CS 4308 – Concepts of Programming Languages Project
3 Deliverables (300 pts Total – 100 pts Per Deliverable)

The project for this course is the development of an interpreter of a language implemented in your choice of programming languages. You may use the same language for each part, or different languages.

This project consists of **developing an interpreter** in 3 parts: lexical analyzer (scanner), syntax analyzer (parser) and interpreter. The **interpreter** is for a **minimal form of the Julia language (see JuliaLang.org)**. This minimal form of Julia has only 1 data type, integer, and the only identifiers are single letters. Additional information is on the next page.

The interpreter will process a Julia file and build some intermediate data structures. These data structures will then be interpreted to execute the program. All tokens in this language are separated by white space. The parsing algorithm should detect any syntactical or semantic error. The first such error discovered should cause an appropriate error message to be printed, and then the interpreter should terminate. Run-time errors should also be detected with appropriate error messages being printed.

Deliverables (see D2L Assignments -> Projects for due dates):

1. **Deliverable P1 Scanner / Lexical Analyzer – (100 pts) Due 6/27/23 at 11:59 PM**
   Develop a complete scanner. Write a short report describing the work performed, include the source code, and show the input and output. You must show the execution of this program by using appropriate input files and the program must show a list of the tokens scanned. Write a short report with READABLE screeshots of your execution output, then your report describing the work performed, followed by (just) your scanner code copied and pasted as text. In addition, submit *copies of your source code files*. If you have more than 5 source code files, please submit your files as zip file.

2. **Deliverable P2 Parser / Syntax Analyzer – (100 pts) Due 7/4/23 at 11:59 PM**
   Develop a complete parser that executes with the scanner. You must show the execution of this program by using several relevant source lines as input, the program must show the corresponding statement recognized. Write a short report with READABLE screeshot of your execution output, then your report describing the work performed, followed by (just) your parser code copied and pasted as text. In addition, submit *copies of your source code files*. If you have more than 5 source code files, please submit your files as zip file..

3. **Deliverable P3 Interpreter – (100 pts) Due 7/18/23 at 11:59 PM**
   Develop a complete interpreter that includes the scanner and parser. You must show the execution of this program by using the relevant source line as input, the program must show the results after executing the statement recognized by the parser. Write a short report with READABLE screeshots of the output, then your report describing the work performed, followed by (just) your interpreter code copied and pasted. In addition, submit *copies of your source code files*. If you have more than 5 source code files, please submit your files as zip file.

Review Project-Assignment-And-Submission-Guidelines for details on submitting to D2L

As always 10% late penalties, per day, are in effect for each deliverable.

Grammar for the Julia language

Syntax Analyzer

<program> → function id ( ) <block> end

<block> → <statement> | <statement> <block>

<statement> → <if_statement> | <assignment_statement> | <while_statement> | <print_statement> | <repeat_statement>

<if_statement> → if <boolean_expression> then <block> else <block> end

<while_statement> → while <boolean_expression> do <block> end

<assignment_statement> -> id <assignment_operator> <arithmetic_expression>

<repeat_statement> -> repeat <block> until <boolean_expression>

<print_statement> → print (<arithmetic_expression>)

<boolean_expression> → <relative_op> <arithmetic_expression> <arithmetic_expression>

<relative_op> → le_operator | lt_operator | ge_operator | gt_operator | eq_operator | ne_operator

<arithmetic_expression> → <id> | <literal_integer> | <arithmetic_op>
<arithmetic_expression>
<arithmetic_expression>

<arithmetic_op> → add_operator | sub_operator | mul_operator | div_operator

Lexical Analyzer

id → letter

literal_integer → digit literal_integer | digit

assignment_operator → =

le_operator → <=

lt_operator → <

ge_operator → >=

gt_operator → >

eq_operator → = =

ne_operator → ~=

add_operator → +

sub_operator → -

mul_operator → *

div_operator → /