Project Report

GROCERY MANAGEMENT SYSTEM

Name : AGRIMA KHANDELWAL
SAP ID : 590026826
B.Tech CSE
Batch : B-45
Subject : C PROGRAMMING
Submitted To : MRS. DOLLY DAS

GitHub Repository Link for Project
https://github.com/agrima-192/Grocery-Management-System.git

Abstract

This report documents the design and implementation of a C-language based **Grocery Management System (GMS)**. The system is designed to efficiently manage inventory, process customer transactions, and track sales data for a small grocery store using a command-line interface. The GMS features a highly modular structure, utilizing separate components for inventory management, shopping cart operations, and bill generation. Data persistence is achieved through binary file I/O, storing inventory items (struct Grocery) and temporary cart items (struct CartItem) in grocery.dat and cart.dat, respectively. The system ensures stock validation during the addition of items to the cart, guaranteeing real-time inventory oversight and data integrity. The primary objective was to create a robust, menu-driven application that provides full CRUD (Create, Read, Update, Delete) functionality for stock and facilitates a precise checkout procedure.

Problem Definition

Background

In a typical small-to-medium grocery environment, manual tracking of inventory, price changes, and stock levels is prone to errors, time-consuming, and makes accurate sales and transaction analysis difficult. The lack of a centralized system often leads to stockouts, difficulty in item traceability, and inefficient customer service at the point of sale.

System Goal

The primary goal of the Grocery Management System is to **computerize the entire operational workflow** from stock maintenance to final billing. This includes:

1. **Inventory Management:** Maintaining a persistent record of all grocery items (ID, name, price, quantity) with full CRUD (Create, Read, Update, Delete) functionality.
2. **Transaction Processing:** Allowing users to add multiple items to a virtual shopping cart, with validation against current stock levels.
3. **Billing Generation:** Calculating the total cost of the cart and displaying a final, itemized bill, simultaneously clearing the cart file.
4. **Persistence:** Ensuring data integrity by storing inventory data persistently using binary files.

System Design

Data Structures

The system uses two primary structures defined in grocery.h to manage data:

```
struct Grocery {
   int id;
   char name[50];
int quantity;
   float price;
};

struct CartItem {
   int id;
   char name[50];
int quantity;
   float price;
};
```

- struct Grocery: Represents an item in the permanent **Inventory**.
- struct CartItem: Represents a temporary item added to the **Shopping Cart** for the current transaction.

Modular Design

The modular separation of concerns is critical for maintenance:

- **Inventory Module (inventory.c):** Handles all master data and stock CRUD operations.
- **Cart Module (cart.c):** Manages the temporary transaction list and performs stock validation and deduction.
- **Billing Module (billing.c):** Handles final cost calculation and transaction cleanup.

Implementation Details

Main Menu Implementation (main.c)

The main.c file provides the central navigation hub, handling user input and dispatching control to the respective module functions.

```
// Code snippet from main.c

int main() {
   int choice;

   while (1) {
      printf("\n========== GROCERY MANAGEMENT SYSTEM ==========\n");
      // Menu options 1 through 9...
      // ...
      scanf("%d", &choice);
      getchar();

      switch (choice) {          case 1:
addItem(); break;          case 6:
addToCart(); break;          case 8:
generateBill(); break;          case 9:
printf("Exiting...\n"); exit(0);
         default: printf("Invalid choice!\n"); pauseProgram();
      }
   }
   return 0;
}
```

Core Inventory Functions (inventory.c)

Adding a New Item (addItem())

The function opens the inventory file in append binary mode ("ab") and uses fwrite() to serialize the new struct Grocery record directly to the end of the file.

```
// Code snippet for addItem() function void
addItem() {
   FILE *fp = fopen(INV_FILE, "ab");     if (!fp) {
printf("Error opening file!\n"); return; }     // ...
input prompts ...
   fwrite(&g, sizeof(g), 1, fp);     fclose(fp);
   printf("\nItem added successfully!\n");
pauseProgram();
}
```

Updating an Item (updateItem())

This function uses the rb+ mode (read/write binary) to allow modification of data in place. After finding the matching ID, fseek() is used to move the file pointer back by the size of one structure (-sizeof(g)), enabling the updated structure to overwrite the old record.

```
// Code snippet for updateItem() function void
updateItem() {
   FILE *fp = fopen(INV_FILE, "rb+");
   // ... search for ID ...
   while (fread(&g, sizeof(g), 1, fp)) {
      if (g.id == id) {
         // ... input new data ...
         fseek(fp, -sizeof(g), SEEK_CUR);
         fwrite(&g, sizeof(g), 1, fp);

         printf("\nItem updated!\n");
         break;
      }    }
```

Deleting an Item (deleteItem())

Deletion is handled by reading all records from the source file (INV_FILE) and writing all records *except* the targeted item into a temporary file (temp.dat). The original file is then removed, and the temporary file is renamed to the original inventory filename, ensuring logical deletion.

Transaction Functions (cart.c & billing.c)

Adding to Cart with Stock Validation (addToCart())

This function is crucial for transaction integrity. It opens the inventory file in rb+ mode and the cart file in ab mode. It first validates if the requested quantity (qty) is less than or equal to the inventory's current quantity (g.quantity). If valid, it writes the item to the cart, then uses fseek and fwrite to deduct the quantity from the original inventory file immediately.

Generating the Bill and Cleanup (generateBill())

The bill function reads all records from the CART_FILE, calculates the total for each item and the final grand total, and prints the itemized receipt. After displaying the bill, it uses the standard C library function remove(CART_FILE) to delete the cart file, resetting the system for the next transaction.

Conclusion & Future Work

Conclusion

The Grocery Management System project successfully utilized C programming concepts, including complex file handling using fseek and binary mode operations, structured data types, and modular programming, to solve a real-world inventory and sales management problem. The system provides a reliable and efficient command-line solution for core grocery operations.

Future Work

Potential enhancements for future versions include:

1. **In-Memory Data Structure:** Migrating from file-only persistence to loading data into a **Linked List** in memory at startup, which would significantly improve search and update speed for large inventories. Changes would only be written back to the file on exit.
2. **Advanced Reporting:** Adding functions to track sales history (by writing bill data to a separate log file), calculate daily/monthly profits, and implement automatic low-stock alerts.
3. **User Authentication:** Implementing a simple login system for different user roles (e.g., administrator, cashier) to restrict access to sensitive CRUD operation