Kenneth Haroldo López López
201906570

ARCHIVO DE GRAMÁTICAS

TYPESTY

Expresiones Regulares

Espacios en blanco: \s+

Comentario de línea: [/ /][.]*

Comentario multilínea: [/][*][^ *]*[*]+([^ / *][^ *]*[*]+)*[/]

Número decimal: ([0-9])+(["."])([0-9])+

Número entero: ([0-9])+

Identificador: ([a-z] | [A-Z] | _ | ñ | Ñ) ([a-z] | [0-9] | [A-Z] | _ | ñ | Ñ)*

Cadena: [“]([^ (“ | \ )] | [\ \ \ “] | [\ \ n] | [\ \ t] | [\ \ r] | [\ \ \ \ ] | [\ \ ‘])*[“]

Carácter: [\ ‘](\ \ n | \ \ r | \ \ t | \ \ ' | \ \ \ " | \ \ \ \ | [^ \ ']) [\ ']

Terminales

Incremento:	++	Menor:	<
Más:	+	Interrog:	?
Decremento:	--	Dospt:	:
Menos:	-	Ptcoma:	;
Por:	*	Ylogico:	&&
Dividido:	/	Ologico:	||
Elevado:	^	Parena:	(
Modulo:	%	Parenc:	)
Equals:	==	Llavea:	{
Igual:	=	Llavec:	}
Diferente:	!=	Coma:	,
Mayorigual:	>=	Elevado:	^
Menorigual:	<=	Corchetea:	[
Mayor:	>	Corchetec:	]

Tint:	int	Vfalse:	false
Tdouble:	double	Si:	if
Tbool:	boolean	Sino:	else
Tchar:	char	Para:	for
Tstring:	string	Mientras:	while
Print:	print	Fswitch:	switch
Vtrue:	true	Caso:	case
Defecto:	default		
Has:	do	Mayusculas:	toupper
Ex:	exec	Tamanio:	length
Tmethod:	void	Truncar:	truncate
Retorno:	return	Redondear:	round
Romper:	break	TypeOf:	typeof
Continuar:	continue	Acadena:	Tostring
Minusculas:	tolower		

No Terminales

INICIO: no terminal inicial de la gramática puede derivar en GLOBALES o en un archivo vacío.

GLOBALES: deriva en GLOBAL una o muchas veces debido a su recursividad por la derecha.

GLOBAL: puede derivar en una sola instrucción valida del ámbito global, como declaraciones, asignaciones, funciones y métodos. Así como la llamada del método inicial por medio de EXEC.

DECLARACION: puede derivar en la estructura normal de una declaración de variable con un tipo, identificador y valor de asignación; así como una declaración por defecto (sin asignación de valor).

ASIGNACION: puede derivar en asignaciones explícitas de valores a identificadores previamente o también en incrementos y decrementos de un identificador.

FUNCION: declaración de un método (void) o una función con tipo. Posee un identificador, lista de parámetros y un bloque de instrucciones (ámbito local).

EXEC: deriva en el terminal utilizado para definir la llamada inicial de un método en el programa.

SYNC: deriva en el terminal de sincronización para el programa.

INSTRUCCIONES: deriva en INSTRUCCION una o muchas veces debido a su recursividad por la derecha. Se concentra en bloques de instrucciones donde no es necesaria la utilización de encapsulamiento de sentencias.

INSTRUCCION: deriva en una instrucción válida de un ámbito local, como declaraciones varias, asignaciones, llamadas a métodos y funciones, sentencias de control, cíclicas y función print.

TRANSFERENCIA: deriva en las sentencias de transferencia utilizadas en el lenguaje; como break, continue y return.

IF: deriva en las diferentes formas en las que se puede realizar una sentencia de control if: con una única condición, condiciones múltiples con else if y bloques else.

SWITCH: deriva en las diferentes formas de realizar una sentencia de control switch: con varios cases, con cases y un default o únicamente un default.

WHILE: deriva en la estructura de la sentencia cíclica while: con la palabra reservada respectiva, una condición encerrada en paréntesis, y su bloque se instrucciones.

DOWHILE: deriva en la estructura de la sentencia cíclica do while: con la palabra reservada respectiva, su bloque se instrucciones, la palabra reservada while, una condición encerrada en paréntesis y terminando con el signo de sincronización.

FOR: deriva en la estructura de la sentencia cíclica for: con su respectiva palabra reservada y, entre paréntesis y separadas por punto y coma, una declaración o asignación, una condición y una actualización de la variable. Junto con su respectivo bloque de instrucciones.

PRINT: deriva en la estructura para la declaración de una impresión en consola: con su respetiva palabra reservada, y encerrada entre paréntesis la expresión a imprimir.

LLAMADA: deriva en la estructura necesaria para invocar una función o método. Con un identificador, y entre paréntesis una colección de valores que corresponden a los parámetros de la función.

BLOQUE: deriva en el inicio de un bloque con un símbolo de encapsulamiento inicial y una derivación de BLOQUE2. Utilizado para delimitar un bloque de instrucciones para cualquier sentencia que lo necesite.

BLOQUE2: deriva en una o muchas instrucciones válidas de un ámbito local debido a su recursividad por la derecha y termina su recursividad con el símbolo final de encapsulamiento.

NATIVA: deriva en cualquier llamada a una función nativa del lenguaje (toUpper, toLower, length, toCharArray, truncate, round, typeOf o toString) con su respectiva palabra reservada y encerrado en paréntesis, la expresión a operar.

CASTEO: deriva en la estructura de un casteo de valor: un tipo encerrado en paréntesis y a su derecha la expresión a operar.

TYPE: deriva en cualquiera de los tipos de datos primitivos del lenguaje.

OPTERNARIO: deriva en la estructura de utilización de un operador ternario: una expresión seguida del carácter '?' seguido de una expresión, luego dos puntos y otra expresión.

ELSE: deriva en la expresión else de un if junto con su respectivo bloque de instrucciones o un else acompañado de una derivación IFSOLO.

IFSOLO: deriva en la estructura simple de un if de una sola condición: la palabra reservada, entre paréntesis su condición y su respectivo bloque de instrucciones.

CASES: deriva en uno o muchos cases de un switch debido a su recursividad por la izquierda. Termina dicha recursividad con una producción simple de la estructura de case: la palabra reservada, una expresión seguida de dos puntos y el bloque de instrucciones sin símbolos de encapsulamiento.

DEFAULT: deriva en la estructura de un caso default de un switch; similar a la estructura de un case, excluyendo únicamente la expresión que viene después del case y reemplazando la palabra reservada case por default.

EXPRL: expresiones con la menor precedencia, incluyendo casteos, operador ternario y expresiones lógicas. También puede derivar en EXP2.

EXP2: producción de expresiones con precedencia media y alta, como las expresiones aritméticas. También puede derivar en EXPVAL.

EXPVAL: expresiones con la precedencia más alta, como valores puntuales, identificadores de variables, llamadas a funciones nativas y negación unitaria.

NUM: deriva en los terminales de entero y decimal para valores puntuales.

PARAM: deriva una o muchas veces en un parámetro de la declaración de una función o método separados por coma con su recursividad por la izquierda. Rompe la recursividad con la estructura de un parámetro unitario: tipo de dato e identificador.

FUNCION: deriva en la estructura de declaración de una función o método: su tipo de retorno, su identificador, entre paréntesis su lista de parámetros y su respectivo bloque de instrucciones.

LISTAVALORES: deriva en una lista de expresiones separadas por coma debido a su recursividad por la izquierda. Rompe la recursividad con una derivación a EXPRL.

Inicio de la gramática: INICIO

Descripción de las producciones

$$INICIO \rightarrow GLOBALES\ EOF$$

Desde el inicio del archivo pueden venir una cantidad indefinida de instrucciones en el ámbito global.

$$INICIO \rightarrow EOF$$

El archivo puede estar vacío.

$$GLOBALES \rightarrow GLOBALES\ GLOBAL$$

Recursividad para reconocer varias producciones de GLOBALES.

$$GLOBALES \rightarrow GLOBAL$$

Fin de la recursividad de GLOBALES

$$GLOBAL \rightarrow DECLARACION\ SYNC$$

Una instrucción global puede ser una declaración de variable.

$$GLOBAL \rightarrow ASIGNACION\ SYNC$$

Una instrucción global puede ser una asignación de valor.

$$GLOBAL \rightarrow FUNCION$$

Una instrucción global puede ser una declaración de función o método.

$$GLOBAL \rightarrow EXEC\ SYNC$$

Una instrucción global puede ser una llamada para el primer método.

$$INSTRUCCIONES \rightarrow INSTRUCCION\ INSTRUCCIONES$$

Recursividad de instrucciones sin símbolos de encapsulamiento.

$$INSTRUCCIONES \rightarrow INSTRUCCION$$

Fin de la recursividad de INSTRUCCIONES.

$$INSTRUCCION \rightarrow DECLARACION\ SYNC$$

Una instrucción local puede ser una declaración.

$$INSTRUCCION \rightarrow ASIGNACION\ SYNC$$

Una instrucción local puede ser una asignación.

$$INSTRUCCION \rightarrow TRANSFERENCIA\ SYNC$$

Una instrucción local puede ser una instrucción de transferencia.

$$INSTRUCCION \rightarrow IF$$

Una instrucción local puede ser un if.

$$INSTRUCCION \rightarrow SWITCH$$

Una instrucción local puede ser un switch.

$$INSTRUCCION \rightarrow WHILE$$

Una instrucción local puede ser un ciclo while.

$$INSTRUCCION \rightarrow DOWHILE$$

Una instrucción local puede ser un ciclo dowhile

$$INSTRUCCION \rightarrow FOR$$

Una instrucción local puede ser un ciclo for

$$INSTRUCCION \rightarrow PRINT\ SYNC$$

Una instrucción local puede ser un print.

$$INSTRUCCION \rightarrow LLAMADA\ SYNC$$

Una instrucción local puede ser una llamada a método.

$$BLOQUE \rightarrow \{\ BLOQUE2$$

Inicio de un bloque con símbolo de encapsulamiento.

$$BLOQUE2 \rightarrow INSTRUCCION\ BLOQUE2$$

Recursividad de instrucciones en el bloque.

$$BLOQUE2 \rightarrow \}$$

Fin de la recursividad o también para bloques vacíos.

$$PRINT \rightarrow print\ (\ EXPRL\ )$$

Función print con valor.

$$PRINT \rightarrow print\ (\ )$$

Print vacío.

$$NATIVA \rightarrow tolower\ (\ EXPRL)$$

Función nativa tolower.

$$NATIVA \rightarrow toupper\ (EXPRL)$$

Función nativa toupper.

$$NATIVA \rightarrow length(\ EXPRL\ )$$

Función nativa length.

$$NATIVA \rightarrow truncate\ (\ EXPRL\ )$$

Función nativa truncate.

$$NATIVA \rightarrow round\ (\ EXPRL\ )$$

Función nativa round.

$$NATIVA \rightarrow typeof\ (\ EXPRL\ )$$

Función nativa typeof.

$$NATIVA \rightarrow tostring\ (\ EXPRL\ )$$

Función nativa tostring.

$$DECLARACION \rightarrow TYPE\ identificador$$

Declaración por defecto.

$$DECLARACION \rightarrow TYPE\ identificador = EXPRL$$

Declaración con asignación de valor.

$$CASTEO \rightarrow (\ TYPE\ )\ EXPRL$$

Casteo de valores.

$$ASIGNACION \rightarrow identificador = EXPRL$$

Asignación de valor.

$$ASIGNACION \rightarrow identificador + +$$

Asignación de incremento.

$$ASIGNACION \rightarrow identificador—$$

Asignación decremento.

$$TYPE \rightarrow int$$

Un tipo de dato puede ser int.

$$TYPE \rightarrow string$$

Un tipo de dato puede ser string.

$$TYPE \rightarrow double$$

Un tipo de dato puede ser double.

$$TYPE \rightarrow boolean$$

Un tipo de dato puede ser booleano.

$$TYPE \rightarrow char$$

Un tipo de dato puede ser un carácter.

$$SYNC \rightarrow ;$$

Símbolo de sincronización es punto y coma.

$$EXEC \rightarrow exec\ LLAMADA$$

Llama al método inicial del programa.

$$LLAMADA \rightarrow identificador\ (\ LISTAVALORES\ )$$

Llamada a un método con parámetros.

$$LLAMADA \rightarrow identificador\ (\ )$$

Llamada a un método sin parámetros.

$$LISTAVALORES \rightarrow LISTAVALORES\ ,EXPRL$$

Recursividad de ingreso de valores en una llamada.

$$LISTAVALORES \rightarrow EXPRL$$

Fin de la recursividad de LISTAVALORES.

$$FUNCION \rightarrow void\ identificador\ (\ PARAM\ )BLOQUE$$

Declaración de método con parámetros.

$$FUNCION \rightarrow void\ identificador\ (\ )\ BLOQUE$$

Declaración de método sin parámetros.

$$PARAM \rightarrow PARAM\ ,TYPE\ identificador$$

Recursividad de múltiples parámetros.

$$PARAM \rightarrow TYPE\ identificador$$

Fin de la recursividad PARAM o parámetro unitario.

$$WHILE \rightarrow while\ (\ EXPRL\ )\ BLOQUE$$

Declaración de ciclo while.

$$DOWHILE \rightarrow do\ BLOQUE\ while\ (\ EXPRL\ )\ parenc\ SYNC$$

Declaración de ciclo dowhile.

$$FOR \rightarrow for\ (\ ASIGNACION\ ;\ EXPRL\ ;\ ASIGNACION)\ BLOQUE$$

Declaración de ciclo for con variable de iteración previamente definida.

$$FOR \rightarrow for\ (\ DECLARACION\ ;EXPRL\ ;ASIGNACION\ )\ BLOQUE$$

Declaración de ciclo for con variable de iteración local.

$$TRANSFERENCIA \rightarrow continue$$

Expresión continue para ciclos.

$$TRANSFERENCIA \rightarrow break$$

Expresión break para ciclos y switch.

$$OPTERNARIO \rightarrow EXPRL\ ?\ EXPRL\ :\ EXPRL$$

Declaración de expresión con operador ternario.

$$IF \rightarrow IFSOLO$$

Un if puede ser de una sola condición.

$$IF \rightarrow IFSOLO\ ELSE$$

Un if puede contener múltiples bloques.

$$ELSE \rightarrow else\ BLOQUE$$

Sentencia else de un if.

$$ELSE \rightarrow else\ IF$$

Un if puede contener múltiples condiciones.

$$IFSOLO \rightarrow if\ (\ EXPRL\ )\ BLOQUE$$

Estructura de un if de solo una condición.

$$SWITCH \rightarrow switch\ (\ EXPRL\ )\ \{\ CASES\ \}$$

Declaración de la sentencia switch con uno o múltiples cases.

$$SWITCH \rightarrow switch\ (\ EXPRL\ )\ \{\ DEFAULT\ \}$$

Declaración de la sentencia switch con expresión default únicamente.

$$SWITCH \rightarrow switch\ (\ EXPRL\ )\ \{\ CASES\ DEFAULT\ \}$$

Declaración de la sentencia switch con uno o múltiples cases e instrucción default.

$$CASES \rightarrow CASES\ caso\ EXPRL : INSTRUCCIONES$$

Recursividad para múltiples casos en el mismo switch.

$$CASES \rightarrow caso\ EXPRL : INSTRUCCIONES$$

Fin de la recursividad de CASES o caso único.

$$DEFAULT \rightarrow default : INSTRUCCIONES$$

Instrucción default de un switch.

$$EXPRL \rightarrow CASTEO$$

Un casteo puede ser utilizado como expresión.

$$EXPRL \rightarrow OPTERNARIO$$

Una operación ternaria también puede ser utilizada como expresión.

$$EXPRL \rightarrow EXPRL\ ||\ EXPRL$$

Una expresión puede ser un or de otras expresiones.

$$EXPRL \rightarrow EXPRL\ \&\&\ EXPRL$$

Una expresión puede ser un and de otras expresiones.

$$EXPRL \rightarrow !\ EXPRL$$

Una expresión puede ser la negación lógica de otra expresión.

$$EXPRL \rightarrow EXPRL == EXPRL$$

Una expresión puede ser la igualdad de otras expresiones.

$$EXPRL \rightarrow EXPRL != EXPRL$$

Una expresión puede ser una diferencia de otras expresiones.

$$EXPRL \rightarrow EXPRL < EXPRL$$

Una expresión puede ser una comparación de menor que de otras expresiones.

$$EXPRL \rightarrow EXPRL > EXPRL$$

Una expresión puede ser una comparación de mayor que de otras expresiones.

$$EXPRL \rightarrow EXPRL \leq EXPRL$$

Una expresión puede ser una comparación de menor o igual que de otras expresiones.

$$EXPRL \rightarrow EXPRL \geq EXPRL$$

Una expresión puede ser una comparación de mayor o igual que de otras expresiones.

$$EXPRL \rightarrow EXP2$$

Una expresión puede derivar en una derivación de EXP2.

$$EXP2 \rightarrow EXPRL + EXPRL$$

Una expresión puede ser una suma de expresiones.

$$EXP2 \rightarrow EXPRL - EXPRL$$

Una expresión puede ser una resta de expresiones.

$$EXP2 \rightarrow EXPRL * EXPRL$$

Una expresión puede ser una multiplicación de expresiones.

$$EXP2 \rightarrow EXPRL \,/\, EXPRL$$

Una expresión puede ser una división de expresiones.

$$EXP2 \rightarrow EXPRL \,\%\, EXPRL$$

Una expresión puede ser el módulo de la división de expresiones.

$$EXP2 \rightarrow EXPRL\text{^}EXPRL$$

Una expresión puede ser el resultado de elevar una expresión a un índice de otra expresión.

$$EXP2 \rightarrow EXPVAL$$

Una expresión puede derivar en alguna derivación de EXPVAL.

$$EXPVAL \rightarrow - EXPRL$$

Una expresión puede ser la negación unitaria de otra expresión.

$$EXPVAL \rightarrow (\ EXPRL\ )$$

Una expresión puede aumentar su precedencia al ser encerrada en paréntesis.

$$EXPVAL \rightarrow NUM$$

Una expresión puede ser un dato numérico.

$$EXPVAL \rightarrow cadena$$

Una expresión puede ser una cadena de caracteres.

$$EXPVAL \rightarrow true$$

Una expresión puede ser un valor verdadero

$$EXPVAL \rightarrow false$$

Una expresión puede ser un valor falso.

$$EXPVAL \rightarrow identificador$$

Una expresión puede ser un identificador de una variable.

$$EXPVAL \rightarrow NATIVA$$

Una expresión puede ser una llamada a una función nativa.

$$NUM \rightarrow entero$$

Un dato numérico puede ser un número entero.

$$NUM \rightarrow decimal$$

Un dato numérico puede ser un número decimal.