

OSCAR RENE CUELLAR MANCILLA

201503712

Gramáticas

P R O Y E C T O 2

Organización de Lenguajes y Compiladores 2

Descripción

J# es un lenguaje de programación basado en c# y javascript, su mayor característica es el control de ámbitos, lo que nos permite llevar un mejor manejo del consumo de memoria. Este lenguaje está formado por un conjunto de herramientas muy flexibles que pueden ampliarse fácilmente mediante paquetes, librerías o definiendo nuestras propias funciones. Este software también cuenta con un entorno de desarrollo integrado (IDE) que proporciona servicios integrales para facilitarle al programador el desarrollo de aplicaciones.

Expresiones Regulares

1. **Nombre: Digito**

**Patrón:** [0-9]

1. **Nombre: Letra**

**Patrón:** [A-Za-zÑñ]

1. **Nombre: Identificador**

**Patrón:** ({letras}|"\_")({letras}+|{digito}\*|"\_")\*

1. **Nombre: String**

**Patrón:** \"([^\"\n\\\\]|\\\"|\\)\*\"

1. **Nombre: Char**

**Patrón:** \'([^'\n\t]|'\n'|'\t'|'\0'|'\"')?\'

1. **Nombre: Double**

**Patrón:** {digito}+"."{digito}+

1. **Nombre: Integer**

**Patrón:** {digito}+

Precedencia Utilizada

/\* **LA PRECEDENCIA MOSTRADA A CONTINUACIÓN VA DE MAYOR A MENOR NIVEL RESPECTIVAMENTE** \*/

%right '='

%left '.'

%left '++' '--'

%left '^'

%left '||'

%left '&&'

%left '!=' '==' '==='

%nonassoc '>' '>=' '<' '<='

%left '+' '-'

%left '\*' '/' '%'

%right Ttypecast

%left '^^'

%right '!' UTmenos UTmas

%left '[' ']'

Enumeración y cantidad de símbolos terminales y no terminales.

**SIMBOLOS TERMINALES:**

1. "import"
2. "private"
3. "public"
4. "void"
5. "var"
6. "const"
7. "global"
8. "define"
9. "as"
10. "integer"
11. "double"
12. "char"
13. "boolean"
14. "true"
15. "false"
16. "if"
17. "else"
18. "case"
19. "default"
20. "switch"
21. "while"
22. "do"
23. "for"
24. "break"
25. "continue"
26. "return"
27. "strc"
28. "print"
29. "throw"
30. "try"
31. "catch"
32. "null"

Gramática Funcional

S : DECLARACIONES

DECLARACIONES: DECLARACIONES DECLARACION

| DECLARACION

DECLARACION : FUNCION

| DECLARACION\_VARIABLE PUEDE\_SEMICOLON

| DECLARACION\_STRUCT PUEDE\_SEMICOLON

| import LISTA\_ID PUEDE\_SEMICOLON

DECLARACION\_STRUCT : define id as '[' LISTA\_ATRIBUTOS ']'

LISTA\_ATRIBUTOS : LISTA\_ATRIBUTOS ',' ATRIBUTO

| ATRIBUTO

ATRIBUTO : TIPO\_DATO id

| TIPO\_DATO id '=' E

FUNCION : TIPO\_DATO id '(' PARAMETROS ')' ''

| void id '(' PARAMETROS ')' ''

DECLARACION\_VARIABLE : TIPO\_VAR id ':' '=' E

| TIPO\_DATO LISTA\_ID '=' E

| TIPO\_DATO LISTA\_ID

LISTA\_ID : LISTA\_ID ',' id

| id

TIPO\_VAR : var

| const

| global

TIPO\_DATO : TYPE '[' E ']'

| TYPE '[' ']'

| TYPE

TYPE : integer

| double

| char

| boolean

| id

ASIGNACION\_VARIABLE : TIPO\_DATO '=' E

| TIPO\_DATO LIST\_ACCESO1 '=' E

LIST\_ACCESO1: LIST\_ACCESO1 '.' id '[' E ']'

| LIST\_ACCESO1 '.' id

| LIST\_ACCESO1 '.' LLAMADA

| '.' id

| '.' id '[' E ']'

PARAMETROS : LISTA\_PARAMETROS

| /\* empty \*/

LISTA\_PARAMETROS : LISTA\_PARAMETROS ',' TIPO\_DATO id

| TIPO\_DATO id

BLOQUES : LISTA\_BLOQUES

| /\* empty \*/

LISTA\_BLOQUES : LISTA\_BLOQUES BLOQUE

| BLOQUE

BLOQUE : SENTENCIA

| INSTRUCCION PUEDE\_SEMICOLON

| DECLARACION\_VARIABLE PUEDE\_SEMICOLON

SENTENCIA : IF

| SWITCH

| WHILE

| DOWHILE

| FOR

| TRY\_CATCH

NATIVAS : print '(' E ')'

INSTRUCCION : break

| continue

| LLAMADA

| ASIGNACION\_VARIABLE

| RETURN ''

| NATIVAS

| id '++'

| id '--'

RETURN : return E

| return

| throw NEW\_EXCEPTION

TRY\_CATCH : try ''

IF : if '(' E ')' ''

| if '(' E ')' '' ELSE

ELSE : else IF

| else ''

WHILE : while '(' E ')' ''

DOWHILE : do '' while '(' E ')' PUEDE\_SEMICOLON

FOR : for '(' INICIO\_FOR '' CONDICION\_FOR '' FIN\_FOR ')' ''

INICIO\_FOR : id '=' E

| TIPO\_DATO id '=' E

|

CONDICION\_FOR : E

|

FIN\_FOR : E

|

SWITCH : switch '(' E ')' ''

{ $$ = $6

$$.elseif = $7

$$.setExpresionSwitch($3)}

LISTA\_CASOS : LISTA\_CASOS case E ':' BLOQUES

| case E ':' BLOQUES

DEFAULT : default ':' BLOQUES

| /\*empty\*/

PARAMETROS\_LLAMADA : LISTA\_PAR

| /\* empty \*/

LISTA\_PAR : LISTA\_PAR ',' PAR

| PAR

PAR : '$' E

| E

LISTA\_E : LISTA\_E ',' E

| E

E : CONSTANTE

| BINARIA

| UNARIA

| '(' E ')'

| LIST\_ACCESO

| E\_ARREGLO

| NEW\_STRUCT

NEW\_STRUCT : strc id '(' ')'

E\_ARREGLO : strc TYPE '[' E ']'

| ''

LIST\_ACCESO : LIST\_ACCESO '.' ACCESO

| ACCESO

ACCESO : id

| LLAMADA

| id '[' E ']'

LLAMADA : id '(' PARAMETROS\_LLAMADA ')'

NEW\_EXCEPTION := strc EXCEPTION '(' ')'

CONSTANTE : int

| true

| false

| double

| null

| string

| char

BINARIA : ARITMETICA

| LOGICA

| RELACIONAL

TIPO\_CASTEO : integer

| char

| double

UNARIA: '(' TIPO\_CASTEO ')' E

| '-' E

| '!' E

| '+' E

| E '++'

| E '--'

ARITMETICA : E '+' E

| E '-' E

| E '\*' E

| E '/' E

| E '^^' E

| E '%' E

LOGICA : E '||' E

| E '&&' E

| E '^' E

RELACIONAL : E '<' E

| E '<=' E

| E '>' E

| E '>=' E

| E '==' E

| E '===' E

| E '!=' E

EXCEPTION : ae

| iobe

| ue

| npe

| ice

| hoe

| soe

PUEDE\_SEMICOLON : ''

|