CSE 6010 Assignment 1: Matrix Multiply

Due Dates:

- Due: 10:00 AM, Friday, September 4, 2015
- Revision (optional): 11:59 PM, Monday September 7, 2015

The objective of this assignment is to write a program library for working with matrices that includes computation of the product of two dense matrices, C = A × B, among other functions. A second objective is to understand the computational performance of your program.

Your software must be structured with the matrix library defined in different files from the programs that utilize the library. You should be able to give the code for the matrix library to other users for them to use for their own purposes, independent of this assignment.

The interface to the matrix multiply library contains the type definitions and function prototypes defined below. These should be placed in a file `matrix.h`. This file defines the complete interface to the library. The implementations of these functions are defined in another file called `matrix.c`.

1. Define a data type for a two-dimensional matrix:

   ```
   typedef double ** matrix;
   ```

   Note that here a 2D matrix is an array of pointers to arrays of double precision numbers (make sure you understand what this means!).

1. Write a function with the prototype

   ```
   matrix make_matrix(int n1, int n2);
   ```

   Allocate memory for a matrix of doubles with `n1` rows and `n2` columns. Return a pointer to the new matrix or NULL if the memory allocation failed.

2. Write a function with the prototype

   ```
   int free_matrix(int n1, int n2, matrix a);
   ```

   Release memory used by a matrix of doubles pointed to by `a` with `n1` rows and `n2` columns. Return 0 if successful, else return 1.

3. Write a function with the prototype

   ```
   int matrix_multiply(int n1, int n2, int n3,
   matrix a, matrix b, matrix c);
   ```

   The matrix A is stored in `a` and has dimensions `n1` by `n2`, and the matrix B is stored in `b` and has dimensions `n2` by `n3`. When your function returns, the product C = A × B is stored in matrix `c`. Your function should return 0 if the computation was successful, and nonzero otherwise (e.g., `n1` is negative, `a` is NULL, or any other invalid condition). From the function prototype above, note that memory for the matrices must have been allocated before calling the `matrix_multiply` function.

4. Write a function with the prototype

```
int matrix_fill_random(int n1, int n2, matrix a);
```

that fills the matrix `a` with random values between −10 and +10. The return value of the function specifies whether or not an error occurred.

5. Write a function with the prototype

```
int matrix_print(int n1, int n2, matrix a);
```

that prints the matrix `a` in a readable format (i.e., columns are aligned).

To exercise the library, define a program in a separate file called `matrix_test.c` whose main function calls the above functions. Use this program to test your functions. In particular, choose A as a 4×3 matrix, and B as a 3×2 matrix. For this case, print A, B, C and hand in your results. Also hand in evidence that you checked that C is correct, e.g., a Matlab script that shows the same computation and results.

6. Write a program called matrix benchmark. This program prints the average time for computing one matrix multiply. The pseudocode for this program looks like this:

```
matrix_fill_random(A)
matrix_fill_random(B)
start timer
loop a large number of times
        C = A*B
endloop
stop timer
print average time for one matrix multiply
```

   The reason for the loop above is to average out any startup effects (i.e., this "warms up" the caches).

7. Run matrix benchmark using square matrices where the dimension of the matrices ranges from 100 to 1000, or as large as your machine can handle. Plot the average time as a function of the cube of the dimension. Provide an explanation of your results. (You may find it useful for your program to input the matrix dimension from the command line.)

Turn in a brief report including your results, and your software in a single zip file. Your software must be well documented and include comments so the code is easy to understand. You should include a README file with instructions on how to compile and run your program on the jinx cluster. Although this is not really necessary due to the relatively simple nature of this program, you should get into the habit of including such documentation with the software you develop. Finally, please note that all code you turn in must be completely developed by yourself, although we encourage you to discuss the problem and issues you are facing with other students (as well as the TA/instructor).

Suggestion: The web is an excellent resource to answer specific questions on C. We encourage you to use it, but be careful not to utilize code copied directly from the web.