```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

// Product structure
struct Product {
    int id;
    string name;
    string category;
    double price;
    int quantity;
    string expirationDate; // Format: YYYY-MM-DD
    int salesCount;        // Tracks popularity
    Product* next;
};

// Supermarket Management System Class
class SupermarketSystem {
private:
    Product* start;  // Head of the linked list
    int productCount;

    // Temporary node for sorting
    struct TempNode {
        Product* product;
        TempNode* next;
    };

    // Helper method to clear the list
    void clearList() {
        while (start != nullptr) {
```

```
        Product* temp = start;
        start = start->next;
        delete temp;
    }
    productCount = 0;
}

// Validate date format (basic check: YYYY-MM-DD)
bool isValidDate(const string& date) {
    if (date.length() != 10 || date[4] != '-' || date[7] != '-') return false;
    for (int i = 0; i < 10; i++) {
        if (i == 4 || i == 7) continue;
        if (!isdigit(date[i])) return false;
    }
    return true;
}

// Get next field from CSV line
string getNextField(const string& line, size_t& pos) {
    string field;
    if (pos < line.size() && line[pos] == '"') {
        pos++; // Skip opening quote
        while (pos < line.size() && line[pos] != '"') {
            field += line[pos];
            pos++;
        }
        if (pos < line.size() && line[pos] == '"') {
            pos++; // Skip closing quote
        }
    } else {
        while (pos < line.size() && line[pos] != ',') {
            field += line[pos];
```

```
                pos++;
            }
        }
        // Skip the comma if present
        if (pos < line.size() && line[pos] == ',') {
            pos++;
        }
        return field;
    }

    // Insert into temporary sorted list
    TempNode* sortedInsert(TempNode* head, Product* newProduct, bool (*compare)(Product*,
Product*)) {
        TempNode* newNode = new TempNode{newProduct, nullptr};
        if (head == nullptr || compare(newProduct, head->product)) {
            newNode->next = head;
            return newNode;
        }
        TempNode* current = head;
        while (current->next != nullptr && !compare(newProduct, current->next->product)) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
        return head;
    }

public:
    SupermarketSystem() : start(nullptr), productCount(0) {}

    ~SupermarketSystem() {
        clearList();
    }
    // Add a new product (sorted by ID, validates inputs)

    void addProduct(int id, string name, string category, double price, int quantity, string
expirationDate) {

        if (searchById(id) != nullptr) {

            cout << "Product with ID " << id << " already exists.\n";

            return;

        }

        if (price < 0 || quantity < 0) {

            cout << "Price and quantity must be non-negative.\n";

            return;

        }

        if (!isValidDate(expirationDate)) {

            cout << "Invalid expiration date format (use YYYY-MM-DD).\n";

            return;

        }

        Product* newProduct = new Product{id, name, category, price, quantity, expirationDate, 0,
nullptr};

        if (start == nullptr || start->id > id) {

            newProduct->next = start;

            start = newProduct;

        } else {

            Product* temp = start;

            while (temp->next != nullptr && temp->next->id < id) {

                temp = temp->next;

            }

            newProduct->next = temp->next;

            temp->next = newProduct;

        }

        productCount++;

        cout << "Product added successfully!\n";

    }
```

```
// Display all products (sorted by ID)
void displayProducts() {
    if (start == nullptr) {
        cout << "No products in the system.\n";
        return;
    }
    cout << "\nSupermarket Inventory (Sorted by ID)\n";
    cout << "ID\tName\tCategory\tPrice\tQty\tExp Date\tSales\n";
    cout << "------------------------------------------------------------\n";
    Product* temp = start;
    while (temp != nullptr) {
        displayProduct(temp);
        temp = temp->next;
    }
    cout << "------------------------------------------------------------\n";
}

// Display a single product
void displayProduct(Product* product) {
    cout << product->id << "\t"
         << product->name << "\t"
         << product->category << "\t"
         << product->price << "\t"
         << product->quantity << "\t"
         << product->expirationDate << "\t"
         << product->salesCount << "\n";
}

// Search by ID (optimized for sorted list)
Product* searchById(int id) {
    Product* temp = start;
    while (temp != nullptr) {
        if (temp->id == id) return temp;
        if (temp->id > id) break; // Early exit
        temp = temp->next;
    }
    return nullptr;
}

// Remove a product by ID
void removeProduct(int id) {
    if (start == nullptr) {
        cout << "Product not found.\n";
        return;
    }
    if (start->id == id) {
        Product* temp = start;
        start = start->next;
        delete temp;
        productCount--;
        cout << "Product removed successfully.\n";
        return;
    }
    Product* prev = start;
    Product* current = start->next;
    while (current != nullptr) {
        if (current->id == id) {
            prev->next = current->next;
            delete current;
            productCount--;
            cout << "Product removed successfully.\n";
            return;
        }
        if (current->id > id) break;
        prev = current;
        current = current->next;
    }
    cout << "Product not found.\n";
}

// Update a product by ID (multiple fields)
void updateProduct(int id) {
    Product* product = searchById(id);
    if (product == nullptr) {
        cout << "Product not found.\n";
        return;
    }
    cout << "Product found: " << product->name << "\n";
    cout << "Choose field to update (0 to finish):\n";
    cout << "1. Name\n2. Category\n3. Price\n4. Quantity\n5. Expiration Date\n";
    int choice;
    do {
        cout << "Choice: ";
        cin >> choice;
        cin.ignore(1000, '\n');
        switch (choice) {
            case 1: {
                string newName;
                cout << "Enter new name: ";
                getline(cin, newName);
                product->name = newName;
                break;
            }
            case 2: {
                string newCategory;
                cout << "Enter new category: ";
                getline(cin, newCategory);
                product->category = newCategory;
                break;
            }
            case 3: {
                double newPrice;
                cout << "Enter new price: ";
                cin >> newPrice;
                if (newPrice < 0) {
                    cout << "Price must be non-negative.\n";
                } else {
                    product->price = newPrice;
                }
                cin.ignore(1000, '\n');
                break;
            }
            case 4: {
                int newQuantity;
                cout << "Enter new quantity: ";
                cin >> newQuantity;
                if (newQuantity < 0) {
                    cout << "Quantity must be non-negative.\n";
                } else {
                    product->quantity = newQuantity;
                }
                cin.ignore(1000, '\n');
                break;
            }
            case 5: {
                string newDate;
                cout << "Enter new expiration date (YYYY-MM-DD): ";
                getline(cin, newDate);
```

```
                if (!isValidDate(newDate)) {
                    cout << "Invalid format. Use YYYY-MM-DD.\n";
                } else {
                    product->expirationDate = newDate;
                }
                break;
            }
            case 0:
                break;
            default:
                cout << "Invalid choice.\n";
        }
        if (choice != 0) cout << "Product updated. Choose another field or 0 to finish.\n";
    } while (choice != 0);
    cout << "Update complete.\n";
}

// Simulate a sale (increment sales count)
void recordSale(int id, int quantitySold) {
    Product* product = searchById(id);
    if (product == nullptr) {
        cout << "Product not found.\n";
        return;
    }
    if (product->quantity < quantitySold) {
        cout << "Insufficient quantity in stock.\n";
        return;
    }
    product->quantity -= quantitySold;
    product->salesCount += quantitySold;
    cout << "Sale recorded successfully.\n";
}
```

```
// Save products to a file
void saveToFile(const string& filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        cout << "Error opening file.\n";
        return;
    }
    Product* temp = start;
    file << "\nSupermarket Inventory (Sorted by ID)\n";
    file << "ID\tName\tCategory\tPrice\tQty\tExp Date\tSales\n";
    file << "------------------------------------------------------------\n";
    while (temp != nullptr) {
        file << temp->id << ",\t" << temp->name << "\",\t"<< temp->category << "\",\t"
             << temp->price << ",\t" << temp->quantity << ",\t" << temp->expirationDate << ",\t"
             << temp->salesCount << "\n";
        temp = temp->next;
    }
    file.close();
    cout << "Products saved to " << filename << " successfully.\n";
}

// Load products from a file
void loadFromFile(const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cout << "Error opening file.\n";
        return;
    }
    clearList();
    string line;
    while (getline(file, line)) {
        size_t pos = 0;
        string idStr = getNextField(line, pos);
        string name = getNextField(line, pos);
        string category = getNextField(line, pos);
        string priceStr = getNextField(line, pos);
        string qtyStr = getNextField(line, pos);
        string expDate = getNextField(line, pos);
        string salesStr = getNextField(line, pos);
        if (idStr.empty() || salesStr.empty()) {
            cout << "Invalid line in file: " << line << "\n";
            continue;
        }
        try {
            int id = stoi(idStr);
            double price = stod(priceStr);
            int quantity = stoi(qtyStr);
            int salesCount = stoi(salesStr);
            addProduct(id, name, category, price, quantity, expDate);
            Product* p = searchById(id);
            if (p) p->salesCount = salesCount;
        } catch (...) {
            cout << "Invalid data in line: " << line << "\n";
        }
    }
    file.close();
    cout << "Products loaded from " << filename << " successfully.\n";
}

// Display products by category
void displayProductsByCategory(const string& category) {
    Product* temp = start;
    bool found = false;
    cout << "\nProducts in Category: " << category << "\n";
    cout << "ID\tName\tCategory\tPrice\tQty\tExp Date\tSales\n";
    cout << "------------------------------------------------------------\n";
    while (temp != nullptr) {
        if (temp->category == category) {
            displayProduct(temp);
            found = true;
        }
        temp = temp->next;
    }
    if (!found) cout << "No products found in this category.\n";
    cout << "------------------------------------------------------------\n";
}

// Search by name
void searchByName(const string& name) {
    Product* temp = start;
    bool found = false;
    cout << "\nProducts with Name: " << name << "\n";
    cout << "ID\tName\tCategory\tPrice\tQty\tExp Date\tSales\n";
    cout << "------------------------------------------------------------\n";
    while (temp != nullptr) {
        if (temp->name == name) {
            displayProduct(temp);
            found = true;
        }
        temp = temp->next;
    }
    if (!found) cout << "No products found with name: " << name << "\n";
    cout << "------------------------------------------------------------\n";
}
```

```
// Display sorted by attribute
void displaySortedBy(bool (*compare)(Product*, Product*), const string& sortBy) {
    TempNode* sortedHead = nullptr;
    Product* current = start;
    while (current != nullptr) {
        sortedHead = sortedInsert(sortedHead, current, compare);
        current = current->next;
    }
    cout << "\nProducts Sorted by " << sortBy << "\n";
    cout << "ID\tName\tCategory\tPrice\tQty\tExp Date\tSales\n";
    cout << "--------------------------------------------------------\n";
    TempNode* temp = sortedHead;
    while (temp != nullptr) {
        displayProduct(temp->product);
        temp = temp->next;
    }
    cout << "--------------------------------------------------------\n";
    while (sortedHead != nullptr) {
        TempNode* toDelete = sortedHead;
        sortedHead = sortedHead->next;
        delete toDelete;
    }
}

// Comparison functions
static bool compareByName(Product* a, Product* b) {
    return a->name < b->name;
}
static bool compareByPrice(Product* a, Product* b) {
    return a->price < b->price;
}
static bool compareByCategory(Product* a, Product* b) {
        return a->category < b->category;
    }
    static bool compareByQuantity(Product* a, Product* b) {
        return a->quantity < b->quantity;
    }
};

// Menu-driven interface
int main() {
    SupermarketSystem system;
    int choice;
    do {
        cout << "\nSupermarket Management System\n";
        cout << "1. Add Product\n";
        cout << "2. Display All Products (Sorted by ID)\n";
        cout << "3. Display Products Sorted by Name\n";
        cout << "4. Display Products Sorted by Price\n";
        cout << "5. Display Products Sorted by Category\n";
        cout << "6. Display Products Sorted by Quantity (Low to High)\n";
        cout << "7. Search Product by ID\n";
        cout << "8. Search Product by Name\n";
        cout << "9. Search Product by Category\n";
        cout << "10. Remove Product\n";
        cout << "11. Update Product\n";
        cout << "12. Record Sale\n";
        cout << "13. Save to File\n";
        cout << "14. Load from File\n";
        cout << "15. Exit\n";
        cout << "Enter your choice: ";
        if (!(cin >> choice)) {
            cin.clear();
            cin.ignore(1000, '\n');
        cout << "Invalid input. Please enter a number.\n";
        continue;
    }
    cin.ignore(1000, '\n');

    switch (choice) {
        case 1: {
            int id, quantity;
            string name, category, expDate;
            double price;
            cout << "Enter ID: "; cin >> id;
            cin.ignore(1000, '\n');
            cout << "Enter name: "; getline(cin, name);
            cout << "Enter category: "; getline(cin, category);
            cout << "Enter price: "; cin >> price;
            cout << "Enter quantity: "; cin >> quantity;
            cin.ignore(1000, '\n');
            cout << "Enter expiration date (YYYY-MM-DD): ";
            getline(cin, expDate);
            system.addProduct(id, name, category, price, quantity, expDate);
            break;
        }
        case 2:
            system.displayProducts();
            break;
        case 3:
            system.displaySortedBy(SupermarketSystem::compareByName, "Name");
            break;
        case 4:
            system.displaySortedBy(SupermarketSystem::compareByPrice, "Price");
            break;
        case 5:
```

```
    system.displaySortedBy(SupermarketSystem::compareByCategory, "Category");
    break;
case 6:
    system.displaySortedBy(SupermarketSystem::compareByQuantity, "Quantity");
    break;
case 7: {
    int id;
    cout << "Enter ID to search: "; cin >> id;
    Product* product = system.searchById(id);
    if (product) {
        cout << "Product found:\n";
        system.displayProduct(product);
    } else {
        cout << "Product not found.\n";
    }
    break;
}
case 8: {
    string name;
    cout << "Enter name to search: ";
    getline(cin, name);
    system.searchByName(name);
    break;
}
case 9: {
    string category;
    cout << "Enter category to search: ";
    getline(cin, category);
    system.displayProductsByCategory(category);
    break;
}
case 10: {
    int id;
    cout << "Enter ID to remove: "; cin >> id;
    system.removeProduct(id);
    break;
}
case 11: {
    int id;
    cout << "Enter ID to update: "; cin >> id;
    system.updateProduct(id);
    break;
}
case 12: {
    int id, qty;
    cout << "Enter product ID: "; cin >> id;
    cout << "Enter quantity sold: "; cin >> qty;
    system.recordSale(id, qty);
    break;
}
case 13: {
    string filename;
    cout << "Enter filename to save: ";
    getline(cin, filename);
    system.saveToFile(filename);
    break;
}
case 14: {
    string filename;
    cout << "Enter filename to load: ";
    getline(cin, filename);
    system.loadFromFile(filename);
    break;
}
```

```
            case 15:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 15);

    return 0;
}
```