National Teachers College
629 J Nepomuceno, Quiapo, Manila, 1001 Metro Manila
Bachelor of Science in Information Technology

Final Project Documentation Data Structure

SGD 3 - Good Health and Well-Being: Health Assessment Checker

A Final Project
Presented to the Faculty of the
College of Information Technology

In partial fulfillment
of the Course Requirements for the degree
of Bachelor of Science in Information Technology

Submitted by:
Arcales, Lad Anthonyel
Diaz, Jereyme James B.
Kismundo, Kimberly A.
Laurino, Jerome E.
Tolentino, Christyn Louise Fatima L.

BSIT 2.2 - 2025

National Teachers College

Instructor:
Neypes, Justin Louise R.

Date:
December 13, 2025

I. INTRODUCTION

1.1. *Project Overview & UN SDG Target*

The Health-Tracker System is a C++ console-based application designed to help users monitor their daily wellness through health assessments, symptom checking, BMI evaluation, and personal health history tracking. It also includes administrative accounts with enhanced access.

This project aligns with UN Sustainable Development Goal (SGD) 3: Good Health And Well-being which specifically focuses on improving the physical and mental well-being of people around the world, and reducing illnesses through early detection and prevention.

1.2. *Problem Statement*

According to WPA, wellbeing is strongly linked to happiness and life satisfaction, and could be described as how you feel about yourself and your life. However, many individuals lack an accessible way to monitor their daily health status. Existing applications for health trackers often require internet access or complex interfaces.

This Health-Tracker System Solves those common problems by providing:

- User-Friendly health monitoring tools
- Symptom evaluation for early illness detection
- Automated storage of user activity
- Local file-based history tracking

II. REQUIREMENTS & ANALYSIS

2.1 Functional & Non-Functional Requirements

2.1.1 Functional Requirements (FR)

The system must be able to:

1. Register a new user and store their credentials.
2. Log-In user and admin using hashed password.
3. Provide a Daily Health Assessment
4. Perform Symptoms Checker and generate diagnosis.
5. Calculate BMI and classify health categories.
6. Record All actions into system logs and user history.
7. Allow users to view their complete health history.

- #### *2.1.2 Non-Functional Requirements (NFR)*

1. Usability: Simple, text-based navigation(CLI)
2. Performance: File I/O and must respond instantly.
3. Security: Password stored using hash function.
4. Compatibility: Must run on any standard C++ compiler

2.2 Data Requirements

Input Data Structures:

- **User Credentials:** Stored in users.txt as name username hashed_password.
- **User History:** Stored in user_history.txt as username|timestamp|category|details.

- **System Logs:** Stored in records.txt as [timestamp] action - User: username.
- **Daily Assessment Inputs:**
    - Water intake (int, number of cups)
    - Sleep hours (int)
    - Minutes of physical activity (int)
- **Symptoms Checker Inputs:** Y/N responses for sore throat, headache, cough, chest pain, back pain.
- **BMI Checker Inputs:**
    - Weight (float, kg)
    - Height (float, cm)

Expected Data Size:

- Users: tens to hundreds
- User history: potentially hundreds of records per user
- Logs: continuous appending, could grow large; handled using simple text files

2.3 Complexity Analysis

Feature	Operation	Complexity
User Log In/Registration	File read/write	O(n)
Daily Assessment	Simple arithmetic and string ops	O(1)
Symptoms Checker	Input check & string concat	O(1)
BMI Checker	Arithmetic calculations	O(1)
Save user/System Records	File append	O(1) per operation
View Records	Build linked list + bubble sort	$O(m^2)$, m = number of records for user
Task Queue	Enqueue/Dequeue	O(1)

Space Complexity:

- User info in memory: O(n), n = number of users
- Pending tasks queue: O(10) (fixed)
- Linked list for user history: O(m), m = number of records for user
- Sorted vector in `ViewRecords`: O(m)

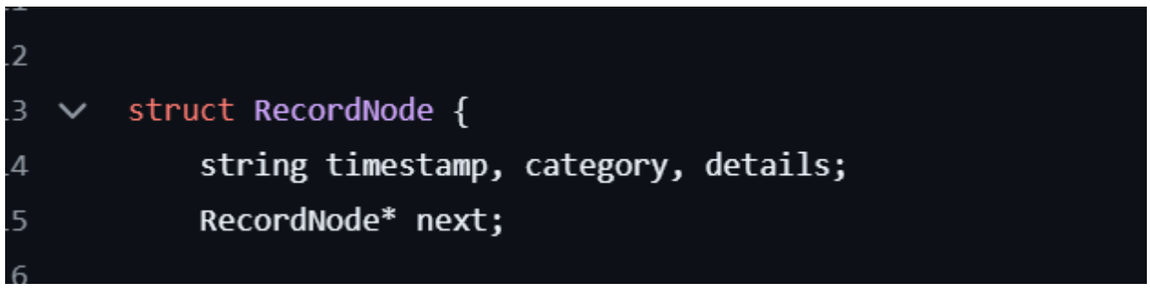
Justification : Bubble sort is used for simplicity since user history is small. For large datasets, more efficient sorting algorithms could reduce time complexity to O(m log m).

III. DESIGN SPECIFICATION

3.1 Core Data Structures

DSA #1: **Linked List**

- A linked list is used because the number of history records is unknown, and it allows the program to store each record dynamically without needing a fixed-size array.

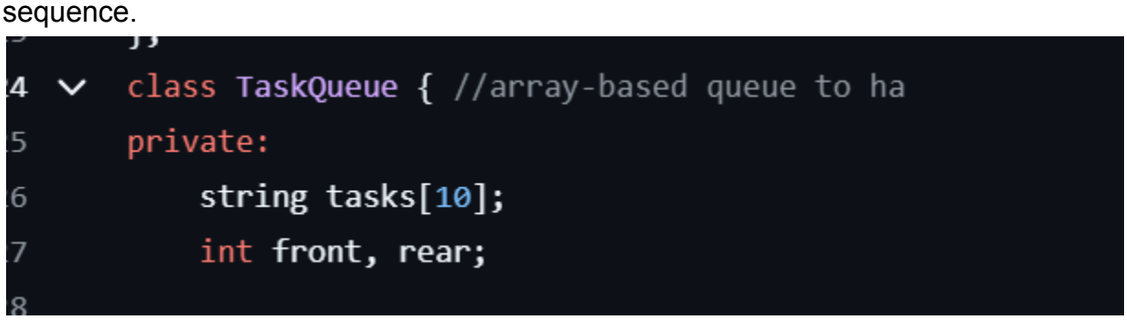


```
12
13  struct RecordNode {
14      string timestamp, category, details;
15      RecordNode* next;
16
```

IMPLEMENTATION : The linked list is implemented using a struct RecordNode that stores the record data and a pointer to the next node, allowing the program to add new records to the front of the list.

DSA #2: **Queue**

- A queue is used for daily tasks because it follows FIFO order, which matches how the system wants users to complete tasks one at a time in the correct sequence.



```
4  class TaskQueue { //array-based queue to ha
5  private:
6      string tasks[10];
7      int front, rear;
8
```

IMPLEMENTATION :The queue is implemented with a fixed-size array and two indices (front and rear), which are updated when tasks are added or removed.

DSA #3: **Bubble Sort**

- Bubble Sort is used because the number of records is small, and it provides a simple way to arrange the user's health history in order based on date and time.

```
80  void bubbleSort(vector<string>& records) {
81      int n = (int)records.size();
82      for (int i = 0; i < n - 1; i++) {
83          for (int j = 0; j < n - i - 1; j++) {
84              if (records[j] > records[j + 1]) {
85                  swap(records[j], records[j + 1]);
86              }
87          }
```

IMPLEMENTATION **:**Bubble Sort is used because the number of records is small, and it provides a simple way to arrange the user's health history in order based on date and time.

DSA #4: **Vector**

- A vector is used to store the records after reading them because it can grow automatically and makes sorting and displaying the data easier.

```
vector<string> sortedRecords;
```

- The vector is implemented using vector<string> and vector<RecordNode*>, where elements are inserted using push_back() to store and organize records for sorting.

DSA #5: **RecordNode struct**

- Linked list traversal is needed to go through each stored record, move the data into a vector, and free memory afterward.

```
RecordNode* current = head;
while (current) {
    // combine into one string: timestamp|category|details
    sortedRecords.push_back(current->timestamp + "|" +
```

- Linked list traversal is implemented using a while (current != nullptr) loop that walks through each node to process, copy, or delete the stored records.

3.2 Flowchart Explanation

1. **Start / Welcome**
   - Corresponds to `main()` calling `Welcome()` and `MainMenu()`.
   - Shows the system initialization and greeting.
2. **Login / Register Decision**
   - Matches the `MainMenu()` choices `[1] Log-In` and `[2] Register`.
   - Show flow: user chooses → goes to `LogIn()` or `Register()` functions.
3. **Authentication**
   - `LogIn()` checks admin via `User::isAdmin()` and regular user via reading `users.txt`.
   - Flow splits: successful login → Secondary Menu, failed login → error / back to main menu.
4. **Secondary Menu**
   - It corresponds to the `SecMenu(username)` function.
   - Options: `DailyAssessment()`, `SymptomsChecker()`, `BMIChecker()`, `ViewRecords()`, `Log-Out`.
   - Pending tasks queue (`TaskQueue`) shows workflow tracking.
5. **Data Processing**
   - Daily assessment: water, sleep, activity → calculates overall wellness.
   - Symptoms checker: user inputs symptoms → generates diagnosis.
   - BMI checker: weight/height → calculates BMI and category.
   - Health history: saved in `user_history.txt` → displayed via `ViewRecords()`.
6. **End / Log-Out**
   - Removes pending tasks, saves log, and returns to main menu.

3.3 Module Breakdown

3.3.1 Classes

1. **`User` class**
   - Stores admin accounts and checks admin authentication (**`isAdmin()`**).
   - Used in login to verify admin users.
2. **`TaskQueue` class**
   - Manages pending tasks per user.
     Functions: **`enqueue()`**, **`dequeue()`**, **`display()`**, **`clear()`**.
   - Ensures sequential flow of health tasks (assessment → symptoms → BMI).
3. **`RecordNode` struct**
   - Used for linked list representation of user history.
   - Fields: **`timestamp`**, **`category`**, **`details`**.

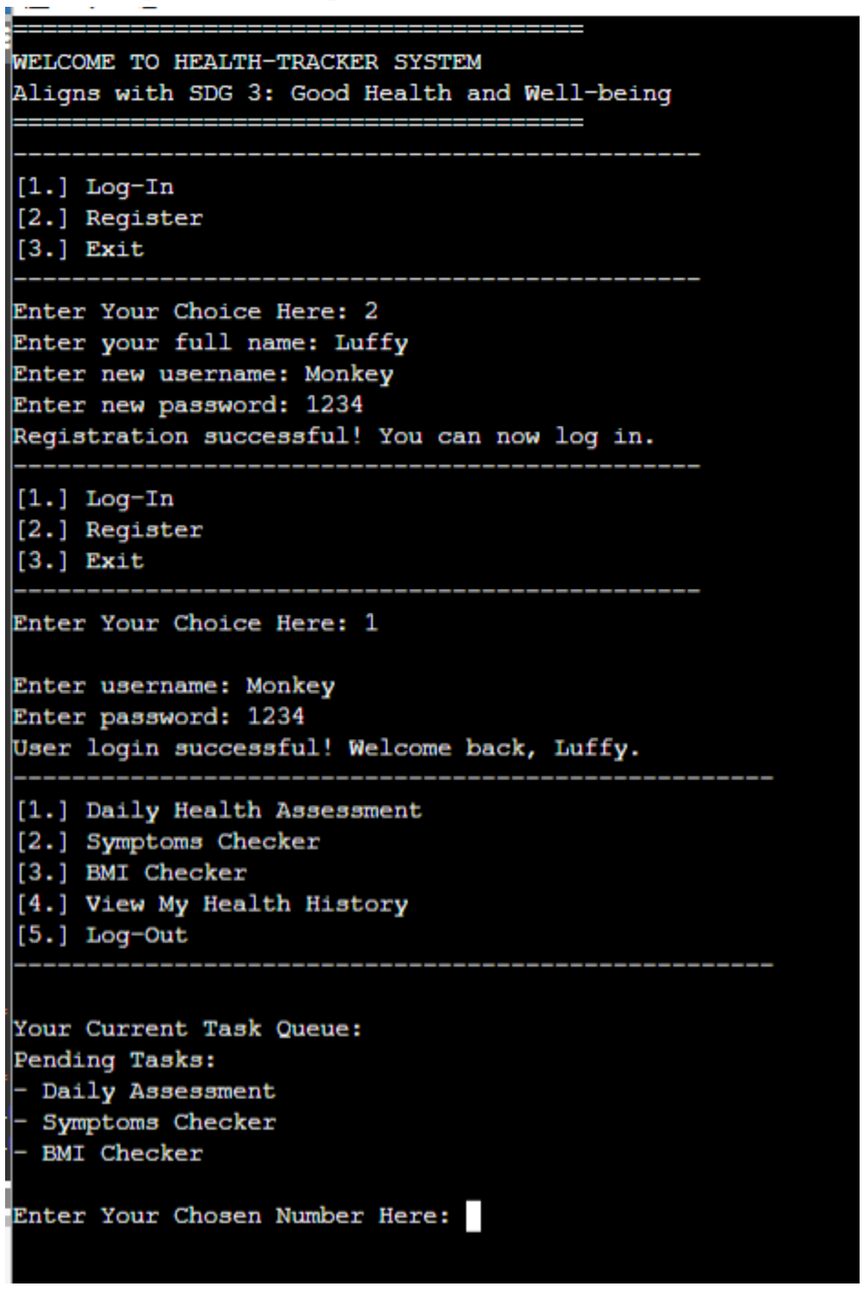
3.3.2 Interaction Sequence

1. **Program Start** → `main()` calls `Welcome()` → shows greeting → calls `MainMenu()`.
2. **Main Menu** → user chooses **Log-In** or **Register.**
    - **Log-In:** checks admin **(**`User::isAdmin`**)** or regular user **(**`users.txt`**).**
    - **Register:** saves new user to `users.txt`**.**
3. **After Login** → initialize `TaskQueue` → enqueue tasks:
    - Daily Assessment
    - Symptoms Checker
    - BMI Checker
4. **Secondary Menu (**`SecMenu`**)** → user selects task:
    - **Daily Assessment** → calculates wellness → saves record → dequeue task
    - **Symptoms Checker** → checks symptoms → saves record → dequeue task
    - **BMI Checker** → calculates BMI → saves record → dequeue task
    - **View Records** → displays history table and statistics
    - **Log-Out** → clears tasks → saves log → back to main menu
5. **End** → user can log in again or exit.

IV. TESTING & RESULTS

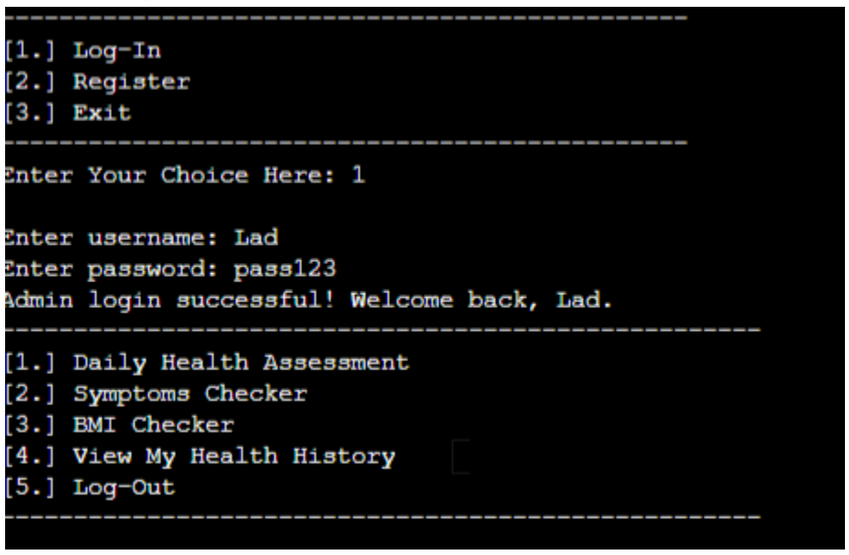
4.1 Test Cases

Test 1: Register and Log in as a user



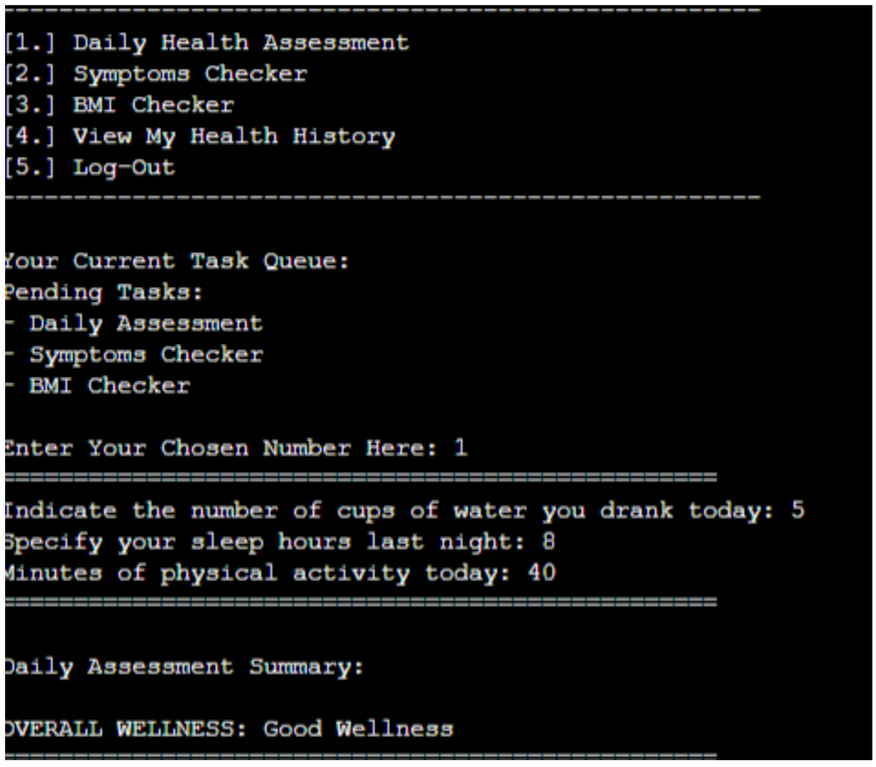
Result: The system successfully accepts new user information, saves it to the file, and allows the same user to log in using the registered credentials.

Test 2: Log in as an admin



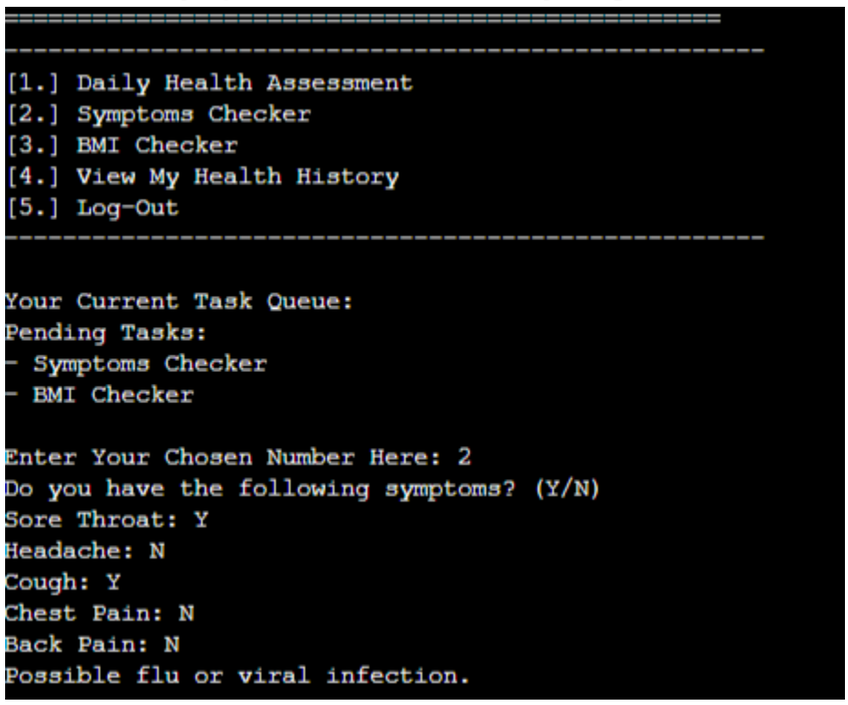
Result: The admin account is correctly recognized, granting access to admin-only features without errors.

Test 3: Perform daily assessment and view saved record



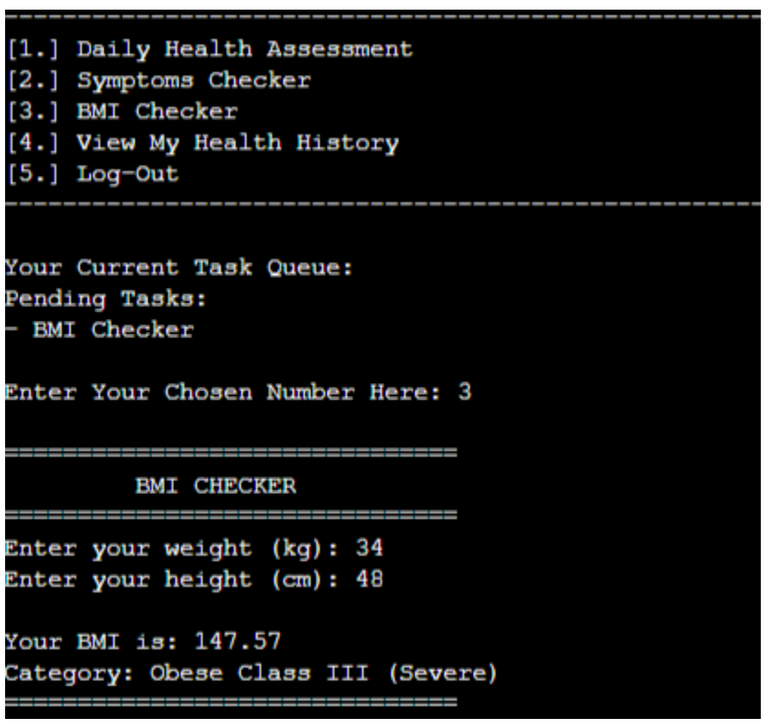
Result: The system records all daily assessment answers, stores them in the history file, and correctly displays the saved record when viewed.

Test 4: Use symptom checker and verify diagnosis



Result: The symptom checker processes the user's inputs, generates an appropriate diagnosis, and displays the expected result without crashing.

Test 5: Calculator BMI and confirm category



Result: The BMI calculator correctly computes BMI, determines the correct category (underweight, normal, overweight, etc.), and outputs accurate results.

4.2 Performance Test

NFR1 is satisfied.

The system continues to perform smoothly even with more than 50 stored records.

V.

main.cpp | records.txt | user_history.txt | users.txt

```
Hello Koi 11194622142508650503
Luffy Monkey 15651099383784684535
Nami Swan 5337992066015343258
Zoro Swords 5608050588507810076
Ussop Sniper 14257932435650712635
Sanji Ladies 8766191141532947413
Uta Genesis 15489772233559468697
Hancock Luffy 18095520000972273125
Rebecca Father 688532784384332204
Shirahoshi Crybaby 14491735811803874703
Vivi Princess 17416982412963892089
Jinbei Warlord 10981009179514145925
Sabo Dispair 6166207990078289545
Ace BigBrother 9379935939828261372
Naruto TailedBeast 16230207193790211793
Sakura Healer 8037141841967606425
Kakashi Sensei 9514281048529261330
Minato Hokage 20739551896078231 49
Onepiece Oda 2033997885575518818
Danaya Lupa 12832163441578054924
Elena Tubig 12503991577727592956
Amihan Hangin 15069345541072875639
Pirena Apoy 13749290555277881070
Hello Wattsup 15020836907419305748
Finn Hero 14139755766493303216
Jake Dog 13659938479592575040
Marceline Vampire 8416810612029012001
Simon IceKing 15651099383784684535
Mitena IceQueen 30413636020120483 45
Lisa DancerQueen 151222382741302987 67
Jennie RapperQueen 4672357166922844825
Rose CuteVoice 5101022709990741963
Jisoo ImOkay 12503991577727592956
Layla LaserCanon 6735654899171688824
Zhask Spawn 10981009179514145925
Hylos IronSteel 8483556159645703315
Estes HelloMyFriend 56789620972122853 82
Saber KillAllAtOnce 7029721069748577707
Granger WannaTakeALookInside 5129806908368014739
Alpha Protect 9606518547543691265
Jihyo Leader 172949025344497331 86
Sana Angel 60185272360872195 60
Nayeon Pop 3797325102496617778
Momo Boom 10284201567629518930
Blackpink Blinks 176379533567255086 53
Twice Once 378499767973530767
BlackBeard Traitor 167501020110069409 47
Daniel Padilla 12922911309609974364
Liza Soberano 17461842440945854522
Kuma Pawpaw 16639525893857482405
```

CONCLUSION & CONTRIBUTIONS

5.1 Conclusion

The Health-Tracker System realized its ambition of developing a user-friendly and convenient method for watching a person's health. The system, through its Daily Assessment, Symptoms Checker, BMI Calculator, and history tracking features, not only provided an easy way to monitor but also helped users to be more conscious of their total health.

With the help of simple data structures like linked lists, queues, vectors, and bubble sort, the system was able to manage the storage, ordering, and user-friendly presentation of information with maximum efficiency. All tests confirmed that the program functions as intended and manages user data accurately.

In summary, the project not only fulfilled the requirements but also delivered a working health-monitoring application that reinforced the concept of health promotion and early awareness.

5.2 Individual Contributions

- **Kismundo, Kimberly :**
  - Github Set-Up
  - TaskQueue (Code)
  - RecordNode (Code)
  - Bubble sort (Code)
  - Main Menu (Code)
  - SecMenu (Code)
  - ViewRecord (Code)
  - Documentation (Intro)
- **Tolentino, Christyn Louise Fatima :**
  - Documentation Paper
  - Daily Assessment Checker (Code)
- **Laurino, Jerome :**
  - Flowchart
  - PPT
  - SymptomsChecker(Code)
  - Checking-Errors (Code)
- **Arcales, Lad Anthonyel :**
  - BMI Checker (Code)
  - Documentation Inputs
  - Github inputs
  - Checking Errors (Code)
- **Diaz, Jereyme James :**
  - Flowchart
  - Log-in (Code)
  - Register (Code)
  - SaveRecord (Code)
  - User Class (Code)