Basics of Programming 1

Name: Tóbiás Máté Levente
Neptun:
VV5EPA
Date:
2023.11.26

Documentation

Table of Contents

1. User Documentation

1.1 Recipe Book Management System

The Recipe Book Management System is a user-friendly and efficient tool for managing recipes, ingredients in the depot, and cooking logs. It simplifies the process of meal planning and ensures that users can make the most of their available ingredients while minimizing food waste.

1.2 User Interface:

1. View Recipes: Display a list of available recipes.
2. Add Recipe: Add a new recipe to the system.
3. Delete Recipe: Delete an existing recipe.
4. View Ingredients: Display a list of available ingredients.
5. Add Ingredient: Add a new ingredient to the system.
6. View Storage: Display the current ingredient inventory in storage.
7. Add to Storage: Add ingredients to storage.
8. Suggest Random Recipe: Suggest a random recipes based on available ingredients.
9. Suggest Recipe by Ingredients: Suggest recipes based on selected ingredients.
10. Suggest Recipe by Expiry Date: Suggest recipes based on ingredients nearing their expiration dates.
11. Select and cook recipe: The user selected the recipe that want to cook, enter the recipe name, and this function decrease the amount of the ingredients in the depot and log the cook event.
12. Logs: See the cooked meals, added or deleted recipes and added ingredients.
13. Exit: Quit the application.

1.3 Input limits:

1.3.1 Character limits:

Recipe name maximum word limit: 50 character.
Recipe Description maximum word limit: 5000 character.
Ingredient name maximum word limit: 50 character.
Ingredient Unit maximum word limit: 10 character.

If more characters are entered by the user than allowed, the program will warn you about this, cancel the started task and throw you back to the menu.

1.3.2 Number limit:

In the menu you can choose numbers from 1 to 13.
- If the user enters a different number, the program indicates this and prompts them to try again.

- If the user writes a rational number instead of an integer, the integer before the decimal point is processed, and the program may behave differently depending on the number. If this number is from 1 to 13, then the program will execute the instruction associated with it, if not, the program will indicate this and ask you to try again.

- If the user enters not a number, but another character, then the program exits.

1.3.3 Date limit:

When entering the date, the following must be true:
- The day should be from 1 to 31.
- The month must be from 1 to 12.

If the user specifies a later day than the recording date, the program will indicate that the given ingredient has already expired

2. Technical documentation

2.1 List of required files:

C – files:

Backend.c

Connection.c

FileRead_write.c

Ingredients.c

Logs.c

main.c

Recipes.c

Storage.c

H – files:

debugmalloc.h

functions.h

structs.h

csv – files:

connection.csv

ingredients.csv

logs.csv

recipes.csv

storage.csv

2.2 Compiler environment

- gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

  Flags:

  - Wall

  -Werror

2.3 Data Storage:

The Recipe Book Management System stores data in CSV (Comma-Separated Values) files to manage recipes, ingredients, storage, connections, and logs. These files are not editable directly by the user. Export and import files are not supported. Here are the details for each CSV file:

• Recipe File (recipes.csv):

  ◦ ID (Primary Key): A unique identifier for each recipe.

  ◦ Name: The name or title of the recipe.

  ◦ Description: How the dish should be cooked.

• Ingredients File (ingredients.csv):

  ◦ ID (Primary Key): A unique identifier for each ingredient.

  ◦ Name: The name of the ingredient.

  ◦ Unit: The unit of measurement for the ingredient.

• Storage of the available ingredients file (storage.csv):

  ◦ ID (Foreign Key): A reference to the corresponding ingredient.

  ◦ Amount: The amount of the ingredient.

  ◦ Expiration Date: The expiration date of the ingredient.

  In this file, could be the same ingredient multiple times.

• Connections File (connections.csv):

  ◦ Recipe_ID (Foreign Key): A reference to the corresponding recipe.

  ◦ Ingredient_ID (Foreign Key): A reference to the corresponding ingredient.

  ◦ Amount: The amount of the ingredient in the recipe.

• Log File (logs.csv):

  ◦ Date: The date when the log entry was recorded.

  ◦ ID: A reference to either a recipe or an ingredient based on the context of the log entry.

  ◦ Operation Description: A string describing the operation performed, such as adding, deleting, cooking recipe and adding ingredient.

Operations: - A : Add

- D: Delete

- C : Cooked

The system reads and writes data to these CSV files to manage recipes, ingredients, storage, connections, and logs. The use of primary and foreign keys ensures data integrity and consistency, and the log file records operations performed within the system for tracking and auditing purposes.

Datatypes:

ID: <string>

Name:<string>

Unit:<string>

Amount:<double>

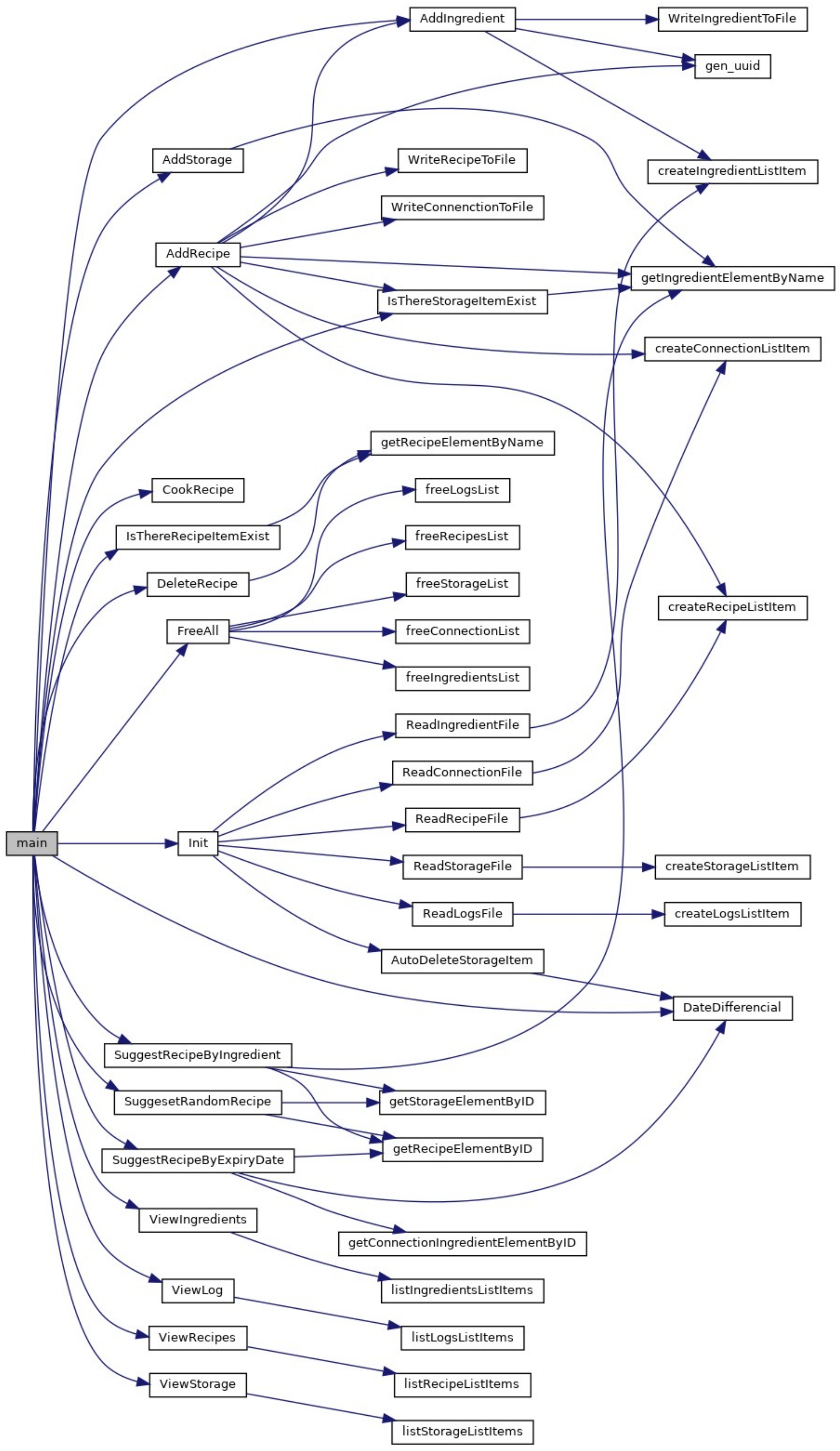
Date: <struct> → day: <int>, month: <int> , year:<int>

Recipe_ID: <string>

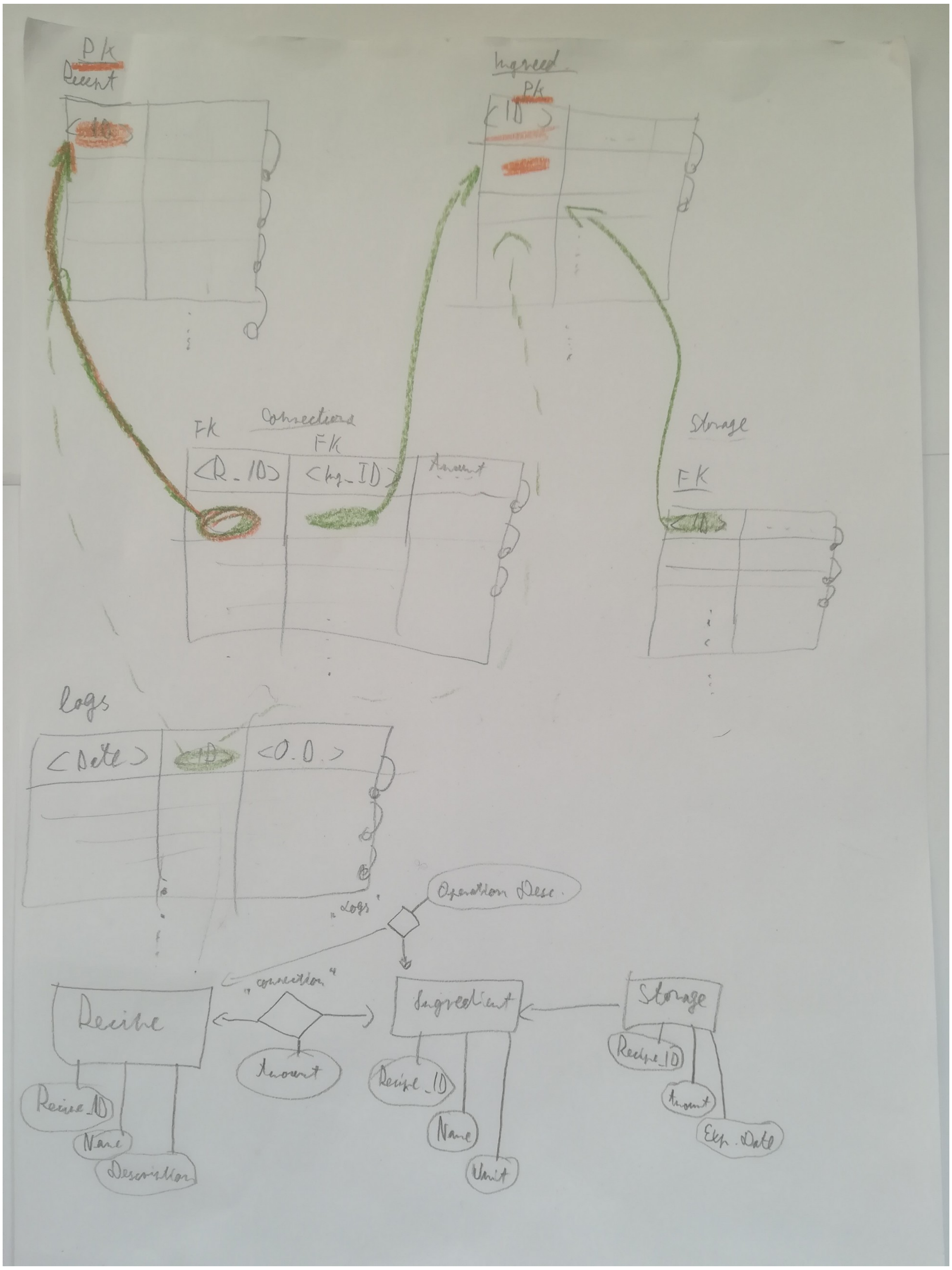
Ingredient_ID: <string>

Operation Description: <string>

2.4 Function call graph:



2.4.1 Tabels Connection



2.5 Functions Descriptions :

2.5.1 Recipes.c functions:

extern void createRecipeListItem(Recipe * R_Recipe, Recipe * R_ptr);

Description: Creates a new Recipe list item and appends it to the end of the list.

Error Cases: Exits the program if memory allocation fails.

Input arguments: R_Recipe - Pointer to the Recipe structure to be added.

R_ptr - Pointer to the head of the Recipe list.

Return type: void

Memory usage: Allocates memory for a new Recipe structure.

Big O notation: O(n), where n is the number of elements in the Recipe list

extern void deleteRecipeListItem(Recipe *R_Recipe, char c_searchedID[]);

Description: Deletes a Recipe list item with a specified recipe ID.

Error Cases: None.

Input arguments: R_Recipe - Pointer to the head of the Recipe list.

c_searchedID - Recipe ID to search for and delete.

Return type: void

Memory usage: Frees memory for the deleted Recipe structure.

Big O notation: O(n), where n is the number of elements in the Recipe list.

extern void listRecipeListItems(Recipe *R_Recipe);

Description: Lists the names of each Recipe list item.

Error Cases: None.

Input arguments: R_Recipe - Pointer to the head of the Recipe list.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Recipe list.

extern void freeRecipesList(Recipe *R_ptr);

Description: Frees the memory allocated for the entire Recipe list.

Error Cases: None.

Input arguments: R_ptr - Pointer to the head of the Recipe list.

Return type: void

Memory usage: Frees memory for all Recipe structures in the list.

Big O notation: O(n), where n is the number of elements in the Recipe list.

extern Recipe* getRecipeElementByID(Recipe *R_Recipe, char c_searchedID[]);

Description: Retrieves a Recipe list item with a specified recipe ID.

Error Cases: None.

Input arguments: R_Recipe - Pointer to the head of the Recipe list.

c_searchedID - Recipe ID to search for.

Return type: Pointer to the found Recipe structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Recipe list.

extern Recipe* getRecipeElementByName(Recipe *R_Recipe, char c_searchedName[]);

Description: Retrieves a Recipe list item with a specified recipe name.

Error Cases: None.

Input arguments: R_Recipe - Pointer to the head of the Recipe list.

c_searchedName - Recipe name to search for.

Return type: Pointer to the found Recipe structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Recipe list.

2.5.2 Ingredients.c Functions:

extern void createIngredientListItem(Ingredient * I_Ingredient, Ingredient *I_ptr) ;

Description: Creates a new Ingredient list item and appends it to the end of the list.

Error Cases: Exits the program if memory allocation fails.

Input arguments: I_Ingredient - Pointer to the Ingredient structure to be added.

I_ptr - Pointer to the head of the Ingredient list.

Return type: void

Memory usage: Allocates memory for a new Ingredient structure.

Big O notation: O(n), where n is the number of elements in the Ingredient list.

extern void deleteIngredientListItem(Ingredient *I_Ingredient, char c_searchedID[]);

Description: Deletes an Ingredient list item with a specified ingredient ID.

Error Cases: None.

Input arguments: I_Ingredient - Pointer to the head of the Ingredient list.

c_searchedID - Ingredient ID to search for and delete.

Return type: void

Memory usage: Frees memory for the deleted Ingredient structure.

Big O notation: O(n), where n is the number of elements in the Ingredient list.

extern void listIngredientsListItems(Ingredient *I_Ingredient);

Description: Lists the names of each Ingredient list item.

Error Cases: None.

Input arguments: I_Ingredient - Pointer to the head of the Ingredient list.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Ingredient list.

extern void freeIngredientsList(Ingredient *I_ptr);

Description: Frees the memory allocated for the entire Ingredient list.

Error Cases: None.

Input arguments: I_ptr - Pointer to the head of the Ingredient list.

Return type: void

Memory usage: Frees memory for all Ingredient structures in the list.

Big O notation: O(n), where n is the number of elements in the Ingredient list.

extern Ingredient* getIngredientElementByID(Ingredient *I_Ingredient, char c_searchedID[]);

Description: Retrieves an Ingredient list item with a specified ingredient ID.

Error Cases: None.

Input arguments: I_Ingredient - Pointer to the head of the Ingredient list.

c_searchedID - Ingredient ID to search for.

Return type: Pointer to the found Ingredient structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Ingredient list.

extern Ingredient* getIngredientElementByName(Ingredient *I_Ingredient, char c_searchedName[]);

Description: Retrieves an Ingredient list item with a specified ingredient name.

Error Cases: None.

Input arguments: I_Ingredient - Pointer to the head of the Ingredient list.

c_searchedName - Ingredient name to search for.

Return type: Pointer to the found Ingredient structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Ingredient list.

2.5.3 Storage.c Functions:

extern void createStorageListItem(Storage * S_StoredItemInStorage, Storage *S_ptr);

Description: Creates a new Storage list item and appends it to the end of the list.

Error Cases: Exits the program if memory allocation fails.

Input arguments: S_StoredItemInStorage - Pointer to the Storage structure to be added.

S_ptr - Pointer to the head of the Storage list.

Return type: void

Memory usage: Allocates memory for a new Storage structure.

Big O notation: O(n), where n is the number of elements in the Storage list.

extern void deleteStorageListItem(Storage *S_Storage, char c_searchedID[]);

Description: Deletes a Storage list item with a specified ingredient ID.

Error Cases: None.

Input arguments: S_Storage - Pointer to the head of the Storage list.

c_searchedID - Ingredient ID to search for and delete.

Return type: void

Memory usage: Frees memory for the deleted Storage structure.

Big O notation: O(n), where n is the number of elements in the Storage list.

extern void listStorageListItems(Storage *S_Storage);

Description: Lists the amount in storage for each Storage list item.

Error Cases: None.

Input arguments: S_Storage - Pointer to the head of the Storage list.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Storage list.

extern void freeStorageList(Storage *S_ptr);

Description: Frees the memory allocated for the entire Storage list.

Error Cases: None.

Input arguments: S_ptr - Pointer to the head of the Storage list.

Return type: void

Memory usage: Frees memory for all Storage structures in the list.

Big O notation: O(n), where n is the number of elements in the Storage list.

extern Storage* getStorageElementByID(Storage *S_Storage, char c_searchedID[]);

Description: Retrieves a Storage list item with a specified ingredient ID.

Error Cases: None.

Input arguments: S_Storage - Pointer to the head of the Storage list.

c_searchedID - Ingredient ID to search for.

Return type: Pointer to the found Storage structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Storage list.

2.5.4 Connection.c Functions:

extern void createConnectionListItem(Connection * C_Connection, Connection * C_ptr);

Description: Creates a new Connection list item and appends it to the end of the list.

Error Cases: Exits the program if memory allocation fails.

Input arguments: C_Connection - Pointer to the Connection structure to be added.

C_ptr - Pointer to the head of the Connection list.

Return type: void

Memory usage: Allocates memory for a new Connection structure.

Big O notation: O(n), where n is the number of elements in the Connection list.

extern int numbersOfSeachedConnectionItem(Connection *C_Connection, char c_searchedID[]);

Description: Counts the number of Connection list items with a specified recipe ID.

Error Cases: None.

Input arguments: C_Connection - Pointer to the head of the Connection list.

c_searchedID - Recipe ID to search for.

Return type: int - Number of Connection items with the specified recipe ID.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Connection list.

extern void deleteConnectionListItem(Connection *C_Connection, char c_searchedID[]);

Description: Deletes Connection list items with a specified recipe ID.

Error Cases: None.

Input arguments: C_Connection - Pointer to the head of the Connection list.

c_searchedID - Recipe ID to search for and delete.

Return type: void

Memory usage: Frees memory for the deleted Connection structures.

Big O notation: O(n^2), where n is the number of elements in the Connection list.

extern void listConnectionListItems(Connection *C_Connection);

Description: Lists the amounts for each Connection list item.

Error Cases: None.

Input arguments: C_Connection - Pointer to the head of the Connection list.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Connection list.

extern void freeConnectionList(Connection *C_ptr);

Description: Frees the memory allocated for the entire Connection list.

Error Cases: None.

Input arguments: C_ptr - Pointer to the head of the Connection list.

Return type: void

Memory usage: Frees memory for all Connection structures in the list.

Big O notation: O(n), where n is the number of elements in the Connection list.

extern Connection* getConnectionRecipeElementByID(Connection *C_Connection, char c_searchedID[]);

Description: Retrieves a Connection list item with a specified recipe ID.

Error Cases: None.

Input arguments: C_Connection - Pointer to the head of the Connection list.

c_searchedID - Recipe ID to search for.

Return type: Pointer to the found Connection structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Connection list.

extern Connection* getConnectionIngredientElementByID(Connection *C_Connection, char c_searchedID[]);

Description: Retrieves a Connection list item with a specified ingredient ID.

Error Cases: None.

Input arguments: C_Connection - Pointer to the head of the Connection list.

c_searchedID - Ingredient ID to search for.

Return type: Pointer to the found Connection structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Connection list.

2.5.5 Logs.c Functions :

extern void createLogsListItem(Logs * L_Log, Logs *L_ptr);

Description: Creates a new Logs list item and appends it to the end of the list.

Error Cases: Exits the program if memory allocation fails.

Input arguments: L_Log - Pointer to the Logs structure to be added.

L_ptr - Pointer to the head of the Logs list.

Return type: void

Memory usage: Allocates memory for a new Logs structure.

Big O notation: O(n), where n is the number of elements in the Logs list.

extern void deleteLogListItem(Logs *L_Log, char c_searchedID[]);

Description: Deletes a Logs list item with a specified log ID.

Error Cases: None.

Input arguments: L_Log - Pointer to the head of the Logs list.

c_searchedID - Log ID to search for and delete.

Return type: void

Memory usage: Frees memory for the deleted Logs structure.

Big O notation: O(n), where n is the number of elements in the Logs list.

extern void listLogsListItems(Logs *L_Log);

Description: Lists the operation descriptions for each Logs list item.

Error Cases: None.

Input arguments: L_Log - Pointer to the head of the Logs list.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Logs list.

extern void freeLogsList(Logs *L_ptr);

Description: Frees the memory allocated for the entire Logs list.

Error Cases: None.

Input arguments: L_ptr - Pointer to the head of the Logs list.

Return type: void

Memory usage: Frees memory for all Logs structures in the list.

Big O notation: O(n), where n is the number of elements in the Logs list.

extern Logs* getLogElementByID(Logs *L_Log, char c_searchedID[]);

Description: Retrieves a Logs list item with a specified log ID.

Error Cases: None.

Input arguments: L_Log - Pointer to the head of the Logs list.

c_searchedID - Log ID to search for.

Return type: Pointer to the found Logs structure or NULL if not found.

Memory usage: No dynamic memory allocation.

Big O notation: O(n), where n is the number of elements in the Logs list.

2.5.6 Backend.c Functions:

extern void Init(void);

Description: Initializes the system by setting up the random seed, reading recipe, ingredient, storage, connection, and logs files.

Error Cases: None.

Input arguments: None.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the size of the data being read from files.

extern char* Generate_uuid(void);

Description: Generates a random UUID.

Error Cases: None.

Input arguments: None.

Return type: char* (pointer to a static buffer containing the UUID).

Memory usage: Uses a static buffer.

Big O notation: O(1).

static void getDate(Date *D_tmp_Date);

Description: Gets the current date.

Error Cases: None.

Input arguments: D_tmp_Date - a pointer to the Date structure to store the current date.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(1).

static void strcpyDate(Date *D_from, Date *D_to);

Description: Copies the values of a source date to a destination date.

Error Cases: None.

Input arguments: D_from - pointer to the source Date structure, D_to - pointer to the destination Date structure.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(1).

extern int DateDifferencial(Date D_date);

Description: Calculates the difference in days between the given date and the current date.

Error Cases: None.

Input arguments: D_date - Date structure representing the target date.

Return type: int - the difference in days.

Memory usage: No additional memory is allocated.

Big O notation: O(1).

extern void ViewRecipes(void);

Description: Displays the list of recipes.

Error Cases: None.

Input arguments: None.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of recipes.

extern void AddRecipe(char c_Recipe_Name[], char c_Recipe_Description[]);

Description: Adds a new recipe with specified name and description, along with required ingredients.

Error Cases: Handles memory allocation errors during ingredient input.

Input arguments: c_Recipe_Name - name of the recipe,

c_Recipe_Description - description of the recipe.

Return type: void.

Memory usage: Allocates memory for temporary connection array.

Big O notation: O(N), where N is the number of ingredients for the recipe.

extern void DeleteRecipe(char c_Recipe_Name[]);

Description: Deletes a recipe with the specified name.

Error Cases: None.

Input arguments: c_Recipe_Name - name of the recipe to be deleted.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of recipes.

extern void ViewIngredients(void);

Description: Displays the list of ingredients.

Error Cases: None.

Input arguments: None.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of ingredients.

extern void AddIngredient(char c_Inredient_Name[], char c_Ingredient_Unit[]);

Description: Adds a new ingredient with specified name and unit.

Error Cases: Handles memory allocation errors.

Input arguments: c_Inredient_Name - name of the ingredient,

c_Ingredient_Unit - unit of the ingredient.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(1).

extern void ViewStorage(void);

Description: Displays the list of storage items.

Error Cases: None.

Input arguments: None.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of storage items.

extern void AddConnection(char c_Recipe_Name[],char c_Ingredient_Name[],double d_amount);

Description: Adds a new connection between a recipe and an ingredient with a specified amount.

Error Cases: None.

Input arguments: c_Recipe_Name - name of the recipe,

c_Ingredient_Name - name of the ingredient,

d_amount - amount required for the recipe.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(1).

extern void AddStorage(char c_Storage_Name[], double amount, Date *Exp_Date);

Description: Adds a new storage item with a specified name, amount, and expiration date.

Error Cases: None.

Input arguments: c_Storage_Name - name of the storage item,

amount - amount in storage,

Exp_Date - pointer to the expiration date.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(1).

extern bool IsThereStorageItemExist(char c_Ingredient_Name[]);

Description: Checks if a storage item with the given ingredient name exists.

Error Cases: None.

Input arguments: c_Ingredient_Name - name of the ingredient.

Return type: bool - true if the storage item exists, false otherwise.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of storage items.

extern bool IsThereRecipeItemExist(char c_Recipe_Name[]);

Description: Checks if a recipe with the given name exists.

Error Cases: None.

Input arguments: c_Recipe_Name - name of the recipe.

Return type: bool - true if the recipe exists, false otherwise.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of recipes.

static void RestoreStorageItemsAmount(char c_Recipe_Name[], Connection* C_end_ptr, Connection *C_Connection_ptr);

Description: Restores the amounts of storage items used by a recipe in case of insufficient quantity.

Error Cases: None.

Input arguments: c_Recipe_Name - name of the recipe,

C_end_ptr - end pointer of the connection list,

C_Connection_ptr - pointer to the connection list.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of connections.

static void SelectAndCookRecipe(char c_Recipe_Name[], Connection *C_Connection_ptr, Logs *L_ptr);

Description: Selects and cooks a recipe by updating storage quantities and creating a log entry.

Error Cases: Handles insufficient quantity of ingredients.

Input arguments: c_Recipe_Name - name of the recipe,

C_Connection_ptr - pointer to the connection list,

L_ptr - pointer to the logs list.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of connections.

extern void CookRecipe(char c_Recipe_Name[]);

Description: Cooks a recipe by calling the necessary functions.

Error Cases: Handles insufficient quantity of ingredients.

Input arguments: c_Recipe_Name - name of the recipe.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of connections.

extern void SuggestRecipeByIngredient(char c_Ingredient_Name[]);

Description: Suggests recipes that can be made using a specific ingredient.

Error Cases: None.

Input arguments: c_Ingredient_Name - name of the ingredient.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of connections.

extern void SuggesetRandomRecipe(void);

Description: Suggests recipes that can be made using a specific ingredient.

Error Cases: None.

Input arguments: c_Ingredient_Name - name of the ingredient.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of connections.

extern void SuggestRecipeByExpiryDate(int i_Differencial_Day);

Description: Suggests recipes with ingredients about to expire within a specified number of days.

Error Cases: None.

Input arguments: i_Differencial_Day - the number of days within which the ingredients will expire.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of storage items.

extern void ViewLog(void);

Description: Displays the list of logs.

Error Cases: None.

Input arguments: None.

Return type: void.

Memory usage: No additional memory is allocated.

Big O notation: O(N), where N is the number of logs.

extern void FreeAll(void);

Description: Frees the memory allocated for all lists.

Error Cases: None.

Input arguments: None.

Return type: void.

Memory usage: Frees the memory allocated for recipes, ingredients, connections, storage, and logs.

Big O notation: O(N), where N is the total number of elements in all lists.

2.5.7 FileRead_write.c Functions:

extern void ReadIngredientFile(Ingredient *I_ptr);

Description: Reads ingredient data from the "ingredients.csv" file and creates an ingredient list.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: I_ptr - Pointer to the Ingredient list.

Return type: void

Memory usage: Allocates memory for temporary Ingredient structures.

Big O notation: O(n), where n is the number of ingredients in the file.

extern void ReadRecipeFile(Recipe *R_ptr);

Description: Reads recipe data from the "recipes.csv" file and creates a recipe list.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: R_ptr - Pointer to the Recipe list.

Return type: void

Memory usage: Allocates memory for temporary Recipe structures.

Big O notation: O(n), where n is the number of recipes in the file.

extern void ReadStorageFile(Storage *S_ptr);

Description: Reads storage data from the "storage.csv" file and creates a storage list.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: S_ptr - Pointer to the Storage list.

Return type: void

Memory usage: Allocates memory for temporary Recipe structures.

Big O notation: O(n), where n is the number of storage in the file.

extern void ReadLogsFile(Logs *L_ptr);

Description: Reads log data from the "logs.csv" file and creates a log list.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: L_ptr - Pointer to the Logs list.

Return type: void

Memory usage: Allocates memory for temporary Logs structures.

Big O notation: O(n), where n is the number of logs in the file.

extern void ReadConnectionFile(Connection* C_ptr);

Description: Reads connection data from the "connection.csv" file and creates a connection list.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: C_ptr - Pointer to the Connection list.

Return type: void

Memory usage: Allocates memory for temporary Connection structures.

Big O notation: O(n), where n is the number of connections in the file.

extern void DeleteLineFromRecipeFile(Recipe *R_tmp_Recipe);

Description: Deletes a specified recipe line from the "recipes.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: R_delete_Recipe - Pointer to the Recipe structure to be deleted.

Return type: void

Memory usage: Allocates memory for temporary Recipe structures.

Big O notation: O(n), where n is the number of recipes in the file.

extern void UpdateStorageFile(char c_Storage_To_Update[], double d_amount);

Description: Update a specified storage line from the "storage.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: c_Storage_To_Update - Name of the Storage structure to be updated.

d_amount - Amount of the storage item, what need to update it.

Return type: void

Memory usage: Allocates memory for temporary Storage structures.

Big O notation: O(n), where n is the number of storages in the file.

extern void AutoDeleteStorageItem(void);

Description: Delete outdated storage element(s) from the "storage.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: void

Return type: void

Memory usage: Allocates memory for temporary Storage structures.

Big O notation: O(n), where n is the number of storages in the file.

extern void WriteRecipeToFile(Recipe *R_Recipe);

Description: Writes recipe data to the "recipes.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: R_Recipe - Pointer to the Recipe structure to be written.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(1)

extern void WriteIngredientToFile(Ingredient *I_Ingredient);

Description: Writes ingredient data to the "ingredients.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: I_Ingredient - Pointer to the Ingredient structure to be written.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(1)

extern void WriteStorageToFile(Storage *S_Storage);

Description: Writes storage data to the "storage.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: S_Storage - Pointer to the Storage structure to be written.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(1)

extern void WriteConnenctionToFile(Connection *C_Connection);

Description: Writes connection data to the "connection.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: C_Connection - Pointer to the Connection structure to be written.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(1)

extern void WriteLogsToFile(Logs *L_Log);

Description: Writes log data to the "logs.csv" file.

Error Cases: Exits the program if the file cannot be opened.

Input arguments: L_Log - Pointer to the Logs structure to be written.

Return type: void

Memory usage: No dynamic memory allocation.

Big O notation: O(1)