UPES

UNIVERSITY OF TOMORROW

C programming Project

PROJECT REPORT

Project title:-Fitness tracker

Submitted By : Yashdeep Lamba
SAP ID: 590023466
Course Title: Programming in C
Course Code: CSEG1032
Semester: 1
Batch:-19

Submitted To : Mr. Rahul Prashad

1. ABSTRACT

This project implements a console-based **Fitness Activity Tracker** using the C programming language. The application allows the user to record multiple physical exercise sessions by storing details such as date, type of activity, duration, steps taken, distance covered, and estimated calories burned. The system uses an array of structures to store up to 100 activity entries and calculates calories automatically using a predefined formula. Through a menu-driven interface, the user can insert new activities and view all previously logged records. The project demonstrates the use of structured programming, modular functions, user input handling, and basic arithmetic operations to solve a real-world tracking problem.

2. OBJECTIVE

The objective of this project is to create a functional digital exercise log that assists users in tracking their fitness activities effectively. The main goals are:

- To apply C language concepts for building real-world applications.
- To utilize structures to store logically grouped data.
- To implement arrays for maintaining multiple fitness entries.
- To provide a simple menu-driven interface to add and view records.
- To calculate calories burned automatically using steps and duration.
- To demonstrate user input management, loops, conditional logic, and function separation in a practical programming scenario.

Overall, the project aims to convert a common health-related task (activity logging) into a simple computational tool, improving usability and reducing manual calculation errors.

3. PROBLEM DEFINITION

In a typical fitness journey, individuals need consistent tracking to measure progress and stay motivated. However:

- Users forget to note activities on a regular basis.
- Paper-based logs are difficult to maintain, edit, or compare.
- Manual calorie calculation is inconvenient and error-prone.
- Apps and online trackers may require registration, internet, or subscriptions.

These challenges result in reduced accountability, incomplete data, and inconsistent exercise routines.

Solution:

The Fitness Activity Tracker offers a lightweight offline solution. As a console program, it:

- Provides instant input without distractions.
- Stores activities systematically using structured formats.
- Automatically calculates calories without external tools.
- Displays all activities at any time in an organized manner.

This approach enhances user discipline, supports habit-building, and enables analysis of long-term trends.

4. SYSTEM DESIGN AND ALGORITHM

Data Structure Used

The project uses a **structure** to represent one fitness activity.

```c
typedef struct {
    char date[20];
    char type[20];
    int duration;
    int steps;
    float distance;
    float calories;
} Activity;
```

Each Activity stores:

- date – the date of the activity
- type – activity type (e.g., walk/run)
- duration – time in minutes

- steps – number of steps taken
- distance – distance covered in km
- calories – estimated calories burned

All activities are stored in an array:

```
#define MAX 100
Activity logData[MAX];
int count = 0;
```

- logData – an array that can store up to MAX activities.
- count – keeps track of how many activities are currently stored.

Functions Used

1. **calculateCalories** – to estimate calories burned

```
float calculateCalories(int steps, int duration) {
    return (steps * 0.04) + (duration * 5);
}
2.
```

This function uses a simple formula:

- Each step contributes some calories (steps * 0.04)
- Each minute of activity contributes some calories (duration * 5)

2.addActivity – to add a new fitness activity

```
void addActivity() {
    if (count >= MAX) {
        printf("Storage full!\n");
        return;
    }

    printf("\nEnter date (DD-MM-YYYY): ");
    scanf("%s", logData[count].date);

    printf("Enter activity type (walk/run): ");
    scanf("%s", logData[count].type);

    printf("Enter duration (minutes): ");
    scanf("%d", &logData[count].duration);

    printf("Enter steps: ");
    scanf("%d", &logData[count].steps);

    printf("Enter distance (km): ");
    scanf("%f", &logData[count].distance);
    logData[count].calories =
calculateCalories(logData[count].steps, logData[count].duration);

    printf("Activity added successfully!\n");
    count++;
}
```

- Checks if storage is full.
- Reads all details for a new activity.
- Calculates calories using calculateCalories.
- Stores everything in logData[count] and increments count.

3.viewActivities – to display all recorded activities

```
void viewActivities() {
    if (count == 0) {
        printf("\nNo activities recorded yet.\n");
        return;
    }

    printf("\n--- Fitness Activity Log ---\n");
    for (int i = 0; i < count; i++) {
        printf("\nEntry %d:\n", i + 1);
        printf("Date        : %s\n", logData[i].date);
        printf("Type        : %s\n", logData[i].type);
        printf("Duration    : %d min\n", logData[i].duration);
        printf("Steps       : %d\n", logData[i].steps);
        printf("Distance    : %.2f km\n", logData[i].distance);
        printf("Calories    : %.2f kcal\n", logData[i].calories);
    }
}
```

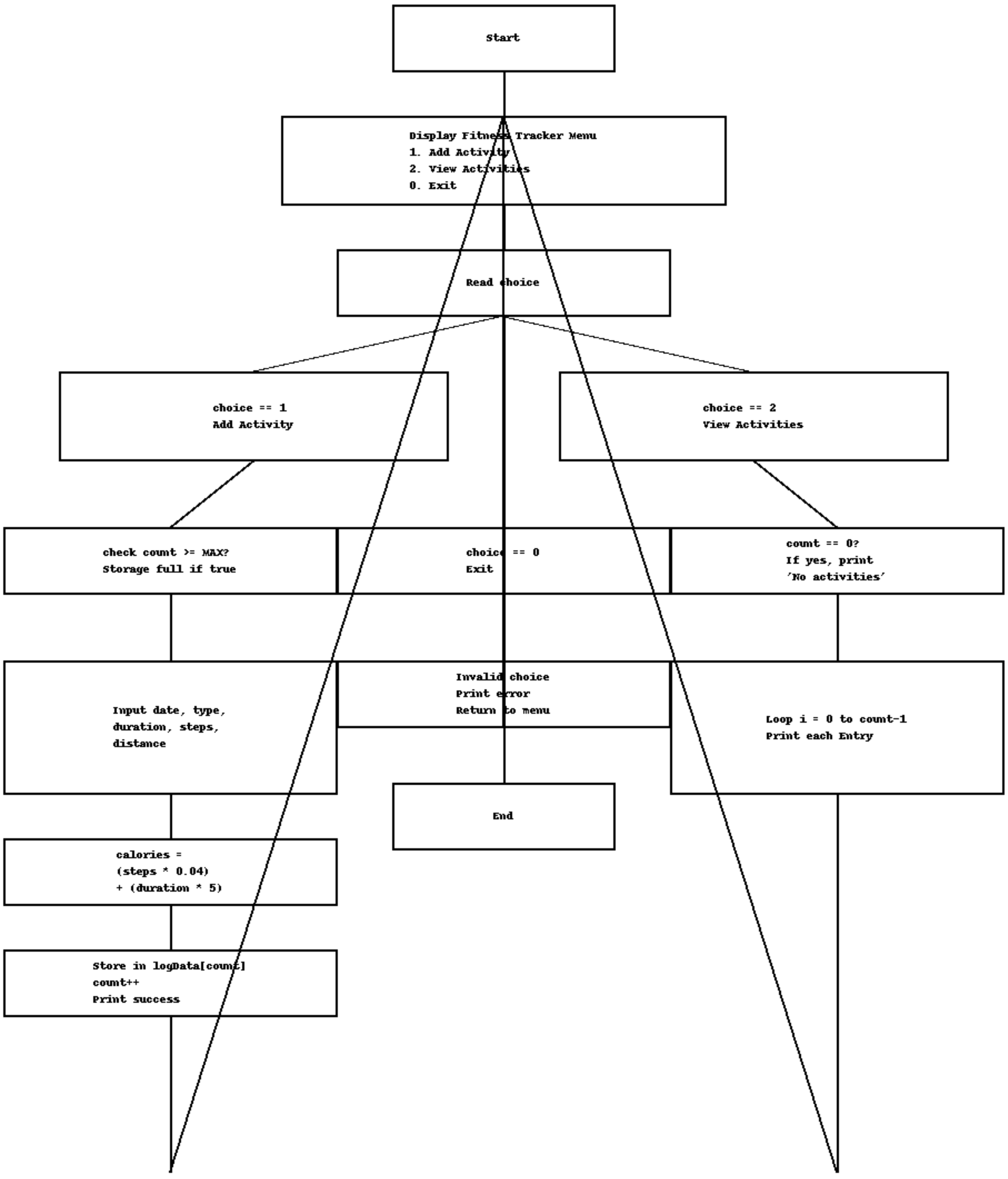
- **f no activities are recorded → message displayed.**
- **Else, prints details of each logged activity.**

Algorithm Example: Add Activity
Function: addActivity()

1. Check if count >= MAX:
    - If yes → show "Storage full!" and return.
2. Prompt user to input:
    - Date (DD-MM-YYYY format)
    - Activity type (walk/run)
    - Duration (minutes)
    - Steps
    - Distance (km)
3. Store data in logData[count]
4. Call calculateCalories()
5. Save returned value in logData[count].calories
6. Increment entry counter count++

7. Display success message
8. End

Flowchart:-


Start
Display Fitness Tracker Menu
1. Add Activity
2. View Activities
0. Exit
Read choice
choice == 1
Add Activity
choice == 2
View Activities
check count >= MAX?
Storage full if true
choice == 0
Exit
count == 0?
If yes, print
'No activities'
Invalid choice
Print error
Return to menu
Input date, type,
duration, steps,
distance
Loop i = 0 to count-1
Print each Entry
End
calories =
(steps * 0.04)
+ (duration * 5)
Store in logData[count]
count++
Print success

5. IMPLEMENTATION DETAILS

The program is implemented in C using the GCC compiler. A structure (Activity) is used to create logically organized containers for workout statistics. An array of structures enables storage of multiple entries without dynamic memory allocation. A separate function handles calorie calculation, supporting modularity and reusability. Loops handle repeated input entry, and conditional checks prevent out-of-bounds data insertion. The menu system is implemented using a while(1)loop with a switch, demonstrating structured and user-controlled flow. The clear separation of logic and computation highlights beginner-friendly modular programming practices.

Key Language Features Used:

File Streams: Used FILE * pointers to manage data persistence.

Standard I/O: Used printf and scanf for the user interface.

Header Files: Created scoreboard.h to share definitions across .c files.

6. Source Code

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

#define MAX 100

typedef struct {
    char date[20];
    char type[20];
    int duration;
    int steps;
    float distance;
    float calories;
} Activity;

Activity logData[MAX];
int count = 0;

float calculateCalories(int steps, int duration) {
    return (steps * 0.04) + (duration * 5);
}

void addActivity() {
    if (count >= MAX) {
        printf("Storage full!\n");
        return;
    }

    printf("\nEnter date (DD-MM-YYYY): ");
    scanf("%s", logData[count].date);

    printf("Enter activity type (walk/run): ");
    scanf("%s", logData[count].type);

    printf("Enter duration (minutes): ");
    scanf("%d", &logData[count].duration);

    printf("Enter steps: ");
    scanf("%d", &logData[count].steps);

    printf("Enter distance (km): ");
    scanf("%f", &logData[count].distance);

    logData[count].calories =
calculateCalories(logData[count].steps, logData[count].duration);

    printf("Activity added successfully!\n");
    count++;
}

void viewActivities() {
    if (count == 0) {
        printf("\nNo activities recorded yet.\n");
        return;
    }

    printf("\n--- Fitness Activity Log ---\n");
    for (int i = 0; i < count; i++) {
        printf("\nEntry %d:\n", i + 1);
        printf("Date        : %s\n", logData[i].date);
        printf("Type        : %s\n", logData[i].type);
        printf("Duration    : %d min\n", logData[i].duration);
        printf("Steps       : %d\n", logData[i].steps);
        printf("Distance    : %.2f km\n", logData[i].distance);
        printf("Calories    : %.2f kcal\n", logData[i].calories);
    }
}

int main() {
    int choice;

    while (1) {
        printf("\n=== Fitness Tracker Menu ===\n");
        printf("1. Add Activity\n");
        printf("2. View All Activities\n");
        printf("0. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addActivity();
                break;
            case 2:
                viewActivities();
                break;
            case 0:
                printf("Exiting... Goodbye!\n");
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}
```

7. TESTING AND RESULTS

SYNTAX

```
=== Fitness Tracker Menu ===
1. Add Activity
2. View All Activities
0. Exit
Enter choice: 1

Enter date (DD-MM-YYYY): 26-10-2006
Enter activity type (walk/run): walk
Enter duration (minutes): 30
Enter steps: 4000
Enter distance (km): 3.2
Activity added successfully!

=== Fitness Tracker Menu ===
1. Add Activity
2. View All Activities
0. Exit
Enter choice: 1

Enter date (DD-MM-YYYY): 26-10-2006
Enter activity type (walk/run): run
Enter duration (minutes): 20
Enter steps: 3000
Enter distance (km): 2.5
Activity added successfully!

=== Fitness Tracker Menu ===
1. Add Activity
2. View All Activities
0. Exit
Enter choice: 2
```

OUTPUT

```
--- Fitness Activity Log ---

Entry 1:
Date       : 26-10-2006
Type       : walk
Duration    : 30 min
Steps      : 4000
Distance    : 3.20 km
Calories    : 460.00 kcal

Entry 2:
Date       : 26-10-2006
Type       : run
Duration    : 20 min
Steps      : 3000
Distance    : 2.50 km
Calories    : 380.00 kcal
```

Testing

The program was tested with:

- Multiple activity entries (walking and running).
- Edge case when **no activities** are stored (viewing shows correct message).
- Maximum limit testing by adding up to near MAX entries.
- Different durations, step counts, and distances.

The outputs were as expected, and all recorded values were printed correctly.

8. CONCLUSION AND FUTURE SCOPE

The Fitness Activity Tracker successfully meets the objectives of creating a lightweight activity log system. It simplifies the process of recording workouts and automatically evaluates calorie expenditure based on time and effort level. The project demonstrates core C concepts including structures, arrays, modular functions, loops, and formatted output. With minimal resources, it provides a useful tool for tracking personal health progress.

Future Scope:

The system may be enhanced using the following additions:

- **File Storage:** Save and reload logs across sessions.
- **Data Editing:** Update or delete entries (CRUD model).
- **Statistical Analysis:** Total weekly stats, averages, or trends.
- **Categorization:** Additional activity types (cycling, gym, swimming).
- **Graph UI:** Basic visual graphs to show improvement.
- **Mobile or GUI App:** Using web or mobile front-end frameworks.
- **BMI/Heart Rate Integration:** More advanced fitness insights.

9. REFERENCES

Kernighan, B. W., & Ritchie, D. M. The C Programming Language.

Lecture Notes

Standard C Library Documentation: cppreference.com