Compilers — Homework 1

Due: Friday, 21 Sep 2012, 13:30

1 Moodle

Enroll in the course on Moodle:

`http://www2.icorsi.ch/course/view.php?id=132`

2 Installation

Be sure you have Python 2.7 installed. You will need this version for the project. Note that Python 3 is a newer, incompatible version of the language, and removes some of the libraries we'll use in the project. Older versions of Python may also work, but no guarantees.

Be sure `gcc` and `gdb` are installed. You will need these later. For Mac users, the developer tools with Xcode should include everything we'll use.

3 Python exercises

The following exercises are intended to give you some practice with the Python library and with the use of Python constructs like lists and dictionaries. Work in pairs. Take a look at the Python documentation at:

`http://docs.python.org/library/`

Familiarize yourself with the language syntax and libraries. If you are new to Python, you'll refer to this site often.

For each exercise N below, submit a Python script called `exercise`N`.py`. Include the names of both members of your pair in each file. Turn in on Moodle by the due date. Scripts that do not parse or that fail with a run-time error (i.e., a traceback) are worth 0 points. Each exercise below uses a text file as input. A ready source of text files to test your solutions can be found at `http://www.textfiles.com/`.

1. [1 pts] Create a script that takes the name of a text file as a command-line argument. The script should read the text file and count the number of occurrences of each word. The script should ignore the case of the letters as well as punctuation and numbers. For example, "car", "car.", "car,", "2car" and "Car" should all be counted as occurrences of the word "car". After reading the file, print on the standard output all the words with their number of occurrences. *Hint*: Take a look at the `re` module.

2. [2 pts] Create a script that takes two or more command-line arguments:

   - the name of a *dictionary* file
   - one or more *words*

   The *dictionary* should contain a list of words and phrases, one per line. For each *word* in *words*, your script should print to the standard output, one per line, all words in the *dictionary* that are anagrams of the given *word*. You should read the *dictionary* only once. Two words are anagrams if they are a permutation of each other. The should ignore case differences and ignore non-letters (punctuation, whitespace, etc). For example, the following words and phrases are anagrams:

   - Elvis — lives
   - funeral — real fun
   - software — swear oft

   Hint: use a dict and sort the letters of each word.

3. [2 pts] Create a script that takes two command-line arguments:

   - an integer, *depth*;
   - the name of a directory, *dir*.

   The script should display on the standard output all the traversed files and directories below *dir*, up to the given *depth*. *dir* itself is at depth 0, so if the *depth* is one is 1, the script should output the immediate children of *dir*. Files and directories traversed should be displayed one per line. The traversal should be depth-first. To avoid possible infinite loops, symbolic links to directories should be printed but not followed. *Hint*: take a look at the `stat` module.

4. [5 pts] In this exercise, you will create a script that builds a dictionary that minimizes the time spent searching for a word. The script takes the name of a text file as a command-line argument. The script should ignore the case of the words, punctuation and numbers.

   The script arranges the words into a tree. Each node of the tree has a key represented by a string, one parent node, and a list of children. To keep the tree as small as possible, each new word is compared against a previous one, and the largest common prefix (if any) is stored into a common parent. Leaves keep a special key to remember read words. Children are ordered alphabetically.

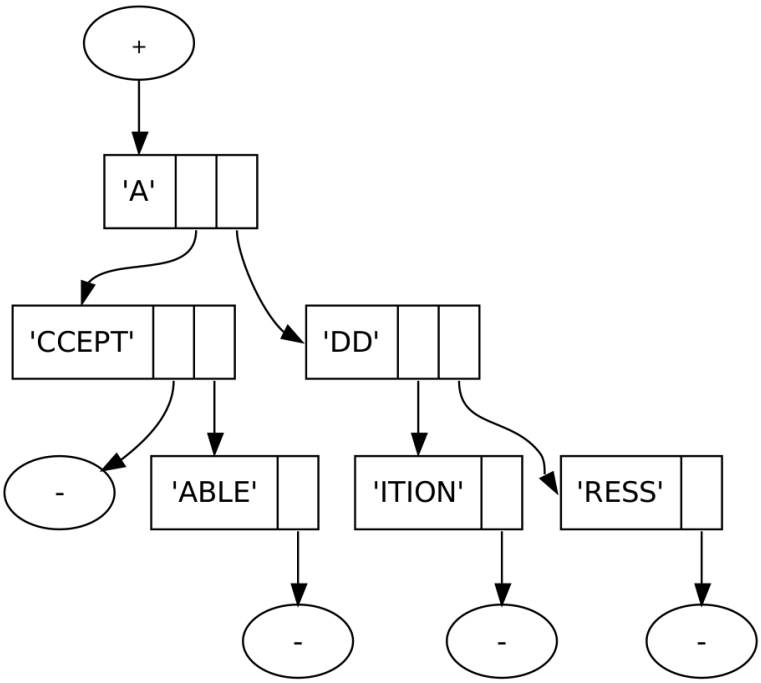
   You should implement this data structure by defining one or more classes.

   The script should on the standard output all the nodes of the tree traversing it in breadth first search.

   The output format is:

   ```
   key number-of-children
   ```

   As an example, if the file contains the words ACCEPT ADDITION ADDRESS ACCEPTABLE, the following tree will be constructed (the node denoted by + is the root of the tree):



   The script should print:

   ```
   A 2
   CCEPT 2
   DD 2
   ABLE 1
   ITION 1
   RESS 1
   ```