

## **CMPE-250 Assembly and Embedded Programming**

### **Laboratory Exercise Six**

#### **Secure String I/O and Number Output**

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students. Other than code provided by the instructor for this exercise, all code was developed by me.

---

Shubhang Mehrotra  
Performed 4<sup>th</sup> March 2021.  
Submitted 11<sup>th</sup> March 2021.

Lab Section 1  
Instructor: Muhammad Shaaban  
TA: Anthony Bacchetta  
Aidan Trabuco  
Sam Myers  
Payton Burak

Lecture Section 1  
Lecture Instructor: Dr. Roy Melton

## Procedure Screen Capture

The activity explored the KL05Z Freedom Board further by writing a program to handle secure String input and output. Building upon the GetChar and PutChar subroutines from Lab 5, the performer now created the GetCharSB and PutCharSB subroutines to handle a buffered string input-output. The Strings used in the activity were limited to being 78 character long. For this implementation, register R0 was used to store and output the String, and register R1 specified the buffer capacity. To ensure security, all strings were null terminated upon receiving a Carriage return as input.

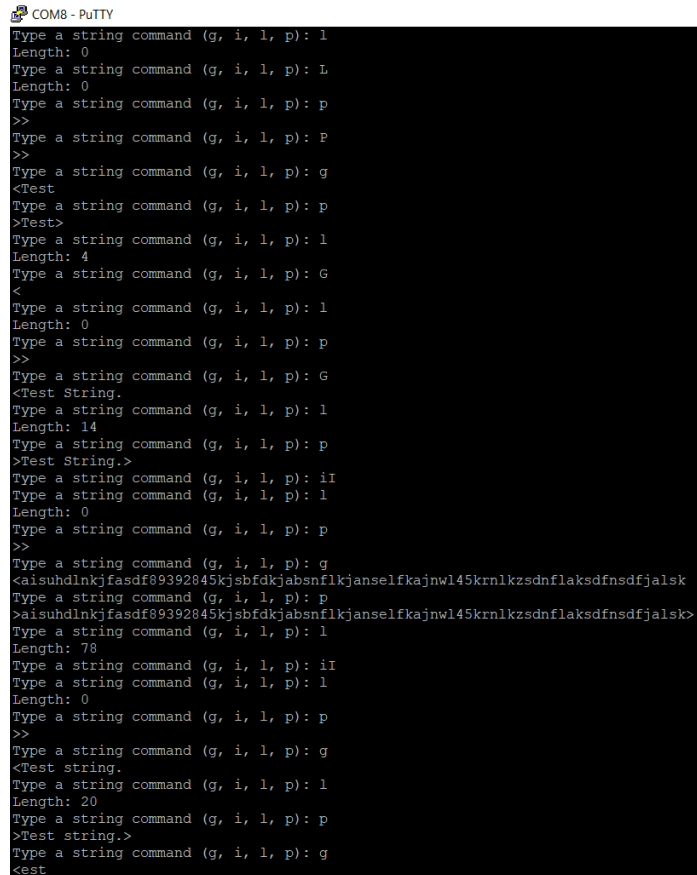
An additional PutNumU subroutine was also defined in the code to display decimal representation of an unsigned word value using the PutChar subroutine.

This connection to the board was again established via UART Polling at 9600 Baud rate using PuTTY and its “Serial” connectivity functionality.

The code for secure string I/O and number output was written and downloaded to the KL05Z board using the Keil IDE. It was then tested against the following cases:

- Uninitialized string: l and p.
- User string between 1 and 77 characters: g, l, and p.
- User empty string, (i.e., only enter key pressed): g, l, and p.
- User string of 78 characters: g, l, and p.
- Initializing string to empty string: i, l, and p.
- User string attempting more than 78 characters: g, l, p.

The result of the test run is shown in Figure 1.



```

COM8 - PuTTY
Type a string command (g, i, l, p): l
Length: 0
Type a string command (g, i, l, p): L
Length: 0
Type a string command (g, i, l, p): p
>>
Type a string command (g, i, l, p): P
>>
Type a string command (g, i, l, p): g
<Test
Type a string command (g, i, l, p): p
>Test>
Type a string command (g, i, l, p): l
Length: 4
Type a string command (g, i, l, p): G
<
Type a string command (g, i, l, p): l
Length: 0
Type a string command (g, i, l, p): p
>>
Type a string command (g, i, l, p): G
<Test String.
Type a string command (g, i, l, p): l
Length: 14
Type a string command (g, i, l, p): p
>Test String.>
Type a string command (g, i, l, p): iI
Type a string command (g, i, l, p): l
Length: 0
Type a string command (g, i, l, p): p
>>
Type a string command (g, i, l, p): g
<aisuhdlnkjfasdf89392845kjsbdfkjabsnflkjanselfkajnw145krnlkzsdnflaksdfnsdfjalsk
Type a string command (g, i, l, p): p
>aisuhdlnkjfasdf89392845kjsbdfkjabsnflkjanselfkajnw145krnlkzsdnflaksdfnsdfjalsk>
Type a string command (g, i, l, p): l
Length: 78
Type a string command (g, i, l, p): iI
Type a string command (g, i, l, p): l
Length: 0
Type a string command (g, i, l, p): p
>>
Type a string command (g, i, l, p): g
<Test string.
Type a string command (g, i, l, p): l
Length: 20
Type a string command (g, i, l, p): p
>Test string.>
Type a string command (g, i, l, p): g
<est

```

Figure 1. Result

Figure 1 shows the PuTTY Serial window with the results of the various inputs. The code written performed the required function only when one of the keywords– “G”, “I”, “L” or “P”, was input. It ignored all other inputs, except the “Enter,” upon which it output a carriage return accompanied with a Line feed character, to move the cursor to the next line. Furthermore, if the character input was in lowercase, the program converted it to uppercase to handle the functionality. It is evident from Figure 1 that the code was successful in its implementation, as it ignores all other inputs and outputs the desired strings when it receives one of the keywords.

The “G” keyword allowed the user to input a string consisting of letters and numbers. The string had a maximum allowed length of 78, as depicted in one of the test cases. It primarily utilized the GetCharSB subroutine.

The “P” keyword printed the saved string to the console, using the PutStringSB subroutine.

The “L” keyword gave the user length of the saved string, utilizing the LengthStringSB subroutine provided in the library.

The “I” keyword initialized the operational string to an empty string, by storing Null in the first byte.

## Memory Ranges

For Lab Activity 6, the Listing file and the Map file were generated and analyzed for the memory addresses of the items listed in Table 1, which lists the start addresses, end addresses and the size occupied by the item in the memory.

Table 1. Memory Ranges

Object	Memory Address Start	Memory Address End (Start + Size - 1)	Size (bytes)
Executable code in MyCode AREA	0x0000011C	0x0000009C	0X9D
MyCode AREA	0x0000011C	0x0000036F	0X370
Constants	0x00000370	0x000003A3	0x3A4
Variables	0x1FFFFD00	0x1FFFFD94	0xDA0
GetStringSB Subroutine	0x0000024A	0x00000279	0x30
PutNumU Subroutine	0x00000296	0x000002D0	0x3A
PutStringSB Subroutine	0x0000027A	0x00000295	0x1C

The Start Address for “MyCode AREA” was obtained from the Map file. Under the section marked “Memory Map of the Image,” is a table which lists the execution address of each entity in the source file. Upon inspection of the table, the start addresses, and the size can be obtained. The End address can be obtained by adding the start address and the size and subtracting 1.

The Listing file contains the offset with respect to the execution addresses and can be analyzed to obtain the addresses for the subroutines and the executable area of the code. For instance, the offset listed for the GetCharSB subroutine is “0x0000012E.” Adding that offset to the execution address of “0x0000011C,” we obtain the start address of GetCharSB subroutine as “0x0000024A.” Similarly, the address for PutCharSB Subroutine is calculated from its listed offset of “0x0000015E.” The ending addresses can be obtained as mentioned above using the size and the starting address and subtracting 1. All other addresses and size were similarly obtained.

## Questions

1. Why does the number of user-typed characters stored in the string have to be one fewer than the number of bytes allocated for the string?  
All strings are required to be null terminated for the secure operations. The last byte allocated is used by the null operator to signify end of string.
2. Why does MAX\_STRING need to be defined as an EQUate rather than as a DCD in the MyConstAREA, (i.e., a constant)?  
MAX\_STRING is only a text substitution for the number '79', which provides additional accessibility to the programmer and as a good coding practice. Moreover, DCD constants are compiled at the end of the program and are only available then, in such a case the program cannot use them before they are compiled and hence stop the functions from performing as required. It adds unnecessary complexity.