CIS*4150/Software Reliability and Testing: F17
Test Suite 1
(Individual work programming assignment)

1 Overview

Your company is adding a text-to-speech system to an existing product line. Your team has been asked to write a tool that will output the set of phonemes for a text-based sentence.

The first milestone your team has been asked to reach is the creation of a robust prototype system that will produce the set of phonemes for a set of input words, one word at a time.

Your company president is going to use this in a *live demonstration* in a couple of weeks time that will either allow you to clinch a major business investment or sink the deal.

Your task is therefore to make sure that when she is on the stage, that the demonstration prototype that your team is producing works flawlessly. Your team lead has divided the work up so that they are writing the C code prototype, and they have asked you to write a testing subsystem. There is a lot on the line for your company, so your team lead and your company president are both relying on you to ensure that the prototype works correctly during this demo.

You have found that there is a standard database of phonemes, provided from a project at Carnegie Mellon University, which is available in the cmudict-0.7b text file.[1] and you have decided that using this standard database of phonemes is a good starting point. (As the CMU project updates its database, you can simply replace the data file.)

Your team lead has already written a stub C program that reads this database, and then accepts keywords either on standard input or in additional files. For each keyword, a set of phonemes will be produced.

1.1 Program behaviour

The C language program is intended to conform to all the behaviours listed below.

(a) Upon startup, it must load the CMU format database file into an in-memory representation, reporting an error on standard error if this fails.
(b) For each subsequent file listed on the command line, your program should read each line in the file as a word, and print on standard output the sequence of phonemes making up this word. If the word cannot be found, the string "`?????`" (five question marks) should be printed on standard output.
(c) Files that cannot be opened should cause an error message to be printed on standard error, and the termination of the program.
(d) Upon successful completion of each file, the string "`=====`" should be printed.
(e) If no keyword files are given as program arguments, keywords are read from standard input. In this case, no terminating "`=====`" string is printed.
(f) Upon successful completion of processing, the string "`DONE! (`*999*` words)`" should be printed, where "*999* is replaced with the actual number of words processed.

[1] This file is also available via the CMU sphinx project:
http://svn.code.sf.net/p/cmusphinx/code/trunk/cmudict/cmudict-0.7b

(g) If multiple pronunciations of a word are present (potato, tomato), then each should be printed by default.

The program is expected to be well-behaved, and report any problems discovered in processing on standard error, and exit with a non-zero exit status in any such case.

2 CMU File format

The CMU database file conforms to a simple format:

- lines beginning with "`;;;`" are comments
- all other lines are *data*
- *data* lines consist of a *tag*, followed by a delimiter of two spaces, followed by a *list of phonemes*, followed by a newline;
- a *list of phonemes* consists of at least one *phoneme* – if there is more than one, they are separated by a single space;
- a *tag* is a string made up of upper-case alphabetic characters, digits, and punctuation, but no "whitespace" characters;
- a *phonemes* are made up of upper-case alphabetic characters and digits;
- all *data* lines are provided in sorted lexicographic order by tag; and
- the file encoding is ISO-8859-1, allowing accented Roman characters.

Further assumptions made on this file format include:

- the maximum length of a line is 128 characters;
- the maximum length of a tag is 32 characters; and
- there will be no more than 150,000 words in the file.

3 Testing task

Your task is to provide a "robust edge testing" testsuite for this program.

Before you begin writing your test cases, answer these questions:

Question 1: How many different variables do you need to control?

Question 2: What are the partitions of each variable?

Question 3: What are the normal test values for each variable?

Question 4: What are the robust test values for each variable?

3.1 Test cases

Based on the above answers, plan your test cases.

Write your test cases so that they exercise the `cmulookup` program, and determine whether it behaves as intended.

When your test suite it run, it should print a PASS or FAIL status for each test case, along with the identifier of which test case has been run. This identifier must clearly indicate which test case is run, and which variables are being exercised.

In addition, your test case must be able to be run separately from the other tests.

This assignment is expected to be based on code that you have written yourself, starting from materials discussed in class and in the lab.

4 Deliverables

Your testsuite should contain at minimum these things:

- a "`README.txt`" or "`README.md`" file – this should be a simple text file describing:
  - who you are (your name and ID)
  - how to run your test suite
  - what files are included in your assignment, and what they are for
  - how complete your assignment is
- a file "`QUESTIONS.txt`" (or "`QUESTIONS.md`") containing your answers to the questions posed above
- a program file called "`runtests`" that should run your entire test suite if run without any arguments
- one or more sub-directories with the naming pattern "`testcases*`" in which all of your individual test cases reside.

Bundle all of these together into a single archive (either of .zip or .tar format) and submit your assignment by uploading this to CourseLink.

Be sure to **copy your file to a test location and unpack it** to ensure that all of the files you want to submit are included, **and that they run as intended** on the machine `linux.socs.uoguelph.ca`.

In particular, you want to make sure that you have:

- all of the code required to run your tests; and
- all of the files containing your written answers.

5 Starting Point

A tar file containing the CMU database and the C program your team lead has written is available here: http://socs.uoguelph.ca/ andrew/teaching/cis4150/CIS4150-F17-TS1.tgz

Use this and the materials from the first lab to get started.