# Homework 6 (due 3/16)

DS-210 @ Boston University

Spring 2022

## Before you start...

**Collaboration policy:** You may verbally collaborate on required homework problems. However, you must write your solutions independently without showing them to other students. If you choose to collaborate on a problem, you are allowed to discuss it with at most 2 other students currently enrolled in the class.

The header of each assignment you submit must include the field "Collaborators:" with the names of the students with whom you have had discussions concerning your solutions. If you didn't collaborate with anyone, write "Collaborators: none." A failure to list collaborators may result in a credit deduction.

You may use external resources such as software documentation, textbooks, lecture notes, and videos to supplement your general understanding of the course topics. You may use references such as books and online resources for well known facts. However, you must always cite the source.

You may **not** look up answers to a homework assignment in the published literature or on the web. You may **not** share written work with anyone else.

**Submitting:** Solutions should be submitted via Gradescope. The entry code is 3Y85PZ.

**Grading:** Whenever we ask for a solution, you may receive partial credit if your solution is not sufficiently efficient or close to optimal. For instance, if we ask you to solve a specific problem that has a polynomial–time algorithm that is easy to implement, but the solution you provide is exponentially slower, you are likely to receive partial credit.

**Late submission policy:** No extensions, except for extraordinary circumstances. We accept submissions submitted up to one day late, but we may deduct 10% of points.

## Questions

To solve problems in this homework, you should use Rust. Your solution to the homework should consist of

- three compilable Rust source files (`.rs`) solving Questions 1–3,
- a report (as a pdf or in another document format accepted by Gradescope) that answers sub-questions denoted by "**Report**" below.

1. **(14 points)** The Fibonacci numbers, $F_k$ for $k \in \mathbb{N}$, are defined as

$$F_k = \begin{cases} 0 & \text{if } k = 0, \\ 1 & \text{if } k = 1, \\ F_{k-2} + F_{k-1} & \text{otherwise.} \end{cases}$$

   (a) Write a function `fib` that takes a parameter $k$ of type `u32` and returns an integer of type `u128` equal to $F_k$. The computation of the function should exactly follow the definition of Fibonacci numbers, i.e., to compute the value of the function for $k \geq 2$, it should recursively call `fib` twice.

   Do not introduce any optimizations such as storing the numbers that have already been computed. For large $k$, the solution will not fit into `u128`, but ignore this issue.

   (b) Your function `main` should iterate from $k = 0$ to $k = 100$ and for each $k$, output the following:
   - $k$,
   - $F_k$ computed using `fib`,
   - the time it took to compute $F_k$ (see a suggestion how this can be done below).

   (c) Run your program twice: once via `cargo run` and once via `cargo run --release`. Since it will take a lot of time for it to finish, interrupt it after a while. (Pressing Ctrl+C may work for this purpose.)

   **Report:** What are the computation times for different $k$ for each of the options? How do the times to compute Fibonacci numbers compare for large $k$ between these two options? Are they roughly the same or are they very different? If they are different, what is the multiplicative difference?

   *Note:* To find the amount if time it takes to perform some computation, you can first place

```
use std::time::SystemTime;
```

   at the beginning of your source file and then place the following around your computation.

```
let before = SystemTime::now();
// YOUR COMPUTATION HERE
let after = SystemTime::now();
let difference = after.duration_since(before);
let difference = difference.expect("Did the clock go back?");
println!("Time it took: {:?}", difference);
```

   See https://doc.rust-lang.org/std/time/struct.SystemTime.html for more details.

2. **(13 points)**

   (a) Create an array `F` of length 101 in which each entry is of type `u128`. Use the `for` loop to make `F[i]` equal $F_i$, starting from $i = 0$ to $i = 100$. Then output all the computed numbers $F_i$. Your solution should use only a constant number of arithmetic operations to compute a given `F[i]` once the previous entries have been computed. This version of your code should be the solution you return.

(b) **Report:** Now conduct the following experiment. Replace the array entry type with `u8` and adjust any other types accordingly so your program still compiles. Try running the modified code with both `cargo run` and `cargo run --release`. Are there any differences in the behavior of the program? If so, what are they?

3. **(13 points)** Write a program that reads a non-negative integer—let us denote it $k$—and computes $\sum_{i=1}^{k} i^2$. Your solution should use the `for` loop to compute it directly as defined (even though there is a smarter way to do this). Assume that the input can be represented, using `u32`. Under this assumption, make sure that the output is computed correctly, i.e., that throughout the computation you do not use an integral type of range too small to represent your intermediate and final numbers.

   **Report:** Explain why the situation described above is not happening, i.e., why the range of integers you use is sufficiently large. This kind of problem is known as *integer overflow*, i.e., you want to explain why integer overflow is not a problem in your code.

   *Note:* To read a line and convert it to an integer, you can do the following. First, put this statement at the beginning of your file:

   ```
   use std::io;
   ```

   Then use the following when you want to read an integer:

   ```
   let mut input = String::new();
   io::stdin().read_line(&input).expect("Failed to read line");
   let input = input.trim();
   let number: u32 = input.parse().expect("Not a good number!");
   ```

4. **(Optional, no credit)**

   **Report:** How much time did you spend on this homework? The answer will have no impact on the credit you receive, but it may help us adjust the difficulty of future homework assignments.