Gramática Código Alto

Expresiones Regulares

Las expresiones regulares que se usaron para el analizador léxico de JISON son las siguientes

```
[0-9]+"."[0-9]+\b                   return 'decimal';
[0-9]+\b                            return 'numero';
\'[^\']\'                           return 'caracter';
[a-zA-Z][-_a-zA-Z0-9nÑ.]*[.]j\b     return 'nombre_archivo';
[a-zA-Z_][_a-zA-Z0-9nÑ]*\b          return 'identificador';
\"[^\"]*\"              { yytext = yytext.substr(1,yyleng-2); return 'cadena'; }
[ \r\t]+            {}
\n                  {}
"//".*                                  // comentario simple línea
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]     // comentario multiple líneas
```

En JISON se tiene una expresión regular para número que consta cualquier digito que se puede repetir 1 o más veces. Seguimos con la expresión regular para números decimales que comienza con 1 o más dígitos seguido de 1 punto y terminado con uno o más números. Para el identificador en JISON se usó otra **ER** llamada letra, que contiene cualquier letra del abecedario español y el identificador se trata de que podría comenzar con cualquier letra o un punto seguido de 1 o más letras, números o diagonales.

JISON posee una **ER** llamada cadena el cual comienza con doble comilla y cualquier cosa hasta otra doble comilla. La declaración un char viene siendo una comilla simple seguido de cualquier símbolo y termina por otra comilla simple.
Por ultimo vienen los comentarios, el comentario de una línea empieza con un ***//*** y termina hasta que venga un salto de línea y retorno de carro. Para los comentarios de múltiples líneas comienza con ***/**** hasta que encuentra ****/***

Precedencia utilizada

La precedencia usada esta especificada en esta tabla

NIVEL	OPERADOR	DESCRIPCION	ASOCIATIVIDAD
11	[ ]	Acceso a elemento de arreglo	Izquierda
10	- !	menos unario not	Derecha
9	* / % ^^	multiplicativas	Izquierda
8	+ -	aditivas	Izquierda
7	< > <= >=	relacionales	No Asociativo
6	= !=	Igualdad Diferencia	Izquierda
5	&	And	Izquierda
4	|	Or	Izquierda
3	^	Xor	Izquierda
2	++ --	Incremento Decremento	Izquierda
1	=	asignación	Derecha

Para JISON se usó la siguiente gramática

```
EXPRESION->  EXPRESION xor EXPRESION
           | EXPRESION op_or EXPRESION
           | EXPRESION op_and EXPRESION
           | EXPRESION igualacion EXPRESION
           | EXPRESION diferencia EXPRESION
           | EXPRESION igualdad_referencia EXPRESION
           | EXPRESION mayor EXPRESION
           | EXPRESION mayor_igual EXPRESION
           | EXPRESION menor EXPRESION
           | EXPRESION menor_igual EXPRESION
           | EXPRESION mas EXPRESION
           | EXPRESION menos EXPRESION
           | EXPRESION por EXPRESION
           | EXPRESION div EXPRESION
           | EXPRESION modulo EXPRESION
           | EXPRESION potencia EXPRESION
           | menos EXPRESION %prec Umenos
           | not EXPRESION %prec Unot
           | para EXPRESION parc
           | para tk_double parc EXPRESION
           | para tk_integer parc EXPRESION
           | para tk_char parc EXPRESION
           | VALOR
```

Donde tiene los operadores con menos precedencia arriba y los de mayor precedencia abajo. Pero esta gramática es ambigua por lo que uso la precedencia que tiene JISON

```
%left 'incremento' 'decremento'
%left 'xor'
%left 'op_or'
%left 'op_and'
%left 'igualacion' 'diferencia' 'igualdad_referencia'
%left 'mayor' 'mayor_igual' 'menor' 'menor_igual'
%left 'mas' 'menos'
%left 'por' 'div' 'modulo'
%left 'potencia'
%left Umenos
%left Unot
//%left UDouble
//%left UInteger
//%left UChar
%left 'para' 'parc'
```

Cantidad de símbolos terminales

En el analizador JISON se usaron **43** Terminales aproximadamente:

Enumeración de los símbolos terminales

Palabras reservadas: Las palabras reservadas son las que nos ayudara para las sentencia de control, sentencias de escape y asignar algún valor a alguna variable.

```
"null"          return 'tk_null';
"integer"       return 'tk_integer';
"double"        return 'tk_double';
"char"          return 'tk_char';
"boolean"       return 'tk_boolean';
"import"        return 'tk_import';
"var"           return 'tk_var';
"const"         return 'tk_const';
"global"        return 'tk_global';
"true"          return 'tk_true';
"false"         return 'tk_false';
"if"            return 'tk_if';
"else"          return 'tk_else';
"switch"        return 'tk_switch';
"case"          return 'tk_case';
"default"       return 'tk_default';
"break"         return 'tk_break';
```

```
"continue"      return 'tk_continue';
"return"        return 'tk_return';
"print"         return 'tk_print';
"public"        return 'tk_public';
"private"       return 'tk_private';
"void"          return 'tk_void';
"for"           return 'tk_for';
"while"         return 'tk_while';
"define"        return 'tk_define';
"as"            return 'tk_as';
"strc"          return 'tk_strc';
"do"            return 'tk_do';
"try"           return 'tk_try';
"catch"         return 'tk_catch';
"throw"         return 'tk_throw';
```

Caracteres Especiales: Los caracteres especiales nos ayudaran para agrupar código, terminar alguna línea, asignar datos a variables o estructuras y sobre todo para poder realizar cualquier operación matemática

```
";"                 return 'ptcoma';
"("                 return 'para';
")"                 return 'parc';
"["                 return 'corizq';
"]"                 return 'corder';
"{"                 return 'llavea';
"}"                 return 'llavec';
":="                return 'equal2';
":"                 return 'dospts';
","                 return 'coma';
```

```
"++"                return 'incremento';
"--"                return 'decremento';
"+"                 return 'mas';
"-"                 return 'menos';
"*"                 return 'por';
"/"                 return 'div';
"^^"                return 'potencia';
"%"                 return 'modulo';
```

```
"<="                return 'menor_igual';
">="                return 'mayor_igual';
"<"                 return 'menor';
">"                 return 'mayor';
"&&"                return 'op_and';
"||"                return 'op_or';
"==="               return 'igualdad_referencia';
"=="                return 'igualacion';
"="                 return 'equal';
"!="                return 'diferencia';
"!"                 return 'not';
"^"                 return 'xor';
```

Expresión Regular: Estas expresiones regulares ya fueron descritas anteriormente en la sección de Expresiones Regulares y son conformadas por:

```
[0-9]+"."[0-9]+\b                   return 'decimal';
[0-9]+\b                            return 'numero';
\'[^\']\'                           return 'caracter';
[a-zA-Z][-_a-zA-Z0-9nÑ.]*[.]j\b     return 'nombre_archivo';
[a-zA-Z_][_a-zA-Z0-9nÑ]*\b          return 'identificador';
\"[^\"]*\"              { yytext = yytext.substr(1,yyleng-2); return 'cadena'; }
[ \r\t]+            {}
\n                  {}
"//".*                                  // comentario simple línea
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]     // comentario multiple líneas
```

Cantidad de símbolos no terminales

En el analizador JISON se usaron **32** No Terminales aproximadamente.
Cada No Terminal devuelve alguna clase que hereda de la interfaz nosoAst

Explicación de cada uno de los símbolos no terminales (cuál fue su uso dentro de la gramática) Y funcional describiendo cada una de las acciones

INICIO

El NoTerminal Inicio es donde comienza nuestra gramática, aquí solo hace indica que va a ir al NoTerminal Programa o si quiere importar funciones de otro fichero.

```
INICIO-> IMPORTACION INSTRUCCIONES
        | INSTRUCCIONES
;
```

IMPORTACION

El NoTerminal Importacion que es una lista con nombres de los archivos que se desean importar

```
IMPORTACION-> tk_import LISTA_ARCHIVO
            | tk_import LISTA_ARCHIVO ptcoma
;

LISTA_ARCHIVO-> LISTA_ARCHIVO coma nombre_archivo
              nombre_archivo
;
```

INSTRUCCIONES

El NoTerminal Instrucciones sirve para generar una lista ascendente de Contenido. En código javascript se va armando el árbol mientras sube

```
INSTRUCCIONES-> INSTRUCCION INSTRUCCIONES
              |   INSTRUCCION
;
```

INSTRUCCION

Programa_Sentencia tendra todas las opciones a donde ir; desde una asignacion, creacion de funcion o ejecuion de alguna sentencia de control. En codigo javascript solo va subiendo el arbol que cada uno devuelve

```
INSTRUCCION-> DECLARACION
            | DECLARACION ptcoma
            | CREAR_FUNCION
;
```

PROGRAMA

El NoTerminal Programa sirve para generar una lista ascendente de Contenido. En código javascript se va armando el árbol mientras sube

```
PROGRAMA:
        PROGRAMA PROGRAMA_SENTENCIA
        | PROGRAMA_SENTENCIA
;
```

PROGRAMA SENTENCIA

Este tiene el mismo funcionamiento de Contenido, con la diferencia de que no posee la producción Declaracion_Funcion y servirá para el cuerpo de funciones y sentenciad de control

```
PROGRAMA_SENTENCIA->  DECLARACION
                    | DECLARACION ptcoma
                    | ASIGNACION
                    | ASIGNACION ptcoma
                    | PRINT
                    | PRINT ptcoma
                    | AUMENTO

                    | SENTENCIA_IF
                    | SENTENCIA_SWITCH
                    | SENTENCIA_WHILE
                    | SENTENCIA_DOWHILE
                    | SENTENCIA_FOR
                    | SENTENCIA_TRANSFERENCIA
                    | CALL_FUNCION
                    | CALL_FUNCION ptcoma
```

DECLARACION

El NoTerminal Declaración sirve para poder declarar variables, estas variables ahora tienen un único tipo, también pueden ser globales, constantes.

```
DECLARACION-> TIPOS LISTA_ID equal EXPRESION
            | tk_var identificador equal2 EXPRESION
            | tk_const identificador equal2 EXPRESION
            | tk_global identificador equal2 EXPRESION
            | TIPOS LISTA_ID
;

LISTA_ID-> LISTA_ID coma identificador
        | identificador
;

TIPOS-> TIPO
        | TIPO corizq corder
;

TIPO-> tk_integer
        | tk_double
        | tk_char
        | tk_boolean
        | identificador
;
```

La declaración usa **Lista_ID** para crear muchas variables, **Tipos**, que nos indica el tipo de la variable

ASIGNACION

En la asignación esta sirve para poder cambiar el valor de una variable

```
ASIGNACION-> identificador equal EXPRESION
           | identificador corder EXPRESION corizq equal EXPRESION
;
```

AUMENTO

En la No Terminal Aumento nos sirve para incrementar o decremento un valor numérico

```
AUMENTO-> identificador incremento
        | identificador decremento
```

PRINT

Este Noterminal sirve para indicar la creación de una función llamada print, este servirá para poder imprimir cualquier estructura

```
PRINT-> print para EXP parc
;
```

SENTENCIA_IF

El NoTerminal Sentencia_If sirve para poder crear una sentencia if. En el código java se crea el Objecto para el if y se retorno

```
SENTENCIA_IF-> tk_if para EXPRESION parc llavea PROGRAMA llavec
             | tk_if para EXPRESION parc llavea PROGRAMA llavec
                                    tk_else llavea  PROGRAMA llavec
             | tk_if para EXPRESION parc llavea PROGRAMA llavec
                                    tk_else SENTENCIA_IF

;
```

SENTENCIA_WHILE

Este NoTerminal sirve para crear una sentencia while, En código java se crea el objeto Sentecia_While

```
SENTENCIA_WHILE-> tk_while para EXPRESION parc llavea llavec
                | tk_while para EXPRESION parc llavea PROGRAMA llavec
;
```

SENTENCIA_DOWHILE

Este NoTerminal sirve para crear una sentencia Do-while, En código java se crea el objeto Sentecia_DoWhile

```
SENTENCIA_DOWHILE->  tk_do llavea llavec tk_while para EXPRESION parc
                   | tk_do llavea PROGRAMA llavec tk_while para EXPRESION parc
;
```

SENTENCIA_FOR

Este NoTerminal sirve para crear una sentencia For, En código java se crea el objeto Sentecia_For

```
SENTENCIA_FOR-> tk_for para INICIOFOR ptcoma EXPRESION ptcoma FINALFOR parc llavea PROGRAMA llavec
              | tk_for para  ptcoma EXPRESION ptcoma FINALFOR parc llavea PROGRAMA llavec
              | tk_for para INICIOFOR ptcoma ptcoma FINALFOR parc llavea PROGRAMA llavec

              | tk_for para INICIOFOR ptcoma EXPRESION ptcoma parc llavea PROGRAMA llavec
              | tk_for para ptcoma ptcoma FINALFOR parc llavea PROGRAMA llavec
              | tk_for para INICIOFOR ptcoma ptcoma parc llavea PROGRAMA llavec

              | tk_for para ptcoma EXPRESION ptcoma parc llavea PROGRAMA llavec
              | tk_for para ptcoma ptcoma parc llavea PROGRAMA llavec
;
```

```
INICIOFOR-> DECLARACION
          | ASIGNACION
;

FINALFOR-> ASIGNACION
         | AUMENTO
;
```

SENTENCIA_SWITCH

Este NoTerminal sirve para crear un switch, este recibe un valor para comparar y una lista de casos

```
SENTENCIA_SWITCH-> tk_switch para EXPRESION parc llavea LIST_CASE llavec
;
```

LIST_CASE

Este NoTerminal sirve para crear una lista de casos para un switch

```
LIST_CASE-> LIST_CASE SENTENCIA_CASE
          | SENTENCIA_CASE
;
```

SENTENCIA_CASE

Este NoTerminal sirve para crear un Objeto Case para una Sentencia switch. Este puede tener o no un cuerpo de ejecución y existe la opción Default que funciona igual a la de un switch

```
SENTENCIA_CASE-> tk_case EXPRESION dospts PROGRAMA
               | tk_default dospts PROGRAMA
;
```

SENTENCIA_TRANSFERENCIA

Este NoTerminal sirve para crear una sentencia de transferencia las cuales sirve para romper o continuar un ciclo y un para retorna algún valor de alguna función

Hay solo 3 opciones:

1) Break
2) Continue
3) Return (CON O SIN VALOR)

```
SENTENCIA_TRANSFERENCIA-> tk_break
                        | tk_continue
                        | tk_return
                        | tk_return EXPRESION
;
```

CREAR_FUNCION

Crear_Funcion sirve para poder declarar alguna función, este va a poder llevar parámetros gracias a la gramática Lista_Parametros, también podrá devolver algún valor ya sea tipo primitivo, arreglo o struct

```
CREAR_FUNCION: TIPOS identificador para parc llavea PROGRAMA llavec
             | TIPOS identificador para LISTA_PARAMETROS parc llavea PROGRAMA llavec
;

LISTA_PARAMETROS: LISTA_PARAMETROS coma TIPOS identificador
             | TIPOS identificador
```

CALL_FUNCION

Call_Funcion sirve para poder hacer una llamada a una función, en este podremos ingresar una lista de expresiones si la función la requiere

```
CALL_FUNCION-> identificador para parc
             |  identificador para LISTA_EXP parc
;
```

EXP

Este NoTerminal sirve para crear lista de expresiones el cual viene todas las operaciones aritméticas, lógicas, relacionales y comparativas

```
EXPRESION->  EXPRESION xor EXPRESION
           | EXPRESION op_or EXPRESION
           | EXPRESION op_and EXPRESION
           | EXPRESION igualacion EXPRESION
           | EXPRESION diferencia EXPRESION
           | EXPRESION igualdad_referencia EXPRESION
           | EXPRESION mayor EXPRESION
           | EXPRESION mayor_igual EXPRESION
           | EXPRESION menor EXPRESION
           | EXPRESION menor_igual EXPRESION
           | EXPRESION mas EXPRESION
           | EXPRESION menos EXPRESION
           | EXPRESION por EXPRESION
           | EXPRESION div EXPRESION
           | EXPRESION modulo EXPRESION
           | EXPRESION potencia EXPRESION
           | menos EXPRESION %prec Umenos
           | not EXPRESION %prec Unot
           | para EXPRESION parc
           | para tk_double parc EXPRESION
           | para tk_integer parc EXPRESION
           | para tk_char parc EXPRESION
           | VALOR
;
```

VALOR

Contiene todas las opciones que puede tener una variable. Desde estructuras hasta llamada de funciones

```
VALOR-> numero
        | decimal
        | caracter
        | cadena
        | tk_true
        | tk_false
        | identificador
        | identificador incremento
        | identificador decremento
        | VALOR_ARRAY
;
```

```
VALOR_ARRAY-> llavea LISTA_EXP llavec
;

LISTA_EXP->     LISTA_EXP coma EXPRESION
                | EXPRESION
;
```

Gramática Código 3D

Expresiones Regulares

Las expresiones regulares que se usaron para el analizador léxico de JISON son las siguientes

```
l[0-9]+                             return 'etiqueta';
[0-9]+"."[0-9]+\b                   return 'decimal';
[0-9]+\b                            return 'numero';
[a-zA-Z_][_a-zA-Z0-9nÑ]*\b          return 'identificador';
[ \r\t]+                {}
\n                      {}
"#".*                                       // comentario simple línea
[#][*][^*]*[*]+([^#*][^*]*[*]+)*[#]         // comentario multiple líneas
```

En JISON se tiene una expresión regular para número que consta cualquier digito que se puede repetir 1 o más veces. Seguimos con la expresión regular para números decimales que comienza con 1 o más dígitos seguido de 1 punto y terminado con uno o más números. Para el identificador en JISON se usó otra **ER** llamada letra, que contiene cualquier letra del abecedario español y el identificador se trata de que podría comenzar con cualquier letra o un punto seguido de 1 o más letras, números o diagonales.

Por ultimo vienen los comentarios, el comentario de una línea empieza con un # y termina hasta que venga un salto de línea y retorno de carro. Para los comentarios de múltiples líneas comienza con **#*** hasta que encuentra ***#**

Cantidad de símbolos terminales

En el analizador JISON se usaron **40** Terminales aproximadamente:

Enumeración de los símbolos terminales

Palabras reservadas: Las palabras reservadas son las que nos ayudara para las los arreglos stack y heap, saltos condicionales y no condicionales, creación de método y su llamada

```
"var"               return 'tk_var';
"stack"             return 'tk_stack';
"heap"              return 'tk_heap';
"goto"              return 'tk_goto';
"begin"             return 'tk_begin';
"end"               return 'tk_end';
"call"              return 'tk_call';
"if"                return 'tk_if';
"print"             return 'tk_print';

"proc"              return 'tk_proc';
```

Caracteres Especiales: Los caracteres especiales nos ayudaran para agrupar código, terminar alguna línea, asignar datos a variables o estructuras y sobre todo para poder realizar cualquier operación matemática

```
";"                 return 'ptcoma';
"("                 return 'para';
")"                 return 'parc';
"["                 return 'corizq';
"]"                 return 'corder';
":"                 return 'dospts';
","                 return 'coma';
"+"                 return 'mas';
"-"                 return 'menos';
"*"                 return 'por';
"/"                 return 'div';
["]"%i"["]                  return 'pInt';
["]"%d"["]                  return 'pDouble';
["]"%c"["]                  return 'pChar';
"%"                 return 'modulo';
```

```
"<="                return 'menor_igual';
">="                return 'mayor_igual';
"<"                 return 'menor';
">"                 return 'mayor';
"=="                return 'igualacion';
"="                 return 'equal';
"<>"                return 'diferencia';
```

Expresión Regular: Estas expresiones regulares ya fueron descritas anteriormente en la sección de Expresiones Regulares y son conformadas por:

```
l[0-9]+                             return 'etiqueta';
[0-9]+"."[0-9]+\b                   return 'decimal';
[0-9]+\b                            return 'numero';
[a-zA-Z_][_a-zA-Z0-9nÑ]*\b          return 'identificador';
[ \r\t]+            {}
\n                  {}
"#".*                                       // comentario simple línea
[#][*][^*]*[*]+([^#*][^*]*[*]+)*[#]         // comentario multiple líneas
```

Cantidad de símbolos no terminales

En el analizador JISON se usaron **23** No Terminales aproximadamente.

Cada No Terminal devuelve alguna clase que hereda de la clase abstracta nosoAst

Explicación de cada uno de los símbolos no terminales (cuál fue su uso dentro de la gramática) Y funcional describiendo cada una de las acciones

INICIO

El NoTerminal Inicio es donde comienza nuestra gramática, aquí solo nos indica el orden del código 3D y la lista de instrucciones que posee

```
INICIO
    -> DECLARAR_VARIABLES DECLARAR_STACK DECLARAR_HEAP DECLARAR_PUNTEROS DECLARAR_PUNTEROS
       INSTRUCCIONES EOF
;
```

DECLARAR_VARIABLES

Este No Terminal sirve para reconocer la declaración de variables y temporales

```
DECLARAR_VARIABLES->
                    tk_var LISTA_ID ptcoma
;
LISTA_ID-> LISTA_ID coma identificador
        | identificador
;
```

DECLARAR_STACK

Este No Terminal sirve para reconocer la declaración stack

```
DECLARAR_STACK->
            tk_var tk_stack corizq corder ptcoma
;
```

DECLARAR_HEAP

Este No Terminal sirve para reconocer la declaración heap

```
DECLARAR_HEAP->
            tk_var tk_heap corizq corder ptcoma
;
```

DECLARAR_PUNTEROS

Este No Terminal sirve para inicializar los punteros stack y heap

```
DECLARAR_PUNTEROS->
            tk_var identificador equal numero ptcoma
;
```

INSTRUCCIONES

El NoTerminal Instrucciones sirve para generar una lista ascendente de Contenido. En código javascript se va armando el árbol mientras sube

```
INSTRUCCIONES
    -> INSTRUCCIONES    INSTRUCCION
    | INSTRUCCION
;
```

INSTRUCCION

Programa_Sentencia tendra todas las opciones a donde ir; desde una asignacion, creacion de funcion o ejecuion de alguna sentencia de control. En codigo javascript solo va subiendo el arbol que cada uno devuelve

```
INSTRUCCION
    -> ASIGNACION
    | DESTINO_SALTO
    | SALTO_INCONDICIONAL ptcoma
    | SALTO_CONDICIONAL ptcoma
    | DECLARAR_METODO
    | LLAMAR_METODO
    | PRINT
    | ASIGNAR_HEAP
    | ASIGNAR_STACK
;
```

PROGRAMA

El NoTerminal Programa sirve para generar una lista ascendente de Contenido. En código javascript se va armando el árbol mientras sube

```
PROGRAMA->
        PROGRAMA PROGRAMA_SENTENCIA
      | PROGRAMA_SENTENCIA
;
```

PROGRAMA SENTENCIA

Este tiene el mismo funcionamiento de Contenido, con la diferencia de que no posee la producción Declaracion_Funcion y servirá para el cuerpo de funciones y sentenciad de control

```
PROGRAMA_SENTENCIA
                -> ASIGNACION
                | DESTINO_SALTO
                | SALTO_INCONDICIONAL ptcoma
                | SALTO_CONDICIONAL ptcoma
                | LLAMAR_METODO
                | PRINT
                | ASIGNAR_HEAP
                | ASIGNAR_STACK
;
```

ASIGNACION

En la asignación esta sirve para poder cambiar el valor de algún temporal

```
ASIGNACION-> identificador equal VALOR OPERACION VALOR ptcoma
           | identificador equal VALOR ptcoma
           | ASIGNAR_MEMORIA ptcoma
;

ASIGNAR_MEMORIA-> identificador equal tk_stack corizq VALOR corder
                | identificador equal tk_heap corizq VALOR corder
;

VALOR-> decimal
      | menos decimal
      | numero
      | menos numero
      | identificador
;
```

OPERACION

Indicia las operaciones que soy permitida

```
OPERACION-> mas
          | menos
          | por
          | div
          | modulo
          | RELACIONAL
;
RELACIONAL-> menor_igual
           | mayor_igual
           | menor
           | mayor
           | igualacion
           | diferencia
;
```

ASIGNAR_STACK

Sirve para poder reconocer instrucciones en donde se quiera asignar un valor a la memoria stack

```
ASIGNAR_STACK-> tk_stack corizq VALOR corder equal VALOR ptcoma
;
```

ASIGNAR_HEAP

Sirve para poder reconocer instrucciones en donde se quiera asignar un valor a la memoria heap

```
ASIGNAR_HEAP-> tk_heap corizq VALOR corder equal VALOR ptcoma
;
```

DESTINO_SALTO

Sirve para poder reconocer la instrucción de una etiqueta

```
DESTINO_SALTO-> etiqueta dospts
;
```

SALTO_INCONDICIONAL

Este Noterminal sirve para reconocer un salto incondicional, que es solamente un goto

```
SALTO_INCONDICIONAL-> tk_goto etiqueta
;
```

SALTO_CONDICIONAL

Este Noterminal sirve para reconocer un salto condicional, que es un goto y un if que evalúa alguna condición

```
SALTO_CONDICIONAL-> tk_if para VALOR RELACIONAL VALOR parc tk_goto etiqueta
;
```

DECLARAR_METODO

Este Noterminal sirve para reconocer la declaración de un método

```
DECLARAR_METODO-> tk_proc identificador tk_begin PROGRAMA tk_end
;
```

LLAMAR_METODO

Este Noterminal sirve para reconocer una llamada de un método

```
LLAMAR_METODO-> tk_call identificador ptcoma
;
```

PRINT

Este Noterminal sirve para usar el método print, el cual podrá imprimir un carácter o numero

```
PRINT-> tk_print para pInt coma VALOR  parc ptcoma
        | tk_print para pDouble coma VALOR  parc ptcoma
        | tk_print para pChar coma VALOR  parc ptcoma
;
```