system programs the login, registration, and the system menu. It starts with no users nor logged in users, then runs a loop. If no users are logged in, it shows a menu that shows "Login, Register, or Exit". The login only works if there is a registered user, if not then the system tells the user to register first. Registration is adding a new user, and Exit stops the program.

When a user is logged in into the system, it shows a menu where the user can mark attendance, view records, view records by date, or log out. Logging out of the system sets the "currentUser" back to NULL and returns to the login menu. This loop continues until the user chooses to exit the system.

Attendance_system.h

This pseudocode lists down all of the main parts of the attendance system. It defines two structures which are "Attendance" and "User". The attendance structure stores the student's details and status, User structure stores the login information and attendance list. It also shows all the functions used in the program, such as getting the current date, converting time, checking status, logging in, registering, marking attendance, and viewing reports.

Attendance_system.cpp

This pseudocode further explains the system with more details. It shows how each function works, such as getting the current date, converting time to minutes, and deciding whether a student is Present, Late, or Absent. It also shows how login and registration are checked, how attendance is added to a student, and how reports are displayed by date.

Data Dictionary

The data dictionary defines each variable's name, size, data type, and purpose to ensure clarity and consistency in the overall system design. It serves as a guide for understanding how data is stored, processed, and managed within the Attendance Report System. By organizing all variables systematically, it helps developers and users interpret the function and behavior of each element used in the program.

The table below lists all the data names, their corresponding types, sizes, and descriptions that are utilized in the code implementation. Each entry provides insight into how the variables, structures, and data types interact to form the core functionality of the Attendance Report System.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
student_name	32 bytes	string	Stores the full name of the student whose attendance
date	16 bytes	string	Represents the date of attendance in the format YYYY-MM-DD.
time_in	15 bytes	string	Holds the time the student arrived
status	15 bytes	string	Indicates attendance status: “Present”, “Late”, or “Absent.”
attendances	depends on number of records	vector<Attendance>	A list (vector) that holds multiple attendance records for each user.
users	depends on number of users	vector<User>	A list containing all registered users in the system.
choice	4 bytes	int	Stores the menu selection input from the user.
currentUser	8 bytes	User*	A pointer that refers to the currently logged-in user.
buffer	11 bytes	char[11]	Stores formatted date (e.g., "2025-11-11").
username	30 bytes	string	Stores the username used during registration and login.

Code

This section presents the header files of the attendance system program. The program is designed using a modular approach, dividing the code into multiple .cpp files along with their corresponding header files. This organization improves the clarity and maintainability of the project by separating different functionalities into

distinct components. By structuring the program in this way, the code becomes more readable, easier to debug, and simpler to test or expand in the future.

```
#ifndef ATTENDANCE_SYSTEM_H
#define ATTENDANCE_SYSTEM_H

#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include <ctime>
#include <limits>
#include <sstream>
#include <cctype>
#include <cstdlib>

using namespace std;

struct Attendance {
    string student_name;
    string date;       // YYYY-MM-DD
    string time_in;
    string status;     // Present, Late, Absent
};

struct User {
    string username;
    string password;
    vector<Attendance> attendances;
};

// Utility functions
string getCurrentDate();
string trim(const string &s);
string lowerCopy(const string &s);
int parseTimeToMinutes(const string &raw);
string calculateStatus(const string &time_in);

// Menu & user functions
void displayMenu();
User* login(vector<User> &users);
void registerUser(vector<User> &users);
```

```
void markAttendance(User &user);
void viewReport(const User &user);
void viewReportByDate(const User &user);

#endif
```

Figure 3 attendance_system.h

The attendance_system.h file serves as the header file for the Attendance System project and defines the essential data structures and function prototypes used throughout the program. It begins with inclusion guards using #ifndef, #define, and #endif to prevent multiple inclusions of the same file, which could otherwise cause compilation errors. The file includes standard libraries such as <iostream>, <string>, <vector>, <iomanip>, <ctime>, <limits>, <sstream>, <cctype>, and <cstdlib>. These libraries provide the necessary functionality for input/output operations, string manipulation, dynamic storage with vectors, formatting output, date and time handling, input validation, string-to-number conversion, character checking, and general utility functions.

In addition to the data structures, the header file declares several utility and system functions. Utility functions handle tasks such as retrieving the current date, formatting strings, converting time to minutes, and determining attendance status. The system functions provide the core functionality of the program, including displaying the menu, registering and logging in users, marking attendance, and viewing reports either fully or by a specific date. By separating these declarations into a header file, the program achieves modularity, making it easier to maintain, expand, and implement in the corresponding .cpp files while keeping the code organized and readable.

```
#include "attendance_system.h"
#include <cstdlib>
#include <cctype>
#include <sstream>
#include <limits>
#include <iomanip>
#include <ctime>

using namespace std;

// --- Utility Functions ---

string getCurrentDate() {
    time_t now = time(NULL);
    tm* tm_info = localtime(&now);
    char buffer[11];
```

```
    strftime(buffer, sizeof(buffer), "%Y-%m-%d", tm_info);
    return string(buffer);
}

string trim(const string& s) {
    size_t a = s.find_first_not_of(" \t\r\n");
    if (a == string::npos) return "";
    size_t b = s.find_last_not_of(" \t\r\n");
    return s.substr(a, b - a + 1);
}

string lowerCopy(const string& s) {
    string out;
    out.reserve(s.size());
    for (size_t i = 0; i < s.size(); i++) {
        unsigned char c = static_cast<unsigned char>(s[i]);
        out.push_back((char)tolower(c));
    }
    return out;
}

int parseTimeToMinutes(const string& raw) {
    string s = trim(raw);
    if (s.empty()) return -1;

    string low = lowerCopy(s);
    bool hasAM = (low.find("am") != string::npos);
    bool hasPM = (low.find("pm") != string::npos);

    string digits = "";
    for (size_t i = 0; i < low.size(); i++) {
        char c = low[i];
        if ((c >= '0' && c <= '9') || c == ':') digits.push_back(c);
        else if (isspace((unsigned char)c)) digits.push_back(' ');
    }
    digits = trim(digits);

    size_t colon = digits.find(':');
    if (colon == string::npos) return -1;

    string hs = trim(digits.substr(0, colon));
    string ms = trim(digits.substr(colon + 1));
```

```
    if (hs.empty() || ms.empty()) return -1;

    for (size_t i = 0; i < hs.size(); i++) {
        if (!isdigit((unsigned char)hs[i])) return -1;
    }
    for (size_t i = 0; i < ms.size(); i++) {
        if (!isdigit((unsigned char)ms[i])) return -1;
    }

    int hours = atoi(hs.c_str());
    int minutes = atoi(ms.c_str());
    if (minutes < 0 || minutes > 59) return -1;

    if (hasAM || hasPM) {
        if (hours < 1 || hours > 12) return -1;
        if (hasPM && hours != 12) hours += 12;
        else if (!hasPM && hours == 12) hours = 0;
    } else {
        if (hours < 0 || hours > 23) return -1;
    }

    return hours * 60 + minutes;
}

string calculateStatus(const string& time_in) {
    int total = parseTimeToMinutes(time_in);
    if (total == -1) return "Absent";

    int present_cutoff = 7 * 60 + 30; // 7:30 AM
    int late_cutoff = 16 * 60;        // 4:00 PM

    if (total <= present_cutoff) return "Present";
    if (total <= late_cutoff) return "Late";
    return "Absent";
}

// --- Menu Functions ---

void displayMenu() {
    cout << "\n--- Attendance System Menu ---\n";
    cout << "1. Mark Attendance\n";
```

```
    cout << "2. View Full Attendance Report\n";
    cout << "3. View Attendance by Date\n";
    cout << "4. Logout\n";
    cout << "Enter your choice: ";
}

User* login(vector<User>& users) {
    string username, password;
    cout << "\n--- Login ---\n";
    cout << "Enter username: ";
    cin >> username;
    cout << "Enter password: ";
    cin >> password;

    for (size_t i = 0; i < users.size(); i++) {
        if (users[i].username == username && users[i].password == password)
{
            return &users[i];
        }
    }
    return NULL;
}

void registerUser(vector<User>& users) {
    string username, password;
    cout << "\n--- Register New User ---\n";
    cout << "Enter new username: ";
    cin >> username;

    for (size_t i = 0; i < users.size(); i++) {
        if (users[i].username == username) {
            cout << "Username already taken. Please choose another.\n";
            return;
        }
    }

    cout << "Enter new password: ";
    cin >> password;

    User newUser;
    newUser.username = username;
    newUser.password = password;
```

```
    users.push_back(newUser);

    cout << "Registration successful! You can now log in.\n";
}

void markAttendance(User& user) {
    string student_name, date, time_in;
    cin.ignore(); // Clear newline

    cout << "\nEnter student name: ";
    getline(cin, student_name);

    cout << "Enter date (YYYY-MM-DD) or press Enter for today: ";
    getline(cin, date);
    if (date.empty()) date = getCurrentDate();

    cout << "Enter time in: ";
    getline(cin, time_in);

    string status = calculateStatus(time_in);

    Attendance att;
    att.student_name = student_name;
    att.date = date;
    att.time_in = time_in;
    att.status = status;

    user.attendances.push_back(att);

    cout << "\nAttendance marked successfully!\n";
    cout << "Student: " << att.student_name << "\n";
    cout << "Date:    " << att.date << "\n";
    cout << "Time In: " << att.time_in << "\n";
    cout << "Status:  " << att.status << "\n";
}

void viewReport(const User& user) {
    cout << "\n--- Attendance Report for " << user.username << " ---\n";
    if (user.attendances.empty()) {
        cout << "No attendance records found.\n";
        return;
    }
```

```
    cout << left << setw(20) << "Student Name"
         << setw(15) << "Date"
         << setw(18) << "Time In"
         << setw(10) << "Status" << "\n";
    cout <<
"-----------------------------------------------------------------\n";

    for (size_t i = 0; i < user.attendances.size(); i++) {
        const Attendance& a = user.attendances[i];
        cout << left << setw(20) << a.student_name
             << setw(15) << a.date
             << setw(18) << a.time_in
             << setw(10) << a.status << "\n";
    }
}

void viewReportByDate(const User& user) {
    string date;
    cin.ignore(); // Clear newline
    cout << "\nEnter date to view attendance (YYYY-MM-DD) or press Enter
for today: ";
    getline(cin, date);
    if (date.empty()) date = getCurrentDate();

    bool found = false;
    cout << "\n--- Attendance Report for " << user.username << " on " <<
date << " ---\n";
    cout << left << setw(20) << "Student Name"
         << setw(18) << "Time In"
         << setw(10) << "Status" << "\n";
    cout << "-----------------------------------------------------\n";

    for (size_t i = 0; i < user.attendances.size(); i++) {
        const Attendance& a = user.attendances[i];
        if (a.date == date) {
            cout << left << setw(20) << a.student_name
                 << setw(18) << a.time_in
                 << setw(10) << a.status << "\n";
            found = true;
        }
    }
```

```
    if (!found) cout << "No attendance records found for this date.\n";
}
```

Figure 4. attendance_system.cpp

The attendance_system.cpp file contains the implementation of all the core functions declared in attendance_system.h and provides the logic that drives the attendance system. It starts with utility functions that handle common tasks such as retrieving the current system date, trimming whitespace from strings, converting strings to lowercase, parsing time strings into minutes, and calculating a student's attendance status based on their time in. These functions ensure that user input is standardized and validated, allowing the system to correctly determine whether a student is Present, Late, or Absent. Functions like parseTimeToMinutes() and calculateStatus() encapsulate the time-based logic, making it easier to maintain and update the rules for attendance.

Furthermore, the file implements the main user and menu operations of the system. Functions such as displayMenu() present options to the user, while login() and registerUser() handle authentication and account creation. The markAttendance() function allows users to record attendance for students, storing the data in the User struct. The reporting functions, viewReport() and viewReportByDate(), provide formatted tables of attendance records, either for all entries or filtered by a specific date. By placing these implementations in a separate .cpp file, the program achieves modularity, readability, and maintainability, ensuring that all operations for tracking and reporting attendance are clearly organized and easy to extend or modify.

```
#include "attendance_system.h"

int main() {
    vector<User> users; // Start empty, no admin
    User* currentUser = NULL;
    int choice = 0;

    while (true) {
        if (currentUser == NULL) {
            cout << "\n--- Welcome to Attendance System ---\n";
            cout << "1. Login\n";
            cout << "2. Register\n";
            cout << "3. Exit\n";
            cout << "Enter your choice: ";
```

```
            if (!(cin >> choice)) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << "Please enter a valid number.\n";
                continue;
            }

            if (choice == 1) {
                if (users.empty()) {
                    cout << "No users registered yet. Please register
first.\n";
                    continue;
                }
                currentUser = login(users);
                if (currentUser == NULL) {
                    cout << "Invalid credentials. Try again.\n";
                    continue;
                }
                cout << "Login successful! Welcome, " <<
currentUser->username << "\n";
            }
            else if (choice == 2) {
                registerUser(users);
                continue;
            }
            else if (choice == 3) {
                cout << "Goodbye!\n";
                break;
            }
            else {
                cout << "Invalid choice. Try again.\n";
                continue;
            }
        }

        displayMenu();
        if (!(cin >> choice)) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Please enter a valid number.\n";
            continue;
        }
```

```
        switch (choice) {
            case 1:
                markAttendance(*currentUser);
                break;
            case 2:
                viewReport(*currentUser);
                break;
            case 3:
                viewReportByDate(*currentUser);
                break;
            case 4:
                cout << "Logged out.\n";
                currentUser = NULL;
                break;
            default:
                cout << "Invalid choice. Try again.\n";
        }
    }

    return 0;
}
```

Figure 5. Main.cpp

The main.cpp file serves as the entry point of the attendance system and controls the overall program flow. It maintains a list of registered users and tracks the currently logged-in user. The program runs in a loop that displays either the login/register menu or the user menu, depending on whether someone is logged in. Input validation ensures that users enter valid choices, and helpful messages guide users when invalid input or unregistered accounts are encountered. Once a user is logged in, the system presents options for marking attendance, viewing all records, viewing records by a specific date, or logging out. Each choice is handled through a switch statement that calls the corresponding functions from attendance_system.cpp. This design keeps the program organized, user-friendly, and modular, allowing the main logic to coordinate seamlessly with the utility and user functions while providing an intuitive interface for interacting with the attendance system.

Results and Discussion

This section presents and analyzes the results of the developed Attendance Report System. It highlights the functionality and performance of the system based on its main modules, including user registration, login authentication, attendance recording, and report generation. The results show how each part of the

program successfully contributes to automating and simplifying the attendance tracking process. The system ensures that users can easily record attendance data and view reports efficiently.

Upon running the program, the system initially displays a main menu that allows the user to register, log in, or exit. Once registered, users can log in using their credentials, confirming that the registration and authentication modules function correctly. After a successful login, the user is directed to the attendance menu, which includes options to mark attendance, view the full attendance report, or view records by date. During testing, the system accurately determined the attendance status (Present, Late, or Absent) based on the time entered, demonstrating the effectiveness of the calculateStatus() and markAttendance() functions. The reporting features properly displayed all attendance records in an organized table, confirming that the system meets its intended objectives of recording, classifying, and displaying attendance data accurately.

```
--- Welcome to Attendance System ---
1. Login
2. Register
3. Exit
Enter your choice: 2

--- Register New User ---
Enter new username: Marqui
Enter new password: 123456
Registration successful! You can now log in.

--- Welcome to Attendance System ---
1. Login
2. Register
3. Exit
Enter your choice: 1

--- Login ---
Enter username: 123456
Enter password: Maqrui
Invalid credentials. Try again.
```

Running the programs first displays a Welcome Menu where the user will choose to Login, Register, or Exit. To register a new user, the system requires an input of a new username and password, after which the system saves the new account and displays a message telling the user that they can now log in. When

logging in the username and password must be the same as the newly registered one, in this output I switched them to check if the system detects the error and prevents any wrong input to continue.

```
--- Welcome to Attendance System ---
1. Login
2. Register
3. Exit
Enter your choice: 1

--- Login ---
Enter username: Marqui
Enter password: 123456
Login successful! Welcome, Marqui

--- Attendance System Menu ---
1. Mark Attendance
2. View Full Attendance Report
3. View Attendance by Date
4. Logout
Enter your choice: 1

Enter student name: Joshua Garcia
Enter date (YYYY-MM-DD) or press Enter for today:
Enter time in: 7:30 AM

Attendance marked successfully!
Student: Joshua Garcia
Date:    2025-11-11
Time In: 7:30 AM
Status:  Present
```

In this output the username and password are correct so the user continues to the Attendance System Menu which shows the user actions that they can possibly do. The user marks attendance for a student using the current date with the time in of 7:30 AM. After that the system displays that the attendance for the student is marked successfully, with full details, and their status is “Present” since the time fits the cutoff.

```
--- Attendance System Menu ---
1. Mark Attendance
2. View Full Attendance Report
3. View Attendance by Date
4. Logout
Enter your choice: 1

Enter student name: Daniel Caesar
Enter date (YYYY-MM-DD) or press Enter for today:
Enter time in: 12:30 PM

Attendance marked successfully!
Student: Daniel Caesar
Date:    2025-11-11
Time In: 12:30 PM
Status:  Late
```

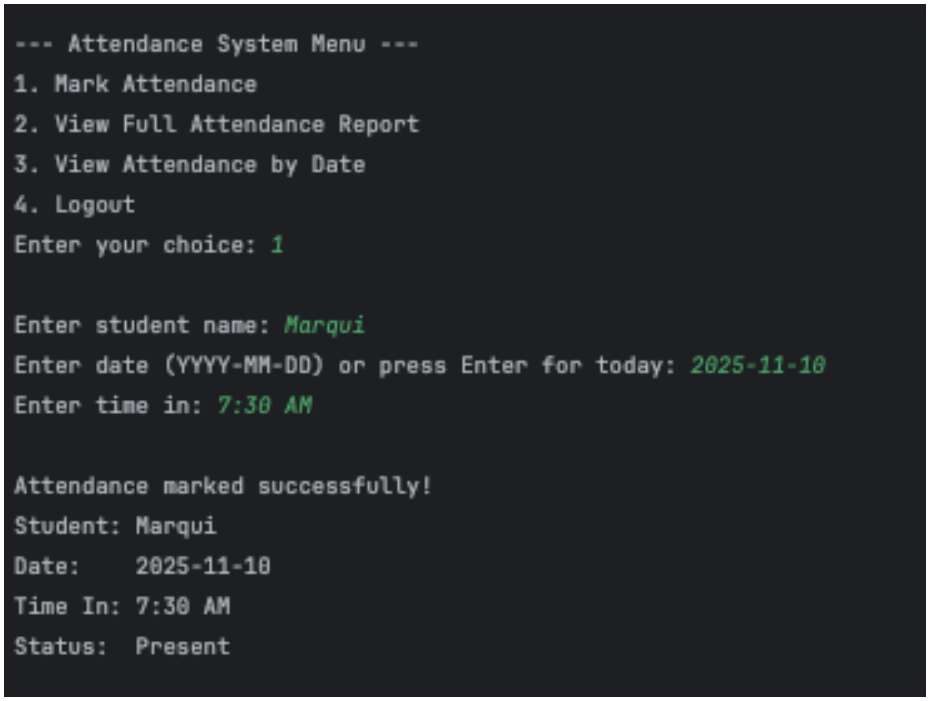
The user marks attendance for another student still using the current date but with a different time. The system shows it has been marked successfully, with full details, and the student's status is "Late" because his time fits the requirement of being marked as "Late".

```
--- Attendance System Menu ---
1. Mark Attendance
2. View Full Attendance Report
3. View Attendance by Date
4. Logout
Enter your choice: 1

Enter student name: Frank Ocean
Enter date (YYYY-MM-DD) or press Enter for today:
Enter time in: 5:00 PM

Attendance marked successfully!
Student: Frank Ocean
Date:    2025-11-11
Time In: 5:00 PM
Status:  Absent
```

Another student's attendance record is being marked with the same date being the current date and still a different time than the previous ones. This student is marked as "Absent" because their time is already beyond the cutoff.



In this output, the user marks attendance for a student but this time enters an exact date that is not the current date with a time of 7:30 AM. The student was marked as "Present" for that date and it showed the full details.

```
--- Attendance System Menu ---
1. Mark Attendance
2. View Full Attendance Report
3. View Attendance by Date
4. Logout
Enter your choice: 2

--- Attendance Report for Marqui ---
Student Name        Date           Time In           Status
------------------------------------------------------------
Joshua Garcia       2025-11-11     7:30 AM           Present
Daniel Caesar       2025-11-11     12:30 PM          Late
Frank Ocean         2025-11-11     5:00 PM           Absent
Marqui              2025-11-10     7:30 AM           Present
```

This output is the Full Attendance Report including the details of each student that was marked by the user. This report shows the student's name, date when they were marked, time in, and status. It clearly shows which students were Present, Late, or Absent on what day and what time did they get to class.

```
--- Attendance System Menu ---
1. Mark Attendance
2. View Full Attendance Report
3. View Attendance by Date
4. Logout
Enter your choice: 3

Enter date to view attendance (YYYY-MM-DD) or press Enter for today: 2025-11-10

--- Attendance Report for Marqui on 2025-11-10 ---
Student Name        Time In           Status
------------------------------------------------
Marqui              7:30 AM           Present
```

This Attendance Report shows only the report for a specific date. Only one student was marked on a specific date so the report only shows one student, including their name, time in, and status which is "Present" for that specific day.

In this output shows the Attendance System Menu and the user's choice to Logout of the system. The system receives its input and display a message indicating that the user has been logged out of the system.

.

Conclusion

Throughout the development of the Attendance Report System, the group went through a detailed and sometimes challenging process of designing, coding, and debugging the program. At first, several errors were encountered, especially in organizing the functions, linking the header file, and ensuring that each part of the system worked together properly. These issues required careful troubleshooting and revisions, which helped the team better understand how C++ handles structures, vectors, and function declarations. Despite the difficulties, these experiences allowed the group to strengthen their problem-solving and coding skills.

The completed program successfully allows users to register, log in, record attendance, and generate attendance reports efficiently. It can automatically determine whether a student is Present, Late, or Absent based on the time entered, providing an accurate and organized record of attendance. Additionally, the system includes features for viewing all attendance data or filtering reports by a specific date, enhancing its usability and accuracy.

Overall, the project demonstrates the effectiveness of using programming to automate and improve traditional attendance recording systems. Despite the technical difficulties encountered during

development, the group was able to create a functioning system that meets its intended purpose and contributes to more efficient attendance monitoring and management.

References

Gazi, S., & Jamal, A. (2019). Academic Honesty among Students of Selected Dental Colleges of Bangladesh. Bangladesh Journal of Medical Education, 10(1), 6–13. https://doi.org/10.3329/bjme.v10i1.44588

Lenuf. (2025, January 16). Challenges of manual attendance marking and how to overcome them. SoftDunamis Business Meal Consulting. https://softdunamis.com/blog/manual-attendance-marking/

Nwabuwe, A., Sanghera, B., Alade, T., & Olajide, F. (2023). Fraud Mitigation in Attendance Monitoring Systems using Dynamic QR Code, Geofencing and IMEI Technologies. International Journal of Advanced Computer Science and Applications, 14(4). https://doi.org/10.14569/ijacsa.2023.01404104

Rahaman, M., Islam, M. M., & Nandi, D. (2025). SmartPresence: Wi-Fi-based online attendance management for smart academic assistance. Journal of Electrical Systems and Information Technology, 12(1). https://doi.org/10.1186/s43067-025-00215-y

Santos, A. B. G., Balba, N. P., & Rebong, C. B. (2021). Attendance Monitoring System of Schools in the Philippines with an Inclusion of Optimization Query Algorithm. International Journal of Innovative Technology and Exploring Engineering, 10(8), 142–146. https://doi.org/10.35940/ijitee.h9149.0610821

Appendices

Appendix A: attendance_system.h

```
#ifndef ATTENDANCE_SYSTEM_H
#define ATTENDANCE_SYSTEM_H

#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
```

```cpp
#include <ctime>
#include <limits>
#include <sstream>
#include <cctype>
#include <cstdlib>

using namespace std;

struct Attendance {
    string student_name;
    string date;       // YYYY-MM-DD
    string time_in;
    string status;     // Present, Late, Absent
};

struct User {
    string username;
    string password;
    vector<Attendance> attendances;
};

// Utility functions
string getCurrentDate();
string trim(const string &s);
string lowerCopy(const string &s);
int parseTimeToMinutes(const string &raw);
string calculateStatus(const string &time_in);

// Menu & user functions
void displayMenu();
User* login(vector<User> &users);
void registerUser(vector<User> &users);
void markAttendance(User &user);
void viewReport(const User &user);
void viewReportByDate(const User &user);

#endif
```

Appendix B: attendance_system.cpp

```cpp
#include "attendance_system.h"
#include <cstdlib>
#include <cctype>
#include <sstream>
#include <limits>
#include <iomanip>
#include <ctime>

using namespace std;

// --- Utility Functions ---

string getCurrentDate() {
    time_t now = time(NULL);
    tm* tm_info = localtime(&now);
    char buffer[11];
    strftime(buffer, sizeof(buffer), "%Y-%m-%d", tm_info);
    return string(buffer);
}

string trim(const string& s) {
    size_t a = s.find_first_not_of(" \t\r\n");
    if (a == string::npos) return "";
    size_t b = s.find_last_not_of(" \t\r\n");
    return s.substr(a, b - a + 1);
}

string lowerCopy(const string& s) {
    string out;
    out.reserve(s.size());
    for (size_t i = 0; i < s.size(); i++) {
        unsigned char c = static_cast<unsigned char>(s[i]);
        out.push_back((char)tolower(c));
    }
    return out;
}

int parseTimeToMinutes(const string& raw) {
    string s = trim(raw);
    if (s.empty()) return -1;

    string low = lowerCopy(s);
    bool hasAM = (low.find("am") != string::npos);
    bool hasPM = (low.find("pm") != string::npos);

    string digits = "";
    for (size_t i = 0; i < low.size(); i++) {
```

```
        char c = low[i];
        if ((c >= '0' && c <= '9') || c == ':') digits.push_back(c);
        else if (isspace((unsigned char)c)) digits.push_back(' ');
    }
    digits = trim(digits);

    size_t colon = digits.find(':');
    if (colon == string::npos) return -1;

    string hs = trim(digits.substr(0, colon));
    string ms = trim(digits.substr(colon + 1));

    if (hs.empty() || ms.empty()) return -1;

    for (size_t i = 0; i < hs.size(); i++) {
        if (!isdigit((unsigned char)hs[i])) return -1;
    }
    for (size_t i = 0; i < ms.size(); i++) {
        if (!isdigit((unsigned char)ms[i])) return -1;
    }

    int hours = atoi(hs.c_str());
    int minutes = atoi(ms.c_str());
    if (minutes < 0 || minutes > 59) return -1;

    if (hasAM || hasPM) {
        if (hours < 1 || hours > 12) return -1;
        if (hasPM && hours != 12) hours += 12;
        else if (!hasPM && hours == 12) hours = 0;
    } else {
        if (hours < 0 || hours > 23) return -1;
    }

    return hours * 60 + minutes;
}

string calculateStatus(const string& time_in) {
    int total = parseTimeToMinutes(time_in);
    if (total == -1) return "Absent";

    int present_cutoff = 7 * 60 + 30; // 7:30 AM
    int late_cutoff = 16 * 60;        // 4:00 PM

    if (total <= present_cutoff) return "Present";
    if (total <= late_cutoff) return "Late";
    return "Absent";
}

// --- Menu Functions ---
```

```
void displayMenu() {
    cout << "\n--- Attendance System Menu ---\n";
    cout << "1. Mark Attendance\n";
    cout << "2. View Full Attendance Report\n";
    cout << "3. View Attendance by Date\n";
    cout << "4. Logout\n";
    cout << "Enter your choice: ";
}

User* login(vector<User>& users) {
    string username, password;
    cout << "\n--- Login ---\n";
    cout << "Enter username: ";
    cin >> username;
    cout << "Enter password: ";
    cin >> password;

    for (size_t i = 0; i < users.size(); i++) {
        if (users[i].username == username && users[i].password == password)
{
            return &users[i];
        }
    }
    return NULL;
}

void registerUser(vector<User>& users) {
    string username, password;
    cout << "\n--- Register New User ---\n";
    cout << "Enter new username: ";
    cin >> username;

    for (size_t i = 0; i < users.size(); i++) {
        if (users[i].username == username) {
            cout << "Username already taken. Please choose another.\n";
            return;
        }
    }

    cout << "Enter new password: ";
    cin >> password;

    User newUser;
    newUser.username = username;
    newUser.password = password;
    users.push_back(newUser);

    cout << "Registration successful! You can now log in.\n";
```

```
}

void markAttendance(User& user) {
    string student_name, date, time_in;
    cin.ignore(); // Clear newline

    cout << "\nEnter student name: ";
    getline(cin, student_name);

    cout << "Enter date (YYYY-MM-DD) or press Enter for today: ";
    getline(cin, date);
    if (date.empty()) date = getCurrentDate();

    cout << "Enter time in: ";
    getline(cin, time_in);

    string status = calculateStatus(time_in);

    Attendance att;
    att.student_name = student_name;
    att.date = date;
    att.time_in = time_in;
    att.status = status;

    user.attendances.push_back(att);

    cout << "\nAttendance marked successfully!\n";
    cout << "Student: " << att.student_name << "\n";
    cout << "Date:    " << att.date << "\n";
    cout << "Time In: " << att.time_in << "\n";
    cout << "Status:  " << att.status << "\n";
}

void viewReport(const User& user) {
    cout << "\n--- Attendance Report for " << user.username << " ---\n";
    if (user.attendances.empty()) {
        cout << "No attendance records found.\n";
        return;
    }

    cout << left << setw(20) << "Student Name"
         << setw(15) << "Date"
         << setw(18) << "Time In"
         << setw(10) << "Status" << "\n";
    cout <<
"-----------------------------------------------------------------\n";

    for (size_t i = 0; i < user.attendances.size(); i++) {
        const Attendance& a = user.attendances[i];
```

```
        cout << left << setw(20) << a.student_name
             << setw(15) << a.date
             << setw(18) << a.time_in
             << setw(10) << a.status << "\n";
    }
}

void viewReportByDate(const User& user) {
    string date;
    cin.ignore(); // Clear newline
    cout << "\nEnter date to view attendance (YYYY-MM-DD) or press Enter
for today: ";
    getline(cin, date);
    if (date.empty()) date = getCurrentDate();

    bool found = false;
    cout << "\n--- Attendance Report for " << user.username << " on " <<
date << " ---\n";
    cout << left << setw(20) << "Student Name"
         << setw(18) << "Time In"
         << setw(10) << "Status" << "\n";
    cout << "-----------------------------------------------------\n";

    for (size_t i = 0; i < user.attendances.size(); i++) {
        const Attendance& a = user.attendances[i];
        if (a.date == date) {
            cout << left << setw(20) << a.student_name
                 << setw(18) << a.time_in
                 << setw(10) << a.status << "\n";
            found = true;
        }
    }

    if (!found) cout << "No attendance records found for this date.\n";
}
```

Appendix C: main.cpp

```
#include "attendance_system.h"

int main() {
    vector<User> users; // Start empty, no admin
    User* currentUser = NULL;
    int choice = 0;

    while (true) {
        if (currentUser == NULL) {
            cout << "\n--- Welcome to Attendance System ---\n";
            cout << "1. Login\n";
            cout << "2. Register\n";
            cout << "3. Exit\n";
            cout << "Enter your choice: ";

            if (!(cin >> choice)) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << "Please enter a valid number.\n";
                continue;
            }

            if (choice == 1) {
                if (users.empty()) {
                    cout << "No users registered yet. Please register
first.\n";
                    continue;
                }
                currentUser = login(users);
                if (currentUser == NULL) {
                    cout << "Invalid credentials. Try again.\n";
                    continue;
                }
                cout << "Login successful! Welcome, " <<
currentUser->username << "\n";
            }
            else if (choice == 2) {
                registerUser(users);
                continue;
            }
            else if (choice == 3) {
                cout << "Goodbye!\n";
                break;
            }
            else {
```

```
                cout << "Invalid choice. Try again.\n";
                continue;
            }
        }

        displayMenu();
        if (!(cin >> choice)) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Please enter a valid number.\n";
            continue;
        }

        switch (choice) {
            case 1:
                markAttendance(*currentUser);
                break;
            case 2:
                viewReport(*currentUser);
                break;
            case 3:
                viewReportByDate(*currentUser);
                break;
            case 4:
                cout << "Logged out.\n";
                currentUser = NULL;
                break;
            default:
                cout << "Invalid choice. Try again.\n";
        }
    }

    return 0;
}
```