This project is an exercise in working with strings, arithmetic operations,writing subroutines (i.e., methods) and interfacing C and X86\_64 Assembly. You will write a menu driven program to manipulate strings as explained in this write up.

The program should store 10 messages for possible manipulation. These will each be initialized to the string: *“I love assembly language and CMSC 450 is my favorite class at UCB!”*

**Main function:**

This is where you collect user inputs and make calls to the other methods as requested by the user. This should be written in X86\_64 assembly.

When the program is launched, the menu must be displayed and the user can make selections. The program must continue to execute until the user chooses to exit by entering “e” or “E”.

Below is the description of each menu choice:

**“d” or “D”:**  Show the current 10 messages to the user, un-encrypted. These messages should all initially be set to: *“I love assembly language and CMSC 450 is my favorite class at UCB!”*

**“u” or “U”:** Get a new message from the user. Validate the message to make sure that it starts with a capital letter and ends with either a period ('.'), question mark ('?') or exclamation point ('!'). If the input is invalid, reject it with an error message to the user. Otherwise, if valid, replace the message in the next spot in the array starting with the first spot (index 0). Once all messages in the array have been replaced, the program should reset to the beginning position and start replacing from there.

**“f” or “F”:** the user must be prompted for a string in the array they would like to work with. Once picked they are then prompted with the message “Please enter the character sequence to find: ” and user input must be collected. Next the message “Please enter the replacement sequence: “ must be displayed and use input should be collected. Then, all occurrences of the first entered character sequence in the text must be replaced by the second entered sequence. For example, if the current value of text is “An apple a day keeps the doctor away!” and the first entered input is “ay” and the second one is “by”, the text string must be changed to “An apple a dby keeps the doctor awby!”.

**“t” or “T”:** If the user selects this choice, they should be prompted for which string in the array they want to transform and a random transformation must be done on the text by calling one of the reverseHalves or scramble operations on it. *The selection of which operation is applied must be done randomly.* The implementation of each of these operations must go into its corresponding helper function, but then you can make calls to them from main. Transforming the string should not change the original string. Here is the explanations for each of these methods:

* reverseHalves: The idea is to divide the input text into two halves, then reverse each half and finally put these halves back into one string. For instance, if the input text is “An apple a day keeps the doctor away!”, after applying reverseHalves to it the result would be: “eek yad a elppa nA!yawa rotcod eht sp”. reverseHalves does not change empty, single character, and two-characters-long texts.

* scramble: This one “scrambles” the text by putting first and last characters next to each other, then it puts the second and the second last characters next to each other, then the third and third last, so on so forth all the way to the middle point of the text. As an example, if the text is “An apple a day keeps the doctor away!”, after scrambling it we would have “A!ny aawpap lreo tac odda ye hkte esp”. scramble does not change empty or single character texts.

**“e” or “E”:** If this one is selected, the program simply terminates without doing or displaying any further messages!

**Note 1:** At any point during the execution if a choice other than the ones described above is entered, the program must show the message “Invalid option! Try again!”, print the menu options and let the user make a choice again.

**Note 2:** See the sample run posted on blackboard. Once you finish the implementation, make sure you can reproduce the sample run.

**Implementation Details & Hints:** There are several key features of this program solution that are required and/or will make your development job much easier. Here is a list of requirements and implementation hints:

* You must break this problem down into subroutines rather than writing everything in main. The required subroutines are;
  + a read message subroutine,
    - **This should be written as a C function**
    - This is called when the user wants to read in a new message
    - It should validate the user’s input using the criteria mentioned above
    - The string the user enters could be any size so you need to account for this. (*Hint: dynamic memory allocation*)
  + a reverseHalfst subroutine,
    - **Should be written in x86-64 assembly language**
    - Should be in a separate file by itself
  + a scramble subroutine,
    - **Should be written in x86-64 assembly language**
    - Should be in a separate file by itself
  + a display subroutine
    - **Should be written as a C function**
    - Displays the current strings in the array
  + A replace subroutine
    - **Should be written as a C function**
    - Used to find and replace a sequence in a string
  + The C functions can be in one .c file.
  + Feel free to add any other subroutines as needed.
* **All code written must be by you and written in assembly aside from the subroutines in C mentioned above.**
  + \* If needed you can have the free command in it’s own subroutine in C.
* **You must program for x86-64 (64 bit architecture).**
* **The transformations should not change the original string being transformed.**

* It is really important to develop this program incrementally! Build your program from the bottom up: Have your main method display the menu and read the user's choices. For each choice, just display a brief message so that you can see that the menu works. Make sure the quit option works and that users can select various options in any order. Next, add the necessary variables for processing, including the initial value for the current message.
* One at a time, implement the various menu operations, testing each with the initial message to be sure they work correctly. Then test them with other inputs, using the menu operation to get new messages.
* You will need to submit the following files in a zipped folder labelled **firstname\_lastname\_project4.zi**p:
  + main.asm
    - Main assembly file that calls the subroutines
  + validate.c
    - C file with c functions
    - You can have your C functions in separate files as well but will need to submit a separate makefile in this case.
  + reverseHalves.asm
    - shift encrypt subroutine
  + scramble.asm
    - jump encrypt subroutine
  + You can submit any other .asm/.c subroutine files, however, you need to make sure you submit a Makefile that assembles/compiles and links all these files.
  + If working in a group of 2, you submission should be **lastname\_lastname\_project4.zi**p and only one person needs to submit
* You must have a comment at the top of the code detailing what the code does.
* This comment should also include your full name and user ID.
  + If working with 2 people both names and user IDs should be included.
* Any late submissions will incur a penalty of 10 points per every 6 hours they are late.

**Grading Breakdown: (remember that code that doesn’t compile gets a 0 grade)**

The 100 total points for this part will be broken down into expected functionality (how well the code works), and how well the code is written, including style and documentation. You are not required to submit pseudocode for any parts of this project, but you are strongly encouraged to write them as part of the solution process. The points for each part include the expected error handling.

* [90] Program functionality:
  + [ 10] Overall main processing of menu and user input.
  + [20] Current messages
    - Initialized
    - updated
    - used properly
    - display operation working.
  + [20] Input and validate new messages.
  + [20] reverseHalves and Scramble
  + [20] Replace

* [10] Program style and adherence to specifications:

**Easter Egg (Extra Credit):**

As a way to earn extra credit for this project, you can implement an easter egg in your project. An easter egg is a hidden feature in the project.

This easter egg should be implemented in the following way:

* While on the main menu, if the user enters the letter ‘c’ as an option 4 times, the easter egg should be activated.
* Each time the user enters ‘c’ but before the easter egg is activated, the program should print an error message letting the user know that they entered an invalid option.
* The easter egg can be whatever you want it to be, a message printed to screen, a special feature in the program, a game and so on. Whatever you like.

Successful implementation of this feature will earn you a minimum of 5 extra points on your project. Depending on how creative your easter egg is, you can get more points up to 10 total points.