Due Date : Due on 5/15 11:59 pm

Project Description

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: *"This is the original message."*

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 "q" or "Q".

Below is the description of each menu choice:

"s" or "S": Show the current 10 messages to the user. These messages should all initially be set to: *"This is the original message."*

"r" or "R": Read 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.

"c" or "C": If the user selects this choice, they should be prompted for which string in the array they want to encrypt using the Caesar cipher. The program will then take the string of choice and shift every letter in the message by the indicated number of positions in the alphabet. When a letter gets shifted past the beginning or end of the alphabet, wrap around to the other end to continue. For example, a 6 would shift any 'e' characters to 'k'. The shift must be applied to all letters in the message. Do not change any non-alphabet characters in the message, and preserve all capitalization.
This will be the option that implements your project 3 but you might need to make some edits to your project 3 to make it fit in this program.

`"f"` or `"F"`: This option should ask the user what string they want to decrypt and then implement a decryption scheme known as frequency analysis on that string. You can read more about this method at this link: Frequency Analysis Example
For this program, we will implement a very basic and not so accurate method of frequency analysis. Our method will involve analyzing a string and picking the 5 most common letters in that string.
We will then produce 5 possible decryptions of the string by:

- Decryption 1 will be from comparing how far the most popular letter is from the letter “e”
- Decryption 2 will be from comparing how far the second most popular letter is from the letter “t”
- Decryption 3 will be from comparing how far the third most popular letter is from the letter “a”
- Decryption 4 will be from comparing how far the fourth most popular letter is from the letter “o”
- Decryption 5 will be from comparing how far the fifth most popular letter is from the letter “i”

See the sample run for examples.

`"q"` or `"Q"`: 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 for an example of this program.

Note 3: Make sure you follow the rules outlined in the Implementation Details & Hints section below.

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 caesar subroutine,
        - **Should be written in x86-64 assembly language**
        - Implements the caesar cipher on a string from the array that the user picks and saves the changes made to the string
        - 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 decryption subroutine
        - **Should be written as a C function**
        - Used to decrypt a string of the user's choice. The decryption does not store changes to the string but simply displays them.
    - 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 its own subroutine in C.

    - The following subroutines will be in C:
        - read
        - display
        - decrypt
        - free
    - **Everything else MUST be done in assembly**
    - **Use of compiler generated assembly code will be counted as a violation of UMBC code of ethics and considered cheating.**

- Declaring and initializing the array of strings **MUST** be done in assembly.
- **You must program for x86-64 (64 bit architecture).**
- **The caesar cipher will change the string in the array but decryption does not.**
- 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 labeled **firstname_lastname_project4.zi**p:
    - main.asm
        - Main assembly file that calls the subroutines
    - cFunctions.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.
    - caesar.asm
        - shift 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. **(LATE DAYS WILL NOT BE APPLIED TO THIS PROJECT)**

Grading Breakdown: (remember that code that doesn't compile/assemble gets a loses 50 points)

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
    - [5] caesar cipher
    - [20] Decryption
    - [15] Dynamic memory allocation and free

- [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 'z' as an option 4 times, the easter egg should be activated.
- Each time the user enters 'z' 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.