CS160 - Introduction to Programming

Personal Finance Manager

Final Project Report

Team members:

Lê Quỳnh Anh	25125071
Trần Nguyễn Đăng Dương	25125072

Date of submission: 28/12/2025

Contents

Chapter 1

Project Overview

In this project, we built a small Personal Finance Manager in C++. The program runs in the console (CLI). It helps users record daily income and expense transactions, manage master data (wallets, income sources, expense categories), set up monthly recurring transactions, and view basic statistics.

Main objectives

- Let the user add income/expense quickly and update wallet balances immediately.
- Store data using IDs internally, but show friendly names on the screen.
- Support recurring monthly income/expense and apply them automatically when the app starts.
- Provide statistics reports for a date range or a year range.
- Save everything into a file so the data is still there after reopening the program.

Chapter 2

Requirements Specification

This section lists the assignment requirements and how our program covers them.

2.1 Requirement summary

1. **Transaction Management:** record income/expense (date, source/category, amount, wallet, description) and keep ID vs displayed name.
2. **Recurring Transactions:** monthly recurring income/expense with start date and optional end date; do not duplicate the same month.
3. **Master Data Management:** manage wallets, income sources, and expense categories.
4. **Statistics & Reporting:** totals in time range, breakdown by wallet, yearly overview, breakdown by income source and expense category.
5. **Data Persistence:** save/load data (binary) using `fstream`.
6. **CLI Dashboard:** show total balance + balances of each wallet; return to dashboard after each function.
7. **Constraints:** no STL containers like std::vector; std::string is allowed.

2.2 Implementation coverage table

Requirement	**Implemented in this project**	**Status**
Transaction Management	Add income/expense via CLI forms, store date/amount/description and reference master-data IDs; wallet balance updates immediately.	Implemented

Requirement	Implemented in this project	Status
ID vs Name separation	Master lists store IDs and names; the UI shows names for selection; logic uses IDs for lookups and statistics.	Implemented
Recurring monthly transactions	Recurring income/expense with start/end date; auto-applied on startup using last-applied month/year to avoid duplicates.	Implemented
Master Data Management	Add/Delete/List wallets, income sources, expense categories. Safe delete is enforced. Wallet edit exists in code but is not shown in menu; edit for sources/categories is not implemented.	Partially
Statistics & Reporting	Time-range totals, wallet breakdown, yearly overview, breakdown by income source and expense category.	Implemented
Data Persistence	Binary save/load (pfm.bin) with manual serialization using fstream.	Implemented
Dashboard and navigation flow	Dashboard shows total money + wallet balances; menu returns to dashboard after tasks.	Implemented
No STL containers	Uses a custom DynamicArray<T> (manual memory) instead of std::vector.	Implemented

Chapter 3

Design Document

3.1 High-level architecture

We separated the code into multiple files so it is easier to manage:

- **UI (pfm_ui.*):** menus and user input.
- **Core (pfm_core.*):** main logic (add transactions, recurring, statistics).
- **State (pfm_state.h):** data structures (AppState, masters, transactions).
- **Utils (pfm_utils.*):** date checking, searching by ID, helper functions.
- **Persist (pfm_persist.h):** save/load binary file.

3.2 Data model

All application data is stored inside `AppState`. We keep master data (wallets, sources, categories) and also keep the lists of income/expense transactions and recurring rules.

3.2.1 Data overview

Instead of a complicated UML diagram, we describe the main structs clearly:

- WalletMaster: wallet ID, wallet name, current balance.
- IncomeSourceMaster: income source ID and name.
- ExpenseCategoryMaster: category ID and name.
- IncomeData: date, source ID, wallet ID, amount, description.
- ExpenseData: date, category ID, wallet ID, amount, description.
- `RecurringItem`: type (income/expense), start/end date, last applied month/year, and a prototype transaction.

3.3 Custom Dynamic Array

Because STL containers are not allowed, we use a custom DynamicArray<T>. It supports:

- Resizing with reserve(),
- Adding with push_back(),
- Removing by index with removeAt().

3.4 Persistence format

All data is saved into pfm.bin using fstream. The file includes counts + entries for wallets, sources, categories, transactions, and recurring items. Strings are saved as length + bytes.

Chapter 4

Implementation Details

4.1 Source code organization

- main.cpp: load data, apply recurring, show dashboard loop, save and exit.
- pfm_state.h: all structs and the global state container.
- dynamic_array.h: custom dynamic array template.
- pfm_ui.*: menus for dashboard, master data, recurring, statistics.
- pfm_core.*: logic for add income/expense, apply recurring, calculate stats.
- pfm_utils.*: date validation, lookups, used-checks for safe deletion.
- pfm_persist.h: save/load in binary format.
- income.*, expense.*: input forms for transactions.

4.2 Key workflows

4.2.1 Startup

1. Load AppState from pfm.bin (if the file exists).
2. Apply recurring rules up to the current month.
3. Show the dashboard and wait for user actions.

4.2.2 Adding income/expense

- The user selects a wallet and a source/category from master data.
- The user enters date, amount, and description.
- The transaction is stored and wallet balance is updated immediately.

4.2.3 Recurring transactions

For recurring items, we store a prototype transaction and the last month/year applied. On startup, the app applies missing months until reaching the current month, then updates the last applied month/year to prevent duplicates.

4.2.4 Statistics

Statistics are calculated by scanning stored transactions and checking:

- date range (converted into a comparable key),
- year range,
- wallet/source/category ID match.

Chapter 5

Testing

5.1 Testing strategy

We tested the program mainly by running it and trying many input cases:

- wrong menu input (letters instead of numbers),
- invalid dates,
- adding/deleting master data,
- checking persistence by closing and reopening,
- verifying recurring transactions do not duplicate.

5.2 Test cases

#	Test case	Expected result
1	Add a wallet, check dashboard	Wallet appears and total money updates
2	Add income source and expense category	They appear in selection lists
3	Add an income to a wallet	Wallet balance increases correctly
4	Add an expense to a wallet	Wallet balance decreases correctly
5	Delete wallet used by transactions	Deletion is blocked
6	Set recurring from previous month, reopen app	Missing months are added once (no duplicates)
7	Run statistics for a date range	Totals match the stored transactions
8	Save, exit, reopen	Data stays the same after reloading

Chapter 6

Known Issues / Limitations

- Editing master data is not fully completed (wallet edit is not shown in menu; edit for sources/categories is not implemented).
- There is no full transaction history screen (we focus on adding and viewing statistics).
- Transaction amounts use int, so extremely large values might overflow.
- `DynamicArray<T>` is minimal and does not support copying safely (we avoid copying AppState).

Chapter 7

Future Improvements

- Add full edit features for wallets, income sources, and expense categories.
- Add a transaction list screen (view/search/edit/delete transactions).
- Improve the save file format (versioning and safer recovery if the file is corrupted).
- Add export to CSV for easier reporting.

Chapter 8

Demo Videos

https://youtu.be/6osH9ae44q0?si=IimR13QdQb2Q9CKp

Chapter 9

Work Division

9.1 Team members

- Lê Quỳnh Anh - 25125071
- Trần Nguyễn Đăng Dương - 25125072

9.2 Responsibilities

- Lê Quỳnh Anh: implemented the whole application (UI, logic, persistence, testing, video, report).
- Trần Nguyễn Đăng Dương: no contribution.

9.3 Collaboration tools

- GitHub for version control.