Part 2:

1. The program takes in 2 command line arguments: one for the file to check for spelling, and another that contains a list of words. The load_dictionary method creates a hashtable for fast lookup. The check_word method is used in the check_words method in order to determine whether or not a given word can be found in the hashtable. The check_words method splits up the input file by word and calls the check_word method for each of the resulting words.

2. The output of Valgrind shows possible memory leaks for when the malloc function is called but the memory allocated has not been freed.

3. I suspect there is some missing edge cases checking with certain ASCII characters. There is also a strong assumption that the list of words used for checking spelling is formatted in a specific way, one word per line.

4. These bugs might occur because the user may not have paid attention to their input files.

5. I have added a check for digits to count numbers which are purely numerical. I have also checked for different types of whitespace that can throw off the spell checking methods.

Part 3:

1. One particular bug that is found from my manual testing is that if the input file list of words to be put in the hashtable is formatted incorrectly, that can cause the program to misbehave.

2. I fuzzed for one day. AFL ran for 2 cycles (yellow) and there were 22 unique crashes. The last unique crash was found 19 hours ago. I have found a bug where the value of the character is not in the expected range of ASCII characters (0 to 127) causes the program to seg-fault.

3. I fixed these bugs mostly with if statements to check for edge cases/incorrect input, or I fixed a segmentation fault when doing part 2 of this assignment with the help of the gdb debugger. I have fixed the bug mentioned in answer 2 found by fuzzing by adding a check in the check_word function to see whether or not the word being spell checked contains any non-ascii characters. This bug fix solved all 22 unique crashes.

4. These bugs can be avoided in the future by assuming that any user input that can be controlled will cause bugs and will be used against your program.