Lexical Conventions and Partial Grammar for Pascal

Lexical Conventions

The symbols of the Pascal vocabulary are divided into the following classes: identifiers, numbers, strings, operators, delimiters, and comments. In the following notation,

- The bar | means you must have one of the two items it separates
- Curly brackets {} are shorthand notation for having zero or more items
- Square brackets [] stands for optional meaning having zero or one item
- Characters to be represented as is are in single quotes.

Note that nothing (identifiers, keywords, standard identifiers, etc.) in Pascal is case-sensitive. The rules governing their representation in terms of the standard character set are the following:

(1) Identifiers are sequences of letters and digits. The first character must be a letter. Identifiers are not case-sensitive, i.e., a lowercase and uppercase letter are considered to be the same character.

Note that in all documentation, lowercase is used, but uppercase is acceptable.

```
identifier --> letter { letter | digit }
```

(2) Numbers are integers or reals (floats). Integers are denoted by sequences of digits. They must not contain spaces. Numbers are unsigned. Integer number examples include `0 1 567`

```
number     -->    integer | real
integer    -->    digit {digit}
```

Reals include a decimal point (digits must surround the decimal point) and an optional exponential part (as in scientific notation). An 'E' can be used as well as 'e' . Real number examples include `0.5 2.0 3.456 7.89e23 7.8e+4 7.9e-456`

```
real          -->    integer '.' integer [exponent]
exponent      -->    expDesignator [ '+' | '-' ] integer
expDesignator -->    'e' | 'E'
```

(3) Strings are sequences of any characters enclosed in quote marks. In order that the closing quote is recognized unambiguously, the string itself cannot contain a quote mark. To allow strings with single or double quote marks, a string may be enclosed within single or double quote marks. Strings containing single quotes would be enclosed in double quotes, and strings containing double quotes would be enclosed in single quotes.

```
string -->  ' " ' {character} ' " ' | " ' " {character} " ' "
```

(4) Operators and delimiters are either special characters or reserved words. Reserved words cannot be used as identifiers.

The operators and delimiters composed of special characters are:

```
:=              assignment
+               (unary or binary), addition, set union
-               (unary or binary), subtraction, set difference
*               multiplication, set intersection
/               real division
div             integer division
mod             integer modulus
=               equal
<>              unequal
<               less than
>               greater than
<=              less than or equal
>=              greater than or equal
in              set membership
not             logical negation
or              logical disjunction
and             logical conjunction
()              parentheses
[]              index brackets for arrays or sets
{}              comment brackets
(* *)           comment brackets
, . ; : .. ^    other symbols
```

Operators are defined by the following:

```
UnaryOperator  --> '+' | '-'
MultOperator --> '*' | '/' | div | mod | and
AddOperator --> '+' | '-' | or
Relation    --> '=' | '<>' | '<' | '>' | '<=' | '>=' | in
```

The reserved words are enumerated in the following list (although we will not use all of them).

```
and         downto      if          or          then
array       else        in          packed      to
begin       end         label       procedure   type
case        file        mod         program     until
const       for         nil         record      var
div         function    not         repeat      while
do          goto        of          set         with
```

Standard identifiers are as follows:

```
Constants:  False, True
Types:      Integer, Boolean, Real, Char
Functions:  Abs, ArcTan, Chr, Cos, EOF, EOLN, Exp, Ln, Odd, Ord,
            Pred, Round, Sin, Sqr, Sqrt, Succ, Trunc
Procedures: Get, New, Dispose, Pack, Page, Put, Read, Readln,
            Reset, Rewrite, Unpack, Write, Writeln
```

From all the functions and procedures, you need only implement the I/O routines:

```
Write  Writeln  Read  Readln
```

And for dynamic allocation and deallocation:

```
New  Dispose
```

(5) Comments may be inserted between any two symbols. They are arbitrary sequences of characters enclosed in the comment brackets `(* *)` or braces `{}`. Comments may not be nested. Comments are skipped by compilers and serve as additional information for the human reader. They may also serve to signal instructions to the compiler.

Grammar

My naming convention: All non-terminals start with a capital letter. All lexical elements (terminals) are lowercase. All reserved words and lexical tokens such as ident are the word itself preceded with the letter 'y' , for example, if is yif, ident is yident, etc.

```
CompilationUnit      -->   ProgramModule
ProgramModule        -->   yprogram  yident  ProgramParameters  ';'  Block  '.'
ProgramParameters    -->   '('  IdentList  ')'
IdentList            -->   yident {','  yident}

Block                -->   [Declarations]  StatementSequence
Declarations         -->   [ConstantDefBlock]
                           [TypeDefBlock]
                           [VariableDeclBlock]
                           [SubprogDeclList]
ConstantDefBlock     -->   yconst  ConstantDef  ';' {ConstantDef  ';'}
TypeDefBlock         -->   ytype  TypeDef  ';'  {TypeDef  ';'}
VariableDeclBlock    -->   yvar  VariableDecl  ';'  {VariableDecl  ';'}
ConstantDef          -->   yident  '='  ConstExpression
TypeDef              -->   yident  '='  Type
VariableDecl         -->   IdentList  ':'  Type

ConstExpression      -->   [UnaryOperator]  ConstFactor
                         | " ' "  ycharacter " ' "
                         | ynil
ConstFactor          -->   yident
                         | ynumber
                         | ytrue  |  yfalse  | ynil
Type                 -->   yident
                         | ArrayType
                         | PointerType
                         | RecordType
                         | SetType
ArrayType            -->   yarray  '['  Subrange {',' Subrange}  ']'  yof Type
Subrange             -->   ConstFactor '..' ConstFactor
                         | " ' "  ycharacter  '..'  ycharacter  " ' "
RecordType           -->   yrecord  FieldListSequence  yend
SetType              -->   yset  yof  Subrange
PointerType          -->   '^'  yident
FieldListSequence    -->   FieldList  {';'  FieldList}
FieldList            -->   IdentList  ':'  Type

StatementSequence    -->   ybegin  Statement  {';'  Statement}  yend
Statement            -->   Assignment
                         | ProcedureCall
                         | IfStatement
                         | CaseStatement
                         | WhileStatement
                         | RepeatStatement
                         | ForStatement
                         | IOStatement
                         | MemoryStatement
                         | StatementSequence
                         | empty
```

```
Assignment          -->   Designator  ':='  Expression
ProcedureCall       -->   yident  [ActualParameters]
IfStatement         -->   yif  Expression  ythen  Statement
                             [yelse  Statement]
CaseStatement       -->   ycase  Expression  yof  Case  {';'  Case}  yend
Case                -->   CaseLabelList  ':'  Statement
CaseLabelList       -->   ConstExpression  {','  ConstExpression }
WhileStatement      -->   ywhile  Expression  ydo  Statement
RepeatStatement     -->   yrepeat  StatementSequence  yuntil  Expression
ForStatement        -->   yfor  yident  ':='  Expression  WhichWay  Expression
                             ydo  Statement
WhichWay            -->   yto  |  ydownto
IOStatement         -->   yread  '('  DesignatorList  ')'
                        | yreadln  [ '(' DesignatorList  ')' ]
                        | ywrite  '('  ExpList  ')'
                        | ywriteln  [ '('  ExpList  ')' ]
DesignatorList      -->   Designator  {','  Designator }
Designator          -->   yident  [ DesignatorStuff ]
DesignatorStuff     -->   {'.'  yident  |  '['  ExpList  ']'  |  '^' }
ActualParameters    -->   '('  ExpList  ')'
ExpList             -->   Expression  { ','  Expression }
MemoryStatement     -->   ynew  '('  yident  ')'  | ydispose '('  yident  ')'

Expression          -->   SimpleExpression  [ Relation  SimpleExpression ]
SimpleExpression    -->   [UnaryOperator]  Term  {AddOperator  Term}
Term                -->   Factor  {MultOperator  Factor}
Factor              -->   ynumber
                        | ystring | ytrue  |  yfalse  | ynil
                        | Designator
                        | '('  Expression  ')'
                        | ynot Factor
                        | Setvalue
                        | FunctionCall
Setvalue            -->   '[' [Element  {','  Element} ] ']'
FunctionCall        -->   yident  ActualParameters
Element             -->   ConstExpression  ['..'  ConstExpression ]

SubprogDeclList     -->   {ProcedureDecl ';' | FunctionDecl ';'}
ProcedureDecl       -->   ProcedureHeading  ';'  Block
FunctionDecl        -->   FunctionHeading  ':'  yident  ';'  Block
ProcedureHeading    -->   yprocedure  yident  [FormalParameters]
FunctionHeading     -->   yfunction  yident  [FormalParameters]
FormalParameters    -->   '('  OneFormalParam  {';'  OneFormalParam}  ')'
OneFormalParam      -->   [yvar]  IdentList  ':'  yident
```

Notes

The above grammar is not a full-blown Pascal, although it is a large subset. The following describes limited aspects of the language:

- -- In an IfStatement, WhileStatement, and RepeatStatement the Expression must be Boolean
- -- In a CaseStatement, the Expression must be one of type Char, Integer, Boolean
- -- All labels have been eliminated from the grammar
- -- Enumeration has been eliminated from the grammar
- -- Subrange types have been eliminated from the grammar (except for arrays)
- -- Packed types and File type have been eliminated from the grammar
- -- No gotos, withs, variant records (like union)
- -- No Formatting on read and write