OOP Concepts in C++ with Project

1. Classes and Objects

- The program uses multiple classes: FileHandler, User, VeganRecipeBook, NonVeganRecipeBook, Recipe, Ingredient, ShoppingList, etc.

- Objects of these classes are instantiated and used throughout the program.

2. Encapsulation

- Classes encapsulate data (attributes) and behaviors (methods) related to specific entities.

- For example, the Recipe class encapsulates recipe name, ingredients, and procedure.

3. Inheritance

- The VeganRecipeBook and NonVeganRecipeBook classes inherit from the RecipeBook class.

- The Recipe and Ingredient classes inherit from the NamedEntity class.

4. Polymorphism

- Method overriding is used in VeganRecipeBook and NonVeganRecipeBook classes (e.g., addRecipe, updateRecipe, deleteRecipe methods).

- The program uses virtual functions in the RecipeBook class, allowing for runtime polymorphism.

5. Abstraction

- The RecipeBook class serves as an abstract base class for VeganRecipeBook and NonVeganRecipeBook.

- It defines a common interface for recipe management operations.

6. Composition

- The Recipe class has a composition relationship with the Ingredient class (a Recipe is composed of Ingredients).

- The ShoppingList class is composed of Ingredients.

7. Association

- The User class has an association with RecipeBook and ShoppingList classes through its methods.

8. Operator Overloading

- The << operator is overloaded for the Recipe and ShoppingList classes to allow easy printing of their contents.

9. Access Specifiers

- Public, private, and protected access specifiers are used to control the visibility of class members.

10. Static Members

- The FileHandler class uses static methods for file operations.

11. Const Correctness

- Const methods are used in several classes to indicate that they don't modify the object's state (e.g., display() in ShoppingList).

12. Exception Handling

- Try-catch blocks are used in the FileHandler class to handle potential errors during file operations.

13. Templates

- While not explicitly used in the provided code, the use of STL containers (vector, unordered\_map) implies the use of template classes.

14. Friend Functions

- The operator<< functions for Recipe and ShoppingList are declared as friend functions to allow them access to private members.

15. Namespaces

- The std namespace is used throughout the code.

These OOP concepts collectively contribute to creating a modular, extensible, and maintainable recipe management system. They allow for clear separation of concerns, code reuse, and flexibility in adding new features or modifying existing ones.

**FUNCTIONALITY –**

**1. RecipeBook.h**

This header defines the RecipeBook class, which stores recipes and allows the user to perform various operations on them.

**Classes:**

* RecipeBook

**Functions:**

* **addRecipe(const Recipe &recipe)**: Adds a recipe to the RecipeBook.
* **searchRecipe(const string &name)**: Searches for a recipe by its name and returns a pointer to it if found.
* **displayAllRecipes() const**: Displays the names and number of ingredients for all recipes.
* **updateRecipe(const string &name)**: Virtual method to update a recipe (default is a message saying it's not implemented).
* **deleteRecipe(const string &name)**: Virtual method to delete a recipe by name.
* **getRecipes() const**: Returns all recipes as an unordered map.

**2. ShoppingList.h**

This header defines the ShoppingList class, which manages a list of ingredients based on recipes.

**Classes:**

* ShoppingList

**Functions:**

* **addIngredients(const Ingredient &ingredient)**: Adds an ingredient to the shopping list, increasing the quantity if it's already present.
* **addRecipeIngredients(const vector<Ingredient> &recipeIngredients)**: Adds the ingredients from a recipe to the shopping list.
* **display() const**: Displays all ingredients in the shopping list.
* **operator<<(ostream &os, const ShoppingList &shoppingList)**: Overloads the << operator to output the shopping list contents to a stream.

**3. User.h**

This header defines the User class, which interacts with the RecipeBook to request recipes and create shopping lists.

**Classes:**

* User

**Functions:**

* **requestRecipe(RecipeBook &recipeBook, const string &recipe\_name)**: Retrieves a recipe by name from the RecipeBook.
* **createShoppingList(const vector<Recipe \*> &recipes, double factor)**: Creates a ShoppingList by scaling the ingredients of the specified recipes.

**4. VeganRecipeBook.h**

This header defines the VeganRecipeBook class, which is derived from RecipeBook and ensures that only vegan recipes are added.

**Classes:**

* VeganRecipeBook

**Functions:**

* **addRecipe(const Recipe &recipe) override**: Overrides RecipeBook::addRecipe to only allow vegan recipes.
* **updateRecipe(const string &name) override**: Overrides RecipeBook::updateRecipe to ensure the updated recipe remains vegan.
* **deleteRecipe(const string &name) override**: Overrides RecipeBook::deleteRecipe to provide custom deletion logic.

**5. FileHandler.h**

This header defines the FileHandler class, which handles loading and saving recipes and shopping lists to and from files.

**Classes:**

* FileHandler

**Functions:**

* **loadRecipes(const string &filename)**: Loads recipes from a file and returns them as a vector of Recipe objects.
* **saveShoppingList(const ShoppingList &shoppingList, const string &filename)**: Saves a ShoppingList to a file.
* **saveRecipes(const RecipeBook &recipeBook, const string &filename)**: Saves all recipes from a RecipeBook to a file.

**6. Ingredient.h**

This header defines the Ingredient class, which represents an ingredient in a recipe.

**Classes:**

* Ingredient

**Functions:**

* **Ingredient(string name, double quantity, string unit)**: Constructor that initializes the ingredient's name, quantity, and unit.

**7. Recipe.h**

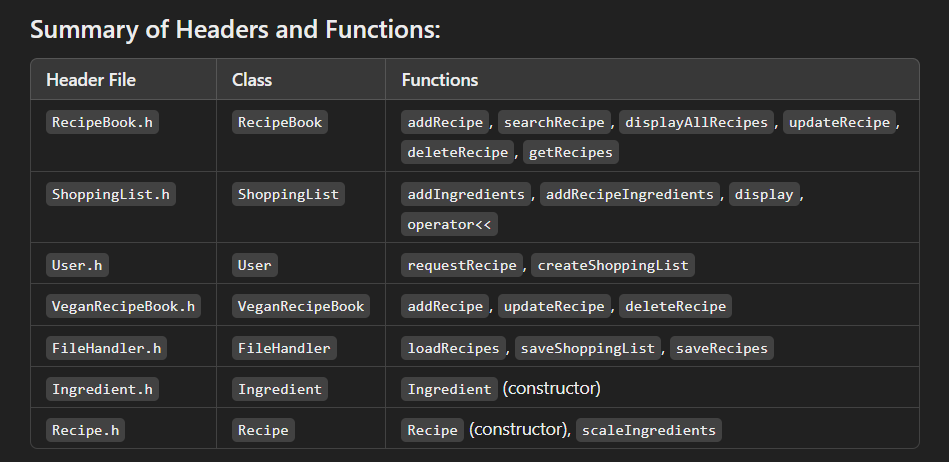
This header defines the Recipe class, which stores a recipe's name, ingredients, and procedure.

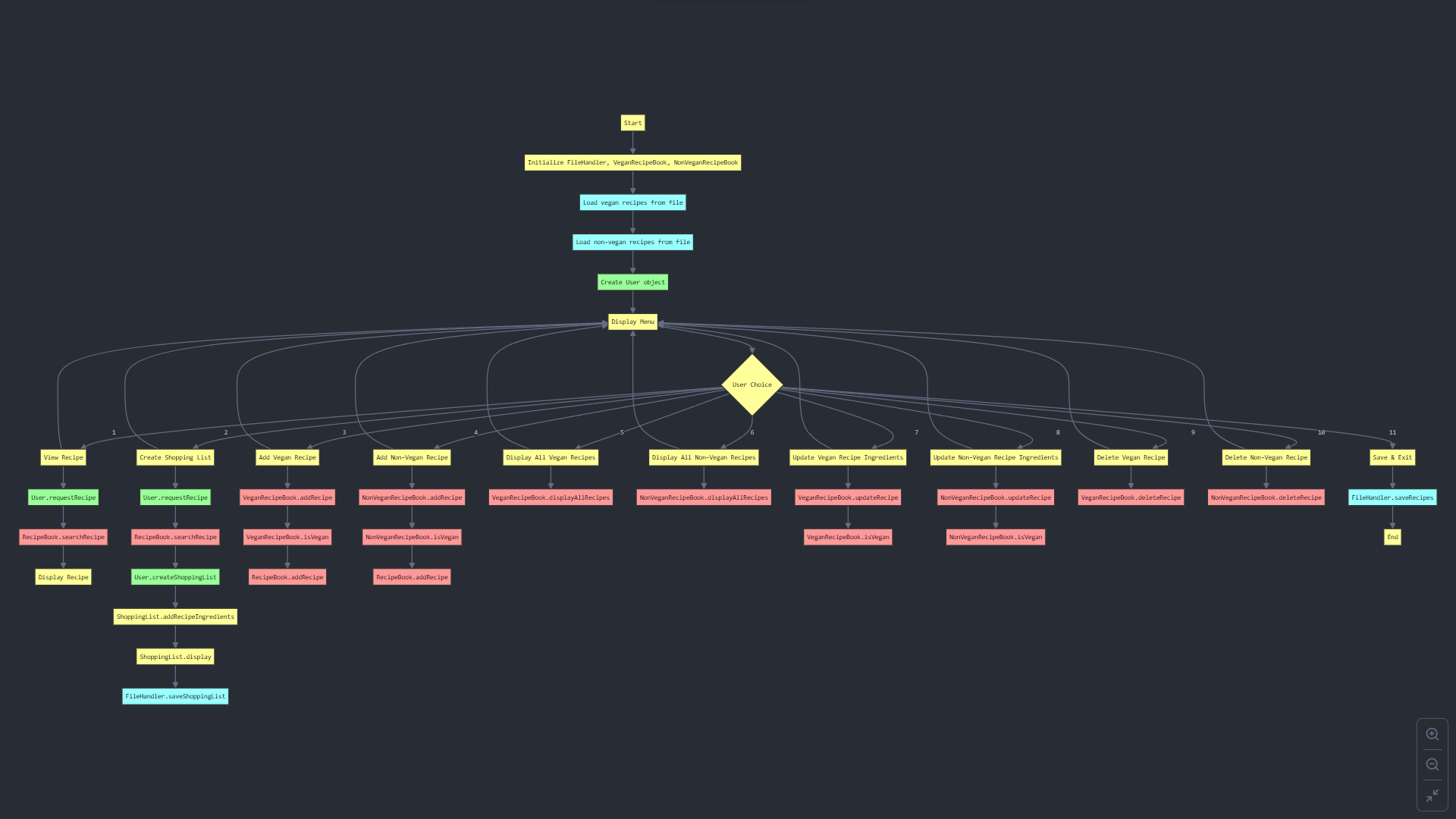
**Classes:**

* Recipe

**Functions:**

* **Recipe(string name, vector<Ingredient> ingredients, string procedure)**: Constructor that initializes a recipe with a name, ingredients, and procedure.
* **scaleIngredients(double factor)**: Scales the ingredient quantities by a given factor and returns the scaled ingredients.

FLOWCHART -



ROLES –

File Management – Gaurang Srivastava and Unmesh Varade

Classes – Ananya Singh and Divit Srivastava

Menu Arrangement (flow)– Everyone