Automated Unit Tests

This assignment has 5 required parts, and 2 bonus parts.

You are starting your first day at a new company and you've been given a project from an engineer who won the lottery and quit their job without giving any notice. You can find the code for your project here: https://github.com/audgster/esep-grade-calculator

After a few hours in onboarding, you meet with the Product Manager Timothy…

You're building a final grade calculator library that will compute final grades for our new educational product. There are three kinds of grades a student can get in the class: assignments, exams, and essays. Depending on the class, each type of grade can have different weights. The final grade is a weighted average of all the assignments. In short, you compute the average grade of each kind of grade, then compute the final grade by summing the average of each kind multiplied by their respective weights. Finally, we convert their numerical grade to a letter grade based on a grade scale.

Part 1: Requirements Clarification (.3 pts)

Written Answer: Based on your conversation with Timothy, write out 3 questions you would ask to clarify the requirements.

- Should the grade weights be fixed or configurable?
- How should the program handle missing assignments?
- How should rounding work for assignments and final grade?

Part 2: Understanding a New Project (1 pt)

Fork the project and go through the setup steps for the project. If you have any questions about setup, please ask in #assignments in Slack. Make sure to check if your question has been answered before.

Written Answer: Answer the following questions:

1. What tech stack is used in this repo?

The language used is Go and it has its own testing framework and toolchain. It uses runtime Go 1.24.4.

2. Write a brief summary of each file in the repository (4 total)

   The readme provides the command necessary for the test cases and coverage. Go.mod declares the module name and version. Grade_calculator.go is the core code file, it contains the code for the grade structure and GradeCalculator and its methods. Grade_calculator_test.go contains the unit tests that use go's built in testing package.

3. Are there tests in the repository?
   1. What do they do?

   TestGetGradeA checks that when all grades are 100's the final grade returned is an A. TestGetGradeB checks that when the grades are an 80, 81, and 85 the final grade returned is a B. TestGetGradeF checks to see that when the grades are a 100, 95, and 91 the final grade returned is an F, which is obviously incorrect.

   2. How do you run them?

   They are run using the commands in the readme, these are go test ./… and go test -coverprofile=coverage.txt ./…

4. What resources did you use to help you understand the code?

   I did a google search for the syntax of Go and the syntax of its testing framework.

Code: Based on your exploration of the code, and your own research, update the readme for the next person to "use" the repo. This should be in its own commit.

Part 3: Updating Tests (1pt)

As you can see, when you run the tests the tests are failing.

Written Answer: Based on the requirements, identify which tests are failing because the test is asserting the wrong thing and which are failing because the code is wrong.

The errors that returned F when supposed to be A or B was fixed in grade_calculator.go by fixing calculateNumericalAverage and computeAverage. The error that returned an A when an F was expected was an issue with the F test case adding A's for each category instead of an F.

Code: Fix the tests and code according to the requirements. This should be in its own commit

For the sake of this section, we can assume the following:
Grade Scale:
A >=90
B >=80
C >=70
D >= 60 F <60

Passing is C or greater. Assignments are 50%, exams are 35%, and essays are 15%.

Part 4: Adding Tests (2 pts)

1. Run the tests with code coverage. **Written Answer:** What is the code coverage of the project after your updates?

88%

2. **Code:** Add tests to reach 100% code coverage. This should be in its own commit.
3. **Written Answer:** How would you refactor this code to make it easier to test?

   I would make configuration explicit using weights and a grade scale. I could also collapse the data []Grade into a single collection then filter by type.

Part 5: Understanding Test Coverage (1 pts)

1. **Written Answer:** Can you achieve 100% code coverage without testing all cases? Why?
   Yes because most tools measure line coverage not all logical paths and branches.
2. Comment out the assertions in your test. **Written Answer:** What happens to your code coverage when you have no assertions?
   Coverage can stay high because the tests still execute the code which marks lines as covered but this no longer checks that answers are right, only that the code runs.

3. **Written Answer:** How would you address this scenario in a team setting?

I would require assertions, set coverage and quality gates, and I would use table driven tests to confirm coverage.

Bonus: Changing Internals (1 pt)

Code: Update the code to use a single list for all types of grades instead of three separate ones. This should be its own commit

Written Answer: Reflect on how well your unit tests held up as you changed your implementation

Bonus: Changing Requirements (1 pt)

Timothy has just come back from a user research session and thinks we should support new functionality. Instead of just supporting letter grades, the calculator should also support Pass/Fail.

Code:

1. Leaving the public interface of GradeCalculator unchanged, add unit tests for the new requirements. You can change the constructor and the GradeCalculator object however you'd like. This should be its own commit.
2. Implement the changes necessary to make your tests pass. This should be its own commit.

Written Answer: This was a brief exercise in test driven development. How did writing the tests help you think about the code?

Deliverables

You will submit a PDF with a link to your repository and the answers to your questions organized by parts.

The last **.7 pts** of the grade are for having commits properly separated as this will be used to snapshot each part of your project.