Project 3 FAQ

1. **How do I begin?**

First, the assignment itself is 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 most important function first: isValidQC. Make a tremendously simplifying assumption: A QC test string is not valid if it contains no data at all. Get the function working with that simplification: it should return false for the empty string.

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 QC test string is valid if it starts with a Q and a number greater than zero. Get the function working with that simplification, updating and adding to your test cases in your main routine. Test data that is valid that your program can currently report as valid. Test data that is invalid that your code can currently report as invalid.

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

1. Require a non-zero digit after the Q for a QC test string to be valid
2. Allow for a number of digits with no leading zeros after the Q for a QC test string to be valid
3. Allow for both p and d following the digits after the Q for a QC test string to be valid
4. Require a digit after the p and after the d for a QC test string to be valid
5. Allow for a number of digits with no leading zeros after the p and after the d for a QC test string to be valid
6. Ensure the number of pass and defect test results equals the total number of tests in the QC test string
7. Create a loop that processes multiple batches of test results for a QC test string to be valid

2. Now that isValidQC is correctly implemented, proceed in a similar fashion to implement totalQC , starting with an extreme simplification, then working your way through successive removals of simplifying assumptions. And yes, this function can call the isValidQC. This is

intended to show you the power of reuse, a key benefit to writing modularized code with functions. Proceed in a similar fashion to implement passQC, defectQC and batches.

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. If you mangle things up in the next phase or 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!