Store Management System - Technical Documentation

Date: May 24, 2025
Authors: Suci Ianis Luca & Tomodan Xeno Tudor
Language: C++
Project Type: Programming Project

Project Overview

The Store Management System is a C++ console application with two separate programs that work together to manage a store's inventory and customer orders.

Team Responsibilities

- **Student 1 (Ion Popescu):** Store Management (Application 1)
  - Add, delete, modify products
  - View orders
- **Student 2 (Maria Popescu):** Customer Interface (Application 2)
  - Manage shopping cart
  - Place orders

Key Features

- Inventory management with real-time stock updates
- Shopping cart functionality
- Order processing and tracking
- File-based data storage

System Requirements

- **Operating System:** Windows, Linux, or macOS
- **Compiler:** G++ with C++11 support
- **Memory:** 256MB RAM minimum
- **Storage:** 10MB for application and data files

Compilation

```
g++ -std=c++11 -o app_1.exe main1.cpp
g++ -std=c++11 -o app_2.exe main2.cpp
```

Data Structures

Product Class

```
class Product {
private:
   string barcode;    // Unique identifier
   string name;       // Product name
   int quantity;      // Stock quantity
   double price;      // Price per unit
public:
   Product(string bc, string n, int q, double p);
   string toString();
};
```

Date Class

```
class Date {
private:
   int day, month, year;
public:
   Date(int d, int m, int y);
   string toString();  // Returns "DD/MM/YYYY"
};
```

Order Class

```
class Order {
private:
   Product[] products;  // Array of ordered products
   Date date;          // Order date
public:
   Order(Product[] prods, Date d);
};
```

File System

The system uses three text files for data storage:

1. stoc.txt (Stock File)

Stores all product information:

```
<number of products>
<barcode1> <name1> <quantity1> <price1>
<barcode2> <name2> <quantity2> <price2>
...
```

Example:

```
3
001 Laptop 5 999.99
002 Mouse 25 29.99
003 Keyboard 15 79.99
```

2. comenzi.txt (Orders File)

Stores order history:

```
<order1 date>
<barcode1> <barcode2> <barcode3>
<order2 date>
<barcode4> <barcode5>
...
```

Example:

```
24/05/2025
001 002
25/05/2025
003 003 002
```

3. cos_cumparaturi.txt (Shopping Cart File)

Stores current shopping cart:

```
<barcode1> <quantity1>
<barcode2> <quantity2>
...
```

Example:

```
001 2
003 1
```

Application Commands

Application 1 (Store Management)

View Stock

```
./app_1.exe view_stock_products
```
Displays all products with their details.

Add Product
```
./app_1.exe add_product <barcode> <name> <quantity> <price>
```
Example: `./app_1.exe add_product 004 "USB_Cable" 50 12.99`

Delete Product
```
./app_1.exe delete_product <barcode>
```
Example: `./app_1.exe delete_product 004`

Modify Product
```
./app_1.exe modify_product <price|quantity> <barcode> <new_value>
```
Examples:

- `./app_1.exe modify_product price 001 899.99`
- `./app_1.exe modify_product quantity 002 30`

View Orders
```
./app_1.exe view_orders
```
Shows all placed orders with dates and products.

Application 2 (Customer Interface)

View Shopping Cart
```
./app_2.exe view_cart
```
Displays current cart contents and total price.

Add to Cart
```
./app_2.exe add_product <barcode> <quantity>
```
Example: `./app_2.exe add_product 001 2`

Modify Cart Item
```
./app_2.exe modify_product <barcode> <new_quantity>
```
Example: `./app_2.exe modify_product 001 3`

Remove from Cart
```
./app_2.exe delete_product <barcode>
```
Example: `./app_2.exe delete_product 001`

Purchase Cart
```
./app_2.exe purchase
```
Converts cart to order and updates stock.

Implementation Guide

Core Functions for Application 1

```
void viewStockProducts() {
  // Read from stoc.txt
  // Display formatted product list
}
void addProduct(string barcode, string name, int quantity, double price) {
  // Check if product exists
  // If exists: update quantity
  // If new: add to stock
  // Save to stoc.txt
}
void deleteProduct(string barcode) {
  // Find product in stock
  // Remove from vector
  // Update stoc.txt
}
void modifyProduct(string type, string barcode, double newValue) {
  // Find product
  // Update price or quantity
  // Save changes
}
```

```
void viewOrders() {
   // Read from comenzi.txt
   // Display formatted order history
}
```

Core Functions for Application 2

```
void viewCart() {
   // Read from cos_cumparaturi.txt
   // Calculate and display total
}
void addToCart(string barcode, int quantity) {
   // Check stock availability
   // Add/update cart item
   // Save to cos_cumparaturi.txt
}
void purchase() {
   // Validate cart contents
   // Check stock availability
   // Create order in comenzi.txt
   // Update stock in stoc.txt
   // Clear cart
}
```

File I/O Helper Functions

```
vector<Product> loadStock() {
   ifstream file("stoc.txt");
   vector<Product> products;
   // Read and parse file
   return products;
}
void saveStock(vector<Product>& products) {
   ofstream file("stoc.txt");
   file << products.size() << endl;
   for(auto& p : products) {
      file << p.toString() << endl;
   }
}
```

Testing

Test Scenarios

Application 1 Tests

1. **Add Product Test**
   - Add new product
   - Add existing product (should update quantity)
   - Verify file updates
2. **Modify Product Test**
   - Change price
   - Change quantity
   - Invalid barcode handling
3. **Delete Product Test**
   - Delete existing product
   - Delete non-existing product

Application 2 Tests

1. **Cart Management Test**
   - Add products to cart
   - Modify quantities

    - Remove products
2. **Purchase Test**
    - Purchase with sufficient stock
    - Purchase with insufficient stock
    - Empty cart purchase

Sample Test Data

Initial stoc.txt:

```
3
001 Laptop 10 999.99
002 Mouse 50 29.99
003 Keyboard 25 79.99
```

Test Commands:

```
# Test adding product
./app_1.exe add_product 004 "Monitor" 5 299.99
# Test cart operations
./app_2.exe add_product 001 2
./app_2.exe add_product 002 1
./app_2.exe purchase
```

Error Handling

Common Errors and Solutions

1. **File Not Found**
    - Create empty data files on first run
    - Check file permissions
2. **Invalid Barcode**
    - Validate barcode format
    - Check product existence
3. **Insufficient Stock**
    - Verify availability before purchase
    - Display appropriate error message
4. **Invalid Input**
    - Validate command-line arguments
    - Check data types and ranges

Error Handling Implementation

```
bool isValidBarcode(string barcode) {
   return !barcode.empty() && barcode.length() <= 10;
}
bool hasEnoughStock(string barcode, int requestedQty) {
   Product* product = findProduct(barcode);
   return product && product->quantity >= requestedQty;
}
```

Conclusion

This Store Management System provides a simple yet effective solution for basic retail operations. The two-application approach ensures clear separation of administrative and customer functions while maintaining data consistency through shared files.

Key Benefits

- Easy to understand and maintain
- File-based storage (no database required)
- Clear command-line interface
- Separate admin and customer functions
- Real-time stock management

Future Improvements

- Add user authentication
- Implement data validation
- Add GUI interface
- Database integration
- Multi-user support