Task. NASM program named `proj4.asm`

The name of your file must be `proj4.asm` and below is a description of what it should do when executed.

Before you start working on any of the programs in the lab project,

1. download `driver.c` to your workstation / laptop, transfer it to ***moore***, and modify it to a UNIX text file by `dos2unix` ;
2. download `simple_io.inc` to your workstation / laptop, transfer it to ***moore***, and modify it to a UNIX text file by `dos2unix` ;
3. download `simple_io.asm` to your workstation / laptop, transfer it to ***moore***, and modify it to a UNIX text file by `dos2unix` ;
4. download `makefile` to your workstation / laptop, transfer it to ***moore***, and modify it to a UNIX text file by `dos2unix` and tailor it for your purpose in this lab project, you may instead prefer to create your own makefile from scratch, or to download the makefile from one of the sample solutions below.

What should `proj4.asm` do:

1. The program checks the number of command line arguments. It should have exactly 1 (`argc` should be 2). If not, an error message is displayed and the program terminates.
2. The program checks the length of the 1st command line argument (i.e. the length of the string `argv[1]`). If it is not exactly 2, an error message is displayed and the program terminates.
3. Then the first character of `argv[1]` is checked. If it is not a lower case letter, an error message is displayed and the program terminates. We shall refer to this letter as `letter`
4. Then the second character of `argv[1]` is checked. If it is not a digit, an error message is displayed and the program terminates.
5. Then the digit is turned to a number and checked if the number is bigger than 1 and odd. If not, an error message is displayed and the program terminates (hence the only admissible digits are 3, 5, 7, and 9). We shall refer to this number as `size` .
6. Then a subroutine `display_shape` is called. It displays the shape and the program terminates.
7. Here is what the subroutine `display_shape` should do: it expects two parameters on the stack, the `size` which is the size of the shape to be displayed and `letter` which is the letter the shape should be made of (*note that the call to display_shape will thus need a fake parameter*).
8. The shapes displayed by `display_shape` (*the name s mandatory*) depend on size and letter passed to it, examples are shown below:

for size = 3
and letter = A

```
 AA
AAA
```

for size = 5
and letter = O

```
  OOO
 OOOO
OOOOO
```

for size = 7
and letter = Y

```
   YYYY
  YYYYY
 YYYYYY
YYYYYYY
```

for size = 9
and letter = M

```
    MMMMM
   MMMMMM
  MMMMMMM
 MMMMMMMM
MMMMMMMMM
```

9. The subprogram `display_shape` displays the shape by calling a subprogram `display_line` (*the name is mandatory*) for each line of the shape (i.e. `display_line` is called either 2 times, or 3 times, or 4 times, or 5 times).
10. The subprogram `display_line` expects three parameters on the stack: the number of spaces it should print, the number of letters it should print, and the letter it should use (in which order is up to you).

Sample run: `proj4`

```
incorrect number of command line arguments
```

Sample run: `proj4 hello`

```
inccorect length of the argument
```

Sample run: `proj4 a5`

```
inccorect first letter of the argument (should be an upper case letter)
```

Sample run: `proj4 A6`

```
inccorect second letter of the argument (should be 3 or 5 or 7 or 9)
```

Sample run: `proj4 A5`

```
  AAA
 AAAA
AAAAA
```

Sample run: `proj4 Y7`

```
   YYYY
  YYYYY
 YYYYYY
YYYYYYY
```

Your should build your program in stages, so if you run out of time, you can submit the stage you have reached and still earn a significant mark:

- If your program is correct but ends at step 6 and does not have `display_shape` nor `display_line` subroutines and instead of calling `display_shape` it displays the parameters that it would pass to `display_shape` , this will earn you 75% of the total mark.
- If your program is correct and has just a dumb `display_shape` and does not have `display_line`, and the dumb `display_shape` just displays the parameters it was passed, this will earn you 80% of the total mark.
- If your program is correct and has a correct and a fully functional `display_shape` and no `display_line`, and instead of calling `display_line` the subroutine `display_shape` just displays the parameters it would have passed to `display_line`, this will earn you 85% of the total.
- If your program is correct and has a correct and a fully functional `display_shape` and a dumb `display_line` which just displays the parameters it got, this will earn you 90% of the total mark.
- If your program is correct and has a correct and a fully functional `display_shape` and a correct and a fully functional `display_line`, this will earn you 100% of the total.

Hints:

- *The structure of* `display_shape`:
  Let `q = size / 2` *(integer division, i.e.* 1 *for* `size`=3, 2 *for* `size`=5, 3 *for* `size`=7, *and* 4 *for* `size`=9*)* -- *this is the # of spaces for the first line. Let* `p = q - size` -- *this is the # of letters for the first line. Subroutine* `display_shape` *performs a loop until q = 0, decrementing* `q` *by* 1 *and incrementing* `p` *by* 1 *at the bottom of the loop. In the body of the loop it calls* `display_line` *with parameters* `q` *and* `p` *and* `letter` .
- *How to compute* `size / 2`:
  The best is to use the 32 bit portions of the registers and compute `(EDX:EAX)/source` *and finding the result in* `EDX:EAX`
  1. *blank register* `edx`
  2. *move* `size` *to* `eax`
  3. *move to* `ebx` *the value of* 2 (*i.e.* `dword 2`)
  4. *call* `div ebx`
  5. *the result will be found in* `eax` *and hence also in* `rax` *(the remainder will be in edx, but we are not interested in the remainder).*