İTÜGAE
GÜNEŞ ARABASI EKİBİ
2004

powered by
ZES
Zorlu Energy Solutions

ISTANBUL TECHNICAL UNIVERSITY
ITU ZES Solar Car Team

Assignment-2

1 Restaurant Bot

In this assignment, you dive into the OOP in C++. You must implement a restaurant bot that helps people with their fancy nights. Overall structure and extra utilities are up to you but you are highly encouraged to do your best. In this assignment, you are given a menu.json which contains your restaurant's menu, and you must read from this file, then within this file, you need to implement a menu bot that runs on the terminal. Refer to this link for the menu.json file.

Classes that you must implement:

- MenuItem
- Starter
- Salad
- MainCourse
- Drink
- Appetizer
- Dessert
- Menu
- User

You must consider the encapsulation concept in OOP design, hold your attributes as private, and reach them by getters and setters. Be careful about attribute naming to be clear, and all over your code please put necessary comments.

The bot you wrote must first introduce itself and its functionalities to the user. Then you need to take the user information and create a User object.

The CLI design is up to you but you can divide it into functions, then call them upon user selections.

User Class

You need to hold user information such as:

- First name
- Last name
- Gender to address (Mr. or Mrs.)
- Menu chosen by the user

User can reach the details of their menu by the command line. All menu operations can be made by the user in the terminal.

Menu Class

The menu attribute of the user is related to the Menu class, which holds such attributes:

- Vector of the MenuItem
- Overall taste_balance of the menu
- Total cost of the menu

You must read, add, delete, and update the current menu of the user from the command line. You must implement these functions in the Menu class.

MenuItem Class

This class is an interface for your food classes, and their functions must be purely virtual and hold such attributes:

- Name of the food
- Price of the food as USD
- Taste balance of the food

You need to also implement getter and setter functions for this base class to be used later in child classes. If needed, any additional functions specific to the child's class must be written also. The other 5 food classes inherit from this base class and you need to overwrite the functions. Some additional attributes may be:

- **Starter:** You need to ask the user whether they want their food hot or cold, and then hold it as an attribute.
- **Salad:** You need to ask the user whether they want topping for their salad and then ask what they want. When the topping is added, it costs 2.25 USD.
- **MainCourse:** You need to ask the user whether they want vegetarian food for the main course, then according to this selection, your suggestions must change.
- **Drink:** You need to ask them whether they want their drink carbonated, and also whether they want additional alcohol shot in their drink. The cost for carbonation is 0.5 USD and the extra shot costs 2.5 USD.
- **Appetizer:** You need to ask the user the time when you must serve the appetizer, whether before the main course or after the main course.
- **Dessert:** You need to ask the user whether they want extra chocolate to their dessert. The cost of the extra chocolate is 1.5 USD.

When you first read the menu from the JSON file you may give default parameters to the constructor of these classes. However, when you are interacting with the user, you can create objects by the user's choices. Do not forget to use the main class' constructor in addition to the overwritten constructor.

All of the classes above must be in the namespace "menu". In addition to these menu items, you must add some intelligence to the bot you wrote. In the beginning, users have a chance to make you create a random menu or the menu with specific taste_balance that the user can give.

AI-Based Menu Recommendation System

To enhance your Restaurant Bot, you are encouraged to integrate a simple AI-based recommendation system that can predict user satisfaction and suggest appropriate menus. The goal of this section is to apply basic machine learning logic, specifically **Linear Regression**, within your C++ implementation.

Objective: Use Linear Regression to model the relationship between the user's *taste balance* and their *satisfaction score*. For instance, each menu can be represented as a set of numerical features:

$$Menu = [sweet, salty, sour, bitter, spicy]$$

and the user's satisfaction can be represented as a numeric value between 0 and 1.

You can use the following simplified regression formula:

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + \ldots + w_5x_5$$

where:

- x_i are the taste balance values (sweetness, saltiness, etc.),
- w_i are the learned weights for each taste dimension,
- $\hat{y}$ is the predicted satisfaction score.

Implementation Details:

- You may initialize the weights w_i randomly or with small constants.
- After each user feedback, update the weights using a simple gradient step:

  $$w_i = w_i + \alpha(y - \hat{y})x_i$$

  where α is the learning rate (e.g., 0.01).
- The bot should store these weights in a JSON or text file so that the system "learns" from previous users.

Expected Outcome:

- The Restaurant Bot gradually learns user preferences through interaction.
- The next time a user comes, the bot predicts their satisfaction for each menu and recommends the one with the highest predicted score.
- The approach simulates a basic supervised learning model integrated into an OOP system.

This section aims to introduce students to the fundamental concept of using simple machine learning methods within object-oriented design and to encourage the implementation of data-driven decision-making in C++.

You must randomly choose one from each of the food classes and output it to the user. The user may like or not like, and repeat the random process. When taste_balance is given by the user, you must choose your food instances to fit the given taste values in each of the 5 tastes at average, also the user may choose the taste_balance as balanced, and then the bot suggests a balanced menu. With the taste_balance option, you must try to fit your suggestions to the given taste_balance at the average for each one of 5 taste types. In brief, your suggestions from 5 different food types must fit in the user's taste_balance profile.

Your other utility functions or intelligence functions must be inside of another meaningful namespace. For the menu suggestion functionality, you may use your own design or search for a solution.

You must write 3 different files which are main.cpp, Menu.cpp, Menu.hpp. You need to separate your work into these 3 files and all of the operations must be done in the main.cpp file. In the main function, you need to read from the JSON file and then hold all of the possible foods to use later in your program.

General Coding Practices

- **Consistent Coding Style:** Whichever style you use in your code does not matter, but be consistent using the same naming convention.
- **Descriptive Variable Names:** Use meaningful and descriptive names for variables that reflect their purpose or content.
- **Consistent Indentation:** Maintain consistent indentation to enhance code readability and structure.
- **Modularization:** Break down code into modular functions or classes to promote code reuse and maintainability.
- **Comments:** Include clear and concise comments to explain complex sections of code or to provide context for future developers.
- **Error Handling and Input Validation:** All over your code, you must consider any bug or error, and handle them correctly to not interrupt the program, especially for the user inputs.

In addition to code work, you need to write a report that briefly explains your class relations. To do that, you must search and explain the concepts of **inheritance, association, composition, and aggregation**, which are the 4 types of relationships in OOP. Explain your class relations using these concepts.

For your comments and explanations of the functions you implemented, you can use the Doxygen format and create documentation for your application, maybe you can use ss' from the generated Doxygen documents.

You must push all of your work to your GitHub repo. You must correctly separate your projects and also write documentation that explains your project. For a sample repo, you may refer to this link.

2 Conclusion

For any questions and errors you find, contact Mehmet Emre Demir. While preparing your work consider readability and scalability, and also do not forget to add necessary comments to your codes.