Data structures:

1. **Linked List** for storing recipes.
2. **Hash Map** (unordered_map) for fast lookup of recipes by categories.
3. **Vector** for storing a collection of recipes, which will allow us to perform sorting and searching efficiently.

Algorithms:

1. **Search Algorithm**: For finding recipes based on category or name.
2. **Sorting Algorithm**: To sort recipes based on category or name.

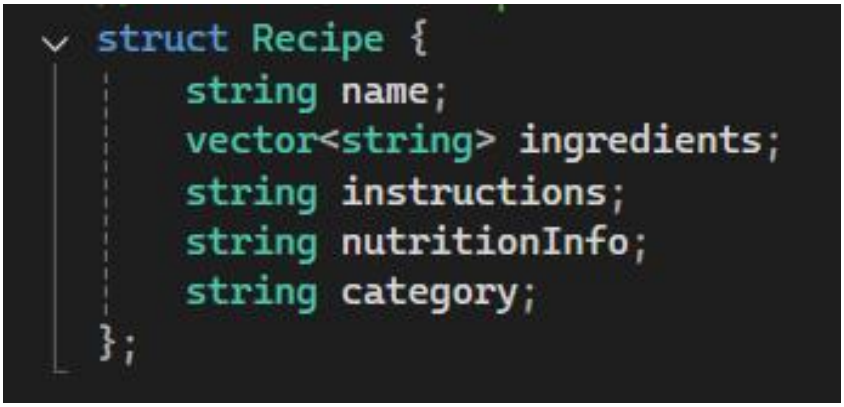
Features:

- **Add recipes**: Each recipe includes a name, ingredients, instructions, nutritional facts, and a category.
- **Categorization**: The user can choose a category for each recipe (e.g., Breakfast, Lunch, Dinner).
- **Search**: The user can search for recipes by category or name.
- **Display recipe details**: The user can view a recipe's ingredients, instructions, and nutritional information.

Explanation of the Code:

1. Recipe Structure:

- Each recipe consists of:
  - `name`: The name of the recipe.
  - `ingredients`: A list (vector) of ingredients.
  - `instructions`: Instructions on how to prepare the recipe.
  - `nutritionInfo`: Nutritional information of the recipe.
  - `category`: The category of the recipe (e.g., Breakfast, Lunch, Dinner).



2. RecipeManager Class:

- The RecipeManager class contains:
  - recipes: A vector that stores all the recipes.
  - categoryMap: A hash map (unordered_map) for fast lookups of recipes based on their category.
  - **addRecipe()**: Adds a new recipe to the collection and updates the category map.
  - **displayRecipes()**: Displays all recipes.
  - **displayByCategory()**: Displays recipes belonging to a specific category.
  - **searchRecipeByName()**: Searches and displays a recipe by its name.
  - **sortRecipesByName()**: Sorts recipes by their name using the sort function from the C++ Standard Library.
  - **sortRecipesByCategory()**: Sorts recipes by category.

```
class RecipeManager {
private:
    vector<Recipe> recipes;  // Vector na i store so all recipes            Ano ang vector? > naga sto
    unordered_map<string, vector<Recipe>> categoryMap;  // HashMap for searching by category

public:
    void saveIngredients() {
        std::ofstream record("Lusi.txt", std::ios_base::app);
        if (!record) {
            std::cout<< "file not found" << std::endl;
        }


    }
    // Add a new recipe
    void addRecipe(const Recipe& recipe) {
        recipes.push_back(recipe);
        categoryMap[recipe.category].push_back(recipe);
    }

    // Display all recipes
    void displayRecipes() {
        if (recipes.empty()) {
            cout << "No recipes available." << endl;
            return;
        }

        for (const auto& recipe : recipes) { // const auto& in a range for to process the elements
            cout << "Recipe: " << recipe.name << endl;
            cout << "Category: " << recipe.category << endl;
            cout << "Ingredients: ";
            for (const auto& ingredient : recipe.ingredients) {
                cout << ingredient << ", ";
            }
            cout << endl;
            cout << "Instructions: " << recipe.instructions << endl;
            cout << "Nutrition Info: " << recipe.nutritionInfo << endl;
            cout << "-----------------------------------" << endl;
        }
```

3. User Input:

- The function inputRecipe() allows the user to input a recipe with the required details (name, category, ingredients, instructions, and nutritional facts).

```
Recipe inputRecipe() {
    Recipe recipe;
    cout << "Enter recipe name: ";
    getline(cin, recipe.name);

    cout << "Enter recipe category (e.g., Breakfast, Lunch, Dinner): ";
    getline(cin, recipe.category);

    cout << "Enter ingredients (comma-separated): ";
    string ingredientsLine;
    getline(cin, ingredientsLine);
    size_t pos = 0; // size_t > can store maximum size of any type(kasabay and array)
    while ((pos = ingredientsLine.find(',')) != string::npos) {
        recipe.ingredients.push_back(ingredientsLine.substr(0, pos));
        ingredientsLine.erase(0, pos + 1);
    }
    if (!ingredientsLine.empty()) {
        recipe.ingredients.push_back(ingredientsLine);
    }

    cout << "Enter instructions: ";
    getline(cin, recipe.instructions);

    cout << "Enter nutritional information: ";
    getline(cin, recipe.nutritionInfo);

    return recipe;
}
```

How it Works:

- **Adding Recipes**: Users can input new recipes with their details. Recipes are added to the vector and categorized using the hash map.
- **Displaying Recipes**: The user can view all recipes or search for recipes by category or name.
- **Sorting**: Users can sort recipes either by name or category.
- **Search**: The user can search for a recipe by its name, and if found, the recipe's details are displayed.