Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Compiladores e intérpretes - Cartago
II Semestre 2016
Proyecto #1
Hack en Español

Estudiantes:
Andrey Sánchez Delgado - 2014010852
Jose Ruiz Jara - 2014096391

Profesor:
Esteban Arias Méndez

30/11/2016

Abstract:

Computers can’t understand the high-level instructions given by the users, because of that, most of the actual programming languages must be compiled or interpreted to a language readable by computers. That is similar to native Spanish speakers who have difficulties sometimes to understand the meaning of some instructions in English, that's why programming languages with syntax in Spanish would be easier and more attractive for new enthusiasts in programming.

Índice

1.Introducción

El actual proyecto consiste en la aplicación de las bases aprendidas a lo largo del curso, acerca del funcionamiento y comportamiento de los compiladores y los intérpretes. a pesar de que ambos están diseñados con el mismo fin, cada uno de ellos realiza sus tareas de una forma diferente a la que la realiza otro. Esa diferencia difícilmente se hace notar por el usuario, porque la tarea de ambos es poder transformar instrucciones de alto nivel a un lenguaje que la máquina pueda ejecutar.

El proyecto está planteado de forma tal que pueden realizarse un scanner y un parser funcional que sea capaz de ejecutar instrucciones y tener manejo de errores, Tomando como base un lenguaje de programación existente para modificar su gramática y traducir sus instrucciones al español.

Para este caso específico se utilizará como base el lenguaje de programación Hack, el cual será interpretado mediante herramientas tipo Lex&Yacc (las cuales se mencionan más adelante). Se decidió utilizar este lenguaje por la similitud que tiene con el lenguaje PHP en su sintaxis, y porque sería interesante observar cómo sería un lenguaje orientado al desarrollo web principalmente, con sus instrucciones escritas en español.

Para dicho lenguaje se deberán Mostrar menos 10 ejemplos de programas funcionales, con o sin errores de tipos léxicos sintácticos y semánticos.

Además ha de ser capaz de poder imprimir lista de talkings del programa en ejecución , e imprimir la tabla de símbolos, La cual muestra una tabla todos los símbolos encontrados de tipo de variables funciones entre otros

Finalmente, si el código fuente de entrada es correcto (sin errores detectados) el programa deberá ejecutarlo (interpretado) y brindar los resultados o acciones que haga el código fuente que se procesa en la pantalla de ejecución de su programa

2. Objetivos

- Modificar la gramática del lenguaje hack para traducir sus instrucciones al español.
- Implementar un scanner y un parser haciendo uso de herramientas tipo Lex&Yacc.
- Utilizar el código generado con las herramientas tipo Lex&Yacc para construir un intérprete o compilador funcional.
- Implementar el manejo de errores léxicos, sintácticos y semánticos.

3.Marco Teórico

A continuación se describirán algunas de las definiciones que pueden encontrarse más adelante en este documento.
-Lenguaje: Manera de expresarse. Conjunto de signos y reglas que permite la comunicación con una computadora.

-Gramática: Parte de la lingüística que estudia los elementos de una lengua, así como la forma en que estos se organizan y se combinan.
-BNF o EBNF: Acrónimo de Backus Naur From, notación usada para especificar la sintaxis de lenguajes de programación.
-Scanner: Analizador de texto que se encarga de separar tipos primitivos y cadenas usando expresiones regulares.
-Parser: Analizador de cadenas de símbolos que forman parte de reglas de una forma gramatical.
-Analizador Semántico: Transforma el código de entrada en estructuras de datos (las mayorías de las veces en árboles,para su posterior análisis y ejecución.
-Compilador: Programa que compila o transforma un elemento a otro, en nuestro caso un compilador toma codigo fuente para convertirlo a código máquina.

Herramientas Utilizadas

Las herramientas utilizadas para la realización del proyecto son las siguientes:

- Java como lenguaje de programación
- JFLEX como herramienta tipo lex
- CUP como herramienta tipo Yacc
- Eclipse Neon como IDE
- GIT como herramienta de control de versiones
- Ubuntu 16.04 como sistema operativo

4. Desarrollo

I) Diseño del lenguaje

Hack es un lenguaje de código abierto bajo licencia BSD, desarrollado por Facebook. Es un lenguaje multiplataforma y es utilizado como una alternativa al lenguaje de programación PHP. Uno de sus principales usos es su implementación en la máquina virtual HipHop(HHVM) y gran parte del código de facebook está escrito en Hack. La única gran modificación que se le realizó al lenguaje, en la realización de este proyecto, es la

traducción de sus instrucciones al español, pues características como tipado dinámico y que es un lenguaje interpretado, se han tratado de mantener en la medida de lo posible.

II) Alfabeto y delimitadores del lenguaje

El alfabeto utilizado en nuestra implementación de hack es la siguiente:
[0-9a-zA-Z><_?():,$=.{}/+-* \t\r\f\n]
Los caracteres del alfabeto utilizados como delimitadores son los siguientes:
“,” para separar valores como parámetros de funciones
“;” para indicar el fin de una instrucción
“ \t\r\f\n” para indicar espacios o cambios de línea

III) Palabras Reservadas

"<?hh" palabra para identificar el inicio de un programa Hack, debe estar al inicio de el código fuente

"funcion"- palabra para indicar el inicio de una funcion

"sinretornar"-para indicar que una funcion no retorna datos

“como”-palabra equivalente a ‘as’ en otros lenguajes

"booleano"-utilizado para indicar valores verdaderos o falsos

"flotante"-utilizado para indicar valores de punto flotante

"entero" -utilizado para indicar valores de numeros enteros

"numero" - para indicar valores que puden ser flotantes o enteros

"recurso"- palabra para indicar que se necesita un recurso externo

"cadena"-utilizado para indicar valores string

"este"-utilizado en clases, equivalente a ‘this’

"nada"-equivalente a ‘void’ para indicar que no tiene valor almacenado

"arreglo"-para indicar el tipo de datos arreglo o lista

"pordefecto"-se utiliza cuando se va a establecer un valor predeterminado

"caso"-se utiliza en los switch(“cambiar”) para denotar las diferentes opciones que se pueden ejecutar

"y"-operador booleano equivalente a ‘and’

"o"-operador booleano equivalente a ‘or’

"verdadero"-valor booleano para denotar que un estado correcto/verdadero

"falso"-valor booleano para denotar que un estado incorrecto/falso

IV) Instrucciones

"si"- instrucción ‘if’ para ejecutar una sentencia cuando el valor condicional es verdadero

"sinosi"-instrucción ‘elseif’ para evaluar otro valor cuando un if u otro elseif no se cumple

"sino"- instrucción ‘else’ para ejecutar cuando no se cumple el valor condicional del if al que pertenece
"mientras"-equivalente a un ‘while’, se ejecuta mientras el valor condicional dado sea verdadero
"para"- equivalente a un ‘for’ donde se ejecuta un grupo de instrucciones hasta que el valor condicional sea falso
"continuar" - instrucción para salir de un ciclo y continuar con la siguiente ejecución del mismo
"detener"- equivalente a un ‘break’ para detener y salir de un ciclo
"retornar"- retorna un valor desde una función
"cambiar"- realiza la misma tarea que un ‘switch’ típico
"hacer" - se utiliza antes de un while para asegurarse que esa parte del código se ejecuta al menos una vez
"imprimir"- imprime el valor de una expresión
“caso” - hace referencia al “case” que se utilizan en los lenguaje de programación, va ligado al switch
“pordefecto” - Equivalente a la instrucción default de los lenguajes de programación, básicamente si no se da ninguno de los “casos” entonces se ejecuta el código que contiene este.

V)Características Agregadas

No se agrego ninguna característica, solamente se modificó la típica estructura del switch la cual básicamente cada uno de sus case tiene 2 puntos como delimitador para indicar las sentencias que en caso de que se cumpla la condición se deben de ejecutar, esto se cambió por llaves: “{“ “}”.

Se cambiaron los operadores binarios || y && por ‘o’ y ‘y’.

VI) Errores Léxicos, Sintácticos y Semánticos

listar los tipos de errores léxicos, sintácticos y semánticos que reconocen y ejemplos de errores, mostrando el código fuente incorrecto, el error que genera su programa (texto mostrado al usuario) y explicando por qué es un error, e indicar también cuál sería la versión correcta y qué hace su programa al respecto

5. Análisis de Resultados

A continuación se mostraran ejemplos de ejecución de los programas que se encuentran en el apéndice 2. Para cada ejemplo se presenta una versión con error y una si error.

--Ejemplo 1.
Ejecución sin error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar asign
acion.hs -s
------------------------------
Se mostrara la tabla de simbolos


30


---Tabla de simbolos---

Variable: newVariable   Valor: 30   Tipo:  entero
Variable: varNum2   Valor: 10   Tipo:  entero
Variable: varNum1   Valor: 20   Tipo:  entero
Variable: varBooleana   Valor: verdadero   Tipo:  Booleano
Variable: varString   Valor: hola_pepe   Tipo:  String
```

Ejecución con error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar asign
acion_error.hs -s
------------------------------
Se mostrara la tabla de simbolos


Error de asignacion en variable: newVariable, Tipos no compatibles
```

Error semántico.
El error en este programa se presenta debido a que se está tratando de hacer una suma entre una cadena y un entero, lo que provoca un error pues son tipos distintos y no pueden operarse entre sí. En específico al realizar “$newVariable = $varNum1 + $varBooleana;”

--Ejemplo 2.
Ejecución sin error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar break
.hs -s
------------------------------
Se mostrara la tabla de simbolos

1
2
3
4
5
6
7
8
9
10


---Tabla de simbolos---

Variable: iteraciones   Valor: 20   Tipo:  entero
```

Ejecución con error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar break
_error.hs -s
------------------------------
Se mostrara la tabla de simbolos

Error: La variable: iteracione no existe
```

Error semántico.
El error en este programa se presenta debido a que se está tratando de realizar una operación con una variable que no ha sido declarada. El error está en la siguiente instrucción: “$iteraciones = $iteracione + 1;” pues $iteraciones nunca fue declarada

--Ejemplo 3.
Ejecución sin error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar conti
nue.hs -s
------------------------------
Se mostrara la tabla de simbolos

890
220


---Tabla de simbolos---

Variable: q   Valor: 890   Tipo:  entero
Variable: u   Valor: 220   Tipo:  entero
Variable: i   Valor: 100   Tipo:  entero
```

Ejecución con error:

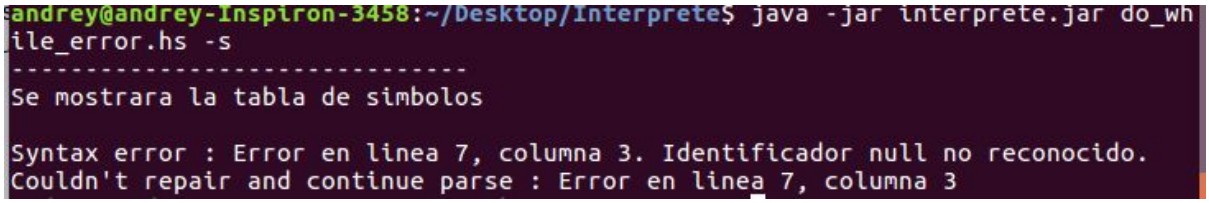
```
nue_error.hs -s
------------------------------
Se mostrara la tabla de simbolos

Exception in thread "main" java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader.main(JarRsrcLoa
der.java:58)
Caused by: compilador.ValorNoEncontrado: Error: El valor muestra_continui no se
ha encontrado.
        at compilador.CUP$Sintactico$actions.ejecutarFuncion(Sintactico.java:771
)
        at compilador.CUP$Sintactico$actions.CUP$Sintactico$do_action(Sintactico
.java:2718)
        at compilador.Sintactico.do_action(Sintactico.java:623)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:584)
        at compilador.Compilador.main(Compilador.java:55)
        ... 5 more
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Error semántico.
El error en este programa se presenta debido a que se está realizando una llamada a una función que no ha sido declarada. El error está en la siguiente instrucción: “muestra_continui();” , pues la funcion que fue declarada se llama “muestra_continue()”

--Ejemplo 4.

Ejecución sin error:

Ejecución con error:

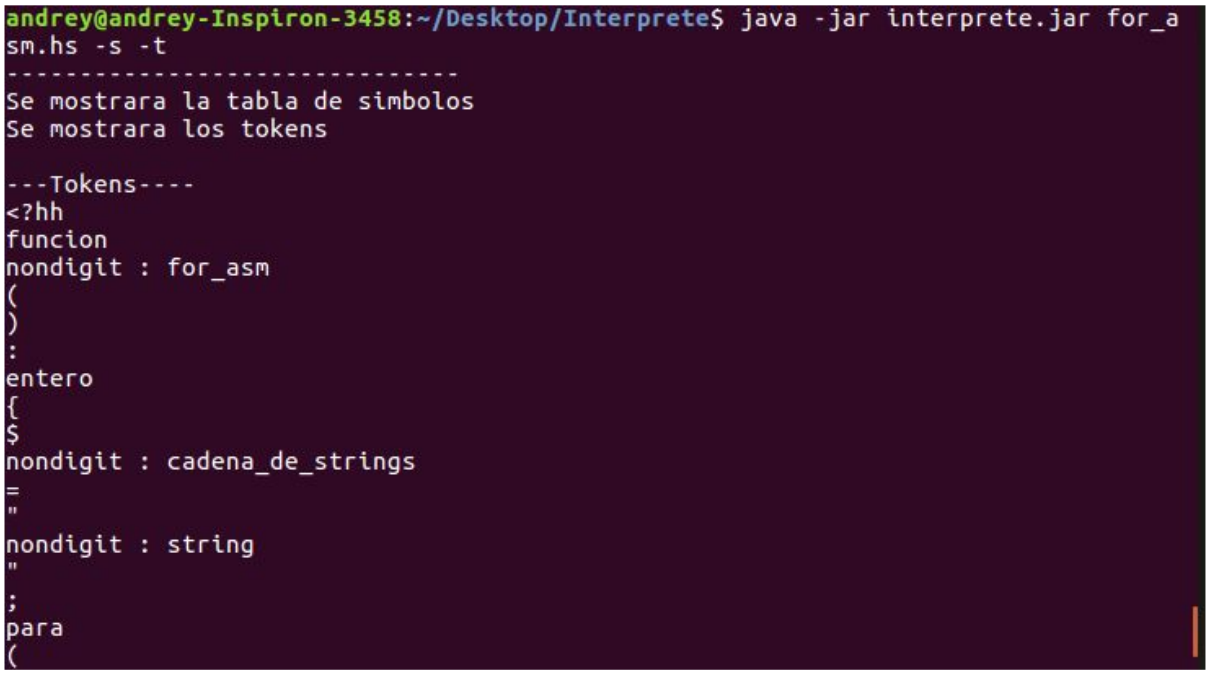


Error sintáctico.

El error en este programa se presenta debido a que falta un “;” al final de una instruccion.. El error está en la siguiente instrucción: “$res = 0”.

--Ejemplo 5 .

Ejecución sin error:



```
para
(
$
nondigit : numero_del_para
=
digito : 30
;
$
nondigit : numero_del_para
<
digito : 32
;
$
nondigit : numero_del_para
=
$
nondigit : numero_del_para
+
digito : 1
)
{
imprimir
$
nondigit : numero_del_para
```

```
$
nondigit : numero_del_para
+
digito : 2
;
}
retornar
$
nondigit : numero_del_para
;
}
nondigit : for_asm
(
)
;
32
33


---Tabla de simbolos---

Variable: numero_del_para   Valor: 32   Tipo:  entero
Variable: cadena_de_strings   Valor: string   Tipo:  String
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Ejecución con error:

```
------------------------------
Se mostrara la tabla de simbolos
Se mostrara los tokens

---Tokens----
<?hh
funcion
nondigit : for_asm
(
)
:
entero
{
$
nondigit : cadena_de_strings
=
"
nondigit : string
"
;
nondigit : forSyntax error : Error en linea 7, columna 9. Identificador for no r
econocido.
Couldn't repair and continue parse : Error en linea 7, columna 9
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Error léxico.
El error en este programa se presenta debido a que se encontró el token “for”, cuando el que deberia de usarse es “para”, al no estar declarado “for” el interprete lanza el error.

--Ejemplo 6 .
Ejecución sin error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar if_as
m.hs -s -a
------------------------------
Se mostrara la tabla de simbolos
Se mostrara el codigo asm

44444


---Tabla de simbolos---

Variable: p   Valor: 3   Tipo:  entero
Variable: jose   Valor: 1921   Tipo:  entero
Variable: varX   Valor: 44444   Tipo:  entero

--Codigo Asm Generado en instrucciones.asm--
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Ejecución con error:

```
m.hs -s -a
------------------------------
Se mostrara la tabla de simbolos
Se mostrara el codigo asm

44444


---Tabla de simbolos---

Variable: p   Valor: 3   Tipo:  entero
Variable: jose   Valor: 1921   Tipo:  entero
Variable: varX   Valor: 44444   Tipo:  entero

--Codigo Asm Generado en instrucciones.asm--
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar if_as
m_error.hs -s -a
------------------------------
Se mostrara la tabla de simbolos
Se mostrara el codigo asm

Syntax error : Error en linea 6, columna 3. Identificador null no reconocido.
Couldn't repair and continue parse : Error en linea 6, columna 3
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Error lexico.
El error en este programa se presenta debido a que se encontró el token “.”,y este no pertenece a al alfabeto del lenguaje.

--Ejemplo 7 .
Ejecución sin error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar mient
ras_asm.hs -s -a
------------------------------
Se mostrara la tabla de simbolos
Se mostrara el codigo asm

12
17
22
27
32
37
42
47


---Tabla de simbolos---

Variable: iteraciones   Valor: 52   Tipo:  entero

--Codigo Asm Generado en instrucciones.asm--
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Ejecución con error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar mient
ras_asm_error.hs -s -a
------------------------------
Se mostrara la tabla de simbolos
Se mostrara el codigo asm

Syntax error : Error en linea 6, columna 14. Identificador null no reconocido.
Couldn't repair and continue parse : Error en linea 6, columna 14
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Error sintáctico.
El error en este programa se presenta debido a que se encontró "((" en "mientras(($iteraciones < 50){", y como no puede encontrar una regla dentro de la gramática que concuerde con eso, muestra el error.

--Ejemplo 8 .
Ejecución sin error:

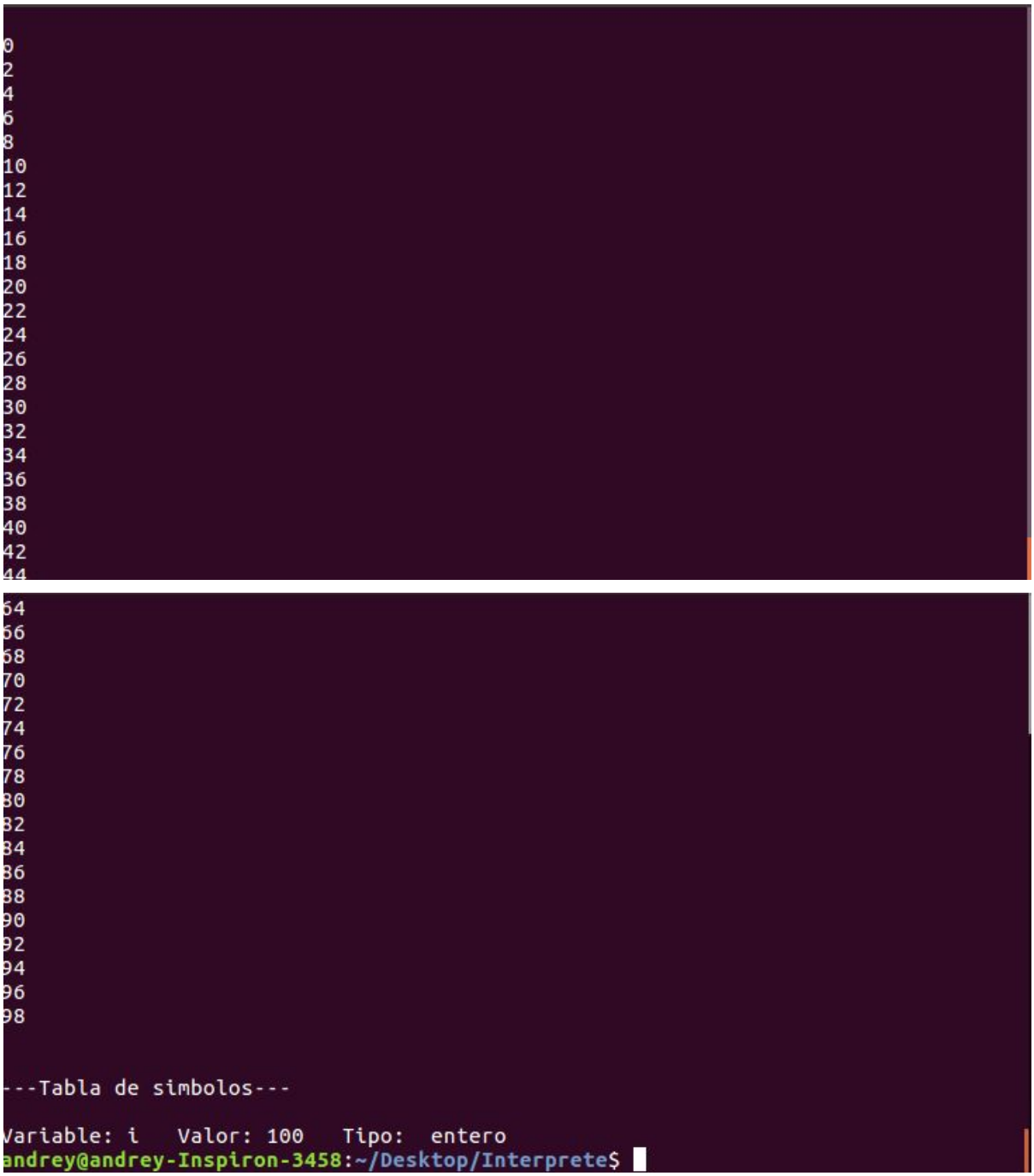
```
ras_menor_diez.hs -s
------------------------------
Se mostrara la tabla de simbolos

1
2
3
4
5
6
7
8
9
10
20
30
40
50


---Tabla de simbolos---

Variable: iteraciones   Valor: 50   Tipo:  entero
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Ejecución con error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar mient
ras_menor_diez_error.hs -s
------------------------------
Se mostrara la tabla de simbolos

Syntax error : Error en linea 7, columna 5. Identificador null no reconocido.
Couldn't repair and continue parse : Error en linea 7, columna 5
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Error lexico.
El error en este programa se presenta debido a que se encontró el token ",",y este no pertenece a al alfabeto del lenguaje.

--Ejemplo 9.
Ejecución sin error:



Ejecución con error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar numer
os_pares_error.hs -s
------------------------------
Se mostrara la tabla de simbolos

Exception in thread "main" java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader.main(JarRsrcLoa
der.java:58)
Caused by: compilador.ValorNoEncontrado: Error: El valor num_paress no se ha enc
ontrado.
        at compilador.CUP$Sintactico$actions.ejecutarFuncion(Sintactico.java:771
)
        at compilador.CUP$Sintactico$actions.CUP$Sintactico$do_action(Sintactico
.java:2718)
        at compilador.Sintactico.do_action(Sintactico.java:623)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:584)
        at compilador.Compilador.main(Compilador.java:55)
        ... 5 more
```

Error semántico.
El error en este programa se presenta debido a que se está realizando una llamada a una función que no ha sido declarada. El error está en la siguiente instrucción: “num_paress();” , pues la funcion que fue declarada se llama”num_pares()”

--Ejemplo 10.
Ejecución sin error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar switc
h.hs -s
------------------------------
Se mostrara la tabla de simbolos

0


---Tabla de simbolos---

Variable: res   Valor: 0   Tipo:  entero
Variable: valor   Valor: 23   Tipo:  entero
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Ejecución con error:

```
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$ java -jar interprete.jar switc
h_error.hs -s
------------------------------
Se mostrara la tabla de simbolos

Syntax error : Error en linea 6, columna 2. Identificador res no reconocido.
Couldn't repair and continue parse : Error en linea 6, columna 2
andrey@andrey-Inspiron-3458:~/Desktop/Interprete$
```

Error sintáctico.
El error en este programa se presenta debido a que se está utilizando mal la asignación de variables pues es necesario que todos los nombres de variable lleven “$” al inicio. El error es causado por la siguiente linea “res = 0;”

6. Conclusiones y Observaciones

Conclusiones:

- Como resultado del proyecto se obtuvo un intérprete capaz de ejecutar instrucciones de control básicas como for,while,if,else,switch,print,break,continue,case, todos en su equivalente en español. También es capaz de declaración de variables de tipo entero, booleano y string, y de las operaciones operaciones básicas de cada uno de estos tipos. Las declaraciones y llamadas a funciones son una parte importante implementada en el intérprete, ambas con sus respectivas declaración de parámetros.
- Las gramáticas de los lenguajes de programación por lo general son extensos debido a la cantidad de instrucciones que contienen y la múltiples combinaciones de éstas que se pueden formar.
- Al tener gramáticas tan extensas puede llegar a ser un poco complejo trabajar con ellas.
- Construir un intérprete, comparado con un compilador, es más sencillo de realizar, pero siempre teniendo en cuenta que cada uno de ellos tiene ventaja sobre el otro.
- Dependiendo del lenguaje que se va a implementar, se debe tomar muy en cuenta el tipo de uso que se la va a dar y la forma de como se espera que trabaje, para elegir si para su ejecución debe ser compilado o interpretado.

Observaciones:

- Se debe tratar de filtrar lo más que se pueda en el analizador léxico, todo lo que se envía al analizador sintáctico, pues de esa forma se pueden evitar problemas con algunas palabras reservadas y valores ingresados en el código que no necesariamente son palabras reservadas pero contienen caracteres en el mismo orden que estas, lo que podría llevar a que el analizador sintáctico lo interprete de manera incorrecta.
- La herramientas tipo Lex&Yacc proveen muchas de las funciones necesarias para realizar analizadores sintácticos y léxicos, pero aún así es mejor realizar una implementación propia de estos analizadores para tener un mejor control sobre los procesos de estos y los pasos a realizar, con el fin de tratar de optimizar los analizadores para el lenguaje que se va implementar.

7. Referencias

[1] Real Academia Española. (2015). Diccionario de la lengua española (23.aed.).[En linea] Consultado en http://dle.rae.es/
[2] Catb.org. (n.d.). *BNF*. [En linea] Consultado en : http://catb.org/jargon/html/B/BNF.html
[3] Docs.oracle.com. (n.d.). *Java Platform SE 8*. [En linea] Consultado en : https://docs.oracle.com/javase/8/docs/api/index.html?java/util/Scanner.html
[4] Louden, K. C.(2004). *Construcción de compiladores: principios y práctica*. Thomson

Apéndice 1: Gramática BNF

```
SCRIPT::=
        hackstart {: Sintactico.asmFinal = ""; :}DECLARATION_LIST_OPT CALL_LIST {:
Sintactico.list.interpret(); :}
       ;
CALL_LIST ::=
       {: Sintactico.valuesList.clear(); :} FUNCTION_CALL_OPT:functionTemp {:
Sintactico.list.add(functionTemp); :} CALL_LIST
       |
       ;

FUNCTION_CALL_OPT ::=
       NAME {: temp = mostrarNombre(); :} parentesis_izquierdo VALUES_DECLARED
parentesis_derecho puntoycoma {: ejecutarFuncion(); :}
       ;

VALUES_DECLARED ::=
       digit:d1 {: Sintactico.valuesList.add(d1); :} coma VALUES_DECLARED
       |digit:d1 {: Sintactico.valuesList.add(d1); :}
       |
       ;
/* The grammar */
EXPRNUM_LIST ::= EXPRNUM_LIST EXPRNUM_PART
             | EXPRNUM_PART
             ;
EXPRNUM_PART ::= EXPRNUM:e
             {: System.out.println("= " + e); :}
             ;
EXPRNUM      ::=
                EXPRNUM:e1 suma EXPRNUM:e2 {: Sintactico.aumentoWhile =
Sintactico.cantidadCiclo ;RESULT = new IntExpression(e1, e2, '+'); :}
             |EXPRNUM:e1 resta EXPRNUM:e2 {: RESULT = new IntExpression(e1, e2, '-'); :}
             |EXPRNUM:e1 multiplicacion EXPRNUM:e2 {: RESULT = new IntExpression(e1, e2,
'*');  :}
             |EXPRNUM:e1 division EXPRNUM:e2{: RESULT = new IntExpression(e1, e2, '/'); :}
             |digit:n {: Sintactico.cantidadCiclo = n; RESULT = new Number(n); :}
             |VARIABLE_NAME {: String temp = mostrarNombre(); Sintactico.asmInst+= "mov
dx, "+temp+"\n"; Sintactico.forBody= "mov dx, "+temp+"\n"; RESULT = new ConsultTable( temp
); :}
             |resta EXPRNUM:e {: RESULT = new IntExpression(e, new Number("-1"), '*');  :}
             ;

EXPRBOOL_LIST ::= EXPRBOOL_LIST EXPRBOOL_PART
                     |EXPRBOOL_PART
                     ;
EXPRBOOL_PART ::= EXPRBOOL:e
             {: System.out.println("= " + e); :}
             ;
EXPRBOOL ::=    /* Esto se tiene que hacer eficiente, esta asi para efectos de pruebas */
       |verdadero y falso {:  Sintactico.asmInst+="mov dx,1\n cmp dx,0\n jne else\n";
RESULT = new Bool("false"); :}
       |falso y verdadero {: Sintactico.asmInst+= "mov dx,0\n cmp dx,1\n jne
else\n";RESULT = new Bool("false"); :}
       |falso y falso {:  Sintactico.asmInst+= "mov dx,0\n cmp dx,0\n je else\n"; RESULT =
new Bool("false"); :}
       |verdadero y verdadero {: Sintactico.asmInst+= "mov dx,1\n cmp dx,1\n jne else\n";
RESULT = new Bool("true");:}
```

```
        |EXPRNUM:e1 tag_izquierda EXPRNUM:e2 {: Sintactico.maximoWhile =
Sintactico.cantidadCiclo;RESULT = new BoolExp(e1, e2,"<"); :}

        |verdadero o falso {: RESULT = new Bool("true"); :}
        |falso o verdadero {: RESULT = new Bool("true"); :}
        |verdadero o verdadero = {: RESULT = new Bool("true"); :}
        |falso o falso {: RESULT = new Bool("falso"); :}
        |verdadero {: RESULT = new Bool("true"); :}
        |falso {: RESULT = new Bool("false"); :}

        ;

EXPR ::=
        EXPRBOOL:e {: RESULT = e; :}
        |EXPRNUM:e {: RESULT = e; :}
        |EXPRSTRING:e {: RESULT = e; :}
        ;

EXPRSTRING ::=
        STRING_S:e {: RESULT = e; :}
        |STRING_S:e1 suma STRING_S:e2  {: RESULT = new StringExp(e1,e2,'+'); :}
        |digit:e suma STRING_S:e2 {: RESULT = new StringExp(new Number(e+""),e2,'+'); :}
        |STRING_S:e2 suma digit:e {: RESULT = new StringExp(new Number(e+""),e2,'+'); :}
        ;

ASSIGNMENT_INT ::=
        VARIABLE_NAME {: temp = mostrarNombre(); anterior = temp; :}  igual EXPR:e {:
Assignment tempp = new Assignment(temp, e);  RESULT = tempp; mostrarNombre();  :}
        ;

LIST_EXPR_VARS ::= LIST_EXPR_VARS SIGN_POSIBBLE POSIBL_EXPR
        |SIGN_POSIBBLE POSIBL_EXPR
        ;
POSIBL_EXPR ::=
        VARIABLE_NAME
        |EXPRNUM_LIST
        ;
SIGN_POSIBBLE ::=
        suma
        |resta
        |multiplicacion
        |division
        |
        ;
INCREMENT ::=
        VARIABLE_NAME suma suma {: temp = mostrarNombre(); Assignment tempp = new
Assignment(temp, new IntExpression(new Number(1), new Number( new ConsultTable(temp) ),
'+')  );  RESULT = tempp;  :}

        ;
ECHO ::=
    imprimir EXPRNUM:e1 puntoycoma {: Sintactico.asmInst+= "mov ah, 9\n int 21h\n"; RESULT
= new Imprimir(e1); :}
    | imprimir EXPRBOOL:e1 puntoycoma {: Sintactico.asmInst+= "mov ah, 9\n int
21h\n";RESULT = new Imprimir(e1); :}
    ;
/*

ECHO_INTRINSIC ::=
        imprimir EXPRESSION
        |imprimir parentesis_izquierdo EXPRESSION parentesis_derecho
        |EXPRESION_LIST_TWO_OR_MORE
```

```
        ;

EXPRESION_LIST_TWO_OR_MORE::=
        |EXPRESSION coma EXPRESSION
        |EXPRESION_LIST_TWO_OR_MORE coma EXPRESSION
        ;


INTRINSIC::=
        ECHO_INTRINSIC
        ;

*/
FOR_STATEMENT ::=
        para parentesis_izquierdo FOR_INITIALIZEROPT:e1 puntoycoma FOR_CONTROLOPT:b1
{:Sintactico.totalCiclos = Sintactico.cantidadCiclo;:} puntoycoma  FOR_END_LOOPOPT:e2
parentesis_derecho STATEMENT:e3 {: crearAsmFor();RESULT = new ForExp(e1, b1, e2, e3); :}
        ;

FOR_INITIALIZEROPT ::=
        ASSIGNMENT_INT:e1 {: RESULT = e1; :}
        |
        ;

FOR_CONTROLOPT ::=
        EXPRBOOL:b1 {: RESULT = b1; :}
        ;

FOR_END_LOOPOPT ::=
        FOR_EXPRESION_GROUP:e1 {: RESULT = e1; :}
        |
        ;

FOR_EXPRESION_GROUP ::=
        ASSIGNMENT_INT:e1 {: RESULT = e1; :}
        |INCREMENT: e2 {: RESULT = e2; :}
        /* |FOR_EXPRESION_GROUP coma EXPRESSION    Esto esta sujeto a discusion, añade
complejidad al modelo actual */
        ;
        /*
PRIMARY_EXPRESSION ::=
        INTRINSIC
        ;
        */
VECTOR_LIKE_ARRAY_TYPE_SPECIFIER ::=
        arreglo tag_izquierda ARRAY_VALUE_TYPE_SPECIFIER tag_derecha
        ;

MAP_LIKE_ARRAY_TYPE_SPECIFIER ::=
        arreglo tag_izquierda ARRAY_VALUE_TYPE_SPECIFIER coma ARRAY_VALUE_TYPE_SPECIFIER
tag_derecha
        ;

ARRAY_VALUE_TYPE_SPECIFIER ::=
  TYPE_SPECIFIER
  ;
CONTINUE_STATEMENT ::=
    continuar puntoycoma {: RESULT = new Continue(); :}
    ;


BREAK_STATEMENT ::=
```

```
    detener puntoycoma {: RESULT = new BreakExp(); :}
    ;

RETURN_STATEMENT::=
    retornar  EXPRNUM:e1 puntoycoma {: RESULT = new Return(e1); :}
    |retornar puntoycoma
    ;

WHILE_STATEMENT ::=
       mientras parentesis_izquierdo EXPRBOOL:e1 {:Sintactico.nombreVariableWhile =
temp;:}parentesis_derecho STATEMENT:s1 {: crearAsmWhile();RESULT = new WhileExp(e1, s1); :}
       ;


DO_STATEMENT ::=
       hacer STATEMENT:s1 mientras parentesis_izquierdo EXPRBOOL:e1 parentesis_derecho
puntoycoma {: RESULT = new Do_While(e1, s1); :}
       ;

IF_STATEMENT ::=
       si parentesis_izquierdo EXPRBOOL:e1 parentesis_derecho STATEMENT:s1
{:Sintactico.asmInst+= "jmp fin_else\nelse:\n";:} ELSEIF_CLAUSES_OPT sino STATEMENT:s2 {:
Sintactico.asmFinal = "fin_else:\n"; RESULT =  new IfElse(e1, s1, s2); :}
       ;
ELSEIF_CLAUSES_OPT ::=
    ELSEIF_CLAUSES
    |
    ;

ELSEIF_CLAUSES ::=
       ELSEIF_CLAUSE
       |ELSEIF_CLAUSES ELSEIF_CLAUSE        /*pot alguna razon no lo puedo hacer
opcional---- revisar*/
       ;
ELSEIF_CLAUSE ::=
       sinosi parentesis_izquierdo EXPRBOOL parentesis_derecho STATEMENT
       ;


DECLARATION_LIST_OPT ::=
    DECLARATION_LIST
    |
    ;
DECLARATION_LIST ::=
    DECLARATION
    |DECLARATION DECLARATION_LIST
    ;
DECLARATION ::=
    FUNCTION_DEFINITION
    | /*Agregar las demas partes de DECLARATION y eliminar ("parentesis_izquierdo")->*/
parentesis_izquierdo
    ;
FUNCTION_DEFINITION ::=
    ATTRIBUTE_SPECIFICATION_OPT FUNCTION_DEFINITION_NO_ATTRIBUTE:f1 {:
Sintactico.declarationList.put(f1.name, f1); :}
    ;
FUNCTION_DEFINITION_NO_ATTRIBUTE ::=
    FUNCTION_DEFINITION_HEADER:f1 STATEMENT:e1 {:  f1.setStatements(e1); RESULT = f1; :}
    ;
FUNCTION_DEFINITION_HEADER ::=
    funcion NAME {: temp = mostrarNombre(); :} /*GENERIC_TYPE_PARAMETER_LIST_OPT*/
parentesis_izquierdo {: Sintactico.parameterList  = new ArrayList(); :}PARAMETER_LIST_OPT
```

```
parentesis_derecho dos_puntos RETURN_TYPE {: RESULT = new
FunctionExpr(temp,Sintactico.parameterList); :} ;
COMPOUND_STATEMENT ::=
    corchete_izquierda {: Flow.ini(); :} STATEMENT_LIST corchete_derecha {: RESULT =
Flow.get(); :}
    ;
STATEMENT_LIST ::=
    STATEMENT:e1 {: Flow.add(e1);  :} STATEMENT_LIST
    |
    ;
ATTRIBUTE_SPECIFICATION_OPT ::=
    ATTRIBUTE_SPECIFICATION
    |
    ;
ATTRIBUTE_SPECIFICATION ::=
    doble_tag_izquierda ATTRIBUTE_LIST doble_tag_derecha
    ;
ATTRIBUTE_LIST::=
    ATTRIBUTE
    | ATTRIBUTE_LIST coma ATTRIBUTE
    ;
ATTRIBUTE::=
    ATTRIBUTE_NAME ATTRIBUTE_VALUE_LIST_OPT
    ;
ATTRIBUTE_VALUE_LIST_OPT::=
    ATTRIBUTE_VALUE_LIST
    |
    ;
ATTRIBUTE_VALUE_LIST ::=
    parentesis_izquierdo ATTRIBUTE_VALUES_OPT parentesis_derecho
    ;
ATTRIBUTE_VALUES_OPT ::=
    ATTRIBUTE_VALUES
    |
    ;
ATTRIBUTE_VALUES ::=
    ATTRIBUTE_VALUE
    | ATTRIBUTE_VALUES coma ATTRIBUTE_VALUE
    ;
ATTRIBUTE_VALUE ::=
    EXPRESSION
    ;
ATTRIBUTE_NAME ::=
    NAME
    ;
NAME ::=
    nondigit:nond {:agregarANombre(nond);:}
    | NAME nondigit:ndn {:agregarANombre(ndn);:}
    | NAME digit:d {:agregarANombre(d);:}
    ;
GENERIC_TYPE_PARAMETER_LIST_OPT ::=
    GENERIC_TYPE_PARAMETER_LIST
    |
    ;
GENERIC_TYPE_PARAMETER_LIST ::=
    tag_izquierda GENERIC_TYPE_PARAMETERS COMA_OPT tag_derecha
    ;
GENERIC_TYPE_PARAMETERS ::=
    GENERIC_TYPE_PARAMETER
    | GENERIC_TYPE_PARAMETERS coma GENERIC_TYPE_PARAMETER
    ;
GENERIC_TYPE_PARAMETER ::=
```

```
    GENERIC_TYPE_PARAMETER_VARIANCE_OPT GENERIC_TYPE_PARAMETER_NAME TYPE_CONSTRAINT_OPT
    ;
TYPE_CONSTRAINT_OPT ::=
    TYPE_CONSTRAINT
    |
    ;
TYPE_CONSTRAINT ::=
    as TYPE_SPECIFIER
    ;
GENERIC_TYPE_PARAMETER_NAME ::=
    NAME {: Sintactico.tabla.insertar(nombre, -2, -1); :}
    ;
GENERIC_TYPE_PARAMETER_VARIANCE_OPT::=
    GENERIC_TYPE_PARAMETER_VARIANCE
    |
    ;
GENERIC_TYPE_PARAMETER_VARIANCE::=
    suma
    | resta
    ;
PARAMETER_LIST_OPT ::=
    VARIABLE_NAME {: Sintactico.parameterList.add(mostrarNombre()); :} coma
PARAMETER_LIST_OPT
    |VARIABLE_NAME {: Sintactico.parameterList.add(mostrarNombre()); :}
    |
    ;
PARAMETER_LIST ::=
    puntos_suspensivos
    | PARAMETER_DECLARATION_LIST COMA_OPT
    | PARAMETER_DECLARATION_LIST coma puntos_suspensivos
    ;
PARAMETER_DECLARATION_LIST ::=
    PARAMETER_DECLARATION
    | PARAMETER_DECLARATION_LIST coma PARAMETER_DECLARATION
    ;
PARAMETER_DECLARATION ::=
        ASSIGNMENT_INT
        |VARIABLE_NAME {: Sintactico.tabla.insertar(mostrarNombre(), -2, -1); :}
    ;
DEFAUlT_ARGUMENT_SPECIFIER_OPT ::=
    DEFAUlT_ARGUMENT_SPECIFIER
    |
    ;
DEFAUlT_ARGUMENT_SPECIFIER ::=
    igual CONST_EXPRESSION
    ;
CONST_EXPRESSION ::=
    EXPRESSION
    ;
COMA_OPT ::=
    coma
    |
    ;
RETURN_TYPE ::=
    TYPE_SPECIFIER
    | noreturn
    ;
TYPE_SPECIFIER ::=
    bool
    | flotante
    | entero
    | num
```

```
    | resource
    | string
    | este
    | nada
    |VECTOR_LIKE_ARRAY_TYPE_SPECIFIER
    |MAP_LIKE_ARRAY_TYPE_SPECIFIER
    |TUPLE_TYPE_SPECIFIER
    /*TODAVIA FALTAN MAS POR HACER*/
    ;

TUPLE_TYPE_SPECIFIER ::=
       parentesis_izquierdo TYPE_SPECIFIER coma TYPE_SPECIFIER_LIST parentesis_derecho
       |tupla parentesis_izquierdo TYPE_SPECIFIER coma TYPE_SPECIFIER_LIST
parentesis_derecho
       ;
TYPE_SPECIFIER_LIST ::=
       TYPE_SPECIFIERS
       ;
TYPE_SPECIFIERS ::=
       TYPE_SPECIFIER
       |TYPE_SPECIFIERS coma TYPE_SPECIFIER
       ;

VARIABLE_NAME ::=
    dolar NAME
    ;
STRING_S ::=
       comilla NAME comilla {: RESULT = new Stringg(mostrarNombre()); :}
       ;
STATEMENT ::=
    SELECTION_STATEMENT:e1 {: RESULT = e1; :}
    |COMPOUND_STATEMENT:e1 {: RESULT = e1; :}
    |ITERATION_STATEMENT:e1 {: RESULT = e1; :}
    |JUMP_STATEMENT:e1 {: RESULT = e1; :}
    |ASSIGNMENT_INT:e1 puntoycoma {: RESULT = e1; :}
    |INCREMENT:e1 puntoycoma {: RESULT = e1; :}
    |LABELED_STATEMENT:e1 {: RESULT = e1; :}
    |ECHO:ech {: RESULT = ech; :}
    ;
SELECTION_STATEMENT ::=
    SWITCH_STATEMENT:e1 {: RESULT = e1; :}
    |IF_STATEMENT:e1 {: RESULT = e1; :}
    /*POR HACER*/
    ;

SWITCH_STATEMENT ::=
       cambiar parentesis_izquierdo EXPRNUM:e1 parentesis_derecho STATEMENT:s1 {: RESULT =
new SwitchExp(e1, s1);  :}
       ;

LABELED_STATEMENT ::=
       CASE_LABEL:e1 {: RESULT = e1; :}
       |DEFAULT_LABEL:e1 {: RESULT = e1; :}
       ;
CASE_LABEL ::=
       caso EXPRNUM:s1 STATEMENT:e1 {: RESULT = new Case(s1, e1); :}
       ;
DEFAULT_LABEL ::=
       pordefecto STATEMENT:e1 {: RESULT = new Default(e1); :}
       ;

ITERATION_STATEMENT ::=
```

```
        FOR_STATEMENT:e1 {: RESULT = e1; :}
        |DO_STATEMENT:e1 {: RESULT = e1; :}
        |WHILE_STATEMENT:e1 {: RESULT = e1; :}
        ;
JUMP_STATEMENT ::=
        RETURN_STATEMENT:e1 {: RESULT = e1; :}
        |BREAK_STATEMENT:e1 {: RESULT = e1; :}
        |CONTINUE_STATEMENT:e1 {: RESULT = e1; :}
        ;
/*  jump-statement:
    continue-statement
    break-statement                          JUMP STATEMENT STRUCTURE
    return-statement
    throw-statement */
/*
  statement:
    function-static-declaration
    compound-statement
    labeled-statement
    expression-statement             STATEMENT STRUCTURE
    selection-statement
    iteration-statement
    jump-statement
    try-statement
*/

EXPRESSION_STATEMENT ::=
                EXPRESSION
                |
        ;
EXPRESSION ::=

    nondigit
    /*|PRIMARY_EXPRESSION*/
    /*POR HACER*/
    ;
```

Apéndice 2: Programas de Ejemplo con errores y versiones corregidas

Ejemplo 1

Sin error:

```
<?hh

funcion funcion_asignacion():entero{
                $varString = "hola_" + "pepe";

                $varNum1 = 20;
                $varNum2 = 10;

                $varBooleana = 3<10;

                $newVariable = $varNum1 + $varNum2;

                imprimir $newVariable;
        retornar $newVariable;
```

```
}



funcion_asignacion();
```

Con error:

```
<?hh

funcion funcion_asignacion():entero{
                $varString = "hola_" + "pepe";

                $varNum1 = 20;
                $varNum2 = 10;

                $varBooleana = 3<10;

                $newVariable = $varNum1 + $varBooleana;

                imprimir $newVariable;
        retornar $newVariable;
}



funcion_asignacion();
```

Ejemplo 2

Sin error:

```
<?hh
funcion muestra_break():entero{
    $iteraciones = 0;
    mientras($iteraciones < 50){
        si($iteraciones < 10){
            $iteraciones =  $iteraciones + 1;
        }sino{
            $iteraciones =  $iteraciones + 10;
        detener;
        }
        imprimir $iteraciones;
    }
    retornar $iteraciones;
}



muestra_break();
```

Con error:

```
<?hh
funcion muestra_break():entero{
    $iteraciones = 0;
    mientras($iteraciones < 50){
        si($iteraciones < 10){
            $iteraciones =  $iteracione + 1;
        }sino{
```

```
            $iteraciones =  $iteraciones + 10;
           detener;
        }
        imprimir $iteraciones;
    }
    retornar $iteraciones;
}



muestra_break();
```

Ejemplo 3

Sin error:

```
<?hh

funcion muestra_continue():entero{
        $q = 0;
        $u = 0;
        para($i = 0;$i<100;$i = $i + 1){
                si(10<$i){
                        $q = $q + 10;
                        continuar;
                        $u = $u + 1;
                }
                sino{
                        $u = $u + 20;
                }
        }
        imprimir $q;
        imprimir $u;
    retornar $q;
}



muestra_continue();
```

Con error:

```
<?hh

funcion muestra_continue():entero{
        $q = 0;
        $u = 0;
        para($i = 0;$i<100;$i = $i + 1){
                si(10<$i){
                        $q = $q + 10;
                        continuar;
                        $u = $u + 1;
                }
                sino{
                        $u = $u + 20;
                }
        }
        imprimir $q;
        imprimir $u;
    retornar $q;
}
```

```
muestra_continui();
```

Ejemplo 4

Sin errores:

```
<?hh



funcion do_while($num):entero{
		$res = 0;
		hacer{
			$res = $res + 1;
		}
		mientras(10<$num);

		imprimir $res;



	retornar $res;
}



do_while(5);
```

Con errores:

```
<?hh

funcion do_while($num):entero{
		$res = 0
		hacer{
			$res = $res + 1;
		}
		mientras(10<$num);

		imprimir $res;



	retornar $res;
}

do_while(5);
```

Ejemplo 5

Sin errores:

```
<?hh



funcion for_asm():entero{
		$cadena_de_strings = "string";
	para($numero_del_para = 30; $numero_del_para<32;$numero_del_para = $numero_del_para
+ 1){
```

```
            imprimir $numero_del_para+2;
        }
        retornar $numero_del_para;
}



for_asm();
```

Con errores:

```
<?hh



funcion for_asm():entero{
               $cadena_de_strings = "string";
        for($numero_del_para = 30; $numero_del_para<32;$numero_del_para = $numero_del_para
+ 1){
            imprimir $numero_del_para+2;
        }
        retornar $numero_del_para;
}



for_asm();
```

Ejemplo 6

Sin errores:

```
<?hh



funcion asm_if():entero{
               $jose = 1921;
               $p = 3;
               $varX= 44444;
               si(verdadero y verdadero){
                      imprimir $varX;
               }sino{
                      imprimir $jose;
               }
        retornar $varX;
}



asm_if();
```

Con errores:

```
<?hh



funcion asm_if():entero{
               .
               $jose = 1921;
               $p = 3;
               $varX= 44444;
```

```
                si(verdadero y verdadero){
                        imprimir $varX;
                }sino{
                        imprimir $jose;
                }
        retornar $varX;
}


asm_if();
```

Ejemplo 7

Sin errores:

```
<?hh


funcion muestra_while($iteraciones):entero{
    mientras($iteraciones < 50){
        imprimir $iteraciones;
        $iteraciones = $iteraciones + 5;
    }
    retornar $iteraciones;
}


muestra_while(12);
```

Con errores:

```
<?hh


funcion muestra_while($iteraciones):entero{
    mientras($iteraciones < 50){
        imprimir $iteraciones;
        $iteraciones = $iteraciones + 5;
    }
    retornar $iteraciones;
}


muestra_while(12);
```

Ejemplo 8

Sin errores:

```
<?hh


funcion muestra_while():entero{
    $iteraciones = 0;
    mientras($iteraciones < 50){
```

```
        si($iteraciones < 10){
            $iteraciones =  $iteraciones + 1;
        }sino{
            $iteraciones =  $iteraciones + 10;
        }
        imprimir $iteraciones;
    }
    retornar $iteraciones;
}



muestra_while();
```

Con errores:

```
<?hh



funcion muestra_while():entero{
    $iteraciones = 0;
    ,
    mientras($iteraciones < 50){
        si($iteraciones < 10){
            $iteraciones =  $iteraciones + 1;
        }sino{
            $iteraciones =  $iteraciones + 10;
        }
        imprimir $iteraciones;
    }
    retornar $iteraciones;
}



muestra_while();
```

Ejemplo 9

Sin errores:

```
<?hh



funcion num_pares():entero{
        para($i = 0; $i < 100 ;$i = $i + 2){
            imprimir $i;
        }
        retornar $i;

}



num_pares();
```

Con errores:

```
<?hh
```

```
funcion num_pares():entero{
        para($i = 0; $i < 100 ;$i = $i + 2){
            imprimir $i;
        }
        retornar $i;

}



num_paress();
```

Ejemplo 10

Sin errores:

```
<?hh



funcion muestra_switch($valor):entero{
       $res = 0;

       cambiar($valor){
              caso 1{
                     $res = 0;
              }
              caso 2{
                     $res = 1;
                     detener;
              }
              caso 3{
                     $res = 2;
                     detener;
              }
              caso 2{
                     $res = 3;
              }
              pordefecto{
                     $res = 4;
              }
       }


       imprimir $res;
    retornar $res;
}
```

muestra_switch(23);

Con errores:

```
<?hh
```

```
funcion muestra_switch($valor):entero{
	res = 0;

	cambiar($valor){
		caso 1{
			$res = 0;
		}
		caso 2{
			$res = 1;
			detener;
		}
		caso 3{
			$res = 2;
			detener;
		}
		caso 2{
			$res = 3;
		}
		pordefecto{
			$res = 4;
		}
	}


	imprimir $res;
    retornar $res;
}



muestra_switch(23);
```

Apéndice 3:

Para poder ejecutar el proyecto, antes es necesario compilar los archivos .flex y .cup para que puedan ser transformados a clases java y poder ser ejecutadas por la JVM.
A continuación se encuentran los comandos para compilar dichos archivos:

$ Jflex Scanner.jflex

$ cup -parser Sintactico -symbols Tokens Parser.cup