ECS 60 **Programming Assignment #1 (50 points)** **Winter 2016**

Due: Friday, January 15th, Write-ups: 4:00pm; Programs: 11:59pm.
Filenames: timetest.cpp, boarding.cpp, and authors.csv.

Format of authors.csv:	author1_email,author1_last_name,author1_first_name
	author2_email,author2_last_name,author2_first_name
For example:	simpson@ucdavis.edu,Simpson,Homer
	potter@ucdavis.edu,Potter,Harry

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! Please note that there should be no spaces in authors.csv unless your last name has spaces in it.

#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 250,000.

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(250000);
```

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:

```
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;
     }  // switch

     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. Airline Boarding Strategies: (20 points, 1 hour coding, 1 hour debugging) Filename: boarding.cpp

Airlines try to minimize the time it takes to load each airplane. There are three strategies commonly used: 1) back to front with six zones, 2) random, 3) and outside in with 3 zones. These are discussed at: http://menkes76.com/projects/boarding/boarding.htm.

You are to create a simulation for a plane that has 48 rows of six seats each, that determines the time it will take to board a plane for each of the strategies. You will find my boarding.out, CreatePassengers.*, passengers18.txt, and three passenger files in ~ssdavis/60/p1.

```
      [ssdavis@lect1 p1]$ boarding.out passengers-1.txt
      Back to front: 3160
      Random: 2625
      Outside in: 2190
      [ssdavis@lect1 p1]$
```

Here are the specifications:

1. Each passenger will have a specific unique seat assigned. Seats A and F are window seats, seats B and E are middle seats, and seats C and D are aisle seats. Rows are numbered 1 to 48.
2. Passenger Files
    - 2.1. The program will be given the name of a passenger file as its only command line parameter.
    - 2.2. The passenger files are guaranteed to be valid.
    - 2.3. Each file has 3 lines. Each line corresponds to a valid listing of 288 assigned seats for an entire planeload of passengers. The entry for each seat is separated by spaces from the others, and is listed with its row followed by its seat letter, e.g. `"12A 5E 36B ….”`
    - 2.4. The first line contains a listing with six zones of 48 passengers presented with the zones listed back-to-front. The seat assignments within each zone are random.
    - 2.5. The second line has 288 passengers listed in a completely random order.
    - 2.6. The third line has a listing with three zones of 96 passengers presented with the first zone being window seats, the second zone being middle seats, and the third zone being the aisle seats. The row assignments within each zone are random.
    - 2.7. Passenger files are generated by CreatePassengers.out which is compiled from CreatePassengers.cpp. CreatePassengers.out asks only for a seed for the random generator, and uses that seed in the name of the passenger file created, e.g., passengers-27.txt will be based on a seed of 27.
3. Passengers always board the plane at row 1.
4. Each passenger takes 10 seconds to store their carry-on luggage.
5. When the aisle in the next row is clear, a passenger takes 5 seconds to move to the next row.
6. Getting to or from one’s seat in a row takes 5 seconds.
7. Each previously seated passenger who is seated closer to the aisle than a newly arrived passenger’s assigned seat, must leave their seat, and step into the aisle, and then return to their seat after the new passenger has seated herself. For example, if a Seat F passenger has stowed her luggage and the passengers in Seat D and Seat E were already seated, then Seat D passenger would take 5 seconds to get to the aisle, Seat E would take 5 seconds to get to the aisle, Seat F would take 5 seconds to reach her seat, Seat E would take 5 seconds to reach his seat, and Seat D would take 5 seconds to reach her seat. For a total of 25 seconds after Seat F had stowed her luggage.
8. No passenger may pass another passenger in an aisle. In the previous example, no one can move past the row until Seat D had reached her seat. To keep things simpler, we assume that seated passengers who must leave their seats somehow find room in the aisle of their same row.
9. The program will print out the simulated time it took to seat each planeload for each strategy in the format <u>exactly</u> matching that shown below.
10. Your program may only #include iostream, fstream, StackAr.h, and QueueAr.h. If your program uses any other #includes it will receive a zero. Since you will only be submitting boarding.cpp, you may not alter StackAr.*, nor QueueAr.*.
11. The only array allowed in the program is the argv command line parameter that contains the filename!! If there is the use of the array bracket operator, other than argv[1], anywhere in your program it will receive a zero.
12. Hints and suggestions.
    - 12.1. Your simulated clock should simply add five seconds at the end of iterating through the all of the rows of the plane.
    - 12.2. A given row’s actions are sometimes dependent on the next row’s action. In which order should you process the plane’s row.
    - 12.3. Create a state transition diagram to describe what should happen to a row under what circumstances. Such diagrams are like flow charts, but the edges have labels based on circumstances. <u>Some</u> of the states could be: EMPTY, WAITING_TO_MOVE, STORING_LUGAGGE1, STORING_LUGGAGE2, AC_OUT, and DF_IN. <u>Some</u> of the circumstances could be: seated passenger in the way, aisle empty in next row, and previously seated passengers in aisle storage. You can read about state transition diagrams at: http://www.cs.unc.edu/~stotts/145/CRC/state.html and see examples at http://users.csc.calpoly.edu/~jdalbey/SWE/Design/STDexamples.html
    - 12.4. Create a Row class that has, among other things, separate data structures for each side of the aisle, a storage data structure for previously seated passengers that have to get up for a new passenger, storage for the passenger in the aisle, an enum for the row’s current state.
    - 12.5. Create a shortened version based on passenger18.txt that has fills only the first 3 rows of a plane.

Contents of passengers18.txt: `1C 2A 3D 2E 2B 1A 1B 2C 1F 2D 3A 2F 1D 3B 3F 1E 3C 3E`

+ = passenger storing luggage

* = aisle of row is not empty because previously seated passengers are still in it.

Time	Row 1 Waiting	Row 2 Waiting	Row 3 Waiting	Row1 Seated	Row 2 Seated	Row 3 Seated	Row 1 Out	Row 2 Out	Row 3 Out
5	1C								
10	1C+								
15	1C+								
20	2A			C					
25	3D	2A		C					
30	3D	2A+		C					
35	3D	2A+		C					
40	2E	3D		C	A				
45	2B	2E	3D	C					
50	2B	2E+	3D+	C					
55	2B	2E+	3D+	C					
60	1A	2B		C	AE	D			
65	1A+	2B+		C	AE	D			
70	1A+	2B+		C	AE	D			
75	1A				ABE	D	C		
80	*			A	ABE	D	C		
85	1B			AC	ABE	D			
90	1B+			AC	ABE	D			
95	1B+			AC	ABE	D			
100	1B			A	ABE	D	C		
105	*			AB	ABE	D	C		
110	2C			ABC	ABE	D			
115	1F	2C		ABC	ABE	D			
120	1F+	2C+		ABC	ABE	D			
125	1F+	2C+		ABC	ABE	D			
130	2D			ABCF	ABCE	D			
135	3A	2D		ABCF	ABCE	D			
140	3A+	2D+		ABCF	ABCE	D			
145	3A+	2D+		ABCF	ABCE	D			
150	2F	3A		ABCF	ABCDE	D			
155	1D	2F	3A	ABCF	ABCDE	D			
160	1D+	2F+	3A+	ABCF	ABCDE	D			
165	1D+	2F+	3A+	ABCF	ABCDE	D			
170	3B	2F		ABCDF	ABCE	AD		D	
175	3B	2F		ABCDF	ABC	AD		DE	
180	3B	*		ABCDF	ABCF	AD		DE	
185	3B	*		ABCDF	ABCEF	AD		D	
190	3F	3B		ABCDF	ABCDEF	AD			
195	1E	3F	3B	ABCDF	ABCDEF	AD			
200	1E+	3F	3B+	ABCDF	ABCDEF	AD			
205	1E+	3F	3B+	ABCDF	ABCDEF	AD			
210	1E		3F	ABCF	ABCDEF	ABD	D		
215	*		3F+	ABCEF	ABCDEF	ABD	D		
220	3C		3F+	ABCDEF	ABCDEF	ABD			
225	3E	3C	3F	ABCDEF	ABCDEF	AB			D
230	3E	3C	*	ABCDEF	ABCDEF	ABF			D
235		3E	3C	ABCDEF	ABCDEF	ABDF			
240		3E	3C+	ABCDEF	ABCDEF	ABDF			
245		3E	3C+	ABCDEF	ABCDEF	ABDF			
250			3E	ABCDEF	ABCDEF	ABCDF			
255			3E+	ABCDEF	ABCDEF	ABCDF			
260			3E+	ABCDEF	ABCDEF	ABCDF			
265			3E	ABCDEF	ABCDEF	ABCF			D
270			*	ABCDEF	ABCDEF	ABCEF			D
275				ABCDEF	ABCDEF	ABCDEF			