



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

Semester II 2023/2024

Subject : Programming Technique 2 - SECJ1023

Section : 08

Task : Final report

Due : 26 june2024

Lecturer : DR LIZAWATI BINTI MI YUSUF

Group : 11

	Name	Matric Number
1	RAMI YASSEIN ELTAYEB MOHAMED	A23CS0022
2	Ammar Abdulrahman Anaam Mudhsh	A23CS0287
3	Mohamed Ali Mohamed Ali	A21EC0287

Table of Contents

1. Section A(proposal):.....	
1.1 Synopsis and general idea.....	1
1.2 Objective and Purpose	1
1.3 System Features User guide	3-4
1.4 Reporting	5
1.5 Flowchart.....	6-8
1.6 UML	9
1.7 Objects.....	10
1.8 Proposed classes involved.....	11-14
1.9 Classes relationship.....	15
2. Section B(Updated).....	
2.1 OOB implementation	16-21
2.2 Conclusion.....	22

1.0 Section A:

1.1 Synopsis and General idea:-

Foodi is a cutting-edge food allergy management system specifically designed for individuals with food sensitivities and allergies. It aims to empower users by providing them with essential tools to identify and avoid foods that may trigger allergic reactions, ensuring their safety and well-being. By leveraging a comprehensive pre-set database of food information, Foodi enables users to make informed dietary choices and prevent potentially life-threatening allergic reactions.

1.2 Objective and Purpose:-

- To Help users Avoid Any food that may triggers their Allergies.
- To Help keeping people safe because some Allergies can lead to death.
- To provide a fast and easy to use Tool that gives specific and concise information about Allergies based on each Unique case.

1.3 System Features and User Guide

Login Page: Upon initial access, users will be prompted to enter essential personal information such as their name, email, and age range. They will also select their allergens from a dropdown list to personalize their profile.

Home Screen: Once logged in, users will navigate to the home screen, which features three primary options:

1. Search Bar:
 - Users can type in the name of a specific food item to search for it. The system will then highlight any allergens present in the food's ingredients list, if applicable.
2. List of Food Items:
 - Users can manually browse through a comprehensive list of food items categorized by type (e.g., dairy products). This is especially useful if the food item cannot be found using the search function.
3. Allergy Information:
 - This section displays the user's personalized allergy profile for quick and easy reference, ensuring that critical information is readily accessible.

Settings:

- Profile:
 - Users can update their personal information by entering new values in the respective input fields.
 - Users can also manage their allergens by adding or removing items from their list.

Reporting:

- Output Based on Selected Allergen:
 - Displays lists of food items that contain, are free of, or are highly likely to contain the specified allergens.
- Output Based on Searched Food:
 - Provides immediate feedback on whether the searched food is safe for the user.
 - Lists which allergens, if any, are present in the searched food item.
- Settings Output:
 - Displays user information and the list of registered allergies for verification and updates.

1.4 Reporting

Output based on selected Allergen:

- Food that contains allergens.
- Food that is free of allergens.
- highly likely to contain allergens

Output based on searched food:

- Will tell the user if the food is safe or not.
- Will tell the user which allergens are present in the searched food.

Settings output:

- User information.
- Registered allergies.

1.5 Proposed Flowchart :-

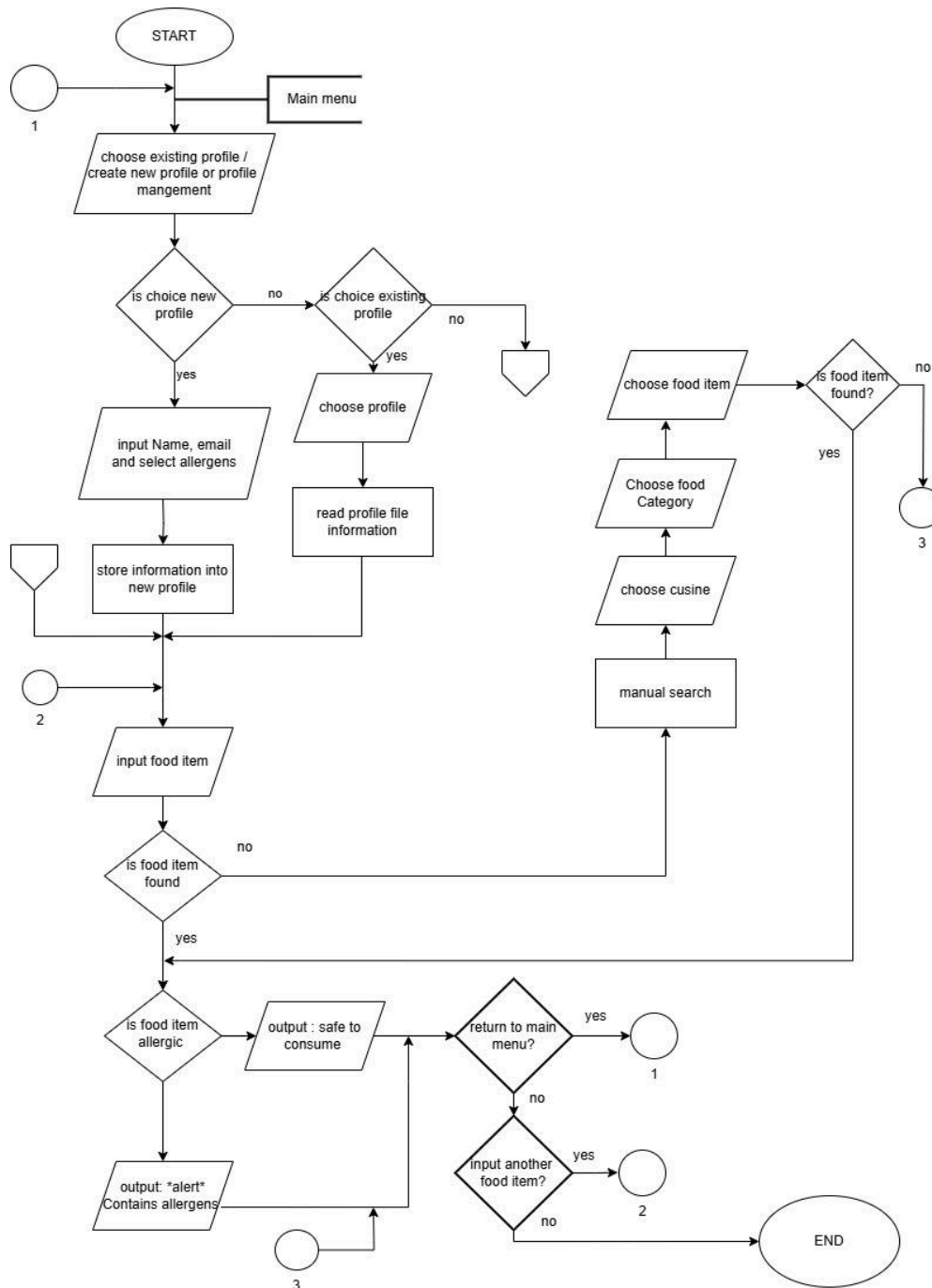


Figure 1

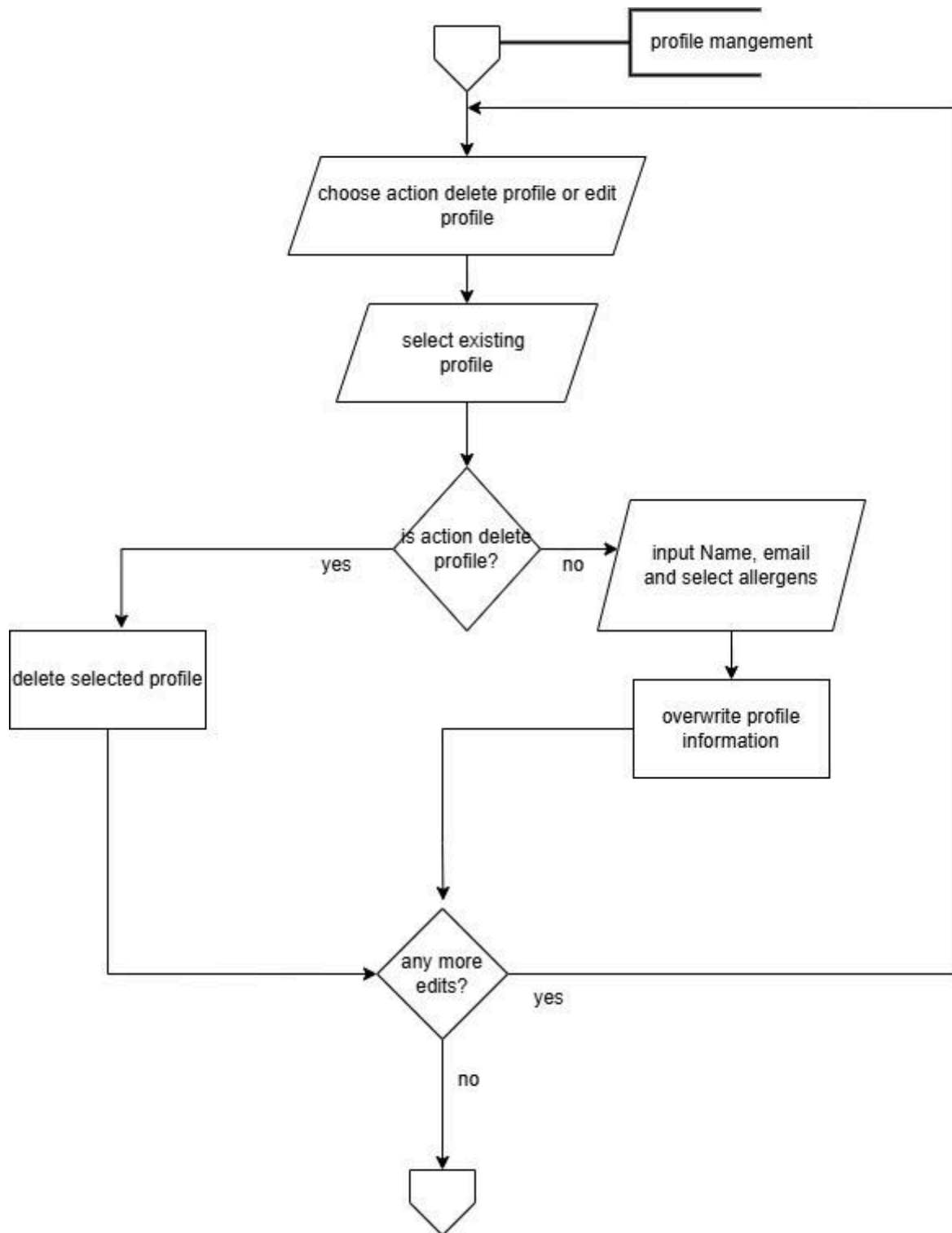


Figure 2

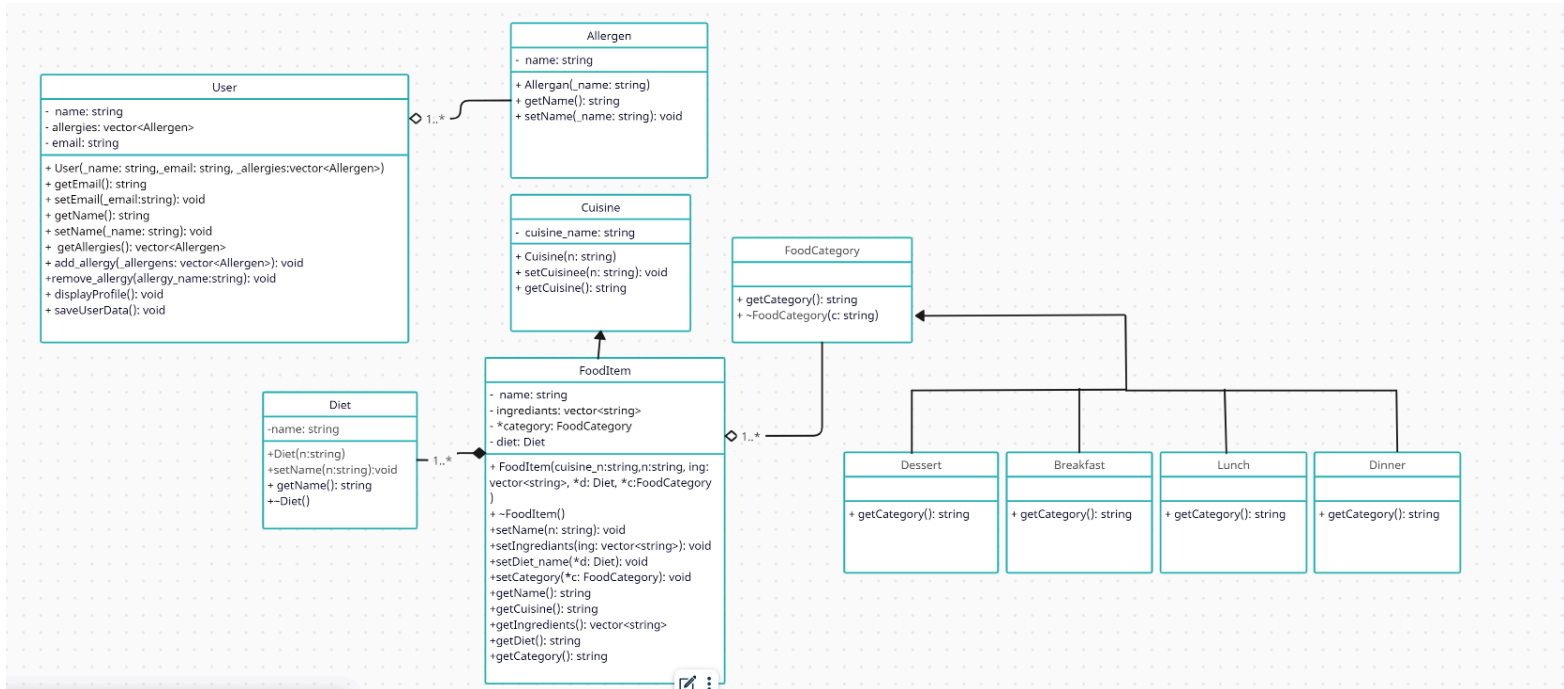
About the flowchart:-

The main process of the system begins with the Main Menu, where users can choose to manage their profiles or search for food items. Users can create a new profile by providing personal details such as name, email, age range, and selecting allergens from a pre-populated list. If an existing profile is chosen, the system retrieves the stored information for the user to review or update.

Once a profile is set up, users can perform food searches. They can either use the Search Bar to type in specific food items or browse through categorized lists of foods. The system checks the ingredients of the selected food against the user's allergy profile to determine if it is safe to consume. If the food contains allergens, the system alerts the user, specifying the allergens found. If no allergens are detected, the system confirms the food is safe.

The system also provides detailed reporting based on search results, informing users whether a food is "Safe to Consume" or "Contains Allergens" along with specifics. Additionally, users can search by allergen to find foods that contain or are free from the allergen and those that may pose a cross-contamination risk. Throughout the process, users can update their profiles to ensure accurate and personalized allergy checks, continually enhancing their experience and safety.

1.6 UML:-



1.7 Objects type(classes):

1. **User:** Represents a user of the system with attributes like name and allergies.
2. **Allergen:** Represents a specific allergen (e.g., peanuts, dairy) with an attribute like name.
3. **Cuisine:** Represents a culinary tradition (e.g., Italian, Thai) with an attribute like cuisine name (protected for access by subclass).
4. **FoodCategory:** Represents a food category (breakfast, lunch, dinner, or snack) with an attribute like category.
5. **Diet:** Represents a dietary restriction (e.g., vegan, vegetarian) with an attribute like name.
6. **FoodItem:** Represents a specific food item with attributes like name, ingredients, category (FoodCategory object), and diet (Diet object). It inherits cuisine_name from the Cuisine class.
7. **Dessert:** type of food category that has polymorphism relationship with Foodcategory class
8. **Breakfast:** type of food category that has polymorphism relationship with Foodcategory class
9. **Lunch :** type of food category that has polymorphism relationship with Foodcategory class
10. **Dinner:** type of food category that has polymorphism relationship with Foodcategory class

1.8 Proposed Classes Involved:

1. User:

○ Attributes:

- name (string)
- allergies (vector <Allergen>)
- Email (string)

All member attributes are private.

○ Methods:

- User(string _name= "", string _email, vector<Allergen> _allergies) is a both a default constructor and parameter constructor
- string getEmail() is an accessor of email
- void setEmail(string _email) is mutator of email
- string getName() is an accessor of name
- void setName(string _name) is mutator of name
- vector<Allergen> getAllergies() is an accessor of allergies
- void add_allergy(vector<Allergen> _allergens) adds a new Allergy object to the allergy list.
- void remove_allergy(string allergy_name) removes an Allergy object with the specified name.
- void displayProfile() shows what the class contain
- void saveUserData() writes user data to a local file

2. Cuisine:

○ Attributes:

- cuisine_name (string)

Member function is protected, then it is derived class FoodItem can access the member attribute (cuisine_name).

○ Methods:

- Cuisine(string n = "") is both a default constructor and parameter constructor
- § string getCuisine() is an accessor of name
- § void setName(string n) is a mutator

3. FoodCategory:

- **Attributes:**

- null

Attribute category is private.

- **Methods:**

- § String getCategory() is an accessor of category.

- § void setCategory(string c) is a mutator of category.

4. Allergen:

- **Attributes:**

- name (string)

Attribute name is private.

- **Methods:**

- Allergen(string _name = "") is both a default constructor and parameter constructor.

- string getName() is an accessor of name

- void setName(string _name) is a mutator of name

Diet:

- **Attributes:**

- name (string)

Attribute name is private.

- **Methods:**

- Diet(string n = "") is both a default constructor and parameter constructor.

- string getName() is an accessor of name

- void setName(string n) is a mutator of name

- ~Diet() is destructor

6. FoodItem (Class inheriting from Cuisine):

- **Attributes:** (Inherits cuisine_name from Cuisine)

- § name (string)

- ingredients (vector<string>)
 - * category (FoodCategory)
 - diet (Diet)

All member attributes are private.

- **Methods:**

- FoodItem(string cuisine_n= "", string n= "", vector< string> ing = {}, string diet_n = "", FoodCategory *c = nullptr) is both a default constructor and parameter constructor.
 - ~FoodItem() is a destructor.
 - string getName() is an accessor of name.
 - vector<string> getIngredients() is an accessor of ingredients
 - string getDiet() is an accessor of diet
 - string getCategory() is an accessor of category
 - void setCategory(FoodCategory*c) is a mutator of category
 - void setName(string n) is a mutator of name
 - void setIngredients(vector<string> ing) is a mutator of ingredients
 - void setDiet(string d_n) is a mutator of diet name

7. Dessert:

- **Attributes:**

- null

Attribute name is private.

- **Methods:**

- String getCategory()

8. Breakfast:

- **Attributes:**

 null

Attribute name is private.

- **Methods:**

- String getCategory()

9. Lunch:

- **Attributes:**

 null

Attribute name is private.

- **Methods:**

- String getCategory()

10. Dinner:

- **Attributes:**

 null

Attribute name is private.

- **Methods:**

- String getCategory()

1.9 Class Relationships:

- **Inheritance:**

- FoodItem inherits from Cuisine. This establishes a parent-child relationship where each FoodItem is a type of Cuisine, this means that FoodItem will inherit cuisine_name attribute from Cuisine making it easier to split FoodItem into categories where the user will find it simpler to find FoodItem normally.

- **Aggregation:**

- User and Allergy: This is an aggregation relationship, where User has a pointer to Allergen object. The Allergen pointer receives a list of Allergen objects created in int main, this means that the relationship between User and Allergen is a “has-a” relationship. The Allergen objects still exist even if a User object is destroyed. This is used to associate a list of allergens a user is allergic to.
- FoodItem and FoodCategory: This is an aggregation relationship, where FoodItem has a pointer to FoodCategory object. The FoodCategory pointer receives an object of FoodCategory created in int main, this means that the relationship between FoodItem and FoodCategory is a “has-a” relationship. This is used to categorize food items to breakfast, lunch, dinner, and snack.

- **Composition**

- FoodItem and Diet: This is a composition relationship. This means that a Diet cannot exist without a FoodItem, if a FoodItem is destroyed diet will be destroyed too. This too is done to enhance the experience of the user while searching for the FoodItem.

- **Polymorphism**

- FoodCategory and (Lunch,breakfast,dinner ,dessert)The FoodCategory interface is designed to be polymorphic. Classes inheriting from FoodCategory implement the getCategory method, allowing FoodItem to treat different FoodCategory objects uniformly.

1.Section B (Implementation Of Concept):

1.1 Implementation Of OOP:-

Foodi leverages key Object-Oriented Programming (OOP) concepts to manage and operate on user profiles and food items effectively. Following is an explanation of how each OOP concept (Encapsulation, Composition , Aggregation , Inheritance , Polymorphism , Array of objects) is applied within Our source code:|

Encapsulation

Encapsulation is the bundling of data with methods that operate on that data. It restricts direct access to some of the object's components, which is a means of preventing accidental interference and misuse of the methods and data.

Example: The Allergen and User classes demonstrate encapsulation. Data members such as name and email in the User class are private and can only be accessed through public getter and setter methods.


```
class User {  
private:  
    string name;  
    string email;  
    vector<Allergen> allergies;  
  
public:  
    User(string _name = "", string _email = "", vector<Allergen> _allergies = {})  
        : name(_name), email(_email), allergies(_allergies) {}  
  
    string getEmail() const {  
        return email;  
    }  
    void setEmail(string _email) { - Setter  
        email = _email;  
    }  
    string getName() const { - getter  
        return name;  
    }  
    void setName(string _name) {  
        name = _name;  
    }  
    vector<Allergen> getAllergies() const {  
        return allergies;  
    }  
};
```

Composition

Composition represents a "has-a" relationship where one class contains objects of another class.

- **Example:** The User class has a vector of Allergen objects, indicating that a user can have multiple allergies. Here, User is composed of Allergen objects.


```
10 class Allergen {
11 private:
12     string name;
13
14 public:
15     Allergen(string _name) : name(_name) {}
16     string getName() const {
17         return name;
18     }
19     void setName(string _name) {
20         name = _name;
21     }
22 };
23
24 // Class representing a User
25 class User {
26 private:
27     string name;
28     string email;
29     vector<Allergen> allergies;
30
31 public:
32     User(string _name = "", string _email = "", vector<Allergen> _allergies = {})
33         : name(_name), email(_email), allergies(_allergies) {}
34
35     string getEmail() const {
36         return email;
37     }
38     void setEmail(string _email) {
39         email = _email;
40     }
41 }
```



Aggregation

Aggregation is a special form of association representing a "whole-part" relationship where the part can exist independently of the whole.

- **Example:** The FoodItem class aggregates FoodCategory objects through a pointer. This indicates that while FoodItem includes a FoodCategory, FoodCategory can exist independently.

```
class FoodItem: public Cuisine {  
private:  
    string name;  
    vector<string> ingredients;  
    FoodCategory* category;   
    Diet diet;  
public:  
    FoodItem(string cuisine_n = "", string n = "", vector<string> ing = {},  
             FoodCategory* c = nullptr, string diet_n = "")  
        : Cuisine(cuisine_n, name(n), ingredients(ing), category(c), diet(diet_n) {}  
};
```

Inheritance

Inheritance allows a class to inherit properties and behavior from another class, facilitating code reusability and a hierarchical classification.

- **Example:** The Cuisine class is a base class, and FoodItem inherits from it, extending its functionalities.
- FoodCategory is another example, where Dessert, Breakfast, Lunch, and Dinner classes inherit from the FoodCategory interface.

```
class FoodItem: public Cuisine {  
private:  
    string name;  
    vector<string> ingredients;  
    FoodCategory* category;  
    Diet diet;  
public:  
    FoodItem(string cuisine_n = "", string n = "", vector<string> ing = {},  
             FoodCategory* c = nullptr, string diet_n = "")  
        : Cuisine(cuisine_n), name(n), ingredients(ing), category(c), diet(diet_n) {}  
    ~FoodItem() {}  
};
```

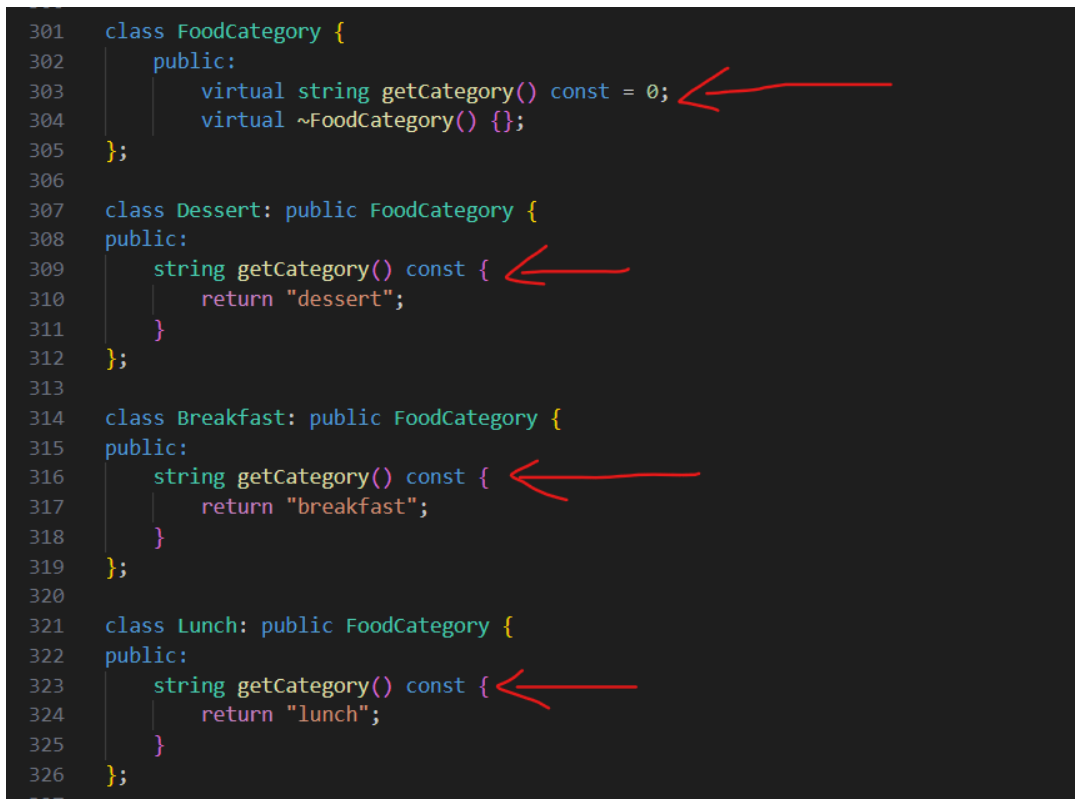
```
class FoodCategory {  
public:  
    virtual string getCategory() const = 0;  
    virtual ~FoodCategory() {};  
};  
  
class Dessert: public FoodCategory {  
public:  
    string getCategory() const {  
        return "dessert";  
    }  
};  
  
class Breakfast: public FoodCategory {  
public:  
    string getCategory() const {  
        return "breakfast";  
    }  
};  
  
class Lunch: public FoodCategory {  
public:  
    string getCategory() const {  
        return "lunch";  
    }  
};
```

Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common base class, particularly useful for dynamic method binding.

- **Example:** The FoodCategory interface is designed to be polymorphic. Classes inheriting from FoodCategory implement the getCategory method, allowing FoodItem to treat different FoodCategory objects uniformly

```
301 class FoodCategory {
302     public:
303         virtual string getCategory() const = 0;
304         virtual ~FoodCategory() {};
305 };
306
307 class Dessert: public FoodCategory {
308     public:
309         string getCategory() const {
310             return "dessert";
311         }
312 };
313
314 class Breakfast: public FoodCategory {
315     public:
316         string getCategory() const {
317             return "breakfast";
318         }
319 };
320
321 class Lunch: public FoodCategory {
322     public:
323         string getCategory() const {
324             return "lunch";
325         }
326 };
327
```



Array of Objects

An array of objects is a data structure where each element is an object. This allows storing and managing a collection of objects in a structured way.

- **Example:** The system uses vector containers to store multiple Allergen objects within a User and multiple FoodItem objects to manage food data.

```
10 class Allergen {
11 private:
12     string name;
13
14 public:
15     Allergen(string _name) : name(_name) {}
16     string getName() const {
17         return name;
18     }
19     void setName(string _name) {
20         name = _name;
21     }
22 };
23
24 // Class representing a User
25 class User {
26 private:
27     string name;
28     string email;
29     vector<Allergen> allergies;
30
31 public:
32     User(string _name = "", string _email = "", vector<Allergen> _allergies = {})
33         : name(_name), email(_email), allergies(_allergies) {}
34
35     string getEmail() const {
36         return email;
37     }
38     void setEmail(string _email) {
39         email = _email;
40     }
41 }
```

```
351 class FoodItem: public Cuisine {
352 private:
353     string name;
354     vector<string> ingredients;
355     FoodCategory* category;
356     Diet diet;
357
358 public:
```

2.2 Conclusion:-

The Foodie system represents a significant advancement in the field of food allergy management, offering a comprehensive solution that caters to the diverse needs of individuals with food sensitivities. By leveraging a user-friendly interface and a robust database of food items and allergens, Foodi provides users with the ability to make informed dietary choices, thereby enhancing their quality of life and ensuring their safety.

The system's design effectively incorporates core object-oriented principles such as encapsulation, inheritance, and polymorphism, which not only facilitate maintainability and scalability but also ensure that the application can be easily extended to include additional features or accommodate new types of food allergens. This modular design allows for the seamless integration of new functionalities without compromising the integrity or performance of the system.

Foodi's key features, including personalized allergen profiles, a searchable food database, and comprehensive reporting capabilities, empower users to quickly and accurately identify foods that are safe for consumption. The ability to update personal information and manage allergen lists ensures that the system remains relevant and useful as users' needs evolve.

In conclusion, Foodi stands as a vital tool for those managing food allergies, providing a reliable and efficient means of avoiding harmful allergens. The system's thoughtful design and implementation underscore its potential as a valuable resource in promoting healthier, safer dietary habits. Through continuous improvement and user feedback, Foodi can expand its capabilities and continue to serve as a trusted companion for individuals navigating the complexities of food allergies.