Linguistics is the study of language. Computational linguistics is the application of techniques in computer science into the study of language. Among the many applications of computational linguistics are machine translation (MT) and computer-aided language learning. In many approaches to creating these types of applications, there is a need to maintain a lexicon or dictionary (which may be represented as a database of information required by these applications). This computational linguistics now forms part of the area of artificial intelligence, specifically on the area of natural language processing (NLP). In DLSU-CCS, we have a laboratory called the Center for Language Technologies (CeLT). In the early days of its inception, we created a Filipino-English translation system, including the language resources needed to support the multi-approach MT system. [In recent years, we also diversify into other NLP tools and applications development and theoretical research, like ontology population, sentiment analysis, and conversational agents, among others. We collaborate with other centers and/or colleges/departments to come up with applications on various domains and disciplines including (but not limited to) health and nutrition, education, and psychology.]

For your project, you are to create some language tools that are related to each other: machine translation and language identification. Note that this project is to give you a glimpse of what NLP is about. There will be many aspects that you will not be expected to incorporate yet due to its complexity (like part of speech and grammar rules).

To support the two language tools, your project will also manage the collection of words as language and translation pairs that will be used. It is, thus, important to note that each translation and language pair can be at most 20 letters each. And there can be at most 10 pairs of language and translation per entry and a count for the number of pairs encoded in the entry. One entry (with a count of 6) would look something like this:

Language	Translation
English	love
Tagalog	mahal
Hiligaynon	gugma
Cebuano	gugma
Spanish	amor
Chinese	ai

Note that there can be at most 150 entries, but not all entries will have the same number of language - translation pairs. Below is a sample of another entry (with a count of 5):

Language	Translation
Tagalog	mahal
Kapampangan	mal
Cebuano	mahal
English	expensive
Chinese	gui

Notice that not all languages in a previous entry may be in the other entries. It may also not be sorted based on translation or the language. Note too that some languages may have the same translation OR even that two different entries have the same language-translation pair.

Your program should allow the user to **Manage Data**, by providing a user-friendly interface, as well as the functionality to do the following:

- ***Add Entry***
  Your program should ask for a language and translation pair (for example, Tagalog and mahal, respectively). The program first checks if there is already an existing entry with the indicated language and translation pair. If yes, then program first shows all information about entries information, same as with the given input (so if the given 2 examples above already exist, then both entries above should be shown first). Then, the user is asked if this is a new entry. If no, then the program goes back to the Manage Data menu options. If the user answered yes, then

a new entry is created with the input as the entry, then language-translation pairs should be taken in as input to be included as part of this new entry. Note that you are not allowed to ask the user how many language-translation pairs will be given as input. However, the program may ask the user if he wants to encode another pair or to use a sentinel value to terminate the input series. Both **`language`** and **`translation`** should each have at least 1 character.

- ***Add Translations***

  Your program should ask for a language and translation pair (for example, Tagalog and mahal, respectively). The program first checks if there is already an existing entry with the indicated language and translation pair. If no, then a message indicating that such an entry do not exist yet and thus the user should first use the Add Entry option. However, if there is an existing entry, then program first shows all information about entries' information. If there is only one entry that exists, then the language and translation pair is asked from the user and it is added to the same entry. The count now should be updated. In situations where there is more than one result, then the user is asked which particular entry is the one where a new language-translation pair will be added to. In this case, an integer input is expected. [Let us say the two samples above exist and we want to add a new translation pair for the first entry. So let's say the user inputs 1 to represent addition to the first entry, then the user is asked to input the language-translation pair (example French and amour, respectively). Lastly the count should be updated to 7.] The user is asked if more translations to the same entry is going to be given, if yes, then language-translation pairs will be asked and added to the entry. If the answer is no, the program reverts back to the Manage Data menu. Note that again language and translation should each have at least 1 character. Also, note that if the entry already has a count of 10 (language-translation pairs), no additional translation pairs can be added to this entry; a message should be shown in this situation before going back to the Manage Data menu.

- ***Modify Entry***

  Your program should allow your user to modify an entry. This option will first display all entries before asking which he wants to modify. [Refer to Display All Entries for details on how info are to be displayed.] The input for this is a number. A valid input starts from 1 to the number of entries initialized. Note that the first entry is referred to as entry 1, but should be stored in index 0. If an invalid number is given, a message is displayed before going back to the Manage Data menu. If a valid number is given, then the user is asked which item is to be modified. Again, an input number 1 signifies the first language-translation pair in this entry, but this is stored in index 0. The user is also asked which of this entry is to be modified, the language or the translation, before proceeding to ask for the new value for that entry. Note that the user is also asked if other language-translation pairs is going to be modified. When the user is satisfied and no more modifications are to be done on this current entry, the program reverts back to the Manage Data menu. If the user wishes to modify a different entry, the user has to chose Modify Entry again. Note that no additional translations will be done in this option and no deletions will also be done in this option.

- ***Delete Entry***

  The program first shows all entries' information before asking which entry the user wants deleted. [Refer to Display All Entries for details on how info are to be displayed.]. The input for this is a number. A valid input starts from 1and ends at the number of entries initialized. Note that the first entry is referred to as entry 1, but should be stored in index 0. If an invalid number is given, a message is displayed before going back to the Manage Data menu. Note that in deleting an entry, all language-translation pairs are included in the deletion.

- ***Delete Translation***

  Your program should allow your user to delete a language-translation pair from an entry. This option will first display all entries before asking which he wants to delete from. [Refer to Display All Entries for details on how info are to be displayed.] The input for this is a number. A valid input starts from 1 to the number of entries initialized. Note that the first entry is referred to as entry 1, but should be stored in index 0. If an invalid number is given, a message is displayed

before going back to the Manage Data menu. If a valid number is given, then the user is asked again which language-translation pair from this entry is to be deleted. Make sure that the input number is valid (before you delete). A valid number is from 1 to 10 only and it should be an initialized entry (meaning, the user cannot choose 7 if there are only 6 language-translation pair under the chosen word). If an invalid value is given, the user is notified with a message and is asked if a deletion will still be done. If not, then the program goes back to the Manage Data menu. If yes, then the program proceeds to ask again for which language-translation pair will be removed and continue to do so as long as the user still wants to delete language-translation pairs from this same entry. If there is only 1 language-translation pair in this entry and this is to be deleted, then the whole entry is also deleted.

- ***Display All Entries***

  Provide a listing of all entries. Display one entry (all language-translation pairs and the count) at a time. Each entry is displayed alphabetically (using the ASCII value) based on language. That is, within an entry (**intra**-entry) if the languages are "Spanish", "English", "Cebuano" and "Filipino", the sequence of the sorted display will be "Cebuano", "English", "Filipino", "Spanish". Of course, their corresponding words are displayed beside the language. [Note that **inter**-entries are not sorted. That is if the first entered word is about the "love", and the second entry is "expensive", these will not be sorted.] Provide a way for the user to view the next or previous word or exit the view (i.e., press 'N' for next, 'P' for previous, 'X' to end the display). Make sure that garbage values are not displayed (i.e., do not display the 7^{th}, 8^{th}, 9^{th}, and 10^{th} language-translation pair if there are only 6 in that entry). Do not also display all 150 entries if there are only 5 entries initialized.

- ***Search Word***

  This option first asks for an input word, then the program proceeds to show a listing of all entries where that input word appears as a translation. Display is similar to how the Display All Entries work (with a way to view next, previous, and exit) but only those that matches the given word. [For example, if the word is to be searched is mahal, then both entries in the given example should be shown. But, if the word to be searched is gui, then only the last entry (all language-translation pairs in this entry) should be displayed. If no word matches, a message should be shown prior to reverting back to the Manage Data menu.

- ***Search Translation***

  This option is similar to Search Word, except that the searching is of a specific language and translation. Meaning input for this option is both the language and the translation and the display are all entries where a language-translation appears. [For example, if the search is for Tagalog mahal, then both entries in the given example should be shown. But, if search is for Cebuano and gugma (the language and translation, respectively), then only the first entry in the example should be shown. If there is no match, a message should be shown prior to reverting back to the Manage Data menu.

- ***Export***

  Your program should allow all data 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 30 characters including the extension. If the file exists, the data will be overwritten.

  This is a sample content of the text file. Make sure to follow the format. Those that are in <> are supposed to be replaced with the character. No <> will be saved in the text file. No additional content should be in stored in the text file. The user should explicitly choose this option for all modifications done (within the options of Manage Data) to be reflected (or stored) to the text file.

  English: love<next line>
  Tagalog: mahal<next line>
  Hiligaynon: gugma<next line>

```
Cebuano: gugma<next line>
Spanish: amor<next line>
Chinese: ai<next line>
<nextline>
Tagalog: mahal<nextline>
Kapampangan: mal<nextline>
Cebuano: mahal<nextline>
English: expensive<nextline>
Chinese: gui<nextline>
<nextline>
<end of file>
```

Note that the sample content above, assumes a scenario where the user chose Export, immediately after a series of Add Entry. Should the user choose other options (like Add Translation or Delete Entry) where the display triggered the sorting, then the exported file will reflect the sorted result.

- ***Import***

  Your program should allow the data stored in the text file to be added to the list of entries in the program. The user should be able to specify which file (input filename) to load. If there are already some entries added (or loaded previously) in the current run, the program shows one entry loaded from the text file and asks if this is to be added to the list of entries (in the array). If yes, it is added as another entry. If no, this entry is skipped. Whichever the choice, the program proceeds to retrieve the next entry in the file and asks the user again if this is to be included in the array or not, until all entries in the file are retrieved. The data in the text file is following the format indicated in Export. [Based on the last note in the Export, **do not assume** that each entry in the file to be imported are already sorted.]

- ***Exit***

  The exit option will just allow the user to quit the Manage Data menu. Only those exported to files are expected to be saved. The information in the lists should be cleared after this option. The program goes back to the Main Menu.

Language Tools

The main meat of your project is to do machine translation and to identify the language of a given text. At the start of choosing the Language Tools menu option (from the main menu), the user is asked for the filename where the data will come from. This is similar to the Import feature in the Manage Data menu.

Once loaded, the following menu options are shown:

- ***Identify Main Language***

  In this feature, the program asks the user to input a phrase or sentence. Assume this input is at most 150 characters and could consist of many words. The program then splits this into words – this is called tokenization. Any comma, period, exclamation point, or question mark should be removed from the word. Assume that a space will separate the tokens. From these words, the program will determine what language it is by checking against the list of entries loaded from the file. Note that it is possible that the sentence used a combination of words from different languages, thus the result should be the one with the highest count. Note that it is possible that some of the words in the sentence cannot be found in the list of entries, in which case, these words are just ignored. For cases where the result is more than one language with the same count, the program can have any of these languages as the result. [For example, let's say the array of entries contain the given 2 example entries in page 1 and if the sentence is "mahal, expensive eh", the count for English is 1, Tagalog is 1 (not 2, since it is the same word), Cebuano is 1, then the program can produce English or Tagalog or Cebuano as the output language. If the example sentence is "mahal, mahal ito ha", then count for Tagalog is 2 (because mahal appeared in the input twice so this means checking is per word) and Cebuano is also 2; again the result of your program can be either of these languages. But if the example

sentence is “love, this is expensive!”, the count for found English words is 2, thus the result should be English. Last example, if none of the words are found in the array of entries, like “mi casa su casa”, then the result is a message to say that it cannot determine the language.]

The program goes back to the Language Tools menu after processing one text.

- ***Simple Translation***

In this feature, the program asks the user for the language of the source text, the text to be translated and the language to be translated to. Assume the text is most 150 characters and could consist of many words. Similar to the previous option, tokenization and removal of the symbols from the words need to be done. The translation need not include the symbols that were removed. For words that do not have corresponding translations, just use the same word as in the given text. If there is more than one translation, the first that it encountered in the array is the one to be used. [For example, if the source text is in English “i love ccprog2” and this is supposed to be translated to Tagalog, then the result is “i mahal ccprog2”. If the source text is in Tagalog “mahal ba ito?” and should be translated to English, the result should be “love ba ito”, assuming the first entry in page 1 is the first that the program found Tagalog-mahal with an available translation to English.]

After processing one text, the user is again asked if another text is to be translated from the same source and output language. If yes, a new text is asked. Once the user says no (maybe the user wants a different source or a different output language or maybe just wants to use other features of the program), then the program goes back to the Language Tools Menu.

- ***Exit***

The exit option will just allow the user to quit the Language Tools menu. The information in the lists should be cleared after this option. The program then reverts back to the Main Menu.

The program terminates when the user chooses to exit from the Main Menu.

Bonus

A **maximum** of **10 points** may be given for features **over & above** the requirements, like (1) producing top 2 of the languages when the program uses a mix of languages (not necessarily with equal count); (2) allowing user input of more than one word for the translations and processes different features like identifying the language and the translation considering multiple words; or other features not conflicting with the given requirements or changing the requirements) 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.

Submission & Demo

- **Final MP Deadline: June 17, 2022 (F), 1200noon via Canvas**. After the indicated time, the submission page will remain open for submission until June 20, 1200noon only but for every day late, there will be a 20% deduction in the MP grade. Note that any amount of time (even a few seconds after 12noon of June 17) will already be considered the next day. Thus, submissions from June 17 12:01PM to June 18 12:00PM (noon) will incur 20% deductions; submissions from June 18, 12:01PM to June 19, 12:00PM will incur 40% deductions; submissions from June 19, 12:01PM until June 20, 12PM will incur 60% deductions in the MP grade. At June 20, 12:01PM, the submission page will be locked and thus considered no submission (equivalent to 0 in the MP grade).

Requirements: Complete Program

- Make sure that your implementation has considerable and proper use of arrays, strings, structures, files, and user-defined functions, as appropriate, even if it is not strictly indicated.
- 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 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** 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**. Goto *label*, exit(), break (except in switch), continue, global variables, calling main() are **not allowed**.
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.

<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 6, 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 Canvas. 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.
7. Use **<surnameFirstInit>.c** & **<surnameFirstInit >.pdf** as your filenames for the source code and test script, respectively. 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. 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.
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:

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 Canvas 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/online during the demo, not just to present their individual demo problem solution, but also to answer questions pertaining to their group submission.

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, 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: June 17, 12:00noon, 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.

1.) Milestone 1 : April 22
    a. Menu options and transitions
    b. Preliminary outline of functions to be created

2.) Milestone 2: May 20
    a. Add Entry
    b. Add Translations
    c. Display all entries
    d. View Clues
    e. View Words
    f. Search Word
    g. Search Translation

3.) Milestone 3: May 30
    a. Modify Entry
    b. Delete Entry
    c. Delete Translation
    d. Identify Main Language[+]
    e. Simple Translation[+]

    [+]For these features, do not clear the contents of the array of entries upon exit from Manage Data menu IF you have not implemented the import and export yet.

4.) Milestone 4: June 10
    a. Export
    b. Import

5.) Milestone 5: June 17
    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