College of Computer Studies
De La Salle University
CCPROG2 Machine Project
Meals4Health

Your task is to create a repository of cooking recipes and to provide tools to add recipes, generate shopping list, and to recommend menu. It is foreseen that there are at least 2 types of users for your program: 1.) the person who creates the recipes (and does not want these to be modified by just anyone) and 2.) the person who will be choosing the recipe to use based on the nutritional facts. Thus, there are two modes of access in this program: Update Recipe Box and Access Recipe Box. Some features in Update Recipe Box may also be available in Access Recipe Box.

Before proceeding to the two modes and to the different features under these, let us first define what a recipe is. A recipe consists of the name of the dish, the classification of this dish (could only be the string starter, main, or dessert), the list of food items needed to complete the dish (these are called ingredients), the list of instructions to prepare or cook the ingredients to become the dish (sometimes these instructions are referred to as steps or procedures), and the number of servings the recipe is for. For your program, let us assume that the name of the dish can be at most 20 characters long and each recipe can have at most 20 ingredients and at most 15 instructions. Each ingredient is further divided into the quantity (may contain fractional part, but assume that the user will give you the input as a real number), the unit (like grams, cups, teaspoon, etc.—just assume that this could be a word with at most 15 characters), and the item (assume that this is a phrase with at most 20 characters). Assume that each instruction will be a phrase or sentence that will not exceed 70 characters. The serving size is a whole number, specifying for how many people it is for.

Now, let's go back to the features of your program. Again, you have 2 modes: Update Recipe Box and Access Recipe Box. Thus, you need to have a menu for this. Include Exit in the choices, of course. Let's call this the Main Menu. Every time the user is in the Main Menu, he can choose which of these options he wants to invoke. For the project, it is left for you as the programmer to decide what input you expect from your user to invoke the function. Suggestion: you could use U for Update, A for Access, and E for Exit (or 1, 2, and 3 respectively…. Better yet, you could use up and down arrow keys).

If the user chooses the Update mode, then he must first input his username and password. For simplicity, the program should accept as username ***admin*** (all small letters) and password is ***ad1234***. Only when these two information are given by the user can he enter the Update Mode. Otherwise, a message stating "Invalid username or password" should be shown. Then, the user is returned back to the Main Menu.

If the user is able to enter the Update Mode, then he can use any of the features there until he chooses to Return to Main Menu. Refer to the list below for the features under Update Mode.

If the user chooses Access Mode, there is no need for any username or password. The features in the Access Mode will be available to the user until he chooses to return to the Main Menu again. Refer to the list below for the features under Access Mode.

If the user chooses Exit, then the program ends.

Update Recipe Box

As mentioned, your program should allow the user to update the recipe box. For this virtual recipe box, let us assume that there can be at most 50 recipes in your collection and at most 50 food items that can be used as ingredients. The recipe collection and food-calorie collection should be implemented as arrays [Do not ask the user how many entries will be given as input]. And in this Update Mode, your program should provide a user-friendly interface, as well as the functionality to do the following:

- ***Add Food-Calorie Info***
  Your program should allow adding of a food item with its corresponding calorie count. It asks for the food item (e.g., olive oil), the quantity (e.g., 1), the unit (e.g., tablespoon), and the calorie count (e.g., 120). Note that the food item should be unique and should have at least 1 character. [For actual data, refer to this sample website: http://whatscookingamerica.net/NutritionalChart.htm]
- ***View Food-Calorie Chart***
  Display entries in the Food-Calorie info in table form in this sequence:
  Food Item Quantity Unit Calories
  Make sure to show only 10 items per screen. Allow the user to press N to view the next 10 items and X to exit the view. Do not show empty/garbage values.
- ***Save Calorie Info***
  Your program should allow all food-calorie information to be saved into a text file. The data stored in the text file can be used later on. The system should allow the user to specify the filename. Filenames have at most 20 characters including the extension. Extension of file to be saved is .txt. If the file exists, the data will be overwritten.

  Save the file in the following format:
  <Food Item1><next line>
  <Quantity><space><Unit><space><Calories><next line>
  <next line>
  <Food Item2><next line>
  <Quantity><space><Unit><space><Calories><next line>
  <next line>

: :
<Food ItemN><next line>
<Quantity><space><Unit><space><Calories><next line>
<next line>
<end of file>

- ***Load Calorie Info***

  Your program should allow the calorie info stored in the text file to be **added** to the list in the program. The user should be able to specify which file to load. Note that for each food-calorie info to be loaded where the food item [name] (e.g. olive oil) already exists, the system has to prompt the user if he wants the existing data in your program to be overwritten. If the user said yes, your program should replace the existing food-calorie info (includes quantity, unit, etc.) with the one from the text file. On the other hand, if the user said no, your program retains the food-calorie info in the program. If the recipe to be loaded is unique, it will be added to your database. The data in the text file is stored in the format specified in Save Calorie Info. When you load the text file, assume the file is already in the correct format. Do not assume that the file will contain sorted data or are all in a specific case (i.e., may not be in all caps or all small letters).

- ***Add Recipe***

  Your program should allow creation of a new recipe by asking the user to input the name of the dish, the classification, the number of servings, the different ingredients and the list of procedures. Make sure that in adding a recipe, the name of the dish should have at least 1 character and the name does not exist yet (meaning it has to be unique). The serving size should be an integer. The classification could only be ***starter***, ***main***, or ***dessert***. The unit and the item should also have at least 1 character each (i.e., it cannot be blank). There should be at least one ingredient and at least one step of instruction for a recipe. The implementation of this feature may call Add Ingredient and Add Step (discussed below). After adding the recipe information, the user is brought back to the Update Recipe Box Menu.

- ***Modify Recipe***

  If the user chooses this option a listing of all recipe titles (names of dishes) should be displayed in alphabetical order (by calling List Recipe Titles). Then, the user is asked to input the name of the dish he wishes to modify. Of course, only an existing recipe may be modified. Note that modification does not mean that the user retypes all information. He is asked which of the information in the chosen recipe he wants to modify. Modifications may be any of the following:

  - ***Add Ingredient***

    If the user chooses this option, a new ingredient is added to the current set of ingredients. Then, the user is brought back to the Modify Recipe Menu.

  - ***Delete Ingredient***

    If the user chooses this option, allow the user to select which of the existing ingredients he wishes to delete (the input for this could be a number starting from 1 until the number of ingredients he has). Note that if there is only 1 ingredient left, the user is not allowed to delete it. Then, the user is brought back to the Modify Recipe Menu.

  - ***Add Step***

    If the user chooses this option, the user is asked where the new step / instruction will be inserted in. Then, the user is also asked to input the new instruction. Note that this is could be inserted at the start or at the middle or added at the end, but there should be no empty / skipped steps. Then, the user is brought back to the Modify Recipe Menu.

  - ***Delete Step***

    If the user chooses this option, allow the user to select which of the existing step he wishes to delete (the input for this could be a number starting from 1 until the number of steps he has). There should at least be 1 step in a recipe. Thus, if there is only 1 step left, the user is not allowed to delete it. Then, the user is brought back to the Modify Recipe Menu.

  - ***Return to Update Recipe Box Menu***

    If the user chooses this option, the user is returned back to the features of Update Mode.

- ***Delete Recipe***

  Once the user chooses this option, a listing of all recipe titles (names of dishes) should be displayed in alphabetical order (by calling List Recipe Titles). Then, the user is asked to input the name of the dish he wishes to delete. Of course, only an existing recipe may be deleted. Meaning, if the user types in a recipe title that does not exist (maybe because of misspelling), nothing is deleted and a message “Recipe is not in the list” should be displayed. Note that in deleting a recipe, all ingredients and instructions under it will automatically be removed.

- ***List Recipe Titles***

  Provide a listing of all recipe titles in alphabetical order. After the display, the user is brought back to the Update Mode Menu. Note that for the basic requirement, the arrangement of alphabetical actually means based on ASCII value. That is, if the recipe titles are “lasagna”, “Beef Stew”, “Egg drop soup”, “carrot cake” and “Egg Pie”, then the display should be in the following sequence (since it is based on ASCII value): are “Beef Stew”, “Egg Pie”, “Egg drop soup”, “carrot cake”, then “lasagna”.

- ***Scan Recipes***

  Display each recipe one at a time until all entries have been displayed. All information per recipe should be displayed. 2. For each recipe information that will be shown, the number of calories for each of the food item should also be displayed and the total calories for the recipe (not per person) should also be displayed. The recommended format is as follows:

<Recipe Title> <Number of Servings> <Total Calories><next line>
Ingredients:<next line>
<Quantity> <Unit> <Food Item> <Calories><next line>
<Quantity> <Unit> <Food Item> <Calories><next line>
: :
<Quantity> <Unit> <Food Item> <Calories><next line>
<next line>
Procedure:<next line>
1. <step1><next line>
2. <step 2><next line>
: :

Provide a way for the user to view the next or previous property or exit the scanning/viewing. The sequence of the display should be alphabetical based on the recipe title. Once the user chooses to exit the scanning or he has already reached the end of the list of recipes, he is brought back to the Update Mode Menu.

- ***Search Recipe by Title***

Show a listing of all recipe titles (in alphabetical order, again using List Recipe Titles) before asking your user to type the recipe title whose recipe information he wants to view. If the recipe title exists, then show all the information about that particular recipe. Then, bring the user back to the Update Mode Menu.

- ***Export Recipes***

Your program should allow all recipes to be saved into a text file. The data stored in the text file can be used later on. The system should allow the user to specify the filename. Filenames have at most 20 characters including the extension. Extension of file to be saved is .txt. If the file exists, the data will be overwritten.

Save the file in the following format:
<Recipe Title1><next line>
<Serving Size><space><Classification><next line>
Ingredients<space><ingredient count><next line>
<Quantity><space><Unit><space><Item1><next line>
<Quantity><space><Unit><space><Item2><next line>
: : :
<Quantity><space><Unit><space><ItemN><next line>
Steps<space><steps count>
<Step1><next line>
<Step2><next line>
: : :
<StepN><next line>
<next line>
<Recipe Title2><next line>
<Serving Size><space><Classification><next line>
Ingredients<space><ingredient count><next line>
<Quantity><space><Unit><space><Item1><next line>
<Quantity><space><Unit><space><Item2><next line>
: : :
<Quantity><space><Unit><space><ItemN><next line>
Steps<space><steps count>
<Step1><next line>
<Step2><next line>
: : :
<StepN><next line>
<next line>
: : :
<Recipe TitleN><next line>
<Serving Size><space><Classification><next line>
Ingredients<space><ingredient count><next line>
<Quantity><space><Unit><space><Item1><next line>
<Quantity><space><Unit><space><Item2><next line>
: : :
<Quantity><space><Unit><space><ItemN><next line>
Steps<space><steps count>
<Step1><next line>
<Step2><next line>
: : :
<StepN><next line>
<next line>
<end of file>

- ***Import Recipes***

Your program should allow the data stored in the text file to be added to the list in the program. The user should be able to specify which file to load. Note that for each recipe to be loaded where the recipe title already exists, the system has to prompt the user if he wants the existing data in your program to be overwritten. If the user said yes, your program should replace the existing recipe (including ingredients and steps) with the one from the text file. On the other hand, if the user said no, your program retains the recipe in the program. If the recipe to be loaded is unique, it will be added to your database. The data in the text file is stored in the format specified in Export. When you load the text file, assume the file is already in the correct format. Note that the recipe titles in the file may not be arranged in alphabetical order. There is also no guarantee that all recipe titles are in all caps or in all small letters. Note that it is possible that the Import feature is called more than once, thus data from multiple files are loaded into the list.

- ***Return to Main Menu***
  This will just allow the user to quit the Update Mode and return back to the Main Menu. The information in the lists should be cleared after this option.

Access Recipe Box

- ***Import Recipes***
  Your program asks the user to specify the filename of a text file where the list of recipes will originate. The information should be loaded into a list. This is actually the same Import feature in the Update Mode. After loading the contents of the text file to the list, the user is brought back to the Access Mode Menu.
- ***List Recipe Titles***
  Provide a listing of all recipe titles (in alphabetical order). Again, this is the same as the one in the Update Mode. After the display, the user is brought back to the Access Mode Menu.
- ***Scan Recipes***
  Display each recipe one at a time until all entries have been displayed. All information per recipe should be displayed. Provide a way for the user to view the next or previous property or exit the scanning/viewing. The sequence of the display should be alphabetical based on the recipe title. Once the user chooses to exit the scanning or he has already reached the end of the list of recipes, he is brought back to the Access Mode Menu.
- ***Search Recipe by Title***
  Show a listing of all recipe titles (in alphabetical order, again using List Recipe Titles) before asking your user to type the recipe title whose recipe information he wants to view. If the recipe title exists, then show all the information about that particular recipe. Then, bring the user back to the Access Mode Menu.
- ***Generate Shopping List***
  Show a listing of all recipe titles (using List Recipe Titles), then ask the user which of these recipes to generate a shopping list for. The user is also asked for how many people he will be cooking for. Recomputation may be necessary based on the number of servings of the recipe. This is then displayed on the screen. Then, bring the user back to the Access Mode Menu.
- ***Scan Recipes by Ingredient***
  Ask the user to type in an ingredient (just the food item, no quantity or unit). Using this, your program shows the recipe information of all recipes where part of its ingredient is the given ingredient. Provide a way for the user to view the next or previous property or exit the scanning/viewing. The sequence of the display should be alphabetical based on the recipe title. Once the user chooses to exit the scanning or he has already reached the end of the list of recipes, he is brought back to the Access Mode Menu.
- ***Recommend Menu***
  The program asks the user to enter a target calorie intake for the meal for 1 person. Based on this information, the program randomly chooses 1 main course from the list of recipes in the list. If the calorie count of the main course does not completely fulfill the target calorie, a starter is also included in the recommendation. If after recommending both a starter and a main course, there is still enough calorie allowance for dessert, a dessert is also recommended. Note that if there is no available course (ex. there are no starter recipes), then just skip and show a recipe of the available course/s (eg. main course and dessert). The menu should not exceed the target calorie intake. If the ingredient (in the recipe) is not included in the list of food-calorie information, then assume that the calorie count for that food is 0 (for example water is an ingredient, but water is not listed in the food-calorie information list). Show the recipe information for the suggested menu one at a time from starter to main course to dessert. Then, bring the user back to the Access Mode Menu.
- ***Return to Main Menu***
  This will just allow the user to quit the Access Mode and return back to the Main Menu. The information in the lists should be cleared after this option.

Bonus

A **maximum** of **10 points** may be given for features **over & above** the requirements (such as use of modules in programming or other features not conflicting with the given requirements or changing the requirements [example: search recipe by multiple ingredients, adding of more user names and passwords which are stored in a file, changing of password, etc.]) subject to **evaluation** of the teacher. **Required features** must be **completed first** before bonus features are credited. Note that use of conio.h, or other advanced C commands/statements may **not** necessarily merit bonuses.

Note though that you are NOT ALLOWED to do the following:

- to declare and use global variables (i.e., variables declared outside any function),
- to use goto statements (i.e., to jump from code segments to code segments), [gotoxy() is allowed]
- to use the break or continue statement to exit a block. Break statement can only be used to break away from the switch block,
- to use the return statement or exit statement to prematurely terminate a loop or function or program,
- to use return; for void functions,
- to use the exit statement to prematurely terminate a loop or to terminate the function or program, and
- to call the main() function to repeat the process instead of using loops.

It is best that you perform your coding "incrementally." This means:

- Dividing the program specification into subproblems, and solving each problem separately according to your algorithm;
- Code the solutions to the subproblems one at a time. Once you're done coding the solution for one subproblem, apply testing and debugging.

Submission & Demo

- **Final MP Deadline: March 30, 0730 AM via AnimoSpace**. No project will be accepted anymore after the submission link is locked and the grade is automatically 0.

  Requirements: Complete Program

  - Make sure that your implementation has considerable and proper use of arrays, structures, files, and user-defined functions, as appropriate, even if it is not strictly indicated. See details on the note regarding "Not Allowed" above.
  - It is expected that each feature (except for the exit or back to menu features) is supported by at least one function that you implemented. Some of the functions may be reused (meaning you can just call functions you already implemented [in support] for other features. There can be more than one function to perform tasks in a required feature.
  - Debugging and testing was performed exhaustively. The program submitted has

    a. NO syntax errors

    b. NO warnings - make sure to activate -Wall (show all warnings compiler option) and that C99 standard is used in the codes

    c. NO logical errors -- based on the test cases that the program was subjected to

Important Notes:

1. Use **gcc -Wall** -std=C99 to compile your C program. Make sure you **test** your program completely (compiling & running).
2. Do not use brute force. Use **appropriate conditional** statements **properly**. Use, **wherever appropriate**, **appropriate loops** & **functions properly**.
3. You **may** use topics outside the scope of CCPROG2 but this will be **self-study**. Refer to restrictions stated above (regarding "Not Allowed"). Be ready to explain how and why you used these.
4. Include **internal documentation** (comments) in your program.
5. The following is a checklist of the deliverables:

Checklist:

- ❒ Upload via AnimoSpace submission:
  - ❒ source code*
  - ❒ test script**
  - ❒ sample text file exported from your program
- ❒ email the softcopies of all requirements as attachments to **YOUR** own email address on or before the deadline

Legend:

***** Source code exhibit readability with supporting inline documentation (not just comments before the start of every function) and follows a coding style that is similar to those seen in the course notes and in the class discussions. The first page of the source code should have the following declaration (in comment) [replace the pronouns as necessary if you are working with a partner]:

/**********

This is to certify that this project is my own work, based on my personal efforts in studying and applying the concepts learned. I have constructed the functions and their respective algorithms and corresponding code by myself. The program was run, tested, and

debugged by my own efforts. I further certify that I have not copied in part or whole or otherwise plagiarized the work of other students and/or persons, nor did I employ the use of AI in any part of the deliverable.

<your full name>, DLSU ID# <number>
**********/

Example coding convention and comments before the function would look like this:

```
/* funcA returns the number of capital letters that are changed to small letters
   @param strWord - string containing only 1 word
   @param pCount - the address where the number of modifications from capital to small are
                   placed
   @return 1 if there is at least 1 modification and returns 0 if no modifications
   Pre-condition: strWord only contains letters in the alphabet
*/
int     //function return type is in a separate line from the
funcA(char strWord[20] ,    //preferred to have 1 param per line
      int * pCount)            //use of prefix for variable identifiers
{  //open brace is at the beginning of the new line, aligned with the matching close brace
   int    ctr;        /* declaration of all variables before the start of any statements -
                         not inserted in the middle or in loop- to promote readability */

   *pCount = 0;
   for (ctr = 0; ctr < strlen(strWord); ctr++)  /*use of post increment, instead of pre-
                                                  increment */
   {  //open brace is at the new line, not at the end
      if (strWord[ctr] >= 'A' && strWord[ctr] <= 'Z')
      {    strWord[ctr] = strWord[ctr] + 32;
           (*pCount)++;
      }
      printf("%c", strWord[ctr]);
   }

   if (*pCount > 0)
      return 1;
   return 0;
}
```

Test Script should be in a table format. There should be at least 3 categories (as indicated in the description) of test cases **per function. There is no need to test functions which are only for screen design (i.e., no computations/processing; just printf).

Sample is shown below.

Function	#	Description	Sample Input Data	Expected Output	Actual Output	P/F
sortIncreasing	1	Integers in array are in increasing order already	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P
	2	Integers in array are in decreasing order	aData contains: 53 37 33 32 15 10 8 7 3 1	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P
	3	Integers in array are combination of positive and negative numbers and in no particular sequence	aData contains: 57 30 -4 6 -5 -33 -96 0 82 -1	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	P

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the sample code in page 7, the following are four distinct classes of tests:

i.) testing with strWord containing all capital letters (or rephrased as "testing for at least 1 modification")
ii.) testing with strWord containing all small letters (or rephrased as "testing for no modification")
iii.) testing with strWord containing a mix of capital and small letters
iv.) testing when strWord is empty (contains empty string only)

The following test descriptions are **incorrectly formed**:

- Too specific: testing with strWord containing "HeLlo"
- Too general: testing if function can generate correct count OR testing if function correctly updates the strWord
- Not necessary -- since already defined in pre-condition: testing with strWord containing special symbols and numeric characters

6. Upload the softcopies via Submit Assignment in AnimoSpace. You can submit multiple times prior to the deadline. However, only the last submission will be checked. Send also to <u>your</u> **DLSU GMail account** a **copy** of all deliverables.
7. You are allowed to create your own modules (.h) if you wish, in which case just make sure that filenames are descriptive.
8. During the MP **demo**, the student is expected to appear on time, to answer questions, in relation with the output and to the implementation (source code), and/or to revise the program based on a given demo problem. Failure to meet these requirements could result to a grade of 0 for the project.

9. It should be noted that during the MP demo, it is expected that the program can be compiled successfully and will run in the command prompt using gcc -Wall -std=C99. If the program does not run, the grade for the project is automatically 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) and other submissions (e.g., non-violation of restrictions evident in code, test script, and internal documentation) will still be checked. ***During the demo, text files for the import and load features will be provided for your use These conform to the format indicated in the specs. There should be no hardcoded data or pre-loaded data once the program is initially launched.***

10. The MP should be an HONEST intellectual product of the student/s. For this project, you are allowed to do this individually or to be in a group of 2 members only. [Note that if you decide to work individually, you may still be subject to the Individual Demo Problem.] Should you decide to work in a group, the following mechanics apply:

Individual Solution: Even if it is a group project, each student is still required to create his/her INITIAL solution to the MP individually without discussing it with any other students. This will help ensure that each student went through the process of reading, understanding, solving the problem and testing the solution.

Group Solution: Once both students are done with their solution: they discuss and compare their respective solutions (ONLY within the group) -- note that learning with a peer is the objective here -- to see a possibly different or better way of solving a problem. They then come up with their group's final solution -- which may be the solution of one of the students, or an improvement over both solutions. Only the group's final solution, with internal documentation (part of comment) indicating whose code was used or was it an improved version of both solutions) will be submitted as part of the final deliverables. It is the group solution that will be checked/assessed/graded. Thus, only 1 final set of deliverables should be uploaded by one of the members in the AnimoSpace submission page. [Prior to submission, make sure to indicate the members in the group by JOINing the same group number.]

Individual Demo Problem: As each is expected to have solved the MP requirements individually prior to coming up with the final submission, both members should know all parts of the final code to allow each to INDIVIDUALLY complete the demo problem within a limited amount of time (to be announced nearer the demo schedule). This demo problem is given only on the day/time of the demo, and may be different per member of the group. Both students should be present during the demo, not just to present their individual demo problem solution, but also to answer questions pertaining to their group submission. No demo, grade is automatically 0.

Grading: the MP grade will be the same for both students -- UNLESS there is a compelling and glaring reason as to why one student should get a different grade from the other -- for example, one student cannot answer questions properly OR do not know where or how to modify the code to solve the demo problem (in which case deductions may be applied or a 0 grade may be given -- to be determined on a case-to-case basis).

11. Any form of **cheating** (**asking other people not in the same group for help**, **submitting as your [own group's] work part of other's work**, **sharing your [individual or group's] algorithm and/or code to other students not in the same group, using AI even partially as part of submitted deliverable, etc.**) can be punishable by a grade of **0.0** for the **course** & a **discipline case**.

Any requirement not fully implemented or instruction not followed will merit deductions.

----------There is only 1 deadline for this project: March 30, 0730 AM, but the following are **suggested** targets.----------
*Note that each milestone assumes fully debugged and tested code/function, code written following coding convention and included internal documentation, documented tests in test script. It is expected that at least one submission per target milestone is done in AnimoSpace. Even if this is not graded, this serves as version control and proof also of continuing efforts.

1.) Milestone 1 : February 28
   a. Menu options and transitions
   b. Preliminary outline of functions to be created
   c. Declaration of Data Structure needed
   d. Checking of username and password prior to allowing access to Update Mode

2.) Milestone 2: March 7
   a. Add Food Calorie Info
   b. View Food Calorie Chart
   c. Add Recipe
   d. Add Ingredient
   e. Add Step
   f. Search by Recipe Title

   Test on Windows System

3.) Milestone 3: March 14
   a. Modify Recipe
   b. Delete Recipe
   c. List Recipe Titles (including sorting/displaying in alphabetical order by ASCII value)
   d. Scan Recipes
   e. Generate Shopping List*
   f. Scan Recipes by Ingredient*

   †For these features, do not clear the contents of the array of entries upon exit from Update menu IF you have not implemented the import and export yet.

4.) Milestone 4: March 21
   a. Recommend Menu
   b. Save Food Calorie Info
   c. Load Food Calorie Info
   d. Export Recipes
   e. Import Recipes
   f. Translate Text File

5.) Milestone 5: March 28
   a. Integrated testing (as a whole, not per function/feature)

b. Collect and verify versions of code and documents that will be uploaded
c. Recheck MP specs for requirements and upload final MP
d. [Optional] Implement bonus features