Super Market System – Software Documentation

Version: 1.0
Author: (Analysis of code from *Super market.cpp*)
Date: 12/19/2025

1. Introduction

The SuperMarket System is a console-based C++ application that simulates a simple supermarket point-of-sale (POS) and inventory management system.
It provides a dual-mode interface:

- **User Mode:**
  Allows customers to view available products, add products to a shopping cart, and generate a final receipt with a total cost.
- **Manager Mode:**
  A password-protected section that allows a manager to add new products to the store, edit existing product names, and update product prices.

The application runs entirely in the console and uses in-memory data storage, meaning all changes are lost when the program terminates.

2. System Architecture

- Language: C++
- Platform: Windows Console (due to the use of <conio.h> for the getch() function)
- Data Storage: In-memory global variables (C-style arrays and std::vector). There is no database or file-based persistence.

Key Data Structures

- **string products[20] :**A fixed-size C-style array holding the names of the products.
- **int prices[20] :** A fixed-size C-style array holding the prices for each product. Each price index corresponds to the product index.
- **int menu_size:** A global integer that tracks the current number of active products.
- **name_manager, password_manager** :Hardcoded global variables storing the manager credentials.
- **vector<int> bag (Global) :**This global vector is unused. A local vector with the same name is declared inside buy_product(), shadowing the global variable.

3. Module & Function Breakdown

This section explains the purpose and functionality of each major function.

3.1 Main Execution

int main()

Purpose: Acts as the main entry point and controls the primary application loop.

Functionality:

1. Displays a welcome screen and waits for user input using getch().
2. Shows the main menu: User, Manager, or Exit.
3. Uses a switch statement to navigate to user_page() or manager_page().
4. Repeats until the user selects Exit.

3.2 User Mode Functions

void user_page()

Purpose: Handles the customer interface.

Functionality:

1. Prompts the user to enter their name (not reused).
2. Displays the user menu:
    - Show the menu
    - Buy a product
    - Exit
3. Calls show_menu() or buy_product() accordingly.
4. Continues until the user exits.

void show_menu()

Purpose: Displays all available products and prices.

Functionality:

1. Iterates from i = 0 to menu_size.
2. Prints each product in the format:
   (i+1) - [Product Name] [Price]

void buy_product()

Purpose: Manages the shopping cart and receipt generation.

Functionality:

1. Declares a local vector<int> bag to store selected product indices (1-based).
2. Displays a submenu:
    - Add products to cart
    - Show the receipt
    - Return to main
3. Add Products:
    - Asks for the number of products.
    - Clears the bag.
    - Displays the menu and records chosen product numbers.
4. Show Receipt:
    - If the bag is empty, notifies the user and recalls buy_product().
    - Otherwise, prints a formatted receipt and calculates the total price.

3.3 Manager Mode Functions

void manager_page()

Purpose: Handles manager authentication.

Functionality:

1. Displays the manager menu:
    - Create new email
    - Login
    - Return to main
2. Calls create() or login().

void create()

Purpose:
Simulates creating a new manager account.

Functionality:

1. Requests a manager name, password, and years of experience.
2. Rejects creation if experience is less than 3 years.
3. Confirms credentials by re-entry.
4. Grants immediate access by calling manager_process().

Limitation: The new credentials are not saved and cannot be reused for login later.

void login()

Purpose:
Authenticates a manager.

Functionality:

1. Prompts for username and password.
2. Compares input with hardcoded credentials.
3. On success, calls manager_process().

void manager_process()

Purpose: Manager dashboard.

Functionality:

1. Displays manager actions:
    - Add product to menu
    - Edit product
    - Return
2. Calls add() or edit().

void add()

Purpose: Adds new products to the catalog.

Functionality:

1. Prompts for a new menu_size.
2. Ensures the size is greater than the original.
3. Adds product names and prices.
4. Displays the updated menu.

void edit()

Purpose: Edits product name or price.

Functionality:

1. Allows choosing between editing name or price.
2. Updates the selected product.
3. Displays the updated menu.

4. How to Compile and Run

1. Prerequisites: Any C++ compiler (g++, MinGW, MSVC).
2. Compile:
3. g++ "Super market.cpp" -o supermarket.exe
4. Run:
5. supermarket.exe

Note: The program is intended for Windows due to <conio.h> usage.

5. Known Issues & Limitations

1. No data persistence — all changes are lost on exit.
2. Manager credentials are hardcoded and insecure.
3. Newly created managers cannot log in later.
4. Product list limited to 20 items.
5. Global variable shadowing (bag).
6. Recursive call in buy_product() may cause stack issues.
7. Minor UI text inconsistencies were present in the original code.