ACMS 20210 Exam 3

Due: 11:59 PM on Wednesday, May 3, 2017

Submit the C++ programs below using Sakai. If you are compiling a single file from the command line on the CRC or another Linux system, I suggest using the command

```
g++ your_program_name.cpp -std=c++1y -o your_executable_name
```

To compile a program using several files, with the main routine in the file main_program.cpp and additional functions, classes, etc., in other source files souce_1.cpp and source_2.cpp with associated headers source_1.h and source_2.h, first, compile each separately into an object file:

```
g++ -std=c++1y -c source_1.cpp
g++ -std=c++1y -c source_2.cpp
g++ -std=c++1y -c main_program.cpp
```

This will produce a .o file for each of your .cpp files. To link them together into an executable, use:

```
g++ source_1.o source_2.o main_program.o -std=c++1y -o your_executable_name
```

The procedure is similar when the program is split over other numbers of source files.

For this exam, to simplify grading, please use the following naming scheme for organizing your programs, unless indicated otherwise. For a given problem number i, use only one .cpp file, with the name exam_3_problem_i.cpp. For example, your submission for problem 1 should be in a single file named exam_3_problem_1.cpp.

You are allowed to work with others, to use notes, and to use outside sources, but you must cite any sources that you use. You may look at code that others have written for similar problems, but you must write your own implementation for the final version that you submit.

1. In this problem, we will simulate moving a knight around an $N \times N$ chess board at random, using only legal moves. The knight will start in the lower left corner of the board. For each step, the knight will move according to one of its available legal moves, selected uniformly at random (i.e. each legal move has the same chance of being selected). We are interested in the expected number of moves required until the knight returns to the lower left corner.

   Your program should take two command line arguments: the number N indicating the size of the board, and a number indicating the number of trials that will be used in your Monte Carlo simulation. If the number N is less than 3, the program should terminate with an appropriate message to the user.

   For an 8×8 board, the expected number of moves before the knight returns to the lower left corner is 168.

2. In this problem, we will simulate moving a king around an $N \times N$ chess board at random, using only legal moves. The king will start in the lower left corner of the board. For each move, the king will move according to one of its available legal moves, selected uniformly at random (i.e. each legal move has the same chance of being selected). We are interested in the expected number of moves until the king has visited every square on the board at least once. After 0 moves, the king has already visited one square (the lower left).

   For a 3×3 board, the expected number of moves to visit every square at least once is approximately 28.9 moves.

3. In this problem, we will write a program that computes student grades using the grading system for this course, according to the policy sheet. Your program should read a file called student_data.dat whose format will be similar to the one in the example directory. For each student, the first line will contain the student's full name, the second the student's homework scores for the six homework assignments, separated by spaces, the third the three exam scores, separated by spaces, and the fourth the final exam score. Each student's data will be separated by a blank line. For the purposes of this exam problem, you may assume that each student has only standard English letters (no diacriticals, etc) and no punctuation (e.g. apostrophes as in O'Connell) in their names.

   Your program should report back to the user the names of the students (in "last comma first" form) along with their final average, sorted in alphabetical order of last name (you may want to use sort from the algorithm library).

   Clarification on the input file: Each of the homeworks in the file is out of 20 possible points, and collectively the homeworks are scaled to be worth 100 out of 550 possible points. Each of the exams is graded out of 100 points, and collectively the exams are worth 300 out of 550 points. The final exam is scored out of 100 points, and is scaled to be worth 150/550 total points.

   Your program should use a C++ class to store the student's data, with a reasonable implementation. One of the member functions should compute the student's final grade.

   Place your Student class definition in a header file called Student.h and define its member functions in a file called Student.cpp.

   For this problem, you should submit three files:

   (a) A main routine in a file named exam_3_problem_3.cpp
   (b) A .h file called Student.h containing the definition of your Student class
   (c) A .cpp file called Student.cpp containing the definitions of the member functions of your student class

4. Write a program that will read a square $N \times N$ matrix $\mathbf{A}$ of doubles from a file, whose name will be supplied by command line argument, and an $N \times 1$ vector $\mathbf{b}$ of doubles from a file whose name will also be supplied by command line argument. The program then solves the linear system

$$\mathbf{Ax} = \mathbf{b}$$

for the vector $\mathbf{x}$ and writes $\mathbf{x}$ to a file called vector_output.dat, in the same format as the vector $\mathbf{b}$. (See the appropriate example files for what these might look like for a small matrix $\mathbf{A}$).

You should solve the linear system using Gaussian elimination (using either back substitution or complete row reduction to finish solving the system). For this problem, the matrix $\mathbf{A}$ is guaranteed to be non-singular (invertible), so that the linear system will have exactly one solution.

Along the way, count the number of arithmetic operations (addition, subtraction, multiplication and division) that were used during Gaussian elimination and the solution process. Report the size of the matrix and the operation count to the user. (You do not have to incorporate updating the count of operations itself as an operation!) Do not count swapping rows during Gaussian elimination in the operation count.