CSE-381: Systems 2

Exercise #15

Max Points: 20

You should save/rename this document using the naming convention MUid.docx (example: `raodm.docx`).

Objective: The objective of this exercise is to:

1. Review the higher-level "professional success" skills practiced in this course
2. Measure some of the computational efficiency aspects covered in this course

Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from the terminal/output window into this document. You may discuss the questions with your instructor.

Name: **Noah Dunn**

Part #1: Computational efficiency & implications

Estimated time: 35 minutes

Background: Technology landscape changes often with another "next new/big thing" being advertised every day. People rave about Just-In-Time (JIT) compiled languages like Java, Scala, Julia, Php, Javascript etc. and interpreted languages like Python, Perl, Ruby. However, core systems and fully-compiled languages like C++ (though, C++ also has JIT and interpreted versions now. For example, see https://hub.gke.mybinder.org/user/quantstack-xeus-cling-yqim32vc/notebooks/notebooks/xcpp.ipynb) have stayed around for over 30 years, of course its perceived popularity waxing and waning. There are a few key aspects of C++ that are often cited for its success, namely:

- It aims to keep simple things, simple (why create classes etc., if we don't really need them)
- You don't pay for what you don't use (this includes avoiding bounds checking etc. if you don't want those overheads)
- Trusting and giving full control to the programmer as the programmer is deemed the expert. Often people end up misunderstanding this aspect and shot themselves in the foot.
- Being a programming language and not a library.

The aforementioned aspects of C++ enable it to provide efficient solutions. In this part of the exercise we will explore aspects of runtime efficiency.

Benchmark: For this part of the exercise, we will use a simple benchmark program that involves string manipulation on a ~287 MB string. Note that, in this era of "big data", 278 MB is actually small because data is often hundreds of gigabytes in size. Our benchmark will be a simple method called `toggleCase`, shown in the adjacent code snippet. This method toggles the case (*i.e.*, lowercase becomes uppercase and vice versa) of a given character `low` (all of its

```
void toggleCase(std::string& str,
                const char low) {
  char up = std::toupper(low);
  for (char& c : str) {
    if (c == low) {
      c = up;
    } else if (c == up) {
      c = low;
    }
  }
}
```

occurrences). For example, calling `toggleCase("abAcda", 'a')` returns `"AbacdA"` (notice how the case of the letter `'a'` has been toggled).

In this experiment, we will measure how much time and memory it takes to toggle the case of a given set of characters by calling the `toggleCase` method several times. This benchmark has been chosen because –

1. This type of string or array processing is very common in many programs.
2. Such string/array operation is common in Natural Language Processing (NLP), data sciences, and bioinformatics, just to name a few examples.
3. It is simple to understand and has some similar solutions in all programming languages.
4. Most importantly, if simple things are not efficient, building complex concepts upon them will not alleviate the situation.

Experimental procedure: In this exercise we will compare the runtime efficiency of statically typed and compiled language, namely C++, versus popular JIT and interpreted language, namely Java and Python. Use the following procedure to collect experimental data:

1. You will be conducting this experiment on the local laboratory computer **and not on the Linux server!** So do not log onto the Linux server to avoid confusion.

2. Open a terminal on your local lab computer. Next, setup your working directory on your lab computer using the following commands:
```
$ scp os1.csi.miamioh.edu:/tmp/toggle_test.zip ./
$ unzip toggle_test.zip
$ cd toggle_test
```

3. Now, briefly study the solutions in the 3 programming languages, from `toggle.cpp`, `Toggle.java`, `toggle.py` while noting the following:
    a. Note the `readFile` method in each language to load the contents of a file into a string.
    b. Note the `toggleCase` method in each language. This method has very similar implementation in each language. The C++ solution is naïve (the C++ solution can be made ~2× faster using SSE instructions) but the solution in Java and Python has been modified to be more optimal (without changing the core logic). So, you will notice the use of arrays etc. in Java. In Python list comprehension and other approaches were tested but were found be slower than the one provided to you.
    c. The main methods in the programs perform the following operation:
        i. Load data from a given file (first command-line argument)
        ii. Toggle the case of a given list of characters (second command-line argument)
        iii. Print the final string.

4. We will run these three programs with identical inputs and record (in the table further below) the following 2 key runtime characteristics using `/usr/bin/time`:
   a. Record their user time (*i.e.*, time taken to run instructions in the program only) – this is the value before the word `user`, on the 1st line of output as shown below.
   b. Memory usage of the program as reported by "`maximum resident set size`" (2nd line of output) and **divide it by 1,000,000** to get memory in MB.

```
$ /usr/bin/time -l ./toggle cdna.txt abcdefghi > /dev/null
        21.73 real          21.62 user         0.10 sys
 300855296  maximum resident set size
         0  average shared memory size
…
```

5. Now compile and run the three programs as shown below and record their runtime (while discarding outputs `to /dev/null`) using the commands shown below:

```
$ g++ -std=c++14 -O3 toggle.cpp -o toggle
$ /usr/bin/time -l ./toggle cdna.txt abcdefghi > /dev/null
```

Record the 2 statistics for the C++ version of the program in the table below.

```
$ javac Toggle.java
$ /usr/bin/time -l java Toggle cdna.txt abcdefghi > /dev/null
```

Record the 2 statistics for the Java version of the program in the table below.

```
$ /usr/bin/time -l python toggle.py cdna.txt abcdefghi > /dev/nul
```

Record the 2 statistics for the Python version of the program in the table below.

Language	**User time (seconds)**	**Speed-up*** **Time ÷ c++'s time**	**Memory usage (MB)**	**Memory-Scale*** **Memory ÷ c++'s memory**
C++	2.51	1	300.8	1
Java	12.71	5.09	3454.6	11.48
Python	147.51	58.7	3003.547	9.99

*__Note__: Speed-up and Memory-scale are computed by dividing an observation with the corresponding value for C++. For example, assume the C++ program uses 10 MB and runs in 10 seconds, while the Java program uses 11 MB and runs in 11 seconds; then C++'s speed-up would be 11÷10 = 1.1× and its memory-scale will be 11÷10 = 1.1× – *i.e.*, C++ is 1.1× faster and uses 1.1× less memory.

Inferences: Now that you have collected some data let's draw some inferences based on the observations.

When drawing inferences, avoid the following common Logical Fallacies. A logical fallacy is a flaw in reasoning. Logical fallacies are like tricks or illusions of thought, and sometimes they lead to incorrect conclusions to be drawn. See for https://yourlogicalfallacyis.com/ for full list, examples, and discussions on common fallacies.

- **Bandwagon**: Millions of people use Java and Python; just look at the job market. They all can't be wrong – The fallacy is along the lines – "*Millions of doctors prescribe opioid-based medicines. So, opioids are good for you.*"
- **Personal Incredulity**: I find Java and Python to be easier than C++. So that must be the case for everyone – The fallacy is along the lines – "*I find Chinese to be hard. Consequently, everyone else would find it hard to speak in Chinese*"
- **Anecdotal**: Architects, CTOs, and other business experts who are way above Dr. Rao's poverty-pay grade have assured me that Java and Python are best – The fallacy is along the lines, say in 2008 – "*The wall-street analysts and CFOs said the economy is great. So, the great recession of 2008 cannot happen*".
- **Strawman / Texas sharpshooter**: These experiments involving array/string processing are not representative and are wrong because – most applications are more complex and as program become more complex, languages become faster (which is not really the case).
- **Appeal to emotion**: Come on, doing small changes is not worthwhile because someone else is going to molest the environment anyways – The fallacy is along the lines – "*If you don't smoke, someone else is going to do it and you will inhale second-hand smoke which is worse, so you might as well smoke.*"

1. For this example (and hence, most likely for similar programs), which programming language provides the fastest runtime? Why do you think that is the case?

   C++. By avoiding a lot of the built in handholding, structural overhead, and protective compilations that a lot of these other languages offer, C++ is freed up to move at a much faster rate without avoid checks

2. For this example (and hence, most likely for similar programs) which processed ~287 MB of data, which programming language provides the smallest memory footprint? Why do you think that is the case?

   C++ is strongly typed and avoids automatic garbage collection, this means the compiler becomes easily aware of what kind of data it is processing, and it does not have to take up extra space to keep track of what the user needs deleted.

3. If you were to develop an algorithm for Machine Learning (ML) / "data science" application that required processing large data sets, what programming-language would you recommend, based on your observations?

   C++ runs very fast at batch processing, C++ seems like a clear choice

4. If you were to develop an algorithm for Artificial Intelligence (AI) application that required training on large data sets (which is typically the case), what programming-language would you recommend, based on your observations?

   C++ runs very fast at batch processing, C++ seems like a clear choice

5. Web-based applications are just string processing programs. If you were to develop high-throughput web-sites, what programming-language would you recommend, based on your observations?

   C++ runs very fast at batch processing, C++ seems like a clear choice

6. All of these applications like data science, AI, web-processing, games etc. are relatively inconsequential as they are dwarfed by environmental issues and energy consumption. There is really no point in having ML/AI etc. if we as human can't even breathe air, drink water, and eat good food due to pollution. Unfortunately, your generation has the inevitable responsibility to address environmental concerns.

   Given that energy consumption is proportional to runtime, what programming language would you recommend so that technologies can minimize their energy usage and be kind to our environment?

   C++ seems like the clear choice again

7. For this question, let's assume (even though in science we like verify like we did in this part of the exercise) that our observations are broadly applicable to many types of programs and we can substantially reduce energy usage. What responsibility do you think students (like yourself) and teachers in educational institutions like Miami University have, if any?

   I think students/teachers have a responsibility to teach and learn the use cases for various programming languages, and they need to understand the consequences of not using a more efficient processing language (like C++).

Part #2: Preparing for professional success in jobs

Estimated time: 20 minutes

Background: Long-term career growth and professional success in any Information-Technology (IT) job requires several key skills as summarized in the video https://youtu.be/1pTnJwxQJwU

1. Adaptability: Learn and use new programming languages, skills, and software tools
2. Attention to quality: Well formatted (style is the first thing people notice and you get only one chance to make "a first impression"), tested, and documented solutions
3. Proactive: Identify and resolve issues well before deadlines (rather than do a postmortem analysis of issues delaying product delivery)

4. Team player: Help colleagues resolve issues (but not do their job or plagiarize their work)

The above long-term success skills require constant practice, not just in school, but even when you are in a job.

Exercise: In this part of the exercise you are expected to reflect upon your experiences and respond to the following questions with regards to the aforementioned 4 key long-term success skills.

1. This course required you to practice problem-solving in a different but imperative programming language. Use of a different IDE was encouraged. Briefly (about 2-to-3 sentences) describe how this course-characteristic addressed one (or more) of the aforementioned 4-key long-term success skills?

   Adapatability was the kicker in this route. NetBeans has a lot of quirks and can prove to be a nuisance at times; however, we came out stronger having to roll with what we were handed.

2. This course emphasized and motivated good programming practices, specifically good programming style. Briefly (about 2-to-3 sentences) describe how this aspect of the course helped to gain experience on one of the aforementioned 4-key long-term success skills?

   Good programming style hits the box of "Attention to detail" as our code was required to be neat, readable, and well commented. Producing garbage shill work was unacceptable in this environment.

3. This course heavily used automated testing and gain experience with working with testing tools. Many students were confounded by this aspect of the course and incorrectly think it was to help the instructor or graders. Briefly (about 2-to-3 sentences) describe how working with automated testing helps practice one of the 4-key long-term success skills?

   Since the thorough testing suites through the CODE plugin enabled us to be Proactive in ensuring our code was correct and accounted for all cases. We were required to produce the output predicated on a given input

4. Jobs and technologies will pose unanticipated challenges. Resolving these challenges in a timely manner by working with your colleagues and your boss is critical. In this course students were expected to resolve issues during office hours or optionally work with TAs to deliver homework problems by a given deadline. Briefly (about 2-to-3 sentences) describe how this experience helps practice one of the 4-key long-term success skills?

Working with students in the lab, as well as asking Dr. Rao for help when needed was the fundamental to team building in this course. We were required to interact in this manner in order to succeed effectively.

5. This course motivated the use of discussion forums to post question and respond to questions from other students. Briefly (about 2-to-3 sentences) describe how this aspect of the course helps practice one of the 4-key long-term success skills?

   Team building is underlying all group discussion threads, which allowed us to cooperate to achieve a desired result. Dr. Rao helping to facilitate these discussions, acting as "the boss" allowed us to succeed.

Submit to Canvas

Once you successfully completed the aforementioned exercises upload the following files to Canvas.

i. This MS-Word document (duly filled-in) saved as a PDF document.

Ensure you actually **submit** the files after uploading them to Canvas.