Uplift - A Daily Wellness Tracker

ABSTRACT

Uplift is a wellness tracking system developed in C. It allows the user to record their mood (on a range of 1-5), sleep hours and amount of water consumed in a day. It also has a feature which allows them to make personal journal entries.

This system also provides an option to check weekly summaries including

- average mood
- average sleep
- total water consumption

This project incorporates core C concepts like file handling, structures, sorting and searching, functions, pointers, memory allocation and arrays.

PROBLEM DEFINITION

Tracking one's daily wellness is often taken lightly or ignored by people today mainly because the do not have a simple and efficient way to record or track their wellbeing. However, without keeping a consistent record, patterns like low mood, poor sleep and dehydration go unnoticed leading to unexplained mental distress.

Most wellbeing apps that I have tried are usually very heavy and distracting beating the main purpose which is *regular usage*.

Therefore the problem is to create a simple, lightweight offline system that allows the user to:

- record their mood (1-5)
- log their daily sleep hours
- track water consumption
- write short journal entries

all of these can be entered date wise and saved permanently in files so they can be retrieved later and analysed.

Uplift solves this problem by providing a clean command line C program that stores each day's information using structured data and file handling. This allows user to

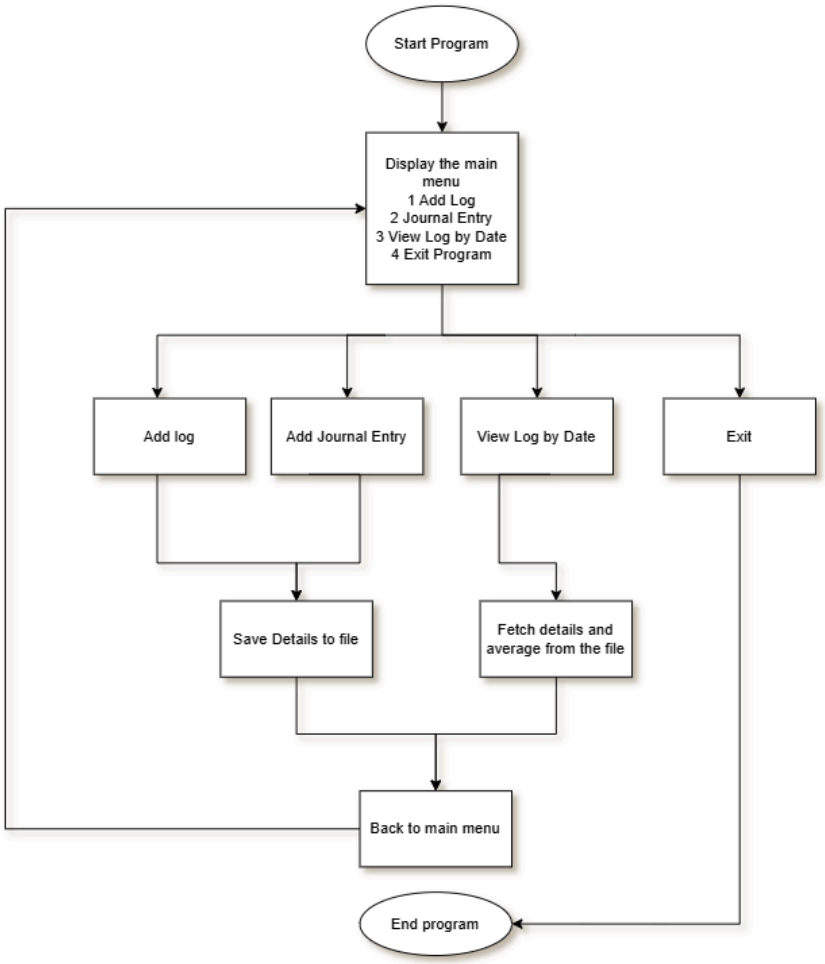
maintain daily logs and observe the patterns in their wellbeing without relying on complicated external applications.

SYSTEM DESIGN

MODULES

- Log module – Handles mood, sleep hours, water intake and stores in “/logs.dat”.
- Journal module – Handles multiline text entries and stores in “/journal.txt”.
- Date-wise retrieval module – Searches logs and entries by date.
- Weekly summary module – takes the start date and checks for the 7 consecutive dates and computes average mood, sleep and water intake.

FLOWCHARTS



ALGORITHMS

Add Daily Log:

```
Enter the date in DD/MM/YYYY format
Check if the date matches the format else show error
Enter mood (1-5)
```

```
Enter sleep hours (float)
Enter water consumed (in glasses)
Open logs.dat in append mode
Create DailyLog structure with all values
Write structure to the file
Close file and confirm save
```

Add Journal entry:

```
Enter the date
Read multiple line input
Stop when user types “END”
Open journal.txt in append mode
Write the date header ##DD/MM/YYYY
Write the journal text
write separator ----
Close file
```

View Logs by Date

```
Enter date
Read all DailyLog entries from logs.dat
Compare date for each entry
If match display the log
If none found print message
```

View Journal by Date

```
Enter the date
Open journal.txt in read mode
Search for header ## DD/MM/YYYY
Print until ---- appears
If not found print “not found”
```

Weekly summary

```
Enter start date
Initialize total counters: sumMood, sumSleep, sumWater, count
For 1-7 days:
    Search for the log of the specific day
    If found
```

```
      add mood to sumMood
      add sleep to sumSleep
      add water to sumWater
      count += 1

    move to next day

If count == 0:

    Print "No data to display" and exit

Find averages:

    avgMood = sumMood/count
    avgSleep = sumMood/count
    avgWater = sumWater/count

Display:

    avgMood
    avgSleep
    avgWater
```

IMPLIMENTATION DETAILS

DATA STRUCTURE

- I used a struct group named `DailyLog` to group all data: date, mood, sleep hours and water consumed.
- This made it easier to pass a day's records in functions

```c
typedef struct
{
    int day;
    int month;
    int year;
    int mood;
    float sleepHrs;
    int waterGlasses;

} DailyLog;
```

ADDING A DAILY LOG

- The program saves daily logs into logs.txt. For this I used file handling .

```c
//adding daily log
int add_daily_log(const char *filename, DailyLog *entry)
{
    FILE *f = fopen(filename, "a");
    if (f == NULL)
    {
        return 0;
    }

    //Writing one log: day month year mood sleep water
    DailyLog e = *entry;

    fprintf(f, "%d %d %d %d %.2f %d\n",
            e.day, e.month, e.year, e.mood, e.sleepHrs,e.waterGlasses);


    fclose(f);
    return 1;
}
```

READING ALL LOGS

- This function read the entire file into a dynamic array. I used fscanf and realloc to extend the memory each time a log is found

```c
DailyLog *read_all_logs(const char *filename, size_t *total_logs)
{
        *total_logs = 0;

        FILE *f = fopen(filename, "r");
        if (f == NULL)
        {
            return NULL;
        }

        DailyLog *arr = NULL;
        DailyLog temp;

        //Reading until EOF
        while (fscanf(f, "%d %d %d %d %f %d",
                      &temp.day, &temp.month, &temp.year,
                      &temp.mood,
                      &temp.sleepHrs,
                      &temp.waterGlasses)==6)
        {
            arr = realloc(arr,(*total_logs + 1) * sizeof(DailyLog));
            arr[*total_logs] = temp;
            (*total_logs)++;
        }

        fclose(f);
        return arr;
```

DATE MATCHING AND SEARCHING LOGS BY DATE

- A function for checking date format to see if the user entered the right format and if the entered date matches an existing log

```c
DailyLog *find_logs_by_date (const char *filename, int d, int m, int y, size_t *total_logs)
{
    *total_logs = 0;
    size_t total = 0;

    DailyLog *all = read_all_logs(filename, &total);
    if (all == 0)
    {
        return NULL;
    }

    DailyLog *result = NULL;

    for (size_t i = 0; i<total; i++)
    {
        if (date_equal(&all[i], d,m,y))
        {
            result = realloc(result, (*total_logs + 1)* sizeof(DailyLog));
            result[*total_logs] = all[i];
            (*total_logs)++ ;
        }
    }

    free (all);
    return result;


}


int date_equal(const DailyLog *a, int d, int m, int y)
{
    DailyLog date = *a;
    return (date.day == d && date.month == m && date.year == y);
}
```

READING A LINE FOR JOURNAL ENTRIES

- I created a custom read_line function with a static buffer for simple and easy line by line reading of journal.txt

```c
char *read_line(FILE *f)
{
    static char buffer[600];

    if (fgets(buffer, sizeof(buffer), f) == NULL)
        return NULL;

    /*
    to remove newline. if \n is encountered it will
    replace with end of string marker
    */
    for (int i = 0; buffer[i] != '\0'; i++) {
    if (buffer[i] == '\n') {
        buffer[i] = '\0';
        break;
    }
}

    return buffer;
}
```

ADDING JOURNAL ENTRIES AND VIEWING

- I created a specific format for each entry for ease of reading the loading the files for the output. When the file is read to fetch the journal entry if a specific date the program finds the matching header (## DD/MM/YYYY) and prints the contents till the separator is encountered (----).

```c
int add_journal_entry(const char *filename, int d, int m, int y, const char *text)
{
    FILE *f = fopen(filename, "a");
    if (f == 0)
    {
        return 0;
    }

    //structure of the journal entry
    fprintf(f, "## %02d/%02d/%04d\n", d, m, y); //date DD/MM/YYYY
    fprintf(f, "%s\n", text );  //entry
    fprintf(f,"----\n"); //seperator four dashes

    fclose(f);
    return 1;


}
```

```c
while ((line = read_line(f)) !=NULL)
{
    if (strcmp(line, header) ==0)
    {
        printf("\nJournal entry written on %02d/%02d/%04d:\n", d,m,y);
        flag= 1;

        while ((line = read_line(f)) != NULL)
        {
            if (strncmp(line, "----",4)== 0)
            {
                break;
            }
            printf("%s\n", line);

        }

        break;
    }
```

WEEKLY SUMMARY CALCULATION

- This function finds logs for 7 consecutive days and computes average. to increment the date i used a leap year aware format, where the “day” , “month” and starts from 1 and as the day increases and reaches the last day of the month, the day resets to 1 and month increments by 1. The number of days in a month is stored in an array. The year entered by the user is also checked for leap year.

- Hence the first day for the 7 day summary is the date entered by the user and the dates increment and checks for the 7 consecutive days using the above mentioned strategy.

```c
static int is_leap(int y)
{
    return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);

}

//to automatically count the seven days for weekly summary
static void next_day (int *d, int *m, int *y)
{
    int daysOfMonth[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};

    if (is_leap(*y)) daysOfMonth[2] = 29;
    (*d)++;

    if (*d >daysOfMonth[*m])
    {
        *d = 1;
        (*m)++;
        if (*m >12)
        {
            *m =1;
            (*y)++;

        }
    }
}
```

MEMORY MANAGEMENT

- I have used multiple dynamically allocated arrays and used proper memory management and freed them after use.

TESTING AND VERIFICATION

Testing and Verification ensures that the program works as expected, follows the requirements, and produces correct results.

I followed the following testing methods for each segment of code

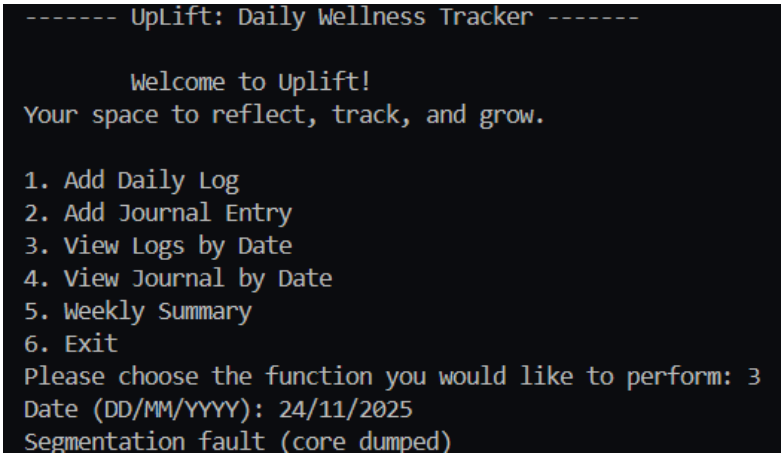
`add_daily_log()`

I added 2 logs with different dates to check if it gets saved accurately in the .txt file. There was no issue in this segment during debugging and testing.

`find_logs_by_date()`

I tried to fetch the logs by date and encountered a segmentation fault. Through debugging i traced the error to incorrect handling of total_logs pointer inside read_all_logs() function. The pointer was being overwritten instead of being

dereferenced which caused invalid memory access when the function was called. The function worked perfectly after that correction.

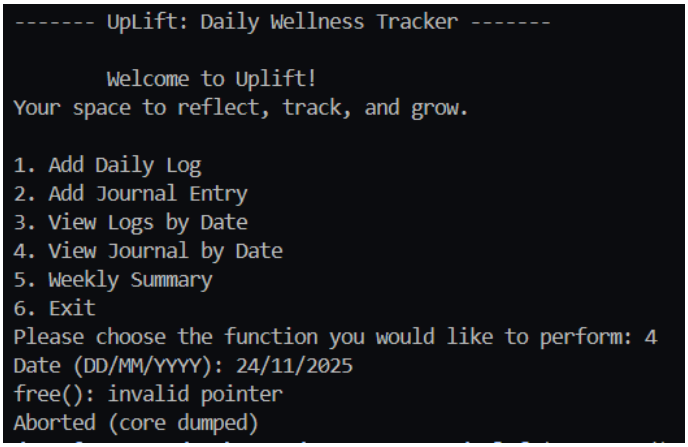


`add_journal_entry()`

I tested the `add_journal_entry()` function by adding entries for two separate days. The program worked correctly each time and showed the success message.

`view_journal_by_date()`

While testing the `view_journal_by_date()` function, I initially encountered an error where the program displayed “No journal entry found” even though an entry existed for the specified date. On reviewing the code, I identified that the month and year parameters had been mistakenly interchanged in `add_journal_entry()` function call. After correcting this, the function successfully retrieved and displayed the appropriate journal entries.



`weekly_summary()`

I tested the `weekly_summary()` function after adding several days of logs. It worked smoothly and correctly calculated the weekly averages without any issues.

CONCLUSION

Working on Uplift helped me understand how a simple wellness tracker can be built by combining the fundamental concepts of C in a practical way. Throughout the project, I used structures, file handling, functions, and dynamic memory allocation to build features that feel meaningful in real use logging daily wellbeing, writing journal entries, viewing past data, and generating weekly summaries.

During testing, I also ran into small mistakes that taught me how important careful debugging is. Fixing pointer issues, correcting parameter mismatches, and validating file reads helped me understand what was actually happening when the code is run. After correction each module worked properly and gave the desired output.

In the end UpLift turned into exactly what i intended, a lightweight, distraction free wellness tracker that is easy to use. This project helped me incorporate the concepts of C and understand them better.