Homework 3 and Gradescope

We have switched to Gradescope for submitting your homework and for grading.

Because the grading of programs compares your outputs and output files with ones on Gradescope, even small differences in output can cause you to not get points. So here is a guide to the output for Homework 3.

Problem 3

For problem 3, here are the formats for the non-error outputs. All are to be written to the standard output. If nothing goes wrong, your program needs to return an exit status code of 0.

- When a word is found in the word list, use a *printf* of the following form:

  ```
  printf("'%s' is word %d in the list\n", w, n);
  ```

  where `w` is the word as a string and `n` the number of comparisons made (which is 1 more than the index, assuming you start your counting at 0 rather than 1).
- When a word is not found in the word list, use a *printf* of the following form:

  ```
  printf("'%s' is not in the word list\n", w);
  ```

  where `w` is the word as a string.
- When you are printing out the number of words looked for, the number of words checked, and the average (mean), use a *printf* of the following form:

  ```
  printf("Checked %d words while looking for %d words; average is %.2f\n", nc, nw,
              ((double)  nc) / ((double) mw));
  ```

  where `nc` is the number of words checked and `nw` the number of words searched for. ***Note the double in the division — if you omit it, or use floats, you will get at least one wrong answer.***

And here are the error messages. All are to be written to the standard error. In all cases, your program needs to return an exit status code of 1.

- If *malloc*() fail, use *perror*(3) to print the message. If, for example, the return value of *malloc* is stored in `p`, then you would do the test as:

  ```
  if (p == NULL)
          perror(s);
          ...
  ```

  where `p` holds the return value of *malloc* and `s` is the string you are saving in the allocated space.
- If there are not exactly 3 command-line arguments (the program name, the word list, and the file of list of words to find), use an *fprintf* of the following form:

  ```
  fprintf(stderr, "Usage: lookup [wordlist] [words]\n");
  ```

- If opening either file fails, use *perror* to print an error message, like this:

  ```
  perror(fname);
  ```

  where `fname` is the name of the file that you were trying to open.
- If the file containing the list of words to be searched is empty, use an *fprintf* of the following form:

  ```
  fprintf(stderr, "No words to search!\n");
  ```

- If the file contaning the list of words to search for is empty, use an *fprintf* of the following form:

  ```
  fprintf(stderr, "No words to find!\n");
  ```

Problem 4

For problem 4, here is the format for the non-error outputs. All are to be written to the standard output. If nothing goes wrong, your program is to return an exit status code of 0.

- The first row of the output will have 2 more "–" (hyphen) characters than there are rows.
- The second through next-to-last rows will begin with a "|" (vertical bar), then have the row from the file, and then conclude with a "|" (vertical bar).
- The last row of the output will have 2 more "–" (hyphen) characters than there are rows.

The result should be the contents of the file surrounded by the border. There cannot be any other lines or characters in the output.

Here are the error messages. All are to be written to the standard error. In all cases, your program needs to return an exit status code of 1.

- If *malloc*() fail, use *perror*(3) to print the message. If, for example, the return value of *malloc* is stored in p, then you would do the test as:

```
if (p == NULL)
        perror(s);
        ...
```

  where p holds the return value of *malloc* and s is the string you are saving in the allocated space.
- If there are not exactly 2 command-line arguments (the program name and the file containing the pattern for the board), use an *fprintf* of the following form:

```
fprintf(stderr, "Usage: %s board_pattern\n", argv[0]);
```

- If opening the file fails, use *perror* to print an error message, like this:

```
perror(fname);
```

  where fname is the name of the file that you were trying to open.
- If a line in the file is of the wrong length, use an *fprintf* of the following form:

```
fprintf(stderr, "%s: line %d is wrong length\n", fname, lineno);
```

  where variablefname is the name of the file and lineno is the line number of the bad line.
- If a line in the file contains an invalid character, use an *fprintf* of the following form:

```
fprintf(stderr, "%s: bad character '%c' in line%d\n", fname, ch, lineno);
```

  where variablefname is the name of the file, ch is theinvalid character, and lineno is the line number of the bad line.

Problem 5

For problem 5, the output should be the contents of the standard input, with lines in reverse order. Do not add any other characters!

As for error messages, you can see them in the source code. Please do not alter them!