Project Phase 1: Updates, Clarifications, and Suggestions

Note: In cases where this document and the phase 1 project description document disagree, the updates in this document supersede the original details.

Updates

2.3.14.a.
An integer constant is now unsigned. (The minus sign is handled by the grammar rules.)

2.3.14.b.
Use longest match when scanning.

2.3.14.c.
Nesting of comments is no longer allowed.

2.11.14.a.
Precedence rules are “grouped” as with other common languages, so addition and subtraction occur left to right and with equal precedence.

Precedence	Operator	Associativity
1	( )	Innermost to outermost grouping
2	unary_minus	Right to left
3	* /	Left to right
4	+ -	Left to right
5	= <> > < >= <=	N/A: See the project document’s “Operators” section.
6	&	Left to right
7	|	Left to right
8	:=	N/A: See the project document’s “Assignment” section.

2.11.14.b.
The “record” types were removed from the language, so ignore any record-related details in the “Assignment” section in the project document.

2.13.14.a.
The following punctuation tokens were holdovers from record types and are no longer part of the language: PERIOD, LBRACE, and RBRACE.

2.13.14.b.
Add the following production rules to the grammar:

<stat> → `break` ;
<const> → `nil`

2.13.14.c.
Example 1 at the end of the project description has been modified slightly. See the test input files for the updated version.

2.14.14.a.
The keyword `return` is now part of the language. Its corresponding token is RETURN.

2.14.14.b.
Add the following production rule to the grammar:

<stat> → `return` <expr> ;

2.16.14.a.
The following production rule is now obsolete:

<type> → `array` [INTLIT] `of` <type-id>

Use the following rules instead:

<type> → `array` [INTLIT] <type-dim> `of` <type-id>
<type-dim> → [INTLIT] <type-dim>
<type-dim> → NULL

2.16.14.b.
The following production rule is now obsolete:

<stat> → `id` (<expr-list>) ;

Use the following rules instead:

<stat> → <opt-prefix> `id` (<expr-list>) ;
<opt-prefix> → <lvalue> :=
<opt-prefix> → NULL

Clarifications

2.3.14.a. There is a *type* keyword in the language that corresponds to the token TYPE. Refer to the grammar rules to decide the token to use for int, string, and user-defined types.
2.3.14.b. An integer constant may contain an arbitrary number of leading zeros.

Suggestions

2.3.14.a. A preprocessing step can be used to remove whitespace and/or comments.
2.3.14.b. You may find it helpful to implement keywords as a subset of identifiers. That is, you may want to first recognize an identifier (keyword or not) as such, then check if it is a keyword,

and, if it is, return the appropriate keyword token instead of ID.