COL216 Assignment-1 Part 3

by Harshit Mawandia
2020CS10348

Report:

I have used 5 seperate files.

1. *test3.s*
2. *String_comp.s*
3. *UsefulFunctions.s*
4. *merge.s*
5. *mergesort.s*

1. test3.s:

This file contains our *main* function and is mainly to take user inputs and give output.

On running this program, The console asks you to enter the number of strings in the list, Then you enter *strings* of length upto 1024 characters terminating with a $\backslash r$ or $\backslash n$ in ascending order and then an Integer 0 or 1 to check for duplicate removal or not and then an Integer 0 or 1 to check for case insensitive or case sensitive comparison respectively.

This user input is done by using *fgets* function.

Then we call the *mergesort* function which sorts the list according to the preferences

We then print the newly formed list by iterating through its pointer.

2. String_comp.s:

This program compares 2 strings and checks which one is lexicographically smaller. This is done by iterating through the lines of code and checking until one character is different between the two strings. If no such character is found, We say the strings are equal.

We also have a chunk of code which converts lower case characters to upper case in case we need to do a case insensitive comparison.

3. UsefulFunctions.s:

We use the *prints* and the *fgets* functions with slight modification to take input and produce an output from and to the console respectively in our *test2.s* file.

4. merge.s:

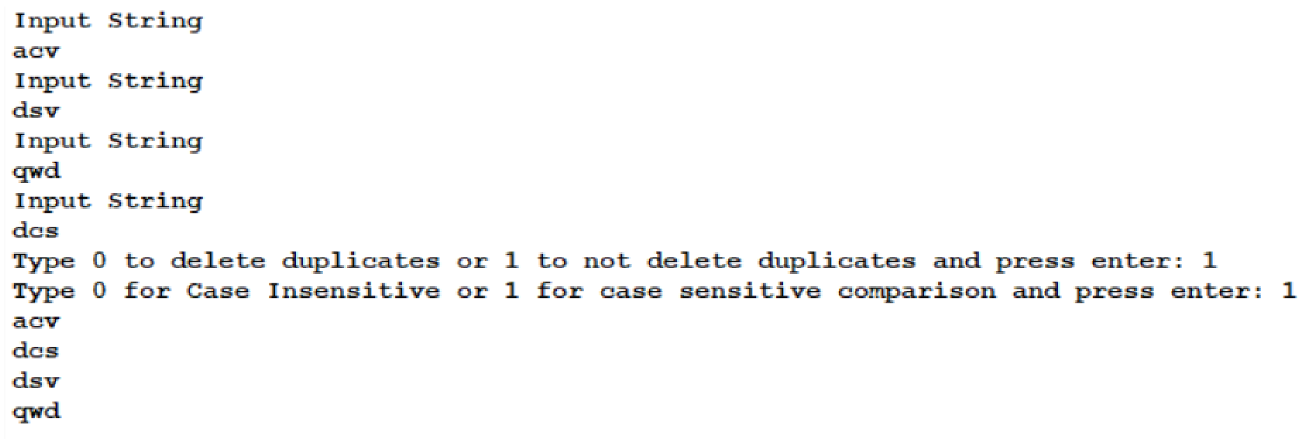
If the input list are of size n and m respectively then we first create a sorted list of size $n + m$. this is done by iterating through the lists and adding the smaller element at any point to our new list and then reiterating. In the end the elements remaining in any list are added without checking as they are definitely larger.

Then we have a function that removes duplicates in the final list. Since the list is sorted we only need to check adjacent elements for duplicates. We go on checking till the end of list and remove the elements which are same. It takes $O(n + m)$ time.

4. mergesort.s:

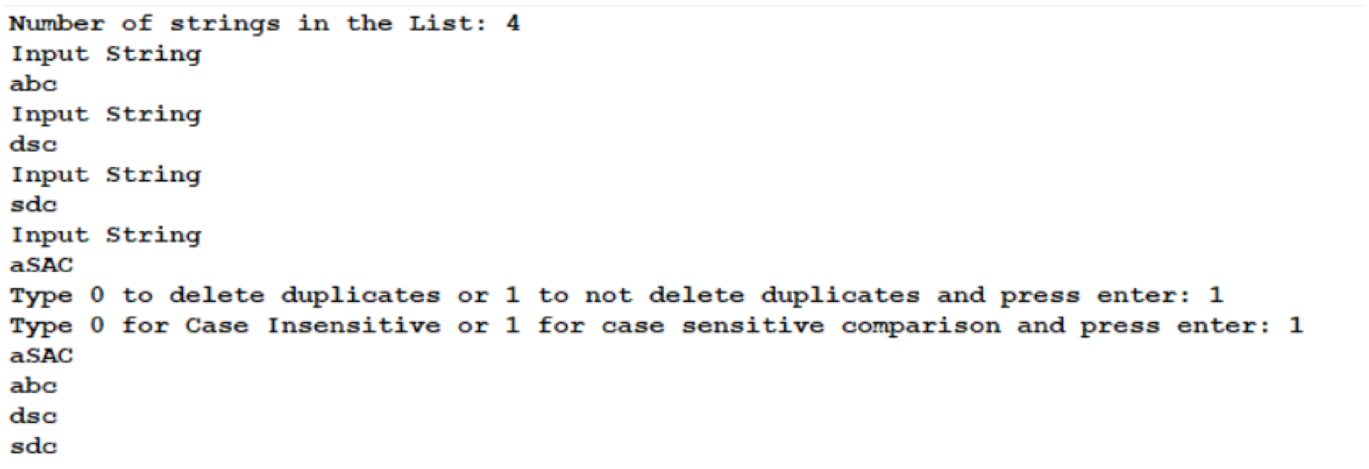
Implements the merge sort algorithm, first divides the list into 2 parts and then recursively applies merge sort to each part until the length of one of them becomes less than 2. Then merges the list back one by one so as to reach the top level again hence forming 1 sorted list. Then we also chexk for duplicate removal.

Results:



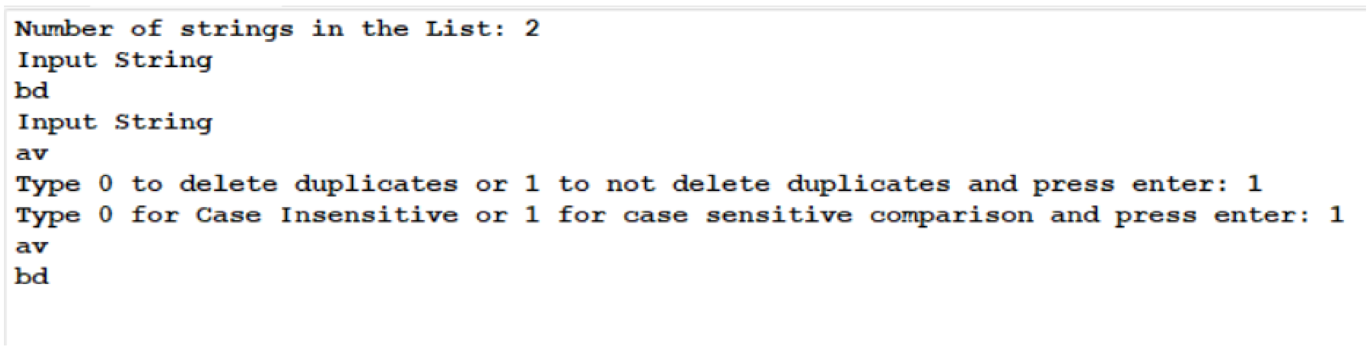
```
Input String
acv
Input String
dsv
Input String
qwd
Input String
dcs
Type 0 to delete duplicates or 1 to not delete duplicates and press enter: 1
Type 0 for Case Insensitive or 1 for case sensitive comparison and press enter: 1
acv
dcs
dsv
qwd
```

Figure 1: Test 1



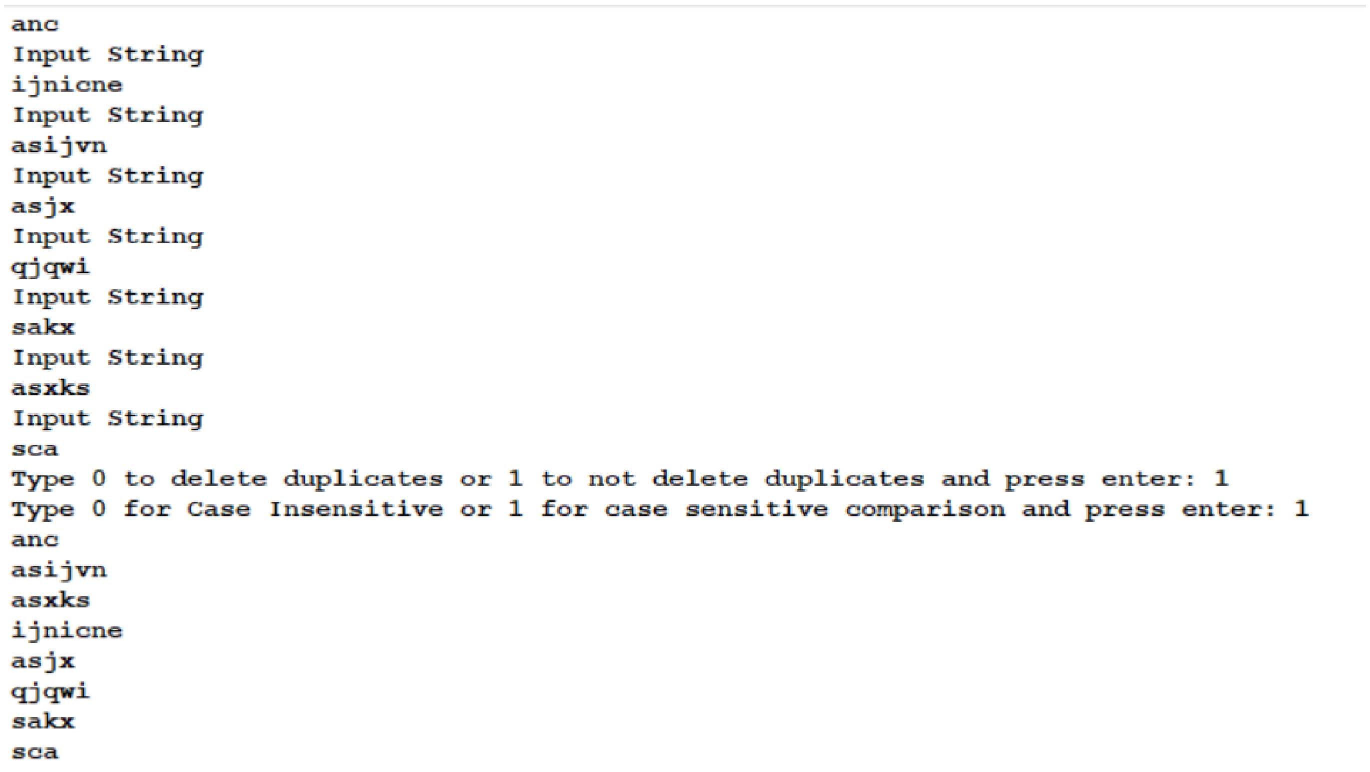
```
Number of strings in the List: 4
Input String
abc
Input String
dsc
Input String
sdc
Input String
aSAC
Type 0 to delete duplicates or 1 to not delete duplicates and press enter: 1
Type 0 for Case Insensitive or 1 for case sensitive comparison and press enter: 1
aSAC
abc
dsc
sdc
```

Figure 2: Test 2



```
Number of strings in the List: 2
Input String
bd
Input String
av
Type 0 to delete duplicates or 1 to not delete duplicates and press enter: 1
Type 0 for Case Insensitive or 1 for case sensitive comparison and press enter: 1
av
bd
```

Figure 3: Test 3



```
anc
Input String
ijnicne
Input String
asijvn
Input String
asjx
Input String
qjqwi
Input String
sakx
Input String
asxks
Input String
sca
Type 0 to delete duplicates or 1 to not delete duplicates and press enter: 1
Type 0 for Case Insensitive or 1 for case sensitive comparison and press enter: 1
anc
asijvn
asxks
ijnicne
asjx
qjqwi
sakx
sca
```

Figure 4: Test 4