COSC2436: Programming and Data Structures
Dense Matrix Multiplication

1 Introduction

You will create a C++ program to multiply a square matrix. You can program matrix multiplication with any algorithm, provided results are correct. It is preferred your algorithm can multiply matrices efficiently, avoiding unnecessary products or data movement in the memory hierarchy.

Given a matrix A of size $n \times n$ you will compute result matrix $C = AA = A^2$. Notice you will not compute AA^T.

2 Input and Output

The input is a single text file, with one matrix. The output is also a single text file, with one matrix.

File format: There will be ONE matrix row per line in the file. Each value is a real number that may or may not have a decimal point (e.g. 1, 2.1, 3.1416). Values are separated by spaces. Comments have a # at the beginning of the line (no spaces before); comments should be skipped, by the program.

Example 1 of input and result matrix

```
#Matrix A, I matrix, 2 dimensions
1 0
0 1

#Matrix C=AA, 2 decimals
1.00 0.00
0.00 1.00
```

Example 2 of input and result matrix

```
# Matrix A, 3 dimensions
0.5 3.0 0.0
0.0 1.0 0.8
1.0 0.0 0.2

#Matrix C=AA, 2 decimals
0.25 4.50 2.40
0.80 1.00 0.96
0.70 3.00 0.04
```

3 Program and input specification

The main C++ program will become the executable to be tested by the TAs. The result matrix should be left on another text file (output file), provided on the command line. Notice the input and output files are specified in the command line, not inside the C++ code. Notice also the quotes in the program call, to avoid Unix/Windows get confused.

The general call to the executable is as follows:

```
densemult "A=<file>;C=<file>"
```

Call example with one input file and another output file.

```
densemult "A=a.txt;C=c.txt"
```

Assumptions: Matrices are given in dense form. Real numbers are separated by one space. The number of columns in the first matrix row determines matrix size. The output matrix will be written in dense form with a fixed number of decimals.

4 Requirements

- You can use the classical $O(n^3)$ algorithm, but you can attempt a more efficient algorithm. Notice the input matrix A is assumed to be squared.
- C++:

  It is encouraged you do not use existing STL, vector classes since you will have to develop your own C++ classes and functions in future homeworks.

  Your C++ code must be clear, indented and commented.

  You must determine matrix size based on number of columns of the first row of the input matrix. Your program must reject input matrices with errors producing an empty file.

  You can use static arrays of a maximum size=20. Your program will be tested with matrices up to 20×20. You can optionally use dynamically-sized 1-dimensional arrays (not dynamic 2D arrays since those require more careful manipulation) whose size is determined at run time.

  In your C++ code you are encouraged to use subscripts $1 \dots n$ in 2D arrays wasting some memory, but you can use C++ $0 \dots n-1$.

  Include comments on each source code file and a README file on how to compile/run with examples, as obvious it may seem.

  Your program will be tested with GNU C++. Therefore, you are encouraged to work only on Unix. You can use other C++ compilers, but the TAs cannot provide support or test your programs with other compilers.
- Output:

  The output file must contain the result, in the same format (one row per line).

  Matrix values are written with 2 decimals separated by one space. Do not use other separators o a different number of decimals. Notice the number of decimals may vary in future homeworks.

  Your program should write error messages to the standard output (cout, ptintf).

- Testing for grading:

  Your main cpp program will be automatically compiled. If there is an error in compilation if will not be executed.

  Your program will be automatically stopped if it does not give an answer in 10 seconds.

  Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, zeroes and inconsistent information. Each exception will be -10.

  Your program will be tested with 10 test cases, going from easy to difficult. You can assume 70% of test cases will be clean, valid input matrices. If your program fails an easy test case 10-20 points will be deducted. A medium difficulty test case is 10 points off on average. Hard test cases are generally worth 5-10 points off. Difficult cases with specific input issues or complex algorithmic aspects are worth 5 points.

- Due date: as posted on my web page. In general the due date cannot changed. No exceptions, unless there are medical or exceptional reasons.

- Grade: A program not submitted by the deadline is zero points. A non-working program is worth 10 points. A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce correct results will get a score of 80 or higher. In general, correctness is more important than speed.

- Plagiarism and cheating: C++ code is individual: it cannot be shared (other than small fragments posted in the newsgroup). Programs will be compared for plagiarism. Notice we also have a repository of C++ examples from the Internet as well as past editions of the course. You must disclose any code you use from the Internet. If the TAs detect a portion of your C++ code is highly similar to C++ on the Internet or other student the grade will be decreased at least 50%.