Problem Set 1
ISE 407 – Computational Methods in Optimization
Due: September 13, 2019
Dr. Ralphs

This first assignment is to get you familiar with and using C and Python, buiding on what we did in Assignment 0. Completing this assignment will require some further research and familiarization with the programming environments we'll use during the rest of the semester. We'll learn in more detail how to do the kinds of testing and analysis asked for in this assignment throughout the course. The objective is simply to get familiar with the tools and take a first try at doing some more substantive computational experiments.

1. Using the source code on the course Web site
   `http://coral.ie.lehigh.edu/~ted/files/ie407/code/matmult/matmult.c`
   and the cache-optimized version from the slides, write functions for multiplying two matrices of a given size in C and Python, according to the following guidelines:

   - Your functions should use the following API:
     C: `int ** matmult (int ** A, int * dimA, int ** B, int * dimB, int ** C)`
     Python: `def matmult(A, B)`
     In both cases, the result of multiplying the two input matrices should be returned to the calling function. For the C function(s), `dimA` and `dimB` should be arrays containing the dimensions of `A` and `B`, respectively. Your function should check to make sure that the dimensions of the matrices are conformable. The argument `C` is the matrix to contain the result and is to be filled out by the method. The function should assume that memory is allocated and deleted by the caller.

     Note that unlike the example C code, your C code will need to allocate the memory for the matrices dynamically (and delete it). The way in which this is done is important. To take full advantage of the cache optimization, you should allocate the matrix as one giant block of memory, as discussed in class, not as separate blocks for each row (feel free to experiment with the difference between these two ways of allocating memory!).
   - For your Python implementation, you should code a "naive" implementation in addition to the one covered in Lecture 4.
   - For your C implementation, you should code (1) the naive version, (2) the version that takes the transpose of one matrix first, and (3) a fully cache-optimized version.

2. Write a test program in each language that generates random matrices in some "reasonable" way. You'll have to research how to generate random numbers on each platform. Use your wrapper programs to compare the speeds of your implementations on one or more graphs. The graphs should show the effect of the size of the matrices somehow and should show how the implementations compare to each other. It is up to you how to do this.

3. As before, you should provide a Makefile that builds your code, runs your experiments and produces the results.

4. Compare and contrast the two languages in light of your results and explain the outcomes of the experiments in as much depth as possible.