Final Project CS/SE 2XA3 2015/16, Term I

The specification release date: November 9, 2015
The submission of deliverables deadline: 11 PM, ~~December 14~~ December 16, 2015
The electronic submission consists of uploading the two deliverable files: lynarr.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 **lynarr** (short for Lyndon Array). **lynarr** gets a string of length 1 to 20 as an input via the command line argument, for instance executing **lynarr *abbacd*** will "give" the program the string *abbacd*, while executing **lynarr *aabaabba*** will "give" the program the string *aabaabba*. The string is to be stored in a byte array in the program's memory.

The **deliverables** are:

1. the source code of the program called **lynarr.asm**, and
2. the corresponding **makefile**. Typing ***make lynarr*** must compile, assemble and link the program and create the executable **lynarr**. Typing ***make clean*** must remove **lynarr.o** and **lynarr**.
3. Your program must use for the I/O routines the program **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 **lynarr**. *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 ***mills*** or other CAS servers may be configured differently; similarly for any NASM downloaded or installed on some other machines. **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 **lynarr *aabaabba***, **lynarr** is the 1st command line argument and ***aabaabba*** the 2nd command line argument.

(2) The program checks if the number of line arguments is correct, i.e. 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 Lyndon array is to be computed. We refer to it as the **input string**.

(4) The program checks that the input string is of length between 1 and 20 (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 the lower case ones , i.e. a, b, $\cdots$, y, z. So, during the length-check the program also checks whether all the letters in the input are lower cases. 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). Call this array X. The length of the input string is also stored in the memory in a variable called N.
For instance, if **lynarr *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' while $N = 6$.

(7) To check that a correct string was input, the copy of the input string (i.e. the array X) is passed to a subroutine **display()** and displayed on the screen. The subroutine **display($Z, n, flag$)** has three arguments, an address of a string (i.e. a byte array) or an integer array Z and its length n. The third argument, $flag$ indicates if Z is an address of a string or an address of an integer array. Based on the value of $flag$, the subroutine either displays Z as a byte array of length n, displaying each byte as a character (using `print_char`), or it displays Z as an integer array of length n, displaying each item as an integer (using `print_int`). In both cases, the subroutine waits for the user to hit enter before terminating (suing `read_char`). Note that if the array Z is being traversed as a byte array, the index must be incremented by 1, while if Z is being traversed as an integer, the index must be incremented by 4.
Thus, the string X, its length N, and a proper value for $flag$ are used as the arguments for **display()**.

(8) The program contains a subroutine **maxLyn(Z, n, k)** with three arguments: a string Z (i.e. a byte array), its length n, and an index $0 \leq k < n$. The subroutine computes and returns the longest Lyndon substring of Z starting at position k. The pseudo-code of the subroutine is given below.

(9) In the main section that must be called `asm_main`, in a loop, for every index $0 \leq k < N$, the subroutine **maxLyn(X, N, k)** is called and the returned value is stored in the k-th item of the (output) array Y.

(10) The resulting array Y is displayed by calling the subroutine **display($Y, N, flag$)** with the appropriate value for $flag$.

(11) After **display** terminates, the program terminates.

Thus, the program consists of two subroutines (**display** and **maxLyn**) and the main section labelled **asm_main**. You have to have these three 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:

aabaabbbabab
12 2 1 9 3 1 1 5 1 3 1 1

abcdefgh
8 7 6 5 4 3 2 1

abababb
7 1 5 1 3 1 1

abbababaaaba
3 1 1 2 1 2 1 4 3 2 1 1

A pseudo-code for **maxLyn** is given below. In general, you should not have a problem to "read" it and understand it.
Symbol $\leftarrow$ indicates "assignment", e.g. $n \leftarrow 2$ means "store 2 in n".
$maxLyn(Z[0\,..\,n-1], n, k)$: *returns integer* means that the subroutine $maxLyn$ expects three arguments, an integer array Z of length n and the position k for which it must find the longest Lyndon substring. The length of the longest Lyndon substring starting at position k is returned (it is an integer).

```
maxLyn(Z[0..n−1], n, k) : returns integer
    if k = n − 1 then return 1
    p ← 1
    for i ← k+1 to n − 1 do
      if Z[i−p] ≠ Z[i] then
        if Z[i−p] > Z[i] then
          return p
        else
          p ← i+1−k
    endfor
    return p
```

For the curious: a string is Lyndon if it is strictly lexicographically smaller than any of its rotations. A ***maximal Lyndon substring of*** X ***at a position*** i *is the longest substring* $X[i..j]$ *which happens to be Lyndon – it always exists though it may be as*

short as one letter or as long as the rest of the string. The Lyndon array Y gives for every i the length of the longest Lyndon substring starting at i.

A hint or suggestion: trying to program it first in Python or any other high-level language you know may simplify the task of programming it in assembler as it will facilitate a better understanding of the program and its structure.