객체지향 프로그래밍 Project 1

Class 3 Team 1

A

❖ Project Title: IIKH

❖ List of team members: Class 03 team 1

20225679 서규민, 20224680 김경민, 20225779 김제신, 20223908 김주영, 20202203 박호근, 20200956 배정환, 20214782 조주원.

❖ Presentation speaker name: 조주원

❖ Brief project description (summary)

This project is to design a kitchen helper and we have made it possible to store a lot of recipes and plan daily meals through these. In the process, this kitchen helper can also search for recipes, add new recipes, delete unnecessary recipes, and update recipes to better ones.

B

❖ How to compile and execute

- Open IIKH.sln with Visual Studio 2022 use C++14 and build

❖ System requirement for compilation and execution

- OS: Microsoft Windows 11
- System requitement: Visual studio 2022 with C++14 and SQLite3

C

❖ Description on functionality that was implemented in your SW system.

(There are a number of features that have been implemented)

- **Greeter class:** The Greeter class displays menus to the user and, depending on user choice, performs actions related to the database or exits the program. This enables you to interact with the user and determines the main control flow of the program.

- **DatabaseManager class:** The Database Manager class abstracts interactions with SQLite databases, providing the ability to retrieve data from databases and process results. These classes will likely be used to read and write data related to recipes and plans.

- **PlanDB:** PlanDB provides the ability to enter plan data, store it in a database, and retrieve, update, and delete stored data.

1) Print plan: Outputs all the plans. Allows users to view all the plans and select a specific plan to view the details.

2) Search plan: Plan information for all dates or names may be retrieved and output according to a user's selection. In addition, when a specific date or name is entered, the plan information about it is printed in the database and the user can view the details and recipes of the selected plan. (if the user enters the date "2023-10-15" or the name of the plan, this system will find and print a plan for that date or name.)

3) Add plan: Receive the name or date from user want to add the plan to, and the breakfast, lunch, and dinner menus, and add the information to the database. Also receive the plan name and date from the user, then take the breakfast, lunch, and dinner menus of the plan that have already been saved with that plan name and add them to the database with the date user entered. This saves new planning information. (if a user enters a date or name 2023-10-15" and a menu for breakfast, lunch, and dinner, that information is added to the database to create a new plan. And when user enter a plan name and date, user create a new plan by adding the breakfast, lunch, and dinner menus stored in that plan name to the database along with that date.)

4) Delete plan: Receive the date or plan name from user that want to delete, delete the plan for that date or name from the database. This deletes the plan information for that date or name. (if a user enters a date or plan name "2023-10-15", the plan for that date or name is found and deleted, and the selected plan information is deleted from the database.)

5) Update plan: Receive the date or name from user that want to update, and find the plan for that date or name in the database and receive the information that want to update. When the user selects the items that they want to update (date or name, breakfast, lunch, dinner) and enters something new, the plan information for that date is updated. (if user enter a date of 2023-10-15 and want to update the date to "2023-10-16", user will receive this information to update the date of the plan for that date.)

6) Grocery List: helps user effectively track and manage the ingredients that need for your planned meals. This makes it easy for users to purchase the ingredients they need according to their plans and to see what ingredients they will use to cook.

- **RecipeDB:** RecipeDB is a database class for recipe management and performs various functions. Also When the RecipeDB object is created, the generator initializes the database and uses a database file named "iikh.db". It also creates a table named "recipe" in the database.

1) Print recipe: Retrieves all recipe information from the database. This information includes the name and brief description of each recipe. Outputs the retrieved recipe information to the user. Each recipe is accompanied by a name and description.

2) Search recipe: Ask the user to choose either "Print All" or "Select". Depending the choice, the following actions are performed. Print all mode: get all recipes from the database and output the name and description. Select mode: select a specific recipe and output the detailed information of that recipe. Invalid input: Outputs a "Wrong Input" message and guides user to re-select. (if the user choose “Print All“, Retrieves and prints all recipes from the database. Or choose ”Select“ and type the name "Recipe C", Prints the recipe details corresponding to ”Recipe C".)

3) Add recipe: Receive the name, description, ingredients, and recipe of the recipe from the user, all required information, create a new recipe using the input, and insert it into the database. (if user enter a recipe name, description, ingredient, and recipe, this system create a new Recipe object and insert it into the database as follows.)

4) Delete recipe: Receive the name of the recipe from user that want to delete and verify that the name of the recipe entered exists in the database. If a recipe name exists, it deletes it from the database, and if not, it displays a "Wrong Input" message and does not perform any deletion action. (if a user tries to delete a recipe named Recipe A, it is deleted if the recipe is found in the database, and if it does not exist, a "Wrong Input" message is displayed and no deletion action is performed.)

5) **Update recipe:** Receive the name of the recipe from user that want to update and verify that the name of the recipe which is entered exists in the database. If the recipe name exists, the user receives the items (name, description, ingredient, recipe) and contents of the recipe information that you want to update, and updates the recipe information with the new information that user have entered. If the recipe name does not exist in the database, it displays a "Wrong Input" message and does not perform an update operation.

- **Plan:** The Plan is used to store meal plan information and output it that can use this to view or update the details of user desired meal menu.

- **Recipe:** The Recipe allows user to store and manage cooking recipe information and print out the name and details of the recipe. This allows users to use it to add and verify cooking recipes.

- **Date:** Perform simple date-related tasks, such as comparing dates and importing current dates. The date comparison function treats the date as a string, and the Get Current Date function returns the current date using the system time.

- **similarity:** It is used as a simple search tool to search for strings in a specific table in a database, output search results, and provide similar items to users. Search jobs can be selected from the "plan" or "recipe" tables using flag parameters.

- **main:** The main function is the entry point of the C++ program and provides a basic program routine that allows the user to select the desired action and control the behavior of the program, such as IIKH logo output or wait for 1 second, Greeter object creation, and infinite loops.

D

❖ **How you implemented (important implementation issues)**

Greeter

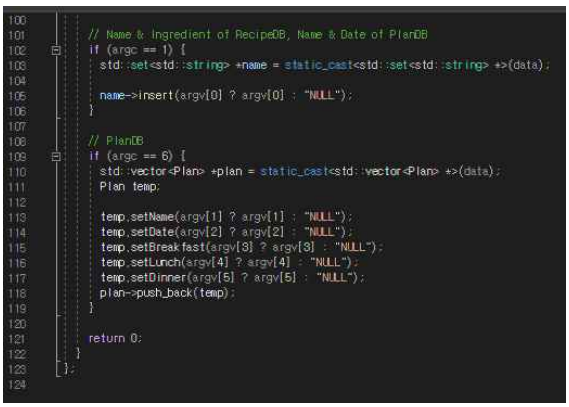
```cpp
#pragma once

#include <iostream>
#include <string>
#include <vector>

#include "PlanDB.h"
#include "RecipeDB.h"

class Greeter {
private:
 std::vector<std::string> menu;
 RecipeDB recipeDB;
 PlanDB planDB;

public:
 Greeter() {
  menu.push_back("SEARCH RECIPE");     // 1
  menu.push_back("ADD RECIPE");        // 2
  menu.push_back("DELETE RECIPE");     // 3
  menu.push_back("UPDATE RECIPE");     // 4
  menu.push_back("SEARCH PLAN");       // 5
  menu.push_back("ADD PLAN");          // 6
  menu.push_back("DELETE PLAN");       // 7
  menu.push_back("UPDATE PLAN");       // 8
  menu.push_back("SHOW GROCERY LIST"); // 9
  menu.push_back("QUIT");              // 10
 }

 void printAndSelectMenu() {
  int selectNum;

  for (int i = 0; i < menu.size(); i++) {
   std::cout << i + 1 << ". " << menu[i] << std::endl;
  }

  std::cout << "Select Menu: ";
  std::cin >> selectNum;
  std::cin.ignore();
  system("cls");

  std::cout << "Selected Menu: " << menu[selectNum - 1] << std::endl;
  switch (selectNum) {
   case 1:
    recipeDB.searchRecipe();
    break;
   case 2:
    recipeDB.addRecipe();
    break;
   case 3:
    recipeDB.deleteRecipe();
    break;
   case 4:
    recipeDB.updateRecipe();
    break;
   case 5:
    planDB.searchPlan();
    break;
   case 6:
    planDB.addPlan();
    break;
   case 7:
    planDB.deletePlan();
    break;
   case 8:
    planDB.updatePlan();
    break;
   case 9:
    planDB.showGroceryList();
    break;
   case 10:
    quit();
    break;
   default:
    std::cout << "Wrong Input" << std::endl;
    break;
  }
 }

 static void quit() {
  std::cout << "Bye" << std::endl;
  exit(0);
 }
};
```

This Greeter class is intended to create menu-based applications that interact with users. Preprocess to prevent duplicate header files with #pragma once, iostream, string, vector headers are included to use standard library functions, and "DatabaseManager.h", "PlanDB.h", "RecipeDB.h", and "sqlite/sqlite3.h" headers are implemented to use SQLite, an external library, and a user-defined class. And this class defines the core functions of menu-based applications, menu that is vector for serves to store menu items, and RecipeDB and PlanDB that these two objects are implemented as classes that perform tasks related to database management and recipe and planning management.

The Greeter constructor initialized the menu vector and added menu items, and the printAndSelectMenu function allowed the user to display menu items and receive user input. You also used a switch statement to perform the action based on the menu items you selected. For example, if the user selects the "SEARCH RECIPE" menu, the recipeDB.searchRecipe() function is called, and if the "QUIT" menu is selected, the quant() function is called. Displays a "Wrong Input" message when the user makes an incorrect input. And a quant static function that is responsible for terminating the program, displaying a "Bye" message and calling exit(0) to terminate the program. So, this class implements a menu-based user interface, which uses RecipeDB and PlanDB classes to perform database tasks, and the code provides a variety of menu options through the menu vector, and has a structure to perform those tasks according to the options you choose.

DatabaseManager

```cpp
#pragma once

#include <iostream>
#include <set>

#include "Plan.h"
#include "Recipe.h"
#include "sqlite/sqlite3.h"

class DatabaseManager {
public:
  int rc;  // SQLite3 return code

  DatabaseManager(const char *dbName) {
    rc = sqlite3_open(dbName, &db);

    if (rc) {
      std::cerr << "Can't Open Database: " << sqlite3_errmsg(db) << std::endl;
    }
  }

  ~DatabaseManager() { sqlite3_close(db); }

  void executeQuery(const char *query) {
    char *errMsg = nullptr;
    rc = sqlite3_exec(db, query, nullptr, nullptr, &errMsg);

    if (rc != SQLITE_OK) {
      std::cerr << "Error: " << errMsg << std::endl;
      sqlite3_free(errMsg);
    }
  }

  // Recipe & Plan
  void executeQuery(const char *query, void *data) {
    char *errMsg = nullptr;
    rc = sqlite3_exec(db, query, getterCallback, data, &errMsg);

    if (rc != SQLITE_OK) {
      std::cerr << "Error: " << errMsg << std::endl;
      sqlite3_free(errMsg);
    }
  }

  // vector
  void executeQuery(const char *query, void *data, int) {
    char *errMsg = nullptr;
    rc = sqlite3_exec(db, query, vectorCallback, data, &errMsg);

    if (rc != SQLITE_OK) {
      std::cerr << "Error: " << errMsg << std::endl;
      sqlite3_free(errMsg);
    }
  }

private:
  sqlite3 *db;

  // argc 5: Recipe, 6: Plan
  static int getterCallback(void *data, int argc, char **argv,
                            char **azColName) {
    // RecipeDB
    if (argc == 5) {
      Recipe *recipe = static_cast<Recipe *>(data);

      recipe->setMenuName(argv[1] ? argv[1] : "NULL");
      recipe->setMenuDescription(argv[2] ? argv[2] : "NULL");
      recipe->setMenuIngredient(argv[3] ? argv[3] : "NULL");
      recipe->setMenuRecipe(argv[4] ? argv[4] : "NULL");
    }

    // PlanDB
    if (argc == 6) {
      Plan *plan = static_cast<Plan *>(data);

      plan->setName(argv[1] ? argv[1] : "NULL");
      plan->setDate(argv[2] ? argv[2] : "NULL");
      plan->setBreakfast(argv[3] ? argv[3] : "NULL");
      plan->setLunch(argv[4] ? argv[4] : "NULL");
      plan->setDinner(argv[5] ? argv[5] : "NULL");
    }

    return 0;
  }

  // vector
  static int vectorCallback(void *data, int argc, char **argv,
                            char **azColName) {
    // RecipeDB
    if (argc == 5) {
      std::vector<Recipe> *recipe = static_cast<std::vector<Recipe> *>(data);
      Recipe temp;

      temp.setMenuName(argv[1] ? argv[1] : "NULL");
      temp.setMenuDescription(argv[2] ? argv[2] : "NULL");
      temp.setMenuIngredient(argv[3] ? argv[3] : "NULL");
      temp.setMenuRecipe(argv[4] ? argv[4] : "NULL");
      recipe->push_back(temp);
    }

    // Name & Ingredient of RecipeDB, Name & Date of PlanDB
    if (argc == 1) {
      std::set<std::string> *name = static_cast<std::set<std::string> *>(data);

      name->insert(argv[0] ? argv[0] : "NULL");
    }

    // PlanDB
    if (argc == 6) {
      std::vector<Plan> *plan = static_cast<std::vector<Plan> *>(data);
      Plan temp;

      temp.setName(argv[1] ? argv[1] : "NULL");
      temp.setDate(argv[2] ? argv[2] : "NULL");
      temp.setBreakfast(argv[3] ? argv[3] : "NULL");
      temp.setLunch(argv[4] ? argv[4] : "NULL");
```



```
    // Name & Ingredient of RecipeDB, Name & Date of PlanDB
    if (argc == 1) {
      std::set<std::string> *name = static_cast<std::set<std::string> *>(data);

      name->insert(argv[0] ? argv[0] : "NULL");
    }

    // PlanDB
    if (argc == 6) {
      std::vector<Plan> *plan = static_cast<std::vector<Plan> *>(data);
      Plan temp;

      temp.setName(argv[1] ? argv[1] : "NULL");
      temp.setDate(argv[2] ? argv[2] : "NULL");
      temp.setBreakfast(argv[3] ? argv[3] : "NULL");
      temp.setLunch(argv[4] ? argv[4] : "NULL");
      temp.setDinner(argv[5] ? argv[5] : "NULL");
      plan->push_back(temp);
    }

    return 0;
  }
};
```

The Database Manager class is responsible for accessing and query execution to SQLite3 databases and is implemented to extract and process data using various callback functions. The Database Manager class defined the classes that interact with the SQLite3 database, and the rc member variable that stores the SQLite3 return code, indicating whether the database operation succeeded or failed.

And the constructor Database Manager (constchar *dbName): opened the database file, established the database connection, and if the database failed to open, it printed an error message and caused the destructor ~DatabaseManager() to close the database connection. The executeQuery method is responsible for executing SQL queries in the database, functions have various formats of overload, and executeQuery(constchar *query) that Executes SQL queries and outputs error messages if an error occurs.

Also, executeQuery(const char *query, void *data) which is run SQL queries, extract and store data using the callback function getterCallback, run SQL queries through executeQuery(const char *query, void *data, int), extract and store data using the callback function vectorCallback to store the handle to the SQLite3 database with the sqlite3 *db member variable, maintaining the database connection. Finally, the getterCallback and vectorCallback static functions run SQL queries, use them to extract data, call each row in the result set, extract and process the data, and both functions receive the argc, argv, azColName parameters, which represent the number of columns, array of values in columns, and array of column names, respectively. In other words, this function has been implemented to store or process data extracted from the database in data parameters, fill Recipe and Plan objects or vectors, or fill string sets, depending on the database table structure.

This class allows us to open the SQLite3 database, run queries, and process results extracted from the database in a variety of ways, which plays an important role in C++ applications that interact with the database.

Main

```cpp
#include <chrono>
#include <iostream>
#include <thread>

#include "Greeter.h"

int main(void) {
  // print the iikh logo
  std::cout << "  III   III   K  K  H  H" << std::endl;
  std::cout << "   I     I    K K   H  H" << std::endl;
  std::cout << "   I     I    KK    HHHH" << std::endl;
  std::cout << "   I     I    K K   H  H" << std::endl;
  std::cout << "  III   III   K  K  H  H" << std::endl;

  // halt for 1s
  std::this_thread::sleep_for(std::chrono::seconds(1));

  Greeter g;

  while (true) {
    system("cls");
    g.printAndSelectMenu();
    system("pause");
  }

  return 0;
}
```

The purpose of main is to implement simple menu-based applications that interact with users using the Greeter class. Using std::this_thread:::sleep_for:std:::chrono::seconds(1) to stop for one second after the program starts and this is responsible for giving the user time to show the logo after displaying it, creating a Greeter object: creating an object g in the Greeter class that interacts with the user to display menus and select menu items.

It also uses an infinite loop: while (true) loop to continue running menu-based applications, erasing the screen through the system ("cls") and initializing the console. This will clear the console window and display a new menu. In addition, g.printAndSelectMenu() invokes the printAndSelectMenu function of the Greeter class to output menus and to select menu items from the user, which is the part that executes the menu management function of the Greeter class.

Finally, the program pauses so that the user can see menu items

with the system ("pause") and when the user selects a menu, the program continues after selecting the menu item.

So, the program continues to run through an infinite loop, allowing users to select menus, use the various functions of the program, and implement the program to represent a simple console application that uses the Greeter class to perform database management and menu management, and interact with users.

RecipeDB

```cpp
#pragma once

#include <iostream>
#include <sstream>
#include <string>
#include <vector>

#include "DatabaseManager.h"
#include "Recipe.h"
#include "Similarity.h"

class RecipeDB {
private:
    DatabaseManager dbm;

public:
    // noArgsConstructor
    RecipeDB() : dbm("ikh.db") {
        // createTable
        dbm.executeQuery(
            "CREATE TABLE IF NOT EXISTS recipe (recipe_id INTEGER PRIMARY KEY "
            "AUTOINCREMENT, name TEXT, description TEXT, ingredient TEXT, recipe "
            "TEXT);");
    }

    void searchRecipe() {
        std::cout << "Select a Mode (1. Print All, 2. Select) : ";
        int selectNum;
        std::cin >> selectNum;
        std::cin.ignore();
        system("cls");

        switch (selectNum) {
        case 1:
            printAllRecipe();
            break;
        case 2:
            selectRecipe();
            break;
        default:
            std::cout << "Wrong Input" << std::endl;
            break;
        }
    }

    void printAllRecipe() {
        std::cout << "All Recipe" << std::endl;
        std::vector<Recipe> recipes;
        dbm.executeQuery("SELECT * FROM recipe;", &recipes, true);
        if (recipes.empty()) {
            std::cout << "No Recipe" << std::endl;
            return;
        }
        for (auto &recipe : recipes) {
            recipe.printNameAndDescription();
        }

        std::cout << "Do you want to see a specific recipe? [y/n] : ";
        char select;
```

```
    std::cout << "Do you want to see a specific recipe? [y/n]: ";
    char select;
    std::cin >> select;
    std::cin.ignore();
    if (select == 'y') {
      std::cout << "Input Recipe Name: ";
      std::string name;
      std::getline(std::cin, name);
      selectRecipe(name);
    }
  }

  void selectRecipe() {
    Recipe recipe;
    std::string name;
    std::cout << "Input Recipe Name: ";
    std::getline(std::cin, name);

    dbm.executeQuery(
        ("SELECT * FROM recipe WHERE name = '" + name + "';").c_str(), &recipe);

    if (recipe.printRecipe() == 1) {
      Similarity similarity(name);
      similarity.checkSimilarity(2);
    }
  }

  void selectRecipe(const std::string &name) {
    Recipe recipe;
    dbm.executeQuery(
        ("SELECT * FROM recipe WHERE name = '" + name + "';").c_str(), &recipe);
    recipe.printRecipe();
  }

  Recipe selectRecipe(const std::string &name, int) {
    Recipe recipe;
    dbm.executeQuery(
        ("SELECT * FROM recipe WHERE name = '" + name + "';").c_str(), &recipe);
    return recipe;
  }

  void addRecipe() {
    Recipe recipe;
    recipe.addRecipe();
    std::set<std::string> recipeNames = getRecipeNames();
    if (recipeNames.find(recipe.getMenuName()) != recipeNames.end()) {
      std::cout << "Already Exist" << std::endl;
      return;
    }

    dbm.executeQuery(
        ("INSERT INTO recipe (name, description, ingredient, recipe) VALUES "
         "('" +
         recipe.getMenuName() + "', '" + recipe.getMenuDescription() + "', '" +
         recipe.getMenuIngredient() + "', '" + recipe.getMenuRecipe() + "');")
            .c_str());
  }
```

```
  void deleteRecipe() {
    std::string menu_recipe;
    std::set<std::string> recipeNames = getRecipeNames();

    std::cout << "Input Target Recipe Name: ";
    std::getline(std::cin, menu_recipe);
    if (recipeNames.find(menu_recipe) == recipeNames.end()) {
      std::cout << "Wrong Input" << std::endl;
      Similarity similarity(menu_recipe);
      similarity.checkSimilarity(2);
      return;
    }

    dbm.executeQuery(
        ("DELETE FROM recipe WHERE name='" + menu_recipe + "';").c_str());
  }

  void updateRecipe() {
    std::string item, content, menu_recipe;
    std::set<std::string> recipeNames = getRecipeNames();

    std::cout << "Input Target Recipe Name: ";
    std::getline(std::cin, menu_recipe);
    if (recipeNames.find(menu_recipe) == recipeNames.end()) {
      std::cout << "Wrong Input" << std::endl;
      Similarity similarity(menu_recipe);
      similarity.checkSimilarity(2);
      return;
    }

    std::cout << "What would you like to change? (name, description, "
                 "ingredient, recipe): ";
    std::getline(std::cin, item);
    if (item != "name" && item != "description" && item != "ingredient" &&
        item != "recipe") {
      std::cout << "Wrong Input" << std::endl;
      return;
    }

    if (item == "recipe") {
      std::cout << "Input Recipe Method (To finish, just press Enter)"
                << std::endl;
      int index = 1;
      while (true) {
        std::cout << "Step " << index << ": ";
        std::string userInput;
        std::getline(std::cin, userInput);
        if (userInput.compare("") == 0) {
          break;
        }
        content += std::to_string(index++) + ". ";
        content += userInput;
        content += "\n";
      }
    } else {
      std::cout << "What would you like to change the " + item + " to?: ";
      std::getline(std::cin, content);
    }
```

```
    if (item == "recipe") {
      std::cout << "Input Recipe Method (To finish, just press Enter)"
                << std::endl;
      int index = 1;
      while (true) {
        std::cout << "Step " << index << ": ";
        std::string userInput;
        std::getline(std::cin, userInput);
        if (userInput.compare("") == 0) {
          break;
        }
        content += std::to_string(index++) + ". ";
        content += userInput;
        content += "\n";
      }
    } else {
      std::cout << "What would you like to change the " + item + " to?: ";
      std::getline(std::cin, content);
    }

    dbm.executeQuery(("UPDATE recipe SET " + item + " = '" + content +
                      "' WHERE name = '" + menu_recipe + "';")
                         .c_str());
  }

  std::set<std::string> getRecipeNames() {
    std::set<std::string> recipeNames;
    dbm.executeQuery("SELECT name FROM recipe;", &recipeNames, true);
    return recipeNames;
  }
};
```

The RecipeDB class is responsible for managing the recipe database, which is implemented to provide a variety of functions to search, add, modify, and delete recipe information stored in the database.

The RecipeDB class initializes objects in the Database Manager class through the constructor, using which it interacts with the SQLite database "iikh.db" If there is no database table "recipe", we have implemented a CREATE TABLE query to create a recipe table, instructing the user to select search mode with searchRecipe() and providing two options for search mode: "Print All" and "Select".

And we searched and printed all the recipe information stored in the database with printAllRecipe(), asked whether the user would like to see a specific recipe, received the recipe name and provided the recipe name with selectRecipe(), and searched and printed the recipe information in the database, and if the searched recipe does not exist, a similar recipe was searched and provided. SelectRecipe(conststd::string &name) then searches for a recipe based on its name and returns the corresponding recipe information, selectRecipe(conststd::string &name, int) searches for a recipe based on its name and returns the corresponding recipe information, which causes the function to return objects in the Recipe class.

Additionally, addRecipe() receives recipe information (name, description, ingredient, recipe) from the user and generates recipes through the Recipe class If the recipe name already exists in the database, display an "AlreadyExist" message and stop adding it.

Otherwise, user were asked to add recipe information to the

database. Then deleteRecipe() receives the recipe name to be deleted from the database, and updateRecipe() receives the recipe name to be updated from the user, selects which part to update (name, description, ingredient, recipe), updates the selected part, and getRecipeNames() searches the database for all recipe names and returns them as a set.

Therefore, the RecipeDB class uses the Recipe class and Database Manager class to manage the recipe database, and provides users with the ability to search, add, modify, and delete recipes, so that the recipe database can be managed and utilized efficiently.

PlanDB

```
#pragma once

#include <algorithm>
#include <iostream>
#include <map>
#include <set>
#include <utility>
#include <vector>

#include "DatabaseManager.h"
#include "Date.h"
#include "Plan.h"
#include "RecipeDB.h"
#include "Similarity.h"

class PlanDB {
private:
    DatabaseManager dbm;
    RecipeDB recipe_db_;
    std::set<std::string> RecipeName;
    std::vector<std::pair<std::string, std::set<std::string>>> ingredients;
    Date _date;

public:
    PlanDB() : dbm("likh.db") {
        // createTable
        dbm.executeQuery(
            "CREATE TABLE IF NOT EXISTS plan (plan_id INTEGER PRIMARY KEY "
            "AUTOINCREMENT, name TEXT, date DATE, breakfast TEXT, lunch TEXT, "
            "dinner text);");

        // insert into map
        makeDatePlanGroceryList();
    }

    std::set<std::string> getNames() {
        std::set<std::string> temp;

        dbm.executeQuery("SELECT name FROM plan WHERE name IS NOT NULL;", &temp,
                         true);

        return temp;
    }

    std::set<std::string> getDates() {
        std::set<std::string> temp;

        dbm.executeQuery(
            "SELECT date FROM plan WHERE date IS NOT NULL ORDER BY date ASC;",
            &temp, true);

        return temp;
    }

    void searchPlan() {
        std::cout << "Select a Mode (1. Print All Date Plan, 2. Print All Name "
                     "Plan, 3. Select Date, 4. Select Name, 5. Select Period) : ";
        int selectNum;
        std::cin >> selectNum;
```

```
                "Plan, 3. Select Date, 4. Select Name, 5. Select Period): ";
    int selectNum;
    std::cin >> selectNum;
    std::cin.ignore();
    system("cls");

    switch (selectNum) {
      case 1:
        printAllPlanByDate();
        break;
      case 2:
        printAllPlanByName();
        break;
      case 3:
        selectPlanByDate();
        break;
      case 4:
        selectPlanByName();
        break;
      case 5:
        selectPeriodList();
        break;
      default:
        std::cout << "Wrong Input" << std::endl;
        break;
    }
  }

  void printAllPlanByDate() {
    std::vector<Plan> plans;

    std::cout << "All Plan" << std::endl;
    dbm.executeQuery(
        "SELECT * FROM plan WHERE date IS NOT NULL ORDER BY date ASC;", &plans,
        true);
    if (plans.empty()) {
      std::cout << "No Date Plan" << std::endl;
      return;
    }
    for (auto &plan : plans) {
      plan.printPlanDate();
    }

    std::cout << "Do you want to see a specific plan? [y/n]: ";
    char select;
    std::cin >> select;
    std::cin.ignore();
    if (select == 'y') {
      std::cout << "Input Plan Date: ";
      std::string plan;
      std::getline(std::cin, plan);
      selectPlanByDate(plan);
    }
  }

  void printAllPlanByName() {
    std::vector<Plan> plans;

    std::cout << "All Plan" << std::endl;
    dbm.executeQuery("SELECT * FROM plan WHERE name IS NOT NULL;", &plans,
                     true);
    if (plans.empty()) {
      std::cout << "No Name Plan" << std::endl;
      return;
    }
    for (auto &plan : plans) {
      plan.printPlanName();
    }

    std::cout << "Do you want to see a specific plan? [y/n]: ";
    char select;
    std::cin >> select;
    std::cin.ignore();
    if (select == 'y') {
      std::cout << "Input Plan Name: ";
      std::string plan;
      std::getline(std::cin, plan);
      selectPlanByName(plan);
    }
  }

  void selectPlanByName() {
    Plan plan;
    std::string name;

    std::cout << "Input Plan Name: ";
    std::getline(std::cin, name);
    dbm.executeQuery(
        ("SELECT * FROM plan WHERE name = '" + name + "';").c_str(), &plan);

    int selectNum = plan.printPlan();
    if (selectNum == 1) {
      std::string breakfast = plan.getBreakfast();
      recipe_db_.selectRecipe(breakfast);
    } else if (selectNum == 2) {
      std::string lunch = plan.getLunch();
      recipe_db_.selectRecipe(lunch);
    } else if (selectNum == 3) {
      std::string dinner = plan.getDinner();
      recipe_db_.selectRecipe(dinner);
    } else {
      Similarity similarity(name);
      similarity.checkSimilarity(1);
      return;
    }
  }

  void selectPlanByName(const std::string &name) {
    Plan plan;

    dbm.executeQuery(
        ("SELECT * FROM plan WHERE name = '" + name + "';").c_str(), &plan);

    int selectNum = plan.printPlan();
    if (selectNum == 1) {
      std::string breakfast = plan.getBreakfast();
```

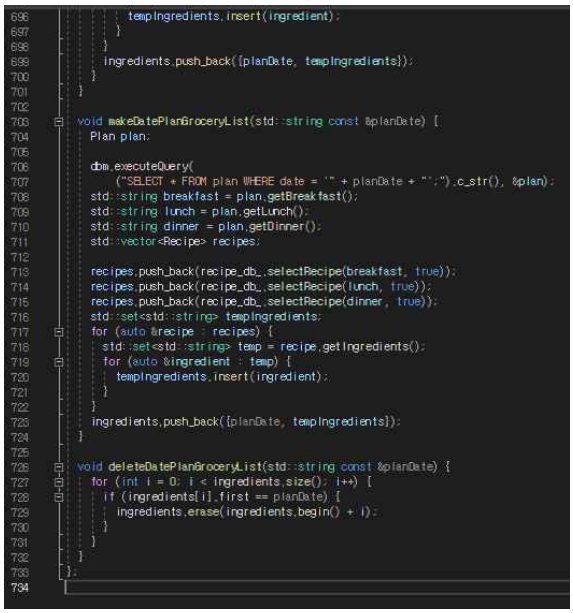
```
    if (selectNum == 1) {
      std::string breakfast = plan.getBreakfast();
      recipe_db_.selectRecipe(breakfast);
    } else if (selectNum == 2) {
      std::string lunch = plan.getLunch();
      recipe_db_.selectRecipe(lunch);
    } else if (selectNum == 3) {
      std::string dinner = plan.getDinner();
      recipe_db_.selectRecipe(dinner);
    } else {
      return;
    }
  }

  void selectPlanByDate() {
    Plan plan;
    std::string date;

    std::cout << "Input Plan Date: ";
    std::getline(std::cin, date);
    dbm.executeQuery(
        ("SELECT * FROM plan WHERE date = '" + date + "';").c_str(), &plan);

    int selectNum = plan.printPlan();
    if (selectNum == 1) {
      std::string breakfast = plan.getBreakfast();
      recipe_db_.selectRecipe(breakfast);
    } else if (selectNum == 2) {
      std::string lunch = plan.getLunch();
      recipe_db_.selectRecipe(lunch);
    } else if (selectNum == 3) {
      std::string dinner = plan.getDinner();
      recipe_db_.selectRecipe(dinner);
    } else {
      return;
    }
  }

  void selectPlanByDate(const std::string &date) {
    Plan plan;

    dbm.executeQuery(
        ("SELECT * FROM plan WHERE date = '" + date + "';").c_str(), &plan);

    int selectNum = plan.printPlan();
    if (selectNum == 1) {
      std::string breakfast = plan.getBreakfast();
      recipe_db_.selectRecipe(breakfast);
    } else if (selectNum == 2) {
      std::string lunch = plan.getLunch();
      recipe_db_.selectRecipe(lunch);
    } else if (selectNum == 3) {
      std::string dinner = plan.getDinner();
      recipe_db_.selectRecipe(dinner);
    } else {
      return;
    }
  }

  void addPlan() {
    std::cout << "Select a Mode (1. Add Date Plan, 2. Add Name Plan, 3. Add "
                 "Date Plan Using Name Plan) : ";
    int selectNum;
    std::cin >> selectNum;
    std::cin.ignore();
    system("cls");

    switch (selectNum) {
      case 1:
        addDatePlan();
        break;
      case 2:
        addNamePlan();
        break;
      case 3:
        addDatePlanUsingNamePlan();
        break;
      default:
        std::cout << "Wrong Input" << std::endl;
        break;
    }
  }

  void addDatePlanUsingNamePlan() {
    std::string planDate;
    std::string planName;
    std::set<std::string> Name = getNames();
    std::set<std::string> Date = getDates();
    Plan inputPlan;

    std::cout << "----------Name Plan List----------" << std::endl;
    for (auto &name : Name) {
      std::cout << "Plan Name : " << name << std::endl;
    }

    std::cout << "----------------------------------" << std::endl;
    std::cout << "Input Name: ";
    std::getline(std::cin, planName);
    if (Name.find(planName) == Name.end()) {
      std::cout << "Wrong Input" << std::endl;
      return;
    }

    std::cout << "Input Date (YYYY-MM-DD): ";
    std::getline(std::cin, planDate);
    std::string today = Date::getToday();
    if (_date.compareDate(today, planDate)) {
      std::cout << "Enter Date After " + today << std::endl;
      return;
    }
    if (Date.find(planDate) != Date.end()) {
      std::cout << "Wrong Input" << std::endl;
      return;
    }
```

```
        return;
    }

    dbm.executeQuery(
        ("SELECT * FROM plan WHERE name = '" + planName + "';").c_str(),
        &inputPlan);
    // planDate & breakfast, lunch, dinner of inputPlan
    dbm.executeQuery(("INSERT INTO plan VALUES(NULL, NULL, '" + planDate +
                      "', '" + inputPlan.getBreakfast() + "', '" +
                      inputPlan.getLunch() + "', '" + inputPlan.getDinner() +
                      "');")
                         .c_str());
    makeDatePlanGroceryList(planDate);
}

void addNamePlan() {
    std::string planName;
    std::string planBreakfast;
    std::string planLunch;
    std::string planDinner;
    RecipeName = recipe_db_.getRecipeNames();

    std::cout << "Input Plan Name: ";
    std::getline(std::cin, planName);

    std::cout << "Input breakfast: ";
    std::getline(std::cin, planBreakfast);
    if (std::find(RecipeName.begin(), RecipeName.end(), planBreakfast) ==
        RecipeName.end()) {
        std::cout << "Wrong input" << std::endl;
        return;
    }

    std::cout << "Input lunch: ";
    std::getline(std::cin, planLunch);
    if (std::find(RecipeName.begin(), RecipeName.end(), planLunch) ==
        RecipeName.end()) {
        std::cout << "Wrong input" << std::endl;
        return;
    }

    std::cout << "Input dinner: ";
    std::getline(std::cin, planDinner);
    if (std::find(RecipeName.begin(), RecipeName.end(), planDinner) ==
        RecipeName.end()) {
        std::cout << "Wrong input" << std::endl;
        return;
    }
    // plan_id & date = NULL
    dbm.executeQuery(("INSERT INTO Plan VALUES(NULL, '" + planName +
                      "', NULL, '" + planBreakfast + "', '" + planLunch +
                      "', '" + planDinner + "');")
                         .c_str());
}
void addDatePlan() {
    std::string planDate;
    std::string planBreakfast;
    std::string planLunch;
    std::string planDinner;
    std::set<std::string> Date = getDates();
    RecipeName = recipe_db_.getRecipeNames();

    std::cout << "Input Date (YYYY-MM-DD): ";
    std::getline(std::cin, planDate);
    std::string today = Date::getToday();
    if (_date.compareDate(today, planDate)) {
        std::cout << "Enter Date After " + today << std::endl;
        return;
    }
    if (Date.find(planDate) != Date.end()) {
        std::cout << "Wrong input" << std::endl;
        return;
    }

    std::cout << "Input breakfast: ";
    std::getline(std::cin, planBreakfast);
    if (std::find(RecipeName.begin(), RecipeName.end(), planBreakfast) ==
        RecipeName.end()) {
        std::cout << "Wrong input" << std::endl;
        return;
    }

    std::cout << "Input lunch: ";
    std::getline(std::cin, planLunch);
    if (std::find(RecipeName.begin(), RecipeName.end(), planLunch) ==
        RecipeName.end()) {
        std::cout << "Wrong input" << std::endl;
        return;
    }

    std::cout << "Input dinner: ";
    std::getline(std::cin, planDinner);
    if (std::find(RecipeName.begin(), RecipeName.end(), planDinner) ==
        RecipeName.end()) {
        std::cout << "Wrong input" << std::endl;
        return;
    }

    // plan_id & name = NULL
    dbm.executeQuery(("INSERT INTO Plan VALUES(NULL, NULL, '" + planDate +
                      "', '" + planBreakfast + "', '" + planLunch + "', '" +
                      planDinner + "');")
                         .c_str());
    makeDatePlanGroceryList(planDate);
}

void deletePlan() {
    std::cout << "Select a Mode (1. Delete Date Plan, 2. Delete Name Plan): ";
    int selectNum;
    std::cin >> selectNum;
    std::cin.ignore();
```

```
389
390  void deletePlan() {
391      std::cout << "Select a Mode (1, Delete Date Plan, 2, Delete Name Plan): ";
392      int selectNum;
393      std::cin >> selectNum;
394      std::cin.ignore();
395      system("cls");
396
397      switch (selectNum) {
398          case 1:
399              deleteDatePlan();
400              break;
401          case 2:
402              deleteNamePlan();
403              break;
404          default:
405              std::cout << "Wrong Input" << std::endl;
406              break;
407      }
408  }
409
410  void deleteNamePlan() {
411      std::string planName;
412      std::set<std::string> Name = getNames();
413
414      std::cout << "Input Target Plan Name: ";
415      std::getline(std::cin, planName);
416      if (Name.find(planName) == Name.end()) {
417          std::cout << "Wrong Input" << std::endl;
418          Similarity similarity(planName);
419          similarity.checkSimilarity(1);
420          return;
421      }
422
423      dbm.executeQuery(
424          ("DELETE FROM plan WHERE name='" + planName + "';").c_str());
425  }
426
427  void deleteDatePlan() {
428      std::string planDate;
429      std::set<std::string> Date = getDates();
430
431      std::cout << "Input Target Plan Date (YYYY-MM-DD): ";
432      std::getline(std::cin, planDate);
433      if (Date.find(planDate) == Date.end()) {
434          std::cout << "Wrong Input" << std::endl;
435          return;
436      }
437
438      dbm.executeQuery(
439          ("DELETE FROM plan WHERE date='" + planDate + "';").c_str());
440      deleteDatePlanGroceryList(planDate);
441  }
442
443  void updatePlan() {
444      std::cout << "Select a Mode (1, Update Date Plan, 2, Update Name Plan): ";
445      int selectNum;
446      std::cin >> selectNum;
447      std::cin.ignore();
448      system("cls");
449
450      switch (selectNum) {
451          case 1:
452              updateDatePlan();
453              break;
454          case 2:
455              updateNamePlan();
456              break;
457          default:
458              std::cout << "Wrong Input" << std::endl;
459              break;
460      }
461  }
462
463  void updateDatePlan() {
464      std::string planDate;
465      std::string item;
466      std::string content;
467      std::set<std::string> Date = getDates();
468
469      std::cout << "Input Target Plan Date (YYYY-MM-DD): ";
470      std::getline(std::cin, planDate);
471      if (Date.find(planDate) == Date.end()) {
472          std::cout << "Wrong Input" << std::endl;
473          return;
474      }
475
476      std::cout << "Which item do you want to update? (date, breakfast, "
477                   "lunch, dinner): ";
478      std::getline(std::cin, item);
479      if (item != "date" && item != "breakfast" && item != "lunch" &&
480          item != "dinner") {
481          std::cout << "Wrong Input" << std::endl;
482          return;
483      }
484
485      std::cout << "What would you like to change the " + item + " to?: ";
486      std::getline(std::cin, content);
487      if (item == "date") {
488          if (Date.find(content) != Date.end()) {
489              std::cout << "Wrong Input" << std::endl;
490              return;
491          }
492      }
493
494      dbm.executeQuery(("UPDATE plan SET " + item + " = '" + content +
495                        "' WHERE date = '" + planDate + "';")
496                           .c_str());
497      deleteDatePlanGroceryList(planDate);
498      if (item == "date") {
499          makeDatePlanGroceryList(content);
500      } else {
501          makeDatePlanGroceryList(planDate);
```

```
                        .c_str());
    deleteDatePlanGroceryList(planDate);
    if (item == "date") {
      makeDatePlanGroceryList(content);
    } else {
      makeDatePlanGroceryList(planDate);
    }
  }

  void updateNamePlan() {
    std::string planName;
    std::string item;
    std::string content;
    std::set<std::string> Name = getNames();

    std::cout << "Input Target Plan Name: ";
    std::getline(std::cin, planName);
    if (Name.find(planName) == Name.end()) {
      Similarity similarity(planName);
      std::cout << "Wrong Input" << std::endl;
      similarity.checkSimilarity(1);
      return;
    }

    std::cout << "Which item do you want to update? (name, breakfast, "
                 "lunch, dinner): ";
    std::getline(std::cin, item);
    if (item != "name" && item != "breakfast" && item != "lunch" &&
        item != "dinner") {
      std::cout << "Wrong Input" << std::endl;
      return;
    }

    std::cout << "What would you like to change the " + item + " to?: ";
    std::getline(std::cin, content);
    if (item == "name") {
      if (Name.find(content) != Name.end()) {
        std::cout << "Wrong Input" << std::endl;
        return;
      }
    }

    dbm.executeQuery(("UPDATE plan SET " + item + " = '" + content +
                      "' WHERE name = '" + planName + "';")
                         .c_str());
  }

  void selectPeriodList() {
    std::string start_date;
    std::string end_date;

    std::cout << "Input Start Date (YYYY-MM-DD): ";
    std::getline(std::cin, start_date);
    std::cout << "Input End Date (YYYY-MM-DD): ";
    std::getline(std::cin, end_date);
    if (_date.compareDate(start_date, end_date)) {
      std::cout << "The order of the start and end dates is not correct. The "
                   "order has been changed automatically."
                << std::endl;
      swap(start_date, end_date);
    }

    std::vector<Plan> plans;
    dbm.executeQuery(("SELECT * FROM plan WHERE date BETWEEN '" + start_date +
                      "' AND '" + end_date + "' ORDER BY date ASC;")
                         .c_str(),
                     &plans, true);
    if (plans.empty()) {
      std::cout << "No Plan Between " + start_date + " and " + end_date
                << std::endl;
      return;
    }
    for (auto &plan : plans) {
      plan.printPlanDate();
    }

    std::cout << "Do you want to see a specific plan? [y/n]: ";
    char select;
    std::cin >> select;
    std::cin.ignore();
    if (select == 'y') {
      std::cout << "Input Plan Date: ";
      std::string plan;
      std::getline(std::cin, plan);
      selectPlanByDate(plan);
    }
  }

  void showGroceryList() {
    std::cout << "Select a Mode (1. Show Period Grocery List, 2. Show Specific "
                 "Date Grocery List): ";
    int selectNum;
    std::cin >> selectNum;
    std::cin.ignore();
    system("cls");

    switch (selectNum) {
      case 1:
        showPeriodGroceryList();
        break;
      case 2:
        showSpecificDateGroceryList();
        break;
      default:
        std::cout << "Wrong Input" << std::endl;
        break;
    }
  }
```

```
}

void showPeriodGroceryList() {
  std::string start_date;
  std::string end_date;
  std::set<std::string> temp;
  std::set<std::string> tempDate;

  std::cout << "Input Start Date (YYYY-MM-DD): ";
  std::getline(std::cin, start_date);
  std::cout << "Input End Date (YYYY-MM-DD): ";
  std::getline(std::cin, end_date);
  if (_date.compareDate(start_date, end_date)) {
    std::cout << "The order of the start and end dates is not correct, The "
                 "order has been changed automatically."
              << std::endl;
    swap(start_date, end_date);
  }

  dbm.executeQuery(
      ("SELECT date FROM plan WHERE date BETWEEN '" + start_date + "' AND '" +
       end_date + "' ORDER BY date ASC;")
          .c_str(),
      &tempDate, true);
  if (tempDate.empty()) {
    std::cout << "No Plan Between " + start_date + " and " + end_date
              << std::endl;
    return;
  }

  for (auto &i : tempDate) {
    std::set<std::string> tempIngredients;
    for (auto const &j : ingredients) {
      if (j.first == i) {
        tempIngredients = j.second;
      }
    }
    for (auto &j : tempIngredients) {
      temp.insert(j);
    }
  }

  std::cout << "-------------Grocery List-------------" << std::endl;
  for (const auto &i : temp) {
    std::cout << i << std::endl;
  }
  std::cout << "--------------------------------------" << std::endl;
}

void showSpecificDateGroceryList() {
  std::string planDate;
  std::set<std::string> Date = getDates();

  std::cout << "Input Target Plan Date (YYYY-MM-DD): ";
  std::getline(std::cin, planDate);
  if (Date.find(planDate) == Date.end()) {
    std::cout << "Wrong Input" << std::endl;
    return;
  }

  std::set<std::string> temp;
  for (auto const &i : ingredients) {
    if (i.first == planDate) {
      temp = i.second;
    }
  }

  std::cout << "-------------Grocery List-------------" << std::endl;
  for (const auto &i : temp) {
    std::cout << i << std::endl;
  }
  std::cout << "--------------------------------------" << std::endl;
}

void makeDatePlanGroceryList() {
  std::vector<Plan> plans;

  dbm.executeQuery("SELECT * FROM plan WHERE date IS NOT NULL;", &plans,
                   true);
  for (auto &plan : plans) {
    std::string planDate = plan.getDate();
    std::string breakfast = plan.getBreakfast();
    std::string lunch = plan.getLunch();
    std::string dinner = plan.getDinner();
    std::vector<Recipe> recipes;
    recipes.push_back(recipe_db_.selectRecipe(breakfast, true));
    recipes.push_back(recipe_db_.selectRecipe(lunch, true));
    recipes.push_back(recipe_db_.selectRecipe(dinner, true));

    std::set<std::string> tempIngredients;
    for (auto &recipe : recipes) {
      std::set<std::string> temp = recipe.getIngredients();
      for (auto &ingredient : temp) {
        tempIngredients.insert(ingredient);
      }
    }
    ingredients.push_back({planDate, tempIngredients});
  }
}

void makeDatePlanGroceryList(std::string const &planDate) {
  Plan plan;

  dbm.executeQuery(
      ("SELECT * FROM plan WHERE date = '" + planDate + "';").c_str(), &plan);
  std::string breakfast = plan.getBreakfast();
  std::string lunch = plan.getLunch();
```



```
            tempIngredients.insert(ingredient);
          }
        }
        ingredients.push_back({planDate, tempIngredients});
      }
    }

    void makeDatePlanGroceryList(std::string const &planDate) {
      Plan plan;

      dbm.executeQuery(
          ("SELECT * FROM plan WHERE date = '" + planDate + "';").c_str(), &plan);
      std::string breakfast = plan.getBreakfast();
      std::string lunch = plan.getLunch();
      std::string dinner = plan.getDinner();
      std::vector<Recipe> recipes;

      recipes.push_back(recipe_db_.selectRecipe(breakfast, true));
      recipes.push_back(recipe_db_.selectRecipe(lunch, true));
      recipes.push_back(recipe_db_.selectRecipe(dinner, true));
      std::set<std::string> tempIngredients;
      for (auto &recipe : recipes) {
        std::set<std::string> temp = recipe.getIngredients();
        for (auto &ingredient : temp) {
          tempIngredients.insert(ingredient);
        }
      }
      ingredients.push_back({planDate, tempIngredients});
    }

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

The PlanDB class was responsible for managing daily meal plans and related grocery lists. This class also provides a variety of capabilities and is implemented to store and retrieve plans and related information through the SQL database.

The PlanDB class contains various header files and member variables. Header files include Database Manager, RecipeDB, Date, Plan, Recipe, and Similarity, and important member variables include DatabaseManager objects for managing interactions with SQL databases, RecipeDB objects used to store and retrieve recipe-related information, std:set<std:string> and std:vector<std::pair<std:string, std:set<std:string>> member variables, and the class's creator was required to establish database connections and create a plan table. It also invoked the makeDatePlanGroceryList method to create a grocery list included in the initial plan and to search for the names and dates of available plans using the getName and getDates methods.

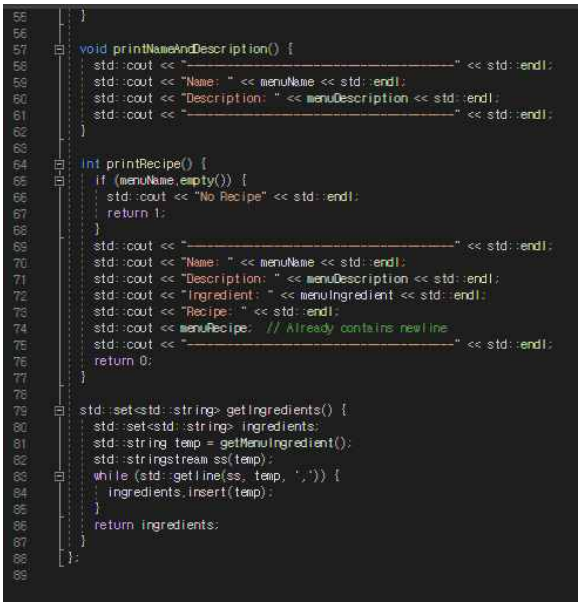
And the searchPlan, printAllPlanByDate, printAllPlanByName, selectPlanByDate, selectPlanByName, and selectPeriodList methods provide a variety of query and search options, addPlan, addDatePlan, addNamePlan, and addDatePlanUsingNamePlan methods to add new plans, store them in the database, delete plans through the deletePlan, deleteDatePlan, and deleteNamePlan methods, and updatePlan, updateDatePlan, and updatePlan, The and updateNamePlan methods have been used to enable you to update existing plans. The show GroceryList, showPeriodGroceryList, and showSpecificDateGroceryList methods display a list of ingredients needed

for a planned period or a specific date, and the makeDatePlanGroceryList method is used to generate a grocery list related to the date of the plan, which is created by combining ingredients from recipes included in the plan for that date and implemented to delete the grocery list for a specific date through the deleteDatePlanGroceryList method.

So the PlanDB class is a class that helps users add, look up, update, and delete daily meal plans and effectively manage related grocery lists, and it also enables them to maintain and utilize plans and related information through interaction with the database.

Recipe

```
#pragma once

#include <iostream>
#include <sstream>
#include <string>
#include <utility>
#include <vector>

class Recipe {
private:
  std::string menuName;
  std::string menuDescription;
  std::string menuIngredient;
  std::string menuRecipe;

public:
  Recipe() = default;

  std::string getMenuName() { return menuName; }
  std::string getMenuDescription() { return menuDescription; }
  std::string getMenuIngredient() { return menuIngredient; }
  std::string getMenuRecipe() { return menuRecipe; }

  void setMenuName(const std::string &name) { menuName = name; }
  void setMenuDescription(const std::string &description) {
    menuDescription = description;
  }
  void setMenuIngredient(const std::string &ingredient) {
    menuIngredient = ingredient;
  }
  void setMenuRecipe(const std::string &recipe) { menuRecipe = recipe; }

  void addRecipe() {
    std::cout << "Input Recipe Name: ";
    std::getline(std::cin, menuName);
    std::cout << "Input Recipe Description: ";
    std::getline(std::cin, menuDescription);
    std::cout << "Input Recipe Ingredient: ";
    std::getline(std::cin, menuIngredient);
    std::cout << "Input Recipe Method (To finish, just press Enter)"
              << std::endl;
    int index = 1;
    while (true) {
      std::cout << "Step " << index << ": ";
      std::string userInput;
      std::getline(std::cin, userInput);
      if (userInput.compare("") == 0) {
        break;
      }

      menuRecipe += std::to_string(index++) + ". ";
      menuRecipe += userInput;
      menuRecipe += "\n";
    }
  }

  void printNameAndDescription() {
    std::cout << "-----------------------------------" << std::endl;
    std::cout << "Name: " << menuName << std::endl;
```



```
    }

    void printNameAndDescription() {
        std::cout << "----------------------------------------" << std::endl;
        std::cout << "Name: " << menuName << std::endl;
        std::cout << "Description: " << menuDescription << std::endl;
        std::cout << "----------------------------------------" << std::endl;
    }

    int printRecipe() {
        if (menuName.empty()) {
            std::cout << "No Recipe" << std::endl;
            return 1;
        }
        std::cout << "----------------------------------------" << std::endl;
        std::cout << "Name: " << menuName << std::endl;
        std::cout << "Description: " << menuDescription << std::endl;
        std::cout << "Ingredient: " << menuIngredient << std::endl;
        std::cout << "Recipe: " << std::endl;
        std::cout << menuRecipe;  // Already contains newline
        std::cout << "----------------------------------------" << std::endl;
        return 0;
    }

    std::set<std::string> getIngredients() {
        std::set<std::string> ingredients;
        std::string temp = getMenuIngredient();
        std::stringstream ss(temp);
        while (std::getline(ss, temp, ',')) {
            ingredients.insert(temp);
        }
        return ingredients;
    }
};

```

The Recipe class is a class used to express and manage recipe information, and it has various member functions and member variables to manage the recipe's name, description, ingredients, and recipe methods and to receive this information from users.

First, menuName, menuDescription, menuIngredient, and menuRecipe implemented as four private variables, defined default constructors, and initialized member variables. and getMenuName(), getMenuDescription(), getMenuIngredient(), getMenuRecipe(), which is a Getter function that returns the recipe's name, description, ingredient, recipe method, setMenuName(conststd::string&name), setMenuDescription(conststd:string &description), setMenuIngredient(conststd:string &redient), and setMenuRecipe(conststd:string &recipe) are the names, descriptions, ingredients, Setter function to set recipe method.

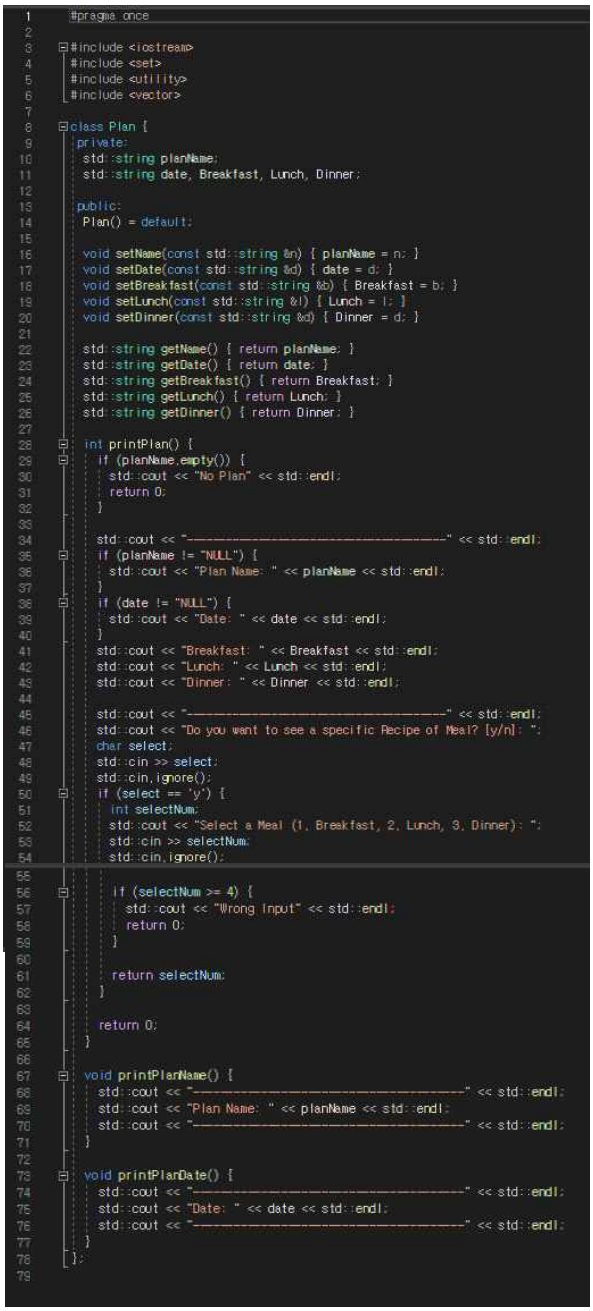
Then, you receive recipe information from the user with addRecipe() and set the value in the menuName, menuDescription, menuIngredient, menuRecipe member variables, where you can get the recipe method (process) in several steps, printNameAndDescription() to display the name and description of the recipe on the screen, and printRecipe() to display the entire recipe information on the screen. It displays all of the names, descriptions, ingredients, and recipe methods, and is implemented to output "No Recipe" if the recipe information is empty, function returns 1, otherwise function returns 0.

We also separated the materials stored in the menuIngredient string

using getIngredients() by a comma(), returning them in std::set<std::string> form, making the material list easy to manage.

As a result, the Recipe class received recipe information from the user, provided the ability to manage it, and implemented it to be used to add and manage recipe data with the RecipeDB class.

Plan



```cpp
#pragma once

#include <iostream>
#include <set>
#include <utility>
#include <vector>

class Plan {
private:
    std::string planName;
    std::string date, Breakfast, Lunch, Dinner;

public:
    Plan() = default;

    void setName(const std::string &n) { planName = n; }
    void setDate(const std::string &d) { date = d; }
    void setBreakfast(const std::string &b) { Breakfast = b; }
    void setLunch(const std::string &l) { Lunch = l; }
    void setDinner(const std::string &d) { Dinner = d; }

    std::string getName() { return planName; }
    std::string getDate() { return date; }
    std::string getBreakfast() { return Breakfast; }
    std::string getLunch() { return Lunch; }
    std::string getDinner() { return Dinner; }

    int printPlan() {
        if (planName.empty()) {
            std::cout << "No Plan" << std::endl;
            return 0;
        }

        std::cout << "-----------------------------------" << std::endl;
        if (planName != "NULL") {
            std::cout << "Plan Name: " << planName << std::endl;
        }
        if (date != "NULL") {
            std::cout << "Date: " << date << std::endl;
        }
        std::cout << "Breakfast: " << Breakfast << std::endl;
        std::cout << "Lunch: " << Lunch << std::endl;
        std::cout << "Dinner: " << Dinner << std::endl;

        std::cout << "-----------------------------------" << std::endl;
        std::cout << "Do you want to see a specific Recipe of Meal? [y/n]: ";
        char select;
        std::cin >> select;
        std::cin.ignore();
        if (select == 'y') {
            int selectNum;
            std::cout << "Select a Meal (1. Breakfast, 2. Lunch, 3. Dinner): ";
            std::cin >> selectNum;
            std::cin.ignore();

            if (selectNum >= 4) {
                std::cout << "Wrong Input" << std::endl;
                return 0;
            }

            return selectNum;
        }

        return 0;
    }

    void printPlanName() {
        std::cout << "-----------------------------------" << std::endl;
        std::cout << "Plan Name: " << planName << std::endl;
        std::cout << "-----------------------------------" << std::endl;
    }

    void printPlanDate() {
        std::cout << "-----------------------------------" << std::endl;
        std::cout << "Date: " << date << std::endl;
        std::cout << "-----------------------------------" << std::endl;
    }
};
```

The Plan class was implemented as a C++ class that stores and manages meal plan information. First, we implemented a three-member variable and a Plan() parameter to store the name of the meal plan, date (a string member variable to store the date of the meal plan), and Breakfast, Lunch, and Dinner (a string member variable to store the recipe name for breakfast, lunch, and dinner). A Plan() parameterless generator, we implemented the member variable to initialize it to its default value.

The Getter and Setter functions return the values of each member variable to the Getter function(), getDate(), getBreakfast(), getLunch(), getDinner(), and the Setter functions(setName(), setDate(), setBreakfast(), setLunch(), setDinner() setDinner() set the values for each member variable.

The printPlan function prints the entire meal plan information on the screen, outputs the name, date, breakfast, lunch, and dinner recipe name of the meal plan, asks the user if they want to see a recipe for a particular meal (morning, lunch, dinner), and returns one of 1, 2 (lunch), and 3 (dinner) depending on your choice. In addition, the printPlanName function was implemented to display only the name of the meal plan on the screen and the printPlanDate function to display only the date of the meal plan on the screen.

Date

```
#pragma once

#include <iostream>
#include <string>
#include <utility>

class Date {
public:
    Date() = default;

    bool compareDate(std::string const &date1, std::string const &date2) {
        int date1Year = std::stoi(date1.substr(0, 4));
        int date1Month = std::stoi(date1.substr(5, 2));
        int date1Day = std::stoi(date1.substr(8, 2));
        int date2Year = std::stoi(date2.substr(0, 4));
        int date2Month = std::stoi(date2.substr(5, 2));
        int date2Day = std::stoi(date2.substr(8, 2));

        if (date1Year > date2Year) {
            return true;
        } else if (date1Year == date2Year) {
            if (date1Month > date2Month) {
                return true;
            } else if (date1Month == date2Month) {
                if (date1Day > date2Day) {
                    return true;
                } else {
                    return false;
                }
            } else {
                return false;
            }
        } else {
            return false;
        }
    }

    static std::string getToday() {
        time_t now = time(nullptr);
        tm *ltm = localtime(&now);
        std::string year = std::to_string(1900 + ltm->tm_year);
        std::string month = std::to_string(1 + ltm->tm_mon);
        std::string day = std::to_string(ltm->tm_mday);

        if (month.length() == 1) {
            month = "0" + month;
        }

        if (day.length() == 1) {
            day = "0" + day;
        }

        return year + "-" + month + "-" + day;
    }
};
```

The Date class is a class that performs date-related actions, providing date-related utility functions such as date comparison and current date import, and implemented without the need to create instances of objects because there are no member variables inside the class and all functions are implemented as static functions.

Once the default constructor is defined, nothing is done. In the comparison date(), the comparison function takes two date strings (date1 and date2) and performs a date comparison, separates each date string into years, months, and days, converting them into integers, and comparing date1 and date2 to return true if the first date is a date in the future than the second date, otherwise it returns false, which handles the date comparison accurately by performing comparisons in the order of year, month, and day.

Also, in getToday(), the getToday function returns the current date as a string and uses the time and tm structures in the <ctime> header to get the current date, takes the current date information (year, month, day) and converts it into a string, and if the month and day are single digits, we add '0' to make it double digits, and implement the returned string in the form of "year-month-day".

As a result, the Date class can be used primarily when you need to obtain date comparisons or current date information, and a static function allows you to perform date-related tasks with a simple interface, making it easier to use date-related behavior in other classes or functions.

Similarity

```
#pragma once

#include <cstring>

#include "DatabaseManager.h"

class Similarity {
private:
 DatabaseManager dbm;
 std::string target;

public:
 Similarity(std::string target) : dbm("iikh.db") { this->target = target; }

 void checkSimilarity(int flag) {
  std::set<std::string> candidate;
  std::set<std::string> ret;

  if (flag == 1) {
   dbm.executeQuery("SELECT name FROM plan;", &candidate, true);
  } else if (flag == 2) {
   dbm.executeQuery("SELECT name FROM recipe;", &candidate, true);
  }

  for (auto& i : candidate) {
   if (strstr(i.c_str(), target.c_str())) {
    ret.insert(i);
   }
  }

  if (ret.size() == 0) {
   return;
  }

  std::cout << "Containing user input '" << target << "' ..." << std::endl;
  for (auto& i : ret) {
   std::cout << i + " " << std::endl;
  }
 }
};
```

The Similarity class is a class that searches and returns a string similar to a given string, and is implemented primarily for searching for items in the database.

The constructor receives a target string of type std::string as input, which initializes the Database Manager object, stores the text passed to the target member variable, and initializes the Database Manager object to perform database-related operations using a database file called "iikh.db". And since the checkSimilarity() function is the main method of checking for similarity, the checkSimilarity function specifies the database to be searched through the flag parameter, performs a search in the "plan" table if the flag value is 1, and in the "recipe" table if the flag value is 2, and takes the names of all items from that database table and stores them in a set of candidates, and then performs a substring search using the strstr function to compare the target string with each candidate string, and if the substring search contains the target string, add that candidate string to the result set ret, and finally, Implemented to output similar strings stored in ret.

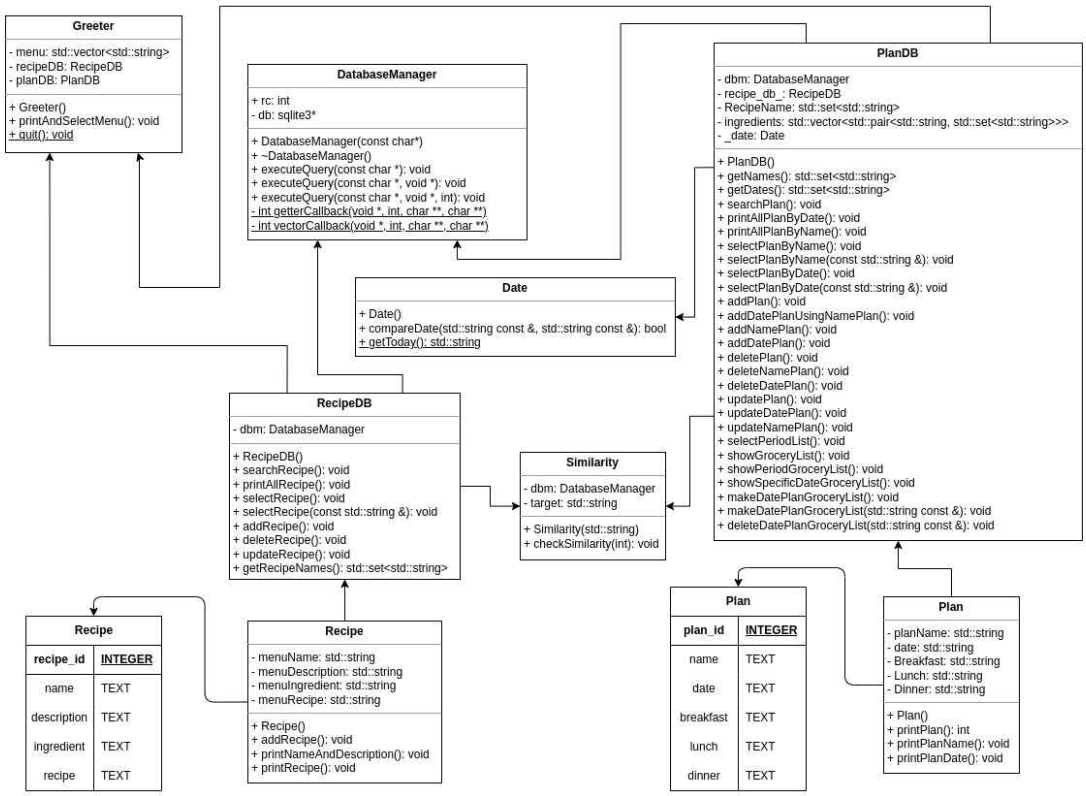
This class performs a name-based search in the database, where substrings search finds items similar to the target string, the database to be searched and the target string is specified through the generator, and the search results can be obtained by calling the checkSimilarity() function, thus implementing a similarity check and providing relevant information to the user.

Issue:

If the program is turned off and on again to detect duplicates in the recipe name or ingredient, the set is initialized, although it is in the DB. So, when selecting names or plans and using various functions under addition, there was a problem that there were some packs that should not be detected or entered duplicate because they were in the DB, but not here. So, to solve this problem, the recipe name brings the recipe DB side class and creates a new get recipe name function from the recipeDB, dragging it from the DB whenever necessary, and creating a set. In addition, ingredient was initially dragged from the DB from the planDB constructor, so that it was brought every time it was executed.

E

❖ The result of SW system design



This UML shows how the SW system of this IIKH is designed.

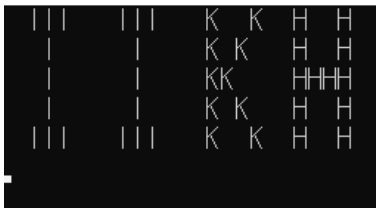
Relationship between classes

The Greeter class has RecipeDB and PlanDB objects that perform database-related tasks. RecipeDB and PlanDB classes also use Database Manager classes to perform database management tasks. And within the PlanDB class, you use the Plan class to deal with the meal plan data, and within the RecipeDB class to manage the recipe data.

F

❖ Execution results : show real examples of program execution. (use screen capture)

```
1. SEARCH RECIPE
2. ADD RECIPE
3. DELETE RECIPE
4. UPDATE RECIPE
5. SEARCH PLAN
6. ADD PLAN
7. DELETE PLAN
8. UPDATE PLAN
9. SHOW GROCERY LIST
10. QUIT
Select Menu:
```



This is the title that comes up before the menu comes up.

When user run the code, this screen appears.

```
Selected Menu: SEARCH RECIPE
Select a Mode (1. Print All, 2. Select):
```

when user select 1, this screen appears.

```
Selected Menu: ADD RECIPE
Input Recipe Name: fried rice
```

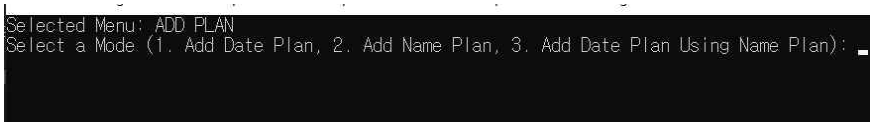
First, let's select 2 as an example and add a recipe (additional process)

```
All Recipe
----------------------------------------
Name: fried rice
Description: Fried rice is one of the most popular Korean dishes, and you can make it quickly and control the taste by u
tilizing various ingredients and spices.
----------------------------------------
Do you want to see a specific recipe? [y/n]:
```

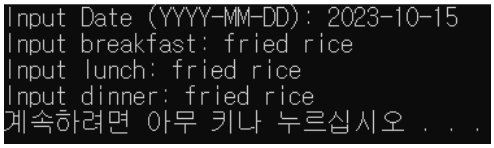
After that, if I choose print All by selecting 1, it will appear like this. and then You can choose between y/n and choose whether you want to see a special recipe or not.

```
Input Recipe Name: fried rice
----------------------------------------
Name: fried rice
Description: Fried rice is one of the most popular Korean dishes, and you can make it quickly and control the taste by utilizing various ingredients and spices.
Ingredient: Rice, chicken breast, pork, shrimp, onion, carrot, green onion, starch
Recipe:
1. Ingredients: Chop chicken brisket, pork, shrimp or tofu, and chop onions, carrots, and green onions. Keep the rice cold in the refrigerator in advance.
2. Preheat the pan/frying pan: Preheat the pan or frying pan over a medium heat. Add a little olive oil or sesame oil and heat.
3. Stir-fry meat: Stir-fry meat in a pan. Stir-fry until the meat is cooked and transfer to another plate.
4. Stir-fry vegetables: Stir-fry onions and carrots in the same pan. Stir-fry onion and carrot until smooth. Add minced garlic and stir-fry.
5. Add the rice: Put the frozen rice in the pan and whisk well. Stir-fry rice, vegetables, and meat until well combined. If you use starch, the rice becomes thicker and crispy.
6. Seasoning: Soy sauce, salt, pepper, red pepper paste (optional), and stir-fry again. You can adjust the soy sauce and red pepper paste to season.
7. Add spring onion and sesame salt: Add chopped spring onion and sesame salt to the fried rice to finish.
8. Plate: Place fried rice on a plate and sprinkle with thick seaweed or sesame salt to complete.
----------------------------------------
계속하려면 아무 키나 누르십시오 . . .
```

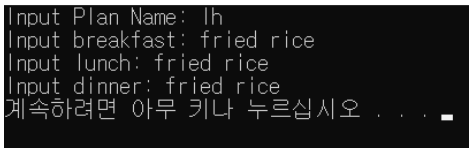
In this case, this is the data found by entering search receipt 1 and selecting number 2 and typing the name of the receipt.



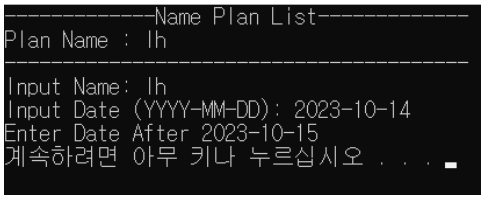
This is the result window when you choose 6



If you choose number 1, you can add it like this.



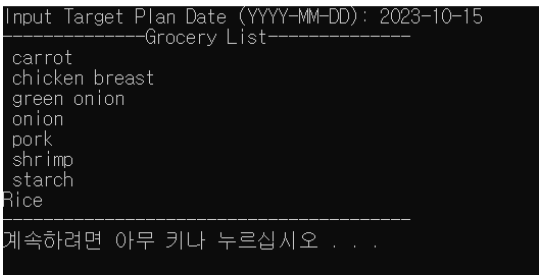
If you choose number 2, you can add it like this.



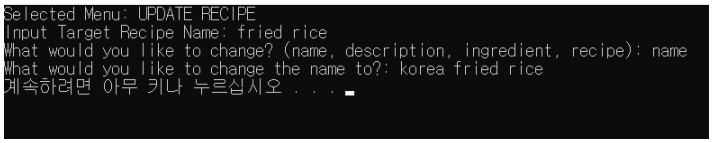
If you choose number 3, you can add it like this

If you choose number 9 on the first screen, this window appears

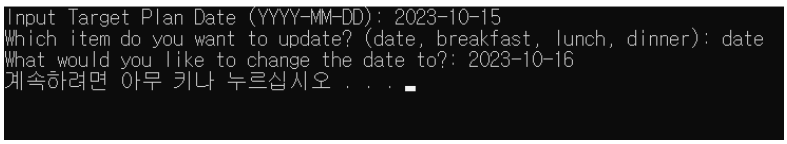
If you choose number 1, you can check the food you need within that period



If you choose number 2, you can see the ingredients you need for that date.



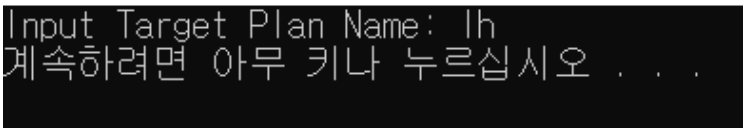
This is the result screen that you are changing by selecting Update recipe. You can change names, materials, etc., but only the name was changed in the example.



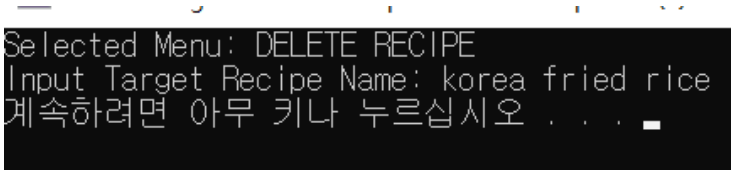
This is the result screen that you are changing by selecting Update plan. You can change the date, breakfast, etc., but only the name was changed in the example.

This is the result window when you select Delete plan.

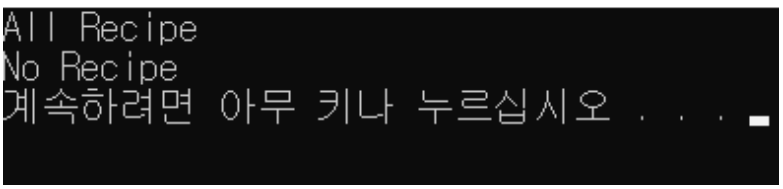
This is the result screen where you select Delete plan to erase the plan. I chose number 1 for the erasing method. As you can see in the example, I changed the name of the recipe before, so I wrote the name with the changed name.



This is the result window when you select Delete plan and choose number 2. The process of entering and erasing the name of the plan added in the example above.



This is the window that comes up when you select Delete recipe. In the example above, you can see that you typed the name you changed, but if you don't type the name you changed, it says wrong and ends.



After that, if you go back to Search recipe and select Print All, it says it's not there as you can see.

G

❖ explain how you applied object oriented concepts to the development for your project. also explain what you felt and learned from the project.

- 김경민(20224680)

I'm a project manager at IIKH. One of the most challenging aspects of this project is HRM (Human Resource Management). The uneven allocation of individual tasks led to a lot of complaints from some team members. This has happened and I am very sorry for them as a team leader. Based on various suggestions and suggestions, the Final Project would like to make more efforts to resolve this point so that the project can proceed smoothly.

The conditions for good code vary. This also depends on the different goals of the project. But I believe that good code comes from a well-defined convention. We used Git, DVCS, for version control of the code. At this time, we wanted to improve the efficiency of work by using branches for feature development and branches that are used stably before the final release, rather than working on only one main branch. In addition, since each code writing style is different, Google's style guide was applied to reduce conflicts when performing merges and to maintain consistency in the way the code is expressed.

Nevertheless, when I finished the project, I saw some shortcomings. Some of the commits pushed contain syntax errors or code that does not guarantee normal operation. At the beginning of the project, we didn't feel that need to automate this process, but as the project progressed and the number of features we needed to support increased rapidly and it became overwhelming to spend time on such simple labor. Therefore, in order to improve this, we want to configure a CI/CD pipeline during the final project to minimize human error and waste of human resources, and to automate tasks such as build and testing.

- 김제신(20225779)

In the process of implementing IIKH, I was able to learn how to use external libraries, how to implement the overall program structure, how to apply code styles for collaboration, and how to use Git. Since it is a team project, it is necessary to design a program on a larger scale than the individual, but since object orientation is not familiar by myself, it is unfortunate that the structure of the class talked about before the start of the team project could not be followed, and the structure was changed. So in the final project, I would like to make up for these deficiencies and design a program with a better structure.

- 김주영(20223908)

I don't have much experience in team projects, so I think I was inexperienced in communicating and collaborating. In particular, as the team project grew in size, the code became more complex and it was difficult to grasp the relationship with each other. Through this experience, I was able to experience what I lacked and I will try to make up for this.

- 박호근(20202203)

I wasn't familiar with Team Play itself, so I think I struggled a lot with writing code. When I created a feature, function, or class, it was hard to make it available to other team members. I will use this experience as a lesson to improve in the next team.

- 배정환(20200956)

It was an experience where I was learn a lot of important things about a project of my own size. I've learned a lot about how important it is to read and use other people's code rather than writing my own. I was also vaguely learned about the existence of various criteria for collaboration. In my next project, I will try to implement and understand more of it.

- 서규민(20225679)

While doing object-oriented teamwork, I learned how difficult sharing codes with others and combining them together, and I felt once again how difficult it was to collaborate. And it was nice to actually experience object-oriented programming through this project. I also found it difficult to interpret other people's codes it made me realize that I still lacked a lot, and during the final project, I will try harder to understand the code faster and more concisely.

- 조주원(20214782)

I understood the concept of object-orientation while using object-oriented language, but I think I was experience it better while working on a full-fledged team project. It was also a great opportunity to experience first-hand the conditions for a good program (such as readability, extendability) that we had only learned about through text through collaboration.

H

❖ Conclusion

This IIKH is part of a console application that interacts with users and recipes or meal plans. Users can manage recipes and set up or search for meal plans. This is a simple application, focused on providing users with recipe management and meal planning capabilities.