CS/CoE 0447 Spring 2017
Lab 4: String Manipulation, Tables, and Simple Function Calls

Released: Thursday 2 February 2017, 11:00am EDT
Due: Monday 12 February 2017, 11:59pm EDT

Each of you should submit your own solution, according to the instructions at your TA's website. Each person must turn in their own copies of the lab. If you choose to work with a neighbor/partner, put your partner's name on your submitted copy of the lab. Submission timestamps will be checked. Late submissions will not be accepted.

For each of the **two parts** in this lab, you must name your files in a specific format. Each part will tell you the name to use. Follow instructions so that your program produces output in the correct format. Submit your files through CourseWeb.

Part 1: Strings (Modifying in Place)

The following lines store a null-terminated string into the data segment:

```
.data
some_string: .asciiz "Surprise Tapestries in the Mail"
```

Each character of the string is stored in one byte as that character's ASCII code. For example, the first byte (at address 0x10010000, which is the right most part of the box in MARS) contains the value 0x53 (83 in decimal), which corresponds to the letter "S" in ASCII code. (You can find a list of the ASCII codes on the second page of the MIPS "green sheet" reference linked on CourseWeb or elsewhere online, for example, at http://www.asciitable.com). The last byte allocated to the string contains the value 0x00, which identifies the end of the string. Mars automatically adds this 0x00 (null) byte to the end of a string when you use the `.asciiz` directive.

Start by writing a function **'**ReplaceLetterWithAsterisk' that takes two arguments as inputs. Use the MIPS calling convention; use `$a` registers to pass the arguments to your function. The first argument ($a0) contains the string address and the second argument ($a1) contains the character to replace. The **'**ReplaceLetterWithAsterisk**'** function will then find all instances of the letter in your string and replace them with an asterisk.

Next, write a function 'Rotate13' that takes the address of a string and obscures the message by "rotating" the letters by 13. That is, adds 13 to their encoded value while looping around the alphabet. "A" becomes "N", "B" becomes "O" and so on. The same for lowercase values ("c" becomes "p", etc) Since adding 13 to some characters would result in a character after "z", values wrap around such that "N" becomes "A" and "Z" becomes "M". Essentially, the value must always be restricted to fall within "A" and "Z" (0x41 and 0x5A) or "a" and "z" (0x61 and 0x7A). This is a type of shift cipher more generally known as a Caesar cipher. (for an interactive reference, see: http://rot13.com) Use the MIPS calling convention to pass the argument to your function.

Now, use these functions to write a complete MIPS program that asks the user for a string and a character, replaces all instances of the specified character with '*' and then rotates the string to obscure

the message. Then print the string to standard output (using a syscall). To call your 'ReplaceLetterWithAsterisk' and 'Rotate13' functions, use the jump and link (`jal`) instruction.

Your TA will test your program on several strings of sufficient length (but no more than 63 bytes). These strings will contain only alpha-numeric characters, asterisks and spaces.

Here is sample output from the program:

```
Please enter your string:
Surprise Tapestries in the Mail <--- this is the input
Please enter the character to replace:
e <--- this is the input
Here is the output: Fhecevf* Gnc*fgev*f va gu* Znvy
```

Please make sure your output line contains the string **"`Here is the output: `"** as shown in the sample output.

Hint 1: Each character of a string is a byte. We need a character buffer (memory space) to hold the input string in memory. To declare memory space for a 63-byte string and its null terminator, you can use the following code:

```
.data
some_string: .space 64
```

You can use **syscall 8** to input a string.

Hint 2: In an ASCII listing ordered by ASCII value, all uppercase letters are adjacent to each other. Similarly, all lowercase letters are adjacent to each other. So, you can simply check whether the value of a given byte falls within a specific range and you'll know it is an uppercase/lowercase letter and not a space or other character. To see how to implement "if statements," i.e., conditional branches, in MIPS assembly language see:

http://people.cs.pitt.edu/~childers/CS0447/lectures/mips-isa3.pdf

Once you have identified it as a character, to rotate the letter you simply add 13. Yet adding 13 to "T" or "Z" etc would not give us another letter. In those cases (for letters greater than or equal to "N"/"n"), we have to *subtract* a certain amount instead. "Z" (0x5A) should map to "M" (0x4D) What is the difference between "Z" and "M"? It may be useful to write out the conditions in code of a higher level language (Java) and then translate by hand. **Note**: Re-running the program and giving the previous output should yield a string that resembles the original! Neat.

Question 1: Submit your program "**lab04part1-<your Pitt Mail ID>.asm"**. For example, "**lab04part1-xyz12.asm" (**Instead of your PeopleSoft number, be sure to use your Pitt Mail ID.) Output format will be strictly checked.

Part 2: Using Tables

In this part, you will declare a dictionary (a table-like data structure) by defining two arrays in parallel as follows:

```
.data
names:   .asciiz  “alex”, “sam”, “jamie”, “andi”, “riley”
cities:  .asciiz  “boston”, “new york”, “chicago”, “pittsburgh”, “denver”
```

This dictionary lists names of five people in the *‘names’* array and the city they live in in the ‘*cities*’ array. You need to write a program that takes a name (as an input string) and prints out the city that person lives in. Here are two sample outputs:

```
Please enter a name:
kate <--- this is the input
City is: pittsburgh
```

```
Please enter a name:
jim <--- this is the input
Not found!
```

You may start with outlining an algorithm and then start your implementation. For each of the names of the array *‘names’*, you should write a function **‘CheckName’** to compare the name with the input string. **‘CheckName’** takes the addresses of two strings and returns 1 if those two strings match or returns 0 otherwise. Again, use the MIPS calling convention; use argument registers ($a) to pass arguments to your functions; use value registers ($v) to return values from your functions.

As you are checking for a name match, keep track of the current index of array *‘names’*. For example, *“alex”* is at index 0, *“jamie”* is at index 2 and so on. If you get a name match, you can use the index to find out the corresponding city from that index of the *‘cities’* array using another function called **‘LookUp’**. The function **‘LookUp’** takes the address of a string array and an index, and returns the address of the string at that index.

You also need to keep track of the address of the current string in the ‘*names*’ array. To increment this address by the correct amount after each loop iteration, you can use another function called **‘StrSize’**. The function **‘StrSize’** takes the address of a string and returns the number of characters before the next *null* (0x00) character (the size of the string). When you give it “riley”, for example, it would return: **5**

If you reach the end of the *‘names’* array, then there is no match of available names with the input name. In that case your program will print *“Not found!”* and exit.

Here is a pseudo-code outline of the flow:

```
input = (user input)
index = 0
names_str = names
while (True) {
        result = CheckName(name_str, input)
        if result==1 {
                theCity = LookUp(cities, index)
                print "City is: ", theCity
                exit
        } else {
                index = index + 1
                if index==5 {
                        print "Not found!"
                        exit
                }
                name_str = name_str + 1 + StrSize(name_str)
        }
}
```

Note: Syscall 8 (https://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html) has a footnote that you should be aware of. Like many string input functions in various programming languages, this will give you literally what a person has typed. The last character (the newline character) might not be something you can use, and you'll have to remove it before you check it against another string.

Question 2: Submit your program "**lab04part2-<your Pitt Mail ID>.asm"**. For example, "**lab04part2-xyz12.asm" (**Instead of your PeopleSoft number, be sure to use your Pitt Mail ID.) Output format will be strictly checked.