Archivo de Gramática

Organización de Lenguajes y Compiladores 1 – Proyecto 2

Aldo Saúl Vásquez Moreira
202109754

TYPEWISE

A continuación, se describirá a detalle el lenguaje utilizando; tanto a nivel léxico como sintáctico.

AREA DE IMPORTACIONES

```
%{
    let Raiz                    =   require("../Entorno/Raiz").Raiz;
    let Tipo                    =   require("../Entorno/Simbolos/Tipo").Tipo;
    let TipoPrimitivo           =   require("../Entorno/Simbolos/TipoPrimitivo").TipoPrimitivo;
    let DeclararVariable        =   require("../Instrucciones/DeclararVariable").DeclararVariable;
    let DeclararFuncion         =   require("../Instrucciones/DeclararFuncion").DeclararFuncion;
    let DeclararArreglo         =   require("../Instrucciones/DeclararArreglo").DeclararArreglo;
    let DeclararLista           =   require("../Instrucciones/DeclararLista").DeclararLista;
    let Asignacion              =   require("../Instrucciones/Asignacion").Asignacion;
    let AsignacionVector        =   require("../Instrucciones/AsignacionVector").AsignacionVector;
    let Ternario                =   require("../Expresiones/Ternario").Ternario;
    let If                      =   require("../Instrucciones/If").If;
    let AccesoVariable          =   require("../Expresiones/AccesoVariable").AccesoVariable;
    let AccesoLista             =   require("../Expresiones/AccesoLista").AccesoLista;
    let AccesoVector            =   require("../Expresiones/AccesoVector").AccesoVector;
    let LlamadaFuncion          =   require("../Expresiones/LlamadaFuncion").LlamadaFuncion;
    let LlamadaPrint            =   require("../Expresiones/LlamadaPrint").LlamadaPrint;
    let OperacionAritmetica     =   require("../Expresiones/OperacionAritmetica").OperacionAritmetica;
    let OperacionLogica         =   require("../Expresiones/OperacionLogica").OperacionLogica;
    let OperacionRelacional     =   require("../Expresiones/OperacionRelacional").OperacionRelacional;
    let FuncionLenguaje         =   require("../Expresiones/FuncionLenguaje").FuncionLenguaje;
    let While                   =   require("../Instrucciones/While").While;
    let ReturnPR                =   require("../Expresiones/ReturnPR").ReturnPR;
    let Valor                   =   require("../Expresiones/Valor").Valor;
    let Incremento              =   require("../Instrucciones/Incremento").Incremento;
    let Decremento              =    require("../Instrucciones/Decremento").Decremento;
    let For                     =    require("../Instrucciones/For").For;
    let DoWhile                 =   require("../Instrucciones/DoWhile").DoWhile;
    let Casteo                  =    require("../Expresiones/Casteo").Casteo;
    let InsertarLista           =    require("../Instrucciones/InsertarLista").InsertarLista;
    let ModificarLista          =    require("../Instrucciones/ModificarLista").ModificarLista;
%}
```

Acá se importan todas las clases que son necesarias para la funcionalidad del lenguaje.

```
%options case-sensitive
```

En la configuración se establece que la gramática será “case-sensitive”, es decir, que no será sensible a las mayúsculas y minúsculas. Por lo tanto “A” = “a”.

AREA DE EXPRESIONES REGULARES

Principalmente se definieron las expresiones regulares de los datos principales del lenguaje, como lo son los números enteros, decimales, etc.

```
digit                   [0-9]
cor1                    "["
cor2                    "]"
esc                     "\\"
int                     (?:[0-9]|[1-9][0-9]+)
exp                     (?:[eE][-+]?[0-9]+)
frac                    (?:\.[0-9]+)
```

Luego se definieron las expresiones regulares de los datos primitivos del lenguaje:

```
([a-zA-ZÑñ]|("_"[a-zA-ZÑñ]))([a-zA-ZÑñ]|[0-9]|"_")*          yytext = yytext.toLowerCase();          return 'id';
\"(?:[{cor1}|{cor2}]|["\\"]["bnrt/["\\"]]|[^"["\\"]])*\"    yytext = yytext.substr(1,yyleng-2);     return 'cadena';
\'(?:{esc}["bfnrt/{esc}]|{esc}"u"[a-fA-F0-9]{4}|[^"{esc}])\'  yytext = yytext.substr(1,yyleng-2);  return 'caracter'
{int}{frac}\b                                                                                    return 'decimal'
{int}\b                                                                                          return 'entero'
```

ÁREA DE SÍMBOLOS

```
"$"                     {return '$';}
"."                     {return '.';}
"++"                    {return '++';}
"--"                    {return '--';}
"+"                     {return '+';}
"-"                     {return '-';}
"*"                     {return '*';}
"/"                     {return '/';}
"^"                     {return '^';}
"%"                     {return '%';}
"("                     {return '(';}
")"                     {return ')';}
"=="                    {return '==';}
"="                     {return '=';}
","                     {return ',';}
":"                     {return ':';}
";"                     {return ';';}
"?"                     {return '?';}
"||"                    {return '||';}
"&&"                    {return '&&';}
"!="                    {return '!=';}
"!"                     {return '!';}
"<="                    {return '<=';}
">="                    {return '>=';}
">"                     {return '>';}
"<"                     {return '<';}
"{"                     {return '{';}
"}"                     {return '}';}
"["                     {return '[';}
"]"                     {return ']';}
. { console.log(`El caracter: "${yytext}" no pertenece al lenguaje`); }
```

Acá se encuentran todos los símbolos o caracteres que el lenguaje acepta. Gracias a la herramienta JISON el análisis léxico se simplifica mucho.

ÁREA DE PALABRAS RESERVADAS

```
"true"                  {   return 'ttrue';     }
"false"                 {   return 'tfalse';    }
"int"                   {   return 'tinteger';  }
"boolean"               {   return 'tboolean';  }
"double"                {   return 'tdouble';   }
"String"                {   return 'tstring';   }
"char"                  {   return 'tchar';   }
"if"                    {   return 'tif';       }
"while"                 {   return 'twhile';    }
"for"                   {   return 'tfor';    }
"else"                  {   return 'telse';     }
"void"                  {   return 'tvoid';     }
"return"                {   return 'treturn';   }
"new"                   {   return 'tnew';     }
"do"                    {   return 'tdo';     }
"list"                  {   return 'tlist';     }
"add"                   {   return 'tadd';     }
"switch"                {   return 'tswitch';     }
"case"                 {   return 'tcase';     }
"default"                 {   return 'tdefault';     }
"toLower"               {   return 'ttoLower';     }
"toUpper"               {   return 'ttoUpper';     }
"truncate"                {   return 'ttruncate';     }
"round"                {   return 'tround';     }
"length"                {   return 'tlength';     }
"typeOf"                {   return 'ttypeOf';     }
"toString"                {   return 'ttoString';     }
"toCharArray"                 {   return 'ttoCharArray';     }
"main"                  {   return 'tmain';     }
"print"                   {   return 'tPrint';    }
```

Acá se encuentran todas las palabras que están predefinidas dentro del lenguaje. Es decir, que al encontrar una de estas palabras durante el análisis léxico estás indican que una instrucción o acción especial continúa.

ÁREA DE PRECEDENCIA

```
/*Operaciones logicas*/
%left '||'
%left '&&'
%left '?'
%left ':'
%left '++' '--'
%left '!=' '==' '==='
%left '>' '<' '<=' '>='

/*Operaciones numericas*/
%left '+' '-'
%left '*' '/' '%'
%right '^'
%right negativo '!' '('
```

Acá se determina la precedencia de cada símbolo. Es decir, para realizar operaciones como lo son las aritméticas existen ciertas reglas para obtener el resultado correcto, entonces, podemos suponer que las precedencias determinan las reglas que se deben seguir para el buen comprendimiento de las entradas.

ANÁLISIS SINTÁCTICO

```
INICIO
    : SENTENCIAS EOF
    {
        console.log("Parse de Jison entrada: OK ");
        let raiz = new Raiz($1);
        $$ = raiz;
        return raiz;
    }
    ;
```

La gramática cuenta con una producción llamada INICIO la cual indica que un nuevo análisis ha comenzado. Según la gramática pueden venir sentencias o bien, podemos toparnos con un archivo vacío.

```
SENTENCIAS :    SENTENCIAS SENTENCIA
                {
                    $1.push($2);
                    $$ = $1;
                }
                |   SENTENCIA
                {
                    let lstsent = [];
                    lstsent.push($1);
                    $$ = lstsent;
                }
;
```

Asimismo, podemos observar que la producción SENTENCIAS es recursiva por la izquierda (LR) y que puede contener una o más sentencias.

```
SENTENCIA :     DECLARACION ';'              { $$ = $1; }
            |   FUNCION                      { $$ = $1; }
            |   LISTA_ADD                    { $$ = $1; }
            |   LISTA_MODIFICAR  ';'         { $$ = $1; }
            |   ASIGNACION       ';'           { $$ = $1; }
            |   VECTOR_ADD                   { $$ = $1; }
            |   IF                           { $$ = $1; }
            |   LLAMADA_FUNCION  ';'         { $$ = $1; }
            |   WHILE                        { $$ = $1; }
            |   FOR                          { $$ = $1; }
            |   DO_WHILE                     { $$ = $1; }
            |   INCREMENTO       ';'         { $$ = $1; }
            |   DECREMENTO       ';'         { $$ = $1; }
            |   PRINT       ';'              { $$ = $1; }
            |   MAIN ';'                     { $$ = $1; }
            |   RETURN                     { $$ = $1; }
            |   error ';'
            |   error '}'
;
```

Y bien, una sentencia es básicamente una instrucción dentro del lenguaje. Estas son el corazón del lenguaje, ya que sin ellas no se ejecutaría ninguna instrucción y por lo tanto, no habría funcionalidad.

A continuación se mostrará cada una de las producciones creadas y un ejemplo del tipo de entrada que permiten:

MAIN: Inicia la ejecución del programa. Recibe la palabra reservada “main” y ejecuta una LLAMADA_FUNCION.

```
MAIN : tmain LLAMADA_FUNCION    { $$ = $2; }
;
```

```
1 main nombreFuncion();
```

PRINT: Imprime en pantalla cualquier tipo de expresión. Recibe la palabra reservada “print” seguido de un paréntesis que abre, una lista de expresiones y un paréntesis que cierra.

```
PRINT : tPrint '(' LISTA_EXP ')' { $$ = new LlamadaPrint($1, $3, @1.first_line, @1.first_column);    }
;
```

```
1 print(1+1);
```

INCREMENTO Y DECREMENTO: Aumenta o disminuye en uno el valor de una variable. Obtiene primero el id de la variable y luego “++” o “--", dependiendo de lo que requiera el usuario.

```
220 INCREMENTO : id '++'
221             {
222                 $$ = new Incremento($1, @1.first_line, @1.first_column)
223             }
224 ;
225
226 DECREMENTO : id '--'
227             {
228                 $$ = new Decremento($1, @1.first_line, @1.first_column)
229             }
230 ;
```

DECLARACIÓN: Inicializa una variable de cualquier tipo. Además de listas y vectores con los distintos tipos de declaración que poseen.

```
DECLARACION : TIPO  id  '=' EXP
            {
                $$ = new DeclararVariable($1, $2, $4, @2.first_line, @2.first_column);
            }
            | TIPO  id
            {
                $$ = new DeclararVariable($1, $2, undefined, @2.first_line, @2.first_column);
            }
            | TIPO '[' ']' id '=' tnew TIPO '[' entero ']'
            {
                $$ = new DeclararArreglo($1, $4, $7,undefined, $9, @2.first_line, @2.first_column);
            }
            | TIPO '[' ']' id '=' '{' LISTA_EXP '}'
            {
                $$ = new DeclararArreglo($1, $4,undefined, $7,undefined, @2.first_line, @2.first_column);
            }
            | tlist '<' TIPO '>' id '=' tnew tlist '<' TIPO '>'
            {
                $$ = new DeclararLista($3, $5, $10,undefined, @2.first_line, @2.first_column);
            }
            | tlist '<' TIPO '>' id '=' EXP
            {
                $$ = new DeclararLista($3, $5, undefined, $7,  @2.first_line, @2.first_column);
            }
;
```

ASIGNACIÓN: Permite asignar o reasignar el valor que contiene una variable. Recibe el id de la variable, el "=" y una expresión que indica lo que podría ser.

```
ASIGNACION  :    id '=' EXP
            {
                $$ = new Asignacion($1, $3, @1.first_line, @1.first_column);
            }
;
```

LISTA/VECTOR ADD: Permiten agregar un valor a una lista o vector.

```
VECTOR_ADD  :   id '[' entero ']' '=' EXP ';'
            {
                $$ = new AsignacionVector($1, $6, $3,@1.first_line, @1.first_column);
            }
;

LISTA_ADD: id '.' tadd '(' EXP ')' ';'
            {
                $$ = new InsertarLista($1, $5, @1.first_line, @1.first_column);
            }
;
```

TERNARIA: Recibe un operador ternario. Recibe tres expresiones, una condición, una verdadera y la otra falsa.

```
TERNARIA: EXP '?' EXP ':' EXP
        {
            $$ = new Ternario($1, $3, $5, @1.first_line, @1.first_column);
        }
;
```

CASTEO: Permite cambiar el tipo de variable de un dato recibido. Recibe paréntesis, el tipo al que desea ser casteado y una expresión.

```
282   CASTEO: '(' TIPO ')' EXP
283       {
284           $$ = new Casteo($2, $4, @1.first_line, @1.first_column);
285       }
286   ;
```

IF/ELSE: Permite cumplir con la funcionalidad de un if dentro de cualquier lenguaje.

```
294   IF      :   tif '(' EXP ')' BLOQUE_SENTENCAS
295           {
296               $$ = new If($3, $5, [], @1.first_line, @1.first_column);
297           }
298           |   tif '(' EXP ')' BLOQUE_SENTENCAS ELSE
299           {
300               $$ = new If($3, $5, $6, @1.first_line, @1.first_column);
301           }
302   ;
303
304   ELSE    :   telse IF
305           {
306               let else_sent = [];
307               else_sent.push($2);
308               $$ = else_sent;
309           }
310           |   telse BLOQUE_SENTENCAS
311           {
312               $$ = $2;
313           }
314   ;
315
```

CICLOS: Permiten ejecutarse varias veces hasta que se cumpla cierta condición.

```
DO_WHILE : tdo BLOQUE_SENTENCAS twhile '(' EXP ')' ';'
        {
            $$ = new DoWhile($2, $5, @1.first_line, @1.first_column );
        }
;

FOR     : tfor '(' DECLARACION ';' EXP ';' ACTUALIZACION_FOR ')' BLOQUE_SENTENCAS
        {
            $$ = new For($3, $5, $7, $9, @1.first_line, @1.first_column );
        }
        | tfor '(' ASIGNACION ';' EXP ';' ACTUALIZACION_FOR ')' BLOQUE_SENTENCAS
        {
            $$ = new For($3, $5, $7, $9, @1.first_line, @1.first_column );
        }
;

WHILE   : twhile '(' EXP ')' BLOQUE_SENTENCAS
        {
            $$ = new While($3, $5, @1.first_line, @1.first_column );
        }
;
```

FUNCIÓN: Contiene los diversos tipos de declaración de una función dentro del lenguaje.

```
FUNCION:        TIPO    id '(' LISTA_PARAM ')' BLOQUE_SENTENCAS
            {
                $$ = new DeclararFuncion($1, $2, $4, $6, @2.first_line, @2.first_column);
            }
            |   tvoid   id '(' LISTA_PARAM ')' BLOQUE_SENTENCAS
            {
                $$ = new DeclararFuncion(new Tipo(TipoPrimitivo.Void), $2, $4, $6, @2.first_line, @2.first_column);
            }
            |   TIPO    id '('  ')' BLOQUE_SENTENCAS
            {
                $$ = new DeclararFuncion($1, $2, [], $5, @2.first_line, @2.first_column);
            }
            |   tvoid   id '('  ')' BLOQUE_SENTENCAS
            {
                $$ = new DeclararFuncion(new Tipo(TipoPrimitivo.Void), $2, [], $5, @2.first_line, @2.first_column);
            }
```

TIPOS: Hace referencia a los tipos primitivos.

```

TIPO    :       tinteger                { $$ = new Tipo(TipoPrimitivo.Integer); }
        |       tboolean                { $$ = new Tipo(TipoPrimitivo.Boolean); }
        |       tstring                 { $$ = new Tipo(TipoPrimitivo.String);  }
        |       tdouble                 { $$ = new Tipo(TipoPrimitivo.Double);  }
        |       tchar                   { $$ = new Tipo(TipoPrimitivo.Char);  }
;
```

FUNCIONES DE LENGUAJE: Realizan el llamado de una función especial del lenguaje.

```
FUNCIONES_LENGUAJE
    : ttoLower '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
    | ttoUpper '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
    | ttruncate '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
    | tround '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
    | ttoCharArray '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
    | ttoString '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
    | ttypeOf '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
    | tlength '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
;
```

EXP: Contiene todas las expresiones permitidas dentro del lenguaje.

```
EXP :   EXP '+' EXP                 { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '-' EXP                 { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '*' EXP                 { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '/' EXP                 { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '^' EXP                 { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '%' EXP                 { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
    |   '-' EXP %prec negativo      { $$ = new OperacionAritmetica($2, "negativo", $2, @2.first_line, @2.first_column);}
    |   '(' EXP ')'                 { $$ = $2;}
    |   EXP '==' EXP                { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '!=' EXP                { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '<'  EXP                { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '>'  EXP                { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '<=' EXP                { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '>=' EXP                { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '&&' EXP                { $$ = new OperacionLogica($1, $2, $3, @2.first_line, @2.first_column);}
    |   EXP '||' EXP                { $$ = new OperacionLogica($1, $2, $3, @2.first_line, @2.first_column);}
    |   id                          { $$ = new AccesoVariable($1, @1.first_line, @1.first_column);        }
    |   id '[' EXP ']'              { $$ = new AccesoVector($1, $3,@1.first_line, @1.first_column);        }
    |   id '[' '[' EXP ']']'        { $$ = new AccesoLista($1,$4,@1.first_line, @1.first_column);        }
    |   LLAMADA_FUNCION             { $$ = $1;}
    |   entero                      { $$ = new Valor($1, "integer", @1.first_line, @1.first_column);}
    |   decimal                     { $$ = new Valor($1, "double", @1.first_line, @1.first_column); }
    |   caracter                    { $$ = new Valor($1, "char", @1.first_line, @1.first_column);   }
    |   cadena                      { $$ = new Valor($1, "string", @1.first_line, @1.first_column); }
    |   ttrue                       { $$ = new Valor($1, "true", @1.first_line, @1.first_column);   }
    |   tfalse                      { $$ = new Valor($1, "false", @1.first_line, @1.first_column);  }
    |   TERNARIA                    { $$ = $1;}
    |   CASTEO                      { $$ = $1;}
    |   FUNCIONES_LENGUAJE          { $$ = $1;}
;
```