https://github.com/Labmann009/Final
GuangyuFu 30054546

Bank-Account Management Program – Executive Summary

Purpose.
This console application satisfies “Assignment 2 – Task 3” by turning a simple balance checker into a persistent mini-bank system: it remembers every deposit and withdrawal (with a timestamp) between runs and rebuilds the balance automatically.

---

1. Functional Requirements Met

1. **Long-term memory** – reads *transactions.txt* at startup, writes it back on exit.
2. **Dynamic storage** – stores all records in std::vector<Transaction>.
3. **Seven-item menu** – Balance, Deposit, Withdrawal, three history views, Exit.
4. **Per-transaction timestamps** – ISO string from currentTimestamp().
5. **Start & final balances** – both printed in the same console view at exit.

---

2. Core Architecture

Component	Responsibility
struct Transaction	Holds amount, type ('D' or 'W'), and timestamp.

class BankAccount	Encapsulates • double balance (live total) • vector<Transaction> history • loadHistory() / saveHistory() • makeDeposit() / makeWithdrawal() • three listing helpers • getBalance() accessor.
currentTimestamp()	Returns local time as **YYYY-MM-DD HH:MM:SS**.
main() loop	Presents the menu, routes user choices, and—on Exit—prints **starting** and **final** balances.

3. Screenshots to Deliver

1. **Build succeeded** – Solution Explorer + Output.
2. **Deposit action** – console line with amount & timestamp.
3. **Withdrawal action** – console line with amount & timestamp.
4. **Deposit History** – table listing deposits.
5. **Withdrawal History** – table listing withdrawals.
6. **All History** – combined list.
7. **Exit view** – starting vs. final balance (single screen).

---

4. Conclusion

The program fulfills every bullet in the assignment: accurate balance tracking, complete transaction logs with timestamps, persistent storage, and a clear seven-item menu. Add the seven required screenshots to this summary, export as PDF, and the submission is ready.

Presentation Script

I'm Guangyu Fu, and I'll spend the next five minutes walking you through the Bank-Account Management program I wrote for Assignment 2, Task 3.

Why this program?

- The assignment asked us to turn a one-shot balance checker into a *persistent* mini-bank system. Which means the application must remember every deposit and withdrawal *between* runs, print time-stamped receipts, and rebuild the balance automatically when you relaunch it. It also had to expose seven clear menu options so the grader could test each feature without digging into the code.

Requirement	**My Implementation**
Long-term memory	I store all transactions in a text file called transactions.txt. When the program starts, it reads that file; when it exits, it rewrites it.
Full history in RAM	While the program is running, every record lives in a std::vector<Transaction> so I can iterate quickly and show filtered lists.
Time-stamps	Each deposit or withdrawal calls a helper currentTimestamp() which returns local time in ISO format—readable and sortable.
Seven-item menu	The main loop prints Balance, Deposit, Withdrawal, three kinds of history, and Exit. Each choice maps to a dedicated method inside my BankAccount class.

Starting vs. Final balance	At launch I cache the initial balance; right before exiting I print both the starting figure and the final figure so the instructor can screenshot them on the same console window.

1. **Transaction struct** – three tiny fields:
   - amount – a positive double.
   - type – a char, D for deposit or W for withdrawal.
   - timestamp – a plain string.
     Keeping it small means fast file I/O and low memory overhead.
2. **BankAccount class** – the heart of the app.
   - **Data**: one double called balance, and the vector<Transaction> I just mentioned.
   - **Constructor**: calls loadHistory(), which opens the text file, pushes every line into the vector, and updates balance in real time.
   - **Persistence**: saveHistory() runs once—exactly when you choose “Exit.” It overwrites the file so there’s no risk of duplicate entries.
   - **Business methods**:
     - makeDeposit() and makeWithdrawal() both validate the amount, modify balance, append a new Transaction, and echo a receipt with the timestamp.
     - listDeposits(), listWithdrawals(), and listAllTransactions() simply iterate over the vector and

filter by type.

    - **Accessor**: getBalance()—only one line of code, but crucial for showing both balances at exit without breaking encapsulation.

3. **main() loop** – about 40 lines of code.
    - Prints the menu, collects user input, and calls the right BankAccount method.
    - Uses std::cin guards so a typo won't crash the program—if you type a letter where a number is expected, it clears the stream and asks again.

---

4. Typical User Flow

"Picture the grader running our executable…"

1. Program greets them, shows an immediate **Balance $500**—or whatever the rebuilt figure is.
2. They deposit $100; the console flashes: *Deposited $100.00 on 2025-05-22 00:30:47.*
3. They withdraw $50; another line appears with a fresh timestamp.
4. Menu option 4 shows **Deposit History**—only that $100 line.
5. Option 5 shows **Withdrawal History**—only that $50 line.
6. Option 6 shows **All History**—both lines stacked.
7. Finally, Option 7 prints:

What did I makes it robust?

- **Input validation** – impossible to over-withdraw or enter a negative deposit.
- **Single source of truth** – balance is never stored twice; it’s derived once at startup and mutated only through the validated methods.

In summary, the application captures every requirement on the checklist: persistent storage, time-stamped receipts, a seven-item menu, and clear beginning-to-end balance reporting. Fully functional and easy to extend if we ever add, say, interest calculations or user authentication.

Thank you!