Assignment 1

Student: Joel Castillo (jc5383)
Repo: https://github.com/joelbcastillo/cs9163-spell-checker

Application Functionality

This application is a shell program that checks for mispelled words in a provided text file.

```
Usage:
    ./spell_check <file_to_check> <wordlist>
        file_to_check - Text file that needs to be checked for mispelled words
        wordlist - Dictionary of words to check the file_to_check against.
```

The main function checks to see if both files (`file_to_check` and `wordlist`) exist and are valid.

Afterwords, the function load_dictionary is called. This function loads the dictionary file (`wordlist`) into a `hashmap_t` data structure for fast lookup. This is done by splitting the file into lines and loading each line into the `hashmap_t` utilizing a provided hash function.

Following, the application then loads the text file to check against the wordlist and splits it into lines. Each line is then split into words (delimited by whitespace). Each word is then checked agains the hashmap (by running the hash function to find the bucket and then traversing the singly linked list at that location.)

In order to avoid issues caused by uppercase / lowercase letters and punctuation, each word is converted to lowercase and stripped of punctuation before being stored in the hashmap and before being checked against the hashmap.

Each mispelled word is added to an array (passed into the check_words function) and a total count of mispelled words is returned.

Main will print out the mispelled words and the total account before shutting down the program.

Valgrind Output

```
$ docker run -ti -v $PWD:/test spell-check:0.1 bash -c "cd /test/; make cleanall; make; val
grind --leak-check=full --show-leak-kinds=all ./spell_check test.txt test_wordlist.txt"
rm dictionary.o spell.o main.o test_main.o check_spell.o dictionary.h.gch
rm: cannot remove 'test_main.o': No such file or directory
rm: cannot remove 'check_spell.o': No such file or directory
Makefile:36: recipe for target 'clean' failed
```

```
make: [clean] Error 1 (ignored)
rm spell_check
gcc -Wall -c dictionary.c dictionary.h
gcc -Wall -c spell.c
gcc -Wall -c main.c
gcc -Wall -g -o spell_check dictionary.o spell.o main.o
==23== Memcheck, a memory error detector
==23== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==23== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==23== Command: ./spell_check test.txt test_wordlist.txt
==23==
==23== Conditional jump or move depends on uninitialised value(s)
==23==    at 0x4C30F78: strlen (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x400CED: load_dictionary (in /test/spell_check)
==23==    by 0x4010A2: main (in /test/spell_check)
==23==
==23== Conditional jump or move depends on uninitialised value(s)
==23==    at 0x4C30F78: strlen (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x4009F4: hash_function (in /test/spell_check)
==23==    by 0x400D0F: load_dictionary (in /test/spell_check)
==23==    by 0x4010A2: main (in /test/spell_check)
==23==
==23== Conditional jump or move depends on uninitialised value(s)
==23==    at 0x400D2D: load_dictionary (in /test/spell_check)
==23==    by 0x4010A2: main (in /test/spell_check)
==23==
strlen(hello): 5
==23== Conditional jump or move depends on uninitialised value(s)
==23==    at 0x4C30F78: strlen (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x4009F4: hash_function (in /test/spell_check)
==23==    by 0x400DFB: check_word (in /test/spell_check)
==23==    by 0x400F12: check_words (in /test/spell_check)
==23==    by 0x401123: main (in /test/spell_check)
==23==
Bucket: 532
==23== Conditional jump or move depends on uninitialised value(s)
==23==    at 0x400E79: check_word (in /test/spell_check)
==23==    by 0x400F12: check_words (in /test/spell_check)
==23==    by 0x401123: main (in /test/spell_check)
==23==
misspelled words count: 1
misspelled_words:
==23== Invalid read of size 1
==23==    at 0x4E88CC0: vfprintf (vfprintf.c:1632)
==23==    by 0x4E8F898: printf (printf.c:33)
==23==    by 0x401178: main (in /test/spell_check)
==23==  Address 0x5206c05 is 0 bytes after a block of size 5 alloc'd
==23==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
```

```
==23==    by 0x400F2D: check_words (in /test/spell_check)
==23==    by 0x401123: main (in /test/spell_check)
==23==
        hello
==23==
==23== HEAP SUMMARY:
==23==     in use at exit: 825 bytes in 6 blocks
==23==   total heap usage: 10 allocs, 4 frees, 10,593 bytes allocated
==23==
==23== 5 bytes in 1 blocks are definitely lost in loss record 1 of 6
==23==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x400F2D: check_words (in /test/spell_check)
==23==    by 0x401123: main (in /test/spell_check)
==23==
==23== 46 bytes in 1 blocks are definitely lost in loss record 2 of 6
==23==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x400B31: remove_punctuation_and_uppercase_characters (in /test/spell_check)
==23==    by 0x400CDD: load_dictionary (in /test/spell_check)
==23==    by 0x4010A2: main (in /test/spell_check)
==23==
==23== 46 bytes in 1 blocks are definitely lost in loss record 3 of 6
==23==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x400B31: remove_punctuation_and_uppercase_characters (in /test/spell_check)
==23==    by 0x400DEB: check_word (in /test/spell_check)
==23==    by 0x400F12: check_words (in /test/spell_check)
==23==    by 0x401123: main (in /test/spell_check)
==23==
==23== 56 bytes in 1 blocks are definitely lost in loss record 4 of 6
==23==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x400CB9: load_dictionary (in /test/spell_check)
==23==    by 0x4010A2: main (in /test/spell_check)
==23==
==23== 120 bytes in 1 blocks are possibly lost in loss record 5 of 6
==23==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x4EA89E7: getdelim (iogetdelim.c:62)
==23==    by 0x400FB5: check_words (in /test/spell_check)
==23==    by 0x401123: main (in /test/spell_check)
==23==
==23== 552 bytes in 1 blocks are still reachable in loss record 6 of 6
==23==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==23==    by 0x4EA7CDC: __fopen_internal (iofopen.c:69)
==23==    by 0x4010DE: main (in /test/spell_check)
==23==
==23== LEAK SUMMARY:
==23==    definitely lost: 153 bytes in 4 blocks
==23==    indirectly lost: 0 bytes in 0 blocks
==23==      possibly lost: 120 bytes in 1 blocks
==23==    still reachable: 552 bytes in 1 blocks
```

```
==23==         suppressed: 0 bytes in 0 blocks
==23==
==23== For counts of detected and suppressed errors, rerun with: -v
==23== Use --track-origins=yes to see where uninitialised values come from
==23== ERROR SUMMARY: 11 errors from 11 contexts (suppressed: 0 from 0)
```

It looks like the valgrind output is mostly complaining about unitialized variables. From what I can tell, the issues are mostly related to heap objects that are not being initalized before usage. However, I believe all of these are false positives that may be caused by initializations in a different part of the code.

There seems to be an intermittent (possibly resolved) issue with the hash function that generates a bucket that is incorrect for the word. I'm not sure how the issue was resolved or if it is being masked by another issue.

In addition, I was unable to get the code to pass the gradescope tests. I ran out of time to debug the issues.

Tests

I added an additional test to see how the function for `remove_punctuation_and_uppercase_characters` would work. Unfortunately, I was unable to get this function working properly in the test cases most likely due to multiple imports from importing spell.c.

Fuzzing

Fuzzing failed spectacularly: ```

```
                  american fuzzy lop 2.52b (spell_check)
```

┌── process timing ──┬── overall results ──────┐ | run time : 0 days, 0 hrs, 5 min, 8 sec | cycles done : 300 | | last new path : n/a (non-instrumented mode) | total paths : 1 | | last uniq crash : 0 days, 0 hrs, 0 min, 0 sec | uniq crashes : 1641 | | last uniq hang : 0 days, 0 hrs, 0 min, 1 sec | uniq hangs : 19 | ├── cycle progress ─────────────────────────────┬── map coverage ─┴──────────────────────────────────┤ | now processing : 0* (0.00%) | map density : 0.00% / 0.00% | | paths timed out : 0 (0.00%) | count coverage : 0.00 bits/tuple | ├── stage progress ─────────────────────────────┼── findings in depth ───────────────────────────┤ | now trying : havoc | favored paths : 0 (0.00%) | | stage execs : 64/256 (25.00%) | new edges on : 0 (0.00%) | | total execs : 78.5k | total crashes : 1641 (1641 unique) | | exec speed : 250.6/sec | total tmouts : 20 (20 unique) | ├── fuzzing strategy yields ─────────────────────┴──────────────────────────────┬── path geometry ───────────┤ | bit flips : 0/48, 3/47, 0/45 | levels : 1 | | byte flips : 0/6, 0/5, 0/3 | pending : 0 | | arithmetics : 1/335, 0/25, 0/0 | pend fav : 0 | | known ints : 0/33, 0/140, 0/132 | own finds : 0 | | dictionary : 0/0, 0/0, 0/0 | imported : n/a | | havoc : 1636/77.6k, 0/0 | stability : n/a | | trim : n/a, 0.00%
├──┘

```
└──────────────────────────────────────────────────────────────────────────────┘
```

[cpu: 43%] [-] PROGRAM ABORT : Unable to execute target application Location : save*if*interesting(), afl-fuzz.c:3281

```
I believe most of these errors are caused by either:
    - Incorrect usage of memory (unlikely due to the valgrind output).
    - Incorrect handling of Unicode characters.



## Conclusions
There are a number of improvements I would make to this application, most of which would re
quire a rewrite.

1) I would refactor to break out each function into smaller, more testable components. This
 would allow me to write actual unit tests for each function and ensure a higher test cover
age percentage.
2) While not a written requirement, the code should be able to fail gracefully when encount
ering non-ascii characters. At this point this is most likely the main culprit for the fuzz
ing failure.
3) TDD: I did not follow TDD practices when building this application. I believe it woudl h
ave been helpful to write out the test cases and then implement the solutions. It would hav
e allowed me to find edge cases and resolve many issues that prevent this application from 
working.
4) Security - The memory management in this program is not very well thought out. As a resu
lt, there are definitly memory access vulnerabilites throughout the code. A large part of t
he rewrite would focus on memory safety first (with regards to application functionality) t
o ensure the code is safe. Unix shell programs should be written in such a way that they fa
il gracefully when there is an issue, instead of possibly blowing up data.
```