Supermarket Inventory Management System

Project Report

Struct 122
Ronil Kag
Rajdev Singh
Anand S Menon

November 25, 2025

Abstract

This report documents the design and development of a C-based Supermarket Management System. The project utilizes dynamic memory allocation and file handling to create a persistent inventory tool for store managers and customers.

Contents

1 Project Description

The Supermarket Management System is a terminal-based software application designed to digitize the inventory tracking process for retail stores. Traditional pen-and-paper methods are prone to human error and data loss; this project solves that problem by creating a digital database of products.

We have built a system that utilizes a **Linked List** as its core data structure. This allows the inventory to be dynamic—growing as new products are added and shrinking as they are deleted—without the fixed-size limitations of standard arrays. The system persists data between sessions by saving the inventory to a binary file (`supermarket.dat`) upon exit and reloading it upon startup.

2 Detailed Functionalities

The project offers two primary categories of functionality: Administrative and Operational.

2.1 Administrative Features

- **Inventory Creation:** Administrators can add new items to the database. The system captures the Product Name, Price, and Quantity, and automatically assigns a unique ID.
- **Stock Visualization:** Users can view the entire list of products in a formatted table, showing ID, Name, Price, and current Quantity.
- **Data Maintenance:** The system allows for updating existing product details (e.g., changing prices or restocking) and deleting obsolete items from the inventory.

2.2 Customer Features

- **Purchase System:** A simulation of the checkout process. Customers select items by ID and specify a quantity. The system calculates the total bill and automatically deducts the purchased amount from the global inventory stock.

2.3 System Features

- **Data Persistence:** The system automatically loads data when the program starts and saves data when the user chooses to exit, ensuring no changes are lost.

3 Implementation Details

This section details the algorithmic approach used to implement the key functionalities.

3.1 Adding a Product

1. **Allocation:** The system uses `malloc` to reserve memory for a new `Product` node.
2. **ID Generation:** The `get_ID` helper function traverses the existing list to find the highest current ID and returns `highest + 1`.
3. **Linking:** The new node's `next` pointer is set to the current head of the list, and the head pointer is updated to point to the new node (Insertion at Head, $O(1)$ complexity).

3.2 Deleting a Product

1. **Search:** The system iterates through the linked list looking for the user-provided ID.
2. **Unlinking:**
   - If the target is the head, the head pointer is moved to the second node.
   - If the target is in the middle, the `previous` node's `next` pointer is set to skip the target node.
3. **Deallocation:** The memory of the unlinked target node is released using `free()`.

3.3 File Persistence (Save/Load)

1. **Saving:** The system opens `supermarket.dat` in binary write mode (`wb`). It iterates through the list, writing the raw memory content of each struct to the file.
2. **Loading:** The system opens the file in binary read mode (`rb`). It reads struct-sized chunks in a loop. For every chunk read, it allocates a new node in the linked list and populates it with the file data.

4 Breakdown of Contributions

The development of this project was divided among the team members as follows:

Team Member	**Contributions**
Ronil Kag	Implemented the `create` and `display` functions; designed the struct definition in `data.h`.
Rajdev Singh	Implemented `update` and `delete` logic; handled pointer manipulation for linked list operations.
Anand S Menon	Developed File I/O (`load_dbfile`, `save_dbfile`) and the `main` menu loop.

Table 1: Task Distribution

5 Function Explanations

Below is an explanation of every function used in the codebase, detailing its input, output, and role.

File: main.c

- `menu()`
  - *Role:* Displays the available options (1-5) to the user.
  - *Input/Output:* Takes no arguments; returns `void`. Prints text to stdout.
- `main(void)`
  - *Role:* The entry point of the program. It handles the main infinite loop, input validation, and switch-case logic to call other functions.
  - *Input/Output:* Takes no arguments; returns integer status (0 for success).

File: crud.c

- `load_dbfile()`
  - *Role:* Reads the binary database file on startup and reconstructs the linked list in memory.
  - *Input/Output:* Takes no arguments; returns `void`. Modifies the global `product_lis_head`.
- `get_ID()`
  - *Role:* Scans the list to find the highest existing ID to ensure the next ID generated is unique.
  - *Input/Output:* Takes no arguments; returns an `int` (the new unique ID).
- `create()`
  - *Role:* Allocates memory for a new node, takes user input for details, and adds it to the front of the list.
  - *Input/Output:* Takes no arguments (uses `scanf`); returns `void`.
- `display()`
  - *Role:* Traverses the linked list and prints the details of every product in a tabular format.
  - *Input/Output:* Takes no arguments; returns `void`. Prints to console.
- `update_product()`
  - *Role:* Finds a product by ID and allows the user to overwrite its Name, Price, or Quantity.

  - *Input/Output:* Takes no arguments (prompts for ID); returns `void`.

- `delete_product()`
  - *Role:* Removes a node from the linked list and frees the associated memory to prevent leaks.
  - *Input/Output:* Takes no arguments (prompts for ID); returns `void`.

- `customer()`
  - *Role:* Handles sales. Reduces the quantity of a specific item based on user purchase amount.
  - *Input/Output:* Takes no arguments; returns `void`. Output is the final bill amount.

- `save_dbfile()`
  - *Role:* Writes the current state of the linked list to the hard drive for persistence.
  - *Input/Output:* Takes no arguments; returns `void`. Writes to `supermarket.dat`.

- `free_memory()`
  - *Role:* Iterates through the entire list and frees every node before the program exits.
  - *Input/Output:* Takes no arguments; returns `void`.