COL216 Lab Assignment-1 (S2)

- Mayank Mangla (2020CS50430)

1 Introduction

This project's aim is to develop a program in ARM Assembly Language for arranging a sequence of character strings in dictionary order (lexicographic order).

Stage 2: This program takes length of two lists, duplicate mode, comp mode and two input strings lists and outputs the length of merged list and the merged list's elements.

The report for the stage 2 program is given here:

2 Program Information

The program starts from input.s

1. **Assumptions:** We assume that the input strings are not more than 99 for each list and the user enters only integers for the lengths and mode inputs (the four integers). Also as mentioned in problem statement, the two lists are in a sorted order.
2. **Software:** This program has been run on **ARMSim (v2.0.1)** in **Windows OS (v windows 10) machine** and the test cases were chosen at random. Also the **Angel Instructions** were used.

2.1 Input and Output

The program prompts for the input by stdin wherever required
User is expected to input four integers, length of first list, length of the second list, comparison mode and 0 or 1 for duplicates. Then the user is asked to enter the strings for list 1 till the user has entered as many strings as mentioned by the length of first list by the user. Similarly, the user is asked to enter the strings for list two.
The output is in the form of a message followed by the length of the merged list The a message prompts user for a new line character for each printing string. When all the strings in merged list are printed then a message is printed stating the end of program.

2.2 Algorithm

- Input is read character by character. First we read the characters for the length of the lists. A function converts this string into the corresponding integer. Similarly the other integer inputs are taken. These 4 integers are stored in memory.
- If the user enters the length as 0 for both the lists then program shows an error.
- Then we read the input strings character by character till we reach the next line character. Next line character is the indication that the user has completed typing that string. This character by character reading requires each character to be stored after each character reading. When we get a next line character, to get a null terminated string, we store a null character after the next line character. This signifies the end of a string.
- During this reading, if the user enters a wrong input and wants to use backspace then this character reading is modified as to get back to overwrite the character that is being deleted.
- Now, once we have stored null terminated strings in memory, we can generate the list of pointers for which we have already allocated the space in data part of the file. Each list has been initiated with a length of 99 and each 4 bytes of the list stores the address of the first character of the corresponding string.

- To create this list, we start from the first character read and update the value in list at first position. Now the list pointer is pointing to the next 4 bytes. We iterate through the memory till we get a null character. As we get a null character, we increase the value of iterator so that the iterator now points to the first character of the next string.
- This address can be stored in the list and list pointer to be updated. This loop goes on till we have not stored n number of strings in first list, where n is the input length of first list. Similarly, we create the list of addresses for list 2.
- In this way we have got 4 integers and 2 lists of addresses from input file and then ***we move to merge.s*** for further computation.
- Now we start to merge the two sorted lists. Here we load each 4 bytes address and access the string starting at that address. We compare the two strings and store the address of lesser one to the address of merged list.
- If the strings are equal then we check if the duplicates are allowed or not. If allowed then we store the address of both the strings and if not then we store only one address of the both.
- During this we appropriately increase the address value of pointers. We do this till we have any one of the counters equal to length of the corresponding list
- After we get any one length equal, we add all the elements remaining in the other list in the merged list. Like this we get our merged list.
- Now we move to ***output.s***.
- Starting of the file prints the message and the length of the merged list stored in a register.
- This file has a function that converts the integer to a string that could be printed ion the console.
- The output file prompts user for a new line character with a message that would tell program that the user wants to see the next string in the merged list.
- Then by each new line character the program outputs the next string in the merged list.
- A message stating the end of program is printed.

2.3 Test Cases and Results

Some random test cases were tested on the program. The results of which are shown below:

- Case in-sensitive comparison and duplicate inclusive

```
Console  stdin/stdout/stderr
Please enter the length of list 1 (< 99) : 4
Please enter the length of list 2 (< 99) : 4
Please enter 0 for case in-sensitive and 1 for case sensitive merge: 0
Please enter 0 for duplicate inclusive and 1 for duplicate exclusive merge: 0
Enter the strings for first list
AFSDGHFGJ
SFGGFHKGKJ
ZFGDHNG
ZZZZGHFCHGJ
Enter the strings for second list
asffjgk
eyfjhjkhd
zghfhjmk
zzzzfgdxfgchm
Size of the merged list is: 8
The strings in merged list are: (Press Enter to get next string)
AFSDGHFGJ
asffjgk
eyfjhjkhd
SFGGFHKGKJ
ZFGDHNG
zghfhjmk
zzzzfgdxfgchm
ZZZZGHFCHGJ
 -------End of Program--------
```

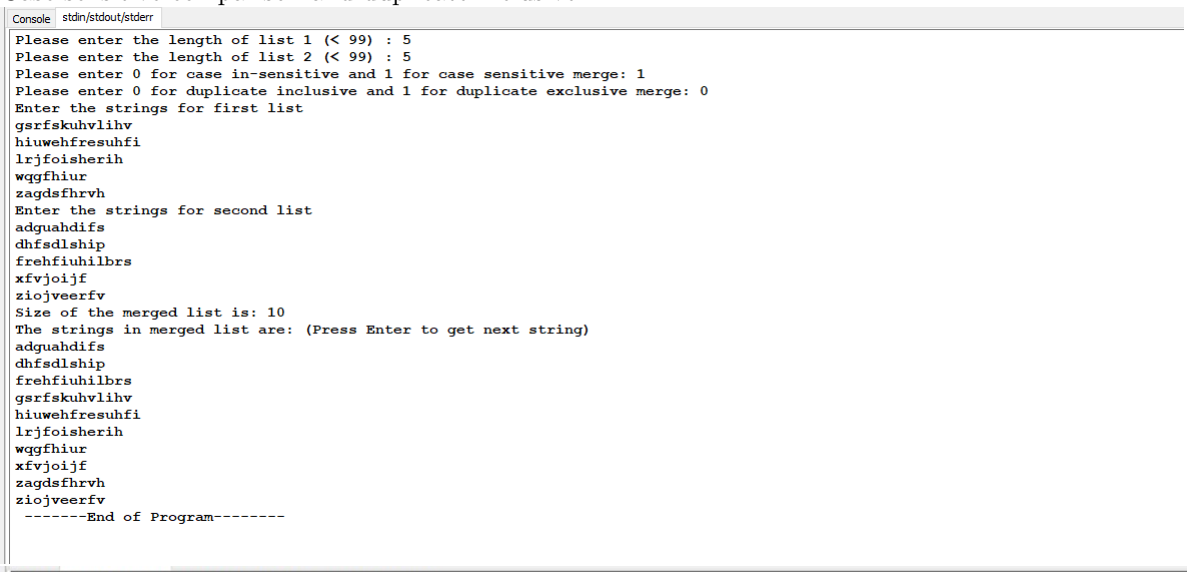
```
Please enter the length of list 1 (< 99) : 6
Please enter the length of list 2 (< 99) : 7
Please enter 0 for case in-sensitive and 1 for case sensitive merge: 0
Please enter 0 for duplicate inclusive and 1 for duplicate exclusive merge: 0
Enter the strings for first list
GJGjkgGJHJKughkjhJKH
HHuhuhuhKUHHUHuhih
ahjgJHGJHGjHGHghJKHGjkGHjk
dhfgkjhhfjkhhkfkjl
xwhkefckjwlerhf
zjkhrfjkeslkggt
Enter the strings for second list
gjgjKGjHjKUGhkjhjkh
hhuhuhuhkuhhuhuhih
zjkhrfjkshshjkrhfv
zzzefkjwkjefrjkgklrtkjkj
zzzz
zzzzz
zzzzzz
Size of the merged list is: 12
The strings in merged list are: (Press Enter to get next string)
GJGjkgGJHJKughkjhJKH
gjgjKGjHjKUGhkjhjkh
HHuhuhuhKUHHUHuhih
hhuhuhuhkuhhuhuhih
ahjgJHGJHGjHGHghJKHGjkGHjk
dhfgkjhhfjkhhkfkjl
xwhkefckjwlerhf
zjkhrfjkeslkggt
zjkhrfjkshshjkrhfv
zzzefkjwkjefrjkgklrtkjkj
zzzz
zzzzz
-------End of Program--------
```

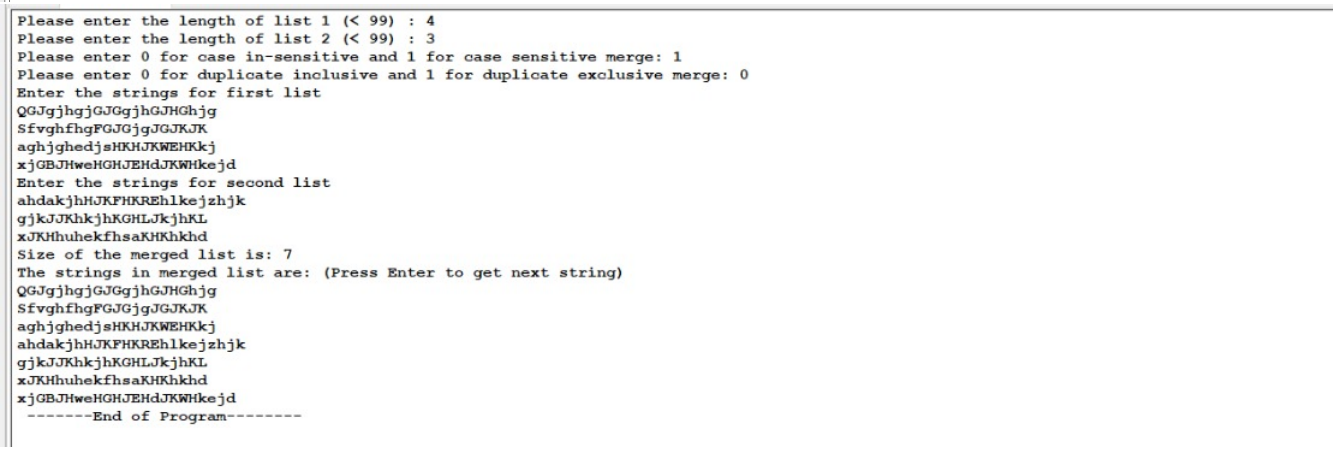
- Case in-sensitive comparison and duplicate exclusive

```
Please enter the length of list 1 (< 99) : 8
Please enter the length of list 2 (< 99) : 3
Please enter 0 for case in-sensitive and 1 for case sensitive merge: 0
Please enter 0 for duplicate inclusive and 1 for duplicate exclusive merge: 1
Enter the strings for first list
AGDWhgfHFFfjyFGYUFFFYjkg
GHFJHghgjhgJHGjhg
HGGKgjgj
LKJHjhJK
abbjkJHjhjkhl
nJGHjhjkhjkhHjkJH
wdgGJHGJHgjhgJHGjh
zcghFJHGjhgJHkj
Enter the strings for second list
agdwhgfhfffjyfgyufffyjkg
lKJHJHJK
zjhuiuighdhweuhf
Size of the merged list is: 9
The strings in merged list are: (Press Enter to get next string)
agdwhgfhfffjyfgyufffyjkg
GHFJHghgjhgJHGjhg
HGGKgjgj
lKJHJHJK
abbjkJHjhj
khlnJGHjhjkhjkhHjkJH
wdgGJHGJHgjhg
JHGjhzcghFJHGjhgJHkj
zjhuiuighdhweuhf
-------End of Program--------
```

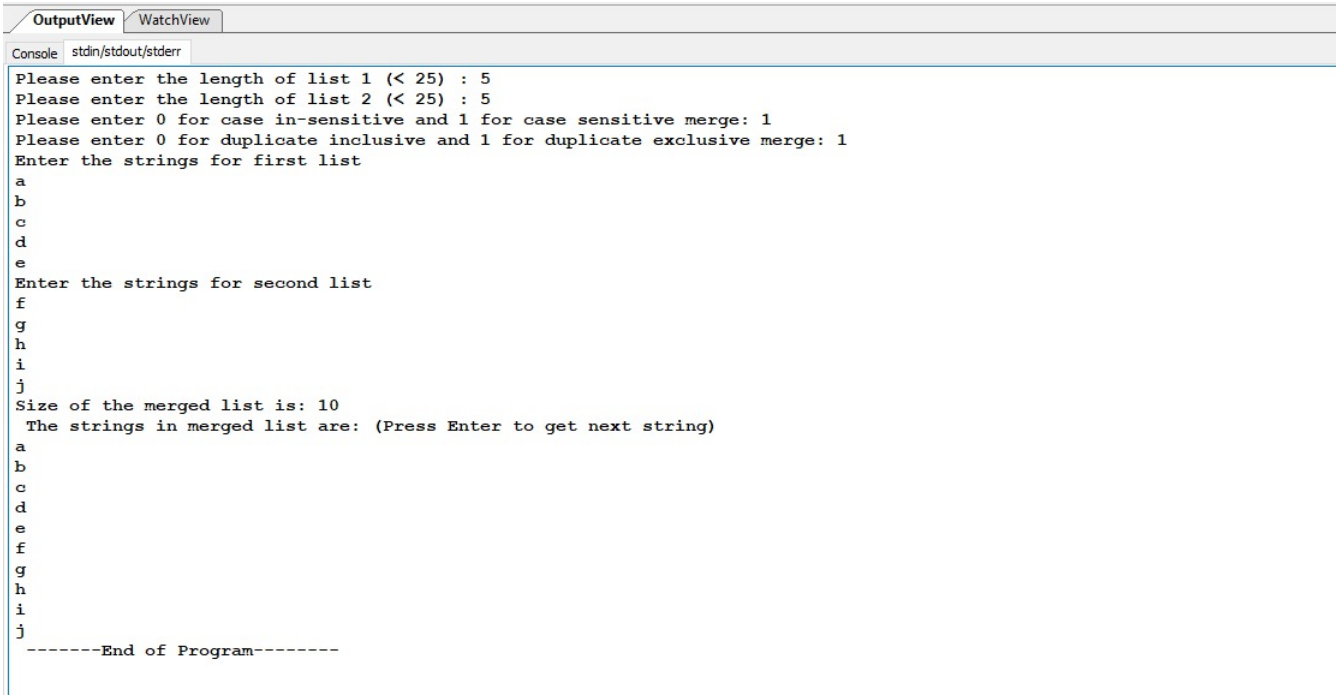
```
Please enter the length of list 1 (< 99) : 13
Please enter the length of list 2 (< 99) : 7
Please enter 0 for case in-sensitive and 1 for case sensitive merge: 0
Please enter 0 for duplicate inclusive and 1 for duplicate exclusive merge: 1
Enter the strings for first list
aqwert
byuiop
casdfg
dhjkl
ezxcv
fvbnm
gqhbwefd
hwhefluih
ivxghd
jwhf
kedd
lwhfd
lzbhewfd
Enter the strings for second list
dghwf
kwhefdiw
miwhfiuh
nhwfd
swhrefd
tjkewf
tjkez
Size of the merged list is: 20
The strings in merged list are: (Press Enter to get next string)
aqwert
byuiop
casdfg
dghwf
dhjkl
ezxcv
fvbnm
gqhbwefd
hwhefluih
ivxghd
jwhf
kedd
kwhefd
iwlwhfd
lzbhewfd
miw
hfiuhnhwf
dswhrefd
tjkewf
tjkez
-------End of Program--------
```

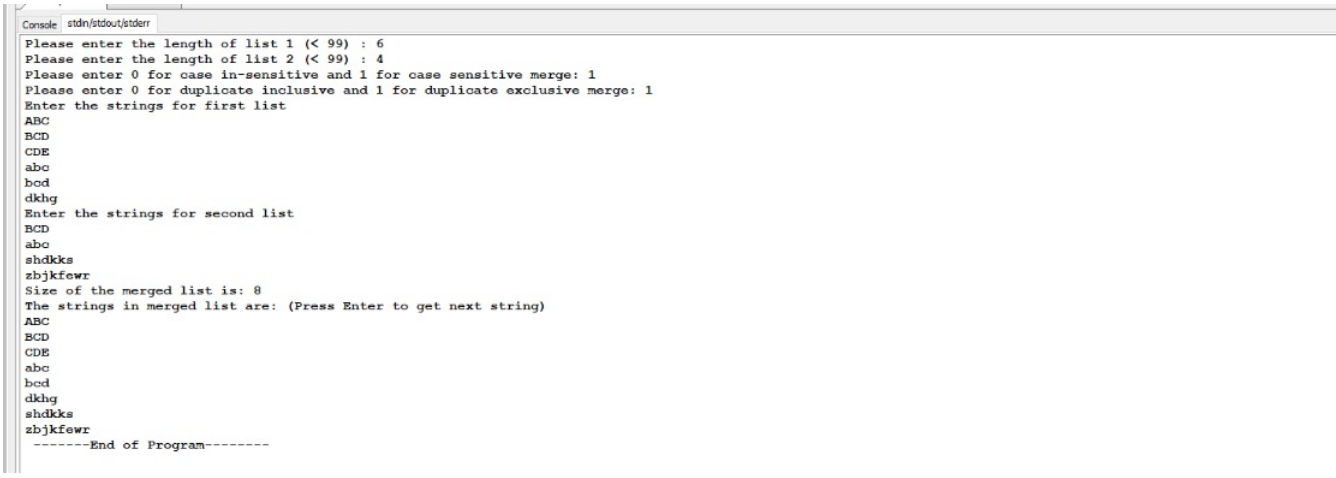
- Case sensitive comparison and duplicate inclusive



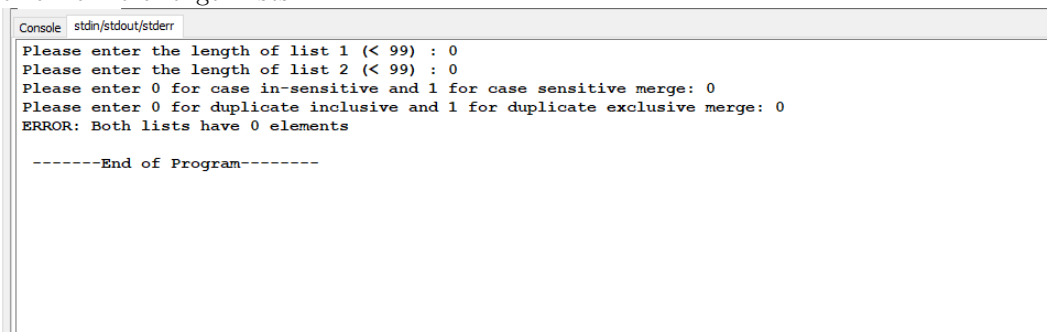


- Case sensitive comparison and duplicate exclusive





- error for zero length lists



2.4 Conclusion

After running these random Test cases and getting the desired output we have verified the program's success.