ECS 60 **Programming Assignment #1 (50 points)** **Winter 2017**

Programs due Wednesday, January 18th by 11:59pm. Write-ups due Monday, January 23rd by 4pm in 2131 Kemper.
Filenames: timetest.cpp, queens.cpp, authors.csv

authors.csv should be in the following format (-5 points for incorrect format):
First line: <e-mail of first partner> <comma> <last name of first partner> <comma> <first name of first partner>
Second line (if needed): <e-mail of second partner> <comma> <last name of second partner> <comma> <first name of second partner>

For example:
```
hpotter@ucdavis.edu,Potter,Harry
fflintstone@ucdavis.edu,Flintstone,Fred
```

Use handin to turn in just your files to cs60. Do not turn in any Weiss files, object files, makefiles, or executable files! You will find copies of my own executables in ~ssdavis/60/p1. All programs will be compiled and tested on Linux PCs. All programs must match the output of mine, except that the CPU times may be different. Do not get inventive in your format! Programs are graded with shell scripts. If your program asks different questions, or has a different wording for its output, then it may receive a zero!

#1. Timetest: (30 points with 25 points for write-up and 5 points for timetest.cpp, 15 minutes)

Write a driver program, timetest.cpp, that will ask for a filename and repeatedly asks the user for the ADT to which he/she wishes to apply the commands from the specified file. You will then run your program and note the performance of ADT in a two or three page typed, double-spaced write-up. **Hand written reports will receive no points!** We will be storing only integers for this assignment. The format of each file will be:

First Line: A string describing the contents of the file.
Second Line: {<command (char)> [values associated with command]}+

The two commands in the files are: 1) insert: 'i' followed by an integer and a space character; and 2) delete 'd' followed by an integer and a space. Since only the list ADTs can delete a specific value, you need delete the specific value for only those three implementations. For stack, simply pop the next integer, and queue simply dequeue the next integer, no matter what the value is specified by the delete command. Some ADT constructors require a maximum size parameter. You should hard code this to 500,001.

Your driver program will need all of the following files from ~ssdavis/60/p1: CPUTimer.h, LinkedList.cpp, StackAr.cpp, dsexceptions.h, CursorList.cpp, LinkedList.h, StackAr.h, CursorList.h, QueueAr.cpp, StackLi.cpp, vector.cpp, QueueAr.h, StackLi.h, vector.h, SkipList.h, SkipList.cpp, File1.dat, File2.dat, File3.dat, and File4.dat. You may copy the .h and .cpp files, but you should simply link to the .dat files using the UNIX ln command.: `ln -s ~ssdavis/60/p1/*.dat .` (Note the period to indicate your current directory.)

To make CursorList compile you should add the following line below your #includes, and pass cursorSpace in the CursorList constructor:

```
vector<CursorNode <int> > cursorSpace(500001);
```

After you've completed your program, apply File1.dat, File2.dat, File3.dat, and File4.dat three times to each ADT and record the values returned. You will find the shell script run3.sh in ~ssdavis/60/p1 that will do that for you, assuming the name of your executable is a.out. Just type "run3.sh", and then look at the file "results" to see the times for all eighteen runs.

In your write-up, have a table that contains the values for each run, and the average for each ADT for each file. Another table should contain the time complexity for each ADT for each file; this should include five big-O values: 1) individual insertion; 2) individual deletion; 3) entire series of insertions (usually N times that of an individual insertion); 4) entire series of deletions (usually N times that of an individual deletion); and 5) entire file. These two tables are not counted as part of the required two or three pages need to complete the assignment. For each ADT, you should explain how the structure of each file determined the big-0. Concentrate on <u>why</u> ADTs took longer or shorter with different files. Do not waste space reiterating what is already in the tables. For example, you could say "Stacks perform the same on the three files containing deletions because they could ignored the actual value specified to be deleted." The last section of

the paper should compare the ADTs with each other. Most of the differences can be directly explained by their complexity differences. The lion's share of the last section should explain why some ADTs with the same complexities have different times. In particular, why is the CursorList slower than the normal list? You should step through Weiss' code with gdb for the answer.

The members of a team may run the program together, but each student must write their own report. Turn in this write-up to the appropriate slot in 2131 Kemper. If you declare ct as a CPUTimer then the essential loop will be (I suggest that you copy it from the online version of this assignment with no fear of plagiarism) :

```
do
{
     choice = getChoice();
     ct.reset();
     switch (choice)
     {
       case 1: RunList(filename); break;
       case 2: RunCursorList(filename); break;
       case 3: RunStackAr(filename); break;
       case 4: RunStackLi(filename); break;
       case 5: RunQueueAr(filename); break;
       case 6: RunSkipList(filename); break;
     }

     cout << "CPU time: " << ct.cur_CPUTime() << endl;
} while(choice > 0);
```

A sample run of the program follows:

```
% timetest.out
Filename >> File2.dat

      ADT Menu
0. Quit
1. LinkedList
2. CursorList
3. StackAr
4. StackLi
5. QueueAr
6. SkipList
Your choice >> 1
CPU time: 15.66

      ADT Menu
0. Quit
1. LinkedList
2. CursorList
3. StackAr
4. StackLi
5. QueueAr
6. SkipList
Your choice >> 0
CPU time: 0
%
```

#2. Eight Queens : (20 points, 1.5 hours) Filename: queens.cpp

from: *Data Structure, Algorithms, and Object-Oriented Programming* by Gregory L. Heilman, p. 187.

In the eight queens problem you are asked to place eight queens on a chessboard (an 8 x 8 grid) so that no queen is in a position to take any other queen. (A queen may take any other queen that is on the same row, column, or diagonal.)

A brute-force algorithm for solving this problem systematically checks all possible ways of placing eight queens on a chessboard to see if a solution has been found. There are 64 possible positions for the first queen, 63 remaining for the second queen, and so forth. This means that the brute force approach may run through 64 * 63 * 62 * 61 * 60 * 59 * 58 * 57 ≈ 3 x 10^{14} iterations.

The brute-force approach checks some arrangements that we can immediately rule out. For instance, we know that no two queens can appear in the same row. Thus, a solution can always be represented using a vector P[1..8], where P[i] is the column placement of the queen in the i-th row, and therefore the actual queen positions are given by (1, P[1]), (2,P[2]), …, (8,P[8]). We say that the vector P is k-feasible if none of the queens in positions (1, P[1]), (2, P[2]), …, (k, P[k]) can take any other queen in these positions.

a) (20 points) Develop a backtracking search algorithm that solves the eight queens problem by systematically producing k-feasible vectors, for k = 1,2,…,8 (Hint: Vector P is k-feasible if for every $i \neq k$ between 1 and k, $P[i] - P[j] \notin \{i - j, 0, j - i\}$.)

b) (5 points extra credit) Extend your algorithm so that it produces all solutions to this problem.

There is no write-up for this problem. Your file, named queens.cpp, may use any Weiss files from part #1, timetest, you wish. You may not use recursive calls. Each line of output should be a solution of the eight queens problem. If you only complete part a, then your program will output one line. The format of each line should be a list of the column placements separated by commas. See the example below for the format

```
[ssdavis@lect1 p1]$ queens.out
1,5,8,6,3,7,2,4
1,6,8,3,7,4,2,5
1,7,4,6,8,2,5,3
1,7,5,8,2,4,6,3
2,4,6,8,3,1,7,5
2,5,7,1,3,8,6,4
2,5,7,4,1,8,6,3
2,6,1,7,4,8,3,5
2,6,8,3,1,4,7,5
2,7,3,6,8,5,1,4
2,7,5,8,1,4,6,3
2,8,6,1,3,5,7,4
3,1,7,5,8,2,4,6
3,5,2,8,1,7,4,6
3,5,2,8,6,4,7,1
3,5,7,1,4,2,8,6
3,5,8,4,1,7,2,6
3,6,2,5,8,1,7,4
3,6,2,7,1,4,8,5
3,6,2,7,5,1,8,4
3,6,4,1,8,5,7,2
3,6,4,2,8,5,7,1
3,6,8,1,4,7,5,2
3,6,8,1,5,7,2,4
3,6,8,2,4,1,7,5
3,7,2,8,5,1,4,6
3,7,2,8,6,4,1,5
3,8,4,7,1,6,2,5
4,1,5,8,2,7,3,6
4,1,5,8,6,3,7,2
4,2,5,8,6,1,3,7
4,2,7,3,6,8,1,5
4,2,7,3,6,8,5,1
4,2,7,5,1,8,6,3
4,2,8,5,7,1,3,6
4,2,8,6,1,3,5,7
4,6,1,5,2,8,3,7
4,6,8,2,7,1,3,5
4,6,8,3,1,7,5,2
4,7,1,8,5,2,6,3
4,7,3,8,2,5,1,6
4,7,5,2,6,1,3,8
4,7,5,3,1,6,8,2
4,8,1,3,6,2,7,5
4,8,1,5,7,2,6,3
4,8,5,3,1,7,2,6
5,1,4,6,8,2,7,3
5,1,8,4,2,7,3,6
5,1,8,6,3,7,2,4
5,2,4,6,8,3,1,7
5,2,4,7,3,8,6,1
5,2,6,1,7,4,8,3
5,2,8,1,4,7,3,6
5,3,1,6,8,2,4,7
5,3,1,7,2,8,6,4
5,3,8,4,7,1,6,2
5,7,1,3,8,6,4,2
5,7,1,4,2,8,6,3
5,7,2,4,8,1,3,6
5,7,2,6,3,1,4,8
5,7,2,6,3,1,8,4
5,7,4,1,3,8,6,2
5,8,4,1,3,6,2,7
5,8,4,1,7,2,6,3
6,1,5,2,8,3,7,4
6,2,7,1,3,5,8,4
6,2,7,1,4,8,5,3
6,3,1,7,5,8,2,4
6,3,1,8,4,2,7,5
6,3,1,8,5,2,4,7
6,3,5,7,1,4,2,8
6,3,5,8,1,4,2,7
6,3,7,2,4,8,1,5
6,3,7,2,8,5,1,4
6,3,7,4,1,8,2,5
6,4,1,5,8,2,7,3
6,4,2,8,5,7,1,3
6,4,7,1,3,5,2,8
6,4,7,1,8,2,5,3
6,8,2,4,1,7,5,3
7,1,3,8,6,4,2,5
7,2,4,1,8,5,3,6
7,2,6,3,1,4,8,5
7,3,1,6,8,5,2,4
7,3,8,2,5,1,6,4
7,4,2,5,8,1,3,6
7,4,2,8,6,1,3,5
7,5,3,1,6,8,2,4
8,2,4,1,7,5,3,6
8,2,5,3,1,7,4,6
8,3,1,6,2,5,7,4
8,4,1,3,6,2,7,5
[ssdavis@lect1 p1]$
```