Homework 2

Due: May 10, 2024 (Note extension) **Points:** 100

All programs are to be submitted through Gradescope. Gradescope will compile and execute your program, sometimes with varying inputs or invocations. You can see your score once executions are done. You can submit your program multiple times, up to the deadline — at that point, the score you have will *usually* be the grade for that problem. I say "usually" because we reserve the right to look at your submission and deduct points if you do not use proper programming style.

You have to name your program as given in the problem. Otherwise you will get an odd error message indicating there is a problem with Gradescope.

Also, your output must match Gradescope's *exactly*, including blanks and tabs — so pay attention to the example output!

1. (*15 points*) Write a program that prints a tic-tac-toe board with an "X" and an "O" in two squares. Each square is to be 7 × 7, with "#" marks. The "X" is to be 5 × 5, centered in the upper left square and composed of "X"s, and the "O" is to be 5 × 5, centered in the middle square, and composed of "O"s. Your output will look like this:

```
       #       #
 X   X #       #
  X X  #       #
   X   #       #
  X X  #       #
 X   X #       #
       #       #
#######################
       #       #
       # OOOOO #
       # O   O #
       # O   O #
       # O   O #
       # OOOOO #
       #       #
#######################
       #       #
       #       #
       #       #
       #       #
       #       #
       #       #
       #       #
```

   When this is done, exit with the status code 0.

   Call your program "ttt2.c".

2. (*30 points*) Write a program that reads lines from the standard input, and prints the words in that line. Each word is to be on its own line, prefixed by the line number (in 3 spaces), a period, and a space. You may assume no line will be longer than 99 characters including the newline (we'll remove that bound later). If there is no input, print nothing.

   For this problem, define a *word* as a maximal sequence of alphanumeric characters, an underscore "_", and a single quote (apostrophe) between 2 letters. For example, "hello24" and "it's" are words, but "++" and "..." are not. Note that in "--xyzzy--", "xyzzy" is a word.

   Here is sample input, in red:

```
Hello, there, my old friend!
How are you today?
I am very well, thank you!
Goodbye ...
```

The corresponding output is:

```
1. Hello
1. there
1. my
1. old
1. friend
2. How
2. are
2. you
2. today
3. I
3. am
3. very
3. well
3. thank
3. you
4. Goodbye
```

Call your program "wordsplit.c".

Hint: There are several ways to do this problem; the simplest is probably to use `fgets` to read the input line.

3. (*30 points*) The Fibonacci numbers play an important role in biology, mathematics, and other sciences. The first two numbers of the sequence are 1 and 1, and the numbers of the sequence are formed by adding the two previous numbers; so, the first few terms of the sequence are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... Please write a function, called `fib`, that takes a single integer *n* as its argument, and prints the first *n* numbers of the sequence. Your function must have the following interface:

```
int fib(int n)
{
    /* your code goes here */
}
```

Your function is to print the numbers on a single line with one blank separating numbers. Do not have any leading or trailing white spaces, and terminate the list of numbers with a newline. If $n = 0$, there is to be no output (as the function is to print 0 Fibonacci numbers). If *n* is negative, print the message "Cannot print a negative number of Fibonacci numbers" followed by a newline on the standard *error* (not the standard output).

If *n* is negative, `fib` is to return 0. Otherwise, it is to return 1. You may assume the argument *n* is an integer, so you do not need to check the argument for being a non-integer. You do need to check that the integer is valid, i.e., non-negative.

The function you write must be stored in a file called "fib.c".

Hint: There is a file that provides an interface (a `main` routine) available on Canvas; it is called "*fibdriver.c*". If you download it and compile it with your function (see **Compiling and Executing Your Program** for details on how to do this), you can focus on writing the function. This file compiles correctly on the CSIF (and Gradescope); no guarantees on any other system, though!

As an example, the output for the input 12 is:

```
1 1 2 3 5 8 13 21 34 55 89 144
```

4. (*25 points*) The program *calc.c*, which implements a simple 5-function calculator, has a bug: a spurious “invalid operator” message is printed after each computation. Find out why and fix it.

   Call your fixed program *calc.c*.