Gramáticas

PROYECTO 2

Organización de Lenguajes y Compiladores 2

OSCAR RENE CUELLAR MANCILLA
201503712

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]

2. **Nombre: Letra**
   Patrón: [A-Za-zÑñ]

3. **Nombre: Identificador**
   Patrón: ({letras}|"_")({letras}+|{digito}*|"_")*

4. **Nombre: String**
   Patrón: \"([^\"\n\\\\]|\\\"|\\)*\"

5. **Nombre: Char**
   Patrón: \'([^'\n\t]|'\n'|'\t'|'\0'|'\'')?\'

6. **Nombre: Double**
   Patrón: {digito}+"."{digito}+

7. **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 : ''

|