DOCUMENTACIÓN JSHARP

DESCRIPCIÓN DE GRAMÁTICA

Luis Leonel Aguilar
201603029

Expresiones Regulares

Expresión regular	Utilizada para	Descripción
[[a-z] | [A-Z] | [ñ] | [Ñ] | [_]] [[a-z] | [A-Z] | [ñ] | [Ñ] | [_][0-9]]*	Identificadores	Un identificador es un token que contiene al menos una letra o guion, seguido de una lista opcional de letras, guiones y números.
[0-9]+"."[0-9]+	Números decimales	Un número decimal contiene al menos un dígito (0-9), seguido de un punto y otra lista de al menos un dígito.
[0-9]+	Números enteros	Un número entero contiene al menos un dígito (0-9).
""	String vacío	Un string vacío es compuesto por dos comillas dobles seguidas.
" ([. "])* "	String	Un string está compuesto por una comilla doble, seguida de una lista de cualquier carácter (excluyendo la comilla doble), seguida de una comilla doble.
' [. ']? '	Char	Un char está compuesto por una comilla simple, seguida de un único carácter cualquiera, seguido de otra comilla simple.
// [.]*	Comentarios de una línea	Un comentario de una línea está compuesto por dos barras "/", seguido de cualquier lista de caracteres excepto un salto de línea "\n".
"/*" [. "*/"]* "*/"	Comentarios multilíneas.	Un comentario multilínea está compuesto por una barra y asterisco "/*" seguido de cualquier cantidad de caracteres, seguido de "*/".

En JISON, se utilizó las siguientes producciones para realizar las expresiones regulares:

```
[a-zA-ZñÑ_][a-zA-ZñÑ0-9_]*    return 'IDENTIFIER';

[0-9]+"."[0-9]+\b      return 'FLOATING_POINT_LITERAL';
[0-9]+\b               return 'DECIMAL_INTEGER_LITERAL';

"\"\""                 return 'STRING_LITERAL';
"\""([^"]|{BSL})*"\""  return 'STRING_LITERAL';
"'"([^']|{BSL})?"'"    return 'CHAR_LITERAL';
```

Precedencia

La precedencia es el orden en el cual una gramática procesa cierto terminal, cuando esta se encuentra en una situación en la cual no se sabe qué reducir o qué desplazar. La Asociatividad es la manera en la cual se agrupan varios operadores a la vez.

Valor de precedencia	Operador	Asociatividad
1	=	Derecha
2	||	Izquierda
3	&&	Izquierda
4	^	Izquierda
5	==, !=	Izquierda
6	<, >, <=, >=	Izquierda
7	+, -	Izquierda
8	*, /, %	Izquierda
9	!	Derecha
10	++, --	Derecha

En JISON, se utilizó la siguiente sintaxis para definir la precedencia y asociatividad.

```
%right  OPERATOR_ASSIGNMENT
%left   OPERATOR_LOGICAL_OR
%left   OPERATOR_LOGICAL_AND
%left   OPERATOR_XOR
%left   OPERATOR_EQUAL OPERATOR_NOT_EQUAL
%left   OPERATOR_LESS_THAN OPERATOR_GREATER_THAN OPERATOR_LESS_THAN_EQUAL OPERATOR_GREATER_THAN_EQUAL
%left   OPERATOR_ADDITION OPERATOR_SUBTRACTION
%left   OPERATOR_MULTIPLICATION OPERATOR_DIVISION OPERATOR_MODULO
%right  OPERATOR_NEGATION
%right  POST_INCREMENT POST_DECREMENT
```

Terminales utilizados

#	Terminal	Descripción
1	{	Utilizado para iniciar y terminar la delimitación de ambientes y arreglos.
2	}	
3	(	Utilizado para hacer jerarquía de operaciones, para enviar parámetros.
4	)	
5	[	Utilizada para acceder a arreglos y para definir estructuras.

6	]	
7	.	Utilizado para hacer accesos a estructuras.
8	,	Utilizado para hacer listas, de parametros, de datos, de expresiones.
9	:	Utilizado en el case, y para declaraciones de variables y constantes.
10	;	Utilizado para delimitar expresiones.
11	import	Iniciar la importación de archivos.
12	var	Declaracion de variables.
13	const	Declaración de constantes.
14	global	Declaración de variables globales.
15	void	Declaración de funciones sin retorno.
16	if	Declaración de sentencia de control if y else if.
17	else	Declaración de sentencias else y else if.
18	for	Declaración de sentencias for
19	while	Declaración de sentencias while
20	do	Declaración de sentencias do_while
21	break	Declaración de sentencias break
22	continue	Declaración de sentencias continue
23	switch	Declaración de sentencias switch
24	case	Declaración de sentencias case
25	default	Declaración de sentencias default en los switch
26	define	Declaración de estructuras
27	as	Declaración de estructuras
28	print	Realizar impresiones a la consola
29	try	Declaración de sentencias try
30	catch	Declaración de sentencias catch
31	throw	Declaración de sentencias throw
32	strc	Operador para declarar arrays, throws, y estructuras
33	return	Declarador de sentencia de retorno
34	boolean	Definir un tipo de dato boolean
35	Integer	Definir un tipo de dato integer
36	double	Definir un tipo de dato double
37	char	Definir un tipo de dato char
38	<=	Operador menor o igual
39	<	Operador menor
40	===	Operador de comparación de referencias
41	==	Operador de comparación de igualdar
42	>=	Operador de mayor o igual
43	>	Operador de mayor
44	!=	Operador de diferencia
45	||	Operador lógico OR

46	^^	Operador aritmético POTENCIA
47	^	Operador lógico XOR
48	&&	Operador lógico AND
49	!	Operador lógico NOT
50	=	Operador de asignación
51	++	Operador aritmético de incremento
52	+	Operador aritmético de suma
53	--	Operador aritmético de decremento
54	-	Operador aritmético de resta
55	*	Operador aritmético de multiplicación
56	/	Operador aritmético de división
57	%	Operador aritmético de módulo
58	null	Tipo de dato null
59	true	Tipo de dato booleano con valor true
60	false	Tipo de dato booleano con valor false
61	Identifier	Representa un nombre de una variable o función
62	Decimal	Representa un número decimal
63	Entero	Representa un número entero
64	String	Representa un string vacío o con caracteres
65	Char	Representa un literal de carácter.
66	EOF	Representa el final de la entrada.
67	INVALID	Representa caracteres inválidos para su utilización en recuperación de errores léxicos

No-terminales utilizados

#	No Terminal	Descripción
1	compilation_unit	Inicio de la gramática, acá se inicia la compilación.
2	member_list	Lista de miembros
3	member_separator	Separador de cada uno de las sentencias, en este caso, “;”
4	member	Miembro, estas son funciones, variables, imports.
5	import_separator	Separador de importaciones, para realizar nombres de importaciones de manera gramática y no léxica.
6	import_content	Contenido de un import.
7	import_name	Nombre del import.
8	import_list	Lista de nombres de importaciones.
9	function	Una declaración completa de una función
10	params	Parametro de la declaración de la función
11	param_list	Lista de parámetros de una función

12	environment_member	Miembro de un ambiente, asignaciones, declaraciones, ifs, whiles, switches, dowhiles, fors, try-catch, declaración de estructuras, etc.
13	environment_body	Cuerpo de un ambiente, es una lista de miembros pero encerrado entre “{“ y “}”.
14	environment	Es un ambiente, utilizado para crear nuevos ids de ámbitos.
15	id_list	Lista de ids para una declaración.
16	declarated_value	Valor de una declaración.
17	declaration	Declaración de variables, globales y constantes.
18	asignation	Asignación de variables
19	simple_if	Sentencia de control IF
20	else	Sentencia de control else
21	else_if	Sentencia de control else if
22	else_if_list	Lista de else-ifs
23	if	Sentencia de control if, utilizada en conjunto con else y else ifs.
24	case	Sentencia de control case.
25	case_list	Lista de sentencias case.
26	default	Sentencia de control default.
27	switch_body	Combinación de sentencias de control case y defaults
28	switch	Sentencia de control switch
29	while_exp	Token While con una expresión
30	while	Sentencia de control while
31	do_while	Sentencias de control do-while
32	for_init	Inicialización en un for
33	for_cond	Condición de un for
34	for_final	Sentencia de finalización de un for
35	for	Sentencia de control FOR
36	break	Sentencia de control de flujo breaj
37	continue	Sentencia de control de flujo continue
38	return	Sentencia return
39	try	Sentencia de control try
40	catch	Sentencia de control catch
41	try_catch	Unión de try y catch
42	throw	Sentencia de control throw
43	print	Sentencia de impresión PRINT.
44	expression	Expresión, esta devuelve un valor.
45	conditional_or_expression	Expresión lógica OR
46	conditional_and_expression	Expresión lógica AND
47	exclusive_or_expression	Expresión lógica XOR
48	equality_expression	Expresión de igualdad y desigualdad
49	relational_expression	Expresiones relacionales mayor-igual, menor-igual, mayor, menor.
50	additive_expression	Expresiones aditivas, suma, resta
51	multiplicative_expression	Expresiones multiplicativas, multiplicación, divisón, módulo.
52	power_expression	Expresion de potencia.

53	unary_expression	Expresiones unarias, menos unaria, expresion lógica NOT.
54	postfix_expression	Expresiones postfijas, auto incremento, decremento, casteos.
55	primary	Expresiones primarias, literales y expresiones encerradas en paréntesis.
56	data_list	Lista de datos, para inicializar arreglos.
57	exp_list	Lista de expresiones, para llamadas a parámetros.
58	array_instance	Inicializacion de un array.
59	call	Llamadas a funciones.
60	call_params_list_with_id	Parametros de funciones delimitadas por nombre y un valor.
61	call_param_with_id	Parámetros de funciones, como una lista de expresiones.
62	struct_head	Inicialización de estructuras.
63	struct_attributes	Lista de atributos de estructuras
64	struct_attribute	Atributos de las estructuras.
65	struct	Una estructura definida, completa.
66	Type	Representa el tipo de un dato, puede ser entero, decimal, double, char, o un identificador.

Gramática utilizada

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
// INICIO DE GRAMATICA
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////

compilation_unit
  : EOF
  | member_list EOF
  ;


// Lista de miembros ejecutables de un programa

member_list
  : member
  | member_list member
  ;

member_separator
    : SEMICOLON
    //| /* epsilum */
    ;

member
  : import_list member_separator
  | function
  | declaration member_separator
  ;
```

```
// lista de imports

import_separator
  : OPERATOR_SUBTRACTION
  | POINT
  ;

import_content
  : identifier
  | literal
  ;

import_name
  : identifier
  | import_name import_separator import_content
  ;

import_list
  : keyword_import import_name
  | import_list COMMA import_name
  ;

////////////////////////////////////////////////
// declaracion de funciones
////////////////////////////////////////////////

function
  : type identifier params environment
  | identifier identifier params environment
  | keyword_void identifier params environment
  ;

params
  : LEFT_PAREN param_list RIGHT_PAREN
  | LEFT_PAREN RIGHT_PAREN
  ;

param_list
  : type identifier
  | identifier identifier
  | param_list COMMA type identifier
  | param_list COMMA identifier identifier
  ;
```

```
////////////////////////////////////////////////////////
// AMBIENTES
////////////////////////////////////////////////////////

environment_member
  : asignation member_separator
  | declaration member_separator

  | if
  | switch

  | for
  | while
  | do_while member_separator

  | break member_separator
  | continue member_separator

  | return member_separator

  | try_catch
  | throw

  | print member_separator
  | struct member_separator
  | postfix_expression member_separator
  ;

environment_body
  : environment_member
  | environment_body environment_member
  ;

environment
  : EMBRACE UNBRACE
  | EMBRACE environment_body UNBRACE
  ;

//////////////////////////////////////////
// Variables
//////////////////////////////////////////
id_list
  : identifier
  | id_list COMMA identifier
  ;

declarated_value
  : OPERATOR_ASSIGNMENT expression
  | /* epsilum */
  ;
```

```
declaration
  : type id_list declarated_value
  | identifier id_list declarated_value
  | keyword_var identifier  COLON declarated_value
  | keyword_const identifier COLON declarated_value
  | keyword_global identifier COLON declarated_value
  ;

asignation
  : identifier OPERATOR_ASSIGNMENT expression
  ;

////////////////////////////////////////////////////
// SENTENCIA DE CONTROL
////////////////////////////////////////////////////

/////// SENTENCIAS DE CONTROL //////////////////////

////// IF

simple_if
  : keyword_if LEFT_PAREN expression RIGHT_PAREN environment
  ;

else
  : keyword_else environment
  ;

else_if
  : keyword_else simple_if
  ;

else_if_list
  : else_if
  | else_if_list else_if
  ;

if
  : simple_if
  | simple_if else
  | simple_if else_if_list
  | simple_if else_if_list else
  ;

//////////// SWITCH
case
  : keyword_case expression COLON environment_body
  ;

case_list
  : case
  | case_list case
  ;

default
  : keyword_default COLON environment_body
  ;
```

```
switch_body
  : EMBRACE case_list UNBRACE
  | EMBRACE case_list default UNBRACE
  | EMBRACE default UNBRACE
  | EMBRACE UNBRACE
  ;

switch
  : keyword_switch LEFT_PAREN expression RIGHT_PAREN switch_body
  ;

/////// SENTENCIAS CICLICAS ////////////////////
while_exp
  : keyword_while LEFT_PAREN expression RIGHT_PAREN
  ;

////// WHILE
while
  : while_exp environment
  ;

////// DO-WHILE
do_while
  : keyword_do environment while_exp
  ;

////// FOR
for_init
  : asignation
  | declaration
  | /* epsilum */
  ;

for_cond
  : expression
  | /* epsilum */
  ;

for_final
  : expression
  | asignation
  | /* epsilum */
  ;

for
  : keyword_for LEFT_PAREN for_init SEMICOLON for_cond SEMICOLON for_final
RIGHT_PAREN environment
  ;

///// SENTENCIAS DE TRANSFERENCIA
break
  : keyword_break
  ;
```

```
continue
  : keyword_continue
  ;

////////// SENTENCIA RETURN ////////////////////////

return
  : keyword_return expression
  | keyword_return
  ;

/////////////////////////////////////////////////////
// EXCEPCIONES
/////////////////////////////////////////////////////
try
  : keyword_try environment
  ;

catch
  : keyword_catch LEFT_PAREN identifier identifier RIGHT_PAREN environment
  ;

try_catch
  : try catch
  ;

throw
  : keyword_throw keyword_strc identifier LEFT_PAREN RIGHT_PAREN
  ;

/////////////////////////////////////////////////////
// IMPRIMIR
/////////////////////////////////////////////////////
print
  : keyword_print LEFT_PAREN expression RIGHT_PAREN
  ;

/////////////////////////////////////////////////////
// EXPRESIONES
/////////////////////////////////////////////////////

// expresiones aritmeticas, logicas, relacionales, de acceso, etc.
expression
  : conditional_or_expression
  | data_list
  | array_instance
  ;

// a || b
conditional_or_expression
  : conditional_and_expression
  | conditional_or_expression OPERATOR_LOGICAL_OR conditional_and_expression
  ;
```

```
// a && b
conditional_and_expression
  : exclusive_or_expression
  | conditional_and_expression OPERATOR_LOGICAL_AND exclusive_or_expression
  ;


// a ^ b
exclusive_or_expression
  : equality_expression
  | exclusive_or_expression OPERATOR_XOR equality_expression
  ;

// a == b
// a != b
equality_expression
  : relational_expression
  | equality_expression OPERATOR_EQUAL relational_expression
  | equality_expression OPERATOR_NOT_EQUAL relational_expression
  | equality_expression OPERATOR_REFERENCE_EQUAL relational_expression
  ;

// a < b
// a <= b
// a > b
// a >= b
relational_expression
  : additive_expression
  | relational_expression OPERATOR_LESS_THAN additive_expression
  | relational_expression OPERATOR_LESS_THAN_EQUAL additive_expression
  | relational_expression OPERATOR_GREATER_THAN additive_expression
  | relational_expression OPERATOR_GREATER_THAN_EQUAL additive_expression
  ;

// a + b
// a - b
additive_expression
  : multiplicative_expression
  | additive_expression OPERATOR_ADDITION multiplicative_expression
  | additive_expression OPERATOR_SUBTRACTION multiplicative_expression
  ;

// a * b
// a / b
// a % b
multiplicative_expression
  : power_expression
  | multiplicative_expression OPERATOR_MULTIPLICATION power_expression
  | multiplicative_expression OPERATOR_DIVISON power_expression
  | multiplicative_expression OPERATOR_MODULO power_expression
  ;

// a ^^ b
power_expression
  : unary_expression
  | power_expression OPERATOR_POT unary_expression
  ;
```

```
// -a
// !a
// (a) b
unary_expression
  : postfix_expression
  | OPERATOR_SUBTRACTION unary_expression
  | OPERATOR_NEGATION unary_expression
  | LEFT_PAREN type RIGHT_PAREN unary_expression
  ;

//a++
//a--
//++a
//--a
postfix_expression
  : primary
  | identifier
  | identifier OPERATOR_INCREMENT
  | identifier OPERATOR_DECREMENT
  | call
  ;

// identificadores
// true, false,
// 0,1,2,3,...
// "strings"
primary
  : literal
  | LEFT_PAREN exclusive_or_expression RIGHT_PAREN
  ;

// lista de corchetes {exp, exp, exp}
data_list
  : EMBRACE exp_list UNBRACE
  ;

// lista de expresiones: exp [,exp]*
exp_list
  : expression
  | exp_list COMMA expression
  ;

// declaracion de arreglos
array_instance
  : keyword_strc type LEFT_BRACKET expression RIGHT_BRACKET
  ;
```

```
////////////////////////////////////////////////////
// LLAMADAS A FUNCIONES
////////////////////////////////////////////////////

// funct1(exp1, exp2, ..., expn)
// funct2(val1= exp1, val2= exp2, ..., valn = expn)
// funct3()
call
  : identifier LEFT_PAREN exp_list RIGHT_PAREN
  | identifier LEFT_PAREN RIGHT_PAREN
  ;

// lista de llamadas de parametro con identificador
call_params_list_with_id
  : call_params_list_with_id COMMA call_param_with_id
  | call_param_with_id
  ;

// llamada de parametro con identificador
call_param_with_id
  : identifier OPERATOR_ASSIGNMENT expression
  ;

////////////////////////////////////////////////////////////
// DEFINICION DE ESTRUCTURAS
////////////////////////////////////////////////////////////

struct_head
  : keyword_define identifier keyword_as
  ;

struct_attributes
  : struct_attribute
  | struct_attributes COMMA struct_attribute
  ;

struct_attribute
  : type identifier
  | type identifier operator_assignment expression
  | identifier identifier
  | identifier identifier operator_assignment expression
  ;

struct
  : struct_head LEFT_BRACKET struct_attributes RIGHT_BRACKET
  ;

type
  : keyword_boolean
  | keyword_integer
  | keyword_double
  | keyword_char
  | keyword_string
  ;
```