Práctica No. 4
Programas en Lex

Objetivo
El objetivo de esta práctica es conocer el programa `flex` y utilizarlo para crear y compilar algunos ejemplos de programas Lex autónomos.

Introducción
Podemos definir a Lex cómo una herramienta para construir analizadores léxicos o*lexers*. Un lexer lee un flujo de entrada (normalmente de `stdin` o de algun otro archivo) y lo divide en unidades léxicas (llamadas tokens) cómo un producto final o para ser procesadas por otro programa. Lex fué desarrollado en los años 70 en los laboratorios Bell de la AT&T dentro del ambiente Unix. Para escribir una especificación léxica en Lex, es necesario crear un conjunto de patrones (Expresiones Regulares) que, cuando el programa se ejecute, servirán para identificar tokens en el flujo de entrada y, si así se especifica, ejecutar ciertas acciones asociadas a cada expresión regular.

Actividades
1.- Captura y compilación de programas lex autónomos
1.1- Usando un editor de texto, captura el siguiente ejemplo de programa lex y guárdalo en un archivo llamado `ejem1.l`

```
/* archivo: ejem1.l */
%{
#include<stdio.h>
%}
%%
[0-9]+                      printf("NUMERO\n");
[a-zA-Z][a-zA-Z0-9]*        printf("PALABRA\n");
%%
main()
{
          yylex();
}

int yywrap()
{
          return 1;
}
```

1.2.- Compila este programa lex con el comando: `flex -oejem1.c ejem1.l`
Este comando debe generar un archivo llamado `ejem1.c`. Usando el comando `dir ejem1*` determina si el archivo existe y su tamaño y fecha:

Archivo `ejem1.c` Tamaño: _____ Fecha: _____

1.3.- Compilación del programa `ejem1.c`
Para ejecutar el lexer, nos falta compilar el programa en C creado por el comando `lex`. Esto lo hacemos tecleando:

```
          tcc ejem1.c
```

Ejecuta el comando anterior y usando el comando `dir ejem1*` determina si existe el archivo ejecutable `ejem1.exe`, su tamaño y fecha

Archivo `ejem1.exe` Tamaño: _____ Fecha: _____

1.4.- Antes de probar la ejecución del lexer, describe en tus propias palabras la expresión regular de cada una de las reglas:

Regla	Interpretación
`[0-9]+`	_____
	_____
`[a-zA-Z][a-zA-Z0-9]*`	
	_____
	_____

1.5.- Ejecución del lexer `ejem1.exe`
Para ejecutar el lexer creado en el inciso anterior sólo debemos teclear su nombre, así que ejecuta el lexer `ejem1.exe` tecleando `ejem1` y reporta las salidas generadas para cada una de las siguientes entradas:

Entrada	Salida del lexer
`a`	_____
`b`	_____
`ab`	_____
`1`	_____
`12`	_____
`12.34`	_____
`abc123`	_____

Para detener la ejecución del lexer, teclea `Ctrl+D` (que genera un `EOF`, fin de archivo). ¿Concuerdan los resultados con tu interpretación de las reglas? Si no, trata de reinterpretar las reglas.
Interpretación correcta?: OK: _____

1.6.- Modificación de `ejem1.l`
Usando un editor de texto vamos ahora a modificar la definición del lexer `ejem1.l` cambiando la segunda regla:
Actual: `[a-zA-Z][a-zA-Z0-9]*`
Modificada: `[a-zA-Z][a-zA-Z0-9]+`

¿Cómo se interpreta ahora esta nueva regla `[a-zA-Z][a-zA-Z0-9]+`?

_____
_____
_____

1.7.- Recompíla el lexer con `flex`: `flex -oejem1.c ejem1.l` OK: ____
y compíla el programa en C generado: `tcc ejem1.c` OK: ____

1.8.- Ejecuta la nueva versión del lexer (tecleando `ejem1`) y reporta las salidas para cada una de las siguientes entradas:

Entrada	Salida del lexer
`a`	_____
`b`	_____
`ab`	_____
`1`	_____
`12`	_____
`12.34`	_____
`abc123`	_____

¿Concuerdan los resultados con tu interpretación de las reglas? Si no, trata de reinterpretar las reglas.
Interpretación correcta?: OK: _____

2.- Segundo ejemplo.

2.1.- Captura el siguiente programa lexer y guárdalo en un archivo llamado `ejem2.l`:

```
/* archivo: ejem2.l */
%{
#include <stdio.h>
%}
%%
end                {printf("Regla 1 -> Fin.\n"); return 0;}
[a-z]              printf("Regla 2: letra minuscula.\n");
[A-Z]              printf("Regla 3: letra mayuscula.\n");
[a-zA-Z]+          printf("Regla 4: Palabra.\n");
[0-9]+             printf("Regla 5: Numero entero.\n");
[0-9]*.[0-9]+      printf("Regla 6: Numero fraccionario.\n");
%%
main()
{
         yylex();
}

int yywrap()
{
         return 1;
}
```

2.2.- Compila este programa lex con el comando: `flex -oejem2.c ejem2.l`
Este comando debe generar un archivo llamado `ejem2.c`. Usando el comando `dir ejem2*` determina si el archivo existe y su tamaño y fecha:

Archivo `ejem2.c` Tamaño: _____ Fecha: _____

Compila el programa en C creado por el comando `flex` tecleando: `tcc ejem2.c`

Ejecuta el comando anterior y usando el comando `dir ejem2*` determina si existe el archivo ejecutable `ejem2.exe`, su tamaño y fecha

Archivo `ejem2.exe` Tamaño: _____ Fecha: _____

2.3.- Antes de probar la ejecución del lexer, describe en tus propias palabras la interpretación de la expresión regular de cada una de las reglas:

Regla	Interpretación
`end`	_____
	_____
`[0-9]+`	_____
	_____
`[a-z]`	_____
	_____
`[A-Z]`	_____
	_____

`[a-zA-Z]+`

_____

_____

`[0-9]*.[0-9]+`

_____

_____

2.4.- Ejecuta el lexer `ejem2` y reporta la salida para cada una de las siguientes entradas:

Entrada	Salida del lexer
`a`	_____
`A`	_____
`ab`	_____
`1`	_____
`12`	_____
`12.34`	_____
`.45`	_____
`Hola`	_____
`end`	_____

¿Concuerdan los resultados con tu interpretación de las reglas? Si no, trata de reinterpretar las reglas.
Interpretación correcta?: OK: _____

2.5.- Modificación de `ejem2.l`
i).- Se quiere modificar la regla de números fraccionarios para que todo número fraccionario tenga al menos un dígito en la parte entera. ¿Cúal es la regla que debe modificarse? ¿Qué modificación se requiere?
Regla a modificarse:

_____

Regla modificada:

_____

ii) Aplica la modificación al archivo `ejem2.l`, compila con `flex`, genera el ejecutable con `tcc` y prueba que la nueva regla funciona adecuadamente con las entradas siguientes:

Entrada	Salida del lexer
`1.23`	_____
`.45`	_____
`123`	_____
`end`	_____

¿Funciona adecuadamente la modificación? Sí: ____ No: ____
En caso negativo, revisa la modificación que se realizó y cámbiala si lo consideras necesario para poder cumplir con lo que se solicita.

3.- Expansión de `ejem2.l`
Se requiere agregar una regla 7 para identificar un nuevo tipo de token llamado `variable` que consiste en un _ (guión bajo) seguido de cualquier letra mayúscula seguido de al menos una letra minúscula. Su acción asociada es desplegar la frase `Regla 7.- Variable`

3.1.- ¿Cual es la expresión regular que define este tipo de token? Y ¿cuál es su acción asociada?

Expresión regular: _____

Acción asociada: _____

3.2.- Agrega esta regla al archivo `ejem2.l`, compila con `flex`, genera el ejecutable con `gcc` y prueba que la nueva regla funciona adecuadamente con las entradas siguientes:

Entrada	Salida del lexer
`hola`	_____
`_Hola`	_____
`_HOLA`	_____
`end`	_____

¿Funciona adecuadamente la modificación? Sí: ____ No: ____
En caso negativo, revisa la modificación que se realizó y cambíala si lo consideras necesario para poder cumplir con lo que se solicita.

4.- Comentarios y conclusiones

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____