CSE 303, Spring 2007, Assignment 5A
Due: Monday 14 May, 9:00AM

Last updated: May 8 (a couple typos)

You will implement a *trie* data structure and unit-tests for it. The data structure maps English-letter strings to "unique indentifiers." Other group members will *independently* develop a "warehouse model" and an "order-filling algorithm." The sample solution is less than 70 lines, *not including* testing code, the header file, or the Makefile.

Requirements:

- Put your code in two files, `indentifier.c` and `identifier_test.c`. Both should include `identifier.h`. Write an appropriate Makefile.
- `identifier.h` (provided) should have just these prototypes plus typical header-file stuff:

```
struct IDSpace;
struct IDSpace* new_id_space();
void free_id_space(struct IDSpace*);

struct ID;
struct ID* string_to_id(struct IDSpace*,char *);
int id_num(struct ID*);
const char * id_string(struct ID*);
```

- In `identifier.c`, define these structs:

```
struct ID { int num; char* word; };
struct Trie { struct ID id; struct Trie * longer; };
struct IDSpace { int counter;  struct Trie * root; };
```

- You may assume that the numeric values for `'a'`, `'b'`, etc. are consecutive and increasing. So if `ch` is a lower-case English character, `ch-'a'` is between 0 and 25, inclusive.
- `new_id_space` should return a pointer to a new heap-allocated object with a counter of 0 and a pointer to an array of 26 `struct Trie` values where each of the 26 values has `id.num`, `id.word`, and `longer` fields that are 0 (or `NULL`).
- `string_to_id` may assume its first argument is not `NULL` and its second-argument points to a (`'\0'`-terminated) string holding only lower-case English letters and at least one letter.

  To lookup the right `struct ID` we follow the correct `struct Trie` pointers. For example, the `struct ID` for "cat" in `space` would be

  ```
  &((((((space->root['c'-'a']).longer)['a'-'a']).longer)['t'-'a']).id)
  ```

  However, while following pointers, we may encounter `NULL`, which of course must not be followed. If we encounter `NULL` for a pointer to an array we need to follow (i.e., for any letter of the word except the last one), set the pointer to a new array of 26 `struct Trie` values (with all fields 0) and continue (recognizing that all subsequent iterations will also encounter `NULL`).

  If this is the first time the word has been looked up (in this IDSpace), the `struct ID` will have a `NULL` word field. Before returning a pointer to the `struct ID`:

  - Set the word field to a *copy* of the word being looked up.
  - Increment the counter field of the IDSpace.
  - Set the num field to the counter field of the IDSpace.

  Hence all the strings ever looked up have "unique identifiers" starting from 1.

- `id_num` and `id_string` just return the appropriate fields of the object the argument points to.
- `free_id_space` deallocates *all* the space used by its argument, including all the space used by all the reachable tries (recursively) and all the reachable strings (recursively). (Hence any strings returned by `id_string` will be dangling pointers, but that is the caller's problem.)
- In `identifier_test.c` put unit tests for your code and a `main` that runs them.

Advice/Hints:

- Understand the data structure before you start coding. This may be the most difficult part.
- Keep `longer` fields `NULL` unless you cannot because a longer word has been added.
- When looking up a word, you need to keep track of the current position in the word and the current position in the data structure.
- The last letter of the word is handled differently because we return an ID rather than follow a pointer.

Assessment and turn-in:
Your solutions should be:

- Correct C code that compiles without warnings using `gcc -Wall` and does not have space leaks
- In good style, including indentation and line breaks
- Of reasonable size

Your test code should provide good *coverage*.

Use `turnin` for course cse303 and project hw5. If you use late-days, use project hw5late1 (for 1 late day) or hw5late2 (for 2).