# CSE 2421 Lab 6

## Learning strings and

## command line arguments in Assembly

## Due: Thursday 27 July 2023 11:59 PM
## Early: 2 bonus points if submitted by Tuesday 25 July 2023 11:59 PM
## Late: Saturday 28 July 2023 11:59 PM

We'll be using Carmen's timestamps to determine lateness or not. If you are not enrolled, you may send me an email with the zip file. If you are unable to submit by the deadline, you need to send me a screenshot of your finished files with the clock visible and then *still* submit to Carmen when you can.

This lab contains three parts. Make sure you do all of them!

Something you should do before getting started on the lab is creating your readme file. Simply create a file called "readme" and include your name at the top. As you progress through the lab, you may see different parts (labeled things like **Part 1**). When you see this, write it down in your readme. While in a numbered part of a lab, you'll come across various questions, prepended with a bolded Q and a number (such as **Q5**). When you see this, include the Q and the number, along with you answer to the question(s) in the file. A brief example:

> Name: Rob LaTour
>
> I certify that I completed all of the work myself with no aid from anyone aside from the instructor or the undergraduate graders.
>
> Part 1:
>
> Q1: Don't make fun of me, but my favorite color is gray
>
> Q2: The definitive answer is that Han did shoot first. The original version is the one that captured the heart of the public, and thus the director lost creative control. In this essay I will…

**Part 1: A simple string problem**

For this lab, we will be implementing some functions that manipulate and modify strings. This lab has almost no questions to answer in the readme, it's all about delivering on the code! First, you'll need a set of strings to manipulate.

**Q1**: In the .section .rodata section of your assembly program, create five strings:
"I'm big fan of small word. No like big word or good grammar. Grammar too hard."
"I really hate how nice the weather is getting."
"What"
"Amazing, isn't it? With just the slightest touch the material warps around the skin effortlessly."
"Mmm, ice cream so good!"

These strings exist for you to test your program, but **note**: the graders will test your code with a set of strings different in number and contents! It is important that your program actually *does* the calculations described here. **Seriously. Do not assume that there will only be five strings or that they will have exactly these contents when we test your files**.

**Q2**: In order to facilitate this, create a section of code that has very vibrant comments around it that can be quickly and easily edited by you or the graders to allow for more or fewer strings to be added into an array on the stack or the heap. This chunk of code should basically create an array with a number of elements equal to the number of strings (which is 5, for now), then should simply put the address of each string into the array. Place a copy of the strings and your instructions for modifying your code in the readme under **Q1** and **Q2**.

It should be incredibly easy for this code to be modified by you or the graders (and thus should have very nice comments describing exactly how to do so).

Once all the setup is finished, you can begin the real work: determining which string is the shortest, and printing that to the console. To accomplish this, you *must* create a function that does calculates the shortest string and prints it. You may not simply do this in main!

**BONUS (2 points)**:

Create a constant at the top of your program that holds a numeric value. For 0, have your function from above simply print the shortest string. For 1, have either the same function or a different function (just don't use main) print all of the strings on separate lines starting with the shortest, then the next shortest, and then the next, and so on.

**Part 2: A slightly more complex string problem**

For this next part, you will create another function that will determine the string with the fewest words and print that to the console. You can assume that the only whitespace used to

separate words will be the space character. This function should also put the number of words in the string into %rax (meaning that it "returns" the number of words).

**BONUS (2 points)**:

Create another constant at the top of your program that holds a numeric value. For 0, have your function from above simply print the string with the fewest words. For 1, have either the same function or a different function (just don't use main) print all of the strings on separate lines starting with string with the fewest words, then the next fewest words, and so on. Furthermore, these lines should be prepended with the number of words in the string. An example output line might read "Words 5: Mmm, ice cream so good!"

**Part 3: And now for something different**

In the final portion of the lab, you will add an entirely different piece of functionality to your program. If run with no command line arguments (beyond the program name), your code should perform the above tasks. You may assume that any command line arguments past the name of the program are the names of C data types. Your task is to provide alignment information to the user as if the arguments to your program were the members of a structure. That is, your program should explain exactly how large the structure is and how many bytes belong to each data type, along with any spacing. Finally, your program should output the alignment of the entire structure.

You may assume that only primitive types are provided as input and that the user will not input anything that is not a C type.

**BONUS (5 points)**:

Expand the functionality to include arrays and structs in the above.