Yet Another Travel Application (YATA)

The definition of *Wanderlust*[1] in the Cambridge Dictionary is "*the wish to travel far away to many different places*". Travelling to places not only quenches a desire to "wander" but also improves one's understanding of other cultures. It allows a person to disconnect and recharge; it relieves stress and anxiety, and the experience might even make one smarter.

For your project, you are to create a simple **personal travel management tool** that aids in planning future travels, recording travel experiences and keeping stock of travel goals.

1. General Instructions.

The following entities are maintained:

1.1. Travel Destinations

The travel destination list contains recommended travel spots. It can contain a maximum of 100 items. Also, the user should have the option to add, delete and update items on the list. Each destination location should provide the following information:

1.1.1. **Destination Longname** - This Should be **50** characters long.
Example - La Union

1.1.2. **Destination Shortname** - This should be **10** characters long and must be **UNIQUE** (no other record can contain the same Shortname) within the list.
Example - Elyu

1.1.3. **Country** - This should be **20** characters long.
For example - *Philippines*

1.1.4. **Geographical Group (GeoGroup)** - This should be **20** characters long.
Some Examples: Luzon, Visayas, Mindanao, Europe, North America

1.1.5. **ToDo** - A short description of recommended activities for this destination. The description should be **100** characters long.
Example: Surf in Urbiztondo Beach, have Breakfast in Masa bakehouse and Visit the Baluarte Watch Tower.

1.2. Travel Goals (Bucket List)

The travel bucket list contains travel destination goals (taken from the travel destination list). The list can only have a maximum of **10** entries. Each record contains the following:

1.2.1. **Destination Shortname** - **10** characters long and should correspond to an entry in the Travel Destination List. *Example - Elyu*

1.2.2. **Priority Rank** - an Integer number indicating the priority to visit. Ranking ranges from **1 - 10,** with **1** being the highest priority (Most desired to see the soonest) and **10** being the least priority. *Note that multiple items can have the same priority. E.g. Three entries can have priority as 1.*

[1] https://dictionary.cambridge.org/us/dictionary/english/wanderlust

1.2.3. **Remarks** - Note to self about this entry. It should be 30 characters long. *E.g. Do this on Term 2 break*

1.2.4. **Achieved Flag -** An indicator that specifies the achievement of the goal. *E.g. Yes or No*

1.3. **Travel Plan.**

A simple travel itinerary contains

1.3.1. **Destination Shortname** - **10** characters long and should correspond to an entry in the Travel Destination List. *Example - Elyu*

1.3.2. **Travel start date** - This should be **10** characters long, marking the start of the travel with the format of *mm/dd/yyyy*

1.3.3. **Daily Itinerary (list)** - A list of activities details.

The number of itinerary entries should correspond to the length of stay. For example, a three (3) day stay should have three (3) entries in the list. See the illustration below.

Day Count	Morning	Afternoon	Evening
Day 1	*Travel to Elyu*	*Lunch and Checkin*	*Dinner and night scene*
Day 2	*Breakfast at Masa*	*Go surfing*	*Dinner and chill*
Day 3	*Breakfast and visit Baluarte*	*Buy Pasalubong*	*Travel back home*

Correspondingly, a four (4) day trip should have four (4) entries and so on.
Below is a more detailed description of the fields:

1.3.3.1. **Day Iteration** - Event sequence number of the Integer. E.g. Day 1

1.3.3.2. **Morning activity** - Short description of the activity in the morning. 30 characters long. *Example: Travel to San Juan, Elyu.*

1.3.3.3. **Afternoon activity** - Short description of the activity in the afternoon. 30 characters long. *Example: Late lunch and hotel check-in.*

1.3.3.4. **Evening activity** - Short description of the activity in the. 30 characters long. *Example: Dinner and check out the night scene*

1.3.4. **Provision for rating the trip** - The application should provide the user with a means to *rate the travel plan when completed*.

1.3.4.1. **Rating** - Number of “Stars” with a scale of (0-5) and increments of 0.5. Five (5) Star is the highest, while zero (0) star is the lowest. It is possible to have half a star, e.g. 2.5 stars.

1.3.4.2. **Comments** - A maximum of 100 characters describing the trip. *Example: Must try Manong Benny sandwich at Masa, love surfing at the beach but hate my hotel*

2. Data Persistence (Storing Information)

All data and information discussed in the section above (Section 1.0) shall be stored as system data files accordingly. Details are as follows

2.1. **File type and name**

2.1.1. A text file **destination.txt** shall contain the travel destinations described in *section 1.1*

2.1.2. A text file **bucketlist.txt** shall contain the list of travel goals as described in *section 1.2*

2.1.3. A text file <*shortname*>- **itinerary.txt** shall contain the travel plan described in *section 1.3*

2.1.3.1. Each travel plan detail is a separate text file with a file naming convention:

<*destination shortname*>**-itinerary.txt**

Where:
<*destination shortname*>
is the shortname of as defined in destination.txt. *E.g. Elyu*

-itinerary.txt is a literal text.

See Examples below::

Elyu-itinerary.txt	This shall be the filename for the travel itinerary to Elyu
Bora-itinerary.txt	This shall be the filename for the travel itinerary to Boracay
Vigan-itinerary.txt	This shall be the filename for the travel itinerary to Vigan

2.2. **File Format**

2.2.1. The text file **destination.txt** shall have the following format:

<shortname><*space*><longname><*new line*>
<country><*new line*>
<geogroup><*new line*>
<todo><*new line*>
<*new line*>
…
<shortname**N**><*space*><longname><*new line*>
<country><*new line*>
<geogroup><*new line*>
<todo><*new line*>
<*new line*>

2.3. The text file **bucketlist.txt** shall have the following format:

<shortname><*new line*>
<priority><*new line*>
<remarks><*new line*>
<achieved><new line>
<*new line*>
…
<shortname**N**><*new line*>
<priority><*new line*>
<remarks><*new line*>
<*new line*>

2.4. The text file <*shortname*>-**itinerary.txt** shall have the following format:

<startdate mm/dd/yyy><*new line*>
<day**1**><*new line*>
morning<*new line*>
<morning itinerary text><new line>
afternoon<new line>

<afternoon itinerary text><new line>
evening<new line>
<evening itinerary text><new line>
<new line>
…
<day**N**><*new line*>
morning<*new line*>
<morning itinerary text><new line>
afternoon<new line>
<afternoon itinerary text><new line>
evening<new line>
<evening itinerary text><new line>
<new line>

3. Initial File Contents.

Default files are available to the students at the start. (see attached resources)

3.1. ***destination.txt*** - shall be provided with an initial list of destinations. Subsequently, it shall be updated as discussed in Section 5 below.

3.2. ***bucketlist.txt*** - shall initially be **empty**. Contents shall be updated accordingly, as discussed in section 5 below.

3.3. There is NO provision for an initial itinerary file. Creating and update of the file is according to the program specifications discussed in sections 2.4 and 5.4

4. On Program Startup

Upon every program startup, the application shall load the contents of the files **destination.txt** and **bucketlist.txt.**

The program should handle gracefully any exception and error conditions that may occur at this point.

5. User Interaction.

Upon the start of the application, a text-based menu shall prompt the user for action and offers a list of options available. The following is a list of the features and functionalities that should be available to the user.

5.1. **Show Dashboard**

The dashboard visually summarises the travel goals. The following information is displayed.

5.1.1. **Information to display**

5.1.1.1. **Top 3 Bucket List destinations**

Each bucketlist destination should include the shortname, longname and priority. Sorting is by priority. Highest priority first and lowest last.

5.1.1.2. **Top 3 Destinations** (according to average rating)

Each destination in the list should include the rating, shortname, longname and geolocation. Sorting is by “stars” rating. The highest rating is first, and the lowest last.

5.1.1.3. **Bucketlist progress**.
The user can view achieved bucketlist goals vis-a-vis the goal list.
Example: If three (3) destination goals are achieved from a list of 10 destinations, they will read as **3 out of 10 or 30%** achieved.

5.1.2. Back option - return to the previous menu

5.2. **Destination List Management.**
The user should have the standard functionality of maintaining the list of destinations. The user should be able to add to the list, remove from the list and edit a record. The following menu options should be available to the user.

5.2.1. **Short Display List** – Display the list's contents showing only on the shortname and longname per item

5.2.2. **Long Display List** – Display the list's content showing all the information per item list. (E.g. include all the record fields)

5.2.3. **Add a record**.
Prompt the user to enter the information described in *section 1.1*. Note that the shortname should be unique, and no duplicates must exist.

5.2.4. **Delete a record.**
Delete an entry from the list one at a time.

5.2.5. **Edit a record.**
Prompt the user to edit the information described in *section 1.1*. Note that all of the fields are editable except for the shortname.

5.2.6. **Back option** - return to the previous menu

5.3. **Bucket list Management**
The user should have the standard functionality of maintaining the bucket list. The user should be able to add to the list, remove from the list and edit a record. The following menu options should be available to the user.

5.3.1. **Display List** – Display the list's content showing all the information per item list. (E.g. include all the record fields)

5.3.2. **Add a record**.
Prompt the user to enter the information described in *section 1.2*. Note that the shortname should be unique, and no duplicates must exist.

5.3.3. **Delete a record.**
Delete an entry from the list one at a time.

5.3.4. **Edit a record.**
Prompt the user to edit the information described in *section 1.2*. Note that all of the fields are editable except for the shortname.

5.3.5. **Mark as Achived**.
Sets the value of the archived to signify the completion of the goal for a particular destination.

5.3.6. **Unmark as Achived**.
Unset the value of a previously marked goal as achieved.

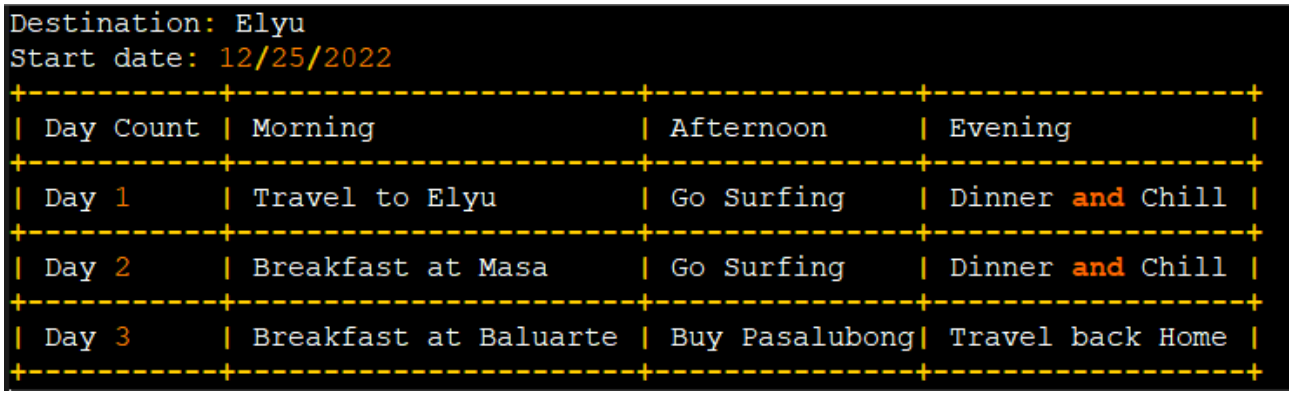
5.3.7. **Back option** - return to the previous menu

5.4. **Trip Management**

The user should have the standard functionality of maintaining the planned trips. The user should be able to add to the list, remove from the list and edit a record. The following menu options should be available to the user.

5.4.1. **Display List** – Display the list of planned trips. The displayed information should include the shortname, start date and rating (if available).

5.4.2. **View Itinerary** – Prompt the user to enter a shortname and start date and display the corresponding itinerary details. Refer to the illustration below as an example.



5.4.3. **Add a record**.
Prompt the user to enter the information described in *section 1.3*. Note that an itinerary record is unique by its **shortname and start date** (shortname+startdate), and no duplicates of the same shortname and start date must exist.
Also, the user must be allowed to enter as many days as needed until the user opts to end the input of itinerary records.

5.4.4. **Delete a record.**
Delete an entry from the list one at a time. Remember that only records with the same shortname and start date will be deleted (shortname+startdate).

5.4.5. **Edit a record.**
Prompt the user to edit the information described in *section 1.3*. All fields are editable EXCEPT for the **shortname** and **start date**.

5.4.6. **Rate a Trip**
The user can rate a completed trip based on the itinerary list. Rating is done for records with the same shortname and start date.
The user should be able to add, edit and delete a rating according to the rules in *sections 1.3.4*

5.4.7. Back option - return to the previous menu

5.5. **Quit the Application.**
Do any necessary cleanup processes and exit the application.

6. On Program Exit.

All data of the destination list, bucket list, itinerary list and other relevant information shall be saved into its corresponding file format and structure. See Section 2 for details.

The application should gracefully handle any error and exception that might occur accordingly.

7. Bonus.

A **maximum** of **10 points** may be given for features **over & above** the requirements (other features not conflicting with the given requirements or changing the requirements). For example: (1) Adding advanced search/display features to filter the destination display by ToDo keywords. E.g. display only records when the word “beach” occur in ToDo ; (2) Add advance statistics grouped by GeoLocation, e.g. the number of places visited by Island groups Luzon, Visayas and Mindanao; (3) Replace the **itinerary** text file to a binary format where the record structure shall be decided upon by the student.

Required features must be **completed first** before bonus features are credited. Note that using conio.h, or other advanced C commands/statements may **not** necessarily merit bonuses.

8. Submission and Demonstration.

8.1. **Final MP Deadline: April 11, 2023, 0800AM via AnimoSpace. –** No project will be accepted anymore after the submission link is locked, and the grade is automatically 0.

8.2. **Requirements – Complete Program**

- 8.2.1. 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.
- 8.2.2. It is expected that each feature 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 multiple functions to perform tasks in a required feature.
- 8.2.3. Debugging and testing were performed exhaustively. The program submitted has
  - 8.2.3.1. NO syntax errors.
  - 8.2.3.2. NO warnings - make sure to activate -Wall (show all warnings compiler option) and that C99 standard is used in the codes.
  - 8.2.3.3. NO logical errors -- based on the test cases the program was subjected to.

8.3. Important Notes.

- 8.3.1. Use gcc -Wall to compile your C program. Make sure you test your program completely (compiling & running).
- 8.3.2. Do not use brute force. Use appropriate conditional statements properly. Use, wherever applicable, suitable loops & functions properly.
- 8.3.3. You may use topics outside the scope of CCPROG2, but this will be self-study. Goto label, exit(), break (except in switch), continue, global variables, and calling main() is not allowed.
- 8.3.4. Include internal documentation (comments) in your program.
- 8.3.5. 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 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 the 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 with 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.

  <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).

See the Sample 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 above, 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.

8.3.6. 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 your **mylasalle account** a **copy** of all deliverables.

8.3.7. You are allowed to create your own modules (.h) if you wish, in which case just make sure that filenames are descriptive.

8.3.8. 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 in a grade of 0 for the project.

8.3.9. It should be noted that during the MP demo, it is expected that the program can be compiled successfully and will run. 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.

8.3.10. 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. Should you decide to work in a group, the following mechanics apply:

8.3.10.1. **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.

8.3.10.2. **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.]

8.3.10.3. **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.

8.3.10.4. **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).

8.3.11. 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, 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.