[1] Aplicación de Lenguajes formales para determinar lenguaje de código para Análisis Estadístico básico codificado en lenguaje Perl, Ruby y Julia

Carlos Andrés Mesa Estupiñán
Juan Camilo Vargas Cardenas

carlos.mesa04@uptc.edu.co
juancamilo.vargas@uptc.edu.co

Universidad Pedagógica y Tecnológica de Colombia

Resumen – El presente proyecto tiene como objetivo la creación de un compilador apto para la determinación de lenguajes de programación Ruby,Pearl y Julia, mediante el uso de expresiones regulares y gramáticas regulares ER Y GR. El proyecto tiene como objetivo implementar los conceptos teóricos, para realizar un compilador que reconocerá 3 diferentes lenguajes, para ello se creó un workspace para el ingreso del lenguaje, con el fin de obtener sus estructuras y componentes, así como sus palabras reservadas para finalmente poder estipular de las medidas de tendencia central con cada uno de los lenguajes.

Palabras clave: Expresiones regulares, Gramáticas regulares, compilador, lenguaje, sentencia, sintaxis, libreria

I. INTRODUCCIÓN

Un compilador generalmente genera lenguaje ensamblador primero, para la sucesión traducir el lenguaje ensamblador al lenguaje máquina. Una utilidad conocida como «enlazador» combina todos los módulos de lenguaje de máquina necesarios en un programa ejecutable que se puede ejecutar en la computadora, los primeros compiladores, al igual que las primeras computadoras, fueron hechos por equipos reducidos o hasta por una sola persona.

El objetivo principal de este proyecto es crear el bosquejo de un compilador desde cero que sea capaz de analizar interpretar y traducir el código estipulado de 3 lenguajes de programación diferentes, se esclarece que los lenguajes de programación que se van a compilar son lenguajes existentes de programación.

En el documento se podrá apreciar varios conceptos relacionados con los compiladores su estructura y cómo se lleva a cabo el desarrollo y creación desde cero, se dará una breve explicación de cómo fue construido y que lenguaje de programación se uso así como las librerías usadas respectivamente, se podrá apreciar avances y fallos que se tuvieron y la terminación del proyecto.

[1]

II. Marco referencial

• Marco Teórico

Aprendizaje regular expressions:

Este libro nos introduce en el mundo de las expresiones regulares, proporcionando una base sobre qué son y cómo se utilizan. Explica en detalle cómo se construyen, qué tipos de expresiones se pueden agrupar, las clases y las diferentes combinaciones. También se aborda cuándo no se deben utilizar las expresiones regulares. El libro incluye varios ejemplos que fueron útiles para el desarrollo del proyecto. [2]

ANÁLISIS LÉXICO FORMALIZACIÓN Y DESARROLLO

Este artículo nos acercó bastante para la creación del analizador Léxico con el uso de Jflex el cual *"JFlex es un generador de analizador léxico para Java, escrito en Java."*[2], este generador toma como entrada una especificación con un conjunto de expresiones regulares y acciones correspondientes, genera un programa lexer que lee la entrada, coincide con la entrada con las expresiones regulares en el archivo de especificaciones para luego ejecutar la acción correspondiente si una expresión regular coincide. Los Lexers generalmente son el primer paso frontal en compiladores, palabras clave coincidentes, comentarios, operadores, etc., y generan una corriente de token de entrada para analizadores. Lexers también se puede usar para muchos otros propósitos.

La aplicación de un análisis léxico a los diferentes lenguajes que son tomados en cuenta para el desarrollo de este proyecto llevaron a la creación de los diferentes Tokens para cada lenguaje, estos tokens pueden ser palabras clave, identificadores, operadores, símbolos, números, cadenas, comentarios.[8]

• Marco Conceptual

Compilador:

Un compilador es un pequeño programa informático, que se encarga de traducir (compilar) el código fuente de cualquier aplicación que se esté desarrollando. En pocas palabras, es un software que se encarga de traducir el programa hecho en lenguaje de programación, a un lenguaje de máquina que pueda ser comprendido por el equipo y pueda ser procesado o ejecutado por este.

Un concepto un poco más elaborado es el siguiente: Un compilador es un programa que convierte o traduce el código fuente de un programa hecho en lenguaje de alto nivel, a un lenguaje de bajo nivel (lenguaje de máquina).

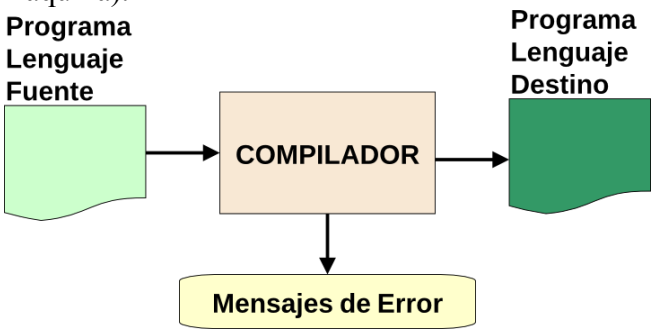


Imagen partes de un compilador [2]

Un compilador tiene varias facetas como se observan en la ilustración 2 se dará una definición de las más importantes para su mejor comprensión.

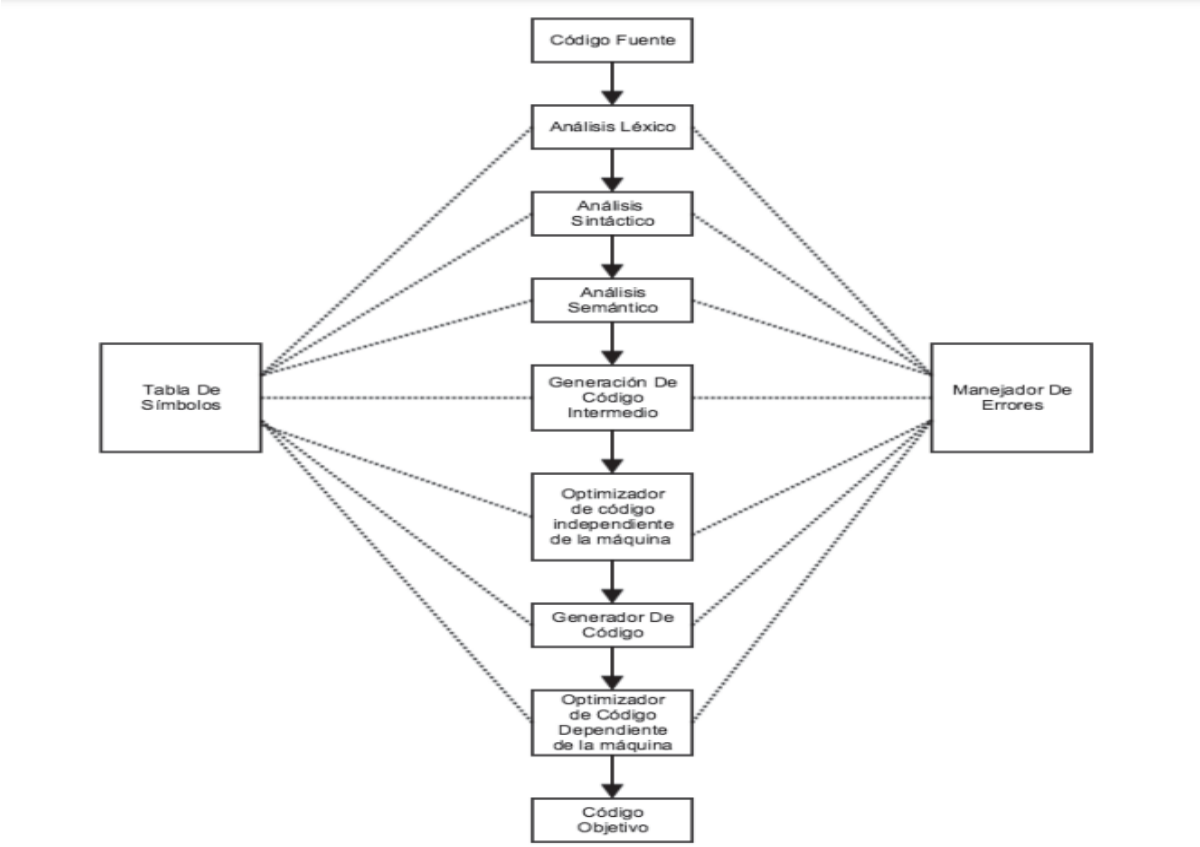


Imagen:Fases del proceso de compilación[6]

Un compilador esencialmente se conforma de los siguientes elementos:

Analizador Léxico:

El análisis léxico es la primera etapa para la definición del lenguaje usado en un compilador, este análisis se encarga de dividir el código fuente en tokens. Se utilizan expresiones regulares para reconocer patrones léxicos, como palabras clave, identificadores, constantes y operadores, los cuales son base para todo lenguaje, estas expresiones se hacen más fáciles de reconocer y procesar para la definición del lenguaje usado.

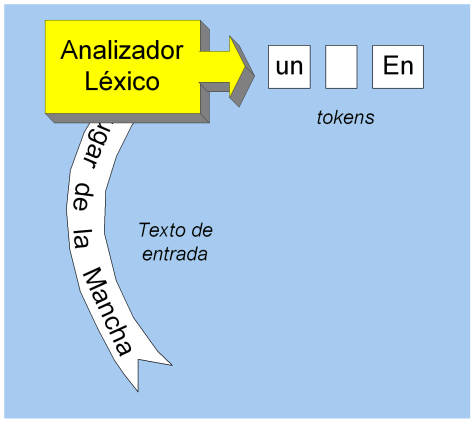


Imagen Interpretación de un Analizador Léxico [3]

Analizador Sintáctico:

La tarea del analizador es, en este caso, la descomposición y transformación de las entradas en un formato utilizable para su posterior procesamiento. Se analiza una cadena de instrucciones en un lenguaje de programación y luego se descompone en sus componentes individuales.
El analizador actual maneja la gramática de los datos de entrada, realiza un análisis sintáctico de éstos y como regla general crea un árbol de sintaxis (árbol de análisis). Esto se puede utilizar para el procesamiento posterior de los datos, por ejemplo, la generación de código por un compilador o ejecutado por un intérprete (traductor). Por lo tanto, el analizador es el software que comprueba, procesa y re envía las instrucciones del código fuente.

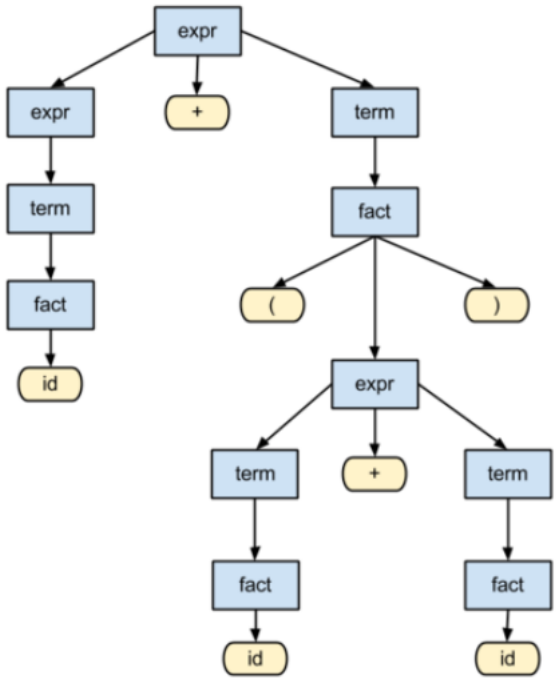


imagen Ejemplo Analizador sintáctico[4]

Analizador Semántico:

El analizador semántico deberá comprobar que estas dos restricciones se cumplen antes de declarar que la sentencia de asignación está bien formada. Pero sucede que la información necesaria para comprobarlas es útil también para generar código. Esto quiere decir que si tuvo éramos una separación estricta entre las fases del compilador, para generar código deberíamos volver a mirar el identificador para saber a qué objeto (variable, función, constante, etc.) corresponde y que tipo tiene y también deberíamos volver a comprobar los tipos de la expresión para generar el código adecuado.

Un método popular de análisis semántico combina el aprendizaje automático y el procesamiento del lenguaje natural para encontrar las ideas principales y las conexiones del texto. Esto puede implicar emplear un modelo de aprendizaje automático entrenado para analizar un texto nuevo y descubrir sus ideas y relaciones clave.[8]

Generación de código:

La generación de código es una de las etapas críticas en el proceso de compilación de un lenguaje de programación. Esta etapa implica la traducción del código fuente de alto nivel (escrito por un programador en un lenguaje de alto nivel) en un programa equivalente en código de máquina o en otro lenguaje de bajo nivel que puede ser ejecutado por una computadora.
La generación de código es una de las partes más desafiantes y críticas del proceso de compilación. Un compilador eficiente debe generar código que sea tanto correcto como rápido de ejecutar. Por esta razón, los compiladores modernos a menudo incorporan técnicas avanzadas de optimización de código para mejorar el rendimiento del programa final.

Tabla de símbolos:

La tabla de símbolos puede iniciarse con cierta información considerada de interés por
el diseñador, como pueden ser constantes, funciones de librería, y si se considerase también
palabras reservadas. De acuerdo con la estrategia que se decida, los identificadores los insertará
en la TS el analizador léxico o el analizador sintáctico, validando antes que no existen ya en la TS.
Tipos simples:

- Entero (int , long,..).
- Real (float, double,...).
- Carácter (char).
- Tipos estructurados:
- Arrays (incluyendo dimensiones).
- Struct o Records (incluyendo cada uno de sus miembros y tipos).
- Tipos archivo o fichero: la representación interna de los archivos
- varía de unos lenguajes a otros.
- Tipos punteros.
- Funciones y procedimientos: con sus parámetros y tipos. En el caso de las funciones el tipo que devuelven.

Manejador de Errores:

Un manejador de errores es una colección de actividades específicas que se ejecutan cuando se genera un error en la actividad con la que el manejador está asociado. Los manejadores de errores se utilizan tanto en actividades de invocación como en actividades de ámbito. Se ejecutan cuando el error lo genera el proceso

en su totalidad, la actividad de invocación o una actividad dentro de la actividad de ámbito. En estos casos, la operación de la actividad padre no se ha confirmado y es posible invertir fácilmente la transacción.

- Captar un error e intentar corregir el problema de modo que el proceso de negocio continúe hasta su finalización normal.
- Captar el error y determinar que no puede resolverse en este ámbito. En este caso, tiene las siguientes opciones adicionales:
- Generar un nuevo error.
- Volver a generar el error original de modo que otro ámbito pueda manejarlo.
- Responder con un error al iniciador del proceso.
- Iniciar una tarea de usuario para corregir el problema. Si el manejador de errores no puede resolver el problema, es posible que deba retrotraer el proceso y compensarlo.
- Para los procesos de larga ejecución, considere la posibilidad de utilizar la propiedad Parar ante errores no manejados del proceso para manejar una situación de error de forma administrativa.

Lenguaje de Programación:

Un lenguaje de programación, en palabras simples, es el conjunto de instrucciones a través del cual los humanos interactúan con las computadoras. Un lenguaje de programación nos permite comunicarnos con las computadoras a través de algoritmos e instrucciones escritas en una sintaxis que la computadora entiende e interpreta en lenguaje de máquina.

Existen una variedad de lenguajes de programación que los programadores pueden usar para comunicarse con una computadora, a través de lo que conocemos como código fuente, y a pesar de que todos comparten las características antes mencionadas, cada lenguaje es diferente y de forma individual puede ser más adecuado utilizarlo para un determinado propósito o propósitos dentro de ciertas industrias.

Los lenguajes de programación se utilizan para crear sistemas operativos, programas de escritorio, aplicaciones móviles, para resolver problemas o interpretar datos. Por ejemplo, existen lenguajes de programación que son más adecuados para crear software o aplicaciones de entretenimiento, para crear dispositivos inteligentes hasta crear robots utilizando inteligencia artificial, los límites de lo que se puede lograr a través de los lenguajes de programación, son desconocidos.

SINTAXIS:

La sintaxis es la estructura de una declaración en un lenguaje de programación, es el conjunto de reglas a respetar para la escritura de código fuente en programas para considerarse correctos cuando se hace uso del lenguaje definido para programar.
Por lo tanto la sintaxis de un lenguaje son las reglas y estructuras gramaticales que posee y gobiernan la escritura de un lenguaje, dichas reglas garantizan que el texto escrito o código, sea comprensible y ejecutable por el intérprete o compilador, en concreto la máquina.

EXPRESIONES REGULARES:

Permiten filtrar textos para encontrar coincidencias, comprobar la validez de fechas, documentos de identidad o contraseñas, se pueden utilizar para reemplazar texto con unas características concretas por otro, y muchos más usos.

El problema es que no son intuitivas a primera vista, por lo que la solución a la que llegan muchos programadores con experiencia que no quieren aprenderse la sintaxis a fondo es tener un conjunto de soluciones que le han servido en el pasado y partir de alguna que se parezca.

JFLEX:

En el analizador léxico el generador toma como entrada una especificación con un conjunto de expresiones regulares y acciones correspondientes. Genera un programa (a lexer) que lee la entrada, coincide con la entrada con las expresiones regulares en el archivo de especificaciones y ejecuta la acción correspondiente si una expresión regular coincide. Los Lexers generalmente son el primer paso frontal en compiladores, palabras clave coincidentes, comentarios, operadores, etc., y generan una corriente de token de entrada para analizadores. Lexers también se puede usar para muchos otros propósitos.[2]

RUBY

Ruby es un lenguaje con un balance cuidado. Su creador, Yukihiro "Matz" Matsumoto, mezcló partes de sus lenguajes favoritos (Perl, Smalltalk, Eiffel, Ada y Lisp) para formar un nuevo lenguaje que incorpora tanto la programación funcional como la imperativa.
Ruby es considerado un lenguaje flexible, ya que permite a sus usuarios alterarlo libremente. Las partes esenciales de Ruby pueden ser quitadas o redefinidas a placer. Se puede agregar funcionalidad a partes ya existentes. Ruby intenta no restringir al desarrollador.

Por ejemplo, la suma se realiza con el operador suma (+). Pero si prefieres usar la palabra sumar, puedes agregar un método llamado sumar a la clase Numeric que viene incorporada.

```
class Numeric
def sumar(x)
self.+(x)
end
end

y = 5.sumar 6
# ahora y vale 11
```

Los operadores de Ruby son simples conveniencias sintácticas para los métodos. Los puede re definir cómo y cuándo se requiera.[16]

JULIA:

Julia es un lenguaje de programación dinámico de alto nivel, diseñado para brindar a los usuarios la velocidad de C/C++ sin dejar de ser tan fácil de usar como Python. Esto significa que los desarrolladores pueden resolver problemas de forma más rápida y eficaz.
Julia es excelente para problemas computacionales complejos. Muchos de los primeros usuarios de Julia se concentraron en dominios científicos como la química, la biología y el aprendizaje automático.
Dicho esto, Julia es un lenguaje de propósito general y puede usarse para tareas como desarrollo web, desarrollo de juegos y más. Julia es visto como el lenguaje de próxima generación para el aprendizaje automático y la ciencia de datos.

```
function mandelbrot(a)
    z = 0
    for i=1:50
        z = z^2 + a
    end
    return z
end
```

```
for y=1.0:-0.05:-1.0
    for x=-2.0:0.0315:0.5
        abs(mandelbrot(complex(x, y))) < 2 ? print("*") : print(" ")
    end
    println()
end
```

PERL

Perl es un lenguaje de programación altamente capaz y rico en funciones con más de 30 años de desarrollo. Perl se ejecuta en más de 100 plataformas desde portátiles hasta mainframes y es adecuado tanto para proyectos de creación rápida de prototipos como para proyectos de desarrollo a gran escala.

"Perl" es una familia de idiomas, "Raku" (anteriormente conocido como "Perl 6"), pero es un idioma separado que tiene su propio equipo de desarrollo. Su existencia no tiene un impacto significativo en la continuación desarrollo de "Perl".

```
#!/usr/bin/perl
$b = 10;    # Assigning value to $b
$c = 30;    # Assigning value to $c

$a = $b + $c;   # Performing the operation
print "$a";     # Printing the result
```

III. Métodos y Resultados

- **Métodos**

VISUAL STUDIO CODE:

Será nuestro id de programación visual studio code es un editor de código de microsoft, en el cual se realizará todo el desarrollo del proyecto, el lenguaje es python por lo tanto se adapta muy bien con este editor.

REPOSITORIO EN GITHUB:

El repositorio del proyecto se encuentra alojado en GitHub en el siguiente enlace:

https://github.com/JuanCVargasC/CompiladoreLenguajes.git

La utilización de GitHub facilita la colaboración entre los miembros del equipo, permitiendo un desarrollo eficiente en el desarrollo proyecto. Además, proporciona un historial detallado de los cambios, lo que es esencial para el proceso de desarrollo.

JAVA:

Java es un lenguaje de programación y una plataforma informática lanzada por primera vez por Sun Microsystems en 1995. Ha evolucionado desde sus humildes comienzos hasta impulsar una gran parte del mundo digital actual, proporcionando una plataforma confiable sobre la cual se construyen muchos servicios y aplicaciones. Los productos nuevos e innovadores y los servicios digitales diseñados para el futuro también siguen confiando en Java. Si bien la mayoría de las aplicaciones Java modernas combinan el

tiempo de ejecución de Java y la aplicación, todavía hay muchas aplicaciones e incluso algunos sitios web que no funcionarán a menos que tenga instalado un Java de escritorio.

El proyecto se fundamenta con la creación de expresiones regulares que permitan reconocer un tipo de lenguaje como por ejemplo Perl, Ruby y Julia, inicialmente realizamos una investigación sobre la creación de un compilador y sus componentes a consideración para la realización del compilador, como los patrones, palabras reservadas, reglas gramaticales de cada lenguaje entre ellos, para la definición en código los mismos se hace uso de la herramienta .flex que permite definir los Tokens de cada lenguaje en un archivo Lexer.flex, los tokens definidos son los siguientes:

```
/* Reservadas Ruby */
def |
end |
if |
else |
elsif |
while |
do |
for |
in |
true |
false |
nil |
self |
class |
module |
return |
puts |
to_f { return token(yytext(), "RUBY_Reserv", yyline, yycolumn); }
```

Imagen 1. Palabras reservadas de Ruby

Para Perl:

```
/* Reservadas Pearl */
my |
List |
def |
use |
end |
if |
else |
elsif |
while |
do |
for |
in |
true |
false |
nil |
self |
class |
module |
return |
puts |
unless |
print |
module { return token(yytext(), "PEARL_Reserv", yyline, yycolumn); }
```

Imagen 2. Palabras reservadas de Perl

Para Julia:

```
/* Tipos de dato julia*/
número |
color |
Int |
UInt |
Float64 |
Bool |
String |
Array |
Matrix |
DataFrame { return token(yytext(), "TIPO_DATO", yyline, yycolumn); }
```

Imagen 3. Palabras reservadas de Julia

El uso de expresiones regulares fue clave para la detección y análisis de las cadenas de texto en los lenguajes, entre ellas tenemos las siguientes:

```
/* Comentario */
Comentario = {ComentarioTradicional} | {FinDeLineaComentario} | {ComentarioDeDocumentacion}

/* Identificador */
Letra = [A-Za-zÑñ_ÁÉÍÓÚáéíóúÜü]
Digito = [0-9]
Identificador = {Letra}({Letra}|{Digito})*
VectorNumeros = "\\[[0-9]+(,[0-9]+)*\\]"
TerminadorDeLinea = \r|\n|\r\n
EntradaDeCaracter = [^\r\n]
/* Número */
Numero = 0 | [1-9][0-9]*
```

Imagen 4. Expresiones regulares

Igualmente se hace el uso de un análisis de sintaxis para generar los errores que pueda generar una mala interpretación del lenguaje.
Los errores pueden darlos por el mal uso de componentes del lenguaje, así como fallas a la hora de completar una expresión, dichos errores fueron capturas en código y eliminados de los bloque de código a compilar, y analizar, a estos bloques se les llamó “Producciones” y cada producción está compuesta por un bloque de código definido correctamente, por ejemplo un bloque de código definido correctamente en Ruby es: puts "¡Hola Mundo!".
En dicho fragmento de código podemos observar que se respetan las normas del lenguaje al hacer uso de las palabras reservadas, así como los signos necesarios para la definición de la palabra a mostrar por pantalla.

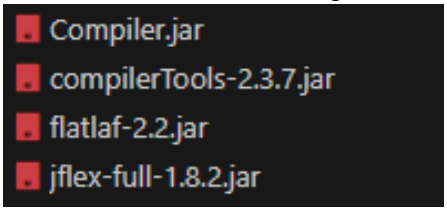
```
/* VAR_PEARLs */
gramatica.group("VAR_PEARL", "TIPO_DATO IDENTIFICADOR_P OP_ASIG VALOR COMA", true, identProd);
gramatica.group("VAR_PEARL", "TIPO_DATO OP_ASIG VALOR", true,
        2, "ERROR, falta Identificador de VAR_PEARL [#,%]");
gramatica.finalLineColumn();
gramatica.group("VAR_PEARL", "TIPO_DATO OP_ASIG IDENTIFICADOR", true,
        3, "ERROR, falta valor de VAR_PEARL [#,%]");

/* VAR_RUBY */
gramatica.group("VAR_RUBY", "IDENTIFICADOR OP_ASIG VALOR", true, identProd);
gramatica.group("VAR_PEARL", "IDENTIFICADOR OP_ASIG VALOR LLAVE_A", true, identProd);
gramatica.group("VAR_PEARL", "IDENTIFICADOR OP_ASIG VALOR PARENTESIS_A", true,
        22, "ERROR, falta Identificador de VAR_RUBY [#,%]");

gramatica.finalLineColumn();

/* ELiminacion de tipos de dato y operadores de asig */
gramatica.delete("TIPO_DATO", 4, "ERROR,Tipo de dato no pertenece a declaracion [#,%]");
gramatica.delete("OP_ASIG", 5, "ERROR,Tipo de asignacion no pertenece a declaracion [#,%]");

/* Agrupar identificadores como un valor */
gramatica.group("VALOR", "IDENTIFICADOR", true);
gramatica.group("PARAMETROS", "VALOR (COMA VALOR)+");
```

Imagen 5. Definición de Errores y bloques de código "Producciones"

- **RESULTADOS**

Se define lenguaje Java para la etapa de desarrollo,dicha elección se lleva cabo teniendo en cuanta la gran cantidad de librerías que facilitan el desarrollo del proyecto, usamos Visual Studio Code como entorno de desarrollo, luego se procede a desarrollar los componentes del compilador, la creación del analizador léxico, analizador sintáctico, la librería conocida como Jflex de Java nos permite crear el analizador léxico, tomando eso como partida realizamos la creación de los Tokens que nos permiten guardar las palabras clave de cada lenguaje los símbolos y caracteres respectivos para poder usar en la comprobación sintáctica.

Estas fueron las librerías que se usaron en el proyecto



La interfaz gráfica se establece de la siguiente manera:

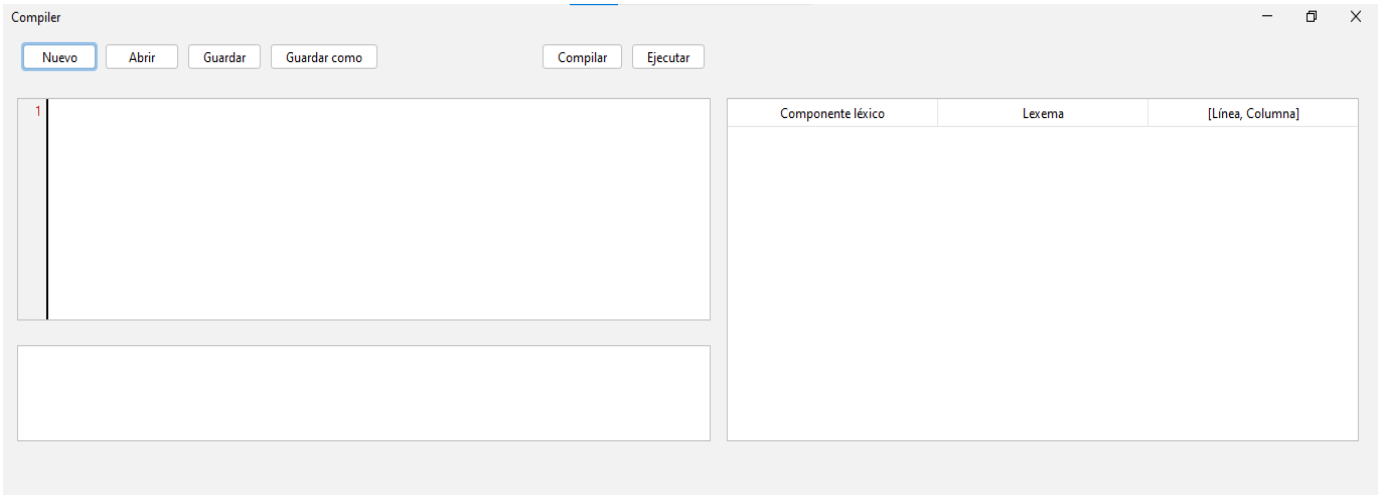


Imagen 5. Interfaz del compilador

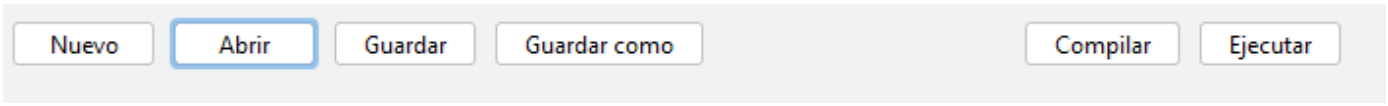


Imagen 6. Menu de opciones

El uso de estas interfaces gráficas nos permiten verificar mejor las funcionalidades de cada una de las etapas del compilador y entender más adecuadamente las funciones y respuestas del compilador en tiempo de ejecución.
la interfaz gráfica nos permite interactuar con el espacio de trabajo asi poder manejar de manera libre los componentes de esta misma, y por la información que se desea transmitir es muy eficaz representarla de este modo se procesa mucho mejor para su respectivo análisis.

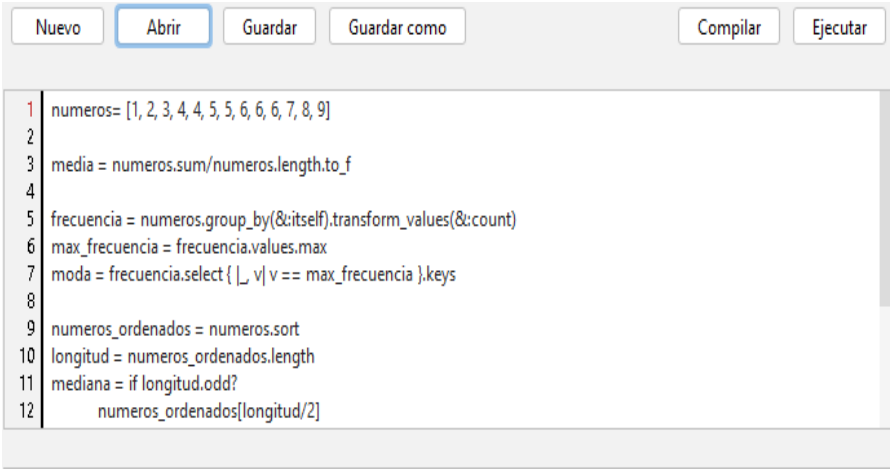


Imagen 7. Prueba de las medidas de tendencia central en perl

Componente léxico	Lexema	[Línea, Columna]
RUBY_VAR	numeros	[1, 1]
OP_ASIG	=	[1, 8]
LLAVE_A	[	[1, 10]
NUMERO	1	[1, 11]
COMA	,	[1, 12]
NUMERO	2	[1, 14]
COMA	,	[1, 15]
NUMERO	3	[1, 17]
COMA	,	[1, 18]
NUMERO	4	[1, 20]
COMA	,	[1, 21]
NUMERO	4	[1, 23]
COMA	,	[1, 24]
NUMERO	5	[1, 26]

Imagen 8. componentes léxicos reconocidos satisfactoriamente

```
Compilación terminada...

ERROR,Tipo de asignacion no pertenece a declaracion [1,8]
 Error sintáctico 23: la llave no está contenida en una agrupación [1, 48]
ERROR,Tipo de asignacion no pertenece a declaracion [3,7]
```

Imagen 9. Errores de compilación

El ejercicio expuesto anteriormente representa el cálculo de las medidas de tendencia central como se observa en la imagen 7 se tiene cargado directamente el código para su ejecución, al realizar la compilación se reconoce el lenguaje en el que está escrito el código y se procede a validar los componentes léxicos del lenguaje para ver si son correctos y aceptados por el compilador así como se observa en la imagen 8,
Finalmente cuando se realiza la compilación se evalúa si hay errores en las funcionalidades del código y se da la retroalimentación de cada error producido en su compilación, como se observa en la imagen 9 se muestra la columna y fila de donde esta el error.

IV. CONCLUSIONES

- La compresión de la estructura y funcionamiento de un compilador nos permite entender de una forma más profunda un correcto manejo de los mismos.
- La aplicación del conocimiento teórico de Lenguajes formales, fomenta la fácil comprensión de diversos lenguajes, generando una mayor versatilidad en el manejo de diferentes lenguajes.
- Se puede observar cómo los diferentes compiladores consideran una tolerancia a los fallos de sintaxis, para garantizar el mínimo de fallos en el código.
- El estudio de expresiones regulares, y definición de palabras reservadas por cada Lenguaje de programación genera una comprensión en las similitudes y diferencias entre los lenguajes aplicados al proyecto, así como otros.

- La comprensión de expresiones regulares y lenguajes formales facilita la comprensión de código y fomenta la aplicación de un proceso de análisis y conocimiento del lenguaje para un correcto uso del mismo en el desarrollo de Software que prioriza la optimización de recursos.
- El estudio del funcionamiento de un compilador, fomenta el estudio y aplicación de buenas prácticas en el desarrollo de software.

V. Referencias

[1] G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.

[2] "Regular Expressions Tutorial," Riptutorial.com, 2022. [En línea]. Disponible:

[3] "JFlex - The Fast Scanner Generator for Java," JFlex.de, 2022. [En línea]. Disponible:

[4] T. Hinostroza, "Jugando con Palabras – Analizadores Léxicos I," Blog de Tito Hinostroza, 19-Dic-2016. [En línea]. Disponible:

[5] "¿Qué es un Analizador Sintáctico?," Ryte Marketing Wiki, 2022. [En línea]. Disponible:

[6] "Semántico," UJI, 2022. [En línea]. Disponible:

[7] J. Domínguez Pérez, "Herramientas de ayuda para compiladores," Archivo Digital UPM, 2022. [En línea]. Disponible:

[8] "Análisis léxico," Cartagena99, 2022. [En línea]. Disponible:

[9] "Registraduría Nacional del Estado Civil - Colombia," Apitude, 2022. [En línea]. Disponible:

[10] EDteam, "Las mejores APIs públicas para practicar," EDteam Blog, 2022. [En línea]. Disponible:

[11] "Analizador léxico (uji.es)," UJI, 2022. [En línea]. Disponible:

[12] IBM, "Using fault handlers," IBM Documentation, 2022. [En línea]. Disponible:

[13] "Función del analizador léxico (uaeh.edu.mx)," Cidecame.uaeh.edu.mx, 2022. [En línea]. Disponible:

[14] J. Barberá Torralvo, "PFC_JOSE_BARBERA_TORRALVO.pdf (upm.es)," Archivo Digital UPM, 2022. [En línea]. Disponible:

[15] Ryte Marketing Wiki, "¿Qué es un Analizador Sintáctico?," Ryte Marketing Wiki, 2022. [En línea]. Disponible:

[16] Ruby-lang.org, "About Ruby (ruby-lang.org)," Ruby-lang.org, 2022. [En línea]. Disponible: