Assignment 3: Spell checker

In this assignment you will develop a spell checker. There are two parts: First you will make a basic spell checker, then you will improve its performance using a trie. Both parts should be submitted to Themis before the same deadline.

Input-output behaviour

For both parts of this assignment we use the same input and output format. The input consists of the dictionary and the text to be checked, after each other.

- The dictionary consists of one word per line. Words consist of letters from '`a`' to '`z`', in lower or upper case. You can assume that no word is longer than 45 characters.
- You are dealing with a very strange dictionary: It is *not* sorted alphabetically.
- The dictionary and the text to be checked are separated by a single line with '`!`'.
- The text to be checked consists of letters from '`a`' to '`z`' in lower and upper case, digits from '`0`' to '`1`', spaces, newlines and the following symbols:

  `.` `,` `'` `/` `-` `"` `:` `;` `?` `!` `(` `)`

- Digits, spaces, newlines and any of the above symbols denote word boundaries.

Your program should output all words that occur in the text but not in the dictionary (one per line), followed by the number of unknown words. Your program should ignore lower/upper case differences.

Example

input	corresponding **output**

`one`
`two`
`three`
`and`
`four`
`!`
`hello?`
`one,twwo; three!`
`And - "for".`
	`hello`
	`twwo`
	`for`
	`3`

Part 1 (2 points)

1. Download `3spellchecker.zip` from Themis and unpack it.
2. The file `speller.c` contains a basic spellchecker. The code compiles, but the resulting program ignores some of the rules above. Your first task is to repair the program.
3. The file `exampleInput.txt` contains the example above. The `Makefile` includes some useful commands. Use '`make example`' to run your program and '`make check-example`' to compare the output of your program against '`exampleOutput.txt`'. You can also use '`make debug-example`' to run your program with valgrind.
4. For this part you only have to upload `speller.c` when submitting to Themis. The files `dictionary.h` and `dictionary.c` are included automatically.

Part 2 (3 points)

1. After you have a solution for part 1, run '`make time-largeExample`'. As you can see in the `Makefile` this will run `time ./speller < large_exampleInput.txt`. Probably your program will take quite long for this example with a real English dictionary.
2. Your new goal is thus to make your spellchecker faster, *using a trie*.
3. You should now modify `dictionary.h` and `dictionary.c`.
   Ideally you should not (have to) change anything in `speller.c`.
4. For this part no files are included automatically by Themis, so you should upload `speller.c`, `dictionary.c` and `dictionary.h`. In fact, you can upload your solution from part 1 and the unmodified `dictionary.c` and `dictionary.h` to part 2, but it will not be fast enough to pass the tests.

Hint

You may use this definition for a standard trie:

```
#include <stdbool.h>

typedef struct TrieNode* Trie;

struct TrieNode {
  bool endNode;
  Trie children[26];
};
```

Report (5 points)

For this assignment you should also write a programming report. You can find a template for this on Nestor. Please follow all guidelines from Appendix E of the lecture notes and submit your report as a single PDF file on Themis. Note: Please do not include long dictionaries in your report PDF.

Further Ideas (just for fun, no bonus points)

- Extend your program with command line arguments to read the dictionary and the text to be checked from two separate text files: `a.out -dictionary english.txt -text mytext.txt`.
- Write a function to print the trie. (This might also be useful for debugging.)
- Store the trie in a file so that it can be reused quickly. (This can be a lot of work.)
- Look up and learn how to output colored text in the terminal. Instead of only printing unknown words, output the whole text and highlight the unknown words.