NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY

SIMPLE ATM SIMULATION

COURSE: DATA STRUCTURES IN C

DEPARTMENT: COMPUTER SCIENCE

SECTION: C

YEAR: 1

SEMESTER: 2

DATE: MAY 04, 2025

SUBMITTED TO: ASHOK SELVA KUMAR E

SUBMITTED BY:

S.D.V. PRASAD REDDY (24KB1A05GP)

S. RAGHAVA (24KB1A05GN)

S. NITHIN KUMAR (24KKB1A05GD)

Acknowledgments

We express our sincere gratitude to NBKR Institute of Science and Technology (NBKRIST) for providing the opportunity and necessary resources to undertake this project.

We are deeply thankful to our guide, Mr. Ashok Selva Kumar E, for his valuable guidance, constant support, and insightful feedback throughout the development of the ATM Simulation System.

We also appreciate the faculty of the CSE Department, our peers, and our families for their encouragement and support.

Abstract

This project implements a simple yet functional ATM simulation system in C. It supports basic operations such as balance inquiry, withdrawal, and deposit through a text-based interface. It also utilizes file handling to maintain persistent account balances and uses a PIN for user authentication. The project offers a practical demonstration of fundamental C programming concepts like loops, conditionals, functions, and file I/O.

Introduction

With the rise of automation in banking, ATMs have become essential interfaces for customers. This project simulates a basic ATM to explore its underlying logic and to enhance our understanding of real-world systems through programming. The idea was chosen to integrate theoretical knowledge with practical application in a meaningful context.

Objectives

- Simulate basic ATM operations using the C programming language.
- Implement file handling for balance storage.
- Practice conditional statements, loops, and switch-case constructs.
- Provide secure access through PIN-based authentication.

System Requirements

Hardware:
- Processor: Intel i3 or higher
- RAM: 2 GB or more
- Storage: 500 MB of free disk space

Software:
- OS: Windows or Linux
- Compiler: GCC (e.g., Code::Blocks, Turbo C)

Methodology

The project followed a step-by-step development process:
1. Identified key ATM features (authentication, transactions).
2. Used C functions to modularize code.
3. Applied file I/O to ensure persistent balance storage.
4. Implemented and tested user scenarios.
5. Debugged and optimized the final version.

Project Description

Problem Statement:
Users need a system to simulate basic banking transactions without actual bank servers.

Proposed Solution:
A console-based program in C that allows secure user authentication and enables balance checking, deposit, and withdrawal.

Key Features:
- PIN authentication
- Balance inquiry
- Cash deposit
- Cash withdrawal
- Persistent data using files

Algorithm

1. Load balance from balance.txt. If it doesn't exist, initialize a default balance.
2. Prompt for PIN input.
3. If the PIN matches:
   a. Display menu options.
   b. Perform transaction based on user choice.
   c. Update and save the new balance.
4. If PIN is incorrect, display an error and exit.

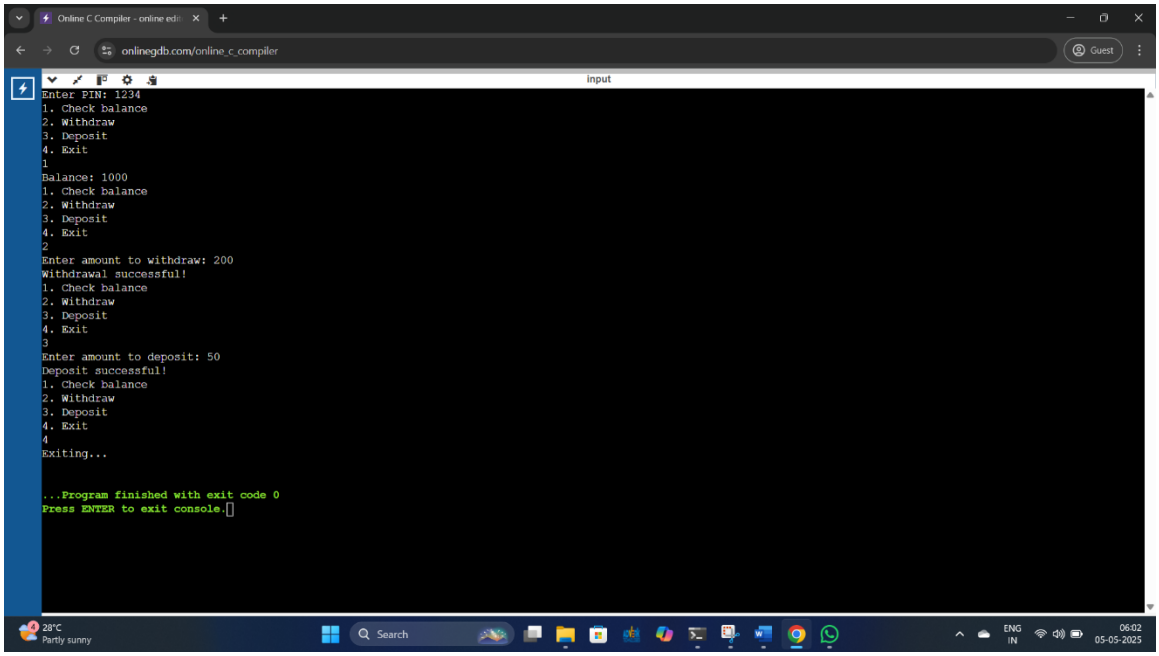
Program Code

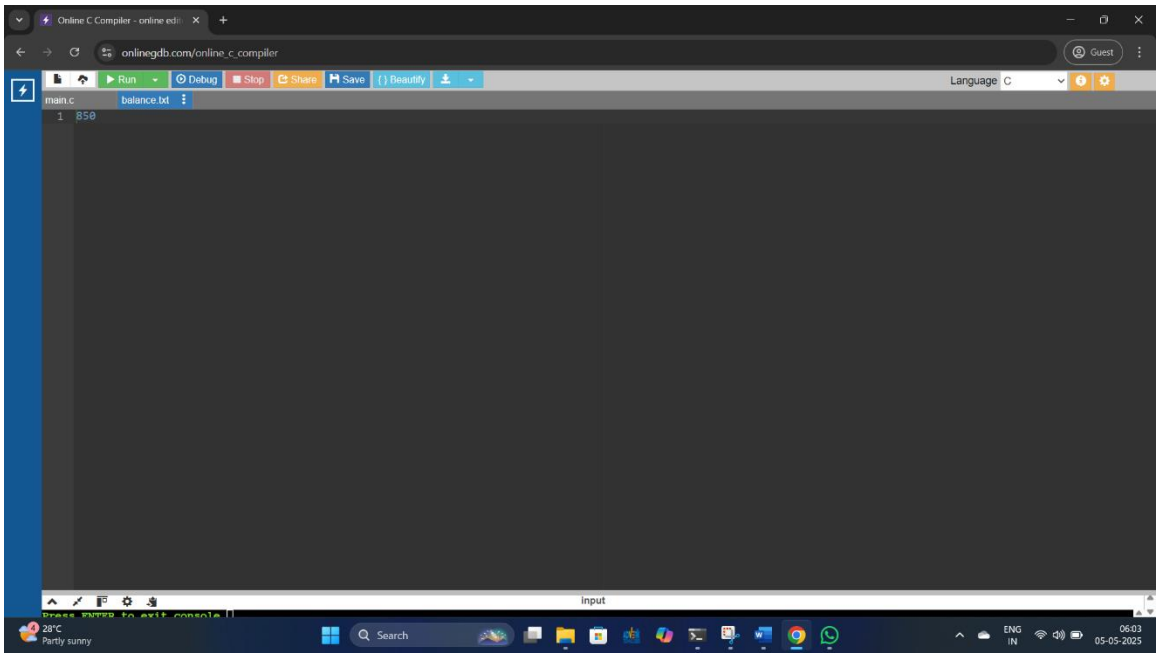
```
#include <stdio.h>

int pin = 1234;
int balance = 1000;

void saveBalance() {
   FILE *fp = fopen("balance.txt", "w");
   fprintf(fp, "%d", balance);
   fclose(fp);
}

void loadBalance() {
   FILE *fp = fopen("balance.txt", "r");
```

```
    if (fp != NULL) {
        fscanf(fp, "%d", &balance);
        fclose(fp);
    } else {
        saveBalance();
    }
}

int main() {
    loadBalance();
    int enteredPin;
    printf("Enter PIN: ");
    scanf("%d", &enteredPin);

    if (enteredPin == pin) {
        while (1) {
            printf("1. Check balance\n");
            printf("2. Withdraw\n");
            printf("3. Deposit\n");
            printf("4. Exit\n");
            int choice;
            scanf("%d", &choice);

            switch (choice) {
                case 1:
                    printf("Balance: %d\n", balance);
                    break;
                case 2:
                    printf("Enter amount to withdraw: ");
                    int withdrawAmount;
                    scanf("%d", &withdrawAmount);
                    if (withdrawAmount <= balance) {
                        balance -= withdrawAmount;
                        saveBalance();
                        printf("Withdrawal successful!\n");
                    } else {
                        printf("Insufficient balance!\n");
                    }
                    break;
                case 3:
                    printf("Enter amount to deposit: ");
                    int depositAmount;
                    scanf("%d", &depositAmount);
```

```
            balance += depositAmount;
            saveBalance();
            printf("Deposit successful!\n");
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice!\n");
      }
    }
  } else {
    printf("Invalid PIN!\n");
  }
  return 0;
}
```

Output Screenshot





Testing and Validation

Testing Phase 1 - Sample Input and Output

Input: 1234 (PIN)
Output: Correct PIN, options displayed

Input: 2 (Withdraw)
Input: 500 (Amount to withdraw)
Output: Withdrawal successful, updated balance displayed

Input: 3 (Deposit)
Input: 1000 (Amount to deposit)
Output: Deposit successful, updated balance displayed

Input: 4 (Exit)
Output: Program exits

Testing Phase 2 - Sample Input and Output

Input: 1111 (PIN)
Output: “INVALID PIN”

Limitations

- Only supports a single user account
- No encryption for PIN storage
- No GUI interface
- No support for multiple sessions simultaneously

Future Enhancements

- Add GUI support
- Support multiple accounts
- Implement database integration for scalability
- Add password encryption for better security
- Include mini statement generation

Conclusion

This project helped us understand how basic financial systems function and offered hands-on experience with file handling, modular programming, and user interaction in C. We gained valuable skills in designing, testing, and improving console applications.

References

1. C Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft
2. Problem Solving with Algorithms and Data Structures" by Brad Miller and David Ranum.

Appendix

- Source code
- Sample test cases
- Screenshot of output