CS-5340/6340, Programming Assignment #1
Due: Thursday, September 13, 2018 by 11:59pm

Your task is to build a **morphological analyzer** from scratch. 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 will consist 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 of tags), 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 file 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 (relative to the prefix/suffix keyword)
3. replacement characters, or a hyphen if no characters should be replaced
4. the part-of-speech tag **required for the originating 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 POS tag required for the originating word from the POS tag that will 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. A sample test file is shown below:

```
carried
recarry
barked
```

Morphology Rule Matching Algorithm:

Given a test word, first look up the word in the dictionary. If the word is present, then its dictionary definition should be returned and the process ends. *No morphological analysis should be done in this case.*

If the test word is not in the dictionary, then apply the morphology rules. When applying the rules, remember that (1) ALL of the rules must be applied because more than one rule may be successful, and (2) the rules must be applied recursively until a match is found in the dictionary OR until no more rules can be applied. When you find a word in the dictionary that matches a rule, that particular derivation path stops and is deemed successful. But remember that you still need to search exhaustively for additional 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 typically 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 rules (shown on the previous page) may all apply! *Every distinct 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. However, 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 that definition once.

OUTPUT SPECIFICATIONS

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

```
<word> <pos> ROOT=<root> SOURCE=<source>
```

Each definition should appear on a separate line. The ROOT should be the word ultimately found in the dictionary during the application of morphology rules. If the word in the dictionary has an explicit root defined, then use its defined root (e.g., use "sit" as the root for "sat"). If no explicit root is defined, then assume that the word itself is a root form.

SOURCE indicates how the definition was obtained, and has 3 possible values:

> **dictionary**: if the word was found in the dictionary *without* applying morphology rules.
> **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 verb ROOT=carry SOURCE=dictionary

carried adjective ROOT=carry SOURCE=morphology
carried verb ROOT=carry SOURCE=morphology

xyz noun ROOT=xyz SOURCE=default
```

GRADING CRITERIA

Your program will be graded based on new test cases! **So please test your program thoroughly to evaluate the generality and correctness of your code!** 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.

Please exactly follow the formatting instructions specified in this assignment. We will deduct points if you fail to follow the specifications because it makes our job much more difficult to grade programs that do not conform to the same standards.

IMPORTANT: You may not use ANY external software packages or dictionaries to complete this assignment except for *basic* libraries. For example, libraries for general I/O handling, math functions, and regular expression matching are ok to use. But you may not use libraries or code from any NLP-related software packages, or external code that performs any functions specifically related to morphological analysis. All submitted code *must be your own.*

SUBMISSION INSTRUCTIONS

Please use CANVAS to submit a gzipped tarball file named "morphology.tar.gz". (This is an archived file in "tar" format and then compressed with gzip. Instructions appear below if you're not familiar with tar files.) Your tarball should contain the following items:

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

   REMINDER: your program **must** be written in Python or Java, and it **must** compile and run on the Linux-based CADE (lab1 or lab2) machines! We will not grade programs that cannot be run on the Linux-based CADE machines.

2. An executable *shell script* named `morphology.sh` that contains the exact commands needed to compile and run your morphology program on the data files provided on CANVAS. We should be able to execute this file on the command line, and it will compile and run your code. For example, if your code is in Python and does not need to be compiled, then your script file might look like this:

   ```
   python morphology.py dictionary.txt rules.txt test.txt
   ```

   If your code is in Java, then your script file might look something like this:

   ```
   javac Morphology/*.java
   java Morphology/MainClass dictionary.txt rules.txt test.txt
   ```

3. A `README.txt` file that includes the following information:

   - Which CADE machine you tested your program on
     (this info may be useful to us if we have trouble running your program)
   - Any known idiosyncracies, problems, or limitations of your program.

4. Submit one trace file called **morphology.trace** that shows the output of your program when given the sample input files provided on Canvas.

HELPFUL HINTS

TAR FILES: First, put all of the files that you want to submit in a directory called "morphology". Then from the parent directory where the "morphology" folder resides, issue the following command:

tar cvfz morphology.tar.gz morphology/

This will put everything underneath the "morphology/" directory into a single file called "morphology.tar.gz". This file will be "archived" to preserve any structure (e.g., subdirectories) inside the "morphology/" directory and then compressed with "gzip".

FYI, to unpack the gzipped tarball, move the morphology.tar.gz to a new location and issue the command:

tar xvfz morphology.tar.gz

This will create a new directory called "morphology" and restore the original structure and contents.

For general information on the "tar" command, this web site may be useful:
https://www.howtogeek.com/248780/how-to-compress-and-extract-files-using-the-tar-command-on-linux/

TRACE FILES: You can generate a trace file in (at least) 2 different ways: (1) print your program's output to standard output and then pipe it to a file (e.g., `morphology dict.txt rules.txt test.txt > morphology.trace`), or (2) print your 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 morphology.trace
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 `morphology.trace`.