EL lenguaje que tomaremos como caso de estudio es mini-pascal.

Descarga aqui su sintaxis:

Syntax of Mini-Pascal (Welsh & McKeag, 1980) tomado de https://www.cs.helsinki.fi/u/vihavain/k10/okk/minipascal/minipascalsyntax.html

A la sintaxis dada, le he hecho algunos ajustes que definirán a nuestro caso de estudio: eliminé de la sintaxis original lo que refiere a <*array type*> y <*procedure declaration part*>. Así también, consideré para los tipos de datos <*type*> ::= **Integer | Real | String**

El texto en negrita se refiere a palabras reservadas y lo que esta entre { } significa la cerradura de Kleene *

Sintaxis en orden descendente recursivo

en

<program> ::= **program** <*identifier*> **;** <*block*> **.**

<*block*> ::= <*variable declaration part*>
<*statement part*>

<*variable declaration part*> ::= <*empty*> |
var <*variable declaration*> **;**
{ <*variable declaration*> **;** }

<*variable declaration*> ::= <*identifier* > { **,** <*identifier*> } **:** <*type*>

<*type*> ::= **Integer** | **Real** | **String**

<*statement part*> ::= <*compound statement*>

<*compound statement*> ::= **begin** <*statement*>{ **;** <*statement*> } **end**

<*statement*> ::= <*simple statement*> **|** <*structured statement*>

<simple statement> ::= *<assignment statement>* | *<read statement>* | *<write statement>*

<assignment statement> ::= *<variable>* := *<expression>*

<read statement> ::= **read (** *<input variable>* { **,** *<input variable>* } **)**

<input variable> ::= *<variable>*

<write statement> ::= **write (** *<output value>* { **,** *<output value>* } **)**

<output value> ::= *<expression>*

<structured statement> ::= *<compound statement>* | *<if statement>* | *<while statement>*

<if statement> ::= **if** *<expression>* **then** *<statement>* | **if** *<expression>* **then** *<statement>* **else** *<statement>*

<while statement> ::= **while** *<expression>* **do** *<statement>*

<expression> ::= *<simple expression>* | *<simple expression>* *<relational operator>* *<simple expression>*

<simple expression> ::= *<sign>* *<term>* { *<adding operator>* *<term>* }

<term> ::= *<factor>* { *<multiplying operator>* *<factor>* }

<factor> ::= *<variable>* | *<constant>* | **(** *<expression>* **)** | **not** *<factor>*

<relational operator> ::= **= | <> | < | <= | >= | >**

<sign> ::= + | - | *<empty>*

<adding operator> ::= + | - | **or**

<multiplying operator> ::= * | **div** | **and**

<variable> ::= *<entire variable>*

<entire variable> ::= *<variable identifier>*

<variable identifier> **::=** *<identifier>*

Lexical grammar

<constant> **::=** *<integer constant>* **|** *<character constant>* **|** *<constant identifier>*

<constant identifier> **::=** *<identifier>*

<identifier> **::=** *<letter>* { *<letter* **|** *digit>* }

<letter or digit> **::=** *<letter>* **|** *<digit>*

<integer constant> **::=** *<digit>* { *<digit>* }

<character constant> **::=** **'***< any character other than ' >***'** **|** **''''**

<letter> **::=** a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<digit> **::=** 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<special symbol> **::=** + | - | * | = | <> | < | > | <= | >= | (|) | [|] | **:=** | . | , | ; | : | .. | **div** | **or** | **and** | **not** | **if** | **then** | **else** | **of** | **while** | **do** | **begin** | **end** | **read** | **write** | **var** | **array** | **procedure** | **program**

<predefined identifier> **::=** integer | Boolean | true | false