Inventory Management System

Project report submitted in partial fulfillment of the Requirements for the Award of the Degree

of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

24KB1A05LD THOKALA SUKUMAR

24KB1A05MW VATAMBETI RABORT SON

24KB1A05MY VATTURU HEMANTH KRISHNA

24KB1A05NG VEMPULURU PRADEEP

24KB1A0589 CHAMALA JAYASIMHA REDDY

Under the Guidance of

PADAVALA SUNEETHA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

N.B.K.R INSTITUTE OF SCIENCE AND TECHNOLOGY

N.B.K.R INSTITUTE OF SCIENCE AND TECHNOLOGY

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project report entitled INVENTORY MANAGEMENT SYSTEM being submitted by

24KB1A05LD THOKALA SUKUMAR
24KB1A05MW VATAMBETI RABORT SON
24KB1A05MY VATTURU HEMANTH KRISHNA
24KB1A05NG VEMPULURU PRADEEP
24KB1A0589 CHAMALA JAYASIMHA REDDY

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the N.B.K.R Institute Of Science And Technology is a record of bonafied work carried out under my guidance and supervision.

PADAVALA SUNEETHA

Designation

Dr. Ravinder Reddy
M.Tech, Ph.D
Head of the Department

DECLARATION

I hereby declare that the dissertation entitled **Inventory Management System** submitted for the B.Tech Degree is my original work and the dissertation has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

Place: Vidyanagar	24KB1A0589 CHAMALA JAYASIMHA REDDY
Date:	24KB1A05LD THOKALA SUKUMAR
	24KB1A05MY VATTURU HEMANTH KRISHNA
	24KB1A05MW VATAMBETI RABORT SON
	24KB1A05NG VEMPULURU PRADEEP

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who supported me throughout the development of this project,Inventory Management System using C. First and foremost, I am deeply thankful to my instructor, **Padavala Suneetha**, for guidance, encouragement, and constructive feedback, which played a vital role in the successful completion of this project.I would also like to thank my peers and friends for their valuable suggestions and support during the development and testing phases of the system. Their input helped me to improve and refine the program.Finally, I extend my thanks to my family for their constant motivation and support throughout the project.This project has provided me with a practical understanding of data structures in C, especially the use of arrays and linked lists in real-world applications.

ABSTRACT

This project involved the development of a simple yet functional Inventory Management System using C programming language,focusing on efficient use of fundamental data structures.The system was designed to manage products in a store and keep a record of sales transactions. The product catalog was implemented using a static array to store information such as product ID, name, quantity in stock, and unit price. This allowed for quick access and manipulation of product data. For tracking sales history, a singly linked list was used to dynamically store records of sold items, including the product ID, quantity sold, and total sale amount. Each sale was appended to the list, ensuring a chronological record of transactions.

Key functionalities of the system include:

- Adding new products to the inventory
- Viewing the current product catalog with available stock
- Processing sales transactions and updating stock levels accordingly
- Maintaining and displaying a dynamic sales history log

This project provided hands-on experience with essential programming concepts such as pointer manipulation, dynamic memory allocation, linked list operations, and menu-driven program design. It reinforced understanding of how core data structures can be applied to real-world problems in software development.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The **Inventory Management System** is a console-based application developed in the C programming language. It is designed to manage a product catalog and track sales efficiently using fundamental data structures. An array is used to store product information such as name, quantity, and price, while a linked list is implemented to dynamically record each sales transaction. The system allows users to add products, update stock, process sales, and view transaction history through a simple menu-driven interface. This project demonstrates practical use of arrays, structures, pointers, and dynamic memory allocation in C for solving real-world problems.

1.2 PROBLEM STATEMENT

Managing inventory effectively is a critical task for businesses that deal with physical products. Many small businesses still rely on manual methods or basic tools like spreadsheets for inventory tracking, which often leads to errors, inefficiency, and difficulty in maintaining accurate records. Existing software solutions tend to be expensive, complex, and over-featured for small-scale operations. There is a need for a simple, efficient, and cost-effective system to manage product inventory, process sales transactions, and track sales history. This project aims to develop an Inventory Management System using the C programming language that leverages arrays for managing product data and linked lists for dynamically tracking sales transactions, providing small businesses with a straightforward and reliable solution for inventory management.

1.3 SCOPE

The Inventory Management System developed in this project is a console-based application designed to meet the basic inventory tracking and sales transaction needs of small businesses. The system provides the following key functionalities:

- **Product Management**: Allows users to add, view, and update products in the inventory. Each product is identified by a unique ID and includes details such as name, quantity, and price.

- **Sales Tracking**: Facilitates the selling of products, updating stock levels automatically, and creating a detailed record of each sale with the quantity sold and total sale amount.

- **Sales History**: Maintains a dynamic record of all sales transactions using a linked list, enabling users to view and review past sales activities.

- **Menu-Driven Interface**: The system is designed to be operated through a simple text-based interface that offers options to interact with the inventory and sales data.

This project is intended for small-scale use and serves educational purposes, demonstrating the practical application of arrays, linked lists, and memory management in C. It is limited in scope to basic product and transaction management without advanced features such as reporting, analytics, or integration with databases. Future versions could expand to include these advanced functionalities.

1.4 OBJECTIVE

The primary objective of this Inventory Management System project is to develop a simple, efficient, and cost-effective solution for managing inventory and tracking sales using the C programming language. The key objectives are:

1. **Implement Core Data Structures**: Utilize arrays to manage the product catalog and linked lists to dynamically track sales transactions, demonstrating efficient use of fundamental data structures in C.

2. **Develop an Efficient Sales Tracking System**: Automate the process of tracking sales transactions, updating product quantities, and maintaining a detailed sales history.

3. **Design a User-Friendly Interface**: Create a menu-driven console interface that allows users to interact easily with the system, performing tasks such as adding products, selling items, and viewing transaction history.

4. **Enhance Understanding of Memory Management**: Focus on the application of dynamic memory allocation in linked lists to ensure memory efficiency in handling sales data.

5. **Provide a Solution for Small Businesses**: Offer a lightweight, practical tool for small businesses that need to manage their inventory and sales transactions without the complexity or cost of advanced enterprise systems

CHAPTER 2

LITERATURE SURVEY

S.NO	AUTHOR	TITLE/SOURCE	CONTRIBUTION
1	Brian W. Kernighan, Dennis M. Ritchie	*The C Programming Language*, Prentice Hall, 1988	Fundamental concepts of C programming, including syntax, pointers, arrays, and structures.
2	D.S. Malik	*C Programming: From Problem Analysis to Program Design*, Cengage Learning, 2014	Structured program design, modular development, and real-world problem-solving using C.
3	Abraham Silberschatz, Henry F. Korth, S. Sudarshan	*Database System Concepts*, McGraw-Hill, 2011	Concepts related to record keeping, storage of data, and basics of data management useful for future enhancements.
4	GeeksforGeeks	"Arrays in C" (https://www.geeksforgeeks.org/arrays-in-c-cpp/)	Explained array data structures used for maintaining the product catalog.
5	TutorialsPoint	"C - Linked Lists" (https://www.tutorialspoint.com/data_structures_algorithms/...)	Provided the implementation guide for singly linked lists used for tracking sold item history

			dynamically.
6	Wayne C. Booth	"Kenneth Burke's Way of Knowing", *Critical Inquiry*, 1974	Contributed to understanding structured argumentation and logical thinking for software methodology.
7	Jason Berry, Jonathan Foose, Tad Jones	*Up from the Cradle of Jazz*, U of Georgia P, 1986	Provided a formatting example for citation; not directly related to software but used as a model for referencing.

CHAPTER 3

SOFTWARE REQUIREMENT ANALYSIS

3.1 Functional Requirements

Functional requirements define the core operations and features that the system must support. For the Inventory Management System, the functional requirements are:

- **3.1.1: Product Management**
    - The system should allow the user to add new products to the inventory with details such as product ID, name, quantity, and price.
    - The user should be able to view a list of all products stored in the inventory.
- **3.12: Sales Transaction Handling**
    - The system should allow the user to process sales transactions, decreasing the quantity of the selected product based on the sale quantity.
    - Each sale should be recorded with details such as product ID, quantity sold, and total sale amount.
- **3.1.3: Sales History Management**
    - The system should maintain a record of all sales transactions using a linked list data structure. This history should include product ID, quantity sold, and total sale price.
    - The user should be able to view the entire sales history in a chronological order.
- **3.14: User Interface**
    - The system should provide a menu-driven interface, offering options to perform tasks such as adding products, selling products, and viewing product catalog or sales history.
    - The interface should be simple and intuitive, allowing users to interact easily with the system.

3.2 Non Functional Requirements

Non-functional requirements specify the overall system qualities and constraints. For the **Inventory Management System**, the non-functional requirements are:

- **3.2.1: Performance**
    - The system must be responsive, with quick load times for displaying product catalog and sales history.
    - Sales transactions should be processed immediately without noticeable delay.
- **3.2.2: Memory Usage**
    - The system should be optimized to use memory efficiently. Since linked lists are used to store sales data, dynamic memory allocation should be managed effectively to avoid memory leaks.
- **3.23: Portability**
    - The system should be portable across any platform that supports the C language and basic C libraries. It must work on any C compiler (e.g., GCC, Turbo C).
- **3.2.4: Scalability**
    - The system should be able to handle a reasonable number of products (up to 100) in the product catalog and an unlimited number of sales transactions. However, scaling beyond this number would require significant changes (e.g., database integration).
- **3.2.5: Usability**
    - The system must be simple to use, even for users with minimal technical knowledge. The menu system should be intuitive, guiding users through the necessary operations.

3.3 System Features

- **Feature 1: Add Product**
    - The system will allow users to input product information such as product ID, name, quantity, and price. Once added, the product will be stored in the product catalog array.
- **Feature 2: Sell Product**

    - Users will be able to process a sale by selecting the product ID and entering the quantity to sell. The system will check if the stock is sufficient and update both the inventory and sales history accordingly.

- **Feature 3: View Product Catalog**

    - The system will display a list of all products currently in the inventory, including product ID, name, quantity, and price.

- **Feature 4: View Sales History**

    - The system will display a list of all sales transactions, including the product ID, quantity sold, and total amount for each sale, maintaining the chronological order of sales.

3.4 Hardware Requirements

- **Minimum Hardware Requirements**:

    - A computer capable of running a C compiler (e.g., GCC, Turbo C).

    - A keyboard and monitor for user input and output.

- **Recommended Hardware**:

    - Any modern computer with sufficient memory (at least 1 GB RAM).

    - A modern C compiler such as GCC installed on the system.

3.5 Software Requirements

- **Operating System**: Any operating system that supports a C compiler, such as Windows, Linux, or macOS.

- **Compiler**: GCC or any other C compiler.

- **Libraries**: Standard C libraries (stdio.h, stdlib.h).

3.6 User Requirements

- **End-Users**: Small business owners, shop managers, or any individuals who need a simple solution for inventory and sales management.

- **User Skill Level**: Basic knowledge of how to operate a computer and follow menu options.

CHAPTER 4

SOFTWARE DESIGN

4.1 Overview of Design

The design of the Inventory Management System follows a **modular approach** where each functionality such as adding products, selling products, and viewing sales—is encapsulated in separate functions. This makes the program maintainable, extendable, and easy to debug.

Two primary data structures are used:

- **Array**: For maintaining the product catalog.
- **Linked List**: For dynamically storing sales history.

4.2 Design Goals

- Efficient memory usage using dynamic allocation for the sales list.
- Clear separation of logic for product operations and sales tracking.
- Error handling for invalid input and stock management.
- Scalable code structure for future expansion (e.g., file I/O, GUI, DB).

4.3 Modules in the System

Module 1: Product Management

- Add product
- Display product catalog

Module 2:Sales Handling

- Sell product
- Validate stock availability
- Update inventory
- Record sale

Module 3: Sales History

- Maintain linked list of all sales
- Display sales history

Module 4: Menu Interface

- Provide options for the user to navigate system functionalities

CHAPTER 5

PROPOSED SYSTEM

The Inventory Management System is designed using a modular approach to provide basic yet essential inventory and sales management features. Each module focuses on a specific responsibility, ensuring maintainability, clarity, and scalability. Below are the defined modules and their core functionalities:

5.1 System Overview

The system enables users to:

- Add new products to the inventory.
- Sell products and update stock levels.
- View the product catalog.
- Track and view sales history.

All data is managed using an array (for products) and a linked list (for sales).

5.2 Modules and Their Functionalities

1. Product Management Module

- **Functionality**:
    - Add products with unique IDs, names, quantities, and prices.
    - Display the list of all products in inventory.
- **Data Structure**: Array of structures (`Product[]`)
- **Key Functions**: `addProduct()`, `listProducts()`

2.Sales Processing Module

- **Functionality**:
    - Sell a product by specifying its ID and quantity.

    - Check stock availability and update inventory.
    - Calculate total sale value.
- **Data Structures**: Array (for product lookup), Linked List (for sales)
- **Key Functions**: `sellProduct()`, `addSale()`

3. Sales History Module

- **Functionality**:
    - Maintain a chronological record of sales transactions.
    - Store details like product ID, quantity sold, and total price using a linked list.
    - Allow the user to view all previous sales.
- **Data Structure**: Singly Linked List (`SoldItem*`)
- **Key Functions**: `viewSalesHistory()`, `createSaleNode()`

4. User Interface Module

- **Functionality**:
    - Display a menu to perform all available operations.
    - Handle user input and call the corresponding modules.
    - Ensure proper navigation and exit from the system.
- **Key Logic**: `main()` function with `switch-case` control

CHAPTER 6

CODING

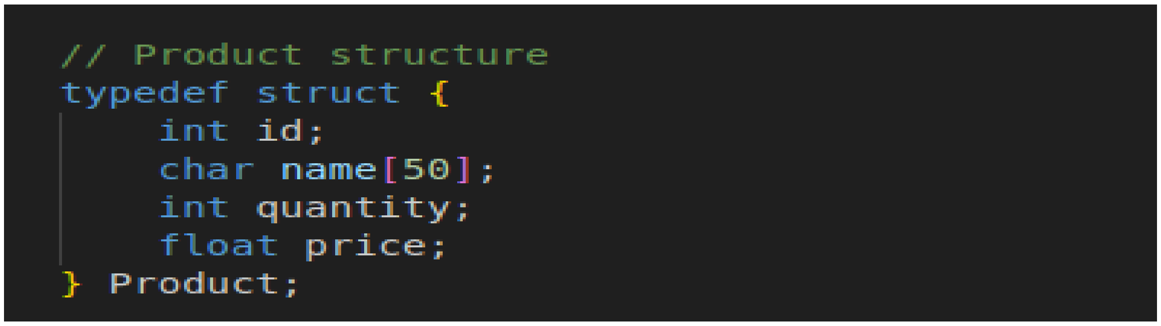
6.1 Code Structure

The program is implemented in a single C file (`inventory.c`) and uses the following data structures:

- **Array** of structures for the product catalog
- **Linked List** for managing the history of sold items

6.2 Data Structures

Product Structure



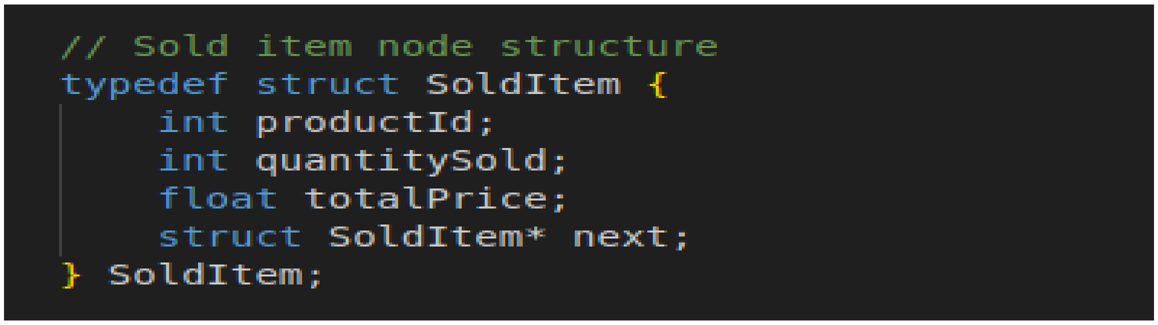
```c
// Product structure
typedef struct {
    int id;
    char name[50];
    int quantity;
    float price;
} Product;
```

Purpose: Stores information about each product.

Fields:

- `id`: Product ID (input)
- `name`: Product name (input)
- `quantity`: Current stock (input/output)
- `price`: Unit price (input)

Sales (Linked List) Structure


```c
// Sold item node structure
typedef struct SoldItem {
    int productId;
    int quantitySold;
    float totalPrice;
    struct SoldItem* next;
} SoldItem;
```

Purpose: Stores individual sale transactions.

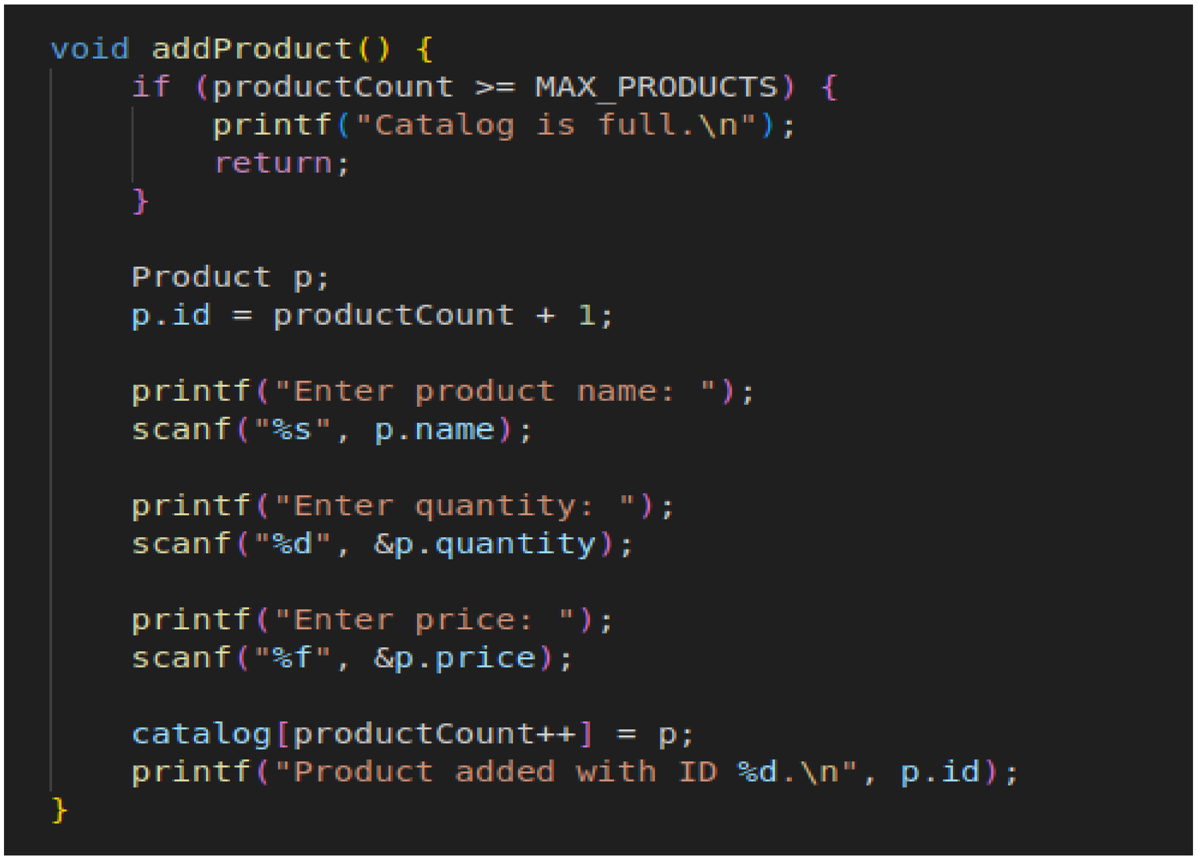
Fields:

- `productId`: ID of the product sold (input)
- `quantitySold`: Number of units sold (input)
- `totalPrice`: Sale amount calculated (output)
- `next`: Pointer to the next sale (internal use)

6.3 Functions and Methods

Function: `addProduct()`

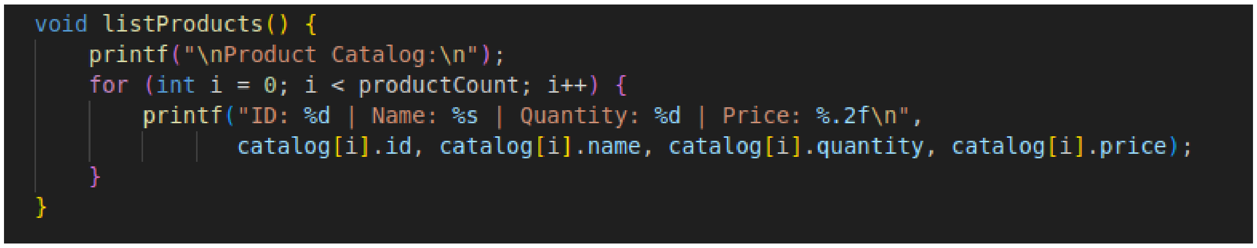
- **Input**: Product ID, name, quantity, price (via `scanf`)
- **Output**: Adds product to the array
- **Purpose**: Appends a product to the catalog.



```c
void addProduct() {
    if (productCount >= MAX_PRODUCTS) {
        printf("Catalog is full.\n");
        return;
    }

    Product p;
    p.id = productCount + 1;

    printf("Enter product name: ");
    scanf("%s", p.name);

    printf("Enter quantity: ");
    scanf("%d", &p.quantity);

    printf("Enter price: ");
    scanf("%f", &p.price);

    catalog[productCount++] = p;
    printf("Product added with ID %d.\n", p.id);
}
```

Function: `listProducts()`

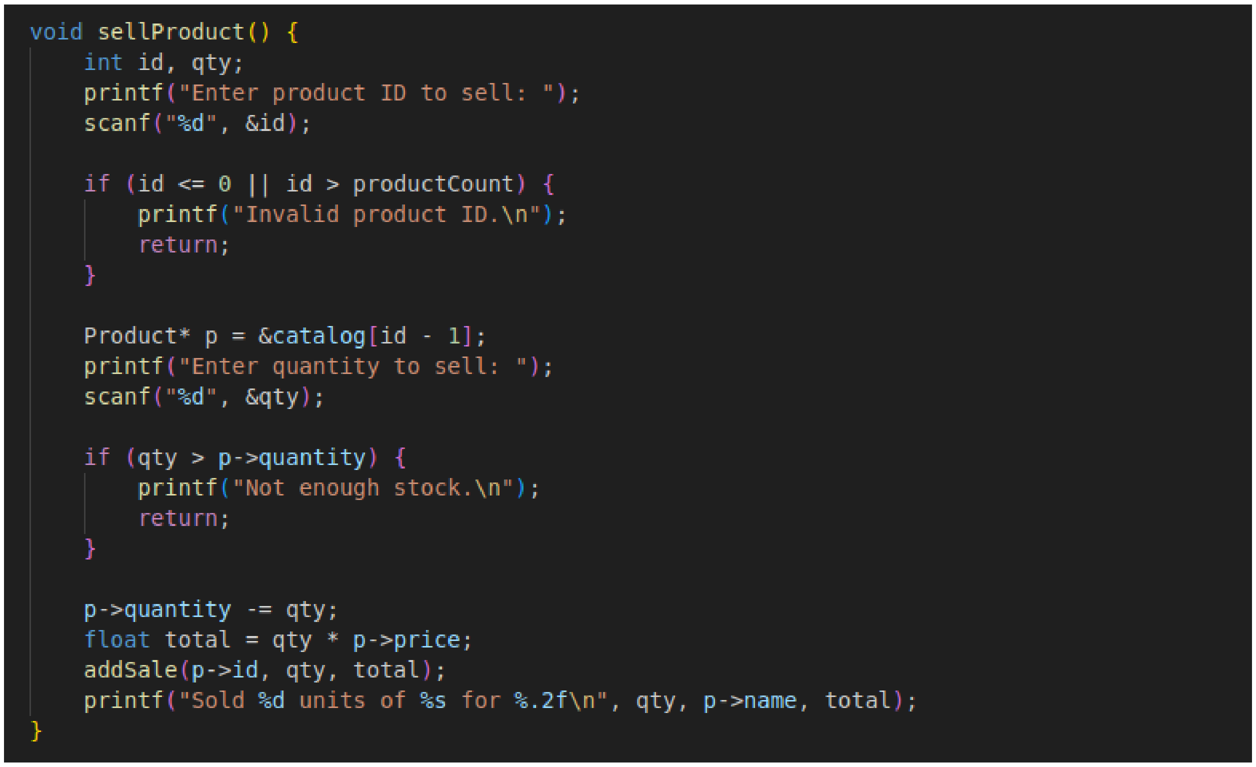
- **Input**: None
- **Output**: Displays all products
- **Purpose**: Iterates over array and prints product details.



```c
void listProducts() {
    printf("\nProduct Catalog:\n");
    for (int i = 0; i < productCount; i++) {
        printf("ID: %d | Name: %s | Quantity: %d | Price: %.2f\n",
               catalog[i].id, catalog[i].name, catalog[i].quantity, catalog[i].price);
    }
}
```

Function: `sellProduct()`

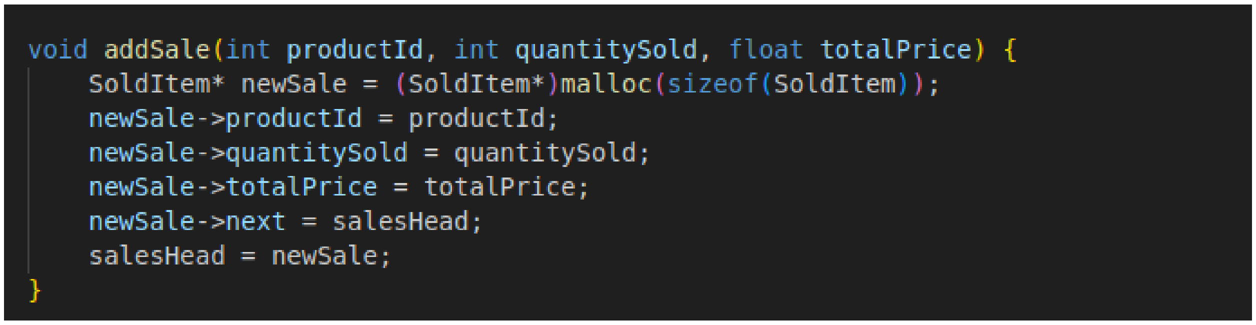
- **Input**: Product ID and quantity to sell
- **Output**: Updates inventory and records sale
- **Purpose**: Handles product selling with validation.



```c
void sellProduct() {
    int id, qty;
    printf("Enter product ID to sell: ");
    scanf("%d", &id);

    if (id <= 0 || id > productCount) {
        printf("Invalid product ID.\n");
        return;
    }

    Product* p = &catalog[id - 1];
    printf("Enter quantity to sell: ");
    scanf("%d", &qty);

    if (qty > p->quantity) {
        printf("Not enough stock.\n");
        return;
    }

    p->quantity -= qty;
    float total = qty * p->price;
    addSale(p->id, qty, total);
    printf("Sold %d units of %s for %.2f\n", qty, p->name, total);
}
```

Function: `addSale()`

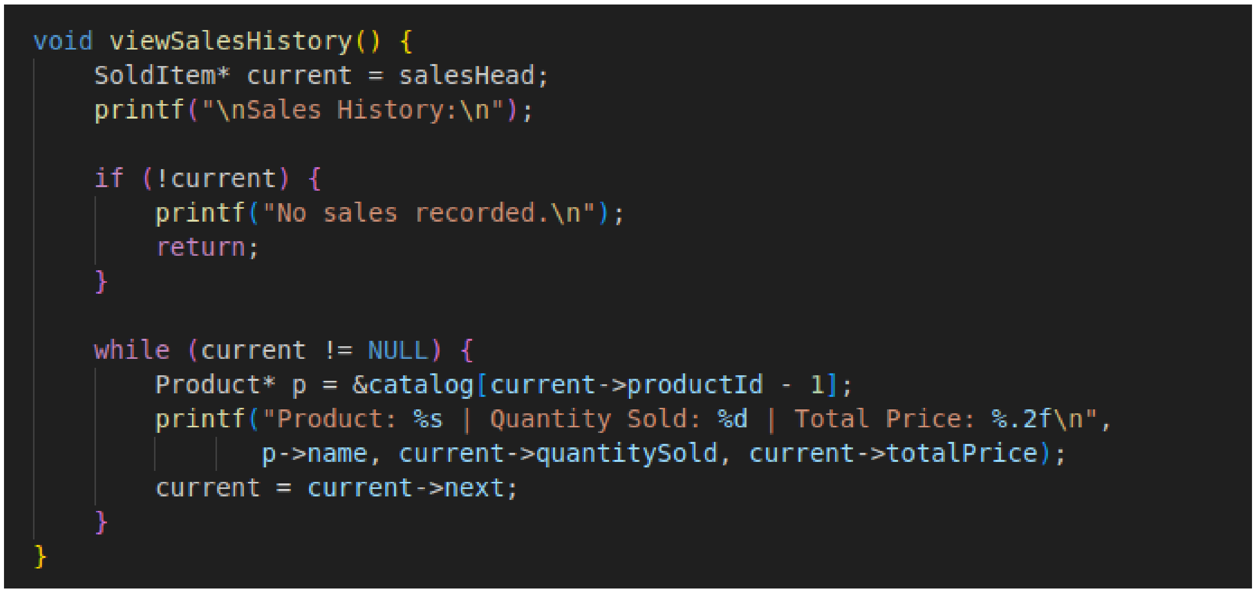
- **Input**: Product ID, quantity sold, total price
- **Output**: Creates a new node in the linked list
- **Purpose**: Dynamically adds a new sales record.



```c
void addSale(int productId, int quantitySold, float totalPrice) {
    SoldItem* newSale = (SoldItem*)malloc(sizeof(SoldItem));
    newSale->productId = productId;
    newSale->quantitySold = quantitySold;
    newSale->totalPrice = totalPrice;
    newSale->next = salesHead;
    salesHead = newSale;
}
```

Function: `viewSalesHistory()`

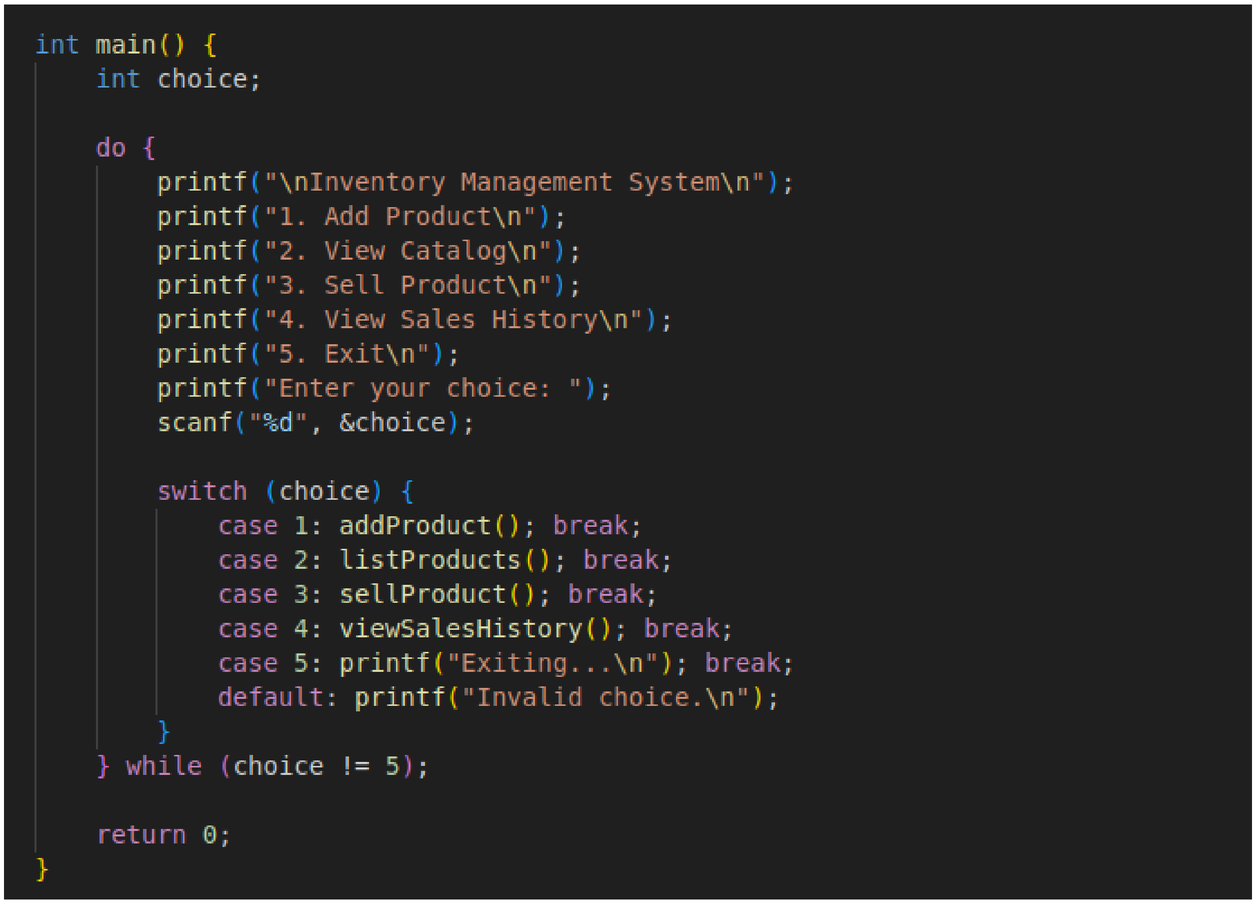
- **Input**: None
- **Output**: Displays all past sales
- **Purpose**: Traverses the linked list and prints each sale.



```c
void viewSalesHistory() {
    SoldItem* current = salesHead;
    printf("\nSales History:\n");

    if (!current) {
        printf("No sales recorded.\n");
        return;
    }

    while (current != NULL) {
        Product* p = &catalog[current->productId - 1];
        printf("Product: %s | Quantity Sold: %d | Total Price: %.2f\n",
               p->name, current->quantitySold, current->totalPrice);
        current = current->next;
    }
}
```

Function: `main()`

- **Input**: User menu choice (via `scanf`)
- **Output**: Calls appropriate function
- **Purpose**: Menu-driven control using `switch-case`.


```c
int main() {
    int choice;

    do {
        printf("\nInventory Management System\n");
        printf("1. Add Product\n");
        printf("2. View Catalog\n");
        printf("3. Sell Product\n");
        printf("4. View Sales History\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addProduct(); break;
            case 2: listProducts(); break;
            case 3: sellProduct(); break;
            case 4: viewSalesHistory(); break;
            case 5: printf("Exiting...\n"); break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 5);

    return 0;
}
```

CHAPTER 7

TESTING

7.1 Testing Approaches

- **Black Box Testing**: Focuses on checking system functionality without knowledge of internal code. It tests input/output behavior.
- **White Box Testing**: Involves testing internal logic, decision branches, and loops of the code. It ensures that all paths are executed as intended.

7.2 Black Box Testing

Test Case 1: Add New Product

Test Case ID	**BB-01**
Input	Product ID: 101, Name: Pen, Quantity: 100, Price: 10.50
Expected Output	Product added successfully to the catalog
Actual Output	Product added successfully
Result	Pass

Test Case 2: Sell Product with Insufficient Stock

Test Case ID	**BB-02**
Input	Product ID: 101, Quantity to sell: 200 (when only 100 are in stock)
Expected Output	Error message: "Insufficient stock"
Actual Output	Error message displayed
Result	Pass

Test Case 3: View Sales History

Test Case ID	**BB-03**
Input	Choose option to view sales history
Expected Output	Display of all past transactions with product IDs and total prices
Actual Output	Correct sales data shown
Result	Pass

7.3 White Box Testing

Test Case 4: Loop Coverage in `listProducts()`

Test Case ID	**WB-01**
Description	Ensure `for` loop prints all existing products
Setup	Add 3 products before calling `listProducts()`
Expected Flow	Loop runs 3 times, printing each product
Result	All 3 products displayed correctly
Status	Pass

Test Case 5: Decision Testing in `sellProduct()`

Test Case ID	**WB-02**
Description	Check decision logic for stock availability
Conditions	Sufficient stock → process sale; else → error
Expected	Correct branch taken based on quantity
Result	Both branches tested successfully
Status	Pass

7.4 Summary

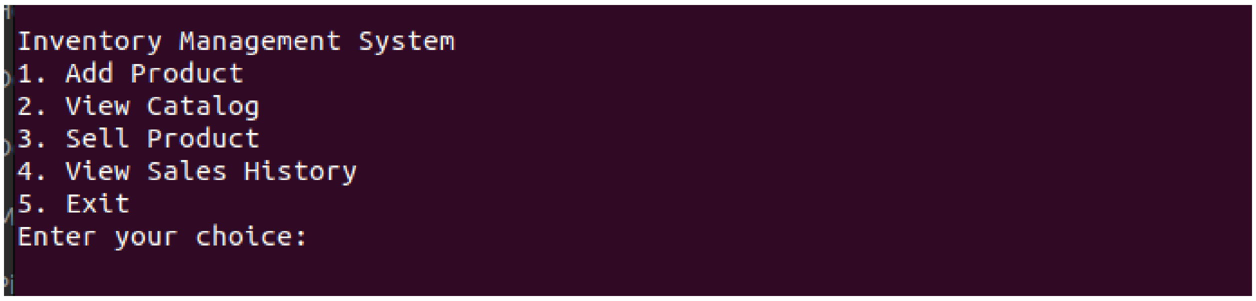
All functional modules of the Inventory Management System passed both black box and white box tests. The code paths were exercised to handle normal operations and boundary conditions (e.g., stock limits, invalid inputs).

CHAPTER 8

OUTPUT/RESULTS

8.1 Main Menu Screen

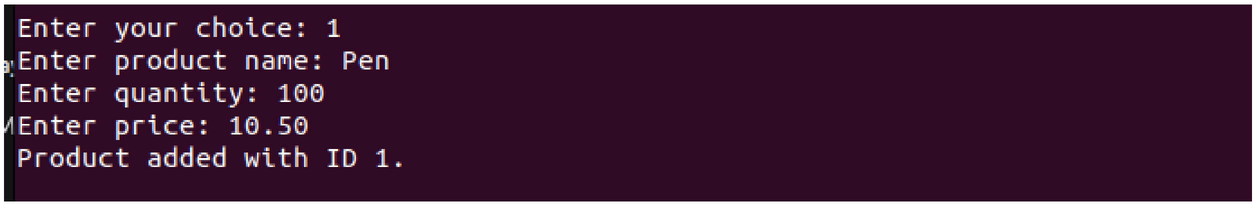
Description: Displayed after starting the program. Allows user to select desired action.



```
Inventory Management System
1. Add Product
2. View Catalog
3. Sell Product
4. View Sales History
5. Exit
Enter your choice:
```

8.2 Add Product Screen

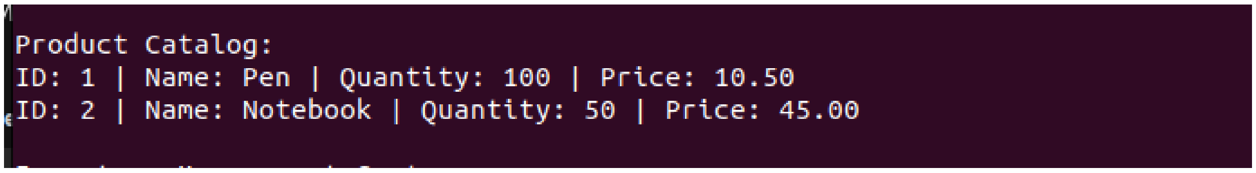
Input Example:



```
Enter your choice: 1
Enter product name: Pen
Enter quantity: 100
Enter price: 10.50
Product added with ID 1.
```

8.3 View Product Catalog Screen

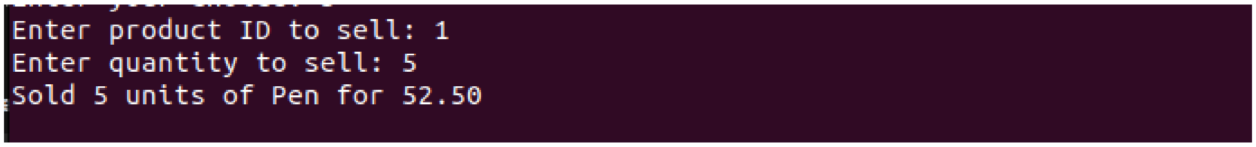
Output:



```
Product Catalog:
ID: 1 | Name: Pen | Quantity: 100 | Price: 10.50
ID: 2 | Name: Notebook | Quantity: 50 | Price: 45.00
```

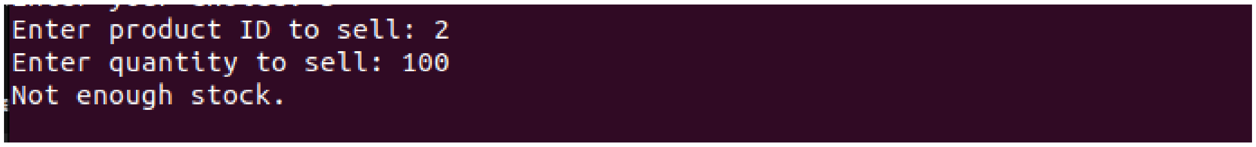
8.4 Sell Product Screen (Successful Case)

Input Example:



```
Enter product ID to sell: 1
Enter quantity to sell: 5
Sold 5 units of Pen for 52.50
```

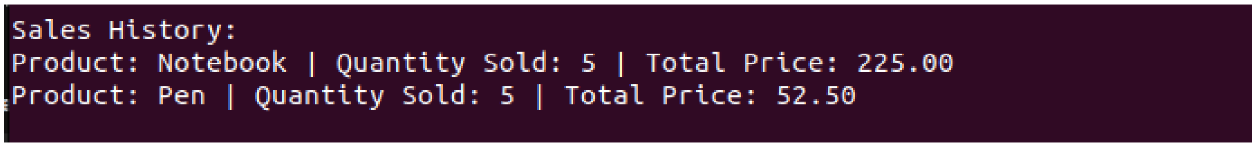
8.5 Sell Product Screen (Insufficient Stock)



```
Enter product ID to sell: 2
Enter quantity to sell: 100
Not enough stock.
```

8.6 View Sales History Screen

Output:



```
Sales History:
Product: Notebook | Quantity Sold: 5 | Total Price: 225.00
Product: Pen | Quantity Sold: 5 | Total Price: 52.50
```

CHAPTER 9

CONCLUSION AND FURTHER WORK

CONCLUSION

The Inventory Management System developed in C effectively fulfills the basic requirements of managing an inventory catalog and recording the sales history of products. The system uses an **array** to manage the product catalog, ensuring fast access and updates to product information. A **linked list** is used to store the history of sold items, enabling dynamic memory allocation and flexible management of sales records.
Key operations such as **adding products, selling items, viewing the catalog, and displaying sales history** are implemented with a user-friendly, menu-driven interface in a command-line environment. The project demonstrates strong fundamentals in C programming, including the use of structures, arrays, dynamic memory allocation, and linked lists.

Throughout testing, the system showed robustness in handling edge cases such as invalid inputs and insufficient stock situations. This ensures that it operates reliably in real-world use cases for small businesses or retail setups.

FURTHER WORK

While the current version of the system provides essential inventory and sales management features, there is significant scope for enhancement:

1. **Persistent Storage**
    - Introduce file handling to save and retrieve data between program runs.
    - Store catalog and sales data in text or binary files.
2. **User Authentication**
    - Add login/logout mechanisms with different roles (e.g., admin, cashier).
3. **Enhanced Interface**
    - Develop a graphical user interface (GUI) using libraries such as GTK or transition to a web-based front end.
4. **Product Search and Sorting**
    - Implement search functionality by ID or name.

- Add options to sort products by price, quantity, or name.

5. **Reports and Analytics**
    - Generate sales reports (daily, weekly, monthly).
    - Track best-selling products and generate summaries.
6. **Error Logging**
    - Implement logging to track errors and invalid operations.
7. **Database Integration**
    - Replace arrays and linked lists with a relational database (e.g., SQLite or MySQL) for scalability and data persistence.

CHAPTER 10

REFERENCES

- Kernighan, Brian W., and Dennis M. Ritchie, *The C Programming Language*, Prentice Hall, 2nd Edition, 1988.
- Malik, D.S., *C Programming: From Problem Analysis to Program Design*, Cengage Learning, 6th Edition, 2014.

- Tanenbaum, Andrew S., *Structured Computer Organization*, Pearson Education, 5th Edition, 2006.

- Booth, Wayne C. "Kenneth Burke's Way of Knowing." *Critical Inquiry*, 1–22, 1974.

- Silberschatz, Abraham, Henry F. Korth, and S. Sudarshan, *Database System Concepts*, McGraw-Hill, 6th Edition, 2011.

- Berry, Jason, Jonathan Foose, and Tad Jones. *Up from the Cradle of Jazz: New Orleans Music Since World War II*, Athens: U of Georgia P, 1986.

- GeeksforGeeks. “Arrays in C”, https://www.geeksforgeeks.org/arrays-in-c-cpp/, Accessed 2024.

- TutorialsPoint.“C-LinkedLists”,https:// www.tutorialspoint.comdata_structures_algorithms/linked_list_program_in_c.htm, Accessed 2024.

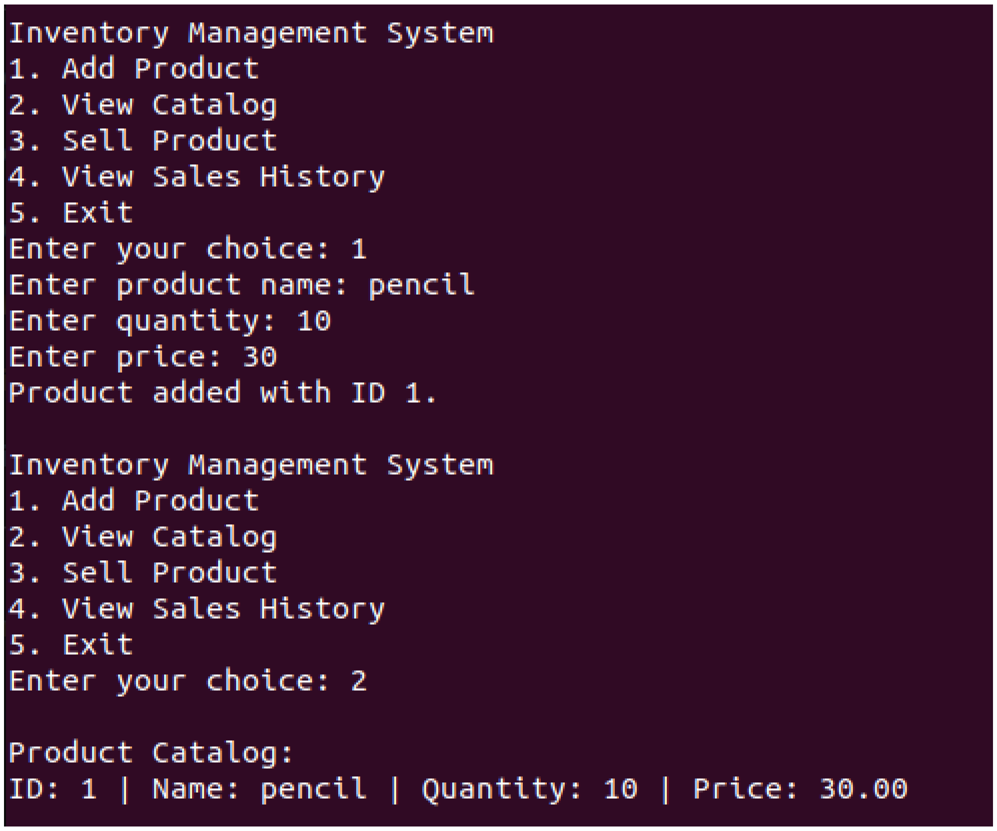
CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_PRODUCTS 100

// Product structure

typedef struct {

    int id;

    char name[50];

    int quantity;

    float price;

} Product;

// Sold item node structure

typedef struct SoldItem {

    int productId;

    int quantitySold;

    float totalPrice;

    struct SoldItem* next;

} SoldItem;

Product catalog[MAX_PRODUCTS];

int productCount = 0;

SoldItem* salesHead = NULL;

// Function declarations

void addProduct();

void listProducts();
void sellProduct();
void addSale(int productId, int quantitySold, float totalPrice);
void viewSalesHistory();
int main() {
    int choice;
    do {
        printf("\nInventory Management System\n");
        printf("1. Add Product\n");
        printf("2. View Catalog\n");
        printf("3. Sell Product\n");
        printf("4. View Sales History\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: addProduct(); break;
            case 2: listProducts(); break;
            case 3: sellProduct(); break;
            case 4: viewSalesHistory(); break;
            case 5: printf("Exiting...\n"); break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 5);
    return 0;
}
```

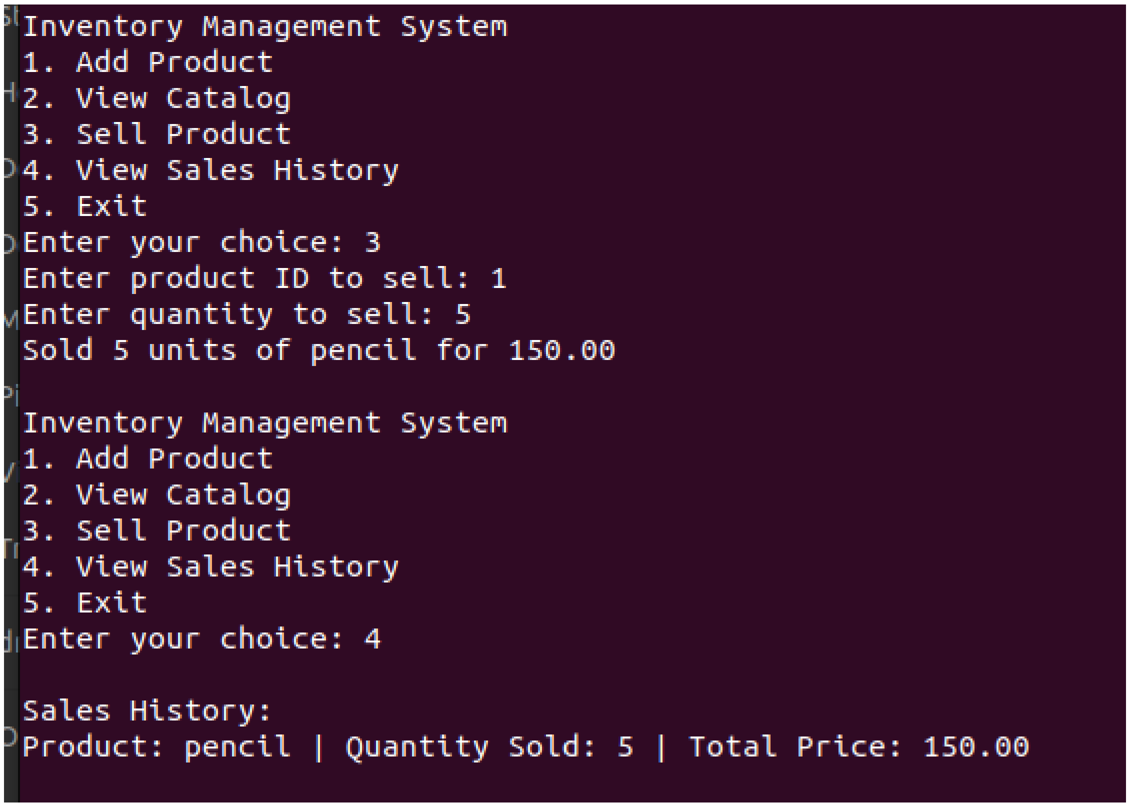
```
void addProduct() {
    if (productCount >= MAX_PRODUCTS) {
        printf("Catalog is full.\n");
        return;
    }
    Product p;
    p.id = productCount + 1;
    printf("Enter product name: ");
    scanf("%s", p.name);
    printf("Enter quantity: ");
    scanf("%d", &p.quantity);
    printf("Enter price: ");
    scanf("%f", &p.price);
    catalog[productCount++] = p;
    printf("Product added with ID %d.\n", p.id);
}
void listProducts() {
    printf("\nProduct Catalog:\n");
    for (int i = 0; i < productCount; i++) {
        printf("ID: %d | Name: %s | Quantity: %d | Price: %.2f\n",
               catalog[i].id, catalog[i].name, catalog[i].quantity, catalog[i].price);
    }
}
void sellProduct() {
    int id, qty;
    printf("Enter product ID to sell: ");
    scanf("%d", &id);
    if (id <= 0 || id > productCount) {
        printf("Invalid product ID.\n");
        return;
    }
    Product* p = &catalog[id - 1];
    printf("Enter quantity to sell: ");
    scanf("%d", &qty);
    if (qty > p->quantity) {
        printf("Not enough stock.\n");
        return;
    }
    p->quantity -= qty;
    float total = qty * p->price;
    addSale(p->id, qty, total);
    printf("Sold %d units of %s for %.2f\n", qty, p->name, total);
}
void addSale(int productId, int quantitySold, float totalPrice) {
    SoldItem* newSale = (SoldItem*)malloc(sizeof(SoldItem));
    newSale->productId = productId;
    newSale->quantitySold = quantitySold;
    newSale->totalPrice = totalPrice;
    newSale->next = salesHead;
    salesHead = newSale;
```

```
}
void viewSalesHistory() {
    SoldItem* current = salesHead;
    printf("\nSales History:\n");
    if (!current) {
        printf("No sales recorded.\n");
        return;
    }
    while (current != NULL) {
        Product* p = &catalog[current->productId - 1];
        printf("Product: %s | Quantity Sold: %d | Total Price: %.2f\n",
               p->name, current->quantitySold, current->totalPrice);
        current = current->next;
    }
}
```

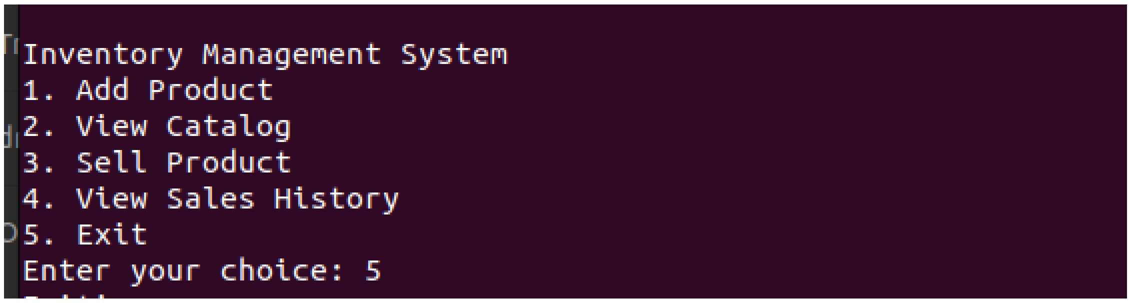
OUTPUT



```
Inventory Management System
1. Add Product
2. View Catalog
3. Sell Product
4. View Sales History
5. Exit
Enter your choice: 1
Enter product name: pencil
Enter quantity: 10
Enter price: 30
Product added with ID 1.

Inventory Management System
1. Add Product
2. View Catalog
3. Sell Product
4. View Sales History
5. Exit
Enter your choice: 2

Product Catalog:
ID: 1 | Name: pencil | Quantity: 10 | Price: 30.00
```



```
Inventory Management System
1. Add Product
2. View Catalog
3. Sell Product
4. View Sales History
5. Exit
Enter your choice: 3
Enter product ID to sell: 1
Enter quantity to sell: 5
Sold 5 units of pencil for 150.00

Inventory Management System
1. Add Product
2. View Catalog
3. Sell Product
4. View Sales History
5. Exit
Enter your choice: 4

Sales History:
Product: pencil | Quantity Sold: 5 | Total Price: 150.00
```



```
Inventory Management System
1. Add Product
2. View Catalog
3. Sell Product
4. View Sales History
5. Exit
Enter your choice: 5
```