Exercise 1

x86 Assembly

Welcome to the first exercise! To be able to run this exercise, and all following exercises, you will need to follow both these documents:

- Setup instructions - Installing the VM and registering on the course website. You will need this to run all exercises.
- Exercise Submission Guidelines - General instructions and course policies that apply to all exercises in the course. You **must** read this document before submitting any exercise.

Ready? Awesome, let's actually begin :)

Log into your VM (`user` / `1234`), open a terminal and type in `infosec pull 1`.

- When prompted, enter your **course website credentials**
  - I.e. the username and password you used to register to the course website (these are unrelated to your university user)
- Once the command completes, your exercise should be ready at `/home/user/1/`

When you finish solving the assignment, submit your exercise with `infosec push 1`.

- This will run some sanity tests to make sure your submissions seems to be OK (it is not a full test of the homework)
- It will submit the homework even if the tests fail
- The last submission is what that matters
  - You can see your submitted files on the course website

Question 1 (30 pt)

In the `q1.c` file, write an x86 Assembly program that receives an integer in `EBX`, computes its **exact square root**, and stores the result in `EAX`; if the integer is less than `1`, or if there is no exact (integer) root, the result should be `0`.

Add your assembly instructions as strings to `q1.c` between our comments, like so:

```
19    asm (
20        /* Your code starts here. */
21
22        "MOV   EAX, 1;"
23        "MOV   EBX, 2;"
24        "ADD   EAX, EBX;"
25
26        /* Your code stops here. */
27    );
```

- To compile your program, run[1] `gcc q1.c -fno-pic -masm=intel -o q1`.
- If compilation succeeds without errors, it will create a program named `q1` within the same directory
- Test your code:
  - To run it, just run `./q1 <number>`
  - For example, `./q1 16` should print `4`, and `./q1 6` should print `0`

Question 2

Part A (30 pt)

In the **`q2a.c`** file, write an x86 Assembly program that receives an integer in `EBX`, **computes its Squarebonacci number**[2] **<u>using recursion</u>**, and stores it in `EAX`; if the integer is less than 0, the result should be 0.

In a similar fashion to Fibonacci numbers, **Square**bonacci numbers are the numbers of the sequence 0, 1, 1, 2, 5, 29, ... defined as:

$$a_0 = 0\,, \qquad a_1 = 1\,, \qquad a_n = (a_{n-1})^2 + (a_{n-2})^2$$

Add your assembly instructions as strings as in question 1, and compile and test in a similar way.

Important Note: Due to automatic testing constraints, please **<u>avoid</u>** using tail-recursion, and make sure each parent call invokes at least 2 recursive calls. You can read more about it in this Wikipedia explanation.

---

[1] `gcc` = the compiler, `q1.c` is our input file, `-fno-pic` means we compile a position dependent program (otherwise `EBX` will be reserved), `-masm=intel` means we use the intel x86 syntax, `-o q1` means to write the result as `q1`.

[2] Given the number `n` in `EBX`, compute a_n

Part B (20 pt)
In the `q2b.c` file, as before, write an x86 Assembly program to compute a Squarebonacci number**, this time without recursion**.

Question 3 (20 pt)

Read the following x86 Assembly program, and describe what it does in `q3.txt`.

```
 1        XOR    EDX, EDX
 2   _LABEL1:
 3        CMP    [EDI], DL
 4        JZ     _LABEL2
 5        INC    EDI
 6        JMP    _LABEL1
 7
 8   _LABEL2:
 9        MOV    AL, [ESI]
10        MOV    [EDI], AL
11        INC    ESI
12        INC    EDI
13        CMP    AL, DL
14        JNZ    _LABEL2
15
16   _END:
```

Please note that the program receives input via 2 registers: `EDI` and `ESI`.

Note: Telling us what every line does, is NOT a valid answer. We want **the key idea of what this code does**, not a translation from Assembly to English.

Final notes:

- Consider edge cases (i.e. negative numbers, etc.)
- **Document your code**
    - You can use C comments to add documentation between the strings of the Assembly
- Don't use any additional third party libraries that aren't already installed on your machine (i.e. don't install anything)
- If your answer takes an entire page, you probably misunderstood the question.