Joseph Jeferson Marroquin Monroy 202010316 OLC1 “A”

Gramática

- ## Expresión Regulares:

Espacios en blanco: \s+

Comentario de simple línea: "//".*

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

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

Número entero: [0-9]+

Identificador: ([a-zA-Z_])[a-zA-Z0-9_]*

Cadena: [\"]("\\n"|"\\r"|"\\t"|"\\'"|"\\\""|"\\\\"|[^\"])*[\"]

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

- ## Terminales:

Signo más “+”

Signo menos “-“

Signo multiplicación “*”

Signo división “/”

Signo potencia “^“

Signo modulo “%”

Igual que “==”

Igual para asignar “=”

Diferente que “!=”

Mayor o igual que “>=”

Menor o igual que “<=”

Mayor que “>”

Menor que “<”

Signo interrogación “?”

Signo de dos puntos “:”

Signo de punto y coma “;”

Signo AND “&&”

Signo OR “||”

Paréntesis izquierdo “(“

Paréntesis derecho “)”

Llave izquierda “{“

Llave derecha “}”

Signo coma “,”

Corchete izquierdo “[“

Corchete derecho “]”

Palabra reservada para enteros “int”

Palabra reservada para decimales “double”

Palabra reservada para booleanos “boolean”

Palabra reservada para cadenas “string”

Palabra reservada para caracteres “char”

Palabra reservada para imprimir en línea “print”

Palabra reservada para imprimir multilínea “println”

Palabra reservada para verdadero “true”

Palabra reservada para falso “false”

Palabra reservada para iniciar un método o función “run”

Palabra reservada para sentencias cíclicas “while, do, for”

Palabra reservada para token si “if”

Palabra reservada para token si no “else”

Palabra reservada para switch “switch”

Palabra reservada para casos de switch “case”

Palabra reservada para valor por defecto de un Switch “default”

Palabra reservada para tipo de método “void”

Palabra reservada para retornar valor o detener “return”

Palabra reservada para detener por completo “break”

Palabra reservada para saltarse una iteración “continue”

Palabra reservada para convertir cadena en minúsculas “toLowe”

Palabra reservada para convertir cadena en mayúsculas “toUpper”

Palabra reservada para obtener tamaño de una cadena o vector “length”

Palabra reservada para redondear números decimales “round”

Palabra reservada para saber el tipo de variable “typeof”

Palabra reservada para castear a una cadena “toString”

- **No Terminales:**
- *SENTENCIAS*: no terminal inicial de la gramática puede derivar en SENTENCIA una o muchas veces por recursividad por la izquierda.
- *SENTENCIA*: derivan las instrucciones de todo el lenguaje como declaraciones, asignaciones, métodos, estructuras de datos, sentencias cíclicas, etc.
- *DECLARACION*: instrucción para declarar una o más variables separados por coma asignándoles un valor por defecto.
- *DECLARACIONYASIGNACION*: declara una variable o más separadas por coma, con tipo de dato, identificador y un valor a asignarle a esa variable.
- *DECLARACION_VECTORES*: declaración de una estructura de datos de tipo 1.
- *DECLARACION_VECTORES_TIPO2*: declaración de una estructura de datos de tipo 2.
- *INCREMENTO_DECREMENTO*: se recibe una variable previamente declarada y se realiza una modificación a su valor incrementando o decremento de 1.
- *MODIFICA_VECTOR*: recibe un identificador con su posición dentro del vector para cambiar su valor.
- *VECTOR_CHAR*: estructura de datos para la función tochararray que recibe una cadena y la devuelve en un arreglo.
- *BLOQUE*: inicia un bloque con un símbolo de encapsulamiento donde se pueden recibir SENTENCIA.
- *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 lineal en consola: con su respetiva palabra reservada, y encerrada entre paréntesis la expresión a imprimir.

- *PRINTLN*: deriva en la estructura para la declaración de una impresión con salto de linea en consola: con su respetiva palabra reservada, y encerrada entre paréntesis la expresión a imprimir.
- *LLAMADAS*: 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.
- *METODOS*: 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.
- *FUNCIONES*: 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.
- *BREAK*: instrucción para detener un ciclo: con su respetiva palabra reservada.
- *RETURN*: instrucción para retornar un valor o detener un ciclo: con su respetiva palabra reservada.
- *CONTINUE*: instrucción para saltarse una iteracion: con su respetiva palabra reservada.
- *ID_LIST*: deriva en una lista de identificadores separadas por coma debido a su recursividad por la izquierda.
- *VECTOR_LIST*: deriva en una lista de expresiones para ingresar en las estructuras de datos, separadas por coma debido a su recursividad por la izquierda.
- *Elif_list:* deriva en una lista de la instrucción elif debido a su recursividad por la izquierda.
- *elif*: deriva en la estructura simple de un elif de una sola condición: la palabra reservada, entre paréntesis su condición y su respectivo bloque de instrucciones.
- *if*: 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.
- *CASE_LIST*: deriva en una lista de case que pueden venir dentro de un SWITCH con su palabra reservado seguido de dos puntos y su bloque de instrucciones, usando recursividad por la izquierda para la lista de case
- *CASE*: deriva en la estructura simple de un CASE : la palabra reservada, dos puntos y su respectivo bloque de instrucciones
- *LISTA_EXP*: deriva en una lista de expresiones para ingresar en las estructuras de datos, separadas por coma debido a su recursividad por la izquierda.
- *LISTA_PARAMETROS*: deriva en una lista de parámetros para ingresar a métodos o funciones separadas por coma debido a su recursividad por la izquierda.
- *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.
- *CASTEO*: deriva en la estructura de un casteo de valor: un tipo encerrado en paréntesis y a su derecha la expresión a operar.
- *EXP*: deriva todas las expresiones que pueden venir dentro del lenguaje como estructuras de datos, variables, enteros, decimales, cadenas, caracteres operaciones relaciones, lógicas, etc

.

- **Inicio de la gramática:**
  S: SENTENCIAS

- **Descripción de las producciones:**

$$S \rightarrow \text{SENTENCIAS } EOF$$

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

$$\text{SENTENCIAS} \rightarrow \text{SENTENCIA}$$

Recursividad para reconocer todas las instrucciones a ejecutar.

$$\text{SENTENCIA} \rightarrow \text{DECLARACION}$$

Instrucción para declarar una o mas variables con su valor por defecto

$$\text{SENTENCIA} \rightarrow \text{ASIGNACION}$$

Instrucción para asignar una o más variables un valor.

$$\text{SENTENCIA} \rightarrow \text{DECLARACIONYASIGNACION}$$

Instrucción para declarar y asignar a una o más variables un valor.

$$\text{SENTENCIA} \rightarrow \text{DECLARACION\_VECTORES}$$

Instrucción para declarar un vector de tipo 1.

$$\text{SENTENCIA} \rightarrow \text{DECLARACION\_VECTORES\_TIPO2}$$

Instrucción para declarar un vector de tipo 2.

$$\text{SENTENCIA} \rightarrow \text{INCREMENTO\_DECREMENTO}$$

Instrucción para asignar a una variable un incremento o decremento de 1.

$$\text{SENTENCIA} \rightarrow \text{MODIFICA\_VECTOR}$$

Instrucción modificar un valor en específico de un arreglo.

$$\text{SENTENCIA} \rightarrow \text{VECTOR\_CHAR}$$

Instrucción asignarle a un arreglo la función tochararry.

$$\text{SENTENCIA} \rightarrow \text{BLOQUE}$$

Recibe un bloque de instrucciones encerrado por símbolos de encapsulamiento.

$$\text{SENTENCIA} \rightarrow IF$$

Una instrucción, puede ser un if solo, con una lista de elif o con un else.

$$\text{SENTENCIA} \rightarrow SWITCH$$

instrucción puede ser un switch con una lista de case o con default.

$$\text{SENTENCIA} \rightarrow WHILE$$

instrucción puede ser un ciclo while.

$$\text{SENTENCIA} \rightarrow DOWHILE$$

instrucción puede ser un ciclo dowhile

$$\text{SENTENCIA} \rightarrow PRINT$$

Instrucción print para impresión lineal.

$$\text{SENTENCIA} \rightarrow PRINT\text{LN}$$

Instrucción print para impresión con salto de linea.

$$\text{SENTENCIA} \rightarrow \text{FOR}$$

instrucción puede ser un ciclo for

$$\text{SENTENCIA} \rightarrow FUNCION\text{ES}$$

Instrucción para declarar una función con o sin parámetros, si viene la palabra reservada run se ejecuta la función.

$$\text{SENTENCIA} \rightarrow \text{METODOS}$$

Instrucción para declarar un método con o sin parámetros, si viene la palabra reservada run se ejecuta el método.

$$\text{SENTENCIA} \rightarrow LLAMADA\text{S}$$

Una instrucción puede ser una llamada a método o función con o sin parámetros, si viene la palabra reservada run se ejecuta el método o función.

$$\text{SENTENCIA} \rightarrow \text{BREAK}$$

Una instrucción que ejecuta la sentencia break que termina un ciclo.

$$\text{SENTENCIA} \rightarrow \text{RETURN}$$

Instrucción para retornar un valor o detener un ciclo o método.

$$\text{SENTENCIA} \rightarrow \text{CONTINUE}$$

Instrucción para saltarse una iteración de un ciclo.

$$PRINT \rightarrow print \text{ (EXP)}$$

Función print con una expresion.

$$\text{EXP} \rightarrow tolower \text{ (EXP)}$$

Función nativa tolower.

$$\text{EXP} \rightarrow toupper \text{ (EXP)}$$

Función nativa toupper.

$$\text{EXP} \rightarrow length\text{(EXP)}$$

Función nativa length.

$$\text{EXP} \rightarrow round \text{ (EXP)}$$

Función nativa round.

$$\text{EXP} \rightarrow typeof \text{ (EXP)}$$

Función nativa typeof.

$$\text{EXP} \rightarrow tostring \text{ (EXP)}$$

Función nativa tostring.

$$CASTEOS \rightarrow \text{ (TIPO\_IDENTIFICADOR) EXP}$$

Casteo de valores.

$$\text{INCREMENTO\_DECREMENTO} \rightarrow identificador++$$

Asignación de incremento.

$$\text{INCREMENTO\_DECREMENTO} \rightarrow identificador\text{- -}$$

Asignación decremento.

$$\text{TIPO\_IDENTIFICADO} \rightarrow int$$

Un tipo de dato puede ser int.

$$\text{TIPO\_IDENTIFICADO} \rightarrow \text{string}$$

Un tipo de dato puede ser string.

$$\text{TIPO\_IDENTIFICADO} \rightarrow \text{double}$$

Un tipo de dato puede ser double.

$$\text{TIPO\_IDENTIFICADO} \rightarrow \text{boolean}$$

Un tipo de dato puede ser booleano.

$$\text{TIPO\_IDENTIFICADO} \rightarrow \text{char}$$

dato puede ser un carácter.

$$\text{LLAMADAS} \rightarrow \text{run } LLAMADA$$

Llama al método inicial del programa.

$$\text{LISTA\_EXP} \rightarrow \text{LIST\_EXP COMA EXP}$$

$$|\text{EXP}$$

Recursividad de ingreso de expresion.

$$\text{ID\_LIST} \rightarrow \text{ID\_LIST COMA ID}$$

$$|\text{ID}$$

Recursividad de ingreso de identificadores.

$$OPTERNARIO \rightarrow \text{EXP ? EXP : EXP}$$

Declaración de expresión con operador ternario.

$$\text{EXP} \rightarrow CASTEO \text{ Un casteo}$$

puede ser utilizado como expresión.

$$\text{EXP} \rightarrow OPTERNARIO$$

Una operación ternaria también puede ser utilizada como expresión.

$$\text{EXP} \rightarrow \text{EXP} \mid \mid \text{EXP}$$

Una expresión puede ser un or de otras expresiones.

$$\text{EXP} \rightarrow \text{EXP \&\& EXP}$$

Una expresión puede ser un and de otras expresiones.

$$\text{EXP} \rightarrow \text{! EXP}$$

Una expresión puede ser la negación lógica de otra expresión.

$$\text{EXP} \rightarrow \text{EXP} == \text{EXP}$$

Una expresión puede ser la igualdad de otras expresiones.

$$\text{EXP} \rightarrow \text{EXP}! = \text{EXP}$$

Una expresión puede ser una diferencia de otras expresiones.

$$\text{EXP} \rightarrow \text{EXP} < \text{EXP}$$

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

$$\text{EXP} \rightarrow \text{EXP} > \text{EXP}$$

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

$$\text{EXP} \rightarrow \text{EXP} \leq \text{EXP}$$

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

$$\text{EXP} \rightarrow \text{EXP} \geq \text{EXP}$$

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

$$\text{EXP} \rightarrow \text{EXP} + \text{EXP}$$

Una expresión puede ser una suma de expresiones.

$$\text{EXP} \rightarrow \text{EXP} - \text{EXP}$$

Una expresión puede ser una resta de expresiones.

$$\text{EXP} \rightarrow \text{EXP} * \text{EXP}$$

Una expresión puede ser una multiplicación de expresiones.

$$\text{EXP} \rightarrow \text{EXP} / \text{EXP}$$

Una expresión puede ser una división de expresiones.

$$\text{EXP} \rightarrow \text{EXP} \,\%\, \text{EXP}$$

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

$$\text{EXP} \rightarrow \text{EXP} \,\hat{}\, \text{EXP}$$

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

$$\text{EXP} \rightarrow (\text{EXP})$$

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

$$\text{EXP} \rightarrow \text{ENTERO}$$

Una expresión puede ser un dato numérico.

$$\text{EXP} \rightarrow \text{Tok\_numero}$$

Una expresión puede ser un dato numérico con decimales.

$$\text{EXP} \rightarrow \text{Tok\_string}$$

Una expresión puede ser una cadena de caracteres.

$$\text{EXP} \rightarrow true$$

Una expresión puede ser un valor verdadero

$$\text{EXP} \rightarrow false$$

Una expresión puede ser un valor falso.