Organización de Lenguajes y Compiladores 2
Gramática J#

JAVIER ALBERTO CABRERA PUENTE

Introducción

A continuación se encuentran todas las especificaciones acerca de la gramática que define a "J#". Las especificaciones incluyen los detalles pertenecientes al análisis léxico y sintáctico especialmente, es decir, los símbolos terminales, no terminales y sus reglas de definición, así como las acciones tomadas para construir el árbol de análisis sintáctico o definición dirigida por la sintaxis.

Análisis Léxico y Símbolos Terminales

El lenguaje cuenta con un total de 72 símbolos terminales, definidos a continuación según su expresión regular, las expresiones regulares siguen las reglas de regex de javascript.

No.	Terminal	Regex
1	espacios en blanco	\s+
2	comentarios simples	// . *
3	comentarios compuestos	[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]
4	nullValue	"null"
5	trueValue	"true"
6	falseValue	"false"
7	intType	"integer"

8	doubleType	“double”
9	charType	“char”
10	booleanType	“boolean”
11	varKW	“var”
12	constKW	“const”
13	globalKW	“global”
14	importKW	“import”
15	ifKW	“if”
16	elseKW	“else”
17	switchKW	“switch”
18	caseKW	“case”
19	defaultKW	“default”
20	breakKW	“break”
21	continueKW	“continue”
22	returnKW	“return”
23	voidKW	“void”
24	forKW	“for”
25	whileKW	“while”
26	defineKW	“define”
27	asKW	“as”
28	strcKW	“strc”
29	doKW	“do”
30	tryKW	“try”
31	catchKW	“catch”
32	throwKW	“throw”
33	printKW	“print”

34	equalsReference	“===”
35	equalsValue	“==”
36	greaterEquals	“>=”
37	colonAsignment	“:=”
38	incOp	“++”
39	decOp	“--”
40	powOp	“^^”
41	notEquals	“!=”
42	lessEquals	“<=“
43	andOp	“&&”
44	orOp	“||”
45	comma	“,”
46	semicolon	“;”
47	colon	“:”
48	dot	“.”
49	asignment	“=”
50	leftP	“(”
51	rightP	“)”
52	leftS	“[”
53	rightS	“]”
54	leftC	“{”
55	rightC	“}”
56	plusOp	“+”
57	minusOp	“-”
58	timesOp	“*”
59	divOp	“/”

60	modOp	“%”
61	notOp	“!”
62	xorOp	“^”
63	greaterThan	“>”
64	lessThan	“<”
65	byValue	“$”
66	charValue (recortada)	[‘][\x00-\xFF][‘]
67	fileName	([a-zA-Z0-9]|”.”|”-”)+”.j”
68	id	([a-zA-Z0-9])[a-zA-Z0-9_]*
69	stringValue	[“](“\\”[“\n\r\t\\]|[^”])[“]
70	doubleValue	[0-9]+”.”[0-9]\b
71	intValue	[0-9]+\b
72	EOF	<<EOF>>

Precedencia Utilizada

Se utilizó una precedencia por niveles y asociaciones para manejar las expresiones, se define en la siguiente tabla:

Nivel	Operador	Descripción	Asociatividad
9	- !	negaciones lógica y aritmética	derecha
8	^^	potencia	izquierda
7	* / %	multiplicación, división y módulo	izquierda
6	+ -	suma y resta	izquierda

5	< <= > >=	menor que, menor igual que, mayor que y mayor igual que	no asociativos
4	!= == ===	no igual, igualdad de valor, igualdad de referencia	derecha
3	&&	conjunción	izquierda
2	||	disyunción	izquierda
1	^	disyunción exclusiva	izquierda

Símbolos No Terminales

En el presente documento los símbolos no terminales se representan con nombres cuyos caracteres son todos mayúsculas, a continuación se encuentran numerados, junto con una explicación corta de su función en la gramática del lenguaje:

No.	No Terminal	Descripción
1	**INIT**	Utilizado únicamente por convención para tener un acceso limpio a la raíz del árbol de análisis sintáctico.
2	**SCRIPT**	Utilizado para encapsular a todas las acciones que pueden ser tomadas desde el entorno global
3	**IMPORT**	Encapsula la instrucción de importación dentro del script.
4	**FILES**	Crea una lista de nombres de archivos para ser importados.
5	**DECL**	Genera una lista de acciones globales que si pueden repetirse.
6	**DECL_OPT**	Encapsula todas las opciones que una opción global

		puede tomar, que pueden ser repetidas en el cuerpo del script.
7	**FUNCTION_DECL**	Encapsula la declaración de una función
8	**TYPE**	Encapsula los tipos primitivos que existen en J#
9	**PARAMETERS**	Encapsula una lista de parámetros dentro de paréntesis.
10	**PARAMETERS_LIST**	Crea una lista de parámetros
11	**PARAMS_VALUE**	Encapsula todas las distintas formas que un parámetro puede tomar.
12	**BLOCK**	Encapsula una lista de sentencias dentro de llaves.
13	**VAR_DECL**	Encapsula todas las diferentes formas de declarar una variable.
14	**ID_LIST**	Genera una lista de identificadores.
15	**VAR_T1**	Encapsula la forma que toma una declaración de variable de tipo 1.
16	**VAR_T2**	Encapsula la forma que toma una declaración de variable de tipo 2.
17	**VAR_T3**	Encapsula la forma que toma una declaración de variable de tipo 3.
18	**VAR_T4**	Encapsula la forma que toma una declaración de variable de tipo 4.
19	**VAR_T5**	Encapsula la forma que toma una declaración de variable de tipo 5.
20	**ARRAY_DECL**	Encapsula la declaración de un arreglo.
21	**E_LIST**	Genera una lista de expresiones.
22	**STRC_DEF**	Define la forma que debe tomar la declaración de una estructura.
23	**ATT_LIST**	Genera una lista de atributos para ser incluidos en la declaración de una estructura.
24	**ATTRIBUTE**	Encapsula las diferentes formas que puede tomar un atributo de una estructura

25	**EXPRESSION**	Encapsula todas las distintas formas que una expresión puede tomar
26	**CAST**	Define las distintas formas que un casteo puede tomar
27	**SENTENCES**	Genera una lista de sentencias
28	**SENTENCE**	Encapsula a todas las acciones que pueden ser realizadas dentro de una función o método.
29	**ASIGNMENT**	Encapsula las diferentes formas que una asignación puede tomar.
30	**ACCESS_LIST**	Genera una lista de accesos.
31	**ACCESS**	Define las formas que un acceso puede tomar.
32	**CALL**	Define la estructura de una llamada a una función.
33	**EXP_LIST**	Define las diferentes formas que puede tomar un parámetro en una llamada y genera una lista de estas.
34	**IF_SENTENCE**	Define la estructura de una sentencia If.
35	**ELSE_SENTECE**	Define la estructura de una sentencia Else.
36	**SWITCH_SENTENCE**	Define la estructura de una sentencia Switch.
37	**SWITCH_BODY**	Define la estructura del cuerpo de una sentencia Switch.
38	**CASES_LIST**	Genera una lista de casos para una sentencia switch.
39	**SINGLE_CASE**	Define la estructura de un solo caso de una setencia switch.
40	**DEFAULT_CASE**	Define la estructura del caso default de una sentencia switch.
41	**WHILE_SENTENCE**	Define la estructura de una sentencia While.
42	**DOWHILE_SENTENCE**	Define la estructura de una sentencia Dowhile.
43	**FOR_SENTENCE**	Define la estructura de una sentencia For.
44	**FOR_BODY**	Define la estructura del recorrido y aumento de la variable de referencia en una sentencia For.

45	**FOR_START**	Define la estructura de la utilización de la variable de referencia en una sentencia For.
46	**FOR_END**	Define la estructura de aumento de la variable de referencia en una sentencia For.
47	**PRINT_SENTENCE**	Define la estructura de una sentencia Print.
48	**THROW_SENTENCE**	Define la estructura de una sentencia Throw.
49	**TRY_SENTENCE**	Define la estructura de una sentencia Try.

Definición de la Gramática

A continuación se definen las reglas gramaticales de cada símbolo no terminal, para fines de formato y claridad se define una tabla por cada símbolo no terminal con sus reglas y acciones semánticas para poder formar el árbol de análisis sintáctico. Para cada símbolo la variable de retorno será denominada retorno, también se utiliza pseudocódigo para definir los métodos de programación utilizados. Los símbolos no terminales se encuentran en mayúsculas y negritas, mientras que los terminales se encuentran en minúsculas, sin negrita. La regla sintáctica se encuentra a la izquierda, las reglas semánticas se encuentran a la derecha.

INIT :=

SCRIPT eof	ret = { script.ret, globales, funciones, estructuras }

SCRIPT :=

IMPORT DECL	ret = new Raiz();
	ret.agregarGlobal(import);
	ret.agregarGlobales(decl);

DECL IMPORT	

DECL IMPORT DECL	ret = new Raiz(); ret.agregarGlobales(decl1);
IMPORT	ret = new Raiz(); ret.agregarGlobal(import);
DECL	ret = new Raiz(); ret.agregarGlobal(decl);

IMPORT :=

importKW **FILES**	ret = new Import(files);
importKW **FILES** semicolon	

FILES :=

FILES comma fileName	ret = files.push(new Identificador(fileName));
fileName	ret = [new Identificador(fileName)];

DECL :=

DECL DECL_OPT	ret = decl.push(decl_opt);
DECL_OPT	ret = [decl_opt];

DECL_OPT :=

VAR_DECL semicolon	ret = var_decl;
VAR_DECL	
FUNCTION_DECL	ret = function_decl;
ARRAY_DECL	ret = array_decl;
ARRAY_DECL semicolon	
STRC_DEF	ret = strc_def;
STRC_DEF semicolon	

FUNCTION_DECL :=

TYPE id **PARAMETERS BLOCK**	ret = new Funcion(type, id, parameters, block);
TYPE leftS rightS id **PARAMETERS BLOCK**	type.esArreglo = true; ret = new Funcion(type, id, parameters, block);
id id **PARAMETERS BLOCK**	ret = new Funcion(new Tipo(id1, false), id2, parameters, block);
id leftS rightS id **PARAMETERS BLOCK**	ret = new Funcion(new Tipo(id1, true), id2, parameters, block);
void id **PARAMETERS BLOCK**	ret = new Funcion(id, parameters, block);

TYPE :=

intType	ret = new Type(intType);

doubleType	ret = new Type(doubleType);
booleanType	ret = new Type(booleanType);
charType	ret = new Type(charType);

PARAMETERS :=

leftP **PARAMETERS_LIST** rightP	ret = parameters_list;
leftP rightP	ret = [ ];

PARAMETERS_LIST :=

PARAMETERS_LIST PARAMS_VALUE	ret = parameters_list.push(params_value);
PARAMS_VALUE	ret = [params_value];

PARAMS_VALUE :=

TYPE id	ret = new Parametro(new Tipo(type), new Identificador(id));
TYPE leftS rightS id	ret = new Parametro(new Tipo(type, true), new Identificador(id));
id id	ret = new Parametro(new Tipo(id), new Identificador(id));
id leftS rightS id	ret = new Parametro(new Tipo(id, true), new Identificador(id));
varKW id	ret = new Parametro(new Tipo(varkw), new Identificador(id);)

BLOCK :=

leftC **SENTENCES** rightC	ret = sentences;
leftC rightC	ret = [ ];

VAR_DECL :=

VAR_T1	ret = var_t1;
VAR_T2	ret = var_t2;
VAR_T3	ret = var_t3;
VAR_T4	ret = var_t4;
VAR_T5	ret = var_t5;

ID_LIST :=

ID_LIST comma id	ret = id_list.push(new Identificador(id));
id	ret = [new Identificador(id)];

VAR_T1 :=

TYPE ID_LIST asignment **EXPRESSION**	ret = new Variable(type, id_list, expression);
id **ID_LIST** asignment **EXPRESSION**	ret = new Variable(new Tipo(id1), id_list, expression);

VAR_T2 :=

varKW id asignment **EXPRESSION**	ret = new Variable(new Type(varkw), , new Identificador(id), expression);

VAR_T3 :=

constKW id asignment **EXPRESSION**	ret = new Variable(new Type(constkw), , new Identificador(id), expression);

VAR_T4 :=

globalKW id asignment **EXPRESSION**	ret = new Variable(new Type(globalkw), , new Identificador(id), expression);

VAR_T5 :=

TYPE ID_LIST	ret = new Variable(type, id_list);
id **ID_LIST**	ret = new Variable(new Tipo(id1), id_list);

ARRAY_DECL :=

TYPE leftS rightS **ID_LIST** asignment strcKW **TYPE** leftS **EXPRESSION** rightS	ret = new Arreglo(type1, id_list, expression);
id leftS rightS **ID_LIST** asignment strcKW id leftS **EXPRESSION** rightS	ret = new Arreglo(new Tipo(id1), id_list, expression);

TYPE leftS rightS **ID_LIST** asignment leftC **E_LIST** rightC	ret = new Arreglo(type, id_list, e_list);
id leftS rightS **ID_LIST** asignment leftC **E_LIST** rightC	ret = new Arreglo(new Tipo(id1), id_list, e_list);

E_LIST :=

E_LIST comma **EXPRESSION**	ret = e_list.push(expression);
EXPRESSION	ret = [expression];

STRC_DEF :=

defineKW id asKW leftS **ATT_LIST** rightS	ret = new Strc(new Identificador(id), att_list);

ATT_LIST :=

ATT_LIST comma **ATTRIBUTE**	ret = att_list.push(attribute);
ATTRIBUTE	ret = [attribute];

ATTRIBUTE :=

TYPE id	ret = new Atributo(type, new Identificador(id));
id id	ret = new Atributo(new Tipo(id), new Identificador(id));

TYPE leftS rightS id	ret = new Atributo(type.esArreglo = true, new Identificador(id));
id leftS rightS id	ret = new Atributo(new Tipo(id1, true), new Identificador(id2));
TYPE id asignment **EXPRESSION**	ret = new Atributo(type, new Identificador(id), expression);
id id asignment **EXPRESSION**	ret = new Atributo(new Tipo(id1), new Identificador(id2), expression);
TYPE leftS rightS id asignment **EXPRESSION**	ret = new Atributo(type.esArreglo = true, new Identificador(id), expression);
id leftS rightS id asignment **EXPRESSION**	ret = new Atributo(new Tipo(id1, true), new Identificador(id2), expression);

EXPRESSION :=

EXPRESSION xorOp **EXPRESSION**	ret = new Binaria(exp1, exp2, xorOp);
EXPRESSION orOp **EXPRESSION**	ret = new Binaria(exp1, exp2, orOp);
EXPRESSION andOp **EXPRESSION**	ret = new Binaria(exp1, exp2, andOp);
notOp **EXPRESSION**	ret = new Unaria(exp, notOp);
EXPRESSION notEquals **EXPRESSION**	ret = new Binaria(exp1, exp2, notEquals);
EXPRESSION equalsValue **EXPRESSION**	ret = new Binaria(exp1, exp2, equalsValue);
EXPRESSION equalsReference **EXPRESSION**	ret = new Binaria(exp1, exp2, equalsReference);
EXPRESSION lessThan **EXPRESSION**	ret = new Binaria(exp1, exp2, lessThan);
EXPRESSION lessEquals **EXPRESSION**	ret = new Binaria(exp1, exp2, lessEquals);
EXPRESSION greaterThan **EXPRESSION**	ret = new Binaria(exp1, exp2, greaterThan);
EXPRESSION greaterEquals **EXPRESSION**	ret = new Binaria(exp1, exp2,

	greaterEquals);
EXPRESSION plusOp **EXPRESSION**	ret = new Binaria(exp1, exp2, plusOp);
EXPRESSION minusOp **EXPRESSION**	ret = new Binaria(exp1, exp2, minusOp);
EXPRESSION timesOp **EXPRESSION**	ret = new Binaria(exp1, exp2, timesOp);
EXPRESSION divOp **EXPRESSION**	ret = new Binaria(exp1, exp2, divOp);
EXPRESSION modOp **EXPRESSION**	ret = new Binaria(exp1, exp2, modOp);
EXPRESSION powOp **EXPRESSION**	ret = new Binaria(exp1, exp2, powOp);
minusOp **EXPRESSION**	ret = new Unaria(exp, minusOp);
leftP **EXPRESSION** rightP	ret = expression;
stringValue	ret = new Cadena(stringvalue);
intValue	ret = new Entero(intvalue);
doubleValue	ret = new Decimal(doublevalue);
trueValue	ret = new Booleano(trueValue);
falseValue	ret = new Booleano(falsevalue);
charValue	ret = new Caracter(charValue);
id	ret = new Identificador(id);
nullValue	ret = new Nulo(nullvalue);
CAST EXPRESSION	ret = new Casteo(cast, expression);
strcKW id leftS **EXPRESSION** rightS	ret = new Strc(new Identificador(id), expression)
strcKW **TYPE** leftS **EXPRESSION** rightS	ret = new Strc(type, expression);
leftC **E_LIST** rightC	ret = new Arreglo(e_list);
strcKw id leftP rightP	ret = new Strc(new Identificador(id));
id dot id	ret = new Acceso(new Identificador(id2));
id dot id ACCESS_LIST	ret = new Acceso(new Identificador(id2));
id leftS **EXPRESSION** rightS	ret = new Acceso(new Identificador(id2));

id leftS **EXPRESSION** rightS **ACCESS_LIST**	ret = new Acceso(new Identificador(id), access_list);
id dot **CALL**	ret = [new Identificador, call]
id dot **CALL ACCESS_LIST**	ret = [call, access_list.hijos];
CALL	ret = call;

CAST :=

leftP intType rightP	ret = new Tipo(intType);
leftP doubleType rightP	ret = new Tipo(doubleType);
leftP charType rightP	ret = new Tipo(chartype);

SENTENCES :=

SENTENCES SENTENCE	ret = sentences.push(sentence);
SENTENCE	ret = [sentence];

SENTENCE :=

STRC_DEF	ret = strc_def;
STRC_DEF semicolon	
VAR_DECL	ret = var_decl;
VAR_DECL semicolon	
ARRAY_DECL	ret = array_decl;

ARRAY_DECL semicolon	
ASIGNMENT	ret = asignment;
ASIGNMENT semicolon	
IF_SENTENCE	ret = if_sentence;
SWITCH_SENTENCE	ret = switch_sentence;
WHILE_SENTENCE	ret = while_sentence;
DOWHILE_SENTENCE	ret = dowhile_sentence;
DOWHILE_SENTENCE semicolon	
FOR_SENTENCE	ret = for_sentence;
PRINT_SENTENCE	ret = print_sentence;
PRINT_SENTENCE semicolon	
THROW_SENTENCE	ret = throw_sentence;
THROW_SENTENCE semicolon	
TRY_SENTENCE	ret = try_sentence;
breakKW	ret = new Break(breakkw);
breakKW semicolon	
continueKW	ret = new Continue(continuekw)
continueKW semicolon	
returnKW semicolon	ret = new Return(returnkw);
returnKW **EXPRESSION**	ret = new Return(returnkw, expression);
returnKW **EXPRESSION** semicolon	
CALL	ret = call;

ASIGNMENT :=

id asignment **EXPRESSION**	ret = new Asignacion(new Identificador(id), expression);
id dot id asignment **EXPRESSION**	ret = new Asignacion(new Identificador(id), expression);
id dot id **ACCESS_LIST** asignment **EXPRESSION**	ret = new Asignacion(new Identificador(id), expression);
id leftS **EXPRESSION** rightS asignment **EXPRESSION**	ret = new Asignacion(new Identificador(id), expression);
id leftS **EXPRESSION** rightS **ACCESS_LIST** asignment **EXPRESSION**	ret = new Asignacion(new Identificador(id), expression);
id dot **CALL** asignment **EXPRESSION**	ret = new Asignacion(new Identificador(id), expression);
id dot **CALL ACCESS_LIST** asignment **EXPRESSION**	ret = new Asignacion(new Identificador(id), expression);

ACCESS_LIST :=

ACCESS_LIST ACCESS	ret = access_list.push(access);
ACCESS	ret = [access];

ACCESS :=

dot id	ret = new Acceso(new Identificador(id));
leftS **EXPRESSION** rightS	ret = new Acceso(expression);
dot **CALL**	ret = new Acceso(call);

CALL :=

idleftP **EXP_LIST** rightP	ret = new Llamada(exp_listl);
id leftP rightP	ret = new Llamada(new Identficador(id));

EXP_LIST :=

EXP_LIST EXP_OPT	ret = exp_list.push(exp_opt);
EXP_OPT	ret = [exp_opt];

IF_SENTENCE :=

ifKW leftP **EXPRESSION** rightP **BLOCK**	ret = new Si(expression, block, null);
ifKW leftP **EXPRESSION** rightP **BLOCK ELSE_SENTENCE**	ret = new Si(expression, block, else_sentence);

ELSE_SENTENCE :=

elseKW **IF_SENTENCE**	ret = new Sino(if_sentence);
elseKW **BLOCK**	ret = new Sino(block);

SWITCH_SENTENCE :=

switchKW leftP **EXPRESSION** rightP leftC **SWITCH_BODY** righC	ret = new Cambiar(expression, switch_body);

SWITCH_BODY :=

CASES_LIST	ret = cases_list;
CASES_LIST DEFAULT_CASE	ret = cases_list.push(default_case);

CASES_LIST :=

CASES_LIST SINGLE_CASE	ret = cases_list.push(single_case);
SINGLE_CASE	ret = [single_case]

SINGLE_CASE :=

caseKW **EXPRESSION** colon **SENTENCES**	ret = new Caso(expression, sentences);

DEFAULT_CASE :=

defaultKW colon **SENTENCES**	ret = new PorDefecto(sentences);

WHILE_SENTENCE :=

whileKW leftP **EXPRESSION** rightP **BLOCK**	ret = new Mientras(expression, block);

DOWHILE_SENTENCE :=

doKW **BLOCK** whileKW leftP **EXPRESSION** rightP	ret = new HacerMientras(block, expression);

FOR_SENTENCE :=

forKW leftP **FOR_BODY** rightP **BLOCK**	ret = new Para(for_body, block);

FOR_BODY :=

FOR_START semicolon **EXPRESSION** semicolon **FOR_END**	ret = new CuerpoFor(for_start, expression, for_end);
FOR_START semicolon semicolon **FOR_END**	ret = new CuerpoFor(for_start,null, for_end);
FOR_START semicolon **EXPRESSION** semicolon	ret = new CuerpoFor(for_start, expression,null);
FOR_START semicolon semicolon	ret = new CuerpoFor(for_start,null, null);
semicolon **EXPRESSION** semicolon **FOR_END**	ret = new CuerpoFor(null, expression, for_end);
semicolon **EXPRESSION** semicolon	ret = new CuerpoFor(null, expression,null);
semicolon semicolon **FOR_END**	ret = new CuerpoFor(null,null, for_end);

semicolon semicolon	ret = new CuerpoFor(null, null, null);

FOR_START :=

TYPE id asignment **EXPRESSION**	ret = new VarT1(type, new Identificador(id), expression);
ASIGNMENT	ret = asignment;

FOR_END :=

EXPRESSION	ret = expression;
ASIGNMENT	ret = asignment;

PRINT_SENTENCE :=

printKW leftP **EXPRESSION** rightP	ret = new Imprimir(expression);

THROW_SENTENCE :=

throwKW strc **CALL**	ret = new Tirar(call);

TRY_SENTENCE :=

tryKE **BLOCK** catchKW leftP **VAR_T5** rightP **BLOCK**	ret = new Trata(block1, vart_t5, block2);