CSE 303, Spring 2005, Assignment 5A (Tries)
Due: Tuesday 17 May, 9:00AM

Last updated: May 11

Summary: You will write a C file that implements a "trie" data structure as described below. You will also write unit tests for the code you write. The sample solution, not including unit tests, is about 50 lines.

A trie is a tree-like data structure that maps "words" to information. At the root of the tree is (1) the information for the word that is zero-characters long and (2) an array containing $2^{8\texttt{sizeof(char)}}$ pointers to tries. Assume the field holding the array is `children`. The i^{th} element of the array points to a trie containing the information for words starting with the character that has integer value i. If there is no information for any words starting with i, the pointer should be NULL.

For example, if you knew that the word "cat" had information in the trie whose root was pointed to by `r`, you could use `r->children[(int)c]->children[(int)a]->children[(int)t]` to get the trie for words starting with "cat". In practice, you use a loop to descend one level of the tree at a time and you check for NULL at each level.

1. Include the following type definitions for tries (which we will also call entries):

   ```
   typedef struct Entry * entry_t;
   struct Entry {
     ...
   };
   ```

   The fields of `struct Entry` should include:

   - An array of pointers to entries. The array should have length 2^{8c} where c is `sizeof(char)`.
   - Two fields of type `int`, referred to below as `this_count` and `total_count`.

2. Implement the function `entry_t new_entry()`. It should return a pointer to a new heap-allocated `struct Entry`. The entries array should have all `NULL` entries and both counts should be initialized to 0.

3. Implement the function `get_count(entry_t w, void *env, char(*next_char)(void*))`. This function returns the `this_count` field associated with the word you get by calling `next_char(env)` until the character `EOF` is returned. For example, if the first time `next_char(env)` is called, `'h'` is returned, the second time `'i'` is returned, and the third time `EOF` is returned, then return the `count` field in the trie associated with the word `"hi"`, starting at root `w`. If there is no such field (because some prefix of the word leads to a `NULL` pointer in the trie), return 0 (which is the correct answer).

4. Implement the function `get_total_count(entry_t w, void *env, char(*next_char)(void*))`. This function is just like `get_count` except it returns the `total_count` field. You should use a shared helper function for this problem and the previous one.

5. Implement the function `void add_word(entry_t w, void *env, char(*next_char)(void*))`. This function uses `next_char` and `env` just like the previous two problems. It modifies the trie `w` to "count" the word being added as follows:

   - It increments the `total_count` field for every word that is a *prefix* of the word, including the word itself. (For example, the root's `total_count` field is always incremented.)
   - It increments the `count` field for the trie representing the word itself.
   - It allocates new entries and changes `NULL` pointers to point to them as necessary. For example, if the first word added is `"hid"`, three new entries will be created. If the next word added is `"hi"`, no new entries will be created. If the next word added is `"hit"`, one new will be created.

6. Write *unit tests* for the functions above. Include comments indicating what different tests accomplish and a `main` function that runs your tests.

Extra Credit: Change your trie implementation such that:

- If any character in any word is not from the alphabet (either case) or the hyphen (`-`) character, the program exits with an error.
- The arrays in the trie data structure have 53 entries instead of $2^{8\texttt{sizeof(char)}}$.
- You do not add any function with 53 cases. Rather you must write a function that computes the index of a character without having more than a few cases.

Assessment: Your solutions should be:

- Correct C programs that compile without warnings using `gcc -Wall`.
- In good style, including indentation and line breaks
- Of reasonable size

Turn-in Instructions:

- Follow the link on the course website and follow the instructions there.
- **Problems 1–5 should have solutions in `trie.h` and `trie.c`; your unit tests should be in `trie_test.c`.**
- We should be able to compile your program via `gcc trie.c trie_test.c`, assuming `trie.h` is in the same directory.