CPT316: Programming Language Implementation and Paradigms
Semester 1 2025/2026
Assignment 1

Weightage: 10%
Due Date: 21 November 2025 (Friday, before 5pm)
Submission: Submit via the elearn@USM by the due date.
Assignment Type: Group Work (4 members per group)

Assignment Overview

Build a small program that accepts an expression statement as input and performs:

1. Lexical Analysis
    - Scan the input to produce a token stream.
    - Identify and list invalid tokens (any character sequence outside the language).
    - Count tokens: show total tokens, and breakdown by token type.
2. Syntax Analysis
    - Parse the token stream against the grammar (below).
    - Generate and display the syntax tree for a valid input.
    - If parsing fails, print a relevant error message.

You may implement in any language (C/C++/Java/Python/etc.) and any paradigm (procedural/OOP/functional) but justify your choice in the report.

COMPY Language Specification

Tokens
Identifiers: single-letter, lowercase [a-z]
Numbers: integers
Operators: +, -, *, /
Assignment: =
Parentheses: ( )
Statement terminator: ;

Grammar
<stmt> → id = <expr> ;
<expr> → <term> | <expr> '+' <term> | <expr> '-' <term>
<term> → <factor> | <term> '*' <factor> | <term> '/' <factor>
<factor> → id | int | (<expr>)

Assignment Tasks

Part A – Implementation

Each group must implement a mini-compiler program that performs the following tasks:

1. Lexical Analyzer
    - Develop a scanner that can recognize and classify tokens: identifiers, numbers, operators (+ - * /), assignment (=), parentheses ((,)), and the statement terminator (;).
    - Output Requirements:
        - Display the token stream for the input expression.
        - Identify invalid tokens (characters not defined in the specification) and report their positions.
        - Provide a count of tokens both in total and broken down by type.
2. Syntax Analyzer
    - Implement a parser based on the grammar provided in the specification.
    - Output Requirements:
        - For valid input: generate and display the syntax tree. Choose the representation of syntax tree either text-based or diagram (higher marks will be awarded *wink*).
        - For invalid input: display a clear error message, specifying the type of error (lexical or syntax)
3. Error Handling
    - Your program must detect and report the identified errors.
    - Examples:
        - LexicalError at position 4: invalid character '$'
        - SyntaxError at position 7: expected ')' before '*'
        - SyntaxError at end of input: missing ';'
4. Program Execution Flow
    - Input: a single-line expression statement. Example, x = (3 + 5) * 2;.
    - Processing:
      perform lexical analysis → generate token stream and counts → parse tokens → build syntax tree or error report.
    - Output: display token list, counts, and either the syntax tree or error messages

Part B – Report

Your report (max 10 pages) should include the following sections:

1. Programming Paradigm & Justification

- State which paradigm you used (e.g., procedural, object-oriented, functional).
- Explain why this paradigm is suitable for building a scanner, parser, and syntax tree.

2. Design & Implementation

- Lexical Analyzer: Describe how tokens are identified, how invalid tokens are handled, and how token counts are generated.
- Syntax Analyzer: Explain the parsing approach (e.g., recursive descent), including how errors are detected and reported.
- Error Handling: Briefly outline how your program distinguishes between lexical errors and syntax errors, and how error messages are structured.
- You may include pseudocode, state diagrams, or flowcharts for clarity.

3. Results & Test Cases
- Provide at least five test cases:
    - 3 valid expressions (token stream + counts + syntax tree)
    - 2 invalid expressions (token stream + error messages)
- Present the outputs using screenshots or logs.
- For each case, add a caption and a short explanation (2–3 sentences).
- Summarize the token counts in a simple table.

4. Reflection
- Discuss challenges faced (e.g., handling precedence, designing error messages, token validation).
- Mention improvements you would make if given more time.

Part C – Demo Video

Prepare a 5–7 minute demo video narrated by at least one group member. The video must clearly demonstrate:

1. Source Code Structure
    - Briefly walk through your program files and explain the role of each major component (e.g., lexer, parser, syntax tree).
2. Program Execution
    - Run at least five test cases:
        - 3 valid expressions → show token stream, token counts, and the generated syntax tree.
        - 2 invalid expressions → show token stream (with any invalid tokens flagged) and the error messages produced.
    - Highlight how the program distinguishes between valid and invalid tokens.
    - Show how the program reports errors with details

Submission Guidelines

1. Source Code
- Submit all source code files
- Upload as a .zip file named: **GroupX_Compy_Code.zip**

2. Report
- Submit a PDF report.
- PDF file named: **GroupX_Compy_Report.pdf**

3. Demo Video
- Upload the video to YouTube (Unlisted), Google Drive, or USM eLearn (not advisable).
- Provide the video link inside the report (last page)

*Note: If uploading directly, name as **GroupX_MiniCompiler_Video.mp4***

Assignment 1 Evaluation Criteria

Item	Criteria	10–9	8–6	5–3	2–1	%
Implementation	**Lexical Analysis & Token Stream**	All tokens correctly recognized; stream complete and accurate	Mostly correct with minor errors	Many missed/misclassified tokens	No clear token stream	15
	Token Validation & Counting	Invalid tokens flagged with positions; accurate total and per-type counts	Counts shown but some mismatches; invalid tokens partly detected	Counts incomplete/incorrect	Counts missing; invalid tokens ignored	10
	Syntax Analysis & Syntax Tree	Grammar fully applied; correct syntax trees for valid inputs	Works for simple inputs; errors in complex cases	Trees incomplete/unclear	No syntax trees generated	20
	Error Handling	Clear, precise error messages (Lexical/Syntax) with type, position	Error messages mostly clear but missing details	Vague error messages; not consistent	No error messages; program crashes	15
Report	**Paradigm Justification**	Clear explanation of paradigm and rationale	Some explanation but shallow	Minimal explanation	No explanation	5
	Lexical & Syntax Analyzer Explanation	Well-explained design & logic; diagrams/pseudocode included	Adequate explanation but limited detail	Very basic; hard to follow	Missing explanation	10
	Demonstration Evidence	5 test cases with screenshots + captions; valid & invalid cases shown	Some test cases missing or poorly explained	Few test cases; unclear	No meaningful evidence	10
Demo Video	**Video Clarity & Structure**	Clear narration; logical code walkthrough	Mostly clear but uneven presentation	Unclear in parts; hard to follow	No clear narration/structure	5
	Test Case Demonstration	All 5 test cases demonstrated (tokens, counts, syntax tree, errors)	Most test cases shown but missing some details	Only a few test cases; unclear results	Missing test cases or demo	10