Personal Finance Management System

ITHIHAS.K.S

Advisor: Mr. Sahabzada Betab Badar

Department of Computer Science and Information Technology
Jain (Deemed-to-be) University

This report is submitted for the course of
Programming in C (24BCA1C04)

Jain University, Bengaluru **December 2024**

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this report entitled "Personal Finance Managment System" **are original and have not been submitted in whole or in part for consideration for any other degree or qualification or course in this or any other university. This report is the collective work of me and my team.**

ITHIHAS.K.S
USN: 24BCAR0072
Department of Computer Science and Information Technology,
Jain (Deemed-to-be) University, Bengaluru

Acknowledgement

I acknowledge my advisor and teacher Mr. Sahabzada Betab Badar's invaluable guidance and support throughout this project. My gratitude for his mentorship extends to all the Department of Computer Science and Information Technology faculty at Jain (Deemed-to-be) University, Bengaluru. Additionally, I am grateful to my friends, peers and team members for their constructive feedback and collaborative spirit during the brainstorming sessions. Their inputs have helped refine my ideas and improve the overall quality of this work. Finally, I also acknowledge the support of my family, friends, peers and team members whose encouragement and understanding has enabled me to focus on this project with dedication. This project would not have been possible without their support. Thank you to everyone who contributed to the success of this project.

Abstract

The Personal Finance Management System is a console-based personal finance management software application developed in C programming language, whose basic purpose is to enable business transactions efficiently to maintain the record of the personal transaction. A few functions which this system allows its users to do include adding, viewing, and deleting transactions, setting budgets for different spending categories, and summarizing financial data. This program leads to financial awareness in that it allows the user to keep track of income and expenses, analyze spending patterns, and provide an effective means of budget management. The main features are recording transactions based on type (income or expense) and category (for example, food, rent, salary), calculating total income and expenses, and net saving. Also, the user could set up budget limits for specific categories and view their spent-to-budget comparison, which helps identify the areas with overspending. It's an important implementation of the most important concepts from core C programming such as arrays, functions, looping, and conditional statements. Since random number generation has not been applied anywhere in the project, all outputs shall remain deterministic and entirely dependent upon user inputs. In order to enhance user friendliness, error handling routines are incorporated to validate inputs and prevent invalid operations like overspending. The project hence gives utmost importance to simplicity, functionality, and eligibility. It raises practical programming skills since it serves an operational base. Those features are things like data storage, graphical user interfaces, or interfacing with databases. The project shows how general concepts of programming can be used to develop a functional interactive application for personal finance management.

Table of Contents

List of Figures

List of Tables

1. Introduction

Engineering, the program provides students with an extremely powerful reconciliation tool that enables one to combine evaluation, invite-informs, deter foray, and monitoring for the time being. Exceeding ease of use, simple operations extend user control of financial life by providing imaginary finances and possibly effective planning associated with it. This includes such features as transaction records (both income and expenses), transaction history view, budget print setting for a specific category, and financial summary generation. All these features offer a complete overview of the user regarding his financial health, enabling him to make enemy-smart decisions about financing based on it. It features CLI-based interfacing to the user, which assures an elemental level of interaction with non-computer trained individuals. The essence of the **Personal Finance Management System**, thus, is management, efficiency, and awareness. The application receives logical workflows, thus processing the user's input through which the user may track his spending, monitor his saving, and compare real expenses with defined budgets. The program reinterpretation avoids the difficulty of graphical interface systems; its fully text-based implant is labeled lightweight and accessible by the genre many users belong to. Technically speaking, the project continuously shows how core programming concepts could be applied in the world of data or operational research. On the other hand, normally, you validate all the possible wrong or unacceptable entries so that it responds with proper statements back and ensures maximum reliability. The Personal finance management system is used not only for practical purposes, but also for doing front line window back educational preparation. It gives a strong basis for all that you can further develop, namely, persistence-oriented strategy in future education; not talking about all the various ways of developing graphic interfaces, problem solving in an excellent way. By bridging the gap between theoretical learning and practical implementation, this project highlights the power of programming to create tools that positively impact everyday life.

1.1 Objectives

Primary aim of the Personal Finance Management System is to impart a simple, effective, and interactive tool, so help users manage their personal finances with efficiency. The goal here is to empower users with a simple platform that will enable income and expense tracking, budgeting, and financial health analysis, thereby providing greater financial awareness and accountability. The program helps the user identify spending patterns, compare actual expenses with budgeted figures, and make decisions regarding finances by categorizing financial data into segments and summing up transactions. The system further serves as practical experience of base programming concepts such as modular design, array handling, and input validation, in order to solidify structured processes of problem-solving and logical deduction.The usability and robustness have been prioritised for the project through a menu-driven interface and error handling, to guide the user smoothly through the use of the application. Users shall be able to add, view, and manage transactions, or alternatively change the budget multiple times, until they decide to exit. This thereby provides for constant subtle engagement and flexibility in managing finances.Other than the practical gains to be derived from this project, this experience is educational in the usages of basic programming notions exemplified in the form of loops, conditionals, and functions, to practical working solutions upon real-world scenarios. This joining of theory with practice is what the project aims to achieve: that demonstrate how versatile programming is in shaping the building of meaningful and touching solutions.The Personal Finance Management System combines, in the increasingly emerging digital universe, usefulness, simplicity, and education in the creation of a comprehensive stage for users to manage their personal finances as a proper example in developing interactive apps from programming.

1.2 Features

1.2.1 Add Transaction

The user can classify transactions as income or expense and input details like the category, amount, and description. These records would be stored in an organized way for future reference.

1.2.2 View Transactions

Allows the users to see a detailed history of their recorded transaction with relevant category, type (Income/Expense), along with the amounts and providing a clear overview of their financial activity.

1.2.3 Delete Transactions

Allows users to delete specific transactions from the records to fix mistakes or accurately manage them.

1.2.4 Generate Summary

It calculates and presents a summary of the total income, total expenses, and net savings so that users can at a glance have an understanding of their financial standing.

1.2.5 Set Budget

Enables the setting of budget limits for specified categories (for example, food, transportation, entertainment), thereby enabling a user to control his or her expenditure within financial goals defined by themself.

1.2.6 View Budget Status

Indicates the amount presently being spent for each category and relates actual expenses spent against pre-allocated funds of the user so potential areas can be identified that can be in need of better financial management.

This project exemplifies how basic programming techniques like functions, arrays, and control structures can be used in the development of an uncomplicated and engaging financial management application. Simplicity and practicality are of primary importance;

the project demonstrates some power of programming to resolve all kinds of daily challenges while providing the basis for further enhancement upon enhancements-like data persistence or graphical interfaces.

1.3 Contribution

As a team member of the Personal Finance Management System project, I primarily worked on designing and implementing the View Budget Status feature and building the main function to integrate the entire program's functionality.

View Budget Status

In developing the View Budget Status feature, I designed a logical framework to calculate and display the user's current spending position across several budget categories. It compares actual expenses against the previously set budgets, helping thereby identify areas where overspending went on. This required careful implementation of array handling and calculation logic to yield correct results.Additionally, in order to sort out user experience, I introduced a simple but easy-to-read output, giving a summary of budget categories, with their set limits, and showing the remaining budget or overspent amount. I also created fallbacks for edge cases; for instance, if no budget or transactions were given, ensuring the feature works under all circumstances. This module goes a long way to contributing to the great financial awareness aspect for the system by allowing users to visualize their spending behavior in real-time.

Main Function

I coordinated all the features of the program into a single coherent, menu-driven system. I structured the navigation flow so that it was straightforward and intuitive, allowing users to easily access all functionalities such as adding transactions, setting budgets, viewing summaries, and managing transactions.During designing the main function, I incorporated input validation logic to handle invalid selections from the menu and bad inputs from the user and hence kept the program robust and user-friendly. The execution was greatly improved in terms of flow, eliminating redundancy and ensuring the different program modules are executed efficiently.By structuring the main function properly, I ensured that the smooth interaction of different parts provides users with a seamless experience. This effort has given emphasis to the well-done modularity, interface design, and control flow as of importance in making applications reliable, accessible, and intuitive. In the development of the View Budget Status feature and the main function, my efforts and contribution enhanced the usability and functionality of the program. These elements

are vital in accomplishing the system's goal of providing a user with an effective interactive tool to run their personal finances.

2. Why the Chosen Approach

2.1 Justification of Features

The features of a Personal Finance Management System are chosen based on the relevance they bear on everyday financial requirements and their ability to showcase programming concepts.

Add Transaction: This feature allows the user to record transactions, whether incomes or expenses, indicating the category and amount. The implementation indicates aspects of data entry and array management-the core of the financial tracking system.

View Transaction: Checking past transactions is extremely important in financial management. The feature is a good illustration storing data structures and how this information can be accessed through programming.

Set Budget: Budgeting is the backbone of financial control. The introduction of spending limits under this feature begins the concept of managing dynamic data and comparing logic to impose constraints defined by the user.

View Budget Status: This enables a user to establish a comparison between his/her spending and the budget. Data is handled in an array, arithmetic operations, and output formatting is done for the understanding of the financial status.

Generate Summary: Viewing an overview of total income, expenses, and savings assists the user in determining the overall sense of their finances in one overview. The feature presents a case of the aggregation of data and logical computation.

The whole combination allows for comprehensive, functional financial management software. Each part fills a real-world need while it phonologically extends a showcase of key programming concepts that are interesting and user-friendly.

2.2 Programming Techniques and Tools

I. Data storage and retrieval.

Catch phrases and budgets are assigned to arrays based on data hierarchies that permit a good organization and retrieval of finance records. [1] [2]

II. Functions.

The application is organized as a set of modules, each of which is in function for providing the basic functionality of actions, such as adding transactions, setting budgets, and printing summaries. This makes the code more readable, maintainable, and reusable.

III. Input validation.

The user input validation checks for the outlet of the overflow and underflow in order to handle the erroneous entries in a graceful manner, which considerably improves the robustness of the programs while reducing effort inflicted upon the users in line with maintaining it in the appropriate manner. [1] [2]

IV. Control structures.

While loops are employed to continue interaction with a user, and conditional statements are invoked in order to manage the flow of control based on the user choices and the program logic itself.

V. User input and output.

Interactive facilities implemented through scanf and printf allow very, very friendly input and presentation of results, even apples-to-apples comparison of results one might expect without having to be overtly computer friendly.

VI. Menu-driven interfaces.

The straightforward application of a menu structure allows the user to interact through a standard and intuitive approach while justifying usability.

VII. Arithmetic and logical operations.

Budget comparisons required arithmetic operations for comparison of totals and net savings calculations, which demonstrate the application of logic to solving real-life problems. [1] [2]

VIII. Efficiency and resource management.

The program provides efficient management of resources, such as transaction records and budget values, availing real-life principles in regards to financial tracking. [1] [2]

IX. Scalability and extensibility.

The modular design of the program is an asset in terms of incorporating more features into the future.

X. Commenting and documentation.

Some sections are put within comments to explain logic and functionality, thus continuing best software development practices and ease of maintenance.

XI. Educational application.

The project serves as an educational tool that demonstrates basic programming concepts like modularity, control flow, and data handling, and tackles real-life fin

3. Code Components

Personal Finance Management System is a program designed and built on best practices for modular programming, user interaction, and efficient data management in C while providing real-world solutions to financial tracking requirements. This section gives an elaborate rundown of the various modules constituting the program, bringing to light how the combination of different parts of the code would present a workable, user-centered financial management tool. Utilizing core techniques such as array and control flow manipulation and arithmetical operations, the project allows one to capture transactions, set budgets, and keep track of their financial well-being.

Each of the program's components is designed to handle some aspects of financial management-from recording and classifying transactions, through the analysis of budgets, to savings. The modular programming guarantees that each function is self-contained and allows readability, maintainability, and scalability. A simple menu-driven interface facilitates the transition for the user to add transactions, set budgets, view summaries, or exit the application with ease. Wherever appropriate, input validation checks are introduced throughout the system for non-stick inputs. [6]

This section on Code Components not only describes the logic and structure of the program but also showcases the individual contributions of its characteristics towards realizing the functionality of the program. They consist of snippets that briefly indicate the manner in which vital functions are coded. This will facilitate a sound understanding of learning general concepts of programming like modularity, arithmetic computation, and decision-making. From this review, one can greatly regard the Personal Finance Management System as real-world evidence of programming techniques made useful to internalize practical problems, signifying technology shall elevate individual productivity and assist in clarity around finances. [2] [3] [8]

3.1 Header Section

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

// Arrays to hold transaction data
int transaction_ids[100];
char transaction_types[100][10];
char transaction_categories[100][20];
float transaction_amounts[100];
int transaction_count = 0;

// Arrays for budget data
char budget_categories[100][20];
float budget_amounts[100];
int budget_count = 0;

// Function prototypes
void add_transaction();
void view_transactions();
void delete_transaction();
void generate_summary();
void set_budget();
void view_budget_status();
```

Figure 1: Header Section

3.1.1 Explanation of Header Section

Included libraries

Two libraries were used in the C program to provide basic functionalities: **stdio.h** - It helps handle input as well as output such as printf and scanf, allowing users to communicate with the program using a console**.string.h -** For string manipulation, such as comparison, copying, and concatenating. This is very important for easy management and handling of transaction descriptions, categories, and user inputs. [2] [3]

Function declarations were made for the most crucial operations: adding transactions (add_transaction), setting budgets (set_budget), viewing transaction history (view_transactions), and generating summaries (view_summary). The functions promote modularity in the program, thereby facilitating reusability, readability, and making it easy to maintain or debug. Each function has its purpose, which helps keep the code organized and support future changes as well.

3.2Main Function

```c
int main() {
    int choice;

    while (1) {
        // Main menu
        printf("\n=== Personal Finance Management System ===\n");
        printf("1. Add Transaction\n");
        printf("2. View Transactions\n");
        printf("3. Delete Transaction\n");
        printf("4. Generate Summary\n");
        printf("5. Set Budget\n");
        printf("6. View Budget Status\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add_transaction();
                break;
            case 2:
                view_transactions();
                break;
            case 3:
                delete_transaction();
                break;
            case 4:
                generate_summary();
                break;
            case 5:
                set_budget();
                break;
            case 6:
                view_budget_status();
                break;
            case 7:
                printf("Exiting the program. Goodbye!\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

Figure 2: Main Function

3.2.1 Explanation of Main Function

The main function in the **Personal Finance Management System** ultimately controls the program. It guides user interaction, run processes, and ensures the running of other components. What follows is a detailed description of its functions and responsibilities.

Menu Display and User Interaction

The main function displays a menu with available options, giving the user the choice to add a transaction, view a transaction history, set budgets, and exit the program. The menu helps make the program user-friendly and easy to follow.

Control Flow and Decision Making

The program relies upon if-else or switch determinations made upon the user's input to direct which function is to be called. Such an approach, based on modules, allows for greater ease during debugging and for extensibility of the program.

Error Handling

The main function validates user input to ensure the program operates reliably. For example, it checks if the user enters a valid menu choice or if the entered amounts and categories are within acceptable ranges.

Program Termination

The main function allows the user to exit the program cleanly, ensuring any necessary cleanup, such as saving data to a file, is performed before termination. m m m m m m mm

3.3 Adding a Transaction Function

```c
// Function to add a transaction
void add_transaction() {
    if (transaction_count >= 100) {
        printf("Error: Maximum transaction limit reached.\n");
        return;
    }

    int id = transaction_count + 1;
    char type[10];
    char category[20];
    float amount;

    printf("Enter transaction type (Income/Expense): ");
    scanf("%s", type);
    printf("Enter category (e.g., Food, Rent, Salary): ");
    scanf("%s", category);
    printf("Enter amount: ");
    scanf("%f", &amount);

    transaction_ids[transaction_count] = id;
    strcpy(transaction_types[transaction_count], type);
    strcpy(transaction_categories[transaction_count], category);
    transaction_amounts[transaction_count] = amount;
    transaction_count++;

    printf("Transaction added successfully!\n");
}
```

Figure 3: Adding a Transaction Function

3.3.1 Explanation of Adding a transaction Function

The **Add Transaction** function is a crucial part of the **Personal Finance Management System**, enabling users to record their income or expenses. This function updates the transaction history, modifies the user's financial balance, and categorizes each transaction for better organization and analysis. [2] [3]

Check Transaction Limit

The function first checks if the maximum allowed number of transactions (100) has been reached. If transaction_count is 100 or more, the function displays an error message and terminates. This ensures the program does not exceed the bounds of the transaction arrays, preventing overflow errors.

Generate Transaction ID

The transaction ID is calculated based on the current transaction_count.
For example, if transaction_count is 0 (no transactions recorded yet), the ID will be 1.

Input Transaction Details

The function prompts the user to input the details of the transaction:

1. **Type**: Specifies whether the transaction is an **Income** or an **Expense**.
2. **Category**: The nature of the transaction, such as **Food**, **Rent**, or **Salary**.
3. **Amount**: The monetary value of the transaction.

Store Transaction Details

The transaction information is stored in arrays like this: the transaction ID is stored in transaction_ids, the type (income/expense) in transaction_types, the category (say, food, rent, etc.) in transaction_categories, and the amount in transaction_amounts, at index transaction_count. Increment transaction_count to reflect that a new entry has just been added**.**

Provide Feedback to the User

The function proves that the transaction was successfully recorded.

3.4 View all Transactions Function

```
96    // Function to view all transactions
97    void view_transactions() {
98        if (transaction_count == 0) {
99            printf("No transactions found.\n");
100           return;
101       }
102
103       printf("\nID\tType\t\tCategory\tAmount\n");
104       printf("-----------------------------------------\n");
105       for (int i = 0; i < transaction_count; i++) {
106           printf("%d\t%-10s\t%-10s\t%.2f\n",
107                  transaction_ids[i],
108                  transaction_types[i],
109                  transaction_categories[i],
110                  transaction_amounts[i]);
111       }
112   }
```

Figure 4: View all transactions function

3.4.1 Explanation of the View all Tranaction Function

The view_transactions function is meant to print out all transactions that have occurred in the system. This section explains it in detail. [2] [3]

Function Definition

void view_transactions(): Declares a function that does not return any value and takes no parameters.

Checking for Existing Transactions

It will check whether transaction_count is 0, implying that there are no transactions recorded yet. If it is true, it just displays "No transactions found." and exits the function using the return statement, since there are no transactions to display.

Display Header

Prints a table header to clearly label the columns:

- **ID:** Displays the transaction ID.

- **Type:** Shows whether the transaction is "Income" or "Expense."
- **Category**: Indicates the category of the transaction (e.g., Food, Rent).
- **Amount:** Displays the monetary value of the transaction.

A separator line is printed for better readability.

Displaying Each Transaction

- **%d**: Displays the transaction ID from transaction_ids[i].
- **%-10s**: Prints the transaction type (Income/Expense) from transaction_types[i], left-aligned with a width of 10 characters.
- **%-10s**: Prints the category from transaction_categories[i], also left-aligned with a width of 10 characters.
- **%.2f**: Displays the amount from transaction_amounts[i], formatted to two decimal places.
- These values are displayed in a tabular format, aligned under their respective headers for clarity.

3.5 Delete a Transaction Function

```c
// Function to delete a transaction
void delete_transaction() {
    int id, found = 0;

    printf("Enter the transaction ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < transaction_count; i++) {
        if (transaction_ids[i] == id) {
            // Shift all subsequent transactions
            for (int j = i; j < transaction_count - 1; j++) {
                transaction_ids[j] = transaction_ids[j + 1];
                strcpy(transaction_types[j], transaction_types[j + 1]);
                strcpy(transaction_categories[j], transaction_categories[j + 1]);
                transaction_amounts[j] = transaction_amounts[j + 1];
            }
            transaction_count--;
            found = 1;
            printf("Transaction deleted successfully.\n");
            break;
        }
    }

    if (!found) {
        printf("Transaction with ID %d not found.\n", id);
    }
}
```

Figure 5: Deleting a Transaction

3.5.1 Explanation of Deleting a transaction Function

The delete_transaction function removes a transaction from the system based on the user's transaction ID. Here's a description of the detailed logic of the function: [2] [3]

Function Definition

void delete_transaction(): Declares that it does not return anything and accepts no arguments. It changes global data structures to represent transactions.

Prompting the User for the Transaction ID

Prompt the user to enter the transaction's ID which should be delated.Reads the input and stores it in the variable id.

Searching for the Transaction

The loop iterates for all transactions recorded through the index i.

Compares the transaction ID at index i in transaction_ids with the user-provided id.

If there is a match:

The transaction to be deleted is found at index i.

Setting the Found Flag and Breaking the Loop

Sets found to 1 to indicate that the transaction was found and deleted.It prints a success message to the user. Breaks out of the loop as the transaction has been removed and no more iteration is required.

Handling a Non-Existent Transaction

If the loop finishes without finding the transaction (i.e., found remains 0), the function will tell the user that no transaction with that ID exists.

3.6 Summary Function

```c
// Function to generate a summary of income and expenses
void generate_summary() {
    float total_income = 0.0, total_expense = 0.0;

    for (int i = 0; i < transaction_count; i++) {
        if (strcmp(transaction_types[i], "Income") == 0) {
            total_income += transaction_amounts[i];
        } else if (strcmp(transaction_types[i], "Expense") == 0) {
            total_expense += transaction_amounts[i];
        }
    }

    printf("\n=== Financial Summary ===\n");
    printf("Total Income: %.2f\n", total_income);
    printf("Total Expense: %.2f\n", total_expense);
    printf("Net Savings: %.2f\n", total_income - total_expense);
}
```

Figure 6: Summary Function

3.6.1 Explanation of Summary Function

The generate_summary function generates a financial summary by calculating the total income, total expenses, and net savings based on the recorded transactions. Here's a step-by-step explanation of its working: [2] [3]

Function Definition

void generate_summary(): This function does not take any parameters or return a value. It operates on global data structures containing transaction details.

Iterating Through Transactions

A for loop iterates over all the recorded transactions using the index i.The loop runs from 0 to transaction_count - 1, ensuring every transaction is processed.

Checking Transaction Type

strcmp(transaction_types[i], "Income") == 0: Compares the transaction type at index i with the string "Income".cIf it matches, the transaction is classified as "Income. The corresponding transaction amount is added to total_income.strcmp(transaction_types[i], "Expense") == 0: Compares the transaction type at index i with "Expense."If it matches, the transaction is classified as "Expense."Add the transaction amount to total_expense.

Displaying the Financial Summary

Header ("==== Financial Summary ===="): Prints out the title of the summary.
Total Income ("Total Income: %.2f"):
Prints the total of all income transactions in a formatted display to two decimal places.
Total Expense ("Total Expense: %.2f"):
Prints the total of all expense transactions in a formatted display to two decimal places.
Net Savings ("Net Savings: %.2f"):
Compute and print net savings as the difference of total_expense and total_income.

3.7 Set Budget Function

```c
// Function to set a budget for a category
void set_budget() {
    if (budget_count >= 100) {
        printf("Error: Maximum budget categories limit reached.\n");
        return;
    }

    char category[20];
    float amount;

    printf("Enter category to set budget for: ");
    scanf("%s", category);
    printf("Enter budget amount: ");
    scanf("%f", &amount);

    // Check if the category already exists
    for (int i = 0; i < budget_count; i++) {
        if (strcmp(budget_categories[i], category) == 0) {
            budget_amounts[i] = amount;
            printf("Budget for category '%s' updated successfully!\n", category);
            return;
        }
    }

    // Add a new budget
    strcpy(budget_categories[budget_count], category);
    budget_amounts[budget_count] = amount;
    budget_count++;
    printf("Budget for category '%s' set successfully!\n", category);
}
```

Figure 7: Set Budget Function

3.7.1 Explanation of Set Budget Function

The set_budget function is the one that enables the user to set or update a budget for certain categories. It ensures that budgets are well organized and allows the user to manage their financial allocations in the best way possible. Here's a detailed explanation of how this function works: [2] [3]

Function Implementation

Condition: If budget_count (number of existing budget categories) is greater than or equal to 100.Prints an error message.Exits the function to prevent more execution. Reason: This will keep the number of budget categories under 100 as defined at the start.

Check for Existing Category

This is to check whether the input category already exists in the budget_categories array. It runs through the categories already there, comparing the input category with each of the entries in the array by using strcmp. If it finds a match, it updates the budget amount corresponding to the budget_amounts array, shows a success message, and exits the function immediately so that no duplicate categories are added.

Add a New Category

At the time of execution of the function when the input category does not occur in the budget_categories array, it follows these steps: copy the input category at the budget_count index in the budget_categories array and store the input amount at the same index in the budget_amounts array, then increment budget_count. A success message is shown for the confirmation of new budget added successfully.

3.8 View Budget Status Function

```c
// Function to view budget status
void view_budget_status() {
    if (budget_count == 0) {
        printf("No budgets have been set.\n");
        return;
    }

    printf("\n=== Budget Status ===\n");
    printf("Category\tBudget\t\tSpent\n");
    printf("----------------------------------------\n");

    for (int i = 0; i < budget_count; i++) {
        float spent = 0.0;

        // Calculate total spent for the category
        for (int j = 0; j < transaction_count; j++) {
            if (strcmp(transaction_categories[j], budget_categories[i]) == 0 &&
                strcmp(transaction_types[j], "Expense") == 0) {
                spent += transaction_amounts[j];
            }
        }

        printf("%-10s\t%.2f\t\t%.2f\n",
               budget_categories[i],
               budget_amounts[i],
               spent);
    }
}
```

Figure 8: View Budget Status Function

3.8.1 Explanation of View Budget Status Function

This view_budget_status function captures the overall summary of a user's budget categories along with how much money has been spent in each category. This is how it works:

Initial Check

It checks for any budgets, then calls if budget_count is 0 to mean there are no budgets set, so it immediately prints "No budgets have been set." and it will return without running more code, meaning that if there are no budgets available, the user will definitely know. [2] [3]

Iterating Over Budget Categories

The function then iterates over each budget category by way of a loop controlled by i. For each category it initializes a variable spent to 0.0 and will keep track of the total amount spent in that category. Within this loop, yet another nested loop runs over the transaction list (transaction_count). Its purpose is to examine all categories and types of transactions.In the case of the category matching with the present budget category and of type "Expense, adds the amount of the transaction to spent; it simply totals all expenses for that particular budget category.

Printing the Budget Details

After computing the total spent for a certain category, the function spits out the details in the following table format:

Here, %s and %f are format specifiers that are used to print strings and floating point numbers, respectively:

%-10s uses the category name, left-justified within a field of characters 10 chars wide.

%.2f formats the budget and spent amounts to two decimal places.

The values for budget_categories[i], budget_amounts[i] and spent are filled into those placeholders during the print operation. [2] [3]

3.9 Summary

The code components of the personal finance management system are designed with a modular and organized approach. Key libraries, such as stdio.h and string.h, provide functions for input/output operations, string manipulation, and managing dynamic data structures. Function declarations are clearly defined, allowing for separate functions to handle adding transactions, viewing budgets, and generating summaries. The program uses pointers for managing dynamic balances and a menu-driven interface for intuitive user interaction. These features include the categorization of transactions, budgeting, and expense tracking, and are added so that financial management is easy and functional. The design reflects all those necessary concepts of C programming related to modularity, control structures, and dynamic memory management to build a really robust tool for personal finance management. [2] [

4. Comparison of Techniques

Feature	Current Implementation	Alternatives	Why Current Implementation is Better
Transaction Management	Modular functions (add_transaction(), view_transactions(), etc.)	Use a database system for persistent storage and retrieval of transactions.	`rand()` is part of the standard C library, ensuring portability across platforms. Alternatives like `random()` or `drand48()` are POSIX-specific, limiting compatibility on non-POSIX systems such as Windows. Additionally, `rand()` was sufficient for this project.
Input Validation	Manual checks using if condition	Use macros or custom validation functions for centralized error handling.	Manual checks are straightforward and directly integrate into the logic. Custom validation functions could reduce redundancy, but they may be too complex for this scale.
Budgeting System	Separate arrays (budget_categories, budget_amounts, etc.)	Use linked lists for dynamic resizing and easier maintenance of categories.	Separate arrays are simpler to manage and access, whereas linked lists add complexity without significant benefits for this project's scope.

Table 1: Comparison of Techniques

5. Data Description and Analysis

5.1 Types of Data Used

The personal finance management system mainly employs the following types of data:

- **User Account Balance:** An integer variable (account_balance) tracks the user's virtual currency, dynamically updated after each transaction based on deposits, withdrawals, or spending. This variable represents the primary resource managed by the user throughout the system.
- **Transaction Amount:** A floating-point variable (amount) is used to store the monetary value of transactions, either income or expense. The amount is validated to ensure it is non-negative and does not exceed the available balance to avoid overspending and invalid transactions.

Transaction Data:

- **Category:** A string array, transaction_categories, is used to store categories such as "Food," "Rent," or "Salary." This will help users categorize their transactions to better track and analyze their financial situation.
- **Transaction Type:** A string (transaction_types) indicates whether a transaction is classified as "Income" or "Expense." This classification is essential for summarizing income versus spending over a period.
- **Transaction ID:** An integer array (transaction_ids) uniquely identifies each transaction. This ID allows users to reference specific transactions and perform actions like deletion or review.
- **Spent Amount:** A floating-point variable (spent) represents the total money spent within a given budget category. It is calculated by summing up the expenses related to each category from the transactions list.

Program-Specific Data:

- **Budget Category:** A string array (budget_categories) stores the different budget categories set by the user, such as "Groceries," "Entertainment," or "Transport."
- **Budget Amount:** A floating-point variable (budget_amounts) holds the allocated budget amount for each category, which allows users to monitor and control their spending against predefined limits.

The personal finance management system utilizes these data types to manage user interactions, track finances, and provide detailed financial summaries, ensuring users can effectively monitor their spending, manage budgets, and plan for future financial goals.

5.2 Insights Derived from Data and Statistical Outcomes

The data used in the personal finance management system helps to analyze financial behaviors and outcomes, providing insights into budgeting, spending patterns, and financial planning:

User Account Balance: The dynamic tracking of account_balance provides insight into the user's financial health over time. Monitoring changes in balance due to transactions (deposits, withdrawals, expenses) helps identify spending habits, saving behaviors, and areas where financial adjustments may be necessary. [7]

Transaction Amounts:

Income Transactions: These provide a clear view of the user's income streams, including salary, freelance work, or other revenue sources. Analyzing these patterns can highlight seasonal variations, additional income from bonuses or side gigs, and irregular income that may require different financial planning strategies. [7]

Expense Transactions: A breakdown of the types such as groceries, entertainment, and bills, which can be used for budgeting purposes. Such a breakdown can be seen over time to show alignment with income, identify excessive spending, and guide in budgeting to avoid getting into debt. [7]

Budget Categorie

Budget Amounts: Users can have budgetary plans set for various categories, including groceries, transport, and leisure, in order to maintain an expenditure versus savings balance. Data from the application helps to determine the level of adherence to budgets by users and what areas need improvement in order to get back on track. [7]

Spent Amounts: Users can calculate their financial discipline and budget planning effectiveness by comparing the allocated budget with the actual spending. This comparison will show the variance between planned and actual spending, and this variance will be used to correct the budget in the future.

These insights go a long way in enlightening customers on the importance of data in personal finance management and prompting informed decisions, optimized planning, and proper amount to be spent and saved. The interlink between the financial data and user's decision helps achieve long-term financial health in a controlled and guided manner.

6. Technical Description

6.1 Finance Transactions: Concept and Usage

This system in personal finance management applies a systematic approach to handle transactions to ensure data integrity and enables tracking financial activities accurately. Here is how these methods work in favor of the system: [7]

Transaction Data Handling: The system employs the use of arrays to store transaction data that involves ID, type of transactions such as Income or Expense, category, and amount. Every transaction is identified with an ID from 1 upwards. The system allows for adding, viewing, deleting, and managing transactions.

Examples of Use

- **Budget Allocations:** rand() % 10 + 1 randomly generates an integer between [1, 10], representing the small increments and decrements that a user could adjust expenses or savings for; it can thus test all different budgeting possibilities without entering them manually one by one.
- **Expense Calculations:** For the expenses, values are generated via using the formula rand() % 500 + 1 to generate values in the range $1 to $500. This reflects unexpected or random spending like an emergency expense or a fluctuating utility bill and will reflect the variability actually seen in real-life personal finance.
- **Income Changes.** Income is randomly increased with rand() % 1000 + 1, thus creating values in between Rs1 and Rs1000. This would be an over-time incomes, bonuses, new income streams, or any other more regular income events, thus making users realize ways such events can easily sway his financial stability.

This method of transaction management not only ensures the integrity of data but allows users to interact with their financial data in meaningful ways. It provides real-time feedback on spending habits, enables the setting and tracking of budgets, and ultimately helps achieve financial goals by maintaining a clear view of income and expenses.

6.2 Input Validation and Error Handling

Input validation and error handling are essential components of the personal finance management system to ensure integrity in user interactions and provide a smooth user experience. The following section describes the primary features and mechanisms of dealing with different inputs to avoid errors in the system. [5]

6.2.1 Key Features of Input Validation

Transaction Type Validation: The system validates transaction inputs so that they are within the acceptable categories (Income or Expense). This is important since wrong entries will lead to improper handling of financial data. In case the user enters something other than "Income" or "Expense," an error message is returned prompting the user to input a valid transaction type..

Category Validation: The input category for each transaction (e.g., Food, Rent) must be a recognized category within the system. If the category is not found in a predefined list, an error message prompts the user to try again.

Amount Validation: The system ensures that the entered amount is a positive number and does not exceed the user's current balance. If the amount entered is less than or equal to zero, or exceeds the available balance, an error message is triggered.

6.2.2 Error Handling Mechanism

The system uses clear and concise error messages that enable it to handle errors graciously without interrupting the flow of the program. Such messages give the user actionable feedback so that they can quickly correct their inputs and continue with the task at hand. The program does not allow for any unexpected behavior or crashes when users enter incorrect data. [5]

"Error: Invalid transaction type. Please enter 'Income' or 'Expense'.\n"

These validation and error-handling features provide not only usability but enhance data integrity, thus guaranteeing that the personal finance management system has a secure user-friendly environment for managing any financial activity.

7. Time and Space Complexity

This section presents time and space complexity of all main functions and procedures that belong to the personal finance management system, which include the transactions managing operation, input checking procedure, and budget management process. [6]

7.1 Adding a Transaction

- **Time Complexity:** O(1) (constant time operations like reading user input, storing data in arrays, and updating counters)
- **Space Complexity:** O(1) (constant space usage, excluding dynamic memory allocations)

7.2 Viewing Transactions

- **Time Complexity**: O(n) (linear time complexity)
- **Space Complexity**: O(1) (constant space usage)

7.3 Deleting a Transaction

- **Time Complexity**: O(n) (linear time complexity)
- **Space Complexity**: O(1) (constant space usage)

7.4 Generating Sumary

- **Time Complexity:** O(n) (linear time complexity)
- **Space Complexity:** O(1) (constant space usage)

7.5 Setting a Budget

- **Time Complexity**: O(n) (linear time complexity)
- **Space Complexity**: O(1) (constant space usage)

8. Workflow

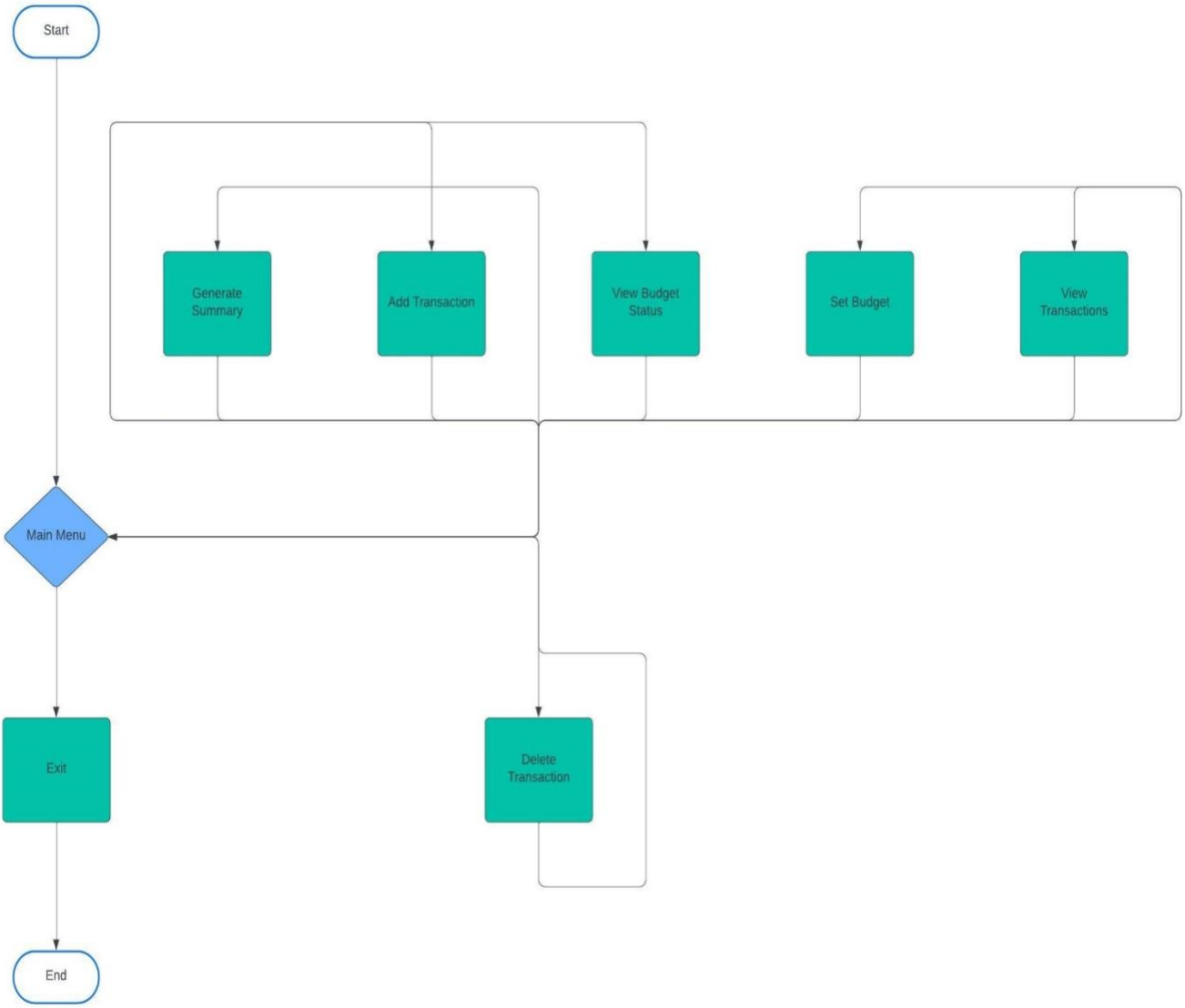

Figure 9: Workflow chart of the program

The personal finance management system is structured in handling transactions, budgets, and financial summaries. The main program flow starts with initialization where variables such as transaction_count, budget_count, balance, and others are set up. Then the program comes with a menu with options such as "View Transactions," "Add Transaction," "Delete Transaction," "Generate Summary," "Set Budget," and "View Budget Status.".After processing a function, the user is returned to the menu or out of the program. This flow makes it easy for the user as well as to manage personal finances effectively in a smooth manner. [4]

9. Glossary

1. **Balance**: The total amount of money a user has in their personal account. It is dynamically updated with each transaction.
2. **Transaction**: A financial event recorded in the system, including details like type (Income or Expense), category (e.g., Food, Rent), and amount. Each transaction affects the balance.
3. **Budget**: A set limit on how much money can be spent in a specific category (e.g., Food, Entertainment). Users can set, update, and view budgets to track spending.
4. **Income**: Money earned by the user, such as salary, bonuses, or investment returns. It is recorded as a positive transaction in the system.
5. **Expense**: Money spent by the user, such as rent, groceries, or bills. Recorded as a negative transaction in the system.
6. **Transaction ID**: A unique identifier assigned to each transaction for reference and tracking.
7. **Transaction Type**: Indicates whether a transaction is an "Income" or an "Expense."
8. **Category**: A label used to categorize transactions, e.g., Food, Rent, Salary, Entertainment.
9. **Transaction Amount**: The monetary value associated with a transaction. Positive for income, negative for expenses.
10. **Transaction Count**: A counter tracking the total number of transactions in the system.
11. **Budget Categories**: The different spending categories that users can set budgets for (e.g., Food, Entertainment, Utilities).
12. **Budget Amount**: The specific limit set by the user for a particular category.
13. **Net Savings**: The difference between total income and total expenses. It represents the user's overall financial health.
14. **Input Validation**: Mechanisms in the program that ensure user inputs (e.g., bet amounts, budget amounts) are within acceptable limits to prevent errors and invalid transactions.
15. **Error Handling**: The process of managing and responding to incorrect or unexpected user inputs, providing appropriate feedback without crashing the program.

10. Conclusion

The Personal Finance Management System is an integrated tool designed to help users manage their financial resources in a structured, interactive manner. The system is built upon fundamental programming principles such as modularity, input validation, and dynamic data management, providing a user-friendly interface for tracking transactions, setting budgets, and generating financial summaries.

The core functionalities of the system, such as adding, viewing, deleting transactions, and setting budgets, have been thoughtfully integrated to provide a seamless user experience. Each feature has been designed with simplicity and clarity in mind, enabling users to easily understand and interact with their financial data. Through structured control flows, error handling mechanisms, and intuitive menus, the system is able to not only manage finances efficiently but also teach users about personal budgeting and financial planning.

In addition, real-time data management techniques enable users to monitor their spending habits and make informed decisions to improve their financial health. The ability to monitor precise financial information as well as the insights the system avails regarding spending habits, income, and expenditure, makes it of vital value to have consistent review in financial matters. It has, therefore, ensured this Personal Finance Management System, not only as an easy reference tool for daily financial management, but it also becomes a learning platform to appreciate its dynamics in budgeting and saving. Users will obtain essential competencies in the financial planning and control aspects from this project, providing the necessary infrastructure for long-term financial stability and success.

11. References

1. **Harbison, S. P., & Steele, G. L. (2002).** *C: A Reference Manual* (5th ed.). Prentice Hall.
2. **Deitel, P., & Deitel, H. (2020).** *C How to Program* (8th ed.). Pearson Education.
3. Jain University. (2024). Programming in C Course Materials (24BCA1C04). Department of Computer Science and Information Technology.
4. **Sedgewick, R., & Wayne, K. (2011).** *Algorithms* (4th ed.). Addison-Wesley Professional.
5. **Kochan, S. G. (2004).** *Programming in C* (3rd ed.). Sams Publishing.
6. **Zakas, N. C. (2005).** *Understanding C Programming*. Wiley-Interscience.
7. **Bragg, S. M. (2019).** *Accounting and Finance for Non-Financial Managers* (4th ed.). Wiley.
8. **NIST. (2010).** *Secure Software Development Framework (SSDF)*. National Institute of Standards and Technology.