Shop Management System

1. 🎯 Objective

- The objective of this project is to design and implement a Shop Management System using object-oriented programming (OOP) principles. The system helps manage product inventory, including adding, updating, deleting, and displaying product information. It ensures modularity, scalability, and secure data handling through encapsulation, inheritance, and polymorphism.

2. 📚 Introduction

- Managing inventory efficiently is essential for any retail business. This Shop Management System automates core operations such as product entry, stock updates, and deadline tracking. It is developed using C++ and follows OOP principles to ensure clean architecture and maintainable code.
- The system supports the following operations:
- - Add new products
- - Update product details
- - Delete products
- - Display product list
- - Track stock and deadlines

3. 🛠️ Methodology

- 🔍 Requirement Analysis
- - Identify core operations: add, update, delete, display.
- - Define product attributes: ID, name, price, quantity, deadline, category.

🧩 System Design

- The system is built using three main classes:

- 🧱 Item Class
- - Base class with attributes like id, name, and price.
- - Provides virtual methods input() and display() for polymorphic behavior.

- 📦 Product Class (Inheritance)
- - Inherits from Item.
- - Adds attributes: quantity, deadline, and category.
- - Overrides input() and display() methods to extend functionality.

- 👨‍💼 Admin Class
- - Manages product operations: add, delete, update, search, and list.
- - Uses encapsulated access to product data.

🔐 Encapsulation

- - All data members are private or protected.
- - Access is controlled through public setters and getters.
- - Example: setPrice(), getId().

🧬 Inheritance

- - Product inherits from Item, reusing and extending its functionality.
- - Promotes code reuse and logical hierarchy.

Polymorphism

- - input() and display() are declared virtual in Item and overridden in Product.
- - Enables dynamic behavior when using base class pointers.

```cpp
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;


class Item {
protected:
    int id;
    string name;
    double price;

public:
    virtual void input() {
        cout << "Enter iD: ";
        cin >> id;
        cin.ignore();
        cout << "Enter name: ";
        getline(cin, name);
        cout << "Enter price: ";
        cin >> price;
    }

    virtual void display() const {
        cout << "ID: " << id << ", Name: " << name << ", Price: " << price;
    }

    int getId() const { return id; }
    string getName() const { return name; }
    double getPrice() const { return price; }

    void setId(int i) { id = i; }
    void setName(const string& n) { name = n; }
    void setPrice(double p) { price = p; }
};

```

```
38  class Product : public Item {
39  private:
40      int quantity;
41      string deadline;
42      string category;
43
44  public:
45      void input() override
46      {
47          Item::input();
48          cout << "Enter Quantity: ";
49          cin >> quantity;
50          cin.ignore();
51          cout << "Enter Deadline (e.g. 2025-12-31): ";
52          getline(cin, deadline);
53          cout << "Enter Category: ";
54          getline(cin, category);
55      }
56
57      void display() const override
58      {
59          Item::display();
60          cout << " Quantity : " << quantity<<endl;
61              cout<< " Deadline : " << deadline<<endl;
62              cout<< " Category : " << category << endl;
63      }
64
65      void saveToFile(ofstream& file) const
66       {
67          file << getId() << endl;
68          file << getName() << endl;
69          file << getPrice() << endl;
70          file << quantity << endl;
71          file << deadline << endl;
72          file << category << endl;
73      }
74
75      void loadFromFile(ifstream& file) {
76          string line;
77          getline(file, line); setId(stoi(line));
78          getline(file, line); setName(line);
79          getline(file, line); setPrice(stod(line));
80          getline(file, line); quantity = stoi(line);
81          getline(file, deadline);
82          getline(file, category);
```

```
        getline(title, category);
    }

    int getQuantity() const { return quantity; }
    string getCategory() const { return category; }
    string getDeadline() const { return deadline; }

    void setQuantity(int q)
    {
        quantity = q;

    }

    void update() {
        cout << "Current Quantity: " << quantity << " Price: " << getPrice() << endl;
        cout << "Enter new Quantity: ";
        cin >> quantity;
        cout << "Enter new Price: ";
        double newPrice;
        cin >> newPrice;
        setPrice(newPrice);
        cin.ignore();
        cout << "Enter new Deadline: ";
        getline(cin, deadline);
        cout << "Enter new Category: ";
        getline(cin, category);
    }
};
```

```
class Shop {
private:
    vector<Product> products;

public:
    void addProduct() {
        Product p;
        p.input();
        products.push_back(p);
        cout << " Product added "<<endl;
    }

    void viewProducts() const {
        cout << " Product List :"<<endl;
        for (const auto& p : products) {
            p.display();
        }
    }

    void viewByCategory() const {
        string cat;
        cout << "Enter category to view: ";
        cin.ignore();
        getline(cin, cat);
        bool found = false;
        for (const auto& p : products) {
            if (p.getCategory() == cat) {
                p.display();
                found = true;
            }
        }
        if (!found) cout << " No products found in the categor"<<endl;
    }
```

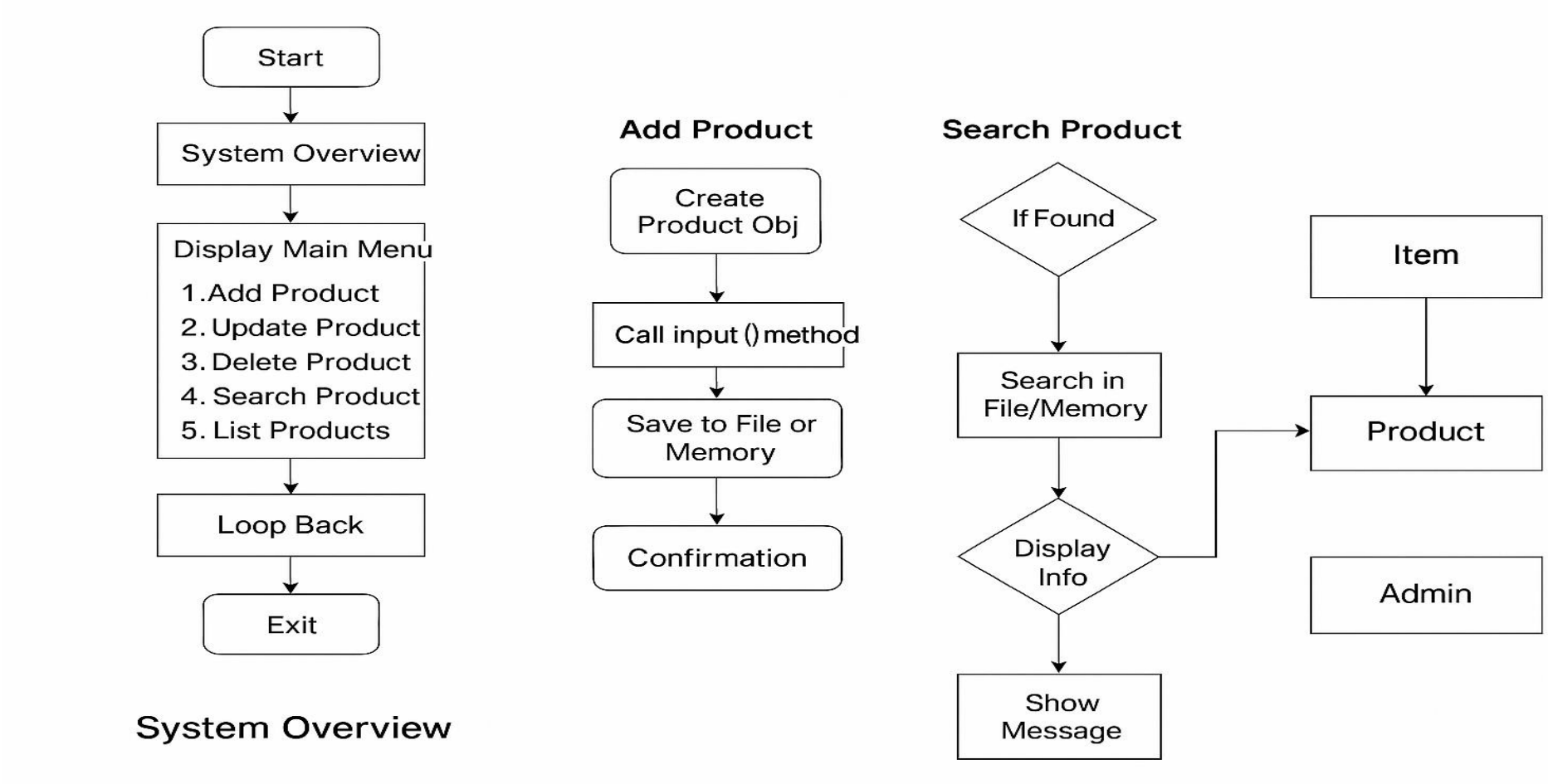
```
146     void searchProduct() const {
147         string name;
148         cout << "Enter product name to search: ";
149         cin.ignore();
150         getline(cin, name);
151         for (const auto& p : products) {
152             if (p.getName() == name) {
153                 cout << " Found: "<<endl;
154                 p.display();
155                 return;
156             }
157         }
158         cout << " Product not found"<<endl;
159     }
160
161     void deleteProduct() {
162         int id;
163         cout << "Enter Product ID to delete: ";
164         cin >> id;
165         for (auto it = products.begin(); it != products.end(); ++it)
166             if (it->getId() == id) {
167                 products.erase(it);
168                 cout << " Product deleted "<<endl;
169                 return;
170             }
171         }
172         cout << " Product not found "<<endl;
173     }
174
175     void updateProduct() {
176         int id;
177         cout << "Enter Product ID : ";
178         cin >> id;
179         for (auto& p : products) {
180             if (p.getId() == id) {
181                 p.update();
182                 cout << " Product updated "<<endl;
183                 return;
184             }
185         }
186         cout << " Product not found "<<endl;
187     }
```

```
void makeSale() {
    int id, qty;
    cout << "Enter Product ID to sell: ";
    cin >> id;
    for (auto& p : products) {
        if (p.getId() == id) {
            cout << "Available Quantity: " << p.getQuantity() << endl;
            cout << "Enter Quantity to sell: ";
            cin >> qty;
            if (qty > p.getQuantity()) {
                cout << "Not enough stock available" << endl;
                return;
            }
            double total = qty * p.getPrice();
            cout << "Total Bill: " << total << " Taka" << endl;


            int updatedQty = p.getQuantity() - qty;
            p.setQuantity(updatedQty);
            cout << "Remaining Quantity: " << updatedQty << endl;
            return;
        }
    }
    cout << "Product not found" << endl;
}
```

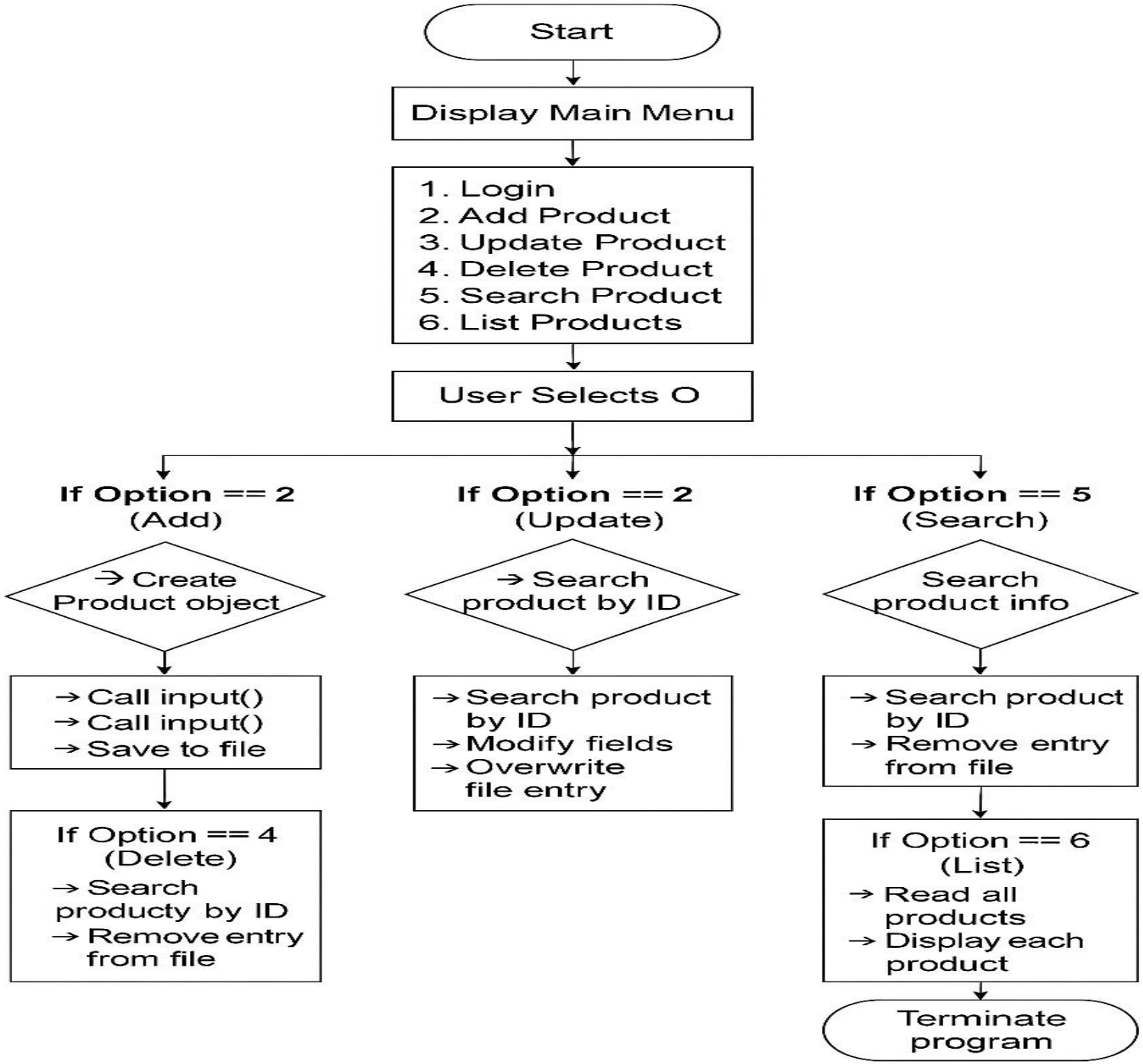
```
    void saveToFile() const {
        ofstream file("project.txt");
        file << products.size() << endl;
        for (const auto& p : products) {
            p.saveToFile(file);
        }
        file.close();
        cout << " Products save to file "<<endl;
    }

    void loadFromFile() {
        ifstream file("project.txt");
        if (!file) {
            cout << "  not found "<<endl;
            return;
        }
        int count;
        file >> count;
        file.ignore();
        products.clear();
        for (int i = 0; i < count; ++i) {
            Product p;
            p.loadFromFile(file);
            products.push_back(p);
        }
        file.close();
        cout << " loaded from file "<<endl;
    }
};
```

```
251 int main() {
252     Shop shop;
253     int choice;
254
255     do {
256         cout << "\n**  Shop Management Menu  **"<<endl;
257         cout << "1. Add Product"<<endl;
258         cout << "2. View All Product"<<endl;
259         cout << "3. View Products by Category"<<endl;
260         cout << "4. Search Product"<<endl;
261         cout << "5. Delete Product  "<<endl;
262         cout << "6. Update Product "<<endl;
263         cout << "7. Save to File"<<endl;
264         cout << "8. Load from File"<<endl;
265         cout << "9. Sale product " << endl;
266         cout << "0. ..Exit.."<<endl;
267         cout << " Enter choice: ";
268         cin >> choice;
269
270         switch (choice) {
271             case 1: shop.addProduct();
272                 break;
273             case 2: shop.viewProducts();
274               break;
275             case 3: shop.viewByCategory();
276                 break;
277             case 4: shop.searchProduct();
278                break;
279             case 5: shop.deleteProduct();
280                break;
281             case 6: shop.updateProduct();
282                break;
283             case 7: shop.saveToFile();
284                 break;
285             case 8: shop.loadFromFile();
286                break;
287                case 9: shop.makeSale();
288                break;
289              case 0: cout << " Exit "<<endl;
290                break;
291             default: cout << "  choice not found "<<endl;
292         }
293     } while (choice != 0);
294
295     return 0;
296 }
297
```

Flow-chart:


Start
System Overview
Display Main Menu
1.Add Product
2. Update Product
3. Delete Product
4. Search Product
5. List Products
Loop Back
Exit
System Overview
Add Product
Create Product Obj
Call input () method
Save to File or Memory
Confirmation
Search Product
If Found
Search in File/Memory
Display Info
Show Message
Item
Product
Admin

Flow-chart:


Start
Display Main Menu
1. Login
2. Add Product
3. Update Product
4. Delete Product
5. Search Product
6. List Products
User Selects O
If Option == 2
(Add)
→ Create Product object
→ Call input()
→ Call input()
→ Save to file
If Option == 4
(Delete)
→ Search producty by ID
→ Remove entry from file
If Option == 2
(Update)
→ Search product by ID
→ Search product by ID
→ Modify fields
→ Overwrite file entry
If Option == 5
(Search)
Search product info
→ Search product by ID
→ Remove entry from file
If Option == 6
(List)
→ Read all products
→ Display each product
Terminate program

Input:

```
D:\p,,,,.exe

=== Shop Management Menu ===
1. Add Product
2. View All Products
3. View Products by Category
4. Search Product
5. Delete Product
6. Update Stock
7. Save List
8. Load List
0. Exit
Enter choice:
```

Input & output:

```
9. ..Exit..
 Enter choice: 1
Enter iD: 14
Enter name: honey
Enter price: 200
Enter Quantity: 2
Enter Deadline (e.g. 2025-12-31): 2025-12-5
Enter Category: honeys
 Product added

**  Shop Management Menu  **
1. Add Product
2. View All Product
3. View Products by Category
4. Search Product
5. Delete Product
6. Update Product
7. Save to File
8. Load from File
9. Sale product
0. ..Exit..
 Enter choice: 2
 Product List :
ID: 14, Name: honey, Price: 200 Quantity : 2
 Deadline : 2025-12-5
 Category : honeys

**  Shop Management Menu  **
1. Add Product
```

4. ⚠ Limitations

- - No GUI; interaction is console-based.
- - No database integration; relies on file handling.
- - Input validation is basic; does not handle special characters or spaces robustly.

5. ✅ Conclusion

The Shop Management System successfully demonstrates the use of object-oriented principles to manage inventory operations. It is modular, maintainable, and scalable. Future improvements could include:

- - GUI integration for better usability
- - Database support for robust data management
- - Advanced search and reporting features
- - User authentication for secure access