SE 2XA3 (2019/20, Term I) Final project-- lab section L03

Back To Lab Menu | Back To Main Menu | Submissions | Log Out

For the Final project, you can use any material from anywhere. The only thing that is not allowed is cooperation on the project with other people.

In the Final project, there are is a single deliverable: text file with NASM 64 bit code named `fproj.asm` . It can be submitted either via the course website, or using `2xa3submit`. For `2xa3submit` submission please use `2xa3submit AAA xproj1 fproj.asm` where `AAA` is your student number. You can submit the file as many times as you wish, the latest submission will be used for marking. The submission will be opened from 8:30 01 Nov 2019 till 23:30 06 Dec 2019; after the closing time no submission will be possible. If submission is not possible for whatever reason (typically right after the submission closes), email the file or files as an attachment immediately (needed for the time verification) to Prof. Franek (franek@mcmaster.ca) with an explanation of the problem; you must use your official McMaster email account for that. Include the course code, your lab section, full name, and your student number. Note that if there is no problem with submission (typically the student using a wrong name for the file), you might be assessed a penalty for email submission, depending on the reason you used email.

NASM 64 program named `fproj.asm`

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

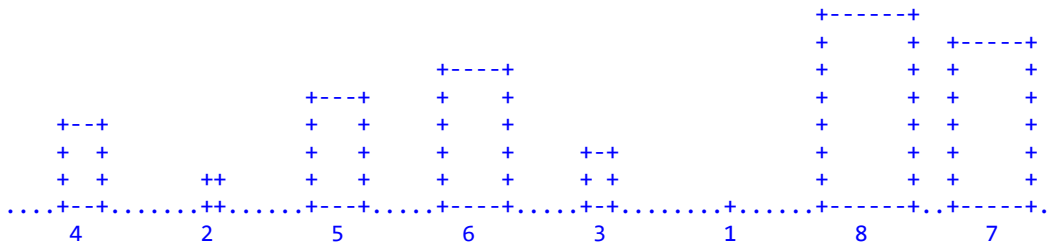
Before you start working on the program `fp.asm`

- Download `simple_io.inc` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file (using `dos2unix`). This file is necessary.
- Download `simple_io.asm` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file (using `dos2unix`). This file is necessary.
- Download `driver.c` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file (using `dos2unix`). This file is necessary. *Note, that this is a different* `driver.c` *than the one we used in the labs.*
- Download `fproj.py` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file (using `dos2unix`). This is a python version of the program you are to write in NASM assembler. You can execute it on ***moore*** by `python3 fproj.py` to gain a better understanding of how the program should behave.
- Download `fproj_skel.asm` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file (using `dos2unix`). This is a skeleton file for `fproj.asm` . It contains to code for the random initialization of the array `array` .
- Download `makefile` from the sample NASM programs and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file (using `dos2unix`). Then modify the file `makefile` for a proper compilation and linking of `fproj.asm` . This file is not necessary, you may create it from scratch if you are so inclined.

What should `fproj.asm` do:

1. It defines an integer array of length 8, its name is up to you, but here we will call it `array` , and the skeleton program `fproj_skel.asm` also uses the name `array` .
2. The array `array` is randomly initialized to the values from 1..8 by the call to `rperm` , see the skeleton program `fproj_skel.asm` .
3. Then in an infinite loop, `array` is displayed on the screen by a subroutine `display` .

4. At the bottom of the screen the user is prompted to either enter a pair of values `i` and `j` from the range 1..8, or 0.
    1. If 0 is entered, the loop is terminated, and then the program terminates.
    2. If a pair of values `i` and `j` is entered, then in `array` the item containing the value `i` is swapped with the item containing the value `j` , and the loop is repeated.
5. The subroutine `display` has two parameters passed on the stack, one is the address of the array `array` , and the other is the size of the array, in this case 8. *You cannot hard code in the subroutine display the address of the array, nor the size.*
6. The subroutine `display` traverses the array and displays the value in each item of the array in a form of a box, for better understanding see the execution of `fproj.py` . It also must display the numeric value of the box underneath.
7. The box of size 1 looks like this
```
+
```
8. The box of size 2 looks like this
```
++
++
```
9. The box of size 3 looks like this
```
+-+
+ +
+-+
```
10. The box of size 4 looks like this
```
+--+
+  +
+  +
+--+
```
11. The box of size 5 looks like this
```
+---+
+   +
+   +
+   +
+---+
```
12. etc. ....
13. The box of size 8 looks like this
```
+-------+
+       +
+       +
+       +
+       +
+       +
+       +
+-------+
```
14. The boxes are displayed adjusted to the same level at the bottom, side by side. Thus, the subroutine `display` must display it from top to the bottom, one line at a time. So, `display` must loop `n`-times, where `n` is the length of the input array. In this loop, it must prepare a line, and then displays it -- i.e. `n` lines from top to bottom. For each line, it must figure out what to put in the line for each item of the array, i.e. must loop `n`-times for each item of the array. For instance:



The first line will be (~signifies a space):

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+------+
```

The second line will be

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+~~~~~~+~~+------+
```

etc.

15. Your subroutine `display` may call other subroutines, similarly as done in the python version of the program, `fproj.py`