



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación
Sede Alajuela

Compiladores e Intérpretes

Profesor:
Esteban Arias Mendez

Proyecto Intérprete: Tercera parte

Esteban Segura y Marvin Carranza

Junio 2015

Abstract:

In this document we present the final part of the Compilers and Interpreters project. In the final part, we will develop a semantic analyzer and a generator of code, which evaluate the semantic of the program write with the Micro Perl language. Also we present some information about the Perl programming language (because we will develop an interpreter for a micro Perl language), some of the most relevant concepts for the project, and the tools used: JFlex, CUP, Java, NetBeans and the result of the Scanner and Parser implementing in the Micro Perl language.

Tabla de Contenidos

1	Introducción	2
2	Marco Teórico	3
2.1	Lenguaje Elegido	3
2.2	Justificación	3
2.3	Objetivos	3
2.4	Definiciones	4
2.5	Lenguaje de Implementación: Java	5
2.6	Herramientas Utilizadas	6
3	Desarrollo Parte 1	7
3.1	Instrucciones del Lenguaje	7
3.2	Tipos de datos del Lenguaje	10
3.3	Símbolos a utilizar	10
3.4	Palabras reservadas	11
3.5	Características del lenguaje	12
3.6	Alfabeto	12
3.7	Delimitadores	13
3.8	Análisis de Resultados. Parte 1:	14
4	Desarrollo Parte 2	19
4.1	Análisis de Resultados. Parte 2:	19
4.2	Errores Sintácticos	24
4.3	Correcciones/Modificaciones	72
5	Desarrollo Parte 3	73
5.1	Correcciones/Modificaciones	73
5.2	Tipos de errores semánticos	73
5.3	Errores semánticos	74
5.4	Declaracion de los cambios realizados en las palabras reservadas o en la estructura del lenguaje	117
5.5	Características agregadas y decisiones de diseño	117
5.6	Analisis de resultados	118
6	Conclusiones	122
7	Apéndice 1: Notación BNF	123
8	Apéndice 2: Programas de ejemplo	126
9	Apéndice 3: Archivo de configuración	132
10	Referencias	146

1 Introducción

En este documento se presenta la ultima parte del proyecto de Compiladores e Intérpretes, en el proyecto se combinan la implementación del analizador léxico, un analizador sintáctico, un analizador semántico y un generador de código para el interprete para Perl, llamado Micro Perl, el analizador léxico se desarrollo utilizando el generador de analizador léxicos Flex, mediante una lista de token establecidos en el archivo de configuración. El analizador sintáctico se desarrollo utilizando el generador de analizador sintáctico por medio del programa CUP, el cual se encarga de generar el analizador sintáctico de acuerdo a un conjunto de reglas especificadas por el equipo de trabajo (Gramática BNF), las cuales tienen la función de especificar la estructura de un programa aceptado por el lenguaje por medio de un conjunto de reglas definidas previamente. El analizador semántico fue desarrollado sin programas secundarios, como en el caso del analizador léxico y sintáctico generados por programas externos, el analizador semántico se desarrollo utilizando la tabla de símbolos generada por el mismo programa en el que se ejecuta el lenguaje correspondiente, en el mismo se va comprobando la tabla de símbolos con el código, validando así la semántica de acuerdo con una serie de reglas establecidas previamente por el equipo de trabajo, las cuales debe cumplir el lenguaje, para ejecutar las instrucciones necesarias.

El documento inicia con una sección dedicada a brindar información sobre el lenguaje de programación Perl, lenguaje elegido en nuestro proyecto, además, la justificación del porque se eligió dicho lenguaje, y el objetivo del proyecto.

Luego se brinda una pequeña explicación de algunos de los conceptos más relevantes para el proyecto, como lo son: Lenguaje, gramática, Forma BNF, Scanner o analizador léxico, Parser o analizador sintáctico, analizador semántico e intérprete.

Además, se extiende el documento con la información presentada anteriormente con la propuesta del proyecto, incluyendo datos sobre el lenguaje a utilizar para desarrollar el proyecto, las herramientas a utilizar, las instrucciones que el interprete va a leer y ejecutar y una gramática BNF, en la cual se especifica las reglas que el interprete debe seguir.

Se incluye la informacion del analisis de resultados del segundo entregable asi como la descripcion de los errores encontrados y tipos de errores presentes en el segundo entregable.

Además se incluye informacion del tercer y ultimo entregable en el cual se especifican los cambios realizados en el documento, se incluye el analisis de resultados, con las diferentes pruebas realizadas para validar el correcto funcionamiento del analizador semantico y la generacion de codigo. El objetivo principal con este proyecto es comprender el desarrollo y funcionamiento de los interpretes modernos, analizando el funcionamiento de las tres partes primordiales utilizadas en la construccion de compiladores o interpretes: Análisis Léxico, Análisis Sintáctico y Análisis Semántico, así como el generador de código.

2 Marco Teórico

2.1 Lenguaje Elegido

El lenguaje elegido para realizar el proyecto se basará en Perl.

Perl es un lenguaje de programación de propósito general creado por Larry Wall en 1987, desarrollado originalmente para la manipulación de texto, y que ahora se utiliza para una amplia gama de tareas. Perl toma las mejores características de lenguajes como C, awk, sed, sh y BASIC entre otros, soporta tanto programación procedural como orientada a objetos. Un programa en Perl consiste en una secuencia de declaraciones y sentencias, permite ciclos, subrutinas y otras estructuras de control para saltar dentro del código, y utiliza ; para cada declaración. (Perl Tutorial, s.f.)

Para el proyecto de Compiladores y Interpretes, el plan es realizar modificaciones al lenguaje Perl, generando un nuevo interprete que acepte estas modificaciones, y con un numero menor de instrucciones.

2.2 Justificación

Se eligio realizar un subconjunto de Perl, debido a que nos asignaron una tarea con el lenguaje Perl en otro curso, por lo que fue necesario estudiar la estructura y sintaxis del lenguaje, y nos parecio muy interesante las funcionalidades que tenia, que otros lenguajes en ocasiones no muestran, ademas de la manera de declarar funciones y variables, anteponiendo un my (esto es porque se utiliza una libreria que obliga colocar el my para que el programa sea valido, en nuestro programa es obligatorio por la gramatica BNF establecida). Ademas de la diferencia de tipos aplicada en Perl con los simbolos \$ para variables,@ para arreglos y % para Hash (llave-valor), por esos motivos de decidio realizar un subconjunto de Perl, con la intencion de comprender mejor ese lenguaje y si es posible realizar un lenguaje basado en Perl, con una sintaxis mas amigable, comparada con Perl que resulta algo confusa.

2.3 Objetivos

Objetivo general:

Creación de un intérprete para un subconjunto o lenguaje reducido de Perl, con algunas modificaciones mínimas, al cual llamaremos MicroPerl.

Objetivo Parte I:

Construir un analizador léxico para el lenguaje MicroPerl utilizando la herramienta JFlex.

Objetivo Parte II: Construir un analizador sintáctico para el lenguaje MicroPerl utilizando la herramienta CUP.

Objetivo Parte III: Construir un analizador semántico y generador de código(Intérprete) para el lenguaje MicroPerl.

2.4 Definiciones

- Lenguaje

Según la Real Academia Española, lenguaje se refiere al estilo y modo de hablar, expresarse, escribir o comunicarse de cada persona en particular. Sin embargo, en el área de la computación, un lenguaje (de programación), según el artículo “Lenguaje de Programación”, se refiere a un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras; puede usarse para crear programas que controlen el comportamiento físico y lógico de una máquina para expresar algoritmos con precisión. Además, está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

- Gramática

Según la Real Academia Española, es la ciencia que estudia los elementos de una lengua y sus combinaciones, además define los usos correctos de una lengua mediante reglas.

En el área de la computación, según el artículos “Compiladores: Análisis Sintáctico” de Gloria Inés Álvarez, la gramática se define como un conjunto de símbolos terminales (símbolos elementales del lenguaje) y no terminales (símbolos sustituidos por grupos de símbolos terminales de acuerdo a las reglas de producción), además de una serie de reglas de producción, que definen la estructura de un lenguaje.

- BNF

Según Lars Marius Garshol, en “BNF and EBNF: What are they and how do they work?”, BNF significa Backus-Naur Form, o Forma Backus-Naur, y es una forma matemática para describir un lenguaje, desarrollada para describir la sintaxis del lenguaje de programación Algol 60. Además, es utilizada para definir formalmente la gramática de un lenguaje. Su nombre se debe a John Backus y Peter Naur. La BNF consiste en algo así como un juego matemático, empezando por un símbolo, llamado símbolo de inicio, y luego una serie de reglas para reemplazarlo, llamadas reglas de producción. El lenguaje definido por la gramática BNF es solo el conjunto de todas las cadenas producidas siguiendo esas reglas.

- EBNF:

- Scanner (Analizador léxico)

Según Eduardo Serna, en “Compiladores”, un scanner es un programa que tiene como función leer el programa fuente como un archivo de caracteres, y dividirlo en tokens, siendo estos las palabras reservadas para un lenguaje y consisten en una secuencia de caracteres que representa una unidad de

información en el programa fuente; además, el scanner también elabora como salida la secuencia de componentes léxicos o tokens que utiliza el analizador sintáctico o parser.

- **Parser (Analizador sintáctico)**

Según Kenneth Louden en su libro “Construcción de Compiladores”, la tarea del analizador sintáctico o parser, es determinar la estructura sintáctica de un programa a partir de los tokens producidos por el scanner, y construir un árbol de análisis gramatical o árbol sintáctico que represente esta estructura. Por lo tanto, el parser se puede ver como una función que toma como entrada la secuencia o lista de tokens generada por el scanner, y produce como salida el árbol sintáctico.

- **Analizador Semántico**

Se entiende como semántica, según su definición en “Definicion.de”, al estudio de los signos lingüísticos y sus combinaciones, y que además, está vinculada al significado, sentido e interpretación de palabras, expresiones o símbolos. Por lo tanto, podría deducirse que el análisis semántico se refiere al proceso de comprobación del sentido de un programa, es decir, que el código ahí descrito tenga sentido según las reglas del lenguaje.

Según “Capítulo 5. Análisis semántico”, el analizador semántico es un programa que utiliza como entrada el árbol sintáctico generado por el parser para comprobar restricciones de tipo y otras limitaciones semánticas y preparar la generación de código.

- **Intérprete**

Según “Intérpretes y Diseño de Lenguajes de Programación”, un intérprete, en computación, se define como un programa escrito en un lenguaje de implementación, que analiza y ejecuta simultáneamente un programa descrito en un lenguaje fuente.

2.5 Lenguaje de Implementación: Java

El lenguaje elegido para la implementación del proyecto fue Java, en nuestro caso, Java JDK 7.

Según “definición.de”, Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems en 1991. La intención de Sun era crear un lenguaje con una estructura y una sintaxis similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel. Entre las facilidades de Java tenemos:

Orientación a objetos: Ofrece un gran control sobre el código y una mejor organización.

Flexibilidad: Preparado especialmente para la reutilización de código, actualizaciones de código con mucha facilidad.

Funciona en cualquier plataforma: Las aplicaciones funcionan en cualquier plataforma, ya que no es el sistema quien las ejecuta, sino la máquina virtual de Java JVM.

Gratis: Para programar en Java basta con descargar el JDK (Java Development Kit), el cual es totalmente gratis.

Fuente abierta: Java ofrece el código de casi todas sus librerías abiertamente.

Elegimos el lenguaje Java ya que, como podemos ver en lo dicho anteriormente, posee muchas facilidades para el programador, además de que es uno de los lenguajes más utilizados y por lo tanto existe mucha documentación y ayuda en internet en caso de ser necesaria. También, debemos mencionar Java posee todas las estructuras y herramientas necesarias para el desarrollo de nuestro proyecto, por ejemplo el caso de JFlex para el desarrollo del Parser y CUP para el Scanner.

2.6 Herramientas Utilizadas

Lenguaje de Implementación:

Java JDK 7

Para nuestro proyecto, se utilizó Java JDK 7, explicado anteriormente.

IDE:

NetBeans Versión 8.0.1

Según la página oficial de Nebeans (netbeans.org), NetBeans es un entorno de desarrollo integrado (IDE), modular, de base estándar (normalizado), escrito en el lenguaje de programación Java, cuyo proyecto consiste en un IDE de código abierto y una plataforma de aplicación, las cuales pueden ser usadas como una estructura de soporte general (framework) para compilar cualquier tipo de aplicación Java.

Flex:

JFlex Version 1.6.1

Según la pagina oficial de JFlex, es una herramienta que permite generar analizadores léxicos, conocidos como scanner, para el lenguaje Java. El analizador léxico generado por JFlex esta basado en un autómata finito determinista (DFA), realizando el análisis de una manera más eficiente en su procesamiento.

Yacc:

CUP Version 0.11a

CUP (Construction of Useful Parsers) es una herramienta que permite generar analizador sintácticos o parsers para un lenguaje determinado desarrollado por C. Scott Ananian, Frank Flannery, Dan Wang, Andrew W. Appel y Michael Petter ([www2.cs.tum.edu,S.F](http://www2.cs.tum.edu/S.F)). El generador CUP funciona estableciendo una gramática, con sus terminales y no terminales, así como el conjunto de reglas a utilizar para generar el analizador sintáctico, en caso de no haber ciclos , el analizador se genera correctamente, en caso contrario retorna un error por posibles ciclos infinitos en la gramática generada.

Para el programa se utilizo como base un tutorial sobre como utilizar JFlex y CUP obtenido de la pagina web Youtube por el usuario Arturo De Casso, el cual

fue muy útil para entender como funcionaban ambas herramientas. En el tutorial se indica donde obtener las librerías, además se brinda un código de ejemplo, el cual se utilizó para ajustar el programa según se necesitara, cambiando la gramática y las reglas sintácticas, este video se encontró al intentar buscar tutoriales completos de JFlex con Jacc (Generador de analizado léxico que se iba a utilizar en primer lugar pero ante la poca documentación encontrada se decidió cambiar por CUP), en el código para generar el analizador léxico, se indican las reglas léxicas y expresiones regulares para validar los tokens aceptados por el programa. Para el código a generar para el analizador sintáctico, se establece la gramática BNF, el cual se define la estructura del lenguaje. Además de los archivos de configuración para generar las clases del Analizador Léxico y del Analizador Sintáctico.

3 Desarrollo Parte 1

3.1 Instrucciones del Lenguaje

1. if

Sentencia condicional utilizada para comprobar si se cumple o no una condición.

Estructura:

```
if (condición==parámetro) {  
    "Hacer algo"  
}
```

2. elseif

Combinación de else con if. Extiende la sentencia if para ejecutar una sentencia diferente en caso de que la condición de if no se cumpla.

Estructura:

```
if (condición==parámetro1) {  
    "Hacer algo"  
}  
elseif (condición==parámetro) {  
    "Hacer algo"  
}
```

3. else

Complemento de la sentencia if. Su idea es que cuando la condición if no se cumpla, la siguiente instrucción es else.

Estructura:

```
if (condición==parámetro) {  
    "Hacer algo"  
}  
else {  
    "Hacer esto si la condición no se cumple"  
}
```

4. while

Esta sentencia ejecutará repetidamente un bloque de código mientras su condición se cumpla.

Estructura:

```
while (condición==parámetro)
    "Hacer algo"
}
```

5. for

Sentencia que ejecuta un bloque de código mientras cumpla la condición. Esta sentencia tiene tres expresiones separadas por punto y coma. Éstas son la inicialización de un contador, la condición, y la modificación del contador.

Estructura:

```
for(contador ; condición{< | > | ==} {parámetro} ; contador{++|-}) {
    "Hacer algo"
}
```

6. foreach

Esta sentencia se ejecuta sobre una lista de variables. Ejecuta el bloque de código por cada elemento en la lista.

Estructura:

```
foreach (elemento_lista) {
    "Hacer algo"
}
```

7. switch

La sentencia switch es una forma de expresión de un anidamiento múltiple de instrucciones if else. Ejecuta el bloque de código del caso que cumpla la condición, de lo contrario, ejecuta el bloque de código default.

Estructura:

```
switch (condición) {
    case valorparámetro:
        "Hacer algo"
    break;
    default:
        "Hacer algo"
}
```

8. return

Sentencia que retorna un valor. Normalmente se utiliza como retorno de resultado de una función, además de cortar la ejecución del código, una vez que se retornó un valor.

Estructura: return valor;

9. print

Sentencia que imprime una cadena de texto o una lista de cadenas en la consola.

Estructura:

```
print parámetro—valor—variable;
```

10. fun

Sentencia utilizada para declarar funciones.

Estructura:

```
fun nombreFunción (parámetros){  
    "Código"  
}
```

11. do-while

Esta sentencia es similar al while, con la diferencia de que el bloque de código del do-while se ejecutará siempre al menos una vez, hasta que se cumpla la condición establecida en el while.

Estructura:

```
do {  
    "Hacer algo"  
} while (condición);
```

12. shift

Sentencia que regresa el primer elemento de un arreglo y reduce su tamaño en 1.

Estructura:

```
shift(arreglo);
```

13. unshift

Sentencia que permite anadir nuevos elementos al principio de un arreglo.

Estructura:

```
unshift (arreglo, elemento);
```

14. pop

Sentencia que regresa el último valor de un arreglo, y reduce su tamaño en 1.

Estructura:

```
pop(arreglo);
```

15. push

Sentencia que agrega elementos en una lista en la ultima posición.

Estructura:

```
push (arreglo1, elemento);
```

16. length

Sentencia que retorna la longitud de una cadena.

Estructura:

```
length(cadena);
```

17. break

Sentencia utilizada para interrumpir el flujo de ejecución de un ciclo, es decir, corta la ejecución normal de un ciclo.

- Estructura:
break;
18. join
 Sentencia que regresa una cadena de la unión de todos los elementos de un arreglo por la cadena. Por ejemplo: join(" >> ",["A","B"]
 retorna A >> B
 Estructura:
 join(cadena, arreglo);
19. split
 Sentencia que divide una cadena o string de acuerdo a un patrón.
 Estructura:
 split (patron, cadena, limite);
20. comentarios
 Los comentarios son bloques de texto que sirven para explicar y documentar código fuente.
 Estructura:
 /* Texto */

3.2 Tipos de datos del Lenguaje

- Entero: El tipo de datos Entero se utilizara para manejar los numeros, tendrá un rango entre -2147483648 y 2147483647 y un tamaño de 32 bits similar a Java, lenguaje sobre el cual se escribirá el intérprete. Para definir un entero se utiliza el "\$", por ejemplo: \$num = 9;
- Cadena: El tipo de datos Cadena se utilizará para manejar las cadenas o "strings", las cuales tienen un tamaño límite según el espacio disponible en memoria. Para la definición de una cadena, al igual que un entero, se utiliza el "\$", por ejemplo: \$var = "HolaMundo!";
- Arreglo: El tipo de datos para Arreglos se utilizara el simbolo @ (aroba) para representar este tipo de datos, por ejemplo: @arr = (1,2,3);. Su tamaño está limitado por el espacio disponible en memoria.

3.3 Símbolos a utilizar

La mayoría de simbolos a utilizar se basaran a los utilizados en PERL, sin embargo se realizaran cambios en el simbolo de concatenacion de cadenas y arreglos ".," en el intérprete a realizar se utilizará +. Otros simbolos a utilizar seran:

Operadores Lógicos:

==, &,&&,||,|||,! =

Operadores Aritméticos:

+, -, *, /, ++, --, <, >, =, <=, >=

Otros Operadores:

{},@,\$,%

3.4 Palabras reservadas

Lista de las palabras reservadas

- my: Para declarar una funcion o una variable.
- fun: Palabra reservada para declarar una funcion.
- if: Palabra reservada utilizada para iniciar el ciclo if.
- else: Palabra reservada que representa el "entonces" en un ciclo if.
- elseif: Palabra reservada que permite la secuencia de codigo en caso de que no se cumpla una condicion anterior, declarar una condicion en el flujo del codigo.
- switch:Palabra reservada utilizada para iniciar el ciclo switch, que permite buscar en diferentes condiciones.
- case: Palabra reservada que permite comprobar una de las condiciones del ciclo switch.
- default: Palabra reservada, la cual permite determinar la accion a tomar en caso de que no se cumplieran los otros casos asignados en el ciclo switch
- while: Palabra reservada que permite iniciar el ciclo while.
- for: Palabra reservada que permite iniciar el ciclo for.
- foreach: Palabra reservada que permite iniciar el ciclo foreach.
- return: Palabra reservada que permite retornar un valor y detener la ejecucion de un ciclo.
- print: Palabra reservada que permite imprimir un valor sin afectar el flujo normal de ejecucion.
- shift: Palabra reservada que permite obtener el primer elemento de un arreglo y reducir su tamano.
- unshift:Palabra reservada que permite agregar un nuevo elemento al inicio de un arreglo
- pop: Palabra reservada que permite retornar el ultimo elemento de un arreglo y reducir el tamano del arreglo.
- push: Palabra reservada que agrega un nuevo elemento en la ultima posicion.

- length: Palabra reservada que permite obtener el largo de un arreglo
- break: Palabra reservada que permite detener el flujo de un ciclo.
- join: Palabra reservada que permite realizar un join con el tipo de datos String.
- split: Palabra reservada que permite dividir un arreglo o un String de acuerdo a un patron.

3.5 Características del lenguaje

El lenguaje mantiene la sintaxis similar a Perl, con subconjunto de instrucciones de Perl, manteniendo la forma en la que se llaman las funciones definidas dentro del lenguaje de Micro Perl, con el nombre de la funcion y entre parentesis, los parametros necesario. Para declarar funciones y variables es necesario colocar la palabra my para indicar que es una funcion o variable, ademas la variable debe de ir precedida por uno de los simbolos establecidos (\$,%,@) segun el tipo que se desee declarar. Los comentarios seran comentarios de una linea iniciando con el simbolo /* y terminando con */. Los rangos de las variables seran los mismos que el lenguaje en el cual se desarrollara el interprete (Java), manejando las variables de tipo Entero como el tipo Integer, de la misma manera con el tipo String.

3.6 Alfabeto

El alfabeto incluye las letras minúsculas y mayúsculas: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z; los números: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; y además, los siguientes símbolos:

1. (
2.)
3. [
4.]
5. {
6. }
7. ;
8. ,
9. =
10. ==

11. !=

12. &

13. &&

14. |

15. ||

16. +

17. ++

18. *

19. -

20. -

21. /

22. <

23. <=

24. >

25. =>

26. ,

27. :

3.7 Delimitadores

La lista de delimitadores utilizados en el lenguaje son los siguientes:

1. Cambio de linea.
2. Retorno de carro.
3. Combinacion de los dos anteriores.
4. Espacios en blanco.
5. Comentarios entre los simbolos /* y */

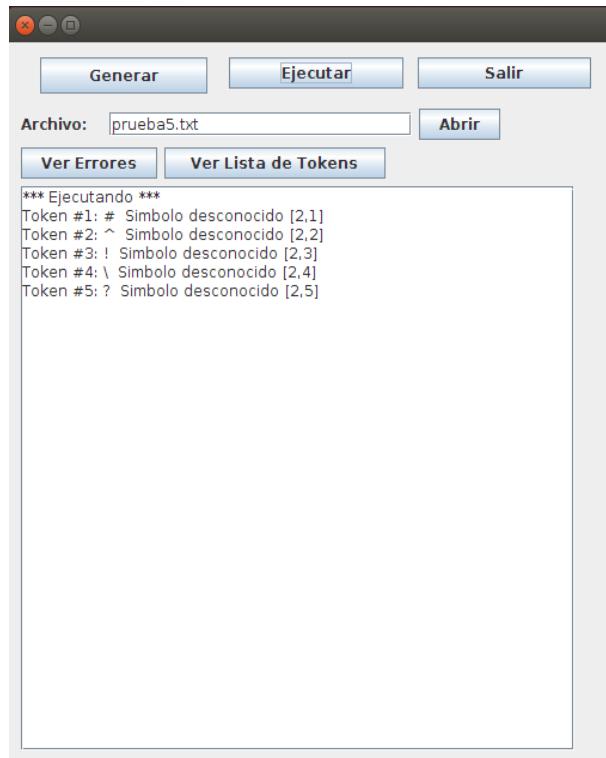
3.8 Análisis de Resultados. Parte 1:

- Prueba realizada sin errores léxicos (Ver Apéndice 2, Prueba 1)

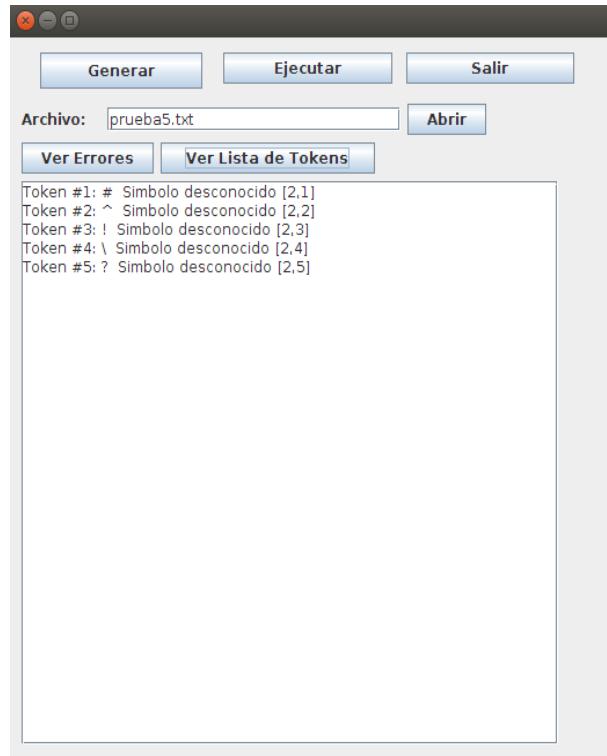
```
Token #1: my MY [1,1]
Token #2: $ DOLAR [1,4]
Token #3: variable1 IDENTIFICADOR [1,5]
Token #4: = ASIGNACION [1,15]
Token #5: 1 NUMERO [1,17]
Token #6: ; PUNTOCOMA [1,18]
Token #7: my MY [2,1]
Token #8: $ DOLAR [2,4]
Token #9: variable2 IDENTIFICADOR [2,5]
Token #10: = ASIGNACION [2,15]
Token #11: 2 NUMERO [2,17]
Token #12: ; PUNTOCOMA [2,18]
Token #13: if CONDICIONIF [3,1]
Token #14: ( OPENPARENTESIS [3,3]
Token #15: $ DOLAR [3,4]
Token #16: variable1 IDENTIFICADOR [3,5]
Token #17: > MAYORQUE [3,14]
Token #18: variable2 IDENTIFICADOR [3,15]
Token #19: ) CLOSEPARENTESIS [3,24]
Token #20: { OPENLLAVES [4,1]
Token #21: my MY [5,5]
Token #22: $ DOLAR [5,8]
Token #23: contador IDENTIFICADOR [5,9]
Token #24: = ASIGNACION [5,18]
Token #25: 0 NUMERO [5,20]
Token #26: ; PUNTOCOMA [5,21]
Token #27: while BUCLEWHILE [6,5]
Token #28: ( OPENPARENTESIS [6,10]
Token #29: contador IDENTIFICADOR [6,11]
Token #30: > MAYORQUE [6,19]
```

- Prueba con el primer archivo con errores léxicos (Ver Apéndice 2, Prueba 5)

Ver Errores

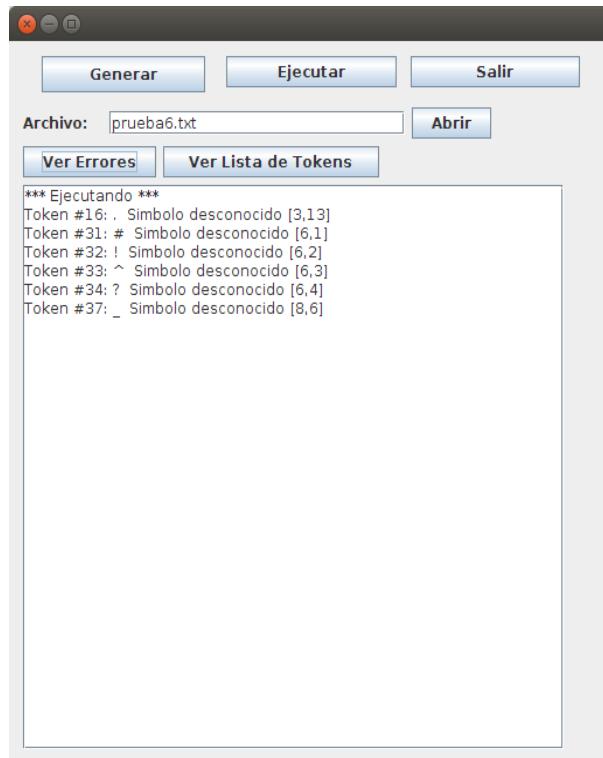


Ver Lista de Tokens

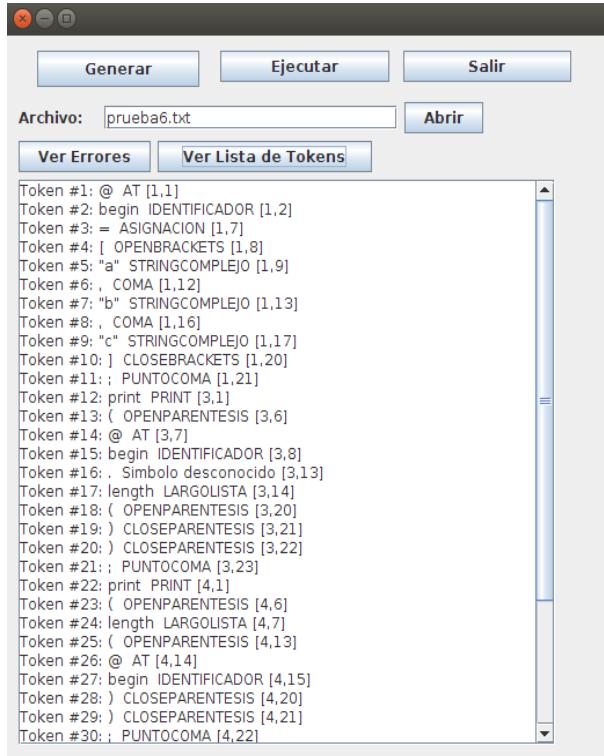


- Prueba con el segundo archivo con errores léxicos (Ver Apéndice 2, Prueba 6)

Ver Errores



Ver Lista de Tokens



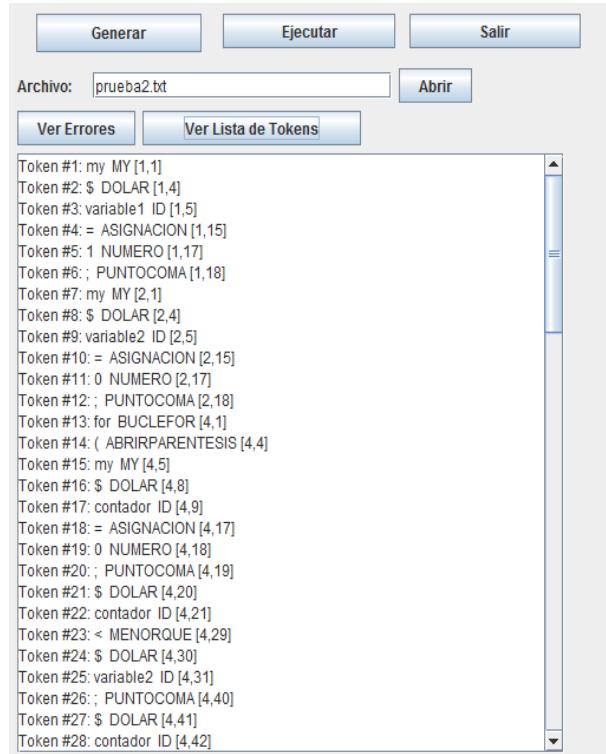
The screenshot shows a software interface for viewing token lists. At the top, there are three buttons: 'Generar' (Generate), 'Ejecutar' (Execute), and 'Salir' (Exit). Below these is a text input field labeled 'Archivo:' containing the value 'prueba6.txt'. To the right of the input field is a 'Abrir' (Open) button. Underneath the input field are two buttons: 'Ver Errores' (View Errors) and 'Ver Lista de Tokens' (View Token List), with 'Ver Lista de Tokens' being the active one. The main area displays a scrollable list of tokens:

```
Token #1: @ AT [1,1]
Token #2: begin IDENTIFICADOR [1,2]
Token #3: = ASIGNACION [1,7]
Token #4: [ OPENBRACKETS [1,8]
Token #5: "a" STRINGCOMPLEJO [1,9]
Token #6: , COMA [1,12]
Token #7: "b" STRINGCOMPLEJO [1,13]
Token #8: , COMA [1,16]
Token #9: "c" STRINGCOMPLEJO [1,17]
Token #10: ] CLOSEBRACKETS [1,20]
Token #11: ; PUNTOCOMA [1,21]
Token #12: print PRINT [3,1]
Token #13: ( OPENPARENTESIS [3,6]
Token #14: @ AT [3,7]
Token #15: begin IDENTIFICADOR [3,8]
Token #16: . Simbolo desconocido [3,13]
Token #17: length LARGOLISTA [3,14]
Token #18: ( OPENPARENTESIS [3,20]
Token #19: ) CLOSEPARENTESIS [3,21]
Token #20: ) CLOSEPARENTESIS [3,22]
Token #21: ; PUNTOCOMA [3,23]
Token #22: print PRINT [4,1]
Token #23: ( OPENPARENTESIS [4,6]
Token #24: length LARGOLISTA [4,7]
Token #25: ( OPENPARENTESIS [4,13]
Token #26: @ AT [4,14]
Token #27: begin IDENTIFICADOR [4,15]
Token #28: ) CLOSEPARENTESIS [4,20]
Token #29: ) CLOSEPARENTESIS [4,21]
Token #30: ; PUNTOCOMA [4,22]
```

4 Desarrollo Parte 2

4.1 Análisis de Resultados. Parte 2:

- Prueba realizada sin errores sintácticos (Ver Apéndice 2, Prueba 2)



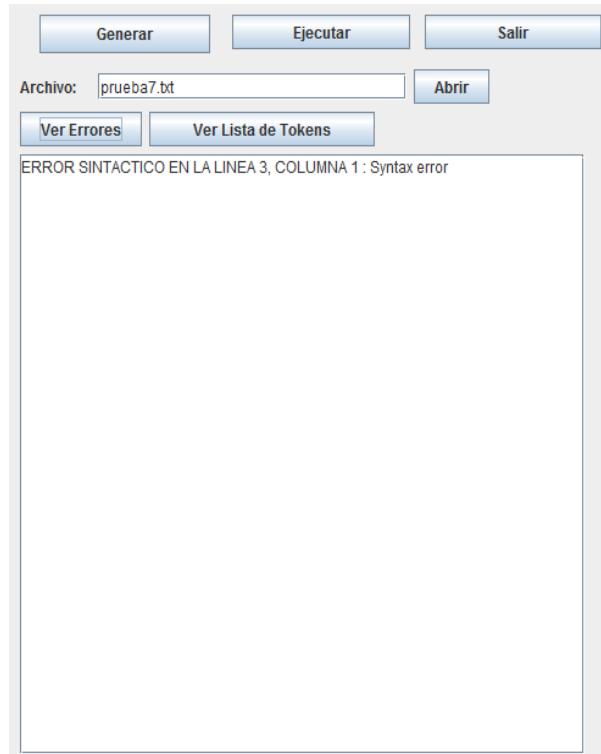
The screenshot shows a software window with the following interface elements:

- Buttons at the top: Generar, Ejecutar, Salir.
- A text input field labeled "Archivo:" containing "prueba2.txt".
- A button labeled "Abrir" next to the file input field.
- Two buttons below the file input: "Ver Errores" and "Ver Lista de Tokens".
- A scrollable list area displaying tokens:

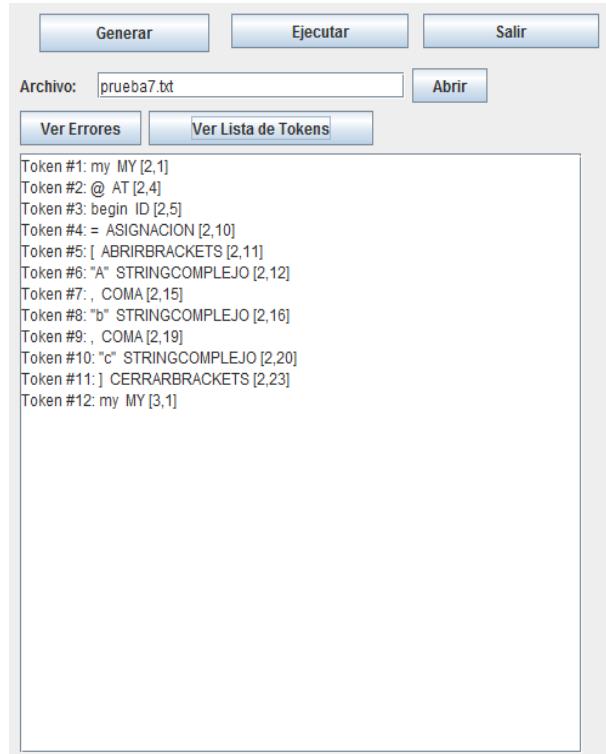
```
Token #1: my MY [1,1]
Token #2: $ DOLAR [1,4]
Token #3: variable1 ID [1,5]
Token #4: = ASIGNACION [1,15]
Token #5: 1 NUMERO [1,17]
Token #6: ; PUNTOCOMA [1,18]
Token #7: my MY [2,1]
Token #8: $ DOLAR [2,4]
Token #9: variable2 ID [2,5]
Token #10: = ASIGNACION [2,15]
Token #11: 0 NUMERO [2,17]
Token #12: ; PUNTOCOMA [2,18]
Token #13: for BUCLEFOR [4,1]
Token #14: ( ABRIRPARENTESIS [4,4]
Token #15: my MY [4,5]
Token #16: $ DOLAR [4,8]
Token #17: contador ID [4,9]
Token #18: = ASIGNACION [4,17]
Token #19: NUMERO [4,18]
Token #20: ; PUNTOCOMA [4,19]
Token #21: $ DOLAR [4,20]
Token #22: contador ID [4,21]
Token #23: < MENORQUE [4,29]
Token #24: $ DOLAR [4,30]
Token #25: variable2 ID [4,31]
Token #26: ; PUNTOCOMA [4,40]
Token #27: $ DOLAR [4,41]
Token #28: contador ID [4,42]
```

- Prueba con el primer archivo con errores sintácticos (Ver Apéndice 2, Prueba 7)

Ver Errores

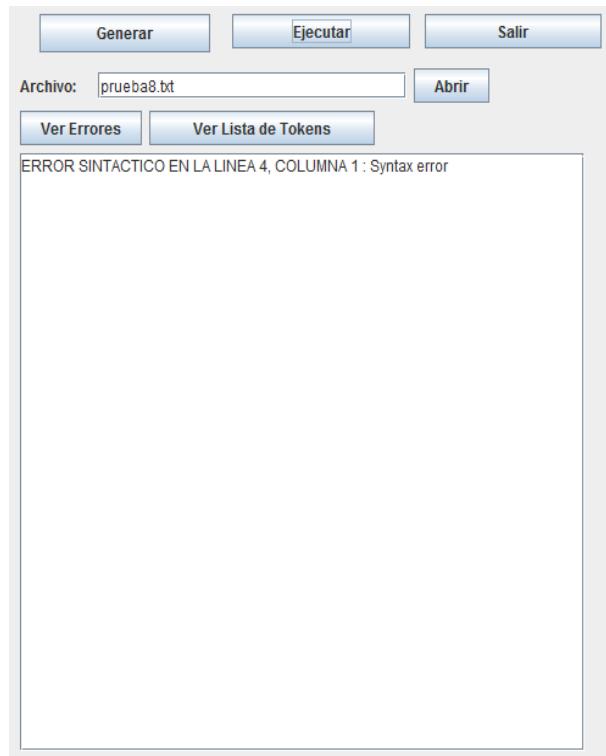


Ver Lista de Tokens

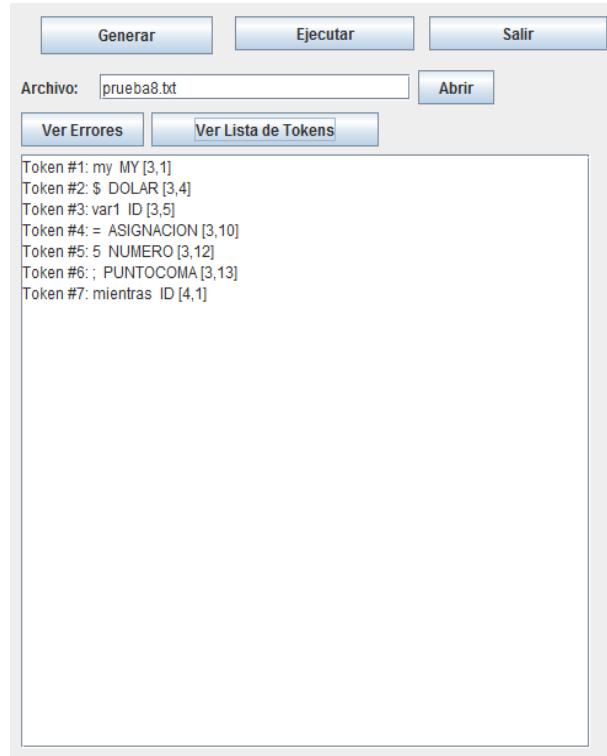


- Prueba con el segundo archivo con errores sintácticos (Ver Apéndice 2, Prueba 8)

Ver Errores



Ver Lista de Tokens



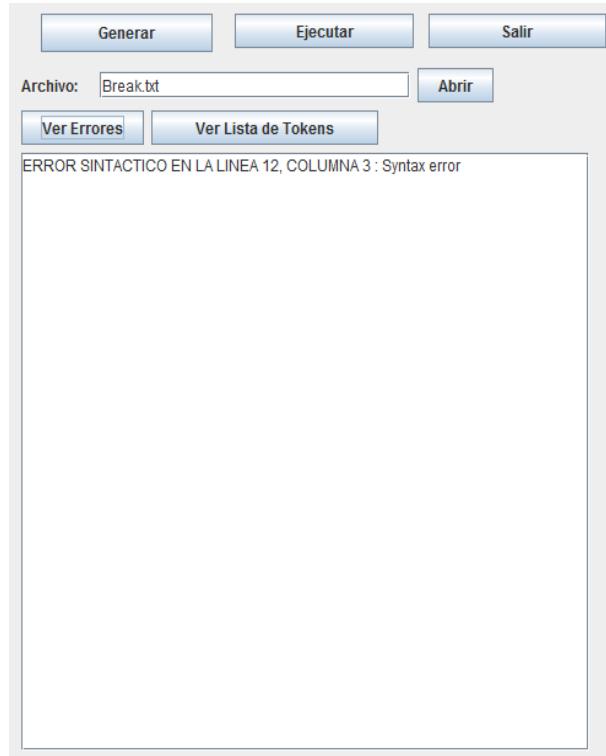
4.2 Errores Sintácticos

1. Break

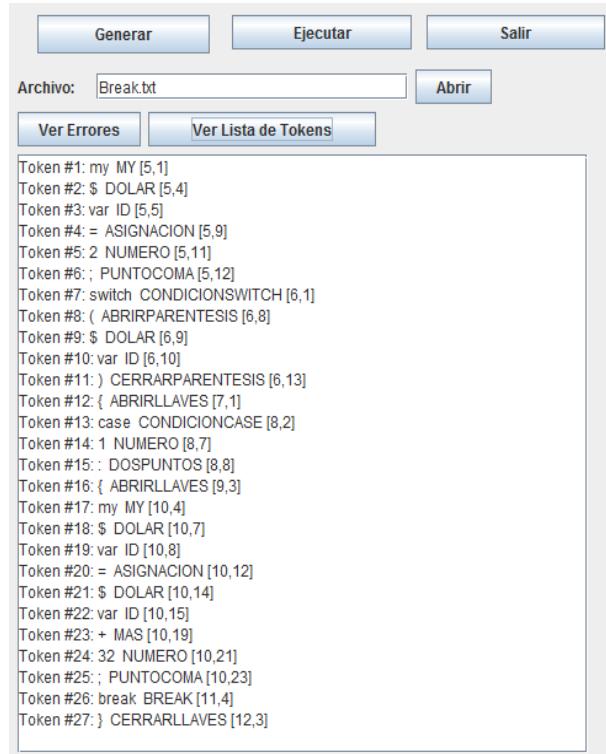
```
/* La estructura del break es : */
/* break; */
/* En este caso falta el ; para completar la estructura. */

my $var = 2;
switch ($var)
{
    case 1:
    {
        my $var = $var + 32;
        break
    }
    case 0:
    {
        my $var = $var + 16;
        break;
    }
    default:
    {
        break;
    }
}
```

Ver Errores



Ver Lista de Tokens



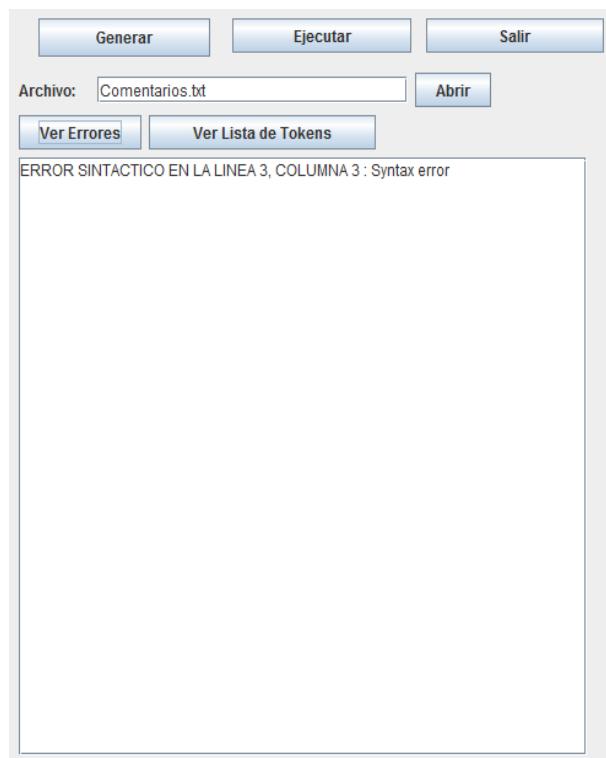
Versión Correcta

```
my $var = 2;
switch ($var)
{
    case 1:
    {
        my $var = $var + 32;
        break;
    }
    case 0:
    {
        my $var = $var + 16;
        break;
    }
    default:
    {
        break;
    }
}
```

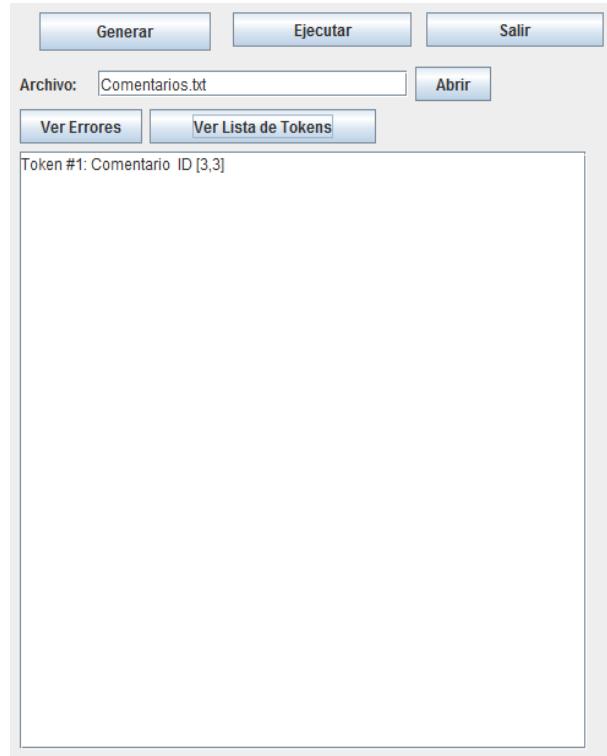
2. Comentarios

```
/*En este caso falta el elemento de cierre de comentario.*/  
/*Comentario
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

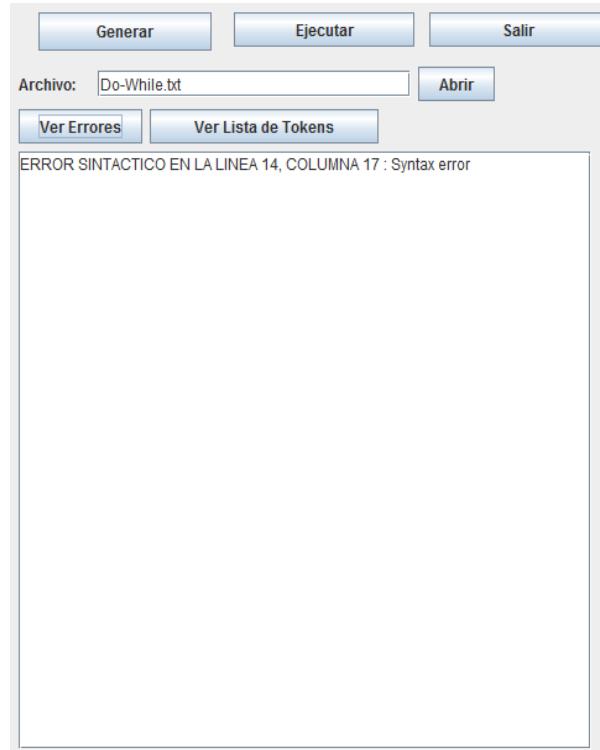
```
/*En este caso falta el elemento de cierre de comentario.*/
/*Comentario*/
```

3. Do-While

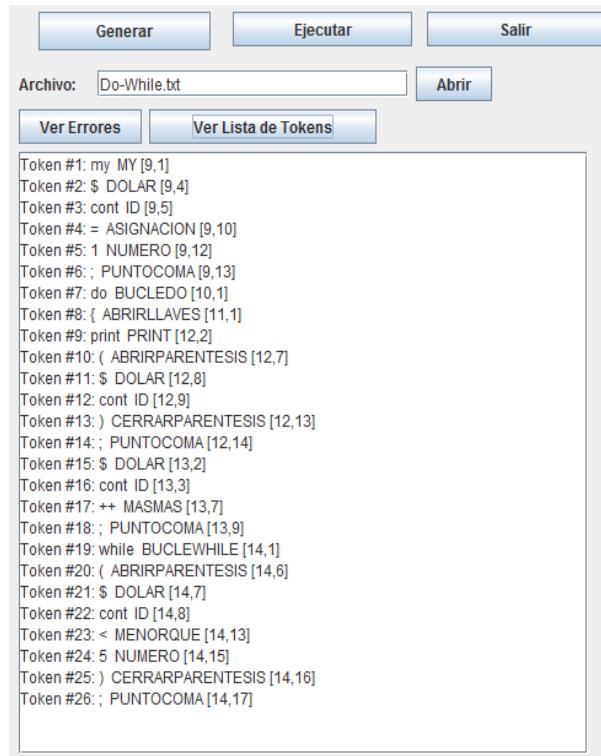
```
/* La estructura del do-while es : */
/* do */
/* { */
/* lista_declaraciones */
/* } while (expresion_logica); */
/* En este caso falta el while esta dentro de las */
/* llaves, por lo tanto incumple la estructura. */

my $cont = 1;
do
{
    print($cont);
    $cont++;
while($cont < 5);
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

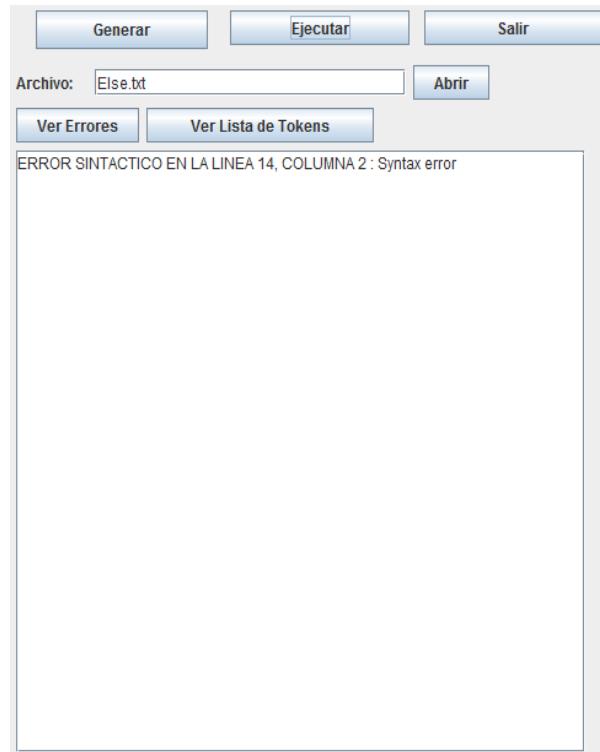
```
my $cont = 1;
do
{
    print($cont);
    $cont++;
} while($cont < 5);
```

4. Else

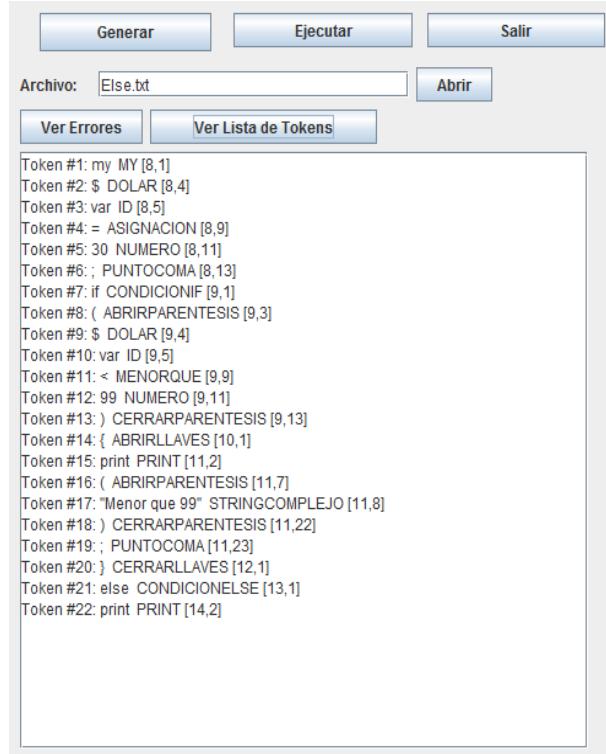
```
/* La estructura del else es : */
/* else */
/* { */
/* lista_declaraciones */
/* } */
/* En este caso falta el { para completar la estructura. */

my $var = 30;
if($var < 99)
{
    print("Menor que 99");
}
else
    print("Mayor que 99");
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

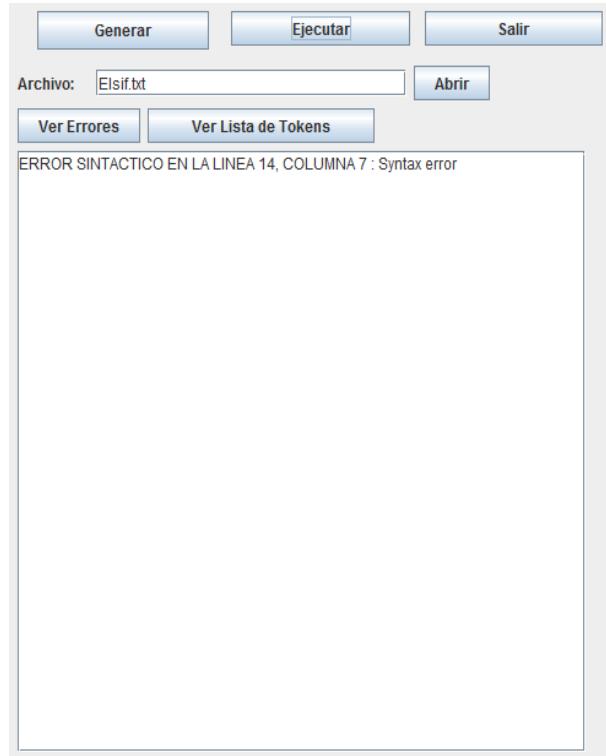
```
my $var = 30;
if($var < 99)
{
    print("Menor que 99");
}
else
{
    print("Mayor que 99");
}
```

5. Elsif

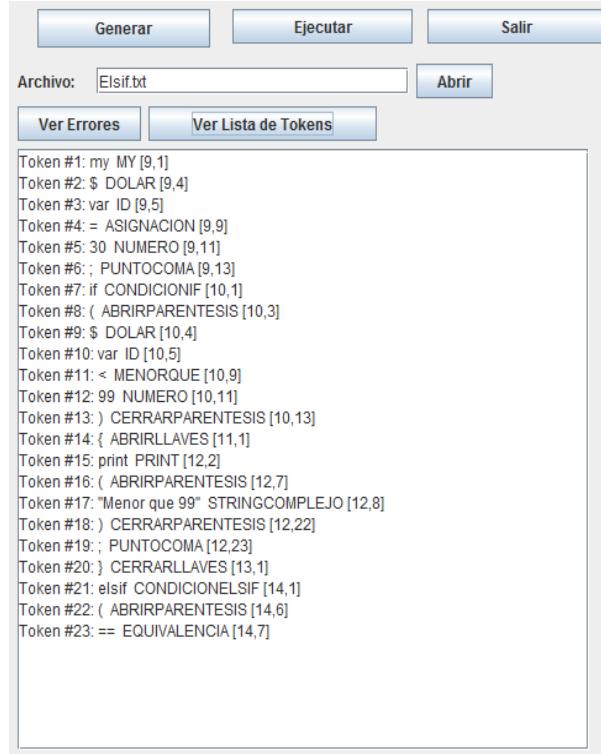
```
/* La estructura del elsif es : */
/* elsif (expresion) */
/* { */
/* lista_declaraciones */
/* } */
/* En este caso ==99 no es una expresión válida. */
/* Por lo tanto, se incumple la estructura. */

my $var = 30;
if($var < 99)
{
    print("Menor que 99");
}
elsif(==99)
{
    print("Igual a 99");
}
else
{
    print("Mayor que 99");
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

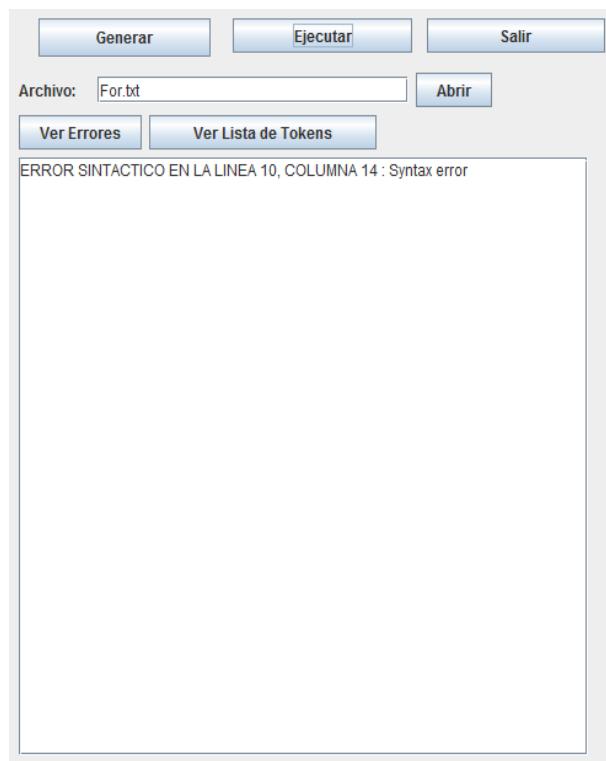
```
my $var = 30;
if($var < 99)
{
    print("Menor que 99");
}
elsif($var == 99)
{
    print("Igual a 99");
}
else
{
    print("Mayor que 99");
}
```

6. For

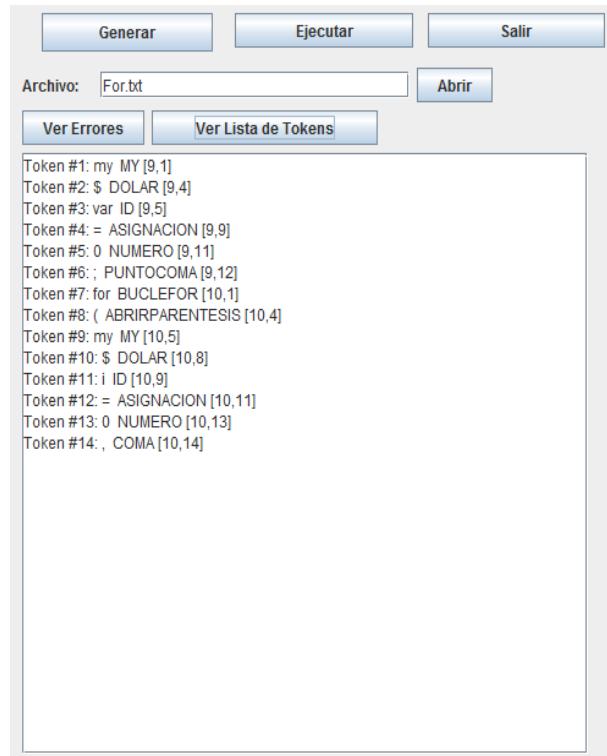
```
/* La estructura del for es : */
/* for (declaracion_variable;expresion_logica;incrementador) */
/* { */
/* lista_declaraciones */
/* } */
/* En este caso hay una coma en vez de punto y coma luego */
/* de la declaración, por lo tanto se incumple la estructura. */

my $var = 0;
for(my $i = 0,$i < 30;$i++)
{
    my $var = $var + $i;
}
print($var);
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

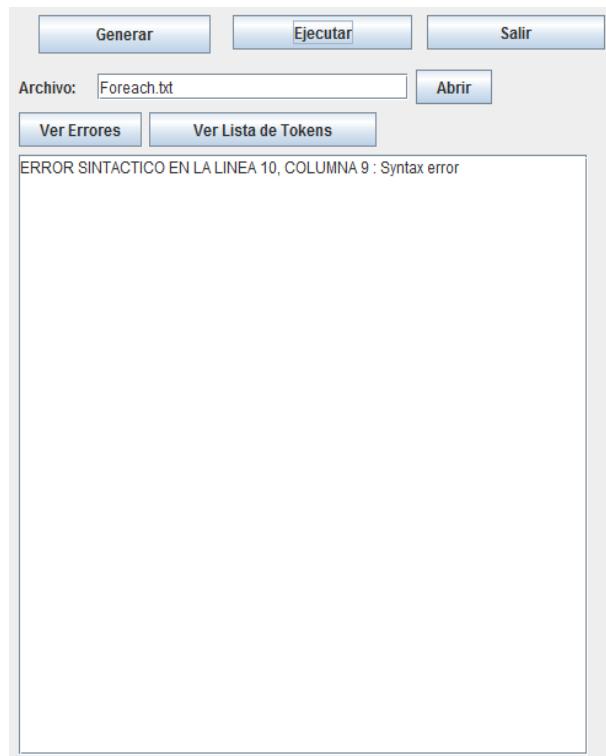
```
my $var = 0;
for(my $i = 0;$i < 30;$i++)
{
    my $var = $var + $i;
}
print($var);
```

7. Foreach

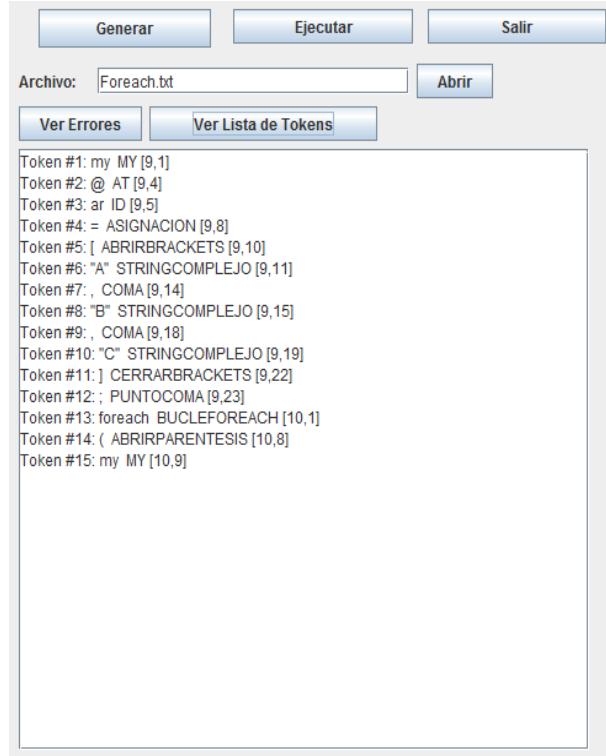
```
/* La estructura del foreach es : */
/* foreach (@ ID) */
/* { */
/* lista_declaraciones */
/* } */
/* En este caso se utiliza el foreach como si fuera un for */
/* Por lo tanto se incumple la estructura. */

my @ar = ["A","B","C"];
foreach(my $i = 0; $i<length(@ar); $i++)
{
    print(shift(@ar));
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

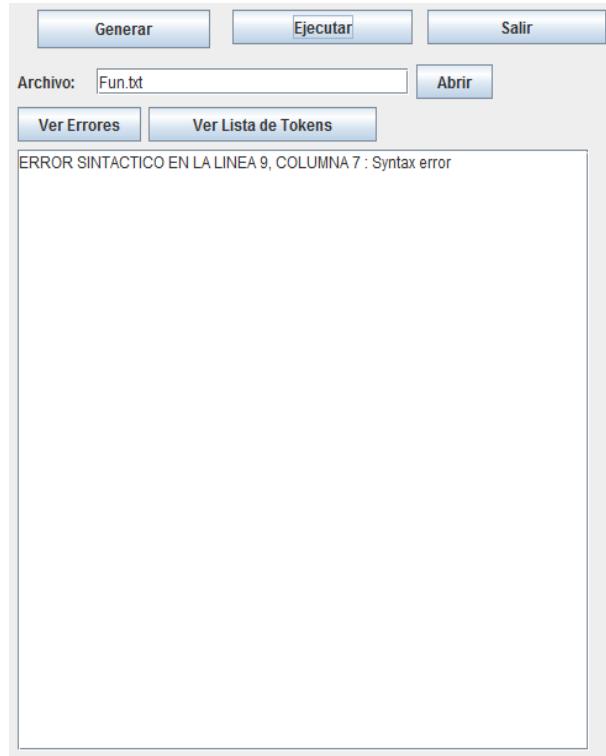
```
my @ar = ["A","B","C"];
foreach(@ar)
{
    print(shift(@ar));
}
```

8. Fun

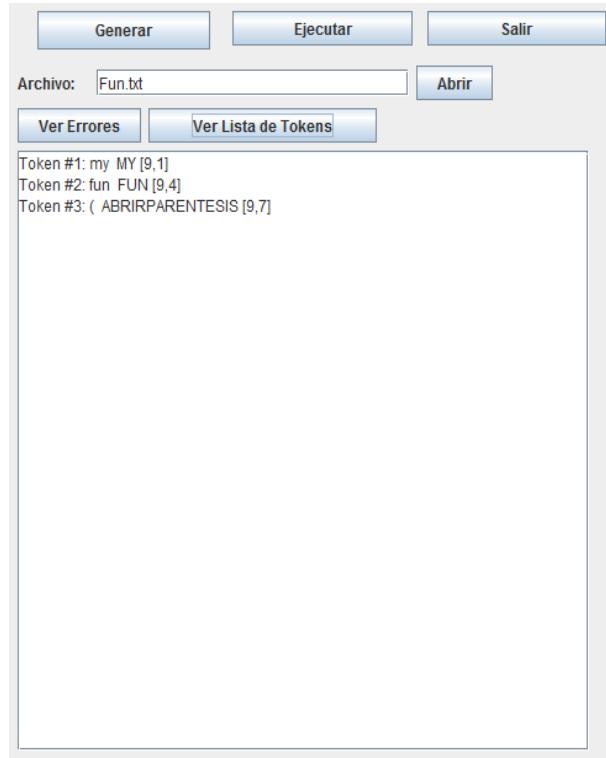
```
/* La estructura del fun es : */
/* my fun ID(parametros) */
/* { */
/* lista_declaraciones */
/* } */
/* En este caso falta el ID o nombre de la funcion */
/* para completar la estructura. */

my fun($param1, $param2)
{
    if ($param1 > $param2)
    {
        print($param1 * 20);
    }
    else
    {
        print($param2 * 10);
    }
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

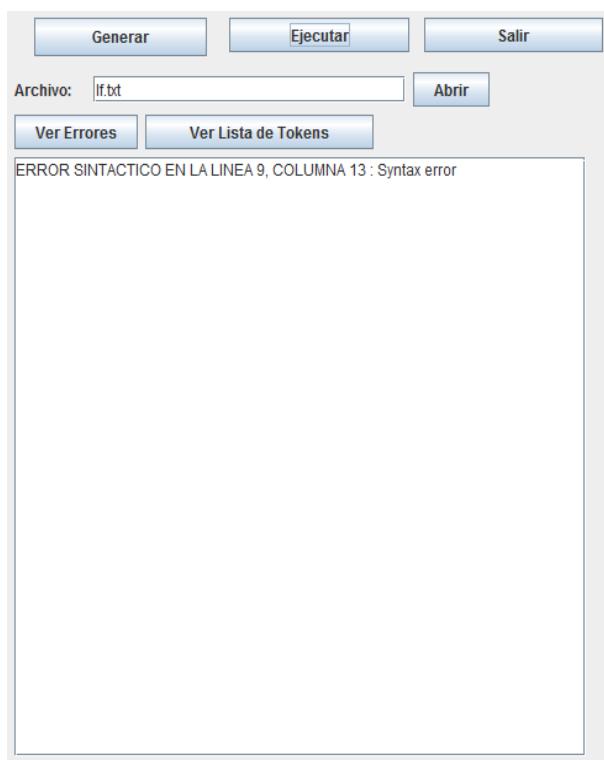
```
my fun funx($param1, $param2)
{
    if ($param1 > $param2)
    {
        print($param1 * 20);
    }
    else
    {
        print($param2 * 10);
    }
}
```

9. If

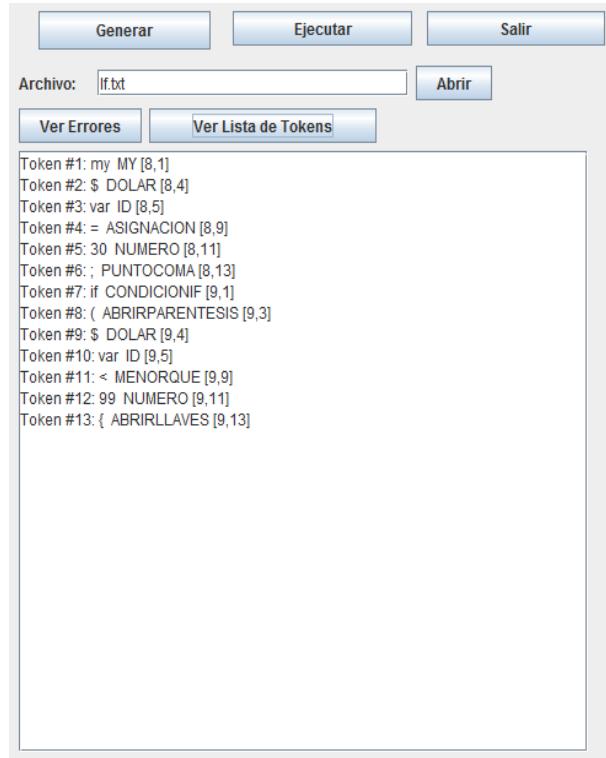
```
/* La estructura del if es : */
/* if (expresion) */
/* { */
/* lista_declaraciones */
/* } */
/* En este caso falta el ) para cerrar la estructura. */

my $var = 30;
if($var < 99{
    print("El número es menor que 99");
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

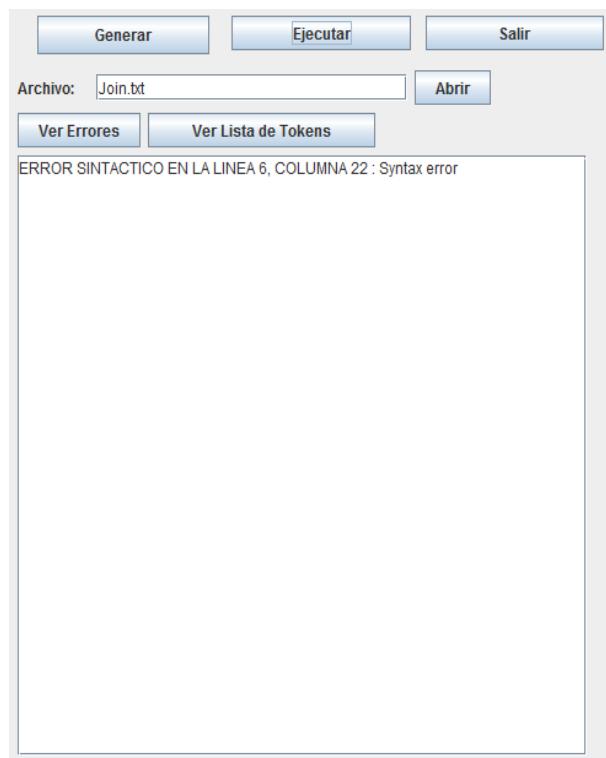
```
my $var = 30;
if($var < 99)
{
    print("El número es menor que 99");
}
```

10. Join

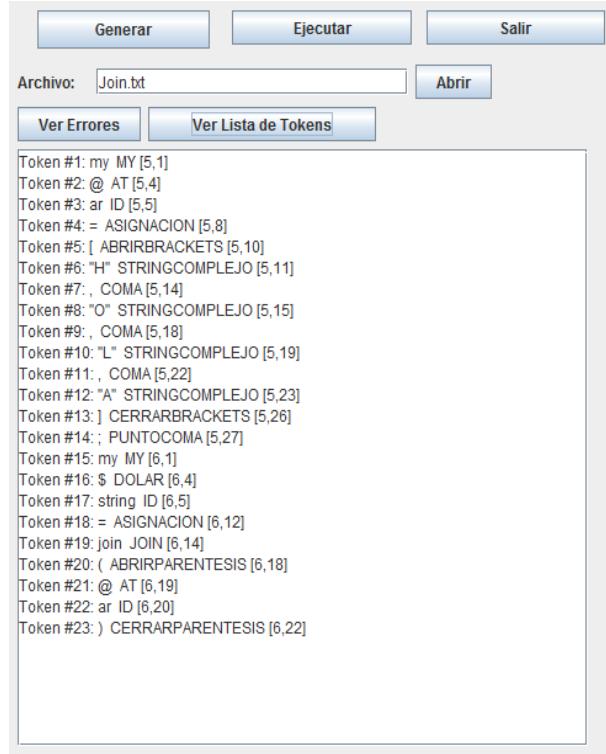
```
/* La estructura del join es : */
/* join (union, @ ID) */
/* En este caso falta el elemento de unión para completar la estructura. */

my @ar = ["H","O","L","A"];
my $string = join(@ar);
print($string);
```

Ver Errores



Ver Lista de Tokens



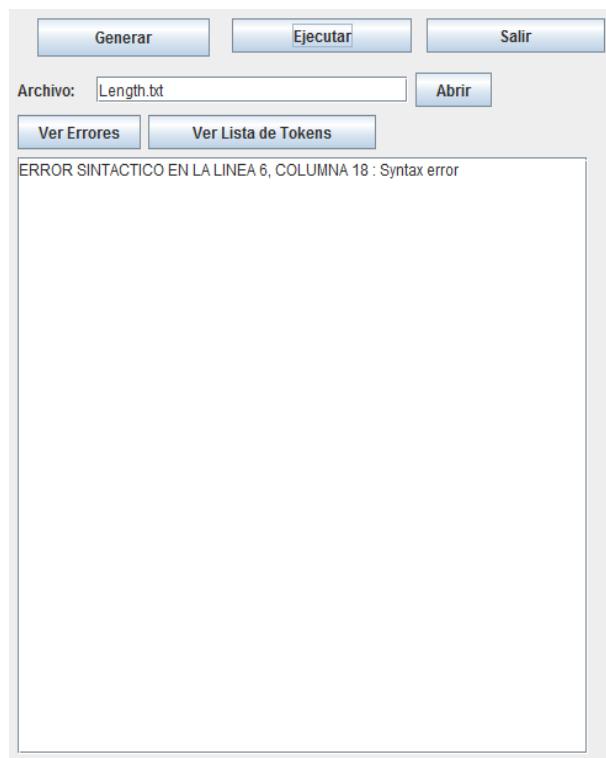
Versión Correcta

```
my @ar = ["H","O","L","A"];
my $string = join('', @ar);
print($string);
```

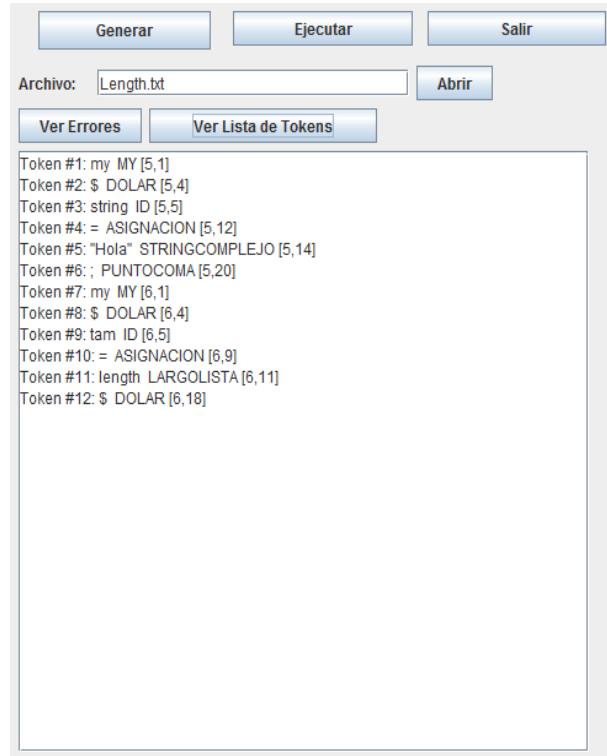
11. Length

```
/* La estructura del length es : */  
/* length(@ ID); ó length($ ID); */  
/* En este caso faltan los paréntesis para completar la estructura. */  
  
my $string = "Hola";  
my $tam = length $string;  
print($tam);
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

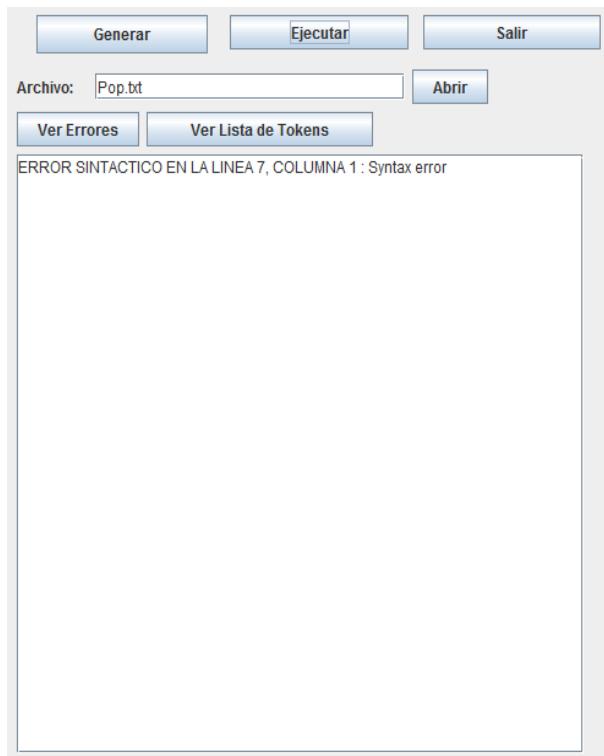
```
my $string = "Hola";
my $tam = length($string);
print($tam);
```

12. Pop

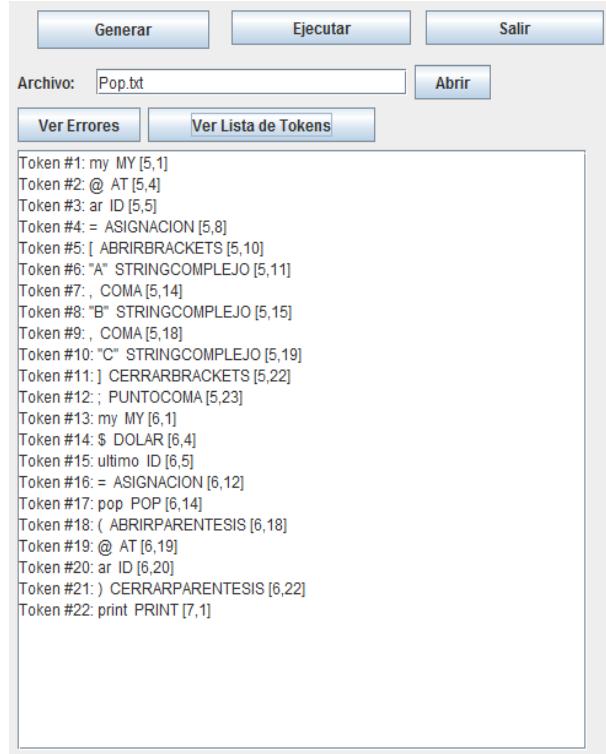
```
/* La estructura del pop es : */
/* pop(@ ID); */
/* En este caso falta el ; para completar la estructura. */

my @ar = ["A","B","C"];
my $ultimo = pop (@ar)
print ($ultimo);
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

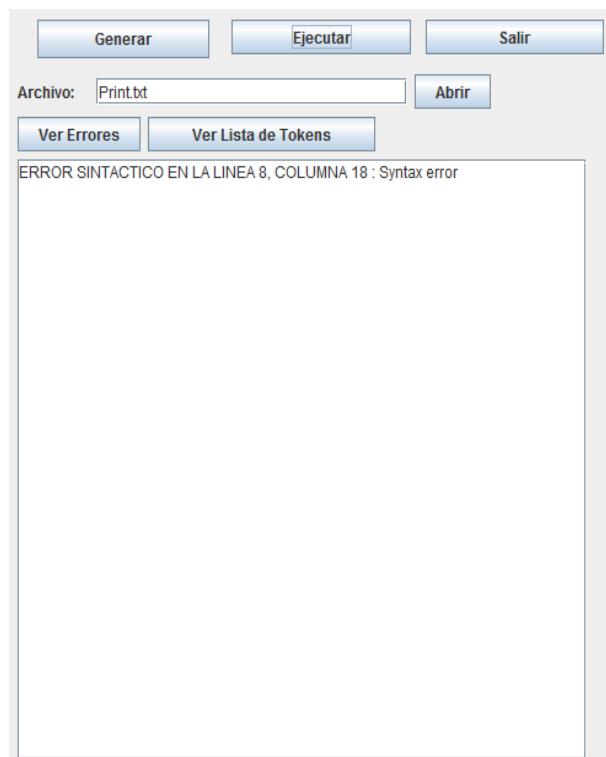
```
my @ar = ["A","B","C"];
my $ultimo = pop(@ar);
print ($ultimo);
```

13. Print

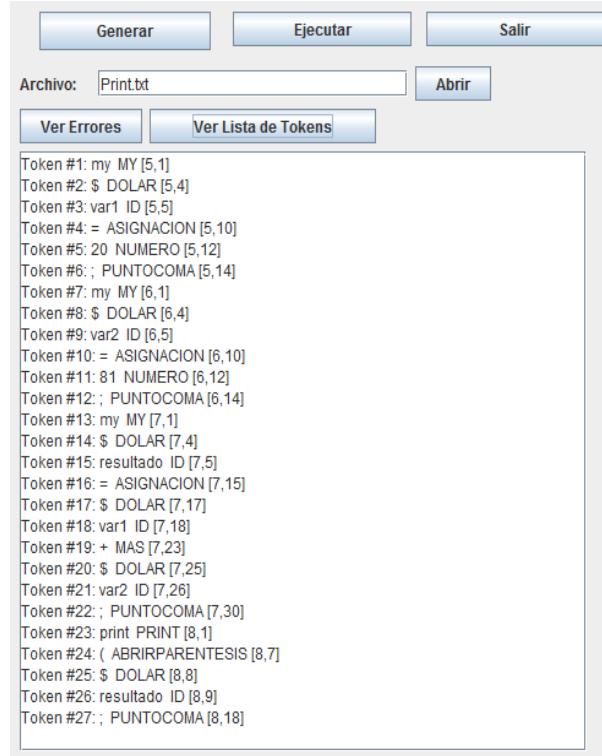
```
/* La estructura del print es : */
/* print(expresión); */
/* En este caso falta el ) para cerrar la estructura. */

my $var1 = 20;
my $var2 = 81;
my $resultado = $var1 + $var2;
print ($resultado);
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

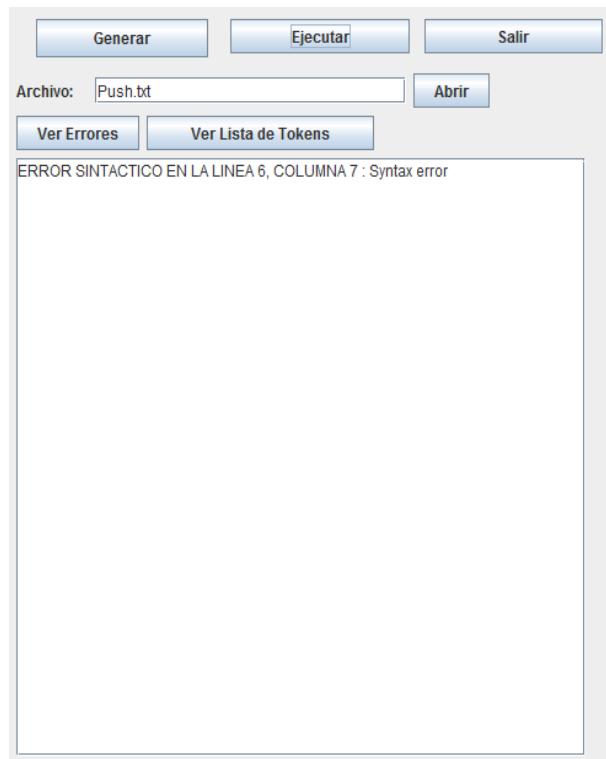
```
my $var1 = 20;
my $var2 = 81;
my $resultado = $var1 + $var2;
print ($resultado);
```

14. Push

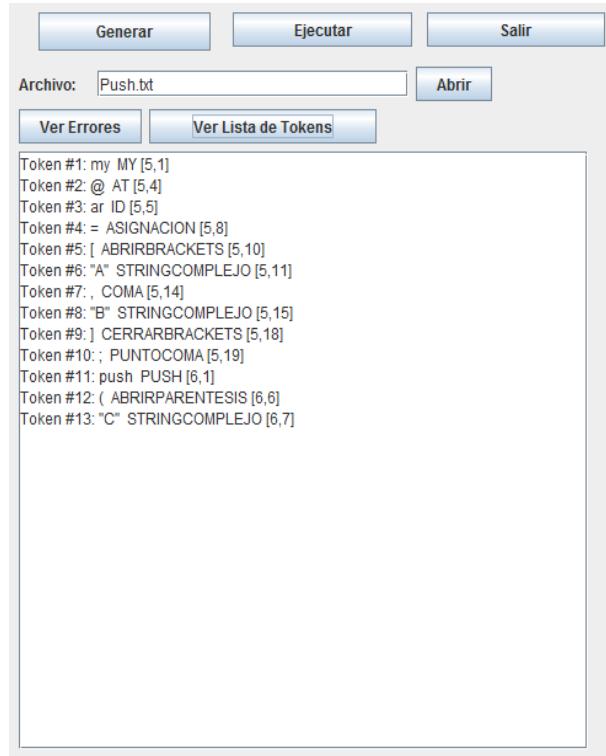
```
/* La estructura del push es : */
/* push(@ ID, expresion); */
/* En este caso falta el arreglo para completar la estructura. */

my @ar = ["A","B"];
push ("C");
foreach(@ar)
{
    print(shift(@ar));
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

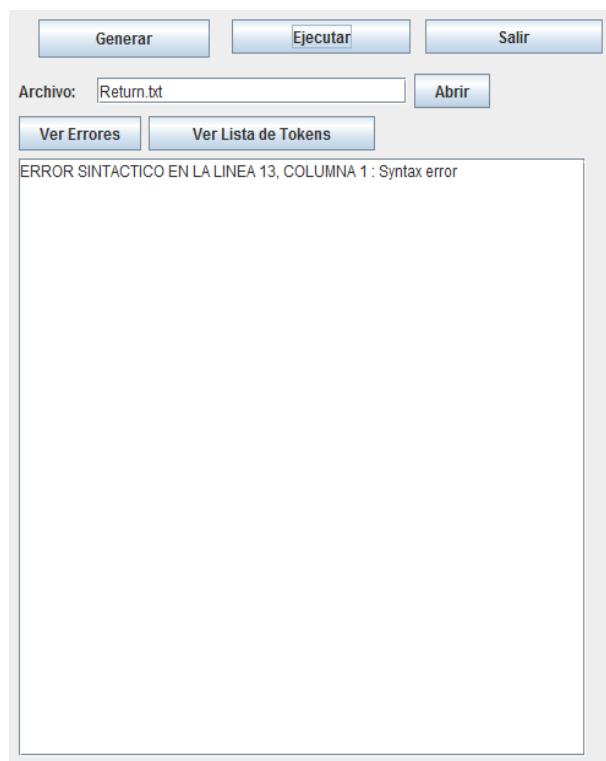
```
my @ar = ["A","B"];
push (@ar, "C");
foreach(@ar)
{
    print(shift(@ar));
}
```

15. Return

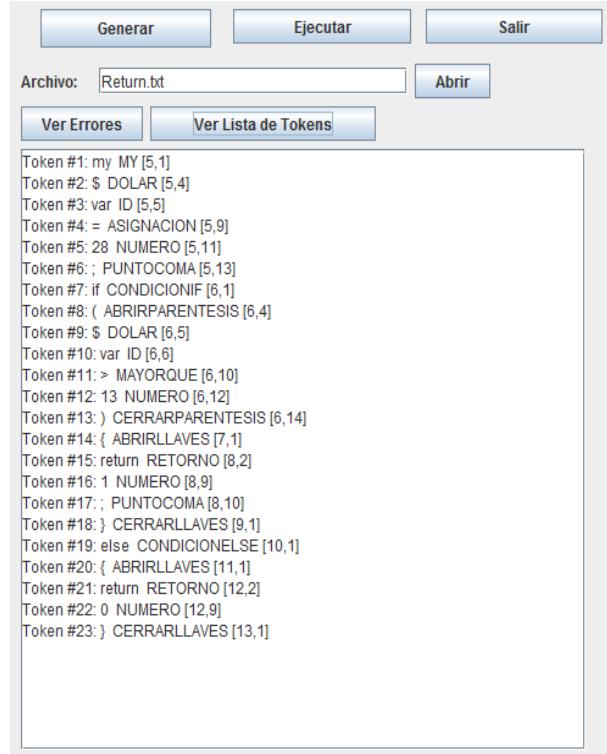
```
/* La estructura del return es : */
/* return expresion; */
/* En este caso falta el ; para completar la estructura. */

my $var = 28;
if ($var > 13)
{
    return 1;
}
else
{
    return 0
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

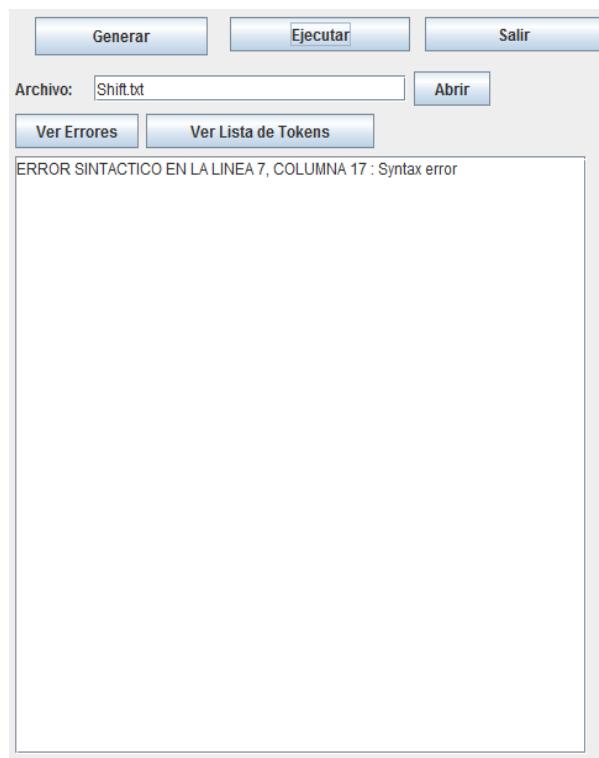
```
my $var = 28;
if ($var < 13)
{
    return 1;
}
else
{
    return 0;
}
```

16. Shift

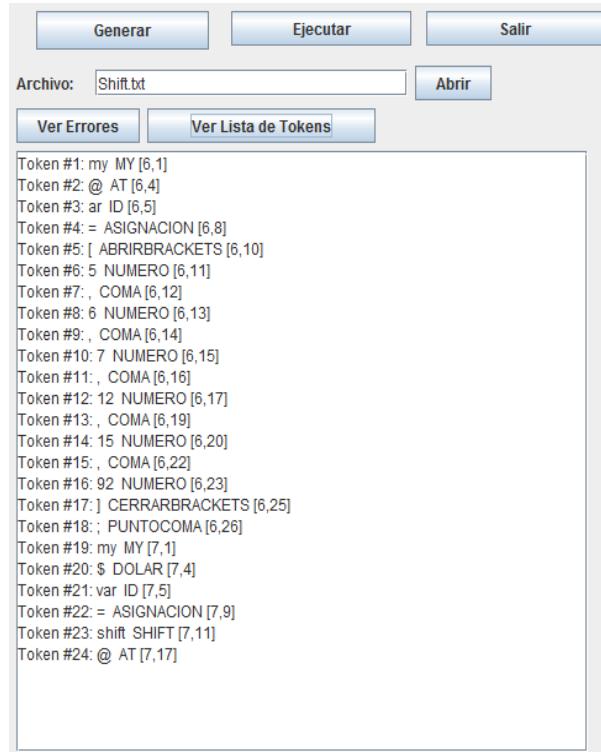
```
/* La estructura del shift es : */
/* shift(@ ID); */
/* En este caso faltan los paréntesis para */
/* completar la estructura. */

my @ar = [5,6,7,12,15,92];
my $var = shift @ar;
print($var);
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

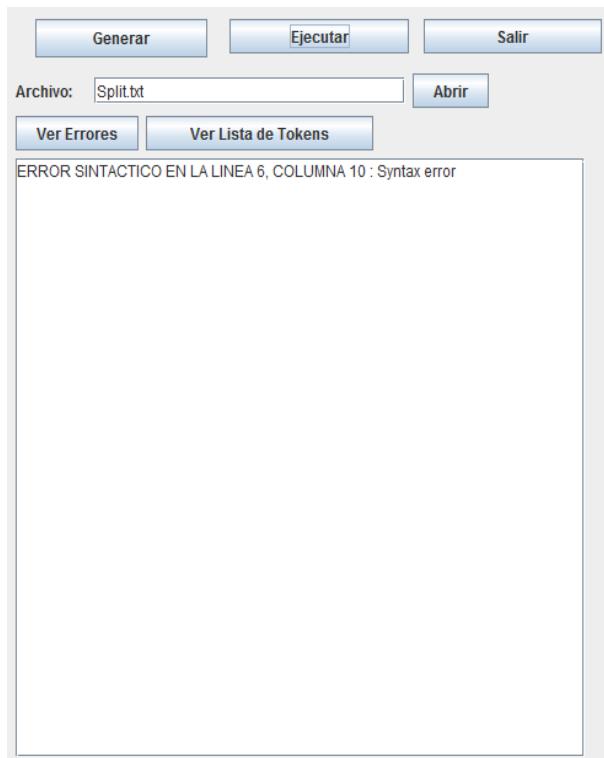
```
my @ar = [5,6,7,12,15,92];
my $var = shift(@ar);
print($var);
```

17. Split

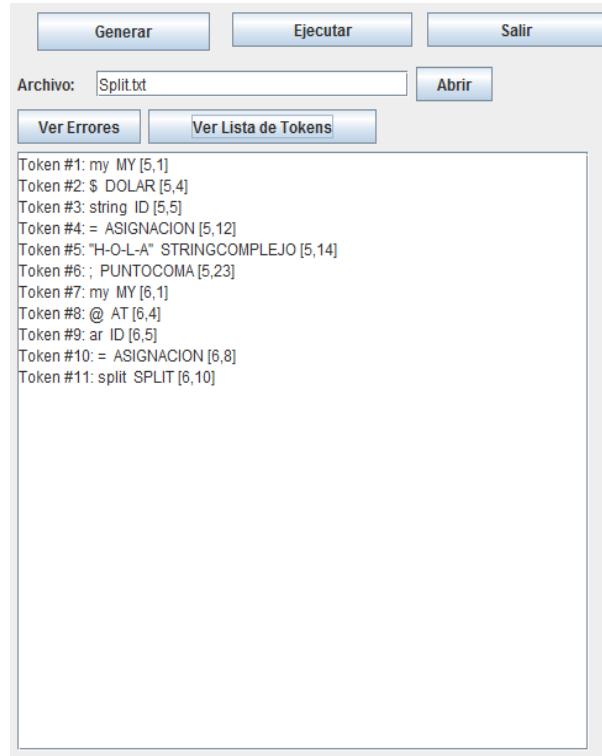
```
/* La estructura del split es : */
/* split(patron, $ ID); */
/* En este caso falta la cadena para completar la estructura. */

my $string = "H-O-L-A";
my @ar = split("-");
foreach(@ar)
{
    print(shift(@ar));
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

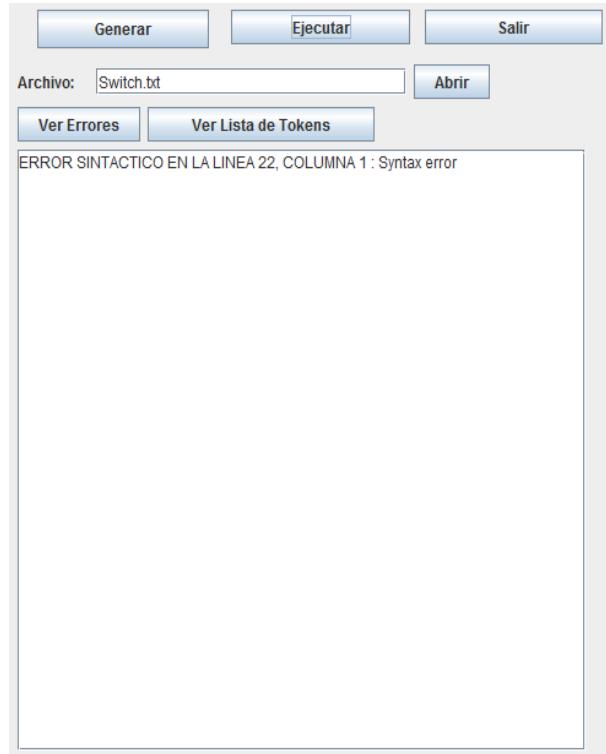
```
my $string = "H-O-L-A";
my @ar = split("-", $string);
foreach(@ar)
{
    print(shift(@ar));
}
```

18. Switch

```
/* La estructura del switch es : */
/* switch (expresion_simple) */
/* { */
/* expresion_case */
/* expresion_default */
/* } */
/* En este caso falta la expresión default. */

my $var = 2;
switch ($var)
{
    case 1:
    {
        my $var = $var + 32;
        break;
    }
    case 0:
    {
        my $var = $var + 16;
        break;
    }
}
```

Ver Errores



Ver Lista de Tokens

The screenshot shows a software window titled "Ver Lista de Tokens". At the top, there are three buttons: "Generar", "Ejecutar", and "Salir". Below these is a text input field labeled "Archivo:" containing the value "Switch.txt". To the right of the input field is a "Abrir" button. Underneath the input field are two tabs: "Ver Errores" and "Ver Lista de Tokens", with "Ver Lista de Tokens" being the active tab. The main area of the window displays a scrollable list of tokens. The tokens listed are:

```
Token #19: var ID [14,8]
Token #20: = ASIGNACION [14,12]
Token #21: $ DOLAR [14,14]
Token #22: var ID [14,15]
Token #23: + MAS [14,19]
Token #24: 32 NUMERO [14,21]
Token #25: ; PUNTOCOMA [14,23]
Token #26: break BREAK [15,4]
Token #27: ; PUNTOCOMA [15,9]
Token #28: } CERRARLLAVES [16,3]
Token #29: case CONDICIONCASE [17,2]
Token #30: 0 NUMERO [17,7]
Token #31: : DOSPUNTOS [17,8]
Token #32: { ABRIRLLAVES [18,3]
Token #33: my MY [19,4]
Token #34: $ DOLAR [19,7]
Token #35: var ID [19,8]
Token #36: = ASIGNACION [19,12]
Token #37: $ DOLAR [19,14]
Token #38: var ID [19,15]
Token #39: + MAS [19,19]
Token #40: 16 NUMERO [19,21]
Token #41: ; PUNTOCOMA [19,23]
Token #42: break BREAK [20,4]
Token #43: ; PUNTOCOMA [20,9]
Token #44: } CERRARLLAVES [21,3]
Token #45: } CERRARLLAVES [22,1]
```

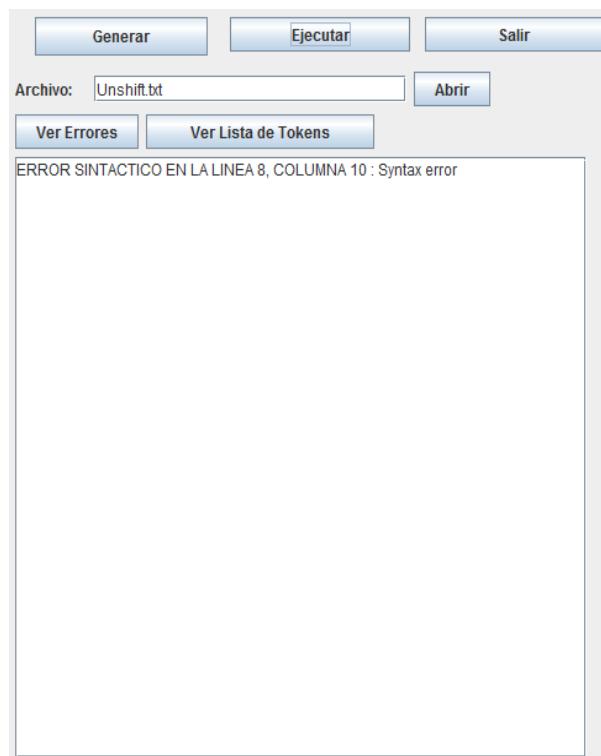
Versión Correcta

```
my $var = 2;
switch ($var)
{
    case 1:
    {
        my $var = $var + 32;
        break;
    }
    case 0:
    {
        my $var = $var + 16;
        break;
    }
    default:
    {
        break;
    }
}
```

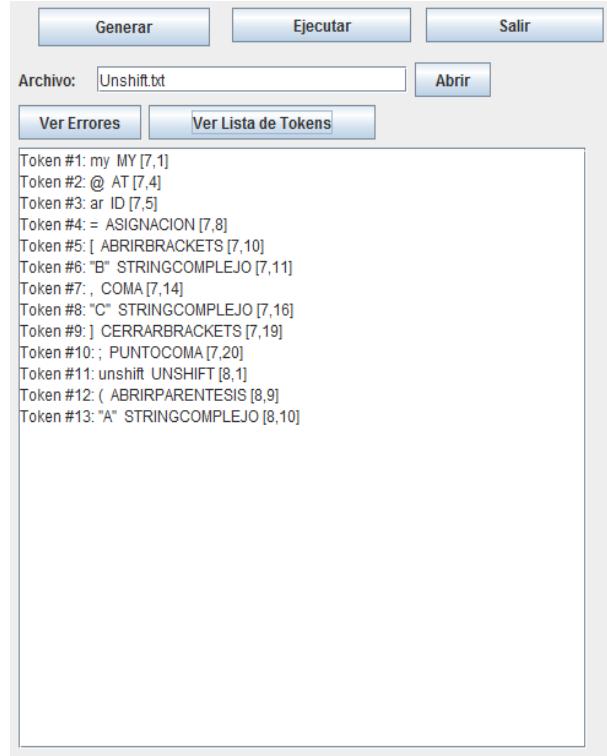
19. Unshift

```
/* La estructura del unshift es : */  
/* unshift(@ ID, expresion); */  
/* En este caso la expresión y el arreglo */  
/* están al revés, por lo tanto esto */  
/* incumple la estructura. */  
  
my @ar = ["B", "C"];  
unshift ("A", @ar);  
foreach(@ar)  
{  
    print (shift(@ar));  
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

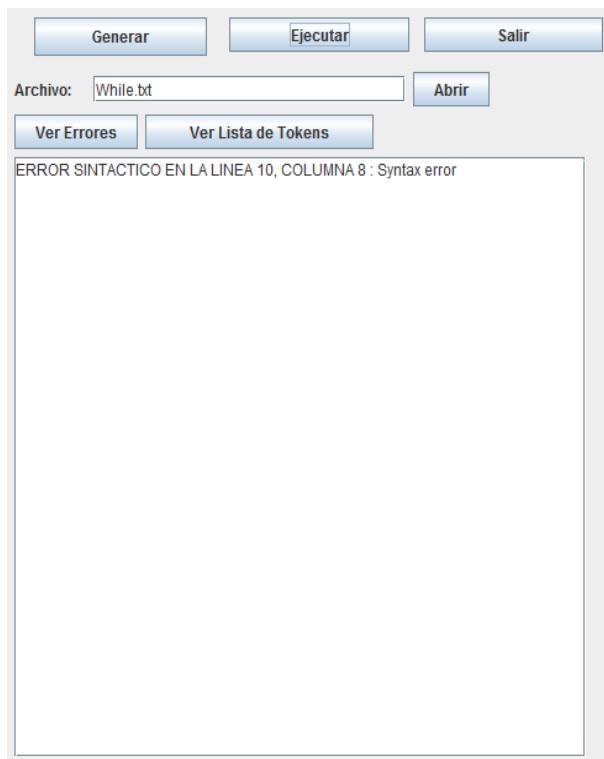
```
my @ar = ["B", "C"];
unshift (@ar, "A");
foreach(@ar)
{
    print (shift(@ar));
}
```

20. While

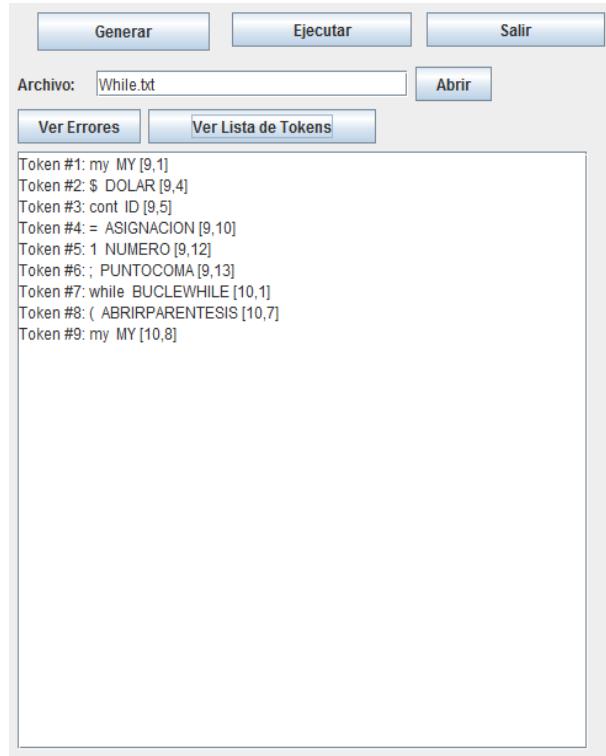
```
/* La estructura del while es : */
/* while (expresión_lógica) */
/* { */
/* lista_declaraciones */
/* } */
/* En este caso la expresión lógica es incorrecta. */
/* Por lo tanto se incumple la estructura. */

my $cont = 1;
while (my $cont=20;)
{
    print($cont);
    $cont++;
}
```

Ver Errores



Ver Lista de Tokens



Versión Correcta

```
my $cont = 1;
while ($cont<20)
{
    print($cont);
    $cont++;
}
```

4.3 Correcciones/Modificaciones

A continuación se indican las correcciones o modificaciones con respecto al documento entregado en la Parte I de el presente proyecto:

- + La gramática BNF fué modificada ya que la anterior presentaba algunos errores que era necesario corregir para el análisis sintáctico.
- + Se cambiaron algunos de los archivos de prueba por otros más completos.
- + Se cambió el archivo de configuración del generador léxico "alexico.flex"

5 Desarrollo Parte 3

5.1 Correcciones/Modificaciones

Entre las principales modificaciones realizadas al interprete de Perl, el cual estamos realizando, se encuentra el cambio realizado sobre el archivo de configuración de Flex, en el cual se indica que dejara de ignorar los espacios en blanco y que se reconociera como un token dentro del lenguaje, además se indicó en el BNF la gramática aceptada con el token de espacio en blanco, esto con el fin de solucionar el problema encontrado en la revisión del segundo entregable en el cual en los casos de que el error sintáctico se encontrara al final de la linea se reportaba en el error en la siguiente linea, esto porque se tomaba la linea donde se encontrara el siguiente token, como solución se decidió que los espacios fueran parte del lenguaje, con esto el error seria reportado en la linea correspondiente. Además se incluyo una derivación propia del generador de análisis sintáctico CUP para manejar errores sintácticos dicha derivación es 'error', el cual ante cualquier error sintáctico el programa va a continuar reportando el error respectivo, esto permitirá detectar varios errores sintácticos en el archivo, lo que no ocurría en la revisión del segundo entregable del proyecto. Se realizaron cambios en la gramática del BNF para algunas instrucciones especificadas, en las cuales fue necesario cambiar la estructura de las instrucciones porque faltaban datos que no fueron considerados anteriormente o no realizaba las acciones esperadas, lo que provocaba que los resultados retornados no fueran los que se esperaban. Una vez realizaron los cambios el programa se comportaba segun lo esperado. Se realizaron cambios a algunos archivos de prueba establecidos en el primer y segundo entregable, para que se ajustaran a las nuevas especificaciones implementadas en el BNF con el fin de que retornara los resultados esperados por el lenguaje

5.2 Tipos de errores semanticos

Se tomaron en cuenta 3 tipos de errores semánticos que el programa detecta en el momento de intentar ejecutar un programa:

1. El primer error es un error semántico que se presenta cuando asignamos a una variable un tipo (Arreglo, Cadena o Entero) y otra variable con otro tipo y si intenta realizar una operación con tipos diferentes, el programa detecta esta acción como incorrecta, ya que los tipos de las variables no coinciden.
2. El segundo error semántico consiste en la asignación de variables con el mismo nombre, esto no esta permitido en el lenguaje definido, buscando así evitar ambigüedades en los posibles programas a generar por el lenguaje.
3. El tercer error, consiste en la verificacion de la declaracion de las variables, en el momento de utilizar las mismas, detectando como error semantico, la utilizacion de variables no iniciadas o no declaradas anteriormente.

5.3 Errores semanticos

1. Break

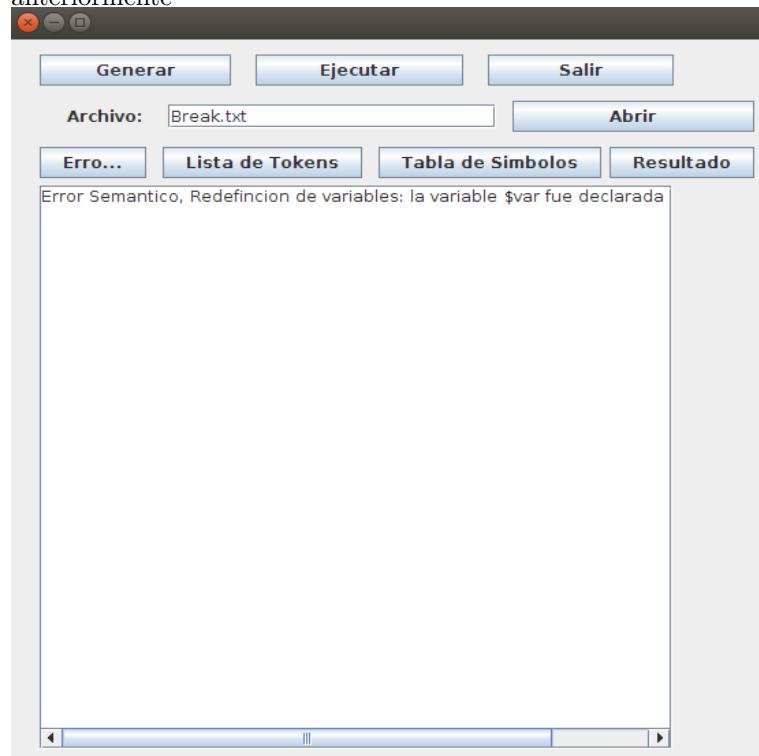
Codigo con Error:

```
/* La estructura del break es : */
/* break; */
/* Redeclaracion de variables: dos variables con el mismo nombre
lo cual no esta permitido */

my $var = 2;
my $var = "Hola";
switch ($var)
{
    case 1:
    {
        my $var = $var + 32;
        break;
    }
    case 2:
    {
        $var = $var + 16;
        break;
    }
    default:
    {
        $var = $var + 67;
        break;
    }
}
```

Mensaje del Error:

Error Semantico, Redefincion de variables: la variable \$var fue declarada anteriormente



Explicacion del Error:

El error semantico anterior se presenta cuando declaramos una variable con un nombre especifico, en el caso del ejemplo anterior tenemos la variable \$var, asignando el valor igual a 2, y despues la misma variable \$var asignandole la cadena "Hola", lo mismo genera un error semantico ya que se estan declarano a dos variables con el mismo nombre

Version Correcta:

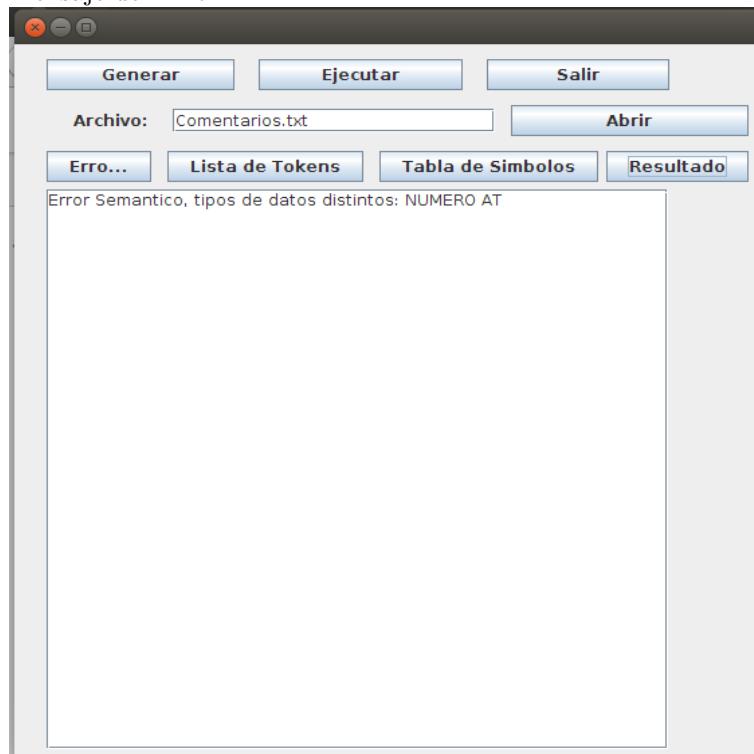
```
my $var = 2;
my $var2 = "Hola";
switch ($var)
{
    case 1:
    {
        my $var = $var + 32;
        break;
    }
    case 2:
    {
        $var = $var + 16;
        break;
    }
    default:
    {
        $var = $var + 67;
        break;
    }
}
print($var1);
```

2. Comentarios

Codigo con Error:

```
/*Comentario*/  
my $var1 = 33;  
my @var = ["1","2","3"];  
my $var2 = $var1+@var2;
```

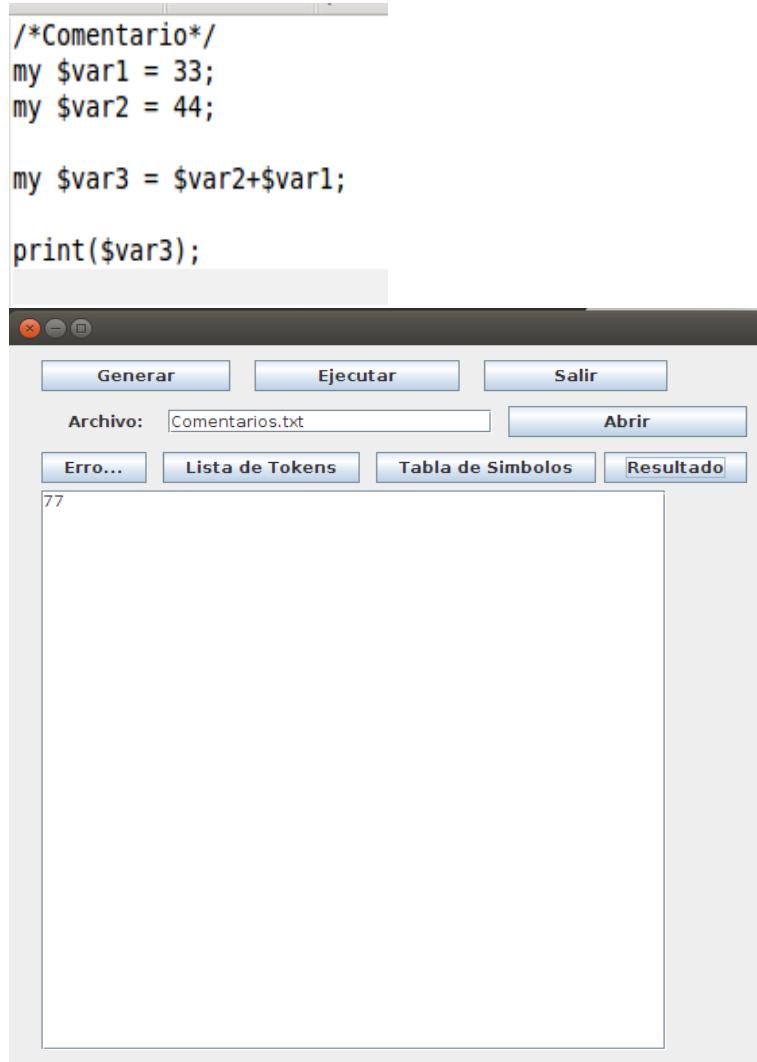
Mensaje del Error:



Explicacion del Error:

En el error anterior, se presenta un error semántico, ya que existe un problema de tipos, en el momento de realizar operaciones con tipos diferentes, ya que el programa no puede sumar una variable tipo Array con una variable de tipo Entero, por lo que se reporta como un error semántico de tipos diferentes. Versión Correcta:

```
/*Comentario*/  
my $var1 = 33;  
my $var2 = 44;  
  
my $var3 = $var2+$var1;  
  
print($var3);
```



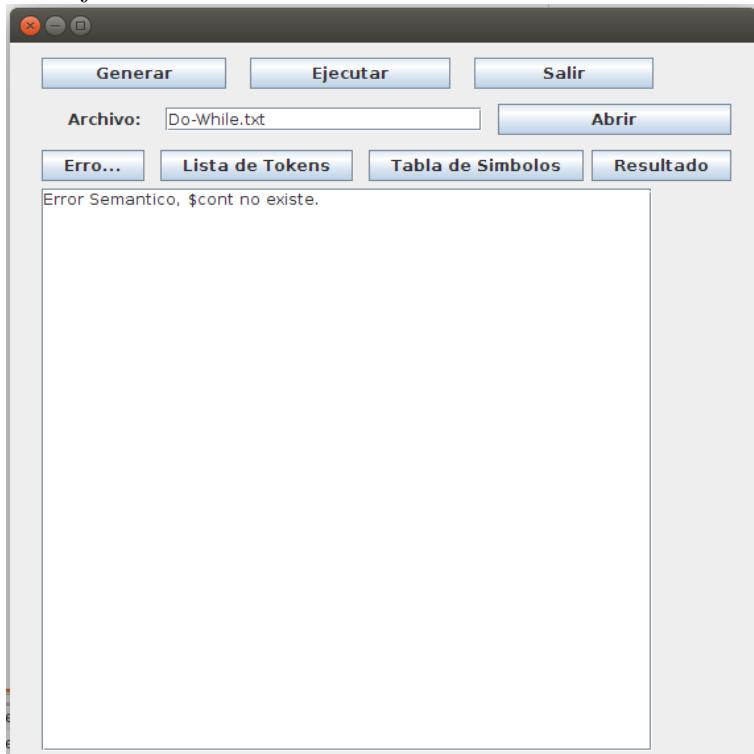
The image shows a software application window with a dark header bar. Below the header are several buttons: 'Generar' (highlighted in blue), 'Ejecutar', 'Salir', 'Archivo: Comentarios.txt', 'Abrir', 'Erro...', 'Lista de Tokens', 'Tabla de Simbolos', and 'Resultado'. The main area contains a large text box with the value '77' at the top left corner.

3. Do-While

Codigo con Error:

```
do
{
    print($cont);
    $cont++;
}
while($cont < 5);
```

Mensaje del Error:

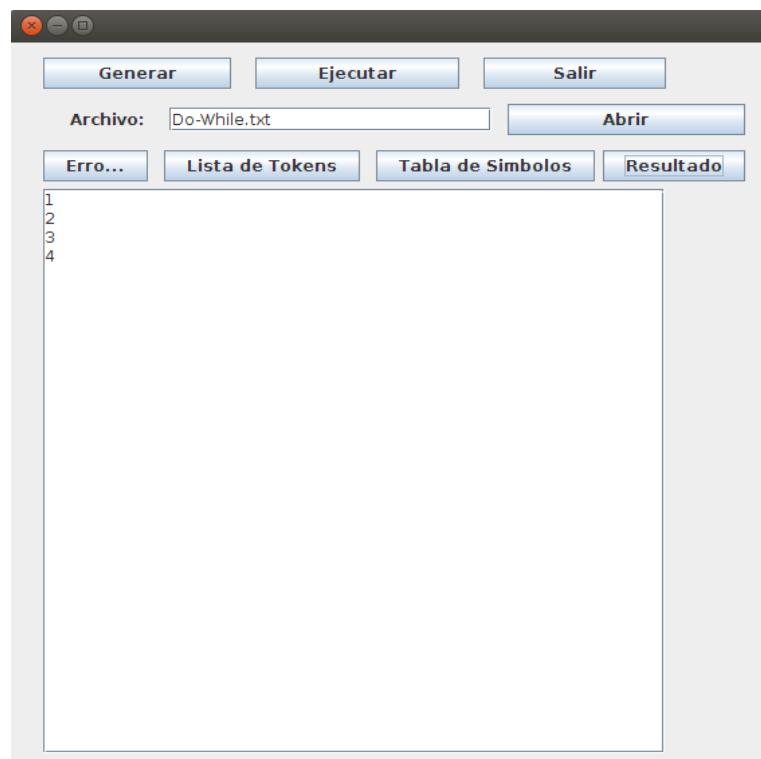


Explicacion del Error:

En el programa anterior se encuentra el error de la no declaracion de una variable que se esta utilizando en las condiciones del ciclo por lo que se el programa retorna un error donde se informa que la variable no ha sido declarada

Version Correcta:

```
'  
my $cont = 1;  
do  
{  
    print($cont);  
    $cont++;  
}  
while($cont < 5);
```

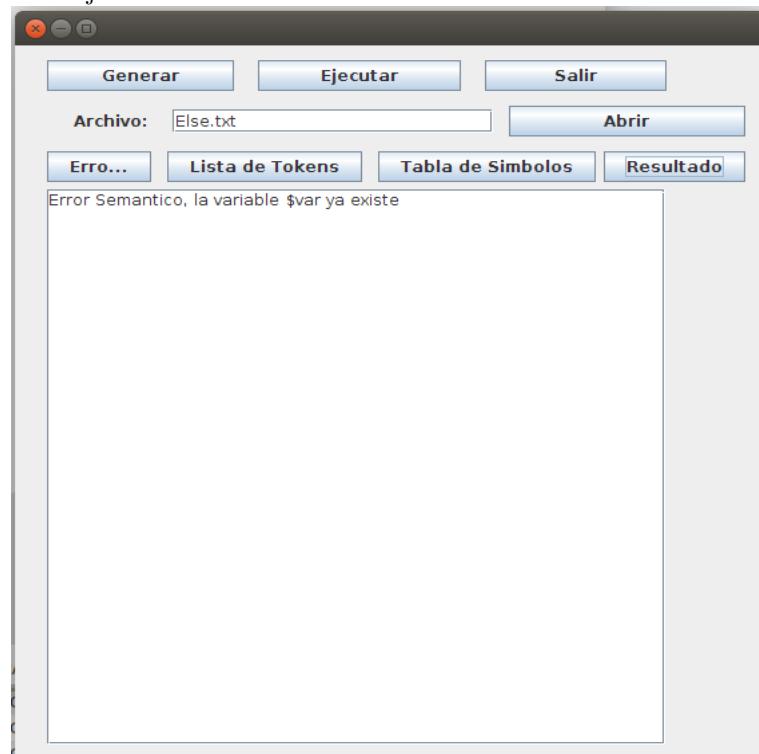


4. Else

Codigo con Error:

```
my $var = 30;
if($var < 99)
{
    my $var = 5;
    print("Menor que 99");
}
else
{
    print("Mayor que 99");
}
```

Mensaje del Error:

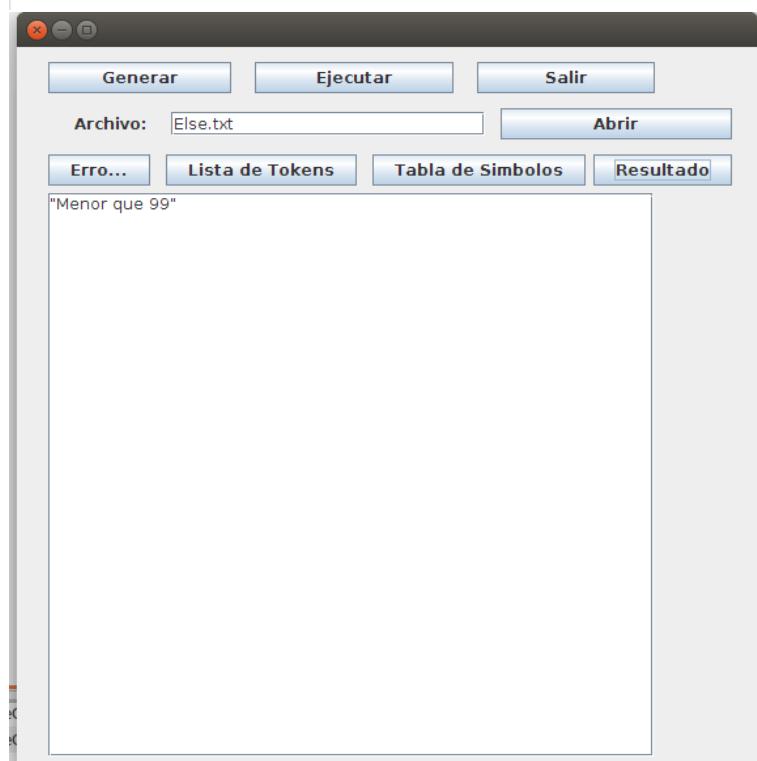


Explicacion del Error:

En el programa anterior se encuentra el error de la no declaracion de una variable que se esta utilizando en las condiciones del ciclo por lo que se el programa retorna un error donde se informa que la variable no ha sido declarada

Version Correcta:

```
my $var = 30;
if($var < 99)
{
    my $var2 = 5;
    print("Menor que 99");
}
else
{
    print("Mayor que 99");
}
```



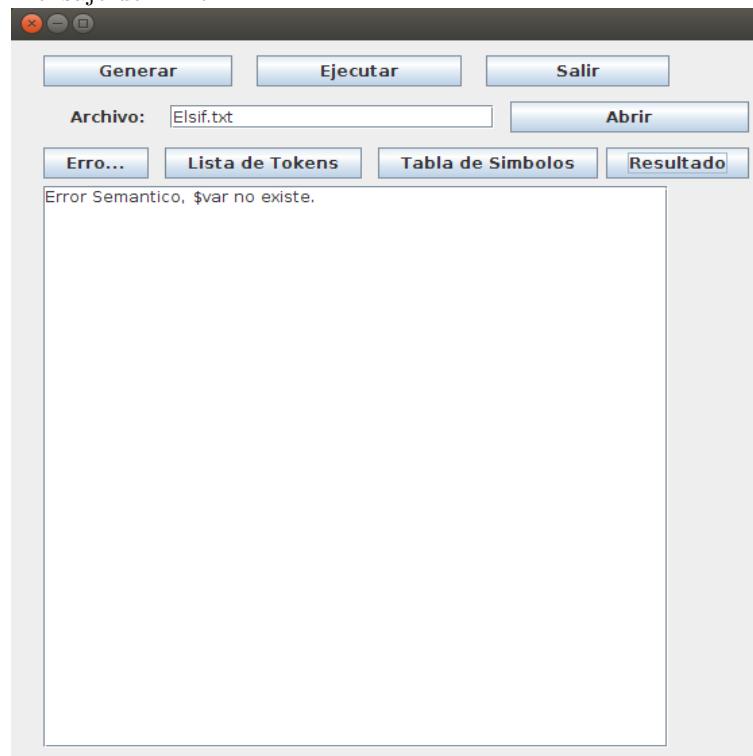
5. Elsif

Codigo con Error:

```
/* La estructura del elsif es : */
/* elsif (expresion) */
/* { */
/* lista_declaraciones */
/* } */

$var = 99;
if($var < 99)
{
    print("Menor que 99");
}
elsif($var ==99)
{
    print("Igual a 99");
}
else
{
    print("Mayor que 99");
}
```

Mensaje del Error:



Explicacion del Error:

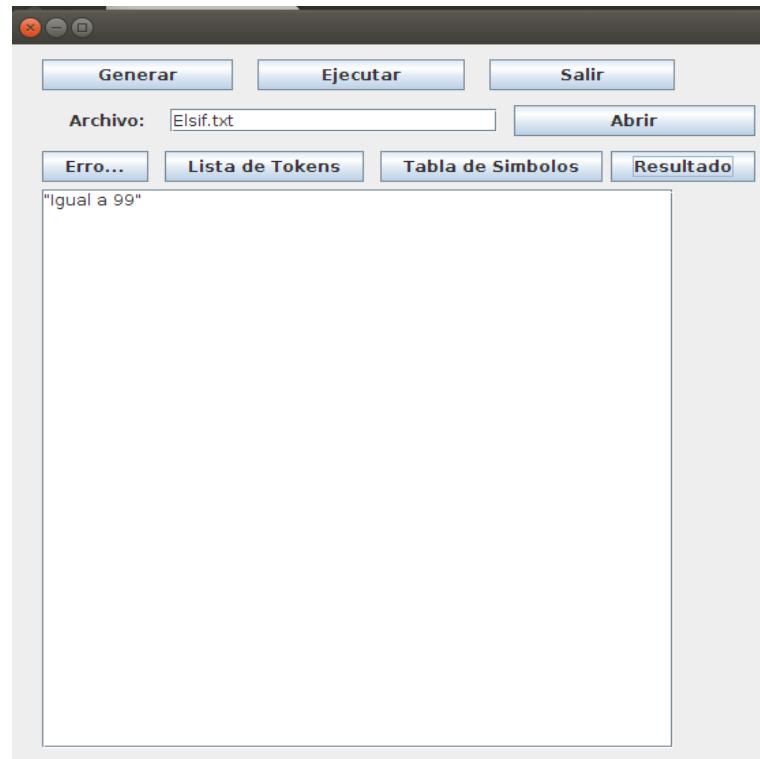
El error semantico anterior se presenta, ya que la variable \$var, no ha sido declarada anteriormente, por lo que al intentar usarla, retornara un error

semantico, de acuerdo a las reglas semanticas establecidas.

Version Correcta:

```
/* La estructura del elsif es : */
/* elsif (expresion) */
/* { */
/* lista_declaraciones */
/* } */
```

```
my $var = 99;
if($var < 99)
{
    print("Menor que 99");
}
elsif($var ==99)
{
    print("Igual a 99");
}
else
{
    print("Mayor que 99");
}
```

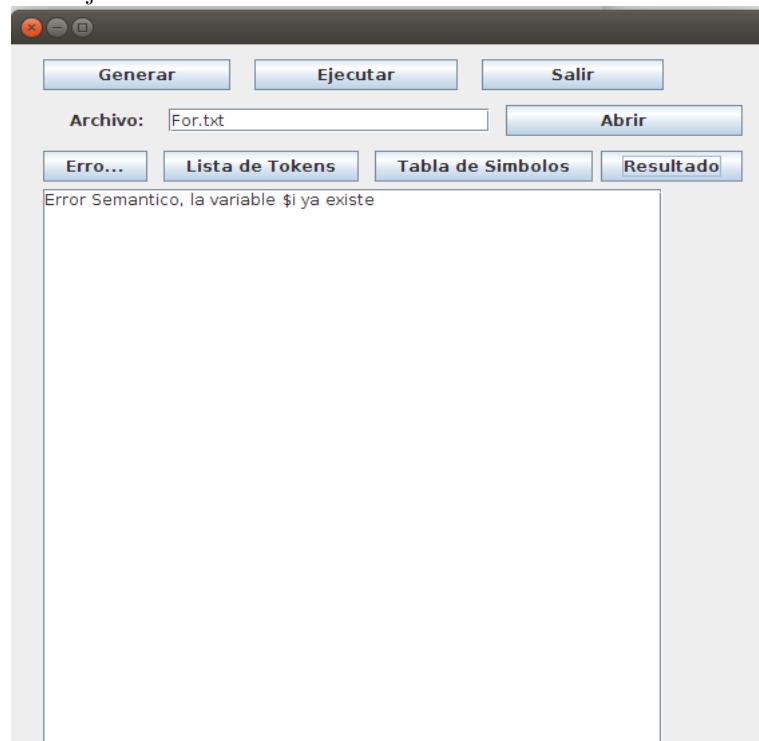


6. For

Codigo con Error:

```
my $i = 0;  
for(my $i = 0;$i < 30;$i++)  
{  
    print($i);  
}
```

Mensaje del Error:

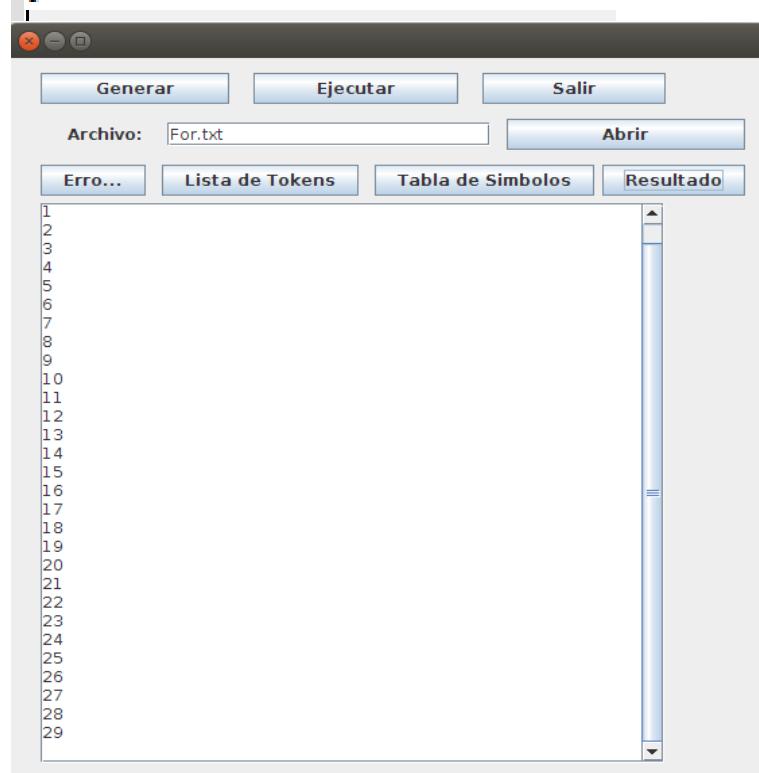


Explicacion del Error:

Una re declaracion de variables no esta permitida, incluyendo las variables que forman parte de la condicion dentro dentro de un ciclo, por lo que el usuario retorna un error donde informa sobre una redeclaracion de variables no permitida

Version Correcta:

```
my $var = 0;
for(my $i = 0;$i < 30;$i++)
{
    print($i);
}
```

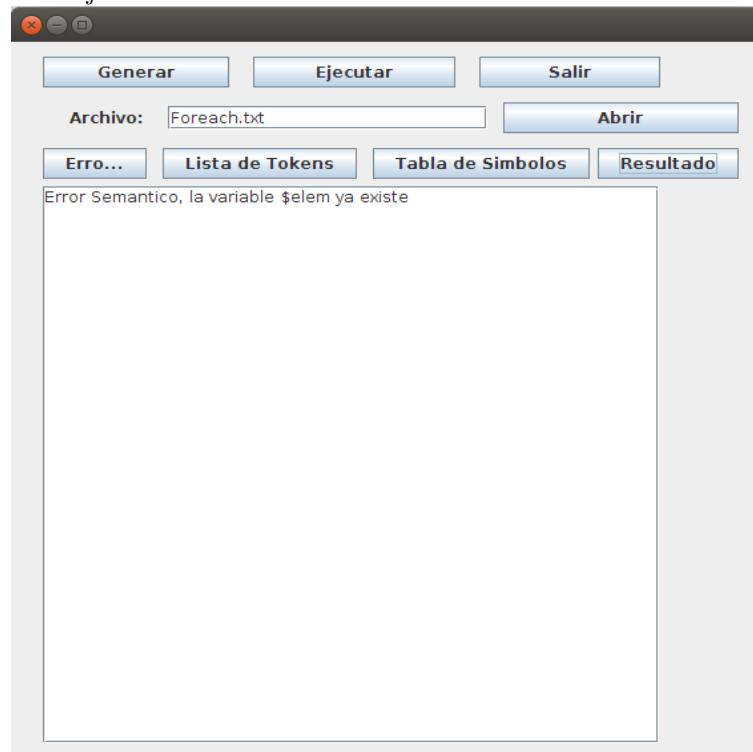


7. Foreach

Codigo con Error:

```
my @ar = ["A", "B", "C"];
my $elem = "D";
foreach($elem,@ar)
{
    print($elem);
}
```

Mensaje del Error:

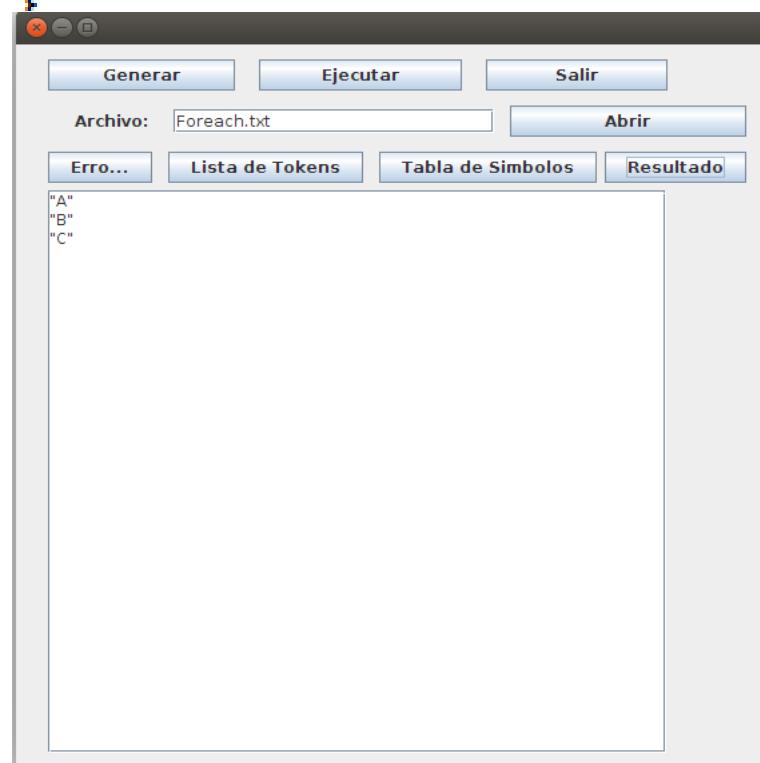


Explicacion del Error:

En caso de que la declaracion sea implicita, no se puede declarar otra variable con el mismo nombre, como en el ejemplo anterior, donde se intenta declarar una variable con el mismo nombre de la variable implicita que se encuentra dentro del ciclo foreach lo cual es una accion no permitida, considerada como error semantico.

Version Correcta:

```
my @ar = ["A", "B", "C"];
my @elem = ["D"];
foreach($elem,@ar)
{
    print($elem);
}
```

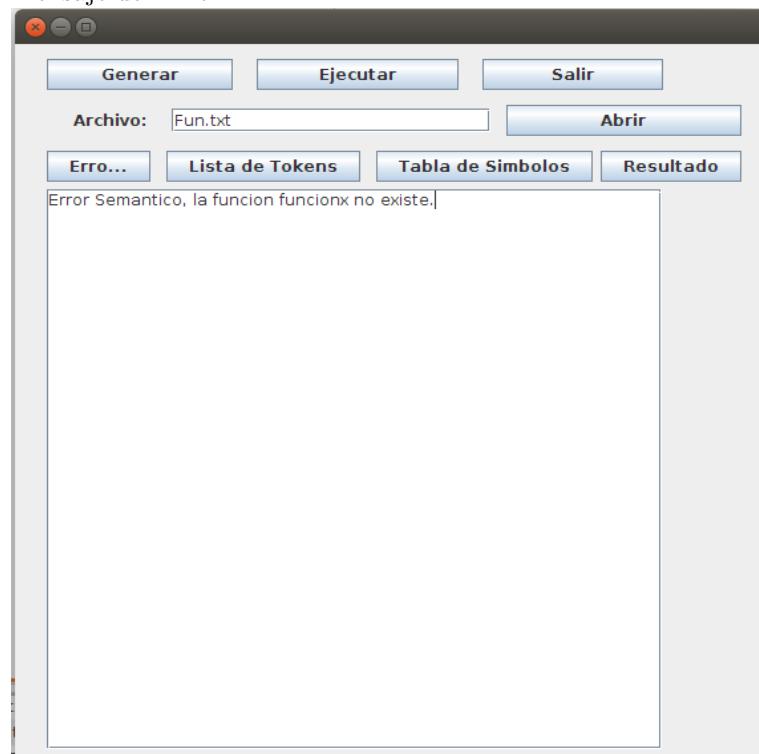


8. Fun

Codigo con Error:

```
fun funcionx(1,2);
```

Mensaje del Error:



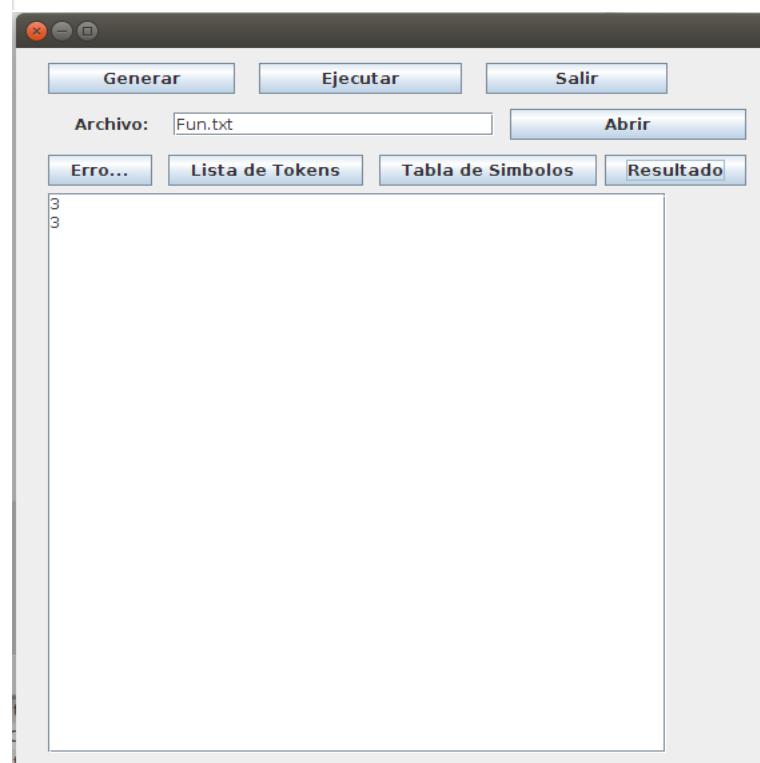
Explicacion del Error:

Si se intenta llamar una función que no ha sido declarada, es considerado un error semántico por parte del lenguaje, y no está permitido, por lo que se retornará el mensaje de error correspondiente, para que el usuario realice los cambios necesarios y pueda continuar con el desarrollo normal del programa.

Version Correcta:

```
my fun funcionx($param1,$param2)
{
    my $var = $param1 + $param2;
    print($var);
}

fun funcionx(1,2);
```

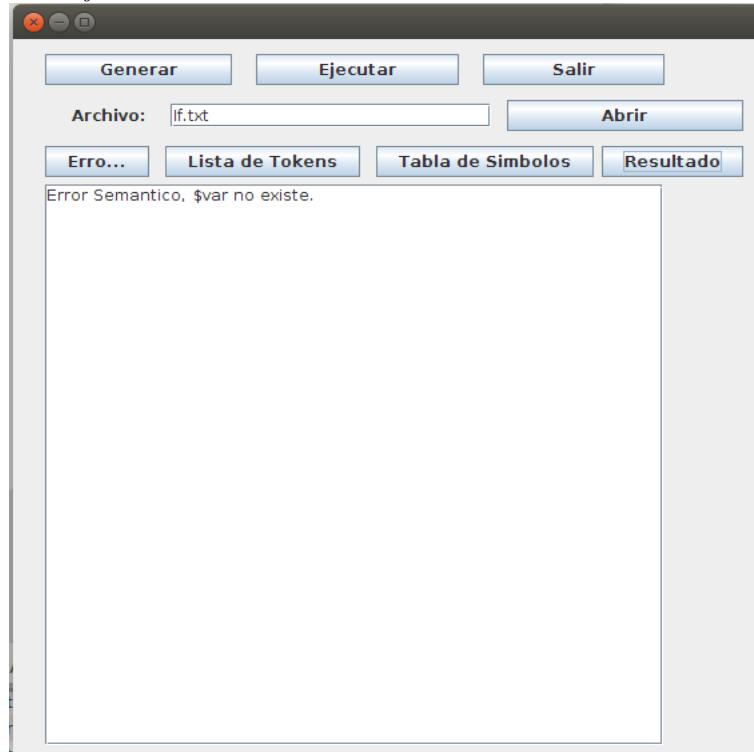


9. If

Codigo con Error:

```
my $var;
if($var < 99)
{
    print("El numero es menor que 99");
}
```

Mensaje del Error:

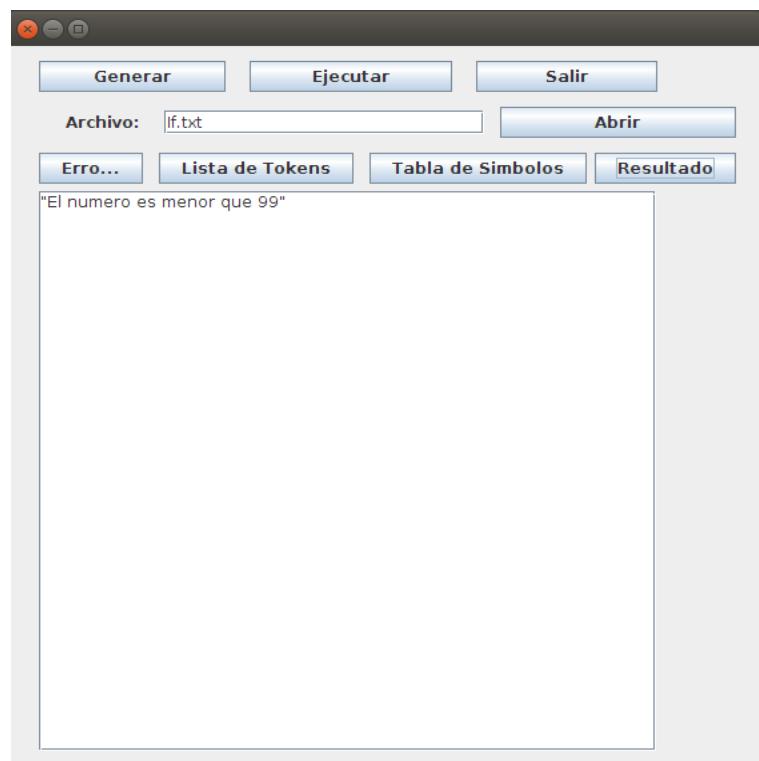


Explicacion del Error:

Es necesario iniciar las variables con algun valor, porque de otro modo el programa detecta esa falta como un error semantic, ta y como ocurrio en el ejemplo anterior, una vez la variable a sido inciada el programa se ejecuta correctamente

Version Correcta:

```
my $var=30;  
if($var < 99)  
{  
    print("El numero es menor que 99");  
}
```

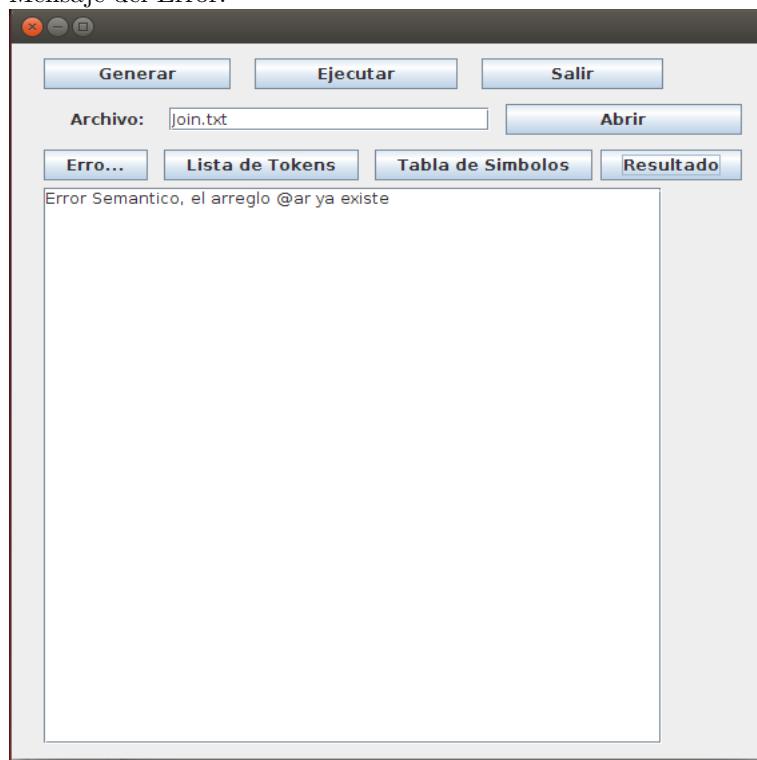


10. Join

Codigo con Error:

```
/* La estructura del join es : */  
/* join (union, @ ID) */  
/* En este caso falta el elemento de unión para completar la estructura. */  
my @ar = ["A","D","I","O","S"];  
my @ar = ["H","O","L","A"];  
my $string = join(@ar, " ");  
print($string);
```

Mensaje del Error:

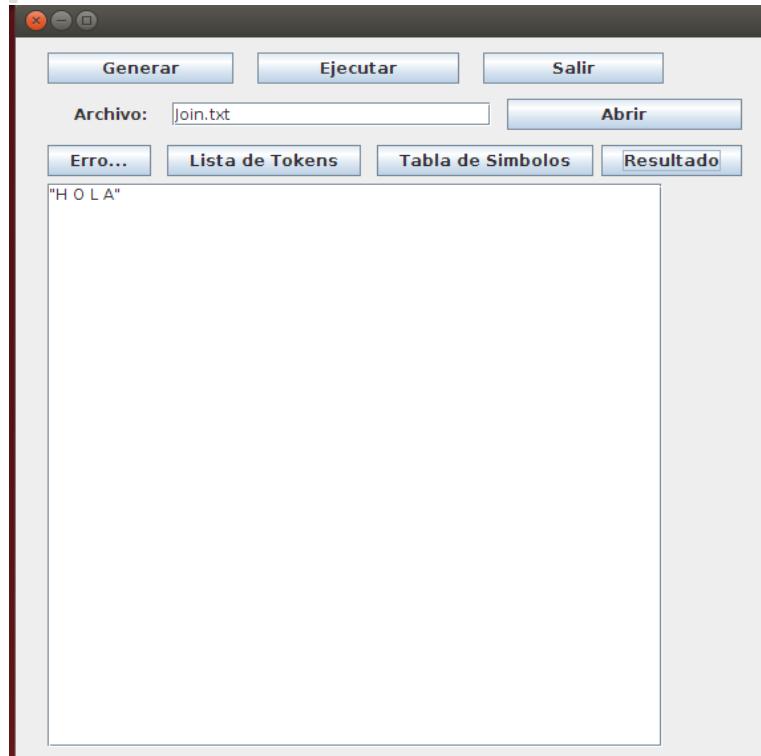


Explicacion del Error:

El error semantico anterior se produce porque existe una doble declaracion de la variable @ar, lo cual para evitar ambiguedades, por reglas semanticas no esta permitido, por lo que se le indica al usuario la existencia de un error semantic, con la dobl declaracion de variables para que los corrigay seguir asi con el funcionamiento del programa.

Version Correcta:

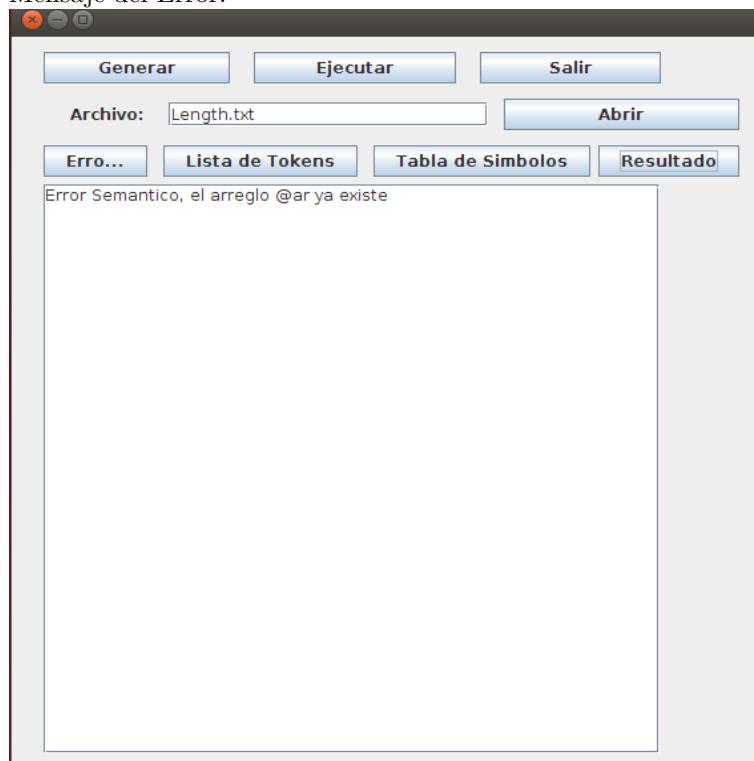
```
/* La estructura del join es : */
/* join (union, @ ID) */
/* En este caso falta el elemento de unión para completar la estructura. */
my @ar = ["A","D","I","O","S"];
my @ar2 = ["H","O","L","A"];
my $string = join(@ar2, " ");
print($string);
```



11. Length Código con Error:

```
my @ar = [ "Hola" ];
my @ar = [ "Hola", "Mundo" ];
my $tam = length($ar);
print($tam);
```

Mensaje del Error:

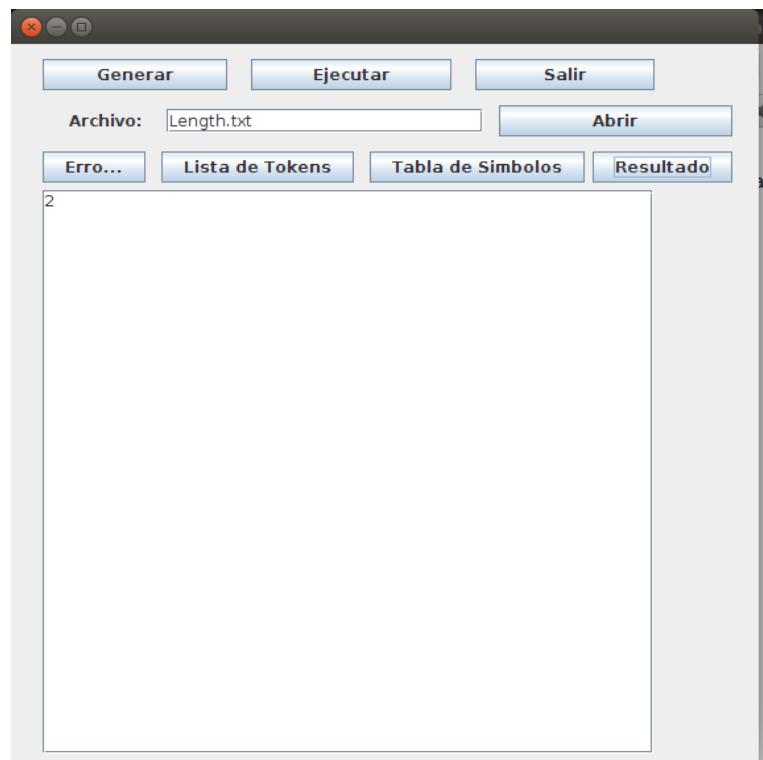


Explicacion del Error:

Cuando se da una re-declaracion de variables, se retorna un error semantico indicando el error correspondiente, tal es el caso de el programa anterior donde hubo una re-declaracion de variables, el cual el programa detecto y reporto satisfactoriamente

Version Correcta:

```
my $ar = "Hola";
my @ar = [ "Hola", "Mundo" ];
my $tam = length($ar);
print($tam);
```

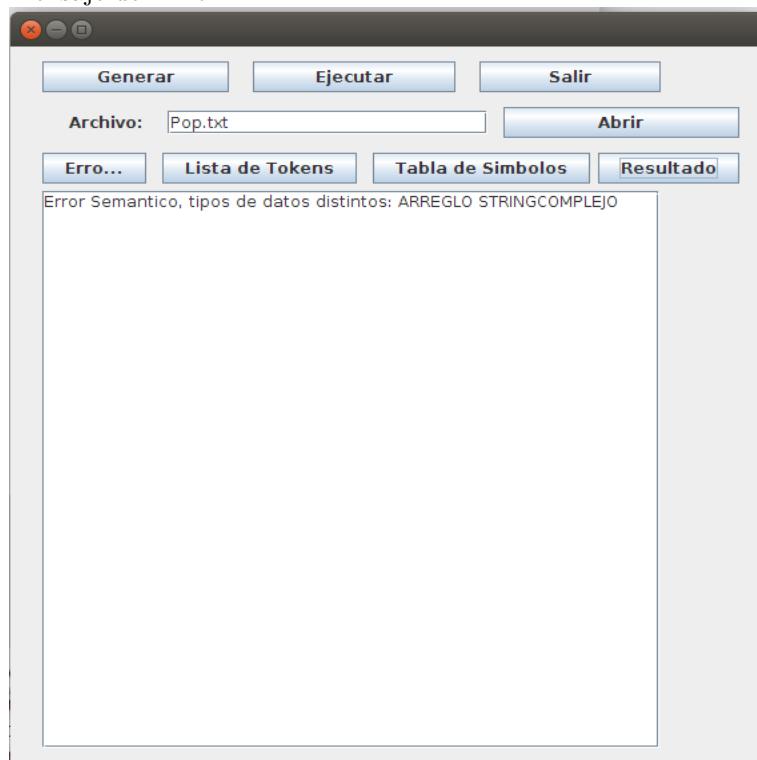


12. Pop

Codigo con Error:

```
my @ar = ["A", "B", "C"];
my $ultimo = pop(@ar);
print ($ultimo);
```

Mensaje del Error:

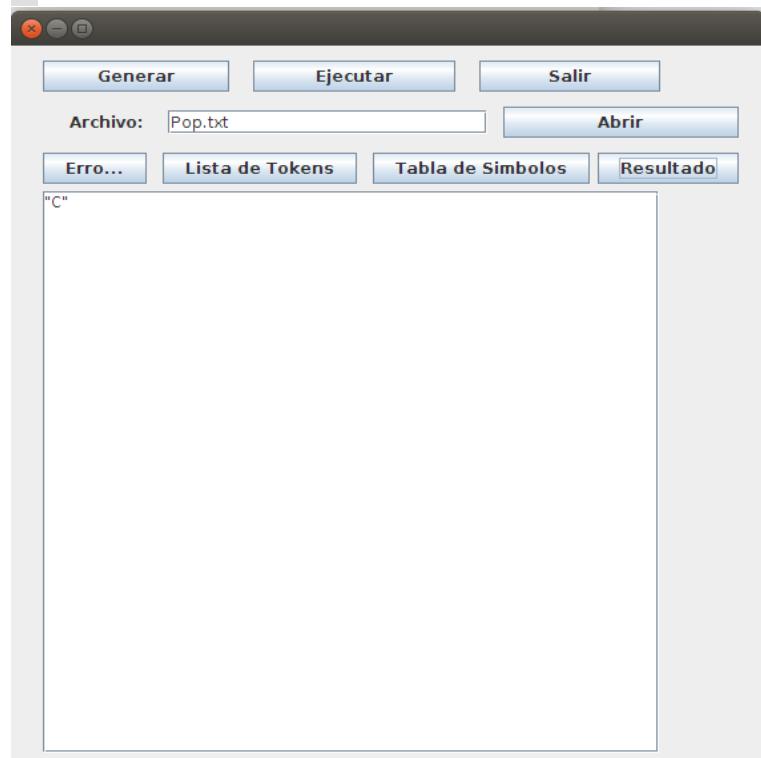


Explicacion del Error:

En el caso de funciones y la asignacion de variables, el lenguaje se asegura de que los errores semanticos sean tomado en cuenta, el error anterior se presenta cuando se intenta asignar a un areglo una funcion cuyo valor de retorno es un string, retornando el mensaje de que encontro un error semantico y deteniendo la ejecucion del programa.

Version Correcta:

```
my @ar = ["A", "B", "C"];
my $ultimo = pop(@ar);
print ($ultimo);
```



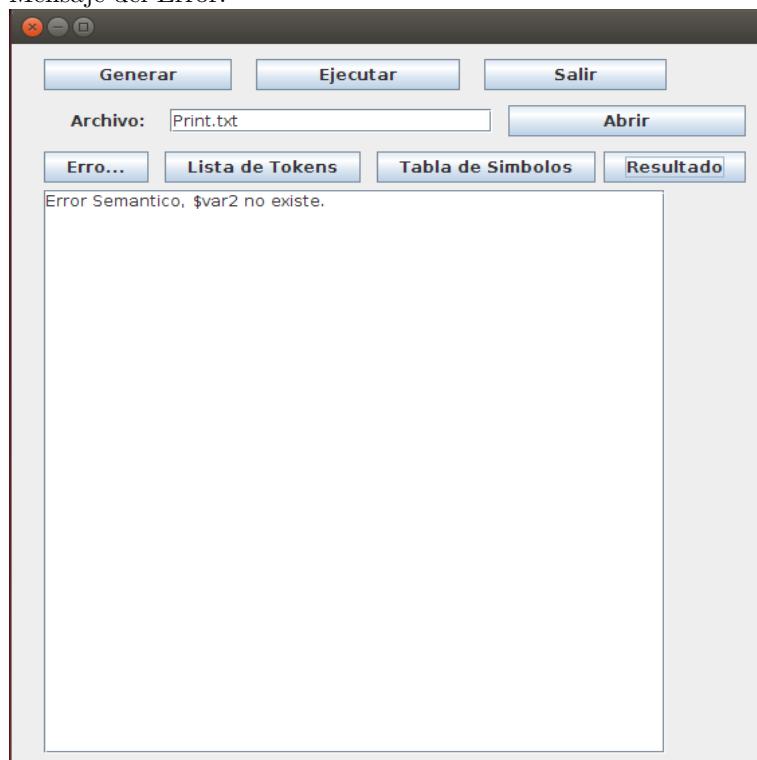
13. Print

Codigo con Error:

```
/* La estructura del print es : */
/* print(expresion); */

my $var1 = 20;
my $resultado = $var1 + $var2;
print ($resultado);
```

Mensaje del Error:



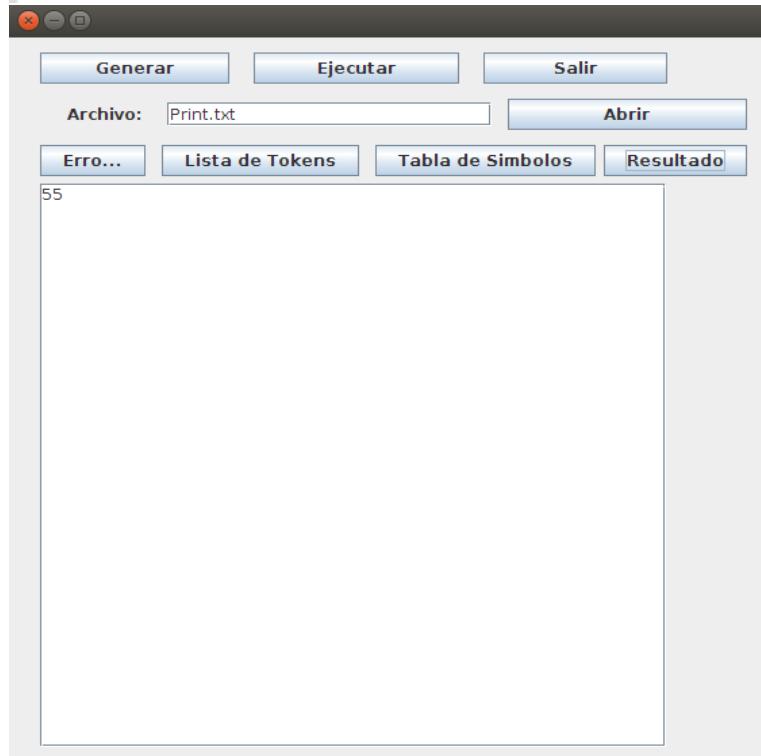
Explicacion del Error:

El error semantico anterior se produce cuando se intenta utilizar una variable que no ha sido declarada anteriormente por lo que el programa retorna un error semantico indicandole al usuario, para su corección correspondiente

Version Correcta:

```
/* La estructura del print es : */
/* print(expresión); */

my $var1 = 20;
my $var2 = 35;
my $resultado = $var1 + $var2;
print ($resultado);
```



14. Push

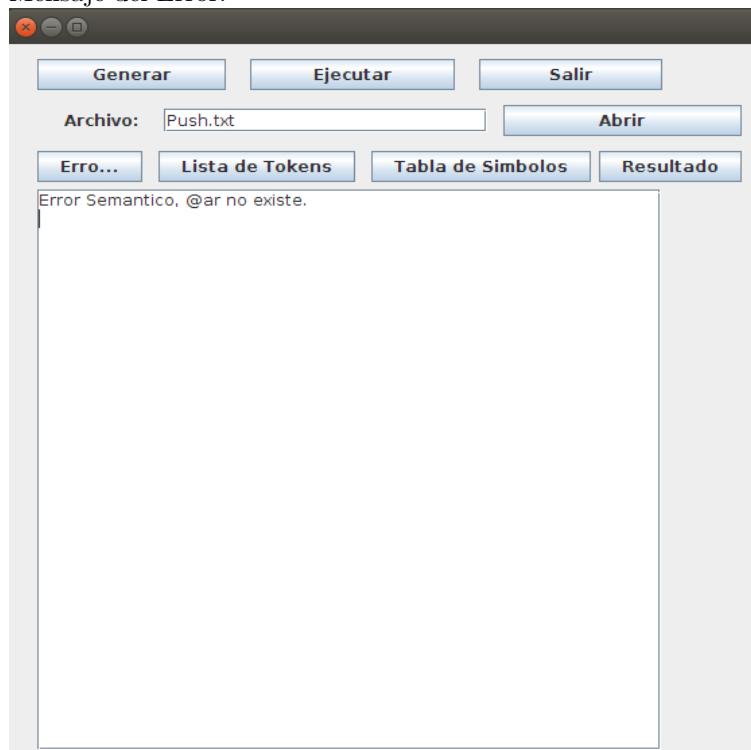
Codigo con Error:

```
/* La estructura del push es : */
/* push(@ ID, expresion); */

/*my @ar = ["A","B","C","D"];*/
my $str = "C";
push(@ar,$str);

foreach($elem,@ar)
{
    print($elem);
}
```

Mensaje del Error:



Explicacion del Error:

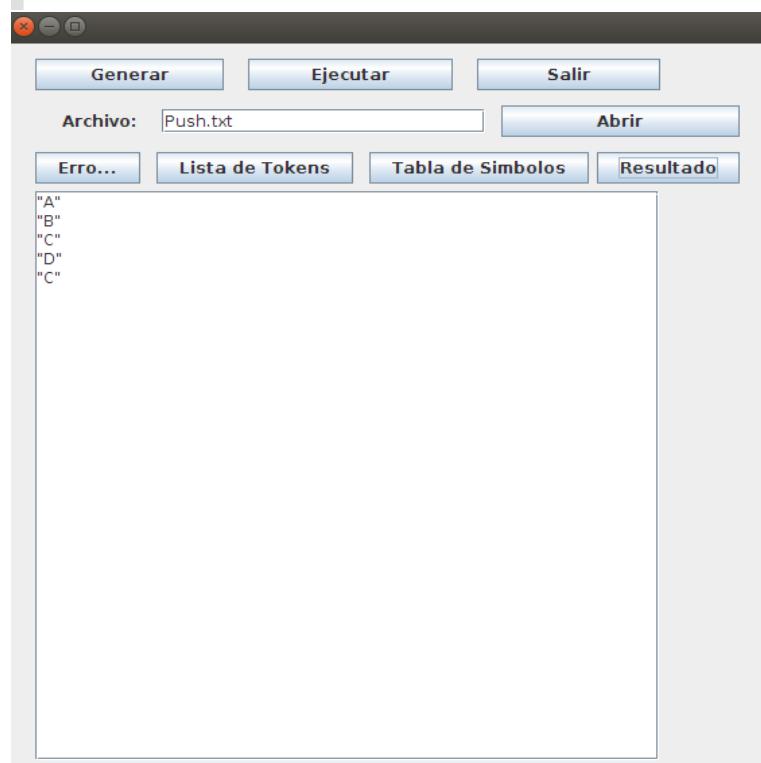
El error semantico encontrado en el programa anterior, se da cuando se procede a realizar la verificacion de la declaracion de las variables, en caso de utilizar los mismos nombres de variables, detectando como error semantico y reportandole al usuario el error correspondiente.

Version Correcta:

```
/* La estructura del push es : */
/* push(@ ID, expresion); */

my @ar = ["A", "B", "C", "D"];
my $str = "C";
push(@ar,$str);

foreach($elem,@ar)
{
    print($elem);
}
```



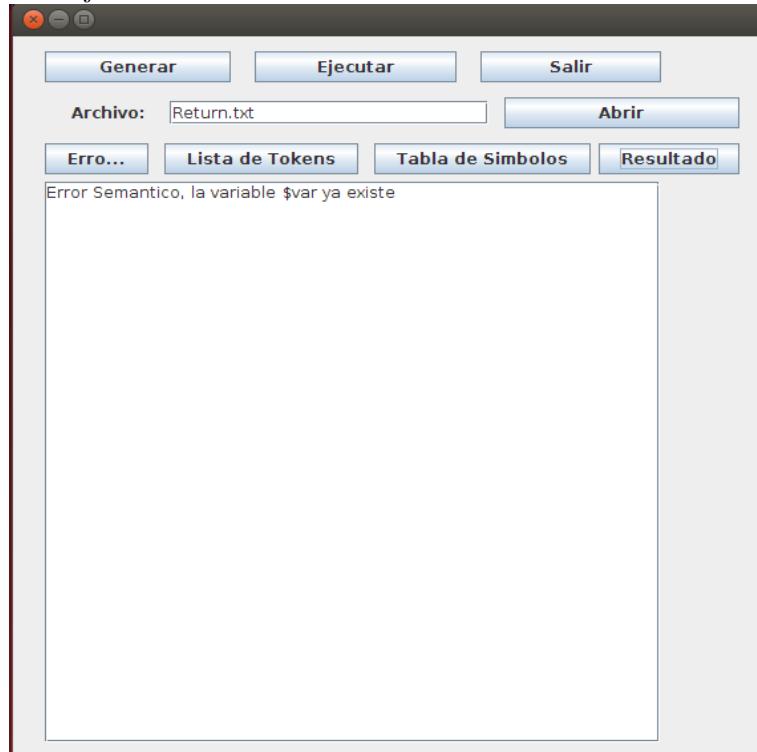
15. Return

Codigo con Error:

```
/* La estructura del return es: */
/* return expresion; */

my $var = 28;
if ($var > 13)
{
    my $var = 3;
    return 1;
}
else
{
    return 0;
}
```

Mensaje del Error:



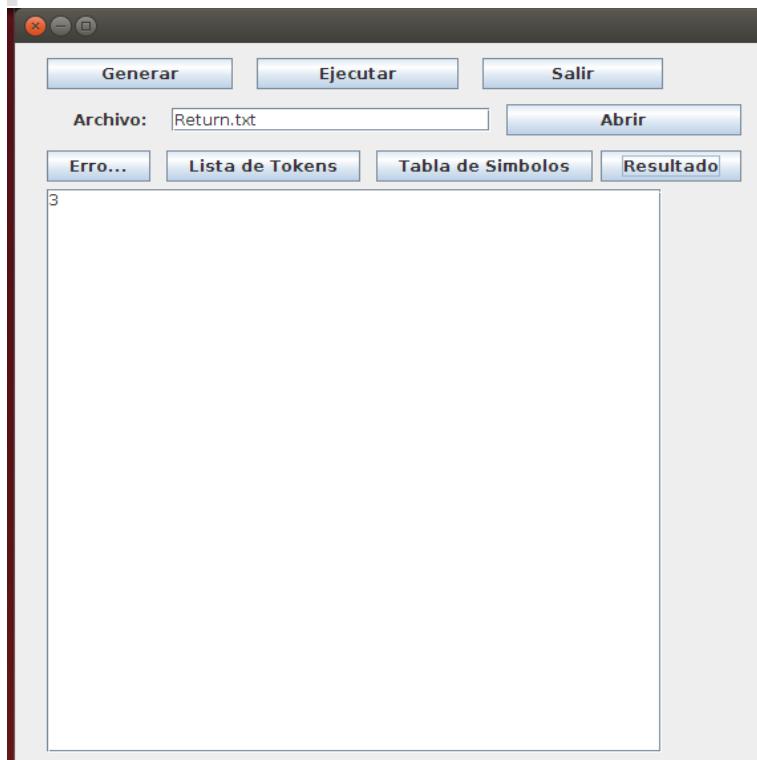
Explicacion del Error:

El error semantico anterior, se da cuando se intenta declarar 2 errores con el mismo, lo cual no esta permitido segun las reglas semanticas definidas en el lenguaje MicroPerl por el equipo de desarrolladores.

Version Correcta:

```
/* La estructura del return es: */
/* return expresion; */

my $var = 28;
if ($var > 13)
{
    my $var2 = 3;
    print($var2);
    return 1;
}
else
{
    return 0;
}
```



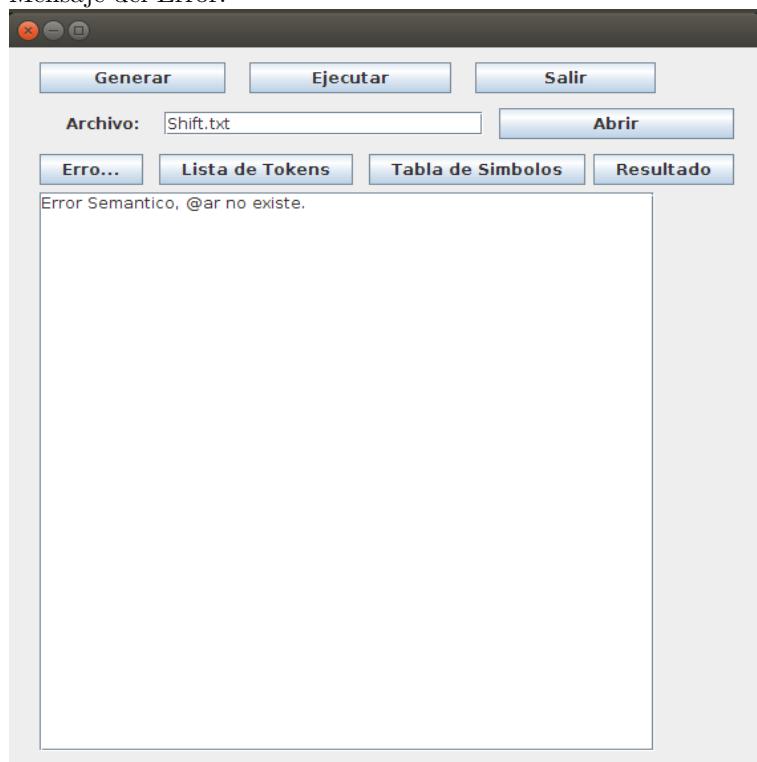
16. Shift

Codigo con Error:

```
/* La estructura del shift es : */
/* shift(@ ID); */

/*my @ar = [5,6,7,12,15,92];*/
my $var = shift(@ar);
print($var);
```

Mensaje del Error:

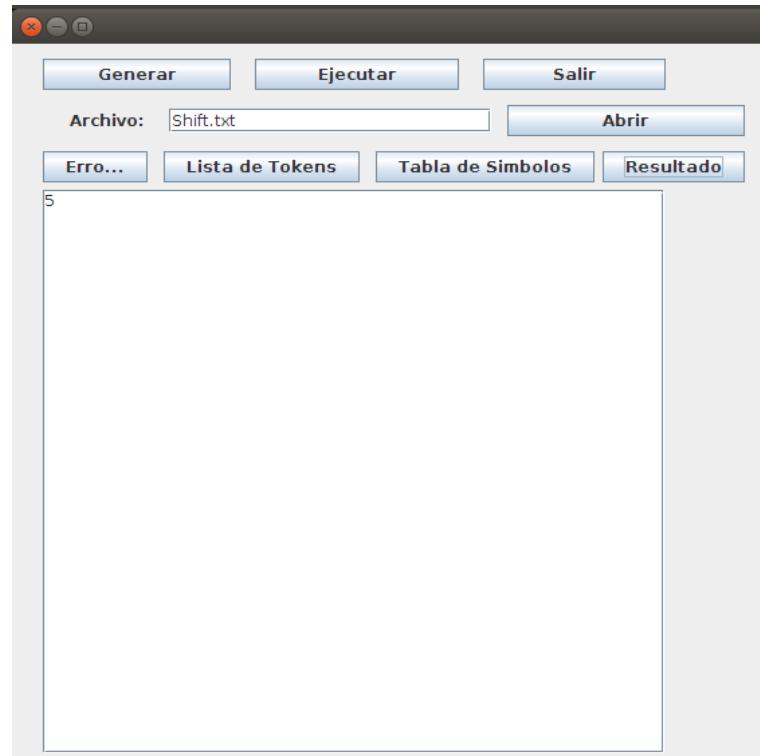


Explicacion del Error:

En caso de encontrar la llamada de una variable que no ha sido declarada, el programa retornara el mensaje indicando que encontro un error semantico al usuario, indicando la variable que no a sido declarada para su corregion.

Version Correcta:

```
/* La estructura del shift es : */  
/* shift(@ ID); */  
  
my @ar = [5,6,7,12,15,92];  
my $var = shift(@ar);  
print($var);
```



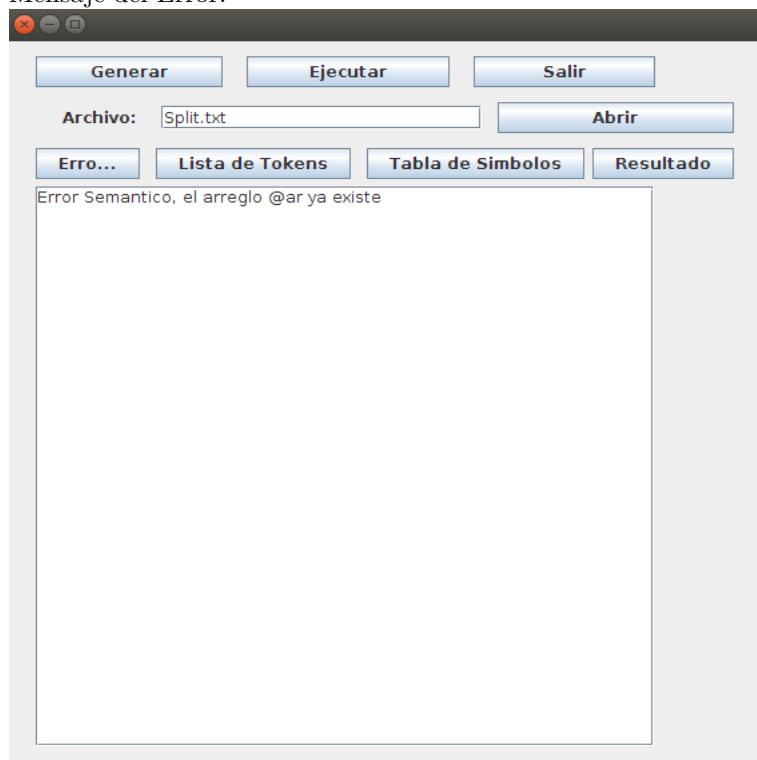
17. Split

Codigo con Error:

```
/* La estructura del split es : */
/* split(patron, $ ID); */

my $string = "H&O&L&A";
my @ar = split($string,"&");
my @ar = ["H","O","L","A"];
foreach($elem,@ar)
{
    print($elem);
}
```

Mensaje del Error:

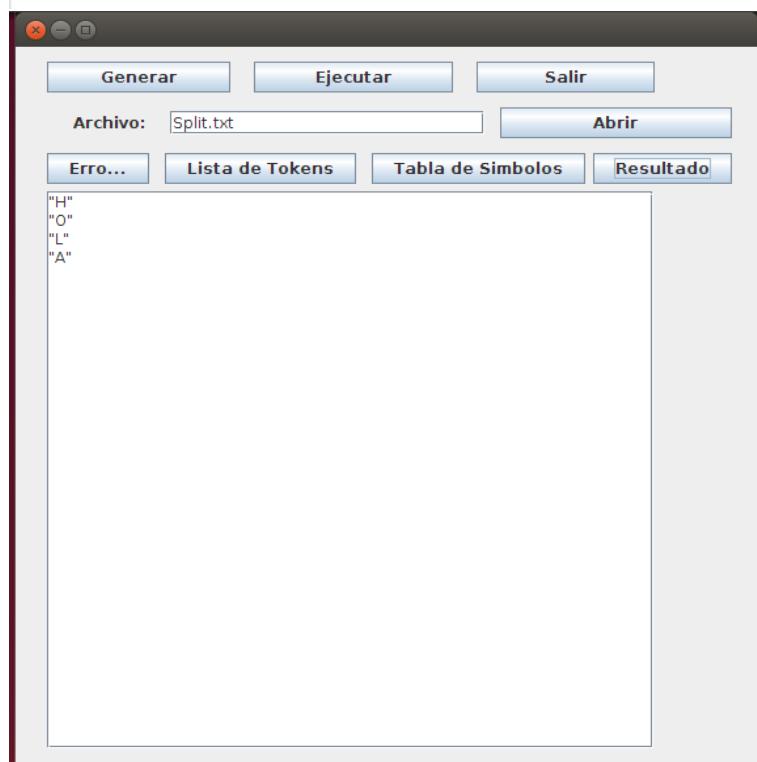


Explicacion del Error:

El error semantico anterior, se presenta al declarar dos variables con el mismo nombre, en ese caso el programa reporta la aparicion de un error semantico por la doble declaracion de variable utilizando el mismo nombre.
Version Correcta:

```
/* La estructura del split es : */  
/* split(patron, $ ID); */
```

```
my $string = "H&O&L&A";  
my @ar = split($string,"&");  
foreach($elem,@ar)  
{  
    print($elem);  
}
```



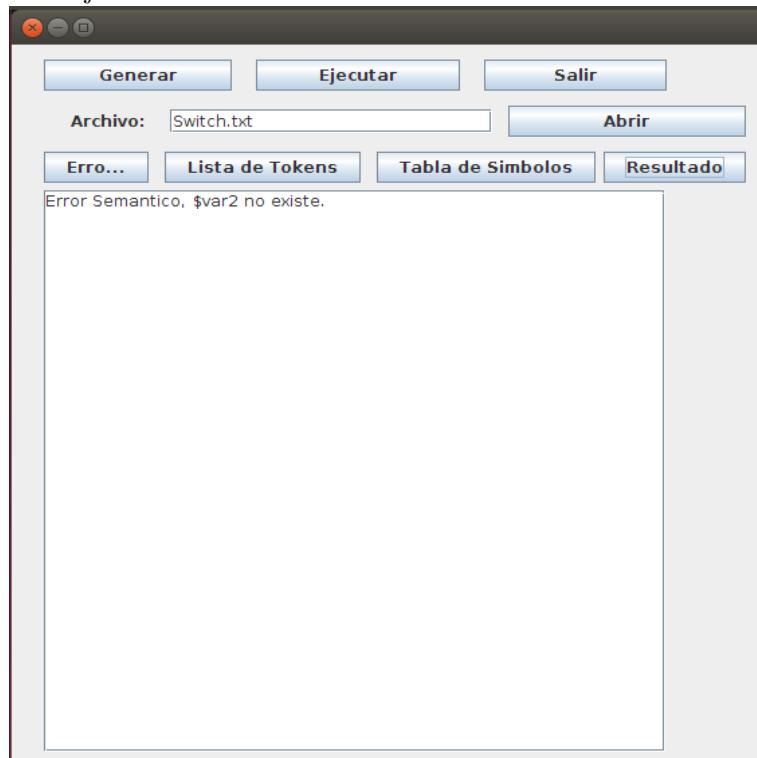
18. Switch

Código con Error:

```
/* La estructura del switch es : */
/* switch (expresion_simple) */
/* { */
/*   expresion_case */
/*   expresion_default */
/* } */
/* En este caso falta la expresión default. */

my $caso = 2;
my $var1 = 0;
switch ($caso)
{
    case 1:
    {
        $var1 = $var2 + 32;
        break;
    }
    case 2:
    {
        $var1 = $var2 + 16;
        break;
    }
    default:
    {
        break;
    }
}
```

Mensaje del Error:



Explicacion del Error:

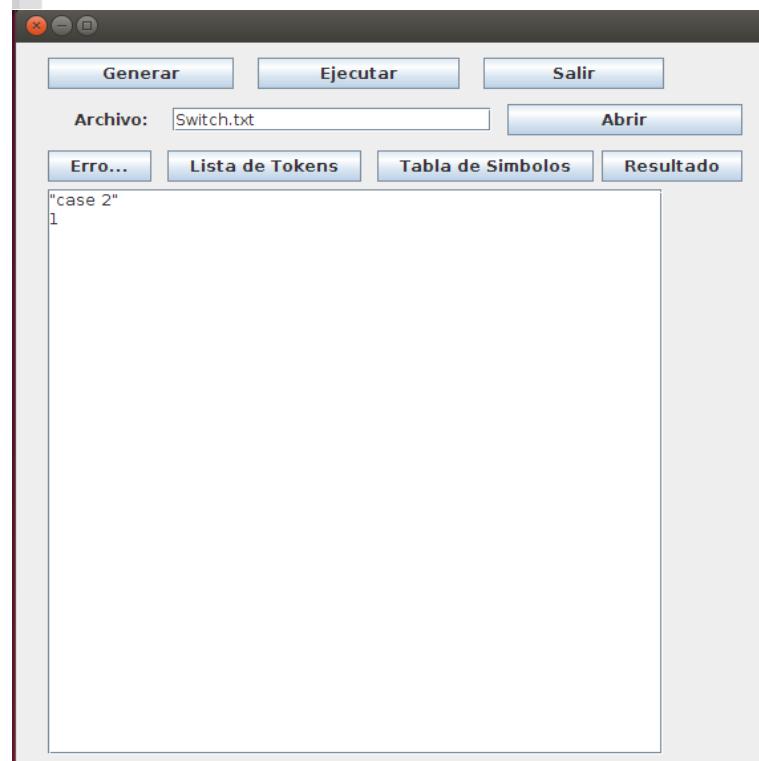
El error semántico anterior se presenta por llamar una variable que no ha sido declarada anteriormente en el código, por lo que dentro del lenguaje

MicroPerl y sintacticamente seria incorrecto, dentro del programa por lo que se reporta el error correspondiente al usuario, con la informacion necesaria.

Version Correcta:

```
/* La estructura del switch es : */
/* switch (expresion_simple) */
/* { */
/*   /* expresion_case */
/*   /* expresion_default */
/* } */

my $caso = 2;
my $vari = 0;
my $var2 = 3;
switch ($caso)
{
    case 1:
    {
        $vari = 2;
        print("case 1");
        break;
    }
    case 2:
    {
        $vari = 1;
        print("case 2");
        break;
    }
    default:
    {
        print("default");
        break;
    }
}
print($vari);
```



19. Unshift

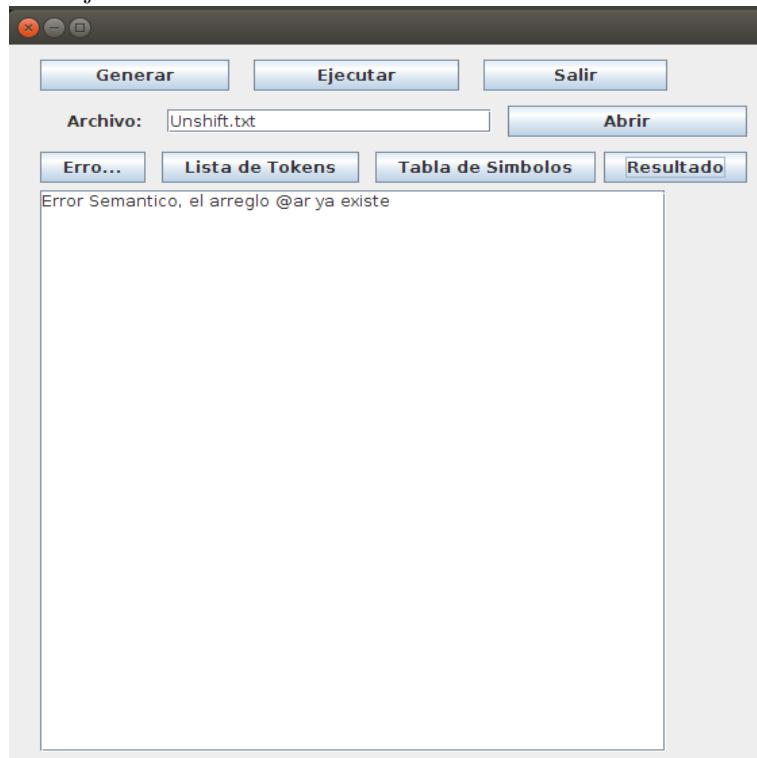
Codigo con Error:

```
/* La estructura del unshift es : */
/* unshift(@ ID, expresion); */

my @ar = ["B","C"];
unshift(@ar,"A");

foreach($elem,@ar)
{
    print($elem);
}
my @ar=[ "D"];
```

Mensaje del Error:



Explicacion del Error:

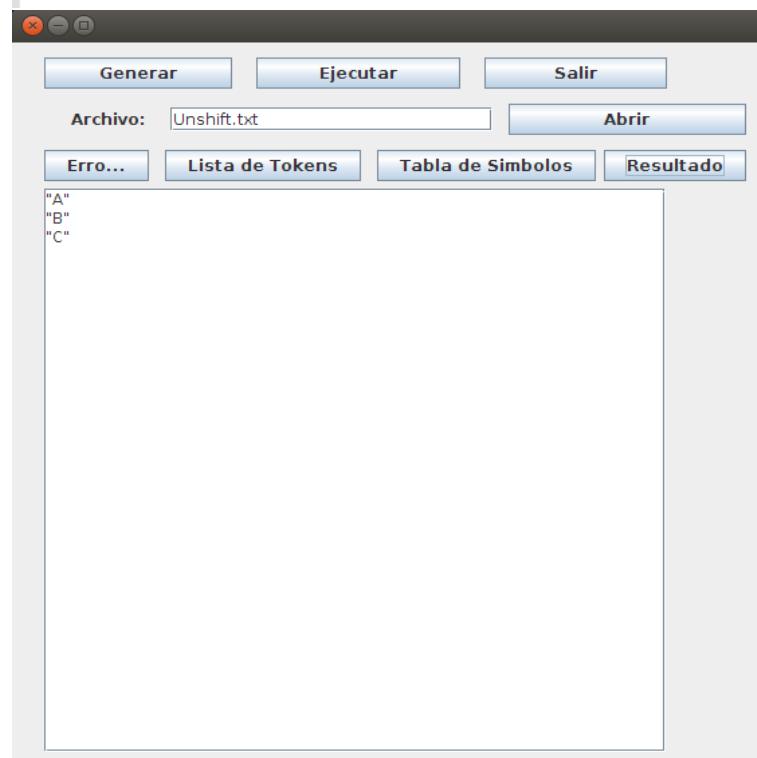
El error del ejemplo anterior ocurre cuando se presenta una declaracion doble de variables, por lo que el programa detecta y reporta una doble declaracion de variables al usuario, y cancela cualquier ejecucion hasta la correccion del error semantico.

Version Correcta:

```
/* La estructura del unshift es : */
/* unshift(@ ID, expresion); */

my @ar = ["B","C"];
unshift(@ar,"A");

foreach($elem,@ar)
{
    print($elem);
}
```



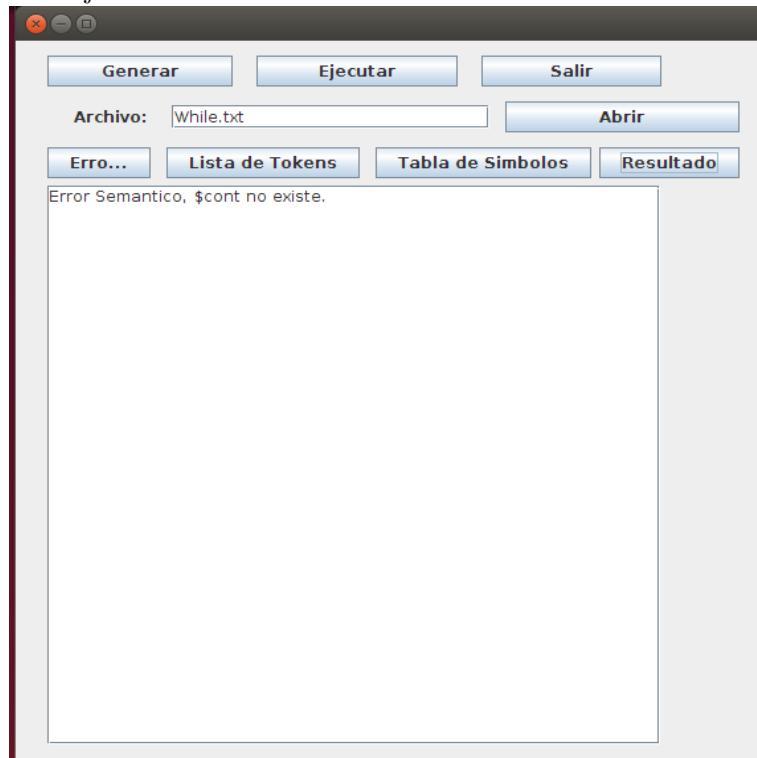
20. While

Codigo con Error:

```
/* La estructura del while es : */
/* while (expresion_lógica) */
/* { */
/* lista_declaraciones */
/* } */

while ($cont<20)
{
    print($cont);
    $cont++;
}
```

Mensaje del Error:

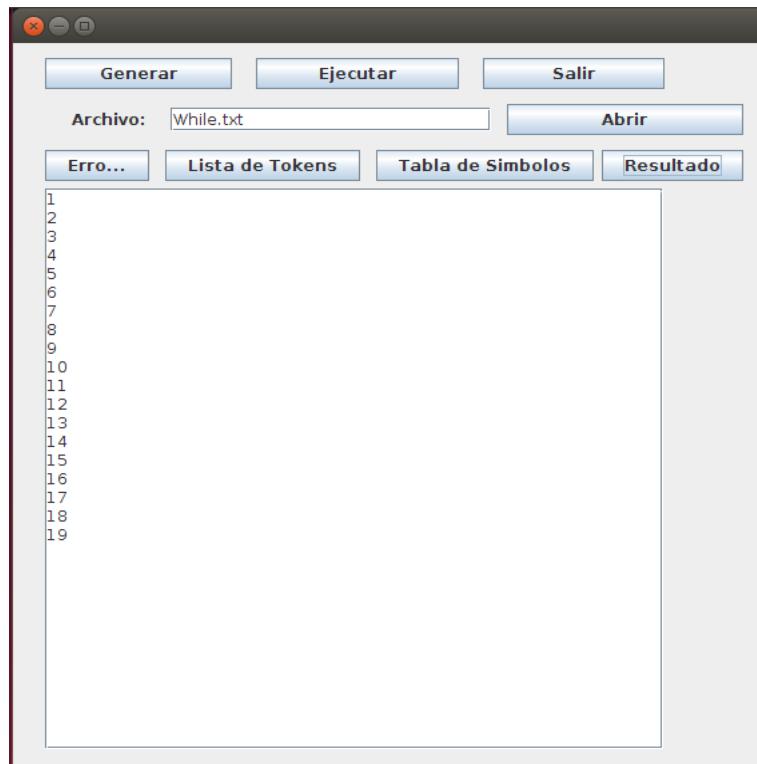


Explicacion del Error:

El error semantico anterior se presenta por llamar una variable que no ha sido declarada anteriormente en el codigo, por lo que dentro del lenguaje MicroPerl y sintacticamente seria incorrecto, dentro del programa por lo que se reporta el error correspondiente al usuario, con la informacion necesaria.

Version Correcta:

```
/* La estructura del while es : */
/* while (expresion_lógica) */
/* { */
/* lista_declaraciones */
/* } */
my $cont = 1;
while ($cont<20)
{
    print($cont);
    $cont++;
}
```



5.4 Declaracion de los cambios realizados en las palabras reservadas o en la estructura del lenguaje

- Se decidio dejar de ignorar los espacios, los cambios de linea y tabulaciones: La decision de dejar de ignorar los espacios, los cambios de linea y tabulaciones, debido a que era necesario detectar errores en la misma linea, ya que se descubrio que los errores se reportaban en la linea siguiente se debia a que reportaba la linea donde se encontraba el siguiente token, y como no coincidia con la sintaxis se reportaba en la linea siguiente cuando deberia ser en la linea anterior, por lo que se decidio no ignorar los espacios para que el error reportado fuera en la misma linea, mejorando con esto los mensajes de error.
- Se agrego una nueva derivacion al inicio de la gramatica BNF, propia del generador de codigo CUP, llamado error, el mismo se deriva en caso de que se presente una estructura que no coincide con las estructuras aceptadas, se reporta el error correspondiente y se continua revisando la estructura de la sintaxis, asi se van acumulando los reportes de errores para ser reportados en cascada.
- Se realizaron leves cambios en la estructura de algunas instrucciones, ya que para su correcto funcionamiento, fue necesario agregar una condicion adicional, para el retorno de los resultados aceptados.
- Con respecto a los tokens establecidos en los entregables de la primera parte y de la segunda parte no se realizo ningun cambio mas alla de los mencionados anteriormente.

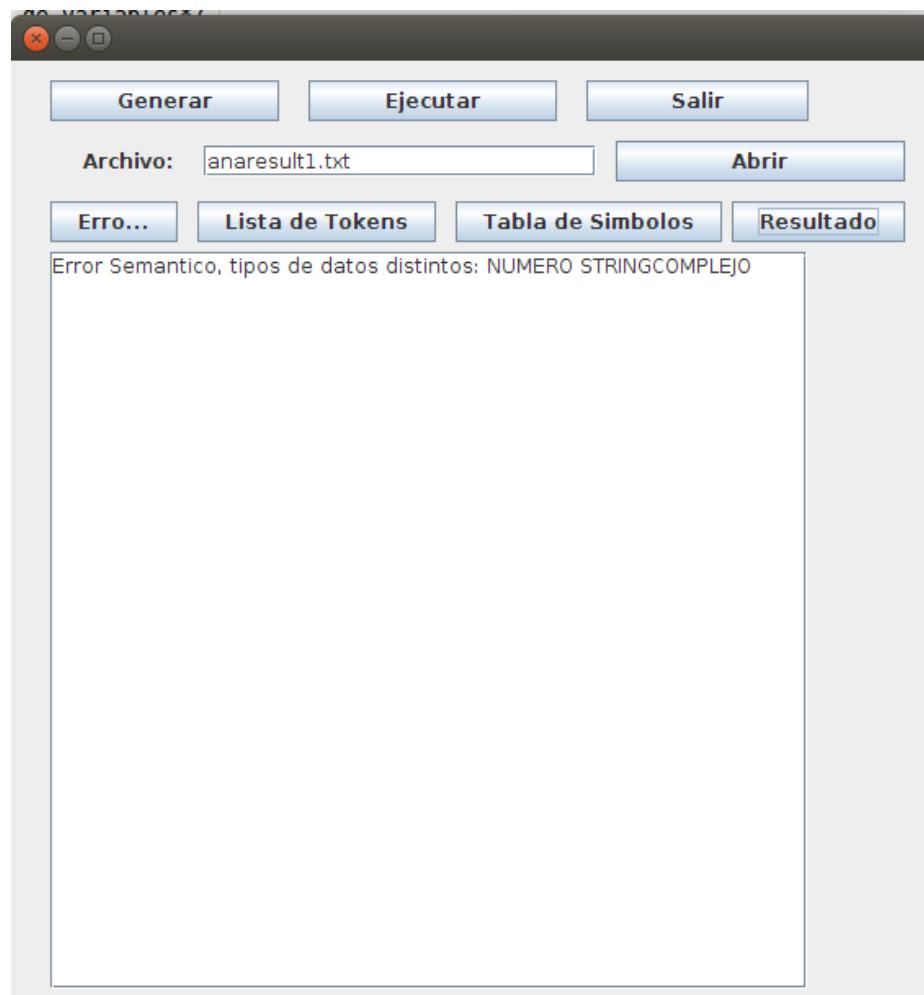
5.5 Características agregadas y decisiones de diseño

- Como se mencionado antes, una de las caracteristicas agregadas al programa es el correcto reporte de mensajes en la linea correspondiente, asi como el reporte de varios errores sintacticos en cascada, lo que no ocurría en los entregables anteriores
- Con respecto a las instrucciones establecidas se mantiene las mismas 20 instrucciones con cambios minimos en su estructura en algunas instrucciones como foreach y el do-while, cambios necesarios para un correcto funcionamiento de las instrucciones, las cuales realizarian su labor segun lo establecido.

5.6 Análisis de resultados

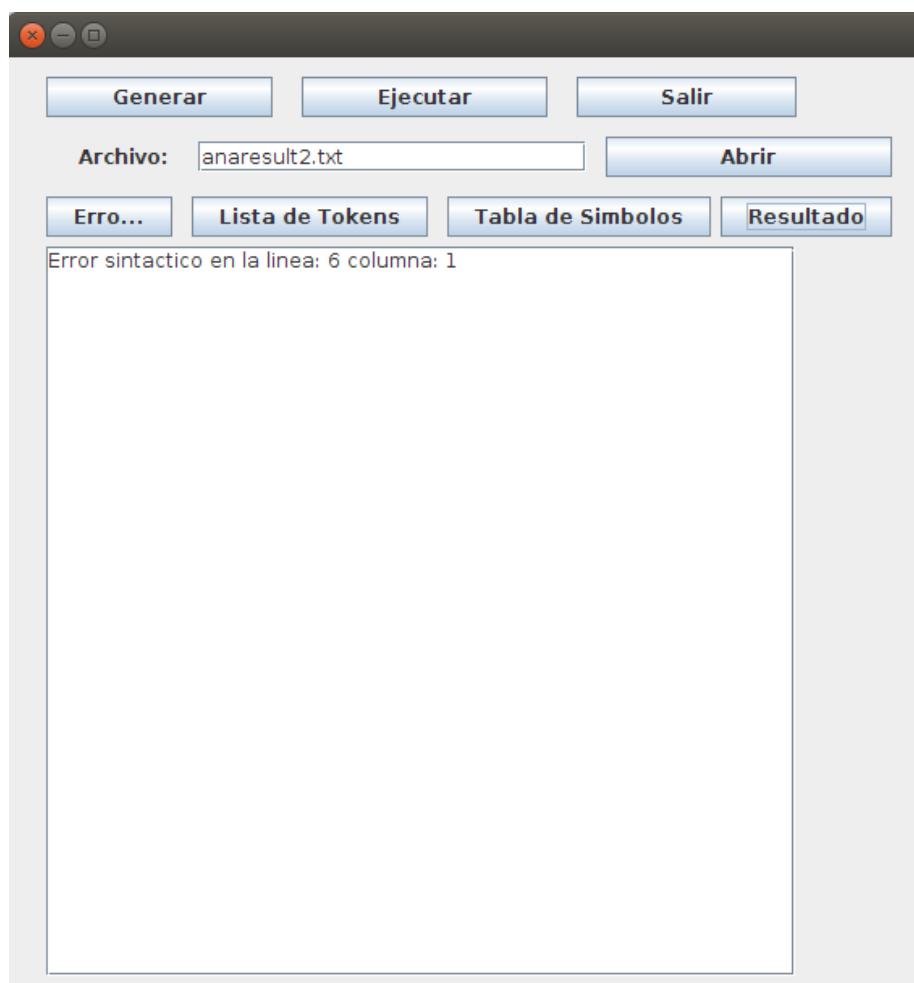
1. El primer error es un error semántico que se presenta cuando asignamos a una variable un tipo (Arreglo, Cadena o Entero) y otra variable con otro tipo y si intenta realizar una operación con tipos diferentes, el programa detecta esta acción como incorrecta, ya que los tipos de las variables no coinciden.

```
/*Analisis de resultados: Operacion con diferentes tipos de variables*/  
my $num = 5;  
my $str = "Hola";  
  
my $var = $num+$str;
```



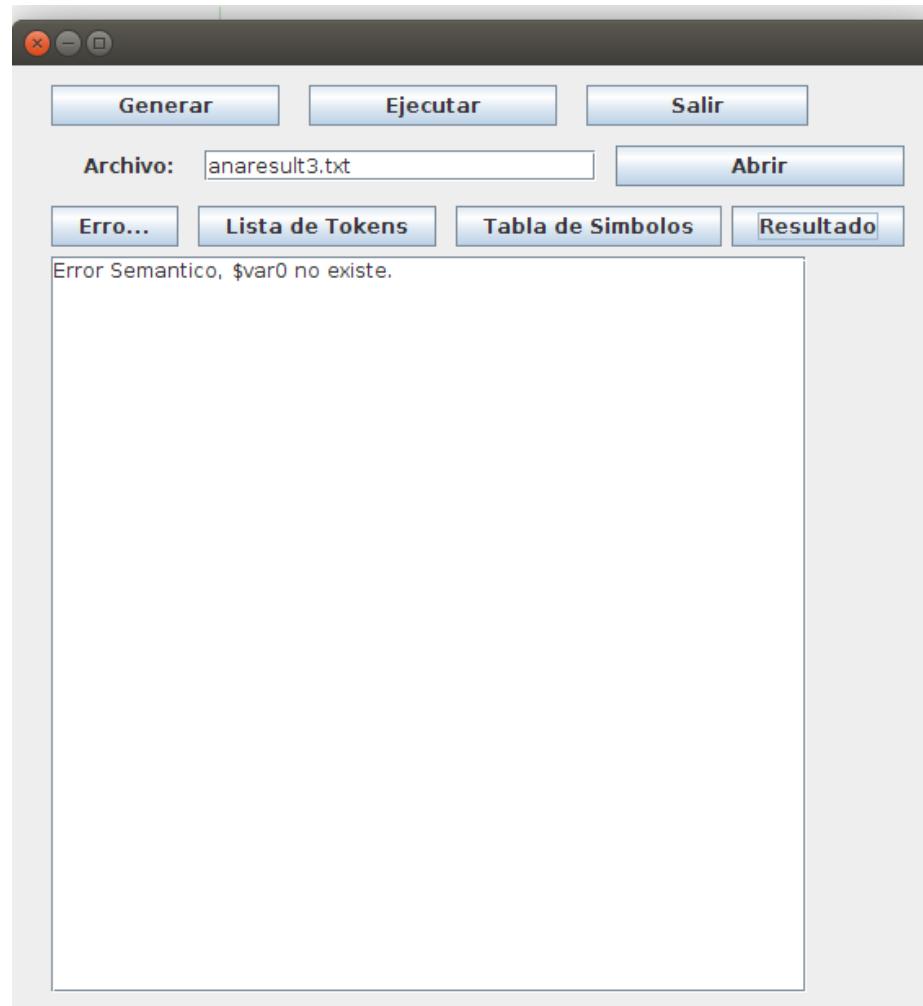
2. El segundo error semántico consiste en la asignación de variables con el mismo nombre, esto no está permitido en el lenguaje definido, buscando así evitar ambigüedades en los posibles programas a generar por el lenguaje.

```
/*Re-asignacion de variables en el codigo*/  
  
my $var1 = 5;  
my $var2 = 4;  
my $var3 = 3  
my $var1 = "5+4+3";
```



3. El tercer error, consiste en la verificacion de la declaracion de las variables, en el momento de utilizar las mismas, detectando como error semanticoo, la utilizacion de variables no iniciadas o no declaradas anteriormente.

```
/*Variables a utlizar sin declaracion previa*/  
  
my $var1 = 5;  
my $var2 = 4;  
my $var3 = $var0+$var1;
```



6 Conclusiones

Como podemos notar con las definiciones brindadas al inicio del documento, el proceso de un compilador o intérprete es un proceso un tanto complicado, en el que cada parte del proceso por separado, debe estar correctamente definida e implementada, para lograr así el objetivo propuesto: la capacidad de ejecutar una serie de instrucciones escritas en un determinado lenguaje. Para iniciar es necesario la identificación correcta del alfabeto; hecho esto, se procede a la definición de la gramática BNF, y de ahí a la construcción de cada una de las herramientas: Scanner, Parser y analizador semántico e intérprete.

El objetivo de la primera parte del proyecto consistió en realizar un analizador léxico para el lenguaje basado en Perl llamado MicroPerl, identificando los tokens permitidos que el lenguaje va a aceptar y mostrando un error en caso de que el token no sea permitido; este proceso se da por medio de la utilización de la herramienta JFlex, la cual permite, por medio de un archivo de configuración, determinar los tokens o palabras reservadas que el programa debe aceptar; únicamente basta generar la clase que tendrá toda la información, para que el programa empiece a aceptar o rechazar tokens, en caso de que un token sea rechazado se le informará al usuario, permitiendo por medio de la interfaz gráfica identificar el tipo de error y la línea en la que se detectó el error; para el analizador léxico, el programa utilizado usa autómatas y expresiones regulares para realizar el trabajo de identificación de tokens, permitiendo aceptar o rechazar hileras recibidas por el programa, de parte del usuario.

El objetivo de la segunda parte del proyecto consistió en realizar un analizador sintáctico para MicroPerl. Este analizador sintáctico se realizó mediante la herramienta CUP, la cual genera el analizador utilizando un archivo de configuración donde se especifica la gramática BNF definida por los miembros de nuestro grupo, además de los símbolos que serán aceptados por el lenguaje. El analizador sintáctico mostrará los errores sintácticos de los programas digitados por el usuario en caso de encontrar código que no cumpla con el BNF establecido en el BNF. Mediante la implementación de un programa que se encarga de realizar el análisis léxico y sintáctico, permitió comprender la manera en que están implementados la mayoría de lenguajes que existen en la actualidad, en la cual se definen una serie de tokens permitidos y una serie de reglas estructurales a seguir, permitiendo generar instrucciones que sean entendibles para el ser humano y ejecutadas por las computadoras.

La tercera y ultima parte del proyecto consistió en realizar el analizador semántico para MicroPerl. Para lo que se utilizó una tabla de símbolos, la cual permitía analizar que las variables estuvieran correctamente definidas, así como detectar posibles errores semánticos, además de desarrollar un generador de código, capaz de ejecutar las instrucciones del lenguaje MicroPerl. Cumpliendo así con el objetivo primordial de todo el proyecto desarrollado a lo largo del semestre, el cual fue el comprender el funcionamiento y desarrollo de las partes involucradas en la construcción de compiladores e interpretes, esto por medio del desarrollo de un lenguaje propio construido desde cero al inicio del semestre, terminando con un programa capaz de interpretar el código en el lenguaje MicroPerl generando

las acciones correspondientes, permitiendo comprender el funcionamiento y la relación entre las mismas de las partes necesarias para construir un Interprete o Compilador según sea el caso.

7 Apéndice 1: Notación BNF

Con respecto al BNF, se realizaron cambios, tal como se indicó en el documento, en las sentencias de algunos instrucciones, en el manejo de errores y en la consideración de los espacios como parte del lenguaje y del BNF, aparte de eso la gramática por parte del lenguaje MicroPerl, se mantuvo de acuerdo a los entregables anteriores.

También se eliminaron algunas declaraciones innecesarias en la gramática BNF anterior, además otras instrucciones se extendieron en varias derivaciones para mayor comodidad y entendimiento de la gramática BNF y para reducir las repeticiones innecesarias de derivaciones en la gramática. A continuación se adjunta el BNF con las modificaciones respectivas:

- programa → lista_declaraciones
- lista_declaraciones → lista_declaraciones declaracion | declaracion | error
- declaracion → declaracion_variable ; | declaracion_funcion linea_espacio | declaracion_bloque
- epsilon →
- espacios → espacio ESPACIO | ESPACIO
- linea_espacio → espacios | epsilon
- declaracion_variable → MY linea_espacio \$ ID linea_espacio = linea_espacio expresion; linea_espacio | MY linea_espacio ID linea_espacio ASIGNACION linea_espacio [linea_espacio parametros]; linea_espacio | MY linea_espacio ID linea_espacio = linea_espacio expresion; linea_espacio | ID linea_espacio = linea_espacio expresion; linea_espacio | \$ ID linea_espacio = linea_espacio expresion; linea_espacio | ID linea_espacio = linea_espacio [linea_espacio parametros]; linea_espacio
- declaracion_funcion → MY linea_espacio FUN linea_espacio ID linea_espacio (linea_espacio parametros) linea_espacio { linea_espacio lista_declaraciones } | FUN linea_espacio ID linea_espacio (linea_espacio parametros) linea_espacio ;
- parametros → parametros , linea_espacio parametro | parametro
- parametro → expresion

- declaracion_bloque → declaracion_seleccion | declaracion_iteracion | declaracion_retorno ; linea_espacio | expresion ; linea_espacio
- declaracion_seleccion → condicion_if | SWITCH linea_espacio (linea_espacio expresion_simple) linea_espacio { linea_espacio condicion_case DEFAULT linea_espacio : linea_espacio { linea_espacio lista_declaraciones} linea_espacio }
- condicion_if → IF linea_espacio (linea_espacio expresion_logica) linea_espacio { linea_espacio lista_declaraciones } linea_espacio | IF linea_espacio (linea_espacio expresion_logica) linea_espacio { linea_espacio lista_declaraciones } linea_espacio condicion_elsif ELSE linea_espacio { lista_declaraciones } | IF linea_espacio (linea_espacio expresion_logica) linea_espacio { linea_espacio lista_declaraciones } linea_espacio ELSE linea_espacio { lista_declaraciones }
- condicion_elsif → condicion_elsif ELSIF linea_espacio (linea_espacio expewaion_logica) linea_espacio { linea_espacio lista_declaraciones } linea_espacio | ELSIF linea_espacio (linea_espacio expewaion_logica) linea_espacio { linea_espacio lista_declaraciones } linea_espacio
- condicion_case → condicion_case CASE linea_espacio expresion : linea_espacio { linea_espacio lista_declaraciones } linea_espacio | CASE linea_espacio expresion : linea_espacio { linea_espacio lista_declaraciones } linea_espacio
- declaracion_iteracion → WHILE linea_espacio (linea_espacio expresion_logica) linea_espacio { linea_espacio lista_declaraciones } linea_espacio | FOR linea_espacio (linea_espacio declaracion_variable expresion_logica ; incrementador) linea_espacio { linea_espacio lista_declaraciones } linea_espacio | FOREACH linea_espacio (linea_espacio \$ ID linea_espacio , @ ID linea_espacio) linea_espacio { linea_espacio lista_declaraciones } linea_espacio | DO linea_espacio { linea_espacio lista_declaraciones } linea_espacio | WHILE linea_espacio (linea_espacio expresion) linea_espacio ; linea_espacio
- declaracion_retorno → RETORNO linea_espacio expresion | PRINT linea_espacio (linea_espacio expresion) | BREAK
- expresion → valor operador_logico expresion | valor operador_aritmetico expresion | expresion_simple
- expresion_simple → incrementador | operador_lista linea_espacio | valor

- $\text{expresion_logica} \rightarrow \text{expresion_simple operador_logico expresion_simple}$
- $\text{operador_logico} \rightarrow \text{EQUIVALENCIA linea_espacio} \mid \text{AND linea_espacio}$
 $\mid \text{OR linea_espacio} \mid \text{OROR linea_espacio} \mid \text{DIFERENCIA linea_espacio} \mid$
 $\mid \text{MAYORQUE linea_espacio} \mid \text{MAYORIGUALQUE linea_espacio} \mid \text{MENORQUE}$
 $\mid \text{MENORIGUALQUE linea_espacio}$
- $\text{operador_aritmetico} \rightarrow + \text{ linea_espacio} \mid - \text{ linea_espacio} \mid * \text{ linea_espacio} \mid /$
 linea_espacio
- $\text{operador_lista} \rightarrow \text{shift (expresion)} \mid$
 $\text{unshift (@ ID , expresion)} \mid$
 $\text{length (@ ID)} \mid$
 $\text{pop (@ ID)} \mid$
 $\text{push (@ ID , expresion)} \mid$
 $\text{join (@ ID , expresion)} \mid$
 $\text{split (tipo ID , expresion)}$
- $\text{valor} \rightarrow \text{NUMERO linea_espacio} \mid \text{STRINGCOMPLEJO linea_espacio} \mid \$$
 $\text{ID linea_espacio} \mid @ \text{ID}$
- $\text{incrementador} \rightarrow \$ \text{ ID } ++ \text{ linea_espacio} \mid \text{NUMERO } ++ \text{ linea_espacio} \mid$
 $\$ \text{ ID } - \text{ linea_espacio} \mid \text{NUMERO } - \text{ linea_espacio}$
- $\text{NUMERO} = \text{NUMERO} = 0 \mid [1-9][0-9]^*$
- $\text{EXP_DIGITO} = [0-9]$
 $\text{EXP_ALPHA} = [\text{A-Z}] \mid [\text{a-z}]$
 $\text{EXP_ALPHANUMERIC} = \text{EXP_ALPHA} \mid \text{EXP_DIGITO} \mid \text{ID} = \text{EXP_ALPHA} (\text{EXP_ALPHANUMERIC})^*$

8 Apéndice 2: Programas de ejemplo

- Prueba 1:

```
/*Programa sin errores #1*/  
  
my $variable1 = 3;  
my $variable2 = 2;  
if($variable1>$variable2)  
{  
    my $contador = 0;  
    while($contador<3)  
    {  
        print("variable 1 es mayor");  
        $contador++;  
    }  
}  
else  
{  
    my $contador = 2;  
    while($contador>0)  
    {  
        print("variable 2 es mayor");  
        $contador--;  
    }  
}
```

- Prueba 2:

```
/*Programa sin errores #2*/  
  
my $variable1 = 3;  
my $variable2 = 0;  
  
for(my $contador=0;$contador<$variable1;$contador++)  
{  
    switch($contador)  
    {  
        case 0:  
        {  
            $variable2 = $variable1 - 1;  
            break;  
        }  
        case 1:  
        {  
            $variable2 = $variable1 - 2;  
            break;  
        }  
        default:  
        {  
            $variable2 = 90;  
            break;  
        }  
    }  
    print($variable2);  
}
```

- Prueba 3:

```
/*Programa sin errores #3*/  
  
my @begin = ["a","b","c"];  
  
my fun funcionx($parametro1,$parametro2)  
{  
    my $tam = length(@begin);  
  
    while($parametro1<=$tam)  
    {  
        print(pop(@begin));  
  
        push(@begin,$parametro2);  
  
        print(shift(@begin));  
  
        unshift(@begin,"d");  
  
        $parametro1++;  
  
        $tam = length(@begin);  
    }  
}  
fun funcionx(1,"x");
```

- Prueba 4:

```
/*Programa sin errores #4*/  
  
/*split: Divide un string psts convertirl en un array a partir del patron indicado*/  
my $cadena = "uno&dos&tres&cuatro";  
my @lista = split($cadena,"&");  
  
/*join: Convierte un array en un string mediante el separador indicado*/  
my @numeros = ["uno","dos","tres","cuatro","cinco"];  
my $string = join(@numeros,"&");  
  
print($string);  
  
foreach($elem,@lista)  
{  
    print($elem);  
}  
/*retorna un arreglo con los valores separados por espacio*/
```

- Prueba 5:

```
/*prueba 5 error lexico #1 simbolos fuera del alfabeto*/  
#^!\?
```

- Prueba 6:

```
/*prueba 6 error lexico #1 simbolos fuera del alfabeto*/  
my @begin=["a","b","c"];  
/*Error lexico en el simbolo fuera del alfabeto ., ademas de la manera incorrecta de llamar una funcion*/  
print(@begin.length());  
print(length(@begin));  
/*Simbolos fuera del alfabeto*/  
#!?  
/*Simbolo _ fuera del alfabeto establecida al lenguaje*/  
$hola_mundo
```

- Prueba 7:

```
/*Prueba 7, error sintactico 1 falta de simbolos de cierre */
my @begin=[ "A","b","c"]
my @variable=[1,2,3];
my $variable=4;
while($variable=>5)
{
    variable++;
```

- Prueba 8:

```
/*Segundo problema sintactico simbolos del alfabeto no esperados*/
/*falta el my necesario en la gramatica BNF ademas de asignacion incorrecta*/
my $var1 = 5;
mientras($variable=5)
{
    print("Hola");
}
```

- Prueba 9:

```
/*Error semantico numero 1*/
my $var1 = 5;
my $var2 = "hola";

/*Diferentes tipos Entero + Cadena*/
my $var3 = $var1 + $var2;

print($var3);
```

- Prueba 10:

```
/*Error semantico numero 2 error logico*/
/*Re-declaracion de una variable ($var1)*/

my $var1 = 5;
my $var1 = 2;

print($var1);
```

9 Apéndice 3: Archivo de configuración

Archivo de configuración para generar el analizador léxico con Jflex: alexico.flex

```
/* -----Codigo de Usuario----- */
package interpreteperlasintactico;
import java_cup.runtime.*;
import java.io.Reader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

//clase de los token devueltos
class Ytoken {
    Ytoken (int numToken, String token, String tipo, int linea, int columna)
    {
        //Contador para el número de tokens reconocidos
        this.numToken = numToken;
        //String del token reconocido
        this.token = new String(token);
        //Tipo de componente léxico encontrado
        this.tipo = tipo;
        //Número de linea
        this.linea = linea;
        //Columna donde empieza el primer carácter del token
        this.columna = columna;
    }
    //Métodos de los atributos de la clase
    public int numToken;
    public String token;
    public String tipo;
    public int linea;
    public int columna;
    //Metodo que devuelve los datos necesarios que escribiremos en un archive de salida
    public String toString() {
        return "Token #"+numToken+": "+token+" "+tipo+" [ "+(linea+1)
        + "," +(columna+1) + " ]";
    }
}

/* //inicio de opciones
/* ----- Sección de opciones y declaraciones de JFlex ----- */

/*
    Cambiamos el nombre de la clase del analizador a Lexer
*/

%class AnalizadorLexico

/*
    Activar el contador de lineas, variable yyline
    Activar el contador de columna, variable yycolumn
*/
%line
%column

/*
    Activamos la compatibilidad con Java CUP para analizadores
    sintacticos(parser)
*/
%cup

/*
    Declaraciones

    El código entre %{ y %} sera copiado integralmente en el
    analizador generado.
*/
%{

    private int contador;
    private ArrayList<Ytoken> tokens;
    0
```

```

private void writeOutputFile(String file,Ytoken t) throws IOException
{
    String filename = file;
    BufferedWriter out = new BufferedWriter(new FileWriter(filename,true));
    // System.out.println("\n*** Tokens guardados en archivo ***\n");

    System.out.println(t);
    out.write(t + "\n");

    out.close();
}

private void writeOutputFileError(String file,String error) throws IOException
{
    String filenameerror = file;
    BufferedWriter out = new BufferedWriter(new FileWriter(filenameerror,true));
    //System.out.println("\n*** Tokens guardados en archivo ***\n");

    out.write(error + "\n");
    out.close();
}

'Obtenemos lista de tokens
public ArrayList<Ytoken> getListatokens() throws IOException
{
    System.out.println("\n*** Lista***\n");
    for(Ytoken t: this.tokens)
    {
        System.out.println(t);
    }
    return this.tokens;
}

/* Generamos un java_cup.Symbol para guardar el tipo de token
encontrado */
private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
}

/* Generamos un Symbol para el tipo de token encontrado
junto con su valor */
private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);
}

}

%init{
try {
    contador = 0;
    tokens = new ArrayList<Ytoken>();
    BufferedWriter out = new BufferedWriter(new FileWriter("file.out"));
    out.close();

    BufferedWriter outError = new BufferedWriter(new FileWriter("error.out"));
    outError.close();

} catch (IOException ex)
{
    System.out.println("error: "+ex);
}
%init}

```

```

%eof{
    /*try
    {
        this.writeOutputFile("file.out");
        //System.exit(0);
    }
    catch(IOException ioe)
    {
        ioe.printStackTrace();
    }*/
%eof}
/*
    Macro declaraciones

    Declaramos expresiones regulares que despues usaremos en las
    reglas lexicas.
*/

/* Un salto de linea es un \n, \r o \r\n dependiendo del SO   */
Salto = [\r|\n|\r\n]

/* Espacio es un espacio en blanco, tabulador \t, salto de linea
   o avance de pagina \f, normalmente son ignorados */
Espacio = [Salto] | [ \t\f]

/* Una literal entera es un numero 0 o System.out.println("\n*** Generado " + archNombre + "***\n"); un digito del 1 al 9
   seguido de 0 o mas digitos del 0 al 9 */
NUMERO = 0 | [1-9][0-9]*
EXP_DIGITO = [0-9]
EXP_ALPHA = [A-Z] | [a-z]
EXP_ALPHANUMERIC=(EXP_ALPHA)|(EXP_DIGITO)
IDENTIFICADOR=(EXP_ALPHA)((EXP_ALPHANUMERIC)*)
COMILLA=[""]
COMILLASIMPLE=['']
STRINGCOMPLEJO = {COMILLA}[a-zA-Z0-9\-\+\^\^/\@\$\$\^\\\$]+{COMILLA}

CERRARCOMENTARIO = [/*/*]
COMENTARIO = {INICIOCOMENTARIO}.*{CERRARCOMENTARIO}

*/ //fin de opciones
/* ----- Seccion de reglas lexicas ----- */

/*
    Esta seccion contiene expresiones regulares y acciones.
    Las acciones son código en Java que se ejecutara cuando se
    encuentre una entrada valida para la expresion regular correspondiente */

/* YYINITIAL es el estado inicial del analizador lexico al escanear.
   Las expresiones regulares solo serán comparadas si se encuentra
   en ese estado inicial. Es decir, cada vez que se encuentra una
   coincidencia el scanner vuelve al estado inicial. Por lo cual se ignoran
   estados intermedios. */

<YYINITIAL> {
    /*se establece la palabra reservada ASIGNACION*/
    "=" {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"ASIGNACION",yyline,yycolumn);
        tokens.add(t);
        /*----- SAVE -----*/
        writeOutputFile("file.out",t);
        /*-----*/
        //System.out.print(" = ");
        return symbol(sym.ASIGNACION);
    }

    /*Se establece la palabra reservada EQUIVALENCIA*/
    "==" {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"EQUIVALENCIA",yyline,yycolumn);
        tokens.add(t);
    }
}

```

```

        return symbol(sym.EQUIVALENCIA);
    }
/*Se establece el token WRITE*/
"!="
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"DIFERENCIA",yyline,yycolumn);
    tokens.add(t);
    /*----- SAVE -----*/
    writeOutputFile("file.out",t);
    /*----- */
    //System.out.print(" != ");
    return symbol(sym.DIFERENCIA);
}

/*Se establece el token AND*/
"&"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"AND",yyline,yycolumn);
    tokens.add(t);
    /*----- SAVE -----*/
    writeOutputFile("file.out",t);
    /*----- */
    //System.out.print(" & ");
    return symbol(sym.AND);
}

/*TOKEN para reconocer la ANDAND*/
"&&"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"ANDAND",yyline,yycolumn);
    tokens.add(t);
    /*----- SAVE -----*/
    writeOutputFile("file.out",t);
    /*----- */
    //System.out.print(" && ");

    return symbol(sym.ANDAND);
}

/* Regresa que el token PUNTOCOMA declarado en la clase sym fue encontrado. */
";"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"OR",yyline,yycolumn);
    tokens.add(t);
    return symbol(sym.OR);
}

"||"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"OROR",yyline,yycolumn);
    tokens.add(t);
    return symbol(sym.OROR);
}

/* Regresa que el token MAS declarado en la clase sym fue encontrado. */
"+"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"MAS",yyline,yycolumn);
    tokens.add(t);
    return symbol(sym.MAS);
}

/* Regresa que el token MASMAS declarado en la clase sym fue encontrado. */
"++"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"MASMAS",yyline,yycolumn);
    tokens.add(t);
    return symbol(sym.MASMAS);
}

/* Regresa que el token MULTI declarado en la clase sym fue encontrado. */
"**"
{

```

```

    }
    "==" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"MULTI",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.MULTI);
        }
    }
    "!=" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"MENOS",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.MENOS);
        }
    }
    "==" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"MENOSMENOS",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.MENOSMENOS);
        }
    }
    "/"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"DIV",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.DIV);
        }
    }
    "<" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"MENORQUE",yyline,yycolumn);
            tokens.add(t);
            return symbol(sym.MENORQUE);
        }
    }
    "<=" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"MENORIGUALQUE",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.MENORIGUALQUE);
        }
    }
    ">" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"MAYORQUE",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.MAYORQUE);
        }
    }
    ">=" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"MAYORIGUALQUE",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.MAYORIGUALQUE);
        }
    }
    "(" {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"ABRIRPARENTESIS",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.ABRIRPARENTESIS);
        }
    }
    ")"
    {
        {
            contador++;
        }
    }

```

```

        Ytoken t = new Ytoken(contador,yytext(),"CERRARPARENTESIS",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.CERRARPARENTESIS);
    }

"[" {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"ABRIRBRACKETS",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.ABRIRBRACKETS);
}

"]" {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"CERRARBRACKETS",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.CERRARBRACKETS);
}

"{" {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"ABRIRLLAVES",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.ABRIRLLAVES);
}

"}" {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"CERRARLLAVES",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);

    return symbol(sym.CERRARLLAVES);
}

";" {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"PUNTOCOMA",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.PUNTOCOMA);
}

"," {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"COMA",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.COMA);
}

";" {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"DOSPUNTOS",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.DOSPUNTOS);
}

"$" {
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),"DOLAR",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.DOLAR);
}

```

```

"@"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"AT",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.AT);
}

"my"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"MY",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.MY);
}

"fun"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"FUN",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.FUN);
}

"if"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"CONDICIONIF",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.CONDICIONIF);
}

"else"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"CONDICIONELSE",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.CONDICIONELSE);
}

"elsif"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"CONDICIONELSIF",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.CONDICIONELSIF);
}

"switch"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"CONDICIONSWITCH",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.CONDICIONSWITCH);
}

"case"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"CONDICIONCASE",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.CONDICIONCASE);
}

"default"
{
    contador++;
    Yytoken t = new Yytoken(contador,yytext(),"CONDICIONDEFAULT",yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    return symbol(sym.CONDICIONDEFAULT);
}

```

```

        }
    "while"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"BUCLEWHILE",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.BUCLEWHILE);
        }
    "do"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"BUCLEDO",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.BUCLEDO);
        }
    "for"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"BUCLEFOR",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.BUCLEFOR);
        }
    "foreach"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"BUCLEFOREACH",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.BUCLEFOREACH);
        }
    "return"
    {
        {
            contador++;

            Yytoken t = new Yytoken(contador,yytext(),"RETORNO",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.RETORNO);
        }
    "print"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"PRINT",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.PRINT);
        }
    "shift"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"SHIFT",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.SHIFT);
        }
    "unshift"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"UNSHIFT",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.UNSHIFT);
        }
    "pop"
    {
        {
            contador++;
            Yytoken t = new Yytoken(contador,yytext(),"POP",yyline,yycolumn);
            tokens.add(t);
            writeOutputFile("file.out",t);
            return symbol(sym.POP);
        }
    }
}

```

```

    }
    "push"
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"PUSH",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.PUSH);
    }
    "length"
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"LARGOLISTA",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.LARGOLISTA);
    }
    "break"
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"BREAK",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.BREAK);
    }
    "join"
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"JOIN",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.JOIN);
    }
    "split"
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"SPLIT",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.SPLIT);
    }

    /* Si se encuentra un entero, se imprime, se regresa un token numero
     que representa un entero y el valor que se obtuvo de la cadena yytext
     al convertirla a entero. yytext es el token encontrado. */
    {NUMERO}
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"NUMERO",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.NUMERO, new Integer(yytext()));
    }

    /*IDENTIFICADOR*/
    {IDENTIFICADOR}
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"ID",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.ID,new String(yytext()));
    }
    /* No hace nada si encuentra el espacio en blanco */
    {Espacio}
    {
        contador++;
        Yytoken t = new Yytoken(contador,yytext(),"ESPACIO",yyline,yycolumn);
        tokens.add(t);
        return symbol(sym.ESPACIO,new String(yytext()));
    }

```

```

(COMENTARIO) /*ignorar el comentario*/
{
    STRINGCOMPLEJO
    {
        contador++;
        Ytoken t = new Ytoken(contador,yytext(),"STRINGCOMPLEJO",yyline,yycolumn);
        tokens.add(t);
        writeOutputFile("file.out",t);
        return symbol(sym.STRINGCOMPLEJO,new String(yytext()));
    }
}

/* Si el token contenido en la entrada no coincide con ninguna regla
   entonces se marca un token ilegal */
{
    /*
    contador++;
    Ytoken t = new Ytoken(contador,yytext(),
    "Error Lexico, caracter desconocido: "+yytext()+" , en la linea "+(yyline+1)+" y la columna "+(yycolumn+1),
    yyline,yycolumn);
    tokens.add(t);
    writeOutputFile("file.out",t);
    this.writeOutputFileError("error.out",
    "Error Lexico, caracter desconocido: "+yytext()+" ,en la linea "+(yyline+1)+" y la columna "+(yycolumn+1));
    */
}

```

Archivo de configuración para el analizador sintáctico: asintactico.cup

```

/* -----Sección de declaraciones preliminares-----*/
package interpreteperlasintactico;

/* Import the class java_cup.runtime.* */
import java_cup.runtime.*;
import java.io.Reader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

parser code {
    String left;
    String right;
    /* Reporte de error encontrado. */
    public void report_error(String message, Object info) {

        if (info instanceof java_cup.runtime.Symbol) {
            java_cup.runtime.Symbol s = ((java_cup.runtime.Symbol) info);
            System.out.println("Left: "+s.left+" "+ "Right: "+s.right);
            left = new String();
            right = new String();
            if (s.left >= 0) {
                left = (s.left+1)+"";
                if (s.right >= 0)
                    right = (s.right+1)+ "";
            }
        }
    }

    public void report_fatal_error(String message, Object info) throws IOException{
        System.out.println("write: "+info);
        writeOutputFile("Error sintactico en la linea: "+left+" columna: "+right, info);
    }
}

private void writeOutputFile(String message, Object info) throws IOException
{
    String filename = "error.out";
    BufferedWriter out = new BufferedWriter(new FileWriter(filename,true));

    out.write(message+ "\n");

    out.close();
}

public static void main(String[] args){
    try {
        Analizadorsintactico asin = new Analizadorsintactico(
            new AnalizadorLexico( new FileReader(args[0])));
        Object result = asin.parse().value;
        System.out.println("\n*** Resultados finales *** \n ");
    } catch (Exception ex) {
        //ex.printStackTrace();
    }
}
};

terminal ASIGNACION,EQUIVALENCIA,DIFERENCIA,AND,ANDAND,OR, OROR, MAS,
MASMAS,MULTI,MENOS,MENOSMENOS,DIV,MENORQUE,MENORIGUALQUE,MAYORQUE,
MAYORIGUALQUE,ABRIRPARENTESIS,CERRARPARENTESIS,ABRIRBRACKETS,
CERRARBRACKETS,ABRIRLLAVES,CERRARLLAVES,PUNTOCOMA,DOSPUNTOS,COMA,
DOLAR,AT, MY,FUN,CONDICIONIF,CONDICIONELSE,CONDICIONELSIF,
CONDICIONSWITCH,CONDICIONCASE,CONDICIONDEFAULT,BUCLEWHILE,BUCLEDO,BUCLEFOR,
BUCLEFOREACH,RETORNO,PRINT,SHIFT,UNSHIFT,POP,PUSH,LARGOLISTA,BREAK,
JOIN,SLIT,ESPACIO;
terminal Integer NUMERO;
terminal String ID,STRINGCOMPLEJO;
non terminal Object     programa,lista_declaraciones,declaracion,declaracion_variable,declaracion_funcion,
                        declaracion_bloque,parametros,parametro,
                        declaracion_seleccion,declaracion_iteracion,declaracion_retorno,expresion,
                        expresion_logica,operador_logico,operador_aritmetico,valor,

```

```

        |           |           |   incrementador,operador_lista,expresion_simple,condicion_case,linea_espacio,
        |           |           |   epsilon,espacios,condicion_elsif,condicion_if ;
        |           |           |   Seccion de la gramatica -----
programa ::= linea_espacio lista_declaraciones
;

lista_declaraciones ::= lista_declaraciones declaracion|
declaracion|
error|
(
    System.out.println("Error encontrado");
    parser.report_fatal_error("Error sintactico", RESULT);
)

declaracion ::= declaracion_variable|
declaracion_funcion linea_espacio|
declaracion_bloque
;

epsilon ::= ;
;

espacios ::= espacios ESPACIO | ESPACIO
;

linea_espacio ::= espacios | epsilon
;

declaracion_variable ::= MY linea_espacio DOLAR ID linea_espacio ASIGNACION linea_espacio expresion PUNTOCOMA linea_espacio |
MY linea_espacio AT ID linea_espacio ASIGNACION linea_espacio ABIRBRACKETS linea_espacio parametros CERRARBRACKETS PUNTOCOMA linea_espacio |
MY linea_espacio AT ID linea_espacio ASIGNACION linea_espacio expresion PUNTOCOMA linea_espacio |
AT ID linea_espacio ASIGNACION linea_espacio expresion PUNTOCOMA linea_espacio |
DOLAR ID linea_espacio ASIGNACION linea_espacio expresion PUNTOCOMA linea_espacio |
AT ID linea_espacio ASIGNACION linea_espacio
ABIRBRACKETS linea_espacio parametros CERRARBRACKETS PUNTOCOMA linea_espacio

;
;

declaracion_funcion ::= MY linea_espacio FUN linea_espacio ID linea_espacio
ABIRPARENTESIS linea_espacio parametros CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio lista_declaraciones CERRARLLAVES |
FUN linea_espacio ID linea_espacio
ABIRPARENTESIS linea_espacio parametros CERRARPARENTESIS linea_espacio PUNTOCOMA
;

parametros ::= parametros COMA linea_espacio parametro | parametro
;

parametro ::= expresion
;

declaracion_bloque ::= declaracion_seleccion |
declaracion_iteracion |
declaracion_retorno PUNTOCOMA linea_espacio |
expresion PUNTOCOMA linea_espacio
;

declaracion_seleccion ::= condicion_if|
CONDICIONSWITCH linea_espacio ABIRPARENTESIS linea_espacio expresion_simple CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
condicion_case|
CONDICIONDEFAUT linea_espacio DOSPUNTOS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
CERRARLLAVES linea_espacio
;

condicion_if ::= CONDICIONIF linea_espacio ABIRPARENTESIS linea_espacio expresion_logica CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
;
;
```

```

CONDICIONES linea_espacio ABRIRPARENTESIS linea_espacio expresion_logica CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
condicion_elseif
CONDICIONELSE linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
CONDICIONELSE linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
;
condicion_elsif ::= condicion_elseif CONDICIONELSEIF linea_espacio ABRIRPARENTESIS linea_espacio expresion_logica CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
CONDICIONELSEIF linea_espacio ABRIRPARENTESIS linea_espacio expresion_logica CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
;
condicion_case ::= condicion_case
CONDICIONCASE linea_espacio expresion DOSPUNTOS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
;
declaracion_iteracion ::= BUCLEWHILE linea_espacio ABRIRPARENTESIS linea_espacio expresion_logica CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
BUCLEFOR linea_espacio
ABRIRPARENTESIS linea_espacio
declaracion_variabile /*;/* expresion_logica PUNTOCOMA incrementador CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
BUCLEFORACH linea_espacio ABRIRPARENTESIS linea_espacio DOLAR ID linea_espacio COMA linea_espacio AT ID CERRARPARENTESIS linea_espacio
ABRIRLLAVES linea_espacio
lista_declaraciones
CERRARLLAVES linea_espacio
BUCLEDO linea_espacio ABRIRLLAVES linea_espacio lista_declaraciones CERRARLLAVES linea_espacio
BUCLEWHILE linea_espacio ABRIRPARENTESIS linea_espacio expresion_logica CERRARPARENTESIS linea_espacio PUNTOCOMA linea_espacio
;
declaracion_retorno ::= RETORNO linea_espacio expresion |
PRINT linea_espacio ABRIRPARENTESIS linea_espacio expresion CERRARPARENTESIS |
BREAK
;
expresion ::= valor operador_logico expresion |
valor operador_aritmetico expresion|
expresion_simple
;
expresion_simple ::= incrementador|
operador_lista linea_espacio|
valor

```

```

;
expresion_logica      ::=      expresion_simple operador_logico expresion_simple
;

operador_logico       ::=      EQUIVALENCIA linea_espacio|
                            AND linea_espacio|
                            ANDAND linea_espacio|
                            OR linea_espacio|
                            OROF linea_espacio|
                            DIFERENCIA linea_espacio|
                            MAYORQUE linea_espacio|
                            MAYORIGUALQUE linea_espacio|
                            MENORQUE linea_espacio|
                            MENORIGUALQUE linea_espacio
;

operador_aritmetico  ::=      MAS linea_espacio|
                            MENOS linea_espacio|
                            MULTI linea_espacio|
                            DIV linea_espacio
;

operador_lista         ::=      SHIFT ABRIRPARENTESIS expresion CERRARPARENTESIS|
                                UNSHIFT ABRIRPARENTESIS AT ID COMA expresion CERRARPARENTESIS|
                                LARGOLISTA ABRIRPARENTESIS AT ID CERRARPARENTESIS|
                                POP ABRIRPARENTESIS AT ID CERRARPARENTESIS|
                                PUSH ABRIRPARENTESIS AT ID COMA expresion CERRARPARENTESIS|
                                JOIN ABRIRPARENTESIS AT ID COMA expresion CERRARPARENTESIS |
                                SPLIT ABRIRPARENTESIS DOLAR ID COMA expresion CERRARPARENTESIS|
                                SPLIT ABRIRPARENTESIS AT ID COMA expresion CERRARPARENTESIS
;

valor                  ::=      NUMERO linea_espacio|
                                STRINGCOMPLEJO linea_espacio|
                                DOLAR ID linea_espacio|
                                AT ID linea_espacio
;

incrementador          ::=      DOLAR ID MASMAS linea_espacio|
                                NUMERO MASMAS linea_espacio|
                                DOLAR ID MENOSMENOS linea_espacio|
                                NUMERO MENOSMENOS linea_espacio
;
```

10 Referencias

1. Perl Tutorial (s.f.). TutorialsPoint, Simple easy learning. Obtenido de: <http://www.tutorialspoint.com/perl/>
2. Real Academia Española. (s.f). Diccionario de la lengua española. Definición de lenguaje. Consultado en: <http://lema.rae.es/drae/?val=lenguaje>
3. ecured(s.f). Lenguaje de Programación. Recuperado de: http://www.ecured.cu/index.php/Lenguaje_de_Programación/
4. Real Academia Española. (s.f). Diccionario de la lengua española. Definición de gramática. Consultado en: <http://lema.rae.es/drae/?val=gramatica>
5. Alvarez, Gloria.(s.f.). Compiladores: Análisis Sintáctico. Obtenido de: http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:compi:comp_sesion9_2008-1.pdf
6. BNF and EBNF: What are they and how do they work? (s. f.). Obtenido de: <http://www.garshol.priv.no/download/text/bnf.html#id1.2>.
7. Serna, Eduardo.(s.f.). Compiladores. Obtenido de: <http://www.paginasprodigy.com/edserna/cursos/compilador/notas/Notas2.pdf>
8. Louden, Kenneth.(s.f). Construcción de Compiladores. Obtenido de: <http://librosysolucionarios.net/construcción-de-compiladores-principios-y-práctica-1ra-edición-kenneth-louden/>
9. Definicion.de.(s.f.). Definición de semántica. Obtenido de: <http://definicion.de/semantica/>
10. Capítulo 5. Análisis semántico.(s.f.). Obtenido de: <http://arantxa.ii.uam.es/alfonsec/docs/compila5.htm>
11. Intérpretes y Diseño de Lenguajes de Programación.(s.f.). Obtenido de: <http://di002.edv.uniovi.es/labra/FTP/Interpretes.pdf>
12. Definicion.de.(s.f.). Definición de Java. Obtenido de: <http://definicion.de/java/>
13. NetBeans IDE.(s.f.). The Smarter and Faster Way to Code. Consultado en: <https://netbeans.org/>
14. JFLEX. (s.f) JFLEX. Obtenido de: <http://jflex.de/>
15. CUP. (s.f). CUP Project. Obtenido de: <http://www2.cs.tum.edu/projects/cup/>
16. Arturo De Casso. (2014). Creación de un analizador léxico con JFlex. Obtenido de: <https://www.youtube.com/watch?v=mHFBLM4GXnk>
17. Arturo De Casso. (2014). Creación de un analizador sintáctico con JFlex y CUP en Java. Obtenido de: <https://www.youtube.com/watch?v=XQHivIfKvMk>