FACULTAD DE INGENIERÍAS Y ARQUITECTURA
ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

COMPILADORES

Integrantes

Johan Leonardo Marquez Zúñiga
Diego Paolo Nova Rosas
Angelica Valeria Castillo Tovar
Renzo Josue Murrillo Alvarez

Introducción

Motivación

Ofrecer una sintaxis simple pero estructurada que permita identificar de forma clara las construcciones fundamentales de un lenguaje de programación. Como funciones, estructuras de control, asignaciones y expresiones.
Nuestro lenguaje se inspira en lenguajes de alto nivel como Python y C, pero con una notación simplificada y un conjunto reducido de tokens. Esto permite centrarse en el análisis léxico y sintáctico, así como en la construcción del árbol de derivación para su posterior visualización o análisis semántico.

Descripción

Sentencias(S)

- Una declaración (DECLARACION)
- Una secuencia de sentencias ($\mathbf{S} \rightarrow \mathbf{S}$)
- Vacía ($\mathbf{S} \rightarrow \epsilon$)

Esto permite que las declaraciones se agrupen o se definan de forma individual.

Declaraciones(Declaración)

- Un símbolo especial: (#)
- Un identificador (id) usado como nombre de variable
- Una declaración extendida (DECL EXTRA), que permite asignaciones

Declaraciones extendidas(DECL EXTRA)

- Un operador de asignación (=)
- Una expresion (E)

Expresiones(E)

- Un término(T) que puede ser un id o un número(#)
- Una continuación de expresión (E') que permite encadenar operadores

Extensión de expresiones(E')

Sirve para definir operaciones aritméticas:
- Un operador como el(+)

- Un nuevo término (T)
- Otra extensión (E') o el fin de la expresión (ϵ)

Operadores(OPER)

- Solo se define el operador de suma (+), lo cual restringe las operaciones posibles.

Especificación léxica

A continuación, se listan los tokens utilizados por el analizador léxico, con sus respectivas expresiones regulares:

Token	Expresión Regular	Descripción
FUNK_RET	funk_ret	Define una función que retorna un valor.
FUNK_NOT_RET	funk_not_ret	Define una función que no retorna valor.
RET	ret	Palabra reservada para retornar un valor desde una función.
IF	if	Estructura condicional: ejecuta un bloque si se cumple la condición.
ELSE	else	Alternativa del if si la condición es falsa.
WHILE	while	Bucle que se ejecuta mientras se cumpla una condición.
FOR	for	Bucle con inicialización, condición y actualización.
IGUAL	=	Operador de asignación de valores a variables.
ADD	+	Operador de suma aritmética.
LESS	-	Operador de resta aritmética.
MULT	*	Operador de multiplicación.
DIV	/	Operador de división.
MENOR_IGUAL	<=	Operador de comparación: menor o igual que.
MAYOR_IGUAL	>=	Operador de comparación: mayor o igual que.
DIFERENTE	<>	Operador de comparación: distinto.
IGUAL_QUE	==	Operador de comparación: igualdad.
MENOR	<	Operador de comparación: menor que.
MAYOR	>	Operador de comparación: mayor que.
AND	@	Operador lógico AND.
OR	//	Operador lógico OR.
LEFT_PAR	(	Paréntesis izquierdo, usado para agrupar expresiones.
RIGHT_PAR	)	Paréntesis derecho.
INICIO	{	Delimita el inicio de un bloque de código.
FIN	}	Delimita el fin de un bloque de código.
COMA	,	Separa elementos, como parámetros en funciones.
DECLARACION DE VARIABLE	#	Simboliza una declaración de variable.
NUM	num	Representa un número entero o real.
ID	id	Representa un identificador, como nombres de variables o funciones.

Gramática Original

S $\rightarrow$ **DECLARACION S**

S $\rightarrow$ **FUNCION S**

S $\rightarrow$ **CICLO S**

S $\rightarrow$ **IF ELSE S**

S $\rightarrow$ **E S**

S $\rightarrow \epsilon$

DECLARACION $\rightarrow \#$ *id* **DECL_EXTRA**

DECL_EXTRA $\rightarrow =$ **E**

DECL_EXTRA $\rightarrow \epsilon$

FUNCION $\rightarrow$ **FUNK_RET**

FUNCION $\rightarrow$ **FUNK_NOT_RET**

FUNK_RET $\rightarrow$ funk_ret(**PARAMS**){**INSTRUCCIONES** ret **E**}

FUNK_NOT_RET $\rightarrow$ funk_not_ret(**PARAMS**){**INSTRUCCIONES**}

PARAMS $\rightarrow$ *id* **PARAMS'**

PARAMS' $\rightarrow ,$ *id* **PARAMS'**

PARAMS' $\rightarrow \epsilon$

INSTRUCCIONES $\rightarrow$ **SENTENCIA INSTRUCCIONES**

INSTRUCCIONES $\rightarrow \epsilon$

ASIGNACION $\rightarrow$ *id* $=$ **E**

SENTENCIA $\rightarrow$ **DECLARACION**

SENTENCIA $\rightarrow$ **FUNCION**

SENTENCIA $\rightarrow$ **IF ELSE**

SENTENCIA $\rightarrow$ **CICLO**

SENTENCIA $\rightarrow$ **ASIGNACION**

IF $\rightarrow$ if(**A**){**INSTRUCCIONES**}

ELSE $\rightarrow$ else{**INSTRUCCIONES**}

ELSE $\rightarrow \epsilon$

CICLO $\rightarrow$ **WHILE**

CICLO $\rightarrow$ **FOR**

WHILE $\rightarrow$ while(**A**){**INSTRUCCIONES**}

FOR $\rightarrow$ for(**DECLARACION**, **A**, **E**){**INSTRUCCIONES**}

A $\rightarrow$ *id* **COMP** *id* **A'**

A' $\rightarrow$ **LOGICO** *id* **COMP** *id* **A'**

A' $\rightarrow \epsilon$

E $\rightarrow$ **T E'**

E' $\rightarrow$ **OPER T E'**

E' $\rightarrow \epsilon$

T $\rightarrow$ (**E**)

T $\rightarrow$ *id*

T $\rightarrow$ *num*

COMP $\rightarrow <$ | $>$ | $<=$ | $>=$ | $<>$ | $==$

LOGICO $\rightarrow$ @ | //

OPER $\rightarrow +$ | $-$ | $*$ | /

Lecturas aceptadas

`funk_ret( id ) { # id = id if( id <id ) { # id } ret id }`
`funk_not_ret( id ) { # id = id funk_not_ret( id ) { if ( id <id ) { # id = num } # id = id + id } }`
`for ( # id , id <id , id + num ) { for ( # id , id <>id , id + num ) { # id = num + id } if ( id >id ) { # id = ( id + num ) * id funk_not_ret ( id ) { while ( id <id ) { id = num + num } } } # id = num }`
`funk_not_ret ( id ) { # id = id funk_not_ret ( id ) { for ( # id , id <id , id + num ) { # id = num } # id = id + id } }`
`if ( id <id @ id <id ) { # id = num + id }`
`if ( id <id @ id <id ) { # id = num + id }`

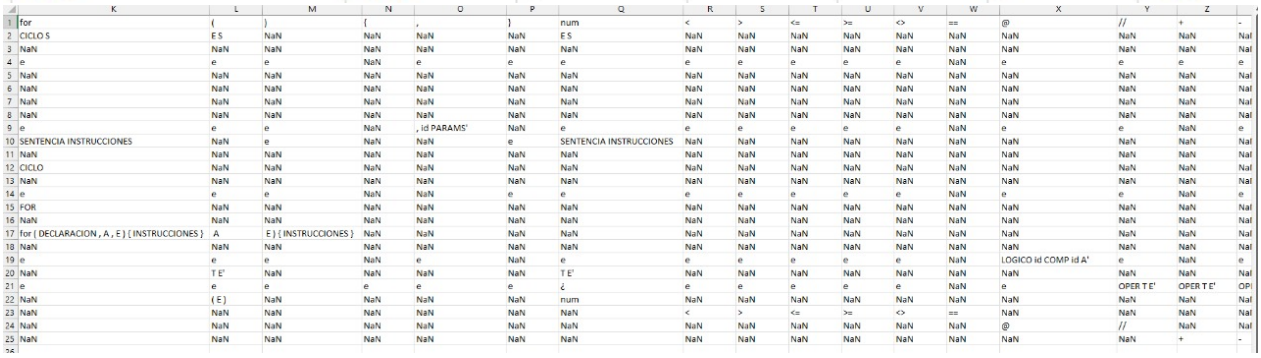
Lecturas no aceptadas

`if ( id <id id <id ) { # id = num + id }` En esta parte quitamos el símbolo `@` and para verificar que manda error.
`for ( ) { for ( # id , id <>id , id + num ) { # id = num + id } if ( id >id ) { # id = ( id + num ) * id funk_not_ret ( id ) { while ( id <id ) { id = num + num } } } # id = num }` El primer `for` no tiene parámetros, y la gramática requiere parámetros en la estructura del ciclo.
`funk_n*ot_ret ( id ) { # id = id funk_not_ret ( id ) { for ( # id , id <id , id + num ) { # id = num } # id = id + id } ret id }` Al final se agregó `ret id`, pero es un error ya que la función fue declarada como una función sin retorno (`funk_not_ret`).

Tabla sintáctica

	A	B	C	D	E	F	G	H	I	J	
1		#	id	=	funk_ret	funk_not_ret	ret	if	else	while	for
2	S	DECLARACION S	E S	NaN	FUNCION S	FUNCION S	NaN	IF ELSE S	NaN	CICLO S	CICLO
3	DECLARACION	# id DECL_EXTRA	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	DECL_EXTRA	e	e	= E	e	e	e	e	e	e	e
5	FUNCION	NaN	NaN	NaN	FUNK_RET	FUNK_NOT_RET	NaN	NaN	NaN	NaN	NaN
6	FUNK_RET	NaN	NaN	NaN	funk_ret (PARAMS) { INSTRUCCIONES ret E }	NaN	NaN	NaN	NaN	NaN	NaN
7	FUNK_NOT_RET	NaN	NaN	NaN	NaN	funk_not_ret (PARAMS) { INSTRUCCIONES }	NaN	NaN	NaN	NaN	NaN
8	PARAMS	NaN	id PARAMS'	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	PARAMS'	e	e	NaN	e	e	e	e	e	e	e
10	INSTRUCCIONES	SENTENCIA INSTRUCCIONES	SENTENCIA INSTRUCCIONES	NaN	SENTENCIA INSTRUCCIONES	SENTENCIA INSTRUCCIONES	e	SENTENCIA INSTRUCCIONES	NaN	SENTENCIA INSTRUCCIONES	SENTE
11	ASIGNACION	NaN	id = E	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	SENTENCIA	DECLARACION	ASIGNACION	NaN	FUNCION	FUNCION	NaN	IF ELSE	NaN	CICLO	CICLO
13	IF	NaN	NaN	NaN	NaN	NaN	NaN	if (A) { INSTRUCCIONES }	NaN	NaN	NaN
14	ELSE	e	e	e	e	e	e	e	else { INSTRUCCIONES }	e	e
15	CICLO	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	WHILE	FOR
16	WHILE	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	while (A) { INSTRUCCIONES }	NaN
17	FOR	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	for (D
18	A	NaN	id COMP id A'	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
19	A'	e	e	e	e	e	e	e	e	e	e
20	E	NaN	T E'	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
21	E'	e	e	e	e	e	e	e	e	e	e
22	T	NaN	id	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23	COMP	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
24	LOGICO	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25	OPER	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

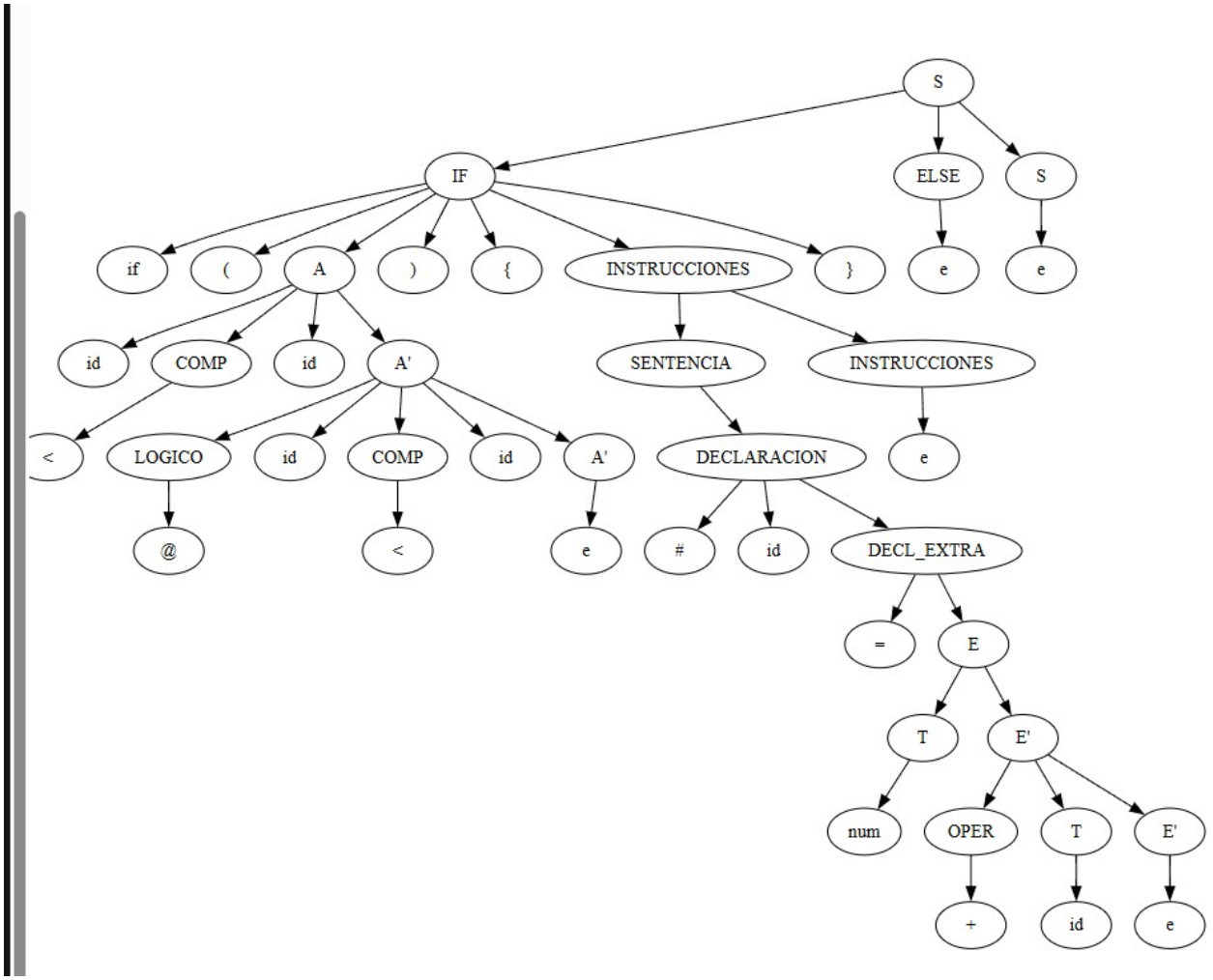
X	Y	Z	AA	AB	AC	AD
@	//	+	-	*	/	$
NaN	NaN	NaN	NaN	NaN	NaN	e
NaN	NaN	NaN	NaN	NaN	NaN	NaN
e	e	e	e	e	e	e
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
e	e	NaN	e	e	e	e
NaN	NaN	NaN	NaN	NaN	NaN	e
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
e	e	NaN	e	e	e	e
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
LOGICO id COMP id A'	e	NaN	e	e	e	e
NaN	NaN	NaN	NaN	NaN	NaN	NaN
e	OPER T E'	OPER T E'	OPER T E'	OPER T E'	e	e
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
@	//	NaN	NaN	NaN	NaN	NaN
NaN	NaN	+	-	*	/	NaN



	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
1	for	(	)	{	,	}	num	<	>	<=	>=	<>	==	@	//	+	-
2	CICLO S	E S	NaN	NaN	NaN	NaN	E S	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
4	e	e	e	NaN	e	e	e	e	e	e	e	e	NaN	e	e	e	e
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
9	e	e	e	NaN	, id PARAMS'	NaN	e	e	e	e	e	e	NaN	e	e	NaN	e
10	SENTENCIA INSTRUCCIONES	NaN	e	NaN	NaN	e	SENTENCIA INSTRUCCIONES	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
12	CICLO	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
14	e	e	e	NaN	NaN	e	e	e	e	e	e	e	NaN	e	e	NaN	e
15	FOR	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
17	for (DECLARACION , A , E) { INSTRUCCIONES }	A	E) { INSTRUCCIONES }	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
19	e	e	e	NaN	e	NaN	e	e	e	e	e	e	NaN	LOGICO id COMP id A'	e	NaN	e
20	NaN	T E'	NaN	NaN	NaN	NaN	T E'	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
21	e	e	e	e	e	e	¿	e	e	e	e	e	NaN	e	OPER T E'	OPER T E'	OPI
22	NaN	(E)	NaN	NaN	NaN	NaN	num	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
23	NaN	NaN	NaN	NaN	NaN	NaN	NaN	<	>	<=	>=	<>	==	NaN	NaN	NaN	Nal
24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	@	//	NaN	Nal
25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	+	-

Arbol

Ejemplo de lectura:
`if ( id <id @ id <id ) { # id = num + id }`


S
IF
ELSE
S
if
(
A
)
{
INSTRUCCIONES
}
e
e
id
COMP
id
A'
SENTENCIA
INSTRUCCIONES
<
LOGICO
id
COMP
id
A'
DECLARACION
e
@
<
e
#
id
DECL_EXTRA
=
E
T
E'
num
OPER
T
E'
+
id
e