



DISEÑO DE COMPILADORES I

Grupo 7

Integrantes:

Ceschini Tomás
Fratti Joaquin Ignacio
Pozzi Matias Nicolas

tomiceschini99@gmail.com
jffati@alumnos.exa.unicen.edu.ar
pozzi.matias.nicolas@gmail.com

Temas particulares asignados

Analizador Léxico

4. Enteros (16 bits): Constantes enteras con valores entre -2^{15} y $2^{15} - 1$. Estas constantes llevarán el sufijo “_i”.

Enteros largos sin signo (32 bits): Constantes enteras con valores entre 0 y $2^{32} - 1$. Estas constantes llevarán el sufijo “_ul”.

Se deben incorporar a la lista de palabras reservadas las palabras INT y ULONG

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-

Considerar el rango $2.2250738585072014D-308 < x < 1.7976931348623157D+308$
 $-1.7976931348623157D+308 < x < -2.2250738585072014D-308$ 0.0

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

12. Incorporar a la lista de operadores, el operador -=.

14. Incorporar a la lista de palabras reservadas, las palabras DO y UNTIL.

18. A definir en Trabajos Prácticos 2/3.

20. Incorporar a la lista de palabras reservadas, las palabras IMPL y FOR.

21. A definir en Trabajos Prácticos 2/3.

23. A definir en Trabajos Prácticos 2/3.

28. A definir en Trabajos Prácticos 2/3.

31. A definir en Trabajos Prácticos 2/3.

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! %

Decisiones de diseño e implementación

Para la realización de la primera etapa se utilizó el lenguaje de programación Java.

Se creó el **Analizador Léxico** el cuál se encarga de leer el programa fuente. Cada vez que lee un carácter, cambia de estado (mediante una matriz de transición de estados) y ejecuta la acción semántica (Almacenada en una matriz de Acciones Semánticas) asociada a ese cambio de estado.

El reconocimiento de un token por parte del Analizador Léxico se realiza mediante un diagrama de transición de estados (autómata finito) el cuál fue implementado con dicha matriz de transición de estados de tamaño $N \times M$ donde N es la cantidad de estados y M la cantidad de símbolos que se pueden reconocer. De la misma manera se creó la matriz de acciones semánticas la cuál almacena las distintas acciones que se deben tomar ante un cambio de estado (arcos del autómata). Ambas matrices son atributos que posee la clase Analizador Léxico.

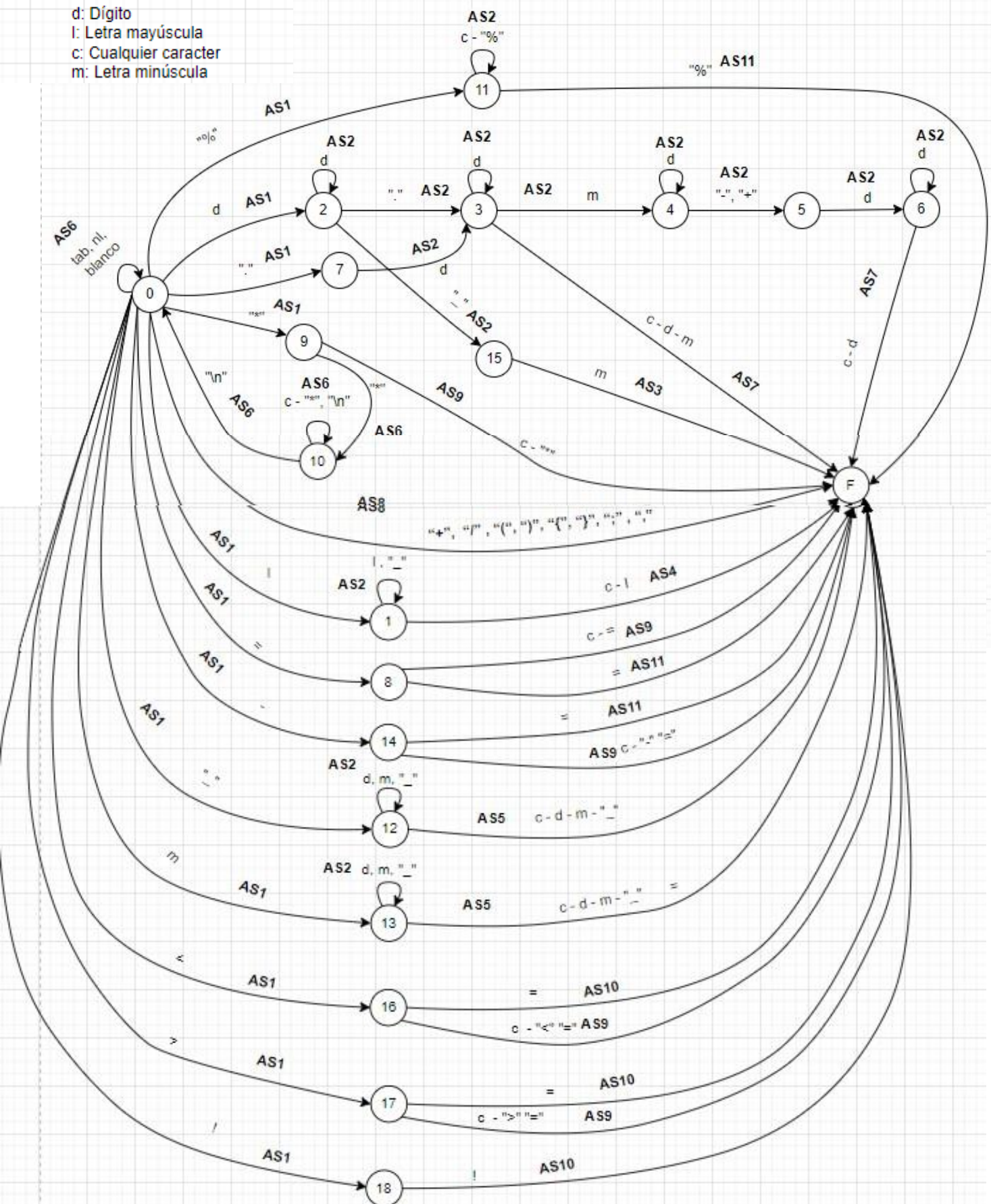
Por otro lado, se implementó la clase **Tabla de Símbolos**, la cual mediante un HashMap almacena los tokens que el Analizador Léxico va generando, donde la clave es un String (Lexema del token) y el valor es una clase conteniendo el valor correspondiente y la cantidad de veces que se repite el lexema en el código.

También se construyó una **Tabla de palabras reservadas** con un HashMap la cual almacena las distintas palabras reservadas del lenguaje, como también los tokens no literales como el "==" o "CADENA".

Con respecto a las **Acciones Semánticas** consideramos implementarla mediante una interfaz con el método abstracto acción, donde cada AS particular extiende de la misma y sobrescribe el método para realizar distintas acciones. Dicho método recibe como parámetros un *StringBuilder* que representa al token actual, y un **Reader** o lector con el objetivo de poder avanzar en la lectura de la correspondiente AS.

Diagrama de transición de estados

d: Dígito
 l: Letra mayúscula
 c: Cualquier caracter
 m: Letra minúscula



Matriz de transición de estados

	l ?	d	m	"*"	"%"	"_"	"="	"/"	"{"	"}"	"["	"]"	","	","	":"	"+"	"_"	>	<	!	BL	tab	nl
0	1	2	13	9	11	14	8	F	F	F	F	F	F	F	7	F	12	17	16	18	0	0	0
1	1	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	1	F	F	F	F	F	F
2	0	2	15	0	0	0	0	0	0	0	0	0	0	0	3	0	15	0	0	0	0	0	0
3	4	3	4	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
4	0	4	0	0	0	5	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
5	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	F	6	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
7	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
9	F	F	F	10	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
10	10	10	10	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0
11	11	11	11	11	F	11	11	11	11	11	11	11	11	F	11	11	11	11	11	11	11	11	11
12	F	12	12	F	F	F	F	F	F	F	F	F	F	F	F	F	12	F	F	F	F	F	F
13	F	13	13	F	F	F	F	F	F	F	F	F	F	F	F	F	13	F	F	F	F	F	F
14	F	F	F	F	F	0	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
15	0	0	F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	0	F	F	F	F
17	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	0	F	F	F	F	F
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	F	0	0	0

Matriz de acciones semánticas

	l ?	d	m	"*"	"%"	"_"	"="	"/"	"{"	"}"	"["	"]"	","	","	":"	"+"	"_"	>	<	!	BL	tab	nl
0	1	1	1	1	1	1	1	8	8	8	8	8	8	8	1	8	1	1	1	1	6	6	6
1	2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	2	4	4	4	4	4	4
2	-1	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	2	-1	-1	-1	-1	-1	-1
3	2	2	2	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
4	-1	2	-1	-1	-1	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	-1	-1	-1
5	-1	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	7	2	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
7	-1	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	9	9	9	9	9	9	10	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	6	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
10	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
11	2	2	2	2	11	2	2	2	2	2	2	2	2	11	2	2	2	2	2	2	2	2	6
12	5	2	2	5	5	5	5	5	5	5	5	5	5	5	5	5	2	5	5	5	5	5	5
13	5	2	2	5	5	5	5	5	5	5	5	5	5	5	5	5	2	5	5	5	5	5	5
14	9	9	9	9	9	6	10	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
15	-1	-1	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
16	9	9	9	9	9	9	10	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
17	9	9	9	9	9	9	10	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	10	-1	-1	-1

Descripción del mecanismo empleado para implementar la matriz de transición de estados y la matriz de acciones semánticas.

Como se planteó antes, son matrices NxM donde N es la cantidad de estados y M la cantidad de símbolos.

La matriz de transición de estados, se llena con el estado al que hay que ir desde el estado en el que se encuentra dependiendo el símbolo que tenga, cada uno son números (del 1 al 18) exceptuando la F que significa finalización, donde en el código vuelve al estado 0.

La matriz de acciones semánticas, se llena con valores donde cada uno representa a una acción semántica, 1 = AS1, 2 = AS2, etc, exceptuando -1 que significa la acción semántica error ASE.

Lista de acciones semánticas

AS1: Inicializa el string y agrega carácter al lexema.

AS2: Agrega carácter al lexema.

AS3: Verifica el rango de int y ulong y devuelve el token.

AS4: Verifica la existencia de la palabra reservada en la tabla de PR y devuelve el token.

AS5: Verifica la longitud de ID y devuelve el token.

AS6: Sin acción.

AS7: Verifica el rango del Double y devuelve el token.

AS8: Reconoce literal y devuelve el token.

AS9: Devuelve el token.

AS10: Agrega el carácter y devuelve el token compuesto

AS11: Finaliza la cadena y devuelve el token.

ASE: AS correspondiente al error.

ASE: AS correspondiente al error

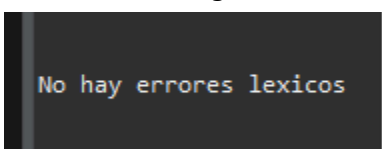
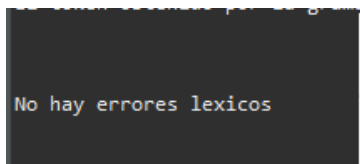
Errores léxicos considerados

El analizador Léxico puede producir ciertos errores léxicos a la hora de compilar el código fuente:

Estos informan un warning con la línea correspondiente al error en el caso que las constantes o identificadores están fuera de rango y luego se truncan al valor correspondiente dependiendo su máximo o mínimo, como también notifica cuando cualquier token no fue escrito correctamente.

Pruebas

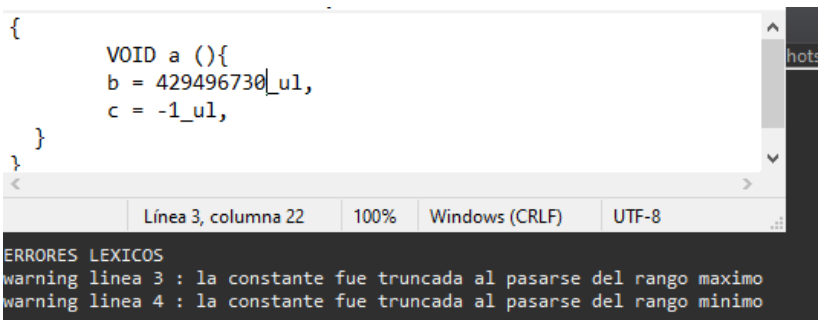
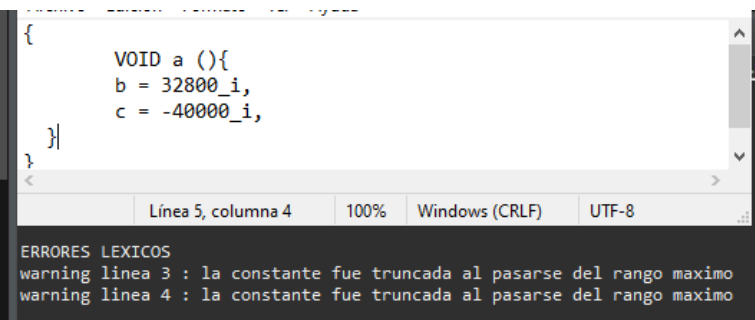
Constantes dentro del rango



```
VOID a (){\nb = 32767_i,\nc = -32768_i,\n}
```

```
VOID a (){\nb = 429496729_ul,\nc = 0_ul,\n}
```

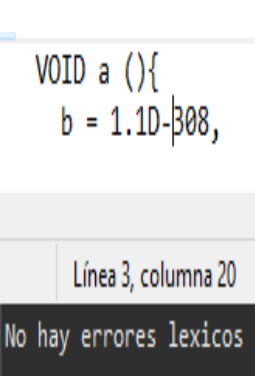
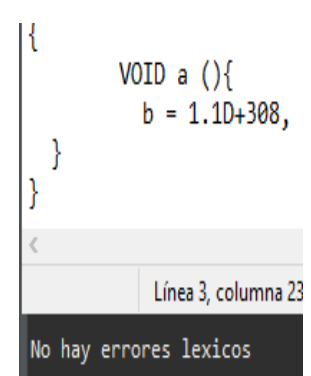
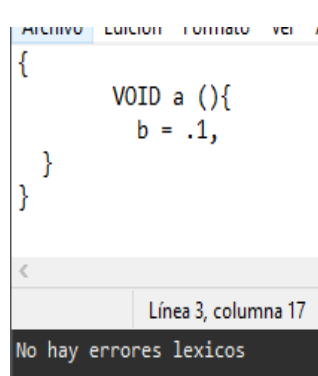
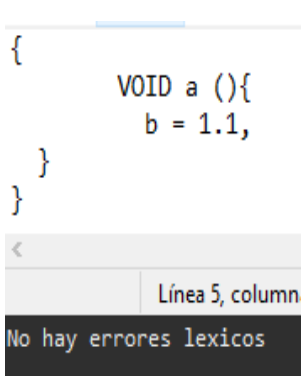
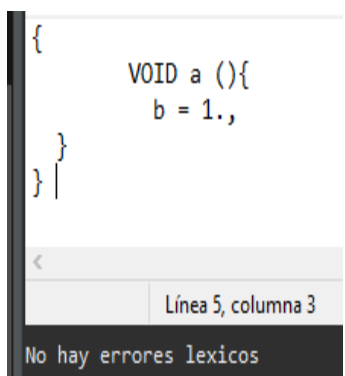
Constantes fuera del rango



```
{\n  VOID a (){\n    b = 32800_i,\n    c = -40000_i,\n  }\n}
```

```
{\n  VOID a (){\n    b = 429496730_ul,\n    c = -1_ul,\n  }\n}
```

Pruebas double



```
{\n  VOID a (){\n    b = 1.,\n  }\n}
```

```
{\n  VOID a (){\n    b = 1.1,\n  }\n}
```

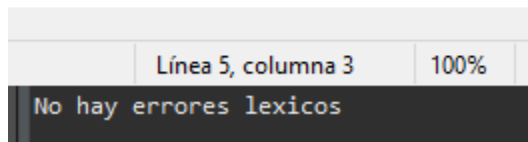
```
{\n  VOID a (){\n    b = .1,\n  }\n}
```

```
{\n  VOID a (){\n    b = 1.1D+308,\n  }\n}
```

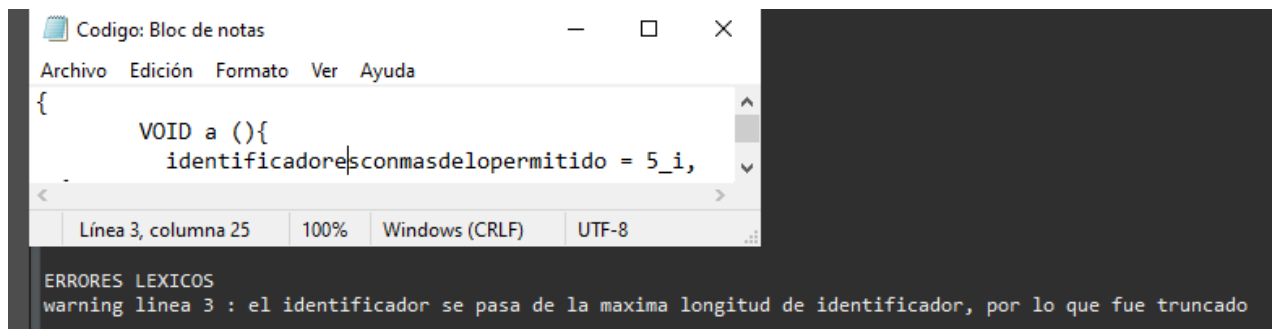
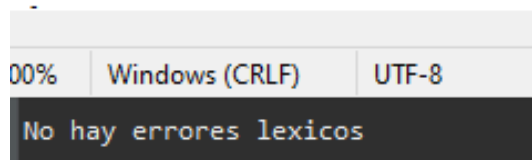
```
VOID a (){\nb = 1.1D-308,\n}
```

Identificadores

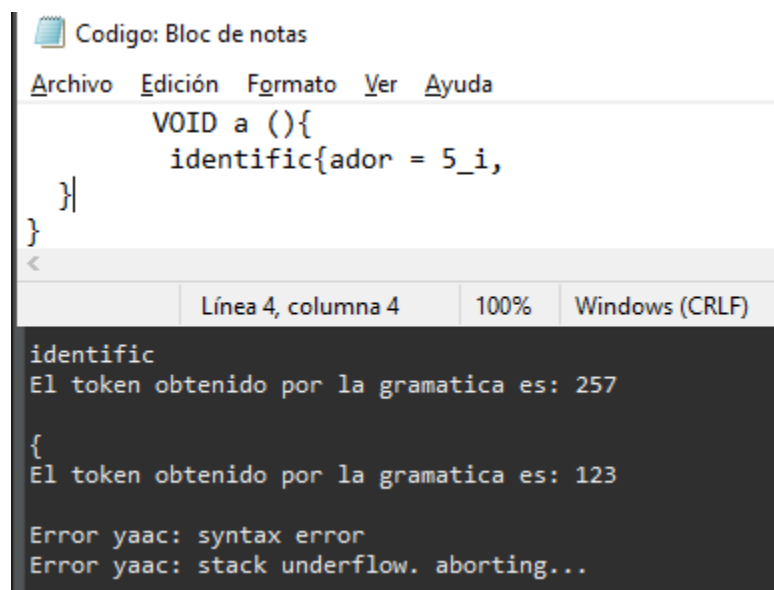
```
VOID a (){  
    identificador_12 = 5_i,  
}
```



```
VOID a (){  
    identificador = 5_i,  
}
```

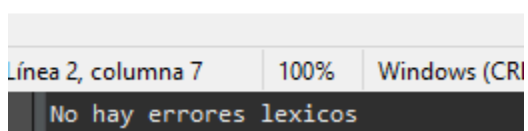


Identificador mal escrito (El léxico separa en 3 tokens el identificador)

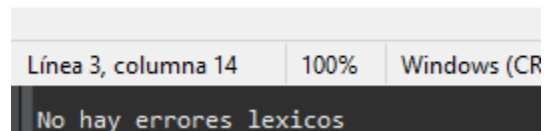


Palabras Reservadas

```
void a (){  
    int identificador = 5_i,  
}
```



```
VOID a (){  
    INT identificador = 5_i,  
}
```



Comentarios

(comentario mal escrito sin resolver)

```
VOID a (){
    **identificador = 5_i,|
}
```

Línea 3, columna 33	100%	Windows
---------------------	------	---------

No hay errores lexicos

```
VOID a (){
    identificador = 5_i,
    *identificador = 5_i,
}
```

Línea 5, columna 4	100%	Windows (CRLF)	UT
--------------------	------	----------------	----

Error yaac: syntax error
Error yaac: stack underflow. aborting...

Cadenas

```
VOID a (){
    PRINT %identificador%,
}
```

Línea 4, columna 4	100%
--------------------	------

No hay errores lexicos

(cadena mal escrita sin resolver)

nivo	edicion	formato	ver	ayuda
------	---------	---------	-----	-------

```
VOID a (){
    PRINT %identificador,
}
```

Línea 3, columna 24	100%
---------------------	------

No hay errores lexicos

Analizador Sintáctico

CONSIDERACIONES GENERALES

- a) Utilizar YACC u otra herramienta similar para construir el Parser.
- b) Adaptar el Analizador Léxico del Trabajo Práctico 1 para convertirlo en el método o función `int yylex()` (o el nombre que el Parser generado requiera). Tener en cuenta que el léxico deberá devolver al parser, en cada invocación, un token. Para los identificadores, constantes y cadenas, deberá devolver, además, la referencia a la entrada de la Tabla de Símbolos donde se ha registrado dicho símbolo, utilizando `yylval` para hacerlo.
- c) Para aquellos tipos de datos que permitan valores negativos (SHORT, INT, LONG, FLOAT, DOUBLE) durante el Análisis Sintáctico se deberán detectar constantes negativas, modificando la tabla de símbolos según corresponda. Será necesario volver a controlar el rango de las constantes, ya que un valor aceptado para una constante por el Analizador Léxico, que desconoce su signo, podría estar fuera de rango si la constante es positiva.
- **Ejemplo:** Las constantes de tipo INT pueden tomar valores desde -32768 a 32767. El Léxico aceptará la constante 32768 como válida, pero si se trata de una constante positiva, estará fuera de rango.
- d) Cuando se detecte un error, la compilación debe continuar. e) Conflictos: Eliminar TODOS LOS CONFLICTOS SHIFT-REDUCE Y REDUCE-REDUCE que se presenten al generar el Parser.

Descripción del proceso de desarrollo

Durante el proceso de desarrollo nos surgieron conflictos shift-reduce y reduce-reduce, por lo que decidimos borrar la regla que nos producía estos.

Para el manejo de errores lo que hicimos fue poner incompleta la regla en la gramática y mostrar imprimiendo por pantalla los errores encontrados.

Lista de no terminales usados

program: Regla principal de la gramática (%start).

bloque: Permite definir el bloque del programa.

sentencia_declarativa: permite declarar sentencias declarativas

declaracion_variables: Permite la declaración de variables junto a su tipo

lista_variables: Permite la declaración de variables

tipo: Permite la declaración de algún tipo de variables

declaracion_funcion: Permite la declaración de una función

inicio_encabezado : Indica el inicio de la declaración de función

fin_encabezado: Indica el fin de la declaración de función

parametro: Permite la declaración de la constante junto a su tipo

cuerpo_funcion: Permite la declaración de el cuerpo de una función

sentencias: Regla que permite la declaración de sentencias

sentencia: Permite la declaración de una sentencia declarativa o ejecutable

lista_sentencias_ejecutables: Permite la declaración de sentencias ejecutables

sentencia_ejecutable: Permite la declaración de sentencias ejecutables

declaracion_asignacion: Permite la declaración de una asignación

expresion_aritmetica: Permite las operaciones aritméticas suma y resta, como poner el factor

termino: Permite las operaciones aritméticas multiplicación y división, como poner el factor

factor: Permite el uso de constantes o identificadores

invocacion_funcion: Permite la invocación a una función

declaracion_if: Permite la declaración del if

condicion_if: Permite la declaración de la condición del if

operador: Permite usar el operador de la condición

cuerpo_if: Permite la declaración del cuerpo if

cuerpo_else: Permite la declaración del cuerpo else

sentencia_return: Permite la declaracion de la sentencia correspondiente al return

sentencia_print: Permite la declaracion de la sentencia correspondiente al print

Lista de errores sintácticos considerados

Los errores que consideración son:

- Falta un "{" en program
- Falta un "}" en program
- Falta una "," en sentencia ejecutable
- Falta una "," en sentencia declarativa
- Falta el TIPO en la declaración de variables
- Falta el identificador de la función
- Falta un "{" en cuerpo función
- Falta un "(" en invocación función
- Falta un "(" en inicio encabezado
- Falta un ")" en inicio condición if
- Falta un "(" en inicio encabezado
- Falta un "{" en cuerpo if
- Falta un "}" en cuerpo if
- Falta el identificador para el tipo en parámetro
- Falta el tipo para el identificador en parámetro
- Falta cuerpo del IF en la declaración del mismo
- Falta la cadena después del PRINT

Consideraciones extras

Cuando se crea un caso de prueba el último token tiene que terminar un espacio para ser reconocido porque espera cualquier cosa luego de este.

Para obtener la línea lo que hacemos es ver si el autómata recibe un NL (new line) lo que hace que entre por la columna 22 de la matriz y sumo 1 a la cantidad de líneas leídas, por lo que cuando el parser encuentra un error sintáctico, al imprimir la línea algunas veces aparece mal.

En la declaración del rango de ULONG pusimos 429496729, quitamos un número ya que no pudimos asignar 429496729.

La longitud del identificador se tomó como 20, ya que en el enunciado dice 20, pero cuando se enuncian los casos de prueba dice 25.

Pruebas

Declaración de función, asignación, sentencia IF y ELSE

Función con parámetros

ERRORES LEXICOS
warning línea 9 : la constante fue truncada al pasarse del rango maximo

No hay errores sintacticos

ESTRUCTURAS DETECTADAS
Declaracion de funcion: línea 2
Declaracion de variable: línea 3
Asignacion: línea 4
Sentencia IF: línea 5
Asignacion: línea 9

TABLA DE SIMBOLOS
Lexema: 0.0 Token: 258
Lexema: 1.2 Token: 258
Lexema: (Token: 40
Lexema: x Token: 257
Lexema:) Token: 41
Lexema: 1.7976931348623157 Token: 258
Lexema: { Token: 123
Lexema: , Token: 44
Lexema: f3 Token: 257
Lexema: = Token: 61
Lexema: } Token: 125
Lexema: > Token: 62

Codigo: Bloc de notas

Archivo Edición Formato Ver A

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

Línea 14, columna 4

No hay errores lexicos

No hay errores sintacticos

ESTRUCTURAS DETECTADAS
Declaracion de funcion: línea 2
Declaración de variable: línea 3
Asignacion: línea 4

TABLA DE SIMBOLOS
Lexema: (Token: 40
Lexema: x Token: 257
Lexema: y Token: 257
Lexema:) Token: 41
Lexema: { Token: 123
Lexema: , Token: 44
Lexema: f3 Token: 257
Lexema: = Token: 61
Lexema: } Token: 125

Codigo: Bloc de notas

Archivo Edición Formato Ver

```
{  
    VOID f3 (INT y) {  
        INT x,  
        x = y,  
        RETURN,  
    }  
}
```

Falta “,” en línea 3 y falta cerrar “}” de fin de programa

Falta el TIPO

No hay errores lexicos

ERRORES SINTACTCOS
Falta una ',' en la línea 3
Se esperaba un } en la línea 6

ESTRUCTURAS DETECTADAS
Declaracion de funcion: línea 2
Declaracion de variable: línea 4
Asignacion: línea 4

TABLA DE SIMBOLOS
Lexema: (Token: 40
Lexema: x Token: 257
Lexema: y Token: 257
Lexema:) Token: 41
Lexema: { Token: 123
Lexema: , Token: 44
Lexema: f3 Token: 257
Lexema: = Token: 61
Lexema: } Token: 125

Codigo: Bloc de notas

Archivo Edición Formato Ver

{

 VOID f3 (INT y){
 INT x
 x = y,
 RETURN,
 }
}

Línea 7, columna 2

ERRORES SINTACTCOS
Falta el TIPO en la línea 3

ESTRUCTURAS DETECTADAS
Declaracion de funcion: línea 2
Asignacion: línea 4

TABLA DE SIMBOLOS
Lexema: (Token: 40
Lexema: x Token: 257
Lexema: y Token: 257
Lexema:) Token: 41
Lexema: { Token: 123
Lexema: , Token: 44
Lexema: f3 Token: 257
Lexema: = Token: 61
Lexema: } Token: 125

Codigo: Bloc de notas

Archivo Edición Formato Ver

{

 VOID f3 (INT y)
 x,
 x = y,
 RETURN,
}
}

Línea 7, columna 3

Falta “(” en línea 2 y el TIPO

Invocación a función

No hay errores lexicos

ERRORES SINTACTCOS
Falta un (en la línea 2
Falta el TIPO en la línea 3

ESTRUCTURAS DETECTADAS
Asignacion: línea 4
Sentencia RETURN: línea 5

TABLA DE SIMBOLOS
Lexema: x Token: 257
Lexema: y Token: 257
Lexema:) Token: 41
Lexema: { Token: 123
Lexema: , Token: 44
Lexema: f3 Token: 257
Lexema: = Token: 61
Lexema: } Token: 125

Codigo: Bloc de notas

Archivo Edición Formato Ver

{

 VOID f3 INT y){
 x,
 x = y,
 RETURN,
 }
}

Línea 5, columna 9

No hay errores lexicos

No hay errores sintacticos

ESTRUCTURAS DETECTADAS
Declaracion de funcion: línea 2
Declaracion de variable: línea 3
Sentencia RETURN: línea 4
Declaracion de funcion: línea 7
Declaracion de variable: línea 8
Asignacion: línea 9
Invocacion a funcion: línea 10
Sentencia RETURN: línea 11

TABLA DE SIMBOLOS
Lexema: (Token: 40
Lexema: x Token: 257
Lexema:) Token: 41
Lexema: y Token: 257
Lexema: { Token: 123
Lexema: , Token: 44
Lexema: f3 Token: 257
Lexema: } Token: 125
Lexema: = Token: 61
Lexema: f4 Token: 257

Codigo: Bloc de notas

Archivo Edición Formato Ver Ayuda

VOID f3 (){
 INT x,
 RETURN,
}

VOID f4 (INT y){
 INT x,
 y = x,
 f3(),
 RETURN,
}

Línea 11, columna 1 100%

No hay errores lexicos

No hay errores sintacticos

ESTRUCTURAS DETECTADAS
Declaracion de funcion: línea 2
Declaracion de variable: línea 3
Asignacion: línea 4
Sentencia RETURN: línea 5
Declaracion de funcion: línea 8
Declaracion de variable: línea 9
Invocacion a funcion: línea 10
Sentencia RETURN: línea 11

TABLA DE SIMBOLOS
Lexema: (Token: 40
Lexema: x Token: 257
Lexema: y Token: 257
Lexema:) Token: 41
Lexema: { Token: 123
Lexema: , Token: 44
Lexema: f3 Token: 257
Lexema: = Token: 61
Lexema: } Token: 125
Lexema: f4 Token: 257

Invocacion a funcion con parametro: BI

Archivo Edición Formato Ver Ayuda

{

 VOID f3 (INT y){
 INT x,
 y = x,
 RETURN,
 }

 VOID f4 (){
 INT y,
 f3(y),
 RETURN
 }

Línea 1, columna 1 11

Invocación a función con parámetros

Conclusión

Con el desarrollo de los trabajos prácticos 1 y 2, pudimos aprender el funcionamiento e implementación de un analizador léxico y un analizador sintáctico Desarrollando paso a paso la comunicación entre ambos para llevar a cabo el envío de tokens desde el analizador léxico hacia el analizador sintáctico y cómo es el chequeo de errores por parte del parser.