CSE 3100 Systems Programming	Spring 2024
Lab #2	

Welcome to Lab 2! Read each question carefully before beginning work and submit your work to gradescope.

Part 1. make. We are going to learn another important tool, `make`. Your lab TA will provide an introduction to the tool and Makefile at the beginning of the lab. The full documentation for GNU Make is available here.

An example of `Makefile` is provided. With this file in your current working directory, running the command `make` compiles `ex-factorial.c` to generate `ex-factorial` if the executable does not exist. If both `ex-factorial.c` and `ex-factorial` exist, running `make` will re-compile `ex-factorial.c` if it has a more recent modification timestamp than `ex-factorial`.

The `Makefile` also has a `clean` rule that removes all object files and the `ex-factorial` executable – this can be used to quickly clean up the assignment folder. To do this, run the command `make clean`.

Change the existing rules and/or add new rules in the provided `Makefile` to support the building of `catalan` in a similar way. The default rule `all` should update both `ex-factorial` and `catalan` executables if necessary. The `clean` rule needs to be revised to remove both executables.

Part 2. factorial. `ex-factorial` reads an integer from stdin and computes its factorial. Find and fix all syntax errors and bugs in the program. This problem is an opportunity for you to practice GDB. Do not rewrite the program.

It is good practice to always check user input. Improve the program to make it robust. The program calls `factorial()` only if a non-negative integer is entered. Otherwise, the program terminates after printing `"Error: Please enter a non-negative integer.\n"`.

There is no need to deal with overflow if the integer is too large.

Part 3. Catalan numbers. The sequence of the Catalan numbers C_n, where $n \geq 0$, can be defined using the following recurrence:

$$\begin{aligned} C_0 &= 1 \\ C_k &= \frac{4k-2}{k+1} C_{k-1} \quad \text{if} \quad k \geq 1 \end{aligned}$$

The goal of this problem is to complete the `catalan_number()` function in `catalan.c`, which computes a Catalan number using the above recurrence. The constraints are as follows:

- The function uses the above recurrence for computing Catalan numbers.
- The function does not use floating-point numbers or operations.
- The function does not print anything.
- The `main()` function should not be changed.

On the next page is a sample session of running the program.

```
$ ./catalan
0
C( 0)=                1
1
C( 1)=                1
2
C( 2)=                2
3
C( 3)=                5
4
C( 4)=               14
5
C( 5)=               42
10
C(10)=            16796
14
C(14)=          2674440
15
C(15)=          9694845
20
C(20)=       6564120420
30
C(30)=  3814986502092304
32
C(32)= 55534064877048198
33
C(33)=212336130412243110
```