Ejercicios Adicionales de "Fundamentos de Informática"

(Curso 2000/2001)

POR FAVOR, **LEA ESTA PÁGINA ENTERA**, ANTES DE EMPEZAR A PROGRAMAR EL EJERCICIO: Cada alumno podrá entregar voluntariamente uno de los siguientes ejercicios. El ejercicio a realizar será calculado del siguiente modo, donde % es el operador módulo (resto de la división, como en **lenguaje C**):

```
(DNI % 14) + 1
```

REQUISITOS FUNDAMENTALES que deben cumplir TODOS los programas:

- Los programas deben incorporar un **menú** de opciones, leer la opción que se desee ejecutar y ejecutarla. Las opciones mínimas son las que se expresan en cada ejercicio y es absolutamente imprescindible que funcionen bien todas ellas. Se valorará muy positivamente incluir más opciones relacionadas con el mismo ejercicio.

- Como mínimo, cada una de las opciones del programa debe estar implementada en una **función** independiente (salvo excepciones justificadas) **sin usar variables globales**. También existirá una función llamada `Menu`, que será encargada de imprimir en pantalla el correspondiente Menú de opciones. Una opción será, obligatoriamente, "SALIR". Además, se valorará una buena modularización del programa usando adecuadamente la filosofía de la Programación Estructurada (eso de "*Divide y Vencerás*").

- Se debe hacer una gestión de todos los posibles **errores** que pueda cometer el usuario del programa. Ejs: Si se pide un número entre 1 y 12 y se introduce un 16 o un dato de otro tipo, no dividir por cero...

- Es **OBLIGATORIO** incluir un **comentario** al principio de cada fichero y de cada función, indicando claramente su cometido y la utilidad de cada uno de sus parámetros formales. Además, al principio de cada fichero se debe incluir, como comentario, el nombre de su autor, Especialidad y Grupo (de teoría y de prácticas). Se valorará que incluya otros comentarios donde sean precisos.

- Es **OBLIGATORIO** un buen **sangrado** (**indentación**) de todo el programa. Un programa mal sangrado estará suspenso, aunque funcione correctamente.

Cada práctica debe ser defendida en el aula de ordenadores por su autor delante del profesor de prácticas y estará formada por un DISCO y la DOCUMENTACIÓN:

- **DISCO**: Disquette de 3.5" **libre de virus**, con el fichero fuente `.C`, y el ejecutable `.EXE` correspondiente, así como otros ficheros que sean necesarios para la compilación y ejecución del programa. El disco será utilizado para probar que el programa funciona.
- **DOCUMENTACIÓN** (preferentemente a doble cara): Listado de todos los ficheros fuente del programa y bibliografía utilizada. Debe etiquetarse con los siguientes datos: Nombre y Apellidos del alumno, DNI, Número de Práctica realizada, Especialidad y Grupo de Mañana (A o C si es de Mecánica) o Tarde y grupo de prácticas (horario y profesor). La documentación será entregada al profesor, el día de la defensa de la práctica. **NO SERÁ EVALUADA** la práctica **si** el alumno **NO ENTREGA** LA DOCUMENTACIÓN.

NOTA 1: No será válida una práctica que no corresponda al DNI del alumno, que esté incompleta, entregada fuera de plazo, que incumpla alguno de los requisitos expresados en este documento o con virus en el disco.

NOTA 2: Se supone que los alumnos que vayan a realizar este ejercicio práctico han resuelto todos o casi todos los ejercicios propuestos en cada tema. Estos ejercicios tienen la complejidad de los ejercicios del Tema 6, pero son más largos y completos, por lo que si no se han resuelto los ejercicios del Tema 6, será difícil su realización.

Fecha de Defensa de las prácticas: Cada alumno tendrá que defender su práctica ante el profesor el **Lunes 4 de Junio por la mañana** en el aula de prácticas: La hora exacta para cada alumno será dicha por los profesores o será publicada la semana anterior, en la puerta del despacho de los profesores (I-326).

1. **Calendario Perpetuo:** El ejercicio 16 del tema 5 tiene por objetivo averiguar el día de la semana del 1 de Enero de cualquier año posterior a 1582. Basándose en ese ejercicio hacer un programa que muestre en pantalla el calendario de un mes y año elegido por el usuario del programa. El programa mostrará el calendario del mes elegido de forma tradicional, es decir, en 7 columnas, una para cada día de la semana y teniendo en cuenta el número de días de cada mes, prestando especial cuidado con el mes de Febrero por si es año bisiesto.

El programa deberá remarcar de alguna forma los días festivos: Todos Domingos y algunas fiestas especiales 28 de Febrero (día de Andalucía), 1 de Mayo (día del trabajo), 12 de Octubre (día de la Hispanidad, Virgen del Pilar), 6 y 8 de Diciembre (día de la Constitución y de la Inmaculada), 25 de Diciembre (Navidad)... y también el Jueves y el Viernes Santo. La fecha de estos últimos dos días se calculará según indica el ejercicio 20 del tema 4.

La mejor forma de remarcar los días festivos es utilizando la función `setcolor()` para cambiar el color de la biblioteca `graphics.h`. Por ejemplo, para establecer el color a rojo debe usarse `setcolor(RED)`. La mejor forma de aprender a usar esta función es buscar esta función en la ayuda del compilador y examinar y probar el ejemplo que incluye. También puede probarse a cambiar el tipo y tamaño de letra con la función `settextstyle()`.

2. **Máximo Multiplicador Cabalístico:** Hay números naturales que al multiplicarlos sucesivamente por 1, 2, 3, 4... se obtienen números que tienen los mismos dígitos que el original pero en distinto orden (sólo al multiplicar por 1 se obtiene un número con los mismos dígitos en el mismo orden). Supongamos que para el número N, se cumple esa propiedad al multiplicarlo por 1, 2, 3..., X. Es decir, si yo multiplico N por cualquier número del intervalo [1,X], el resultado será un número con los mismos dígitos que N, pero en distinto orden. Entonces, decimos que el "**máximo multiplicador cabalístico**" de N es X: **MMC(N)=X**.

Ejemplo: **MMC(142857) = 6;** → Por ejemplo, 142857*5=714285. Para el número 142857 la propiedad se cumple, además de para el número 5, también para cualquier número del intervalo [1,6].

Hacer un programa que muestre (usando funciones independientes):

- La mayor lista de números posible, en la que todos cumplen que MMC(N) es mayor o igual a 2. Al final debe mostrar el N cuyo valor MMC(N) es el mayor de todos los números analizados.
- Dado un número N, mostrar su valor MMC(N).

3. **Punto de corte de dos rectas:** Dados dos puntos de una recta y otros dos puntos de otra recta, calcular: El punto donde se cortan, si son paraletas o si son coincidentes. El Menú contendrá las opciones "Cambiar puntos de la recta 1: (X1,Y1)-(X2,Y2)" (donde X1, Y1, X2 e Y2 son los últimos puntos introducidos para la recta 1), "Cambiar puntos de la recta 2: (X3,Y3)-(X4,Y4)" (donde X3, Y3, X4 e Y4 son los últimos puntos introducidos para la recta 2), "Calcular relación entre recta 1 y recta 2". Por defecto, al principio del programa se debe suponer un valor cualquiera para los 4 puntos.

Notas: La pendiente de una recta dados dos puntos es: m=(Y2-Y1)/(X2-X1).
La ecuación punto-pendiente de una recta es: y=m(x-X1)+Y1.
OJO con las rectas verticales, tienen pendiente infinita (∞).

4. **Comparación de cadenas en español:** Implementar la función `ComparaCads()` que se comportará de forma similar a la función `strcmp()` de la biblioteca `string.h`, pero para **comparación de cadenas en español**. La función tendrá tres argumentos. Los dos primeros argumentos de la función serán dos cadenas de caracteres. Una cadena es "menor" que otra cuando está antes siguiendo el orden alfabético.

Para este ejercicio no se puede utilizar la biblioteca `string.h`. La función `ComparaCads` debe solucionar dos problemas que tiene la función `strcmp()` para comparar texto en español:

- Debe comparar correctamente cualesquiera palabras en español: Considerando como iguales las vocales si están acentuadas (mayúsculas o minúsculas), la letra "u" con diéresis ("ü" y "Ü") y teniendo en cuenta la letra "ñ" (y "Ñ"), situada tras la letra "n" en el abecedario pero situada en otra posición en la tabla ASCII.
- La función tendrá un tercer argumento que será de tipo entero: Si su valor es 1 (`TRUE`) la comparación se efectuará teniendo en cuenta que las mayúsculas y las minúsculas son iguales, o sea, al comparar "hola" y "HOLA" devolverá 0. Si el valor de este último argumento es (`FALSE`), no considerará que las mayúsculas y las minúsculas son iguales.

5. Dividir calculando periodos: Implementar un programa para **dividir** dos números de tipo `long double`. El resultado debe tener hasta 10 decimales, redondeando este último según el undécimo decimal. El programa debe detectar e indicar si entre los 10 primeros decimales se produce algún **periodo** (puro o mixto). El Menú contendrá las opciones "Cambiar dividendo: X" (donde X es el último dividendo introducido), "Cambiar divisor: Y" (donde Y es el último divisor introducido) y "Calcular división". Por defecto, al principio del programa se debe suponer un valor cualquiera para el dividendo y el divisor.

Ejemplos: 12/9 = 1.3 con 3 como periodo (1.3333333333...).
12/7 = 1.714285 con 714285 como periodo (1. 714285714285714285...).
13/6 = 2.16 con 6 como periodo (2.166666666...).

Compruebe que el divisor no sea nunca cero. Para efectuar diversas divisiones cambiando sólo el divisor bastará con usar la segunda y tercera opciones sucesivamente, ya que el dividendo permanecerá constante mientras no se cambie usando la primera opción. Para calcular los periodos se pueden calcular los restos obtenidos al sacar decimales en la división y, en cada resto comprobar si ese resto ya se ha repetido anteriormente, en cuyo caso tenemos un periodo a partir de ese dígito.

6. Juego del ahorcado: Hacer un programa que permita jugar al **juego del ahorcado** entre dos personas. Un jugador escribe una palabra (sin que el otro la vea) y el otro jugador trata de adivinarla introduciendo letras individuales. El programa tendrá dos cadenas de caracteres, una con la palabra introducida por el primer jugador y otra cadena en la que se introducirán tantos guiones `'-'` como letras tenga la palabra a adivinar.

Tras leer la palabra, se borrará la pantalla (función `clrscr()`), se creará y se mostrará en pantalla la cadena con los guiones. Entonces el ordenador leerá una letra introducida por el segundo jugador y comprobará si dicha letra existe en la palabra original:

- SI EXISTE: Pondrá dicha letra en su posición correspondiente en la cadena de guiones y escribirá el resultado. Se deberá comprobar:
  - Si existen más letras que coincidan: Se deben poner TODAS las coincidencias para cada letra leída.
  - Si ya no existen guiones en la segunda cadena, entonces indicar que se ha acertado la palabra, mostrar el número de fallos cometidos y terminar.
- SI NO EXISTE: Contabilizar un fallo y volver a pedir otra letra.

El programa debe terminar si se acierta la palabra so si se cometen un cierto número de fallos que debe visualizar previamente. El programa permitirá "rendirse" por parte del segundo jugador, es decir, que permita terminar el programa en cualquier momento.

Consejo: Programar una función que acepte 3 argumentos: La cadena original, la cadena con guiones y la letra introducida. La función modificará la cadena con guiones insertando la letra en las posiciones que le

correspondan. La función devolverá 1 si ha habido alguna modificación en la cadena de guiones y 0 si no ha habido ninguna modificación.

7. Implementar un programa para **gestión de un almacén**. Supondremos que el almacén gestionará un máximo de 50 productos. Por supuesto, el programa permitirá trabajar con menos de 50 productos. Para cada producto se almacenará la siguiente información: Código, Nombre, Peso y Volumen. Para ello se utilizará un array de estructuras (o registros).

El programa permitirá las siguientes operaciones, en un menú de opciones:

- Añadir nuevo producto (si es posible).
- Borrar producto ya existente (dando su posición en la lista y/o su nombre, como se prefiera).
- Cambiar algún dato de algún producto: Modificar su peso...
- Mostrar todos los datos de todos los productos.
- Ordenar la lista de productos según su nombre (utilizando la función `strcmp()`).
- Ordenar la lista de productos según sus pesos.

8. Implementar un programa para **gestión de una clase**. Supondremos que el profesor almacenará un máximo de 50 alumnos. Por supuesto, el programa permitirá trabajar con menos de 50 alumnos. Para cada alumno se almacenará la siguiente información: Nombre, Apellidos, DNI, Nota en número, y Nota en letra. Para ello se utilizará un array de estructuras (o registros).

El programa permitirá las siguientes operaciones, en un menú de opciones:

- Añadir nuevo alumno (si es posible).
- Borrar alumno ya existente (dando su posición en la lista y/o su nombre, como se prefiera).
- Cambiar algún dato de algún alumno: Modificar sus notas...
- Mostrar todos los datos de todos los alumnos.
- Cambiar la nota en letra de cada alumno según su nota numérica: No presentado (<0), Suspenso (<5), Aprobado (<7), Notable (<9), Sobresaliente (<10) y Matrícula de Honor (10).
- Calcular la nota media y la varianza de todos los alumnos presentados a examen de la lista, teniendo en cuenta que si la nota es negativa el alumno no se ha presentado y no será tenido en cuenta en los cálculos.
- Mostrar el total de alumnos de cada nota (Suspenso, Aprobado...) y el porcentaje que suponen respecto al total de alumnos en la lista y respecto al total de alumnos presentados (excluyendo los No Presentados).

9. En este problema se recreará la carrera clásica entre **la liebre y la tortuga**. Utilizará una generación de números aleatorios para desarrollar una simulación de la carrera. Los dos contendientes empiezan la carrera en el "cuadro 1" de 70 cuadros. Cada cuadro representa una posición posible a lo largo de la carrera. La línea de meta está en el cuadro 70. El primer contendiente que llegue o que pase el cuadro 70 gana. El desarrollo de la pista trascurre sobre la ladera de una montaña resbaladiza, por lo que ocasionalmente los corredores pierden terreno. El programa deberá ajustar la posición de los animales de acuerdo con las reglas dadas en la tabla siguiente:

Animal	Tipo de Movimiento	Porcentaje de Tiempo	Movimiento Real
Tortuga	Paso rápido	50%	3 cuadros a la dcha
	Resbalón	20%	6 cuadros a la izda
	Paso lento	30%	1 cuadro a la dcha
Liebre	Dormido	20%	Ningún movimiento
	Salto grande	20%	9 cuadros a la dcha
	Resbalón grande	10%	12 cuadros a la izda

	Salto pequeño	30%	1 cuadro a la dcha
	Resbalón pequeño	20%	2 cuadros a la izda

Cada vez que se haga un movimiento de los dos animales se deberá esperar un cierto tiempo antes del siguiente movimiento (por ejemplo esperar 1 segundo usando la función delay()) para que se pueda ver la carrera. Utilice variables para llevar control de las posiciones de los animales (es decir, los números de posición son del 1 al 70). Inicie cada animal en la posición 1. Si un animal resbala antes del cuadro 1, el animal volverá al cuadro 1.

Genere los porcentajes de la tabla anterior mediante un número entero al azar, i, en el rango 1<=i<=10. Por ejemplo, para la tortuga, ejecute un "paso rápido" cuando 1<=i<=5, un "resbalón" cuando 6<=i<=7, o un "paso lento" cuando 8<=i<=10. Utilice una técnica similar para mover a la liebre. Para generar un número aleatorio puede usar las funciones randomize() (inicializa el generador de números aleatorios) y random(num) (devuelve un número aleatorio entre 0 y num-1), ambas de la librería stdlib.h. Consulte la ayuda del compilador para obtener más información. En la web de la asignatura, en los ejercicios resueltos del tema 5, tiene un ejemplo de generación y uso de números aleatorios

Empiece la carrera imprimiendo BANG!!! Entonces, para cada movimiento de ambos animales, imprima una línea de 70 posiciones que muestre la letra T en la posición de la tortuga y la letra L en la posición de la liebre. Ocasionalmente ambos contendientes pueden coincidir en el mismo cuadro. En esta condición, la tortura muerde a la libre y su programa deberá imprimir OUCH!!! empezando en esta posición. Todas las posiciones de impresión, distintas a la de T o la L deberán estar vacías. Cada vez que se imprime la línea hay que comprobar si alguno o ambos ha llegado o ha pasado el cuadro 70. Si es así, se imprime el ganador y se dá por terminada la simulación. También se debe indicar si ambos han llegado a la vez.

El programa deberá contar, como mínimo con las siguientes funciones: MoverLiebre(), MoverTortuga(), PintarLinea(). Se valorará que los gráficos de la liebre, la tortuga y el recorrido sean realistas.

10. Quizás el más famoso de todos los sistemas de codificación es el **código Morse**, desarrollado por Samuel Morse en 1832, para uso en el sistema telegráfico. El código Morse asigna una serie de puntos y rayas a cada letra del alfabeto, a cada dígito y a unos cuantos caracteres especiales. La separación entre palabras se indica por un espacio o por la ausencia de un punto o una raya. La versión internacional del código Morse aparece en la tabla siguiente:

Carácter	Código	Carácter	Código
A	.-	T	-
B	-...	U	..-
C	-.-.	V	...-
D	-..	W	.--
E	.	X	-..-
F	..-.	Y	-.--
G	--.	Z	--..
H	....		
I	..	Números	
J	.---	1	.----
K	-.-	2	..---
L	.-..	3	...--
M	--	4	....-
N	-.	5	.....
O	---	6	-....
P	.--.	7	--...
Q	--.-	8	---..
R	.-.	9	----.
S	...	0	-----

Escriba un programa que lea una frase escrita en español y cifre dicha frase en código Morse y que también lea una frase en código Morse y la convierta en el equivalente en español. Utilice un espacio en blanco entre cada letra codificada Morse y tres espacios en blanco entre cada palabra codificada en Morse.

El programa deberá incorporar una función Menu() que muestre las siguientes opciones: 1) Pasar una frase a código Morse, lo cual se implementará en una función que se llame Frase2Morse(), 2) Pasar código Morse a una frase, implementando una función que se llame Morse2Frase() y 3) Salir.

11. Análisis de texto. La disponibilidad de computadoras con capacidades de manipulación de cadenas nos proporciona interesantes métodos para analizar lo escrito por grandes autores. Se ha puesto, por ejemplo, gran atención al hecho de saber si William Shakespeare alguna vez existió. Algunos estudiosos creen que existen evidencias indicando que Christopher Marlowe fue el que escribió las obras maestras atribuidas a Shakespeare. Los investigadores han utilizado ordenadores para localizar similitudes en los textos de estos dos autores. Realice un programa que lea varias líneas de texto y analice las siguientes características del texto:

a) Imprimir una tabla indicando el número de veces que aparece cada letra del alfabeto en dicho texto.

b) Imprimir una tabla que indique el número de palabras de una letra, de dos letras, de tres letras ... que aparecen en el texto.

c) Imprimir una tabla indicando el número de ocurrencias de cada palabra distinta en el texto. Para ello supondremos que el texto tiene como máximo 100 palabras distintas, con lo que deberá almacenarlas en un array de estructuras de tamaño 100. Cada estructura deberá contener una cadena de caracteres con la palabra (máximo 20 caracteres) y otro campo con el número de veces que aparece esa palabra en el texto. Considere opcionalmente la posibilidad de que las palabras aparezcan ordenadas alfabéticamente. Para ordenarlas puede utilizar cualquier algoritmo de ordenación teniendo en cuenta que hay que intercambiar estructuras completas (la cadena y el número de ocurrencias). Para comparar las cadenas de caracteres en la ordenación utilice la función strcmp() que dice si dos cadenas son iguales, si una es mayor que la otra o viceversa.

El programa deberá mostrar un menú con las siguientes opciones: 1) Introducir texto, 2) Número de instancias de cada letra, 3) Número de palabras de cada longitud, 4) Número de ocurrencias de cada palabra distinta, 5) Salir.

12. Hacer un programa para evaluar la calidad de las comidas en **dos comedores universitarios**. Para ello se ha pedido a un conjunto de usuarios de dichos comedores que puntuen de 1 a 5 (muy malo, malo, regular, bueno, muy bueno) cada comedor. La encuesta se hace de forma anónima, pero cada usuario tiene que decir si es alumno (A) o no (O). Esta información la guarda el programa junto con las dos puntuaciones. Sobre estos datos se deben obtener las siguientes estadísticas:

a) La puntuación media obtenida por cada uno de los comedores.
b) Calcular y visualizar el histograma. Este se calcula contando el número de veces que se obtiene cada puntuación y guardándolo en un array.
c) La moda de cada uno de los comedores. La moda es el valor más frecuente y se puede calcular fácilmente utilizando el histograma.
d) La mediana para cada uno de los comedores. Este valor es el que divide a los demás en dos partes iguales. Se puede calcular ordenando los valores de menor a mayor y la mediana será aquel valor que queda en medio de los demás.
e) La varianza de las puntuaciones dadas para cada uno de los dos comedores.
f) La covarianza de las puntuaciones para los dos comedores en común.

El programa deberá mostrar un menú con las siguientes opciones: 1) Introducir valores, 2) Calcular la media, 3) Calcular la moda, 4) Calcular la mediana, 5) Visualizar el histograma, 6) Calcular la varianza, 7) Calcular la covarianza, 8) Salir.

Cada una de las opciones de la 2 a la 6 deberán mostrar los valores para los dos conjuntos de datos, es decir, para cada comedor independientemente, mientras que la opción 6 se calcula para los dos conjuntos de datos en común. En cualquiera de los casos se deben mostrar los resultados calculados sobre todos los encuestados, sólo sobre los encuentados que son alumnos y sólo sobre los encuentados que no son alumnos. Por ejemplo, si se da la opción 2 el programa mostrará la media calculada para todos los encuentados, para los alumnos y para los no alumnos para el primer comedor y lo mismo para el segundo comedor, es decir, 6 valores en total.

Nota: Para introducir los valores se puede utilizar un proceso aleatorio. El generador de números aleatorios se inicializa al principio del programa con la función randomize(). La función random(num) devuelve un número aleatorio entre 0 y num-1 (consultar la ayuda del compilador). Para introducir si el encuestado es alumno o no se supone que el 75% de los encuestados son alumnos y el resto no lo son. En este caso se genera un número aleatorio entre 1 y 100, si el número es menor que 75, el encuestado es alumno y si es mayor que 75 no lo es. Para introducir los valores de las puntuaciones el número se genera entre 1 y 5 y el valor obtenido se pone en la puntuación. Así se pueden almacenar inicialmente las encuestas que se deseen (50 por ejemplo) sin tener que introducirlas por teclado cada vez que se utiliza el programa. En la web de la asignatura, en los ejercicios resueltos del tema 5, tiene un ejemplo de generación y uso de números aleatorios.

13. En la Secretaría de una Universidad hay un programa para que los alumnos puedan introducir sus preferencias a la hora de cursar **asignaturas optativas**. Después se impartirán aquellas asignaturas en las que el número de solicitudes sea mayor que un tope establecido (suponer que es 3). Cada alumno cursa entre 1 y 3 asignaturas optativas y registra: Nombre, Curso, lista de asignaturas que prefiere (como máximo 3).

Las asignaturas serán opt1, opt2, opt3, opt4, opt5 y opt6. Suponer que cada asignatura tiene un código [1-6]. El alumno introducirá el código de la asignatura.

a) Hacer una función que permita insertar los datos para un alumno.
b) Otra función que para una asignatura determinada indique el número de alumnos que la han solicitado.
c) Otra función que para una asignatura indique si se va a impartir o no.
d) En el programa principal utilizar las funciones anteriores para solicitar las preferencias de un conjunto de alumnos (se termina la lectura de alumnos cuando se introduzca un nombre vacío) y mostrar como resultado las asignaturas que se impartirán y el número de alumnos de cada asignatura.

14. Supongamos que queremos obtener algunos **datos de tres empresas que tienen cinco sucursales** cada una. De cada empresa guardaremos su nombre, y de cada sucursal el nombre, el número de sucursal ([1..5]), sus ingresos y sus gastos. Por ejemplo, para una empresa (de las tres que vamos a gestionar):

ZARA:	1 Armengual de la Mota	1550000	700000
	2 Centro	2850000	934000
	3		
	4		
	5		

Realizar un programa que tenga las siguientes opciones (usando funciones independientes):

a) Introducir los datos de las distintas empresas.
b) Presentar la sucursal (con todos sus datos incluyendo la empresa a la que pertenece) que tenga el beneficio más alto.
c) Con los datos de las empresas obtenidos mediante el procedimiento a), presentar estas empresas en orden descendente por beneficio total (suma de beneficios de todas sus sucursales). Se visualizarán todos los datos de cada empresa.