Programming Paradigms and Pragmatics (CS202)

LAB - 4

Kaleidoscope is a toy programming language designed to introduce the basics of compiler and language design. The language is procedural, supporting functions, conditionals, loops, and mathematical expressions. The purpose of Kaleidoscope is to keep the syntax simple while demonstrating key concepts of a language frontend, including tokenization, parsing, abstract syntax trees (AST), and interpretation or compilation.

Key Features of Kaleidoscope:

1. Minimal Syntax: Only one data type – a 64-bit floating-point number (double in C).
2. No Type Declarations: All values are implicitly of type double.
3. Procedural: Supports function definitions, conditionals (if/then/else), loops (for), and standard operators.
4. Extensibility: The language can be easily extended to include user-defined operators, JIT compilation, and more.

Please refer to the following link:
https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/LangImpl02.html

For each of these questions, you should:
a) Print a parse tree.
b) Implement new parsing functions or modify existing ones as required
c) Update the precedence handling system if necessary

1. **1.** Implement the necessary functions to parse the expression *"a+b*c"* ensuring correct operator precedence. Your implementation should:
   a) Parse variables 'a', 'b', and 'c' as primary expressions.
   b) Handle the multiplication (*) with higher precedence than addition (+).
   c) Return an AST representation of the expression.

   Provide the C++ code for the ParseExpression(), ParsePrimary(), and ParseBinOpRHS() functions, and explain how they work together to parse the given expression correctly.
2. **2.** Add support for the modulo operator (%):
   Extend the parser to handle the modulo operator. Update the BinopPrecedence map and

modify the ParseBinOpRHS function to incorporate this new operator. Ensure it has the same precedence as multiplication and division.

Input Example: *14 % 3 + 2 * 5*

3. Add support for comparison operators:
   Implement the less than (<), greater than (>), less than or equal to (<=), and greater than or equal to (>=) operators. Update the AST, precedence handling, and parsing functions to accommodate these new operators.

   Input Example: *a < b && c >= d*

4. Add support for string literals:
   Extend the lexer and parser to handle string literals enclosed in double quotes. Create a new AST node for string literals and update the parsing functions accordingly.

   Input Example: *"Hello" + " " + "World"*