Customer Relationship Management (CRM) System - Detailed Project Summary

Project Overview

This Customer Relationship Management (CRM) system was developed for a fictional electronics store to manage customer information, track sales transactions, and provide role-based access for employees. The system implements core Object-Oriented Programming (OOP) principles, efficient data structures, and various C++ features to create a functional and optimized application suitable for a second-year computer engineering project.

System Architecture

Class Hierarchy

The system is built around a carefully designed class hierarchy:

1. **User Class (Base)**
   - Handles authentication functionality
   - Stores username and password
   - Contains virtual methods for polymorphic behavior
2. **Employee Class (Derived from User)**
   - Extends User with employee-specific attributes
   - Manages customer interactions
   - Handles transaction viewing
3. **Admin Class (Derived from Employee)**
   - Further extends Employee with administrative capabilities
   - Manages employee records
   - Generates reports and analytics
4. **Customer Class**
   - Stores customer personal information
   - Contains a collection of transactions
   - Calculates customer metrics (total spending)
5. **Transaction Class**
   - Records purchase details

- Links to customer accounts
- Tracks monetary values

6. **CRMSystem Class**
   - Acts as the central controller
   - Manages overall system flow
   - Integrates all components

Data Flow

1. Data is loaded from text files at startup
2. User authentication determines access level
3. Menu system directs program flow based on user role
4. Operations modify in-memory data structures
5. Data is saved back to files on exit

Technical Implementation Details

OOP Principles Implementation

1. Encapsulation

- **Private Data Members**: All classes use private attributes with public getter/setter methods
- **Information Hiding**: Implementation details are hidden from external access
- **Example**: Customer class keeps customerId, name, email, and phone private, with controlled access via methods

2. Inheritance

- **Is-A Relationship**: Admin is-an Employee, Employee is-a User
- **Code Reuse**: Derived classes inherit and extend base functionality
- **Method Overriding**: displayInfo() is overridden in derived classes

3. Polymorphism

- **Virtual Functions**: displayInfo() is declared virtual in User class
- **Dynamic Binding**: User pointers can reference Employee or Admin objects
- **Runtime Behavior**: Correct version of methods called based on actual object type

4. Abstraction

- **Simplified Interfaces**: Complex operations encapsulated in simple method calls
- **Implementation Hiding**: Users of classes don't need to know internal workings

C++ Features Utilized

1. Operator Overloading

- **Stream Operators**: Overloaded `<<` and `>>` for Customer and Transaction classes
- **Format**: Streamlined data formatting for file I/O operations
- **Usage**: `customerFile << customer` for simple data persistence

2. Templates

- **Generic Programming**: Template functions for sorting and searching
- **Type Independence**: Same code works with different object types
- **Example**: `sortByName<T>()` works with both Customer and Employee vectors

3. STL Containers

- **std::vector**: Used for storing collections of customers, employees, and transactions
- **std::unordered_map**: Used for fast O(1) customer lookups by ID
- **Memory Management**: STL containers handle dynamic memory allocation

4. STL Algorithms

- **std::find_if**: Used for efficient searching
- **std::sort**: Used for sorting customers and transactions
- **std::copy_if**: Used for filtered searches

5. Exception Handling

- **try-catch Blocks**: Wrapped critical operations for robust error handling
- **std::exception**: Used standard exception hierarchy
- **Error Messages**: Specific error reporting for better debugging

6. File I/O

- **fstream**: Used for reading and writing persistent data
- **Serialization**: Simple text-based data serialization
- **Data Persistence**: Customer, employee, and transaction data stored between sessions

Data Management

Data Structures

1. **Vector-based Collections**:
   - `std::vector<Customer> customers`: Main customer repository
   - `std::vector<Employee> employees`: Employee records
   - `std::vector<Admin> admins`: Administrator records
   - `std::vector<Transaction>`: Per-customer transaction history
2. **Hash-based Lookups**:
   - `std::unordered_map<int, Customer*> customerMap`: O(1) customer lookups by ID

Data Persistence

- **File Format**: Simple space-delimited text format
- **File Organization**:
  - customers.txt: ID, name, email, phone
  - employees.txt: username, password, ID, name, position
  - admins.txt: username, password, ID, name
  - transactions.txt: transactionID, customerID, date, amount, productName

Search and Sort Capabilities

- **Name-based Searching**: Case-sensitive substring matching
- **ID-based Lookup**: O(1) hash-based lookup for customers
- **Sorting Options**: By name (alphabetical), by spending (numerical)

Functional Modules

Authentication System

- Username/password verification
- Role determination (Employee vs Admin)
- Session management (current user tracking)

Customer Management

- Add new customers with unique IDs
- Modify existing customer information

- Remove customers from the system
- Search for customers by name or ID
- Display comprehensive customer information

Transaction Processing

- Record new purchases with product details
- Associate transactions with customer accounts
- Calculate customer spending metrics

Employee Management (Admin only)

- Add new employee accounts
- Remove employee accounts
- Employee role assignment

Reporting System (Admin only)

- Top customers by spending
- Customer count summary
- Basic analytics

Menu System

1. **Main System Menu**:
   - Login
   - Exit
2. **Employee Menu**:
   - Manage Customers
   - View Transactions
   - Add Transaction
   - Logout
3. **Admin Menu**:
   - Manage Customers
   - Manage Employees
   - Generate Reports
   - Add Transaction

  - Logout

4. **Customer Management Submenu**:
   - Add Customer
   - Remove Customer
   - Search Customer
   - Modify Customer
   - Display All Customers

Project Strengths & Implementation Highlights

Code Organization

- **Separation of Concerns**: Each class has a single, well-defined responsibility
- **Modularity**: Components can be modified independently
- **Maintainability**: Clear structure makes code easier to maintain and extend

Optimization Techniques

- **Fast Lookups**: O(1) hash map lookup for customer ID searches
- **Efficient Algorithms**: STL algorithms for searching and sorting
- **Memory Management**: Appropriate use of references to avoid unnecessary copying

User Experience

- **Intuitive Menus**: Clearly structured menu system
- **Error Handling**: Robust error messages for invalid operations
- **Data Validation**: Basic input validation for critical operations

Security Features

- **Role-based Access Control**: Functionality restricted based on user role
- **Authentication**: Basic username/password verification

Development Approach

- **Top-down Design**: Started with high-level class design
- **Incremental Development**: Built core classes first, then integrated them
- **Test-driven Development**: Tested individual components before integration

Technical Specifications

- **Language**: C++17
- **Compilation**: Compatible with modern C++ compilers (GCC/G++)
- **Dependencies**: Standard C++ library only
- **Platform**: Cross-platform compatible

Testing Strategy

- **Unit Testing**: Individual class testing
- **Integration Testing**: Testing interaction between components
- **System Testing**: End-to-end workflow testing
- **Test Cases**:
  - Customer addition, modification, removal
  - Transaction processing
  - Employee management
  - Authentication
  - Report generation

Future Enhancements

1. **Security Improvements**:
   - Password hashing and encryption
   - Session timeouts
   - Input sanitization
2. **Feature Extensions**:
   - Customer categorization (regular, premium, etc.)
   - Product inventory management
   - Discount and promotion handling
   - Email notifications
3. **Technical Improvements**:
   - Database integration (SQLite or MySQL)
   - Multi-threading for concurrent operations
   - Advanced reporting with data visualization
   - JSON/XML data format

- GUI interface

4. **Performance Optimizations**:
   - Caching frequently accessed data
   - More efficient search algorithms for large datasets
   - Batch processing for transactions

Documentation Standards

- **Header Comments**: Purpose, author, date
- **Function Documentation**: Purpose, parameters, return values
- **Implementation Notes**: Complex algorithm explanations
- **User Guide**: System operation instructions

Conclusion

This CRM system demonstrates a comprehensive application of Object-Oriented Programming principles and effective use of C++ features. The modular design ensures maintainability and extensibility while meeting all the core functional requirements of a basic Customer Relationship Management system. The implementation is appropriate for a second-year computer engineering project, balancing complexity with readability and educational value.