Assignment 1

Introduction to Operating Systems (COMP 2004)

Fall 2025

Important notes:

1. **Due date:** Sep 26.
2. You need to submit one `.c` file for each one of the problems. Each file must include a `main()`.
3. You can use onlinegdb C Compiler and Debugger to develop your programs (to submit, copy paste your code from the online debugger into a text editor and save it as `.c`). If you wish to use other development environments, make sure that your code also runs correctly on onlinegdb.

Problem 1 (20 pts). Create a data type for *complex numbers* using `C struct` and `typedef` (check out this page for further information on `typedef`). Define two functions that implement the addition of two complex numbers with the newly defined data type: the first function performs the task *by value*, and the second function performs the task *by reference*. Test your functions in `main()`.

Problem 2 (30 pts). Write a function that computes the *distance matrix* of 5 points based on the *Euclidean* distance measure. The function inputs two arrays `x` and `y` of size 5 consisting of floating point numbers, where `(x[0],y[0])` are the coordinates of the first point, `(x[1],y[1])` are the coordinates of the second point, and so on. The function then outputs the distance matrix; that is, a two-dimensional array where each element indicates the distance between the corresponding pairs. For example, the value at the intersection of row 3 and column 4 indicates the distance between the fourth point and the fifth point. Test you program with `x=[2.0,3.0,3.0,1.0,-1.0]` and `y=[2.0,1.0,-3.0,4.0,-4.0]` in the `main()`.

Problem 3 (20 pts). Write a program that continuously inputs a string of characters until 'q' is entered, and then outputs the number of occurrences of the character 'a' in the input string. Your program should store the number of

occurrences in a `static` variable.

Problem 4 (30 pts). For this problem, you need to submit two files: a `.py` file and a `.c` file. We want to compare the speed of operations in *C arrays* with the speed of operations in *Python's lists*. Create two Python's lists each initialized with 10,000 ones. Perform an element-wise addition on the lists and store the result in a third list (which should contain 10,000 twos after the operation). Measure the *execution time* of the operation using Python's `timeit`. Do the same thing this tome with arrays in C. You will need to use `time.h` to measure execution time. Check out this thread for instructions on how to do this. Note that both files must print the execution time in the same unit of time (not a part of the assignment: if you are curious, do a bit of research to explain the result).