Algorithmic Adventures: The Virtual Bistro – Catering to Dietary Needs

---

Project Overview

Welcome back to your culinary adventure! In this project, you will continue to enhance your virtual bistro simulation by introducing dietary accommodations. Building upon the `Dish`, `Appetizer`, `MainCourse`, `Dessert`, and `Kitchen` classes from previous projects, you will modify these classes to enable dishes to be adjusted based on specific dietary needs.

As the head chef and manager, you will now cater to customers with dietary restrictions and preferences. You will implement functionality that allows dishes to be modified to accommodate these needs, and you will manage these dishes dynamically. This project will test your understanding of advanced Object-Oriented Programming (OOP) concepts such as abstract classes, polymorphism, virtual functions, dynamic memory allocation, and file I/O.

Here is the link to accept this project on GitHub Classroom:
https://classroom.github.com/a/YS2C9Xk7

Part 1: Documentation Requirements

As with all previous projects, documentation is crucial. You will receive 15% of the points for proper documentation. Ensure that you:

1. **File-level Documentation**: Include a comment at the top of each file with your name, date, and a brief description of the code implemented in that file.
2. **Function-level Documentation**: Before each function declaration and implementation, include a comment that describes:
   - **@pre**: Preconditions for the function.
   - **@param**: Description of each parameter.
   - **@return**: Description of the return value.
   - **@post**: Postconditions after the function executes.
3. **Inline Comments**: Add comments within your functions to explain complex logic or important steps.

Remember: All files and all functions must be commented, including both `.hpp` and `.cpp` files.

Part 2: Additional Resources

If you need to brush up on or learn new concepts, the following resources are recommended:

- **Abstract Classes and Polymorphism**:
  - Abstract Classes in C++ (GeeksforGeeks)
  - Polymorphism in C++ (Programiz)
- **File I/O in C++**:
  - File Handling (cplusplus.com)
  - Reading from Files in C++ (GeeksforGeeks)
- **Dynamic Memory Allocation**:
  - Pointers and Dynamic Memory (cplusplus.com)
  - Smart Pointers in C++ (GeeksforGeeks)
- **String Manipulation**:
  - String Stream in C++ (GeeksforGeeks)
  - String Functions (cplusplus.com)

Part 3: Modifying Classes for Polymorphism and Dietary Accommodations

You will modify the `Dish` class and its subclasses to enable dynamic adjustments based on dietary needs.

Key Concepts

- **Abstract Classes and Pure Virtual Functions**: The `Dish` class will become an abstract class with pure virtual functions to enforce that each subclass implements specific functionalities.
- **Polymorphism**: The `Kitchen` will store pointers to `Dish` objects, allowing it to handle different dish types polymorphically.
- **Dynamic Memory Allocation**: You will use dynamic memory to manage `Dish` objects through pointers.

- **File I/O**: The kitchen will be initialized by reading dishes from a CSV file through a parameterized constructor.

Step 1: Modify the `Dish` Class

- **Add a Structure for Dietary Accommodations**

  Inside the `Dish` class, define a new structure `DietaryRequest` that will store information about dietary accommodations.

  ```
  /**
   * Structure to store dietary accommodation details.
   */
  struct DietaryRequest {
      bool vegetarian;
      bool vegan;
      bool gluten_free;
      bool nut_free;
      bool low_sodium;
      bool low_sugar;
  };
  ```

- **Make the `display()` Function Pure Virtual**

  ```
  /**
   * Pure virtual function to display dish details.
   * Must be overridden by derived classes.
   */
  ```

- **Add a Pure Virtual Function `dietaryAccommodations()`**

  ```
  /**
   * Modifies the dish to accommodate specific dietary needs.
   * @param request A reference to a DietaryRequest structure specifying
  the dietary accommodations.
   * @post Modifies the dish's attributes based on the accommodations.
   */
  ```

- **Update Other Member Functions as Needed**

If there are any, ensure that any functions that need to be overridden by subclasses are modified accordingly. Review the lecture slides and provided additional resources for the correct syntax!

Step 2: Modify the Subclasses

Each subclass (`Appetizer`, `MainCourse`, `Dessert`) will implement the `display()` and `dietaryAccommodations()` functions.

Appetizer Class

- **Implement `display()`**

```
/**
 * Displays the appetizer's details.
 * @post Outputs the appetizer's details, including name, ingredients,
preparation time, price, cuisine type, serving style, spiciness level, and
vegetarian status, to the standard output.
 * The information must be displayed in the following format:
 *
 * Dish Name: [Name of the dish]
 * Ingredients: [Comma-separated list of ingredients]
 * Preparation Time: [Preparation time] minutes
 * Price: $[Price, formatted to two decimal places]
 * Cuisine Type: [Cuisine type]
 * Serving Style: [Serving style: Plated, Family Style, or Buffet]
 * Spiciness Level: [Spiciness level]
 * Vegetarian: [Yes/No]
 */
```

Note: When printing the enums, if the exact format (case-sensitive) is not specified in the function-level documentation that you're provided above, they're expected to be printed in all caps. If it is specified, then you are expected to output it as specified.

- **Implement `dietaryAccommodations()`**

  ```
  /**
   * Modifies the appetizer based on dietary accommodations.
   * @param request A DietaryRequest structure specifying the dietary
  accommodations.
   * @post Adjusts the appetizer's attributes to meet the specified
  ```

```
dietary needs.
 *       - If `request.vegetarian` is true:
 *           - Sets `vegetarian_` to true.
 *           - Searches `ingredients_` for any non-vegetarian
ingredients and replaces the first occurrence with "Beans". If there are
other non-vegetarian ingredients, the next non-vegetarian ingredient is
replaced with "Mushrooms".  If there are more, they will be removed
without substitution.
 *             Non-vegetarian ingredients are: "Meat", "Chicken",
"Fish", "Beef", "Pork", "Lamb", "Shrimp", "Bacon".
 *       - If `request.low_sodium` is true:
 *           - Reduces `spiciness_level_` by 2 (minimum of 0).
 *       - If `request.gluten_free` is true:
 *           - Removes gluten-containing ingredients from
`ingredients_`.
 *             Gluten-containing ingredients are: "Wheat", "Flour",
"Bread", "Pasta", "Barley", "Rye", "Oats", "Crust".
 */
```

MainCourse Class

- **Implement `display()`**

```
/**
 * Displays the main course's details.
 * @post Outputs the main course's details, including name, ingredients,
preparation time, price, cuisine type, cooking method, protein type,
side dishes, and gluten-free status to the standard output.
 * The information must be displayed in the following format:
 *
 * Dish Name: [Name of the dish]
 * Ingredients: [Comma-separated list of ingredients
 * Preparation Time: [Preparation time] minutes
 * Price: $[Price, formatted to two decimal places]
 * Cuisine Type: [Cuisine type]
 * Cooking Method: [Cooking method: e.g., Grilled, Baked, etc.]
 * Protein Type: [Type of protein: e.g., Chicken, Beef, etc.]
 * Side Dishes: [Side dish name] (Category: [Category: e.g., Starches,
Vegetables])
 * Gluten-Free: [Yes/No]
 */
```

- **Implement `dietaryAccommodations()`**

```
/**
 * Modifies the main course based on dietary accommodations.
 * @param request A DietaryRequest structure specifying the dietary
accommodations.
 * @post Adjusts the main course's attributes to meet the specified
dietary needs.
 *       - If `request.vegetarian` is true:
 *           - Changes `protein_type_` to "Tofu".
 *           - Searches `ingredients_` for any non-vegetarian
ingredients and replaces the first occurrence with "Beans". If there are
other non-vegetarian ingredients, the next non-vegetarian ingredient is
replaced with "Mushrooms". If there are more, they will be removed
without substitution.
 *             Non-vegetarian ingredients are: "Meat", "Chicken",
"Fish", "Beef", "Pork", "Lamb", "Shrimp", "Bacon".
 *       - If `request.vegan` is true:
 *           - Changes `protein_type_` to "Tofu".
 *           - Removes dairy and egg ingredients from `ingredients_`.
 *             Dairy and egg ingredients are: "Milk", "Eggs", "Cheese",
"Butter", "Cream", "Yogurt".
 *       - If `request.gluten_free` is true:
 *           - Sets `gluten_free_` to true.
 *           - Removes side dishes from `side_dishes_` whose category
involves gluten.
 *             Gluten-containing side dish categories are: `GRAIN`,
`PASTA`, `BREAD`, `STARCHES`.
 */
```

Dessert Class

- **Implement `display()`**

```
/**
 * Displays the dessert's details.
 * @post Outputs the dessert's details, including name, ingredients,
preparation time, price, cuisine type, flavor profile, sweetness level, and
whether it contains nuts.
 * The information must be displayed in the following format:
 *
 * Dish Name: [Name of the dish]
 * Ingredients: [Comma-separated list of ingredients]
 * Preparation Time: [Preparation time] minutes
 * Price: $[Price, formatted to two decimal places]
 * Cuisine Type: [Cuisine type]
 * Flavor Profile: [Flavor profile: Sweet, Bitter, Sour, Salty, or Umami]
 * Sweetness Level: [Sweetness level]
 * Contains Nuts: [Yes/No]
 */
```

- **Implement `dietaryAccommodations()`**

  ```
  /**
   * Modifies the dessert based on dietary accommodations.
   * @param request A DietaryRequest structure specifying the dietary
  accommodations.
   * @post Adjusts the dessert's attributes to meet the specified dietary
  needs.
   *       - If `request.nut_free` is true:
   *           - Sets `contains_nuts_` to false.
   *           - Removes nuts from `ingredients_`.
   *             Nuts are: "Almonds", "Walnuts", "Pecans", "Hazelnuts",
  "Peanuts", "Cashews", "Pistachios".
   *       - If `request.low_sugar` is true:
   *           - Reduces `sweetness_level_` by 3 (minimum of 0).
   *       - If `request.vegan` is true:
   *           - Removes dairy and egg ingredients from `ingredients_`.
   *             Dairy and egg ingredients are: "Milk", "Eggs", "Cheese",
  "Butter", "Cream", "Yogurt".
   */
  ```

Note for all subclasses: When multiple ingredients are removed, at most two replacement ingredients should be added, and they should replace the first and second (when applicable) occurrences of the ingredients to be removed.

Step 3: Modify the `Kitchen` Class

- **Store Pointers to `Dish` Objects**

  Modify the `Kitchen` class to store pointers to `Dish` objects (`Dish*`), allowing for polymorphic behavior.
- **Implement a Parameterized Constructor**

```
/**
 * Parameterized constructor.
 * @param filename The name of the input CSV file containing dish
information.
 * @pre The CSV file must be properly formatted.
 * @post Initializes the kitchen by reading dishes from the CSV file and
storing them as `Dish*`.
 */
```

Details:

- The constructor reads the CSV file line by line.
- For each line, it creates a new dish object dynamically based on the `DishType`.
- It sets the dish's attributes accordingly.
- Adds the dish to the kitchen.

- **Implement `dietaryAdjustment()`**

```
/**
 * Adjusts all dishes in the kitchen based on the specified dietary
accommodation.
 * @param request A DietaryRequest structure specifying the dietary
accommodations.
 * @post Calls the `dietaryAccommodations()` method on each dish in the
kitchen to adjust them accordingly.
 */
```

- **Implement `displayMenu()`**

```
/**
 * Displays all dishes currently in the kitchen.
 * @post Calls the `display()` method of each dish.
 */
```

- **Destructor**

```
/**
 * Destructor.
 * @post Deallocates all dynamically allocated dishes to prevent memory
leaks.
 */
```

Example CSV File

To help you understand the format expected for the CSV file, here is a brief model CSV file that you can use as a reference:

```
DishType,Name,Ingredients,PreparationTime,Price,CuisineType,AdditionalAttributes
APPETIZER,AA,A;B;C,5,10.00,ITALIAN,PLATED;3;true
MAINCOURSE,MA,A;B;C,5,10.00,ITALIAN,GRILLED;BEEF;RICE:STARCHES|BEANS:VEGETABLE;false
DESSERT,DA,A;B;C,5,10.00,ITALIAN,SWEET;3;true
```

Explanation:

- **APPETIZER:**
  - **Name**: AA
  - **Ingredients**: A, B, C
  - **PreparationTime**: 5 minutes
  - **Price**: $10.00
  - **CuisineType**: ITALIAN
  - **AdditionalAttributes**:
    - **ServingStyle**: PLATED
    - **SpicinessLevel**: 3
    - **Vegetarian**: true
- **MAINCOURSE:**
  - **Name**: MA
  - **Ingredients**: A, B, C
  - **PreparationTime**: 5 minutes
  - **Price**: $10.00
  - **CuisineType**: ITALIAN
  - **AdditionalAttributes**:
    - **CookingMethod**: GRILLED
    - **ProteinType**: BEEF
    - **SideDishes**:
      - Rice (Category: STARCHES)
      - Beans (Category: VEGETABLE)
    - **GlutenFree**: false
- **DESSERT:**

- **Name**: DA
- **Ingredients**: A, B, C
- **PreparationTime**: 5 minutes
- **Price**: $10.00
- **CuisineType**: ITALIAN
- **AdditionalAttributes**:
    - **FlavorProfile**: SWEET
    - **SweetnessLevel**: 3
    - **ContainsNuts**: true

Testing

You will be provided with a large csv file that you can use to test your code. *It would not be good practice to test it only using this full file, you should shorten it for testing or make your own debug csv with clearer inputs.* Once you can verify that code is working with that shortened version, you should then test it with the larger one to ensure all of the different cases are handled correctly, then (if they are) submit to gradescope!

If you are experiencing errors reading in the csv file, especially if you are on a Windows device, you should confirm what the line endings of the file are. For most people, it will be the exactly as we went over in class. Some devices may use different methods of encoding, which can result in different line endings. If this applies to you, you should either ensure the endings/encoding on your device match the ones on the lab machines or account for both in your code.

To help you establish a good practice for testing, the thorough testing of your code is part of the assignment. Although you are not required to submit your test file, you must continue to thoroughly and methodically test your code.

Start by stubbing all expected functions. Have all function declarations in the `.hpp` files and stubs for all functions in the `.cpp` files. When submitted as such, your program will compile, although you will fail all tests since you have not implemented any functions yet. If your program compiles, you will have at least established that all functions have the correct names, parameters, and return types. If it does not compile, you should refer to the Common Compilation Errors document.

What is a stub?

A stub is a dummy implementation that always returns a single value for testing (or has an empty body, if void). Don't forget to go back and implement the stub! If you put the word **STUB** in a comment, some editors will make it more visible.

Now you can start implementing and testing your project, **one function at a time**!

Write a `main()` function to test your implementation. Choose the order in which you implement your methods so that you can test incrementally: i.e., implement constructors, then accessor functions, then mutator functions.

For each class, test each function you implement with all edge cases before you move on to implement the next function. This includes all constructors.

TEST THE FUNCTIONS LOCALLY BEFORE SUBMITTING TO GRADESCOPE! If it does not work on your machine, it *definitely* won't work on the autograder.

Submission

You will submit your solution to Gradescope via GitHub Classroom. The autograder will grade the following files:

- `Dish.hpp`
- `Dish.cpp`
- `Appetizer.hpp`
- `Appetizer.cpp`
- `MainCourse.hpp`
- `MainCourse.cpp`
- `Dessert.hpp`
- `Dessert.cpp`
- `Kitchen.hpp`
- `Kitchen.cpp`

Do not modify any other files from previous projects unless instructed.

Submission Instructions:

- Ensure that your code compiles and runs correctly on the Linux machines in the labs at Hunter College.
- Use the provided `Makefile` to compile your code.
- Push the code you want to submit to the GitHub repository created by GitHub Classroom.

- Submit your assignment on Gradescope by linking it to your GitHub repository.

Grading Rubric

- **Correctness**: 80% (distributed across unit testing of your submission)
- **Documentation**: 15%
- **Style and Design**: 5% (proper naming, modularity, and organization)

Due Date

This project is **due on 10/28 at 5:30 PM EST.**
No late submissions will be accepted.**

Important Notes

- **Start Early**: Begin working on the project as soon as it is assigned to identify any issues and seek help if needed.
- **No Extensions**: There will be no extensions and no negotiation about project grades after the submission deadline.
- **Help Resources**: Help is available via drop-in tutoring in Lab 1001B (see Blackboard or Discord for schedule). Starting early will allow you to get the help you need.

Authors: Michael Russo, Georgina Woo, Prof. Wole

Credit to Prof. Ligorio