Phase 2 Documentation

All changes were made with reference to phase 2 documentation and project description. Documentation for the tests is available for each test in the /testSuite/phase2 directory.

Change made	**Location in parser.ssl**	**Explanation/Justification**
Updated input token definitions	{ Input Tokens }	🍏 Removed old PT parser input tokens 🍎 **Keywords:** pDiv, pMod, pUntil, pDo, pArray, pProgram, pConst, pType, pProcedure, pBegin, pCase 🍎 **Non-compound tokens:** pColonEquals, pDot 🍏 Added new Like parser input tokens 🍎 **Keywords:** pChoose, pElseif, pFun, pIs, pLike, pPkg, pPublic, pUsing, pVal, pWhen 🍎 **Non-compound tokens:** pSlash, pPercent, pHash, pBang, pBar, pDoubleBar, pPlusEquals, pMinusEquals, pStarEquals, pSlashEquals, pPercentEquals, pDoubleEquals 🍏 Changed string synonym of pNotEqual from “<>” to “!=”
Updated output token definitions	{ Output Tokens }	🍏 Removed old PT sType output token 🍏 Added new Like output tokens 🍎 sLike, sPackage, sPublic, sConcatenate, sRepeatString, sSubstring, sLength, sInitialValue, sCaseElse
Update parser.pt tokens	Different file – parser.pt	🍏 Pasted the contents of parser.def into the corresponding section, replacing the previous pasted parser.def contents
Merge ‘Statements’ rule into ‘Block’ rule	‘Block’ rule	🍏 Like makes no distinction between declarations and statements, so the alternatives from the ‘Statement’ rule were added into the ‘Block’ rule
Added null statement	‘Block’ rule	🍏 Like allows for a null statement (represented by a semicolon) so this was added as a choice in the ‘Block’ rule
Emit sBegin token at beginning of the ‘Block’ rule and sEnd at the end of the rule	‘Block’ rule	🍏 It is desirable to minimize the differences between Like and PT in the semantic phase, so a sequence of Like declarations and statements will always be treated as if it were a PT **begin...end** statement by explicitly emitting those tokens at the beginning and end of the ‘Block’ rule 🍏 This also means that the scope of declarations in a Like statement sequence is from the declaration itself until the end of the sequence

Add ‘pkg’ keyword handling	‘Block’ rule	🍏 Added parsing of packages by adding case for ‘pkg’ keyword inside the ‘Block’ rule and emitting the sPackage token 🍏 Packages is a new Like feature that’s not in PT
Change program handling	‘Program’ rule	🍏 Changed to check for the “using” string synonym instead of “program” at the beginning of a Like program 🍏 Removed pIdentifier token consumed for the program name, since programs are no longer named.
Remove TypeDefinitions, TypeBody, SimpleTypes, BeginStmt	N/A	🍏 ‘TypeDefinition’, ‘TypeBody’, and ‘SimpleTypes’ rules were removed as Like does not allow for typing using the PT ‘type’ keyword 🍏 ‘BeginStmt’ rule removed as Like does not recognize the PT ‘begin’ keyword
Change keywords for constant declarations and procedures	‘Block’ rule	🍏 Changed PT’s ‘const’ case in the ‘Block’ rule to check for the Like equivalent ‘val’ 🍏 Changed PT’s ‘procedure’ case in the ‘Block’ rule to check for the Like equivalent ‘fun’
Change ‘repeat’ statement to ‘repeat while’ statement	‘Block/Statement’ and ‘WhileStmt’ rules	🍏 Moved ‘while’ case to be inside the ‘repeat’ case in the ‘Block/Statement’ rule 🍏 Removed ‘do’ from the ‘WhileStmt’ rule 🍏 PT uses a ‘while (condition) do’’ statement but the equivalent in Like is a “repeat while (condition)…end”
Change ‘repeat…until’ statement to repeat…while’ statement	‘Block/Statement’ and ‘RepeatStmt’’ rules	🍏 Changed ‘RepeatStmt’ call in the ‘Block/Statement’ rule to be the default so that “repeat while (condition)…end” will work with the ‘repeat’ case 🍏 Emit an .sNot after the conditional expression 🍏 Changed ‘until’ inside the ‘RepeatStmt’ block to a ‘while’ 🍏 Like’s “repeat...while (condition)” statement has the opposite meaning to PT’s “repeat...until (condition)”
Public (var, val, fun) + associated rules	‘Block’ rule, ‘PublicValueOrLike’ rule, ‘PublicConstantDeclarations’ rule	🍏 Changed to allow for variables, constants, and functions to be declared public in Like 🍏 A ‘public’ choice was added to the Block rule, with nested choices for ‘var’ keyword that calls ‘PublicValueOrLike’, ‘val’ keyword that calls ‘PublicConstantDeclarations’, and ‘fun’ keyword that is the same as the outer ‘fun’ block, with the addition of an sPublic token emitted following the sIdentifier 🍏 ‘PublicValueOrLike’ is the same as the ‘ValueOrLike’ rule, with the addition of an sPublic

		token emitted following the sIdentifier 🍏 'PublicConstantDeclarations' is the same as the 'ConstantDeclarations' rule, with the addition of an sPublic token emitted following the sIdentifier
Sequence of constant variable declarations	'ConstantDefinitions' rule	🍏 Changed to allow for sequences of constant variable declarations that are separated by commas in Like 🍏 In the 'ConstantDefinitions' rule, recognize sequences using a comma choice and handle with recursion instead of looping on semicolons
Add ValueOrLike and LikeClause rules	'ValueOrLike' rule, 'LikeClause' rule	🍏 Added a 'ValueOrLike' rule to determine whether a variable is declared with an initial value or typed using the 'like' keyword 🍏 'ValueOrLike' called from 'var' and 'public var' in 'Block' rule 🍏 If the statement has '=', the variable is declared with an initial value 🍏 If the statement has ':', the variable does not have an initial value, is typed using the 'like' keyword, and calls the 'LikeClause' rule 🍏 If the statement has '['. the variable is an array declared with a 'like' clause, and calls the 'LikeClause' rule 🍏 The 'LikeClause' rule handles statements with the 'like' keyword and the 'file' keyword preceding a 'like' keyword
Update ProcedureHeading	'ProcedureHeading' rule	🍏 Updated rule to only allow function parameters to be typed using 'like' keyword 🍏 Check for multiple parameters separated by commas 🍏 Check for 'is' keyword following parameter declaration instead of a semicolon
Add Short Form Assignments	''AssignmentOrCallStmt' rule	🍏 Added parsing for Likes new short for assignment statements 🍎 +=, -=, *=, /= and %= 🍏 When using short form assignments (e.g. i += 1), outputting the equivalent token stream (e.g. i = i + 1) makes it so the semantic phase doesn't have to deal with short form assignments
Check for ; at end of statements	'AssignmentOrCallStmt' rule	🍏 Added explicit expectation of a semicolon at the end of the 'AssignmentOrCallStmt' rule that was previously being expected in the 'BeginStmt' rule (which was removed)
Elseif	'IfStmt' rule, 'Block' rule	🍏 An option for the 'elseif' keyword is added within the choice block. It is handled by first emitting an sElse token to the semantic analyzer before calling the 'IfStmt' rule recursively. 🍏 This causes the 'elseif' block in code to be treated as an else block with a nested if

<table>
<tr><td></td><td></td><td>statement within.
🍏 After the call to ‘IfStmt’, the choice block for the ‘if’ keyword handles the ‘end’ and ‘;’ that is expected afterwards. This is because not all options within the choice loop in the ‘Block’ rule require the end keyword, so we need to handle that requirement individually.</td></tr>
<tr><td>Add then to CaseAlt</td><td>‘CaseAlternative’ rule</td><td>🍏 The ‘then’ keyword is expected at the end of a case alternative (before declarations and/or statements are called). It replaces the ‘:’ that was previously expected.
🍏 The ‘Block’ rule is called instead of the ‘Statements’ rule (since it has been deleted).</td></tr>
<tr><td>Add else, when to CaseStmt</td><td>‘‘CaseStmt’ rule
‘Block’ rule</td><td>🍏 At least one ‘CaseAlternative’ is required within the ‘CaseStmt’ rule, so after entering, check that the ‘when’ keyword is provided before calling ‘CaseAlternative’.
🍏 Within the loop, check for more optional ‘when’ keywords to call ‘CaseAlternative’. If ‘when’ is not provided, exit the loop.
🍏 After a case statement has ended by emitting the .sCaseEnd token, optionally check for the ‘else’ keyword. If it is provided, emit the .sCaseElse token (to be handled in the next phase) and call ‘Block’ to handle the declaration or statement.
🍏 After the call to ‘CaseStmt’, the choice block for the ‘if’ keyword handles the ‘end’ and ‘;’ that is expected afterwards. This is because not all options within the choice loop in the ‘Block’ rule require the end keyword, so we need to handle that requirement individually.</td></tr>
<tr><td>Change Operator Syntaxes</td><td>‘AssignmentOrCallStmt’, ‘Expression’ and ‘Term’ and rules</td><td>🍏 Changed PT operator syntaxes to use the new Like operator syntaxes
🍏 ‘:=’ changed to ‘=’ in ‘AssignmentOrCallStmt’ rule
🍏 ‘=’, and ‘<>’ changed to ‘==’ and ‘!=’ in ‘Expression’ rule
🍏 ‘div’, and ‘mod’ changed to ‘/’ and ‘%’ in ‘Term’ rule
<table>
<tr><th>PT</th><th>Like</th></tr>
<tr><td>:=</td><td>=</td></tr>
<tr><td>=</td><td>==</td></tr>
<tr><td><></td><td>!=</td></tr>
<tr><td>div</td><td>/</td></tr>
<tr><td>mod</td><td>%</td></tr>
</table></td></tr>
<tr><td>LikeString</td><td>‘Factor’ and ‘Term’ rules</td><td>🍏 In the ‘Factor’ rules, add an option to handle a pHash input, by calling the ‘Factor’ rule to handle the string (or identifier) that follows the hash.</td></tr>
</table>

		🍏 In the ‘Term’ rule, for the pSlash option, add a nested choice block to differentiate between emitting sDivide or sSubstring (when the factor is followed by a colon and another factor). Also add options for pBar and pDoubleBar to handle sConcatenate and sRepeatString respectively. 🍏 Note: errors such as concatenating an integer to a string, or repeating a string by something that is not an integer are not caught in the parsing phase (similar to how the PT compiler does not catch these errors in the parser phase for the old PT operations such as div and mod).