Fall 2018 CS 31

Project 3 FAQ

1. **How do I begin?**

   First, the announcements dated 10/19/18 and 10/21/18 are loaded with resources to give you the tools you need. Read all the writeups. Even if you don't see why they're relevant at first, as you progress in the project you'll be thinking at various points "Now I need to …. Wait! Didn't I read something related to that?", and can read the appropriate writeup again, this time with greater understanding. Do the Project 3 warmup to make sure you understand the mechanics of using functions.

   Again, as in Project 2, one secret of success is to develop incrementally. Work on the easier function first: `hasProperSyntax`. Make a tremendously simplifying assumption: A polldata string is syntactically correct if it contains no state forecasts at all. Get the function working with that simplification: it should return true for the empty string, and false otherwise.

   This seems ridiculously trivial, but it gets you past an important hurdle. You now know how to declare a function, implement it, and write a main routine that tests it. Further work then becomes merely adjusting the implementation and the test cases.

   Now go with a simplification that is a little less tremendous: A poll data string is syntactically correct if it contains no state forecasts at all, or exactly one state forecast with either no party results or exactly one party result with exactly one digit. Get the function working with that simplification, updating and adding to your test cases in your main routine.

   Continue in this manner, removing more and more simplifications until you're implementing the definition of a poll data string as in the spec. There are a number of orders in which you could remove the simplifications; here's one such ordering:

   1. No state forecasts at all.
   2. No state forecasts, or exactly one state forecast with either no party results or exactly one party result with exactly one digit.
   3. Like the previous simplification, but allow an optional second digit.
   4. Like the previous simplification, but allow any number of party results. At this point, your definition of a syntactically correct poll data string is zero or one state forecast.
   5. Zero or more state forecasts. At this point, you're implementing what the spec requires of this function.

   Now that `hasProperSyntax` is correctly implemented, proceed in a similar fashion to implement `tallySeats`, starting with an extreme simplification, then working your way through successive removals of simpifying assumptions.

   As we said in the FAQ for Project 2, it might seem counterintuitive, but following this incremental approach to developing your program will take *less* total time than trying to write the whole thing at once.

   At each step where you've got something working, save a copy of the program. That way, if you mangle things up in the next phase, or you run out of time, you'll be able to turn in a program that will compile and pass at least some of our test cases. That beats getting a zero for correctness!

2. **Can we assume the `pollData` parameter never contains *blah blah blah*? Can we assume the `pollData` always contains *blah blah blah*?**

   No! The `pollData` parameter might contain *any* string! Of course, if that string is not a syntactically correct poll data string, the spec tells you what the function must do.

3. **Is such-and-such a syntactically correct poll data string?**

   If you think it is, how many state forecasts does it have? What are they? Are you sure they each fit the definition of a state forecast? Are you sure that every character in the supposed poll data string is part of some state forecast? For each supposed state forecast, how many party results does it have? Are you sure they each fit the definition of a party result?

4. **When must `tallySeats` return 2 and not 1?**

```
int s;
int r1 = tallySeats("MA9D", '@', s);
        // r1 must be 2, since the party parameter '@' is not a letter.
        // r1 must not be 1, since "MA9D" *is* a poll data string.

int r2 = tallySeats("MA9@", 'D', s);
        // r2 must be 1, since "MA9@" is not a poll data string.
        // r2 must not be 2, since the party parameter 'D' *is* a letter.

int r3 = countVotes("MA9@", '@', s);
        // r3 must be 1 or 2, your choice which one.
```

5. **What must this do?**

```
int s;
int r = tallySeats("", 'D', s);
```

   Since `""` is a valid poll data string and `'D'` is a letter, `r` must be set to 0, and s must be set to the total number of seats that `""` predicts party D will win, which is 0.

6. **Must `hasProperSyntax("MA9D,,KS4R")` return `true` or `false`?**

   Yes. :-) The wording in the spec, *A poll data string is a sequence of zero or more state forecasts separated by commas …* is ambiguous, since a phrase like *separated by whatevers* is sometimes used to mean *each separated from a following one by exactly one whatever* (which is what we intended) and sometimes *each separated from a following one by one or more whatevers* (which we did not intend). At this late date, we won't make you rewrite your code if you chose the latter: We'll resolve this by proclaiming that you must choose one or the other of those interpretations and use it consistently in your program. For example, you must pass this test:

```
assert(hasProperSyntax("MA9D,KS4R")  &&  hasProperSyntax("KS4R,MA9D"));
assert(hasProperSyntax("MA9D,,KS4R") == hasProperSyntax("KS4R,,MA9D"));
```

   You would *not* be consistent if the second assertion failed because one of the calls to `hasProperSyntax` returned false and the other true; either both must return false (which was our original intent) or both must return true.