Name: ____

This assignment focuses on the following topics: (1) studying and implementing a simple graph algorithm efficiently; (2) learning the use of *JFlex*[1] and *CUP*[2] to implement a simple language; and (3) packaging and documenting code following good conventions. These topics are very closely related to the assigned readings, especially those on Lex[3] and Yacc[4], and classroom exercises and should help solidify that material. Further details and clarifications will be announced in class or the newsgroup.

You should submit (1) a hard-copy of pages 1–12 of this assignment with your answers filled in, and (2) an electronic package that contains the source files for your work on the programming questions, following the submission procedure described below and in class. The hard-copy is due in class before the due date and time, and the electronic package must be submitted before the hard-copy using the interface at `http://cs.umaine.edu/~chaw/u/` only. No other forms of submission are accepted.

You are welcome to use any inanimate resources (e.g., books, Web sites, publicly available code) to help you with your work. However, *all such help must be clearly noted* in your submissions. Further, no matter what you use, *you must be able to explain, in detail, how it works.* (You may be called upon to explain your homework in person.) Refer to the class policy for details, and ask for clarifications if you are unsure if something is allowed.

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) Package and submit your solutions to the programming questions, and any other electronic components of your work, as a *single file* named using the template `cos397-L-F-hw01-N.jar`, replacing `L` and `F` with your last and first names, and `N` with an arbitrary 4-digit integer (and `jar` with `tgz` or another extension if appropriate). You may make multiple submissions (within reason) by using different values of `N`. *After* submitting your work, fill in the following:

File name:	________________________
Size, in bytes:	________________________
MD5 checksum:	________________________
Timestamp:	________________________

[1] Gerwin Klein, JFlex User's Manual, 2004.

[2] Scott E. Hudson, CUP User's Manual, 1999.

[3] Michael E. Lesk and Eric Schmidt, "Lex—A Lexical Analyzer Generator," in Andrew G. Hulme and M. Douglas McIlroy (eds.), *UNIX Vol. II: research system*, 10th edition (Philadelphia, Pennsylvania: W. B. Saunders Company, 1990).

[4] Stephen C. Johnson, Yacc: Yet Another Compiler-Compiler, Unix Programmer's Manual, Volume 2b. AT&T Bell Laboratories. Murray Hill, New Jersey, 1978.

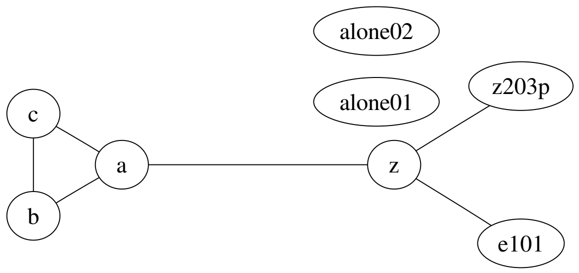
3. (18 pts.) *Well-known graphs.* The notation K_n denotes the *complete* graph on n vertices while $K_{m,n}$ denotes the *complete bipartite* graph with m vertices on one side and n on the other. P_n denotes a *path* graph on n vertices, C_n denotes an n-vertex *cycle*, W_n denotes an n-vertex *wheel*, and S_n denotes an n-vertex *star*.

   Depict K_n, $K_{n-1,n+1}$, P_n, C_n, W_n, and S_n for $n = 3, 5, 10$ in a manner that best illustrates their structure.

[additional space for answering the earlier question]

4. (10 pts.) *Linear representation.* We define a linear representation of graphs, illustrated by the following example.

```
c--a z--a z
alone01 z--e101 z203p--z
alone02 b--c a--b
```



The input consists of whitespace-separated tokens. Tokens represent either edges or vertices. A vertex token consists simply of an identifier that follows the conventions of C. An edge token has the format *u*`--`*v* where *u* and *v* are vertex identifiers. A vertex may be introduced either directly, using a vertex token, or indirectly, by its use as an endpoint of an edge (or both).

Provide linear representations for each of the graphs of Question 3, labeling them clearly.

[additional space for answering the earlier question]

5. (10 pts.) *Normalized representation.* A linear representation of a graph is in normal form if

   - there is no whitespace other single spaces separating adjacent tokens;
   - no vertex identifier appears in both an edge token and a vertex token;
   - the vertices of each edge are listed lexicographically; and
   - all the tokens are sorted lexicographically.

   For example, the following is a normalized version of the linear representation of Question 4.

   ```
   a--b a--c alone01 alone02 a--z b--c e101--z z--z203p
   ```

   Convert each of the linear representations of Question 4 into normal form.

[additional space for answering the earlier question]

6. (10 pts.) For each graph of Question 3, indicate if the graph is Eulerian and, if so, exhibit an Eulerian circuit in the normalized representation of Question 5.

[additional space for answering the earlier question]

7. (10 pts.) Describe, in English and as clearly as possible, a *linear-time* algorithm for finding an Eulerian circuit in a given graph, or determining that no such circuit exists. *Explain* why your algorithm is correct. Check carefully that you handle all cases.

8. (20 pts.) Provide pseudocode for your algorithm of Question 7, ensuring that the resulting description is clear enough for a colleague to implement without much additional work.

9. (20 pts.) Prove that your algorithm's running time is $O(n)$ where n is the input size. What is n, in terms of standard graph parameters? What is your algorithm's space complexity?

10. (50 pts.) Implement your algorithm of Question 8. Your program should read from standard input and write to standard output. The input is a graph in the linear notation of Question 4. The output should be an Eulerian circuit, if it exists, and empty otherwise. The Eulerian circuit should be represented by listing the vertices separated by two dashes in a manner similar to the input (e.g., `a--b101--fz--y--a`).

    You should write your program in Java, using JFlex and CUP for parsing the input. (These parsing routines form the basis for parsing more complicated input in the next assignment, so are worthwhile even for this simple input.) Ensure that your code runs properly on *gandalf*. You must document, package, and submit your source code properly to receive any credit. Do not submit object code or other binary blobs. Ensure that your submission includes a README file with conventional instructions and a Makefile that permits everything to be compiled using a single 'make' command. You should also include at least a few test inputs and outputs. For a good example of how to package simple code, including tests, refer to the sample code included in the *JFlex* and *CUP* packages. Please ask for clarifications if any of the requirements are unclear, especially because **submissions that do not follow these instructions will receive zero credit.**

11. (50 pts.) Conduct suitable experiments using your implementation of Question 10 to provide an empirical quantification of the running time and memory footprint of your implementation. Compare your results with the answer to Question 9 and explain the agreement or discrepancies.

    Summarize your results appropriately using tables, charts, and text, and include a suitably named file with this material in your electronic submission. In that file, be sure to explain exactly how the experiments were conducted, and how you ensured that the reported results are accurate and significant.