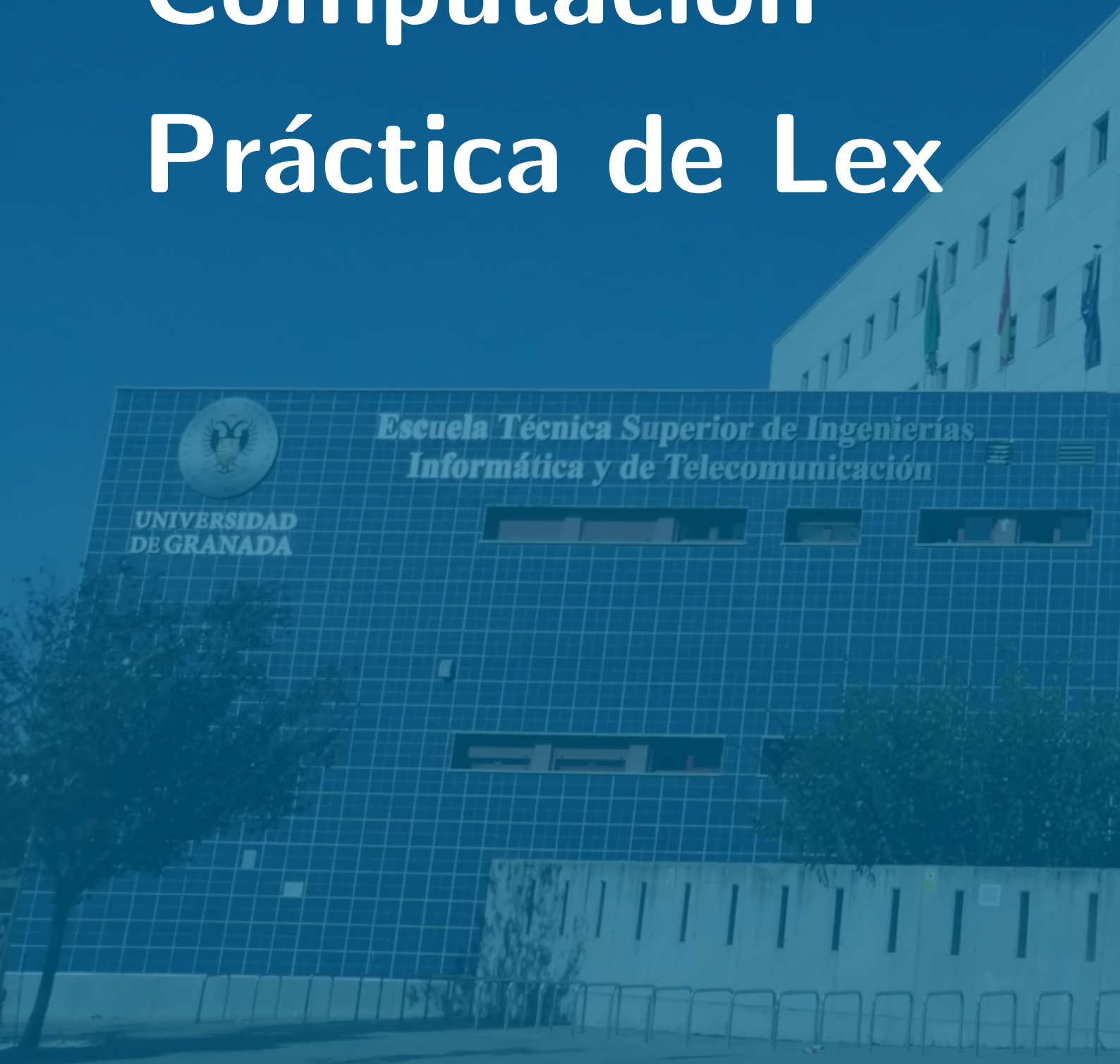
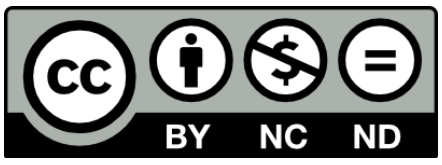
Modelos de Computación
Práctica de Lex



Los Del DGIIM, losdeldgiim.github.io

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada



Modelos de Computación
Práctica de Lex

Los Del DGIIM, `losdeldgiim.github.io`

José Juan Urrutia Milán
Irina Kuzyshyn Basarab

Granada, 2024

1. Motivación

En la gestión de cualquier organismo o empresa es necesaria la recopilación de datos de trabajadores, usuarios o clientes de la misma. Debida a la cantidad de datos que estas requieren, este procedimiento suele ser automático, para lo que se necesitan una serie de sistemas y algoritmos a desarrollar y mantener.

En esta práctica, supuesto que tenemos un programa de recopilación de datos tipo formulario (un documento donde los usuarios puedan escribir sus datos personales), crearemos un programa que tenga como objetivo obtener toda la información válida introducida por el usuario (en cualquier formato), para:

- Identificar el tipo de información: detectar si se trata de un nombre, de un número de teléfono, ...
- Poner la información en un formato estándar, con el objetivo de introducirla en una base de datos y que los datos introducidos por distintos usuarios estén en el mismo formato.

2. Desarrollo de la práctica

Desarrollamos pues un programa que sirva como filtro entre un programa de recopilación de datos (por ejemplo, una página web) y un programa que introduce datos en una base de datos. Como no contamos con ninguno de estos programas, usaremos los ficheros `stdin` y `stdout` como la salida del programa de recopilación y entrada al programa de la base de datos, respectivamente. Los datos que reconoceremos serán los siguientes:

- Nombres y Apellidos.
- Teléfonos.
- DNIs y NIEs.
- Correos electrónicos.
- Cuentas bancarias (números de IBAN de cuentas españolas).
- Fechas.

El funcionamiento del programa es simple: se van introduciendo datos por teclado (uno en cada línea), y los datos que vaya reconociendo el programa como alguno de los arriba nombrados, se comprobará que sean válidos en algunos casos, y se estandarizarán mostrándolo por pantalla indicando el dato que es.

Ejemplo. Veamos un sencillo ejemplo:

```
> 12345678z
DNI: 12345678Z
> a
> +34123456789
Teléfono: +34 | 123456789
```

Como, vemos cuando lee algo que no encaja en ninguno de los patrones buscados simplemente lo ignora.

Pasemos ahora a explicar los formatos que aceptamos como válidos, pues aunque la idea sea aceptar formatos variados y estandarizarlos, hay ciertas restricciones que vamos a poner; y los formatos estándar para cada tipo de dato. Una restricción común a todos los tipos es que un dato tiene que ir en una misma línea, si se haya un salto de línea se considerará que el dato se ha completado.

2.1. Nombres y Apellidos

Nombre válido.
Consideraremos que un nombre es válido cuando contemos con una palabra formada por al menos dos cadenas de caracteres alfabéticos. De esta forma, no aceptamos que el nombre de una persona esté conformado por una única cadena alfabética.

Formato estándar.
Dado un nombre válido, este estará conformado por al menos dos cadenas de caracteres separadas por un espacio. En función del número de cadenas de caracteres que tengamos, tendremos formas distintas de nombres:

- Si solo tenemos dos cadenas de caracteres, consideramos que la primera es el nombre de la persona y que la segunda es su único apellido.
- Si tenemos más de dos cadenas de caracteres, consideramos que las dos últimas son sus apellidos y que el resto forman el nombre de la persona (que puede estar formado por tantas cadenas de caracteres como se desee).

Haremos una distinción especial para los *nombres compuestos con partículas preposicionales*, como por ejemplo, "María del Carmen Fernández Martín", de forma que consideramos estas cinco cadenas como cuatro, por lo que tenemos a una persona cuyo primer nombre es "María" y segundo nombre es "del Carmen".

Finalmente, si el usuario introdujo un nombre que no tiene su primera letra en mayúscula o que tiene alguna letra que no corresponde a la primera de una cadena en mayúscula, lo modificaremos de forma que se quede la primera letra de cada cadena (salvo las preposicionales) en mayúscula y el resto de caracteres en minúscula.

Ejemplo. Mostramos un ejemplo de funcionamiento del programa con las siguientes cadenas de entrada:

```
> josé sáncHez de las nieves
Nombre: José || Sánchez | de las Nieves
> María   del carmen pilar lóPEZ
Nombre: María | del Carmen || Pilar | López
```

2.2. Teléfonos

Teléfono válido.

Consideramos que una palabra es un número de teléfono cuando:

- Bien está conformada por 9 caracteres numéricos que pueden estar separados por ".", "-", "/" o un espacio en blanco.
- Cuenta con un prefijo, es decir, un "+" seguido de una cadena numérica formada por 1, 2 o 3 caracteres (que pueden estar o no parentizados) seguido de un número válido (esto es, una palabra de la forma indicada en el punto superior).

Formato estándar.

Dado un número de teléfono válido, eliminaremos todos los separadores que puedan aparecer en él, de forma que al final mostraremos su prefijo (si no se indicó ninguno, suponemos que es un número de España e indicamos el prefijo "+34") seguido del número en cuestión.

Ejemplo. Mostramos un ejemplo del funcionamiento del programa:

```
> 666 66 66 66
Telefono: +34 | 666666666
> + (128)   666-666-666
Telefono: +128 | 666666666
```

2.3. DNIs y NIEs

Cadenas válidas.

Una cadena se considerará un DNI o un NIE si:

- Un DNI si consta de 8 dígitos juntos (sin espacios entre ellos) seguido de una letra.
- Un NIE si consta de una letra seguida de 7 dígitos juntos y de otra letra.

Formato estándar.

Dada una cadena que puede ser un DNI o NIE válido, eliminaremos todos los espacios en blanco que aparezcan separando las letras de los números y convertiremos todas las letras que aparezcan en mayúscula.

DNI o NIE válidos.

Una vez tengamos nuestra cadena en formato estándar, comprobaremos si efectivamente se trata de un DNI o NIE válido. Para ello:

- En el caso de los DNIs, consideramos el número formado por los 8 dígitos, calculamos su resto módulo 23 y consultamos la siguiente tabla para ver si la letra es correcta:

Resto	0	1	2	3	4	5	6	7	8	9	10	11
Letra	T	R	W	A	G	M	Y	F	P	D	X	B
Resto	12	13	14	15	16	17	18	19	20	21	22	
Letra	N	J	Z	S	Q	V	H	L	C	K	E	

- En el caso de los NIEs, sustituimos la primera letra por un número, tal y como indica la siguiente tabla:

Letra	X	Y	Z
Número	0	1	2

En caso de que la primera letra no sea una de esas, el NIE no es válido. Una vez hecha la sustitución, hemos de comprobar que lo que nos queda es un DNI válido.

Ejemplo. Ejemplo de ejecución

```
> 00473914   E
DNI: 00473914E
> Z 9530626  Y
NIE: Z9530626Y
> 67861125 H
> R0564785D
```

2.4. Correos electrónicos

Correos válidos.

Una palabra será un correo válido cuando este conste de una cadena de caracteres alfanuméricos (admitiendo también "_"), siendo el primer carácter de dicha cadena una letra; seguida de una "@" y de al menos dos cadenas de caracteres separadas por un punto.

Formato estándar.

Dada una cadena que consideramos un correo válido, la estandarizaremos simplemente pasando todos los caracteres alfabéticos a minúscula.

Ejemplo. Un ejemplo de ejecución:

```
> USUARIO@correo.ugr.es
Correo: usuario@correo.ugr.es
> usuario_47@gmail.com
Correo: usuario_47@gmail.com
```

2.5. Cuentas bancarias

Cadenas válidas.

Diremos que una palabra será una cuenta bancaria válida si está en el formato IBAN para cuentas españolas, que consta de 24 caracteres: 2 letras ("ES") seguidas de 22 dígitos. Admitiremos también dichas letras en minúscula, espacios entre las letras y los dígitos y entre cada grupo de dígitos, tal y como mostramos debajo:

$$ES\ XX\ XXXX\ XXXX\ XXXX\ XXXX\ XXXX$$

Admitimos que el usuario no introduzca alguno de dichos espacios (o que introduzca más de dos espacios seguidos en sitios donde podemos esperar al menos uno).

Formato estándar.
Dada una cuenta bancaria válida, eliminaremos todos los espacios en blanco y convertiremos todos los caracteres a mayúscula.

Cuentas válidas.
Una cuenta bancaria es válida si siendo una cadena para cuenta válida, ponemos los 4 primeros caracteres (las 2 letras y los 2 dígitos) al final de la palabra, quedando:

$$XXXXXXXXXXXXXXXXXXXXESXX$$

Transformamos las letras a números (de forma que a "A" le corresponde el 10, a la "B" el 11, ...) y el resto de dividir el número que nos queda entre 97 es 1.

Ejemplo. Mostramos ahora un ejemplo de ejecución, donde vemos que el programa no nos da la segunda salida por ser una cuenta no válida (su resto entre 97 no es 1):

```
> ES91 2100 0418 4502 0005 1332
IBAN: ES9121000418450200051332
> ES92 2100 0418 4502 0005 1332
```

2.6. Fechas

Fechas válidas.
Dado que las fechas se pueden introducir de un montón de formas, en esta práctica vamos a aceptar algunos de ellos que pasaremos a explicar a continuación. Algo que afecta a todas ellas es que el rango de años que vamos a tomar como válidos es desde 1800 hasta 2200.

- Lo que llamamos fecha simple, púramente numérica. Entre los números que representan el día, mes y año puede haber más de un separador (".", "/", "-" o espacio) o un "de" o "del". Además aceptamos varios órdenes distintos:
  - DD/MM/YYYY (por defecto)
  - MM/DD/YYYY
  - YYYY/MM/DD
  - DD/MM/YY (por defecto)
  - MM/DD/YY

  En primer lugar, a lo que nos referimos como por defecto, es que si entre las dos primeras (o las dos últimas) valieran las dos se toma la primera (y la penúltima). Por otro lado, en los casos YY si 20YY es menor o igual al año actual se pone 20YY, en otro caso se pone 19YY.

- La fecha con el mes escrito, en formato larto (ej: febrero, Febrero) o corto (ej: Feb, FEB, feb). En concreto las cadenas que aceptaremos son de la forma: dia + separadores + mes + separadores + año. Con separadores nos referimos a lo mismo que en la fecha simple. También se acepta el año acortado y se siguen las mismas reglas que antes.
- El último caso que vamos a tratar es que podamos ponerle el día de la semana (ej: Miercoles, Miércoles, miercoles, miércoles, x, X) por delante de los otros dos casos, seguido de una posible "," y de separadores (".", "/", "-" o espacio), al menos 1. Hay que destacar que el programa comprueba que el día de la semana introducido coincide con el día correspondiente a la fecha introducida, sino se descarta.

El diagrama de la Figura 1 representa cómo hemos conseguido que el programa identifique formatos de fechas tan variados.

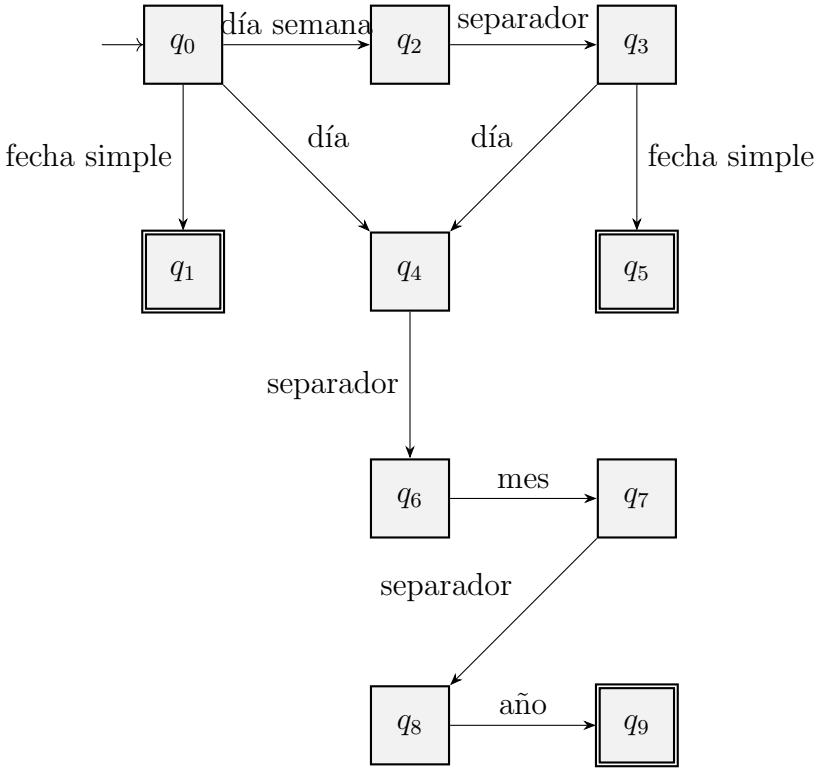


Figura 1: Diagrama del reconocedor de fechas.

- El estado q_0 simboliza el estado inicial.
- Los estados finales indican que ya se ha leído una fecha válida y se llama a una función que comprueba que la fecha es correcta e imprime la fecha, si es correcta, en el formato estándar que más tarde detallaremos.
- Las transiciones indican qué se lee (y guarda en unas variables que luego usaremos) para cambiar a otro estado, en el que se espera que se lean otras cosas.
- Tanto llegar a un estado final, como leer un salto de línea nos sitúa de nuevo en el estado inicial.

- Las transiciones de los separadores, como es evidente, no guardan nada, simplemente cambian de estado.

Formato estándar.
Dada una fecha válida, el formato estándar que hemos escogido es: DD/MM/YYYY.

Ejemplo. Un ejemplo de ejecución:

```
> 23--5   2003
Fecha: 23/05/2003
> 12/ene  del 23
Fecha: 12/01/2023
> lunes,   15 octubre---73
Fecha: 15/10/1973
```

Observación. Cabe destacar que hemos tenido que tener en cuenta algunas prioridades. Por ejemplo cadenas como `999 999 999` son claramente un teléfono pero también puede reconocerse como fecha, que en ningún caso sería válida, tomemos la combinación de espacios que tomemos. Por otro lado tenemos que tener en cuenta que los días de la semana deben reconocerse antes que los nombres también.

3. El programa

Puede encontrar el programa en el archivo `formateador.l`, el cual incluye archivos `.h`, cada uno por cada tipo de cadena que aceptamos en el programa.

Se encuentra disponible un archivo `makefile` para compilar y probar el programa, con las opciones:

- `make`, para generar el archivo ejecutable `formateador` (compila con `c++20`, ya que hemos usado una funcionalidad especial para las fechas).
- `make clean`, para eliminar el archivo ejecutable generado.
- `make test`, que ejecutará el archivo ejecutable (que ha de ser previamente generado) con todas las cadenas de prueba que se encuentran en el archivo `prueba.txt`.

  Recomendamos por una parte abrir el archivo y por otra ejecutar `make test`, de forma que por cada cadena de prueba se vea si ha sido aceptada por el programa o si no.