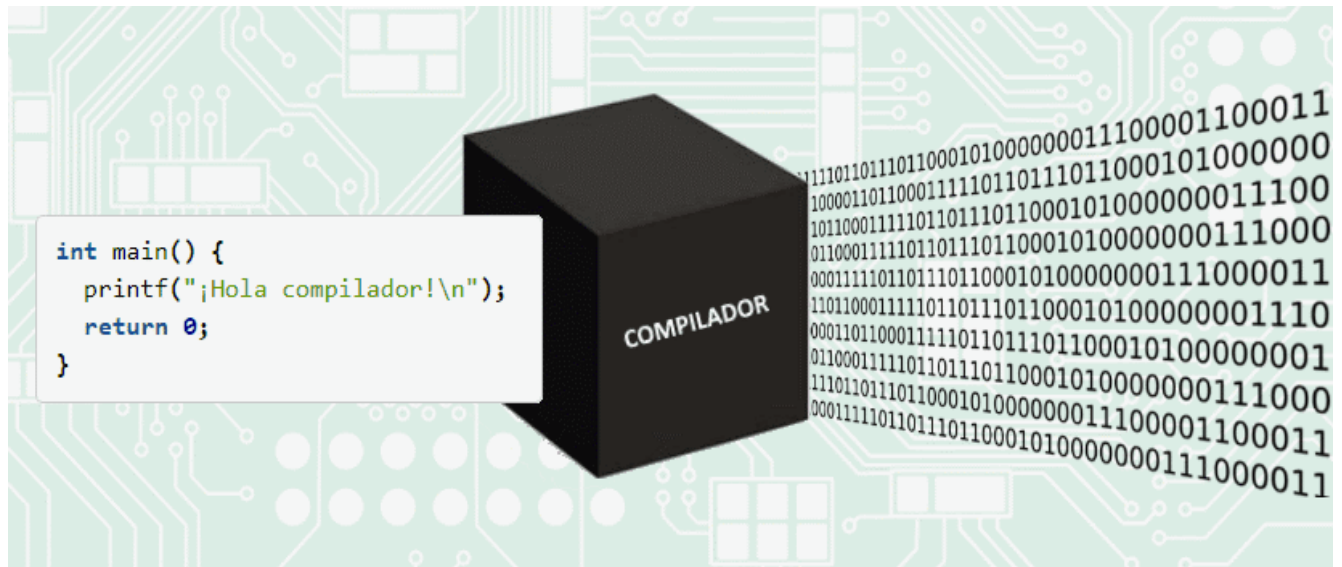


# Diseño de Compiladores 1

Grupo 10 - Tp 1 y 2



## Integrantes

Medioli Nicolas - [mediolifnicolas@gmail.com](mailto:mediolifnicolas@gmail.com)

Arias Facundo - [facundoarias1231@gmail.com](mailto:facundoarias1231@gmail.com)

## Docente asignado

Jose Fernandez Leon

## Temas asignados

6 8 12 14 18 20 21 24 28 29 32 35

---

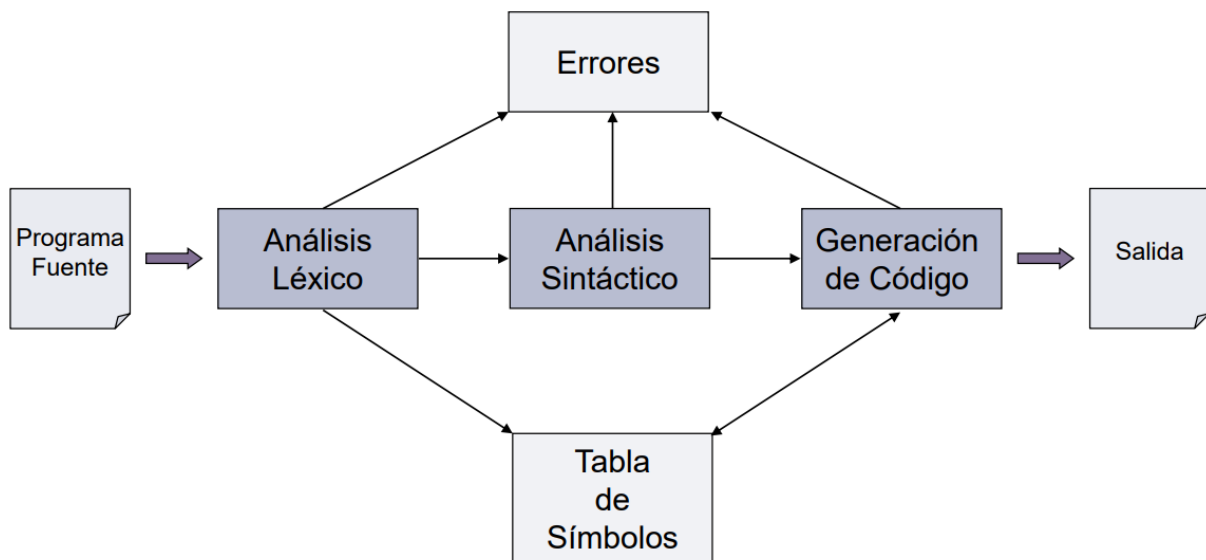
## índice

|   |           |
|---|-----------|
| <b>Introducción.....</b>                          | <b>3</b>  |
| <b>Especificaciones asignadas.....</b>            | <b>4</b>  |
| <b>Parte 1 - Analizador Léxico.....</b>           | <b>7</b>  |
| Estructura general del analizador léxico.....     | 8         |
| Implementación general - LexycalAnalyzer.....     | 9         |
| Diagrama de transición de estados.....            | 9         |
| Acciones semánticas.....                          | 11        |
| Errores léxicos considerados.....                 | 13        |
| <b>Parte 2 - Analizador Sintáctico.....</b>       | <b>14</b> |
| Especificaciones generales.....                   | 14        |
| implementaciones generales - Yacc.....            | 15        |
| Gramática.....                                    | 15        |
| Lista de no terminales.....                       | 16        |
| Errores sintácticos considerados.....             | 18        |
| <b>Casos de prueba.....</b>                       | <b>19</b> |
| Casos de prueba-Léxico.....                       | 19        |
| Casos de prueba-Sintáctico.....                   | 27        |
| <b>Correcciones realizadas - 2da entrega.....</b> | <b>37</b> |
| <b>Conclusión.....</b>                            | <b>38</b> |

---

## Introducción

En este informe se detalla el desarrollo y la implementación de un compilador y las dos primeras partes que lo conforman: Análisis Léxico y Análisis Sintáctico. Las especificaciones del mismo, fueron dadas por la cátedra.



*Figura 1: "Estructura general del compilador"*

El ejecutable del compilador se ejecuta abriendo una consola dentro de la carpeta "entrega grupo 10" y escribiendo el siguiente comando de ejemplo:

```
java -jar build\compilador.jar tests\lexico\case1.fn
```

---

## Especificaciones asignadas

**6.** Enteros largos (32 bits): Constantes enteras con valores entre  $-2^{31}$  y  $2^{31} - 1$ . Estas constantes llevarán el sufijo “\_l”.

**8.** Punto flotante de 64 bits: Números reales con signo y parte exponencial. La parte exponencial puede estar ausente. Si está presente, el exponente comienza con la letra “D” (mayúscula o minúscula) y el signo del exponente es obligatorio. Puede estar ausente la parte entera o la parte decimal, pero no ambas. El “.” es obligatorio.

Ejemplos válidos: 1. .6 -1.2 3.d-5 2.D+34 2.5D-1 13. 0. 1.2d+10

Considerar el rango  $2.2250738585072014D-308 < x < 1.7976931348623157D+308$

$-1.7976931348623157D+308 < x < -2.2250738585072014D-308$

Se debe incorporar a la lista de palabras reservadas la palabra DOUBLE.

**12.** En las sentencias de asignación, puede utilizarse el operador ‘-=’ en lugar del operador de asignación ‘=’. Por ejemplo:  $a -= 10\_i$ ,  $z -= a * b$ ,

**14.** DO UNTIL DO UNTIL ( ), tendrá la misma definición que la condición de las sentencias de selección. podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por llaves.

**18.** Herencia por Composición - Uso con nombre

|  |  |
|--|--|
| <p>Incorporar, como sentencia declarativa, la declaración de clases con la estructura que se muestra en los siguientes ejemplos:</p> <pre> CLASS ca {     INT a;c,      // declaración de atributos     VOID m() {    // declaración de método         ...     }, }  CLASS cb{     FLOAT b,      // declaración de atributo     FLOAT a,      // declaración de atributo     VOID n() {    // declaración de método         ...     },     ca,           // nombre de clase } ... </pre> <p>Incorporar, como sentencia declarativa, la declaración de objetos de una clase determinada:</p> <pre> ca a1; a2, cb b1; b2; b3, </pre> | <p>Incorporar, dentro de las sentencias ejecutables, la posibilidad de utilizar referencias a métodos y atributos de objetos indicando explícitamente cuando el atributo o método corresponden a la clase heredada por composición.</p> <p>Ejemplos:</p> <pre> a1.a = 3_i, b1.ca.a = 3_i, b1.a = 2.3, b1.b = 1.2, b1.ca.c = 1_i, b1.ca.m(), b1.n(), </pre> <p>(todos los chequeos semánticos para estas referencias, se efectuarán en la etapa 3 del TP)</p> |
|--|--|

**20.** Declaración de métodos distribuida La implementación de los métodos de una clase puede efectuarse mediante una cláusula IMPL como se indica en el siguiente ejemplo

```

CLASS ca {
    INT a;c,      // declaración de atributos
    VOID m() {    // declaración de método
        ...
    },

    VOID p(),     // prototipo de método
}

IMPL FOR ca: {
    VOID p() {    // implementación del método p de la clase ca
        ...
    },
    ...
}

```

// LÉXICO: Incorporar a los símbolos detectados, el símbolo ‘.’.

**21.** Forward declaration

---

Incorporar la posibilidad de declarar las clases con posterioridad a su uso.

Por ejemplo:

```
CLASS ca {  
    VOID m() {  
        ...  
    }  
}  
...  
CLASS cc,  
CLASS cd {  
    cc cl,  
    ca al,  
    VOID j() {  
        cl.p(),    // será un error, pero se detectará en la siguiente etapa  
        al.m(),    // ok  
    }  
}  
...  
CLASS cc {    // declaración de clase cc  
    INT z,  
    VOID p() {  
        ...  
    }  
}
```

**24.** Sobreescritura de atributos La semántica de este tema se explicará en la etapa 3 del Trabajo Práctico

**28.** La semántica de estos temas se explicará en la etapa 3 del Trabajo Práctico

**29.** Conversiones Explícitas: Se debe incorporar en todo lugar donde pueda aparecer una expresión, la posibilidad de utilizar la siguiente sintaxis:

TOF () // para grupos que tienen asignado el tema 7

TOD() // para grupos que tienen asignado el tema 8

//LÉXICO: Incorporar a la lista de palabras reservadas, la palabra TOF o TOD según corresponda.

**32.** Comentarios de 1 línea: Comentarios que comiencen con “\*\*” y terminen con el fin de línea.

**35.** Cadenas multilínea: Cadenas de caracteres que comiencen y terminen con “%”. Estas cadenas pueden ocupar más de una línea. (En la Tabla de símbolos se guardará la cadena sin los saltos de línea).

Ejemplo: % ¡Hola

mundo! %

---

## Parte 1 - Analizador Léxico

Para desarrollar el analizador léxico tuvimos en cuenta el objetivo y las especificaciones que proporciona la cátedra:

### Objetivo

Desarrollar un Analizador Léxico que reconozca los siguientes tokens:

- Identificadores cuyos nombres pueden tener hasta 20 caracteres de longitud. El primer puede ser una letra o '\_', y el resto pueden ser letras, dígitos y "\_". Los identificadores con longitud mayor serán truncados y esto se informará como Warning. Las letras utilizadas en los nombres de identificador sólo pueden ser minúsculas.
- Constantes correspondientes al tema particular asignado a cada grupo.  
**Nota:** Para aquellos tipos de datos que pueden llevar signo, la distinción del uso del símbolo '-' como operador aritmético o signo de una constante, se postergará hasta el trabajo práctico Nro. 2.
- Operadores aritméticos: "+", "-", "\*\*", "/" agregando lo que corresponda al tema particular.
- Operador de asignación: "="
- Comparadores: ">=", "<=", ">", "<", "==", "!="
- "{", "}", "(", ")", ";", ":" y ","
- Cadenas de caracteres correspondientes al tema particular de cada grupo.
- Palabras reservadas (en mayúsculas):  
**IF, ELSE, END\_IF, PRINT, CLASS, VOID**
- y demás símbolos / tokens indicados en los temas particulares asignados a cada grupo.

El Analizador Léxico debe eliminar de la entrada (reconocer, pero no informar como tokens al Analizador Sintáctico), los siguientes elementos.

- Comentarios correspondientes al tema particular de cada grupo.
- Caracteres en blanco, tabulaciones y saltos de línea, que pueden aparecer en cualquier lugar de una sentencia.

### Analizador Léxico. Especificaciones

a) El Analizador Léxico deberá leer un código fuente, identificando e informando:

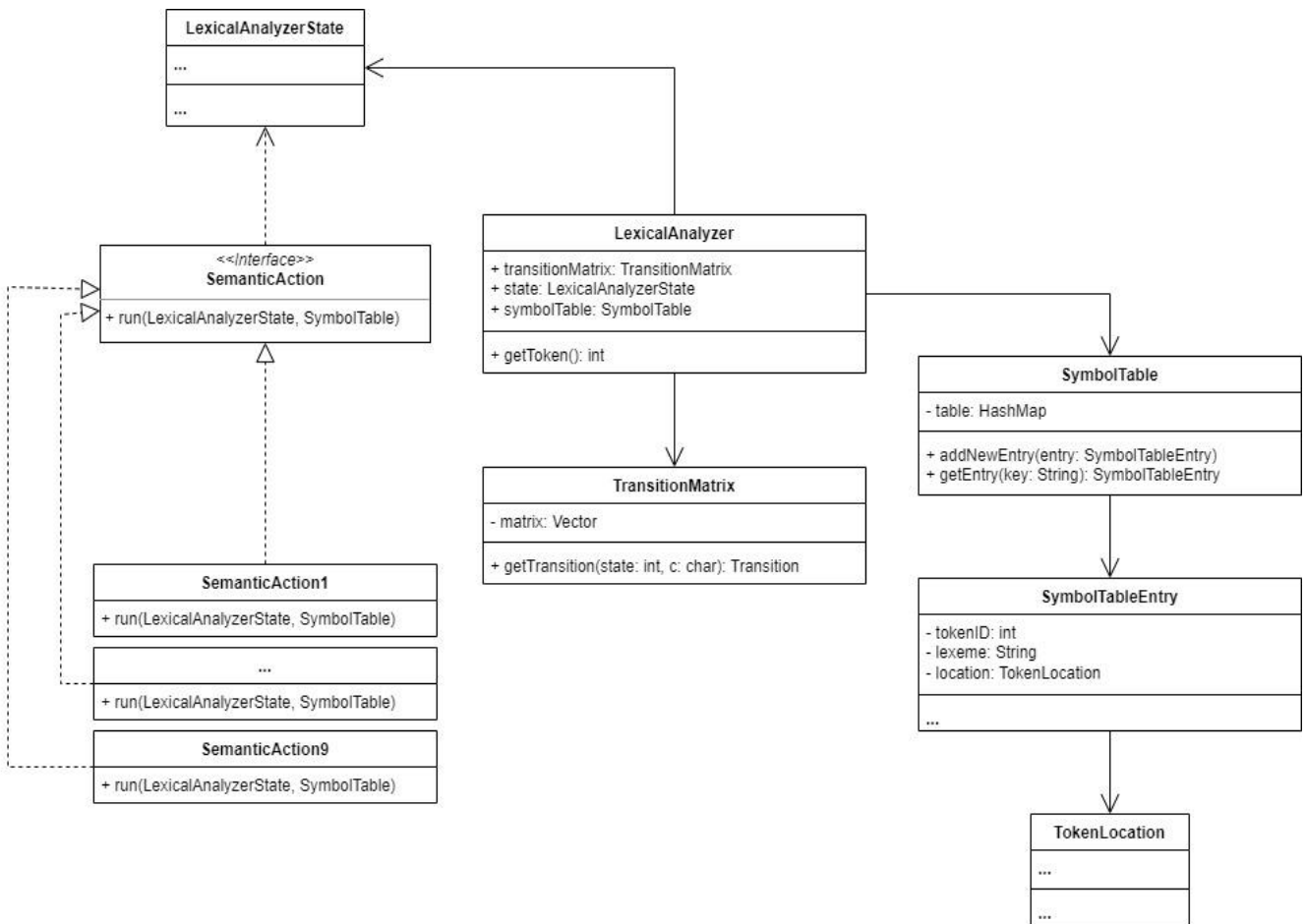
- Tokens detectados en el código fuente. Por ejemplo:  
Palabra reservada **IF**  
(  
Identificador **var\_x**  
+  
Constante entera **25**  
Palabra reservada **ELSE**  
etc.
- Errores léxicos detectados en el código fuente, indicando: nro. de línea y descripción del error. Por ejemplo:  
Línea 24: Constante entera fuera del rango permitido
- Contenidos de la Tabla de Símbolos.

Se sugiere la implementación de un consumidor de tokens que invoque al Analizador Léxico solicitándole tokens. En el trabajo práctico 2, esta funcionalidad estará a cargo del Analizador Sintáctico.

- b) El código fuente **debe ser leído desde un archivo**, cuyo nombre **debe poder ser elegido** por el usuario del compilador. Se espera que el compilador pueda ejecutarse desde línea de comandos.
- c) La numeración de las líneas de código debe comenzar en 1. De este modo, la información de cada error coincidirá con el número de línea en el archivo del código fuente.
- d) Para la programación se podrá elegir el lenguaje. Para esta elección, tener en cuenta que el analizador léxico se integrará luego a un Parser (Analizador Sintáctico) generado utilizando una herramienta tipo Yacc. Por lo tanto, es necesario asegurarse la disponibilidad de dicha herramienta para el lenguaje elegido.
- e) El Analizador Léxico deberá implementarse mediante una matriz de transición de estados y una matriz de acciones semánticas, de modo que cada cambio de estado y acción semántica asociada, sólo dependa del estado actual y el carácter leído.
- f) Implementar una Tabla de Símbolos donde se almacenarán identificadores, constantes, y cadenas de caracteres. Es requisito para la aprobación del trabajo, que la tabla sea implementada con una estructura dinámica.
- g) La aplicación deberá mostrar, además de tokens y errores léxicos, los contenidos de La Tabla de Símbolos.

Decidimos utilizar el lenguaje Java porque es el lenguaje que más conocemos en común y se complementa fácilmente con Yacc, teniendo mucha documentación a disposición.

## Estructura general del analizador léxico





---

## Implementación general - LexicalAnalyzer

En primera instancia creamos una clase “Compilador” que llama a otra clase “LexicalAnalyzer”, esta última tiene como atributos la matriz de transición, la tabla de símbolos, un arreglo con acciones semánticas y un estado actual, entre otros. Luego se inicia la clase (asignando sus respectivos valores cada atributo). Esta, tiene como método principal “getToken”, el mismo, mientras esté leyendo el token, hará una serie de funciones, entre ellas, pasar al siguiente estado, ejecutar acciones semánticas, etc.

Otra clase que es importante mencionar es “LexicalAnalyzerState”, esta es una instancia del analizador léxico que tiene como atributos el índice del archivo que se está leyendo, estado y lexema actual, una variable booleana que te indica si estas leyendo el token, el token a devolver, etc. En resumen, funciona como un registro que contiene los datos actuales del token que se está leyendo.

La tabla de símbolos es implementada con un HashMap, donde la clave es generada de manera aleatoria a partir de un UUID, y cada “SymbolTableEntry” tiene información de:

- El tokenID
- El lexema
- La ubicación

Por último, esta clave es utilizada para guardar en el yylval la entrada a la información del token reconocido.

## Diagrama de transición de estados

En primer lugar, debíamos realizar el diagrama de transición de estados. Utilizando [draw.io](https://draw.io) creamos todos los posibles estados y sus respectivas transiciones hasta llegar al nodo final (por una cuestión de mejor legibilidad, hay dos nodos finales).

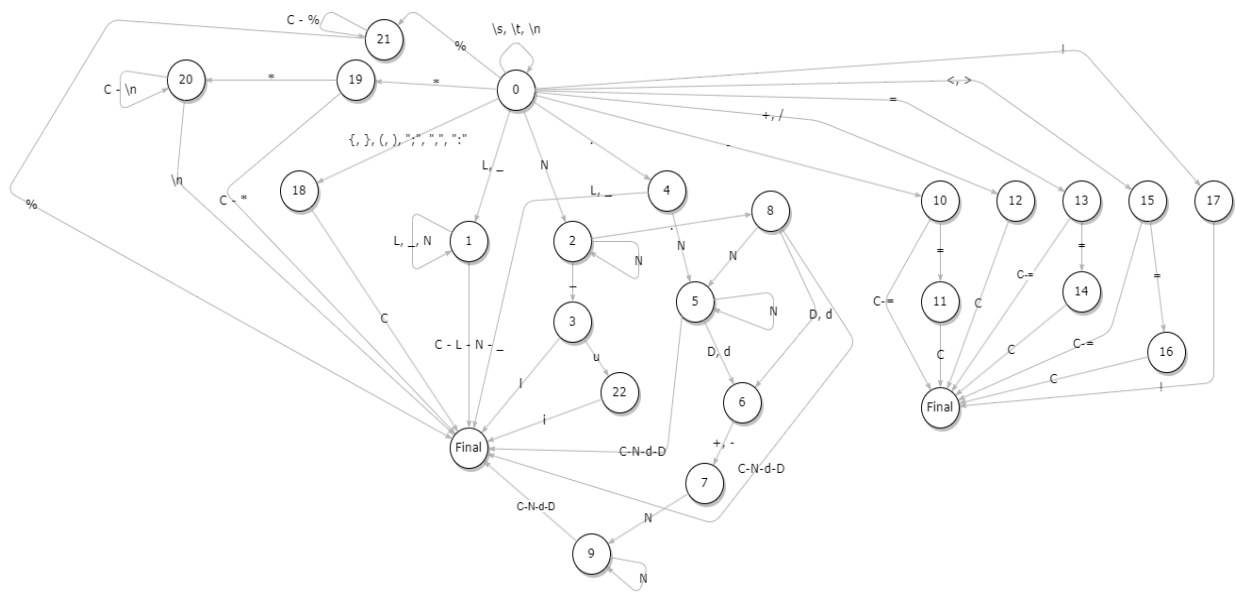


figura 2: "Diagrama de transición de estados"

Para implementar este diagrama en código, se podría haber utilizado una matriz de transición de estados como la que aparece en la teoría, pero tomamos la decisión de volver a utilizar un HashMap como aparece a continuación:

```
public class TransitionMatrix {  
  
    private static Vector<HashMap<Character, Transition>> matrix;
```

---

```
public class Transition {  
  
    private Integer newState;  
    private List<Integer> semanticActionList;  
  
}
```

*figura 3: "Atributos de la clase Transition"*

Por ejemplo:

si el analizador detecta el carácter ' \_ ' se pasará al estado 1 (como se marca en el diagrama de estados) ejecutándose la acción semántica 2:

```
HashMap<Character, Transition> s0 = new HashMap<>();  
s0.put(key: '_', new Transition(newState:1, List.of(e1:2)));
```

*figura 4: inicialización de HashMap y ejemplo de inserción de dupla*

Las acciones semánticas están cargadas en un arreglo en la clase "LexicalAnalyzer.java", entonces cuando se quiera ejecutar alguna acción, simplemente hay que acceder al arreglo en el índice que indica la transición y ejecutarla.

## Acciones semánticas

En total generamos 9 acciones semánticas, estas, implementan un método que implementan la interfaz SemanticAction (método run), que recibe como parámetro el estado actual del léxico (anteriormente mencionado) y la tabla de símbolos.

- **Acción semántica 1:** Se encarga de decrementar el índice de lectura del archivo.

- 
- **Acción semántica 2:** Se encarga de resetear el lexema actual sumándole el último carácter leído.
  - **Acción semántica 3:** A diferencia de la acción anterior, esta, simplemente le suma el último carácter leído al lexema actual. Pero si el carácter leído es un salto de línea, se omite.
  - **Acción semántica 4:** Si el lexema tiene más de 20 caracteres, es truncado avisando al usuario mediante un warning. Luego lo busca en la tabla de símbolos predefinidos, y si está, devuelve el token junto a su ubicación. En caso de que no se encuentre, se agrega a la tabla de símbolos dinámica (de ahora en adelante, simplemente tabla de símbolos) el token 'ID', su ubicación y el lexema. Finalmente se retorna el token y su referencia en la tabla de símbolos
  - **Acción semántica 5:** Verifica el rango de la constante y la da de alta en la tabla de símbolos, luego devuelve el token y su referencia en la tabla.
  - **Acción semántica 6:** Devuelve un token predefinido
  - **Acción semántica 7:** Resetea el lexema actual para comenzar con una nueva lectura.
  - **Acción semántica 8:** Agrega el string (presente en el lexema actual) a la tabla de símbolos y retorna el token 'CTE\_STRING' junto a su referencia en la tabla.

- 
- ***Acción semántica 9:*** Incrementa el índice de línea que se está leyendo del archivo.  
Para incluir la información de ubicación de los tokens en la tabla de símbolos.

## Errores léxicos considerados

- Enteros largos de 32 bits que no tengan como sufijo “\_l”
- Punto flotante:
  - no contener “.”
  - si se declara la parte exponencial, no contener “d” o “D”
- Comentarios:
  - no comenzar con “\*\*”
  - no finalizar con un salto de línea
- Las cadenas de caracteres que no empiezan y terminan con “%”
- Chequeo de rangos para las constantes.
- Identificadores cuyos nombres tengan más de 20 caracteres (Warning).

## Parte 2 - Analizador Sintáctico

Al igual que el analizador léxico, tuvimos en cuenta las especificaciones generales dadas por la cátedra para realizar la parte sintáctica:

### OBJETIVO

Construir un Parser (Analizador Sintáctico) que invoque al Analizador Léxico creado en el Trabajo Práctico N° 1, y que reconozca un lenguaje con las siguientes características:

### SINTAXIS GENERAL:

#### **Programa:**

- Programa constituido por un conjunto de sentencias, que pueden ser declarativas o ejecutables.  
**Las sentencias declarativas pueden aparecer en cualquier lugar del código fuente, exceptuando los bloques de las sentencias de control.**  
Los elementos declarados sólo serán visibles a partir de su declaración (esto será chequeado en etapas posteriores).
- El programa estará delimitado por llaves '{' y '}'.
- Cada sentencia debe terminar con coma ','.

#### **Sentencias declarativas:**

- Sentencias de declaración de datos para los tipos de datos correspondientes a cada grupo según la consigna del Trabajo Práctico 1, con la siguiente sintaxis:

<tipo> <lista\_de\_variables> ,

Donde <tipo> puede ser (Según tipos correspondientes a cada grupo): **SHORT, INT, LONG, USHORT, UINT, ULONG, FLOAT, DOUBLE**

Las variables de la lista se separan con punto y coma (',' )

- Incluir declaración de funciones **VOID**, con la siguiente sintaxis:

```
VOID ID (<parametro>)  
{  
  <cuerpo_de_la_funcion>  
}
```

Donde:

- <parametro> será un identificador precedido por un tipo.:

<tipo> ID

Se permite hasta un parámetro, y puede no haber parámetros. **Este chequeo debe efectuarse durante el Análisis Sintáctico**

- <cuerpo\_de\_la\_funcion> es un conjunto de sentencias declarativas (incluyendo declaración de otras funciones) y/o ejecutables, incluyendo sentencias de retorno con la siguiente estructura:

**RETURN,**

Ejemplos válidos:

```
VOID f1 (INT y)  
{  
  INT x,  
  x = y,  
  ...  
  RETURN,  
}
```

```
VOID f2 (LONG x)  
{  
  INT x,  
  x = y,  
  ...  
  RETURN,  
}
```

```
VOID f3 ()  
{  
  FLOAT x,  
  x = 1.2,  
  ...  
  IF (x > 0.0)  
    RETURN,  
  ELSE  
  {  
    x = 2.0,  
    RETURN,  
  }  
  END_IF,  
}
```

---

#### Sentencias ejecutables:

- Asignaciones donde el lado izquierdo puede ser un identificador, y el lado derecho una expresión aritmética. Los operandos de las expresiones aritméticas pueden ser variables, constantes, u otras expresiones aritméticas.

**No se deben permitir anidamientos de expresiones con paréntesis.**

- invocación a una función, con el siguiente formato:

**ID(<parametro\_real>),** // Función con parámetro

o

**ID ( ),** // Función sin parámetro

El parámetro real puede ser cualquier expresión aritmética, variable o constante.

Ejemplos:

`f1 (a) ,                      f2 ( ) ,                      f3 (a+b) ,                      f4 (5_i) ,`

- Cláusula de selección (IF). Cada rama de la selección será un bloque de sentencias. La estructura de la selección será, entonces:

**IF (<condicion>) <bloque\_de\_sent\_ejecutables> ELSE <bloque\_de\_sent\_ejecutables> END\_IF,**

El bloque para el **ELSE** puede estar ausente.

La condición será una comparación entre expresiones aritméticas, variables o constantes, y debe escribirse entre "( " )".

El bloque de sentencias ejecutables puede estar constituido por una sola sentencia, o un conjunto de sentencias ejecutables delimitadas por llaves.

- Sentencia de salida de mensajes por pantalla. El formato será

**PRINT<cadena>,**

Ejemplos:

`PRINT#Hola mundo#,                      //Tema 34`

`PRINT %Hola                              //Tema 35`

`Mundo%,`

## Implementación General - Yacc

Para implementar el analizador sintáctico, utilizamos Yacc para analizar estructuralmente una entrada, esta herramienta requiere 3 especificaciones:

- Una gramática (Conjunto de reglas)
- Código a ser invocado cuando una regla es reconocida
- Un analizador léxico que le provea los tokens

## Gramática

Tuvimos que realizar una gramática que reconozca las reglas, esta está contenida en un archivo llamado "compilador.y".

En primer lugar declaramos como tokens las palabras reservadas que actúan como estados finales (además de los tokens con relación 1:1 con ascii) y luego definimos la gramática.

```
%token IF ELSE END_IF PRINT CLASS VOID ID
      LONG UINT DOUBLE STRING
      CTE_LONG CTE_UINT CTE_DOUBLE CTE_STRING
      CMP_GE CMP_LE CMP_EQUAL CMP_NOT_EQUAL
      SUB_ASSIGN
      DO UNTIL IMPL FOR RETURN TOD
```

*"Figura 5: Definición de palabras reservadas"*

Para poder mostrar por pantalla el conjunto de reglas detectados en el código, utilizamos una cola de prioridad en la que se almacenan las diferentes estructuras sintácticas encontradas (SyntacticStructureResult) en base a su ubicación en el archivo.

Esta implementación nos permite imprimir de forma ordenada el conjunto de reglas sintácticas que reconoce el parser.

## Lista de no terminales

- **programa:** encierra todo el programa entre {}.
- **comparador:** operadores de comparación.
- **condicion:** comparación de expresiones.
- **tipo:** incluye palabras reservadas de los tipos de datos.
- **lista\_identificadores:** lista recursiva de identificadores.
- **sentencia\_declarativa:** incluye las sentencias de:
  - declaración de variable
  - definición de función y clases
  - implementación de métodos distribuidos.
- **sentencia\_ejecutable:**
  - asignación a variables/atributos
  - invocación a función/método
  - sentencia if, do until, PRINT y RETURN
- **lista\_sentencias:** lista de sentencias de cualquier tipo.
- **lista\_sentencias\_ejecutables:** sólo sentencias ejecutables.
- **invocacion\_funcion:** invocación a función con o sin parámetro.
- **op\_asignacion\_aumentada:** asignación con = o -=.
- **sentencia\_if:** sentencia condicional.
- **constante:** constante de cualquier tipo de dato.



- 
- **expr:** *expresión con casting.*
  - **basic\_expr:** *expresión sin anidamiento.*
  - **term:** *término de una expresión.*
  - **factor:** *factor de una expresión.*
  - **parametro\_formal:** *sintaxis de parámetro en declaración de funciones.*
  - **parametro\_real:** *sintaxis de parámetro en invocación a funciones.*
  - **definicion\_funcion:** *definición de función.*
  - **procedimiento:** *definición de procedimiento (función o método)*
  - **do\_until:** *sentencia de bucle condicional*
  - **metodo:** *definición de método.*
  - **acceso\_atributo:** *acceso a atributos mediante operador de acceso.*
  - **definicion\_clase:** *definición de clase.*
  - **cuerpo\_clase:** *cuerpo de clase según la especificación.*
  - **clase\_lista\_atributos:** *lista de atributos dentro de clase.*
  - **clase\_lista\_metodos:** *lista de métodos dentro de clase.*
  - **clase\_lista\_composicion:** *lista de clases para composición.*
  - **implementacion:** *implementación de métodos de forma distribuida.*

## Errores sintácticos considerados

- Sentencias que no terminen con coma ','

- 
- Programa que no esté delimitado por llaves '{' '}'
  - Sentencias declarativas:
    - Lista de variables declaradas que no estén separadas con un punto y coma ';'
    - Declaración de funciones:
      - Parámetro que no sea de la forma <tipo> ID
      - Cuerpo de la función que no esté delimitado con llaves '{' '}'
      - Funciones con mas de 1 parametro
  - Sentencias ejecutables:
    - Asignaciones donde el lado izquierdo no sea un identificador
    - Anidamiento de expresiones con paréntesis
    - Cláusula de selección **IF**:
      - Ramas de selección que no sean bloques de sentencias
      - Condición que no esté delimitada con paréntesis '(' ')'
      - Bloques de sentencias que no estén delimitadas por llaves '{' '}'
    - Símbolo de negativo en constantes tipo UINT

## Casos de prueba

---

Para manejar y mostrar por pantalla los errores detectados, decidimos guardar como información adicional en la tabla de símbolos la línea donde se encuentran cada token, por lo tanto cuando se encuentre un error de cualquier tipo, se puede identificar de qué línea surge y mostrarla.

Luego de las incorporaciones nuevas agregadas en las partes correspondientes a la generación de código intermedio y código assembler, a la hora de probar la salida del compilador, se van a mostrar datos y variables que no se explican en este informe (ya que posteriormente son explicadas). Se espera que el lector haga énfasis en la correspondiente impresión de la tabla de símbolos y las estructuras sintácticas encontradas.

## Casos de prueba - Léxico

- Case 1: para cada tipo de datos asignado, probamos si funciona el primer y último valor del rango. Esperamos que funcione bien y muestre por pantalla los siguiente para el siguiente código:

```
1  ** Constantes con el primer y ultimo valor dentro del rango (Para cada tipo de datos asignado).
2  ** Se espera que funcione BIEN
3  {
4      LONG var1,
5      UINT var2,
6      DOUBLE var3,
7
8      ** tipo long
9      var1 = -2147483648_l,
10
11     ** tipo long
12     var1 = 2147483647_l,
13
14     ** tipo uint
15     var2 = 0_ui,
16
17     ** tipo uint
18     var2 = 65535_ui,
19
20     ** tipo double
21     var3 = -1.7976931348623157d+308,
22
23     ** tipo double
24     var3 = 1.7976931348623157d+308,
25 }
```

```

Parsing correcto
Lista de tokens leídos:
{' ID '=' '-' CTE_DOUBLE ',' ID '=' CTE_DOUBLE ',' ID '=' CTE_UINT ',' ID '=' CTE_UINT ',' ID '=' '-' TOKEN NO CONOCIDO ',' ID '=' TOKEN NO CONOCIDO ',' '}'

Estructuras sintacticas encontradas:
Linea 5, estructura: Asignacion a variable
Linea 8, estructura: Asignacion a variable
Linea 11, estructura: Asignacion a variable
Linea 14, estructura: Asignacion a variable
Linea 17, estructura: Asignacion a variable
Linea 20, estructura: Asignacion a variable

Tabla de simbolos:
Clave: 82224a TokenID: 270 Ubicacion: Linea 17 Lexema: '-1.7976931348623157d+308'
Clave: eedd0c TokenID: 45 Ubicacion: Linea 17
Clave: cb7efc TokenID: 61 Ubicacion: Linea 14
Clave: 658f2e TokenID: 263 Ubicacion: Linea 20 Lexema: 'variable'
Clave: a46168 TokenID: 263 Ubicacion: Linea 14 Lexema: 'variable'
Clave: 76c000 TokenID: 125 Ubicacion: Linea 21
Clave: 252f1a TokenID: 269 Ubicacion: Linea 11 Lexema: '0_ui'
Clave: 6e115c TokenID: 270 Ubicacion: Linea 20 Lexema: '1.7976931348623157d+308'
Clave: 3abb7d TokenID: 61 Ubicacion: Linea 8
Clave: 37d20e TokenID: 269 Ubicacion: Linea 14 Lexema: '65535_ui'
Clave: 0585e5 TokenID: 61 Ubicacion: Linea 5
Clave: e27ba4 TokenID: 44 Ubicacion: Linea 17
Clave: dc6e16 TokenID: 61 Ubicacion: Linea 20
Clave: be2bf3 TokenID: 44 Ubicacion: Linea 20
Clave: b4d349 TokenID: 123 Ubicacion: Linea 3
Clave: 3b3eba TokenID: 268 Ubicacion: Linea 8 Lexema: '2147483647_1'
Clave: 4fd2ef TokenID: 44 Ubicacion: Linea 5
Clave: d93940 TokenID: 45 Ubicacion: Linea 5
Clave: bd07c4 TokenID: 268 Ubicacion: Linea 5 Lexema: '-2147483648_1'
Clave: c6ff61 TokenID: 263 Ubicacion: Linea 11 Lexema: 'variable'
Clave: d20581 TokenID: 44 Ubicacion: Linea 8
Clave: 0fd221 TokenID: 44 Ubicacion: Linea 11
Clave: d93dc5 TokenID: 263 Ubicacion: Linea 8 Lexema: 'variable'
Clave: 52f32c TokenID: 263 Ubicacion: Linea 5 Lexema: 'variable'
Clave: 388cfa TokenID: 44 Ubicacion: Linea 14
Clave: 771864 TokenID: 263 Ubicacion: Linea 17 Lexema: 'variable'
Clave: 8dfab8 TokenID: 61 Ubicacion: Linea 17
Clave: 7d5fe2 TokenID: 61 Ubicacion: Linea 11
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10>

```

- Case 2: Similar a la anterior, pero los valores asignados están fuera del rango, por lo tanto se espera que funcione mal, mostrando:

```

1  ** Constantes con el primer y ultimo valor fuera del rango (Para cada tipo de datos asignado).
2  ** Se espera que funcione MAL
3  {
4      LONG var1,
5      UINT var2,
6      DOUBLE var3,
7
8      ** tipo LONG
9      var1 = -2147483649_l,
10
11     ** tipo LONG
12     var1 = 2147483648_l,
13
14     ** tipo UINT
15     var2= -1_ui,
16
17     ** tipo UINT
18     var2 = 65536_ui,
19
20     ** tipo DOUBLE
21     var3 = -1.7976931348623159d+308,
22
23     ** tipo DOUBLE
24     var3 = 1.7976931348623159d+308,
25 }

```

PS C:\Users\Facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\lexico\case2.fn

```

[Lexico: Linea 5] Constante LONG fuera de rango: 2147483649_l
[Lexico: Linea 8] El rango de LONG es [-2147483648, 2147483647]
[Lexico: Linea 11] Las constantes tipo UINT no pueden ser negativas
[Lexico: Linea 14] Constante UINT fuera de rango: 65536_ui
[Lexico: Linea 17] Constante DOUBLE fuera de rango: 1.7976931348623159d+308
[Lexico: Linea 20] Constante DOUBLE fuera de rango: 1.7976931348623159d+308

```

Hubo errores en el parsing

Lista de tokens leídos:

```
{ ' ID '=' '-' CTE_DOUBLE ', ' ID '=' '-' CTE_DOUBLE ', ' ID '=' '-' CTE_UINT ', ' ID '=' '-' CTE_UINT ', ' ID '=' '-' TOKEN NO CONOCIDO ', ' ID '=' TOKEN NO CONOCIDO ', ' ' }
```

Estructuras sintacticas encontradas:

```

Linea 5, estructura: Asignacion a variable
Linea 8, estructura: Asignacion a variable
Linea 11, estructura: Asignacion a variable
Linea 14, estructura: Asignacion a variable
Linea 17, estructura: Asignacion a variable
Linea 20, estructura: Asignacion a variable

```

Tabla de símbolos:

|               |              |                     |                        |
|---------------|--------------|---------------------|------------------------|
| Clave: a36d17 | TokenID: 263 | Ubicacion: Linea 14 | Lexema: 'variable'     |
| Clave: 641af1 | TokenID: 44  | Ubicacion: Linea 14 |                        |
| Clave: ed2aad | TokenID: 270 | Ubicacion: Linea 20 |                        |
| Clave: 93b7d9 | TokenID: 61  | Ubicacion: Linea 11 |                        |
| Clave: 4dc4d1 | TokenID: 44  | Ubicacion: Linea 20 |                        |
| Clave: 610a61 | TokenID: 44  | Ubicacion: Linea 5  |                        |
| Clave: 9049c1 | TokenID: 125 | Ubicacion: Linea 21 |                        |
| Clave: 35e032 | TokenID: 44  | Ubicacion: Linea 17 |                        |
| Clave: 29f6f8 | TokenID: 44  | Ubicacion: Linea 8  |                        |
| Clave: cc895d | TokenID: 270 | Ubicacion: Linea 17 | Lexema: '-null'        |
| Clave: a86449 | TokenID: 123 | Ubicacion: Linea 3  |                        |
| Clave: 6cda8d | TokenID: 61  | Ubicacion: Linea 8  |                        |
| Clave: b0e633 | TokenID: 263 | Ubicacion: Linea 17 | Lexema: 'variable'     |
| Clave: a74a3d | TokenID: 44  | Ubicacion: Linea 11 |                        |
| Clave: a2560f | TokenID: 45  | Ubicacion: Linea 11 |                        |
| Clave: b40553 | TokenID: 263 | Ubicacion: Linea 5  | Lexema: 'variable'     |
| Clave: 131cc0 | TokenID: 263 | Ubicacion: Linea 11 | Lexema: 'variable'     |
| Clave: 1a8873 | TokenID: 269 | Ubicacion: Linea 11 | Lexema: '1_ui'         |
| Clave: 4408bf | TokenID: 61  | Ubicacion: Linea 5  |                        |
| Clave: e76a67 | TokenID: 263 | Ubicacion: Linea 20 | Lexema: 'variable'     |
| Clave: 91e3fc | TokenID: 61  | Ubicacion: Linea 20 |                        |
| Clave: b7dfdf | TokenID: 263 | Ubicacion: Linea 8  | Lexema: 'variable'     |
| Clave: 635d1e | TokenID: 268 | Ubicacion: Linea 8  | Lexema: '2147483648_l' |
| Clave: d937c9 | TokenID: 45  | Ubicacion: Linea 5  |                        |
| Clave: da71f0 | TokenID: 269 | Ubicacion: Linea 14 |                        |
| Clave: 605142 | TokenID: 268 | Ubicacion: Linea 5  | Lexema: '-null'        |
| Clave: ad9b8a | TokenID: 61  | Ubicacion: Linea 17 |                        |
| Clave: 145979 | TokenID: 61  | Ubicacion: Linea 14 |                        |
| Clave: 10d630 | TokenID: 45  | Ubicacion: Linea 17 |                        |

- Case 3: para los números de punto flotante, parte entera con y sin parte decimal, parte decimal con y sin parte entera, con y sin exponente y exponente positivo o negativo. Debe funcionar bien mostrando:

```
parsing correcto
```

```
Lista de tokens leídos:
```

```
{' ID '=' TOKEN NO CONOCIDO ',' ID '=' TOKEN NO CONOCIDO ',' ID '=' TOKEN NO CONOCIDO ',' ID '=' TOKEN NO CONOCIDO ',' ID '=' TO  
COCIDO ',' ID '=' TOKEN NO CONOCIDO ',' }'
```

```
Estructuras sintacticas encontradas:
```

```
Línea 4, estructura: Asignacion a variable  
Línea 5, estructura: Asignacion a variable  
Línea 6, estructura: Asignacion a variable  
Línea 7, estructura: Asignacion a variable  
Línea 8, estructura: Asignacion a variable  
Línea 9, estructura: Asignacion a variable  
Línea 10, estructura: Asignacion a variable
```

```
Tabla de símbolos:
```

|               |              |                     |                        |
|---------------|--------------|---------------------|------------------------|
| Clave: dc973f | TokenID: 270 | Ubicacion: Línea 10 | Lexema: '.45d+24'      |
| Clave: fa8aed | TokenID: 44  | Ubicacion: Línea 7  |                        |
| Clave: 9f139e | TokenID: 125 | Ubicacion: Línea 11 |                        |
| Clave: e271a1 | TokenID: 44  | Ubicacion: Línea 8  |                        |
| Clave: 620f9b | TokenID: 44  | Ubicacion: Línea 10 |                        |
| Clave: 07bcf3 | TokenID: 270 | Ubicacion: Línea 9  | Lexema: '.45d-24'      |
| Clave: 875969 | TokenID: 263 | Ubicacion: Línea 6  | Lexema: 'variable'     |
| Clave: abe2bd | TokenID: 270 | Ubicacion: Línea 7  | Lexema: '*2442.45d+24' |
| Clave: feeae1 | TokenID: 44  | Ubicacion: Línea 6  |                        |
| Clave: fc3f05 | TokenID: 270 | Ubicacion: Línea 4  | Lexema: '*2442.'       |
| Clave: b59139 | TokenID: 44  | Ubicacion: Línea 9  |                        |
| Clave: 2e30eb | TokenID: 44  | Ubicacion: Línea 5  |                        |
| Clave: bc73a7 | TokenID: 61  | Ubicacion: Línea 7  |                        |
| Clave: f58163 | TokenID: 270 | Ubicacion: Línea 6  | Lexema: '.634'         |
| Clave: a47890 | TokenID: 270 | Ubicacion: Línea 8  | Lexema: '*2442.45d-24' |
| Clave: a8200f | TokenID: 123 | Ubicacion: Línea 3  |                        |
| Clave: 5d4037 | TokenID: 61  | Ubicacion: Línea 8  |                        |
| Clave: a6dede | TokenID: 44  | Ubicacion: Línea 4  |                        |
| Clave: b16f78 | TokenID: 263 | Ubicacion: Línea 5  | Lexema: 'variable'     |
| Clave: 806a46 | TokenID: 61  | Ubicacion: Línea 6  |                        |
| Clave: 67727e | TokenID: 61  | Ubicacion: Línea 9  |                        |
| Clave: d9bb25 | TokenID: 61  | Ubicacion: Línea 10 |                        |
| Clave: 4dlb60 | TokenID: 263 | Ubicacion: Línea 8  | Lexema: 'variable'     |
| Clave: ec143d | TokenID: 263 | Ubicacion: Línea 4  | Lexema: 'variable'     |
| Clave: 17e3f3 | TokenID: 61  | Ubicacion: Línea 5  |                        |
| Clave: ed955a | TokenID: 263 | Ubicacion: Línea 10 | Lexema: 'variable'     |
| Clave: 5099f0 | TokenID: 263 | Ubicacion: Línea 7  | Lexema: 'variable'     |
| Clave: 941bb6 | TokenID: 61  | Ubicacion: Línea 4  |                        |
| Clave: 92e027 | TokenID: 263 | Ubicacion: Línea 9  | Lexema: 'variable'     |
| Clave: d92a91 | TokenID: 270 | Ubicacion: Línea 5  | Lexema: '*2442.523'    |

- 22

```

1  ** Identificadores de menos y mas de 25 caracteres.
2  ** Se espera que funcione BIEN (con warning)
3  {
4      LONG identificador_corto,
5      LONG identificador_muy_largo,
6
7      identificador_corto = 25_l,
8      identificador_muy_largo = 23_l,
9  }

```

```

[Linea 5] Lexema truncado
    identificador_muy_largo -> identificador_muy_la
Parsing correcto
Lista de tokens leidos:
{' ID '=' CTE_DOUBLE ', ' ID '=' CTE_DOUBLE ', ' '}'
Estructuras sintacticas encontradas:
Linea 4, estructura: Asignacion a variable
Linea 5, estructura: Asignacion a variable
Tabla de simbolos:
Clave: 66381c | TokenID: 125   Ubicacion: Linea 6
Clave: 283c5b | TokenID: 263   Ubicacion: Linea 5   Lexema: 'identificador_muy_la'
Clave: a20df9 | TokenID: 61    Ubicacion: Linea 4
Clave: 58f12f | TokenID: 123   Ubicacion: Linea 3
Clave: 445c18 | TokenID: 44    Ubicacion: Linea 4
Clave: 58ea5e | TokenID: 268   Ubicacion: Linea 5   Lexema: '23_l'
Clave: 582b5d | TokenID: 44    Ubicacion: Linea 5
Clave: f08131 | TokenID: 263   Ubicacion: Linea 4   Lexema: 'identificador_corto'
Clave: 53bbc5 | TokenID: 268   Ubicacion: Linea 4   Lexema: '25_l'
Clave: 0da5f9 | TokenID: 61    Ubicacion: Linea 5
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 5: identificadores con letras, dígitos y guiones(\_). Se espera que funcione bien:

```

1  ** Identificadores con letras, digitos y _
2  ** Se espera que funcione BIEN
3  {
4      STRING id_con_guiones_y_num,
5      UINT id_25,
6      id_con_guiones_y_num = %test%,
7      id_25 = 25_ui,
8  }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\lexico\case5.fn
Parsing correcto

Lista de tokens leídos:
{' ID '=' CTE_STRING ',' ID '=' CTE_UINT ',' '}'

Estructuras sintácticas encontradas:
Línea 4, estructura: Asignación a variable
Línea 6, estructura: Asignación a variable

Tabla de símbolos:
Clave: cf69c3 | TokenID: 44 | Ubicación: Línea 5
Clave: 621b01 | TokenID: 61 | Ubicación: Línea 6
Clave: 1e42c5 | TokenID: 269 | Ubicación: Línea 6 | Lexema: '25_ui'
Clave: c1a1b1 | TokenID: 123 | Ubicación: Línea 3
Clave: a18a9a | TokenID: 44 | Ubicación: Línea 6
Clave: c887b4 | TokenID: 61 | Ubicación: Línea 4
Clave: d161a5 | TokenID: 263 | Ubicación: Línea 4 | Lexema: 'id_con_guiones_y_num'
Clave: 550f80 | TokenID: 271 | Ubicación: Línea 4 | Lexema: 'test'
Clave: bbd4cf | TokenID: 125 | Ubicación: Línea 7
Clave: 286670 | TokenID: 263 | Ubicación: Línea 6 | Lexema: 'id_25'
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 6: Similar al anterior, pero uso Identificadores con caracteres inválidos (que no sean letras, dígitos o guiones). Debe funcionar mal y mostrar:

```

1  ** Intento de incluir en el nombre de u
2  ** Se espera que funcione MAL
3  {
4  |   LONG hol-a,
5  |   STRING variab*e,
6  |
7  |   hol-a = 25_l,
8  |   variab*e = %hola%,
9  }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\lexico\case6.fn
Hubo errores en el parsing

Lista de tokens leídos:
{' ID '-'

No se encontraron estructuras sintácticas

Tabla de símbolos:
Clave: 6aac73 | TokenID: 123 | Ubicación: Línea 3
Clave: 9f3018 | TokenID: 263 | Ubicación: Línea 4 | Lexema: 'hol'
Clave: 68d504 | TokenID: 45 | Ubicación: Línea 4
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 7: Utilizamos palabras reservadas escritas en minúsculas y mayúsculas. Se espera que funcione bien:



```

1  ** Palabras reservadas escritas en
2  ** Se espera que funcione BIEN
3  {
4      LONG do,
5      STRING string,
6      STRING return,
7
8      do = 25_1,
9      return = %hola%,
10     string = %string%,
11 }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\lexico\case7.fn
Parsing correcto
Lista de tokens leidos:
{' ID '=' CTE_DOUBLE ', ID '=' CTE_STRING ', ID '=' CTE_STRING ', ' '}'
Estructuras sintacticas encontradas:
Linea 4, estructura: Asignacion a variable
Linea 5, estructura: Asignacion a variable
Linea 7, estructura: Asignacion a variable
Tabla de simbolos:
Clave: cf57bc | TokenID: 125 | Ubicacion: Linea 10
Clave: cfa556 | TokenID: 268 | Ubicacion: Linea 4 | Lexema: '25_1'
Clave: a1aaf9 | TokenID: 44 | Ubicacion: Linea 6 | Lexema: 'string'
Clave: daaef3 | TokenID: 271 | Ubicacion: Linea 7 | Lexema: 'return'
Clave: a680ae | TokenID: 263 | Ubicacion: Linea 5 | Lexema: 'do'
Clave: 11178a | TokenID: 44 | Ubicacion: Linea 8 | Lexema: 'hola'
Clave: fbb473 | TokenID: 263 | Ubicacion: Linea 4 | Lexema: 'string'
Clave: c4c51c | TokenID: 123 | Ubicacion: Linea 3 | Lexema: 'do'
Clave: 71f830 | TokenID: 61 | Ubicacion: Linea 7 | Lexema: 'hola'
Clave: 7c622f | TokenID: 271 | Ubicacion: Linea 5 | Lexema: 'string'
Clave: c35331 | TokenID: 61 | Ubicacion: Linea 4 | Lexema: 'do'
Clave: 72ec30 | TokenID: 44 | Ubicacion: Linea 4 | Lexema: 'hola'
Clave: 5471dc | TokenID: 61 | Ubicacion: Linea 5 | Lexema: 'string'
Clave: d807e9 | TokenID: 263 | Ubicacion: Linea 7 | Lexema: 'string'
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 8: Comentarios bien y mal escritos, por lo tanto se espera que funcione mal:

```

1  ** Comentarios bien y mal escritos.
2  ** Se espera que funcione MAL
3  {
4      ** Comentario bien escrito de una linea
5
6      ** Comentario donde se
7      pretende que sea multilinea **
8  }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\lexico\case8.fn
Hubo errores en el parsing
Lista de tokens leídos:
{' ID ID ID
No se encontraron estructuras sintacticas
Tabla de simbolos:
Clave: 537dfe | TokenID: 263   Ubicacion: Linea 7   Lexema: 'pretende'
Clave: 12b1f4 | TokenID: 263   Ubicacion: Linea 7   Lexema: 'sea'
Clave: 54d87f | TokenID: 123   Ubicacion: Linea 3   Lexema: ''
Clave: db2d97 | TokenID: 263   Ubicacion: Linea 7   Lexema: 'que'
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 9: Cadenas bien y mal escritas, por lo tanto se espera que funcione mal:

```

1  ** Cadenas bien y mal escritas.
2  ** Se pretende que funcione MAL
3  {
4      STRING str_bien_escrita,
5      STRING str_mal_escrita,
6
7      str_bien_escrita = %esta es una cadena
8      bien escrita%,
9      str_mal_escrita = "cadena mal escrita",
10 }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\lexico\case9.fn
[Lexico: Linea 6] Caracter no reconocido: '%'
Hubo errores en el parsing
Lista de tokens leídos:
{' ID '=' CTE_STRING ', ID '=' ID ID
Estructuras sintacticas encontradas:
Linea 4, estructura: Asignacion a variable
Tabla de simbolos:
Clave: 189905 | TokenID: 61   Ubicacion: Linea 4
Clave: b06312 | TokenID: 123  Ubicacion: Linea 3
Clave: fd9317 | TokenID: 271  Ubicacion: Linea 4   Lexema: 'esta es una cadena
bien escrita'
Clave: cd96f6 | TokenID: 44   Ubicacion: Linea 5
Clave: b2557c | TokenID: 263  Ubicacion: Linea 4   Lexema: 'str_bien_escrita'
Clave: f84c14 | TokenID: 263  Ubicacion: Linea 6   Lexema: 'cadena'
Clave: 330b1f | TokenID: 263  Ubicacion: Linea 6   Lexema: 'mal'
Clave: ce272f | TokenID: 263  Ubicacion: Linea 6   Lexema: 'str_mal_escrita'
Clave: 23cbcc | TokenID: 61   Ubicacion: Linea 6
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```



```

1  ** Se espera que ande BIEN
2  {
3      CLASS persona
4      {
5          STRING nombre,
6
7          VOID set_nombre(STRING nuevo_nombre)
8          {
9              nombre = nuevo_nombre,
10             },
11         },
12
13         persona p1,
14         p1.set_nombre(%Facundo%),
15     }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\sintactico\case2.fn
Parsing correcto
Lista de tokens leidos:
{' CLASS ID '{' STRING ID ',' VOID ID '(' STRING ID ')' '{' ID '=' ID ',' '}' ',' '}' ',' ID ID ',' ID '.' ID '(' ')' '}' '}'
Estructuras sintacticas encontradas:
Linea 3, estructura: Definicion de clase
Linea 7, estructura: Implementacion de metodo dentro de clase
Linea 9, estructura: Asignacion a variable
Linea 13, estructura: Declaracion de variables tipo objeto
Linea 14, estructura: Invocacion a metodo
Tabla de simbolos:
Clave: fbf194 | TokenID: 41 | Ubicacion: Linea 14
Clave: 5639fa | TokenID: 263 | Ubicacion: Linea 7 | Lexema: 'set_nombre'
Clave: 3dac64 | TokenID: 263 | Ubicacion: Linea 14 | Lexema: 'set_nombre'
Clave: a7248c | TokenID: 44 | Ubicacion: Linea 10
Clave: f23d1c | TokenID: 44 | Ubicacion: Linea 11
Clave: 6f5db9 | TokenID: 123 | Ubicacion: Linea 2
Clave: cc5b28 | TokenID: 40 | Ubicacion: Linea 7
Clave: b4be66 | TokenID: 263 | Ubicacion: Linea 9 | Lexema: 'nuevo_nombre'
Clave: 272890 | TokenID: 263 | Ubicacion: Linea 9 | Lexema: 'nombre'
Clave: 980a3f | TokenID: 263 | Ubicacion: Linea 14 | Lexema: 'p1'
Clave: a63282 | TokenID: 267 | Ubicacion: Linea 5
Clave: 82d396 | TokenID: 263 | Ubicacion: Linea 13 | Lexema: 'p1'
Clave: fa800c | TokenID: 263 | Ubicacion: Linea 7 | Lexema: 'nuevo_nombre'
Clave: b3b926 | TokenID: 44 | Ubicacion: Linea 5
Clave: 87153c | TokenID: 263 | Ubicacion: Linea 3 | Lexema: 'persona'
Clave: e5db21 | TokenID: 61 | Ubicacion: Linea 9
Clave: 4ef250 | TokenID: 125 | Ubicacion: Linea 15
Clave: 7bbf87 | TokenID: 44 | Ubicacion: Linea 9
Clave: afdc5e | TokenID: 123 | Ubicacion: Linea 4
Clave: 42894f | TokenID: 44 | Ubicacion: Linea 13
Clave: 0bfdac | TokenID: 44 | Ubicacion: Linea 14
Clave: cc6a23 | TokenID: 263 | Ubicacion: Linea 5 | Lexema: 'nombre'
Clave: de96aa | TokenID: 40 | Ubicacion: Linea 14
Clave: 0b6636 | TokenID: 262 | Ubicacion: Linea 7
Clave: f1ff80 | TokenID: 125 | Ubicacion: Linea 11
Clave: 98df51 | TokenID: 263 | Ubicacion: Linea 13 | Lexema: 'persona'
Clave: 2ff636 | TokenID: 123 | Ubicacion: Linea 8
Clave: 3a4321 | TokenID: 267 | Ubicacion: Linea 7
Clave: 4a3bb6 | TokenID: 46 | Ubicacion: Linea 14
Clave: 138f6a | TokenID: 261 | Ubicacion: Linea 3
Clave: d10bf9 | TokenID: 41 | Ubicacion: Linea 7
Clave: e71d67 | TokenID: 125 | Ubicacion: Linea 10
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 3: Cláusula de selección IF con las dos ramas (else). Debe andar bien, mostrando:

```
1  ** Se espera que ande BIEN
2  {
3      LONG a; b,
4
5      a = 25_1,
6      b = 35_1,
7
8      IF (a < b)
9      {
10         PRINT %a es menor que b%,
11     }
12     ELSE
13     {
14         IF (a > b)
15         {
16             PRINT %a es mayor que b%,
17         }
18         ELSE
19         {
20             PRINT %a es igual a b%,
21         }
22     }
23     END_IF,
24 }
25 }
```

```

Parsing correcto

Lista de tokens leídos:
'{' LONG ID ',' ID ',' ID '=' CTE_DOUBLE ',' ID '=' CTE_UINT ',' IF '(' ID '<' ID ')' '{' PRINT CTE_STRING ',' '}' ELSE '{' IF '(' ID '>' ID ')' PRINT CTE_STRING ',' ELSE PRINT CTE_STRING ',' IF ',' '}'

Estructuras sintacticas encontradas:
Linea 3, estructura: Declaracion de variables primitivas
Linea 5, estructura: Asignacion a variable
Linea 6, estructura: Asignacion a variable
Linea 8, estructura: Sentencia IF
Linea 10, estructura: Sentencia PRINT
Linea 15, estructura: Sentencia IF
Linea 16, estructura: Sentencia PRINT
Linea 19, estructura: Sentencia PRINT

Tabla de simbolos:
Clave: d500b7 TokenID: 44 Ubicacion: Linea 3
Clave: 625f2d TokenID: 269 Ubicacion: Linea 6 Lexema: '35_ui'
Clave: 798bd1 TokenID: 271 Ubicacion: Linea 10 Lexema: 'a es menor que b'
Clave: b2e0e2 TokenID: 263 Ubicacion: Linea 8 Lexema: 'b'
Clave: 5527f2 TokenID: 44 Ubicacion: Linea 11
Clave: 7f1a6d TokenID: 271 Ubicacion: Linea 16 Lexema: 'a es mayor que b'
Clave: f18eab TokenID: 258 Ubicacion: Linea 18
Clave: 2e7c00 TokenID: 263 Ubicacion: Linea 15 Lexema: 'a'
Clave: e2fc77 TokenID: 125 Ubicacion: Linea 22
Clave: 5064b3 TokenID: 263 Ubicacion: Linea 6 Lexema: 'b'
Clave: b7d11c TokenID: 271 Ubicacion: Linea 19 Lexema: 'a es igual a b'
Clave: 9e2b0a TokenID: 59 Ubicacion: Linea 3
Clave: a8cb03 TokenID: 258 Ubicacion: Linea 13
Clave: 0e64d1 TokenID: 263 Ubicacion: Linea 15 Lexema: 'b'
Clave: babe8c TokenID: 268 Ubicacion: Linea 5 Lexema: '25_1'
Clave: 7b8920 TokenID: 125 Ubicacion: Linea 24
Clave: 7a762f TokenID: 123 Ubicacion: Linea 2
Clave: 133ded TokenID: 44 Ubicacion: Linea 21
Clave: 890f71 TokenID: 41 Ubicacion: Linea 8
Clave: 11ea6a TokenID: 260 Ubicacion: Linea 16
Clave: 750039 TokenID: 44 Ubicacion: Linea 6
Clave: ecd7c0 TokenID: 257 Ubicacion: Linea 15
Clave: a445e0 TokenID: 44 Ubicacion: Linea 23
Clave: 9c943e TokenID: 264 Ubicacion: Linea 3
Clave: 33ae57 TokenID: 40 Ubicacion: Linea 15
Clave: 73c3ae TokenID: 263 Ubicacion: Linea 3 Lexema: 'b'
Clave: 8877c9 TokenID: 260 Ubicacion: Linea 19
Clave: a0eeab TokenID: 123 Ubicacion: Linea 14
Clave: 95285b TokenID: 62 Ubicacion: Linea 15
Clave: 44ed49 TokenID: 259 Ubicacion: Linea 23
Clave: b56f5d TokenID: 41 Ubicacion: Linea 15
Clave: c45bf1 TokenID: 263 Ubicacion: Linea 8 Lexema: 'a'
Clave: 6b56fe TokenID: 123 Ubicacion: Linea 9
Clave: 7be931 TokenID: 61 Ubicacion: Linea 5
Clave: e4882f TokenID: 125 Ubicacion: Linea 12
Clave: 1a0712 TokenID: 44 Ubicacion: Linea 20
Clave: 8a0098 TokenID: 44 Ubicacion: Linea 5
Clave: 6dd6cc TokenID: 44 Ubicacion: Linea 17
Clave: b8607e TokenID: 260 Ubicacion: Linea 10
Clave: 8cccbb TokenID: 263 Ubicacion: Linea 3 Lexema: 'a'
Clave: a4aeb6 TokenID: 61 Ubicacion: Linea 6
Clave: 044578 TokenID: 257 Ubicacion: Linea 8
Clave: fe8a72 TokenID: 40 Ubicacion: Linea 8
Clave: 21c096 TokenID: 60 Ubicacion: Linea 8
Clave: 7e1a54 TokenID: 263 Ubicacion: Linea 5 Lexema: 'a'
Clave: 5bc542 TokenID: 259 Ubicacion: Linea 21
PS C:\Users\Facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 4: probamos la cláusula DO UNTIL. Debería funcionar bien:

```

1  ** Se espera que ande BIEN
2  ✓ {
3      LONG acumulador; c,
4      acumulador = 0_1,
5      c = 10_1,
6
7      DO
8      ✓ {
9          acumulador -= c,
10         c -= 1_1,
11     } UNTIL (c < 0_1),
12 }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\sintactico\case4.fn
Parsing correcto
Lista de tokens leídos:
{' ' LONG ID ';' ID ' ' ID '=' CTE_DOUBLE ' ' ID '=' CTE_DOUBLE ' ' DO '{' ID '-' ID ' ' ID '-' CTE_DOUBLE ' ' ' '}' UNTIL '(' ID '<' CTE_DOUBLE ')' ' ' ' '}'
Estructuras sintacticas encontradas:
Linea 3, estructura: Declaracion de variables primitivas
Linea 4, estructura: Asignacion a variable
Linea 5, estructura: Asignacion a variable
Linea 7, estructura: Estructura DO UNTIL
Linea 9, estructura: Asignacion a variable
Linea 10, estructura: Asignacion a variable
Tabla de simbolos:
Clave: fbdce0 TokenID: 123 Ubicacion: Linea 8
Clave: f552e0 TokenID: 44 Ubicacion: Linea 10
Clave: 34c3f0 TokenID: 277 Ubicacion: Linea 7
Clave: 23664c TokenID: 263 Ubicacion: Linea 10 Lexema: 'c'
Clave: 93b333 TokenID: 61 Ubicacion: Linea 4 Lexema: '0_1'
Clave: d8e931 TokenID: 268 Ubicacion: Linea 11 Lexema: '0_1'
Clave: 956288 TokenID: 278 Ubicacion: Linea 11 Lexema: 'acumulador'
Clave: bcd165 TokenID: 263 Ubicacion: Linea 3 Lexema: '0_1'
Clave: 827636 TokenID: 268 Ubicacion: Linea 4 Lexema: 'c'
Clave: 8a7a13 TokenID: 40 Ubicacion: Linea 11 Lexema: 'c'
Clave: c2ad81 TokenID: 263 Ubicacion: Linea 3 Lexema: 'c'
Clave: 850901 TokenID: 263 Ubicacion: Linea 9 Lexema: 'c'
Clave: de3e45 TokenID: 125 Ubicacion: Linea 11 Lexema: 'c'
Clave: 875c12 TokenID: 264 Ubicacion: Linea 3 Lexema: 'c'
Clave: 29cab7 TokenID: 125 Ubicacion: Linea 12 Lexema: 'c'
Clave: aeeb12 TokenID: 276 Ubicacion: Linea 9 Lexema: 'c'
Clave: 35cf45 TokenID: 44 Ubicacion: Linea 9 Lexema: 'c'
Clave: 7ff2e8 TokenID: 59 Ubicacion: Linea 3 Lexema: 'c'
Clave: ea5716 TokenID: 44 Ubicacion: Linea 4 Lexema: 'c'
Clave: 723991 TokenID: 263 Ubicacion: Linea 11 Lexema: 'c'
Clave: 47d9bd TokenID: 123 Ubicacion: Linea 2 Lexema: 'c'
Clave: 0f4322 TokenID: 41 Ubicacion: Linea 11 Lexema: 'c'
Clave: d96bf3 TokenID: 44 Ubicacion: Linea 5 Lexema: '1_1'
Clave: c86583 TokenID: 276 Ubicacion: Linea 10 Lexema: '1_1'
Clave: 8c6bb9 TokenID: 268 Ubicacion: Linea 10 Lexema: 'acumulador'
Clave: a1ca29 TokenID: 44 Ubicacion: Linea 11 Lexema: 'acumulador'
Clave: 8d44c4 TokenID: 263 Ubicacion: Linea 9 Lexema: 'acumulador'
Clave: 14b1e2 TokenID: 263 Ubicacion: Linea 4 Lexema: 'c'
Clave: 8193d7 TokenID: 44 Ubicacion: Linea 3 Lexema: 'c'
Clave: bd4e84 TokenID: 60 Ubicacion: Linea 11 Lexema: 'c'
Clave: 3d1c61 TokenID: 263 Ubicacion: Linea 5 Lexema: 'c'
Clave: e43730 TokenID: 61 Ubicacion: Linea 5 Lexema: '10_1'
Clave: 699c5c TokenID: 268 Ubicacion: Linea 5 Lexema: '10_1'
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 5: Herencia por composición e implementación de método a la clase hija. Se espera que funcione bien:

```
1  ** Se espera que ande BIEN
2  {
3      CLASS persona
4      {
5          STRING nombre,
6          LONG dni,
7      },
8
9      CLASS empleado
10     {
11         LONG id_empleado,
12
13         persona p1, ** Herencia por composicion
14     },
15
16     IMPL FOR empleado:
17     {
18         VOID enviar_encomienda()
19         {
20             PRINT %se envia encomienda%,
21         },
22     },
23 }
```



```

PS C:\Users\Facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\sintactico\case5.fn
Parsing correcto
Lista de tokens leídos:
{' CLASS ID {' STRING ID ',' LONG ID ',' '}' ',' CLASS ID {' LONG ID ',' ID ',' '}' ',' IMPL FOR ID ':' {' VOID ID '(' ' ' '}' PRINT CTE_STRING ',' ' ' '}' ',' '}'
Estructuras sintácticas encontradas:
Línea 3, estructura: Definición de clase
Línea 9, estructura: Definición de clase
Línea 16, estructura: Implementación de metodo fuera de clase
Línea 16, estructura: Implementación de metodo
Línea 18, estructura: Implementación de metodo dentro de clase
Línea 20, estructura: Sentencia PRINT
Tabla de símbolos:
Clave: 52be83 TokenID: 123 Ubicacion: Línea 19
Clave: f09733 TokenID: 44 Ubicacion: Línea 22
Clave: 94cf9d TokenID: 123 Ubicacion: Línea 17
Clave: f3e9bd TokenID: 41 Ubicacion: Línea 18
Clave: e1c6b6 TokenID: 44 Ubicacion: Línea 11
Clave: ea69cf TokenID: 125 Ubicacion: Línea 7
Clave: 581acf TokenID: 264 Ubicacion: Línea 11
Clave: b40c30 TokenID: 263 Ubicacion: Línea 9 Lexema: 'empleado'
Clave: 67792f TokenID: 125 Ubicacion: Línea 24
Clave: 8a9b12 TokenID: 263 Ubicacion: Línea 13 Lexema: 'persona'
Clave: d51642 TokenID: 280 Ubicacion: Línea 16
Clave: 0da079 TokenID: 44 Ubicacion: Línea 5
Clave: c3a1ec TokenID: 44 Ubicacion: Línea 7
Clave: b9f82a TokenID: 44 Ubicacion: Línea 14
Clave: 76e07b TokenID: 123 Ubicacion: Línea 4
Clave: 00a9c8 TokenID: 263 Ubicacion: Línea 6 Lexema: 'dni'
Clave: e0e412 TokenID: 44 Ubicacion: Línea 23
Clave: 1710ca TokenID: 261 Ubicacion: Línea 3
Clave: 10d80e TokenID: 260 Ubicacion: Línea 20
Clave: 0741d5 TokenID: 279 Ubicacion: Línea 16
Clave: b9cb51 TokenID: 125 Ubicacion: Línea 23
Clave: cc9551 TokenID: 123 Ubicacion: Línea 10
Clave: c556bd TokenID: 261 Ubicacion: Línea 9
Clave: 571458 TokenID: 125 Ubicacion: Línea 22
Clave: 5a8420 TokenID: 44 Ubicacion: Línea 6
Clave: 72bdd5 TokenID: 271 Ubicacion: Línea 20 Lexema: 'se envia encomienda'
Clave: dd2d37 TokenID: 263 Ubicacion: Línea 16 Lexema: 'empleado'
Clave: 1e3481 TokenID: 44 Ubicacion: Línea 21
Clave: fa90bd TokenID: 267 Ubicacion: Línea 5
Clave: 7003eb TokenID: 263 Ubicacion: Línea 5 Lexema: 'nombre'
Clave: d450d0 TokenID: 264 Ubicacion: Línea 6
Clave: 729459 TokenID: 125 Ubicacion: Línea 14
Clave: 5769fd TokenID: 263 Ubicacion: Línea 11 Lexema: 'id_empleado'
Clave: 5ceca5 TokenID: 58 Ubicacion: Línea 16
Clave: 1bd9ca TokenID: 40 Ubicacion: Línea 18
Clave: df8913 TokenID: 263 Ubicacion: Línea 3 Lexema: 'persona'
Clave: d65992 TokenID: 44 Ubicacion: Línea 13
Clave: ec6c1b TokenID: 263 Ubicacion: Línea 18 Lexema: 'enviar_encomienda'
Clave: 7ce900 TokenID: 262 Ubicacion: Línea 18
Clave: 35fa21 TokenID: 123 Ubicacion: Línea 2
PS C:\Users\Facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 6: Función dentro de otra función, se espera que funcione bien:

```

1  ** Se espera que funcione BIEN
2  {
3      VOID exterior()
4      {
5          VOID interior(LONG p)
6          {
7              PRINT %hola%,
8          },
9          interior(5_1),
10     },
11     exterior(),
12 }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\sintactico\case6.fn
Parsing correcto
Lista de tokens leídos:
{' VOID ID '(' ')' '{' VOID ID '(' LONG ID ')' '{' PRINT CTE_STRING ',' '}' ' ID '(' CTE_DOUBLE ')' ' ',' '}' ' ID '(' ')' ' ',' '}'
Estructuras sintacticas encontradas:
Linea 3, estructura: Definicion de funcion
Linea 5, estructura: Definicion de funcion
Linea 7, estructura: Sentencia PRINT
Linea 10, estructura: Invocacion a funcion
Linea 12, estructura: Invocacion a funcion
Tabla de simbolos:
Clave: 63d81b TokenID: 44 Ubicacion: Linea 12
Clave: 6f7baa TokenID: 264 Ubicacion: Linea 5
Clave: e820b2 TokenID: 125 Ubicacion: Linea 9
Clave: db8bfa TokenID: 268 Ubicacion: Linea 10 Lexema: '5_1'
Clave: 08392a TokenID: 260 Ubicacion: Linea 7
Clave: 209ab7 TokenID: 125 Ubicacion: Linea 13
Clave: 9ade86 TokenID: 41 Ubicacion: Linea 10
Clave: 9da863 TokenID: 125 Ubicacion: Linea 11
Clave: 4b83d0 TokenID: 271 Ubicacion: Linea 7 Lexema: 'hola'
Clave: c15fc3 TokenID: 40 Ubicacion: Linea 12
Clave: 5040ef TokenID: 41 Ubicacion: Linea 3
Clave: 0c1a19 TokenID: 44 Ubicacion: Linea 11
Clave: 37dcfc TokenID: 123 Ubicacion: Linea 6
Clave: 5e7492 TokenID: 44 Ubicacion: Linea 10
Clave: 53aa7f TokenID: 41 Ubicacion: Linea 12
Clave: bdf540 TokenID: 262 Ubicacion: Linea 5
Clave: 81cf62 TokenID: 263 Ubicacion: Linea 5 Lexema: 'interior'
Clave: d116b5 TokenID: 263 Ubicacion: Linea 3 Lexema: 'exterior'
Clave: fb1670 TokenID: 123 Ubicacion: Linea 2
Clave: 5fb859 TokenID: 40 Ubicacion: Linea 10
Clave: cbd546 TokenID: 123 Ubicacion: Linea 4
Clave: 075e15 TokenID: 40 Ubicacion: Linea 3
Clave: 5b43f6 TokenID: 44 Ubicacion: Linea 9
Clave: 2a6327 TokenID: 263 Ubicacion: Linea 10 Lexema: 'interior'
Clave: 7e05ce TokenID: 44 Ubicacion: Linea 8
Clave: d0e505 TokenID: 40 Ubicacion: Linea 5
Clave: 335e8c TokenID: 263 Ubicacion: Linea 5 Lexema: 'p'
Clave: 368601 TokenID: 262 Ubicacion: Linea 3
Clave: 8355c6 TokenID: 41 Ubicacion: Linea 5
Clave: 37761d TokenID: 263 Ubicacion: Linea 12 Lexema: 'exterior'
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 7: IF con una condición inválida ya que el número 53 no es una constante válida por el lenguaje, por lo tanto se espera que funcione mal mostrando:

```

1  ** Se espera que no funcione
2  ** (53 no es una constante valida)
3  {
4      LONG a,
5      a = 100_1,
6
7      IF (a < 53)
8      {
9          PRINT %error%,
10     }
11     ELSE
12     {
13         PRINT %error%,
14     }
15     END_IF,
16 }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\sintactico\case7.fn
[Lexico: Linea 7] Caracter no reconocido: ')'
[Sintactico: Linea 7] Error en IF

Hubo errores en el parsing

Lista de tokens leidos:
{' LONG ID ', ' ID '=' CTE_DOUBLE ', ' IF '(' ID '<' '{' PRINT CTE_STRING ', ' '}' ELSE '{' PRINT CTE_STRING ', ' '}' END_IF ', ' '}'

Estructuras sintacticas encontradas:
Linea 4, estructura: Declaracion de variables primitivas
Linea 5, estructura: Asignacion a variable

Tabla de simbolos:
Clave: c40a66 TokenID: 268 Ubicacion: Linea 5 Lexema: '100_1'
Clave: b070b8 TokenID: 44 Ubicacion: Linea 4
Clave: 115a23 TokenID: 258 Ubicacion: Linea 12
Clave: 621b6b TokenID: 263 Ubicacion: Linea 7 Lexema: 'a'
Clave: 63c5a7 TokenID: 271 Ubicacion: Linea 9 Lexema: 'error'
Clave: 26989b TokenID: 123 Ubicacion: Linea 13
Clave: 4da826 TokenID: 263 Ubicacion: Linea 4 Lexema: 'a'
Clave: edb6b4 TokenID: 125 Ubicacion: Linea 18
Clave: 119885 TokenID: 44 Ubicacion: Linea 17
Clave: 33a820 TokenID: 44 Ubicacion: Linea 5
Clave: a4ad4f TokenID: 260 Ubicacion: Linea 14
Clave: 1ab1ae TokenID: 40 Ubicacion: Linea 7
Clave: c8ed40 TokenID: 125 Ubicacion: Linea 11
Clave: c03a1a TokenID: 257 Ubicacion: Linea 7
Clave: 0d250b TokenID: 264 Ubicacion: Linea 4
Clave: 4103c0 TokenID: 259 Ubicacion: Linea 17
Clave: 4c9cbc TokenID: 60 Ubicacion: Linea 7
Clave: 6b7f71 TokenID: 44 Ubicacion: Linea 10
Clave: 5ae509 TokenID: 123 Ubicacion: Linea 8
Clave: 5425c1 TokenID: 123 Ubicacion: Linea 3
Clave: 443ca8 TokenID: 44 Ubicacion: Linea 15
Clave: 48a2b4 TokenID: 260 Ubicacion: Linea 9
Clave: 044499 TokenID: 125 Ubicacion: Linea 16
Clave: 807995 TokenID: 61 Ubicacion: Linea 5
Clave: b7a8bf TokenID: 263 Ubicacion: Linea 5 Lexema: 'a'
Clave: d6c22e TokenID: 271 Ubicacion: Linea 14 Lexema: 'error'
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

- Case 8: Pasamos dos datos como parametro a un método, pero esto no es válido, ya sea para un método o una función, por lo tanto debería andar mal y mostrar:

```

1  ** Se espera que no funcione
2  ** No se pueden pasar dos parametros a funciones/metodos
3  {
4      CLASS persona
5      {
6          STRING nombre,
7          LONG edad,
8
9          VOID set_datos(STRING nuevo_nombre, LONG nueva_edad)
10         {
11             nombre = nuevo_nombre,
12             edad = nueva_edad,
13         },
14     },
15
16     persona p1,
17     p1.set_nombre(),
18 }

```

```

PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10> java -jar .\build\compilador.jar .\tests\sintactico\case8.fn
[Syntactico: Linea 9] Error en funcion/metodo

Hubo errores en el parsing

Lista de tokens leidos:
{' CLASS ID '{' STRING ID ',' LONG ID ')' VOID ID '(' STRING ID ',' LONG ID ')' '{' ID '=' ID ',' ID '=' ID ',' '}' ',' '}' ',' ID ID ',' ID '.' ID '(' ')' ',' '}'

Estructuras sintacticas encontradas:
Linea 4, estructura: Definicion de clase
Linea 9, estructura: Implementacion de metodo dentro de clase
Linea 16, estructura: Declaracion de variables tipo objeto
Linea 17, estructura: Invocacion a metodo

Tabla de simbolos:
Clave: f98880 TokenID: 44 Ubicacion: Linea 14
Clave: 5ef63b TokenID: 40 Ubicacion: Linea 17
Clave: 2e405a TokenID: 123 Ubicacion: Linea 3
Clave: b6393a TokenID: 46 Ubicacion: Linea 17
Clave: 290ad6 TokenID: 44 Ubicacion: Linea 11
Clave: ad8ed9 TokenID: 44 Ubicacion: Linea 12
Clave: 75faad TokenID: 44 Ubicacion: Linea 7
Clave: a330b9 TokenID: 263 Ubicacion: Linea 6 Lexema: 'nombre'
Clave: a81971 TokenID: 263 Ubicacion: Linea 9 Lexema: 'set_datos'
Clave: 99db43 TokenID: 264 Ubicacion: Linea 9
Clave: d979a7 TokenID: 267 Ubicacion: Linea 6
Clave: 34a628 TokenID: 61 Ubicacion: Linea 12
Clave: 9bdc5b TokenID: 263 Ubicacion: Linea 12 Lexema: 'edad'
Clave: 706e67 TokenID: 263 Ubicacion: Linea 9 Lexema: 'nuevo_nombre'
Clave: e51260 TokenID: 44 Ubicacion: Linea 16
Clave: b006f7 TokenID: 41 Ubicacion: Linea 17
Clave: 686846 TokenID: 44 Ubicacion: Linea 9
Clave: 5fa955 TokenID: 263 Ubicacion: Linea 17 Lexema: 'set_nombre'
Clave: 6956f7 TokenID: 261 Ubicacion: Linea 4
Clave: e500ec TokenID: 125 Ubicacion: Linea 18
Clave: 796731 TokenID: 263 Ubicacion: Linea 12 Lexema: 'nueva_edad'
Clave: 1f3c38 TokenID: 123 Ubicacion: Linea 5
Clave: d3a162 TokenID: 264 Ubicacion: Linea 7
Clave: 345ea6 TokenID: 123 Ubicacion: Linea 10
Clave: aab4ae TokenID: 262 Ubicacion: Linea 9
Clave: 52cd43 TokenID: 263 Ubicacion: Linea 17 Lexema: 'p1'
Clave: bb4277 TokenID: 44 Ubicacion: Linea 6
Clave: b1b1b6 TokenID: 44 Ubicacion: Linea 17
Clave: 4082ec TokenID: 263 Ubicacion: Linea 11 Lexema: 'nombre'
Clave: 6f0fb1 TokenID: 41 Ubicacion: Linea 9
Clave: 4e2674 TokenID: 125 Ubicacion: Linea 14
Clave: b2b8a4 TokenID: 267 Ubicacion: Linea 9
Clave: 719ba7 TokenID: 263 Ubicacion: Linea 11 Lexema: 'nuevo_nombre'
Clave: c240b2 TokenID: 44 Ubicacion: Linea 13
Clave: e7e0ed TokenID: 263 Ubicacion: Linea 16 Lexema: 'persona'
Clave: 8d77b4 TokenID: 125 Ubicacion: Linea 13
Clave: 46c137 TokenID: 263 Ubicacion: Linea 16 Lexema: 'p1'
Clave: c76d05 TokenID: 263 Ubicacion: Linea 7 Lexema: 'edad'
Clave: f71ac8 TokenID: 263 Ubicacion: Linea 4 Lexema: 'persona'
Clave: 0d7cfe TokenID: 40 Ubicacion: Linea 9
Clave: d3bf07 TokenID: 263 Ubicacion: Linea 9 Lexema: 'nueva_edad'
Clave: 7b2650 TokenID: 61 Ubicacion: Linea 11
PS C:\Users\facun\OneDrive\Escritorio\entrega grupo 10\entrega grupo 10>

```

---

## Correcciones realizadas - 2da entrega

1. “El ‘compilador.jar’ no funciona en cualquier caso que se pruebe, ya sea con un código que tiene o no errores”  
Ejecutable arreglado.
2. “No hay indicación en el informe de cómo ejecutar el compilador ‘jar’, como lo generaron en cuanto a que versión de Java usar, etc”  
Indicación puesta en la introducción (Pag 3)
3. “No se explica en el informe, ni se dan ejemplos de cuales son los resultados que se reportan al ejecutar el compilador”  
Casos de prueba (pág 20)
4. “No se informa como se reporta la línea donde se encuentra algún error y como se informa tal error. Tampoco se habla de la política de manejo de errores utilizada”  
No se explicó en el informe el manejo de errores, la salida por pantalla muestra los errores y en la línea donde se encuentra.
5. “No se da un ejemplo de como se listan las reglas, tabla de símbolos, etc por pantalla.”  
Corregido
6. “Deben listar en la tabla de símbolos cuál es la entrada y que token representan más allá de TokenID. Validen nuevamente el diagrama de transición de estados para cubrir ese, y otros caso que detecten para arreglar.”  
Corregido, y el diagrama de transición fue validado
7. “ El compilador no detecta palabras reservadas respecto a su identificación de mayúsculas y minúsculas; e.g. int VARIABLE,”  
No pudimos corregirlo
8. “No encuentro casos de testeo de todos los puntos asignados al grupo; e.g. Herencia por Composición.”  
Caso de pruebas que faltaban añadidos
9. “Sería conveniente que trabajen con los archivos de testeo creados como “.txt” y no como “.fn”, principalmente por comodidad al crearlos y editarlos.”  
Corregido

---

## Conclusión

Al principio de la cursada no sabíamos cómo se podría traducir un código fuente a un código ejecutable, pero a medida que avanzábamos en este trabajo empezamos a comprender cómo se implementa un compilador, profundizando en su estructura y sus partes, tuvimos que informarnos sobre ciertos temas, tomar decisiones de implementación y aprender a usar herramientas nuevas. Si bien nos quedan dos etapas más que debemos aprender y profundizar, ya hemos podido realizar las dos primeras partes del compilador, hemos aplicado técnicas aprendidas en la carrera, manejamos los errores de manera efectiva y documentamos el proceso.

Para finalizar, la experiencia de crear un software complejo como este, nos puso a prueba y hemos podido mejorar nuestras habilidades y capacidades adquiriendo conocimientos nuevos.