Fall 2018 CS 31

Project 2 FAQ

1. **Do I have to *blah blah blah*? Is it OK to *blah blah blah*? What should the program do if *blah blah blah*?**

   One skill you'll need as a professional in whatever technical field you're in is the ability to read a technical specification. If your question is about what your program is required or permitted to do, then if you carefully re-read the Project 2 spec and this FAQ with your question in mind, you'll probably find that there's a sentence or two that answers it. If *after* re-reading the spec and this FAQ, you still don't know whether that thing is required or permitted, then ask us. If we think the answer to your question is right there in the spec, our answer may be merely *The spec says: [some sentence we quote from the spec]*. If that happens, use that as an opportunity to improve your technical reading skills by analyzing why you didn't see that yourself even though that was what you were looking for.

2. **How do I return from `main()`, thus ending my program, other than by reaching the end of it?**

   You can do something like this:

```
int main()
{
    do some stuff
    if (something bad happened)
    {
        write an error message
        return 1;    // non-zero return value means program couldn't do its job
    }
    do some more stuff
}
```

   If *something bad happened*, this will *write an error message* and terminate the program without doing *some more stuff*.

3. **Should the very last character that my program writes be a newline?**

   It won't matter to our grading tool, but it's better style. In other words, if the last line your program is going to write in some cases is `You must enter a product.`, then saying

```
cout << "You must enter a product." << endl;
```

   or

```
cout << "You must enter a product.\n";
```

   is better than

```
cout << "You must enter a product.";
```

   with no further output. Oh, and don't worry about the `Press any key to continue` or `Program ended with exit code` message. That's something the Visual Studio or the Xcode environment writes after your program finishes, but the way our grading tool runs your program, that message isn't produced.

4. **When I run my program, why does the dialog start like this?**

```
Odometer at start: 2417
Odometer at end: 2754
Rental days: 4
Customer name: Luxury car? (y/n):
```

   Why didn't it give me a chance to type the customer name?

   If you had read the Some Things about Strings writeup, you'd know why. (Look at the discussion of the Floral Tapestry example and the use of `cin.ignore(…)`.)

5. **Is zero positive?**

   No — never was, never will be. Why would you think that when we use a word in a specification, we're using some idiosyncratic definition like Humpty Dumpty would? There's glory for you! Anyway, perhaps you should learn a bit more about zero.

6. **I wrote the program and now I'm trying to get it to work. It's not reading the input correctly, it's printing multiple messages, and many output numbers are wrong. What can I do?**

   Most importantly, learn this lesson from your mistake: You should develop your programs incrementally. It seems that you did not do this; instead, you wrote a lot of code and then tried to test it, so all the problems with your code are hitting you at once.

   It may seem like a step backward, but what you should do is start over again. Create a new project and solve a simplified version of the problem. For example, assume the user will always enter correct input, and have your program use a very simple rental charge formula (e.g., the base charge is always $33 per day plus $0.27 per mile for all miles). Get that working, making sure the format of any output you write is correct. Once you get that working, enhance the program to check for an empty customer name, and make sure your code works for both empty and non-empty customer names. Then add checking for bad month numbers and test the code. Continuing to add error-checking code in this fashion, you should end up with a program that does everything except always computing the right rental charge in the case of valid input.

   That's a good state to be in, because now you know the only thing you have to do is change the computation of the rental charge; that's independent of everything else you've written and know to be correct. Since the actual rental charge calculation rules are complex, continue by implementing a simplification that's only a little more complex than your original simplification. For example, distinguish between luxury/non-luxury car rentals. Get that working. Implement the mileage tiers, ignoring the the winter/non-winter distinction. Get that working. Add the winter/non-winter distinction. Get that working. You're done!

   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 on something other than the computer you're doing your development on (e.g., online or on a flash drive). That way, if you mangle things up in the next phase, or you run out of time, or your hard drive fails, 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!

7. **I've tested my program thoroughly and I believe my program is behaving the way it's required to. How can I shake the feeling I have, though, that I might have overlooked something fundamental that will cost me a lot of correctness points when your testing script runs it?**

   Follow the instructions in the g++ with Linux writeup to run your program using g31 on `cs31.seas.ucla.edu`. Then, when you are satisfied that it works, run this command on that server:

   ```
   curl -s -L http://cs.ucla.edu/classes/fall18/cs31/Utilities/p2tester | bash
   ```

   It will test your program on a couple of cases in the manner that our testing script will. (Of course, our testing script will test many more cases than just a couple.)