CS206 Introduction to Data Structures

Lab 3

How long did that take?

UNIX

Two topics for today: copying files and recording output. (Recording is at the end of the lab.)

Copying files: the UNIX comment to copy files is cp. For instance

```
cp x y
```

would make a copy of the file named x under the name y (assuming x exists in the current directory).

```
cp DIR1/x DIR2/y
```

makes a copy of the file named x that is in the the directory DIR1 and puts that copy into a file named y in the directory DIR2. (Assuming x, DIR1 and DIR2 exist.)

```
cp DIR1/*.java DIR2/
```

will make a copy of every file that end with .java in DIR1 and put those copies into DIR2 (assuming DIR1 and DIR2 exist.) The * is a UNIX wildcard, it allow you to say something like “all” or in this case all that ends in .java. Further specialization is possible — for instance A*.java would copy copy on those files that start with A and end in .java

(Note that scp is a generalization of cp which allows you to copy from one machine to another)

How long did that take:

Accurately timing things on computers is hard because many factors get in the way. In class we will discuss abstractions for timing to ease this problem. Still sometimes the actual time is important. Consider the following code for collecting the time required for the doWork() method to complete.

```
public class Timer {
    public static void main(String[] args) {
        long startTime = System.nanoTime(); // yes, timing in 
nanoseconds
        new Timer().doWork(1000);
        long endTime = System.nanoTime();
        // now covert nanoseconds to seconds
        System.out.println("Time: " + (endTime-startTime)/
1000000000.0); 
    }
    public void doWork(int amt) {
        double res=0.0;
        for (int i=0; i<amt; i++) {
            for (int j=0; j<amt; j++) {
                for (int k=0; k<amt; k++) {
                    res += Math.sqrt(1.0*i*j*k);
                }
            }
        }
        System.out.println(res);
    }
}
```

Enter this into VSC and try it. On my computer it takes about 4 seconds. How fast is it on yours? (if it takes more than 10 seconds, change 1000 to 500 and make the same adjustment below. This code allows you fairly fine grained control over what you are timing. Note that the time also includes the time required to create an instance of Timer. Revise the code to eliminate that issue.

Experiment a little. How does the time taken by this program change is you change 1000 to 2000 or 500? Divide the times before and after your change. With some vigorous rounding you should get a factor of 8 change when you double (or halve) the value passed to doWork.
Why?

Another way of timing is to time the entire program run using UNIX utilities. To do so in a terminal

```
UNIX> javac Timer.java
UNIX> time java Timer
```

On my machine the time reported by UNIX is about 0.1 seconds more than the internal time. Why?

Finally, notice that in the provided code, the time includes the time required to create an instance of the Timer class. How can you revise the code so that the time does not include the creation of an instance of Timer?

UNIX capturing the output of programs into files

It is common to create a file to hold the output of your program. You can do this in UNIX using something called IO redirection. The simplest form of IO redirection is to cause what would have appeared on the screen to instead be written to a file. For instance, using the Timer code you just worked with (first change the value passed to doWork to 500)

```
UNIX> javac Timer.java
UNIX> java Timer > timerout.txt
UNIX> cat timerout.txt
```

The first line compiles Timer; the second line runs it, capturing the output into the file timerout.txt. The final line just prints that file to the screen. All of the work is done by the ">" which the operating system interprets as "take everything that is being written to standard output (in Java System.out) and put it in the file timerout.txt. The name of the file is completely your choice it could be "a" or "thisistheoutfilefromtimer". Similarly the extension ".txt" is your choice, UNIX does not care.

To the timer program, add the following line just blow the existing println of the the value of the variable res.

```
System.err.println("This is writing to the error stream");
```

Now do the second UNIX command again. The println you just added still shows on your screen. This is because the UNIX ">" writes from standard output (i.e. System.out in Java) to a file. It handles standard error (System.err in Java) separately. To capture standard error to a file use "2>". For example

```
UNIX> java Timer > timerout.txt 2> timererr.txt
```

You can use one, the either or both of ">" and "2>".

Note that each time you use ">" it creates a new file, replacing the file that was there. Sometimes you want to add a new run a file rather than replacing. To do so, use ">>". for example:

```
UNIX> java Timer > timerout.txt
UNIX> java Timer >> timerout.txt
```

The result of this is to have two runs the Timer program with output in the same file.

Finally, sometimes you do not care about the output, you just do not want it on the screen or even in a file (which you would then have to delete). In this case rather do the following:

```
UNIX> java Timer > /dev/null
```

/dev/null is effectively a trash can; so this command say to take the output and throw it away without every showing it to me.

What to Hand In:

Send email to gtowell206@cs.brynmawr.edu with the following:

A table of the time required for each of the runs in the "how long did that take" section. Also, describe the machine you ran the code on. For instance, for me this is:

```
2020 powermac with 2.3 GHz 8-Core Intel Core i9
500 0.56
1000 4.32
2000 36.65
```

If you did the lab on a CS department machine, just say CS lab for the machine description

After this table, answer the question about why the numbers in this table differ by a factor of 8. If you do not have a good answer make one up.