Assignment 3: Indexing words

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 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 proceeds to read lines from standard input and gets the first word ignoring the rest of each line. It then searches for that 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 data structure 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 these checks will not test all functionality of your array.*
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. Be sure to add some functions of your own to better divide up the code's functionality. The command `make check` will also run several tests on your hash table, *but again the provided tests are incomplete.*
3. Write the code in `main.c`. We already provide the implementation of the `main()` function. You will need to complete the function that creates the hash table and the function that performs the lookup in the hash table. Make sure to convert the input as specified by the requirements above and study the output format in the example below. The last test that `make check` runs looks up a couple of words in a large text file to test your complete program.
4. Add more hash functions to `hash_func.c` (and modify `hash_func.h` accordingly). You can write your own hash functions 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. We have included several books 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 hash table is built, the program should read words from standard input and print the line numbers for each word. Only the first word of every input line on stdin should be processed, any words after the first word should be ignored. The first word must be converted to lowercase, with any non-alphabetical character treated as spacing.

The program prints the converted lowercase word, and on every next line, a * followed by the line number on which that word occurred. The program should print an empty line between word entries. Starting with a simple example you should expect the following output:

```
$ nl test.txt   # 'nl' adds line numbers to print a file.
     1      hello
     2      how are you
     3      today. I am fine. Thank
     4      you. hi
     5      Hi hi

$ cat words.txt
you
I
hi

$ ./lookup test.txt  < words.txt
you
* 2
* 4

i
* 3

hi
* 4
* 5
* 5

$
```

Note that the $ represents the bash prompt and is not part of the output that your program should produce. It is shown here to indicate that last line of your output should be a blank line.

As a more complex test we type the command `./lookup origin-of-species-ascii.txt`, with the input `Creature's`, and we should see the following:

```
$ echo "Creature's" | ./lookup origin-of-species-ascii.txt
creature
* 3863
* 5878
* 7797
* 11876
* 13333
* 13627
* 13873

$
```

If a word occurs several times on the same line, the line number is printed multiple times, 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* it should look like this:

```
$ echo "esoteric" | ./lookup origin-of-species-ascii.txt
esoteric

$
```

Grading

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

- +2pt if you have submitted an archive in the right format, your source code builds without errors and you have clearly made a real effort to implement `array.c` and `hash_table.c`.
- +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.
- +0.5pt if your hash table correctly deletes keys.
- +0.5pt if your program returns the correct line numbers when given a test file and test input.
- +0.5pt if your program correctly handles non-alphabetical characters and uppercase characters.
- +0.5pt 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 implementation has the correct style and the correct complexity.
- -1pt if your code produces any warnings using the flags `-Wpedantic -Wall -Wextra` when compiling.