Revista CIENCIA Y TECNOLOGÍA V.17 N.1
ISSN 1810-6781 Rev. Cienc. Tecnol. 17(1): 85-96, (2021)

Desarrollo e implementación de un analizador sintáctico utilizando el compilador Javacc para el reconocimiento de errores sintácticos en el lenguaje PHP

Development and implementation of a parser using the Javacc compiler for recognition of syntax errors in the PHP language

Patricia Gissela Pereyra Salvador*; Richard Alexander Rosario Sánchez

Facultad de Ciencias Físicas y Matemáticas, Universidad Nacional de Trujillo, Av. Juan Pablo II - s/n - Ciudad Universitaria, Trujillo, Perú

* Autor correspondiente: patty_pereyra31@hotmail.com (P. Pereyra)
DOI: 10.17268/rev.cyt.2021.01.08

RESUMEN

Se implementó un reconocedor de errores sintácticos mediante el desarrollo de un analizador sintáctico para el lenguaje php. Para la realización del trabajo, se tomó la ayuda de tecnologías como JavaCC, HTML, CSS, PHP, XAMPP y Visual Studio Code. Se presenta un programa (AnalizadorLexSin.jj), compuesto por cinco partes, las cuales son: Área de opciones, cláusulas PARSER_BEGIN y PARSER_END, tokens bajo la cláusula SKIP, tokens bajo la cláusula TOKEN y área de funciones BNF. Luego de desarrollar el programa se llevó a cabo su compilación con la ayuda del compilador JavaCC. El código a analizar se ingresó por el usuario a través de una interfaz mostrada en un navegador web, para realizar el proceso de análisis sintáctico. Una vez ingresado el código, éste se guardó en un archivo txt, el cual fue el archivo que se verificó y comprobó si cumple con las reglas sintácticas del lenguaje php, escritas previamente en el programa AnalizadorLexSin.jj. De esta manera, se comprobó si el código presenta errores sintácticos o no. Llegando a reducir la experiencia de encontrarse con errores comunes que se presentan al momento de programar en el lenguaje PHP.

Palabras clave: analizador sintáctico; JavaCC; lenguaje PHP; reconocimiento de errores sintácticos.

ABSTRACT

A syntax error recognizer has been implemented through the development of a syntactic parser for the php language. To carry out this work, use the help of technologies such as JavaCC, HTML, CSS, PHP, XAMPP and Visual Studio Code. The project presents a program (AnalyzerLexSin.jj), consisting of five parts, which are: Options area, PARSER_BEGIN and PARSER_END clauses, tokens under the SKIP clause, tokens under the TOKEN clause and BNF function area. After developing the program, its compilation was carried out with the help of the JavaCC compiler. The analysis code was entered by the user through an interface shown in a web browser, to perform the syntactic analysis process. Once the code was entered, it was saved in a txt file, which was the file that was verified and checked if it complies with the php language syntactic rules, previously written in the AnalyzerLexSin.jj program. In this way the student checked if his code presents syntactic errors or not. Getting to reduce the experience of encountering common errors that occur when programming in the PHP language.

Keywords: syntactic analyzer; JavaCC; PHP language; recognition of syntactic errors.

1. INTRODUCCIÓN

A lo largo de los años se desarrollaron innumerables lenguajes de programación, empezando por lenguajes que se programaban en código binario, cuyas instrucciones ejercían un control directo sobre el hardware (Correa, 2015), pasando por el lenguaje ensamblador, a principios de los 50, siendo aún complicado y tedioso para el programador. Posteriormente a fines de la misma década, aparecieron los lenguajes de alto nivel, llegando a solucionar problemas de una manera fácil y rápida, no obstante, aquellos lenguajes necesitan ser traducidos.

Los lenguajes de programación de alto nivel evitarían que el programador se involucre con el mundo físico del hardware. Aquellos lenguajes aparecieron junto con programas (compilador o intérprete) que les permitía pasar

Autor Responsable: Patricia Gissela Pereyra Salvador Fecha de envío: 03-09-2019 Fecha de aprobación: 17-02-2020

de la aproximación del lenguaje natural a lenguaje máquina.

El estudiante que recientemente se involucra con el lenguaje de programación php, se enfrenta a nuevos conceptos y sintaxis que presenta tal lenguaje. Por tal motivo, demandará tiempo y dedicación aprenderlos, tal es el caso que, aún dominando el lenguaje, se encontrará con problemas del día a día del programador, como errores sintácticos en el código.

Aun así, programando en un lenguaje que se aproxima al lenguaje natural, hoy en día, se presentan errores que el programador experimenta al momento de codificar, errores que son casi imposible de evitar (Traver, 2010). Uno de los errores más comunes al momento de programar, son los errores sintácticos, es por ello que en la presente se tomará como tema principal dicho error.

El compilador es un tipo de traductor que presenta la característica de transformar el código fuente a lenguaje máquina (Ibañez et al., 2015). Generando un programa ejecutable que puede ser almacenada o reutilizada, mientras que el intérprete es un procesador que realiza la operación de conversión ejecutando paso a paso dicho código fuente.

Los errores sintácticos ocurren cuando el programador escribe código que no va de acuerdo a las reglas de escritura del lenguaje de programación, cuyos errores son detectados por el analizador sintáctico, el cual su función es la verificación que el programa fuente cumpla con la sintaxis que el lenguaje fuente tiene definido para sus programas (Vanegas, 2005). Recibiendo los tokens producidos por el scanner (analizador léxico) para verificar que estén correctamente combinados de tal manera que cumplan con las reglas sintácticas del lenguaje fuente que se representan a través de una gramática. El análisis de la cadena de tokens se realiza mediante la gramática, y se representa mediante un árbol de sintaxis, el cual constituye la salida de la fase del análisis sintáctico.

Por lo expuesto, el objetivo general en este proyecto es la implementación de un reconocedor de errores sintácticos mediante el desarrollo de un analizador sintáctico para el lenguaje php.

De esta manera; el estudiante, futuro programador en php, pueda comprobar que las sentencias que componen su código fuente son las correctas para el lenguaje php, ayudando a disminuir los errores comunes que se presentan al momento de programar en dicho lenguaje, a la vez aumentando su grado de satisfacción al programar en el nuevo lenguaje y por consiguiente disminuir el tiempo que empleará al momento de tratar de encontrar el error sintáctico.

2. MATERIALES Y MÉTODOS

Para la realización del reconocedor de errores sintácticos, el proyecto toma la ayuda de diferentes tecnologías.

JavaCC

Es un generador de analizador sintáctico para el lenguaje de programación Java, generando un parser para una gramática presentada en notación BNF, tomando como salida un código escrito en Java (Norvell, 2011). Admite el uso de estados léxicos y la capacidad de agregar acciones léxicas incluyendo un bloque de código Java tras el identificador de un token. Se trata de una herramienta que facilita la construcción de analizadores léxicos y sintácticos por el método de las funciones recursivas (Gálvez & Mora, 2005). De esta manera, los analizadores generados utilizan la técnica descendente a la hora de obtener el árbol sintáctico. Entre los generadores de analizadores sintácticos descendentes, JavaCC es uno de los que poseen mejor gestión de errores. Los analizadores generados por JavaCC son capaces de localizar exactamente la ubicación de los errores, proporcionando información diagnóstica completa.

HTML

HyperText Markup Language, Es un lenguaje de marcado de hipertexto, para la elaboración de páginas web.

CSS

Cascading Style Sheets, es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.

PHP

PHP es el acrónimo de Hipertext Preprocesor. Es un lenguaje interpretado para programar scripts del lado del servidor. Php permite la creación de páginas web dinámicas que se incrustan dentro del código HTML (Como se cita en Sierra & Espinoza, 2018). Así mismo permite trabajar con bases de datos.

XAMPP

Cuando se trabaja en php, el primer requisito es contar con un servidor, para ello, el proyecto usará xampp. siendo un servidor independiente de plataforma, un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl.

Visual Studio Code

Es un editor de código fuente desarrollado por Microsoft, en el cual se llevará a cabo todo el trabajo. En dicho editor, se escribirá los tokens necesarios, junto con la sintaxis a la cual pertenece el lenguaje php, siendo un editor de código fuertemente centrado en el desarrollo de sitios web, soportando una gran cantidad de lenguajes de programación, como lo son, C#, Visual Basic, PHP, Phyton, Java, HTML, CSS, JavaScript entre otros (Medina, 2015).

Desarrollo

El proyecto presenta un programa con la extensión .jj, como todo programa JavaCC, suele almacenarse en ficheros con extensión. jj (Gálvez & Mora, 2005). Tal programa producirá los siguientes ficheros de salida: AnalizadorLexSin.java, (es el archivo de analizador sintáctico), AnalizadorLexSinTokenManager.java (es el analizador lexicográfico), AnalizadorLexSinConstants.java (interface que asocia un código a cada token). Además, creará las clases necesarias como: ParseException.java, SimpleCharStream.java, Token.java, TokenMgrError.java. (Gálvez & Mora, 2005). Dicho programa presenta una estructura formada por cinco partes: un área de **opciones**, cláusulas **PARSER_BEGIN y PARSER_END**, tokens bajo la cláusula **SKIP**, tokens bajo la cláusula **TOKEN** y finalmente el **área de funciones BNF**, que será convertido por JavaCC, en funciones Java.

Parte 1: Área de opciones.

En la sección opcions, se hará uso de la opción lookeadhead inicializándolo en 5. JavaCC crea analizadores de descendencia recursivos, el cual observa al siguiente símbolo para decidir qué regla elegir, si el lookahead se iguala a uno, por defecto, mira solo al siguiente símbolo (Gálvez & Mora, 2005), sin embargo, en el proyecto se iguala a cinco para así pueda buscar los siguientes cinco símbolos y decidir que gramática elegir, como se muestra en la imagen:

```
options{
    LOOKAHEAD = 5;
}
```

Figura 1. Área de opciones.

Parte 2: Cláusulas PARSER_BEGIN y PARSER_END.

En esta sección se le indica a JavaCC el nombre de la clase principal y está delimitada por dos palabras reservadas, Parser_Begin y Parser_End acompañada por el nombre del proyecto entre las palabras delimitadoras se encuentra código en java (Gálvez & Mora, 2005), compuesta por una clase llamada con el mismo nombre de la especificación (AnalizadorLexSin), la clase presenta un atributo de tipo entero y estático, llamada numeroDeErrores, inicializada en cero, dicho atributo ayudará a contar la cantidad de errores del código a analizar.

```
options{
    LOOKAHEAD = 5;
}

PARSER_BEGIN(AnalizadorLexSin)
class AnalizadorLexSin {
    public static int numeroDeErrores = 0;

    public static void main(String[] args) throws ParseException {
        AnalizadorLexSin analizador = new AnalizadorLexSin(System.in);
        analizador.Programa();
        System.out.println("CANTIDAD DE ERRORES: " + numeroDeErrores);
        System.out.println("EL ANALISIS HA TERMINADO.");
    }
}
PARSER_END(AnalizadorLexSin)
```

Figura 2. Cláusulas PARSER_BEGIN y PARSER_END.

A la vez se cuenta con el método principal (main) que dará inicio a la ejecución del programa. Se crea el objeto analizador para aplicar al método Programa(), siendo el método asociado al símbolo inicial de la gramática sintáctica, terminando el análisis, imprimirá por pantalla la cantidad de errores, y un aviso de análisis terminado, dando por finalizado el análisis sintáctico.

Parte 3: tokens bajo la cláusula SKIP.

Son aquellos que serán consumidos sin ser pasados al analizador sintáctico (Gálvez & Mora, 2005). Por ejemplo, los espacios en blando, saltos de línea, los tabuladores y retorno de carro.

```
SKIP:
{
    " " | "\n" | "\r" | "\r\n" | "\t"
}
```

Figura 3. Tokens bajo la cláusula SKIP.

Parte 4: tokens bajo la cláusula TOKEN.

La declaración de cada token se agrupa entre paréntesis angulares y está formada por el nombre del token seguido por el patrón asociado y separado de éste por dos puntos (Gálvez & Mora, 2005). Cada token describe a los caracteres que usa el lenguaje php.

A continuación, se dará una breve descripción de los tokens que presenta el proyecto.

En la figura 4, se muestran los tokens para signos de puntuación, por ejemplo, el dólar, el punto y coma, el punto, la coma, dos puntos, etc.

```
TOKEN: {
    <DOLAR : "$">   {System.out.println("DOLAR -> " + image + "\r\n");}
    |<PUNTOYCOMA : ";"> {System.out.println("PUNTO Y COMA ->" + image + "\r\n");}
    |<PUNTO : "."> {System.out.println("PUNTO ->" + image + "\r\n");}
    |<COMA : ","> {System.out.println("COMA ->" + image + "\r\n");}
    |<DOSPUNTOS : ":"> {System.out.println("DOSPUNTOS ->" + image + "\r\n");}
    |<PARENIZQ : "("> {System.out.println("PARENIZQ ->" + image + "\r\n");}
    |<PARENDER : ")"> {System.out.println("PARENDER ->" + image + "\r\n");}
    |<LLAVEIZQ : "{"> {System.out.println("LLAVEIZQ ->" + image + "\r\n");}
    |<LLAVEDER : "}"> {System.out.println("LLAVEDER ->" + image + "\r\n");}
    |<CORCHIZQ : "["> {System.out.println("CORCHIZQ ->" + image + "\r\n");}
    |<CORCHDER : "]"> {System.out.println("CORCHDER ->" + image + "\r\n");}
    |<AMPERSAND : "&"> {System.out.println("AMPERSAND ->" + image + "\r\n");}
}
```

Figura 4. Token para signos de puntuación.

En la figura 5, se muestra los tokens para operadores, como es el caso del operador de asignación, aritmético, combinado, comparación, etc.

```
TOKEN: {
    <OPERASIGNACION : "="> {System.out.println("OPERASIGNACION ->" + image + "\r\n");}
    |<OPERARITMETICO : ("+") | ("-") | ("*") | ("/") | ("%") | ("**")>
                        {System.out.println("OPERARITMETICO ->" + image + "\r\n");}
    |<OPERCOMBINADO_ART : ("+=") | ("-=") | ("*=") | ("/=") | ("%=") | ("**=")>
                       {System.out.println("OPERCOMBINADO_ART ->" + image + "\r\n");}
    |<OPERCOMBINADO_CAD : ".="> {System.out.println("OPERCOMBINADO_CAD ->" + image + "\r\n");}
    |<OPERCOMPARACION : ("==") | ("===") | ("!=") | ("<>") | ("!==") | ("<") | (">") | ("<=")
                     | (">=") | ("<=>")> {System.out.println("OPERCOMPARACION ->" + image + "\r\n");}
    |<OPERINCDEC : ("++") | ("--")> {System.out.println("OPERINCDEC ->" + image + "\r\n");}
    |<OPERLOGICO : ("and") | ("or") | ("xor") | ("&&") | ("||")>
                                {System.out.println("OPERLOGICO ->" + image + "\r\n");}
    |<OPERLOGNOT : "!"> {System.out.println("OPERLOGNOT ->" + image + "\r\n");}
    |<OPERASIG_ARRAY : "=>"> {System.out.println("OPERASIG_ARRAY ->" + image + "\r\n");}
}
```

Figura 5. Token para operadores.

En la figura 6, se muestran los tokens para estructuras de control, como es el caso del if, else, do, while, endwhile, etc.

```
TOKEN:{
    <IF : "if"> {System.out.println("IF ->" + image + "\r\n");}
    | <ELSE : "else"> {System.out.println("ELSE ->" + image + "\r\n");}
    | <DO : "do"> {System.out.println("DO ->" + image + "\r\n");}
    | <WHILE : "while"> {System.out.println("WHILE ->" + image + "\r\n");}
    | <ENDWHILE : "endwhile"> {System.out.println("ENDWHILE ->" + image + "\r\n");}
    | <FOR : "for"> {System.out.println("FOR ->" + image + "\r\n");}
    | <SWITCH : "switch"> {System.out.println("SWITCH ->" + image + "\r\n");}
    | <CASE : "case"> {System.out.println("CASE ->" + image + "\r\n");}
    | <ENDSWITCH : "endwswitch"> {System.out.println("ENDSWITCH ->" + image + "\r\n");}
    | <BREAK : "break"> {System.out.println("BREAK ->" + image + "\r\n");}
    | <CONTINUE : "continue"> {System.out.println("CONTINUE ->" + image + "\r\n");}
    | <DEFAUL : "default"> {System.out.println("DEFAULT ->" + image + "\r\n");}
}
```

Figura 6. Token para estructuras de control.

En la figura 7, se muestran los tokens para palabras reservadas, como es el caso de inicio, fin, echo, global, stactic, etc.

```
TOKEN:{
    <INICIO : "<?php"> {System.out.println("INICIO ->" + image + "\r\n");}
    | <FIN : "?>"> {System.out.println("FIN ->" + image + "\r\n");}
    | <ECHO : "echo"> {System.out.println("ECHO ->" + image + "\r\n");}
    | <GLOBAL : "global"> {System.out.println("GLOBAL ->" + image + "\r\n");}
    | <STATIC : "static"> {System.out.println("STATIC ->" + image + "\r\n");}
    | <CONST : "const"> {System.out.println("CONST ->" + image + "\r\n");}
    | <PRINT : "print"> {System.out.println("PRINT ->" + image + "\r\n");}
    | <FUNCTION : "function"> {System.out.println("FUNCTION ->" + image + "\r\n");}
    | <RETURN : "return"> {System.out.println("RETURN ->" + image + "\r\n");}
    | <NULL: "null"> {System.out.println("NULL ->" + image + "\r\n");}
    //CLASES Y OBJETOS
    | <CLASS: "class"> {System.out.println("CLASS ->" + image + "\r\n");}
    | <PUBLIC: "public"> {System.out.println("PUBLIC ->" + image + "\r\n");}
    | <PROTECTED: "protected"> {System.out.println("PROTECTED ->" + image + "\r\n");}
    | <PRIVATE: "private"> {System.out.println("PRIVATE ->" + image + "\r\n");}
    | <NEW: "new"> {System.out.println("NEW ->" + image + "\r\n");}
    | <EXTENDS: "extends"> {System.out.println("EXTENDS ->" + image + "\r\n");}
}
```

Figura 7. Token para palabras reservadas.

En la figura 8, se muestran los tokens especiales, como es el caso de cadena, entero, real e identificador.

```
TOKEN:{
    <CADENA : "\""(~["\""])*"\""> {System.out.println("CADENA ->" + image + "\r\n");}
    | <ENTERO : (["0"-"9"])+> {System.out.println("ENTERO ->" + image + "\r\n");}
    | <REAL : (["0"-"9"])+ "." (["0"-"9"])+> {System.out.println("REAL ->" + image + "\r\n");}
    | <IDENTIFICADOR : ["a"-"z","A"-"Z"](["a"-"z","A"-"Z","0"-"9","_"])*>
                        {System.out.println("IDENTIFICADOR ->" + image + "\r\n");}
}
```

Figura 8. Tokens especiales.

Parte 5: Área de funciones BNF

Como ya se ha comentado, JavaCC genera un analizador sintáctico descendente implementado a base de funciones recursivas, de manera que cada no terminal de la gramática será convertido una función diferente, cuya implementación será generada por JavaCC.

En la creación de reglas en BNF, se define la sintaxis del lenguaje php, cada regla contiene la estructura del lenguaje php. Las expresiones BNF puede hacer uso de los siguientes componentes: **<>:** permiten hacer referencia a un terminal declarado en el área de tokens. Ej.: <NUMERO>. **“”:** permiten expresar tokens anónimos, entrecomillando un literal de tipo cadena de caracteres. ***:** indica repetición 0 o más veces de lo que le precede entre paréntesis. **+:** indica repetición 1 o más veces de lo que le precede entre paréntesis. **?:** indica que lo que le precede entre paréntesis es opcional. **[]**: tiene igual significado que el ?, de manera que (patrón)? es equivalente a [patrón]. **|:** indica opcionalidad. Ej.: (“texto1” | “texto2”) encaja con “texto1” y con “texto2”, (Gálvez & Mora, 2005).

A continuación, se dará una breve descripción de las reglas que presenta el proyecto, cabe mencionar que dichas reglas serán convertidas a métodos por el compilador JavaCC.

Programa(): método que da inicio al análisis sintáctico, reconociendo al token <INICIO> y a la vez llamando al método Bloque().

Bloque(): método que contiene try catch permitiendo encontrar el error durante el análisis, a la vez acumula la cantidad de errores que se encontró y muestra como mensaje “Error sintáctico”.

ExpresionesGlobales(): método que identifica diferentes expresiones que se encuentra en un bloque de código php conteniendo métodos como Variable(), FuncionAnonima(), LlamadaFuncion(), Constante().

BloqueLocal(): método que se mandará a llamar siempre y cuando esté dentro de la sentencia **if** o **else.**

Variable(): método que se encarga en identificar una variable en php, presentado por un token <DOLAR> y el nombre de la variable con el token <IDENTIFICADOR>.

TipoVariable(): método que identifica el tipo de la variable, el cual puede ser, <GLOBAL>, <STATIC>, <PUBLIC>, <PROTECTED> o <PRIVATE>.

Constante(): método que identifica una constante, presentado por un token <CONST> y el nombre de la constante con el token <IDENTIFICADOR>.

Sentencias(): presenta un grupo de métodos las cuales son, SentenciaIf(), SentenciaWhile(), SentenciaDoWhile(), SentenciaFor(), SentenciaSwitch().

Los métodos **SentenciaIf(), SentenciaElse(), SentenciaWhile(), SentenciaDoWhile(), SentenciaFor(), SentenciaSwitch() y BloqueCase(),** serán los métodos que identificaran a la estructura del lenguaje php, de acuerdo al nombre en que han sido declarados.

Comparacion(): método que será llamado cuando se necesite realizar una comparación en el código, por ejemplo, cuando se use la estructura **if**, **while**, **do while** o **for**, que necesitarán realizar por lo menos una comparación.

Funciones(): identifica funciones a usar en el código php, devolviendo o no valores.

FuncionAnonima(): método que no necesita ser identificada, es decir, no llevará un nombre.

Argumentos(): método que identifica los argumentos que se les pasa a las funciones.

DevolverValores(): método que se invocará cuando se requiera devolver valores, ya sea variables, valores de tipo entero o real.

LlamadaFuncion(): método que permite invocar a una función, junto con sus argumentos en caso los tenga.

PasarArgumentos(): método que contiene a los argumentos que serán llevados a la función quien los invocó, pueden ser, valores enteros, valores reales, cadena de caracteres o variables.

Clase(): método que permite la creación de clases, tomando la estructura de Java, contando así solo con la presencia de atributos y métodos dentro de cada clase y restringiendo el método de bloque local.

Atributos(): permite la creación de atributos para la clase, conteniendo el método TipoVariable() y Variable().

Metodos(): permite la creación de métodos, mandando a llamar a la sentencia Funciones().

Esta área toma la siguiente estructura:

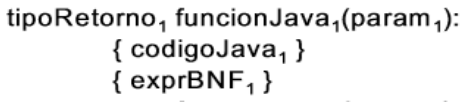

Figura 9: Estructura del área de funciones BNF.

Fuente: Java a tope: Traductores y compiladores con Lex/Yacc, JFlex/Cup y JavaCC.

A modo de ejemplo se observa la estructura de la sentencia switch para el lenguaje PHP.

```
void SentenciaSwitch():{}{
    <SWITCH> <PARENIZQ> Variable() <PARENDER> (<LLAVEIZQ> BloqueCase() <LLAVEDER>
        | <DOSPUNTOS> BloqueCase() <ENDSWITCH> <PUNTOYCOMA>)
}
```

Figura 10: Estructura de la sentencia switch.

A continuación, se presenta la estructura formada por todos los métodos que fueron utilizados para desarrollar e implementar el analizador sintáctico. Se puede observar que algunos métodos se dividen en otros métodos, tomando la idea de invocación de métodos, cada método presenta un color único para un mayor entendimiento.

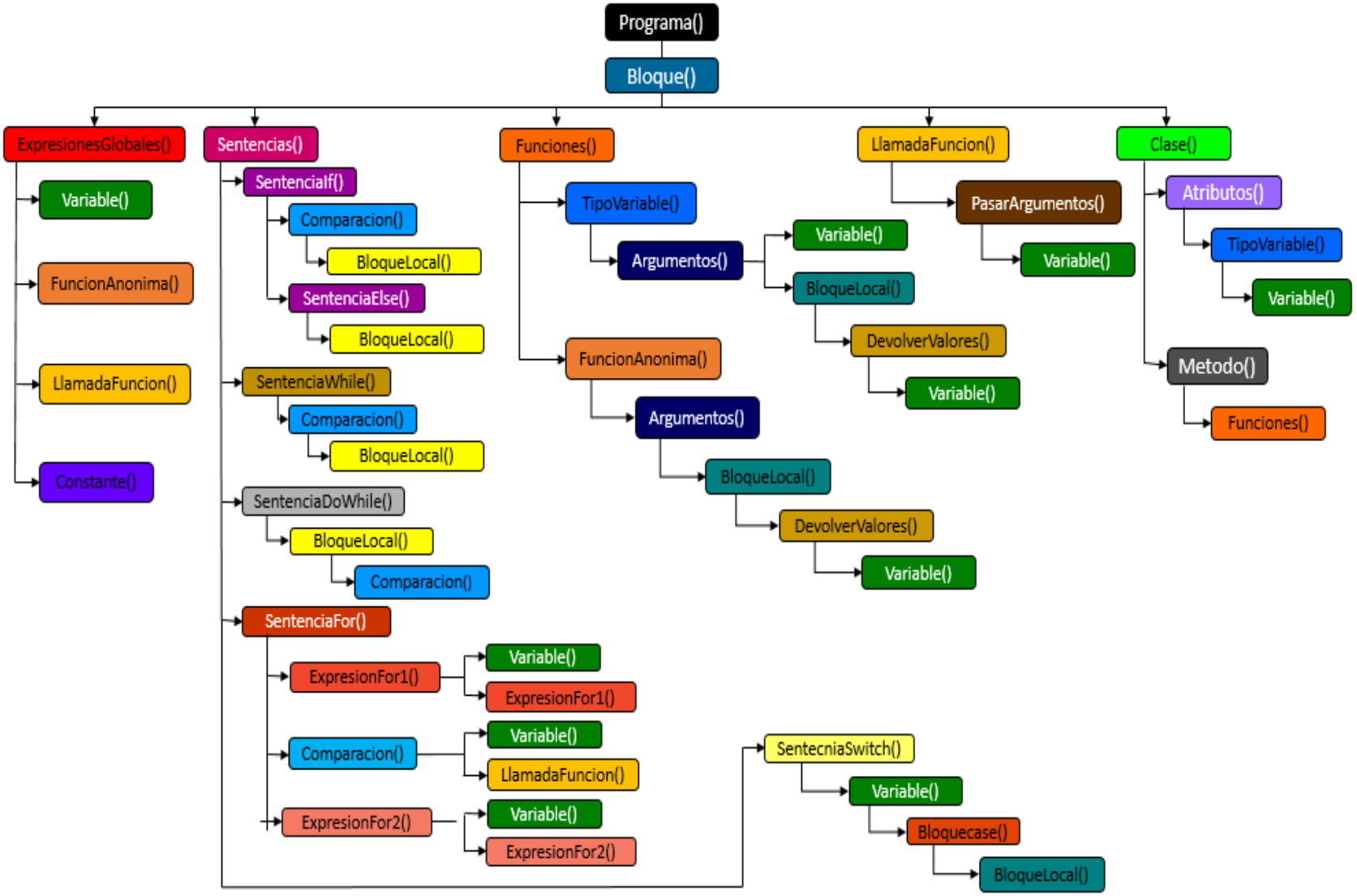

Figura 10. Estructura de los métodos empleados en el proyecto.

Luego de haber descrito el programa (AnalizadorLexSin.jj), se pasará a la implementación del código, tomando la ayuda del compilador JavaCC, el proyecto se presenta en un navegador web (Google Chrome), se contó con un formulario, el cual pedirá el ingreso del código a analizar, a la vez presenta un botón que guardará el código ingresado por el usuario, dicho código se almacenará en un archivo .txt, que en un futuro será el archivo a compilar por JavaCC. Así mismo se presenta dos opciones para alimentar el conocimiento acerca del lenguaje php, y tratar de avanzar, los cuales son los botones que les llevará a ¿Qué es PHP? y a la Documentación. Para su diseño se tomó la ayuda del lenguaje CSS, y así tomar la apariencia como se muestra a continuación:

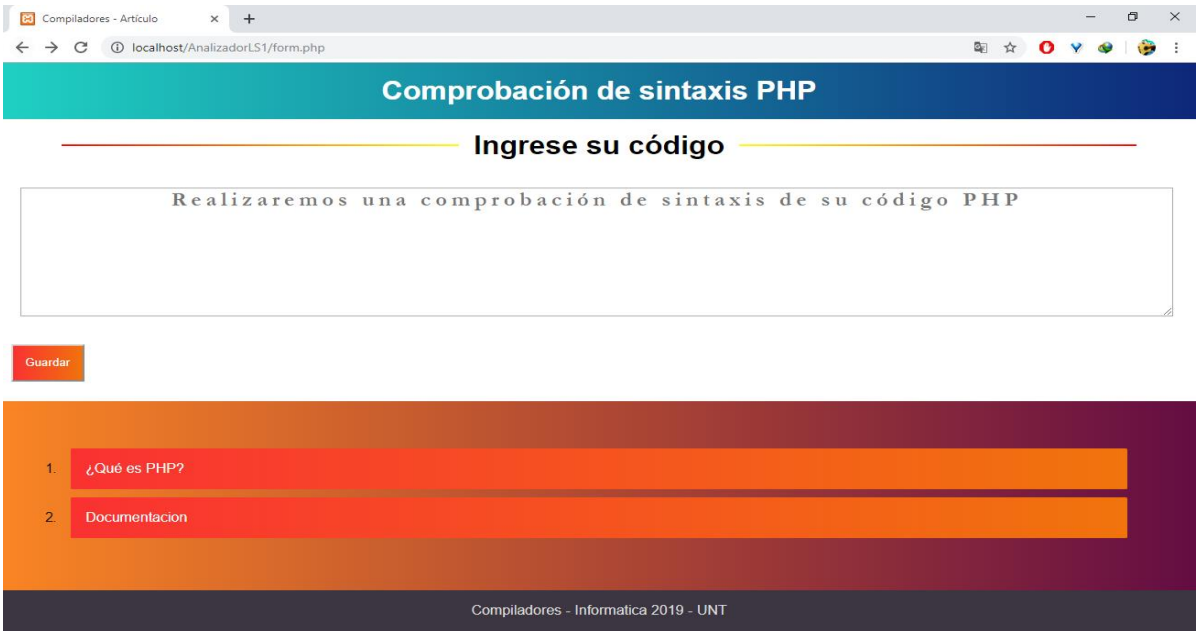

Figura 11: Interfaz para el ingreso del código a analizar.

A continuación, se presenta el ingreso de un pequeño código en php, el famoso Hola Mundo.

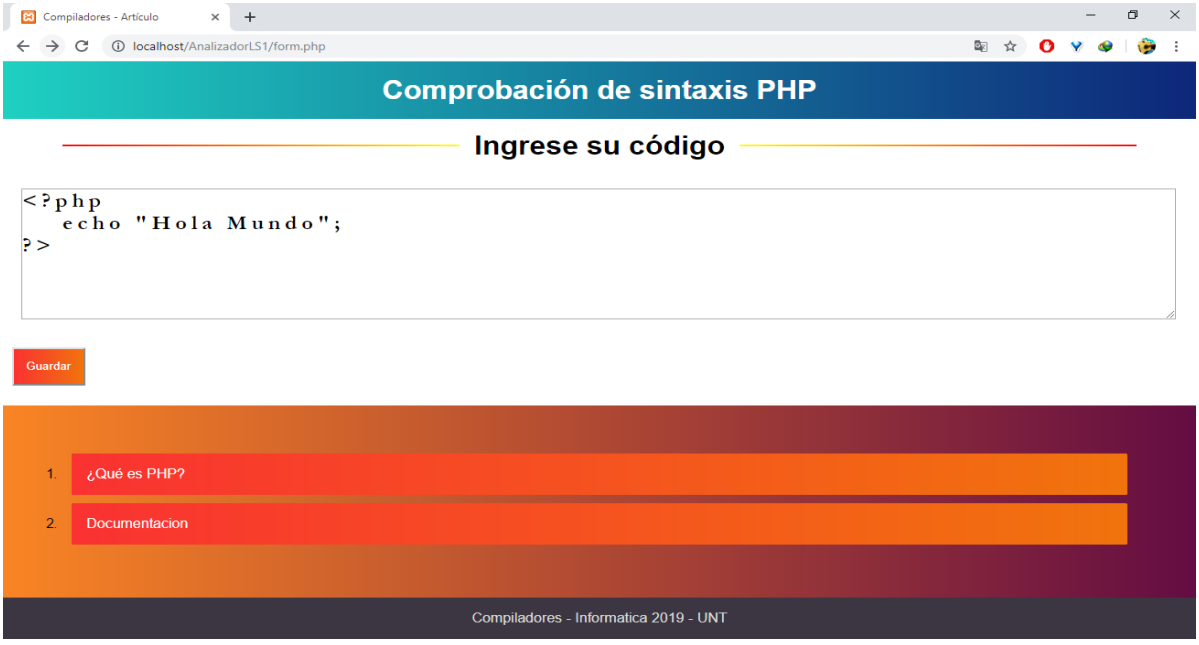

Figura 12: Interfaz de ingreso del código a analizar.

Luego de presionar el botón guardar, el código se almacenará en un archivo .txt (codigophp.txt), como se muestra a continuación:

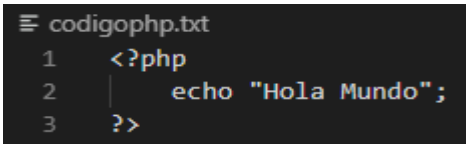

Figura 13. Archivo que contiene el código a analizar.

Continuando con la ejecución del proyecto, se dirigirá al símbolo del sistema, para llevar a cabo la compilación empleando JavaCC, como primer paso se busca a la carpeta que almacena al programa (AnalizadorLS1), estando dentro de la carpeta xampp y htdocs, para que se pueda visualizar en un navegador.

Figura 14. Carpeta Analizador LS1.

Posteriormente se compila empleando el comando javacc y el nombre del archivo que contiene el programa (AnalizadorLexSin.jj).

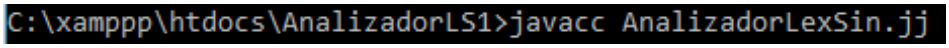

Figura 15. Archivo AnalizadorLexSin.jj

A continuación, se ingresa el comando javac *.java, para la compilación de los archivos .java que fueron creados En el paso anterior, obteniendo como resultado archivos .class.

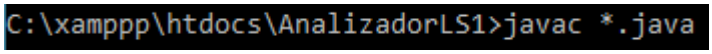

Figura 16. Comando javac *.java

Por último, se ingresa el nombre del archivo que lleva el programa (AnalizadorLexSin) junto con el nombre el archivo (codigophp.txt) que almacenó al código que se espera analizar, el cual fue ingresado por la interfaz.

Figura 17. Archivo codigophp.txt

3. RESULTADOS Y DISCUSIÓN

Luego de haber ingresado el código a analizar y realizar el proceso de análisis sintáctico, el programa muestra los tokens que se empleó al momento de analizar el código ingresado por el usuario, como es el caso de INICIO, ECHO, CADENA, PUNTO Y COMA y FIN, también muestra la cantidad de errores que se presentan en el código y el número de fila y columna en que se posiciona el error sintáctico, en el ejemplo de ingreso de código, no se programó con errores, es por ello que se muestra la cantidad de errores igual a cero, como se observa en la figura 18.

Sin embargo, se probará también ingresando el mismo código, pero con un error sintáctico, por ejemplo, eliminar el punto y coma. Por consiguiente, se mostrará un aviso de ERROR SINTACTICO, la ubicación en que se encuentra dicho error (línea 2, columna 5). y a la vez la cantidad de errores, (igual a 1), Como se observa en la figura 19.

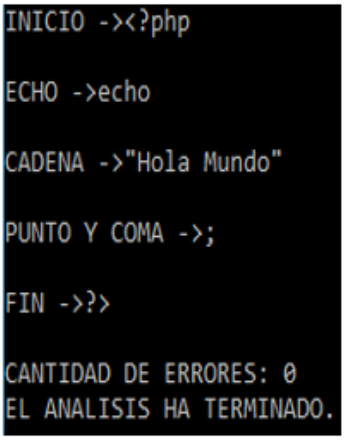

Figura 18. Resultado sin errores.

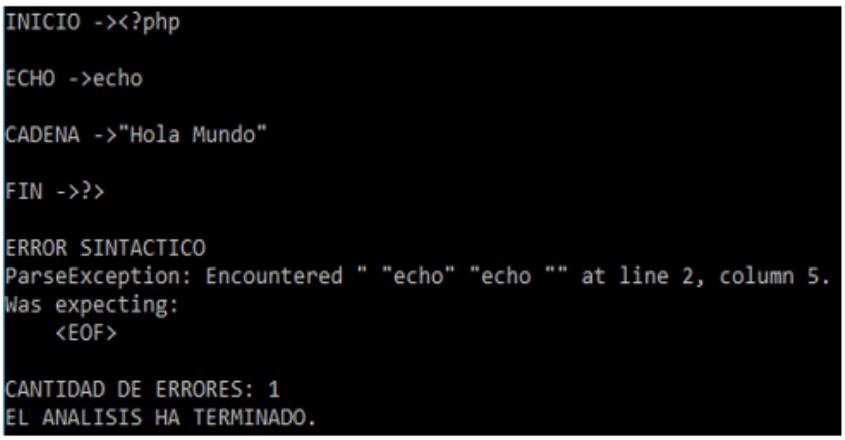

Figura 19. Resultado con un error.

De esta manera, el estudiante comprobará si su código está o no perfectamente escrito, respetando las reglas sintácticas del lenguaje php, así mismo evitará volver a recurrir en los mismos errores sintácticos que presentaba al momento de programar. Alimentando sus ganas de seguir descubriendo y aprendiendo el lenguaje php.

4. CONCLUSIONES

En este artículo se presentó el desarrollo y la implementación de un analizador sintáctico para el lenguaje php, todo ello se llevó a cabo gracias a la ayuda de diferentes tecnologías como el compilador JavaCC, HTML, CSS, lenguaje PHP, XAMPP y Visual Studio Code.

Se pudo demostrar que a pesar de programar en lenguajes que se aproxima al lenguaje natural, aun así, el programador experimenta errores, siendo los más comunes, los errores sintácticos, tal y como sucedió con el ejemplo empleado anteriormente, el cual fue la eliminación del carácter punto y coma, comprobando que el código no respetaba con las reglas sintácticas del lenguaje php.

Como trabajo futuro se continuará con el desarrollo del analizador sintáctico, tomando como objetivo, el poder mostrar la cantidad de errores y la posición de dicho error, ya no por el símbolo del sistema, sino más bien, por la misma interfaz, el cual se ingresó el código a analizar. Convirtiendo a la aplicación en un programa completo y mucho más fácil de interactuar con ella, transformándolo en un programa de muy fácil uso, evitando el manejo de comandos y diferentes nombres de archivos que contiene el programa. Así mismo reducir los pasos al momento de conocer el error y su ubicación.

Concluido con el proyecto, se espera llevar a cabo con la instalación del aplicativo en los laboratorios de la escuela para así apoyar en la formación del alumno que empieza a programar en el lenguaje php.

REFERENCIAS BIBLIOGRÁFICAS

Correa, P. 2015. webnode. Disponible en: https://itm201511.webnode.es/archivos-del-sistema/lenguajes/lenguajes-de-bajo-nivel/

Gálvez, S.; Mora, M. 2005. Java a Tope: Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC. Editorial Universidad de Málaga. Málaga, España. 319 pp.

Ibañez, F.; Díaz , D.; Oviedo, S.; Otazu, A.; Alves, M. 2015. COMPI, una herramienta interactiva para la enseñanza de construcción de compiladores. 5 pp.

Medina, E. 2015. muylinux. Disponible en: https://www.muylinux.com/2015/04/30/visual-studio-code-editor-codigo-microsoft-windows-os-x-gnu-linux/

Norvell, T. 2011. Las preguntas frecuentes de JavaCC. Disponible en: http://www.engr.mun.ca/~theo/JavaCC-FAQ/javacc-faq-moz.htm#tth_sEc1.2

Sierra, A.; Espinoza, M. 2018. Análisis Comparativo entre ASP.NET y PHP. Revista INNOVA Research Journal, 19 pp.

Traver, J. 2010. Sobre los mensajes de error de los compiladores. 4 pp.

Vanegas, C. 2005. Compiladores: un enfoque. 13 pp.