

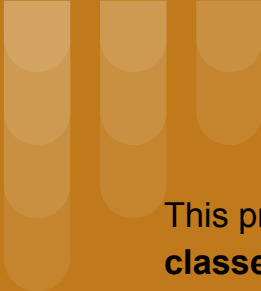
Bank Accounts/ATM simulator

By Lidya Ghede




This C++ program simulates an **ATM System** with the ability to manage multiple bank accounts. It uses **classes**, **pointers**, **dynamic memory**, **file I/O**, and **binary search** to perform core banking operations like deposit, withdrawal, and account lookup.

When the program runs, it performs a series of predefined operations such as adding sample accounts, depositing and withdrawing money, and printing account details to external files. All transaction activities are saved in atm_output.txt, while final summaries including transaction history are saved in accounts.txt. The program demonstrates real-world banking logic in a simple, structured format.



This project makes use of several key C++ programming concepts. It uses **classes and objects** to encapsulate account behavior, and **private member variables** to protect sensitive data like balance and transaction history. The program relies on **pointers and dynamic memory allocation** (using `new[]` and `delete[]`) to manage arrays of accounts and transactions.

To efficiently locate accounts, the program uses a **binary search algorithm** on a sorted list of accounts. **File input and output (I/O)** is used to log all activities and generate reports in external .txt files. The program also demonstrates the use of **arrays**, **string manipulation** for formatting transaction logs, and **formatted output** using `setprecision` to display monetary values correctly. Throughout the code, control structures like `if`, `for`, and `while` are used to manage logic flow and handle conditions.



1. The Account class stores user information such as the account holder's name, account number, and balance. It also uses a dynamically allocated array (transactionHistory) to store a fixed number of transaction logs.

2. The constructor initializes the account with default or given values and allocates memory for storing up to 10 transactions. The addTransaction function logs each deposit or withdrawal as a string.

```
9  class Account {
10 private:
11     string accountHolder;
12     int accountNumber;
13     double balance;
14     string* transactionHistory; // for pointer to dynamic array
15     int transactionCount;
16     int maxTransactions;
17
```

```
public:
    // constructor
    Account(string holder = "Unknown", int accNum = 0, double bal = 0.0) {
        accountHolder = holder;
        accountNumber = accNum;
        balance = bal;
        maxTransactions = 10; // Fixed-size array for transactions
        transactionHistory = new string
            [maxTransactions]; // dynamic allocation
        transactionCount = 0;
    }
    void addTransaction(string transaction) {
        if (transactionCount < maxTransactions) {
            transactionHistory[transactionCount] = transaction;
            transactionCount++;
        } else {
            cout << "Transaction history full!" << endl;
        }
    }
}
```

These functions update the balance based on user actions. After each successful transaction, a formatted message is stored in the transaction history.

```
// Deposit money
void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        string trans = "Deposited $" + to_string(amount) + ", New Balance: $" + to_string(balance);
        addTransaction(trans);
    }
}

// Withdraw money
bool withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        string trans = "Withdrew $" + to_string(amount) + ", New Balance: $" + to_string(balance);
        addTransaction(trans);
        return true;
    }
    return false;
}
```

1. The ATM class manages a dynamic array of Account objects. It keeps track of how many accounts exist and allows adding, searching, and performing actions on them.

```
70
71 // Class 2: ATM to manage multiple accounts and operations
72 class ATM {
73 private:
74     Account* accounts; // Pointer to dynamic array of accounts
75     int accountCount;
76     int maxAccounts;
77
```

2. Binary search is used to quickly locate an account by its number. It reduces the number of comparisons compared to linear search, especially for large datasets.

```
100
101 // Binary search to find account by account number
102 int binarySearch(int accNum) {
103     if (accountCount == 0) {
104         return -1; // No accounts to search
105     }
106
107     int left = 0, right = accountCount - 1;
108     while (left <= right) {
109         int mid = left + (right - left) / 2;
110         if (accounts[mid].getAccountNumber() == accNum) {
111             return mid;
112         }
113         if (accounts[mid].getAccountNumber() < accNum) {
114             left = mid + 1;
115         } else {
116             right = mid - 1;
117         }
118     }
119     return -1; // Account not found
120 }
121
```

1. This function writes the details of a specific account (including transaction history) to the output file. It helps simulate a bank statement log.

2. The main() function creates an ATM object, adds sample accounts, performs transactions, and displays results by writing them to a text file.

```
// Display account details
void displayAccount(int accNum, ofstream& outFile) {
    int index = binarySearch(accNum);
    if (index != -1) {
        outFile << "\nAccount Details:\n";
        outFile << "Holder: " << accounts[index].getHolder() << endl;
        outFile << "Account Number: " << accounts[index].getAccountNumber() << endl;
        outFile << "Balance: $" << fixed << setprecision(2) << accounts[index].getBalance() << endl;
        accounts[index].displayHistory(outFile);
    } else {
        outFile << "Account " << accNum << " not found!" << endl;
    }
}
```

```
// Create ATM instance
ATM atm;

// Add sample accounts
atm.addAccount("John Doe", 1001, 500.0);
atm.addAccount("Jane Smith", 1002, 1000.0);
atm.addAccount("Alice Johnson", 1003, 250.0);

// Perform transactions
atm.deposit(1001, 200.0, outFile);
atm.withdraw(1002, 300.0, outFile);
atm.deposit(1003, 50.0, outFile);
atm.withdraw(1001, 100.0, outFile);

// Display account details
atm.displayAccount(1001, outFile);
atm.displayAccount(1002, outFile);
atm.displayAccount(1004, outFile); // Non-existent account
```


Data Type Sizes (in bytes):

int: 4

double: 8

string: 24

Account*: 8

Deposited \$200 to Account 1001

Withdrew \$300 from Account 1002

Deposited \$50 to Account 1003

Withdrew \$100 from Account 1001

Account Details:

Holder: John Doe

Account Number: 1001

Balance: \$600.00

Transaction History for Account 1001 (John Doe):

Deposited \$200.000000, New Balance: \$700.000000

Withdrew \$100.000000, New Balance: \$600.000000

Account Details:

Holder: Jane Smith

Account Number: 1002

Balance: \$700.00

Transaction History for Account 1002 (Jane Smith):

Withdrew \$300.000000, New Balance: \$700.000000

Account 1004 not found!

Account: 1001, Holder: John Doe, Balance: \$600.00

Transaction History for Account 1001 (John Doe):

Deposited \$200.000000, New Balance: \$700.000000

Withdrew \$100.000000, New Balance: \$600.000000

Account: 1002, Holder: Jane Smith, Balance: \$700.00

Transaction History for Account 1002 (Jane Smith):

Withdrew \$300.000000, New Balance: \$700.000000

Account: 1003, Holder: Alice Johnson, Balance: \$300.00

Transaction History for Account 1003 (Alice Johnson):

Deposited \$50.000000, New Balance: \$300.000000

Account: 1001, Holder: John Doe, Balance: \$600.00

Transaction History for Account 1001 (John Doe):

Deposited \$200.000000, New Balance: \$700.000000

Withdrew \$100.000000, New Balance: \$600.000000

Account: 1002, Holder: Jane Smith, Balance: \$700.00

Transaction History for Account 1002 (Jane Smith):

Withdrew \$300.000000, New Balance: \$700.000000

Account: 1003, Holder: Alice Johnson, Balance: \$300.00

Transaction History for Account 1003 (Alice Johnson):

Deposited \$50.000000, New Balance: \$300.000000

Account: 1001, Holder: John Doe, Balance: \$600.00

Transaction History for Account 1001 (John Doe):

Deposited \$200.000000, New Balance: \$700.000000

Withdrew \$100.000000, New Balance: \$600.000000

Account: 1002, Holder: Jane Smith, Balance: \$700.00

Transaction History for Account 1002 (Jane Smith):

Withdrew \$300.000000, New Balance: \$700.000000

Account: 1003, Holder: Alice Johnson, Balance: \$300.00

Transaction History for Account 1003 (Alice Johnson):

Deposited \$50.000000, New Balance: \$300.000000



Challenges Faced and How I Solved Them

One of the main challenges I faced was managing dynamic memory using pointers for both the list of accounts and each account's transaction history. Initially, I encountered memory issues due to forgetting to deallocate the memory with `delete[]` in the destructor. I fixed this by carefully implementing proper memory cleanup in the ATM class to avoid memory leaks.

Another difficulty was implementing **binary search** correctly. At first, the search returned wrong results because the list of accounts wasn't sorted. I solved this by making sure accounts were added in order, allowing binary search to function correctly.

Finally, formatting file output with proper precision and structure was tricky. I used `setprecision()` and fixed from the `<iomanip>` library to ensure money values displayed as \$100.00 instead of \$100.

Despite these challenges, I was able to complete the project successfully by breaking problems into smaller steps, testing frequently, and researching solutions online.



Conclusion

This project allowed me to apply core C++ concepts in a practical and structured way. By building an ATM simulator with classes, pointers, dynamic memory, file handling, as i learn from my class and also add some search algorithms, I was able to simulate a real-world banking system using fundamental programming skills. The use of object-oriented design helped organize the logic clearly, and file output allowed for persistent transaction records.

Through this process, I strengthened my understanding of memory management and program structure. Despite facing challenges with dynamic arrays and formatting output, I was able to solve them effectively and complete a fully functional, well-documented program. This project has boosted my confidence in using C++ to build larger, real-world systems.