6 Appendices

A Prompt Templates

The complete system prompts used for each of the three prompting strategies are provided below for reproducibility. The {function} placeholder was dynamically replaced with the specific function name for each programming problem.

A.1 IO System Prompt

```
You are generating simulated student Java code submissions for a programming problem.

### Goal
- Produce compilable Java code submissions, each featuring exactly one non-trivial
    logical error (no syntax errors or runtime crashes).
- Each submission should reflect a genuine attempt by a novice programmer.

### Constraints
- Implement the required Java method using the exact name provided, represented as
    {function}.
- Include only one logical error per submission; also, the code must compile and run
    without crashing.
- Adhere to all I/O and method signature requirements as outlined in the problem
    description.
- Code blocks must contain only Java  c o d e exclude  comments, hints, debug output, or
    external explanations.

### Variety
- Each submission should contain a distinct, plausible logical error pertinent to the
    specific problem.
- Do not repeat the same error type across different submissions; ensure diversity
    until reasonable variation is exhausted.

### Output Format

For each submission:
Submission i:
```java
// Substitute this block with Java code that implements the {function} method, using
    the parameters and return type specified in the problem statement. Introduce
    exactly one plausible, non-trivial logical error relevant to the problem.
```
```

A.2 CoT System Prompt

```
You are generating simulated student Java code submissions for a programming problem.

### Goal
- Produce compilable Java code submissions, each featuring exactly one non-trivial
    logical error (no syntax errors or runtime crashes).
- Each submission should reflect a genuine attempt by a novice programmer.

### Reasoning policy
- For EACH submission, FIRST provide a SHORT reasoning (1 3 sentences) describing
    the intended approach and the kind of subtle logical slip it might contain (no
    test cases, no step-by-step).
- Then immediately provide the code block.
- Keep the reasoning concise and high-level.

### Constraints
- Implement the required Java method using the exact name provided, represented as
    {function}.
- Include only one logical error per submission; also, the code must compile and run
    without crashing.
- Adhere to all I/O and method signature requirements as outlined in the problem
    description.
- Code blocks must contain only Java c o d e exclude comments, hints, debug output, or
    external explanations.

### Variety
- Each submission should contain a distinct, plausible logical error pertinent to the
    specific problem.
- Do not repeat the same error type across different submissions; ensure diversity
    until reasonable variation is exhausted.

### Output Format
For each submission:
Submission i:
Reasoning: <1 3 sentences, concise, high-level>
```java
// Substitute this block with Java code that implements the {function} method, using
    the parameters and return type specified in the problem statement. Introduce
    exactly one plausible, non-trivial logical error relevant to the problem.
```

A.3 Self-Refine System Prompts

A.3.1 Initial Generation Prompt.
```

```
You are generating simulated student Java code submissions for a programming problem.

### Goal

- Produce compilable Java code submissions, each featuring exactly one non-trivial
    logical error (no syntax errors or runtime crashes).
- Each submission should reflect a genuine attempt by a novice programmer.

### Reasoning policy

- For **each** submission, first output a single short line starting with
    **  Reasoning  :  ** (1  3  sentences, high-level; no test cases; no
    step-by-step).
- Immediately after the Reasoning line, output the code block.

### Constraints

- Implement the required Java method using the exact name **{function}**.
- Exactly **one** logical error per submission; code must compile and not crash.
- Respect the problems I/O and method signature.
- No trivial sabotage (no hardcoded answers, no skipping essential logic, no
    deliberate exceptions).
- Avoid `main`, I/O, randomness, external resources, and imports unless explicitly
    required by the problem.
- Inside code blocks: **Java code only** (no comments, hints, or debug output).

### Variety

- Across submissions, each logical error must be **distinct** and **plausible** for
    the problem.
- Do not repeat the same error type unless reasonable variety is exhausted.

### Output format (strict)

For each submission:

````
Submission i:
Reasoning: <1 3 sentences, concise, high-level>
```java
// code only; one fenced block per submission
````

```
```

```
Do not include any extra text before the first submission or after the last
    submission.
```

A.3.2 *Feedback Prompt (Critic).*

```
You are reviewing a **set** of simulated student Java submissions with the following
    objective:

- Each submission compiles and contains **exactly one** non-trivial logical error.
- Submissions look like genuine novice attempts (no trivial sabotage).
- The required method name is exactly **{function}**.
- Output from the generator includes a  Reasoning  :  line (for CoT setups)
    followed by a single fenced Java code block containing **only** Java code, and it
    respects the problems I/O/signature.

### Your task
- Evaluate the **entire set** and judge each submission against the objective.
- For any submission that **fails** a requirement, provide concise guidance on what
    to change so that the **next revision still contains exactly one non-trivial
    logical error** but becomes compliant (e.g., fix method name/signature, remove
    comments/debug prints, avoid multiple bugs, ensure plausibility, preserve
    formatting, maintain distinctness across the set).
- If there are cross-submission conflicts (e.g., duplicate error types), identify
    them and specify which submission(s) should be revised and in what direction (no
    code).

### Style
- Be concise and concrete (aim for 2 6  sentences per submission verdict).
- Do not reveal hidden tests or provide step-by-step chain-of-thought.
- **Do not provide code** or code fragments.

### Required output structure (strict)
- A numbered list covering **all** submissions in the form:

```

Submission i: Compliant   <brief justification>

```
or
```

Submission i: Revise   <specific issues and how to adjust while keeping exactly one
    non-trivial logical error>
```
```

```
```

- Optionally, an **Overall:** paragraph for cross-submission issues (e.g., duplicate
    error types, inconsistent formatting).
```

### *A.3.3 Refinement Prompt.*

```
You are revising a **set** of simulated student Java submissions based on batch
    feedback.

### Goal
- Produce a **revised set** that preserves the original **count** of submissions.
- For items marked **Compliant** in feedback: **return the exact original code
    unchanged**.
- For items marked **Revise**: return a revised version that **compiles** and
    **intentionally contains exactly one non-trivial logical error**, while
    addressing the feedback.

### Constraints
- Keep the method name **exactly {function}** and preserve the required I/O and
    signature from the problem.
- Ensure the logical error is plausible (no hardcoding answers, no skipping logic, no
    deliberate exceptions).
- Maintain **distinct** error types across the set where required by feedback.
- Avoid `main`, I/O, randomness, external resources, and imports unless explicitly
    required.
- Inside code blocks: **Java code only** (no comments, hints, or debug prints).

### Output policy
- Output **the same number** of submissions as the input set.
- For each submission, output **one** fenced Java code block (no extra prose).
- No text before the first submission or after the last submission.

### Output format (strict)
For each submission **i** in order:
```

Submission i:

```java
// one fenced Java block; if Compliant, reproduce original unchanged; if Revise,
    provide the revised code with exactly one non-trivial logical error
```
```