# Designing model:

Departmental Store Management

We made a departmental store management system which manage 5 stores:

1. Crafted Curiosities: Handcraft
2. The Modern Maven: Home Decor
3. MegaMart Express: Hyper Market
4. GadgetHub: Technology Gadgets
5. Bookworm Corner: Book Shop

It mange it’s main and userOption function are the main of our system with other **11 functions, 5 functions for the store, 2 function for files and 4 functions for customer.**

**Main functions:**

**Description:** This function initializes the program by loading data from a file and calling the `userOptions()` function to display the main menu and perform actions based on user input.

**Input:** None

**Output:** 0 (indicating successful program completion)

**Steps:**

1. Initialize the storeNo variable to 0.

2. Call the `loadFromFile()` function to load data from a file into the `myStores` variable, passing in the `storeNo` variable as an argument.

3. Call the `userOptions()` function, passing in the `storeNo` variable as an argument.

4. Return 0 to indicate successful program completion.

**Function: userOptions**

**Description:** This function displays a menu of options for the user to choose from and performs the corresponding action based on the user's input.

**Input:**

- An integer representing the store number (storeNo)

**Output:** None

**Steps:**

1. Display a menu of options for the user to choose from and ask the user to enter their choice using the scanf() function.

2. If the user chooses to terminate the program, print a termination message and exit the program using the exit() function.

3. If the user enters an invalid choice, print an error message and continue to the next iteration of the loop.

4. Ask the user to enter the store number they want to perform the action on using the scanf() function.

5. If the user enters an invalid store number, print an error message and continue to the next iteration of the loop.

6. If the user chooses to terminate the program, print a termination message and exit the program using the exit() function.

7. Based on the user's choice, perform the corresponding action:

a. If the user chooses to add a good, call the AddGood() function with the appropriate arguments.

b. If the user chooses to display goods, call the ListSoldGoodQuantities() function with the appropriate arguments.

c. If the user chooses to sell goods, call the Sold() function with the appropriate arguments.

d. Ifthe user chooses to display sold-out goods, call the ListSoldOutGood() function with the appropriate arguments.

e. If the user chooses to search for a good, call the searchGoods() function with the appropriate arguments.

f. If the user chooses to add a customer, create a new customer struct and call the AddCustomer() function with the appropriate arguments.

g. If the user chooses to edit a customer, ask the user to enter the customer ID and call the EditCustomer() function with the appropriate arguments.

h. If the user chooses to display customers, call the DisplayCustomers() function with the appropriate arguments.

i. If the user chooses to delete a customer, ask the user to enter the customer ID and call the DeleteCustomer() function with the appropriate arguments.

j. If the user chooses to save data, call the saveToFile() function with the appropriate arguments.

k. If the user chooses to load data, call the loadFromFile() function with the appropriate arguments.

l. If the user enters an invalid choice, print an error message and continue to the next iteration of the loop.

**5 functions for the store:**

**Function: AddGood**

**Description:** This function adds a new good to a Store.

**Input:**

- A pointer to a Store struct (Stores)

- An integer representing the store number (storeNo)

- A pointer to a goods struct (Goods)

**Output:** None

**Steps:**

1. Ask the user to enter the name, quantity, and price of the new good using the scanf() function.

2. Get the current quantity of goods in the store and use it as the index for the new good.

3. Copy the name of the new good to the name field of the goods array in the store.

4. Assign the correct quantity and price to the new good.

5. Increment the quantity variable in the Store.

6. Print a message indicating that the good has been added successfully.

**Function: ListSoldGoodQuantities**

**Description:** This function lists all the goods in a Store along with their quantities and prices.

**Input:**

- A pointer to a Store struct (stores)

- An integer representing the store number (storeNo)

**Output:** None

**Steps:**

1. Print a header row for the goods table.

2. Iterate over the goods array in the store and print the quantity, price, and name of each good.

**Function: searchGoods**

**Description:** This function searches for a good with a specified name in a Store and displays its details.

**Input:**

- A pointer to a Store struct (stores)

- An integer representingthe store number (storeNo)

**Output:** None

**Steps:**

1. Ask the user to enter the name of the good to search using the scanf() function.

2. Iterate over the goods array in the store and compare the name of each good with the search name using the strcmp() function.

3. If a matching good is found, print its details (name, price, and available quantity) and set the found flag to 1.

4. If no matching good is found, print a message indicating that the good was not found.

**Function: Sold**

**Description:** This function allows the user to sell a specified quantity of a good from a Store.

**Input:**

- A pointer to a Store struct (Stores)

- An integer representing the store number (storeNo)

**Output:** None

**Steps:**

1. Ask the user to enter the name of the good to sell using the scanf() function.

2. Iterate over the goods array in the store and compare the name of each good with the search name using the strcmp() function.

3. If a matching good is found, ask the user to enter the quantity to sell using the scanf() function.

4. If the quantity to sell is greater than the available quantity of the good, print an error message and return from the function.

5. If the quantity to sell is less than or equal to the available quantity of the good, subtract the quantity to sell from the available quantity of the good.

6. Print a message indicating thatthe quantity of the good has been sold and update the quantity field of the good in the Store.

7. If no matching good is found, print a message indicating that the good was not found.

**Function: ListSoldOutGood**

**Description:** This function lists all the goods in a Store that have a quantity of 0.

**Input:**

- A pointer to a Store struct (stores)

- An integer representing the store number (storeNo)

**Output:** None

**Steps:**

1. Print a header row for the sold-out goods table.

2. Iterate over the goods array in the store and check if the quantity of each good is 0.

3. If the quantity of a good is 0, print its quantity, price, and name.

**2 functions for the files :**

**Function: loadFromFile**

**Description:** This function reads data from a text file and populates a Store struct with the data.

**Input:**

- A pointer to a Store struct (stores)

- An integer representing the store number (storeNo)

**Output:** None

**Steps:**

1. Open the "store\_data.txt" file using the fopen() function.

2. Check if the file was successfully opened. If not, print an error message and return.

3. Read an integer value from the file using the fscanf() function and store it in a variable named "quantity". This value represents the number of goods stored in the store.

4. Set the "quantity" value for the store using the "storeNo" parameter.

5. For each good in the store, read the quantity, price, and name from the file using the fscanf() function and store them in the appropriate variables of the current good pointer.

6. Close the file using the fclose() function.

7. Print a message indicating that the data has been successfully loaded from the file.

**Function: saveToFile**

**Description:** This function writes data from a Store struct to a text file.

**Input:**

- A pointer to a Store struct (stores)

- An integer representing the store number (storeNo)

**Output:** None

**Steps:**

1. Open the "store\_data.txt" file using the fopen() function with the write mode ("w").

2. Write the quantity of goods in the store to the file using the fprintf() function.

3. For each good in the store, write the quantity, price, and name to the file using the fprintf() function.

4. Close the file using the fclose() function.

5. Print a message indicating that the data has been successfully saved to the file.

**4 functions for the customer:**

**Function: AddCustomer**

**Description:** This function adds a new customer to a CustomerList.

**Input:**

- A pointer to a CustomerList struct (Customers)

- A pointer to a customer struct (Customer)

Output: None

**Steps:**

1. Print a message asking the user to enter customer details.

2. Ask the user to enter the customer's ID, name, address, and phone number using the scanf() function.

3. Add the customer to the Customers array in the CustomerList using the quantity variable.

4. Increment the quantity variable in the CustomerList.

5. Print a message indicating that the customer has been added successfully.

**Function: EditCustomer**

**Description:** This function allows the user to edit the details of an existing customer in a CustomerList.

**Input:**

- A pointer to a CustomerList struct (Customers)

- An integer representing the ID of the customer to be edited (customerID)

**Output:** None

**Steps:**

1. Iterate over the Customers array in the CustomerList to find the customer with the specified ID.

2. If the customer is found, print a message asking the user to enter new details for the customer.

3. Print the current details of the customer and a list of options for the user to choose from (name, address, phone, or exit).

4. Ask the user to enter their choice using the scanf() function.

5. Use a switch statement to handle the user's choice and update theappropriate field of the customer struct.

6. Print a message indicating that the field has been updated successfully.

7. If the user chooses to exit, return from the function.

8. If the customer is not found, print a message indicating that the customer was not found.

**Function: DisplayCustomers**

**Description:** This function displays the details of all customers in a customer array.

**Input:**

- A pointer to an array of customer structs (customers)

- An integer representing the number of customers (numCustomers)

**Output:** None

**Steps:**

1. Print a header row for the customer table.

2. Iterate over the customer array and print the ID, name, and phone number of each customer.

**Function: DeleteCustomer**

**Description:** This function deletes a customer with a specified ID from a CustomerList.

**Input:**

- A pointer to a CustomerList struct (Customers)

- An integer representing the ID of the customer to be deleted (customerID)

**Output:** None

**Steps:**

1. Iterate over the Customers array in the CustomerList to find the customer with the specified ID.

2. If the customer is found, shift all subsequent customers in the array to the left by one position to overwrite the deleted customer.

3. Decrement the quantity variable in the CustomerList.

4. Print a message indicating that the customer has been deleted successfully.

5. If the customer is not found, print a message indicating that the customer was not found.

# Code:

//Departmental Store Management

 // including header files

    #include <stdio.h> // for printf & scanf

    #include <string.h> // for string & its functions (strcpy,strcmp,strcat)

    #include <stdlib.h> // for the system("cls") which literally clears the console to make it cleaner

    #define C "cls" // this is instead of system("cls") it will be system(c);

    typedef struct { // goods struct

        int quantity;

        float price;

        char name[50];

    } goods;

    typedef struct { // store struct

        int quantity;

        goods Goods[100];

    } Store;

    typedef struct { // customer struct

    int id;

    char name[50];

    char address[100];

    char phone[15];

} customer;

    typedef struct { // customerList struct

    int quantity;

    customer Customers[100];

} CustomerList;

// global struct variables initialized to 0 in order to be modified by all functionas

    CustomerList myCustomers = { 0 };

    customer myCustomer;

    Store myStores = { 0 };

    goods myGoods;

    //Function to add goods or product to store

    void AddGood(Store\* Stores, int storeNo, goods\* Goods);

    // Function to list the quantities of sold goods

    void ListSoldGoodQuantities(Store\* stores, int storeNo);

    // Function to display user options and handle their choices

    void userOptions(int storeNo);

    // Function to search for goods in the store

    void searchGoods(Store\* Stores, int storeNo);

    // Function to mark goods as sold

    void Sold(Store\* Stores, int storeNo);

    // Function to list sold-out goods

    void ListSoldOutGood(Store\* stores, int storeNo);

    //  Function to save file

    void saveToFile(Store\* myStores, int storeNo);

    // Function to load from file

    void loadFromFile(Store\* stores, int storeNo);

    // Function to add customer to file

    void AddCustomer(CustomerList\* Customers, customer\* Customer);

    // Function to edit customer info

    void EditCustomer(CustomerList\* Customers, int customerID);

    // Function to display customers

    void DisplayCustomers(customer\* customers, int numCustomers);

    // Function to delete a customer from the customer list

    void DeleteCustomer(CustomerList\* Customers, int customerID);

  int main() {

    int storeNo = 0;

    loadFromFile(&myStores, storeNo);//load data from file

    userOptions(storeNo);//start user interaction

    return 0;

        }

void userOptions(int storeNo){

    int choice;

    while (1) {

        //display the avaliable option that user have

        printf("\n\t\t\t\t\tSelect action:\n");

        printf("\n\t\t\t\t\t1. Add Goods\n");

        printf("\n\t\t\t\t\t2. Display Goods\n");

        printf("\n\t\t\t\t\t3. Sell Goods\n");

        printf("\n\t\t\t\t\t4. Display Sold Out Goods\n");

        printf("\n\t\t\t\t\t5. Search Goods\n");

        printf("\n\t\t\t\t\t6. Add Customer\n");

        printf("\n\t\t\t\t\t7. Edit Customer\n");

        printf("\n\t\t\t\t\t8. Display Customer\n");

        printf("\n\t\t\t\t\t9. Delete Customer\n");

        printf("\n\t\t\t\t\t10. Save Data\n");

        printf("\n\t\t\t\t\t11. Load Data\n");

        printf("\n\t\t\t\t\t12. Terminate :(\n");

        printf("\n\t\t\t\t\t---------------------------------------");

        printf("\n");

        printf("\t\t\t\t\tEnter Action : ");

        scanf("%d", &choice);

        system(C);//clear the console screen

        if(choice==12){

            system(C);

            printf("\n\n\n\t\t\t\t\t    Terminating StoreMasters program.......   \n\n\n");

            printf("\n\n\n\t\t\t\t\t    Thanks for using StoreMasters!   \n\n\n");

            exit(0);

            break;

        }

        else if(choice>12){system(C);printf("\t\t\t\t\t\tINVALID CHOICE!!\n");break;}

        //to keep the user to select a store

        printf("\n\t\t\t\t\tSelect store:\n");

        printf("\n\t\t\t\t\t1. Crafted Curiosities: Handcraft\n");

        printf("\n\t\t\t\t\t2. The Modern Maven: Home Decor\n");

        printf("\n\t\t\t\t\t3. MegaMart Express: Hyper Market\n");

        printf("\n\t\t\t\t\t4. GadgetHub: Technology Gadgets\n");

        printf("\n\t\t\t\t\t5. Bookworm Corner: Book Shop\n");

        printf("\n\t\t\t\t\t6. Terminate:((\n");

        printf("\n\t\t\t\t\t---------------------------------------");

        printf("\n");

        printf("\n\t\t\t\t\tEnter Store : ");

        scanf("%d", &storeNo);

        if (storeNo < 1 || storeNo >6) {

            printf("\t\t\t\t\t\tINVALID STORE NUMBER!!\n"); // validation incase the user enters a store number that doesn't exist

            continue;

        }

        else if(storeNo==6){

            printf("\n\n\n\t\t\t\t\t    Terminating StoreMasters program.......   \n\n\n");

            printf("\n\n\n\t\t\t\t\t    Thanks for using StoreMasters!!!   \n\n\n");

            exit(0);

        }

        // switch cases that call function, it's in infinite loop to trap the user until he basically enters a valid input & to finish his actions

        switch(choice){

        case 1:

            AddGood(&myStores, storeNo, &myGoods);

            break;

        case 2:

            system(C);

            ListSoldGoodQuantities(&myStores, storeNo);

            break;

        case 3:

            system(C);

            Sold(&myStores, storeNo);

            break;

        case 4:

            system(C);

            ListSoldOutGood(&myStores, storeNo);

            break;

        case 5:

            system(C);

            searchGoods(&myStores, storeNo);

            break;

        case 6:

           {

           customer newCustomer;

           AddCustomer(&myCustomers, &newCustomer);

           break;

        }

       case 7:

          {

    // declaring customerId and taking info.

    int customerID;

    printf("\n\t\t\t\t\tEnter customer ID to edit: ");

    scanf("%d", &customerID);

    EditCustomer(&myCustomers, customerID);

    break;

         }

      case 8:

      {

    DisplayCustomers(myCustomers.Customers, myCustomers.quantity);

    break;

       }

      case 9:

          {

    int customerID;

    printf("\n\t\t\t\t\tEnter customer ID to delete: ");

    scanf("%d", &customerID);

   DeleteCustomer(&myCustomers, customerID);

    break;

            }

        case 10:

            system(C);

            saveToFile(&myStores, storeNo);

            break;

        case 11:

            system(C);

            loadFromFile(&myStores, storeNo);

            break;

        default:

            system(C);

            printf("\n\t\t\t\t\tERROR! INVALID ACTION CHOICE!!!!"); // validation for customer

            continue;

        }

    }

}

void AddGood(Store\* Stores, int storeNo, goods\* Goods) {

        printf("\n\t\t\t\t\tEnter the name of the goods: ");

        scanf("%s", Goods->name);

        printf("\n\t\t\t\t\tEnter the quantity of the goods: ");

        scanf("%d", &(Goods->quantity));

        printf("\n\t\t\t\t\tEnter the price of the goods: ");

        scanf("%f", &(Goods->price));

        int index = Stores[storeNo - 1].quantity; // Get the current quantity as the index

        strcpy(Stores[storeNo - 1].Goods[index].name, Goods->name);

        Stores[storeNo - 1].Goods[index].quantity = Goods->quantity; // Assign the correct quantity

        Stores[storeNo - 1].Goods[index].price = Goods->price;

        Stores[storeNo - 1].quantity++;

    }

void ListSoldGoodQuantities(Store\* stores, int storeNo) {

        printf("\n\t\t\t\t\tGoods in Store #%d:\n", storeNo);

        printf("\n\t\t\t\t\tQuantity:\tPrice:\tName:\n");

        for (int i = 0; i < stores[storeNo - 1].quantity; i++) {

            goods\* currentGood = &(stores[storeNo - 1].Goods[i]);

            printf("\n\t\t\t\t\t%d\t\t%.2f\t%s\n", currentGood->quantity, currentGood->price, currentGood->name);

        }

    }

void searchGoods(Store\* stores, int storeNo) {

    // making user enter the goods name and searching for it with a loop

    char searchName[50];

    printf("\n\t\t\t\t\tEnter the name of the good to search: ");

    scanf("%s", searchName);

    int found = 0;

    goods\* currentGood;

    for (int i = 0; i < stores[storeNo - 1].quantity; i++) {

        currentGood = &(stores[storeNo - 1].Goods[i]);

        if (strcmp(currentGood->name, searchName) == 0) { // this is to check whether the user-entered name matches the actual name thats inside the program

            printf("\n\t\t\t\t\tStore: %d\n", storeNo);

            printf("\n\t\t\t\t\tName: %s\n", currentGood->name);

            printf("\n\t\t\t\t\tPrice: %.2f\n", currentGood->price);

            printf("\n\t\t\t\t\tAvailable quantity: %d\n", currentGood->quantity);

            found = 1;

            break;

        }

    }

    if (!found) {

        printf("\t\t\t\t\t\t\t\t\t\tGood not found!\n");

    }

}

void Sold(Store\* Stores, int storeNo) {

    //searching for the good name first to sell

    char searchName[50];

    printf("\n\t\t\t\t\tEnter the name of the good to sell: ");

    scanf("%s", searchName);

    int found = 0; // this is like a flag to update once goods are found

    goods\* currentGood;

    for (int i = 0; i < Stores[storeNo - 1].quantity; i++) {

        currentGood = &(Stores[storeNo - 1].Goods[i]);

        if (strcmp(currentGood->name, searchName) == 0) {

            found = 1;

            int sellQuantity;

            printf("\n\t\t\t\t\tEnter the quantity to sell: ");

            scanf("%d", &sellQuantity);

            if (sellQuantity > currentGood->quantity) {

                printf("\n\t\t\t\t\tERROR: Not enough quantity available!\n");

                return;

            }

            currentGood->quantity -= sellQuantity;

            printf("\n\t\t\t\t\tQuantity %d of %s sold from Store #%d.\n", sellQuantity, currentGood->name, storeNo);

            break;

        }

    }

    if (!found) {

        printf("\n\t\t\t\t\t Whoops!, good is not found!\n");

    }

}

void ListSoldOutGood(Store\* stores, int storeNo) {

    printf("\n\t\t\t\t\tSold out goods in Store #%d:\n", storeNo);

    printf("\n\t\t\t\t\tQuantity:\tPrice:\tName:\n");

    for (int i = 0; i < stores[storeNo - 1].quantity; i++) { // loop to display the sold out good

        goods\* currentGood = &(stores[storeNo - 1].Goods[i]);

        if (currentGood->quantity == 0) {

            printf("\n\t\t\t\t\t%d\t\t%.2f\t%s\n", currentGood->quantity, currentGood->price, currentGood->name);

        }

    }

}

void saveToFile(Store\* stores, int storeNo) {

    // this is incase the user wants to terminate and he want the data to be saved on his progress

    FILE\* fp;

    fp = fopen("store\_data.txt", "w");

    fprintf(fp, "%d\n", stores[storeNo - 1].quantity);

    for (int i = 0; i < stores[storeNo - 1].quantity; i++) {

        goods\* currentGood = &(stores[storeNo - 1].Goods[i]);

        fprintf(fp, "%d %f %s\n", currentGood->quantity, currentGood->price, currentGood->name);

    }

    fclose(fp);

    printf("\n\t\t\t\t\tData saved to file!\n");

}

void loadFromFile(Store\* stores, int storeNo) {

    // this is incase the user wants to terminate and he want the data to be  loaded on his progress

    FILE\* fp;

    fp = fopen("store\_data.txt", "r");

    if (fp == NULL) {

        printf("\n\t\t\t\t\tERROR: File not found!\n");

        return;

    }

    int quantity;

    fscanf(fp, "%d", &quantity);

    stores[storeNo - 1].quantity = quantity;

    for (int i = 0; i < quantity; i++) {

        goods\* currentGood = &(stores[storeNo - 1].Goods[i]);

        fscanf(fp, "%d %f %s", &(currentGood->quantity), &(currentGood->price), currentGood->name);

    }

    fclose(fp);

    printf("\n\t\t\t\t\tData loaded from file!\n");

}

void AddCustomer(CustomerList\* Customers, customer\* Customer) {

    // making user enter the customer details

    printf("\n\t\t\t\t\tEnter customer details:\n");

    printf("\n\t\t\t\t\tID: ");

    scanf("%d", &Customer->id);

    printf("\n\t\t\t\t\tName: ");

    scanf("%s", Customer->name);

    printf("\n\t\t\t\t\tAddress: ");

    scanf("%s", Customer->address);

    printf("\n\t\t\t\t\tPhone: ");

    scanf("%s", Customer->phone);

    Customers->Customers[Customers->quantity] = \*Customer;

    Customers->quantity++;

    printf("\n\t\t\t\t\tCustomer added successfully!\n");

}

void EditCustomer(CustomerList\* Customers, int customerID) {

    int i;

    for (i = 0; i < Customers->quantity; i++) { // loop to print all customer infos that were added by user

        if (Customers->Customers[i].id == customerID) {

            printf("\n\t\t\t\t\tEnter new details for customer %d:\n", customerID);

            printf("\n\t\t\t\t\t1. Name: %s\n", Customers->Customers[i].name);

            printf("\t\t\t\t\t2. Address: %s\n", Customers->Customers[i].address);

            printf("\t\t\t\t\t3. Phone: %s\n", Customers->Customers[i].phone);

            printf("\t\t\t\t\t4. Exit\n");

            int choice;

            do {

                printf("\n\t\t\t\t\tEnter your choice (1-4): ");

                scanf("%d", &choice);

                switch (choice) {

                    case 1:

                        printf("\n\t\t\t\t\tEnter new name: ");

                        scanf("%s", Customers->Customers[i].name);

                        printf("\n\t\t\t\t\tName updated successfully!\n");

                        break;

                    case 2:

                        printf("\n\t\t\t\t\tEnter new address: ");

                        scanf("%s", Customers->Customers[i].address);

                        printf("\n\t\t\t\t\tAddress updated successfully!\n");

                        break;

                    case 3:

                        printf("\n\t\t\t\t\tEnter new phone number: ");

                        scanf("%s", Customers->Customers[i].phone);

                        printf("\n\t\t\t\t\tPhone number updated successfully!\n");

                        break;

                    case 4:

                        return;

                    default:

                        printf("\n\t\t\t\t\tInvalid choice. Please try again.\n"); // validation for customer info choice.

                        break;

                }

            } while (choice < 1 || choice > 4);

        }

    }

    printf("\n\t\t\t\t\tCustomer not found!\n");

}

void DisplayCustomers(customer\* customers, int numCustomers){

    printf("\nCustomer ID\tName\t\tPhone Number\n");

    printf("------------------------------------------------\n");

    for (int i = 0; i < numCustomers; i++) // Loop to print the customer obv

    {

        printf("%d\t\t%s\t\t%s\n", customers[i].id, customers[i].name, customers[i].phone);

    }

}

void DeleteCustomer(CustomerList\* Customers, int customerID) {

    int i, j;

    for (i = 0; i < Customers->quantity; i++) {

        if (Customers->Customers[i].id == customerID) {

            for (j = i; j < Customers->quantity - 1; j++) {

                Customers->Customers[j] = Customers->Customers[j + 1];

            }

            Customers->quantity--;

            printf("\n\t\t\t\t\tCustomer deleted successfully!\n");

            return;

        }

    }

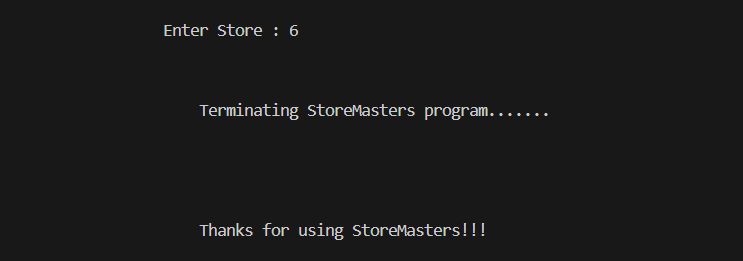
    printf("\n\t\t\t\t\tCustomer with ID %d not found.\n", customerID); // if the customer Id was not found

}

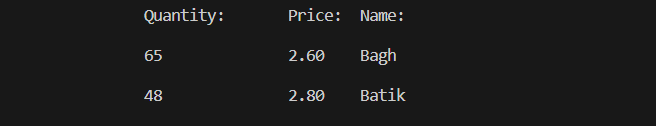
# Screenshot of project’s output:

A screenshot of a computer

Description automatically generated with medium confidence



Display Function:



A black background with white text

Description automatically generated with low confidence

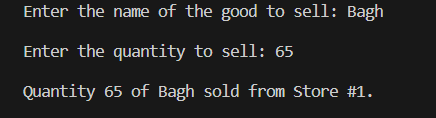


A screenshot of a computer

Description automatically generated with medium confidence

A black background with white text

Description automatically generated with low confidence



A black screen with white text

Description automatically generated with low confidence

Sold out function:

A black background with white text

Description automatically generated with low confidence