ECS 60 **Programming Assignment #1 (50 points)** **Spring 2014**

Due: Wednesday, April 9th, Write-ups: 4:00pm. Programs 11:59 pm.
Filenames: timetest.cpp, LongInt.cpp, LongInt.h

Use handin to turn in just your files to cs60. Do not turn in any object files, data files, or executable files! Place all authors names on only the first line of timetest.cpp, and puzzle.cpp. 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 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 why 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;
     }

     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 (20 points, 1 hour) Filenames: LongInt.cpp, LongInt.h

I have written a program, LongIntMain.cpp, to add unsigned integers of UNLIMITED length. It uses the class LongInt. Write the implementation code for the LongInt class. LongIntMain.cpp, and a Makefile are provided in ~ssdavis/60/p1. You may not alter either of these files. You may assume that the user will enter only digits for the numbers, but the number of digits entered may be very large. You may not: read the digits into an array, nor may you use the STL. Your two files will be copied into a directory where LongIntMain.cpp, Makefile, and the Weiss files used for timetest will be located. The script will then we will run make using my Makefile.
Hints: I used Weiss List, Weiss StackLi, isdigit(), peek(), and a static local variable

```
[ssdavis@lect1 p1]$ cat LongIntMain.cpp
#include <iostream>
#include "LongInt.h"
using namespace std;

int main()
{
  LongInt int1, int2, int3;
  cout << "Please enter first long integer >> ";
  cin >> int1;
  cout << "Please enter second long integer >> ";
  cin >> int2;
  int3 = int1 + int2;
  cout << int3 << endl;
  return 0;
}

[ssdavis@lect1 p1]$ LongInt.out
Please enter first long integer >> 172340958712349871234931287
Please enter second long integer >>
432598741390874309587243098572340985723098571908731924
432598741390874309587243098744681944435448443143663211
[davis@lect2 p1]$
```