PROJECT REPORT
Topic – HealthDash

FUNDAMENTALS OF COMPUTER PROGRAMMING

SUBMITTED BY: **SUBMITTED TO:**

1. Project Title - HealthDash

2. Project Overview

Healthdash, is a personal wellness management system that allows users to log and track their health-related activities such as sleep, weight, hydration, diet, and exercise. It features user authentication (login/signup), record management (add, view, delete), progress tracking with graphical reports, and health reminders.

3. Objectives

- **Health Record Management**: Allows users to track and manage their daily health data, including sleep, weight, hydration, diet, and exercise, with options to add, view, and delete records.
- **Progress Tracking**: Provides graphical reports to visualize health data trends over time, helping users monitor their progress.
- **Health Reminders**: Enables users to set and view personalized health reminders to encourage healthy habits and maintain wellness goals.

4. Functional Requirements

- **User Authentication**: Users must be able to securely sign up, log in, and log out using unique credentials stored in a file.
- **Data Processing**: The system should allow users to manage health data (e.g., sleep, weight, hydration), processing it by reading/writing to text files and generating CSV exports.
- **Output Generation**: The system should generate the following outputs:
  - **Health Reports**: Display records of health data (such as hydration, weight, diet, etc.) to users, sorted by date.
  - **Graphs**: Visual representations of health data (e.g., sleep and weight trends) should be generated in the form of graphs using external tools like Gnuplot.
  - **Reminders**: Personalized health reminders should be displayed, based on user input. These reminders will be stored and retrieved from a centralized reminders file.

5. Non – Functional Requirements

1. **Performance**
   • **Response Time**:
   The app should respond to user inputs within **2-3 seconds** for typical operations such as logging health data or viewing records.
   • **Efficiency**:
   The app should minimize the consumption of system resources (CPU, memory) during its execution. It must provide smooth performance even on lower-end systems.

2. **Scalability**
   • **Data Storage**:
   As the user's health data grows over time, the app should be able to handle an increasing volume of logs without performance degradation. It should accommodate multiple health records while maintaining efficient operation.
   • **User Growth**:
   The app supports multiple users, ensuring that it can scale to handle additional users as the system grows.

3. **Security**
   • **Data Privacy**:
   Although higher-level data privacy measures are not yet implemented, password authentication is required to secure access to the system.
   • **Access Control**:
   Each user has a separate profile with individual data files, preventing unauthorized access to another user's health data.

4. **Usability**
   • **Ease of Use**:
   The app should provide an intuitive, user-friendly interface, enabling users to easily input health data. The text-based interface should offer clear prompts, instructions, and helpful messages to guide users.
   • **Error Handling**:
   The system must provide clear error messages when users make invalid inputs, helping them understand and correct mistakes.

5. **Availability**
   • **Reliability**:
   The system must be reliable and should not crash during normal operations. Basic exception handling must be implemented to manage unexpected inputs and ensure stability.
   • **Data Availability**:
   Health logs should be accessible to the user at any time the app is run. As data is stored locally in files, users should be able to access their logs even after restarting the app.

6. **Maintainability**
   • **Code Modularity**:
   The app should be developed using a modular approach, where components such as user input handling, data storage, and retrieval are separated into manageable modules for easier maintenance.
   • **Ease of Updates**:
   Future updates, bug fixes, and improvements should be easy to implement without requiring a complete rewrite of the application. Clear and consistent coding practices should be followed to ensure maintainability.

6. System Architecture Overview

- **User Interface (UI)**:
  The front-end of the system is a command-line interface (CLI) that facilitates user interaction, allowing them to perform operations such as sign-up, login, and managing health records. The UI handles user inputs, providing appropriate prompts and feedback.
- **Authentication & User Management**:
  Authentication is handled via credential verification stored in a flat text file (`users.txt`). Upon successful login, a user session is initiated, and the system grants access to user-specific data files, ensuring secure access control for health data management.
- **Health Data Management**:
  Health-related metrics (such as weight, sleep, hydration, workout, and diet) are stored in a set of user-specific text files (e.g., `username_Sleep.txt`, `username_Weight.txt`). The system

enables CRUD (Create, Read, Update, Delete) operations on this data and supports data export in CSV format for analytics and reporting purposes.

- **Reminder Management**:
  Reminders, such as hydration or sleep-related alerts, are stored in `reminders.txt`. The system checks these reminders at periodic intervals and notifies users as needed, ensuring adherence to their health goals. Reminders are scheduled and dynamically updated based on user preferences.
- **Data Processing Layer**:
  The data processing layer handles the core operations of parsing and modifying health data files, calculating averages, and generating analytics reports when requested. It integrates with the file storage system to read and write data efficiently and handles any data export functionality in CSV format for further analysis.
- **File System**:
  The system utilizes a file-based storage model with plain text files for simplicity and efficiency. Each user's data is segregated into individual text files, categorized by health metric type (e.g., sleep, weight). The file system is structured in a way that provides easy data retrieval and modification based on user identification.

7. Technologies Used

- **Programming Languages**: C for backend logic and user interface.
- **Libraries/Frameworks**: Standard C libraries (e.g., stdio.h, stdlib.h, string.h, time.h) for file handling, data processing, and user interaction.
- **Graphing Tool**: **Gnuplot** for plotting graphs based on exported CSV data.
- **Data Storage**: Text files for storing user health data and records (e.g., sleep, weight, hydration).
- **File Format**: CSV for exporting data to facilitate easy graph plotting.

8. Data Requirements

- The system manages user-related data, including credentials and health metrics, stored in text files. User credentials are maintained in users.txt, while health records (such as sleep, weight, hydration, workout, and diet) are stored in user-specific files (e.g., username_Sleep.txt, username_Weight.txt). Health data can be exported to CSV format (sleep_data.csv, weight_data.csv) for analysis and reporting. Additionally, health reminders are stored in a central file (reminders.txt). All data is stored in plain text format, ensuring ease of access and manipulation across different system functionalities.

9. Assumptions and Constraints

- **User Base**:
  The application assumes that users have basic familiarity with technology and can interact with a text-based interface. No advanced technical knowledge is required for usage.
- **Device Availability**:
  The app assumes users will have access to devices with sufficient local storage to store health data files.
- **Single User Focus**:
  While the app supports multiple users, each user is assumed to be using the app independently on separate devices or profiles, and it is assumed that user data will not be shared across multiple devices simultaneously.
- **Cross-Platform Compatibility**:
  As the application is text-based, it is assumed to work on all platforms (Windows, macOS, Linux) that support the programming language used. However, future updates may be required to optimize the app for each platform.
- **Limited Features**:
  The current version does not include advanced features such as real-time data syncing across devices or integration with external health data sources (e.g., wearables, third-party APIs), which may be considered in future versions.

10. Timeline and Milestones

- **Phase 1: Planning and Requirements** – 01.11.24
- **Phase 2: Designing the Flow** – 05.11.24
- **Phase 3: Core Features Implementation** – 09.11.24
- **Phase 4: Advanced Features & Error Handling** – 13.11.24
- **Phase 5: Testing and Debugging** – 16.11.24
- **Phase 6: Final Optimization & Documentation** – 12.12.24
- **Phase 7: Deployment and Completion** – 18.12.24

11. References

https://www.youtube.com/watch?v=hCLIDph7-mU

https://www.geeksforgeeks.org/basics-file-handling-c/

https://www.geeksforgeeks.org/top-healthcare-app-development-ideas-for-startups/

12. Appendices (if any)

Drive Link containing demonstration videos :

https://drive.google.com/drive/folders/1aL9hm24pUDvRVwx49yP8mBzWNhGGSdAO?usp=sharing

SOURCE CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAXLEN 50 // Maximum length for username and password

// Main submenu functions
void mng_record(char *username);  // Manage Records (Add, Delete, View)
#include <stdio.h>
```

```c
#include <string.h>
#include <stdlib.h>

// Function to display the contents of a file
void display_file_content(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file %s.\n", filename);
        return;
    }

    char line[200];
    while (fgets(line, sizeof(line), file)) {
        printf("%s", line);
    }

    fclose(file);
}



void export_sleep_data_to_csv(char *username) {
    char sleep_filename[100], csv_filename[100];
    snprintf(sleep_filename, sizeof(sleep_filename), "%s_Sleep.txt", username);
    snprintf(csv_filename, sizeof(csv_filename), "sleep_data.csv");

    FILE *sleep_file = fopen(sleep_filename, "r");
    if (sleep_file == NULL) {
        printf("No sleep records found.\n");
        return;
    }

    FILE *csv_file = fopen(csv_filename, "w");
    if (csv_file == NULL) {
        printf("Error opening CSV file for writing.\n");
        return;
    }

    // Write headers for CSV
    fprintf(csv_file, "Date, Sleep Duration (minutes)\n");

    char line[200];
    char date[100];
    int sleep_duration;

    // Read and write sleep data
    while (fgets(line, sizeof(line), sleep_file)) {
        if (sscanf(line, "Sleep: %d minutes, DateTime: %s", &sleep_duration, date)
== 2) {
```

```
            fprintf(csv_file, "%s, %d\n", date, sleep_duration);
        }
    }

    fclose(sleep_file);
    fclose(csv_file);

    printf("Sleep data exported to %s\n", csv_filename);
}

void export_weight_data_to_csv(char *username) {
    char weight_filename[100], csv_filename[100];
    snprintf(weight_filename, sizeof(weight_filename), "%s_Weight.txt", username);
    snprintf(csv_filename, sizeof(csv_filename), "weight_data.csv");

    FILE *weight_file = fopen(weight_filename, "r");
    if (weight_file == NULL) {
        printf("No weight records found.\n");
        return;
    }

    FILE *csv_file = fopen(csv_filename, "w");
    if (csv_file == NULL) {
        printf("Error opening CSV file for writing.\n");
        return;
    }

    // Write headers for CSV
    fprintf(csv_file, "Date, Weight (kg)\n");

    char line[200];
    char date[100];
    float weight;

    // Read and write weight data
    while (fgets(line, sizeof(line), weight_file)) {
        if (sscanf(line, "Weight: %f kg, DateTime: %s", &weight, date) == 2) {
            fprintf(csv_file, "%s, %.2f\n", date, weight);
        }
    }

    fclose(weight_file);
    fclose(csv_file);

    printf("Weight data exported to %s\n", csv_filename);
}


void plot_graph(const char *csv_filename) {
```

```
    // Plotting using gnuplot with a pause to keep the window open until the user
closes it manually
    char command[200];
    snprintf(command, sizeof(command),
             "gnuplot -e \"set datafile separator ','; set title 'Health Data'; set
xlabel 'Date'; set ylabel 'Value'; plot '%s' using 1:2 with linespoints title '%s';
pause -1\"",
             csv_filename, csv_filename);

    int result = system(command);
    if (result == 0) {
        printf("Graph plotted successfully.\n");
    } else {
        printf("Failed to plot the graph.\n");
    }
}

// Progress function to view records
void progress(char *username) {
    int choice;

    printf("Choose an option:\n");
    printf("1. View all records for your username\n");
    printf("2. Graphical Report\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        // Existing functionality to view records
        int records_found = 0;

        // Define the list of possible file types
        const char *file_types[] = {"Workout", "Diet", "Hydration", "Sleep",
"Weight", "Steps"};

        // Try to open each file and display its contents
        for (int i = 0; i < 6; i++) {
            char filename[100];
            sprintf(filename, "%s_%s.txt", username, file_types[i]);

            FILE *file = fopen(filename, "r");
            if (file != NULL) {
                records_found = 1;
                printf("\nContents of %s file:\n", filename);
                display_file_content(filename);  // Display content of the file
                fclose(file);
            }
```

```
        }

        if (!records_found) {
            printf("No records found for user %s.\n", username);
        }
    } else if (choice == 2) {
        // Graphical Report Option
        int graph_choice;
        printf("Which graph do you want to plot?\n");
        printf("1. Sleep Graph\n");
        printf("2. Weight Graph\n");
        printf("Enter your choice: ");
        scanf("%d", &graph_choice);

        if (graph_choice == 1) {
            export_sleep_data_to_csv(username);
            plot_graph("sleep_data.csv");
        } else if (graph_choice == 2) {
            export_weight_data_to_csv(username);
            plot_graph("weight_data.csv");
        } else {
            printf("Invalid choice. Returning to main menu.\n");
        }
    } else if (choice == 3) {
        printf("Exiting progress view.\n");
    } else {
        printf("Invalid choice. Returning to main menu.\n");
    }
}

void hlth_remndr(char *username); // Health Reminders (Set and View)

// Functions for user entry
int userenter();    // User login/signup
int login();        // Login
int signup();       // Signup

// Manage records: Hydration, Weight, Exercise, Sleep, Diet
void update_record(char *username, const char *type);
void delete_record(char *username, const char *type);
void view_record(char *username, const char *type);

// Health reminders functions
void set_reminder(char *username);
void view_reminders(char *username);

// Main menu
int mainmenu() {
```

```c
    int d;
    printf("_\n*------------\n|------ MAIN MENU ------|\n");
    printf("|1. Manage Daily Records|\n");
    printf("|2. View Progress       |\n");
    printf("|3. Health Reminders    |\n");
    printf("|4. Exit Program        |\n_");
    printf("\nEnter your choice: ");
    scanf("%d", &d);
    return d;
}

// Main function
int main() {
    printf("*\n");
    printf("              Welcome to HEALTHDASH   \n         your personal wellness
companion\n");
    printf("*\n");
    printf("Please login or sign up to continue:\n");

    char a; // Login exit variable
    char username[MAXLEN]; // Store username globally for session
    while (1) { // Loop to repeat userenter
        if (userenter(username) == 1) { // userenter true
            while (1) { // mainmenu infinite loop
                int choice = mainmenu(); // mainmenu only once
                switch (choice) { // Check input
                    case 1: mng_record(username); break;  // Manage Records
                    case 2: progress(username); break;    // View Progress
                    case 3: hlth_remndr(username); break; // Health Reminders
                    case 4:
                        printf("Exiting the program. Goodbye!\n");
                        return 0; // Exit
                    default:
                        printf("Invalid input. Try again.\n");
                }
            }
        } else { // userenter false
            printf("Invalid choice, user or password. Do you want to try again? Y/N
\n");
            scanf(" %c", &a);
            if (a == 'N' || a == 'n') { // If user wants to exit
                printf("Exiting\n See you next time!\n");
                return 0; // Exit
            }
        }
    }
}
```

```c
// User Entry (Login/Signup)
int userenter(char *username) {
    int a;
    printf("Are you already a user, or do you want to sign up?\n");
    printf("1. Already a user\n2. Sign up\n");
    scanf("%d", &a);
    switch (a) {
        case 1: return login(username);  // Pass username pointer
        case 2: return signup(username); // Pass username pointer
        default: return 0;
    }
}

// Login Function
int login(char *username) {
    char password[MAXLEN];
    char file_username[MAXLEN], file_password[MAXLEN];
    FILE *file = fopen("users.txt", "r");
    if (file == NULL) {
        printf("No users found. Please sign up first.\n");
        return 0; // Failure
    }
    printf("Enter username: ");
    scanf("%s", username);
    printf("Enter password: ");
    scanf("%s", password);
    while (fscanf(file, "%s %s", file_username, file_password) != EOF) {
        if (strcmp(username, file_username) == 0 && strcmp(password, file_password)
== 0) {
            fclose(file);
            printf("Welcome to Healthdash user %s\n", username);
            return 1; // Success
        }
    }
    fclose(file);
    return 0;
}

// Signup Function
int signup(char *username) {
    char password[MAXLEN];
    FILE *file = fopen("users.txt", "a+");
    if (file == NULL) {
        printf("Error opening users.txt\n");
        return 0;
    }

    int is_duplicate;
```

```
    do {
        is_duplicate = 0;
        printf("Enter new username: ");
        scanf("%s", username);

        char file_username[MAXLEN], file_password[MAXLEN];
        rewind(file);
        while (fscanf(file, "%s %s", file_username, file_password) != EOF) {
            if (strcmp(username, file_username) == 0) {
                printf("Username already exists. Please choose another.\n");
                is_duplicate = 1;
                break;
            }
        }
    } while (is_duplicate);

    printf("Enter new password: ");
    scanf("%s", password);
    fprintf(file, "%s %s\n", username, password);
    fclose(file);

    // Create user-specific health files
    char health_filename[100];
    sprintf(health_filename, "%s_health_metrics.txt", username);
    FILE *health_metrics = fopen(health_filename, "w");

    if (health_metrics) {
        fclose(health_metrics);
    } else {
        printf("Error initializing user files.\n");
        return 0;
    }

    printf("Signup successful! Welcome user %s\n", username);
    return 1;
}

// Manage Records (Add, View, Delete)
void mng_record(char *username) {
    int choice;
    printf("Choose a category to manage records:\n");
    printf("1. Hydration\n2. Diet\n3. Workout\n4. Sleep\n5. Weight\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
        case 2:
```

```
        case 3:
        case 4:
        case 5: {
            // Inner menu for add, view, delete
            int action;
            printf("\n1. Add record\n2. View records\n3. Delete record\n");
            printf("Enter your choice: ");
            scanf("%d", &action);
            const char *type = (choice == 1) ? "Hydration" :
                               (choice == 2) ? "Diet" :
                               (choice == 3) ? "Workout" :
                               (choice == 4) ? "Sleep" : "Weight";

            switch (action) {
                case 1: update_record(username, type); break;
                case 2: view_record(username, type); break;
                case 3: delete_record(username, type); break;
                default:
                    printf("Invalid choice. Returning to main menu.\n");
            }
            break;
        }
        default:
            printf("Invalid choice. Try again.\n");
    }
}

// Update Record (Hydration, Weight, Exercise, Sleep, Diet)
void update_record(char *username, const char *type) {
    char filename[100];
    sprintf(filename, "%s_%s.txt", username, type);
    FILE *file = fopen(filename, "a");

    if (file == NULL) {
        printf("Error opening %s file.\n", type);
        return;
    }

    // Get current time and date
    time_t now = time(NULL);
    struct tm *t = localtime(&now);
    char date_time[100];
    strftime(date_time, sizeof(date_time), "%Y-%m-%d %H:%M:%S", t);

    char record[100];
    if (strcmp(type, "Workout") == 0) {
        int workout_type;
```

```
        printf("Select workout type: \n1. Cardio\n2. Yoga\n3. Gym\n4. Running\n5.
Sport\n");
        printf("Enter your choice: ");
        scanf("%d", &workout_type);

        char *workout_type_str;
        switch (workout_type) {
            case 1: workout_type_str = "Cardio"; break;
            case 2: workout_type_str = "Yoga"; break;
            case 3: workout_type_str = "Gym"; break;
            case 4: workout_type_str = "Running"; break;
            case 5: workout_type_str = "Sport"; break;
            default: workout_type_str = "Unknown"; break;
        }

        int duration;
        printf("Enter duration in minutes: ");
        scanf("%d", &duration);
        fprintf(file, "Workout: %s, Duration: %d minutes, DateTime: %s\n",
workout_type_str, duration, date_time);
    } else if (strcmp(type, "Diet") == 0) {
        char food_item[100];
        int quantity;
        printf("Enter the food item you ate: ");
        scanf(" %[^\n]%*c", food_item);
        printf("Enter quantity in grams: ");
        scanf("%d", &quantity);
        fprintf(file, "Food: %s, Quantity: %d grams, DateTime: %s\n", food_item,
quantity, date_time);
    } else if (strcmp(type, "Hydration") == 0) {
        float hydration;
        printf("Enter hydration amount in liters: ");
        scanf("%f", &hydration);
        fprintf(file, "Hydration: %.2f liters, DateTime: %s\n", hydration,
date_time);
    } else if (strcmp(type, "Weight") == 0) {
        float weight;
        printf("Enter weight in kg: ");
        scanf("%f", &weight);
        fprintf(file, "Weight: %.2f kg, DateTime: %s\n", weight, date_time);
    } else if (strcmp(type, "Sleep") == 0) {
        int sleep_duration;
        printf("Enter sleep duration in minutes: ");
        scanf("%d", &sleep_duration);
        fprintf(file, "Sleep: %d minutes, DateTime: %s\n", sleep_duration,
date_time);
    }
```

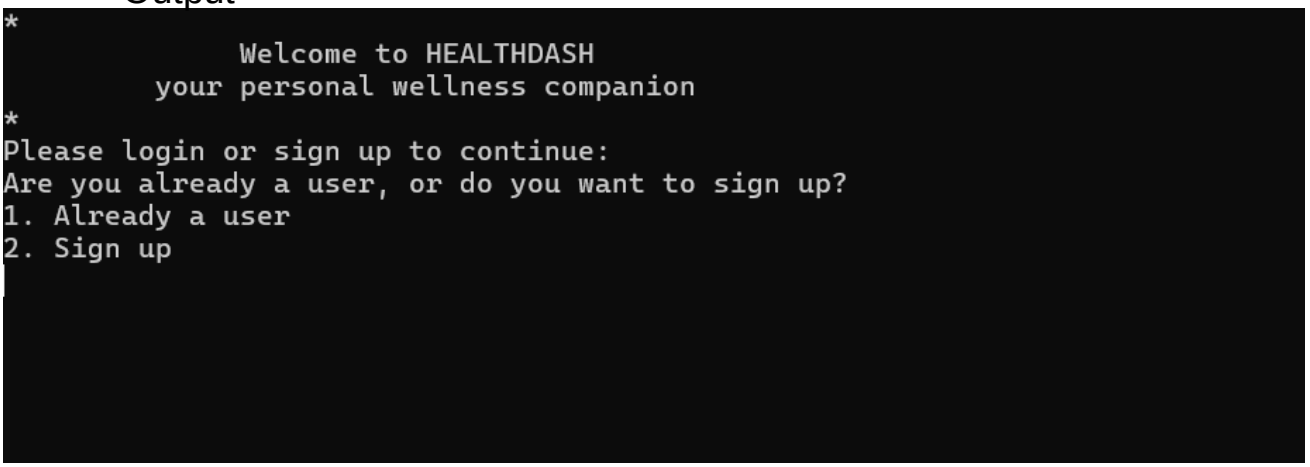
```c
    fclose(file);
    printf("%s record added successfully!\n", type);
}

// Delete Record (Hydration, Weight, Exercise, Sleep, Diet)
void delete_record(char *username, const char *type) {
    char filename[100];
    sprintf(filename, "%s_%s.txt", username, type);
    FILE *file = fopen(filename, "r");

    if (file == NULL) {
        printf("No records found for %s. Nothing to delete.\n", type);
        return;
    }

    // Read and display all records
    char line[200];
    int record_count = 0;
    while (fgets(line, sizeof(line), file)) {
        record_count++;
        printf("%d: %s", record_count, line);
    }
    fclose(file);

    if (record_count == 0) {
        printf("No records to delete.\n");
        return;
    }

    int action;
    printf("\n1. Delete all records\n2. Delete specific record\nEnter your choice:
");
    scanf("%d", &action);

    if (action == 1) {
        if (remove(filename) == 0) {
            printf("All %s records deleted.\n", type);
        } else {
            printf("Error deleting all %s records.\n", type);
        }
    } else if (action == 2) {
        int record_to_delete;
        printf("Enter the record number to delete: ");
        scanf("%d", &record_to_delete);

        FILE *file = fopen(filename, "r");
        FILE *temp_file = fopen("temp.txt", "w");
```

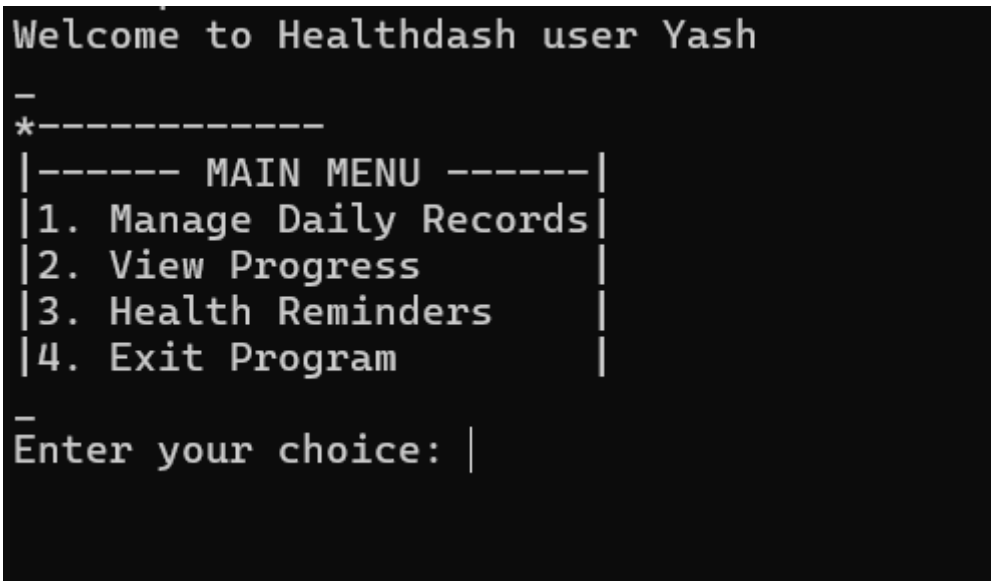
```
        if (file == NULL || temp_file == NULL) {
            printf("Error opening files.\n");
            return;
        }

        // Copy all records except the selected one to a temp file
        int current_line = 1;
        int deleted = 0;
        while (fgets(line, sizeof(line), file)) {
            if (current_line != record_to_delete) {
                fprintf(temp_file, "%s", line);
            } else {
                deleted = 1;
            }
            current_line++;
        }

        fclose(file);
        fclose(temp_file);

        // If record is found, replace the original file
        if (deleted) {
            if (remove(filename) == 0 && rename("temp.txt", filename) == 0) {
                printf("Record %d deleted successfully.\n", record_to_delete);
            } else {
                printf("Error deleting record %d.\n", record_to_delete);
            }
        } else {
            printf("Record %d not found.\n", record_to_delete);
            remove("temp.txt"); // Clean up temp file
        }
    } else {
        printf("Invalid choice. Returning to main menu.\n");
    }
}

// View Record (Hydration, Weight, Exercise, Sleep, Diet)
void view_record(char *username, const char *type) {
    char filename[100];
    sprintf(filename, "%s_%s.txt", username, type);
    FILE *file = fopen(filename, "r");

    if (file == NULL) {
        printf("No records found for %s.\n", type);
        return;
    }

    char line[200];
```

```
    printf("Viewing records for %s:\n", type);
    while (fgets(line, sizeof(line), file)) {
        printf("%s", line);
    }

    fclose(file);
}

// Health Reminders
void hlth_remndr(char *username) {
    int action;
    printf("\n1. Set reminder\n2. View reminders\nEnter your choice: ");
    scanf("%d", &action);

    if (action == 1) {
        set_reminder(username);
    } else if (action == 2) {
        view_reminders(username);
    } else {
        printf("Invalid input. Returning to main menu.\n");
    }
}

// Set Health Reminder
void set_reminder(char *username) {
    char reminder[100];
    printf("Enter a health reminder: ");
    scanf(" %[^\n]%*c", reminder);

    FILE *file = fopen("reminders.txt", "a");
    if (file == NULL) {
        printf("Error opening reminders file.\n");
        return;
    }

    // Get current time and date for reminder
    time_t now = time(NULL);
    struct tm *t = localtime(&now);
    char date_time[100];
    strftime(date_time, sizeof(date_time), "%Y-%m-%d %H:%M:%S", t);

    fprintf(file, "Reminder: %s, DateTime: %s, User: %s\n", reminder, date_time,
username);
    fclose(file);
    printf("Reminder set successfully!\n");
}

// View Health Reminders
```

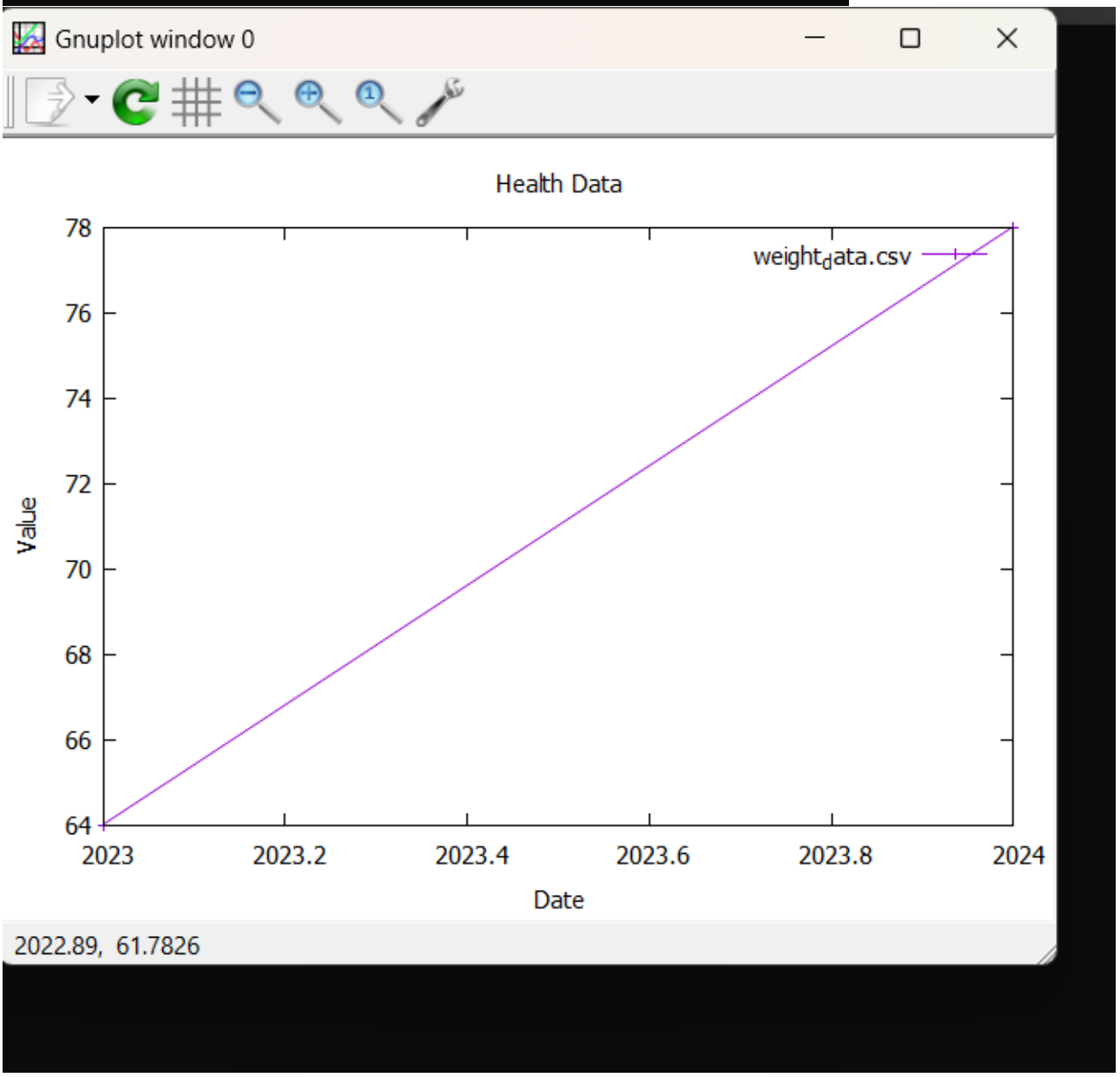
```c
void view_reminders(char *username) {
    FILE *file = fopen("reminders.txt", "r");

    if (file == NULL) {
        printf("No reminders found.\n");
        return;
    }

    char line[200];
    printf("Viewing reminders for user %s:\n", username);
    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, username)) {
            printf("%s", line);
        }
    }

    fclose(file);
}
```

Output





```
_
Enter your choice: 2
Choose an option:
1. View all records for your username
2. Graphical Report
3. Exit
Enter your choice: 2
```



```
Choose a category to manage records:
1. Hydration
2. Diet
3. Workout
4. Sleep
5. Weight
Enter your choice: 3

1. Add record
2. View records
3. Delete record
Enter your choice: 1
Select workout type:
1. Cardio
2. Yoga
3. Gym
4. Running
5. Sport
Enter your choice: 5
Enter duration in minutes: 120
Workout record added successfully!
```