Informe de desarrollo de Scanner

Proyecto Scanner - Lenguaje ABS
Compiladores e Intérpretes

Edgar André Araya Vargas
Daniel Zeas Brown
Olman Gerardo Alvarado Zúñiga

23 de septiembre de 2025
Versión 1.0

Índice

1. Resumen Ejecutivo

El presente informe detalla los aspectos más relevantes en el desarrollo del SCANNER para el lenguaje de programación denominado ABS, componente fundamental en la primera etapa de análisis léxico de un compilador. El proyecto se desarrolló utilizando la herramienta JFlex para la generación automática del analizador léxico, implementando todas las especificaciones requeridas para el reconocimiento de tokens válidos del lenguaje ABS.

Los resultados obtenidos demuestran un funcionamiento exitoso del scanner, logrando identificar correctamente palabras reservadas, operadores, identificadores, literales numéricos y de cadena, así como la detección apropiada de errores léxicos. El sistema procesó satisfactoriamente 8 casos de prueba exhaustivos, validando la funcionalidad completa del analizador léxico.

2. Introducción

El presente informe recoge el desarrollo e implementación de un scanner o analizador léxico, componente fundamental en el proceso de compilación de un lenguaje de programación. El scanner es responsable de transformar la secuencia de caracteres del código fuente en unidades mínimas con significado, conocidas como tokens, que servirán como entrada para el análisis sintáctico y las etapas posteriores del compilador.

En el desarrollo de este componente, se consideraron aspectos cruciales como la definición precisa de los patrones léxicos que identifican los distintos tipos de tokens, la gestión eficiente de la lectura del código fuente, la correcta eliminación de espacios y comentarios, así como el manejo y reporte de errores léxicos. Asimismo, se diseñó una interfaz funcional que permite la interacción fluida entre el scanner y el resto del compilador.

Este informe detalla las bases teóricas del análisis léxico, describe el proceso de desarrollo del scanner, y analiza las decisiones técnicas tomadas para garantizar un funcionamiento correcto y eficiente. El documento busca servir como un recurso tanto para la comprensión de la tarea del análisis léxico como para la implementación práctica de scanners en proyectos de compiladores.

3. Estrategia de Solución

3.1. Herramientas Utilizadas

Para el desarrollo del scanner se seleccionó JFlex como herramienta principal para la generación automática del analizador léxico. JFlex permite definir expresiones regulares que describen los patrones de tokens válidos del lenguaje ABS, generando automáticamente el código Java correspondiente para el reconocimiento de dichos patrones.

3.2. Arquitectura del Sistema

El sistema se compone de los siguientes elementos principales:

- **Archivo JFlex (.flex):** Contiene las definiciones de expresiones regulares y reglas léxicas

- **Lexer generado:** Código Java generado automáticamente por JFlex
- **Clases de soporte:** Token, TokenType y Main para el manejo y procesamiento de tokens
- **Sistema de reportes:** Generación de listados de tokens encontrados y errores léxicos

3.3. Decisiones Técnicas

1. **Case-insensitive para palabras reservadas:** Se implementó reconocimiento de palabras reservadas sin distinción entre mayúsculas y minúsculas
2. **Soporte múltiple de comentarios:** Se incluyó soporte para comentarios de línea (//) y de bloque ({}, (*), /**/)
3. **Manejo de diferentes bases numéricas:** Implementación de reconocimiento para números decimales, hexadecimales, binarios y octales
4. **Recuperación de errores:** El scanner continúa procesando después de encontrar errores léxicos

4. Análisis de Resultados

4.1. Funcionalidades Implementadas

Funcionalidad	Estado	Observaciones
Reconocimiento de palabras reservadas	100 %	Implementado con soporte case-insensitive
Números decimales	100 %	Reconocimiento completo de enteros
Números hexadecimales	100 %	Formato $FF, $abc implementado
Números binarios	100 %	Formato %1010 implementado
Números octales	100 %	Formato &777 implementado
Números reales	100 %	Soporte para notación científica
Strings y caracteres	100 %	Manejo de secuencias de escape
Operadores aritméticos	100 %	Todos los operadores reconocidos
Operadores de comparación	100 %	Incluyendo <=, >=, <>
Comentarios	100 %	Soporte para todos los tipos
Identificadores básicos	95 %	Limitación: no acepta _ al inicio
Detección de errores	85 %	Detecta caracteres ilegales básicos

4.2. Limitaciones Identificadas

1. **Caracteres UTF-8:** El lexer no maneja correctamente caracteres especiales como 'ñ'

2. **Identificadores con underscore:** No se permiten identificadores que inicien con ’_’
3. **Validación de números:** Números malformados como "5..º ".5"no se detectan como errores

5. Lecciones Aprendidas

5.1. Aspectos Técnicos

- **Orden de reglas en JFlex:** Las reglas léxicas necesitan un orden específico para funcionar bien. Por ejemplo, -+"debe ir antes que -"para que no se tokenice como dos signos más separados.
- **Estados para comentarios:** Usar estados diferentes para comentarios multilínea ayudó a manejar mejor los comentarios anidados sin romper el análisis.
- **Expresiones regulares:** Definir bien las expresiones regulares tomó varios intentos. Era fácil que fueran muy específicas o muy generales.
- **Palabras reservadas case-insensitive:** Para que "BEGINƨ "begin"fueran iguales, convertimos todo a minúsculas al comparar, no en las expresiones regulares.
- **Continuar después de errores:** Hacer que el lexer siga funcionando después de encontrar errores permitió mostrar todos los problemas de una vez.

5.2. Trabajo en Equipo

- **División de tareas:** Cada uno se encargó de una parte diferente (Olman las reglas JFlex, André el código Java, Daniel las pruebas) y funcionó bien.
- **Comunicación por Telegram:** Usar el chat de grupo ayudó a mantener a todos informados sobre cambios y problemas que iban surgiendo.
- **Revisar el código entre todos:** Que cada uno revisara el trabajo de los otros ayudó a encontrar errores que uno solo no ve.

5.3. Aspectos Personales

- **Aprender JFlex desde cero:** Nunca habíamos usado JFlex, así que tomó tiempo entender cómo funcionaba y cómo escribir las reglas.
- **Debugging paso a paso:** Aprendimos a probar cosas de a poco para encontrar problemas más fácil, en vez de cambiar todo y no saber qué falló.
- **Limitaciones técnicas:** Encontrar problemas como que no maneja bien caracteres especiales (ñ) nos enseñó a documentar qué no funciona.

6. Casos de Prueba

6.1. Caso de Prueba #1: Palabras Reservadas Básicas

Objetivo: Verificar que el scanner reconoce correctamente las palabras reservadas del lenguaje ABS en diferentes contextos y que es case-insensitive.

Archivo: sample1.txt

Código de entrada:

```
begin
  if x > 0 then
    while y < 10 do
      for i := 1 to 5 do
        case z of
          1: procedure test;
          2: function calc;
        end;
    end;
  else
    repeat
      var x: integer;
    until false;
end.
```

Resultado obtenido:

```
=== Tokens encontrados ===
begin           3
if              4
x               4, 14
>               4
0               4
then            4
while           5
y               5
<               5
10              5
do              5, 6
for             6
i               6
:               6, 8, 9, 14
=               6
1               6, 8
to              6
5               6
case            7
z               7
of              7
procedure       8
test            8
;               8, 9, 10, 11, 14, 15
2               9
function        9
calc            9
end             10, 11, 16
else            12
repeat          13
var             14
```

```
integer          14
until            15
false            15
.                16

=== Errores léxicos ===
No se encontraron errores léxicos.
```

Análisis:

- Palabras reservadas detectadas correctamente: begin, if, then, while, do, for, to, case, of, procedure, function, end, else, repeat, var, until
- Operadores reconocidos: >, <, :, =, ;, .
- Identificadores válidos: x, y, i, z, test, calc, integer, false
- Números: 0, 10, 1, 5, 2
- Conteo de líneas correcto: Los tokens aparecen en las líneas esperadas
- Sin errores léxicos: El código es sintácticamente válido a nivel léxico

Conclusión: CASO EXITOSO - El scanner identifica correctamente todas las palabras reservadas y demás elementos léxicos.

6.2. Caso de Prueba #2: Números (Todos los Formatos)

Objetivo: Verificar que el scanner reconoce correctamente números decimales, hexadecimales, binarios, octales y reales con notación científica.

Archivo: sample2.txt

Código de entrada:

```
var
  decimal := 123;
  negativo := -456;
  hexadecimal := $FF;
  hexMinuscula := $abc;
  binario := %1010;
  binarioLargo := %11110000;
  octal := &777;
  octalPequeño := &123;
  realBasico := 3.14;
  realCientificoPos := 2.5E+3;
  realCientificoNeg := 1.5E-4;
  realSoloE := 123E5;
```

Resultado obtenido:

```
=== Tokens encontrados ===
var              3
decimal          4
:                4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
=                4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
123              4
;                4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
negativo         5
```

```
-               5
456             5
hexadecimal     6
$FF             6
hexMinuscula    7
$abc            7
binario         8
%1010           8
binarioLargo    9
%11110000       9
octal           10
&777            10
octalPeque      11
o               11
&123            11
realBasico      12
3.14            12
realCientificoPos 13
2.5E+3          13
realCientificoNeg 14
1.5E-4          14
realSoloE       15
123E5           15

=== Errores léxicos ===
Error léxico en línea 11, columna 13: lexema «ñ»
```

Análisis:

- Números decimales: 123, 456 detectados correctamente
- Números hexadecimales: $FF, $abc reconocidos como HEX_NUMBER
- Números binarios: %1010, %11110000 detectados como BIN_NUMBER
- Números octales: &777, &123 reconocidos como OCTAL_NUMBER
- Números reales: 3.14, 2.5E+3, 1.5E-4, 123E5 clasificados como REAL_NUMBER
- PROBLEMA: .ºctalPequeño"se dividió por el carácter 'ñ'

Conclusión: MAYORMENTE EXITOSO - Todos los tipos numéricos funcionan, limitación con caracteres UTF-8.

6.3. Caso de Prueba #3: Strings y Caracteres

Objetivo: Verificar reconocimiento de strings con escapes y caracteres individuales.
Archivo: sample3.txt
Código de entrada:

```
var
  mensaje := "Hola mundo";
  vacio := "";
  conEscapes := "Linea 1\nLinea 2\tTabulado";
  comillas := "Texto con \"comillas\" internas";
  caracter := 'A';
  caracterEscape := '\n';
```

```
  simbolo := '@';
  numero := '5';
```

Resultado obtenido:

```
=== Tokens encontrados ===
var             3
mensaje         4
:               4, 5, 6, 7, 8, 9, 10, 11
=               4, 5, 6, 7, 8, 9, 10, 11
"Hola mundo"    4
;               4, 5, 6, 7, 8, 9, 10, 11
vacio           5
""              5
conEscapes      6
"Linea 1\nLinea 2\tTabulado" 6
comillas        7
"Texto con \"comillas\" internas" 7
caracter        8
'A'             8
caracterEscape  9
'\n'            9
simbolo         10
'@'             10
numero          11
'5'             11

=== Errores léxicos ===
No se encontraron errores léxicos.
```

Análisis:

- Strings básicos detectados correctamente ("Hola mundo")
- String vacío reconocido ()
- Secuencias de escape procesadas correctamente (\n, \t, \")
- Caracteres individuales válidos ('A', '@', '5')
- Caracteres con escape manejados correctamente ('\n')
- Strings complejos con escapes internos funcionan

Conclusión: CASO EXITOSO - El procesamiento de strings y caracteres funciona perfectamente.

6.4. Caso de Prueba #4: Operadores y Delimitadores

Objetivo: Verificar que todos los operadores y delimitadores del lenguaje ABS son reconocidos correctamente.

Archivo: sample4.txt

Código de entrada:

```
begin
  x := y + z - w * v / u;
  potencia := base ** exponente;
```

```
  comparacion := (a < b) and (c > d) or (e <= f) and (g >= h);
  diferente := x <> y;
  incremento := counter++;
  decremento := index--;
  arreglo := lista[0];
  record := objeto.campo;
  puntero := variable^;
end.
```

Resultado obtenido:

```
=== Tokens encontrados ===
begin           3
x               4, 7
:               4, 5, 6, 7, 8, 9, 10, 11, 12
=               4, 5, 6, 7, 8, 9, 10, 11, 12
y               4, 7
+               4
z               4
-               4
w               4
*               4
v               4
/               4
u               4
;               4, 5, 6, 7, 8, 9, 10, 11, 12
potencia        5
base            5
**              5
exponente       5
comparacion     6
(               6
a               6
<               6
b               6
)               6
and             6
c               6
>               6
d               6
or              6
e               6
<=              6
f               6
g               6
>=              6
h               6
diferente       7
<>              7
incremento      8
counter         8
++              8
decremento      9
index           9
--              9
arreglo         10
lista           10
[               10
0               10
```

```
]                10
record           11
objeto           11
.                11, 13
campo            11
puntero          12
variable         12
^                12
end              13

=== Errores léxicos ===
No se encontraron errores léxicos.
```

Análisis:

- Operadores aritméticos: +, -, *, /, ** todos detectados
- Operadores de comparación: <, >, <=, >=, <>reconocidos
- Operadores lógicos: and, or clasificados correctamente
- Operadores especiales: ++, – detectados
- Delimitadores: (,), [,], ., îtodos reconocidos

Conclusión: CASO EXITOSO - Todos los operadores y delimitadores funcionan perfectamente.

6.5. Caso de Prueba #5: Comentarios

Objetivo: Verificar que todos los tipos de comentarios son ignorados correctamente por el scanner.

Archivo: sample5.txt

Código de entrada:

```
// Comentario de línea simple
begin
  x := 5; { comentario de llaves en línea }

  {
    comentario de llaves
    multilínea
  }

  (* comentario de paréntesis
     también multilínea *)

  /* comentario estilo C
     que también funciona */

  y := 10; // otro comentario al final
end.
```

Resultado obtenido:

```
=== Tokens encontrados ===
begin           3
x               4
:               4, 17
=               4, 17
5               4
;               4, 17
y               17
10              17
end             18
.               18

=== Errores léxicos ===
No se encontraron errores léxicos.
```

Análisis:

- Comentarios // ignorados correctamente (no aparecen tokens de comentarios)
- Comentarios { } ignorados
- Comentarios (* *) ignorados
- Comentarios /* */ ignorados
- Solo tokens válidos aparecen en salida

Conclusión: CASO EXITOSO - Todos los tipos de comentarios son ignorados correctamente.

6.6. Caso de Prueba #6: Identificadores Válidos

Objetivo: Verificar el reconocimiento de identificadores con diferentes formatos y convenciones de nombrado.

Archivo: sample6.txt

Código de entrada:

```
var
  variable1: integer;
  _underscore: real;
  CamelCase: string;
  snake_case_var: boolean;
  numeroAlFinal123: integer;
  _123conNumeros: real;
  MAYUSCULAS: string;
  minusculas: integer;
  MeZcLaDo: boolean;
```

Resultado obtenido:

```
=== Tokens encontrados ===
var             3
variable1       4
:               4, 5, 6, 7, 8, 9, 10, 11, 12
integer         4, 8, 11
;               4, 5, 6, 7, 8, 9, 10, 11, 12
underscore      5
```

```
real            5, 9
CamelCase       6
string          6, 10
snake_case_var  7
boolean         7, 12
numeroAlFinal123 8
123             9
conNumeros      9
MAYUSCULAS      10
minusculas      11
MeZcLaDo        12

=== Errores léxicos ===
Error léxico en línea 5, columna 3: lexema «_»
Error léxico en línea 9, columna 3: lexema «_»
```

Análisis:

- Identificadores con números al final funcionan (variable1, numeroAlFinal123)
- PROBLEMA: Identificadores que inician con _ generan error
- CamelCase detectado correctamente
- snake_case detectado como un solo identificador
- Mayúsculas y minúsculas mezcladas funcionan

Conclusión: PARCIALMENTE EXITOSO - Funciona bien excepto identificadores con _ inicial.

6.7. Caso de Prueba #7: Errores Léxicos

Objetivo: Verificar que el scanner detecta y reporta errores léxicos apropiadamente.
Archivo: sample7.txt
Código de entrada:

```
begin
  x := 123@;        // @ es ilegal
  y := #invalid;    // # no definido
  z := 5.;          // real mal formado
  w := .5;          // real mal formado
  error := 123abc; // número seguido de letras
  otro := $GZ;      // hex inválido
  malo := %102;     // binario inválido
end.
```

Resultado obtenido:

```
=== Tokens encontrados ===
begin           3
x               4
:               4, 5, 6, 7, 8, 9, 10
=               4, 5, 6, 7, 8, 9, 10
123             4, 8
;               4, 5, 6, 7, 8, 9, 10
y               5
invalid         5
```

```
z               6
5               6, 7
.               6, 7, 11
w               7
error           8
abc             8
otro            9
GZ              9
malo            10
%10             10
2               10
end             11

=== Errores léxicos ===
Error léxico en línea 4, columna 11: lexema «@»
Error léxico en línea 5, columna 8: lexema «#»
Error léxico en línea 9, columna 11: lexema «$»
```

Análisis:

- Caracteres ilegales detectados (@, #, $) en posiciones correctas
- PROBLEMA: Números malformados (5., .5) no detectados como errores
- PROBLEMA: "123abc"se separó en lugar de error único
- Errores reportan línea y columna correctas
- Programa continúa después de errores

Conclusión: PARCIALMENTE EXITOSO - Detecta caracteres ilegales pero validación de números incompleta.

6.8. Caso de Prueba #8: Programa Completo

Objetivo: Verificar que el scanner maneja correctamente un programa completo con múltiples elementos léxicos mezclados.

Archivo: sample8.txt

Código de entrada:

```
program TestCompleto;
var
  contador: integer;
  mensaje: string;
  valor: real;

begin
  contador := 0;
  mensaje := "Procesando...";

  // Loop principal
  while contador < 100 do
  begin
    if (contador mod 2) = 0 then
      valor := contador * 3.14
    else
      valor := contador / 2.0;
```

```
    contador++;
  end;

  { Fin del programa }
end.
```

Resultado obtenido:

```
=== Tokens encontrados ===
program         3
TestCompleto    3
;               3, 5, 6, 7, 10, 11, 19, 21, 22
var             4
contador        5, 10, 14, 16, 17, 19, 21
:               5, 6, 7, 10, 11, 17, 19
integer         5
mensaje         6, 11
string          6
valor           7, 17, 19
real            7
begin           9, 15
=               10, 11, 16, 17, 19
0               10, 16
"Procesando..." 11
while           14
<               14
100             14
do              14
if              16
(               16
mod             16
2               16
)               16
then            16
*               17
3.14            17
else            18
/               19
2.0             19
++              21
end             22, 25
.               25

=== Errores léxicos ===
No se encontraron errores léxicos.
```

Análisis:

- Programa completo procesado sin errores
- Todas las palabras reservadas detectadas
- Números enteros y reales mezclados correctamente
- Strings y comentarios ignorados correctamente
- Estructura de control completa reconocida

Conclusión: CASO EXITOSO - El scanner maneja perfectamente programas complejos.

6.9. Resumen de Casos de Prueba

Caso de Prueba	Estado	Observaciones
1. Palabras Reservadas	EXITOSO	Reconoce todas las palabras reservadas correctamente
2. Números	PARCIAL	Funciona bien, pero falla con caracteres UTF-8 (ñ)
3. Strings y Caracteres	EXITOSO	Maneja strings y escapes perfectamente
4. Operadores	EXITOSO	Todos los operadores funcionan bien
5. Comentarios	EXITOSO	Ignora correctamente todos los tipos de comentarios
6. Identificadores	PARCIAL	Funciona bien excepto identificadores que empiezan con _
7. Errores Léxicos	PARCIAL	Detecta caracteres ilegales pero no valida formato de números
8. Programa Completo	EXITOSO	Procesa programas complejos sin problemas

Cuadro 2: Resumen de resultados de casos de prueba

Estadísticas finales:

- Total de casos ejecutados: 8/8
- Casos completamente exitosos: 5
- Casos parcialmente exitosos: 3
- Casos fallidos: 0

Conclusión general: El scanner cumple con los requisitos principales del proyecto. Las limitaciones encontradas son menores y no afectan la funcionalidad core requerida.

7. Manual de Usuario

7.1. Requisitos Previos

1. **Visual Studio Code:** Descargue e instale Visual Studio Code

2. **Java JDK:** Instale la versión más reciente del JDK de Java
3. **Configuración VS Code:** Asegúrese de que VS Code detecte el JDK
4. **Extensiones necesarias:**
   - Java Extension Pack
   - JFlex

7.2. Ejecución del Programa

Para compilar y ejecutar el scanner:

1. Presione `Ctrl + Shift + P`
2. Escriba Run Taskz seleccione la opción
3. Escriba Runz presione Enter

Este proceso realizará automáticamente:

- Generación del archivo Lexer.java con JFlex
- Compilación de todos los archivos .java
- Ejecución de la clase principal

7.3. Descripción del Funcionamiento

El programa:

- Crea una instancia del Lexer
- Escanea el archivo sample.txt
- Imprime tokens encontrados clasificados por tipo
- Reporta errores léxicos con ubicación

8. Bitácora de Trabajo

Fecha	**Observación**	**Otros**
8/9/2025	Consulta con asistente sobre ejemplos de formatos JFlex	Brandon atiende a Olman durante horario de atención
11/9/2025	Se comparte formato JFlex tentativo al grupo	N/A
12/9/2025	Reunión rápida para revisar formato JFlex	André indica revisión más profunda

16/9/2025	Se recibe ejemplo de formato JFlex de Brandon	Solicitud de Olman
16/9/2025	Creación de Drive compartido y formato de documento	Olman genera carpeta e inicia documento
19/9/2025	André logra ejecutar el scanner exitosamente	Implementación en Java completada
20/9/2025	Daniel ejecuta casos de prueba exhaustivos	8 casos de prueba documentados

9. Referencias Bibliográficas

Referencias

IJRASET. (s.f.). *Developing programming language with compilers using JFlex in Netbean*. Recuperado el 16 de septiembre de 2025, de `https://www.ijraset.com/research-paper/developing-programming-language-with-compilers-using-jflex-in-netbean`

JFlex. (s.f.). *JFlex manual*. Recuperado el 10 de septiembre de 2025, de `https://jflex.de/manual.html`

OpenAI. (2023). *ChatGPT* (versión GPT-4) Validación de orden de tokens. Recuperado el 11 de septiembre de 2025, de `https://chat.openai.com/`