CS-5340/6340, Programming Assignment #1
Due: Tuesday, September 6, 2011 by NOON (12:00pm)

Your task is to build a **morphological analyzer**. Your morphological analysis program should accept three input files: (1) a dictionary file, (2) a rules file, and (3) a test file. Your program should be named **"morphology"** and should accept these files as command-line arguments in the following order:

morphology <dictionary_file> <rules_file> <test_file>

The Dictionary File

The dictionary file consists of words paired with their possible parts-of-speech (POS). If a word can have multiple parts-of-speech, then each POS will be on a separate line. The POS tags should be treated as arbitrary strings (i.e., you should *not* assume a finite set), but you can assume that all POS tags will be a single word (i.e., not a phrase). Some dictionary entries for irregular word forms may also be followed by a "ROOT" keyword and the root word. For example, the word "sat" is an irregular verb so a dictionary entry for "sat" would include the "ROOT" keyword followed by the root word "sit".

Here is a sample dictionary file:

carry	verb
furry	adjective
alumni	noun
bark	noun
bark	verb
sit	verb
sat	verb ROOT sit

Your program should treat the dictionary as **case insensitive** (e.g., the words "dog", "Dog", and "DOG" should all be considered the same word). You can assume that any root listed in the dictionary will have its own entry in the dictionary (e.g., "sit" is the root of "sat" in the example above and has its own dictionary entry). You should <u>not</u> assume that the dictionary is sorted in any way.

The Rules File

The rules file will consist of morphology rules, one per line. Each rule will have 5 parts:

1. PREFIX or SUFFIX keyword
2. beginning or ending characters (depending on prefix/suffix keyword)
3. replacement characters, or a hyphen if no characters should be replaced
4. the part-of-speech tag required for the original word from which the new word is derived
5. the part-of-speech tag that will be assigned to the new (derived) word

The characters –> will be used to separate the part-of-speech tag required for the root from the part-of-speech tag to be assigned to the derived word. Each rule will end with a period. A sample rule file is shown below:

```
SUFFIX   ly   -   adjective  –>  adverb .
SUFFIX   ily  y   adjective  –>  adverb .
SUFFIX   ed   -   verb  –>  adjective .
SUFFIX   ed   -   verb  –>  verb .
SUFFIX   ied  y   verb  –>  adjective .
SUFFIX   ied  y   verb  –>  verb .
PREFIX   re   -   verb  –>  verb .
SUFFIX   s    -   noun  –>  noun .
SUFFIX   s    -   verb  –>  verb .
```

The Test File

The test file will contain words that your morphological analyzer should be applied to. The file will contain one word per line.

When grading your program, we will run your morphological analyzer both on the test file that we give you in advance, as well as a completely new, more comprehensive test file. So please test your code extensively!!!

How to Apply the Rules

First, look up the word in the dictionary. If the word is present, then its dictionary definition should be returned. If the word is not in the dictionary, then apply your morphological analyzer. When applying the morphology rules, remember that (1) ALL of the rules need to be applied because more than one rule may be successful, and (2) the rules need to be applied recursively until a root form is found in the dictionary or until no more rules apply. As soon as you find a word in the dictionary that matches the most recent rule that was applied, the derivation can be deemed successful and the recursion can stop. (But remember that you still need to search for multiple derivations.)

If your morphological analyzer cannot find any derivation for a word, then your program should generate a default definition for the word as a NOUN. [In practice, unknown words are often assumed to be nouns because unfamiliar words are often proper names.]

Multiple Derivations: Multiple derivations for a word will be common! For example, if a word ends in "ied" then the 3rd, 4th, 5th, and 6th rules in the sample rule file may all apply! Every unique definition that is produced should be printed. For example, if "carried" can be derived as both a verb and an adjective, then both the verb and adjective definitions should be printed. Duplicate definitions, however, should not be printed. If your program generates exactly the same definition by applying different rules, or by applying the same rules in a different order, then only print one of these definitions.

OUTPUT SPECIFICATIONS

Your program should output each *word definition* in the following format:

WORD ROOT SOURCE POS

Each definition should appear on a separate line. SOURCE indicates how the definition was obtained, and has 3 possible values:

dictionary: if the word was found in the dictionary (without applying morphology).
morphology: if the morphological analyzer successfully derived the word.
default: if the word is not in the dictionary and the morphological analyzer failed to find a derivation for the word.

Remember that the morphological analyzer can generate multiple definitions for a word. For the sake of readability, please print a blank line between each set of definitions generated for the same word. For example, the words "carry", "carried" and "xyz" should produce the following output (based on the sample dictionary and rule files shown earlier):

```
carry carry dictionary verb

carried carry morphology adjective
carried carry morphology verb

xyz xyz default noun
```

ELECTRONIC SUBMISSION INSTRUCTIONS
(a.k.a. "What to turn in and how to do it")

You need to submit 3 things:

1. The source code files for your program. Be sure to include all files that we will need to compile and run your program ourselves!

2. A README file that includes the following information:

   - how to compile and run your code
   - which CADE machine you tested your program on
     (this info may be useful to us if we have trouble running your program)
   - any known bugs or problems with your program

   REMINDER: your program **must** compile and run on the CADE machines! We will not grade programs that cannot be run on the CADE machines.

3. A trace file that shows the output of your program given the dictionary, rules, and test files on our class web page:

   http://www.eng.utah.edu/~cs5340/assignments/morphology/dict.txt
   http://www.eng.utah.edu/~cs5340/assignments/morphology/rules.txt
   http://www.eng.utah.edu/~cs5340/assignments/morphology/test.txt

   You can generate a trace file in (at least) 3 different ways: (1) print your program's output to a file called *trace.txt*, (2) print your program's output to standard output and then pipe it to a file (e.g., *morphology dict.txt rules.txt test.txt* > *trace.txt*), or (3) print your program's output to standard output and use the unix *script* command before running your program on the test files. The sequence of commands to use is:

   ```
   script trace.txt
   morphology dict.txt rules.txt test.txt
   exit
   ```

   This will save everything that is printed to standard output during the session to a file called `trace.txt`.

How To Submit

To turn in your files, the CADE provides a web-based facility for electronic handin, which can be found here:

`https://cgi.eng.utah.edu/webhandin/index.cgi`

Or you can log in to any of the CADE machines and issue the command:

`handin cs5340 morphology <filename>`

HELPFUL HINT: you can see a list of the files that you've already turned in by using the 'handin' command without giving it a filename. For example:

`handin cs5340 morphology`

will show you all of the files that you've turned in thus far. If you submit a new file with the same name as a previous file, the new file will overwrite the old one.

GRADING CRITERIA

We will run your program on both the input files that we give you as well as new test, more comprehensive files to evaluate the generality and correctness of your code. **So please test your program thoroughly!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on new test cases.

HELPING US HELP YOU

If your program does not work correctly, then we will try to give you as much partial credit as possible. But grading incorrect or incomplete programs is difficult. If you print debugging output while developing your code, it may be useful for us to see this debugging output to help us understand exactly what your code is doing. So we encourage you to turn in an additional file that contains more detailed print statements and/or debugging output that shows what your program is doing. Please save this output separately, though, in a different file with an appropriate name, such as *debugging.txt*. The official trace file that you hand in should not contain any debugging statements.