CS 410 | **Compiler Construction** | Phase 2: Parser

EXPECTATIONS

As a team, **review your minimal dialect** (as in Phase 1) and update it only when necessary. Then…

A **Create a context-free grammar** for your minimal dialect. Your grammar should:

- Generate the language of your minimal dialect using necessary and sufficient productions, nonterminals, and terminals.
  - Represent nonterminals in upper-case or title-case without bracketing, and represent terminals in lower-case.
  - Use the tokens accepted by your scanner as the terminals in your grammar.
  - Don't define productions for lexical analysis, such as individual letters or digits.
- Follow either the LL(1) or LR(1) or simpler property.
- Optionally, use an attributed grammar and/or translation grammar.

B **Define an abstract machine** for syntactic analysis of your dialect in step A. Your abstract machine should:

- Use one of these parser models:

Parser Model	**Parsing Direction**	**Grammar**	**Presentation Format**
One-state pushdown machine parser	Top down	LL(1) or simpler	Table of instructions with rows for stack tokens and columns for input tokens
Recursive descent parser	Top down	LL(1) or simpler	Pseudocode functions for all productions
Shift reduce parser	Bottom up	LR(1) or simpler	Instructions for tracing execution and handling shift/reduce and reduce/reduce conflicts

- Accept the sentences of your dialect, but reject any other sentences.
- Follow conventional notation for your parser model.

C **Program a parser** in Java based on your abstract machine in step B. Your parser should:

- Define a set of data structures suitable for encoding your model.
- Input a stream of tokens from the output of your scanner (as in Phase 1) or from a user, file, or standard channel.
- Output a stream of atoms to a user, file, or standard channel. Your stream of atoms should:
  - Use these atom definitions:

<table>
<tr><th>Class</th><th>Atom with Operands</th><th colspan="7">Semantics</th></tr>
<tr><td><b>1</b></td><td><code>(ADD, left, right, result)</code></td><td colspan="7">result = left + right</td></tr>
<tr><td><b>2</b></td><td><code>(SUB, left, right, result)</code></td><td colspan="7">result = left – right</td></tr>
<tr><td><b>3</b></td><td><code>(MUL, left, right, result)</code></td><td colspan="7">result = left × right</td></tr>
<tr><td><b>4</b></td><td><code>(DIV, left, right, result)</code></td><td colspan="7">result = left ÷ right</td></tr>
<tr><td><b>5</b></td><td><code>(JMP, , , , , dest)</code></td><td colspan="7">goto dest unconditionally</td></tr>
<tr><td><b>10</b></td><td><code>(NEG, left, , result)</code></td><td colspan="7">result = –left</td></tr>
<tr><td><b>11</b></td><td><code>(LBL, , , , , dest)</code></td><td colspan="7">label dest</td></tr>
<tr><td rowspan="3"><b>12</b></td><td rowspan="3"><code>(TST, left, right, , cmp, dest)</code></td><td colspan="7">goto dest conditionally, if left cmp right is true,
where cmp is from 0 to 6 and the result is true when…</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr>
<tr><td>always</td><td>equal</td><td>lesser</td><td>greater</td><td>lesser or equal</td><td>greater or equal</td><td>unequal</td></tr>
<tr><td><b>13</b></td><td><code>(MOV, s, , d)</code></td><td colspan="7">d ← s</td></tr>
</table>

- Be hand-written in Java, not be made by a compiler generator (which is reserved for a future assignment).

Assessment

As a team, submit your compiler phase deliverables to the assignment submission page in eCampus.

- **Create and attach a credits table** based on the actual contributions within your team.
  - For each step from A through C (table column), credit precisely 2 to 3 authors and 1 to 2 reviewers.
  - For each team member (table row), credit them as either author, reviewer, or neither on each step.
  - If contributions aren't equitable, notify the instructor in writing before or after submission.
- **Attach your context-free grammar** as a PDF file.
- **Attach your abstract machine** as a PDF file.
- **Attach your parser** as one or more source code files.
- **Verify your submission** with the instructor before the compiler phase is due.