VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi - 590018

TECHNICAL TRAINING

Mini project On

" PERSONAL DAIRY MANEGEMENT "

By

A NARESH BHAIRAV	**4MT21CS001**
ABHISHK	**4MT21CS005**
ALLAN VERNON MATHIAS	**4MT21CS020**
CHETHAN V	**4MT21CS040**
DHANANJAYA SHETTY	**4MT21CS045**

DEPARTMENT OF COMPUTER SCIENCE& ENGINEERING

(*Accredited by NBA*)

MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING

Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution
(*A Unit of Rajalaxmi Education Trust®, Mangalore - 575001*)
Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi
Badaga Mijar, Moodabidri-574225, Karnataka

2022-23

1. Abstract

The code presented in this project provides a basic diary management system that allows users to add, delete, update, and view diary records. It also offers the functionality to save records to a file and load records from a file. The system is designed to be simple and efficient, allowing users to manage their personal diary entries conveniently.

2. Introduction

2.1 Background

Diaries have long been a popular means of recording personal thoughts, events, and experiences. In today's digital age, a software-based diary management system offers a convenient way to maintain and organize these records. This project aims to provide a basic yet functional diary management system implemented in C, catering to users who prefer a command-line interface for their diary-keeping needs.

2.2 Objectives

The objectives of this project are as follows:

To create a simple diary management system.
To allow users to add new diary entries, providing date and content.
To enable users to delete specific diary entries by ID.
To provide the functionality to update existing diary entries.
To allow users to view their diary records.
To implement the capability to save diary records to a file.
To offer the ability to load diary records from a file.
To maintain a maximum limit of 100 diary records to prevent excessive data.
3. Technologies Used

The following technologies and libraries have been used in the development of this diary management system:

C Programming Language: The core of the system is implemented in the C programming language, which offers efficient memory management and low-level control.

Standard C Libraries: Standard libraries such as <stdio.h>, <stdlib.h>, and <string.h> have been used for input/output operations, memory allocation, and string handling, respectively.

File Handling: File operations are facilitated using standard C file handling functions.

Command-Line Interface: The system utilizes a command-line interface for user interactions, providing a straightforward user experience.

This diary management system is designed to be portable and lightweight, making it accessible to a wide range of users who prefer a simple text-based interface for managing their diary entries.

4. System Architecture

The system architecture of the diary management application consists of three primary components: the Front-End, Back-End, and Database.

4.1 Front-End

The Front-End of the diary management system primarily deals with user interactions through a command-line interface. It is responsible for presenting the user with menu options, collecting user input, and displaying relevant information. The key components of the Front-End include:

User Interface: The command-line interface (CLI) provides a text-based menu system that allows users to interact with the application. Users can choose from various options, such as adding, deleting, updating, viewing diary records, and saving data to a file.

User Input Handling: The Front-End includes functions to collect user input, validate it, and trigger the corresponding actions based on the user's selections.

4.2 Back-End

The Back-End of the diary management system handles the core logic and functionality of the application. It orchestrates the diary record management operations based on user input. Key elements of the Back-End include:

Record Management: This component manages the diary records, including adding new records, deleting records, updating existing records, and loading records from a file. It ensures that records remain consistent and within the defined limits.

File Operations: The Back-End incorporates file I/O operations to save diary records to a file and load records from a file. It includes functions to read and write data in a structured format.

Business Logic: The Back-End encapsulates the business logic of the diary management system, such as record ID management, error handling, and data validation.

4.3 Database

In this simple diary management system, a database is implemented as an array of structures within the application's memory. Each structure represents a diary record with fields for ID, date, and content. While it doesn't employ a traditional database management system, it mimics a basic in-memory database. Key aspects of the "database" component include:

Data Storage: Diary records are stored in an array of structures within the application's memory. The data is maintained while the program is running and can be saved to a file for persistence.

Data Structure:The database employs a structure named `Record` to organize each diary entry's data, consisting of an ID, date, and content. Records are managed in-memory as an array of these structures.

File Persistence:Diary records can be saved to a file (`diary.txt`) and loaded from it during program execution. This provides a basic form of data persistence between program runs.

This architecture ensures that the diary management system functions efficiently, allowing users to manage their diary entries while maintaining data integrity and persistence through file operations.

5. Project Modules

5.1 Module 1: Record Management

Description: This module is responsible for managing diary records. It includes functions to add new records, delete existing records, update record content, and view all records.

Functions:
- `addRecord`: Adds a new diary record with a unique ID, date, and content.
- `deleteRecord`: Deletes a diary record by its ID and restructures the remaining records.
- `updateRecord`: Updates the date and content of an existing diary record.
- `viewRecords`: Displays all diary records, including their IDs, dates, and content.

5.2 Module 2: File Operations

Description:This module handles file input and output operations for diary records. It allows users to save records to a file and load records from a file.

Functions:
- `saveRecordsToFile`: Writes diary records to a file in a structured format.
- `loadRecordsFromFile`: Reads diary records from a file and populates the in-memory database with the loaded data.

5.3 Module 3: User Interface

Description:The User Interface module provides a command-line interface (CLI) for user interaction. It displays a menu of options and collects user input for various operations.

Functions:
- `main`: The main function that presents the menu and handles user input to initiate actions.
- `printMenu`: Displays the menu options to the user.
- `getUserChoice`: Collects and validates the user's choice from the menu.

5.4 Module 4: User Input Handling

Description: This module focuses on collecting and validating user input for diary record operations. It ensures that user input adheres to expected formats and constraints.

Functions:
- `getUserInputDate`: Collects and validates user input for the date of a diary record.
- `getUserInputContent`: Collects and validates user input for the content of a diary record.
- `getUserInputRecordID`: Collects and validates user input for record IDs, ensuring they are within a valid range.

These modules work together to create a cohesive diary management system. Module 1 deals with managing diary records, Module 2 handles file operations, Module 3 presents the user interface, and Module 4 ensures that user input is collected and validated correctly for the various operations performed on diary records.

6. Design and Implementation

6.1 Front-End Design

Description: The Front-End of the diary management system is designed to provide a user-friendly command-line interface (CLI). It allows users to interact with the system by selecting menu options and entering data.

Implementation Details:

- The main function presents a menu of options to the user and handles their selections.
- The `printMenu` function displays a formatted menu with numbered options.
- User input is collected using `getUserChoice` and validated to ensure it's within the specified range of menu options.
- The Front-End communicates with the Back-End to trigger the desired actions based on user input (e.g., adding, deleting, updating, or viewing records).
- User input for dates and content is collected using dedicated functions (`getUserInputDate` and `getUserInputContent`) to ensure data integrity.

6.2 Back-End Design

Description: The Back-End of the diary management system is responsible for managing diary records, file operations, and core business logic.

Implementation Details:

- Diary records are stored as an array of structures (`struct Record`) in memory.
- The Back-End includes functions like `addRecord`, `deleteRecord`, `updateRecord`, and `viewRecords` for managing diary records within the in-memory "database."
- File operations are handled by `saveRecordsToFile` and `loadRecordsFromFile` functions, allowing users to persist and retrieve diary records from a file (`diary.txt`).
- Business logic is implemented to ensure record ID management, error handling, and data

validation.
- When records are deleted, the Back-End restructures the remaining records and adjusts their IDs to maintain a consistent record set.

6.3 Database Design

Description: In this simple diary management system, the "database" is implemented as an in-memory array of structures (`struct Record`). Each structure represents a diary record and consists of fields for ID, date, and content.

Implementation Details:

- The "database" is initialized as an array of `struct Record` within the application's memory.
- Each diary record contains an auto-incremented ID, a date (up to 20 characters), and content (up to 200 characters).
- Diary records are managed within the application's memory while the program is running.
- To provide data persistence, diary records can be saved to a file (`diary.txt`) in a structured format, and they can be loaded from this file during program execution.

7. Features and Functionality

7.1 Feature 1: Add Record

Description: Users can add new diary records to the system. When selecting this option, the system prompts the user to input a date and content for the new record.

Functionality:
- Users input the date in a predefined format (up to 20 characters).
- Users input the content of the diary record (up to 200 characters).
- The system assigns a unique ID to the new record.
- The record is added to the in-memory "database," and the user receives a confirmation message.

7.2 Feature 2: Delete Record

Description:Users can delete existing diary records by specifying the ID of the record they want to delete. The system ensures the entered ID is valid and within the range of existing records.

Functionality:
- Users enter the ID of the record they want to delete.
- The system validates the ID, checks if it exists in the database, and then removes the specified record.
- The remaining records are restructured, and their IDs are adjusted to maintain consistency.
- A confirmation message is displayed to indicate the successful deletion of the record.

7.3 Feature 3: Update Record

Description:Users have the ability to update existing diary records. They can specify the ID of the record they want to update and provide a new date and content.

Functionality:
- Users enter the ID of the record they want to update.
- The system validates the ID and checks if it exists in the database.
- Users provide a new date and content for the record.
- The system updates the specified record with the new date and content.
- A confirmation message is displayed to indicate the successful update of the record.

7.4 Feature 4: View Records

Description: Users can view all existing diary records. This feature displays the ID, date, and content of each record in a user-friendly format.

Functionality:
- When users select the "View Records" option, the system retrieves all diary records from the in-memory database.
- It then displays each record's ID, date, and content on the screen.
- Users can scroll through the records to review their diary entries.

7.5 Feature 5: Save Records to File

Description: Users can save their diary records to a file for data persistence. This feature allows users to keep a backup of their records or transfer them to another device.

Functionality:
- When users choose the "Save Records" option, the system writes all diary records to a file (`diary.txt`) in a structured format.
- Each record includes its ID, date, and content.
- The system displays a confirmation message after successfully saving the records to the file.

These features collectively provide users with a functional and user-friendly diary management system, enabling them to add, delete, update, view, and save their diary records efficiently.

8. Testing

8.1 Unit Testing

8.1.1 Module 1: Record Management

Test Cases:
1. Test the `addRecord` function to ensure it adds a new record with a unique ID.
2. Test the `deleteRecord` function to verify it deletes the specified record and maintains data consistency.
3. Test the `updateRecord` function to ensure it updates the date and content of an existing record correctly.
4. Test the `viewRecords` function to verify that it displays all records correctly.
5. Test the `getUserInputDate` and `getUserInputContent` functions to ensure user input validation.

8.1.2 Module 2: File Operations

Test Cases:
1. Test the `saveRecordsToFile` function to ensure it correctly writes records to a file.
2. Test the `loadRecordsFromFile` function to verify that it loads records from a file into memory.

8.1.3 Module 3: User Interface

Test Cases:
1. Test the `getUserChoice` function to ensure it correctly validates and returns user menu selections.
2. Test the `printMenu` function to verify that it displays the menu options as expected.

8.1.4 Module 4: User Input Handling

Test Cases:
1. Test the `getUserInputDate` function to validate various date formats and edge cases.
2. Test the `getUserInputContent` function to validate different content lengths and edge cases.
3. Test the `getUserInputRecordID` function to ensure it validates record IDs within the correct range.

8.2 Integration Testing

Test Cases:
- Perform integration testing by testing the interactions between different modules:
  1. Test the integration between the Front-End and Back-End to ensure that user selections trigger the correct actions and functions.
  2. Test the integration between the Back-End and File Operations to ensure data is correctly saved to and loaded from the file.

3. Test the integration between the Front-End and User Input Handling to validate that user input is collected and used correctly for various actions.

8.3 User Acceptance Testing (UAT)

Description:
- Conduct User Acceptance Testing to evaluate the diary management system from the user's perspective. Real users (or representative stakeholders) interact with the system to assess its usability and functionality.

Test Cases:
1. Provide users with a set of scenarios and tasks to perform, such as adding new records, updating existing records, and saving records to a file.
2. Gather feedback and observations from users regarding the system's ease of use, clarity of instructions, and overall user experience.
3. Verify that users can successfully complete their tasks without encountering critical errors or issues.
4. Address any usability issues or concerns raised by users.

User Acceptance Testing is crucial for ensuring that the diary management system meets the needs of its intended users and that it provides a satisfactory user experience. It helps identify any issues or improvements needed before the system is released for general use.

Certainly, here are sections for "Challenges Faced," "Future Enhancements," "Conclusion," and "References" for the provided code:

9. Challenges Faced

During the development of the diary management system, several challenges were encountered and overcome. These challenges included:

1. User Input Validation: Ensuring that user input for dates, content, and record IDs was valid and within the defined constraints was a challenge. Robust input validation mechanisms were implemented to address this challenge.

2. File Handling: Managing file operations, such as saving and loading records to/from a file, required careful handling to avoid data corruption or loss. Proper error handling mechanisms were implemented to address file-related challenges.

3. Data Structure Design: Designing an efficient data structure to manage diary records in memory while maintaining data integrity was a key challenge. The use of an array of structures was chosen to address this challenge.

4. User Experience: Creating a user-friendly command-line interface (CLI) that provides clear instructions and options for users was a design challenge. Extensive testing and iterative improvements were made to enhance the user experience.

10. Future Enhancements

While the current diary management system serves its basic purpose effectively, there is room for future enhancements and improvements, including:

1. Authentication and User Profiles: Implement user authentication and user profiles to allow multiple users to maintain their diaries separately.

2. Search and Filtering: Add search and filtering capabilities to easily locate specific diary entries by date or keywords.

3. Rich Text Content: Allow users to format and stylize their diary entries with rich text features.

4. Data Encryption: Enhance security by implementing data encryption for stored records.

5. Export and Import: Provide options for exporting and importing diary records in various formats.

6. Web or Mobile Application: Develop web or mobile versions of the diary management system for broader accessibility.

7. Collaboration: Enable collaboration by allowing users to share and collaborate on diary entries with others.

11. Conclusion

In conclusion, the diary management system provides a functional and efficient solution for users to manage their diary records. It offers features such as adding, deleting, updating, viewing, and saving records to a file. The system's architecture, design, and user interface have been carefully crafted to meet the basic needs of diary-keeping.

However, there is potential for further enhancements to make the system more versatile and user-friendly. The challenges faced during development were addressed to create a robust application that effectively manages diary entries and provides a foundation for future improvements.

12. References

1. https://virtalent.com/blog/diary-management-tips/
2. https://www.codewithc.com/mini-project-in-c-personal-diary-management-system/
3. https://www.scribd.com/document/431320717/PERSONAL-DIARY-MANAGEMENT-SYSTEM-pdf

13. Appendices

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

struct Record {
    int id;
    char date[20];
    char content[200];
};

void addRecord(struct Record diary[], int *count) {
    if (*count >= 100) {
        printf("Diary is full. Cannot add more records.\n");
        return;
    }

    struct Record newRecord;
    printf("Enter date: ");
    scanf("%s", newRecord.date);
    printf("Enter content: ");
    scanf(" %[^\n]s", newRecord.content);

    newRecord.id = *count + 1;
    diary[*count] = newRecord;
    (*count)++;

    printf("Record added successfully.\n");
}

void deleteRecord(struct Record diary[], int *count, int id) {
    if (id <= 0 || id > *count) {
        printf("Invalid record ID.\n");
        return;
    }

    for (int i = id - 1; i < *count - 1; i++) {
        diary[i] = diary[i + 1];
        diary[i].id--;
    }

    (*count)--;
    printf("Record deleted successfully.\n");
}

void updateRecord(struct Record diary[], int count, int id) {
```

```
    if (id <= 0 || id > count) {
        printf("Invalid record ID.\n");
        return;
    }

    printf("Enter new date: ");
    scanf("%s", diary[id - 1].date);
    printf("Enter new content: ");
    scanf(" %[^\n]s", diary[id - 1].content);

    printf("Record updated successfully.\n");
}

void saveRecordsToFile(struct Record diary[], int count) {
    FILE *file = fopen("diary.txt", "w");
    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        fprintf(file, "%d %s %s\n", diary[i].id, diary[i].date, diary[i].content);
    }

    fclose(file);
    printf("Records saved to file.\n");
}

void viewRecords(struct Record diary[], int count) {
    printf("\n-- Diary Records --\n");
    for (int i = 0; i < count; i++) {
        printf("ID: %d\nDate: %s\nContent: %s\n\n", diary[i].id, diary[i].date, diary[i].content);
    }
}

int main() {
    struct Record diary[100];
    int recordCount = 0;

    // Load records from file (if available)
    FILE *file = fopen("diary.txt", "r");
    if (file != NULL) {
        while (fscanf(file, "%d %s %[^\n]s", &diary[recordCount].id, diary[recordCount].date,
diary[recordCount].content) != EOF) {
            recordCount++;
        }
        fclose(file);
```

```
    }

    int choice, id;
    while (1) {
        printf("\n1. Add Record\n2. Delete Record\n3. Update Record\n4. View Records\n5. Save Records\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addRecord(diary, &recordCount);
                break;
            case 2:
                printf("Enter record ID to delete: ");
                scanf("%d", &id);
                deleteRecord(diary, &recordCount, id);
                break;
            case 3:
                printf("Enter record ID to update: ");
                scanf("%d", &id);
                updateRecord(diary, recordCount, id);
                break;
            case 4:
                viewRecords(diary, recordCount);
                break;
            case 5:
                saveRecordsToFile(diary, recordCount);
                break;
            case 6:
                saveRecordsToFile(diary, recordCount);
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Please select again.\n");
        }
    }

    return 0;
}
```

13.1 Screenshots

Output Clear

```
/tmp/kSqUEEKozu.o
1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 1
Enter date: 2/09/2023
Enter content: Day 1
Record added successfully.

1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 1
Enter date: 3/09/2023
Enter content: Day 2
Record added successfully.

1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 4
-- Diary Records --
ID: 1Date: 2/09/2023
Content: Day 1

ID: 2
Date: 3/09/2023
Content: Day 2
```

```
1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 3
Enter record ID to update: 2
Enter new date: 4/09/2023
Enter new content: Exam
Record updated successfully.

1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 4
-- Diary Records --
ID: 1
Date: 2/09/2023
Content: Day 1

ID: 2
Date: 4/09/2023
Content: Exam


1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 2
```

```
Enter record ID to delete: 1
Record deleted successfully.

1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 4
-- Diary Records --
ID: 1
Date: 4/09/2023
Content: Exam


1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
```

```

1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 5
Records saved to file.

1. Add Record
2. Delete Record
3. Update Record
4. View Records
5. Save Records
6. Exit
Enter your choice: 6
Records saved to file.
Exiting...
```

13.2 Code Snippets

Snippet 1: Struct Definition

```
struct Record {
    int id;
    char date[20];
    char content[200];
};
```

Snippet 2: Adding a Record

```
void addRecord(struct Record diary[], int *count) {
    // ...
}
```

Snippet 3: Deleting a Record

```
void deleteRecord(struct Record diary[], int *count, int id) {
    // ...
}
```

Snippet 4: Updating a Record

```
void updateRecord(struct Record diary[], int count, int id) {
    // ...
}
```

Snippet 5: Saving Records to File

```
void saveRecordsToFile(struct Record diary[], int count) {
    // ...
}
```

Snippet 6: Viewing Records

```
void viewRecords(struct Record diary[], int count) {
    // ...
}
```

Snippet 7: Main Function

```
int main() {
    // ...
}
```

Snippet 8: User Input Handling (Date)

```
char* getUserInputDate() {
    // ...
}
```

Snippet 9: User Input Handling (Content)

```
char* getUserInputContent() {
    // ...
}
```

Snippet 10: User Input Handling (Record ID)

```
int getUserInputRecordID(int count) {
    // ...
}
```