Recipe Management App

Documentation(Mid+End Sem-Version)

Indian Institute of Information Technology, Design & Manufacturing, Kancheepuram

PREPARED BY

Rohit Kumar - CS23B2053

Gyan Chandra - CS23I1053

1 Project Overview

This section provides an overview of the Recipe Management App, which is developed as part of this project using core Object-Oriented Programming (OOP) principles.

1.1 Objective

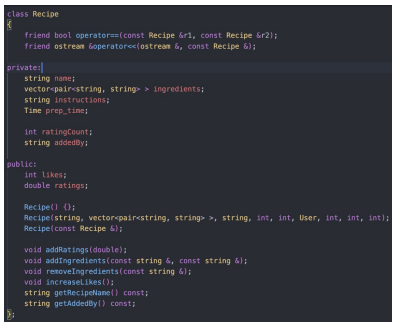
The primary objective of this project is to create a user-friendly Recipe Management Application that allows users to store, manage, and retrieve recipe information efficiently. This application leverages OOP concepts such as encapsulation, abstraction, classes, and composition to ensure modular and maintainable code.

1.2 Features Implemented

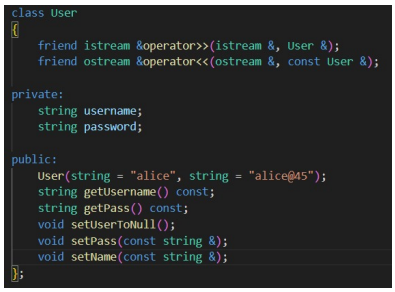
In this project, the following OOP features and techniques have been applied:

- **Encapsulation:** Encapsulation involves bundling data (attributes) and functions(behaviours) within a class, ensuring that only the necessary details are accessible to the outside. Each class represents a real-world entity, such as a recipe or a user, and controls access to its data to maintain integrity and modularity.
- **Abstraction:** Abstraction is the concept of exposing only essential attributes and behaviors of an object while hiding its internal implementation details. By keeping data members private and providing access through public member functions, abstraction simplifies interaction with the class and helps protect data integrity.
- **Classes and Composition:** Classes are defined to represent various entities, such as `Recipe`, `User`, and `Time`. Composition is used to show a has-a relationship.
- **Function Templates:** To improve flexibility, function templates are implemented for sorting/-filtering recipes based on various attributes (such as likes and ratings). These templates enable the use of generic code for different data types and operations, enhancing code reusability.
- **File Handling Implementation:** File handling is a crucial feature implemented in this project to allow users to save and load recipe data. By storing data in text files, the application provides persistent storage for recipes, users, and other relevant information.
- **User Authentication and Authorization:** User authentication and authorization are implemented to secure the Recipe Management App. Users must log in with a valid username and password to access their accounts and manage recipes.

2 Encapsulation-Abstraction



(a) Recipe Class



(b) User Class

(c) Time Class

Figure 1: Interface of Recipe, User, and Time Classes

The `Recipe` class demonstrates **encapsulation** by making key data members private, allowing access only through public **member functions** like `addRatings`, `addIngredients`, and `getRecipeName`. This ensures controlled data access and modification.

It also illustrates **abstraction** by exposing essential behaviors, such as managing ingredients and ratings, while hiding `internal implementation details`. This clean interface simplifies use, enabling interaction without needing to know the internal structure.

The `User` class demonstrates **encapsulation** by keeping sensitive data members, like `username` and `password`, private. Access to these members is controlled through public member functions such as `getUsername`, `getPass`, `setPass`, and `setName`, which allow safe and regulated interaction with the userâs information.

The class also provides a clean and simple interface through **abstraction**, exposing essential functionalities (e.g., setting and retrieving username and password) without revealing internal data handling, thus ensuring security and simplicity for users of the class.

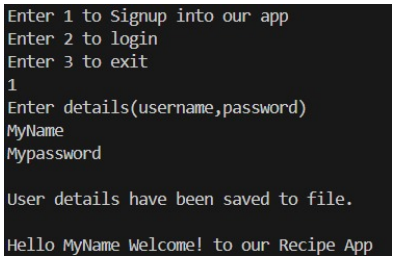
The `Time` class demonstrates **encapsulation** by keeping its data members, such as `hour`, `minute`, and `second`, private and accessible only through public methods.

In this context, the `Time` class is used to implement **class composition**. Specifically, an object of the `Time` class is included as a data member of the `Recipe` class to represent the preparation time.

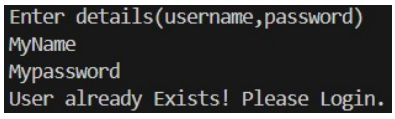
3 User Authentication/Authorization

This part of project implements a basic **user authentication system** with functionalities for **signup, login, and duplicate account prevention**. Below are the key features:

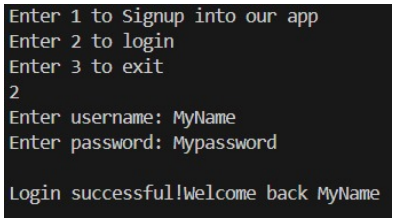
1. **Signup**:
   - User selects the **Signup** option and enters a **username** and **password**.
   - Credentials are saved to a file.
   - A welcome message confirms a successful signup.
2. **Login**:
   - User selects the **Login** option and inputs their credentials.
   - If the details match the saved data, the system displays "Login successful!".
3. **Duplicate Check**:
   - If the user tries to sign up with an existing username, the system prevents duplicate accounts.
   - Displays a message: "User already exists! Please Login."



(a) Signup



(b) Duplicate Signup



(c) Login

Figure 2: Outputs of Authentication

4 Definitions(Authentication)

```cpp
void signUp(User &u)
{
    cout << "Enter details(username,password)" << endl;
    cin >> u;
    ofstream out("user.txt", ios::app);
    if (login(u.getUsername(), u.getPass()))
    {
        cout << "User already Exists! Please Login." << endl;
        u.setUserToNull();
        return;
    }
    if (!out)
    {
        cerr << "Error while opening file for saving user data." << endl;
        return;
    }
    out << u << endl;
    out.close();
    cout << endl;
    cout << "User details have been saved to file." << endl;
    cout << endl;
}
```

(a) Signup Function

```cpp
int handleLogin(User &u)
{
    string name, password;
    cout << "Enter username: ";
    cin >> name;
    cout << "Enter password: ";
    cin >> password;
    cout << endl;

    if (login(name, password))
    {
        cout << "Login successful!" << "Welcome back " << name << endl
             << endl;
        ;
        u.setName(name);
        u.setPass(password);
        return 1;
    }
    else
    {
        cout << "Invalid username or password." << endl
             << endl;
        return 0;
    }
}
```

(b) Handle login

```cpp
int handleSignup(User &u)
{
    signUp(u);
    if (u.getUsername() == " ")
    {
        return 1;
    }
    cout << "Hello " << u.getUsername() << " Welcome! to our Recipe App" << endl;
    return 0;
}
```

(c) Handle Signup

```cpp
// login function
bool login(const string &username, const string &password)
{
    ifstream in("user.txt");
    if (!in)
    {
        cerr << "Error opening file for reading." << endl;
        return false;
    }
    User u;
    while (in >> u)
    {
        if (u.getUsername() == username && u.getPass() == password)
        {
            return true;
        }
    }
    return false;
}
```

(d) Handle Login

Figure 3: Handlers Functions

5 Function Templates

Function Templates is used as generic programming practice.we cam write a generic function that can be used for different data types. Examples of function templates are sort(), max(), min(), printArray().

1. **Template Flexibility**: The `bubbleSortRecipes` template function sorts `Recipe` objects by any specified member (e.g., `likes` or `ratings`) where likes is of int type while ratings is of double data type.
2. **Bubble Sort Optimization**: The use of a `swapped` flag in the bubble sort stops sorting early if the list is already sorted, improving efficiency.
3. **Descending Order Sorting**: The code correctly sorts recipes in descending order for both `likes` and `ratings`, as shown in the output.
4. **User Prompt**: The program lets the user choose the sorting criterion, enhancing usability.

```
Enter 1 to filter based on likes(decreasing order)
Enter 2 to filter based on ratings(decreasing order)
2
Sorted by ratings:
Grilled Cheese Sandwich : 5
Pancakes : 4
Overnight Oats : 3
```

(a) Sorting by ratings

```
Enter 1 to filter based on likes(decreasing order)
Enter 2 to filter based on ratings(decreasing order)
1
Sorted by likes:
Grilled Cheese Sandwich : 10
Overnight Oats : 7
Pancakes : 5
```

(b) Sorting by likes

```
template <typename T>
void bubbleSortRecipes(vector<Recipe> &recipes, T Recipe::*member)
{
    bool swapped;
    for (int i = 0; i < recipes.size() - 1; ++i)
    {
        swapped = false;
        for (int j = 0; j < recipes.size() - i - 1; ++j)
        {
            if (recipes[j].*member < recipes[j + 1].*member)
            {
                swap(recipes[j], recipes[j + 1]);
                swapped = true;
            }
        }
        if (!swapped)
            break;
    }
}
```

(c) Code

Figure 4: Function Templates

6 Constructors Definitions

```cpp
Time::Time(int hour, int minutes, int seconds)
{
    if ((hour < 0 || hour > 23) || (minutes < 0 || minutes > 59) || (seconds < 0 || seconds > 59))
    {
        this->hour = 0;
        this->minutes = 30;
        this->seconds = 0;
    }
    else
    {
        this->hour = hour;
        this->minutes = minutes;
        this->seconds = seconds;
    }
}
```

(a) `Time` Constructor

```cpp
// Constructor definition
Recipe::Recipe(string name, vector<pair<string, string>> ingredients, string instructions,
               int likes, int ratings, User user,
               int hour, int minutes, int seconds)
    : prep_time(hour, minutes, seconds)
{
    this->name = name;
    this->ingredients = ingredients;
    this->instructions = instructions;
    this->likes = likes;
    this->ratings = ratings;
    this->addedBy = user.getUsername();
    ratingCount = 1;
}
```

(b) `Recipe` Constructor

```cpp
Recipe::Recipe(const Recipe &other)
{
    name = other.name;
    ingredients = other.ingredients;
    instructions = other.instructions;
    prep_time = other.prep_time;
    likes = other.likes;
    ratings = other.ratings;
    ratingCount = other.ratingCount;
    addedBy = other.addedBy;
}
```

(c) `Recipe` Copy Constructor

```cpp
User::User(string username, string pass)
{
    this->username = username;
    this->password = pass;
}
```

(d) `User` Constructor

Figure 5: Constructors Overloading

7 Operator and Function Overloading

Operator Overloading is the process of giving special meaning to an existing operator in the context of user-defined data types.

Function Overloading is defined as when two or more functions have the same name but differ in their signature (type, number of parameters).

1. **Overloaded < < Operator for Output**:
   - This operator is used to display recipe details in a well-formatted way.
2. **Overloaded > > Operator for Input**:
   - This operator is used to take inputs of data members of objects in one go.
3. **Overloaded == Operator for Comparing Recipes**:
   - This operator checks for common ingredients between two recipes.
   - When you use `recipe1 == recipe2;`, it:
     - Prints out the ingredients that both recipes share.
     - Returns `true` if there are common ingredients, otherwise `false`.
4. **Function Overloading for Recipe**:
   - The `Recipe` constructor is overloaded where the parameterized constructor and copy constructor have been used.

```
Recipe() {};
Recipe(string, vector<pair<string, string>>, string, int, int, User, int, int, int);
Recipe(const Recipe &);
```

(a) Constructor Overloading Example

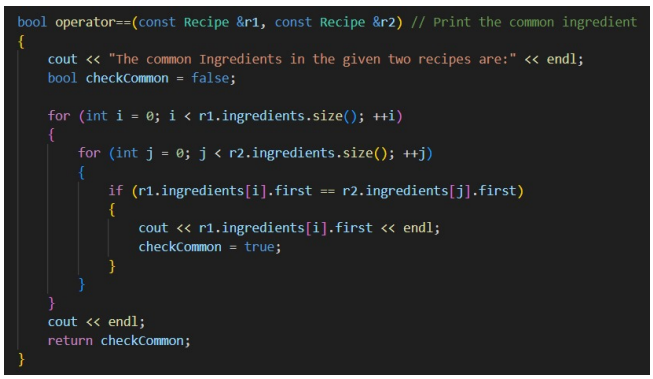
Figure 6: Function Overloading in Recipe Class

8 Overloading Definitions

(a) < < Operator

(b) < < Operator

(c) > > Operator



(d) == Operator

Figure 7: Operator Overloading

9 Implemented Features in App

In the following page, we describe two functions that allow users to modify the ingredients of a recipe: addIngredients and removeIngredients. These functions allow the user to add or remove ingredients in the recipes they have created.

addIngredients Function

```
void Recipe::addIngredients(const string &takeIngredient, const string &quantity)
{
    ingredients.push_back(make_pair(takeIngredient, quantity));
}
```

Figure 8: addIngredients Function

removeIngredients Function

```
void Recipe::removeIngredients(const string &removeIngredients)
{
    for (int i = 0; i < ingredients.size(); i++)
    {
        if (ingredients[i].first == removeIngredients)
        {
            ingredients.erase(ingredients.begin() + i);
            break;
        }
    }
}
```

Figure 9: removeIngredients Function

Conditions

Both functions only apply to recipes created by the currently logged-in user. This ensures that users can only modify the ingredients of their own recipes and prevents unauthorized changes to other users' recipes.

10 Implemented Features Contd..

In the following page, we describe functions like `increaseLikes` and `addRatings`.

IncreaseLikes

The `increaseLikes()` method adds **1 to the total likes**, indicating the recipe's popularity.

```
void Recipe::increaseLikes() /
{
    likes++;
}
```

Figure 10: Increaselikes function

addRatings

The `addRatings(double addedRating)` method manages ratings by:

- **Counting Ratings**: Each new rating increases the `ratingCount` by 1.
- **Updating Average Rating**: Uses this formula to recalculate the average each time:

$$\text{ratings} = \frac{(\text{ratings} \times \text{ratingCount} + \text{addedRating})}{(\text{ratingCount} + 1)}$$

```
void Recipe::addRatings(double addedRating)
{
    ratingCount++;
    ratings = (ratings * ratingCount + addedRating) / (ratingCount + 1);
}
```

Figure 11: addRatings Function

11 Implemented Features Contd..

11.1 AddRecipe

1. **Create Recipe**:
   - A new `Recipe` object is created using user input (name, ingredients, instructions, etc.).
2. **Open File**:
   - Opens a file named `addedRecipe.txt` in append mode using `ofstream`.
   - If the file fails to open, it shows an error message and exits.
3. **Check for Duplicates**:
   - Uses the function `isRecipeAdded()` to check if a recipe with the same name already exists.
   - If a duplicate is found, it informs the user and skips saving the recipe.
4. **Save Recipe**:
   - **To File**: Adds the new recipe to `addedRecipe.txt`.
   - **To Vector**: Adds the recipe to a vector called `recipes` for fast access.
5. **Close File**:
   - Ensures the file is properly closed after writing to prevent data corruption.

```
Recipe r1(name, ingredients, instruct, 0, 0, u, h, m, s);

ofstream out("addedRecipe.txt", ios::app);
if (!out)
{
    cerr << "Error opening file for writing." << endl;
    return;
}
if (isRecipeAdded(r1.getRecipeName()))
{
    cout << "This Recipe already exist.please have a look at all recipes" << endl;
    continue;
}

out << r1;
recipes.push_back(r1);
out.close();
```

Figure 12: addRecipe function

12 Implemented Features Contd..

12.1 isRecipeAdded

- The function `isRecipeAdded()` is used to check if a recipe with the same name already exists.
- If a duplicate is found, it informs the user and skips saving the recipe.

```
Recipe r1(name, ingredients, instruct, 0, 0, u, h, m, s);

ofstream out("addedRecipe.txt", ios::app);
if (!out)
{
    cerr << "Error opening file for writing." << endl;
    return;
}
if (isRecipeAdded(r1.getRecipeName()))
{
    cout << "This Recipe already exist.please have a look at all recipes" << endl;
    continue;
}

out << r1;
recipes.push_back(r1);
out.close();
```

Figure 13: `isRecipeAdded` Function

12.2 Recipe of the Day

- The program checks if the recipe list is empty; if not, it seeds the random number generator, selects a random recipe, and displays it as the **"Recipe of the Day"**.

```
case 6: // "Recipe of the Day"
{
    if (recipes.empty())
    {
        cout << "No recipes available!" << endl;
        break;
    }

    srand(time(0));                          // Seed the random number generator
    int randomIndex = rand() % recipes.size(); // Generate a random index

    cout << "Recipe of the Day: " << endl;
    cout << recipes[randomIndex] << endl; // Display the randomly selected recipe
    break;
}
```

Figure 14: Recipe of the Day

13 End-sem Version

13.1 Inheritance and Polymorphism

```cpp
//Abstract class
class Recipe
{
    friend bool operator==(const Recipe&, const Recipe&);
protected:
    string name;
    vector<pair<string, string> > ingredients;
    string instructions;
    Time prep_time;

    int ratingCount;
    string addedBy;

public:
    int likes;
    double ratings;
    void addRatings(double);
    void addIngredients(const string &, const string &);
    void removeIngredients(const string &);
    void increaseLikes();
    string getRecipeName() const;
    string getAddedBy() const;
    virtual void display() const = 0;  //pure virtual function
    virtual ~Recipe()
    {
        cout << "Recipe destructor called." << endl;
```

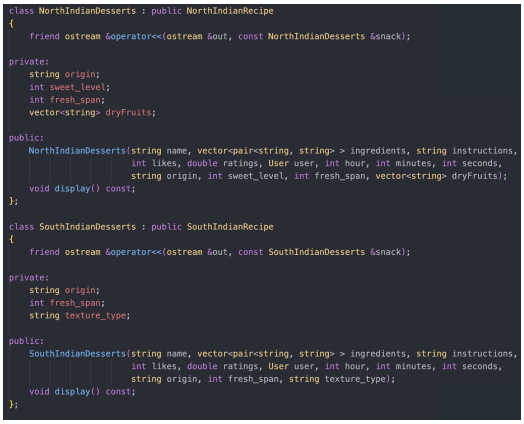
Figure 15: `Recipe-Abstract` Class

```cpp
// Derived Class: NorthIndianRecipe(Abstract class)
class NorthIndianRecipe : public Recipe
{

public:
    NorthIndianRecipe(string name, vector<pair<string, string> > ingredients, string instructions, int likes, dou

    virtual void display() const = 0;
};

// Derived Class: SouthIndianRecipe(Abstract class)
class SouthIndianRecipe : public Recipe
{

public:
    SouthIndianRecipe(string name, vector<pair<string, string> > ingredients, string instructions, int likes, dou

    virtual void display() const = 0;
};
```

Figure 16: `Derived Abstract` Classes

```
//concrete classes
class NorthIndianSnacks : public NorthIndianRecipe
{
    friend ostream &operator<<(ostream &out, const NorthIndianSnacks &snack);

private:
    string origin;
    string prep_type;
    int spice_level;
    string beverage_pairing;

public:
    NorthIndianSnacks(string name, vector<pair<string, string> > ingredients, string instructions,
                      int likes, double ratings, User user, int hour, int minutes, int seconds,
                      string origin, string prep_type, int spice_level, string beverage_pairing);
    void display() const;
};

class SouthIndianSnacks : public SouthIndianRecipe
{
    friend ostream &operator<<(ostream &out, const SouthIndianSnacks &snack);

private:
    string origin;
    string prep_type;
    int crisp_level;
    string chutney_type;

public:
    SouthIndianSnacks(string name, vector<pair<string, string> > ingredients, string instructions,
                      int likes, double ratings, User user, int hour, int minutes, int seconds,
                      string origin, string prep_type, int crisp_level, string chutney_type);
    void display() const;
};
```

Figure 17: `Derived Concrete` Classes



```
class NorthIndianDesserts : public NorthIndianRecipe
{
    friend ostream &operator<<(ostream &out, const NorthIndianDesserts &snack);

private:
    string origin;
    int sweet_level;
    int fresh_span;
    vector<string> dryFruits;

public:
    NorthIndianDesserts(string name, vector<pair<string, string> > ingredients, string instructions,
                        int likes, double ratings, User user, int hour, int minutes, int seconds,
                        string origin, int sweet_level, int fresh_span, vector<string> dryFruits);
    void display() const;
};

class SouthIndianDesserts : public SouthIndianRecipe
{
    friend ostream &operator<<(ostream &out, const SouthIndianDesserts &snack);

private:
    string origin;
    int fresh_span;
    string texture_type;

public:
    SouthIndianDesserts(string name, vector<pair<string, string> > ingredients, string instructions,
                        int likes, double ratings, User user, int hour, int minutes, int seconds,
                        string origin, int fresh_span, string texture_type);
    void display() const;
};
```

Figure 18: `Derived Concrete` Classes

14 Exception Handling

```cpp
// Custom Signup exception handler
class SignUpException {
private:
    string message;

public:
    // Constructor to initialize the error message
    SignUpException(const string &msg) : message(msg) {}

    // Method to retrieve the error message
    const char *what() const
    {
        return message.c_str();
    }
};

//login exception handler
class LoginException {
private:
    string message;

public:
    // Constructor to initialize error message
    LoginException(const string &msg) : message(msg) {}

    // Method to retrieve the error message
    const char *what() const {
        return message.c_str();
    }
};
```

Exception Class

```cpp
void signUp(User &u)
{
    try
    {
        cout << "Enter details (username, password):" << endl;
        cin >> u;

        // Check if the user already exists
        if (login(u.getUsername(), u.getPass()))
        {
            u.setUserToNull();
            throw SignUpException("User already exists! Please login.");
        }

        // Open file in append mode
        ofstream out("user.txt", ios::app);
        if (!out)
        {
            throw SignUpException("Error while opening the file for saving user data.");
        }

        // Save user details to file
        out << u << endl;
        out.close();

        cout << "\nUser details have been successfully saved to the file.\n" << endl;
    }
    catch (const SignUpException &e)
    {
        cerr << "SignUp Error: " << e.what() << endl;
    }
    catch (...)
    {
        cerr << "An unknown error occurred." << endl;
    }
```

Signup Exception implementation

Figure 19: Custom Exception Handler

15 Polymorphism-Implementation

```cpp
//array of abstract base class pointers
extern vector<Recipe *> recipes;
```

Figure 20: Base Pointer

```
vector<pair<string, string> > jalebiIngredients;
jalebiIngredients.push_back(make_pair("Flour", "2 cups"));
jalebiIngredients.push_back(make_pair("Sugar", "3 cups"));
jalebiIngredients.push_back(make_pair("Saffron", "Pinch"));
jalebiIngredients.push_back(make_pair("Cardamom", "1 tsp"));
jalebiIngredients.push_back(make_pair("Ghee", "500ml"));
jalebiIngredients.push_back(make_pair("Water", "2 cups"));
vector<string> dryFruits;
dryFruits.push_back("Pistachios");
NorthIndianDesserts* recipe5 = new NorthIndianDesserts("Jalebi", jalebiIngredients, "Make batter, fry spirals, and soak in syrup.", 400
recipes.push_back(recipe5);
```

Figure 21: `Recipes object Addition` in Base pointer array

```
Recipe *newRecipe = nullptr;
switch (typeChoice)
{
case 1:
{
    string origin, prepType, beveragePairing;
    int spiceLevel;
    cout << "Enter origin: ";
    cin.ignore();
    getline(cin, origin);
    cout << "Enter preparation type: ";
    getline(cin, prepType);
    cout << "Enter spice level (1-5): ";
    cin >> spiceLevel;
    cout << "Enter beverage pairing: ";
    cin.ignore();
    getline(cin, beveragePairing);

    newRecipe = new NorthIndianSnacks(name, ingredients, instructions, 0, 0, u, h, m, s, origin, prepType, spiceLevel, beveragePairing);
    break;
}
```

Figure 22: `Recipes object Addition` in Base pointer array

```
case 2:
{
    cout << "Recipes available:" << endl;
    for (const auto &recipe : recipes)
    {
        recipe->display();
        cout << endl;
    }
    break;
}
```

Figure 23: `Recipes Display` Using Base pointer (virtual functions)

```
case 4:
{
    cout << "Enter the name of the 1st recipe: ";
    string name1;
    cin.ignore();
    getline(cin, name1);

    cout << "Enter the name of the 2nd recipe: ";
    string name2;
    getline(cin, name2);

    Recipe *recipe1 = nullptr, *recipe2 = nullptr;

    for (const auto &recipe : recipes)
    {
        if (recipe->getRecipeName() == name1)
        {
            recipe1 = recipe;
        }
        if (recipe->getRecipeName() == name2)
        {
            recipe2 = recipe;
        }
    }

    if (!recipe1 || !recipe2)
    {
        cout << "One or both recipes not found!" << endl;
        break;
    }
    if(*recipe1==*recipe2)
    {
```

Figure 24: `Common Ingredients Check` Using polymorphism

```cpp
void SouthIndianDesserts::display() const
{
    cout << "============================" << endl;
    cout << "       Recipe Details       " << endl;
    cout << "============================" << endl;
    cout << "🍮 Dessert Name: " << name << endl;
    cout << "----------------------------" << endl;

    cout << "🔪 Ingredients:" << endl;
    cout << "----------------------------" << endl;
    for (const auto &ingredient : ingredients)
    {
        cout << ingredient.first << "\t\t" << ingredient.second << endl;
    }
    cout << "----------------------------" << endl;

    cout << "📖 Preparation Time:" << endl;
    cout << prep_time << endl;

    cout << "📖 Cooking Instructions:" << endl;
    cout << instructions << endl;

    cout << "👍 Likes: " << likes << endl;
    cout << "⭐ Average Rating: " << ratings << endl;

    cout << "👤 Added By: " << addedBy << endl;

    cout << "🌍 Origin: " << origin << endl;
    cout << "📅 Freshness Span: " << fresh_span << " days" << endl;
    cout << "🌟 Texture Type: " << texture_type << endl;
    cout << "============================" << endl;
}
```

Figure 25: `Overridden Display Function` Definition

16 Test drive(Outputs)

```
Enter 1 to Signup into our app
Enter 2 to login
Enter 3 to exit
1
Enter details (username, password):
user12
123

User details have been successfully saved to the file.

Hello user12 Welcome! to our Recipe App
Enter 1 to add your Recipe
Enter 2 to display all recipes
Enter 3 to search and display a particular recipe
Enter 4 to display common ingredients between two recipes
Enter 5 to sort/filter recipes based on likes/ratings
Enter 6 to get the recipe of the day
Enter 7 to exit
```

Figure 26: `Signup`

```
Enter 1 to Signup into our app
Enter 2 to login
Enter 3 to exit
2
Enter username: user12
Enter password: 123

Login successful!Welcome back user12

Enter 1 to add your Recipe
Enter 2 to display all recipes
Enter 3 to search and display a particular recipe
Enter 4 to display common ingredients between two recipes
Enter 5 to sort/filter recipes based on likes/ratings
Enter 6 to get the recipe of the day
Enter 7 to exit
```

Figure 27: `Login`

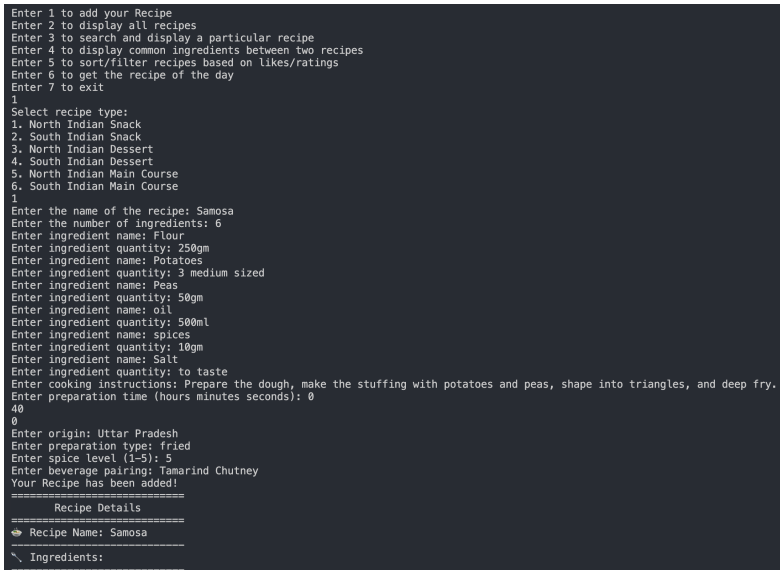


Figure 28: `Recipe Add`

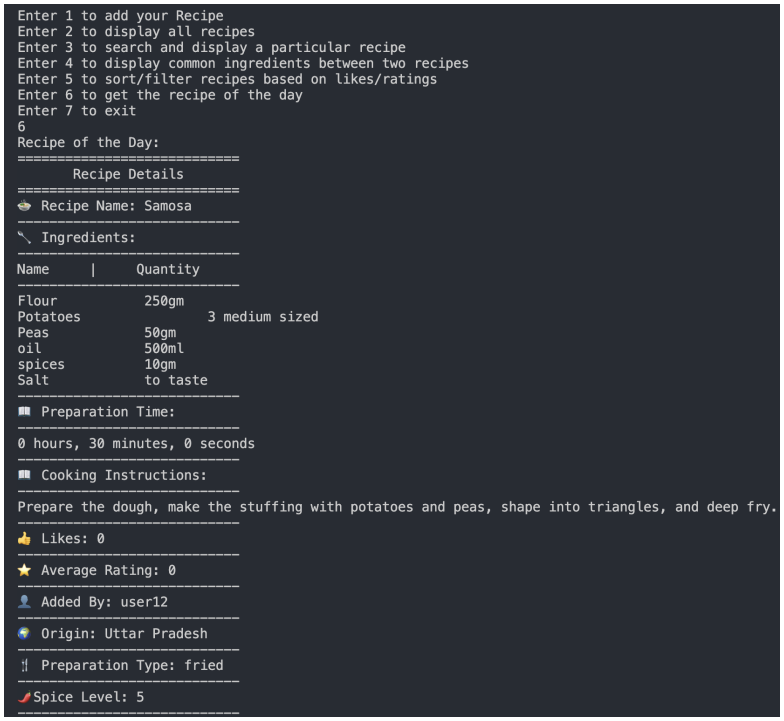


Figure 29: `Recipe of the Day`

```
Enter 1 to add your Recipe
Enter 2 to display all recipes
Enter 3 to search and display a particular recipe
Enter 4 to display common ingredients between two recipes
Enter 5 to sort/filter recipes based on likes/ratings
Enter 6 to get the recipe of the day
Enter 7 to exit
4
Enter the name of the 1st recipe: Samosa
Enter the name of the 2nd recipe: Kachori
The common Ingredients in the given two recipes are:
Flour
Salt
```

Figure 30: `Common Ingredients`

```
Enter the name of the recipe to be displayed:
Samosa
==========================
       Recipe Details
==========================
🍲 Recipe Name: Samosa
--------------------------
🔪 Ingredients:
--------------------------
Name      |     Quantity
--------------------------
Flour          250gm
Potatoes               3 medium sized
Peas           50gm
oil            500ml
spices         10gm
Salt           to taste
--------------------------
📖 Preparation Time:
--------------------------
0 hours, 30 minutes, 0 seconds
--------------------------
📖 Cooking Instructions:
--------------------------
Prepare the dough, make the stuffing with potatoes and peas, shape into triangles, and deep fry.
--------------------------
👍 Likes: 0
--------------------------
⭐ Average Rating: 0
--------------------------
👤 Added By: user12
--------------------------
🌍 Origin: Uttar Pradesh
--------------------------
🍴 Preparation Type: fried
--------------------------
🌶️Spice Level: 5
--------------------------
🍷 Beverage Pairing: Tamarind Chutney
==========================
```

Figure 31: `Display Particular Recipe`

```
Enter 1 to give a rating to this recipe.
Enter 2 to like this recipe.
Enter 3 to add more ingredients to this recipe.
Enter 4 to remove ingredients from this recipe.
Enter 5 to exit to the main menu.
4
Enter the name of ingredient to be removed:
Salt
Ingredient removed.
============================
       Recipe Details
============================
🍲 Recipe Name: Kachori
----------------------------
🥄 Ingredients:
----------------------------
Name     |     Quantity
----------------------------
Flour           200g
Lentils         100g
Oil             500ml
Ginger          1 tsp
Spices          10g
----------------------------
📖 Preparation Time:
----------------------------
0 hours, 30 minutes, 0 seconds
----------------------------
📖 Cooking Instructions:
----------------------------
Make dough, fill lentils, and deep fry.
----------------------------
👍 Likes: 250
----------------------------
⭐ Average Rating: 4.7
----------------------------
👤 Added By: gyan
----------------------------
🌍 Origin: Rajasthan
----------------------------
🍴 Preparation Type: Fried
```

Figure 32: `Display Particular Recipe`