**CMPE-250 Laboratory Exercise Eight**

**String Operations**

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; however, other than code provided by the instructor for this exercise, all code was developed by me.

John Judge

Performed 3/31/16

Submitted 4/7/16

Lab Section 02

Instructor: Dr.Shaaban

TAs: Peter Muller

Stephen Moore

Connor Goss

Lecture Section 01

Professor: Allesandro Sarra

**Abstract**

In this exercise, basic string operations were explored on the Freescale Freedom KL46Z microcontroller using a peripheral machine for user input. The objective of the exercise was familiarization of string operations using subroutines to control operations and program to test them. During the exercise the program was created and tested. The results were then analyzed to determine that the exercise had been successful.

**Procedure**

Just as in previous exercises, this exercise required the use of the Freescale Freedom KL46Z microcontroller and a peripheral machine for user input. To begin the exercise the project had to be correctly configured to work with in conjunction with the peripheral device. Using the same program templates and starting files used in the previous exercises the project contained all of the necessary files and once the settings with the Kiel IDE had been configured the project could run as intended.

From here the program was constructed. In order to achieve the string operations required for the exercise three subroutines needed to be written. The first of these, *CopyString,* was needed to initialize the initial string. The subroutine would copy the contents of one string into another string. This was achieved by passing the subroutines the respective address of the initial string and the operational string in registers. The subroutine then loaded every element in the initial string and stored the element in the same position in the operational string. This was achieved by implementing a for-loop and index mode addressing. This subroutine would later be used to store the initial string, “Initial String,” in the location of the operational string.

The next operation required was modifying a string so that all space characters were replaced with underscores, all number replaced by pound signs, and all uppercase characters replaced with lowercase characters. This was achieved by passing the subroutine, *ModifyString,* with the starting address of the operational string. From there the subroutine loaded each element of the string and compared it to the Ascii value for a space, all uppercase numbers, and all numbers. If a space was detected then a underscore character was stored at the address the space, in memory, that the space was. If a number was detected then the Ascii value for a pound sign was stored at the address, in memory, of the number. Any uppercase character was loaded and then 32 was added to it in order to turn the character into its lowercase equivalent and then the new character was stored at the original character’s position.

The final string operation required was achieved by the subroutine *ReverseString*. When passed the starting address of the operational string *ReverseString* would reverse the position of each element in the string in regards to their position from the starting address. In order to achieve this the subroutine would first find the length of the string by iterating through each element from the starting address to the first null character while incrementing a counter for each non-null element. Within a for-loop the subroutine would then load the characters at the starting address of the string and, using the length of the string as an offset, the last element in the string. So long as the loop had not closed in on the middle of the string then the elements were stored at the other’s address. This resulted in a reversed string.

From here the subroutines *GetStringSB, PutStringSB, GetChar,* and *PutChar* from previous exercises were added to the project. Then the main program was created. First the initial string’s address was loaded into a register and using *CopyString* it was copied into the operational string. Then the program would print the possible commands to the console using *PutString*. Then by using a loop the program would poll for input using *GetChar*. If one of the possible commands (G, g, H, h, M, m, P, p, R, or r) was detected then the character was echoed to the console using *PutChar*, if not it would echo the character and print the string, “:Invalid command” and then return to asking for a command. Had a possible command been echoed to the console then the program would branch to a helper function, one for each possible command, which in would call the subroutine necessary to preform the operation. G or g would call *GetStringSB* to poll the user for input and then store each character in the operational string sequentially. H or h would print the help menu that defined each of the possible commands. This was achieved by using *PutStringSB* and the help string, which was located in the constants. P or p would print the operational string to the console using *PutStringSB*. When R or r was entered the program would use *ReverseString* to reverse the operational string. After any of these operations took place the program branched back to print the possible commands and wait for input.

**Results**

After the program was developed a series of test cases were conducted to test the program. The tests were created to test cases that could be considered common errors when operating on strings in this nature. The result of the terminal after these test cases can be seen in Figure (1.0).

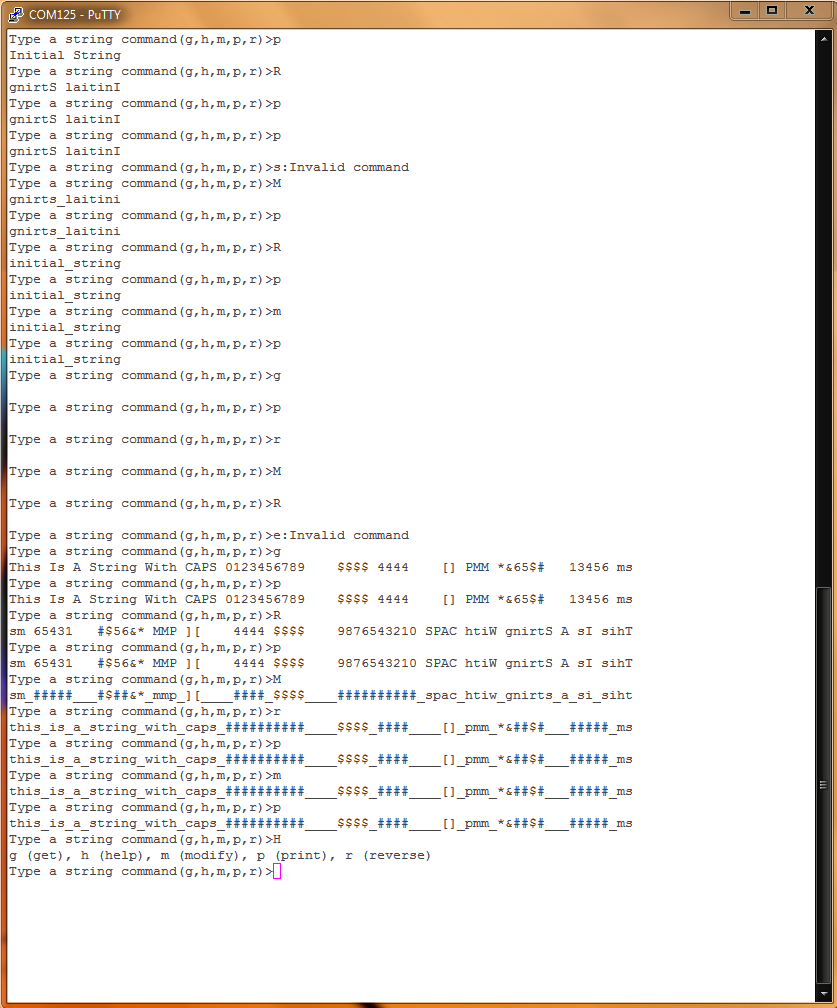


Figure (1.0): Terminal Results

To begin the tests the print command was tested in order to conclude *CopyString* had been correctly used to initialize the initial string. The program successfully achieved this functionality and printed the initial string. To continue, the program was tested with the reverse command using an uppercase “R” for input. This checked that the program could carry out commands with valid uppercase inputs. The program successfully handled the case and *ReverseString* successfully reversed the initial string and printed the resulting string to the terminal. The print command was used to print the string from memory. The result of this operation was identical to the previous printed string ensuring the program was successfully storing, retrieving, and printing the string at the correct address. An invalid command was then entered and the program successfully handled the unknown command by printing “:Invalid command” followed by the menu on the next line of the terminal. The last operation was tested when an uppercase “M” was input. The program successfully printed the operational string with the space replaced with an underscore. The previous commands were used again to manipulate the same string even more and the program successfully handled each command and responded with the correct results.

To follow this the g command was used and a null string was input. Then all of the commands were once again used to test the functionality of the program when operating on a null string. The program responded correctly and continued to display a null string as the result for each of the operations.

After another invalid command was input, another string was stored but this time the string contained number, space, and punctuation characters. The string was printed correctly and then a reverse operation took place. The result of the reverse matched the expected output. When the string was then modified all the numbers, spaces, and uppercase characters were replaced with the their modified counterparts. These tests ensured the program could correctly respond to all characters that needed to be replaced.

Lastly the help command was used and the program successfully printed the help displayed, returned to the next line, and printed the prompt. After all of the correct results for the test cases it was concluded that the program could successfully handle any case and therefore the program correctly developed.

The memory map file created by the program was then examined. Using the Local Symbol Table, which can be seen in Figure (1.1), the memory ranges of the program were examined.

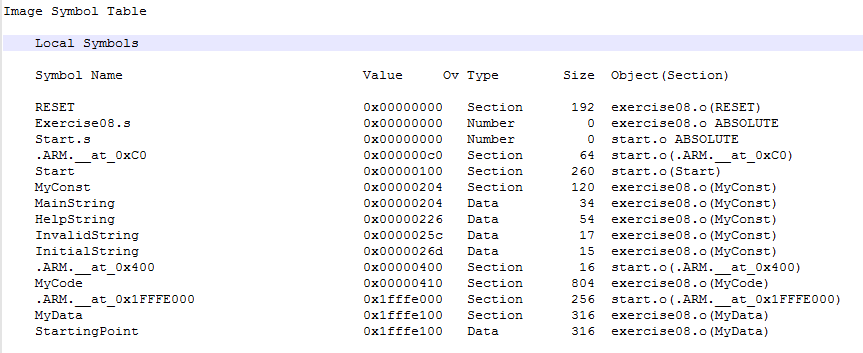


Figure (1.1): Local Symbol Table

From the Local Symbol Table, it was concluded that the range of the executable code was between 0x00000410 and 0x00000733 with a total size of 80410. The constants for the program, which included the equate statements necessary for UART polling as well as the maximum length of the string (*Max\_String*) and the constant string the program would print took up a total space of 12010 and could be found between 0x00000204 and 0x0000027B. The only variable used in the program, the space allocated for the operational string, *StartingPoint* could be found between the range 0x1FFFE100 and 0x1FFFE23C with a total size of 31610.

**Conclusion**

In this exercise, basic string operations were explored on the Freescale Freedom KL46Z microcontroller and tested with a series of test cases. The program correctly handled all string operations and as a result the theory behind string operations, taught in lecture, were verified. The objective of the exercise, familiarization of string operations, was achieved and the exercise was considered to be successful as a result. The exercise will prove to be a valuable asset in future exercises or endeavors that incorporate operational strings or string theory.