Report: Personal Diary Management System

Md Abul Bashar Nirob- 2022198042,
Jannatul Islam Eshita- 2524186042,
Tashin Binte Taiba- 2522283642,
Sara Tasnim- 2522727042,
Tasnia Afrin- 2523855642

Course: cse-115, Group Number: 07
August 12, 2025

***Abstract*—**This report details the design and implementation of a Personal Diary Management System (PDMS) developed in C. The application provides essential functionalities for managing daily entries, including adding, viewing, searching, editing, and deleting records. Key features include data persistence through file I/O, an interactive console-based user interface with colored output, and a basic encryption mechanism for enhanced security. The system also incorporates a backup utility to safeguard user data. This document outlines the system's architecture, implementation specifics, and functional capabilities, demonstrating a practical application of C programming concepts for personal data management.

***Index Terms*—About four key words Diary Management, C Programming, File Handling, Encryption, Console Application.**

I. INTRODUCTION

ersonal diaries have been a traditional means of documenting ppersonal thoughts, events, and experiences. With technological advancement, the need for digital diary systems has grown significantly. Digital diaries not only provide convenience but also address challenges such as data security, easy retrieval, and backups.

The Personal Diary Management System (PDMS) developed in this project addresses these requirements in a simple, modular, and offline format. It is implemented entirely in C, using a command-line interface with color-coded menus for improved user interaction. Unlike many modern applications, PDMS does not require internet access, ensuring privacy and user control over stored data.

Key features include:

- Persistent storage of diary entries in a text file.
- Search and retrieval based on entry dates.
- Editing and deletion of existing entries.
- Encryption and decryption using a user-defined password.
- Automatic backup generation with timestamps.

This report details the design, implementation, features, and results of PDMS, along with potential future enhancement.

II. BACKGROUND AND RELATED WORK

A. Stage one:

While many diary applications exist on both desktop and mobile platforms, most rely on internet-based storage or proprietary frameworks. Common challenges in existing systems include:

- ✓ Lack of full data control (cloud dependency).
- ✓ Weak or absent encryption mechanisms.
- ✓ Overhead from heavy frameworks for simple tasks.

B. Stage two:

Several research works have highlighted the importance of secure personal data storage in offline environments. The proposed PDMS addresses these issues by:

1. Operating fully offline.
2. Using local text files for data storage.
3. Incorporating encryption and backup functionalities.

The modular approach of PDMS also makes it extendable, allowing for future integration with more advanced features such as GUI support and database integration. author only.

III. SYSTEM DESIGN

A. Architecture

PDMS follows a modular architecture, dividing functionality into distinct components for clarity and maintainability:

- Core Module: Handles basic file operations (CRUD: Create, Read, Update, Delete).
- Security Module: ImplementsXOR-based encryption/ decryption for data protection.
- Backup Module: Generates timestamped backups to prevent data loss.
- User Interface (UI): Provides a color-coded console menu for intuitive interaction.

B. Functional Requirements

1. Add New Entry: Users can input a date (DD-MM-YYYY) and a diary entry (up to 1000 characters). Entries are appended to diary.txt in a structured format.

2. View All Entries: Displays all stored entries with clear formatting.

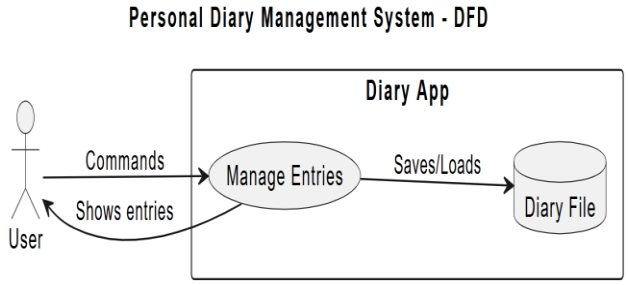
3. Search by Date: Retrieves specific entries using a date-based search.

4. Edit/Delete Entries: Modifies or removes existing entries via temporary file handling.

5. Encrypt/Decrypt Diary: Secures data using password-based XOR encryption. Decryption requires the correct password for access.

6. Backup Creation: Automatically generates backups with timestamps (e.g., diary_backup_20250813_153045.txt).

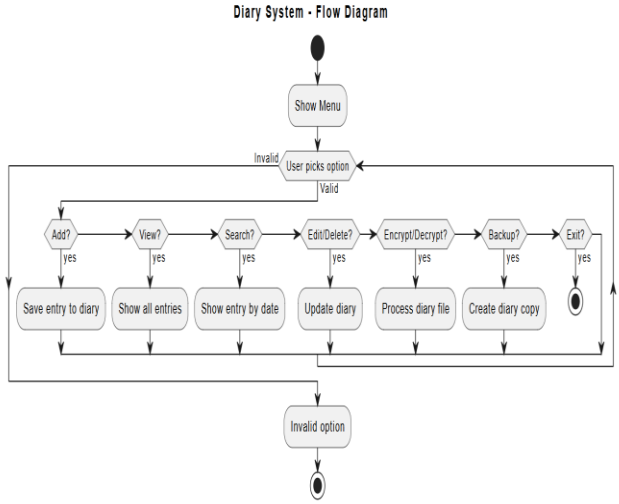
C. *Data flow diagram*



D. *Assumptions and Constraints*

1. Windows environment for color support.
2. Entries limited to 1000 characters.
3. Basic encryption; not resistant to advanced attacks.

E. *Flow Diagram*



The system relies heavily on standard C file I/O operations (fopen, fclose, fprintf, fgets, fgetc, fputc) to manage data persistence. Temporary files are used during edit and delete operations to ensure data integrity before committing changes to the main diary file.

IV. CODES OF PDMS

0. Header file

```c
#ifndef DIARY_H
#define DIARY_H

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

// Function Declarations
void displayMenu();
void addEntry();
void viewEntries();
void searchEntry();
void editEntry();
void deleteEntry();
void encryptFile();
void decryptFile();
void backupDiary();

#endif
```

1.Add Entry

```c
#include "diary.h"
void addEntry() {
    FILE *file = fopen("diary.txt", "a");

    if (file == NULL) {
        printf("Error opening file!\n");
        return;
    }
    char date[20];
    char entry[1000];

    printf("Enter date (DD-MM-YYYY): ");
    scanf("%s", date);
    printf("Write your diary entry:\n");
    scanf(" %[^\n]s", entry);  // Read until newline

    fprintf(file, "Date: %s\nEntry: %s\n\n", date,entry);
    fclose(file);

    printf("Entry saved successfully!\n");
}
```

2. View and Search

```c
#include "diary.h"

void viewEntries() {
    FILE *file = fopen("diary.txt", "r");

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

    char line[1000];
    printf(" _____________________\n");
    printf("|                     |\n");
    printf("| !..All Diary Entries..! |\n");
    printf("|_____________________|\n\n");

    while (fgets(line, sizeof(line), file)) {
        printf("%s", line);
    }
    fclose(file);
}

void searchEntry() {
    FILE *file = fopen("diary.txt", "r");
    char searchDate[20];
    char line[1000];
    int found = 0;

    printf("Enter date to search (DD-MM-YYYY): ");
    scanf("%s", searchDate);

    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, searchDate) != NULL) {
            printf("Entry found:\n%s", line);
            found = 1;
            while (fgets(line, sizeof(line), file)) {
                if (strstr(line, "Date:") != NULL) break;
                printf("%s", line);
            }
        }
    }

    if (!found) printf("No entry found for %s\n",
searchDate);
    fclose(file);
}
```

3. Edit and Delete

```c
#include "diary.h"
#include <ctype.h>

// Add this function that's currently missing
void editEntry() {
    FILE *file = fopen("diary.txt", "r");
    if (file == NULL) {
        printf("Error: Diary file not found!\n");
        return;
    }

    FILE *temp = fopen("temp.txt", "w");
    if (temp == NULL) {
        printf("Error creating temporary file!\n");
        fclose(file);
        return;
    }

    char searchDate[20];
    char line[1000];
    int found = 0;

    printf("Enter date of the entry to edit (DD-MM-YYYY):
");
    scanf("%19s", searchDate);

    while (fgets(line, sizeof(line), file)) {
        if (!found && strstr(line, searchDate) != NULL) {
            found = 1;
            printf("Current entry:\n%s", line);

            // Get the existing entry content
            if (fgets(line, sizeof(line), file)) {
                printf("%s", line);
            }

            // Get new entry
            char newEntry[1000];
            printf("Enter new entry text: ");
            scanf(" %[^\n]", newEntry);

            // Write updated entry
            fprintf(temp, "Date: %s\nEntry: %s\n\n",
searchDate, newEntry);
        } else {
            fputs(line, temp);}
    }
```

```
    fclose(file);
    fclose(temp);

    if (!found) {
        printf("No entry found for %s\n", searchDate);
        remove("temp.txt");
    } else {
        remove("diary.txt");
        rename("temp.txt", "diary.txt");
        printf("Entry updated successfully!\n");
    }
}

// Your existing delete function
void deleteEntry() {
    FILE *file = fopen("diary.txt", "r");
    if (file == NULL) {
        printf("Error: No entries found!\n");
        return;
    }

    FILE *temp = fopen("temp.txt", "w");
    if (temp == NULL) {
        printf("Error creating temporary file!\n");
        fclose(file);
        return;
    }

    char searchDate[20];
    char line[1000];
    int found = 0;

    printf("Enter date of the entry to delete (DD-MM-
YYYY): ");
    scanf("%19s", searchDate);

    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, searchDate) != NULL) {
            found = 1;
            printf("Deleted entry:\n%s", line);
            // Skip the entry content
            fgets(line, sizeof(line), file);
        } else {
            fputs(line, temp);
        }
    }

    fclose(file);
    fclose(temp);

    if (!found) {
        printf("No entry found for %s\n", searchDate);
        remove("temp.txt");
    } else {
        remove("diary.txt");
        rename("temp.txt", "diary.txt");
        printf("Entry deleted successfully!\n");
    }
}
```

4. Encrypt and Decrypt

```
#include "diary.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
void encryptFile() {
    char password[50];
    printf("Enter encryption password: ");
    scanf("%49s", password);

    FILE *source = fopen("diary.txt", "rb");
    if (!source) {
        printf("Error: Could not open diary.txt\n");
        return;
    }

    FILE *dest = fopen("diary_encrypted.txt", "wb");
    if (!dest) {
        printf("Error: Could not create encrypted
file\n");
        fclose(source);
        return;
    }

    // Store password length as verification
    int key_length = strlen(password);
    fprintf(dest, "DPASS:%d:", key_length); // Header
format

    int ch, i = 0;
    while ((ch = fgetc(source)) != EOF) {
        fputc(ch ^ password[i % key_length], dest);
        i++;
    }
```

```
    fclose(source);
    fclose(dest);
    printf("Encryption successful! Remember your password
is: %s\n", password);
}

void decryptFile() {
    char password[50];
    printf("Enter decryption password: ");
    scanf("%49s", password);

    FILE *source = fopen("diary_encrypted.txt", "rb");
    if (!source) {
        printf("Error: Encrypted file not found\n");
        return;
    }
    // Verify file header
    char header[20];
    int stored_length;
    if (fscanf(source, "DPASS:%d:", &stored_length) != 1
||
        strlen(password) != stored_length) {
        printf("ACCESS DENIED! Invalid password
length\n");
        printf("You may now view/edit your diary
entries\n");
        fclose(source);
        return;
    }

    // Decrypt to temporary memory first
    char temp_content[10000];
    int content_size = 0;
    int ch, i = 0;

    while ((ch = fgetc(source)) != EOF && content_size <
9999) {
        temp_content[content_size++] = ch ^ password[i %
stored_length];
        i++;
    }
    temp_content[content_size] = '\0';
    fclose(source);
    // Verify content structure
    if (strstr(temp_content, "Date: ") == NULL ||
        strstr(temp_content, "Entry: ") == NULL) {
        printf("ACCESS DENIED! Wrong password\n");
        return; }
    // Save to file if valid
    FILE *dest = fopen("diary_decrypted.txt", "w");
    if (dest) {
        fputs(temp_content, dest);
        fclose(dest);
        printf("SUCCESS! Diary restored perfectly!\n");
    } else {
        printf("Error saving decrypted file\n");
    }
}
```

5. Backup data

```
// backup_diary.c
#include "diary.h"
#include <time.h>
void backupDiary() {
    time_t now = time(NULL);
    struct tm *t = localtime(&now);
    char backupFilename[50];

    // Create timestamped filename
    sprintf(backupFilename,
"diary_backup_%04d%02d%02d_%02d%02d%02d.txt",
            t->tm_year + 1900, t->tm_mon + 1, t->tm_mday,
            t->tm_hour, t->tm_min, t->tm_sec);

    FILE *source = fopen("diary.txt", "r");
    if (!source) {
        printf("Error: No diary entries found to
backup!\n");
        return;
    }
    FILE *dest = fopen(backupFilename, "w");
    if (!dest) {
        printf("Error creating backup file!\n");
        fclose(source);
        return;
    }
    char ch;
    while ((ch = fgetc(source)) != EOF) {
        fputc(ch, dest);
    }
    fclose(source);
    fclose(dest);
    printf("Backup created successfully as: %s\n",
backupFilename);
}
```

6. Main Function

```
#include "diary.h"
#include <locale.h>
#include <windows.h>

// Extended color definitions
#define COLOR_RED       12
#define COLOR_GREEN     10
#define COLOR_YELLOW    14
#define COLOR_BLUE       9
#define COLOR_MAGENTA   13
#define COLOR_CYAN      11
#define COLOR_WHITE     15
#define COLOR_PURPLE     5
#define COLOR_LIME      10
#define COLOR_TEAL       3
#define COLOR_ORANGE     6
#define COLOR_PINK      13
#define COLOR_GRAY       8

void setColor(int color) {
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hConsole, color);
}
int main() {
    setlocale(LC_ALL, "");
    system("chcp 65001");

    int choice;
    while(1) {
        displayMenu();
        setColor(COLOR_CYAN);
        printf("Choose a option...! ");
        setColor(COLOR_WHITE);
        scanf("%d", &choice);

        switch(choice) {
            case 1: setColor(COLOR_LIME); addEntry();
break;
            case 2: setColor(COLOR_TEAL); viewEntries();
break;
            case 3: setColor(COLOR_ORANGE);
searchEntry(); break;
            case 4: setColor(COLOR_PINK); editEntry();
break;
            case 5: setColor(COLOR_RED); deleteEntry();
break;
            case 6: setColor(COLOR_PURPLE);
encryptFile(); break;
            case 7: setColor(COLOR_YELLOW);
decryptFile(); break;
            case 8: setColor(COLOR_GRAY); backupDiary();
break;  // New option
            case 9: setColor(COLOR_CYAN);
                    printf("\nExiting... See you next
time! \n\n");
                    exit(0);
            default: setColor(COLOR_RED);
                     printf("\nInvalid choice! Please try
again.\n");
        }
        setColor(COLOR_WHITE);
    }
    return 0;
}

void displayMenu() {
    setColor(COLOR_CYAN);

    printf(" _____\n");
    printf("|                                    |\n");
    printf("| Personal Diary Management System   |\n");
     ("|_____|\n\n");

    setColor(COLOR_LIME);
    printf("1. Add New Entry\n");
    setColor(COLOR_TEAL);
    printf("2. View All Entries\n");
    setColor(COLOR_ORANGE);
    printf("3. Search Entry\n");
    setColor(COLOR_PINK);
    printf("4. Edit Entry\n");
    setColor(COLOR_RED);
    printf("5. Delete Entry\n");
    setColor(COLOR_PURPLE);
    printf("6. Encrypt Diary File\n");
    setColor(COLOR_YELLOW);
    printf("7. Decrypt Diary File\n");
    setColor(COLOR_GRAY);
    printf("8. Backup Diary\n");  // New menu item
    setColor(COLOR_BLUE);
    printf("9. Exit\n");
    setColor(COLOR_WHITE);
    printf("\n");
}
```

V. Implementation Details

A. Technologies Used

The system is implemented in C using standard libraries like stdio.h, stdlib.h, string.h, and time.h. Windows-specific features use windows.h for console colors.

1. Programming Language: C
2. Libraries:
   - ✓ <stdio.h> – Input/Output operations.
   - ✓ <stdlib.h> – Memory and process control.
   - ✓ <string.h> – String manipulation.
   - ✓ <time.h> – Timestamp for backups.
   - ✓ <windows.h> – Console color handling.
   - ✓ <locale.h> – UTF-8 support.

B. User Interface

The system features a color-coded console interface to enhance user experience. Different colors are assigned to various menu options and feedback messages. The interface includes:

1. A decorative header for the application
2. Color-coded menu options
3. Colored feedback messages based on user actions

So, the main.c file controls the program's flow. It presents a main menu (displayMenu()) to the user in a continuous loop, allowing them to choose from various operations (add, view, search, edit, delete, encrypt, decrypt, backup, exit). User input is captured using scanf(), and a switch statement directs the program to the appropriate function. The use of predefined color macros (COLOR_RED, COLOR_LIME, etc.) and the setColor() function provides visual feedback and improves menu readability.

VI. Results and Testing

The following test cases were executed to validate the system:

A. Functionality Testing

- ➢ Add/View: Successfully adds and displays entries.
- ➢ Search/Edit/Delete: Accurately locates and modifies entries by date.
- ➢ Encrypt/Decrypt: Encrypts file; decrypts only with correct password. Invalid passwords deny access.
- ➢ Backup: Creates dated backups without errors.

B. Edge Cases

- ✓ Non-existent file: Graceful error messages.
- ✓ Invalid dates: Handled by user input validation (basic).
- ✓ Large entries: Limited by buffer size; no overflows observed.

C. Performance

Runs instantaneously on standard hardware; file operations are efficient for small diaries. (No figures included as this is text-based; in a Word file, insert screenshots of console output).

VII. Challenges & Solutions

Challenge	Solution
1. Multi-line entry handling	1. Used scanf("%[^\n]s", entry)
2. Password verification	2. Stored length in encrypted file header
3. Temporary file management	4. temp.txt for edit/delete operations

VIII. Limitations of PDMS

The encryption method, while functional, uses a basic XOR algorithm that may not provide sufficient security against sophistica ted attacks. The system is limited to text-based entries and does not support multimedia content. The search functionality is restricted to date-based searches only.

1. No GUI: Currently console-based; a graphical interface would enhance usability.
2. No Cloud Sync: Entries are stored locally.

IX. Discussion

The implementation of the PDMS demonstrates the effective application of fundamental programming concepts in developing a practical personal information management tool. The modular design approach facilitated the development process and allows for future enhancements.

The XOR encryption, while not suitable for highly sensitive information, provides a basic level of security suitable for personal diary content. The backup functionality ensures data preservation, addressing a critical concern for digital diary users.

X. Future Work

The Personal Diary Management System presented in successfully addresses the need for a simple, secure, and efficient platform for managing personal diary entries. The system provides all essential features for diary management while ensuring data security.

Several enhancements could be implemented to improve in future:

- ✓ Implement AES encryption for stronger security.
- ✓ Add multi-user authentication.
- ✓ Create a GUI version using C++/Qt or Python.
- ✓ Enable export/import in formats like PDF or Word.
- ✓ Introduce search by keyword in entry content.

XI. Conclusion

The Personal Diary Management System successfully demonstrates the application of fundamental C programming concepts to create a functional and practical utility for personal record-keeping. The system effectively manages diary entries through file I/O, provides an interactive user experience with colored console output, and incorporates essential features like basic data encryption and automated backups. This project serves as a solid foundation for understanding file management, string manipulation, basic security implementations in C. While the encryption is rudimentary, it

showcases the potential for integrating security features into such applications. This project underscores the importance of modular design and effective file handling for developing software tools.

XII. References

[1] Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.

[2] Deitel, P., & Deitel, H. (2016). C How to Program (8th ed.). Pearson.

[3] B. W. Kernighan and D. M. Ritchie, The C Programming Language, 2nd Ed., Prentice Hall, 1988.

[4] Day One Journal App. [Online]. Available: https://dayoneapp.com/

[5] RedNotebook. [Online]. Available: https://rednotebook.sourceforge.io/

[6] Vim-Diary Plugin. [Online]. Available: https://github.com/vim-denops/deno-vim-channel-command (example reference).

[7] GNU Project, "GNU C Library Documentation," Available: https://www.gnu.org/software/libc/manual/