Final Project CS 2XA3 2016/17, Term I

The specification release date: November 7, 2016
The submission of deliverables deadline: December 16, 2016, by 23:00 hours
The electronic submission consists of uploading the two deliverable files: `sufsort.asm` and `makefile` by the deadline. The submission can be performed any time between the posting of the project and the deadline, and the project can be submitted multiple times, just like the lab projects.

Your task is to write a NASM program. The executable should be called `sufsort` (short for suffix sort). The program `sufsort` gets a string of length 1 to 30 as an input via the command line argument, for instance executing `sufsort abbacd` will "pass" to the program the string `abbacd`, while executing `sufsort aabaabba` will "pass" to the program the string `aabaabba`. The string is to be stored in a byte array terminated by 0 in the program's memory, then displayed on the screen, then its suffixes are sorted according to the lexicographical ordering and displayed on the screen in the ascending manner. For definition of lexicographical ordering see for instance wikipedia at `https://en.wikipedia.org/wiki/Lexicographical_order`

The **deliverables** are:

1. the source code of the program called `sufsort.asm`, and
2. the corresponding `makefile`. Typing `make sufsort` must compile, assemble and link the program and create the executable `sufsort`. Typing `make clean` must remove `sufsort.o` and `sufsort`.
3. Your program must use for the I/O routines the routines of `asm_io.asm` and hence in addition to `asm_io.asm` you will also need files `asm_io.inc`, `cdecl.h`, and `driver.c` in order to create the executable `sufsort`. *These files are available for download on the final project website, and are **not to be uploaded** for submission. Even though these files can be found elsewhere on the Internet, use the ones from the final project website as they were specifically tuned for* `moore`.
4. The source code and the makefile must be **specifically** prepared for the `moore` machine! The NASM on other CAS servers are different release versions and may also be configured differently; similarly for any NASM downloaded and/or installed on some other machine. **In simple terms, the project must be able to be assembled, compiled, linked, and executed on** `moore`.

The ***program specification***:

(1) To prevent confusion in terminology, in the following we consider the *first command line argument* to be the name of the program. For instance, if the program is executed by `sufsort aabaabba`, `sufsort` is the 1st command line argument and `aabaabba` is the 2nd command line argument.

(2) The program checks if the number of line arguments is correct, i.e. must equal to 2. If not, an error message is displayed and the program terminates.

(3) The program first accesses the second command line argument, i.e. the string whose suffixes are to be sorted. We refer to it as the ***input string***.

(4) The program checks that the input string is of length between 1 and 30 (inclusive). If not, the program terminates and an error message describing the error in length is displayed.

(5) The letters that can be used in the input string are only characters `'0'`, `'1'`, and `'2'`. So, during the input string length check, the program also checks whether all the letters in the input string are these three characters. If the length was OK, but not the letters, an error message describing the error of improper letters being used is displayed and the program terminates.

(6) If the input string has a correct size and consists of correct letters, it is then copied (by a loop one character at a time) into the program's memory (stored in a byte array terminated with 0). Call this array `X`. The length of the input string is also stored in the memory in a variable called `N`.
For instance, if `sufsort abbacd` was executed on the command line, the array `X` would contain: `X`[0] = `'a'`, `X`[1] = `'b'`, `X`[2] = `'b'`, `X`[3] = `'a'`, `X`[4] = `'c'`, `X`[5] = `'d'`, `X`[6] = `0`, while `N` = 6.

(7) To visually check the input string, the copy of the input string (i.e. the array `X`) is displayed on the screen using the I/O subroutine `print_string()`.

(8) The program must contain a subroutine `sufcmp(Z,i,j)` with three arguments: an address of a string (i.e. a byte array terminated with 0), and two indeces `i` and `j` so that $0 \le \texttt{i} < \texttt{j} < \texttt{N}$. The subroutine returns -1 if the suffix `Z`[`i`..`N`−1] is lexicographically smaller than the suffix `Z`[`j`..`N`−1], and +1 if `Z`[`i`..`N`−1] is lexicographically bigger than `Z`[`j`..`N`−1]. Note that since `i` < `j`, `Z`[`i`..`N`−1] ≠ `Z`[`j`..`N`−1] .

(9) The main section that must be called `asm_main` performs a bubble sort on the suffixes using calls to `sufcmp(Z,i,j)` for the comparisons. After the bubble sort is finished, the order of suffixes is displayed on the screen and the program waits for the user to press enter. After that, the program terminates.

Thus, the program consists of a subroutine (`sufcmp`) and the main section labelled `asm_main`. You have to have these two parts and they must be named accordingly to allow us to comprehend and mark your program. You may not modularize the program any further – i.e. no more subroutines can be used.

A couple of results for testing and debugging of the program and to illustrate how the output should look like:

```
00100102
sorted suffixes:
00100102
00102
0100102
0102
02
100102
102
2

22010100
sorted suffixes:
0
00
0100
010100
100
10100
2010100
22010100

012012012
sorted suffixes:
012
012012
012012012
12
12012
12012012
2
2012
2012012
```

The same program written in Python (except that the input string is passed as a regular argument rather than as a command line argument):

```
def sufcmp(Z,i,j):
        n = len(Z)-i  # length of the suffix Z[i..N-1]
        m = len(Z)-j  # length of the suffix Z[j..N-1]
        k = min(n,m)
        for o in range(k):
                if Z[i+o]<Z[j+o]:
                        return -1
                if Z[i+o]>Z[j+o]:
                        return 1
        # so all the letters are the same
        if n<m:
                return -1
        else:
                return 1
##############################################################
def sufsort(Z):
        # display the input string
        print(Z)
        N = len(Z)
        # check the length of the input string
        if N <= 0 or N > 30:
                print("length error")
                return
        # check the composition of the input string
        for i in range(N):
                if not (Z[i]=='0' or Z[i]=='1' or Z[i]=='2'):
                        print("composition error")
                        return
        # so the input string is fine
        # we will keep the suffix indeces in an array y
        # we create array holding 0, 1, 2, ..., N-1
        y = [];
        for i in range(N):
                y.append(i);

        # so we have an array y = [0,1,2,...N-1]
        # we will sort array y[] using bubble sort and comparing i with j
        # by comparing Z[i..N-1] and Z[j..N-1] using the subroutine
        # sufcmp(Z,i,j)

        for i in range(N,0,-1):
                for j in range(1,i):
                        k = sufcmp(Z,y[j-1],y[j])
                        if k > 0:
                                t = y[j-1]
                                y[j-1]=y[j]
                                y[j]=t

        # display the sorted suffixes
        print("sorted suffixes:")
        for i in range(N):
                print(Z[y[i]:N])
```