Program 2: Recursive descent, top-down parser

Hashim Abdulla

Due Date: 3/31/2025

**Development Process**

**Development Summary:**

The development of this recursive descent, top-down parser for a simple calculator language involved multiple iterations, debugging cycles, and optimizations. The goal was to implement a parser that follows a structured grammar and correctly processes input files containing arithmetic and logical expressions. The parser ultimately follows the defined grammar and successfully builds parse trees.

Experience with Different LLMs

Initially, OpenAI's GPT-4o and DeepSeek were tested for generating the parser. However, both models consistently produced code with significant issues, including:

* Syntax errors due to incorrect Racket-specific constructs
* Logical errors in parsing expressions and handling tokenization
* Infinite loops due to improper recursion management
* Poor handling of edge cases in scanning and parsing tokens

Given these challenges, I switched to Anthropic’s Claude 3.7 Sonnet, which proved to be far more reliable for generating functional Racket code. This model provided:

* Structurally sound recursive functions
* Improved token management logic
* Clearer handling of operator precedence and parsing flow
* Syntactically correct code

**Key Areas Where Claude 3.7 Sonnet Was Helpful:**

1. Tokenization Improvements:
   * Helped refine the scanning process to ensure correct token recognition, particularly for multi-character comparison operators (e.g., <=, >=, ==, !=).
   * Suggested better ways to handle whitespace and end-of-file conditions without causing unexpected behavior.
2. Parsing Logic Enhancements:
   * Helped refine recursive parsing logic, ensuring termination and avoiding infinite loops.
   * Provided better structuring for handling optional elements (like epsilon transitions in stmt\_list and etail).
3. Error Handling and Debugging Support:
   * Assisted in improving error messages for unexpected tokens, making debugging easier.
   * Helped refine structured error reporting with line numbers and expected token types.

**Challenges Faced & Lessons Learned**

* LLMs Need Strong Prompting for Syntax Sensitivity:
  + Racket’s syntax rules were often misinterpreted by GPT-4o and DeepSeek. Being explicit about expected syntax and structures in prompts helped me mitigate some of these issues.
* Recursion Management Requires Manual Verification:
  + Claude 3.7 Sonnet improved recursion handling, but careful manual testing was necessary to avoid stack overflows and infinite recursion loops.
* Edge Cases Matter:
  + Certain edge cases, like handling negative numbers or ensuring if statements had proper endif; termination, required explicit testing beyond what was initially generated by the LLM.

Recommendations for Future Development with LLMs

1. Always cross-check generated code manually for logical consistency.
2. Use LLMs for structured code generation but validate recursion rigorously.
3. Test with diverse input cases early to catch syntax and logic errors.
4. If one model fails, try another—some are better suited for specific tasks.
5. Iterate interactively—explain issues in the prompt for better refinements.

**My Conclusion**

The final parser successfully processes arithmetic and conditional expressions, demonstrating a good recursive descent approach. Claude 3.7 Sonnet significantly outperformed GPT-4o and DeepSeek in Racket-related syntax correctness and logical structure. However, manual debugging and testing were essential in ensuring reliability. This experience underscores the importance of choosing the right tool for the job and maintaining a critical eye when using LLM-generated code.