¿Qué ocurre si en la primera sección se quitan las llaves al nombre de la macro letra?

```
/*palabra={letra}+*/
palabra=letra+
espacio=[ \t\n]
```

El macro palabra será igual a la cadena letra , donde “a” puede repetirse infinitamente , por eso las expresiones regulares que definimos no separan en lexemas la cadena de entrada

```
┌──(josefer㉿kali2)-[~/Documents/7_septimo/Compiladores/JFLEX]
└─$ java Lexer input.txt
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
HayEncontré un número: 100
nubesenelcielohoy
```

¿Qué ocurre si en la segunda sección se quitan las llaves a las macros?

En la regla espacio si le quitamos las llaves entonces deja de hacer referencia a los macros y se convierte en una cadena “espacio” y solo si en el archivo de entrada está la palabra espacio entonces se ejecutara esta regla

```
21
22   /** seccion de expresiones regulares
23   espacio {/* La acción léxica puede
24   {digito}+ { System.out.print("Encon
25   {palabra} { System.out.print("Encon
26
27   /** Si hubiera seccion de codigo te
28
```

```
┌──(josefer㉿kali2)-[~/Documents/7_septimo/Compiladores/JFLEX]
└─$ java Lexer input.txt
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Encontré una palabra: Hay
 Encontré un número: 100
 Encontré una palabra: nubes
 Encontré una palabra: en
 Encontré una palabra: el
 Encontré una palabra: cielo
 Encontré una palabra: hoy
```

Mismo caso cuando quito las llaves a {digito}+ se convierte en una simple cadena.

```
19
20   %%
21
22   /** seccion de expresiones regula
23   {espacio} {/* La acción léxica pu
24   digito+ { System.out.print("Encon
25   {palabra} { System.out.print("Enc
26
27   /** Si hubiera seccion de codigo
28
29
```

```
┌──(josefer㉿kali2)-[~/Documents/7_septimo/Compiladores/JFLEX]
└─$ java Lexer input.txt
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Encontré una palabra: Hay
100Encontré una palabra: nubes
Encontré una palabra: en
Encontré una palabra: el
Encontré una palabra: cielo
Encontré una palabra: hoy

┌──(josefer㉿kali2)-[~/Documents/7_septimo/Compiladores/JFLEX]
└─$
```

Mismo caso que antes , ahora la regla espera hacer match con una cadena igual “palabra ”

```
%%

/** seccion de expresiones
{espacio} {/* La acci□n l□x
{digito}+ { System.out.print
palabra { System.out.print("

/** Si hubiera seccion de c
```

```
┌──(josefer㉿kali2)-[~/Documents/7_septimo/Compiladores/JFLEX]
└─$ java Lexer input.txt
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
HayEncontré un número: 100
nubesenelcielohoy
```

Si se lo quitamos a todas las reglas entonces ya no hace match con ninguna

```
%%

/** seccion de expresione
espacio {/* La acci□n l□
digito+ { System.out.pri
palabra { System.out.pri

/** Si hubiera seccion d
```

```
┌──(josefer㉿kali2)-[~/Documents/7_septimo/Compiladores/JFLEX]
└─$ java Lexer input.txt
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Hay 100 nubes en el cielo hoy
```

¿Cómo se escribe un comentario en flex?
Con /** */

¿Qué se guarda en yytext?
El acceso al lexema del último token reconocido se realiza a través de la variable yytext.

¿Qué pasa al ejecutar el programa e introducir cadenas de caracteres y de dígitos en el archivo de entrada?

Flex está íntimamente relacionado con la teoría de autómatas porque, internamente, el analizador léxico generado por Flex funciona como un **Autómata Finito Determinista (DFA, por sus siglas en inglés)**.Esto se aplica de la siguiente manera :

1. **Expresiones Regulares y Autómatas**:
    - Las expresiones regulares que se definen en el archivo de especificación son, en esencia, descripciones de lenguajes regulares. Según la teoría de autómatas, cualquier lenguaje regular puede ser representado por un autómata finito.
2. **Conversión a DFA**:
    - Flex convierte las expresiones regulares de la especificación en un autómata finito no determinista (NFA). Luego, mediante un proceso llamado determinización, este NFA se convierte en un DFA, que es un autómata finito determinista. Un DFA es más eficiente porque, para cualquier cadena de entrada y estado actual, tiene exactamente una transición posible.
3. **Simulación del DFA**:
    - El código java generado por jflex simula el funcionamiento de este DFA. Cada vez que el programa analiza un carácter del texto de entrada, cambia de estado en el DFA según las transiciones definidas por los patrones (expresiones regulares). Cuando el DFA llega a un estado final (un estado que corresponde a un patrón completo), se ejecuta la acción asociada a ese patrón.
4. **Eficiencia del DFA**:
    - Los DFA son muy eficientes para el análisis léxico porque pueden procesar cada carácter de la entrada en un tiempo constante, sin necesidad de retroceder o hacer cálculos complicados para decidir la próxima acción. Esto es crucial para el rendimiento de compiladores e intérpretes.

¿Qué ocurre si introducimos caracteres como "*" en el archivo de entrada?

No lo reconoce y por la construcción del programa con jflex se imprime en pantalla ese carácter pero no hace match con ninguna expresión regular descrita