Universidad Nacional Autónoma de México

Facultad de Ciencias

Práctica 1: Analizadores Léxicos con Lex (Flex)

Materia: Compiladores 2026-1

Profesora: Ariel Adara Mercado Martínez

Alumno: Adrián Lima García

19 de septiembre de 2025

Objetivo

Que el alumno conozca y utilice los principios para generar analizadores léxicos utilizando la herramienta **Lex/Flex**.

Introducción

Un analizador léxico es el encargado de leer la cadena de entrada y dividirla en unidades significativas llamadas **tokens**. Para construirlo, se usan expresiones regulares que describen los patrones válidos en el lenguaje.

Lex (y su versión en Linux llamada Flex) permite definir estas expresiones regulares y generar de forma automática el código del analizador léxico en C o C++.

Desarrollo

Estructura de un programa en Lex

Un programa de Lex se divide en tres secciones:

- **Sección de declaraciones**: se incluyen librerías y variables globales.
- **Sección de expresiones regulares**: reglas de patrones y sus acciones.
- **Sección de código de usuario**: funciones auxiliares, incluyendo la función principal.

Primer programa en Flex

```
1  // Archivo: primer.ll
2  %{
3    #include <iostream>
4  %}
5
6  %option c++
7  %option noyywrap
8
9  digito  [0-9]
10 letra   [a-zA-Z]
11 palabra {letra}+
12 espacio [ \t\n]
13
14 %%
15
16 {espacio}  { /* ignorar espacios y tabuladores */ }
17 {digito}+  { std::cout << "Encontre␣un␣numero:␣" << yytext << std::
     endl; }
```

```
18 {palabra}  { std::cout << "Encontre una palabra: " << yytext << std
       ::endl; }
19
20 %%
21
22 int main() {
23   FlexLexer* lexer = new yyFlexLexer;
24   lexer->yylex();
25 }
```

Ejecución

```
$ flex++ primer.ll
$ g++ lex.yy.cc -o programa
$ ./programa
```

—

Ejercicios y respuestas

1. **¿Qué ocurre si en la primera sección se quitan las llaves al nombre de la macro letra?** El compilador marcará error, porque al definir una macro en Lex siempre se deben usar llaves {} al llamarla dentro de las reglas.

2. **¿Qué ocurre si en la segunda sección se quitan las llaves a las macros?** No se reconocerán correctamente, Lex interpretará el nombre como una cadena literal y no como una macro.

3. **¿Cómo se escribe un comentario en flex?** Se pueden escribir comentarios en C/C++: `/* comentario */` o `// comentario`.

4. **¿Qué se guarda en yytext?** La cadena exacta de caracteres que coincide con la expresión regular reconocida.

5. **¿Qué pasa al ejecutar el programa e introducir cadenas de caracteres y de dígitos por la consola?** Detecta y muestra si son palabras o números. Los espacios se ignoran.

6. **¿Qué ocurre si introducimos caracteres como "*.^{en} la consola?** No existe una regla para este símbolo, por lo que el programa lo imprime sin clasificar o lo ignora según la configuración.

—

Ejercicio 7

Se modificó el programa para que reconozca:

- Números hexadecimales en C++ (`0x[0-9a-fA-F]+`).
- 5 palabras reservadas: `int`, `float`, `if`, `else`, `while`.
- Identificadores válidos de C++ de hasta 32 caracteres.
- Espacios en blanco.

```
// Archivo: ejercicio.ll
%{
  #include <iostream>
%}

%option c++
%option noyywrap

hexa        0[xX][0-9a-fA-F]+
reservada   int|float|if|else|while
id          [a-zA-Z_][a-zA-Z0-9_]{0,31}
espacio     [ \t\n]

%%

{espacio}      { /* ignorar espacios */ }
{hexa}         { std::cout << "Hexadecimal: " << yytext << std::endl; }
{reservada}    { std::cout << "Reservada: " << yytext << std::endl; }
{id}           { std::cout << "Identificador: " << yytext << std::endl; }
.              { std::cout << "Caracter desconocido: " << yytext << std::endl; }

%%

int main() {
  FlexLexer* lexer = new yyFlexLexer;
  lexer->yylex();
}
```

—

Analizador léxico para el lenguaje C_1

En la carpeta `src/C_1` se implementó un analizador con los archivos:

- `lexer.ll` – reglas en Flex.
- `tokens.hpp` – definiciones de tokens.
- `prueba` – archivo de prueba de entrada.
- `Makefile` – para compilar fácilmente.

La salida generada es la que se pide en el enunciado.
—

Conclusión

Con esta práctica comprendí cómo Lex permite traducir expresiones regulares a código que reconoce patrones en texto. Aunque al inicio parece complejo, el uso de macros y reglas facilita mucho el trabajo. Esta práctica me dio las bases para generar analizadores léxicos más avanzados en siguientes prácticas.