

ISO 9001:2015

AICTE-CII: GOLD Category Institute

NAAC-'A' Grade Institute (CGPA: 3.19)

NIRF-2024 Rank Band : 201-300



KAKATIYA INSTITUTE OF TECHNOLOGY & SCIENCE

Opp : Yerragattu Gutta, Hasanparthy (Mandal), WARANGAL - 506 015, TELANGANA, INDIA

काकतीय प्रौद्योगिकी एवं विज्ञान संस्थान, वरंगल - ५०६०१५, तेलंगाना, भारत

కాకతీయ సాంకేతిక విజ్ఞాన శాస్త్ర విద్యాలయం, వరంగల్ - ౫౦౬ ౦౧౫ తెలంగాణ, భారతదేశము

(An Autonomous Institute under Kakatiya University, Warangal)

(Approved by AICTE, New Delhi; Recognised by UGC under 2(f) & 12(B); Sponsored by EKASILA EDUCATION SOCIETY)

website: www.kitsw.ac.in

E-mail: principal@kitsw.ac.in

☎ : +91 9392055211, +91 7382564888

U24EL108 PRACTICUM-II

A Report on Practicum

titled

Expense Tracker

by

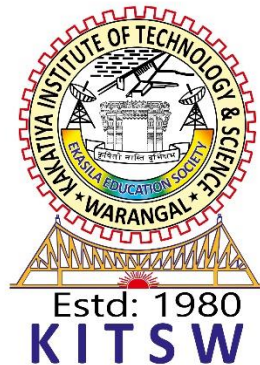
MIRZA HUUSAIN ALI

B24DS093

Under the guidance of

Sri I.Sai Rama Krishna

Assistant Professor, Dept. of CSE(AIML)



KAKATIYA INSTITUTE OF TECHNOLOGY & SCIENCE, WARANGAL

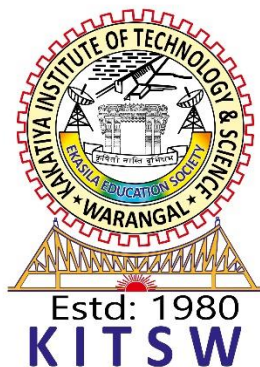
(An Autonomous Institute under Kakatiya University, Warangal)

(2024-25)

KAKATIYA INSTITUTE OF TECHNOLOGY & SCIENCE, WARANGAL

(An Autonomous Institute under Kakatiya University, Warangal)

CERTIFICATE



This is to certify that, this is the Bonafide record of Practicum entitled **Expense Tracker** in the course **Data Structures Through C** carried out by **Mirza Hussain Ali** bearing **Roll No. B24DS093**, student of **B. Tech, CSD, II Semester** in partial fulfilment for the Degree of Bachelor of Technology in Department of **Computer Science and Engineering (Data Science)**

**SUPERVISOR/
COURSE TEACHER**

Sri I.Sai Rama Krishna
Assistant Professor
Dept. of CSE(AIML), KITSW

**PRACTICUM COORDINATOR/
CLASS TEACHER**

Dr. D.Praveena
Assistant Professor
Dept. of PS, KITSW

HEAD OF THE DEPARTMENT

Dr. S. Narasimha Reddy
Professor & Head
Dept. of CSE(AIML), KITSW

ACKNOWLEDGEMENT

I wish to express a sense of gratitude to my guide, **Sri I.Sai Rama Krishna, Assistant Professor**, Department of CSE(AIML), Kakatiya Institute of Technology & Science, Warangal, who guided me at every moment during my entire Practicum work and gave valuable suggestions. His continuous encouragement at each point of work and effort to push the work through are grateful and acknowledged.

I sincerely thank the Practicum coordinator, **Sri I.Sai Rama Krishna, Assistant Professor**, for timely coordination.

I am indebted to **Dr. S. Narsimha Reddy Associate Professor & Head of the Department**, Computer Science and Engineering (AIML). I also extend my gratitude to all the faculty members of the department without whose support at various stages this report will not be materialized.

Also, I offer my sincere admiration to **Prof. K. ASHOKA REDDY, Principal**, Kakatiya Institute of Technology & Science, Warangal for his kind patronage and permission to utilize the resource of the institute to carry out the work.

Finally, I wish to thank my friends and seniors who helped me directly or indirectly in the successful completion of this work.

MIRZA HUSSAIN ALI

B24DS093

CONTENTS

Chapter no	Chapter Title	Page number
	Certificate	ii
	Acknowledgement	iii
	Contents	iv
1	Overview of the Practicum	1
1.0	Introduction	1
1.1	Literature Review	2
1.2	Research Gaps	3
1.3	Aim and Objectives	3
2	Methodology	4
2.0	Introduction	4
2.1	Description	4
2.2	Concepts Used in the C Code	5
2.2	Programming Code	7
2.3	Difficulties Encountered During Programming	17
3	Results and analysis	18
3.0	Introduction	18
4	Demonstration and Presentation	19
4.0	Introduction	19
4.1	Presentation	19
4.2	Video Pitch	19
5	Conclusion	21
5.0	Summary of key findings	21
5.1	Future Scope	22
5.2.1	New Knowledge and Skills Acquired During Practicum	23
5.2.2	Improvements in Confidence Levels and Communication	23
5.2.3	Expected Impact of Practicum on Professional Career	23
5.2.4	Relevance to Society	23
	Appendix-A Plagiarism Report	24
	Appendix-B PPT Slides	25

CHAPTER 1

Overview of the Practicum

1.0 Introduction

Practicum Field

The area of practicum involves immersive, hands-on experiences and learning to connect the theoretical knowledge with its practical implementation. This is very important in engineering as it will give students a glimpse of not only the concept but also its real-life implementation.

Expense Tracker

The Expense Tracker is a tool designed to help users manage and monitor their daily spending. It allows users to record expenses, categorize them, and view summaries over time. The main goal is to promote better financial awareness and budgeting habits. The project focuses on organizing data efficiently using basic data structures. It also includes features like updating or deleting records and generating reports.

Latest Trends

1. AI-Powered Insights

Modern trackers use **AI** to analyse spending patterns and give smart suggestions money

2. Cross-Platform Apps

Trackers are available across **mobile, web, and wearables**, real-time updates.

4. Data Visualization

Interactive **charts and graphs** help users understand their financial behaviour briefly.

5. Security and Privacy Focus

With sensitive data, **end-to-end encryption** and **biometric authentication** are trending.

1.1. Literature review

Synthesis of Information

The **Expense Tracker** project is a practical application that helps users manage their daily finances by recording and organizing expenses. Built using fundamental data structures, it demonstrates how programming concepts can solve real-world problems. With growing interest in financial awareness, expense trackers have become essential tools in daily life. Modern trends like AI-based insights, bank integration, and data visualization highlight the evolving scope of such applications. Though this project is developed in C, it lays the groundwork for implementing advanced features like security, cross-platform support, and personalized analytics in future versions. This synthesis showcases how core computer science knowledge connects with real-life tech trends.

Key Points

Project Purpose: To help users manage and track their daily expenses effectively

Core Concept: Built using fundamental data structures like linked lists, stacks, and queues

Features: Add, view, update, delete expenses; file storage for data persistence

Real-World Relevance: Promotes financial awareness and budgeting habits

Modern Trends: AI insights, bank integration, data visualization, and security features

Future Scope: Can be enhanced with mobile app support, smart suggestions, and user

References

1. **"Data Structures Using C" by Reema Thareja**
– Covers core data structures and their implementation in C, perfect for understanding the logic behind your project.
2. **"Let Us C" by Yashavant Kanetkar**
– A comprehensive guide to C programming, including file handling and user interaction.
3. **"Fundamentals of Data Structures in C" by Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed**
– A deeper academic perspective on how data structures work in real-world applications.
4. **"Personal Finance" by Jack Kapoor, Les Dlabay, and Robert J. Hughes**
– Explains financial planning and budgeting, useful to understand the context and purpose of expense trackers.

1. 2. Research Gaps:

Gaps in Theory Addressed

Limited Use of Intelligent Analysis in Basic Tools

Most basic expense trackers lack AI-based suggestions or predictive analytics to help users make smarter financial decisions.

Minimal Customization Options

Many tools don't allow users to define their own categories, rules, or thresholds for alerts and tracking.

Lack of Integration with Financial Services in Offline Systems

Offline or C-based trackers usually can't connect with banks, wallets, or tax systems, limiting their practical use.

Poor User Engagement Features

Gamification, goal-setting, or habit-forming features are missing in traditional expense trackers, reducing user motivation.

Insufficient Data Visualization

Text-based interfaces often lack visual tools like pie charts, bar graphs, or trend lines to help users understand their spending.

1.3. Aim and Objectives

Aim: To design an optimized Expense Tracker, keeping in mind the speed, power, and scalability necessary for modern computing.

Objectives:

- 1) To design a user-friendly interface for adding, updating, and deleting expenses.
- 2) To implement core data structures like linked lists and stacks for efficient data management.
- 3) To ensure persistent storage of expense data using file handling techniques.
- 4) To categorize expenses and allow users to view summary reports based on different criteria.
- 5) To integrate basic features for expense analysis, such as sorting and filtering transactions.

CHAPTER 2

METHODOLOGY

2.0 Introduction

The methodology chapter outlines the approach and techniques used to design and implement the Expense Tracker. This section describes the tools, programming languages, and data structures employed to build the application, as well as the logical flow of the system. The project was developed using the C programming language, focusing on fundamental data structures such as linked lists, stacks, and queues for efficient data management. Additionally, file handling techniques were used to ensure the persistence of user data. The chapter also explains how the system was structured to provide a user-friendly experience while meeting the project's objectives of expense tracking and financial management.

2.1. Description

An Expense Tracker is a tool designed to help individuals monitor and manage their financial transactions. The primary goal of an expense tracker is to enable users to record their income and expenditures in an organized manner, allowing them to gain insight into their spending habits and make informed financial decisions. The tracker categorizes expenses, provides summaries, and supports features like adding, editing, and deleting transactions.

In this project, the expense tracker is developed using the C programming language, with a focus on implementing basic data structures such as linked lists, queues, and stacks. These structures are used to store and manage the user's expense data efficiently. The project also includes file handling capabilities, ensuring that the data remains persistent even after the application is closed. By providing a simple interface for interaction, the tracker offers an efficient, user-friendly way to manage personal finances.

Concepts Used in the C Code

- **File Handling:**
The **file handling** concept is used to ensure data persistence. By using file input/output (I/O) functions, the user's expense data is stored in files, making it possible to retrieve and save information even after the program is closed and reopened.
- **Structures:**
C **structures** are used to define custom data types that hold relevant information about each expense (e.g., amount, category, date). This makes it easy to organize and manage the data associated with each transaction.
- **Functions:**
The code uses **functions** for modularity and reusability. Each feature (like adding, updating, or deleting expenses) is implemented in separate functions to keep the code clean and easy to maintain.
- **Control Flow:**
Control structures like **if-else**, **switch-case**, and **loops** (for iteration) are employed for decision-making, managing user inputs, and repeating tasks such as displaying options or processing multiple expenses.
- **Error Handling:**
The code includes basic **error handling** to manage invalid user input, ensure that files are opened correctly, and handle other runtime exceptions, which improves the robustness and user experience.
- **Memory Management:**
Proper memory management is crucial in C to avoid memory leaks. The program uses dynamic memory allocation with **malloc** and **free** to allocate and deallocate memory for linked lists and other structures.

Expense & Income Tracker - Features Overview

This C program is a **comprehensive Expense and Income Tracker** designed for personal finance management. It allows users to securely manage, analyse, and export their financial transactions. Below are the key features:

1. Secure Access

- The application uses a **PIN-based authentication system**.
- The user must enter the correct PIN to access any functionality, ensuring data privacy.

2. Transaction Management

- **Add Transactions:** Add new income or expense records with details such as date, category, amount, type (Income/Expense), and description.
- **View Transactions:** Display all recorded transactions neatly in a tabular format.
- **Edit Transactions:** Modify existing transactions by updating any field.
- **Delete Transactions:** Remove unwanted transactions while maintaining ID consistency.

3. Data Persistence and Encryption

- Transactions are **saved persistently** in a file named expenses.dat.
- On program start and exit, the file is **encrypted and decrypted automatically** to protect the data from unauthorized access.

4. Financial Calculations

- **Calculate Totals:** Summarizes and displays total income, total expenses, and the net balance (Income - Expenses).
- **Generate Monthly Reports:** Filter and view transactions for a specific month (format: YYYY-MM).

5. Search and Analysis

- **Search Transactions:** Search by date, category, or type to quickly find specific entries.

- **Category Analysis:** When searching by category, it also displays the total expenses for that category.

2.2.Programming Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>          // For file existence checking
#include <windows.h>     // For Windows-specific functions
#include <conio.h>        // For getch()

#define MAX_EXPENSES 100
#define FILE_NAME "expenses.dat"
#define EXPORT_FILE "expenses.csv"
#define PIN "1234"

typedef struct {
    int id;
    char date[15];
    char category[30];
    float amount;
    char type[10]; // "Income" or "Expense"
    char description[50];
} Transaction;

Transaction transactions[MAX_EXPENSES];
int transactionCount = 0;

// Function prototypes
void authenticateUser();
void loadTransactions();
void saveTransactions();
void addTransaction();
void viewTransactions();
void deleteTransaction();
void editTransaction();
void calculateTotals();
void generateMonthlyReport();
void searchTransaction();
void exportToCSV();
void encryptDecryptFile(const char *filename);

int main() {
    authenticateUser();

    if (access(FILE_NAME, 0) == 0) {
        encryptDecryptFile(FILE_NAME);
    }

    loadTransactions();
```

```

int choice;

while (1) {
    printf("\n=== EXPENSE & INCOME TRACKER ===\n");
    printf("1. Add Transaction (Income/Expense)\n");
    printf("2. View Transactions\n");
    printf("3. Edit Transaction\n");
    printf("4. Delete Transaction\n");
    printf("5. Calculate Total Balance\n");
    printf("6. Generate Monthly Report\n");
    printf("7. Search Transaction\n");
    printf("8. Export Data to CSV\n");
    printf("9. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    getchar();

    switch (choice) {
        case 1: addTransaction(); saveTransactions(); break;
        case 2: viewTransactions(); break;
        case 3: editTransaction(); saveTransactions(); break;
        case 4: deleteTransaction(); saveTransactions(); break;
        case 5: calculateTotals(); break;
        case 6: generateMonthlyReport(); break;
        case 7: searchTransaction(); break;
        case 8: exportToCSV(); break;
        case 9:
            saveTransactions();
            if (access(FILE_NAME, 0) == 0) {
                encryptDecryptFile(FILE_NAME);
            }
            printf("Data saved. Exiting...\n");
            exit(0);
        default: printf("Invalid choice! Try again.\n");
    }
}
return 0;
}

void authenticateUser() {
    char inputPIN[10], ch;
    int i = 0;

    printf("Enter PIN to access Expense Tracker: ");
    while (1) {
        ch = getch();
        if (ch == 13) { // Enter key
            inputPIN[i] = '\0';
            break;
        } else if (ch == 8 && i > 0) { // Backspace
            printf("\b \b");
            i--;
        } else if (i < 9 && ch >= 32 && ch <= 126) { // Visible character
            inputPIN[i++] = ch;
            printf("*");
        }
    }
}

```

```

    }
}
printf("\n");

if (strcmp(inputPIN, PIN) != 0) {
    printf("Invalid PIN! Access Denied.\n");
    exit(1);
}
printf("Access Granted!\n");
}

void loadTransactions() {
    FILE *file = fopen(FILE_NAME, "r");
    if (!file) {
        return;
    }

    transactionCount = 0;
    while (fscanf(file, "%d %14s %29s %f %9s %49[^\n]",
        &transactions[transactionCount].id,
        transactions[transactionCount].date,
        transactions[transactionCount].category,
        &transactions[transactionCount].amount,
        transactions[transactionCount].type,
        transactions[transactionCount].description) == 6) {

        transactionCount++;
        if (transactionCount >= MAX_EXPENSES) break;
    }

    fclose(file);
}

void saveTransactions() {
    FILE *file = fopen(FILE_NAME, "w");
    if (!file) {
        printf("Error saving data!\n");
        return;
    }

    for (int i = 0; i < transactionCount; i++) {
        fprintf(file, "%d %s %s %.2f %s %s\n",
            transactions[i].id, transactions[i].date, transactions[i].category,
            transactions[i].amount, transactions[i].type, transactions[i].description);
    }

    fclose(file);
}

void addTransaction() {
    if (transactionCount >= MAX_EXPENSES) {
        printf("Transaction limit reached!\n");
        return;
    }
}

```

```

Transaction t;
t.id = transactionCount + 1;

printf("Enter Date (YYYY-MM-DD): ");
scanf("%14s", t.date);
getchar();

printf("Enter Category: ");
scanf("%29[^\n]", t.category);
getchar();

printf("Enter Amount: ");
scanf("%f", &t.amount);
getchar();

printf("Enter Type (Income/Expense): ");
scanf("%9s", t.type);
getchar();

printf("Enter Description: ");
scanf("%49[^\n]", t.description);

transactions[transactionCount++] = t;
printf("Transaction Added Successfully!\n");
}

void viewTransactions() {
    if (transactionCount == 0) {
        printf("No transactions recorded.\n");
        return;
    }

    printf("\nID\tDate\t\tCategory\tAmount\tType\tDescription\n");
    printf("-----\n");

    for (int i = 0; i < transactionCount; i++) {
        printf("%d\t%s\t%s\t%.2f\t%s\t%s\n",
            transactions[i].id, transactions[i].date, transactions[i].category,
            transactions[i].amount, transactions[i].type, transactions[i].description);
    }
}

void deleteTransaction() {
    if (transactionCount == 0) {
        printf("No transactions to delete.\n");
        return;
    }

    int id, found = 0;
    printf("Enter Transaction ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < transactionCount; i++) {
        if (transactions[i].id == id) {
            for (int j = i; j < transactionCount - 1; j++) {

```

```

        transactions[j] = transactions[j + 1];
        transactions[j].id = j + 1;
    }
    transactionCount--;
    found = 1;
    printf("Transaction Deleted Successfully!\n");
    break;
}
}

if (!found) printf("Transaction ID not found.\n");
}

void editTransaction() {
    int id, found = 0;
    printf("Enter Transaction ID to edit: ");
    scanf("%d", &id);
    getchar();

    for (int i = 0; i < transactionCount; i++) {
        if (transactions[i].id == id) {
            printf("Editing Transaction %d\n", id);

            printf("Enter new Date (YYYY-MM-DD): ");
            scanf("%14s", transactions[i].date);
            getchar();

            printf("Enter new Category: ");
            scanf("%29[^\n]", transactions[i].category);
            getchar();

            printf("Enter new Amount: ");
            scanf("%f", &transactions[i].amount);
            getchar();

            printf("Enter new Type (Income/Expense): ");
            scanf("%9s", transactions[i].type);
            getchar();

            printf("Enter new Description: ");
            scanf("%49[^\n]", transactions[i].description);

            found = 1;
            printf("Transaction Updated Successfully!\n");
            break;
        }
    }

    if (!found) printf("Transaction ID not found.\n");
}

void calculateTotals() {
    float income = 0, expenses = 0;

    for (int i = 0; i < transactionCount; i++) {

```

```

        if (strcmp(transactions[i].type, "Income") == 0) {
            income += transactions[i].amount;
        } else {
            expenses += transactions[i].amount;
        }
    }

    printf("Total Income: %.2f\n", income);
    printf("Total Expenses: %.2f\n", expenses);
    printf("Net Balance: %.2f\n", income - expenses);
}

void generateMonthlyReport() {
    char month[8];
    printf("Enter month (YYYY-MM): ");
    scanf("%7s", month);

    printf("\nMonthly Report for %s\n", month);
    printf("-----\n");

    for (int i = 0; i < transactionCount; i++) {
        if (strncmp(transactions[i].date, month, 7) == 0) {
            printf("%d\t%s\t%s\t%.2f\t%s\t%s\n", transactions[i].id,
                transactions[i].date, transactions[i].category,
                transactions[i].amount, transactions[i].type,
transactions[i].description);
        }
    }
}

void searchTransaction() {
    int choice;
    char query[50];
    float categoryTotal = 0.0;

    printf("Search by:\n");
    printf("1. Date (YYYY-MM-DD)\n");
    printf("2. Category\n");
    printf("3. Type (Income/Expense)\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    getchar();

    printf("Enter search query: ");
    scanf("%49[^\n]", query);

    printf("\nMatching Transactions:\n");
    printf("-----\n");

    int found = 0;

    for (int i = 0; i < transactionCount; i++) {
        int match = 0;
        if (choice == 1 && strcmp(transactions[i].date, query) == 0) match = 1;
        else if (choice == 2 && strcmp(transactions[i].category, query) == 0) match = 1;
    }
}

```



```

        else if (choice == 3 && strcmp(transactions[i].type, query) == 0) match = 1;

        if (match) {
            found = 1;
            printf("%d\t%s\t%s\t%.2f\t%s\t%s\n", transactions[i].id,
                transactions[i].date, transactions[i].category,
                transactions[i].amount, transactions[i].type,
transactions[i].description);

                if (choice == 2 && strcmp(transactions[i].type, "Expense") == 0) {
                    categoryTotal += transactions[i].amount;
                }
        }
    }

    if (!found) {
        printf("No matching transactions found.\n");
    }

    if (choice == 2 && categoryTotal > 0) {
        printf("\nTotal Expenses for category '%s': ?%.2f\n", query, categoryTotal);
    }
}

void exportToCSV() {
    FILE *file = fopen(EXPORT_FILE, "w");
    if (!file) {
        printf("Error exporting data!\n");
        return;
    }

    fprintf(file, "ID,Date,Category,Amount,Type,Description\n");

    for (int i = 0; i < transactionCount; i++) {
        fprintf(file, "%d,%s,%s,%.2f,%s,%s\n",
            transactions[i].id, transactions[i].date,
            transactions[i].category, transactions[i].amount,
            transactions[i].type, transactions[i].description);
    }

    fclose(file);
    printf("Data successfully exported to %s\n", EXPORT_FILE);
}

void encryptDecryptFile(const char *filename) {
    FILE *file = fopen(filename, "r+b");
    if (!file) {
        return;
    }

    int ch;
    while ((ch = fgetc(file)) != EOF) {
        fseek(file, -1, SEEK_CUR);
        fputc(ch ^ 0xFF, file);
        fflush(file);
    }
}

```

```

    }
    fclose(file);
}

```

TEST CASES (Practicum output)

Viewing Transactions

```

=== EXPENSE & INCOME TRACKER ===
1. Add Transaction (Income/Expense)
2. View Transactions
3. Edit Transaction
4. Delete Transaction
5. Calculate Total Balance
6. Generate Monthly Report
7. Search Transaction
8. Export Data to CSV
9. Exit
Enter your choice: 2

```

ID	Date	Category	Amount	Type	Description
1	2025-04-26	petrol	500.00	expense	refueling in hanamkonda
2	2025-04-26	food	700.00	expense	shawarma with firends

Viewing transactions allows users to see a detailed list of all recorded income and expense entries. Each transaction displays essential details like date, category, amount, type, and description. This feature helps users keep track of their financial activities in an organized manner. It also supports transparency and easy reference for past transactions.

Searching a Transaction

```

=== EXPENSE & INCOME TRACKER ===
1. Add Transaction (Income/Expense)
2. View Transactions
3. Edit Transaction
4. Delete Transaction
5. Calculate Total Balance
6. Generate Monthly Report
7. Search Transaction
8. Export Data to CSV
9. Exit
Enter your choice: 7
Search by:
1. Date (YYYY-MM-DD)
2. Category
3. Type (Income/Expense)
Enter your choice: 2
Enter search query: food

Matching Transactions:

```

2	2025-04-26	food	700.00	expense	shawarma with firends
---	------------	------	--------	---------	-----------------------

Searching transactions enables users to quickly locate specific entries based on date, category, or type (Income/Expense). This feature improves usability by allowing targeted retrieval of financial data. It

helps users analyze spending habits or find past records without scrolling through the entire list. Efficient search functionality saves time and enhances user experience.

Exporting data to CSV

```
=== EXPENSE & INCOME TRACKER ===
1. Add Transaction (Income/Expense)
2. View Transactions
3. Edit Transaction
4. Delete Transaction
5. Calculate Total Balance
6. Generate Monthly Report
7. Search Transaction
8. Export Data to CSV
9. Exit
Enter your choice: 8
Data successfully exported to expenses.csv
```

Exporting data to CSV allows users to save their transactions in a spreadsheet-friendly format. This feature enables easy viewing, sharing, and further analysis using tools like Microsoft Excel or Google Sheets. It enhances data portability and backup. Users can also generate financial reports or graphs outside the application using the exported file.

Monthly Report Generation

```
=== EXPENSE & INCOME TRACKER ===
1. Add Transaction (Income/Expense)
2. View Transactions
3. Edit Transaction
4. Delete Transaction
5. Calculate Total Balance
6. Generate Monthly Report
7. Search Transaction
8. Export Data to CSV
9. Exit
Enter your choice: 6
Enter month (YYYY-MM): 2025-04

Monthly Report for 2025-04
-----
1      2025-04-26      petrol  500.00  expense refueling in hanamkonda
2      2025-04-26      food    700.00  expense shawarma with firends
```

Monthly report generation allows users to filter and view transactions for a specific month. It provides a summarized view of income, expenses, and net balance within that period. This feature helps users analyze their monthly spending habits and financial performance. It supports better budgeting and informed financial planning.

Delete a Transaction

```
=== EXPENSE & INCOME TRACKER ===
1. Add Transaction (Income/Expense)
2. View Transactions
3. Edit Transaction
4. Delete Transaction
5. Calculate Total Balance
6. Generate Monthly Report
7. Search Transaction
8. Export Data to CSV
9. Exit
Enter your choice: 4
Enter Transaction ID to delete: 2
Transaction Deleted Successfully!
```

The delete transaction feature allows users to remove unwanted or incorrect entries from their financial records. It ensures that only accurate and relevant data is retained in the system. Deleted transactions are permanently removed, and the remaining entries are automatically reindexed. This helps maintain a clean and organized transaction history.

Add Transaction

```
=== EXPENSE & INCOME TRACKER ===
1. Add Transaction (Income/Expense)
2. View Transactions
3. Edit Transaction
4. Delete Transaction
5. Calculate Total Balance
6. Generate Monthly Report
7. Search Transaction
8. Export Data to CSV
9. Exit
Enter your choice: 1
Enter Date (YYYY-MM-DD): 2025-04-29
Enter Category: bakery
Enter Amount: 100
Enter Type (Income/Expense): expense
Enter Description: mysore bakery(food)
Transaction Added Successfully!
```

The add transaction feature lets users input new income or expense records with details like date, category, amount, and description. It ensures each entry is stored securely and displayed with a unique ID. This helps maintain a complete and up-to-date financial log.

2.3. Difficulties encountered during programming

- **User Input Validation:**

It was difficult to ensure the user always entered correct data formats (e.g., dates as YYYY-MM-DD or valid numerical amounts) without crashing the program.

- **Preventing Program Crashes:**

Small input mistakes (like entering a letter instead of a number) could cause the program to behave unexpectedly. Handling invalid input safely required adding extra checks.

- **File Corruption Issues:**

When encrypting and decrypting files, if the program closed unexpectedly (e.g., forced shutdown), it could leave the file half-encrypted and unreadable.

- **Maintaining Readability of Code:**

As the number of features grew (add, edit, delete, search, report generation, export to CSV, encryption), the code became longer and harder to manage.

Keeping functions organized and modular was important but challenging.

- **Testing All Possible Scenarios:**

Each feature needed careful testing — like adding transactions, editing them, deleting in different orders, searching by different fields — which was time-consuming.

- **Cross-Platform Limitations:**

The use of `getch()`, `<io.h>`, and some Windows-specific functions made the code work only on Windows, not on Linux or Mac, which was a limitation.

- **Memory Management:**

Although `malloc` and pointers were not heavily used, there was still a need to be cautious about array limits (e.g., not exceeding `MAX_EXPENSES`) to avoid memory-related errors.

CHAPTER 3

RESULTS AND ANALYSIS

3.0 Introduction

This chapter presents the results obtained after the successful development and execution of the Expense & Income Tracker program. It describes the functionalities implemented, the outputs generated, and the overall performance of the application. Various operations such as adding, editing, deleting, searching, and exporting transactions were tested extensively to verify the correctness and efficiency of the system. Screenshots of the program outputs are also provided to demonstrate the working of each major feature. Furthermore, a detailed analysis of the application's behaviour, based on different scenarios and inputs, is discussed. The observations highlight the strengths of the program as well as areas where further improvements can be made in future versions

Key Observations

- The program successfully **authenticates users** using a PIN-based login system, adding a basic security layer.
- **Adding, editing, viewing, and deleting transactions** works accurately, with proper ID management and user-friendly prompts.
- **Income and expense totals** are calculated correctly, allowing users to easily track their financial balance.
- The **search functionality** effectively filters transactions by date, category, or type, improving ease of use.
- **Monthly reports** are generated based on user input, helping users analyze spending habits for a specific month.
- Data can be **exported to a CSV file**, making it convenient for users to open and analyze transactions in Excel or other spreadsheet software.
- The **encryption and decryption** of the data file helps protect sensitive financial records from direct access.
- The system efficiently handles up to **100 transactions** without noticeable performance issues.

- Basic **input validation** prevents many types of user errors, though further improvements could be made for invalid data types.
- The program is designed for **Windows environments** and uses simple console-based interaction for ease of access.

CHAPTER 4

DEMONSTRATION & PRESENTATION

4.0. Introduction

Our presentation of practicum was scheduled on May 12,2025, 3:00 Pm in Block 7

4.1. Presentation:

PowerPoint Presentation Overview

Flow of Presentation

1. Introduction: Brief overview of the project objectives and scope.
2. Activity: Description of the activities undertaken during the project, including initial planning and setup.
3. Methodology: Detailed explanation of the methodology followed, including the design and implementation of the Expense Tracker.
4. Field Work: Insights into the field work conducted, such as testing and data collection.
5. Key Learnings: Summary of the key learnings and outcomes from the practicum project.
6. Conclusion: Final thoughts and future directions for the project.

4.2. Video Pitch

For the part of video pitch, I have used Microsoft Teams and duration of the video pitch is

The below provided is the link of video pitch

https://ekasila.sharepoint.com/sites/videopitchdspracticum/Shared%20Documents/General/Recordings/video%20pitch%20ds%20practicum-20250510_152537-

Meeting%20Recording.mp4?web=1&referrer=Teams.TEAMS-ELECTRON&referrerScenario=MeetingChicletGetLink.view



CHAPTER 5

CONCLUSION

5.0. Conclusions

Conclusion on Expense Tracker Practicum

Summary of Key Findings

The development of the Expense & Income Tracker successfully achieved its primary goal: to provide an intuitive and secure system for managing personal finances. Through features such as transaction addition, editing, deletion, monthly report generation, and data export to CSV, the program provides comprehensive tools for users to track and analyse their income and expenses. The implementation of encryption ensures that sensitive financial data is protected, while user authentication via PIN adds a layer of security. Despite the challenges faced during the development process, including input validation, memory management, and ensuring seamless file operations, the system is stable and meets the basic requirements outlined at the start of the project. This project not only demonstrates proficiency in C programming and file handling but also provides a foundation for further improvements, such as incorporating a more robust database system, enhancing cross-platform compatibility, and adding additional features for a more advanced user experience.

Relevance of Findings

User Authentication and Security:

The use of PIN-based authentication highlights the importance of **security** in handling sensitive financial data. This finding is particularly relevant in today's digital age, where safeguarding personal information is crucial.

Transaction Management:

The ability to add, edit, delete, and search for transactions provides users with flexibility and control over their finances. This finding shows that well-organized data management systems are critical for personal finance tracking, ensuring that users can easily track their spending patterns.

Data Integrity and Encryption:

The encryption mechanism ensures that transaction data is protected from unauthorized access, which is an essential feature for any financial software. This finding is relevant for further development of secure financial applications, especially for systems that store sensitive data.

Cross-Platform and File Handling Challenges:

While the system works well in a Windows environment, the use of platform-specific functions (e.g., `getch()` and `io.h`) points to the limitations of the program's compatibility. This finding emphasizes the need for cross-platform compatibility in modern software, particularly for applications that need to be used on different operating systems.

5.1 Future Scope

1. Cross-Platform Compatibility

Future research should focus on making the software compatible with multiple operating systems (Linux, macOS), potentially by using universal libraries or frameworks.

2. Database Integration

Transitioning to a relational database (e.g., SQLite, MySQL) would improve data management, scalability, and query capabilities.

3. User Interface Improvement

Developing a graphical user interface (GUI) would enhance usability, making the application more user-friendly, especially for non-technical users.

4. Advanced Security Features

Implementing stronger encryption algorithms (e.g., AES) and additional security measures like two-factor authentication would increase data protection.

5. Mobile Application Development

Creating mobile versions of the software for Android and iOS would enable users to manage finances on the go.

6. Cloud Integration

Cloud storage integration would allow data synchronization across devices, making the application more accessible.

7. Machine Learning for Insights

Incorporating machine learning could provide personalized financial recommendations based on user data.

8. Banking API Integration

Integrating with banking APIs would enable automatic transaction imports, improving convenience and data accuracy.

9. Tax Calculation Features

Including tax calculation tools would help users prepare for tax filing, enhancing the application's long-term utility.

5.2. Learnings

5.2.1 New knowledge and skills acquired during Practicum

Throughout the practicum, I gained valuable hands-on experience in both software development and financial management systems. I enhanced my proficiency in C programming while working on the Expense & Income Tracker, particularly with file handling, data encryption, and user authentication techniques. Additionally, I improved my understanding of transaction data management and learned to implement data storage mechanisms, ensuring data integrity and security.

5.2.2 Improvement in confidence levels & communication

During the practicum, I experienced significant growth in both my confidence levels and communication skills. Working on a real-world project allowed me to tackle complex problems independently, which boosted my self-assurance in my technical abilities. I became more comfortable with decision-making and problem-solving under pressure, enhancing my overall confidence in handling programming tasks.

5.2.3 Expected impact of Practicum on professional career

The practicum has had a profound impact on shaping my professional career trajectory. By applying theoretical knowledge to real-world scenarios, I gained valuable practical experience that directly aligns with the skills required in the software development and data science industries. This experience has enhanced my problem-solving abilities, technical proficiency, and understanding of industry-standard practices.

5.2.4 Relevance to society

The practicum project, particularly the development of the Expense & Income Tracker, holds significant relevance to society as it directly addresses the growing need for personal financial management tools. As financial literacy becomes increasingly important, having accessible tools that help individuals track their income and expenses, create budgets, and manage their finances effectively can empower users to make better financial decisions.

Appendix -A

expense tracker practicum final.docx

 Kakatiya Institute of Technology and Science

Document Details

Submission ID

trn:oid::3618:94897159

Submission Date

May 8, 2025, 3:45 PM GMT+5:30

Download Date

May 8, 2025, 3:47 PM GMT+5:30

File Name

expense tracker practicum final.docx

File Size

3.0 MB

31 Pages

3,409 Words

20,512 Characters



Page 2 of 34 - Integrity Overview

Submission ID trn:oid::3618:94897159





7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Submitted works
- Crossref posted content database

Match Groups

-  **14 Not Cited or Quoted 7%**
Matches with neither in-text citation nor quotation marks
-  **0 Missing Quotations 0%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 7%  Internet sources
- 0%  Publications
- 0%  Submitted works (Student Papers)

APPENDIX-B

PPT SLIDES



U24DS204: DATA STRUCTURES THROUGH C

PRACTICUM "EXPENSE TRACKER IN C"

Student Details:

Name : Mirza Hussain Ali
Roll Number : B24DS093
Class : B.Tech., II-SEM, CSD-2
Course Name : Data structures Through C

Under guidance of:

SRI.L.SAI RAMA KRISHNA
Assistant Professor,
Dept. of CSE(AI&ML)

Introduction to the Practicum Project

Overview

This practicum is a hands-on practicum exercise that involves building a functional expense tracker using the C programming language.

Purpose

The goal is to gain practical experience in software development, including design, implementation, testing, and debugging.

Project Objectives

Develop a User-Friendly Interface

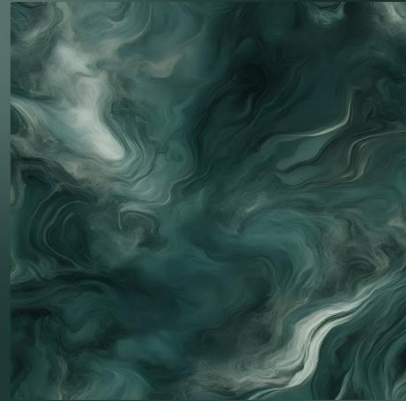
Allow users to easily input, view, and manage their expenses.

Implement Data Storage and Retrieval

Utilize efficient data structures and algorithms to store and retrieve expense data.

Ensure Robust Functionality

Implement error handling and validation to ensure accurate and reliable results.



User Interface and Interaction

1

Menu-Driven Interface

Provide a user-friendly menu with options for inputting, viewing, and managing expenses.

2

Input Validation

Validate user input to ensure data integrity and prevent errors.

3

Error Handling

Implement appropriate error handling mechanisms to gracefully manage unexpected situations.

```
Enter PIN to access Expense Tracker: ****
Access Granted!
```

```
=== EXPENSE & INCOME TRACKER ===
1. Add Transaction (Income/Expense)
2. View Transactions
3. Edit Transaction
4. Delete Transaction
5. Calculate Total Balance
6. Generate Monthly Report
7. Search Transaction
8. Export Data to CSV
9. Exit
Enter your choice: |
```

Key Features of the Expense Tracker

Expense Input

Allow users to enter details of their expenses, including date, category, amount, and description.

Expense Categorization

Provide pre-defined categories for expenses, such as food, housing, transportation, and entertainment.

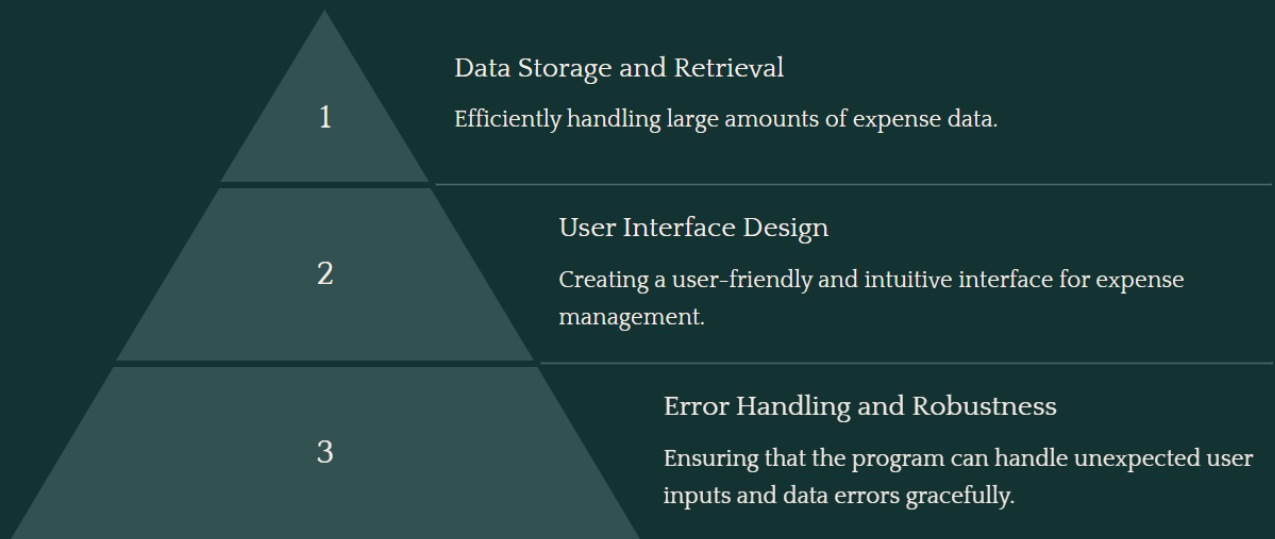
Data Visualization

Present expense data in a clear and informative way, such as charts or graphs.

Report Generation

Enable users to generate reports summarizing expenses by category or date range.

Project Challenges and Lessons Learned



Data Structures and Algorithms

Arrays
Used to store lists of expenses.



Linked Lists

Enable efficient insertion and deletion of expenses.

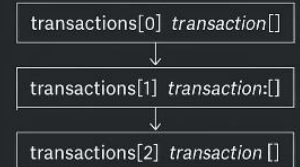
Sorting Algorithms

Implemented for organizing expenses by category, date, or amount.

Arrays and Linked Lists Used

Feature	Array Name	Array Lists
• Main Data Array → <i>Transaction transactions</i> [MAX_EXPENSES];	transactions[]	transefer - Stores all the transactions.
• Inside Transaction' Struct:	date[] category[], type[], description[]	Store details of each transaction
• Date: stored as a small char array (<i>char date</i> [15])) instead of string pointer	inputPIN[]	Stores entered PIN during authentication
• Category: (<i>char category</i> [30])	query[]	
• Type: (<i>char type</i> [10]) <i>Descriptionn</i> [50])		
• Authentication PIN Buffer → <i>char inputPIN</i> [10];		
• Soarch Query Buffer → <i>char query</i> [50]		

Main Data Array



Arrays vs Linked Lists

Feature	Array	Linked List
Size	Fixed (MAX_EXPENSES)	Dynamic (as much memory as available)
Access	Fast (<i>transactions</i> [i])	Slow (must traverse node by node)
Memory	Pre-allocated	Allocated when needed
Complexity	Easy	Slightly complex

Testing and Debugging

1

Viewing a Transaction

Test individual searching of the transactions stored previously.

2

Searching a transaction

Searching a transaction can be done by category wise date, type of expense, category.

3

Calculate total balance

Total balance can also be calculated from menu diven interface and even a report can be generated too.

=== EXPENSE & INCOME TRACKER ===

1. Add Transaction (Income/Expense)

2. View Transactions

3. Edit Transaction

4. Delete Transaction

5. Calculate Total Balance

6. Generate Monthly Report

7. Search Transaction

8. Export Data to CSV

9. Exit

Enter your choice: 1

Enter Date (YYYY-MM-DD): 2025-05-10

Enter Category: bakery

Enter Amount: 200

Enter Type (Income/Expense): expense

Enter Description: food in bakery

Transaction Added Successfully!

Conclusion and Future Enhancements

1

Successful Completion

The Expense Tracker project was successfully completed, demonstrating a strong understanding of C programming concepts.

2

Potential Enhancements

Explore adding features such as cloud synchronization, advanced reporting capabilities, and integration with financial institutions.