THE UNIVERSITY OF CHICAGO

MASTERS PROGRAM IN COMPUTER SCIENCE

2013/14 Programming Placement Exam

Instructions

Exam Format

In this programming exam, you will have 90 minutes to write programs that solve a series of simple problems. You will have access to a single computer. During the exam, network access (both to the Internet and to other machines in the room) will be disabled. This exam is meant to be done entirely on a computer, and you are expected to write code that compiles and runs correctly.

During the exam, you will submit your solutions using a tool called PC^2 (you will be provided with instructions for PC^2 during the exam). This tool will run your solution with a series of test cases, and will tell you whether your solution passed the test cases or not. Although the results of the test cases will be a factor in evaluating your solution, your code will also be stored in a database for further evaluation by a human grader.

Problem Style

All the exercises in the exam require you to write a program that will read its input from a text file in a specific format, which you must then process in a way specified in the problem statement, and produce an output in a specific format. You must write your solution in a single file (Java programmers, note that you can include additional top-level classes in a single source file as long as they are unqualified: just "class", without "public").

The format of the input and output is described in each exercise, and ***you must follow them to the letter***. Each problem includes some sample input/output data that you can use to test your solution. Take into account that we will test your solution with larger test cases.

Each problem statement also specifies the name of the file you should read from. Do not use drive and/or path specifications when naming input files. If a problem indicates that the input file is named **`sample.in`**, then you must open **`sample.in`** and not **`/home/cspp/sample.in`** or **`../../sample.in`** or anything else. Your program should read the specified file, process it, and print the correct output without any human intervention. *Do not write a program that expects a human operator to enter the data through the keyboard*.

All output should be printed to standard output. Anything printed to standard error will be ignored (i.e., you can use standard error to print debugging statements).

PC^2 "Judgments"
When you submit a solution, the PC^2 system runs your solutions with a series of test cases (more exhaustive than the sample data provided in the problem statement), and will produce a "judgment" (you will get a notification of this judgment from the PC^2 system)

The possible judgments are:

- **Yes**. The solution you submitted passes our extended test cases.
- **No – Wrong Answer**. The solution you submitted ran to completion but did not pass our extended test cases.
- **No – Presentation Error**. The solution you submitted produced output that essentially matches that of a correct solution, but your output is not correctly formatted.
- **No – Time Limit Exceeded**. Your solution ran for more than 60 seconds. This is usually a sign that your solution has fallen into an infinite loop somewhere, or waiting for input from the keyboard. Remember: Your solution should read data from a file *not from standard input*.
- **No – Runtime Error**. Your solution crashed when we ran it (e.g., by producing a segfault, or a runtime exception in Python)
- **No – Submission Error**. You submitted your solution to the wrong problem, or to the wrong programming language.
- **No – Compilation Error**. We were unable to compile your program.

Please note that a "No – Wrong Answer" ***does not*** mean you get a "zero" on that problem. Your solution may still be mostly correct, but failing on some corner case (in fact, a "No – Wrong Answer" submission could still get close to full credit). So, if your solution passes the sample test cases but you still get a "No – Wrong Answer", you should not waste too much time trying to get a "Yes". Move on to another problem, and revisit the "wrong" problems if you have time left over at the end.

Programming Environment
All the machines are GNU/Linux Ubuntu 12.04 machines, and include the following compilers and interpreters:

- Java 1.6.0_27
- GCC 4.6.3
- Python 2.7.3
- Ruby 1.8.7
- Perl 5.14.2
- Mono C# compiler 2.10.8
- PHP 5.3.10

The following programming tools are also provided:

- Eclipse 3.7.2 (with C/C++ Development Tools)
- GNU Make 3.81
- GDB 7.4
- GNU Emacs 23.3.1
- vim 7.3

Since network access will be disabled, you will be provided with local copies of the Java API documentation, the C++ STL documentation, and the Python documentation.

Preparing for the exam

The goal of this exam is not to test your ability to read from a file and print to standard output. We want you to focus on how that data is processed to produce the expected output. We *strongly* encourage you to look at the provided sample problems, and to code up a solution for them to get accustomed to reading data from a file and printing output in a very specific format.

Take into account that *all* the problems in the exam will have similar input specifications: files that include some number of datasets, with individual values in a dataset separated by a single space or a newline, and the file terminated by an explicit end-of-file marker. You can assume that all input data is correctly formatted, so there is no need to do error checking or validation.

Furthermore, it is useful to think of the input as a stream of tokens (with each value, or *token*, separated by a space or a newline). All modern programming languages include libraries to easily read in this kind of data, without having to read in the file byte by byte or doing any special parsing. In particular, you may want to look at:

- C: `fscanf()`
- C++: The `iostream` library, including the << and >> operators.
- Python: `*file*.read().split()`
- Java: `StreamTokenizer`

Make sure you familiarize yourself with this functionality in your programming language of choice *before* the exam.

Finally, the room where the exam will take place will be open prior to the start of the exam. During this time, you will be able to familiarize yourself with the environment, and with the PC^2 tool. We encourage you to arrive early so you can hit the ground running when the exam starts.

During the exam
The exam will include 3-5 exercises. The difficulty of the exercises will be similar to the sample exam we have provided. We need to see evidence that you know how to write code that compiles and runs correctly: Three partial solutions (as long as they compile and run) are actually much better to us than a single solution (even if it is 100% correct). The more code we see from you, the better.

We suggest you follow this strategy during the exam:

1. Select an exercise that you feel you can solve easily.

2. Write a solution. Double-check that your solution reads its input from the filename specified in the exam.

3. Test the solution with the sample input provided. Check whether it produced the **exact** same output shown on the handout. If it does not, your solution is incorrect.

4. If your output matches the output shown in the handout, submit it with PC^2. If you get a "No - Wrong Answer" judgement, it's likely that your solution is substantially correct, but is failing on a corner case.

   Don't obsess over this! If your solution (1) compiles, (2) runs, and (3) solves the problem correctly for the sample input/output shown in the handout, then your solution is very likely good enough. Move on to the next exercise, and revisit any "No - Wrong Answer" problems if you have time at the end.