PROJECT REPORT

On

“Ration Store Management”

Submitted by:

Mehul Avdesh Shukla (CS23095)

Guided by:

Mrs. Roshani Talmale

DEPARTMENT OF FIRST YEAR ENGINEERING

S. B. JAIN INSTITUTE OF TECHNOLOGY MANAGEMENT AND RESEARCH, NAGPUR

(2023-2024)

Contents

Introduction:

The purpose of this program is to serve as a simple inventory management system for a Ration store. It allows users to perform various operation related to managing product information, processing sales, and maintaining transaction history.

This program gives some features:

- **Add product:** This function allows user to add products in the inventory with the type, brand, product id and price.

- **Display inventory:** This function allows to see current inventory with the following info like brand etc.

- **Search product:** This function allows user to search product in the inventory either using product id or name.
  It comprises of character by character search for easy and efficient functioning.

- **Process sale:** This function allows user to make sale and sell his added product it also calculate the total money and save it.

- **Display Transaction History:** The program can display the transaction history, showing details of all transactions, including the product name, quantity, and total amount.

- **Save and Exit**: Users can save the current inventory and transaction data to files before exiting the program.

Overall, the program provides a basic framework for managing the inventory of a Ration store, keeping track of product details, processing sales, and maintaining a record of transactions. It can be used as a starting point for a more comprehensive inventory management system with additional features and improvements.

Major Library and Functions:

Structures:

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

```
struct Transaction {
    int productId;
    int quantity;
    float totalAmount;
};
```

•***struct Product:*** Represents information about a product, including its name, brand, ID, quantity, and price.

•***struct Transaction:*** Represents information about a transaction, including the product ID, quantity, and total amount.

Arrays:

```
struct Product inventory[MAX_PRODUCTS];
struct Transaction transactions[MAX_TRANSACTIONS];
```

•inventory: An array to store product information.

•transactions: An array to store transaction information.

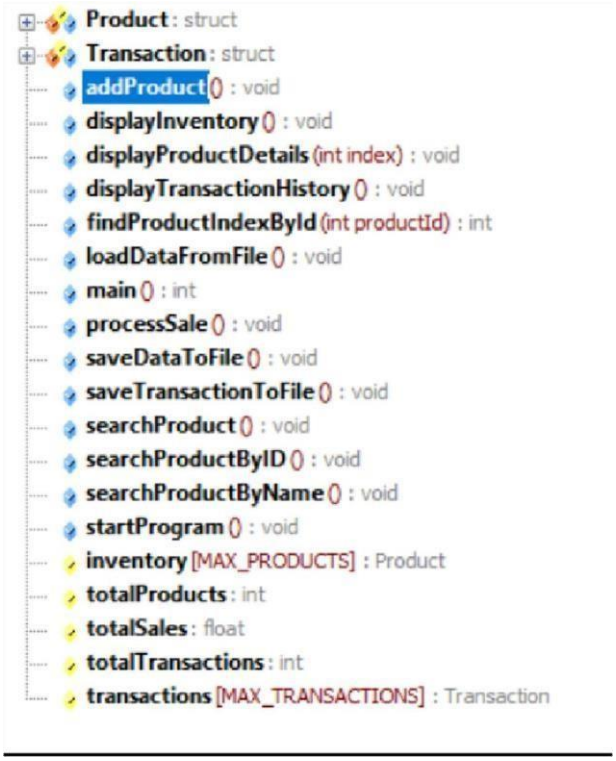
Constants:

```
#define MAX_PRODUCTS 100
#define MAX_TRANSACTIONS 100
#define FILENAME "inventory.txt"
#define TRANSACTION_FILENAME "transactions.txt"
```

•MAX_PRODUCTS and MAX_TRANSACTIONS: Maximum number of products and transactions the system can handle.

•FILENAME and TRANSACTION_FILENAME: File names for storing product and transaction data.

Functions:



1. findProductIndexById(int productId)

int findProductIndexById(int productId)

- Purpose: Finds the index of a product in the inventory array based on its ID.
- Iterates through the inventory array to search for the specified productId.
- Returns the index if found, otherwise returns -1.

2. saveDataToFile()

void saveDataToFile()

- Purpose: Saves product data and total sales to a file.
- Opens the file specified by FILENAME for writing.
- Writes product information and total sales to the file.

3. saveTransactionToFile()

void saveTransactionToFile()

- Purpose: Saves transaction data to a file.
- Opens the file specified by TRANSACTION_FILENAME for writing.

• Writes transaction information to the file.

4. loadDataFromFile()

void loadDataFromFile()

• Purpose: Loads product and transaction data from files.

• Opens the file specified by FILENAME for reading.

• Reads product information and total sales from the file.

• Opens the file specified by TRANSACTION_FILENAME for reading.

• Reads transaction information from the file.

5. displayTransactionHistory()

void displayTransactionHistory()

• Purpose: Displays transaction history, including product details.

• Prints a table of all transactions with product details.

6. addProduct()

void addProduct()

• Purpose: Adds a new product to the inventory.

• Takes user input for product details.

• Increments totalProducts and adds the new product to the inventory array.

7. displayInventory()

void displayInventory()

• Purpose: Displays the current inventory.

• Prints a table of all products with their details.

8. searchProduct()

void searchProduct()

• Purpose: Allows searching for a product by ID or name.

• Presents a menu to choose between searching by ID or name.

• Calls either searchProductByID() or searchProductByName() based on user choice.

9. searchProductByID()

void searchProductByID()

- Purpose: Searches for a product by ID.
- Takes user input for the product ID.
- Calls findProductIndexById() and displays product details if found.

10. searchProductByName()

void searchProductByName()

- Purpose: Searches for a product by name.
- Takes user input for the product name.
- Compares each character of the input with product names in a case-insensitive manner.
- Displays product details if a match is found.

11. displayProductDetails(int index)

void displayProductDetails(int index)

- Purpose: Displays details of a specific product.
- Takes the index of the product in the inventory array and prints its details.

12. processSale()

void processSale()

- Purpose: Processes a sale, updating inventory and transaction history.
- Takes user input for the product ID and quantity.
- Validates the quantity and updates the transactions and inventory arrays accordingly.

13. startProgram()

void startProgram()

- Purpose: The main function to start the program.
- Calls loadDataFromFile() to load existing data.
- Displays a menu for user interaction.
- Performs actions based on user choices until the user decides to exit.

14. main()

int main() { startProgram(); return 0; }

- Calls the startProgram() function to initiate the main program loop.

Source Code:

```
#include   <stdio.h>
#include   <stdlib.h>
#include <string.h>

#define MAX_PRODUCTS 100
#define MAX_TRANSACTIONS 100
#define FILENAME "inventory.txt"
#define TRANSACTION_FILENAME "transactions.txt"

struct Product {
    char name[50];
    char brand[50];
    int id;
    int quantity;
    float price;
};

struct Transaction {
    int productId;
    int quantity;
    float totalAmount;
};
```

```
struct Product inventory[MAX_PRODUCTS];
struct Transaction transactions[MAX_TRANSACTIONS];
int totalProducts = 0;
int totalTransactions = 0;
float totalSales = 0.0;

// Function to find the index of a product by ID
int findProductIndexById(int productId) {
    int i;
    for (i = 0; i < totalProducts; i++) {
        if (inventory[i].id == productId) {
            return i;
        }
    }
    return -1;
}

// Function to save product data to a file
void saveDataToFile() {
    int i;
    FILE *file = fopen(FILENAME, "w");
    if (file == NULL) {
        printf("Error opening file for writing.\n");
        return;
    }
```

```
    fprintf(file, "%d\n", totalProducts);
    for (i = 0; i < totalProducts; i++) {
        fprintf(file, "%s %s %d %d %.2f\n", inventory[i].name, inventory[i].brand,
inventory[i].id,
                inventory[i].quantity, inventory[i].price);
    }

    fprintf(file, "%.2f\n", totalSales);

    fclose(file);
}
// Function to save transaction data to a file
void saveTransactionToFile() {
    int i;
    FILE *file = fopen(TRANSACTION_FILENAME, "w");
    if (file == NULL) {
        printf("Error opening transaction file for writing.\n");
        return;
    }
    for (i = 0; i < totalTransactions; i++) {
        fprintf(file, "%d %d %.2f\n", transactions[i].productId,
transactions[i].quantity, transactions[i].totalAmount);
    }
    fclose(file);
}
// Function to load data from files
void loadDataFromFile() {
    int i;
```

```
    FILE *file = fopen(FILENAME, "r");
    if (file == NULL) {
        printf("No previous  data found.\n");
        return;
    }
    fscanf(file, "%d", &totalProducts);
    for (i = 0; i < totalProducts; i++) {
        fscanf(file, "%s %s %d %d %f", inventory[i].name,  inventory[i].brand,
&inventory[i].id,
                &inventory[i].quantity,  &inventory[i].price);
    }
    fscanf(file, "%f", &totalSales);  // Load total sales
    fclose(file);
    file  =  fopen(TRANSACTION_FILENAME,  "r");
    if (file  == NULL) {
        printf("No previous transaction data found.\n");
        return;
    }
    fscanf(file, "%d", &totalTransactions);
    for (i = 0; i < totalTransactions;  i++) {
        fscanf(file, "%d %d %f", &transactions[i].productId,
&transactions[i].quantity,  &transactions[i].totalAmount);
    }
    fclose(file);
}
// Function to display transaction history
void displayTransactionHistory()  {
    int i;
```

```
    printf("\n=========================================\n");
    printf("          Ration Store Management\n");
    printf("=========================================\n");

    printf("All Transactions:\n");
    printf("_____________________________________________\n");
    printf("| %-4s | %-20s | %-10s | %-6s |\n", "ID", "Product Name", "Quantity",
"Total Amount");
    printf("_____________________________________________\n");

    for (i = 0; i < totalTransactions; i++) {
        int productId = transactions[i].productId;
        int quantity = transactions[i].quantity;
        float totalAmount = transactions[i].totalAmount;
        int productIndex = findProductIndexById(productId);
        if (productIndex != -1) {
            printf("| %-4d | %-20s | %-10d | Rs.%-6.2f |\n", productId,
inventory[productIndex].name, quantity, totalAmount);
        }
    }
}
// Function to add a new product to the inventory
void addProduct() {
    int i;
    if (totalProducts < MAX_PRODUCTS) {
        printf("Enter product name: ");
        scanf("%s", inventory[totalProducts].name);
```

```
        printf("Enter product brand: ");
        scanf("%s", inventory[totalProducts].brand);

        printf("Enter product ID: ");
        scanf("%d", &inventory[totalProducts].id);

        printf("Enter quantity: ");
        scanf("%d", &inventory[totalProducts].quantity);

        printf("Enter price: ");
        scanf("%f", &inventory[totalProducts].price);

        totalProducts++;
        printf("Product added successfully!\n");
    } else {
        printf("Inventory is full. Cannot add more products.\n");
    }
}
// Function to display the current inventory
void displayInventory() {
    int i;
    printf("\n=========================================\n");
    printf("          Ration Store Management\n");
    printf("=========================================\n");

    printf("Inventory:\n");
    printf("_____________________________________________________\n");
```

```
    printf("| %-4s | %-20s | %-15s | %-8s | %-6s |\n", "ID", "Name", "Brand",
"Quantity", "Price");
    printf("_____\n");

    for (i = 0; i < totalProducts; i++) {
        printf("| %-4d | %-20s | %-15s | %-8d | Rs.%-6.2f |\n", inventory[i].id,
inventory[i].name, inventory[i].brand,
               inventory[i].quantity, inventory[i].price);
    }
}

// Function to search for a product by ID or Name
void searchProduct() {
    int choice;
    printf("\n=========================================\n");
    printf("          Rashan Store Management\n");
    printf("=========================================\n");

    printf("1. Search by ID\n");
    printf("2. Search by Name\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            searchProductByID();
            break;
        case 2:
```

```
            searchProductByName();
            break;
        default:
            printf("Invalid  choice.\n");
    }
}

// Function to search for a product by ID
void searchProductByID() {
    int productId;
    printf("Enter product ID to search: ");
    scanf("%d", &productId);

    int productIndex = findProductIndexById(productId);
    if (productIndex != -1) {
        displayProductDetails(productIndex);
    } else {
        printf("Product not found in the inventory.\n");
    }
}

// Function to search for a product by Name
void searchProductByName() {
    char searchName[50];
    printf("Enter product name to search: ");
    scanf("%s", searchName);
```

```
    int i, j,matchFound;
    for (i = 0; i < totalProducts; i++) {
        matchFound = 1; // Assume match until proven otherwise

        // Compare each character of searchName with the product name
        for ( j = 0; searchName[j] != '\0'; j++) {
            if (tolower(searchName[j]) != tolower(inventory[i].name[j])) {
                matchFound = 0; // Characters do not match
                break;
            }
        }

        // If a match is found, display product details and return
        if (matchFound) {
            displayProductDetails(i);
            return;
        }
    }

    printf("Product not found in the inventory.\n");
}


// Function to display details of a specific product
void displayProductDetails(int index) {
    printf("\n=========================================\n");
    printf("          Ration Store Management\n");
```

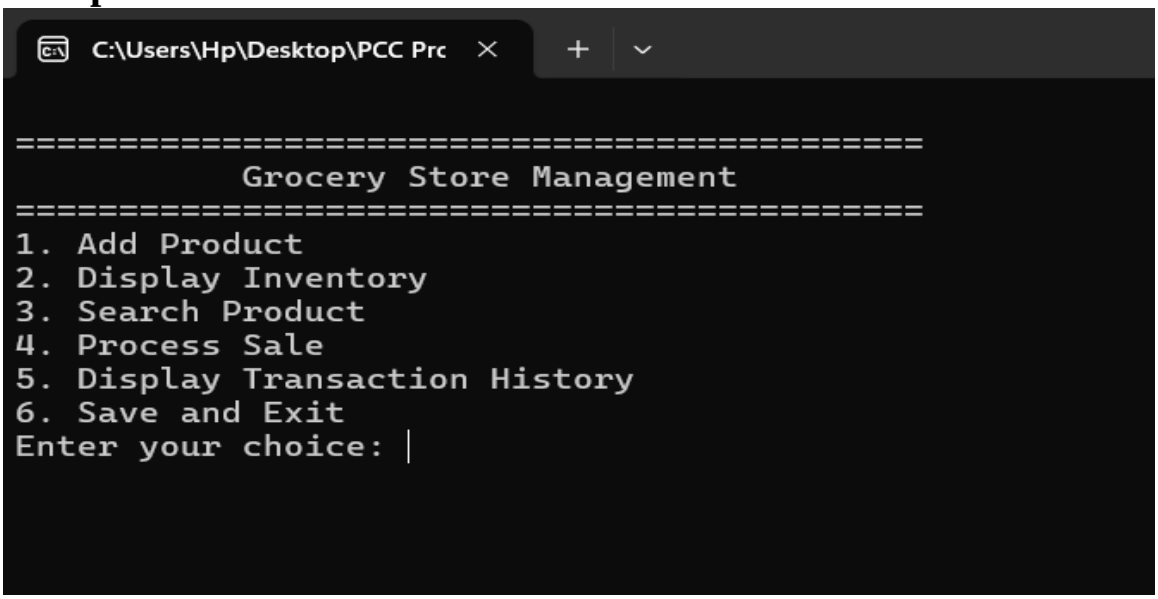
```
    printf("=========================================\n");

    printf("Product found:\n");
    printf("ID:%d\n", inventory[index].id);
    printf("Name:%s\n",  inventory[index].name);
    printf("Brand:%s\n",  inventory[index].brand);
    printf("Quantity:%d\n", inventory[index].quantity);
    printf("Price:  Rs.%.2f\n", inventory[index].price);
}

// Function to process a sale
void processSale()  {
    int productId, quantity, productIndex;
    float totalAmount;

    printf("Enter product ID: ");
    scanf("%d", &productId);

    productIndex  =  findProductIndexById(productId);

    if (productIndex != -1) {
        printf("Enter quantity: ");
        scanf("%d", &quantity);

        if (quantity > 0 && quantity <= inventory[productIndex].quantity) {
            totalAmount = quantity * inventory[productIndex].price;
            transactions[totalTransactions].productId  = productId;
```

```
            transactions[totalTransactions].quantity  = quantity;
            transactions[totalTransactions].totalAmount  = totalAmount;
            totalTransactions++;

            // Update inventory
            inventory[productIndex].quantity -= quantity;
            totalSales += totalAmount;

            printf("Sale processed successfully!\n");
        } else {
            printf("Invalid quantity. Sale not processed.\n");
        }
    } else {
        printf("Product not found. Sale not processed.\n");
    }
}

// Function to start the main program
void startProgram() {
    loadDataFromFile();

    int choice;

    do {

printf("\n=========================================\n");
        printf("          Ration Store Management\n");
        printf("=========================================\n");
```

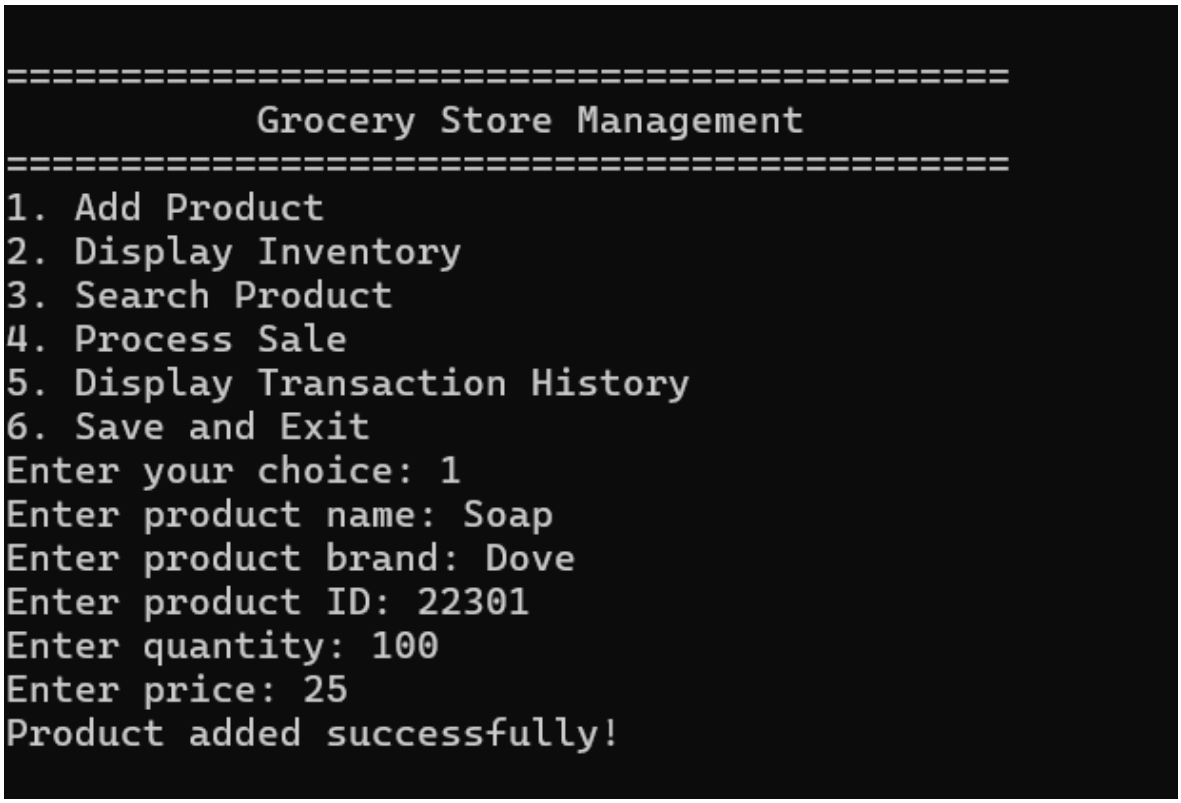
```
printf("1. Add Product\n");
printf("2. Display Inventory\n");
printf("3. Search Product\n");
printf("4. Process Sale\n");
printf("5. Display Transaction History\n");
printf("6. Save and Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        addProduct();
        break;
    case 2:
        displayInventory();
        break;
    case 3:
        searchProduct();
        break;
    case 4:
        processSale();
        break;
    case 5:
        displayTransactionHistory();
        break;
    case 6:
```

```
            saveDataToFile();
            saveTransactionToFile();
            printf("Exiting the program. Data saved. Goodbye!\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
    }
  } while (choice != 6);
}
int main() {
  startProgram();
  return 0;
}
```

Output:



Press 1: Adding the product with their type, brand, id and price



Press 2: Shows the items present in the inventory file

```
==========================================
          Grocery Store Management
==========================================
1. Add Product
2. Display Inventory
3. Search Product
4. Process Sale
5. Display Transaction History
6. Save and Exit
Enter your choice: 2

==========================================
          Grocery Store Management
==========================================
Inventory:
-------------------------------------------------------------------------
| ID    | Name                | Brand           | Quantity | Price   |
-------------------------------------------------------------------------
| 22301 | Soap                | Dove            | 100      | Rs.25.00 |
```

Press 3: Search the items present in the inventory file

```
==========================================
1. Add Product
2. Display Inventory
3. Search Product
4. Process Sale
5. Display Transaction History
6. Save and Exit
Enter your choice: 3

==========================================
          Grocery Store Management
==========================================
1. Search by ID
2. Search by Name
3. Search by Brand
Enter your choice: 2
Enter product name to search: Soap

==========================================
          Grocery Store Management
==========================================
Product found:
ID:22301
Name:Soap
Brand:Dove
Quantity:100
Price: Rs.25.00
```

Press 4: Proccess sale in the items present in the inventory file

```
==========================================
           Grocery Store Management
==========================================
1. Add Product
2. Display Inventory
3. Search Product
4. Process Sale
5. Display Transaction History
6. Save and Exit
Enter your choice: 4
Enter product ID: 22301
Enter quantity: 100
Sale processed successfully!
```

Press 5: Shows the transaction in the items present in the inventory file

```
==========================================
          Grocery Store Management
==========================================
1. Add Product
2. Display Inventory
3. Search Product
4. Process Sale
5. Display Transaction History
6. Save and Exit
Enter your choice: 5

==========================================
          Grocery Store Management
==========================================
All Transactions:
-------------------------------------------------------------------------
| ID    | Product Name         | Quantity    | Total Amount |
-------------------------------------------------------------------------
| 22301 | Soap                 | 100         | Rs.2500.00 |
```

Conclusion:

In conclusion, the Ration Store Management program provides a user-friendly interface for managing inventory, processing sales, and maintaining transaction history. With features such as adding products, displaying inventory, searching for products by ID, name, or brand, processing sales, and viewing transaction history, the program offers a comprehensive solution for efficient Ration store management.

the Ration Store Management program serves as a valuable tool for small to medium-sized Ration stores, aiding in efficient inventory management, sales processing, and transaction tracking.

References:

1. https://chat.openai.com
2. https://google.com
3. https://openai.com