En nuestro lenguaje la sintaxis estará inspirada en el lenguaje C, con palabras reservadas en ingles.

Nuestra estructura de datos elemental serán las listas con subprogramas organizados como funciones.

Las estructuras de control adicionales serán for con sintaxis de Pascal.

<Programa> ::= <Cabecera\_programa> <bloque>

<bloque> ::= <Inicio\_de\_bloque>

<Declar\_de\_variables\_locales>

<Declar\_de\_subprogs>

<Sentencias>

<Fin\_de\_bloque>

<Declar\_de\_subprogs> ::= <Declar\_de\_subprogs> <Declar\_subprog>

|

<Declar\_subprog> ::= <Cabecera\_subprograma> <bloque>

<Declar\_de\_variables\_locales> ::= <initVar>

<Variables\_locales>

<endVar>

|

<Cabecera\_programa> ::= main

<Inicio\_de\_bloque> ::= {

<Fin\_de\_bloque> ::= }

<Variables\_locales> ::= <Variables\_locales> <Cuerpo\_declar\_variables>

| <Cuerpo\_declar\_variables>

<Cuerpo\_declar\_variables> ::= <tipo><list\_id> ;

<Cabecera\_subprog> ::= <tipo><id>(<argumentos>)

<argumentos> ::= <argumentos> , <argumento>

<argumento> ::= <tipo><id>

<Sentencias> ::= <Sentencias> <Sentencia>

| <Sentencia>

<Sentencia> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while>

| <sentencia\_entrada>

| <sentencia\_salida>

| <sentencia\_return>

| <sentencia\_for>

| <sentencia\_list>

<sentencia\_asignacion> ::= <id> = <expresion>;

<sentencia\_if> ::= if ( <expresion> ) then <Sentencia>

| if ( <expresion> ) then <Sentencia> else <Sentencia>

<sentencia\_while> ::= while ( <expresion> ) <Sentencia>

<sentencia\_entrada> ::= <nomb\_entrada> <list\_id>;

<sentencia\_salida> ::= <nomb\_salida> <list\_expresiones\_o\_cadena>;

<sentencia\_return> ::= return <expresion> ;

<sentencia\_for> ::= for ( <expresion> to <expresion>, <expresion> ) do <Sentencia>

| for ( <expresion> ) do <Sentencia>

<sentencia\_list> ::= <expresion> >>

| <expresion> <<

| $ <expresion>

<expresion> ::= ( <expresion> )

| <op\_unario> <expresion>

| <expresion> <op\_binario> <expresion>

| <id>

| <constante>

| <funcion>

| <expresion> ++ <expresion> @ <expresion>

<funcion> ::= <id> ( <list\_expresiones> )

<list\_expresiones\_o\_cadena> ::= <list\_expresiones\_o\_cadena> , <exp\_cad>

| <exp\_cad>

<exp\_cad> ::= <expresion>

| <cadena>

<cadena> ::= “<cadena> <const\_char>”

| <const\_char>

<constante> ::= <const\_int>

| <const\_double>

| <const\_boolean>

| <const\_char>

| <const\_list\_int>

| <const\_list\_double>

| <const\_list\_boolean>

| <const\_list\_char>

<list\_expresiones> ::= <list\_expresiones>, <expresion>

| <expresion>

<op\_unario> ::= !

| ++

| --

| +

| -

| #

| ?

<op\_binario> ::= +

| -

| \*

| /

| <

| <=

| >

| >=

| ==

| and

| or

| xor

| @

| --

| %

| \*\*

<tipo> ::= <tipo\_elemental>

| list of <tipo\_elemental>

<tipo\_elemental> ::= int

| double

| boolean

| char

<const\_int> ::= <int><dig>

| <dig>

<const\_double> ::= <const\_int>.<const\_int>

<const\_boolean> ::= <true>

| <false>

<const\_char> ::= ascii

<const\_list\_int> ::= [<list\_int>]

<list\_int> ::= <list\_int> , <const\_int>

| <const\_int>

<const\_list\_double> ::= [<list\_double>]

<list\_double> ::= <list\_double> , <const\_double>

| <const\_double>

<const\_list\_boolean> ::= [<list\_boolean>]

<list\_boolean> ::= <list\_boolean> , <const\_boolean>

| <const\_boolean>

<const\_list\_char> ::= [<list\_char>]

<list\_char> ::= <list\_char> , <const\_char>

| <const\_char>

<list\_id> ::= <list\_id> , <id>

| <id>

<id> ::= <id> <char>

| <char>

| <id> <dig>

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