

INVENTORY MANAGEMENT SYSTEM ARRAYS FOR PRODUCT CATALOG, LINKED LIST FOR SOLD ITEMS HISTORY

PRESENTED BY

24KB1A05KA

24KB1A05DP

24KB1A05CU

24KB1A05GA

OBJECTIVE

The primary objective of an Inventory Management System is to efficiently track, control, and manage inventory levels, ensuring that the right products are available at the right time, while minimizing costs and maximizing profitability.

INTRODUCTION

- ▶ An Inventory Management System is a software application designed to help businesses and organizations efficiently track, manage, and control their stock of products or materials. It plays a crucial role in ensuring that the right amount of inventory is available at the right time, reducing both shortages and excess stock.
- ▶ This system allows users to add new products, update stock quantities, process sales, and monitor inventory levels in real time. By automating these tasks, it minimizes manual errors, saves time, and provides accurate records that can be used for analysis and decision-making.
- ▶ Whether in retail, manufacturing, or services, an inventory management system is essential for improving operational efficiency, maintaining customer satisfaction, and supporting business growth.

Data Structures Overview

Array

Arrays are used to store products in the format **Product products[MAX_PRODUCTS]**. This provides indexed storage with a fixed size, enabling efficient access to product information.

- Indexed storage
- Fixed size allocation

Structure

Structures are defined for both Product and Sale entities. The **Product** structure includes fields for **id**, **name**, **quantity**, and **price**, while the **Sale** structure includes **product_id**, **quantity sold**, and **total price**.

- Product: id, name, quantity, price
- Sale: product_id, quantity sold, total price

Simulated Linked List

Sales data is stored sequentially in the **sales[]** array. Index-based traversal simulates linked list-style behavior, allowing for efficient management of sales records.

- Sequential storage in array
- Index-based traversal

Why use an array for the product catalog?

1. Fixed and predictable size:

The number of products is usually known to be limited (e.g., under 100), so a **static** array works well.

2. Fast access by index:

Arrays allow $O(1)$ time access, making it easy to search or update a product by index.

3. No frequent insertions/deletions:

The product list doesn't change often. You typically just add products occasionally. Arrays are efficient when you don't need to insert/delete in the middle.

4. Simplicity:

Arrays are easier to implement and debug, especially in C.

Why prefer a linked list for sales history?

1. Dynamic size: Sales are made continuously, so a linked list avoids the need to **predefine MAX_SALES**.
2. Frequent additions: Linked lists are efficient for appending new records at runtime.
3. Unbounded growth: You may want to keep sales history indefinitely (or save to a file), so dynamic memory is more appropriate.
4. Rare lookups: Unlike products, you usually just display recent sales — so sequential access is fine.

Feature	Array (Products)	Linked List (Sales)
Access Speed	Fast ($O(1)$)	Slower ($O(n)$)
Insertion Speed	Moderate (with shifting)	Fast ($O(1)$ at end)
Deletion	Complex (shifting needed)	Simple (pointer change)
Memory Usage	Fixed at compile time	Dynamic at runtime
	Relatively static datasets	

System Functionalities

1 Add Product

The system collects product details, such as name, quantity, and price, and stores this information in the product array. This ensures all products are tracked for inventory purposes.

2 List Products

The system displays all available products, showing key details like product ID, name, quantity, and price, providing a clear overview of the current inventory.

3 Sell Product

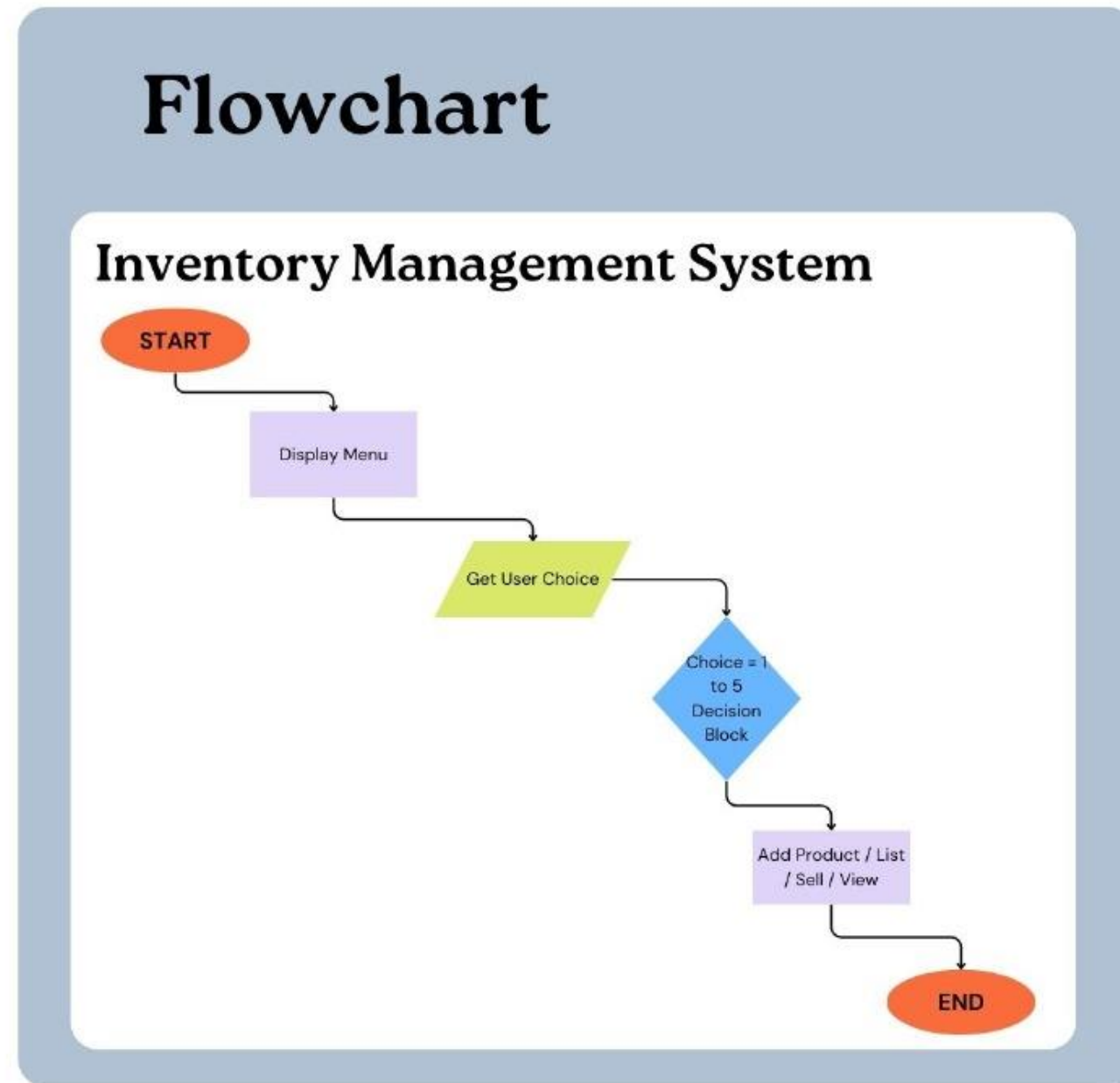
Upon a sale, the system updates the quantity of the product and records the sale in the sales array, tracking product movement and sales history.

4 View Sales

The system displays a list of sold items with the total price, allowing users to review sales records and analyze sales performance.



System Flowchart



Code Snapshot: Product Handling

Product Structure

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

addProduct()

```
void addProduct() {  
    // Code to collect product  
    details  
    // and store in product array  
}
```

listProducts()

```
void listProducts() {  
    // Code to display all available  
    // products in the system  
}
```

The code includes the structure of the Product and sample functions for adding and listing products. The highlights include ID-based product search and input validation for product count, ensuring data integrity.

```
1  // Function to add a new product  
2  {  
3      // Prompt user for product details  
4      printf("Enter product name: ");  
5      char name[50];  
6      fgets(name, sizeof name, stdin);  
7      // Prompt user for product price  
8      printf("Enter product price: ");  
9      float price;  
10     scanf("%f", &price);  
11     // Prompt user for product quantity  
12     printf("Enter product quantity: ");  
13     int quantity;  
14     scanf("%d", &quantity);  
15     // Validate quantity  
16     while (quantity < 0)  
17     {  
18         printf("Quantity must be non-negative. Enter again: ");  
19         scanf("%d", &quantity);  
20     }  
21     // Add product to the array  
22     products[productCount] = {id, name, quantity, price};  
23     productCount++;  
24     // Save the data to a file  
25     saveProductsToFile();  
26 }  
27  
28 // Function to list all products  
29 {  
30     printf("List of products in the system:\n");  
31     for (int i = 0; i < productCount; i++)  
32     {  
33         printf("%d\t%s\t%d\t%.2f\n", products[i].id, products[i].name, products[i].quantity, products[i].price);  
34     }  
35 }
```

Code Snapshot: Sales Handling

Sale Structure

```
struct Sale {  
    int product_id;  
    int quantity_sold;  
    float total_price;  
};
```

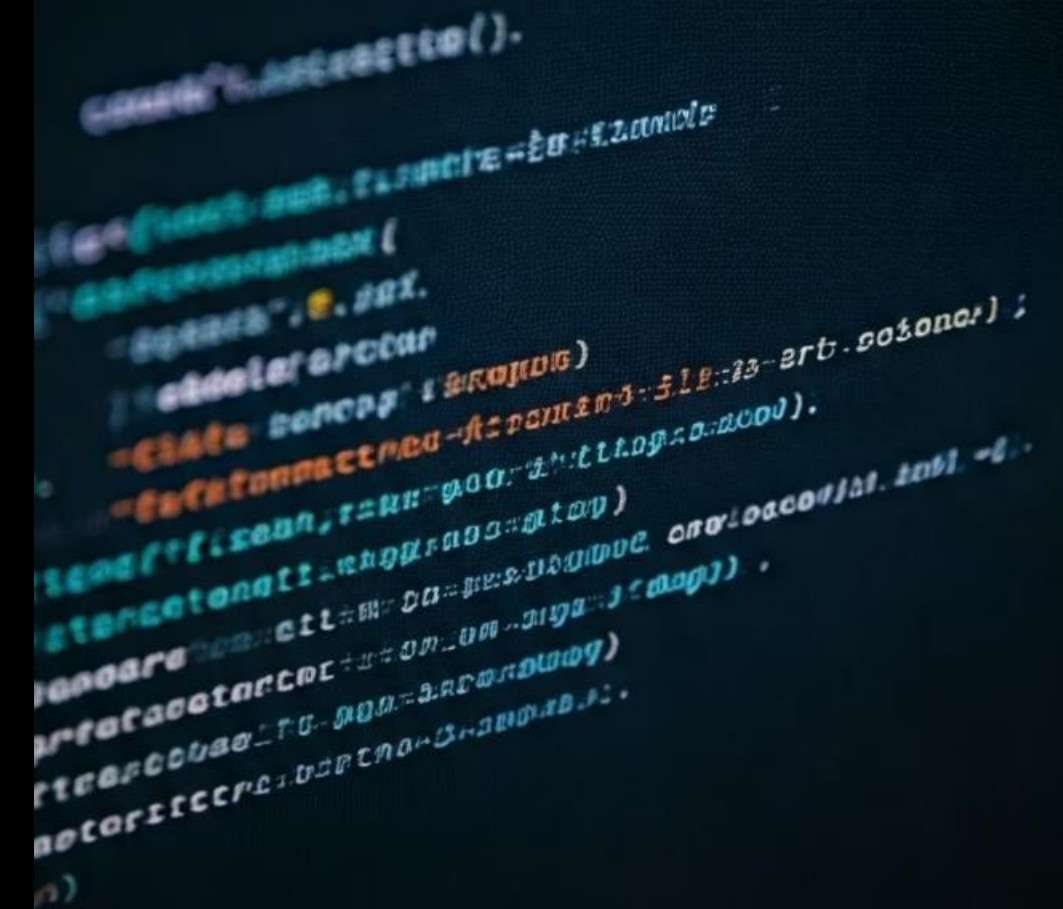
sellProduct()

```
void sellProduct() {  
    // Code to update quantity  
    after sale  
    // and record sale in sales  
    array  
}
```

listSales()

```
void listSales() {  
    // Code to display list of sold  
    // items with total price  
}
```

The sales handling code includes the structure of the Sale and sample functions for selling products and listing sales. Key features include quantity check before sale, sale record creation, and sales listed with total value, ensuring accurate sales tracking.



```
...sellProduct();  
...updateQuantity(...)  
...recordSale(...)  
...listSales();  
...displaySales();  
...calculateTotalPrice(...)  
...checkQuantity(...)  
...createSaleRecord(...)
```

Sample Test Case & Output

[illegible]

Adding Products:

Apple (ID: 101, Qty: 50, Price: 2.5)

Banana (ID: 102, Qty: 100, Price: 1.0)

Selling:

Apple $\times 10 \rightarrow ₹25.00$

Output:

ID	Name	Qty	Price
----	------	-----	-------

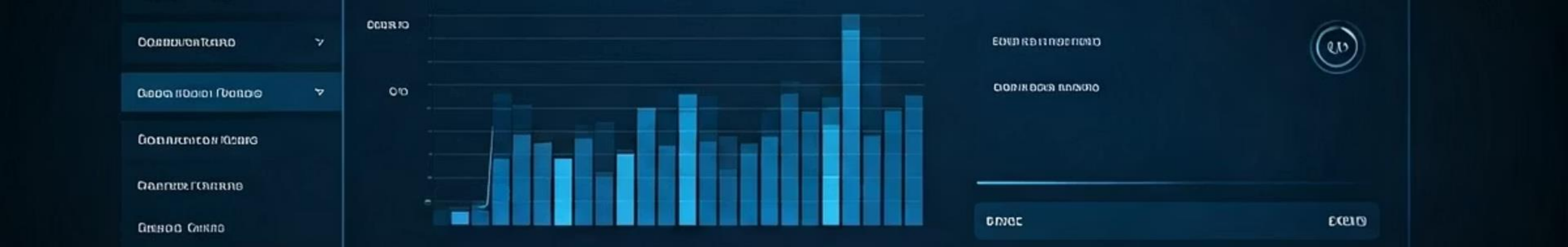
101 Apple 40 2.50

102 Banana 100 1.00

Sales: Product ID Qty Total Price

101 10 25.00

The test case demonstrates adding products such as Apple and Banana, selling a quantity of Apple, and displaying the output showing updated inventory and sales records. This showcases the system's functionality and data accuracy.



Conclusion & Future Scope

Achievements

Basic inventory system built with arrays and structures provides a functional command-line interface for managing products and sales.

The inventory system has achieved its goal of providing a basic, functional command-line interface. Future developments could focus on implementing file handling for persistent storage, dynamic memory with linked lists, a GUI for better usability, and additional features such as search, update, and delete.

Future Scope

Enhancements include using file handling for persistent storage, dynamic memory with linked lists, and a GUI for better usability.

Additional Features

Implement search, update, and delete features to improve data management and system functionality.

The background features abstract, overlapping geometric shapes in various shades of green, ranging from light lime to dark forest green. These shapes are primarily located on the right side of the image, creating a modern, layered effect. The rest of the background is a solid, very light green.

THANK YOU