First assignment (2022/2023, this is the one you need to do, due 05-12-2022)
Speed run (part 1)

A road is subdivided into road segments of approximately the same length. Each road segment has a speed limit. The speed is measured by the number of road segments a car is able to advance in a single "move." In each move the car can i) reduce its speed by one (brake), ii) maintain the speed (cruise), or iii) increase its speed by one (accelerate). The car is placed at the first segment of the road with a speed of zero. It has to reach the last segment of the road with a speed of one (at which point it can reduce the speed to zero and so, stop). The purpose of this assignment is to determine the **minimum** number of moves required to reach the final position.

The car position, i.e., the road segment number, is stored in the integer variable `position`. Its final position is stored in `final_position`, and its speed in `speed`. A car move consists in doing the following:

1. Choose its `new_speed`. It can be `speed-1`, `speed`, or `speed+1`. The new speed must be positive.
2. Advance to the new position: `new_position=position+new_speed`.
3. **However**, it can only do so if it never exceeds the speed limit, i.e., for `i=0,1,...,new_speed`, it must be true that `new_speed<=max_road_speed[position+i]`.

In the following example the road has 31 segments (i.e. `final_position=30`).

5	5	6	6	8	7	8	9	8	9	8	9	9	8	9	7	7	6	6	6	5	6	5	4	4	3	5	3	3	3	3
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]

The optimal solution has 10 moves (the 11 car positions are shown in gray):

5	5	6	6	8	7	8	9	8	9	8	9	9	8	9	7	7	6	6	6	5	6	5	4	4	3	5	3	3	3	3
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]

Speed run (part 2, what is given)

In the archive `A01.tgz` you will find the files:

- `A01/makefile` — this makefile can be used to compile the program.
- `P02/elapsed_time.h` — to declare and define the `cpu_time` function that is used to measure the time the program takes to run.
- `A01/make_custom_pdf.c` — code to produce directly a custom PDF file. (This is your chance to study how a PDF file is structured.) It is possible to produce uncompressed or compressed PDF files; just take a look at the `makefile` to see how this can be done. Uncompressed PDF files are text files! Warning: on `ubuntu`, you may need to install the development version of the `zlib` library:

  ```
  sudo apt install zlib1g-dev
  ```

- `A01/speed_run.c` — the main code, where the `max_road_speed` array is initizalized, and where a correct but slow solution of the problem is provided (look at the functions `solution_1_recursion` and `solve_1`).

You should study the entire contents of the `speed_run.c` file. In particular, you **must** study and understand how the `solution_1_recursion` recursive function works, because improving it will be your main task.

The `speed_run` can take one command line argument. If it is `-ex` it generates and solves the example given in the previous page. Otherwise, the argument should be a student number. It will be used to initialize the pseudo-random number generator used by the program to produce perturbations of the maximum speed array data. In this way, each student will have to solve a slighty different problem.

Speed run (part 3, what has to be done)

Proposed tasks:

- See how large the final position can be using a `_time_limit_` of 3600.0 (one hour). This has to be done using each one of the student numbers of the group as a (command line) input to the program. Record the execution times as a function of the final position. Try to find a formula that gives a reasonably good estimate of the execution time (as a function of the final position). For a final position of 800, estimate how long would the program take to give an answer.

- As it would be great to be able to reach a `final_position` of 800, which is not possible using the given solution, **invent** other solution methods. Do not change the `solution_1_recursion` and `solve_1` functions. Instead, create new ones. [Hint: it is possible to solve the problem with a final position of 800 in a few microseconds.] Which of your methods is the fastest?
  Consider solving the problem using dynamic programming (that will be explained later in this course). That is not mandatory, but should be attempted by groups with very good students (or so they believe themselves to be). A warning, though. The teachers will give no support regarding using dynamic programming at this early stage. You will be on your own. Impress us!

Speed run (part 4, the written report)

The written report must have:

- A title page (front page) with the name of the course, the name of the report, date, the numbers and names of the students, and an estimate of the percentage each student contributed to the project. The sum of percentages should be 100%.
- A short introduction describing the problem.
- A **small** description of the methods used to find the solutions. This must also include a description of the solution already provided. Explanations why a given solution is better (or worse) than some others should also be given.
- A description of the solutions found; this should include graphs of the execution time of the program as a function of the size of the problem. Place in the report only a few of the PDF files generated by the program.
- Comments or attempts at explanations of the results found. This can be placed near where the results are presented in the report.
- The source of any material adapted from the internet must be properly cited.
- An appendix with all the code. Use a small mono-spaced font such as `courier` or `consolas`.
- **Deliverable (via elearning site): one PDF file. No archives!**