Roman Assignment Language: Reflections and Learnings

My journey through this project presented an opportunity to not only challenge my understanding of programming concepts but also to grow as a learner. The task was to create an evaluator for the Roman Assignments Language (RAL), a language involving Roman numerals and Latin operator terms, with specific requirements for error handling.

Learning Outcomes:

1. **Deepened Understanding of Parsing and Evaluating Expressions:** This project facilitated a deeper comprehension of how to parse and evaluate expressions. Creating the evaluator demanded the careful breakdown of expressions into understandable segments, followed by their evaluation according to RAL rules.
2. **Proficiency in Error Handling:** This project emphasized the importance of error handling. Distinguishing between lexical, syntactic, and semantic errors was challenging, but it honed my ability to detect and report specific errors accurately.
3. **Improved Skills in the Go Language:** Implementing this project in Go was both daunting and exciting. As Go is a statically typed, compiled language, it required meticulous attention to detail. It helped me solidify my understanding of Go's syntax and furthered my proficiency in using this powerful language.
4. **Command-Line Interaction:** This project exposed me to the real-world usage of command-line interfaces. Running the evaluator as a command-line program and reading input from a disk file, helped me comprehend how real-world applications interact with the operating system.

Challenges:

1. **Understanding Roman Numerals and Latin Operator Terms:** As a modern programmer, I found it challenging initially to adapt to Roman numerals and Latin operator terms. It took time to comprehend and implement them correctly, especially considering the RAL rules.
2. **Implementing Operator Precedence and Associativity:** Implementing the precedence and associativity of Latin operators akin to JavaScript's was a demanding task. It required thoughtful design to correctly parse and evaluate the expressions.

3. **Semantic Error Handling:** The specific requirements for semantic error handling in RAL were unique. Handling cases where computations would result in zero or negative numbers was challenging. Understanding the historical context of Roman numeral usage and developing suitable error messages to address these scenarios required extra research.
4. **Caret Positioning:** Determining the correct placement of the caret character ('^') to point to the error location in the assignment expression was one of the trickiest aspects of this project. The challenge was to accurately identify the first character of the offending token and terminate the execution upon detecting the very first error.

This project provided me with a unique blend of challenges and learning opportunities. Despite the difficulties, the experience was rewarding and insightful, as it deepened my understanding of the Go language, command-line interaction, parsing, evaluating expressions, and error handling.