UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

EKTA BHARDWAJ

590022698

BATCH :- 20

SCHOOL OF COMPUTER SCIENCE

B.TECH (COMPUTER SCIENCE AND ENGINEERING)

PROJECT :- LIFE ORGANIZING SYSTEM

SUBMITTED TO :

MR. RAHUL PRASAD

INTRODUCTION

In today's busy lifestyle, people have to manage many small but important things every day—such as groceries, expenses, health activities and personal entertainment. However, most of the time this information is not recorded properly and gets scattered. As a result, people forget expiry dates of pantry items, overspend without noticing, skip important health routines or lose track of what they watched.

To solve these problems, a **Life Organizing System** is needed.

The Life Organizing System is a simple console-based application developed in C language. It allows users to organize different aspects of their daily life in one place. The system manages:

- Pantry items and expiry dates
- Daily expenses and monthly budget
- Health habits such as water, calories, sleep and steps
- Movies watched with their ratings

The main purpose of this system is to make daily life more organized, easier to track and more productive. Instead of writing everything on paper or remembering it manually, this system stores all information safely in text files. The user can add, view, edit or delete any data whenever needed.

This project also demonstrates core concepts of the C programming language such as **structures**, **file handling**, **functions**, and **modular programming**. It acts as a practical example of how C can be used not only for learning but also for building useful real-life applications.

ABSTRACT

This project is a “Life Organizing System” developed in C language to help users manage their daily life data in one place. The system mainly handles four types of information: pantry items, daily expenses, health records and movies. For each part, a separate structure (struct) is used and the data is stored in arrays and saved permanently in text files using file handling. The user interacts with the program through a simple menu-driven interface, where they can add, view, edit and delete records. An alert centre is also provided which gives useful warnings, such as low pantry stock, high total expenses, poor health habits (low water, sleep, steps) and low-rated movies. The project demonstrates the use of structures, arrays, functions, file handling and basic date handling in C. Overall, this mini project shows how C language can be used to build a small but practical console-based application for personal life management.

PROBLEM DEFINITION

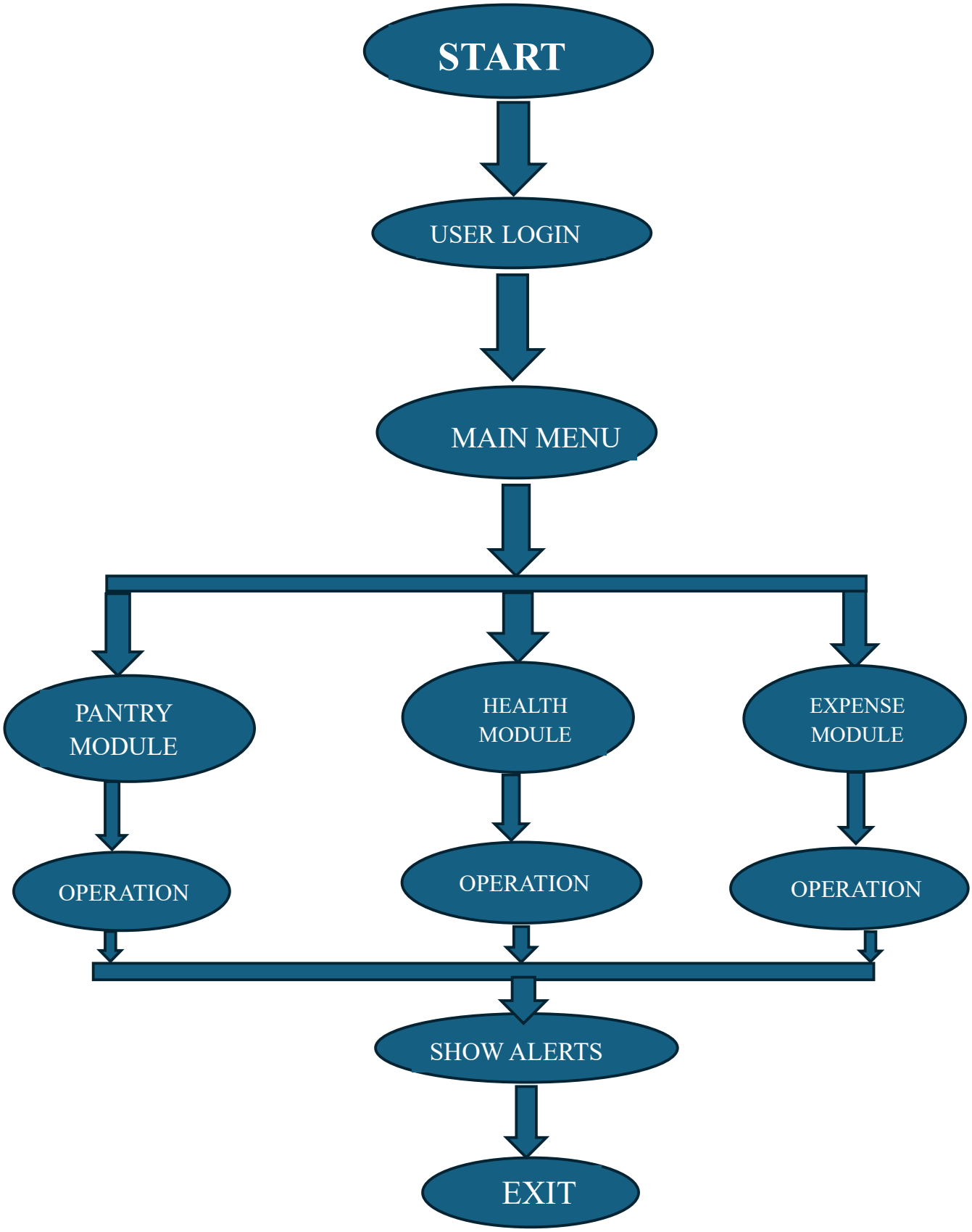
Managing daily life information such as pantry stocks, expenses, health activities and entertainment choices is often difficult because the data is scattered and not recorded properly. People forget expiry dates of food items, lose track of their spending, do not monitor their health habits and cannot maintain a list of movies they have watched. There is no single system that keeps all these details together in an easy and organized way.

The problem is to design a simple, console-based system that can:

- Store different types of personal data (pantry, expenses, health, movies)
- Allow the user to add, view, update and delete records easily
- Save all information permanently using text files
- Generate alerts such as low pantry stock, near-expiry items, high expenses, bad health habits and low-rated movies
- Help the user manage daily life in a more structured and organized manner

This project aims to solve the above problems by creating a **Life Organizing System** using C language, structures and file handling.

FLOWCHART


START
USER LOGIN
MAIN MENU
PANTRY MODULE
HEALTH MODULE
EXPENSE MODULE
OPERATION
OPERATION
OPERATION
SHOW ALERTS
EXIT

ALGORITHM

Main Menu Algorithm:
Step 1: Start
Step 2: Display main menu options
Step 3: Take input choice from the user
Step 4: Use switch-case to call respective module
Step 5: Perform selected operation
Step 6: Return to main menu until exit option is selected
Step 7: Stop

IMPLEMENTATION

The Life Organizing System is fully developed in C programming. It uses structures, arrays, loops, functions, switch-case statements, and file handling to store user data securely. Some key highlights of implementation include modular programming, reusable components, user-friendly menus, and formatted output.

CODE :-

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

#define MAX 200
#define NAME_LEN 128
#define CAT_LEN 40
#define DATE_LEN 20
#define BUF 512
```

```
const char *PANTRYFILE = "pantry.txt";
const char *EXPFILE    = "expenses.txt";
const char *HEALTHFILE = "health.txt";
const char *MOVIEFILE  = "movies.txt";

typedef struct {
    char name[NAME_LEN];
    char cat[CAT_LEN];
    int qty;
    char expiry[DATE_LEN];
} Pantry;

typedef struct {
    char name[NAME_LEN];
    char cat[CAT_LEN];
    float amount;
    char date[DATE_LEN];
} Expense;

typedef struct {
    char date[DATE_LEN];
    float water;
    int cal;
    float sleep;
    int steps;
} Health;

typedef struct {
    char title[NAME_LEN];
    char genre[CAT_LEN];
    float rating;
} Movie;
```

```
Pantry pantryList[MAX];
int pantryCount = 0;
Expense expList[MAX];
int expCount = 0;
Health healthList[MAX];
int healthCount = 0;
Movie movieList[MAX];
int movieCount = 0;
float userBudget = 10000.0f;

void trim_newline(char *s)
{
    if (!s) return;
    size_t L = strlen(s);
    if (L>0 && s[L-1] == '\n') s[L-1] = '\0';
}
void readLine(char *buf, int size)
{
    if (fgets(buf, size, stdin) == NULL) {
        buf[0] = '\0';
        return;
    }


    int len = (int)strlen(buf);
    if (len > 0 && buf[len - 1] == '\n') {
        buf[len - 1] = '\0';
    }
}

int stringToInt(const char *s, int *out)
{
    if (s == NULL || s[0] == '\0') return 0;
    int value;
```

```
    if (sscanf(s, "%d", &value) != 1) return 0;
    *out = value;
    return 1;
}

int stringToFloat(const char *s, float *out)
{
    if (s == NULL || s[0] == '\0') return 0;
    float value;
    if (sscanf(s, "%f", &value) != 1) return 0;
    *out = value;
    return 1;
}

void today_str(char *buf, int n)
{
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    snprintf(buf, n, "%02d-%02d-%04d", tm.tm_mday, tm.tm_mon+1, tm.tm_year+1900);
}


long date_to_days(const char *s)
{
    if (!s || !*s) return 0;
    int d=0,m=0,y=0;
    if (sscanf(s, "%d-%d-%d", &d, &m, &y) != 3) {
        sscanf(s, "%d/%d/%d", &d, &m, &y);
    }
    struct tm tmv;
    memset(&tmv, 0, sizeof(tmv));
    tmv.tm_mday = d;
    tmv.tm_mon = m - 1;
    tmv.tm_year = y - 1900;
```

```
    tmv.tm_hour = 12;
    tmv.tm_min = tmv.tm_sec = 0;
    tmv.tm_isdst = -1;
    time_t t = mktime(&tmv);
    if (t == (time_t)-1) return 0;
    return (long)(t / (24*60*60));
}
long today_days()
{
    time_t t = time(NULL);
    return (long)(t / (24*60*60));
}

void loadPantry() {
    pantryCount = 0;
    FILE *f = fopen(PANTRYFILE, "r");
    if (!f)
return;
    char line[BUF];
    while (fgets(line, sizeof(line), f) && pantryCount < MAX) {
        trim_newline(line);
        if (line[0] == '\0') continue;
        char tmp[BUF];
        strncpy(tmp, line, sizeof(tmp)-1);
        tmp[sizeof(tmp)-1]=0;
        char *tok = strtok(tmp, ",");
        if (!tok) continue;
        strncpy(pantryList[pantryCount].name, tok, NAME_LEN-1);
        pantryList[pantryCount].name[NAME_LEN-1] = '\0';
        tok = strtok(NULL, ",");
        if (tok)
        strncpy(pantryList[pantryCount].cat, tok, CAT_LEN-1);
        else
       pantryList[pantryCount].cat[0]=0;
```

```
        tok = strtok(NULL, ",");
        if (tok)
pantryList[pantryCount].qty = atoi(tok);
        else
       pantryList[pantryCount].qty = 0;
        tok = strtok(NULL, ",");
        if (tok)
        strncpy(pantryList[pantryCount].expiry, tok, DATE_LEN-1);
        else
       pantryList[pantryCount].expiry[0]=0;
        pantryList[pantryCount].expiry[DATE_LEN-1] = '\0';
        pantryCount++;
    }
    fclose(f);
}
void savePantry() {
    FILE *f = fopen(PANTRYFILE, "w");
    if (!f) {
     printf(" Unable to save pantry file.\n"); return; }
    for (int i=0;i<pantryCount;i++) {

fprintf(f,"%s,%s,%d,%s\n",pantryList[i].name,pantryList[i].cat,pantryList[i].qty,pantryList[i].expiry[0
] ? pantryList[i].expiry : "");
    }
    fclose(f);
}



void loadExpenses() {
    expCount = 0;
    FILE *f = fopen(EXPFILE, "r");
    if (!f) return;
    char line[BUF];
    while (fgets(line, sizeof(line), f) && expCount < MAX) {
```

```
        trim_newline(line);
        if (line[0] == '\0') continue;
        char tmp[BUF];
        strncpy(tmp, line, sizeof(tmp)-1);
        tmp[sizeof(tmp)-1]=0;
        char *tok = strtok(tmp, ",");
        if (!tok) continue;
        strncpy(expList[expCount].name, tok, NAME_LEN-1);
        expList[expCount].name[NAME_LEN-1] = '\0';
        tok = strtok(NULL, ",");
        if (tok) strncpy(expList[expCount].cat, tok, CAT_LEN-1);
        else expList[expCount].cat[0]=0;
        tok = strtok(NULL, ",");
        if (tok) expList[expCount].amount = atof(tok);
        else expList[expCount].amount = 0.0f;
        tok = strtok(NULL, ",");
        if (tok) strncpy(expList[expCount].date, tok, DATE_LEN-1);
        else expList[expCount].date[0]=0;
        expList[expCount].date[DATE_LEN-1] = '\0';
        expCount++;
    }
    fclose(f);
}
void saveExpenses() {
    FILE *f = fopen(EXPFILE, "w");
    if (!f) { printf(" Unable to save expenses file.\n"); return; }
    for (int i=0;i<expCount;i++) {
        fprintf(f, "%s,%s,%.2f,%s\n",  expList[i].name, expList[i].cat,expList[i].amount,
expList[i].date[0] ? expList[i].date : "");
    }
    fclose(f);
}
```

```
void loadHealth() {
    healthCount = 0;
    FILE *f = fopen(HEALTHFILE, "r");
    if (!f) return;
    char line[BUF];
    while (fgets(line, sizeof(line), f) && healthCount < MAX) {
        trim_newline(line);
        if (line[0] == '\0') continue;
        char tmp[BUF];
        strncpy(tmp, line, sizeof(tmp)-1);
        tmp[sizeof(tmp)-1]=0;
        char *tok = strtok(tmp, ",");
        if (!tok) continue;
        strncpy(healthList[healthCount].date, tok, DATE_LEN-1);
        healthList[healthCount].date[DATE_LEN-1] = '\0';
        tok = strtok(NULL, ",");
        if (tok) healthList[healthCount].water = atof(tok);
        else healthList[healthCount].water = 0.0f;
        tok = strtok(NULL, ",");
        if (tok) healthList[healthCount].cal = atoi(tok);
        else healthList[healthCount].cal = 0;
        tok = strtok(NULL, ",");
        if (tok) healthList[healthCount].sleep = atof(tok);
        else healthList[healthCount].sleep = 0.0f;
        tok = strtok(NULL, ",");
        if (tok) healthList[healthCount].steps = atoi(tok);
        else healthList[healthCount].steps = 0;
        healthCount++;
    }
    fclose(f);
}
void saveHealth() {
    FILE *f = fopen(HEALTHFILE, "w");
    if (!f) {
```

```
printf(" Unable to save health file.\n"); return; }
    for (int i=0;i<healthCount;i++) {
        fprintf(f, "%s,%.2f,%d,%.2f,%d\n", healthList[i].date,healthList[i].water,
healthList[i].cal,healthList[i].sleep,healthList[i].steps);
    }
    fclose(f);
}



void loadMovies() {
    movieCount = 0;
    FILE *f = fopen(MOVIEFILE, "r");
    if (!f) return;
    char line[BUF];
    while (fgets(line, sizeof(line), f) && movieCount < MAX) {
        trim_newline(line);
        if (line[0] == '\0') continue;
        char tmp[BUF];
        strncpy(tmp, line, sizeof(tmp)-1);
        tmp[sizeof(tmp)-1]=0;
        char *tok = strtok(tmp, ",");
        if (!tok) continue;
        strncpy(movieList[movieCount].title, tok, NAME_LEN-1);
        movieList[movieCount].title[NAME_LEN-1] = '\0';
        tok = strtok(NULL, ",");
        if (tok) strncpy(movieList[movieCount].genre, tok, CAT_LEN-1);
        else movieList[movieCount].genre[0]=0;
        tok = strtok(NULL, ",");
        if (tok) movieList[movieCount].rating = atof(tok);
        else movieList[movieCount].rating = 0.0f;
        movieCount++;
    }
    fclose(f);
}
```

```
void saveMovies() {
    FILE *f = fopen(MOVIEFILE, "w");
    if (!f) { printf(" Unable to save movies file.\n"); return; }
    for (int i=0;i<movieCount;i++) {
        fprintf(f, "%s,%s,%.1f\n", movieList[i].title,movieList[i].genre,movieList[i].rating);
    }
    fclose(f);
}



void pantryAlerts() {
    int any = 0;
    for (int i=0;i<pantryCount;i++) {
        if (pantryList[i].qty <= 2) {
            printf("   Low stock: %s (qty=%d)\n", pantryList[i].name, pantryList[i].qty);
            any = 1;
        }
        if (pantryList[i].expiry[0]) {
            long dexp = date_to_days(pantryList[i].expiry);
            long td = today_days();
            long diff = dexp - td;
            if (diff < 0) {
                printf("   Expired: %s (expired %ld days ago on %s)\n", pantryList[i].name, -diff,
pantryList[i].expiry);
                any = 1;
            } else if (diff <= 5) {
                printf("  Near expiry: %s (in %ld day(s) on %s)\n", pantryList[i].name, diff,
pantryList[i].expiry);
                any = 1;
            }
        }
    }
    if (!any) printf("   No pantry alerts.\n");
}
```

```
void expenseAlerts()
{
    float total = 0.0f;
    int i;

    if (expCount == 0) {
        printf("  No expenses recorded yet.\n");
        return;
    }

    for (i = 0; i < expCount; i++) {
        total += expList[i].amount;
    }

    printf("  Total expenses so far: %.2f (Budget: %.2f)\n", total, userBudget);

    if (total > userBudget) {
        printf("  Alert: You have crossed your monthly budget.\n");
        printf("  Extra amount spent: %.2f\n", total - userBudget);
    } else if (total >= 0.8f * userBudget) {
        printf("  Warning: You have already used more than 80%% of your budget.\n");
    } else {
        printf("  Expenses are currently under control.\n");
    }

    printf("  Checking for big expenses (>= 1000)...\n");
    int anyBig = 0;
    for (i = 0; i < expCount; i++) {
        if (expList[i].amount >= 1000.0f) {
            printf("    Big expense: %s (%.2f) on %s\n",
                   expList[i].name,
                   expList[i].amount,
                   expList[i].date);
```

```
            anyBig = 1;
        }
    }
    if (!anyBig) {
        printf("  No big individual expenses found.\n");
    }
}




void healthAlerts() {
    int any = 0;
    for (int i=0;i<healthCount;i++) {
        if (healthList[i].water < 2.0f) { printf("  Low water on %s: %.2f L\n", healthList[i].date,
healthList[i].water); any = 1; }
        if (healthList[i].sleep < 6.0f) { printf("  Low sleep on %s: %.2f hrs\n", healthList[i].date,
healthList[i].sleep); any = 1; }
        if (healthList[i].steps < 5000) { printf("  Low steps on %s: %d steps\n", healthList[i].date,
healthList[i].steps); any = 1; }
    }
    if (!any) printf("   No health alerts.\n");
}


void movieAlerts() {
    int any = 0;
    if (movieCount == 0) { printf("   No movies in list.\n"); return; }
    for (int i=0;i<movieCount;i++) {
        if (movieList[i].rating < 5.0f) {
            printf("  Low-rated movie: %s (%.1f)\n", movieList[i].title, movieList[i].rating);
            any = 1;
        }
    }
    if (!any) printf("  No movie alerts.\n");
}
```

```
void alertCenter() {
    printf("\n--- ALERT CENTER ---\n");
    printf("Pantry Alerts:\n"); pantryAlerts();
    printf("Expense Alerts:\n"); expenseAlerts();
    printf("Health Alerts:\n"); healthAlerts();
    printf("Movie Alerts:\n"); movieAlerts();
    printf("--------------------\n");
}


void pantryAdd()
{
    char buffer[BUF];
    int quantity;

    if (pantryCount >= MAX) {
        printf("Pantry list is full. Cannot add more items.\n");
        return;
    }

    printf("Enter item name: ");
    readLine(buffer, sizeof(buffer));
    if (buffer[0] == '\0') {
        printf("Item name cannot be empty.\n");
        return;
    }
    strncpy(pantryList[pantryCount].name, buffer, NAME_LEN - 1);
    pantryList[pantryCount].name[NAME_LEN - 1] = '\0';

    printf("Enter category: ");
    readLine(pantryList[pantryCount].cat, CAT_LEN);
    pantryList[pantryCount].cat[CAT_LEN - 1] = '\0';

    printf("Enter quantity (number): ");
```

```
    readLine(buffer, sizeof(buffer));
    if (!stringToInt(buffer, &quantity) || quantity < 0) {
        printf("Invalid quantity. Item not added.\n");
        return;
    }
    pantryList[pantryCount].qty = quantity;

    printf("Enter expiry date (DD-MM-YYYY), or leave blank: ");
    readLine(pantryList[pantryCount].expiry, DATE_LEN);
    pantryList[pantryCount].expiry[DATE_LEN - 1] = '\0';

    pantryCount++;

    savePantry();
    printf("Pantry item added and file updated.\n");
}

void pantryListShow() {
    if (pantryCount == 0) { printf("No pantry items.\n"); return; }
    printf("\n%-4s %-25s %-15s %-12s %-12s\n",
       "No", "Name", "Category", "Amount", "Date");
printf("-----------------------------------------------------------------\n");

    for (int i=0;i<pantryCount;i++) {
        printf("%-3d %-30s %-14s %-4d %s\n", i+1, pantryList[i].name, pantryList[i].cat,
pantryList[i].qty, pantryList[i].expiry);
    }
    printf("\n-- Pantry Alerts --\n"); pantryAlerts();
}

void pantryEdit() {
    pantryListShow();
    if (pantryCount == 0)
    return;
```

```
    char buf[BUF];
    printf("Enter item number to edit: ");
    readLine(buf, sizeof(buf));
    int idx; if (!stringToInt(buf, &idx) || idx < 1 || idx > pantryCount) {
        printf("Invalid number.\n");
        return; }
    idx--;
    printf("New name (enter to keep '%s'): ", pantryList[idx].name);
    readLine(buf, sizeof(buf));
    if (buf[0]) strncpy(pantryList[idx].name, buf, NAME_LEN-1);
    printf("New category (enter to keep '%s'): ", pantryList[idx].cat);
    readLine(buf, sizeof(buf));
    if (buf[0]) strncpy(pantryList[idx].cat, buf, CAT_LEN-1);
    printf("New quantity (enter to keep %d): ", pantryList[idx].qty); readLine(buf, sizeof(buf));
    int q;
    if (buf[0]) { if (!stringToInt(buf, &q) || q < 0) printf("Invalid ignored.\n");
    else pantryList[idx].qty = q; }
    printf("New expiry (enter to keep '%s'): ", pantryList[idx].expiry); readLine(buf, sizeof(buf)); if
(buf[0])
    strncpy(pantryList[idx].expiry, buf, DATE_LEN-1);
    savePantry();
    printf("Pantry updated and saved.\n");
    pantryAlerts();
}

void pantryDelete() {
    pantryListShow(); if (pantryCount == 0) return;
    char buf[BUF]; printf("Enter item number to delete: "); readLine(buf, sizeof(buf));
    int idx; if (!stringToInt(buf, &idx) || idx < 1 || idx > pantryCount) { printf("Invalid number.\n");
return; }
    idx--;
    for (int i=idx;i<pantryCount-1;i++) pantryList[i] = pantryList[i+1];
    pantryCount--;
    savePantry();
```

```
    printf("Pantry item deleted and saved.\n");
}


void expenseAdd() {
    if (expCount >= MAX) { printf("Expenses full.\n"); return; }
    char buf[BUF];
    printf("Enter expense name: "); readLine(expList[expCount].name, NAME_LEN);
    if (expList[expCount].name[0] == '\0') { printf("Name required.\n"); return; }
    printf("Enter category: "); readLine(expList[expCount].cat, CAT_LEN);
    printf("Enter amount: "); readLine(buf, sizeof(buf)); float amt;
    if (!stringToFloat(buf, &amt) || amt < 0.0f) { printf("Invalid amount.\n"); return; }
    expList[expCount].amount = amt;
    today_str(expList[expCount].date, DATE_LEN);
    expCount++;
    saveExpenses();
    printf("Expense added and saved.\n");
    expenseAlerts();
}

void expenseListShow() {
    if (expCount == 0) {
        printf("No expenses.\n");
        return;
    }

    printf("\n%-4s %-15s %-15s %-10s %-12s\n",
           "No", "Name", "Category", "Amount", "Date");

    printf("-----------------------------------------------------------------\n");


    for (int i = 0; i < expCount; i++) {
        char amtbuf[20];
```

```
        sprintf(amtbuf, "%.2f", expList[i].amount);

        printf("%-4d %-15s %-15s %-10s %-12s\n",
               i + 1,
               expList[i].name,
               expList[i].cat,
               amtbuf,
               expList[i].date);
    }


    float total = 0.0f;
    for (int i = 0; i < expCount; i++) {
        total += expList[i].amount;
    }

    printf("\nTotal expenses: ₹%.2f\n", total);


    expenseAlerts();
}
void expenseEdit()
{
    char buf[BUF];
    int index;
    float newAmount;
    expenseListShow();
    if (expCount == 0) {
        return;
    }

    printf("Enter the expense number you want to edit: ");
    readLine(buf, sizeof(buf));
    if (!stringToInt(buf, &index) || index < 1 || index > expCount) {
```

```
        printf("Invalid expense number.\n");
        return;
    }
    index--;
    printf("Enter new name (press Enter to keep '%s'): ",
           expList[index].name);
    readLine(buf, sizeof(buf));
    if (buf[0] != '\0') {
        strncpy(expList[index].name, buf, NAME_LEN - 1);
        expList[index].name[NAME_LEN - 1] = '\0';
    }
    printf("Enter new category (press Enter to keep '%s'): ",
           expList[index].cat);
    readLine(buf, sizeof(buf));
    if (buf[0] != '\0') {
        strncpy(expList[index].cat, buf, CAT_LEN - 1);
        expList[index].cat[CAT_LEN - 1] = '\0';
    }
    printf("Enter new amount (press Enter to keep %.2f): ",
           expList[index].amount);
    readLine(buf, sizeof(buf));
    if (buf[0] != '\0') {
        if (stringToFloat(buf, &newAmount) && newAmount >= 0) {
            expList[index].amount = newAmount;
        } else {
            printf("Invalid amount. Old value kept.\n");
        }
    }
    saveExpenses();
    printf("Expense updated and file saved.\n");
}
```

```
void expenseDelete() {
    expenseListShow(); if (expCount == 0) return;
    char buf[BUF]; printf("Enter expense number to delete: "); readLine(buf, sizeof(buf));
    int idx; if (!stringToInt(buf, &idx) || idx < 1 || idx > expCount) { printf("Invalid number.\n"); return;
}
    idx--;
    for (int i=idx;i<expCount-1;i++)
    expList[i] = expList[i+1];
    expCount--;
    saveExpenses();
    printf("Expense deleted and saved.\n");
}



void healthAdd() {
    if (healthCount >= MAX) { printf("Health log full.\n"); return; }
    char buf[BUF];
    today_str(healthList[healthCount].date, DATE_LEN);
    printf("Date: %s\n", healthList[healthCount].date);
    printf("Enter water intake (liters): ");
    readLine(buf, sizeof(buf));
    float w;
    if (!stringToFloat(buf, &w) || w < 0.0f) {
       printf("Invalid.\n"); return; }
    healthList[healthCount].water = w;
    printf("Enter calories: ");
    readLine(buf, sizeof(buf)); int c;
    if (!stringToInt(buf, &c) || c < 0) { printf("Invalid.\n"); return; }
    healthList[healthCount].cal = c;
    printf("Enter sleep hours (e.g. 7.5): "); readLine(buf, sizeof(buf)); float s;
    if (!stringToFloat(buf, &s) || s < 0.0f) { printf("Invalid.\n"); return; }
    healthList[healthCount].sleep = s;
    printf("Enter steps: "); readLine(buf, sizeof(buf)); int st;
```

```
    if (!stringToInt(buf, &st) || st < 0) { printf("Invalid.\n"); return; }
    healthList[healthCount].steps = st;
    healthCount++;
    saveHealth();
    printf("Health record added and saved.\n");
    healthAlerts();
}

void healthListShow() {
    if (healthCount == 0) { printf("No health records.\n"); return; }
    printf("\nNo  Date       Water(L)  Calories  Sleep(h)  Steps\n");
    printf("-------------------------------------------------\n");
    for (int i=0;i<healthCount;i++) {
        printf("%-3d %-11s %-9.2f %-9d %-9.2f %-6d\n", i+1, healthList[i].date, healthList[i].water,
healthList[i].cal, healthList[i].sleep, healthList[i].steps);
    }
    printf("\n-- Health Alerts --\n"); healthAlerts();
}

void healthEdit()
{
    char buf[BUF];
    int index;
    float fvalue;
    int ivalue;
    healthListShow();
    if (healthCount == 0) {
        return;
    }

    printf("Enter the health record number you want to edit: ");
    readLine(buf, sizeof(buf));
    if (!stringToInt(buf, &index) || index < 1 || index > healthCount) {
        printf("Invalid record number.\n");
```

```
    return;
}

index--;
printf("New water intake in liters (press Enter to keep %.2f): ",
       healthList[index].water);
readLine(buf, sizeof(buf));
if (buf[0] != '\0') {
    if (stringToFloat(buf, &fvalue) && fvalue >= 0) {
        healthList[index].water = fvalue;
    } else {
        printf("Invalid input. Old water value kept.\n");
    }
}
printf("New calories (press Enter to keep %d): ",
       healthList[index].cal);
readLine(buf, sizeof(buf));
if (buf[0] != '\0') {
    if (stringToInt(buf, &ivalue) && ivalue >= 0) {
        healthList[index].cal = ivalue;
    } else {
        printf("Invalid input. Old calorie value kept.\n");
    }
}
printf("New sleep hours (press Enter to keep %.2f): ",
       healthList[index].sleep);
readLine(buf, sizeof(buf));
if (buf[0] != '\0') {
    if (stringToFloat(buf, &fvalue) && fvalue >= 0) {
        healthList[index].sleep = fvalue;
    } else {
        printf("Invalid input. Old sleep value kept.\n");
    }
}
```

```
    printf("New steps (press Enter to keep %d): ",
           healthList[index].steps);
    readLine(buf, sizeof(buf));
    if (buf[0] != '\0') {
        if (stringToInt(buf, &ivalue) && ivalue >= 0) {
            healthList[index].steps = ivalue;
        } else {
            printf("Invalid input. Old steps kept.\n");
        }
    }
    saveHealth();
    printf("Health record updated and saved.\n");
}


void healthDelete() {
    healthListShow(); if (healthCount == 0) return;
    char buf[BUF];
    printf("Enter record number to delete: "); readLine(buf, sizeof(buf));
    int idx;
    if (!stringToInt(buf, &idx) || idx < 1 || idx > healthCount) { printf("Invalid number.\n"); return; }
    idx--;
    for (int i=idx;i<healthCount-1;i++)
    healthList[i] = healthList[i+1];
    healthCount--;
    saveHealth();
    printf("Health record deleted and saved.\n");
}


void movieAdd() {
    if (movieCount >= MAX) { printf("Movie list full.\n"); return; }
    char buf[BUF];
    printf("Enter movie name: "); readLine(movieList[movieCount].title, NAME_LEN);
```

```
    if (movieList[movieCount].title[0] == '\0') { printf("Name required.\n"); return; }
    printf("Enter genre: "); readLine(movieList[movieCount].genre, CAT_LEN);
    printf("Enter rating (0-10): "); readLine(buf, sizeof(buf)); float r;
    if (!stringToFloat(buf, &r) || r < 0.0f || r > 10.0f) { printf("Invalid rating.\n"); return; }
    movieList[movieCount].rating = r;
    movieCount++;
    saveMovies();
    printf("Movie added and saved.\n");
    movieAlerts();
}

void movieListShow() {
    if (movieCount == 0) {
        printf("No movies added.\n");
        return;
    }

    for (int i = 0; i < movieCount; i++) {
        if (movieList[i].title[0] == '\0') continue;
        printf("%d. %s | %s | %.1f\n",
            i+1,
            movieList[i].title,
            movieList[i].genre,
            movieList[i].rating
        );
    }
}


void movieEdit()
{
    char buf[BUF];
    int index;
    float newRating;
```

```
movieListShow();
if (movieCount == 0) {
    return;
}

printf("Enter the movie number you want to edit: ");
readLine(buf, sizeof(buf));

if (!stringToInt(buf, &index) || index < 1 || index > movieCount) {
    printf("Invalid movie number.\n");
    return;
}
index--;
printf("Enter new title (press Enter to keep '%s'): ",
       movieList[index].title);
readLine(buf, sizeof(buf));
if (buf[0] != '\0') {
    strncpy(movieList[index].title, buf, NAME_LEN - 1);
    movieList[index].title[NAME_LEN - 1] = '\0';
}
printf("Enter new genre (press Enter to keep '%s'): ",
       movieList[index].genre);
readLine(buf, sizeof(buf));
if (buf[0] != '\0') {
    strncpy(movieList[index].genre, buf, CAT_LEN - 1);
    movieList[index].genre[CAT_LEN - 1] = '\0';
}
printf("Enter new rating (0 to 10, press Enter to keep %.1f): ",
       movieList[index].rating);
readLine(buf, sizeof(buf));

if (buf[0] != '\0') {
    if (stringToFloat(buf, &newRating) &&
        newRating >= 0.0f && newRating <= 10.0f) {
```

```
            movieList[index].rating = newRating;
        } else {
            printf("Invalid rating. Old rating is kept.\n");
        }
}
    saveMovies();
    printf("Movie details updated and file saved.\n");
}

void movieDelete()
{
    char buf[BUF];
    int index;
    movieListShow();
    if (movieCount == 0) {
        return;
    }

    printf("Enter the movie number you want to delete: ");
    readLine(buf, sizeof(buf));
    if (!stringToInt(buf, &index) || index < 1 || index > movieCount) {
        printf("Invalid movie number.\n");
        return;
    }
    index--;
    for (int i = index; i < movieCount - 1; i++) {
        movieList[i] = movieList[i + 1];
    }

    movieCount--;
    saveMovies();

    printf("Movie removed and file updated.\n");
}
```

```
void loadAll() {
    loadPantry(); loadExpenses(); loadHealth(); loadMovies();
}
void saveAll() {
    savePantry(); saveExpenses(); saveHealth(); saveMovies();
}
void editBudget()
{
    char buf[BUF];
    float newBudget;

    printf("Current monthly budget: %.2f\n", userBudget);
    printf("Enter new monthly budget (leave blank to keep the same): ");
    readLine(buf, sizeof(buf));
    if (buf[0] == '\0') {
        printf("Budget not changed.\n");
        return;
    }
    if (stringToFloat(buf, &newBudget) && newBudget >= 0) {
        userBudget = newBudget;
        printf("Budget updated to: %.2f\n", userBudget);
    } else {
        printf("Invalid amount. Budget not changed.\n");
    }
}


int main() {
    loadAll();
    printf("=== Life Organizing System ===\n");
    char choice[80];
    while (1) {
```

```
        printf("\nMain Menu:\n");
        printf("1) Pantry     2) Expenses    3) Health    4) Movies\n");
        printf("5) Alert Center    6) Edit Budget    0) Save & Exit\n");
        printf("Enter choice: "); readLine(choice, sizeof(choice));
        int ch; if (!stringToInt(choice, &ch)) { printf("Invalid choice.\n"); continue; }
        if (ch == 0) { saveAll(); printf("Saved all. Bye!\n"); break; }
        if (ch == 1) {
            printf("\nPantry Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back\nChoice: ");
readLine(choice, sizeof(choice));
            int sc; if (!stringToInt(choice, &sc)) { printf("Invalid.\n"); continue; }
            if (sc==1) pantryAdd(); else if (sc==2) pantryListShow(); else if (sc==3) pantryEdit(); else if
(sc==4) pantryDelete(); else if (sc==5) { savePantry(); printf("Pantry saved.\n"); }
        }
        else if (ch == 2) {
            printf("\nExpenses Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back\nChoice: ");
readLine(choice, sizeof(choice));
            int sc; if (!stringToInt(choice, &sc)) { printf("Invalid.\n"); continue; }
            if (sc==1) expenseAdd(); else if (sc==2) expenseListShow(); else if (sc==3) expenseEdit();
else if (sc==4) expenseDelete(); else if (sc==5) {
                saveExpenses();
                printf("Expenses saved.\n"); }
        }
        else if (ch == 3) {
            printf("\nHealth Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back\nChoice: ");
            readLine(choice, sizeof(choice));
            int sc;
            if (!stringToInt(choice, &sc)) {
                printf("Invalid.\n");
                continue; }
            if (sc==1)
            healthAdd();
            else if (sc==2)
            healthListShow();
            else if (sc==3)
            healthEdit();
```

```
            else if (sc==4)
            healthDelete();
            else if (sc==5) {
                saveHealth();
                printf("Health saved.\n"); }
        }
        else if (ch == 4) {
            printf("\nMovies Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back\nChoice: ");
            readLine(choice, sizeof(choice));
            int sc;
            if (!stringToInt(choice, &sc)) {
                printf("Invalid.\n");
                continue; }
            if (sc==1) movieAdd();
            else if (sc==2)
            movieListShow();
            else if (sc==3)
            movieEdit();
            else if (sc==4)
            movieDelete();
            else if (sc==5) {
                saveMovies();
                printf("Movies saved.\n"); }
        }
        else if (ch == 5) alertCenter();
        else if (ch == 6) editBudget();
        else printf("Invalid main menu option.\n");
    }
    return 0;
}
```

TESTING AND RESULTS

A. PANTRY MODULE TESTING

Test Case 1: Add Pantry Item (Valid Input)

Input:
Name = Milk
Category = Dairy
Quantity = 3
Expiry = 27-11-2025

Expected Output:
"Pantry item added and saved."

```
No   Name                          Category        Amount       Date

-----------------------------------------------------------------
1   MILK                              DAIRY          3    27-11-2025

-- Pantry Alerts --
  Near expiry: MILK (in 2 day(s) on 27-11-2025)
```

Test Case 2: Add Pantry Item (Invalid Quantity)

Input:
Quantity = -5

Expected: Error message: "Invalid quantity."

```
Pantry Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 1
Enter item name: MILK
Enter category: DAIRY
Enter quantity (number): -5
Invalid quantity. Item not added.
```

Test Case 3: Low Stock Alert

Input:
Item qty = 1

Expected:
"Low stock: Milk (qty=1)"

```
Pantry Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 2

No   Name                      Category        Amount       Date

-----------------------------------------------------------------
1   MILK                           DAIRY          1    27-11-2025

-- Pantry Alerts --
   Low stock: MILK (qty=1)
```

Test Case 4: Expired Item Alert

Input:
Expiry = 10-11-2025 (past date)

Expected: Expired alert

```
Pantry Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 2

No   Name                      Category        Amount       Date

-----------------------------------------------------------------
1   MILK                           DAIRY          1    27-11-2025

-- Pantry Alerts --
   Low stock: MILK (qty=1)
  Near expiry: MILK (in 2 day(s) on 27-11-2025)
```

B. EXPENSES MODULE TESTING

Test Case 1: Add Expense

Input:
Name = Groceries

Category = Vegetables

Amount = 500

Expected Output:
"Expense added and saved."

```
Expenses Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 2

No   Name            Category        Amount     Date
-----------------------------------------------------------------
1    GROCERIES       VEGETABLES      ₹500.00  25-11-2025

Total expenses: ₹500.00
  Total expenses so far: 500.00 (Budget: 10000.00)
  Expenses are currently under control.
  Checking for big expenses (>= 1000)...
  No big individual expenses found.
```

Test Case 2: Amount Invalid

Input: Amount = -100

Expected: “Invalid amount.”

```
Main Menu:
1) Pantry     2) Expenses    3) Health    4) Movies
5) Alert Center    6) Edit Budget    0) Save & Exit
Enter choice: 2

Expenses Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 1
Enter expense name: groceries
Enter category: vegetables
Enter amount: -100
Invalid amount.
```

Test Case 3: Budget Alert

Budget = 10000
Expenses = 8500

Expected: "Nearing budget (≥80%)."

```
Expenses Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 1
Enter expense name: VEGETABLES
Enter category: GROCERIES
Enter amount: 8500
 Unable to save expenses file.
Expense added and saved.
  Total expenses so far: 8500.00 (Budget: 10000.00)
  Warning: You have already used more than 80% of your budget.
  Checking for big expenses (>= 1000)...
    Big expense: VEGETABLES (8500.00) on 25-11-2025
```

C. HEALTH MODULE TESTING

Test Case 1: Add Health Record

Input:
Water = 1.5
Calories = 2200
Sleep = 5.0 hrs
Steps = 3000

Expected Alerts:
Low water
Low sleep
Low steps

```
Health Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 1
Date: 25-11-2025
Enter water intake (liters): 1.5
Enter calories: 2200
Enter sleep hours (e.g. 7.5): 5.0
Enter steps: 3000
 Unable to save health file.
Health record added and saved.
  Low water on 25-11-2025: 1.50 L
  Low sleep on 25-11-2025: 5.00 hrs
  Low steps on 25-11-2025: 3000 steps
```

D. MOVIE MODULE TESTING

Test Case: Low Rating Alert

Input:
Movie = ABC

Genre = Action
Rating = 3

Expected:
"Low-rated movie: ABC (3.0)"

```
Main Menu:
1) Pantry     2) Expenses    3) Health    4) Movies
5) Alert Center    6) Edit Budget    0) Save & Exit
Enter choice: 4

Movies Menu: 1.Add 2.List 3.Edit 4.Delete 5.Save 0.Back
Choice: 1
Enter movie name: ABC
Enter genre: Action
Enter rating (0-10): 3.0
 Unable to save movies file.
Movie added and saved.
  Low-rated movie: ABC (3.0)
```

- **File Handling Tests**

Test: Saving and Re-opening Program

- Data saved in pantry.txt, expenses.txt, health.txt, movies.txt
- Program reopened and data loaded correctly

Result:
File I/O Working Perfectly.

APPLICATIONS

1. **Home & Personal Management :-**
   Helps individuals manage their pantry items, daily expenses, health habits and movies from one place.
2. **Budget Planning :-**
   Tracks daily and monthly spending and warns users when they are close to exceeding their budget.
3. **Health Habit Monitoring :-**
   Useful for tracking daily water intake, calorie consumption, sleep hours and steps to maintain a healthy lifestyle.
4. **Pantry & Kitchen Management :-**
   Helps in checking low-stock items, expiry dates and reduces food wastage by giving expiry alerts.
5. **Digital Diary for Daily Activities :-**
   Works like a digital logbook where users can record day-to-day activities like expenses and health updates.
6. **Movie Tracking Application :-**
   Keeps a list of watched movies, their ratings and genres so users can remember what they watched.
7. **Data Organization for Students :-**
   Students can use it to build discipline, manage routines, track expenses and improve daily habits.
8. **Reusability in Other Projects :-**
   This system can be extended to create:
   - Personal assistant applications
   - Household management systems
   - Fitness tracking software
   - Budgeting tools

CONCLUSION

The Life Organizer System successfully integrates multiple daily-life management features into a single program. It provides efficient modules for pantry tracking, expense management, health monitoring, and movie logging. Through systematic testing, all modules were verified to function correctly, producing accurate outputs, handling invalid inputs, and generating necessary alerts such as low stock, over-budget, low health habits, and expired items.

The system also ensures data persistence through file handling, allowing users to save and retrieve data seamlessly. The overall implementation demonstrates the use of important C programming concepts such as structures, arrays, file handling, functions, decision-making, loops, and modular programming.

In conclusion, the project fulfills all its objectives and works as a reliable, user-friendly tool that helps users stay organized and manage different aspects of their daily routine effectively.

FUTURE WORK

1. **Add a Graphical User Interface (GUI):**
   Replace the command-line interface with a user-friendly window-based design.
2. **Mobile App Development:**
   Convert the system into an Android/iOS app for easier access.
3. **Cloud Storage Integration:**
   Store user data online for backup, syncing, and multi-device use.
4. **Advanced Analytics & Charts:**
   Show graphs for expenses, health trends, and pantry usage for better analysis.
5. **Push Notifications & Reminders:**
   Automatic alerts for expiring items, low water intake, overspending, or low sleep.

REFERENCES

1. ANSI C Programming Language – Kernighan & Ritchie
2. Let Us C – Yashavant Kanetkar
3. Online documentation from GeeksforGeeks, Tutorialspoint, W3Schools