SECJ2013

Data Structures and Algorithms

Lecturer

Ts Dr Johanna binti Ahmad

Project Name

Food Ordering System Report

Demo Video

https://drive.google.com/drive/folders/1jgtDCq_H9MqWkVLBrCAGRmwULt3wVuTu?usp=sharing

Team Members

Ahmed Ghaleb - A21EC9120
Altayeb Mustafa - A21EC9115

Objective

This project aims to demonstrate the application of data structures, specifically arrays, in a medium-sized C++ program. The primary focus is on utilizing sorting and searching techniques within a food ordering system.

System Overview

The Food Ordering System is designed to manage a menu of food items, offering functionalities like displaying, sorting, and searching the menu. It operates on a console-based interface, allowing user interactions for various operations.

Technical Details

Attributes and Member Functions

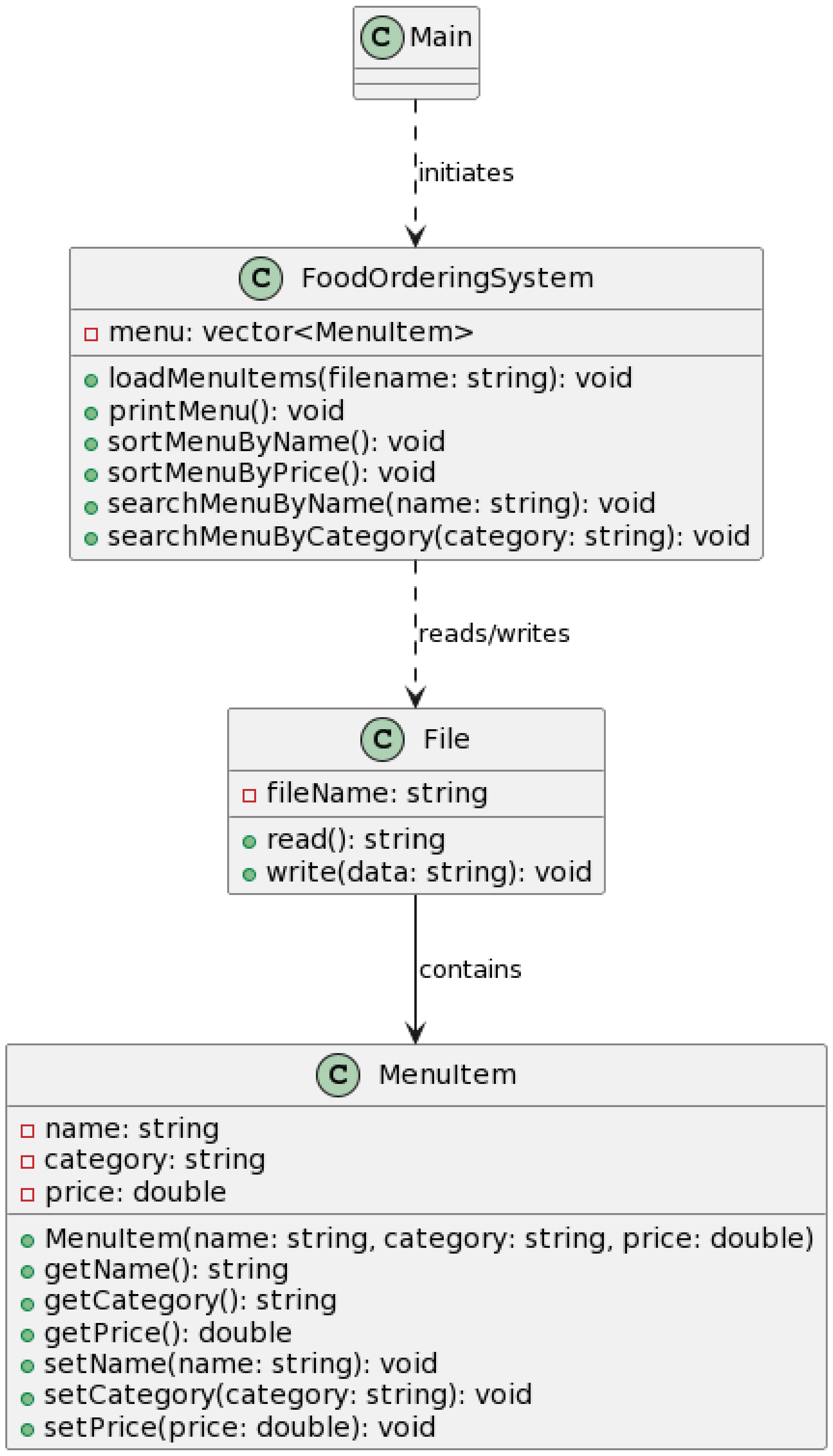
1. `MenuItem` Class

- **Attributes**:
  - `name` (string): The name of the menu item.
  - `category` (string): The category of the menu item (e.g., Appetizer, Main Course).
  - `price` (double): The price of the menu item.
- **Member Functions**:
  - Constructor initializing name, category, and price.

2. `FoodOrderingSystem` Class

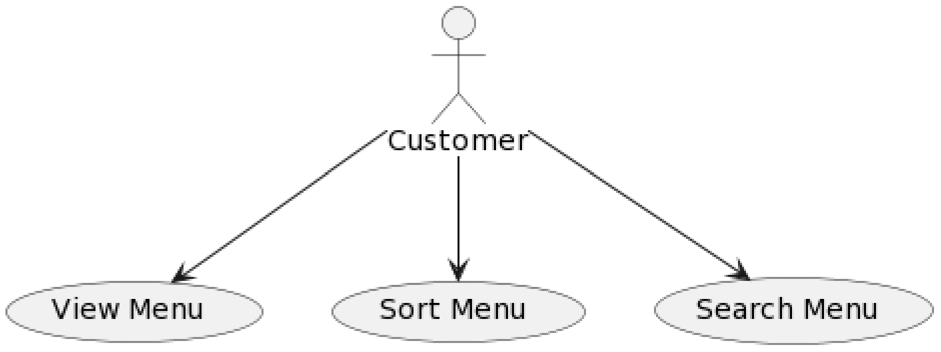
- **Attributes**:
  - `menu` (vector<MenuItem>): A dynamic array to store menu items.
- **Member Functions**:
  - `loadMenuItems`: Reads menu items from a file and stores them in `menu`.
  - `printMenu`: Displays all menu items.
  - `sortMenuByName`: Sorts menu items by name.
  - `sortMenuByPrice`: Sorts menu items by price.
  - `searchMenuByName`: Searches for menu items by name.
  - `searchMenuByCategory`: Searches for menu items by category.

Class Diagram



Main
initiates
FoodOrderingSystem
- menu: vector<MenuItem>
+ loadMenuItems(filename: string): void
+ printMenu(): void
+ sortMenuByName(): void
+ sortMenuByPrice(): void
+ searchMenuByName(name: string): void
+ searchMenuByCategory(category: string): void
reads/writes
File
- fileName: string
+ read(): string
+ write(data: string): void
contains
MenuItem
- name: string
- category: string
- price: double
+ MenuItem(name: string, category: string, price: double)
+ getName(): string
+ getCategory(): string
+ getPrice(): double
+ setName(name: string): void
+ setCategory(category: string): void
+ setPrice(price: double): void

Use Case Diagram



Implementation of Member Functions

```
void loadMenuItems(const std::string& filename) {

   std::ifstream file(filename);

   std::string line, name, category;

   double price;


   while (std::getline(file, line)) {

       std::stringstream ss(line);

       std::getline(ss, name, ',');

       std::getline(ss, category, ',');

       ss >> price;

       menu.push_back(MenuItem(name, category, price));

   }
```

```
void sortMenuByName() {

   std::sort(menu.begin(), menu.end(), [](const MenuItem& a, const MenuItem& 
b) {

       return a.name < b.name;

   });

}
```

```cpp
void sortMenuByPrice() {
   std::sort(menu.begin(), menu.end(), [](const MenuItem& a, const MenuItem& b) {
       return a.price < b.price;
   });
}
```

```cpp
void searchMenuByName(const std::string& searchName) {
   bool found = false;
   for (const auto& item : menu) {
       if (item.name.find(searchName) != std::string::npos) {
           std::cout << "Found: " << item.name << ", Category: " << item.category << ", Price: $" << item.price << std::endl;
           found = true;
       }
   }
   if (!found) {
       std::cout << "No items found with name: " << searchName << std::endl;
   }
}
```

```
void searchMenuByCategory(const std::string& searchCategory) {

   bool found = false;

   for (const auto& item : menu) {

       if (item.category == searchCategory) {

           std::cout << "Found: " << item.name << ", Category: " <<
item.category << ", Price: $" << item.price << std::endl;

           found = true;

       }

   }

   if (!found) {

       std::cout << "No items found in category: " << searchCategory <<
std::endl;

   }

}
```

Main Program

The `main` function serves as the entry point of the application. It creates an instance of `FoodOrderingSystem`, loads menu items from a data file, and provides a menu-driven interface for users to interact with the system.

Sorting and Searching Techniques

Sorting: Implemented using the standard library's `std::sort` function. The menu can be sorted either by name or price, which demonstrates an understanding of lambda expressions for custom comparison.

Searching: Linear search is used to find menu items by name or category. It iterates through the vector and matches user input with item attributes.

Challenges and Solutions

Challenge: Efficiently implementing and integrating sorting and searching algorithms with the dynamic array. **Solution**: Utilized lambda expressions and standard algorithms from the C++ Standard Library for concise and effective sorting and searching.

Conclusion

The Food Ordering System successfully demonstrates the application of arrays, sorting, and searching techniques in C++. It serves as a practical example of data structure utilization in a real-world scenario, with potential for further enhancements like integrating other data structures (linked lists, stacks, queues) for more complex operations.

Appendix

A. Data Files

`Menu.txt`

This file contains the sample data for the menu items. Each line in the file represents a single menu item, with its attributes (name, category, price) separated by commas.

Content of `menu.txt`:

```
Chicken Burger,Sandwich,5.99
Veggie Burger,Sandwich,4.99
Caesar Salad,Salad,6.49
Pasta Carbonara,Pasta,8.99
Margherita Pizza,Pizza,7.99
Pepperoni Pizza,Pizza,8.49
Mushroom Risotto,Risotto,9.99
Tofu Stir Fry,Vegan,7.49
Beef Tacos,Tacos,6.99
Chicken Tacos,Tacos,6.49
Quinoa Salad,Salad,5.49
Spaghetti Bolognese,Pasta,8.49
Greek Salad,Salad,6.99
Hawaiian Pizza,Pizza,8.99
Vegetable Curry,Vegan,7.99
Chicken Wings,Appetizer,5.49
Garlic Bread,Appetizer,3.99
Chocolate Cake,Dessert,4.49
Cheesecake,Dessert,4.99
Apple Pie,Dessert,3.99
```

B. Source Code

`MenuItem` Class

```
class MenuItem {
public:
   std::string name;
   std::string category;
   double price;

   MenuItem(std::string name, std::string category, double price) :
name(name), category(category), price(price) {

   }
};
```

`FoodOrderingSystem` Class

```
class FoodOrderingSystem {
private:
   std::vector<MenuItem> menu;

public:
   void loadMenuItems(const std::string& filename) {
       std::ifstream file(filename);
       std::string line, name, category;
       double price;

       while (std::getline(file, line)) {
           std::stringstream ss(line);
           std::getline(ss, name, ',');
           std::getline(ss, category, ',');
           ss >> price;
           menu.push_back(MenuItem(name, category, price));
       }
   }

   void printMenu() const {
       for (const auto& item : menu) {
           std::cout << "Name: " << item.name << ", Category: " <<
item.category << ", Price: $" << item.price << std::endl;
       }
   }


   void sortMenuByName() {
       std::sort(menu.begin(), menu.end(), [](const MenuItem& a, const
MenuItem& b) {
           return a.name < b.name;
       });
```

```
    }

    void sortMenuByPrice() {
        std::sort(menu.begin(), menu.end(), [](const MenuItem& a, const
MenuItem& b) {
            return a.price < b.price;
        });
    }

    void searchMenuByName(const std::string& searchName) {
        bool found = false;
        for (const auto& item : menu) {
            if (item.name.find(searchName) != std::string::npos) {
                std::cout << "Found: " << item.name << ", Category: " <<
item.category << ", Price: $" << item.price << std::endl;
                found = true;
            }
        }
        if (!found) {
            std::cout << "No items found with name: " << searchName <<
std::endl;
        }
    }

    void searchMenuByCategory(const std::string& searchCategory) {
        bool found = false;
        for (const auto& item : menu) {
            if (item.category == searchCategory) {
                std::cout << "Found: " << item.name << ", Category: " <<
item.category << ", Price: $" << item.price << std::endl;
                found = true;
            }
        }
        if (!found) {
            std::cout << "No items found in category: " << searchCategory <<
std::endl;
        }
    }

};
```

Main Function

```cpp
int main() {
   FoodOrderingSystem system;
   system.loadMenuItems("menu.txt"); // Load menu items from the file

   int choice;
   std::string input;

   do {
       std::cout << "\nFood Ordering System Menu\n";
       std::cout << "1. Print Menu\n";
       std::cout << "2. Sort Menu by Name\n";
       std::cout << "3. Sort Menu by Price\n";
       std::cout << "4. Search Menu by Name\n";
       std::cout << "5. Search Menu by Category\n";
       std::cout << "0. Exit\n";
       std::cout << "Enter your choice: ";
       std::cin >> choice;

       switch (choice) {
           case 1:
               system.printMenu();
               break;
           case 2:
               system.sortMenuByName();
               system.printMenu();
               break;
           case 3:
               system.sortMenuByPrice();
               system.printMenu();
               break;
           case 4:
               std::cout << "Enter name to search: ";
               std::cin.ignore(); // Clear newline left in the buffer
               std::getline(std::cin, input);
               system.searchMenuByName(input);
               break;
           case 5:
               std::cout << "Enter category to search: ";
               std::cin.ignore(); // Clear newline left in the buffer
               std::getline(std::cin, input);
               system.searchMenuByCategory(input);
               break;
           case 0:
               std::cout << "Exiting the program.\n";
               break;
           default:
               std::cout << "Invalid choice. Please try again.\n";
       }
   } while (choice != 0);

   return 0;
}
```