Instituto Tecnológico y de Estudios Superiores de Monterrey

**Desarrollo de aplicaciones avanzadas de ciencias computacionales (Gpo 501)**

**A blue and black logo

Description automatically generated**

**Proyecto de compiladores**

Leonardo Mojica Amézquita - A00571960

03/05/2024

El proyecto esta basado en los siguientes diagramas.

A diagram of a computer program

Description automatically generated**Lista de tokens:**

Símbolos:

* (): paréntesis para encerrar expresiones, y para el uso de funciones.
* {}: llaves que se usan para dar inicio y fin de un “body”.
* []: Corchetes que se usan para encerrar la lógica de una función.
* ;: punto y coma que sirve como un símbolo terminador, el cual se pone cada que termines una asignación de id, llames a una función, declares una función, termines un ciclo, hagas una condición, asignación, etc.
* +: símbolo de suma, se usa para sumar variables.
* -: simbol de resta, se usa para restar variables.
* \*: asterisco que se usa como símbolo de multiplicación.
* /: símbolo que se usa para división.
* =: símbolo de asignación.
* ,: coma, se usa como separador, por ejemplo, para declarar varias variables, en una sola línea, o para poner los parámetros de una función.
* <: menor que, símbolo para comparar dos expresiones.
* >: mayor que, símbolo para comparar dos expresiones.
* !=: símbolo diferente de para comparar dos expresiones
* :: dos puntos que van entre id y su tipado, se usa para especificar el tipado de una variable.

Palabras reservadas:

* program: indica el inicio del programa
* main: indica el inicio del bloque de código principal de tu programa
* end: indica el final del main
* var: indica el inicio de una declaración de variable
* int: indica que tu variable es un entero
* float: indica que tu variable es flotante
* print: función reservada para mostrar información en consola
* if: indica el inicio del condicional if
* else: indica el inisio de lo que se hace en caso de que no se cumpla el if
* void: indica el inicio de una función
* while: indica el inicio de un ciclo while, donde se pondrá la condicional
* do: indica el inicio del cuerpo del while donde se indicará lo que se hará mientras el while se cumpla

Tokens con regex:

* id
  + El id tiene que empezar con una letra (mayúscula o minuscula) y puede seguir con 0 o n letras más (mayúscula o minúscula), o 0 o n números o 0 o n guiones bajo, en cualquier orden.
  + [A-Z|a-z] [A-Z|a-z|0-9|\_]\*
* cte.string
  + una cadena de caracteres (cualquier carácter menos salto de linea), la cual esta encerrada por dos comillas dobles (“”), o dos comillas sencillas (‘’).
  + ("[^\n|"]\*") | ('[^\n|']\*'), el símbolo ^ representa todos los caracteres menos la lista de caracteres a continuación.
* cte.int
  + Toma valor de números enteros decimales.
  + [0-9]+
* cte.float
  + Únicamente acepta valores numéricos seguidos de un único punto (.) seguido de 1 o más caracteres numéricos
  + [0-9]+ (.) [0-9]+

**Gramática libre de contexto**

<Program> 🡪 program id ; <a\_vars> <a\_funcs> main <BODY> end

<a\_vars> 🡪 ε | <VARS>

<a\_funcs> 🡪 ε | <FUNCS> <a\_funcs>

<VARS> 🡪 var <list\_vars>

<list\_vars> 🡪 id <list\_id> : <TYPE>; <more\_vars>

<more\_vars> 🡪 ε | id <list\_id> : <TYPE>; <more\_vars>

<list\_id> 🡪 ε | , id

<TYPE> 🡪 int | float

<Body> 🡪 {<List\_statements>}

<List\_statements> 🡪 ε | <STATEMENT> <List\_statements>

<STATEMENT> 🡪 <ASSIGN> | <CONDITION> | <CYCLE> | <F\_Call> | <Print>

<Print> 🡪 print (<List\_Expresion>);

<List\_Expresion> 🡪 <exp\_o\_string> |<exp\_o\_string> , <List\_Expresion>

<exp\_o\_string> 🡪 <EXPRESION> | cte.string

<ASSIGN> 🡪 id = <EXPRESION> ;

<CYCLE> 🡪 while <Body> do ( <EXPRESION> );

<CONDITION> 🡪 if (<EXPRESION>) <Body> <ELSE>;

<ELSE> 🡪 else <Body> | ε

<EXPRESION> 🡪 <EXP> < comparar\_exp>

<comparar\_exp> 🡪 < <EXP> | > <EXP> | != <EXP> | ε

<EXP> 🡪 <List\_Terminos>

<List\_Terminos> 🡪< TERMINO > <next\_termino>

<next\_termino> 🡪 <sum\_rest> < TERMINO ><next\_termino> | ε

<sum\_rest> 🡪 + | -

<TERMINO> 🡪 <List\_Factor>

<List\_Factor> 🡪<FACTOR> <next\_Factor>

<next\_Factor> 🡪 <mult\_divt> <FACTOR><next\_factor> | ε

<mult\_div> 🡪 \* | /

<FACTOR> 🡪 (<EXPRESION>) | <sum\_rest><id\_cte> |<id\_cte>

<id\_cte> 🡪 id | <CTE>

<CTE> 🡪 cte\_int | cte\_float

<FUNCS> 🡪 void id (<list\_params>) [<var\_no\_var> <BODY>];

<list\_params> 🡪 id : <TYPE> | <more\_params>

<more\_params> 🡪 ε | , id : <TYPE> <more\_params>

<var\_no\_var> 🡪 ε | <VARS>

<F\_Call> 🡪 id ();

<f\_list\_expresion> 🡪 ε | <EXPRESION> <f\_more\_expresion>

<f\_more\_expresion> 🡪 ε | ,<EXPRESION><f\_more\_expresion>