COMP 273 Assignment 2

Prepared by Prof. M. Langer

Posted: Thurs. Feb. 11, 2016
Due: Sun. Feb. 21, 2016 at 23:59

General Instructions

- **TA's handing this assignment** are Ben (tzu-yang.yu@mail.mcgill.ca) and Hasan (sh.mozafari@mail.mcgill.ca). Office hours and locations will be posted on the course web page.

- **Late assignments** will be accepted up to only 3 days late, and will be penalized by 20 percent per day. If you submit one minute late, we reserve the right to treat this as equivalent to submitting 23 hours and 59 minutes late, etc. Verify that your submission was submitted correctly. If it was not, then your assignment will be considered late.

- **Bonus points:** If you discover ambiguities or minor errors with the assignment, please notify us immediately. If you are correct in identifying problems, you will be eligible for bonus points.

- **Plagiarism:** We encourage you to discuss the assignment with each other, but the discussion should be public in the sense that anyone should be able to listen in. You are not allowed to tell each other how to do the assignment, although some help is permitted e.g. if you are stuck on a small detail. Under no circumstances should you show solutions to each other. Any strongly suspected cases of plagiarism will be reported to the Faculty of Science disciplinary officer.

- You must properly comment your code. Points will be taken off for code that is difficult to follow or not commented.

- The programs will be tested only with valid inputs. For example, you do not need to test if maximum string length is negative, or if the user has entered too many strings**, [ADDED Feb 16] or if the total lengths of the strings (including the null and linefeed) has exceeded the allotted Memory space. When inputs are invalid, the program will crash or produce errors. It is instructive for you to examine these edge cases, but we are not asking you to deal with them.**

- Submit your code as a single file in a zip directory (as you did with A1).

Introduction

This assignment introduces you to MIPS programming using MARS. You can download MARS from the link on the public course web page (top right).

The main goal of this assignment is for you to gain experience with MIPS Memory, and how addresses are used. We will work with arrays of strings. A string is just a sequence of ASCII characters that is terminated by a null character (ASCII value 0).

Part 1: user enters a sequence of strings (30 points)

Your task in this part is to add instructions to the starter code which ask the user to input a sequence of strings. The starter code includes examples of how to use MIPS system calls to read from and write to the console. You can read about MIPS system calls from the MARS Help.

Your solution must use a loop. For each pass through the loop, a message should be posted in the console window asking the user to enter a string. The user types the string and then hits <return> to enter the string. To facilitate grading, we insist that the program echos the string by printing the entered string to the console.

As the user enters strings, the program should keep track of how many strings have been entered, or equivalently, what is the array index for the current string. The user indicates he/she is done entering strings by hitting <return>.

Here is an example of console I/O. The black text is the programmed output from the system calls. The red text is entered by the user. The green text is echoed. (These colors will not appear in MARS.)

ENTER MAXIMUM LENGTH OF A STRING: 40
ENTER A STRING : I love assembly language.
I love assembly language.
ENTER A STRING : Even more than I love Java or C.
Even more than I love Java or C.
ENTER A STRING : I am so glad that I am taking COMP 273
I am so glad that I am taking COMP 273
ENTER A STRING : because I am learning so much.
because it is such fun.
ENTER A STRING : <return>

Note that, according to the starter code, your array can reference up to 5 strings and the space allocated to hold all of your strings is 100 bytes. Each string that you read in must use only as much memory as that string needs, however. e.g. a string can be less than or greater than 100/5 = 20 bytes.

Part 2: Printing out the list of strings (20 points)

When the user has finished entering the sequence of strings, the program must print out the list of the strings that were entered. Before each string, the program should print the index of the string and some punctuation and text to make the output easy to read. This punctuation is given in the starter code.

Here is an example of output after the user hits <return> in Part 1. Note that the string order corresponds to the order in which the strings were entered.

NUMBER OF STRINGS IS: 4
0.) I love assembly language
1.) even more than I love Java or C.
2.) I am so glad that I am taking COMP 273
3.) because I am learning so much.

Part 3: move-to-front (40 points)

After the list of strings has been entered and stored, the program should enter a second loop. In each pass through this second loop, the user will be asked to select a string by entering a number, namely the index of a string in the list. The program must then modify the list by moving the indexed string to the front of the list. *However, the program must not move the string in Memory (which would be very messy, since the strings in general have different sizes).* Rather, the program must modify only the array of string references. See lecture 10 slides.

Move the selected string to the front as follows. First, save the address of the indexed string j in a temporary variable. Then shift the addresses of strings 0 to j-1 from array positions 0 to j-1 to array positions 1 to j. Then copy the saved address of string j into array position 0. This algorithm should be familiar to you e.g. it is used in insertion sort which many of you have seen in COMP 250.

After the selected element has been moved to the front of the list, the program should print the list of strings again, showing the new order. If the user entered index 0, then the order should not have changed since the selected string was already at the front of the list. The move-to-front loop can be repeated arbitrarily many times.

The user indicates she/he is done by entering an index greater than or equal to the number of strings. The program should then exit and print the message, "Bye for now!"

Here is an example of the output when the user enters "2", moving string at index 2 to the front of the list:

```
MOVE TO FRONT STRING (ENTER NUMBER): 2

 0.) I am so glad that I am taking COMP 273
 1.) I love assembly language
 2.) even more than I love Java or C.
 3.) because I am learning so much.

MOVE TO FRONT STRING (ENTER NUMBER): 5
BYE FOR NOW !
```

Part 4: correct handling of string terminators (10 points)

This part deals a curious technical issue, concerning the maximum length of a string that the user can enter. The issue only comes up if you enter strings that have the maximum length.

The system call for entering a string requires that you specify the **size** of a "buffer" which the OS/kernel will need for your string. This buffer must be large enough to hold the string you will enter, including the null character (ASCII value 0) which terminates every string. The kernel needs to know the maximum **size** because the kernel does not know in advance how long a string the user will enter, and it needs to make room.

The MIPS "system call" for reading a string has the following property. If the number of characters in the string is less than **size** - 1, then when the kernel reads the characters of the string into the buffer and copies that string from the buffer into Memory, it terminates the string with a linefeed, i.e. the <return> character (ASCII value 10), followed by a null character (terminator) that indicates the end of the string. If, however, the user enters a string with **size** -1 or more characters, the system call takes the first **size** – 1 characters that were entered, terminates the string only with a null character, and copies the string into Memory. Note: a string that can be entered by the user therefore has at most **size** – 1, not including the null terminator. This may be slightly confusing because, when you refer to the length of a string, you typically are not including the null terminator or line feed *e.g.* If you enter "hi", then you think of the length of this string as 2.

Ok, enough preamble: what do you need to do ? For every string that the user enters, your program should ensure that the string that is stored in Memory is indeed terminated by a linefeed, followed by a null terminator. For strings that are of maximum length, you will need to manipulate the string in Memory to add the line feed.

Here is what should happen if you attempt to enter strings that exceed the maximum length. Note the missing linefeeds at the string entry stage, namely the echoed string is not on a new line. However when the strings are printed out, the line feed has been correctly inserted.

```
ENTER MAXIMUM LENGTH OF A STRING:  8
ENTER A STRING :  I love aI love a
ENTER A STRING :  ssembly ssembly
ENTER A STRING :

NUMBER OF STRINGS IS:  2
0.)  I love a
1.)  ssembly
```

Get started early! Have fun! Good luck !