CS 366  
Assignment #2  
Matrix Multiplication Matters  
Due 2/7/18, in class

1. Goal  
The goal of this assignment is to write your first real C program while admiring the cache’s impact on computation time by implementing the matrix multiplication algorithm, and its variants, from Assignment 1.

2. Problem Statement [client’s statement of their need]  
Mr. Krabs needs a lightning quick matrix multiplier.  
Back Story: (see Assignment 1)

3. Analysis [What is the client's problem?]  
This too is the same as Assignment 1.

Assignment Requirements  
1. Clone the assignment repo using the invitation at  
   https://classroom.github.com/a/rGUeDODk.
2. Edit mm.c to start with the following header comment  
   // This is my code  
   // <Your Name>  
   // CS366  
3. Next include (as a comment right after the header comment) the machine name, processor model, cache size, and alignment (i.e., the block or line size), as well as the sizes of the L1, L2, and L3 caches. The first two can be obtained using the commands  
   grep model /proc/cpuinfo | head -2  
   grep cache /proc/cpuinfo | head -2  
   (try these commands without the “| head -2”)  
   Include also the cache information from  
   lscpu  
   and the output of  
   hostname  
   Cache and other information can be found on cites such as http://www.cpu-world.com.
4. Then, write the body of the function  
   void multiply(array A, array B, array C, int n)  
   where A, B, and C are each square, n \times n, matrices. After you have it is working, create functions  
   patrick and sponge for the two corresponding variants.
5. When writing code, be sure to use clear internal documentation and careful formatting. See the general notes from Day 1. In particular, be consistent in indentation (2 or at most 4 spaces please) and the alignment of braces. Each open brace “{” must be on its own line.
6. Finally, make sure that your code generates no warnings or errors when compiled using gcc -Wall.
What to hand in

1. A well-formatted 2-up printout of your source code including all three versions of multiply. You must use a2ps after removing all the tabs from your code. (Ask me how vi or sed can be used to accomplish this.)
   Consider creating an alias in your bash_profile to include the following options
   
   ```
   a2ps -T 4 -q -Avirtual -2 -o mycode.ps.
   ```
   (and before printing, checking the formatting using `gv mycode.ps`.)

2. A no-more-than one page summary explaining who was right and why you think the order does or does not matter. Include and refer to a copy of your program’s output in your explanation.

3. Your final GitHub repo.
   - Yes git log will tell me if you Hail Mary it.
   - Include a useful Readme.md file in your repository.
   - It is bad (for your grade) if you commit derivable files (e.g., .o and the executable files).
   - I expect to pull your code, run make, and then run my test script.

Grading Rubric
Here is a list things that I will look for

- use of a2ps
- source indentation
- labeled output (think about creating an impressive presentation)
- function header comments
- comment quality (e.g., don’t comment the obvious)
- the existence of gcc warnings
- the file header
- quality and insight in the write up (omitting the data or not labeling the output are a bad start)

Important: “git commit; git push” is not what you do once to turn in your code. I expect you to commit and push your code each time you reach a notable milestone. I will use the output “git log” as part of your grade.

Extra Fun!

Run the command `grep processor /proc/cpuinfo` and then count the number of processors (or try `grep processor /proc/cpuinfo | wc`). Let $n$ be this number divided by 2. Compared to running one copy of your code, is running $n - 1$ copies faster or slower? How about $n + 1$ copies??