Assignment 3: Indexing words

Date: February 21st 2019
Deadline: March 2nd 2019 23:59

Objectives

You will need to implement a resizing array and hash table API, and use those to create a complete index of all words in a book or file.

Requirements

Your program must be named `lookup` and its basic operation is as follows:

- It takes a single command-line argument, the name of the file to build the index for, and builds the corresponding word index table.
- It then reads lines from standard input, treating each line a new word, and searches for the word in the index table. *If the word is in the index table, it will list all the line numbers this word occurred on in the original file.*
- Alternatively, it takes one additional argument `-t` and performs a series of timing tests for different parameter sets when building the table.

For the input file, all non-alphabetical characters should be treated as **spacing** and all letters should be converted to lowercase, before being stored in the table. *The words read on standard input should be converted in the same way, to ensure they match the format in the table.*

You must submit your work as a tarball. The command `make tarball` will create a a tarball called `hash_table_submit.tar.gz` containing the relevant files.

Getting started

1. Start by writing the code for `array.c`, which is the datastructure that will contain all the line numbers for a specific word. Some words might only occur once, while common words will occur many times, so your array should be able to scale appropriately, depending on the number of elements inserted. The command `make check` will run some basic test, *but not all functionality of your array will be tested by these checks.*
2. Next start writing the code for `hash_table.c`. The code here will depend on your `array.c` working, as the table should return the complete array of line numbers for a given word when it is looked up. Be sure to add some functions of your own, to better divide up the functionality of the code. The command `make check` will also run several tests on your hash table, *but again the provided tests are not a complete check.*
3. Write the code in `main.c`. You will only need to complete 2 functions there, as the actual body of the `main` has already been provided. Make sure convert the input as specified by the requirements above and study the output format in the example below. The last test `make check` runs will try some simple words on your complete program.
4. Add more hash functions to `hash_func.c` (and modify `hash_func.h` accordingly). You may write your own, or search for existing solutions online. If you use existing solutions, attribute the original author and provide a link to the source.
5. Study the provided function `timed_construction`, which builds the table many times with different parameters. Add your own hash functions to the parameter set and expand the other parameter options as you think would be sensible. Rerun the timing tests using the `-t` option, several books have been included for you to test with. Include your best parameter set in the default `#define` parameters at the top of the file.

Output format

Once the table is built, the program should read words from standard input and print the line numbers for each word. Every line should be considered a separate word, converted to lowercase, with any non-alphabetical character treated as spacing. Only the first word of every input line should be processed, so any words after the first should be ignored.

The program should the print the converted lowercase word and on every next line a `*` followed by the line number on which the word occurred. Finally the program should print an empty line. So for example, when testing with the command `./lookup origin-of-species-ascii.txt`, the with the input `Creature's.`, the output should be:

```
creature
* 3863
* 5878
* 7797
* 11876
* 13333
* 13627
* 13873
<blank line>
```

If a word occurs several times on the same line, the line number should be included multiple times as well, each time on a new line.

If a word does not occur in the text at all, your program should just output a blank line. So for example, with the input `Esoteric` in *Origin of Species* the output should just be:

```
esoteric
<blank line>
```

Grading

Your grade starts from 0, and the following tests determine your grade:

- +1pt if you have submitted an archive in the right format and your source code builds without errors and you have modified `array.c` or `hash_table.c` in any way.
- +1pt if your resizing array implementation works correctly
- +2pt if your hash table correctly supports basic inserts and lookup of integers
- +1pt if your hash table correctly resizes when the maximum load factor is exceeded
- +1pt if your hash table correctly extends the existing value array if the key was already present
- +1pt if your hash table correctly deletes keys
- +1pt if your program returns the correct line numbers when given a test file and test input
- +1pt if your program correctly handles non-alphabetical characters and uppercase characters.
- +1pt if you have added hash functions and included your best parameter set in the default `#define` parameters for the table in `main.c`
- -1pt if your code produces any warnings using the flags `-Wpedantic -Wall -Wextra` when compiling.
- -1pt if enabling the `address sanitizer` or running `valgrind` reports errors while running your code. *Note that you cannot test both of these at the same time, so disable ASAN in the Makefile when testing with valgrind.*