CS3104 Compiler Lab, Autumn 2025
Assignment 2

Date: 08 August 2025

Instructions: Write your name and roll number on top of your source code as comments. Source code file name must be `"Assignment2.c"`.
Submit your code at: `http://10.22.10.100/~abyaym/CS3104/submission/`

Building upon your experience from Assignment 1, you will now develop a **Variable-based Calculator** that can handle variable assignments and evaluate expressions containing variables. This assignment introduces you to symbol tables and extends your parsing techniques to handle a simple programming language construct.

Problem Statement

Your task is to build a calculator that can handle variable assignments, evaluate arithmetic expressions containing variables, support all basic arithmetic operators (`+`, `-`, `*`, `/`, `%`), process multiple statements in sequence, and maintain proper operator precedence and associativity.

Input Format and Example Session

Your program should read multiple lines of input, which can be an assignment (`variable = expression`), an expression to be evaluated, or an exit command (`quit` or `exit`).

```
> x = 10
x = 10
> y = x + 5
y = 15
> z = x * y - 8
z = 142
> x + y * z / 2 - 15
1050
> w = (x + y) * (z - x) % 100
w = 0
> a + b
Error: Variable 'a' is not defined
> x / 0
Error: Division by zero
> quit
Goodbye!
```

Listing 1: Sample Input/Output Session

Core Functionality and Requirements

- **Variable Storage and Assignment:** Implement a symbol table to store variable names (single letters a-z, A-Z) and their integer values. Parse and execute assignment statements.
- **Expression Evaluation:** Evaluate expressions containing variables, positive integers, the operators `+`, `-`, `*`, `/`, `%`, and parentheses for grouping.
- **Operator Precedence:** Follow standard mathematical precedence: Parentheses (highest), then `*`, `/`, `%` (left-to-right), and finally `+`, `-` (left-to-right).
- **Error Handling:** Your program must detect and report errors such as undefined variables, division by zero, and invalid syntax.
- **Additional Features:** Handle whitespace flexibly, support an interactive mode with a prompt (`>`), display assignment results as `variable = value`, and allow a clean exit.

Algorithm Suggestions
You may extend your recursive approach from Assignment 1. Use an array of structures for a symbol table. Consider functions like `parseStatement()` to differentiate between assignments and expressions, `evalExpression()`, `evalTerm()` for handling different operator precedences, and `evalFactor()` for numbers, variables, and parenthesized sub-expressions.

Implementation Notes

- Variables are case-sensitive (`x` and `X` are different).
- All arithmetic is integer-based.
- Ensure your program provides clear error messages and does not crash on invalid input.
- Test your program with a variety of cases, including complex expressions, errors, and boundary conditions.

This assignment will help you understand how interpreters handle variable bindings and symbol tables - fundamental concepts in compiler design. The parsing techniques you develop here will be essential for more complex language constructs in future assignments.

Good luck and happy coding!