Lexical Analyzer Comparison Report

Assignment 01 - Compiler Construction

1. Output Comparison

Both the Manual Scanner and the JFlex-generated scanner were executed against the five provided test files (test1.lang to test5.lang). The outputs were compared for token recognition, line/column tracking, and error handling.

Test Case 1: Valid Tokens (test1.lang)

Feature	Manual Scanner Output	JFlex Scanner Output	Verdict
Token Format	<KEYWORD, "start", Line: 1, Col: 1>	<KEYWORD, "start", Line: 1, Col: 1>	**Match**
Identifiers	<IDENTIFIER, "Count", Line: 2, Col: 13>	<IDENTIFIER, "Count", Line: 2, Col: 13>	**Match**
Literals	<INTEGER, "0", Line: 2, Col: 21>	<INTEGER, "0", Line: 2, Col: 21>	**Match**
Operators	<OPERATOR, "++", Line: 6, Col: 14>	<OPERATOR, "++", Line: 6, Col: 14>	**Match**

Test Case 3: Strings & Escapes (test3.lang)

Feature	Manual Scanner Output	JFlex Scanner Output	Verdict
String Literal	<STRING, "Hello\n", Line: 2, Col: 10>	<STRING, "Hello\n", Line: 2, Col: 10>	**Match**
Char Literal	<CHAR, '\t', Line: 3, Col: 5>	<CHAR, '\t', Line: 3, Col: 5>	**Match**

Test Case 5: Comments (test5.lang)

Feature	Manual Scanner Output	JFlex Scanner Output	Verdict
Single-line	Ignored (Comment removed)	Ignored (Skipped by Regex)	**Match**
Multi-line	Ignored (Comment removed)	Ignored (Skipped by Regex)	**Match**

---

2. Explanation of Implementation Differences

While the outputs are identical, the internal mechanisms differ significantly:

A. Token Recognition

- **Manual Scanner:** Uses a specific order of `if-else` checks. It manually peeks at characters (`peek()`, `advance()`) to determine if a token is a keyword, identifier, or number. It explicitly implements the "Longest Match Principle" by checking for multi-character operators (like `==`) before single-character ones (like `=`).
- **JFlex Scanner:** Uses a generated Deterministic Finite Automaton (DFA). The priority is determined by the order of rules in the `.flex` file. JFlex compiles these rules into a minimized transition table, allowing it to recognize tokens without manual character checks.

B. Error Handling

- **Manual Scanner:** Has custom logic to detect specific errors (e.g., detecting `12.34.56` as a "Malformed Literal"). It provides detailed, custom error messages dependent on the specific `if` block where the error occurred.
- **JFlex Scanner:** Relies on a fallback rule (`.` matches any character) to catch invalid input. While effective, it generally groups all unrecognized characters into a generic `ERROR` token unless specific error regex patterns are defined.

---

3. Performance Comparison

Metric	Manual Scanner	JFlex Scanner

Initialization	**Fast.** Zero setup time required as it is just a Java class.	**Slower.** Requires the JFlex tool to generate the Java code before compilation.
Execution Speed	**Moderate.** Uses many method calls (`peek`, `advance`) and string manipulations, which adds overhead for every character processing.	**Fast.** Uses a pre-computed DFA transition table (array lookups), making it O(n) complexity and highly optimized for speed.
Maintenance	**Hard.** Adding a new token requires writing new logic, handling edge cases, and updating multiple `if-else` blocks (approx. 300 lines of code).	**Easy.** Adding a new token only requires adding a single regex line in the `.flex` file (approx. 50 lines of code).

Conclusion

Both scanners successfully implement the lexical specifications of NovaLang. The Manual Scanner offers finer control over error reporting and symbol table management, while the JFlex Scanner offers superior execution performance and maintainability through concise regular expressions.