B: SE 2XA3 (2018/19, Term I) Major Lab 5 -- lab section L01

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

sample solutions: `proj5_1.asm` `proj5_2.asm` `bonus.txt`

If you are in the correct lab room and during the SCHEDULED time slot for your lab section for Major Lab 4 and working on one of the lab's workstations, go into Marks+Comments to see if your attendance was registered. If you signed on the course website too early (the most common problem), your attendance would not be recorded; in such case, please log out and sign on again. If your attendance is still not recorded, please, ask your TA to record your attendance and email Prof. Franek right away. If you are not using one of the lab's workstations, please, have the TA record your attendance.

The Major Labs are open book labs, you can bring and use any notes etc. You can access any website, Google, Wikipedia etc. The only thing that is not allowed is cooperation with other people, inside or outside the lab. You must submit your own work. The TA can only help with administrative and technical aspects and not with the solutions of the problems of the Major Lab.

In this lab, there are two tasks and two deliverables: NASM programs `proj5_1.asm` and `proj5_2.asm` , and a text file `bonus.txt` for the bonus if you decide to do the bonus question -- note that the bonus is not mandatory and the bonus mark of 3% will be applied to the overall grade. Each file can be submitted either via the course website, or using `2xa3submit`. For `2xa3submit` submission please use `2xa3submit AAA proj5 BBB` where `AAA` is your student number and `BBB` the name of the file you want to submit. You can submit every file as many times as you wish, the latest submission will be used for marking. The submission is opened from 8:30-11:20 or 14:30-17:20 depending on the lab section; after the closing time no submission is 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.

Task 1. NASM program named `proj5_1.asm`

Do this one first, `proj5_2.asm` is an extension of `proj5_1.asm`

The name of your file must be `proj5_1.asm` and below is a description of what it should do when executed.

Before you start working on the program `proj5_1.asm`:

- Decide what will be your working directory.
- Download a text file asm_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`).
- Download a text file asm_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.
- Download a text file cdecl.h and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file.
- Download a C++ program 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.
- Download a bash script makefile and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file.

What should `proj5_1.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 digit, an error message is displayed and the program terminates.
4. Then the digit is turned to a number and the number is checked if it 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.
5. Then the second character of `argv[1]` is checked. If it is not an upper case letter (i.e. `'A'` .. `'Z'`), an error message is displayed and the program terminates. We shall refer to this letter as letter.
6. The program displays a massage of displaying a shape of the given size and made of the given letter.

Hints:

1. *To check that `argv[1]` is of length exactly two, just check that the first character of `argv[1]` is not a `NULL` character and the second character of `argv[1]` is not a `NULL` character and the third character of `argv[1]` is a `NULL` character.*
2. *To check if a byte is an upper case letter, use two separate `cmp` statements to see if it is below `'A'` or above `'Z'` . If it is not, then it is an upper case letter.*
3. *To check if a byte is a digit, use two separate `cmp` statements to see if it is below `'0'` or above `'9'` . If it is not, then it is a digit.*
4. *To turn a digit into a number, load it to a register and subtract `dword '0'`.*
5. *To determine if a number is even or odd requires division by 2 and checking if the remainder is 0 (then it is even) or 1 (then it is odd), see `div` instruction for bytes. The other option is to check whether the number equals 3 or 5 or 7 or 9 by four separate `cmp` statements.*

A sample run `proj5_1 3A` will output `Displaying shape of size 3 made of letters A`
A sample run `proj5_1 5U` will output `Displaying shape of size 5 made of letters U`

Task 2. NASM program named `proj5_2.asm`

`proj5_2.asm` is an extension of `proj5_1.asm`

The name of your file must be `proj5_2.asm` and below is a description of what it should do when executed.

1. Copy `proj5_1.asm` to `proj5_2.asm` (make sure not to destroy `proj5_1.asm`).
2. The program `proj5_2.asm` instead of displaying the final message (`Displaying shape of size ...`) calls a subprogram `display_shape` -- *the name of the subroutine is mandatory*.
3. The subroutine `display_shape` 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.
4. The shapes displayed by `display_shape` depend on `size` and `letter` passed to it, examples are shown below:

for `size` = 3 and `letter` = a

```
  a
  a
a a a
```

for `size` = 5 and `letter` = o

```
  o
  o
  o
  o
o o o o o
```

for `size` = 7 and `letter` = x

```
    x
    x
    x
    x
    x
    x
x x x x x x x
```

for `size` = 9 and `letter` = w

```
    w
    w
    w
    w
    w
    w
    w
    w
w w w w w w w w w
```

5. The subprogram `display_shape` displays the shape by calling a subprogram `display_line` (*again, the name is mandatory*) for each line of the shape (i.e. `display_line` is called either 3 times, or 5 times, or 7 times, or 9 times).

6. The subprogram `display_line` expects three parameters on the stack, first is the number of spaces it should print, the second is the number of letters it should print, and the third is the letter it should use. *Note the space between two consecutive letters!*

Hint: Subroutine `display_shape` *performs a loop of length* `size` *; at the beginning of the loop the number of spaces is set to* `size-1` *and the number of letters is set to* `1` *. In each pass through the loop, the number of spaces and the number of letters remains the same, except for the last pass when the number of spaces is set to* `0` *and the number of letters is set to* `size` *. In each pass through the loop, the subroutine* `display_line` *is called with three parameters: number of spaces, number of letters, and* `letter` *.*

Bonus task.

Download a text file bonus.asm and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file. Compile the program by `make bonus` . Execute the program `bonus`

The output is

```
You have in your account 250 dollars
The Swindler's account has 0 dollars
Amount in your account after doubling is 500
Amount in Swindler's account after doubling is 0
Let's check both accounts after the doubling of all accounts is done
You have in your account 0 dollars
The Swindler's account has 0 dollars


How is this possible ? Does he work in IT department of the bank?
```

Describe why the program removes all of your money when the program never touches your account after the amount in it was correctly doubled, and describe how to fix it in a text file `bonus.txt` that you submit in the usual way.