6 Appendices

521 522

523 524

525

526 527

528 529

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

```
530
       You are generating simulated student Java code submissions for a programming problem.
531
532
533
       ### Goal
534
       - Produce compilable Java code submissions, each featuring exactly one non-trivial
535
            logical error (no syntax errors or runtime crashes).
536
       - Each submission should reflect a genuine attempt by a novice programmer.
537
538
       ### Constraints
539
       - Implement the required Java method using the exact name provided, represented as
540
541
            {function}.
542
       - Include only one logical error per submission; also, the code must compile and run
543
           without crashing.
       - Adhere to all I/O and method signature requirements as outlined in the problem
545
           description.
546
       - Code blocks must contain only Java code exclude comments, hints, debug output, or
547
           external explanations.
548
549
550
       ### Variety
551
       - Each submission should contain a distinct, plausible logical error pertinent to the
552
            specific problem.
553
       - Do not repeat the same error type across different submissions; ensure diversity
554
            until reasonable variation is exhausted.
555
556
       ### Output Format
558
559
       For each submission:
560
       Submission i:
561
       ```java
562
 // Substitute this block with Java code that implements the {function} method, using
563
 the parameters and return type specified in the problem statement. Introduce
564
 exactly one plausible, non-trivial logical error relevant to the problem.
565
 . . .
566
567
```

#### A.2 CoT System Prompt

568569

12 Anon.

You are generating simulated student Java code submissions for a programming problem. 573 574 575 ### Goal 576 - Produce compilable Java code submissions, each featuring exactly one non-trivial 577 logical error (no syntax errors or runtime crashes). 578 - Each submission should reflect a genuine attempt by a novice programmer. 579 580 581 ### 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. 586 - Keep the reasoning concise and high-level. 587 588 589 ### Constraints 590 - Implement the required Java method using the exact name provided, represented as 591 {function}. 592 - Include only one logical error per submission; also, the code must compile and run 593 without crashing. 594 - Adhere to all I/O and method signature requirements as outlined in the problem 595 description. - Code blocks must contain only Java code exclude comments, hints, debug output, or 598 external explanations. 599 600 ### Variety 601 - Each submission should contain a distinct, plausible logical error pertinent to the 602 specific problem. - Do not repeat the same error type across different submissions; ensure diversity 604 until reasonable variation is exhausted. 605 606 607 ### Output Format For each submission: Submission i: Reasoning: <1 3 sentences, concise, high-level> ```java 612 // Substitute this block with Java code that implements the {function} method, using 613 the parameters and return type specified in the problem statement. Introduce 614 615 exactly one plausible, non-trivial logical error relevant to the problem. 616

# A.3 Self-Refine System Prompts

A.3.1 Initial Generation Prompt.

617 618

619

```
You are generating simulated student Java code submissions for a programming problem.
625
626
627
 ### Goal
628
629
 - Produce compilable Java code submissions, each featuring exactly one non-trivial
630
 logical error (no syntax errors or runtime crashes).
631
 - Each submission should reflect a genuine attempt by a novice programmer.
632
633
 ### Reasoning policy
636
 - For **each** submission, first output a single short line starting with
637
 ** Reasoning : ** (1 3 sentences, high-level; no test cases; no
638
 step-by-step).
639
 - Immediately after the Reasoning line, output the code block.
640
641
642
 ### Constraints
643
644
 - Implement the required Java method using the exact name **{function}**.
645
 - Exactly **one** logical error per submission; code must compile and not crash.
646
 - Respect the problems I/O and method signature.
647
 - No trivial sabotage (no hardcoded answers, no skipping essential logic, no
 deliberate exceptions).
649
650
 - Avoid `main`, I/O, randomness, external resources, and imports unless explicitly
651
 required by the problem.
652
 - Inside code blocks: **Java code only** (no comments, hints, or debug output).
653
654
 ### Variety
655
656
 - Across submissions, each logical error must be **distinct** and **plausible** for
657
658
 the problem.
659
 - Do not repeat the same error type unless reasonable variety is exhausted.
 ### Output format (strict)
662
663
 For each submission:
664
665
666
667
 Submission i:
668
 Reasoning: <1 3 sentences, concise, high-level>
669
 ···java
670
 // code only; one fenced block per submission
671
672
673
 . . .
674
676
```

14 Anon.

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

```
submission.
679
680
681
 A.3.2 Feedback Prompt (Critic).
682
683
 You are reviewing a **set** of simulated student Java submissions with the following
684
685
 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}**.
690
 - Output from the generator includes a Reasoning :
 line (for CoT setups)
691
 followed by a single fenced Java code block containing **only** Java code, and it
692
 respects the problems I/O/signature.
693
694
695
 ### Your task
696
 - Evaluate the **entire set** and judge each submission against the objective.
697
 - For any submission that **fails** a requirement, provide concise guidance on what
698
 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
701
702
 formatting, maintain distinctness across the set).
703
 - If there are cross-submission conflicts (e.g., duplicate error types), identify
704
 them and specify which submission(s) should be revised and in what direction (no
705
 code).
706
707
 ### Style
708
 - Be concise and concrete (aim for 2 6 sentences per submission verdict).
709
 - Do not reveal hidden tests or provide step-by-step chain-of-thought.
710
711
 - **Do not provide code** or code fragments.
 ### Required output structure (strict)
 - A numbered list covering **all** submissions in the form:
716
717
718
719
 Submission i: Compliant
 <brief justification>
720
721
722
 or
723
724
725
 <specific issues and how to adjust while keeping exactly one</pre>
 Submission i: Revise
727
 non-trivial logical error>
```

Manuscript submitted to ACM

728

```
729
730
 . . .
731
732
 - Optionally, an **Overall:** paragraph for cross-submission issues (e.g., duplicate
733
 error types, inconsistent formatting).
734
735
736
 A.3.3 Refinement Prompt.
737
738
739
 You are revising a **set** of simulated student Java submissions based on batch
740
 feedback.
741
742
 ### Goal
743
 - Produce a **revised set** that preserves the original **count** of submissions.
744
 - For items marked **Compliant** in feedback: **return the exact original code
745
746
 unchanged**.
 - For items marked **Revise**: return a revised version that **compiles** and
747
748
 intentionally contains exactly one non-trivial logical error, while
749
 addressing the feedback.
750
 ### Constraints
 - Keep the method name **exactly {function}** and preserve the required I/O and
753
 signature from the problem.
754
 - Ensure the logical error is plausible (no hardcoding answers, no skipping logic, no
755
756
 deliberate exceptions).
757
 - Maintain **distinct** error types across the set where required by feedback.
758
 - Avoid `main`, I/O, randomness, external resources, and imports unless explicitly
 required.
760
 - Inside code blocks: **Java code only** (no comments, hints, or debug prints).
761
762
 ### Output policy
763
 - Output **the same number** of submissions as the input set.
 - For each submission, output **one** fenced Java code block (no extra prose).
766
 - No text before the first submission or after the last submission.
767
768
 ### Output format (strict)
769
 For each submission **i** in order:
770
771
772
773
 Submission i:
774
775
       ```java
776
       // one fenced Java block; if Compliant, reproduce original unchanged; if Revise,
777
```

provide the revised code with exactly one non-trivial logical error

778