Exercise 3: Specification-Guided Test Improvement

This is an individual assignment. The final submission must be completed and submitted independently by each student.

Deadline: Nov 25, 11:59PM EST*. The deadline is sharp -- late submissions will not be accepted. Assignments must be submitted through GradeScope. We understand that circumstances may arise where you may need to submit the assignment late.* ***Seek approval by contacting TA Mingzhe Li (mingzhel@umass.edu) at least 24 hours in advance (unless it's a last-minute emergency and you cannot).*** *Medical conditions, religious or funerary events, university-related events (conference visit, athletic event, field trip, or performance), or extenuating non-academic reasons (military obligation, family illness, jury duty, automobile collision) that need extension will be accommodated with written documentation. Assignments or exams from other courses, interviews, or paper deadlines are not legitimate reasons for an extension.*

Objective:

Explore how formal specifications, automatically generated from natural language problem statements, can guide test improvement. (Total: 10 points)

Problem Selection & Requirement:

Select two problems from Exercise 2 (Part 1) where your testing coverage is weakest. You may use the same "Problem Statement × LLM solution" pairs you analyzed earlier in Exercise 2. You will:
1. Prompt an LLM with each problem's natural language description (not its code) and method signature to generate formal specifications as test assertions.
2. Manually evaluate and correct these assertions.
3. Use the refined specifications to create test cases.
4. Compare whether the new tests improve coverage or reveal additional insights.

Even if coverage does not increase, clear reasoning and case-specific explanations earn full credit.

Part 1: Generate, Evaluate, and Refine Specifications (5 pts)

1. For each problem:
   - Identify the function or method signature (input/output types).
   - Copy the natural language problem description.
   - Use this description and signature as the LLM prompt input to generate formal specifications written as assertions.

Your prompt must clearly instruct the LLM to:
- Write formal specifications in the form of assertions (`assert` statements).
- Not call the target function/method itself inside the assertions (no self-reference).
- Avoid all methods that cause side effects, such as:
  - modifying data structures (add, append, remove, set)
  - performing I/O (print, read, write, input, outputStream)
  - using randomness or timing (Math.random(), System.currentTimeMillis(), datetime.now()).

Number of Specifications Guideline:
- There is no fixed number of specifications required for each method.
- You may ask the LLM to generate about 5 specifications.
- It's possible that a method is very simple and yields fewer than 5 unique specifications; keep only the distinct ones.
- For more complex methods, more than 5 meaningful specifications may exist — you can either select 5 correct ones or include more.
- The key requirement is to use logically correct specifications, not to produce a complete set.

Java Example:
Prompt:
Problem description: Write a method that checks whether an integer 'n' is even.
Method signature: public static boolean isEven(int n)
Please write formal specifications as Java assertions that describe the correct behavior of this method.
Let 'res' denote the expected return value of 'isEven(n)'.
Do not call 'isEven()' in your assertions.
Do not use methods with side effects such as System.out.println, file I/O, random number generation, or timing functions.
Express the relationship between 'n' and 'res' using pure arithmetic and boolean logic only.

Expected Output:
```*java*
*// Specification-style assertion for isEven(int n)*
*assertTrue(res == (n % 2 == 0));  // The result must equal "n is divisible by 2"*
```

2. Review each generated assertion and mark it as Correct or Incorrect.
3. Compute the accuracy rate:
  Accuracy Rate = (# Correct Assertions) / (Total Assertions)
4. For every incorrect assertion, provide:
  - A brief explanation of the problem (e.g., missing boundary condition, use of side effects).
  - A manually corrected version that accurately represents the intended behavior.

After completing your manual corrections, use only the logically correct specifications in Part 2 to guide test-case generation.

Part 2: Use Specifications to Guide Test Improvement (5 pts)

1. Generate Test Cases from Specifications
Feed your final, corrected specifications back into the LLM and ask it to generate test cases that reflect the logical properties in the assertions. Label these tests clearly as spec-guided and add them to your Exercise 2 test suite.

2. Compare with Exercise 2 Tests
Run both your original tests (from Exercise 2) and the spec-guided tests. Report:
- Statement coverage (%)
- Branch coverage (%)
- Any new failing tests or insights

If no improvement is observed, provide a case-specific explanation.
Example:
"Coverage did not increase because the generated specification restates a condition already tested by t3, one of the original test cases from Exercise 2."

Deliverables (same format as Exercise 2). Submit one PDF document on Gradescope. The PDF must include:

- LLM prompt, generated assertions before correction, and accuracy rate.
- A table of incorrect assertions, the issues, and corrected assertions.
- LLM prompt used to generate test cases with specifications as guidance.
- A list of the resulting tests labeled as spec-guided
- Before/after coverage table:
  | Problem | Old Stmt % | New Stmt % | Old Branch % | New Branch % |
- Case-specific insight per problem
- A link to your GitHub repository, which must contain:
  - Source code of the problems, test files (baseline + improved), and any scripts/config for coverage.
  - Generated and revised assertions.
  - Generated coverage reports (HTML/XML) or instructions to reproduce locally.