CSE 303, Spring 2007, Assignment 4
Due: Friday 4 May, 9:00AM

Last updated: April 23

(As usual, line counts provide a rough sense of how much work is required; they are not requirements.)

Overview: You will complete a Boggle program that is more useful than the one in homework 3. The user inputs commands that make the program do things like create a new grid, add a word to the found-words (if it is in the grid), print the words found so far in the current grid, print the current grid, etc. Instead of reading grids from a file, grids are filled in by choosing letters randomly, but using probabilities that favor common letters. These probabilites are computed by another program (written by the instructor; called `probability_maker`) by reading a (presumably) representative text file.
Much of the code has been written for you and you need not add any files. What is left to do is:

- Write a Makefile. This is interesting because `probability_maker` creates a C file used by the `boggle` program (which you are completing).
- Change a function in `grid_utilities.c` to uses the correct probabilities to choose a letter for a grid position.
- Write most of `string_set_array.c`, which implements a set of strings using "extensible arrays" (explained below).
- Write most of `boggle_main.c` to process the user's commands (using the grid-utilities and the string-set provided by other files).

The code provided to you (on the course web-site) compiles but does not yet "do anything." The file `buildScript` does everything necessary to compile `boggle` (including compile and run `probability_maker`), so it is not necessary to do problem (1) first.

Problems:

1. Complete the `Makefile` for this project. The provided file needs 6–7 more targets, 2 of which are for the programs `boggle` and `probability_maker`. These programs should be built out of only the `.o` files they need. As usual:

   - A `.o` should be remade if the corresponding `.c` file or any of the non-library headers it uses change.
   - A program should be remade if any of its `.o` files change.

   In addition, the file that `probability_maker` creates (`letter_probabilities.c`) should be remade if the `probability_maker` program changes or `/usr/share/dict/words` changes. (`probability_maker` takes one file and produces a C file defining an array described in the next problem. We choose to pass the Linux words file, so if this file changes, the created C file will probably change too.)

   As noted above, it is not necessary to do this problem first.

2. Change the implementation of `get_weighted_letter` in `grid_utilities.c` to use the array `letter_probabilities` as follows. The array has 26 entries and the number in position i is the fraction of the time that we want to pick one of the first $i+1$ letters of the alphabet. (So, as an extreme example, the last entry is 1.0 because we always pick some letter.) The caller to `get_weighted_letter` passes a (pseudo)random number evenly distributed between 0 and 1 (you need not understand how this is done). So all you have to do is return the i^{th} letter of the alphabet when the argument is between the $(i-1)^{th}$ and i^{th} entries in the array. Sample solution is 7 lines.

3. In `string_set_array.c`, define the functions declared in `string_set.h` using the provided definition of `struct StringSet`. Sample solution is 44 lines (4 blank). These functions should behave as follows:

   - `new_string_set` heap-allocates a new set with 0 elements but with an array large enough to hold `INITIAL_SET_SIZE` elements.

- `string_set_member` returns 0 unless one of the first argument's elements is a string equal to the second argument. Use the `strcmp` library function.
- `string_set_add` does nothing if the second argument is already a member of the first. Else it adds a *copy* of the second argument (use the `strdup` library function) to the set. If there is no room in the array, first make a new array that is *twice as large*, copy over all the pointers to the string, and (to avoid leaking space), deallocate the too-small array.
- `string_set_foreach` takes a function-pointer and calls the pointed-to-function on each string in the first argument (so if there are n strings, it calls the function n times).
- `free_string_set` deallocates *all* the space used for a set, including the strings, the array, and the struct. (It is okay to deallocate the strings because `string_set_add` uses copies; see above.)

4. Implement `run_boggle` in `boggle_main.c` to behave as follows. Sample solution is 70 lines including one other very short function. You should use several other functions provided to you.

   (a) First, make a grid of size `INITIAL_GRID_SIZE` and an empty set of strings for "words found so far." Print the possible commands and this initial grid.

   (b) Then enter an infinite loop that repeatedly reads a line from stdin. Use the `getline` function. (See the man page; the `probability_maker` code also has an example use.) If the line has only a `\n` immediately read another line (i.e., do nothing), else there are different cases depending on the first letter of the line:

   (c) If the first letter is `'p'`, print the current grid.

   (d) If the first letter is `'a'`, there should be one space after the `'a'` and then one or more English letters. If not, print an appropriate message. Else convert the word to all lower-case and see if the result is in the current grid. If not, print an appropriate message. Else see if the word is already in the "words found so far." If so, print an appropriate message. Else add the word to the "words found so far."

   (e) If the first letter is `'n'`, there should be one space after the `'n'` and then the rest of the line should be something the standard-library function `atoi` can convert to a positive number. If not, print an appropriate message. Else, (1) replace the current grid with a new grid with the number being the size, (2) replace the "words found so far" with a new empty set, and (3) print the new current grid. Do not leak space.

   (f) If the first letter is `'w'`, print the "words found so far," one on a line. Hint: Use `string_set_foreach`, passing it a function you write that prints a string.

   (g) If the first letter is `'q'`, return from `run_boggle`, after preventing *all* space leaks.

   (h) If the first letter is `'?'`, print the possible commands.

   (i) For any other first letter, print a message like: `bad command; enter '?' for usage`.

5. **Extra Credit:** Do one or more of the following.

   (a) Change `probability_maker` and the grid-printing code so that `qu` is treated as one-letter (ignore any `q` that is not followed by a `u`). In printing a grid, any column that has a `qu` should put an extra space after all other letters in the column so that subsequent columns still line up.

   (b) In `string_set_list.c`, implement string-sets using linked lists. Update the Makefile to choose between the two implementations depending on whether the make variable `USE_LISTS` is set.

   (c) Add a `t` command that takes a number and after that many seconds have elapsed no more words can be added until a new grid is created. Change the `a` command to report if time has expired. After the `t` command has been used once on the current grid, any subsequent `t` commands do not need a number and they print how many seconds are remaining.

   (d) Change `probability_maker` to have an option `-a` to, if `letter_probabilities.c` already exists, add the file's contents to the results rather than replacing the results. (Hint: Put the old total number of letters in a C comment.) Your solution should work for up to 2^{64} total letters.

Assessment: Your solutions should be:

- Correct C programs that compile without warnings using `gcc -Wall` and do not leak memory.
- In good style, including indentation and line breaks
- Of reasonable size

Turn-in: Use the `turnin` command (man turnin) for course cse303 and project hw4. Given the large number of files, it is easiest to run turn-in on a whole directory *after* running make clean and removing any temporary files. For example:

```
turnin -ccse303 -phw4 myhw4_code
```

If you use one late-day (see the syllabus) use the project hw4late1 instead of hw4 and similarly hw4late2 for two late days. If you do the extra credit, turn in extra files as necessary *and turn in a text file titled* `extra_credit` *that explains what extra credit you did.*