HW4: ASCII Strings & Procedures

This assignment gives you practice with procedures (functions) and strings of ASCII characters in MIPS. You will work with your current partner. As usual make sure **both of your names are at the top of your file** and that you comment your code!

Deliverables: The file *user1_user2.asm* containing your solution program.

Code Provided

I have given you an initial program in start.asm with some functions and a main function to test them. Here are details about the procedures I provide:

`readline` – takes 1 argument which is the memory address at which to store the string that is read. Reads all text that is input until <enter> is pressed (up to 99 characters), storing each byte including the <enter> itself at the address given. A null character (simply the value 0) is added to the end of the string. If string was "!q" returns 1, otherwise returns 0.

`writeline` – takes 1 argument which is the memory address of a null-terminated string. Writes the string to terminal, returns nothing.

`writechar` – takes 1 argument which is the ASCII value of the character to write. Writes that character to the display and returns nothing.

You should **not modify the code** in the 3 procedures above, but you may use them to create the loop in main (described in the later section of this description on modifying main).

Your Task

You will add 3 new procedures and modify the main function as described below. Make sure you use the exact names given, the grader will both run your main function and use a main function I provide as part of the testing. They may also swap my version of any of the 3 new procedures in for yours. This means **you must follow all of the standard conventions for using registers** in functions. You can trust that my main function will also follow all of the standards, but otherwise you don't know anything about it.

New procedures

`getLength` – takes 1 argument which is the starting memory address of a null-terminated string. Returns the number of characters in the string, not including the null terminating char. Remember that when you enter a string on the terminal the newline character is included in the string, e.g. "abcd\n" would return a length of 5, where "\n" denotes the <return> character.

`printMiddleChar` – takes 1 argument which is the starting memory address of a null-terminated string. Finds the middle character in the string (if the string has an even number of characters either of the middle characters is acceptable), then prints it by calling the writechar function. Also RETURNS the ASCII value of the middle character. This procedure MUST CALL the getLength procedure as part of finding the middle char, and MUST CALL the writechar function to actually print the char before returning it.

`reverseString` – takes 1 argument which is the starting memory address of a null-terminated string. Returns the starting memory address of a string that is the reverse of the original string, except that if the last character of the original string is a newline, you should keep that newline at the end of the reversed string. For example, given the string "hello\n" where "\n" represents a newline, the reversed string should be "olleh\n". This procedure should leave the original string untouched, and place the reversed string in memory at the start of the next word after the original string. For example, if the original string is "hello\n" at memory address 0x10010000, the reversed string should begin at address 0x10010008, even though the original string is only 6 characters long.

Update to main program

You should create a loop in the main function that continues until the user enters "!q" as their string. The loop should ask the user to enter a string, print the length of the string, print the middle character of the string (on the next line), and then print the reverse of the string (on the line after the middle character). If the user enters "!q" the program should quit without printing anything further. Use the readline function I provide to create your loop, it already does the work of checking if the string is "!q"!

HELPFUL NOTES

1. The instructions lb (load byte) and sb (store byte) are useful when working with ASCII characters. They follow the same format as lw and sw, but only load/store a single byte. Both of these instructions always use the last byte of a register to load into or store from.

2. MARS has an option to show the memory in the data section in ASCII, just check the ASCII box at the bottom of the data segment window. However, it turns out that MIPS can handle either big or little endian memory, and MARS chooses to simulate little endian (with no option for big endian). This means that when you store a string in memory, then look at that memory as displayed by MARS, each set of 4 characters is shown in reverse order.

   So if what you have stored in memory is the following bytes:

0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
a	b	c	d	e	f	g	h

   Mars will display it like:

Address	Value (+0)	Value (+4)
0x0	d c b a	h g f e

   This doesn't affect the memory itself at all nor should affect how you think about writing your program, but if you are looking at the MARS memory display in order to help your debugging, make sure you don't let this throw you off!