Debugging with Valgrind

On the course homepage, you'll find a source file, wordlist.c, a text input file, input.txt, and an expected output file. You can also download these with the following curl commands:

```
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise15/wordlist.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise15/input.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise15/expected.txt
```

This program expects a filename as a command-line argument. It reads all the words from the file, storing them in nodes in linked list. For this program, a word is a sequence of up to 8 characters, consisting of letters or the apostrophe character. Words are converted to lower-case as they're stored, and words longer than 8 characters are truncated to just the first 8. After building the linked list, it sorts it using mergesort, then prints out all distinct words from the input file, reporting the number of occurrences of each word.

When it's working, you should be able to run the program on the sample input file as follows:

```
$ ./wordlist input.txt
a (7)
aching (2)
and (16)
any (1)
around (6)
ask (1)
been (4)
… several lines omitted ...
what's (2)
wouldn't (1)
you (41)
you're (3)
your (2)
```

This program is already implemented for you, but it contains some bugs. You're going to use valgrind to help you find and correct the bugs.

Debugging with Valgrind

This program has some bugs that you need to fix. You're going to fix them with help from valgrind. Compile the program the way you normally would. Be sure to use the -g compiler option, so valgrind can give you more detailed information about where it catches your program doing something wrong.

Then, try debugging the program with help from valgrind. You should be able to find and fix the following bugs. Be sure to re-compile the program after fixing each bug.

- Run the program in valgrind, using the default tool (memcheck) and giving it the input.txt file as a command-line argument:

  ```
  valgrind ./wordlist input.txt
  ```

  This will probably generate a lot of output, from the program and from valgrind. Scroll back and have a look at the first error reported by valgrind. It should be a complaint about an invalid write. The program is writing past the end of a dynamically allocated block of memory. This means either the block of memory being allocated isn't big enough for what the program is trying to store, or there's a bug in the program that's causing it to try to write more to that block than it should.

  In this case, the problem is that the dynamically allocated block of memory isn't big enough. Look at the definition of the Node structure. Is the word array large enough to hold an 8-character string? Almost, but not quite. Fix the definition of this structure so it can store a string of up to 8 characters.

- After fixing this bug and re-compiling, try the program again with valgrind. Now, its first complaint should be about a conditional jump or move depending on an uninitialized value. Look at the line of source code it's complaining about and find any uninitialized variables that might be accessed by this line. Correct this error and re-compile the program.

- Run the program again in valgrind. There should still be a complaint about an invalid read. This is the kind of error we talked about when we were looking at linked list code in class. You can't access the contents of a block of memory after you free it.

  Change the code so that it frees the list of words without using nodes that have already been freed.

- After these corrections, the program still has a file leak. Add the --track-fds=yes option when you run valgrind, and it will report all leaked files at program termination. You don't have to worry about the three standard streams, standard input, standard output and standard error, but any additional files should be closed before the program exits.

  Valgrind will give you a stack trace to tell you where you open any files that are subsequently leaked. Look at the valgrind output and see where the leaked file is being opened. Add code to close this file before returning from the function that opened it.

- After fixing these errors, you should still have one block of leaked memory. Run valgrind with the --leak-check=full option, and it will show where this block of memory is allocated`. Consider what happens after the program has read all the words from the input file, and add code to free this block of memory before returning to the main function.

After fixing these bugs, your program should produce the correct output, and it should run in valgrind without any error messages.

Submitting Your Work

Once you've fixed all the bugs, submit your modified source file to the exercise_15 assignment in Moodle. We'll test it by making sure it produces the right output and valgrind no longer has anything to complain about.

Don't forget to make sure your output is right. In the past, some students have lost most (or all) of their points for this exercise because they concentrated on fixing the memory errors without checking to make sure the program's output is right. The grading script for this problem first checks to make sure you have the right output. If you do, it goes on to try your program with valgrind.