CMSC 313 Project 2
Fall ’18 Due: 10/02/2018 @ 11:59pm

Objective

The objectives of the programming assignment are 1) to gain experience writing larger assembly language programs, 2) to gain further experience with branching operations, and 3) to practice some indexed addressing modes

Background

In computer science, string-searching algorithms, sometimes called string-matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text wikipedia .

The most basic case of string searching involves one (often very long) string, sometimes called the "**haystack**", and one (often very short) string, sometimes called the "**needle**". The goal is to find one or more occurrences of the "**needle**" within the "**haystack**". For example, one might search for "**needle**" within:

Pattern: in a haystack, needle in a haystack, needle in a haystack
text: needle

"**needle**" appears after 14 characters from the first character in the pattern.

Although the text to be search appears more than once, a simple substring search algorithm would indicate only its first appearance in the pattern.

Your goal is to determine the **first** occurrence of the text to be searched (specified via standard input) within the following text pattern (provided as a part of the code snippet). The text to be searched can include white spaces and punctuation and may even be a partial string within a word (substring).

```
record:
row1:   db "Knight Rider a shadowy flight"
row2:   db "into the dangerous world of a"
        db " man who does not exist. Mich"
        db "ael Knight, a young loner on "
        db "a crusade to champion the cau"
        db "se of the innocent, the innoc"
        db "ent, the helpless in a world "
        db "of criminals who operate abov"
        db "e the law. Knight Rider, Keep"
        db " riding brave into the night."
```

Assignment

Your assignment for this project is to write an assembly language program that computes the **first** occurrence of a string (provided via standard input) within **text embedded** within the code snippet. Your program may execute like this:

```
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: K
Text you searched for, appears at  0 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: Rider
Text you searched for, appears at  7 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: er,
Text you searched for, appears at  253 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: ht.
Text you searched for, appears at  287 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: anger
Text you searched for, appears at  39 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: criminals who operate above the law
Text you searched for,  appears at  206 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: cent
Text you searched for,  appears at  159 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: Knight Rider,
Text you searched for,  appears at  243 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: a crusade
Text you searched for,  appears at  116 characters after the first.
[deepakk1@linux3 ~]$ ./textSearch
Enter search string: random@3456
String not found!
```

Implementation Notes

1. A good starting point for your program is the textSearch.asm program shown in class. It already queries the user for input and prints out digits for a number in the register RAX. The source code for textSearch.asm is available in the class's blackboard via the navigation pane, at:

   Assignments -> Projects -> Section 02 -> Project 2 -> textSearch.asm

2. Your can assume that no more than **100** characters will ever be input to your program. The text to be searched can include white spaces and punctuation and may even be a partial string within a word (substring).

3. Although you may modify the embedded text within the provided code snippet, you are **required** to preserve the text **as is** when you make your final code submission. **Your code will be evaluated on test cases that presume the original embedded text is preserved with no modifications.**

4. Remember, all characters input into the system are ASCII values. Refer to an ASCII table when determining how to implement your code.

5. Your code must be case-sensitive: if the input is upper-case, then the corresponding output must be upper-case; if the input is lower-case, then the corresponding output must be lower-case, as in the following example (also above):

   ```
   [deepakk1@linux3 ~]$ ./textSearch
   Enter search string: Rider
   Text you searched for, appears at  7 characters after the first.
   ```

6. If the input string was not found, then your code should print out the following message

   ```
   [deepakk1@linux3 ~]$ ./textSearch
   Enter search string: random@3456
   String not found!
   ```

7. To make things easier, you are also provided a code snippet that prints out the digits of the location found. You are allowed to change/optimize the provided code if required.

8. There are many ways to implement your **search** routine. Although, you may be creative in your implementation, points are awarded, on a 'correctness' basis, where you are supposed to clear all provided test cases. You may use indexed addressing instructions and conditional loops to achieve the objective. You may also explore efficient techniques for searching substrings such as the KMP Algorithm. However, since efficiency is not the goal, you may use a naive string search technique. One aspect that you should definitely consider, is that your code should produce results in a few seconds (not minutes, or hours at a stretch!).

9. It may be helpful to write the code to perform the **Substring Search** in a higher-level language of your choosing (C, C++, Java, Python, …). This will help you understand the mechanics of your assembly language program. ***This is not required and you should NOT submit this if you choose to do it.***

What to submit

Use the UNIX submit command on the GL system to turn in your project. You should submit the assembly language program, the readme file (Documentation) and typescript file that demonstrates a sequence of test runs of your program taking in user input and printing out the location of the string if it exists. The class name for submit is CMSC313_Deepak and the project name is `proj2`. The UNIX command to do this should look something like:

```
submit CMSC313_Deepak proj2 textSearch.asm Readme.txt typescript
```