CSE30: Computer Organization and Systems FA20
Assignment 6
Due: Wednesday, December 2nd (time TBD)
Instructors: Leo Porter, Keith Muller

Intro: Translate command line arguments to upper case as needed and print to stdout

Requirements

1. You will write this program using only the arm-32 instructions listed on the CSE-30 green card
2. You will process each argument in order from argument 1 to argument n.
3. For each argument you will convert each lower-case letter (letters between lower-case 'a' and lower-case 'z' inclusive) to its upper-case equivalent, all other characters will remain unaltered.
4. You will not print any characters in an argument that falls outside the 7-bit ascii printable range from the space (0x20) character to ~ (0x7e) character.
5. You will print each argument after processing to stdout.
6. You will print one space between each argument printed.
7. A single argument that includes any whitespace (e.g., space, etc.) will have that whitespace preserved.
8. The last argument printed will be followed by a single newline.
9. At normal program termination EXIT_SUCCESS (0) will be returned to the caller
10. The program will terminate and return EXIT_FAILURE (1) to the caller if any libc library function(s) you use has a failure return.
11. You will write your program starting with the supplied template for shout.s
12. If there are no arguments, a newline should not be printed.

Use of Linux library functions

You may only use any number of the following Linux library functions in your code

```
putchar()
putc()
puts()
```

Sample output

```
%./shout tHis iS a TeSt of 44 trumpets at OnCe!
THIS IS A TEST OF 44 TRUMPETS AT ONCE!
%./shout "This   -  is just one arg! "
THIS   -  IS JUST ONE ARG!
%./shout 1 2 3 4 "12 daNcerS"      Dancing
1 2 3 4 12 DANCERS DANCING
%./shout "@ # $% ^&* ()a+{}"
@ # $% ^&* ()A+{}
```

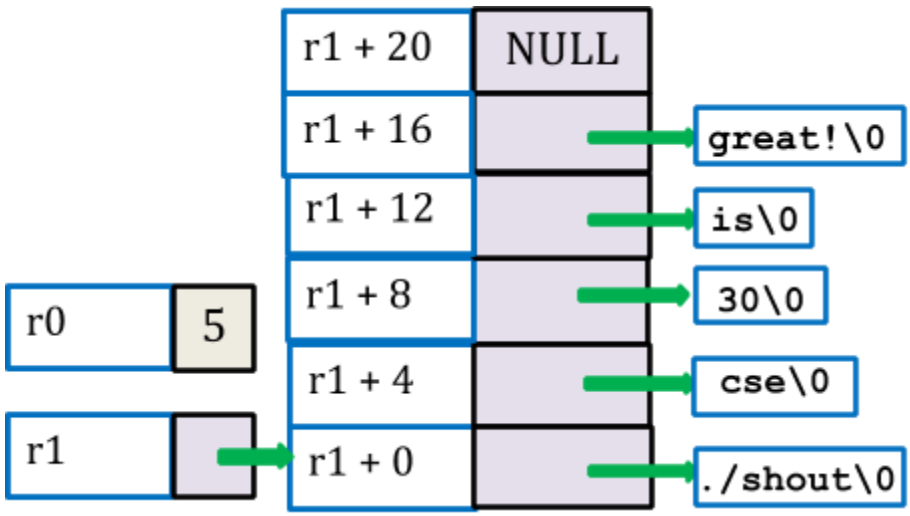
Note: If you try to pass chars like #&* etc. as args, the command interpreter might process them in a different way since they have their own meanings on the shell. The program will never see them unless you have them in between the quotation marks like the example above.

How to access a Linux library function from arm-32 assembly

1. Export the function using an .extern assembly directive at the top of your function
   `.extern putchar // example`
2. Follow the directions on the man page for the `putchar`, `putc`, and `puts` function(s) in terms of parameters. See below for additional details on `putchar`.
3. Call the function with bl and normal parameter passing conventions.

Accessing argv from arm-32

Main() is passed argc in r0 and argv in r1. r1 contains a pointer (the address of) to the base array of pointers. Each pointer is 4 bytes wide.



To get the pointer for argv[0] which is pointed at by r1 you would write the following in arm

```
ldr   r3, [r1] //r3 contains a pointer (address) to the ./shout string
```

To get the first character of the argv[1] string the next line would be

```
ldr   r3, [r1, 4] //r3 now points at the string "cse"
ldrb  r2, [r3]    //r2 now contains the first character of argv[1] 'c'
```

To test for the end of the argv array you would write

```
ldr   r3, [r1]
cmp   r3,  NULL  // where .equ NULL, 0 else where
```

Writing to Stdout with `putchar`

In order to write to stdout you will want to use the function `putchar` which will write one character at a time. The function signature is:

```
int putchar(int char)
```

The function expects a character which it will output to standard out. If it is successful, it returns the numeric value of the character written. If unsuccessful, it returns EOF (-1). Your program should exit with "EXIT_FAILURE" if you are unable to write a character.

Getting started

1. Download the starter code (shout.s) from here.
2. Assemble your program with gcc shout.s -o shout
3. Start editing. You may want to try to make sure you can output a character to start, then start working with argv.
4. Feel free to write helper functions before main if you find the code complexity to start getting difficult to follow. You are not required to have a helper function, but if you do so, be careful about properly handling calling conventions.
5. Note that you are not limited to writing code to the region below "ADD YOUR CODE HERE". You can change code above and below as needed.

Survey for week 8

Complete this survey about your experiences this term. Double check your PID when entering to ensure you receive credit.

Submission and Grading

1. Submit your files to Gradescope under the assignment titled Homework6. You will submit the following file: shout.s

2. After submitting, the autograder will run a few tests:
    a. Checks that all required files were submitted

Make sure to check the autograder output after submitting! We will be running additional tests after the deadline passes to determine your final grade.

The assignment will be graded out of 16 points, with 12 points allocated to the shout program(autograded). 3 points dedicated to style (see above ARM assembly style requirements). 1 point for completing the survey. Make sure your assignment compiles correctly on the ieng6 (pi-cluster) machines. **Any assignment that does not compile will receive 0 for the autograded part.** We will be comparing the standard out of your program to ours, so please ensure nothing else is printed than what is printed by the provided main.