COS 335 Spring 2011
Assignment 1 Due Thursday Jan 27

The purpose of the last problem in this assignment is to provide a demonstration that organization and architecture of a computer can impact even relatively simple programs. The program involved creates large arrays in virtual memory, then accesses them in row-major and column-major order. The slowdown in column-major access is an artifact of the way that C and Java compilers store arrays in memory, combined with cache assumptions about memory access, and with larger array sizes virtual memory being paged out to disk.

Problems 1-7 are 1 point each. Homework is graded on a 0 - 10 scale.

1. Manually convert the following numbers from unsigned hex to decimal. Show your work.
a) 2011 b) BEEF c) EAD4 d) DC

2. Assuming 16-bit numbers, convert the numbers in question 1 from signed 2's complement hex to decimal. Also, for d) convert from 8 bit-signed 2's complement hex to decimal.

3. Manually convert the following numbers from decimal to unsigned 16-bit hex:
a) 2011 b) 64206 c) 21333 d) 142

4. Negate the numbers in the last question and convert to 32 bit 2's complement hex. (For example, convert -2010 to 2's complement hex) This is easiest to do using the answers from 3 (almost trivial).

5. Assuming an 8-bit representation, list 4 pairs of bit strings whose binary interpretations as unsigned and 2's complement numbers have the following properties under 8-bit addition:
a) Both signed and unsigned addition are correct.
b) Signed addition is correct, but unsigned is not.
c) Unsigned addition is correct, but signed is not.
d) Neither signed nor unsigned addition are correct.

6. Show what the bit string 1001 0111 represents assuming a) 8-bit unsigned, b) signed magnitude, c) one's complement, d) two's complement, e) bias-127

7. Show 8-bit binary bit strings representing the number -67 in a) 8-bit signed magnitude, b) two's complement, c) bias-127 and d) one's complement

7. (4 pts) The two programs at the end of the assignment (matrix.cpp and Matrix.java) declare a large 2 dimensional array (64 MB in the example) and then initialize the array as an identity matrix, which has 1's in all cells where the row index is equal to the column index and 0's elsewhere. The program does this three times. The first pass accesses the entire array and makes sure that all memory required by the program is physically allocated by the operating system. (This is called "touching" memory). The second pass redoes the first pass, but times the program to see how long it takes and then displays the results. Both first and second passes use "row-major" order, accessing the array row by row. The third pass uses "column-major" order, accessing the array column by column.

For this problem, do the following, in one language or the other:
a. Find out how much physical memory is present in the computer on which you are running this program.

b. Change the MATSIZE constant for each run (note that an int is 4 bytes on both 32 and 64-bit machines, at least with all compilers with which I am familiar) so that the array occupies the following amounts of memory
128MB, 256MB, 512MB, 1024MB, 1536MB

c. Run the program once for each array size noted above. **Submit a report showing the output for each run and note the amount of physical memory and the operating system used on the report. Calculate memory accesses per second and graph these against array size for row and column major orders.**

The program will require several to many seconds to run for the larger array sizes, depending on machine speed. Don't panic and think your computer has crashed.

The C++ source code is not designed for Visual Studio IDE. It should be compiled and run the executable from the command line. I also recommend that you close ALL other applications while running the program so that the maximum amount of physical memory is available to your program.

Notes:

1. Array memory requirements in megabytes = ((MATSIZE ^ 2) * 4) / 1048576

2. When running in a command prompt you can redirect output to a file use the output redirection operator >. For example, with the C++ exe you can give the following command:

```
matrix > run1.txt
```

to place the output in a file called run1.txt. With Java (see note 4 below)

```
java -Xms1024M -Xmx1024M Matrix > run.txt
```

3. The C++ program will compile as is in the from the command line in Visual Studio 6 and from the command line in Visual Studio.NET 20xx, and gcc (g++). To compile from the command line in VS.NET use Start > Programs > Microsoft Visual Studio .NET 20xx > Visual Studio .NET Tools > Visual Studio .NET 20xx Command Prompt to open a command prompt, then issue the command:

Visual 6	:	cl matrix.cpp /EHsc
Visual Studio.NET:		cl matrix.cpp /GX

You can run this program on any type of OS that you like; Windows, Linux, Mac, Sun etc. You may also translate the program into another language and run the program using that language. If so make a note of the language used in your report. **Please submit: a) your report and b) your program if you did it in another language.**

5. To run the java program, first compile with javac (e.g., javac Matrix.java) then run with java (e.g., java -Xms100M -Xmx100M Matrix). The command line switch -Xms specifies starting heap size, -Xmx specifies max heap. These should be at least 10-20M more than the matrix size. If using a Java IDE, see your IDE documentation. Almost any machine can should be able to handle a heap of 1600MB

8. Extra Credit: 2 points. Compare the Java and C++ versions of the matrix code for all memory sizes. Graph your results.

matrix.cpp

```
#include <iostream>
#include <iomanip>
#include <time.h>
using namespace std;

#define MATSIZE 4096         //modify as per assignment

int main() {
  // ALLOCATE HEAP SPACE
  //**IMPORTANT TO USE THE DELETE OPERATOR WHEN DONE WITH THIS MATRIX**
  int (*mat)[MATSIZE];
  mat = new int[MATSIZE][MATSIZE];
  int i,j;
  clock_t start, finish;

  cout.setf(ios::fixed | ios::showpoint);

  cout << "Starting" << endl;
  // create identity matrix. Do one pass untimed just so that we
  // "touch" and therefore physically allocate all required memory
  for (i=0; i<MATSIZE; i++)
      for (j=0; j<MATSIZE; j++)
          mat[i][j] = i == j ? 1 : 0;


  // do it again and time it
  cout << "Timing row order access..." << endl;
  start = clock();
  for (i=0; i<MATSIZE; i++)
      for (j=0; j<MATSIZE; j++)
          mat[i][j] = i == j ? 1 : 0;

  finish = clock();
  cout << "MATSIZE: " << MATSIZE << " Row First TIME " << setprecision(3)
  <<  (finish - start) / (double) CLOCKS_PER_SEC << " seconds." << endl;

  cout << "Timing column order access..." << endl;
  start = clock();
  for (j=0; j<MATSIZE; j++)
      for (i=0; i<MATSIZE; i++)
          mat[i][j] = i == j ? 1 : 0;
  finish = clock();
  cout << "MATSIZE: " << MATSIZE << " Col First TIME " << setprecision(3)
  <<  (finish - start) / (double) CLOCKS_PER_SEC << " seconds." << endl;
  cout << "Array is " << (int)(MATSIZE * MATSIZE * sizeof(int)) / 1048576 <<
"MB";

  // if running from IDE add some code to pause here
  delete [] mat;  // DEALLOCATE SPACE
return 0;
}
```

Matrix.java

```
class Matrix
// to run with 280 megabytes of heap: java -Xms280M -Xmx280M Matrix
{
public static final int MATSIZE = 4096;        //modify as per assignment
public static int[][] mat = new int[MATSIZE][MATSIZE];

// for 2gb 23170
// memory occupied = 20067 * 20067 * 4 bytes = 1.5gb
public static void main(String args[]) {
  int i,j;
  long start, finish;
  System.out.println("Starting\n");
  // create identity matrix. Do one pass untimed just so that we
  // "touch" and therefore physically allocate all required memory
  for (i=0; i<MATSIZE; i++)
      for (j=0; j<MATSIZE; j++)
          mat[i][j] = i == j ? 1 : 0;


  // do it again and time it
  System.out.println("Timing row order access...\n");
  start = System.currentTimeMillis();
  for (i=0; i<MATSIZE; i++)
      for (j=0; j<MATSIZE; j++)
          mat[i][j] = i == j ? 1 : 0;
  finish = System.currentTimeMillis();
  System.out.println("MATSIZE: " + MATSIZE + " Row First TIME " +
  + (double)(finish - start)/1000.0 + " seconds.\n");

  System.out.println("Timing column order access...\n");
  start = System.currentTimeMillis();
  for (j=0; j<MATSIZE; j++)
      for (i=0; i<MATSIZE; i++)
          mat[i][j] = i == j ? 1 : 0;
  finish = System.currentTimeMillis();
  System.out.println("MATSIZE: " + MATSIZE + " Col First TIME " +
  + (double)(finish - start)/1000.0 + " seconds.\n");

  System.out.println("Array is " + (int)(MATSIZE * MATSIZE * 4) / 1048576 +
"MB\n");

  }
}
```