Universidad de Alcalá

Departamento de Automática

INFORMÁTICA

Práctica 4

Programación básica en C

Grado en Ingeniería en Electrónica y Automática Industrial

Curso 2023-2024

V3.2 (08.09.2023)

A continuación figuran una serie de ejercicios propuestos, agrupados por bloques de dificultad creciente. La práctica consiste en hacer la mayor cantidad posible de ellos, teniendo en cuenta que los que se propongan para el examen de laboratorio habrán sido extraídos de entre ellos con muy pequeñas variantes. En todos los casos se pide cumplir las especificaciones completamente, aunque para aquellos detalles del ejercicio que no se especifiquen en el enunciado se admite la decisión justificada del alumno.

Salvo que se indique algo distinto en el propio enunciado, se debe utilizar la entrada estándar para obtener los datos y la salida estándar para mostrar los resultados.

En varios de los enunciados que siguen encontramos que el programa pedido dispone de un espacio máximo fijado (es decir, el programador lo fija en el momento de crear el programa) para almacenar algún dato (por ejemplo, para una lista, para una matriz, etc.). En todos esos casos, el programa debe asegurarse de que los datos introducidos por el usuario al ejecutar el programa no superan el espacio disponible.

Bloque 1

1. Crear un programa que imprima una tabla de números consecutivos junto a su cuadrado respectivo. El programa pedirá al usuario el valor del primer número a partir del cual comenzará la impresión; y pedirá también cuántos números ha de imprimir.
2. Calcular el promedio de los elementos de una lista. El programa debe pedir al usuario el número total de elementos de la lista y, después de comprobar que no supera el máximo, le pedirá dichos elementos. Finalmente, emitirá el promedio.
3. Clasificar los números de una lista en pares e impares. El resultado del programa será dos listas: una con los pares y otra con los impares. Las listas tendrán un tamaño máximo fijado por el programa.
4. Crear un programa que obtenga del usuario una cadena de caracteres (con un tamaño máximo fijado por el programa) e imprima por pantalla la misma cadena, pero al revés, es decir, empezando desde el último carácter y acabando con el primero.
5. Escribir la sucesión de Fibonacci. El programa debe leer dos números enteros de teclado: el primer dato leído será el número de elementos de la secuencia a mostrar; el segundo, el elemento de la sucesión a partir del que hay que mostrar, es decir, a partir de ese elemento, el programa deberá mostrar tantos elementos de la sucesión como indique el primer número leído. La sucesión de Fibonacci es n_0, n_1, ..., n_i, ..., y se define como sigue:
$$\begin{aligned} n_0 &= 0; \\ n_1 &= 1; \\ n_i &= n_{i-1} + n_{i-2}. \end{aligned}$$
6. Crear un programa que obtenga del usuario una cadena de texto (de longitud máxima fijada) consistente únicamente en letras y espacios, y muestre por pantalla el resultado de dejar, entre cada dos palabras, un único espacio. Se entiende que solo los espacios separan palabras.
7. Crear un programa que obtenga del usuario una cadena de caracteres (con un tamaño máximo fijado por el programa) y un carácter, y muestre por pantalla la misma cadena eliminando el carácter indicado.
8. Crear un programa que obtenga del usuario una cadena de caracteres (con un tamaño máximo fijado por el programa) e imprima por pantalla las letras mayúsculas que encuentre en esa cadena.
9. Escribir un programa que calcule la suma y la resta de dos cantidades de dinero expresadas en el sistema inglés de la época victoriana: peniques, chelines y libras. Tener en cuenta que 1 chelín = 12 peniques, y 1 libra = 20 chelines.

Ejercicios de ampliación para el Bloque 1

10. Generar una tabla de equivalencias entre kilómetros y millas náuticas. El programa preguntará el intervalo (por ejemplo, desde 10 a 100 kilómetros) y el paso (por ejemplo, cada 10 kilómetros) y creará un tabla en la que cada línea dará la distancia en millas para cada distancia en kilómetros. Tómese que una milla náutica son 1.852 metros.

11. Calcular el producto escalar de dos vectores. Los vectores tendrán un número de elementos especificado por el usuario (con un máximo fijado por el programa) y obtendrá los elementos de los vectores del usuario.
12. Crear un programa que obtenga del usuario una cadena de caracteres (con un tamaño máximo fijado por el programa) y dos caracteres, y muestre por pantalla la misma cadena pero agregando el segundo carácter solicitado a continuación del primero allí donde este aparezca.
13. Realizar una calculadora que sea capaz de sumar y restar horas expresadas en el formato HH:MM:SS. El programa pedirá al usuario las dos horas y devolverá la suma y la diferencia (restando siempre el menor del mayor) de tales horas expresadas en el mismo formato. Por ejemplo:

```
Introducir primera hora:   10:29:12
Introducir segunda hora:    8:42:25
Suma de las horas:         19:11:37
Diferencia de las horas:    1:46:47
```

14. Realizar un programa que lea una cadena de caracteres (con un tamaño máximo fijado por el programa) y muestre por pantalla el resultado de cambiar las letras minúsculas en mayúsculas y viceversa. Si hay otros caracteres que no sean letras, deben salir intactos.
15. Realizar un programa que pida al usuario dos cadenas de caracteres (con tamaños máximos fijados por el programa) y las imprima por pantalla 'mezcladas', es decir, sacando alternativamente un carácter de cada una de ellas. Los caracteres que le sobren a la más larga se imprimirán al final. Por ejemplo, supongamos que las cadenas introducidas por el usuario han sido: `elefante` y `GATO`. El programa debería sacar por pantalla la siguiente cadena:
    `eGlAeTfOante`
16. Comprobar si una cadena de caracteres es un palíndromo, es decir, si se puede leer igual de izquierda a derecha que de derecha a izquierda. Al hacer la comprobación deben ignorarse los espacios.
17. Crear un programa que obtenga del usuario dos vectores de enteros de igual longitud (pero con una longitud máxima fijada por el programa) y calcule la suma de ambos, rotando el primero un número de posiciones hacia la derecha que indique el usuario. Cada rotación implica que la componente de índice i pasa a ocupar índice $i + 1$; y que la componente de índice máximo pasa al índice mínimo.
18. El programa obtendrá del usuario una lista de números de longitud arbitraria (pero con una longitud máxima fijada por el programa) y, a continuación, de cada grupo de tres números, intercambiará las posiciones del primero y el tercero, mostrando el resultado por pantalla. Si el último grupo no estuviera completo, lo completará con ceros.
19. Crear un programa que lea de teclado una lista de números. Para ello, primero indicará al usuario el tamaño máximo de la lista y averiguará de él la cantidad de números que el usuario quiere introducir. A continuación, leerá la lista. Una vez leída, el programa mostrará por pantalla otra lista donde el elemento en la posición i es el promedio de los elementos desde la primera posición hasta la i-ésima en la lista original.
20. Sumar un determinado número de elementos de una progresión aritmética. El primer elemento de la progresión, la razón y el número de elementos que hay que sumar los especificará el usuario. Cada elemento de una progresión

aritmética se construye a partir del anterior sumándole la razón, es decir, tiene la forma:

$$a_{i+1} = a_i + r,$$

donde a_1 es el primer elemento y r es la razón.

21. Crear un programa que obtenga del usuario dos números enteros positivos e indique si son primos entre sí.
    Nota: dos números son primos entre sí si no comparten ningún divisor, es decir, si su máximo común divisor es 1. Por ejemplo, 12 y 5 lo son, mientras que 10 y 6 no lo son.
22. Euclides se dio cuenta de que el máximo común divisor de dos números a y b (suponiendo que $a > b$) es el mismo que el de b y r, donde r es el resto de dividir a entre b. Si ahora consideramos que b y r hacen el papel de a y b podemos reiterar las divisiones cuantas veces queramos, hasta que el resto r termina por ser 0, momento en el cual el valor de b es precisamente el máximo común divisor. Usar esa propiedad para hacer un programa que obtenga el máximo común divisor de dos números (positivos) cualesquiera.
23. Realizar el sumatorio de $1/n^2$ desde $n = 1$ hasta el máximo que el usuario especifique por teclado. El programa pedirá al usuario el valor máximo de n y dará como salida la suma total de los valores $1/1^2 + 1/2^2 + 1/3^2 + ... + 1/n^2$.
    Nota: Las matemáticas nos dicen que, si n es suficientemente alto, ese valor se aproxima a $\pi^2/6$ ($\approx 1{,}64493$). Pero, ¡atención!, la división ha de ir en coma flotante para que todo vaya bien.
24. Crear un programa que sume los elementos de la sucesión de Padovan entre dos posiciones indicadas por el usuario (ambas inclusive). La sucesión de Padovan $p_0, p_1, p_2, ...$ se define como:

$$p_0 = p_1 = p_2 = 1;$$
$$p_n = p_{n-2} + p_{n-3}.$$

Bloque 2

1. Evaluar un polinomio de grado arbitrario en un punto x_0 especificado por el usuario. Los coeficientes y el grado del polinomio también serán especificados por el usuario.
2. Calcular la suma de dos matrices. Las matrices tendrán unas dimensiones máximas fijadas por el programa. El programa solicitará al usuario las dimensiones de las matrices, comprobará que no superan los límites fijados y mostrará la matriz resultado fila por fila.
3. Extraer la diagonal principal de una matriz. Se mostrará la matriz original, fila por fila y, después, la diagonal, en una única lista. Las dimensiones máximas de la matriz se fijarán por el programa. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.
4. Tomando como entrada una cadena de texto (de longitud máxima fijada), eliminar las consonantes y mostrar ambas cadenas: la original y la cadena con las consonantes eliminadas.
5. Crear un programa que obtenga del usuario una cadena de caracteres (de longitud máxima fijada) y sustituya en ella la vocal más frecuente en ella por la menos frecuente en ella (si hay varias que son la menos frecuente, se elegirá una de ellas de forma arbitraria).
6. Cuando una matriz es grande pero tiene muchos elementos de valor 0, se puede almacenar usando el *formato comprimido*. Este consiste en almacenar en una única lista únicamente aquellos elementos de la matriz distintos de cero, conjuntamente con sus índices de fila y columna. Ejemplo (recordar que en C, los índices empiezan en 0):

   Matriz original:

   $$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

   La misma, pero en formato comprimido:

   (1,0,1), (4,1,3), (2,3,0)

   Se pide convertir una matriz dispersa a formato comprimido. El programa pedirá al usuario las dimensiones de la matriz y sus elementos. A continuación, mostrará la matriz original se mostrará fila por fila y después el formato comprimido, de manera similar al ejemplo. Las dimensiones máximas de la matriz y de la lista de tripletas se fijarán por programa. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.
7. Dados una matriz y un vector, calcular el vector que se obtiene al multiplicar la matriz por el vector. Mostrar únicamente el vector resultado. Las dimensiones máximas de la matriz y del vector se fijarán por programa. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.
8. Mostrar una lista con los números primos que se encuentran entre dos números dados.
9. Hacer un histograma de las vocales de un texto, es decir, indicar cuántas veces aparece cada vocal en ese texto. Se leerá una cadena de caracteres (de tamaño

máximo fijado) y se mostrará el número de veces que aparece cada vocal en él. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.

10. Contar el número de palabras de un texto. Se leerá una cadena de caracteres (de tamaño máximo fijado) y se mostrará cuántas palabras tiene. El programa ha de comprobar, en primer lugar, que el texto contiene únicamente letras, espacios en blanco y signos de puntuación (se consideran signos de puntuación válidos la coma, el punto y coma, el punto y los dos puntos) y, si no es así, debe terminar de inmediato dando un error. Si la cadena es correcta, debe devolver el número de palabras de que consta, teniendo en cuenta que cualquier espacio en blanco o signo de puntuación separa palabras y que estos separadores podrían estar repetidos.
11. Llevar un histograma de resultados del evento "tirar un dado". Un histograma es una lista que cuenta, para cada elemento, el número de veces que ha salido ese resultado. Por ejemplo, la lista (el histograma): `5,3,7,2,6,1` indicaría que el `1` ha salido cinco veces, el `2` tres veces, el `3` siete veces, y así sucesivamente. Para ello, utilizar la función `random` y asociadas, presente en la biblioteca `stdlib.h` (Ver nota (*) al final del cuaderno).
12. Repetir el ejercicio anterior, pero simulando la tirada de dos dados (tener en cuenta que ahora los posibles resultados son los números comprendidos entre 2 y 12).
13. Dibujar en pantalla una pirámide construida con el carácter '*'. El programa leerá, en primer lugar, los pisos que tiene la pirámide, y la mostrará centrada sobre su base. Ejemplo:

```
4 pisos:
   *
  ***
 *****
*******
```

14. Visualizar un entero en binario codificado en ASCII (Ver nota (**) al final del cuaderno). Se visualizarán tanto el entero leído como el resultado de su conversión.
    Nota: la función `printf` no dispone de esa conversión.

Ejercicios de ampliación para el Bloque 2

15. Extraer la diagonal secundaria de una matriz. Se mostrará la matriz original, fila por fila y, después, la diagonal secundaria, en una única lista. Las dimensiones máximas de la matriz se fijarán por el programa. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.
16. Imprimir el primer carácter de cada palabra que exista en una cadena de caracteres de tamaño máximo fijado. El programa ha de comprobar, en primer lugar, que el texto contiene únicamente letras, espacios en blanco y signos de puntuación (se consideran signos de puntuación válidos la coma, el punto y coma, el punto y los dos puntos) y, si no es así, debe terminar de inmediato dando un error. Si la cadena es correcta, el programa imprimirá en la línea siguiente el primer carácter de cada palabra, separándolos entre sí por un espacio.

17. Crear un programa que obtenga del usuario los elementos de una matriz de dimensiones arbitrarias (pero como máximo 20) y no necesariamente cuadrada. A continuación, indicará por pantalla cuál es la columna la suma de cuyos elementos sea mayor, así como el valor de esa suma.
18. Calcular las soluciones de una ecuación de segundo grado. Téngase en cuenta que las soluciones pueden ser valores reales **o complejos**.
19. Crear un programa que lea de teclado una matriz de dimensiones especificadas por el usuario (pero menores que 20). El programa generará (y mostrará por pantalla, fila por fila), a partir de esa matriz, otra con una fila y una columna más. En cada elemento de la nueva columna almacenará el promedio del resto de elementos de su misma fila. En cada elemento de la nueva fila almacenará el promedio del resto de elementos de su misma columna. En el elemento que pertenece tanto a la nueva fila como a la nueva columna almacenará el promedio de todos los elementos de la matriz. Los datos de la matriz serán números en coma flotante.
20. Imprimir la lista de las parejas (sin repetición) de números enteros, comprendidos entre dos límites dados por el usuario, tales que su suma sea igual a cierto número entero dado. La pareja (*a*,*a*) es válida.
21. Crear un programa que obtenga del usuario una matriz cuadrada de dimensión arbitraria (pero como máximo 20) y muestre la matriz resultante de transponerla respecto a su diagonal secundaria.
22. Repetir el ejercicio 13 mostrando únicamente los bordes de la pirámide.
23. Repetir el ejercicio 13 mostrando una pirámide invertida.
24. Repetir el ejercicio 13 mostrando un rombo.
25. Repetir el ejercicio 13 mostrando un diábolo. Ejemplo (para 5 pisos):

```
* * * * *
 * * *
  *
 * * *
* * * * *
```

26. Implementar el método de Newton-Raphson para obtener la raíz cuadrada de un número *a*. Este método consiste en iterar la siguiente fórmula para valores $n = 0, 1, 2, 3, \ldots$:

$$x_{n+1} = \frac{x_n^2 + a}{2x_n},$$

tomando $x_0 = 1$. Cuanto más alto es el valor de *n*, más se acerca el valor de x_n a la raíz cuadrada de *a*. El programa obtendrá la raíz con un error máximo *E* para la raíz. Esto equivale a decir que la iteración se debe detener tan pronto como ocurra que

$$|x_{n+1} - x_n| < E.$$

En ese momento, el valor de x_{n+1} es el resultado del problema.

27. Contar el número de veces que un determinado patrón aparece en una cadena de texto. Se leerá en primer lugar un patrón y, a continuación, la cadena de texto; después se mostrará el número de veces que aparece el patrón en ella (que

podría ser 0). Tanto la cadena de texto como la del patrón tendrán unas longitudes máximas fijadas. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible para cada una de ellas.

28. Crear un programa que obtenga del usuario una cadena de texto de longitud arbitraria (pero menor de 256 caracteres) y un número positivo. A continuación, mostrará la cadena rotada tantas posiciones a la izquierda como indique el número, insertando por la derecha los caracteres que "se salgan" por la izquierda. Repetirá e imprimirá la rotación tantas veces como sean necesarias hasta que vuelva a imprimirse la cadena original.

    Ejemplo:
    ```
    Teclee cadena: Hola que tal
    Rotación: 3

          'Hola que tal'
          'a que talHol'
          'ue talHola q'
          'talHola que '
          'Hola que tal'
    ```

29. El programa obtendrá del usuario una matriz cuadrada de dimensión como máximo 10, y un número positivo. A continuación, mostrará por pantalla el resultado de rotar las columnas de la matriz a la derecha tantas posiciones como indique el número.

    Ejemplo:
    ```
    Matriz:          Número: 1          Matriz rotada:
    1 3 2 5                             5 1 3 2
    2 2 3 0                             0 2 2 3
    7 4 2 1                             1 7 4 2
    ```

30. Crear un programa que obtenga del usuario una matriz de enteros de dimensión arbitraria (pero como máximo de 10 × 10), y a continuación obtenga del usuario un entero positivo. El programa mostrará por pantalla el resultado de eliminar de la matriz la fila y la columna correspondientes al entero que se obtuvo del usuario.

    Ejemplo:
    ```
    Matriz:     Entero: 2  (eliminar fila 2 y columna 2, donde la primera es la 0)
    1 3 2 5
    2 2 3 0
    7 2 4 1
    4 1 4 0

    Matriz resultado:
    1 3 5
    2 2 0
    4 1 0
    ```

31. Realizar un programa que cambie un número de base 10 a otra base elegible por el usuario. Para representar dígitos superiores a 9 se utilizarán las letras del alfabeto: A, B, C, D, E, F, G, H, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y,

Z, desde la A, para el 10, hasta la Z para el 34. Pedirá cuál es la nueva base, después pedirá el número y finalmente imprimirá el resultado. Obsérvese que usando esas letras, el valor máximo de la base es 35.

32. Crear un programa que calcule el intervalo de tiempo entre dos fechas de un mismo año, ambas inclusive. Condiciones:
    - Las fechas se expresan por medio de un par de números, que definen el día y el mes. Por ejemplo, (3,11) sería el 3 de noviembre.
    - El intervalo de tiempo se expresa por medio de un par de números que definen el número de semanas y el número de días, donde éste último no puede ser superior a 6 (ya que eso indicaría una semana más). Por ejemplo: (5,3) sería 5 semanas + 3 días = 38 días.

    Ejemplo:

    ```
    Fecha 1: 16,8
    Fecha 2: 10,9
    Resultado: 3 semanas y 5 días.
    ```

    Explicación: Entre el 16 de agosto y el 10 de septiembre hay 16 días de agosto y 10 de septiembre (se incluyen ambas fechas). En total, 26 días, que hacen 3 semanas y 5 días.
33. Crear un programa que solicite al usuario una matriz de números en coma flotante (de dimensión máxima 10 × 10) y un vector de, como máximo, 10 elementos. El programa buscará si la matriz contiene el vector en alguna de sus columnas y, de ser así, indicará cuál. En caso contrario, indicará que el vector no está contenido en ninguna de las columnas.

Bloque 3

1. Escribir un programa que informe de la proporción de aparición de cada letra del alfabeto con respecto al total de letras que se introduzcan. El programa leerá de teclado un texto multilínea, arbitrariamente largo, cuya entrada finaliza cuando el usuario teclee el carácter fin de fichero, EOF (en Linux, Control-D). Solo se mostrará la proporción de las letras presentes.
2. Realizar la multiplicación de dos matrices. Se leerán dos matrices y se mostrará el resultado. Se debe comprobar que la multiplicación es posible. Las dimensiones máximas de las matrices estarán fijadas por el programa. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.
3. Implementar una calculadora básica (suma, resta, multiplicación y división) que permita trabajar con números flotantes. La selección de la operación a realizar se puede solicitar por menú, incluyendo también una opción para terminar. Para este ejercicio, utilizar exclusivamente el teclado y la pantalla como entrada y salida estándar, respectivamente.
   Nota: El programa debe evitar las divisiones por cero.
4. Calcular el número combinatorio $\binom{m}{n}$, con $m \geq 0$, $n \geq 0$ y $0 \leq n \leq m$, teniendo en cuenta que $\binom{m}{n} = \binom{m-1}{n-1} + \binom{m-1}{n}$ y que $\binom{m}{0} = \binom{m}{m} = 1$. Se sugiere utilizar recursividad.
5. Sustituir un patrón por otro en una cadena. El programa leerá un texto, a continuación el patrón a sustituir, y por último el patrón por el que debe sustituirlo. Finalmente, mostrará la cadena modificada. Si no se ha encontrado el patrón, se mostrará la cadena sin modificar. El programa fijará tamaños máximos para las diversas cadenas utilizadas y deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible en cada caso.

Ejercicios de ampliación para el Bloque 3

6. Generar el bit de paridad de un entero. El bit de paridad es un bit que se añade a un número de forma que el número total de bits a uno del número sea par (en el caso de paridad par) o impar (en el caso de paridad impar). Para este ejercicio, utilizar paridad par.
7. Ordenar una lista de números. El programa obtendrá del usuario una lista de números cuyo tamaño especificará el propio usuario, y, a continuación, la mostrará ordenada en orden creciente. Para ordenar la lista, se puede utilizar alguna variante del *algoritmo de la burbuja* (Ver nota (***) al final del cuaderno).
8. Ordenar una lista de palabras. Se leerán las palabras de la lista, y a continuación se mostrará la lista ordenada alfabéticamente. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.
9. Escribir un filtro: el programa leerá un patrón, y a continuación un texto multilínea. Mostrará sólo aquellas líneas del texto que contengan el patrón.
10. Dado un número entero, calcular el número que se obtiene de invertir el orden de sus bits. El programa leerá el número, y a continuación mostrará los bits del

número original y los del número resultado, ambos en binario codificado en ASCII (Ver nota (**) al final del cuaderno).

11. El programa obtendrá del usuario dos listas de números de longitud arbitraria (pero inferior a 100) y seleccionará el número que, trasladado a la otra lista, hace que las sumas de los elementos de ambas sean lo más parecidas entre sí. Las listas no tendrán necesariamente la misma longitud.

    El programa indicará qué número de qué lista se debe mover (no es necesario mostrar las listas modificadas).

    Nota: Se trata de elegir el elemento de la lista con la suma mayor cuyo doble está más cerca de la diferencia de sumas.

    Ejemplo:

    Lista 1: `2, 5, 1, 7, 3`
    Lista 2: `5, 3, 6, 8, 1`

    La suma de la lista 1 es 18, y la de la lista 2 es 23. La diferencia entre ambas es 5. El número de la lista 2 cuyo doble es más cercano a 5 es el 3. Si se lleva el 3 a la lista 1, la suma para esta lista será de 21, y para la lista 2 será de 20. Así la diferencia entre ambas listas es la menor posible.

12. Crear un programa que resuelva operaciones básicas de números expresadas en notación "polaca inversa". El programa leerá la secuencia de operaciones por teclado en forma de una única cadena de caracteres (de longitud máxima fijada por el programa) y mostrará el resultado por pantalla. En la cadena, operadores y operandos estarán separados por un único espacio en blanco. Se puede suponer que la cadena leída es correcta. La cadena nula es admisible.

    Ejemplo:
    Secuencia de operaciones: 5 2 + 3 *
    Resultado: 21

    Explicación:
    Los dos primeros números (5 y 2) son los operandos de la operación que aparece a continuación (la suma). 5+2=7. El resultado (7) es el primer operando de la siguiente operación (la multiplicación) y 3 es el segundo. 7*3=21.

13. Realizar un programa que lea una frase y, conservando el orden de las palabras, escriba cada una de ellas al revés (entendemos por palabra una secuencia de caracteres distinta de espacio en blanco). Por ejemplo:

    ```
    La casa de la pradera
    ```

    Daría lugar a:

    ```
    aL asac ed al aredarp
    ```

Bloque 4

1. Determinar si una matriz está contenida en otra. El programa obtendrá dos matrices del usuario (con dimensiones especificadas por él) e indicará si una está contenida en la otra.
2. Hacer un "traductor" básico. El programa cargará de un archivo una serie de correspondencias entre palabras. A continuación, para cada palabra leída de teclado mostrará su traducción (si existe; en caso contrario la dejará igual). El proceso se repetirá hasta que se lea una palabra clave predefinida.
3. Resolver una sopa de letras. El programa cargará de un archivo una matriz de caracteres. A continuación, leerá una palabra y la buscará en la matriz en cualquier orientación (vertical, horizontal, diagonal). Si la encuentra, indicará su situación por medio de las coordenadas de su letra de inicio y las de su letra de fin. Si no la encuentra, lo indicará convenientemente.
4. Gestionar una base de datos sencilla. Los elementos de la base de datos tendrán un campo código (entero), un campo nombre (cadena de caracteres), y un campo edad (entero). Las únicas operaciones admitidas son: *insertar nuevo elemento*, *eliminar elemento*, y *mostrar elemento*. *Insertar* crea un nuevo elemento, y solicita los campos del mismo. *Eliminar* solicita un código, y, si está presente en la lista, lo elimina. *Mostrar* solicita un código y muestra la información asociada al elemento correspondiente. Al realizar la inserción, el programa debe asegurarse de que el código asociado al nuevo elemento no exista previamente.
5. Repetir el ejercicio 1 del bloque 3 con matrices en formato disperso.
6. Calcular el determinante de una matriz cuadrada de tamaño arbitrario. Dada una matriz A, cuadrada, de tamaño $n \times n$, se define la submatriz $A_{p,q}$ como la matriz, de tamaño $(n-1) \times (n-1)$, que resulta de eliminar en A la fila p y la columna q. Con esto, la siguiente fórmula recursiva nos permite calcular el determinante de A:

   $$\det(A) = \sum_{j=1}^{n} (-1)^{j+k} \cdot a_{k,j} \cdot \det(A_{k,j}),$$

   en donde $a_{k,j}$ es el elemento de la fila k y columna j en A. Dicho con palabras, el determinante de A se calcula en función de los determinantes de sus submatrices. El valor de la fila k se fija arbitrariamente de entre los valores que cumplen que $1 \leq k \leq n$.

   La recursión se termina cuando una submatriz tiene una sola fila y una sola columna, en cuyo caso tiene un solo elemento, que coincide con su determinante.

NOTAS:

(*) Las funciones `random` y `srandom`, de la librería `stdlib.h`, permiten generar números pseudo-aleatorios. Cada vez que se invoca, `random` devuelve un entero pseudo-aleatorio comprendido entre los valores 0 y $(2^{31} - 1)$. Los números los obtiene de una secuencia generada por medio de un algoritmo que hace que parezcan aleatorios. Sin embargo, esto no es realmente así. Cada vez que se ejecute un programa, la secuencia de números que devuelve la función `random` es la misma. Para evitar esto, se debe usar la función `srandom` al inicio del programa. Esta función permite establecer la "semilla", es decir, la posición de la secuencia por la que se empiezan a generar los números. Recibe un parámetro, que es un número a partir del cual obtiene la posición inicial de la secuencia, de modo que, si cada vez que se ejecuta el programa, la función `srandom` utiliza un número diferente como "semilla", los números que irá generando la función `random` serán diferentes. Una forma de que el número que se utiliza como semilla sea diferente en cada invocación del programa es utilizar la función `time`, de la librería `time.h`, que devuelve un entero con el tiempo en segundos desde las 0 horas, 0 minutos y 0 segundos desde el 1 de enero de 1970. Dado que esta cuenta es diferente cada vez que se ejecute el programa, se puede utilizar este valor como semilla para la función `srandom`.

Evidentemente, para la depuración del programa lo interesante es que la secuencia sea precisamente siempre la misma, por lo que, en este caso, el valor que se usa como semilla deberá ser siempre el mismo.

(**) El binario codificado en ASCII es un código que representa cada bit del dato por un carácter ASCII, bien el '0', bien el '1', dependiendo del valor del bit. De esta manera, un número entero se representa por una cadena de caracteres ASCII. Por ejemplo, el número decimal 5, en binario puro de 8 bits, se representaría así:

```
00000101.
```

Por lo tanto, en binario codificado en ASCII, se utilizaría un código ASCII para cada bit del número:

```
'0' '0' '0' '0' '0' '1' '0' '1' '0',
```

donde `'0'` representa el código ASCII del `0` y `'1'` el del `1`.

(***) El *algoritmo de la burbuja*, en la variante propuesta para este ejercicio, consiste en recorrer la lista recolocando los elementos que estén en orden relativo incorrecto. Se van comparando elementos consecutivos de la lista, y si están mal colocados, se colocan bien. A continuación se pasa al siguiente par y se repite la operación. Cuando se ha recorrido toda la lista, se evalúa si se ha realizado alguna recolocación. Si ha sido así, se vuelve a hacer un recorrido completo de la lista. En caso contrario, ya está ordenada. Por ejemplo, para ordenar con orden creciente la lista:

1, 4, 5, 3.

se toma el primer par (1,4). Como está bien ordenado, no se hace nada. A continuación, se pasa al siguiente par (4,5). También está en el orden correcto, por lo que se pasa al siguiente (5,3). Este está desordenado, así que se ordena (3,5). Después de recorrer toda la lista, el resultado es:

1, 4, 3, 5.

Como en el recorrido completo se ha recolocado un par (el 3,5), hay que repetir otra vez el proceso. Se toma el primer par (1,4), se deja igual; se toma el segundo (4,3), que está desordenado, se reordena (3,4); se pasa al siguiente (4,5, ya que el 4 ha tomado la posición anterior del 3), y como está ordenado, se deja igual. La lista, después de recorrerla, ha quedado:

1, 3, 4, 5.

Dado que se ha hecho una modificación en el último recorrido, hay que hacer otro más. Pero ahora la lista ya está ordenada, por lo que en este recorrido no se modificará nada: eso significa que hemos llegado al final del proceso.