HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
OFFICE FOR INTERNATIONAL STUDY PROGRAM
FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING

COMPUTER SYSTEM AND PROGRAMMING
LC3 PROJECT REPORT
STRING SORTING PROGRAM

Lecturer:	Assoc. Prof. Đặng Thành Tín
Class:	TT03
Semester:	212
Members:	Lương Triển Thắng
Student ID:	2051194

Hồ Chí Minh City – 10th May, 2022

Table of Contents

I. Requiremnts

- ✓ Use subroutines as much as possible.
- ✓ Create user interface as clear and beautiful as possible.
- ✓ Check range for every value input and output appropriately.
- ✓ The program should be organized so well for structure programming.
- ✓ The program needs to comment as mamy as possible.
  The detail explanation of the program will be describe more here.
- ✓ Input *n* (input from keyboard) strings of characters with the length unlimited (it is defined by the program, not by the compiler).
- ✓ Sort them in descending or ascending order depending on the request input.
- ✓ These strings are sorted in ascending/descending by the order of dictionary with deleting redundant characters in each string: blank ‘ ’, comma ‘,’ if they exist in string.

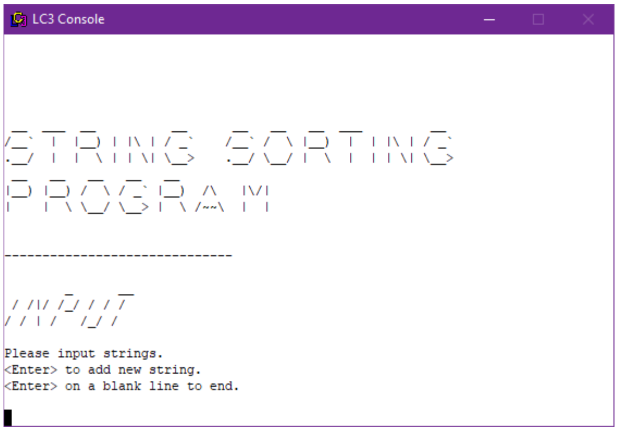
II. Features

- Beautiful and friendly UI.
- User aren’t required to input the number of strings. Input procedure ends when user pressed <Enter> twice (<Enter> on a blank line).
- 36864 characters can be stored (`0x4000` to `0xCFFF`).
  11776 addresses of strings can be stored (`0xD000` to `0xFDFF`).
- `PUTSP` is used instead of `PUTS` to save memory.
- Screen and memory is cleared when (re)starting the program.
- The program can be restarted after it’s finished.

III. Instructions

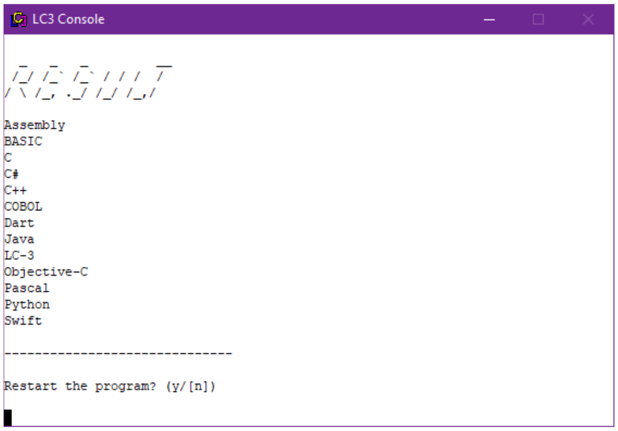
- User will be greeting with a beautiful welcome screen.
- User will input a sequence of strings (without inputing the amount of strings needed to input).
- After inputing all strings, <Enter> twice (<Enter> on a blank line) to end the input procedure.
- A prompt asking whether descending order is applied, y to accept, other keys to decline.
- Sorted strings array with redundant spaces and commas removed will be displayed.
- A prompt asking if user want to restart the program, y to accept, other keys to end the program.

IV. Images



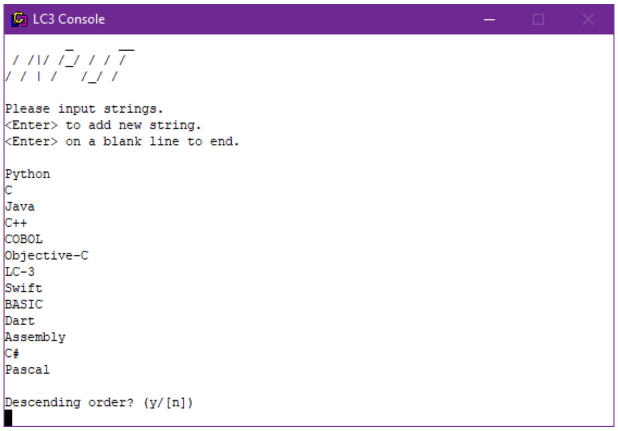
```
 __  ___  __         __     __   __   __  ___         __
/__`  |  |__) | |\ | / _`   /__` /  \ |__)  |  | |\ | / _`
.__/  |  |  \ | | \| \__>   .__/ \__/ |  \  |  | | \| \__>
 __   __   __   __   __
|__) |__) /  \ / _` |__)  /\   |\/|
|    |  \ \__/ \__> |  \ /~~\  |  |


------------------------------


       _     __
 / /|/ /_/ / / /
/ / | /   /_/ /

Please input strings.
<Enter> to add new string.
<Enter> on a blank line to end.
```

Figure 1 - Welcome Screen



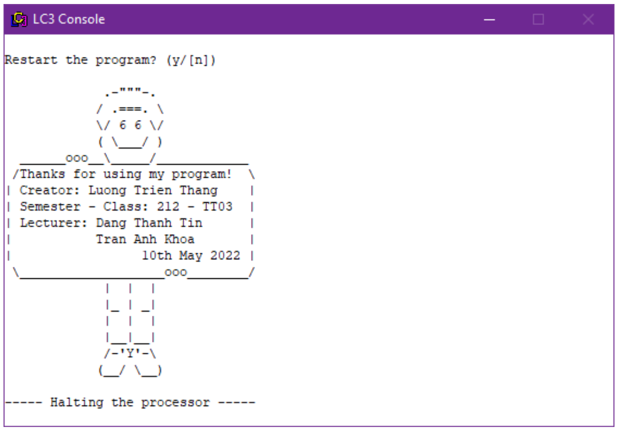
```
   _   _   _       __
 /_/ /_` /_` / / / /
/ \ /_, ._/ /_/ /_,/

Assembly
BASIC
C
C#
C++
COBOL
Dart
Java
LC-3
Objective-C
Pascal
Python
Swift

------------------------------

Restart the program? (y/[n])
```

Figure 2 - Result of sorted strings



```
       _     __
 / /|/ /_/ / / /
/ / | /   /_/ /

Please input strings.
<Enter> to add new string.
<Enter> on a blank line to end.

Python
C
Java
C++
COBOL
Objective-C
LC-3
Swift
BASIC
Dart
Assembly
C#
Pascal

Descending order? (y/[n])
```

Figure 3 - Input strings

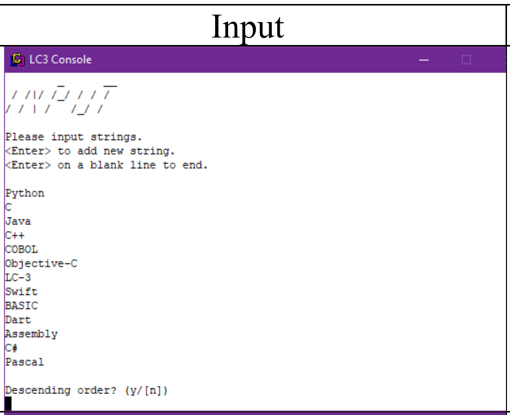


```
Restart the program? (y/[n])

             .-"""-.
            / .===. \
            \/ 6 6 \/
            ( \___/ )
  ______ooo__\_____/____________
 /Thanks for using my program!  \
| Creator: Luong Trien Thang     |
| Semester - Class: 212 - TT03   |
| Lecturer: Dang Thanh Tin       |
|           Tran Anh Khoa        |
|                 10th May 2022  |
 \___________________ooo________/
             |  |  |
             |_ | _|
             |  |  |
             |__|__|
             /-'Y'-\
            (__/ \__)

----- Halting the processor -----
```

Figure 4 - Program ends

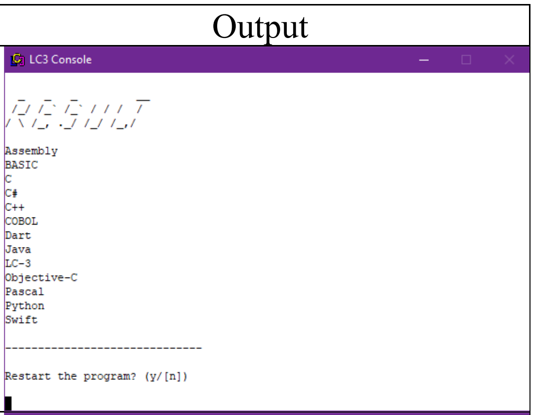
V. Demonstration

<table>
<tr><th></th><th>Input</th><th>Output</th></tr>
<tr><td>One-word strings</td><td>



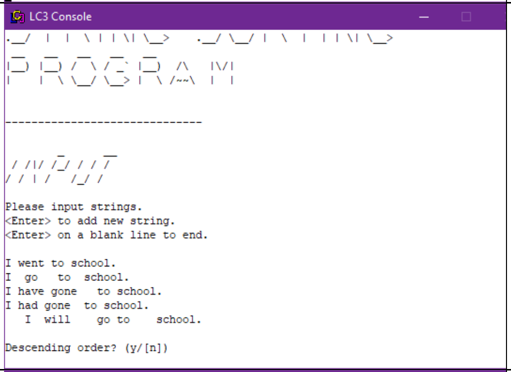
```
      _    __
 / /|/ /_/ / / /
/ / | /   /_/ /

Please input strings.
<Enter> to add new string.
<Enter> on a blank line to end.

Python
C
Java
C++
COBOL
Objective-C
LC-3
Swift
BASIC
Dart
Assembly
C#
Pascal

Descending order? (y/[n])
```

</td><td>



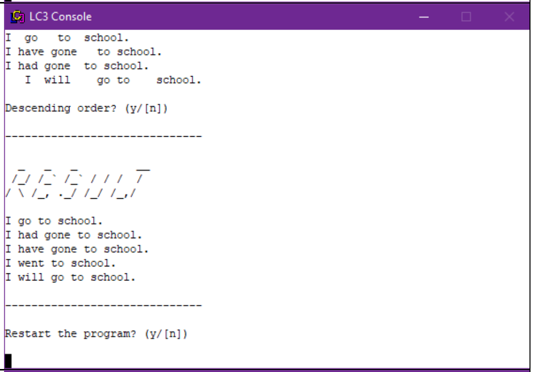
```
  _   _   _        __
 /_/ /_` /_` / / /  /
/ \ /_, ._/ /_/ /_,/

Assembly
BASIC
C
C#
C++
COBOL
Dart
Java
LC-3
Objective-C
Pascal
Python
Swift

------------------------------

Restart the program? (y/[n])
```

</td></tr>
<tr><td>Strings with redundant spaces</td><td>



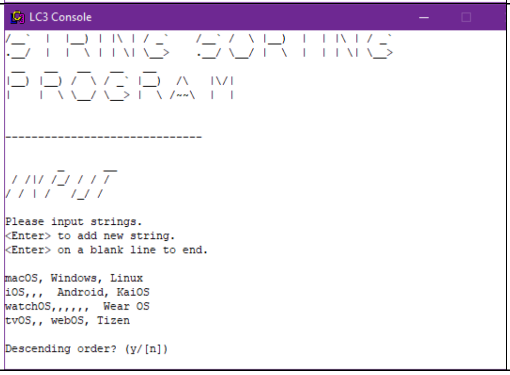
```
.__/  |  |  \ | | \| \__>    .__/ \__/ |  \  |   | | \| \__>
 __   __   __   __   __
|__) |__) /  \ / _` |__)  /\   |\/|
|    |  \ \__/ \__> |  \ /~~\  |  |


------------------------------


      _    __
 / /|/ /_/ / / /
/ / | /   /_/ /

Please input strings.
<Enter> to add new string.
<Enter> on a blank line to end.

I went to school.
I  go   to  school.
I have gone   to school.
I had gone  to school.
   I  will    go to    school.

Descending order? (y/[n])
```

</td><td>



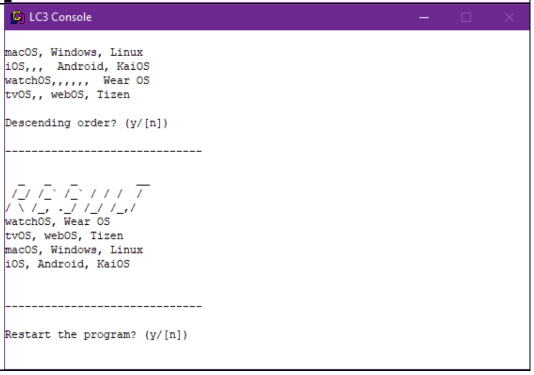
```
I  go   to  school.
I have gone   to school.
I had gone  to school.
   I  will    go to    school.

Descending order? (y/[n])

------------------------------


  _   _   _        __
 /_/ /_` /_` / / /  /
/ \ /_, ._/ /_/ /_,/

I go to school.
I had gone to school.
I have gone to school.
I went to school.
I will go to school.

------------------------------

Restart the program? (y/[n])
```

</td></tr>
<tr><td>Strings with redundant spaces and commas with descending order</td><td>



```
/__`  |  |__) | |\ | / _`    /__` /  \ |__)  |   | |\ | / _`
.__/  |  |  \ | | \| \__>    .__/ \__/ |  \  |   | | \| \__>
 __   __   __   __   __
|__) |__) /  \ / _` |__)  /\   |\/|
|    |  \ \__/ \__> |  \ /~~\  |  |


------------------------------


      _    __
 / /|/ /_/ / / /
/ / | /   /_/ /

Please input strings.
<Enter> to add new string.
<Enter> on a blank line to end.

macOS, Windows, Linux
iOS,,,  Android, KaiOS
watchOS,,,,,,  Wear OS
tvOS,, webOS, Tizen

Descending order? (y/[n])
```

</td><td>



```

macOS, Windows, Linux
iOS,,,  Android, KaiOS
watchOS,,,,,,  Wear OS
tvOS,, webOS, Tizen

Descending order? (y/[n])

------------------------------


  _   _   _        __
 /_/ /_` /_` / / /  /
/ \ /_, ._/ /_/ /_,/
watchOS, Wear OS
tvOS, webOS, Tizen
macOS, Windows, Linux
iOS, Android, KaiOS


------------------------------

Restart the program? (y/[n])
```

</td></tr>
</table>

VI. Ideas

- Characters will be stored from `0x4000` to `0xEFFF`.
- Every string will end with `0` (null character).
- Address of every first character of a string will be stored from `0xD000` to `xFDFF`.
- Strings are compared based on strcmp function in C.
- When sorting, only addresses are changed, not the characters.

VII. Algorithms

1. String compare (strcmp)

- Input: str1, str2.
- Return: 1 if str1 > str2, else 0.
- Let str1, str2 as 2 strings, i as index.
- Denote str1[i], str2[i] as ith character of str1, str2.
- Procedure:

```
    If str1[i] = str2[i]
        If str1[i] ≠ 0 and str2[i] ≠ 0 then i = i +1 and loop.
    Return str1[i] - str2[i]
```

- C code:

```
int strcmp(char *str1, char *str2){
    int i = 0;
    while(str1[i] == str2[i] && (str1[i] != '\0' || str2[i] != '\0')) i++;
    return str1[i] - str2[i];
}
```

- LC3 code:
    - Input: R1: str1, R2: str2
    - Return: R4: str1 – str2

```
strcmp
ST R3, saveR3
ST R5, saveR5
AND R5, R5, #0           ; R5 = i = 0
loop
ADD R3, R1, R5           ; Get character address offset of str1
ADD R4, R2, R5           ; Get character address offset of str2
LDR R3, R3, #0           ; if str1[i] = 0
BRz next                 ;    goto next
LDR R4, R4, #0           ; if str2[i] = 0
BRz next                 ;    goto next
NOT R4, R4               ;
ADD R4, R4, #1           ;
ADD R3, R3, R4           ; if str1[i] != str[2]
BRnp next                ;    goto next
ADD R5, R5, #1           ; i = i + 1
BR loop                  ; Loop

next
ADD R3, R1, R5
ADD R4, R2, R5
LDR R3, R3, #0           ; Get str1[i]
LDR R4, R4, #0           ; Get str2[i]
NOT R4, R4
ADD R4, R4, #1
ADD R4, R4, R3           ; R4 = str1[i] - str2[i]
LD R3, saveR3
LD R5, saveR5
RET
```

2. Sorting algorithm: bubble sort

- Time complexity: $O(n^2)$
- Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. [2]
- Let `strArr` as an array of strings, `n` as the size of the array.
- C code with strcmp implementation:

```
for(int i = n-1; i >=0; i--)
    for(int j = 0; j < i; j++)
        if(strcmp(strArr[j], strArr[j+1]) == 1){
            char *t;
            strcpy(t, strArr[j+1]);
            strcpy(strArr[j+1], strArr[j]);
            strcpy(strArr[j], t);
        }
```

- LC3 code with strcmp implementation:
    - Input: `R4`: number of strings

```
sortRoutine
ST R0, saveR0
ST R1, saveR1
ST R2, saveR2
ST R4, saveR4
ST R5, saveR5
ST R6, saveR6
ST R7, saveR7

ADD R5, R3, #0          ; i
ADD R6, R5, #0          ; j

outer                   ; loopn n times (i from n to 0)
ADD R5, R5, #-1         ; Decrease i
ADD R6, R5, #0          ; j = i
BRn doneSort            ; If i or j is negative, then done
LD R0, stringArray      ; Load string addresses array
inner                   ; loop i times
LDR R1, R0, #0          ; Get a string
LDR R2, R0, #1          ; Get the next string
JSR strcmp              ; R4 <- strcmp(R1, R2) > 0
ADD R4, R4, #0          ; Check if R4 <= 0?
BRnz swapped            ; If no, then no swap
; swap                  ; Else, swap
STR R1, R0, #1
STR R2, R0, #0
swapped
ADD R0, R0, #1          ; Increase the address
ADD R6, R6, #-1         ; Decrease j
BRp inner               ; If positive, loop inner loop
BR outer                ; Else loop outer loop
```

```
doneSort
LD R0, saveR0
LD R1, saveR1
LD R2, saveR2
LD R4, saveR4
LD R5, saveR5
LD R6, saveR6
LD R7, saveR7
BR return
```

VIII. Main code

1. Global variables and declarations

```
; DEFINE R3: strings count
saveR0 .FILL #0
saveR1 .FILL #0
saveR2 .FILL #0
saveR3 .FILL #0
saveR4 .FILL #0
saveR5 .FILL #0
saveR6 .FILL #0
saveR7 .FILL #0
yNeg .FILL #-121
stringAddress .FILL x4000
stringArray .FILL xD000
```

2. A look at PUTSP TRAP routine

- The traditional `PUTS` (PUT String) reads and prints each 2-byte ASCII in an address.
- `PUTSP` (PUT String Packed) reads and decode two 1-byte ASCIIs in an address.
- Example: "Hello" string needs to be encoded like this

PUTS	PUTSP
`0x0048   ; H`	`0x6548  ; eH`
`0x0065   ; e`	`0x6C6C  ; ll`
`0x006C   ; l`	`0x006F  ; \0o`
`0x006C   ; l`	
`0x006F   ; o`	
`0x0000   ; \0` (null character)	

- Thus, `PUTSP` will save half the amount of memory.
- Since there is no (or I haven't found yet) ultility or program to encode, I wrote a simple program in Python to encode.

```
string = """Hello"""
for i in range(0, int(len(string)), 2):
    firstByte = format(ord(string[i]), "02x")
    try:
        nextByte = format(ord(string[i+1]), "02x")
    except:
        nextByte = "00"
    print("\t.FILL x"+nextByte+firstByte)
print("\t.FILL x0000")
```

- Using this, we can print a large ASCII art text without worry about the PCOffset issue.

3. Subroutines

a. UI related

```
; Print dashed line
; (Print 30 dashes)
printLine
ST R0, saveR0
ST R1, saveR1
ST R7, saveR7
AND R0, R0, #0
ADD R0, R0, #10 ; newline
OUT
LD R0, dash
AND R1, R1, #0
ADD R1, R1, #15
ADD R1, R1, #15
printLineLoop
OUT
ADD R1, R1, #-1
BRp printLineLoop
AND R0, R0, #0
ADD R0, R0, #10 ; newline
OUT
OUT
LD R0, saveR0
LD R1, saveR1
LD R7, saveR7
RET
dash .FILL #45
```

```
; Clear screen
; (Print 15 new lines)
clearScreen
ST R0, saveR0
ST R1, saveR1
ST R7, saveR7
AND R1, R1, #0
ADD R1, R1, #15
newLineLoop
AND R0, R0, #0
ADD R0, R0, #10 ; newline
OUT
ADD R1, R1, #-1
BRp newLineLoop
LD R0, saveR0
LD R1, saveR1
LD R7, saveR7
RET
```

b. Input

- Keys with ASCII less than 32 is ignored (eg. Backspace, Escape, etc.).
- <Enter> on a blank line will end the input procedure.
- Redundant spaces and commas still echo, but not stored.
  Spaces at the start of string also count as redundant spaces.
- `R4`, `R5` keep track of space, comma input.
  - If space/comma is typed, `R4`/`R5` will be added by `1`.
  - The program will not store space/comma if `R4`/`R5` ≠ `0`.
  - If other key has pressed, `R4`/`R5` will be cleared.

```
inputRoutine
ST R0, saveR0
ST R1, saveR1
ST R4, saveR4
ST R5, saveR5
ST R6, saveR6
ST R7, saveR7

LD R1, stringAddress     ; R1: character pointer
LD R2, stringArray       ; R2: string address pointer
```

```
AND R3, R3, #0          ; n: number of strings
ADD R3, R3, #1          ; n = n + 1
resetCounter
AND R4, R4, #0          ; Keep track of spaces (spaces counter)
ADD R4, R4, #1          ; Prevent start with space(s)
AND R5, R5, #0          ; Keep track of commas (commas counter)
STR R1, R2, #0          ; Store first string address
LOOP
GETC                    ; Get a character
ADD R6, R0, #-10        ; Check if <Enter>
BRz enter               ;     is pressed?
ADD R6, R6, #-15        ; Check
ADD R6, R6, #-7         ;     if <Space>
BRn LOOP                ;          or ASCII < 32
BRz space               ;               is pressed?
ADD R6, R6, #-12        ; Check if <,>
BRz comma               ;     is pressed?
OUT                     ; Print character
AND R4, R4, #0          ; Reset spaces counter
AND R5, R5, #0          ; Reset commas counter
BR store                ; Goto store

space
AND R5, R5, #0          ; Reset commas counter
OUT                     ; Print character
ADD R4, R4, #0          ; Check if space counter
BRp LOOP                ;     is positive?
ADD R4, R4, #1          ; Increase spaces counter
BR store                ; Goto store

comma
AND R4, R4, #0          ; Reset spaces counter
OUT                     ; Print character
ADD R5, R5, #0          ; Check if commas counter
BRp LOOP                ;     is positive?
ADD R5, R5, #1          ; Increase commas counter

store
STR R0, R1, #0          ; Store the character
ADD R1, R1, #1          ; Increase R1 pointer
BR LOOP                 ; Loop

enter
OUT                     ; Print character
LDR R7, R1, #-1 ; Check if previous character is a null
BRz doneInput   ;    character? (to check blank line)
ADD R1, R1, #1          ; Increase R1 pointer
ADD R2, R2, #1          ; Increase R2 pointer
STR R1, R2, #0          ;     and store
ADD R3, R3, #1          ; Increase n
BR resetCounter         ; Goto resetCounter (loop)
```

```
doneInput
LD R0, saveR0
LD R1, saveR1
LD R4, saveR4
LD R5, saveR5
LD R6, saveR6
LD R7, saveR7
BR return
```

c. *Sort (explained in VII.2)*

d. *Output*

- Input:
  - o `R3`: number of strings
  - o `R5`: step
  - o `R6`: from address
- Order (need to be set before calling subroutine):
  - o Ascending: `R5 = 1`, `R6 = stringArray`
  - o Descending: `R5 = -1`, `R6 = stringArray + R3 - 1`

```
outputFunc
ST R0, saveR0
ST R3, saveR3
ST R6, saveR6
ST R7, saveR7
output
ADD R3, R3, #-1
BRn outputDone          ; Check if the end of array
LDR R0, R6, #0          ; Get the address from stringArray
BRz outputDone          ; Check for blank array
PUTS
AND R0, R0, #0
ADD R0, R0, #10
OUT                     ; Newline
ADD R6, R6, R5          ; R6 is increased/decreased depends on R5
BR output               ; Loop
outputDone
LD R0, saveR0
LD R3, saveR3
LD R6, saveR6
LD R7, saveR7
RET
```

e. *Miscellaneous*

```
; Reset All
; Reset all data from x4000 to xFCFF and registers (except R7)
resetAll
AND R0, R0, #0
LD R1, resetCount
LD R2, stringAddress
resetLoop
STR R0, R2, #0
ADD R2, R2, #1
```

```
ADD R1, R1, #-1
BRnp resetLoop
AND R0, R0, #0
AND R1, R0, #0
AND R2, R0, #0
AND R3, R0, #0
AND R4, R0, #0
AND R5, R0, #0                       resetCount .FILL xBCFF
AND R6, R0, #0                       stringAddress .FILL x4000
RET
```

4. Main program

```
mainProgram
;;; Reset. Input. Sort
JSR clearScreen
LEA R0, title           ; Print title
PUTSP
JSR resetAll            ; Reset all data
JSR printLine           ; Print a line
LEA R0, inputMsg        ; Print input text
PUTSP
LEA R0, instrMsg        ; Print instruction message
PUTSP
JSR inputRoutine        ; Call input routine
JSR sortRoutine         ; Call sort routine

;;; Asking to print in descending
; If <y> is pressed, descending is chosen, else ascending.
AND R5, R5, #0          ; R5: step
LD R6, stringArray      ; R6: from
LEA R0, orderMsg        ; Prompt a asking message
PUTSP
GETC                    ; Get a character
LD R1, yNeg             ; Check if <y>
ADD R1, R1, R0          ;     is pressed?
BRnp ascending          ; If not, ascending is chosen
ADD R5, R5, #-1         ; R5 = -1
ADD R6, R6, R3
ADD R6, R6, #-1         ; R6 += number of strings - 1
BR print                ; Goto print

ascending
ADD R5, R5, #1          ; R5 = 1

;;; Output
print
JSR printLine           ; Print a line
LEA R0, resultMsg       ; Print result text
PUTSP
JSR outputFunc          ; Call output subroutine

JSR printLine           ; Print a line
LEA R0, restartMsg      ; Prompt for asking restart program
PUTSP
GETC
LD R1, yNeg             ; Check if <y>
ADD R1, R1, R0          ;     is pressed?
BRz mainProgram         ; If yes, restart the program
HALT                    ; Else HALT
```

5. Full code (with ASCII art texts/strings)

```
.ORIG x3000
;; DEFINE R3: strings count
BR mainProgram
;; UI
;;;;;;;;;;;
; STRINGS ;
;;;;;;;;;;;
title
.FILL x200a
.FILL x5f5f
.FILL x2020
.FILL x5f5f
.FILL x205f
.FILL x5f20
.FILL x205f
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x5f20
.FILL x205f
.FILL x2020
.FILL x2020
.FILL x5f20
.FILL x205f
.FILL x2020
.FILL x5f5f
.FILL x2020
.FILL x5f20
.FILL x205f
.FILL x5f20
.FILL x5f5f
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x5f20
.FILL x0a5f
.FILL x5f2f
.FILL x605f
.FILL x2020
.FILL x207c
.FILL x7c20
.FILL x5f5f
.FILL x2029
.FILL x207c
.FILL x5c7c
.FILL x7c20
.FILL x2f20
.FILL x5f20
.FILL x2060
.FILL x2020
.FILL x2f20
.FILL x5f5f
.FILL x2060
.FILL x202f
.FILL x5c20
.FILL x7c20
.FILL x5f5f
.FILL x2029
.FILL x7c20
.FILL x2020
.FILL x207c
.FILL x5c7c
.FILL x7c20
.FILL x2f20
.FILL x5f20
.FILL x0a60
.FILL x5f2e
.FILL x2f5f
.FILL x2020
.FILL x207c
.FILL x7c20
.FILL x2020
.FILL x205c
.FILL x207c
.FILL x207c
.FILL x7c5c
.FILL x5c20
.FILL x5f5f
.FILL x203e
.FILL x2020
.FILL x2e20
.FILL x5f5f
.FILL x202f
.FILL x5f5c
.FILL x2f5f
.FILL x7c20
.FILL x2020
.FILL x205c
.FILL x7c20
.FILL x2020
.FILL x207c
.FILL x207c
.FILL x7c5c
.FILL x5c20
.FILL x5f5f
.FILL x0a3e
.FILL x5f20
.FILL x205f
.FILL x2020
.FILL x5f5f
.FILL x2020
.FILL x5f20
.FILL x205f
.FILL x2020
.FILL x5f5f
.FILL x2020
.FILL x5f20
.FILL x0a5f
.FILL x5f7c
.FILL x295f
.FILL x7c20
.FILL x5f5f
.FILL x2029
.FILL x202f
.FILL x5c20
.FILL x2f20
.FILL x5f20
.FILL x2060
.FILL x5f7c
.FILL x295f
.FILL x2020
.FILL x5c2f
.FILL x2020
.FILL x7c20
.FILL x2f5c
.FILL x0a7c
.FILL x207c
.FILL x2020
.FILL x7c20
.FILL x2020
.FILL x205c
.FILL x5f5c
.FILL x2f5f
.FILL x5c20
.FILL x5f5f
.FILL x203e
.FILL x207c
.FILL x5c20
.FILL x2f20
.FILL x7e7e
.FILL x205c
.FILL x7c20
.FILL x2020
.FILL x0a7c
.FILL x0000
```

```
; "Please input strings.\n<Enter> to add new string.\n<Enter> on blank line to end\n"
instrMsg
.FILL x6c50
.FILL x6165
.FILL x6573
.FILL x6920
.FILL x706e
.FILL x7475
.FILL x7320
.FILL x7274
.FILL x6e69
.FILL x7367
.FILL x0a2e
.FILL x453c
.FILL x746e
.FILL x7265
.FILL x203e
.FILL x6f74
.FILL x6120
.FILL x6464
.FILL x6e20
.FILL x7765
.FILL x7320
.FILL x7274
.FILL x6e69
.FILL x2e67
.FILL x3c0a
.FILL x6e45
.FILL x6574
.FILL x3e72
.FILL x6f20
.FILL x206e
.FILL x2061
.FILL x6c62
.FILL x6e61
.FILL x206b
.FILL x696c
.FILL x656e
.FILL x7420
.FILL x206f
.FILL x6e65
.FILL x2e64
.FILL x000a

orderMsg
.FILL x6544
.FILL x6373
.FILL x6e65
.FILL x6964
.FILL x676e
.FILL x6f20
.FILL x6472
.FILL x7265
.FILL x203f
.FILL x7928
.FILL x5b2f
.FILL x5d6e
.FILL x0029

inputMsg
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x205f
.FILL x2020
.FILL x2020
.FILL x5f20
.FILL x0a5f
.FILL x2f20
.FILL x2f20
.FILL x2f7c
.FILL x2f20
.FILL x2f5f
.FILL x2f20
.FILL x2f20
.FILL x2f20
.FILL x2f0a
.FILL x2f20
.FILL x7c20
.FILL x2f20
.FILL x2020
.FILL x2f20
.FILL x2f5f
.FILL x2f20
.FILL x000a

resultMsg
.FILL x2020
.FILL x205f
.FILL x2020
.FILL x205f
.FILL x2020
.FILL x205f
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x5f5f
.FILL x200a
.FILL x5f2f
.FILL x202f
.FILL x5f2f
.FILL x2060
.FILL x5f2f
.FILL x2060
.FILL x202f
.FILL x202f
.FILL x202f
.FILL x2f20
.FILL x2f0a
.FILL x5c20
.FILL x2f20
.FILL x2c5f
.FILL x2e20
.FILL x2f5f
.FILL x2f20
.FILL x2f5f
.FILL x2f20
.FILL x2c5f
.FILL x002f
```

```
;;;;;;;;;;;;;;;;
; MAIN PROGRAM ;
;;;;;;;;;;;;;;;;
mainProgram
;;; Reset. Input. Sort
JSR clearScreen
LEA R0, title                                   ; Print title
PUTSP
JSR resetAll                                    ; Reset all data
JSR printLine                                   ; Print a line
LEA R0, inputMsg                                ; Print input text
PUTSP
LEA R0, instrMsg                                ; Print instruction message
PUTSP
JSR inputRoutine                                ; Call input routine
JSR sortRoutine                                 ; Call sort routine

;;; Asking to print in descending
; If <y> is pressed, descending is chosen, else ascending.
AND R5, R5, #0                                  ; R5: step
LD R6, stringArray                              ; R6: from
LEA R0, orderMsg                                ; Prompt a asking message
PUTSP
GETC                                            ; Get a character
LD R1, yNeg                                     ; Check if <y>
ADD R1, R1, R0                                  ;     is pressed?
BRnp ascending                                  ; If not, ascending is chosen
ADD R5, R5, #-1                                 ; R5 = -1
ADD R6, R6, R3
ADD R6, R6, #-1                                 ; R6 += number of strings - 1
BR print                                        ; Goto print

ascending
ADD R5, R5, #1                                  ; R5 = 1

;;; Output
print
```

```
JSR printLine                                   ; Print a line
LEA R0, resultMsg                               ; Print result text
PUTSP
JSR outputFunc                                  ; Call output subroutine

JSR printLine                                   ; Print a line
LEA R0, restartMsg                              ; Prompt for asking restart program
PUTSP
GETC
LD R1, yNeg                                     ; Check if <y>
ADD R1, R1, R0                                  ;     is pressed?
BRz mainProgram                                 ; If yes, restart the program
LEA R0, goodbyeMsg
PUTSP
HALT                                            ; Else HALT
restartMsg
.FILL x6552       |  .FILL x2074       |  .FILL x7270       |  .FILL x3f6d       |  .FILL x6e5b
.FILL x7473       |  .FILL x6874       |  .FILL x676f       |  .FILL x2820       |  .FILL x295d
.FILL x7261       |  .FILL x2065       |  .FILL x6172       |  .FILL x2f79       |  .FILL x000a

;;;;;;;;;;;;;;;
; SUBROUTINES ;
;;;;;;;;;;;;;;;

;;;;;;;;;;; UI RELATED
; Print line
; (Print 30 dashes)
printLine
ST R0, saveR0
ST R1, saveR1
ST R7, saveR7
AND R0, R0, #0
ADD R0, R0, #10                                 ; newline
OUT
LD R0, dash
AND R1, R1, #0
ADD R1, R1, #15
ADD R1, R1, #15
printLineLoop
OUT
ADD R1, R1, #-1
BRp printLineLoop
AND R0, R0, #0
ADD R0, R0, #10                                  ; newline
OUT
OUT
LD R0, saveR0
LD R1, saveR1
LD R7, saveR7
RET
dash .FILL #45

; Clear screen
; (Print 15 new lines)
clearScreen
ST R0, saveR0
ST R1, saveR1
ST R7, saveR7
AND R1, R1, #0
ADD R1, R1, #15
newLineLoop
AND R0, R0, #0
ADD R0, R0, #10                                  ; newline
OUT
ADD R1, R1, #-1
BRp newLineLoop
LD R0, saveR0
LD R1, saveR1
LD R7, saveR7
RET

;;;;;;;;;;; MISC
;;; resetAll
;;; Reset all data from x4000 to xFCFF and registers (except R7)
```

```
resetAll
AND R0, R0, #0
LD R1, resetCount                                ; counter
LD R2, stringAddress
resetLoop
STR R0, R2, #0
ADD R2, R2, #1
ADD R1, R1, #-1
BRnp resetLoop
AND R0, R0, #0
AND R1, R0, #0
AND R2, R0, #0
AND R3, R0, #0
AND R4, R0, #0
AND R5, R0, #0
AND R6, R0, #0
RET
resetCount .FILL xBCFF


; Return
return
RET

;;;;;;;;;;; MAIN SUBROUTINES
;;; INPUT
;;; Returns:
;;;   R3: numbers of strings
inputRoutine
ST R0, saveR0
ST R1, saveR1
ST R4, saveR4
ST R5, saveR5
ST R6, saveR6
ST R7, saveR7

LD R1, stringAddress                             ; R1: character pointer
LD R2, stringArray                              ; R2: string address pointer

AND R3, R3, #0                                  ; n: number of strings
ADD R3, R3, #1                                  ; n = n + 1
resetCounter
AND R4, R4, #0                                  ; Keep track of spaces (spaces counter)
ADD R4, R4, #1                                  ; Prevent start with space(s)
AND R5, R5, #0                                  ; Keep track of commas (commas counter)

STR R1, R2, #0                                  ; Store first string address

LOOP
GETC                                            ; Get a character
ADD R6, R0, #-10                                ; Check if <Enter>
BRz enter                                       ;     is pressed?
ADD R6, R6, #-15                                ; Check
ADD R6, R6, #-7                                 ;     if <Space>
BRn LOOP                                        ;          or ASCII < 32
BRz space                                       ;                is pressed?
ADD R6, R6, #-12                                ; Check if <,>
BRz comma                                       ;     is pressed?
OUT                                             ; Print character
AND R4, R4, #0                                  ; Reset spaces counter
AND R5, R5, #0                                  ; Reset commas counter
BR store                                        ; Goto store

space
AND R5, R5, #0                                  ; Reset commas counter
OUT                                             ; Print character
ADD R4, R4, #0                                  ; Check if space counter
BRp LOOP                                        ;     is positive?
ADD R4, R4, #1                                  ; Increase spaces counter
BR store                                        ; Goto store

comma
AND R4, R4, #0                                  ; Reset spaces counter
OUT                                             ; Print character
```

```
ADD R5, R5, #0                                  ; Check if commas counter
BRp LOOP                                        ;     is positive?
ADD R5, R5, #1                                  ; Increase commas counter

store
STR R0, R1, #0                                  ; Store the character
ADD R1, R1, #1                                  ; Increase R1 pointer
BR LOOP                                         ; Loop

enter
OUT                                             ; Print character
LDR R7, R1, #-1                                 ; Check if previous character is a null
BRz doneInput                                   ;     character? (to check blank line)
ADD R1, R1, #1                                  ; Increase R1 pointer
ADD R2, R2, #1                                  ; Increase R2 pointer
STR R1, R2, #0                                  ;     and store
ADD R3, R3, #1                                  ; Increase n
BR resetCounter                                 ; Goto resetCounter (loop)

doneInput
LD R0, saveR0
LD R1, saveR1
LD R4, saveR4
LD R5, saveR5
LD R6, saveR6
LD R7, saveR7
BR return

;;; SORT
;;; Using bubble sort
;;; Input: R3: number of srings
sortRoutine
ST R0, saveR0
ST R1, saveR1
ST R2, saveR2
ST R4, saveR4
ST R5, saveR5
ST R6, saveR6
ST R7, saveR7

ADD R5, R3, #0                                  ; i
ADD R6, R5, #0                                  ; j

outer                                           ; loopn n times (i from n to 0)
ADD R5, R5, #-1                                 ; Decrease i
ADD R6, R5, #0                                  ; j = i
BRn doneSort                                    ; If i or j is negative, then done
LD R0, stringArray                              ; Load string addresses array
inner                                           ; loop i times
LDR R1, R0, #0                                  ; Get a string
LDR R2, R0, #1                                  ; Get the next string
JSR strcmp                                      ; R4 <- strcmp(R1, R2) > 0
ADD R4, R4, #0                                  ; Check if R4 <= 0?
BRnz swapped                                    ; If no, then no swap
; swap                                          ; Else, swap
STR R1, R0, #1
STR R2, R0, #0

swapped
ADD R0, R0, #1                                  ; Increase the address
ADD R6, R6, #-1                                 ; Decrease j
BRp inner                                       ; If positive, loop inner loop
BR outer                                        ; Else loop outer loop

doneSort
LD R0, saveR0
LD R1, saveR1
LD R2, saveR2
LD R4, saveR4
LD R5, saveR5
LD R6, saveR6
LD R7, saveR7
BR return
```

```
;;; strcmp(R1: str1, R2: str2)
;;; Manipulating strcmp of String.h in C
;;; return R4 <- 1 if strcmp(R1, R2) > 0 else 0
strcmp
ST R3, saveR3
ST R5, saveR5
AND R5, R5, #0                                   ; R5 = i = 0
loop
ADD R3, R1, R5                                   ; Get character address offset of str1
ADD R4, R2, R5                                   ; Get character address offset of str2
LDR R3, R3, #0                                   ; if str1[i] = 0
BRz next                                         ;     goto next
LDR R4, R4, #0                                   ; if str2[i] = 0
BRz next                                         ;     goto next
NOT R4, R4                                       ;
ADD R4, R4, #1                                   ;
ADD R3, R3, R4                                   ; if str1[i] != str[2]
BRnp next                                        ;     goto next
ADD R5, R5, #1                                   ; i = i + 1
BR loop                                          ; Loop

next
ADD R3, R1, R5
ADD R4, R2, R5
LDR R3, R3, #0                                   ; Get str1[i]
LDR R4, R4, #0                                   ; Get str2[i]
NOT R4, R4
ADD R4, R4, #1
ADD R4, R4, R3                                   ; R4 = str1[i] - str2[i]
LD R3, saveR3
LD R5, saveR5
RET


; OUTPUT(R3: number of strings, R5: step, R6: from)
outputFunc
ST R0, saveR0
ST R3, saveR3
ST R6, saveR6
ST R7, saveR7
output
ADD R3, R3, #-1
BRn outputDone                                  ; Check if the end of array
LDR R0, R6, #0                                  ; Get the address from stringArray
BRz outputDone                                  ; Check for blank array
PUTS
AND R0, R0, #0
ADD R0, R0, #10
OUT                                             ; Newline
ADD R6, R6, R5                                  ; R6 is increased/decreased depends on R5
BR output                                       ; Loop

outputDone
LD R0, saveR0
LD R3, saveR3
LD R6, saveR6
LD R7, saveR7
RET


;;;;;;;;;;;;;
; VARIABLES ;
;;;;;;;;;;;;;
saveR0 .FILL #0
saveR1 .FILL #0
saveR2 .FILL #0
saveR3 .FILL #0
saveR4 .FILL #0
saveR5 .FILL #0
saveR6 .FILL #0
saveR7 .FILL #0
yNeg .FILL #-121
stringAddress .FILL x4000
stringArray .FILL xD000
```

```
goodbyeMsg
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2e20
.FILL x222d
.FILL x2222
.FILL x2e2d
.FILL x200a
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2f20
.FILL x2e20
.FILL x3d3d
.FILL x2e3d
.FILL x5c20
.FILL x200a
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x5c20
.FILL x202f
.FILL x2036
.FILL x2036
.FILL x2f5c
.FILL x200a
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2820
.FILL x5c20
.FILL x5f5f
.FILL x2f5f
.FILL x2920
.FILL x200a
.FILL x5f20
.FILL x5f5f
.FILL x5f5f
.FILL x6f5f
.FILL x6f6f
.FILL x5f5f
.FILL x5f5c
.FILL x5f5f
.FILL x5f5f
.FILL x5f2f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x0a5f
.FILL x2f20
.FILL x6854
.FILL x6e61
.FILL x736b
.FILL x6620
.FILL x726f
.FILL x7520
.FILL x6973
.FILL x676e
.FILL x6d20
.FILL x2079
.FILL x7270
.FILL x676f
.FILL x6172
.FILL x216d
.FILL x2020
.FILL x0a5c
.FILL x207c
.FILL x7243
.FILL x6165
.FILL x6f74
.FILL x3a72
.FILL x4c20
.FILL x6f75
.FILL x676e
.FILL x5420
.FILL x6972
.FILL x6e65
.FILL x5420
.FILL x6168
.FILL x676e
.FILL x2020
.FILL x2020
.FILL x0a7c
.FILL x207c
.FILL x6553
.FILL x656d
.FILL x7473
.FILL x7265
.FILL x2d20
.FILL x4320
.FILL x616c
.FILL x7373
.FILL x203a
.FILL x3132
.FILL x2032
.FILL x202d
.FILL x5454
.FILL x3330
.FILL x2020
.FILL x0a7c
.FILL x207c
.FILL x654c
.FILL x7463
.FILL x7275
.FILL x7265
.FILL x203a
.FILL x6144
.FILL x676e
.FILL x5420
.FILL x6168
.FILL x686e
.FILL x5420
.FILL x6e69
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x0a7c
.FILL x207c
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x7254
.FILL x6e61
.FILL x4120
.FILL x686e
.FILL x4b20
.FILL x6f68
.FILL x2061
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x0a7c
.FILL x207c
.FILL x2009
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x3031
.FILL x6874
.FILL x4d20
.FILL x7961
.FILL x3220
.FILL x3230
.FILL x2032
.FILL x0a7c
.FILL x5c20
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x6f5f
.FILL x6f6f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x5f5f
.FILL x0a2f
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x7c20
.FILL x2020
.FILL x207c
.FILL x7c20
.FILL x200a
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x5f7c
.FILL x7c20
.FILL x5f20
.FILL x0a7c
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x7c20
.FILL x2020
.FILL x207c
.FILL x7c20
.FILL x200a
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x5f7c
.FILL x7c5f
.FILL x5f5f
.FILL x0a7c
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2f20
.FILL x272d
.FILL x2759
.FILL x5c2d
.FILL x200a
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2020
.FILL x2820
.FILL x5f5f
.FILL x202f
.FILL x5f5c
.FILL x295f
.FILL x0000
.END
```

IX. Appendix

The source code will be available on Github after 10th May, 2022 via this link.

X. References

[1] cplusplus.com, "strcmp - C++ Reference," [Online]. Available: https://www.cplusplus.com/reference/cstring/strcmp/.

[2] Wikipedia, "Bubble sort," [Online]. Available: https://en.wikipedia.org/wiki/Bubble_sort.