First:_____ Last:_____

Scoring Your grade will be based both on the numerical results returned by your program and on your programming style. In particular, write code that is easy to understand, easy to debug, easy to change. Please employ good labels, pretty structure, and good comments.

Performance Score= Run by TA at the checkout		TA:

I promise to follow these rules

This is a closed book exam. You must develop the software solution using the **Keil uVision** simulator. You have 55 minutes, so allocate your time accordingly. You must bring a laptop and are allowed to bring only some pens and pencils (no books, cell phones, hats, disks, CDs, or notes). You will have to leave other materials up front. Each person works alone (no groups). You have full access to **Keil uVision**, with the **Keil uVision** help. You may use the Window's calculator. You sit in front of a computer and edit/assemble/run/debug the programming assignment. You do NOT have access the book, internet or manuals. You may not take this paper, scratch paper, or rough drafts out of the room. You may not access your network drive or the internet. You are not allowed to discuss this exam with other EE319K students until Thursday.

The following activities occurring during the exam will be considered scholastic dishonesty:

1) running any program from the PC other than **Keil uVision**, or a calculator,
2) communicating with other students by any means about this exam until Friday,
3) using material/equipment other than a pen/pencil.

Students caught cheating will be turned to the Dean of Students.

Signed: _____ date:_____

Procedure

First, you will log onto the computer and download files from the web as instructed by the TAs.

Web site xxxx
user: xxxx
password: xxxx

Unzip the folder placing it in a temporary folder. You are not allowed to archive this exam. Within **Keil uVision** open the project, put your name on the first comment line of the file **Exam2.c**. Before writing any code, please build and run the system. You should get output like the figure above (but a much lower score). You may wish create backup versions of your program. If you wish to roll back to a previous version, simply open one of the backup versions.

My main program will call your subroutines multiple times, and will give your solution a performance score of 0 to 100. *You should not modify my main program or my example data.* Each time you add a block of code, you should run my main program, which will output the results to the **`UART#1`** window. After you are finished, raise your hand and wait for a TA. The TA will direct you on how to complete the submission formalities. The TA will run your program in front of you and record your performance score on your exam cover sheet. The scoring page will not be returned to you.

Part a) The first subroutine, called **`Size`**, determines the number of **8-bit ASCII** characters in a null-terminated string. The same string format will also be used for the third subroutine. The null character, used to signify the end of the string, *is NOT one of the characters.* Look at the source code to see all five sets of test data. One data set is the following:

```
String1 DCB "cat",0      ;size=3
```

Input parameter: A pointer to the string is passed into your program in RegR0.

Output parameter: The number of data characters returned in RegR0.

Error conditions: If the string is empty, return RegR0=0.

A typical calling sequence (from assembly) is

```
LDR R0,=String1 ; pointer to the null-terminated string
BL  Size        ; should return RegR0=3
```

Part b) Write the second assembly language subroutine, called **`Compare`**, which compares two ASCII characters. You do not convert upper case to lower case or convert lower case to upper case. Rather, you simply compare 8-bit unsigned values.

Input parameters: *first* and *second* are two 8-bit ASCII character.

Output parameter: Return -1 if the *first* is greater than the *second*, Return 0 if the *first* equals the *second*, and Return +1 if the *first* is less than the *second*

Error conditions: none.

A typical calling sequence (from assembly) is

```
MOV R0,#0x79   ; first is letter 'y'
MOV R1,#0x5A   ; second is letter 'Z'
BL  Compare    ; should return R0=-1, because 0x79 > 0x5A
```

Part c) Write the third assembly language subroutine, called **`StringCompare`**, which compares two strings. If the two strings are equal, return 0. To be equal, all letters must match and the strings must be the same length. If the first string is alphabetically before the second string return +1. If the first string is alphabetically after the second string return -1. You do not convert upper case to lower case or convert lower case to upper case. Rather, you simply compare the 8-bit unsigned values in each string from beginning to end, stopping at the end of either string or if a letter is not equal. Empty strings are alphabetically before all strings except another empty string. Consider these test cases

First string	Second string	Result
Cat	Dog	+1 because first letter 'c'< 'd'
Cattle	cobra	+1 because second letter 'a'< 'o' (length doesn't matter)
Horse	Hose	+1 because third letter 'r'< 's' (length doesn't matter)
Cat	Cattle	+1 because all letters of the first string match, but the first string is shorter (length does matter)
horse	horse	0 because the strings are equal

Input parameter: A pointer to the first string is passed into your program in buffer1
A pointer to the second string is passed into your program in buffer2

Output parameter: The result is returned in the R0

Error conditions: none

```
First2  DCB "cattle",0
Second2 DCB "cobra",0
```

A typical calling sequence (from assembly) is

```
  LDR  R0,=First2    ; pointer to the null-terminated string
  LDR  R1,=Second2   ; pointer to the null-terminated string
  BL   StringCompare ; should return R0 = +1
```

Your subroutines should work for all cases shown in the starter file. The input strings, however, may be empty. *Handle the simple cases first and the special cases last.*