Sesión 02**: Expresiones regulares**

I. OBJETIVOS

- Compreender las expresiones regulares
- Utilizar las funcionalidades básicas de flex para definir expresiones regulares
- Crear una aplicación con expresiones regulares

II. TEMAS A TRATAR

- Flex
- Expresiones regulares

III. MARCO TEÓRICO

Expresión regular

Las expresiones regulares son una forma de especificar **patrones,** entendiendo por patrón la forma de describir cadenas de caracteres. Es la forma de definir los **tokens o componentes léxicos** y, como veremos, cada patrón concuerda con una serie de cadenas.

De esta forma, utilizamos las expresiones regulares para darle nombre a estos patrones. Ejemplo:

Token (componente léxico)	**Lexema**	**Patrón**
Identificador	a, valor, b	[a-zA-Z]+
Número	5, 3,25, 56	[0-9]+(\.[0-9]+)?

Vemos que para describir un identificador que se define solo por letras utilizando el patrón [a-zAZ]+, que reconoce cualquier letra mayúscula o minúscula seguido del símbolo +, estamos indicando que al menos tiene que haber una letra para describir un identificador, pero no hay límite para el número de caracteres que puede tener ese token (identificador). Una expresión regular se puede construir a partir de otras expresiones regulares más simples. Cuando definamos los símbolos mediante los que se especifican las expresiones regulares veremos ejemplos de estos, definiendo letras y dígitos y como un identificador es una combinación de ambos.

Para definir las expresiones regulares usamos **metacaracteres** o **metasímbolos** que especifican las acciones que se pueden reconocer sobre un determinado carácter o símbolo. Algunos de estos metacaracteres son: *****, **+**, **?**, **|**.

Cuando queremos utilizar estos símbolos, como por ejemplo * con su significado normal, para la operación de multiplicar se utiliza un carácter de escape que anula el significado especial del metacarácter. En este ejemplo la forma correcta sería *, y así tenemos el símbolo de multiplicar en un patrón.

Operaciones con expresiones regulares

Hay tres operaciones básicas con expresiones regulares:

Selección entre alternativas	Se denota por el metacarácter **|**. Ejemplo: a|b, significa que puede ser a ó b. Esta operación equivale a la unión, puesto que tanto a como b valdrían como lexemas para este patrón (obsérvese que utilizamos patrón y expresión regular de forma indistinta).
Concatenación	Se construye poniendo un símbolo al lado del otro y no utiliza ningún metacarácter. Ejemplo: ab, significa que el lexema equivalente tiene que ser "ab", sin alternativa posible
Repetición	También se la denomina cerradura de Kleene y se denota por el matacarácter *. Identifica una concatenación de símbolos incluyendo la cadena vacía, es decir "0 o más instancias" del símbolo afectado. Ejemplo: a*, significa que los lexemas para este patrón podrían ser: , a, aa, aaa, aaaa,...

Algunos operadores de FLEX son:

`x`
: empareja el carácter `'x'`

`.`
: cualquier carácter excepto una línea nueva

`[xyz]`
: un conjunto de caracteres; en este caso, el patrón empareja una `'x'`, una `'y'`, o una `'z'`

`[abj-oZ]`
: un conjunto de caracteres con un rango; empareja una `'a'`, una `'b'`, cualquier letra desde la `'j'` hasta la `'o'`, o una `'Z'`

`[^A-Z]`
: cualquier carácter menos los que aparecen en el conjunto. En este caso, cualquier carácter EXCEPTO una letra mayúscula.

`[^A-Z\n]`
: cualquier carácter EXCEPTO una letra mayúscula o una línea nueva

r`*`
: cero o más *r*'s, donde *r* es cualquier expresión regular

r`+`
: una o más *r*'s

r`?`
: cero o una *r* (es decir, "una *r* opcional")

r`{2,5}`
: entre dos y cinco concatenaciones de *r*

r`{4}`
: exactamente 4 *r*'s

`{nombre}`
: la expansión de la definición de "nombre" (ver más abajo)

`"[xyz]\"foo"`
: la cadena literal: [xyz]"foo

`\x`
: si *x* es una `'a'`, `'b'`, `'f'`, `'n'`, `'r'`, `'t'`, o `'v'`, entonces la interpretación ANSI-C de \x (por ejemplo \t sería un tabulador). En otro caso, un literal `'x'` (usado para la concordancia exacta de caracteres especiales (\\ \. \?)

`(r)`
: empareja una *R*; los paréntesis se utilizan para anular la precedencia (ver más abajo)

`rs`
: la expresión regular *r* seguida por la expresión regular *s*; se denomina "concatenación"

`r|s`
: bien una *r* o una *s*

`r/s`
: una *r* pero sólo si va seguida por una *s*.

`^r`
: una *r*, pero sólo al comienzo de una línea

`r$`
: una *r*, pero sólo al final de una línea (es decir, justo antes de una línea nueva). Equivalente a "*r*/\n".

`<s>r`
: una *r*, pero sólo en la condición de arranque *s* (ver más adelante).

`<s1,s2,s3>r`
: lo mismo, pero en cualquiera de las condiciones de arranque *s1*, *s2*, o *s3*

IV. ACTIVIDADES (La práctica tiene una duración de 2 horas académicas)

Realiza las siguientes actividades:

1. Imprimir “numero del 0,1,2,3 y 4”, cuando se escriba 0,,1,2,3 o 4

   ```
   %option main
   %{

   %}
   %%
   [01234] { printf("numero 1,2 o 3\n");}
   %%
   ```

2. Imprimir “número 1,2,3,4,5,6,7,8 o 9”, cuando se escribe del 0 al 9

   ```
   %option main
   %{

   %}
   %%
   ```

```
[0-9] { printf("numero 1,2,3,4,5,6,7,8 o 9\n");}
%%
```

3. Imprimir “una letra cualquiera”, cuando se escribe de la letra “a” a la “z”

```
%option main
%{

%}
%%
[a-z] { printf("una letra cualquiera\n");}
%%
```

4. Imprimir “es una consonante” cuando se escribe una consonante

```
%option main
%{

%}
%%
[b-df-hj-np-tv-z] { printf("es una consonante\n");}
%%
```

5. Imprimir “es una MAYUSCULA” cuando se escribe una letra en mayúsculas

```
%option main
%{

%}
%%
[A-Z] { printf("es una MAYUSCULA\n");}
%%
```

6. Imprimir “es una a” cuando se escribe una a, o más.

```
%option main
%{

%}
%%
[a]* { printf("es una a\n");}
%%
```

7. Imprimir “es una a” cuando se escribe una a, o más con signo de +

```
%option main
%{

%}
%%
[a]+ { printf("es una a\n");}
```

```
%%
```

8. Imprimir “es un 0“cuando se escribe uno o más cero

```
%option main
%{

%}
%%
[0]+ { printf("es una cero\n");}
%%
```

9. Imprimir “es binario” cuando se escribe 0 o 1

```
%option main
%{

%}
%%
[01]+ { printf("es binario\n");}
%%
```

10. Crear un archivo “digitos” que reconozca números naturales

```
%option main
%{

%}
%%
[0-9] {printf("Es digito\n");}
%%
```

```
%option main
%{
%}

digito [0-9]
%%
{digito}  {printf("Es un digito");}
%%
```

11. Definir el patrón DIGITO fuera de la zona de reglas y definir la regla de combinación de digitos para que sea reconocido como entero.

```
%option main
DIGITO [0-9]
%{

%}
%%
```

```
{DIGITO}{DIGITO}* {printf("\nEs un entero");}

%%
```

12. Imprimir “Es un entero” usando la función atoi y la variable yytext.

```
%option main
DIGITO [0-9]
%{

%}
%%

{DIGITO}{DIGITO}* {printf("\nEs un entero %d",atoi(yytext));}

%%
```

13. Imprimir es un entero a pesar del punto decimal

```
%option main
DIGITO [0-9]
%{

%}
%%

{DIGITO}{DIGITO}* {printf("\nEs un entero %d",atoi(yytext));}
, ;
%%
```

14. Imprimir es un entero a pesar de la coma decimal

```
%option main
DIGITO [0-9]
%{

%}
%%

{DIGITO}{DIGITO}* {printf("\nEs un entero %d",atoi(yytext));}
, ;
%%
```

15. Imprimir “Es una cadena” usando solo la variable “yytext”.

```
%option main
%{

%}
%%
```

```
[a-zA-Z_][a-zA-Z_]* {printf("\nEs una cadena %s \n",yytext);}
, ;
%%
```

16. Imprimir la cantidad de cifras que tiene un número ingresado

```
%option main
DIGITO [0-9]
%{

%}
%%

{DIGITO}{DIGITO}* {printf("\nel número tiene longitud %d",yyleng);}
. ;
%%
```

17. Imprimir la longitud de una cadena ingresada usando “yyleng”

```
%option main
%{

%}
%%

[a-zA-Z_][a-zA-Z_]* {printf("\nEs una cadena de longitud:%d \n",yyleng);}
, ;
%%
```

18. Prueba el siguiente programa y describe para a las variables “yytext” y “yyleng”

```
%option main
DIGITO [0-9]
%%
{DIGITO}{DIGITO}*       { printf(" \nEs n entero %d", atoi(yytext));}
[a-zA-Z_][a-zA-Z0-9_]*  { printf(" \nEs una cadena %s \n", yytext);
                          printf("Que tiene %d caracteres",yyleng);}
. ;
%%
```

19. Crear un programa que reconozca números reales con y sin signo: (números con punto flotante) e imprima que es un número real.
20. Crear un programa que reconozca sumas (de la forma número_natural + número_natural) e imprima que es una suma.
21. Crear un programa que reconozca restas (de la forma número_natural - número_natural) e imprima que es una suma.
22. Crear un programa que reconozca multiplicaciones (de la forma número_real * número_real = número_real) e imprima que es una multiplicación.
23. ¿Qué cadenas reconoce el siguiente programa?

```
%option main
%{

%}
letra  [a-zA-Z]
digito [0-9]
%%
"if"     {printf("IF\n");}
"else"   {printf("ELSE\n");}
{letra}({letra}|{digito})* {printf("IDENTIFICADOR: %s \n",yytext);}
%%
```

VII. Ejercicios

1. Diseñar patrones flex que reconozcan los operadores DIV y MOD entre dos números reales (de la forma número_real DIV número_real).
2. Diseñar un patrón flex que reconozca operaciones combinadas entre números reales y enteros reconociendo los patrones: (+ ,- ,*, /, (,) ,=)
3. Diseñar un patrón flex que reconozca números de celular, (nueve números que pueden incluir espacios o guiones) e imprima que es un número de celular o que no lo es en caso contrario.
4. Diseñar un patron flex que reconozca el numero de dni
5. Diseñar un patron que reconozca una dirección ip
6. Redactar un informe de los ejercicios propuestos

VII. Bibliografía y referencias

1. https://es.qwe.wiki/wiki/Flex_(lexical_analyser_generator)
2. https://docplayer.es/46330679-Lex-flex-generacion-de-analizador-lexico-p-1.html
3. http://web.archive.org/web/20130627190411/
4. http://wwwdi.ujaen.es:80/~nacho/Flex.htm