Grammar for Dopl

Each rule of the grammar has a ‘non-terminal’ symbol on the left-hand side of the ‘::=’ symbol. On the right-hand side of ‘::=’ is a mix of ‘terminal’ symbols, non-terminal symbols and grammatical notation, terminated with a semicolon. The meaning of the grammatical notation is described below.

- The notation `::=` means 'is defined as'.
- A semicolon marks the end of each non-terminal 'rule'.
- Parentheses group terminal and non-terminal symbols as a single item.
- A `?` means the single preceding item is optional or the preceding parenthesized items as a group are optional. For instance:
  `a b ? c` means that both `ac` and `abc` are valid and `(a b) ? c` means that both `c` and `abc` are valid.
- One or more items enclosed between curly brackets means the enclosed items occur zero or more times. For instance:
  `{ a b }` means that ab may occur 0, 1, 2 or more times; e.g., `abab`, `abababab`, `ababababababab` are all valid.
- A `|` separates alternatives. For instance:
  `{ a | b | c }` means that either `a` or `b` or `c` is valid.
- Items all in upper-case are terminal symbols: an identifier (`IDENTIFIER`), keyword (`START`, `FINISH`, etc.), integer constant (`INTEGER_CONSTANT`) or character constant (`CHARACTER_CONSTANT`).
- Items enclosed in single quotes are terminal symbols, e.g. `';'` and `'<-'`.

In the grammar be careful to distinguish between those characters not enclosed between single-quote characters (e.g., `::=` and `;`) and those that look similar but are enclosed (e.g., `';'`).

```
program ::= START declarations statements FINISH ;

declarations ::= { declaration ';' } ;

declaration ::=     dataType identifiers ;

dataType ::= INTEGER | CHARACTER | LOGICAL;

identifiers ::= IDENTIFIER { ',' IDENTIFIER } ;

statements ::= { statement ';' } ;

statement ::=  assignment |
conditional |
print |                 loop
;

assignment ::= IDENTIFIER '<-' expression ;

conditional ::=     IF expression THEN statements
                    ( ELSE statements ) ? ENDIF ;

print ::= PRINT expression ;

loop ::= LOOPIF expression DO statements ENDLOOP;

expression ::= term { binaryOp term } ;

binaryOp ::= arithmeticOp | logicalOp | relationalOp ;

arithmeticOp ::= '.plus.' | '.minus.' | '.mul.' | '.div.' ;

logicalOp ::=       '.and.' | '.or.' ;

relationalOp ::=    '.eq.' | '.ne.' |
                    '.lt.' | '.gt.' | '.le.' | '.ge.' ;


term ::=       INTEGER_CONSTANT |
               CHARACTER_CONSTANT |
               IDENTIFIER |
               '(' expression ')' |
               unaryOp term ;

unaryOp ::=    '.minus.' | '.not.' ;
```

Comments

Comments are not permitted in Dopl programs.

Example program

The following (nonsense) program illustrates the main syntactic features defined by the grammar.

```
     start
         integer num, sum;
     character ch;       sum
     <- 0;     num <- 1;
     ch <- "a";     loopif
     ch .lt. "z"     do
             if (num .div. 3) .ne. 0 .and. (ch .ne. "y")
```

```
        then
            sum <- sum .plus. num .mul. 2;
ch <- ch .plus. 1;
        else
            sum <- sum .minus. num;
            ch <- "m";
endif;         num <- num
.plus. 1;
    endloop;
print sum;
print ch;
finish
```

Definitions of terminal symbols: lexical analysis

- The following are all keywords of the language: `character`, `do`, `else`, `endif`, `endloop`, `finish`, `if`, `integer`, `logical`, `loopif`, `print`, `start`, `then`. They are case-sensitive and must be all lower-case in the source. These are represented in the grammar by upper-case versions (e.g., `LOOPIF`).
- All symbols within a pair of single quote characters in the grammar are literals that will be found in the source code; e.g. , `'<-'`.
- IDENTIFIER: A sequence of one or more alphabetic, digit or underscore characters, of which the first must be alphabetic. E.g., `Ng1_f3` is valid but `1e4` is not. It is not permitted to define a variable that is the same as a keyword.
- INTEGER_CONSTANT: A sequence of one or more digits (0-9).
- CHARACTER_CONSTANT: A single character enclosed within a pair of *double-quote* characters. E.g., `"!"`, `"a"`, etc. Multi-character character constants are not permitted.
- All symbols between a pair of periods (full stops) are operators; e.g., `.plus.`

Limited type checking

Type checking involves ensuring that variables have been defined and that data types are consistent in assignments and expressions. Identifiers and associated data type information are normally stored in a 'symbol table'. So, in order to complete this part, you will need to maintain a symbol table for identifiers.

Some type checking of expressions and assignments is required, so you will also need to associate a 'data type' (*character*, *integer, logical*) with each expression, according to the following rules:

- If the expression contains at least one `relationalOp`, `logicalOp` or unary logical negation operator (`.not.`) then its data type is *logical*.
- If the expression is not *logical* and it contains at least one `CHARACTER_CONSTANT` or an `IDENTIFIER` of data type `CHAR` then its data type is *character*.
- If the expression is neither *logical* nor *character* then its data type is *integer*.

Any of the following should result in a failed parse, with the error message given previous being printed and the parse stopping.

- Use of an `IDENTIFIER` in a statement that has not been declared within a `variable`.

- Assignment of an expression of one type to a variable of a different type in an `assignment`. It is only legal to assign a character expression to a variable of type character, an integer expression to a variable of type integer, and a logical expression to a variable of type logical.

- Use of an expression of type integer or character in the expression (condition) of either an `conditional` or `loop`. All condition expressions must be of type logical.