INFORMÁTICA

Práctica 4.

Programación básica en C.

Grado en Ingeniería en Electrónica y Automática Industrial

Curso 2018-2019

v2.4 (24.09.2018)

A continuación figuran una serie de ejercicios propuestos, agrupados por dificultad. La práctica consiste en hacer la mayor cantidad de ellos posible, teniendo en cuenta que los que se propongan para el examen de laboratorio serán muy parecidos. Para cada uno de ellos, se pide cumplir las especificaciones completamente. Para aquellos detalles del ejercicio que no se especifiquen en el enunciado se admite la decisión justificada del alumno.

Para todos los ejercicios, y salvo que se especifique en el propio enunciado, utilizar la entrada estándar para obtener los datos, y la salida estándar para mostrar los resultados.

Bloque 1.

1. Calcular el promedio de los elementos de una lista. La lista tendrá un número de elementos especificado por el usuario (y será menor que un máximo dado por una constante) y obtendrá los elementos de la lista del usuario.
2. Clasificar los números de una lista en pares e impares. El resultado del programa serán dos listas, una con los pares y otra con los impares. Las listas tendrán un tamaño máximo especificado por medio de constantes. El programa debe asegurarse de que el número de datos leídos no supera el espacio disponible.
3. 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.
4. Invertir el orden de los caracteres de una cadena de texto. Se mostrarán la cadena original y la cadena invertida. Las cadenas tendrán una longitud máxima especificada por una constante. El programa debe asegurarse de que los caracteres de la cadena no superan el espacio disponible.
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:
    1. $n_0=0$;
    2. $n_1=1$;
    i. $n_i=n_{i-1}+n_{i-2}$;
6. Crear un programa que obtenga del usuario una cadena de texto de longitud arbitraria (pero inferior a 256 caracteres) consistente únicamente en letras y espacios, y muestre por pantalla el resultado de dejar, entre cada dos palabras, un único espacio.

Ejercicios de ampliación para el Bloque 1.

7. Calcular el producto escalar de dos vectores. Los vectores tendrán un número de elementos especificado por el usuario (menor que una constante dada) y obtendrá los elementos de los vectores del usuario.
8. 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.

9. Crear un programa que obtenga del usuario una cadena de caracteres y un carácter, y muestre por pantalla la cadena pero sin el carácter indicado.
10. 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 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
```

11. Realizar un programa que cambie las letras minúsculas en mayúsculas y viceversa. El programa irá leyendo una cadena de caracteres desde la entrada estándar y escribirá el resultado de la transformación por salida estándar.
12. Se pide realizar un programa que pida al usuario dos cadenas de caracteres y las imprima por pantalla 'mezcladas', es decir, sacando alternativamente un carácter de cada una de ellas. Por ejemplo, supongamos que las cadenas introducidas por el usuario del programa han sido: `elefante` y `GATO`. El programa debería sacar por pantalla la siguiente cadena:
    `eGlAeTfOante`
13. Comprobar si una cadena de caracteres es un palíndromo.
14. Crear un programa que obtenga del usuario dos vectores de longitud arbitraria (pero igual para los dos e inferior a 20) y calcule la suma de ambos, rotando el primero un número de posiciones hacia la derecha que indique el usuario.
15. El programa obtendrá del usuario una lista de números de longitud arbitraria (pero inferior a 100) 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.
16. 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 de las posiciones 0 a i en la lista original.
17. 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. Una progresión aritmética tiene la forma:

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

18. 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.
19. 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 se define como:

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

Bloque 2.

1. Calcular la suma de dos matrices. Las matrices tendrán unas dimensiones máximas especificadas por constantes. El programa solicitará al usuario las dimensiones de las matrices, y comprobará que no superan los límites especificados por las constantes. La matriz resultado se mostrará fila por fila.
2. De una matriz, extraer su diagonal principal. Mostrar la matriz original, fila por fila, y la diagonal en una única lista. Las dimensiones máximas de la matriz se fijarán por medio de constantes. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.
3. Tomando como entrada una cadena de texto, extraer las consonantes. Mostrar ambas cadenas. Las cadenas tendrán una longitud máxima especificada por una constante. El programa debe asegurarse de que los caracteres de la cadena no superan el espacio disponible.
4. Crear un programa que obtenga del usuario una cadena de caracteres 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).
5. Convertir una matriz dispersa a formato comprimido. Mostrar ambas matrices. La matriz original se mostrará fila por fila. La matriz comprimida es una única lista de tripletas (valor, fila, columna). El formato comprimido 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:

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

   matriz comprimida:

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

   Las dimensiones máximas de la matriz y de la lista de tripletas se fijarán por medio de constantes. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.

6. 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 medio de constantes. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.
7. Mostrar una lista con los números primos que se encuentran entre dos números dados.
8. Hacer un histograma de las vocales de un texto. Se leerá un texto, y se mostrará el número de veces que aparece cada vocal en él. Opcionalmente, el texto se puede almacenar en una cadena. En ese caso, la cadena tendrá una longitud máxima determinada por una constante. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.
9. Contar el número de palabras de un texto. Se leerá un texto y se mostrará cuántas palabras tiene. Considerar que el texto contiene únicamente letras, espacios en blanco, y signos de puntuación. Son signos de puntuación válidos la coma, el punto y coma, el punto y los dos puntos. Tras un signo de puntuación podrá o no haber un espacio en blanco. Opcionalmente, el texto se puede almacenar en una cadena. En ese caso, la cadena tendrá una longitud máxima determinada por una constante. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.
10. 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:
   *
  ***
 *****
*******
```

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 5 veces, el `2` 3, el `3` 7 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 11, pero simular la tirada de dos dados.
13. 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.

Ejercicios de ampliación para el Bloque 2.

14. De una matriz, extraer su diagonal secundaria. Mostrar la matriz original, fila por fila, y la diagonal secundaria en una única lista. Las dimensiones máximas de la matriz se fijarán por medio de constantes. El programa debe asegurarse de que el número de elementos no supera el espacio disponible.

15. Imprimir el primer carácter de cada palabra que exista en una cadena de caracteres. El programa solicitará al usuario la cadena de entrada y a continuación imprimirá en la línea siguiente dichos caracteres separados entre sí por un espacio.
16. Crear un programa que obtenga del usuario una matriz de dimensiones arbitrarias (pero inferiores a 20) e indicará por pantalla cuál es la columna la suma de cuyos elementos es mayor.
17. Calcular la solución de una ecuación de segundo grado. Téngase en cuenta que la ecuación puede tener como solución valores reales o complejos.
18. 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), 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.
19. 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.
20. Crear un programa que obtenga del usuario una matriz cuadrada de dimensión arbitraria (pero menor de 20) y muestre la matriz resultante de transponerla respecto a su diagonal secundaria.
21. Repetir el ejercicio 10 mostrando únicamente los bordes de la pirámide.
22. Repetir el ejercicio 10 mostrando una pirámide invertida.
23. Repetir el ejercicio 10 mostrando un rombo.
24. Repetir el ejercicio 10 mostrando un diábolo. Ejemplo (para 5 pisos):

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

25. Implementar el método de Newton-Raphson para obtener la raíz cuadrada de un número `a`. Este método consiste en iterar la siguiente fórmula:

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

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 el error máximo `E` para la raíz. De esta manera, la iteración se detendrá cuando

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

26. Realizar un programa que calcule el determinante de una matriz 3x3.
27. Contar el número de veces que un determinado patrón aparece en una línea. Se leerá en primer lugar un patrón, y a continuación la línea, y se mostrará el número de veces que aparece. Opcionalmente, la línea y el patrón se pueden

almacenar en sendas cadenas. En ese caso, las cadenas tendrán una longitud máxima determinada por sendas constantes. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.

28. 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.
29. 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 derecha como indique el número. Repetirá la lectura del número y la rotación hasta que la cadena leída esté vacía.
30. El programa obtendrá del usuario una matriz cuadrada de dimensión inferior a 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
```

31. Crear un programa que obtenga del usuario una matriz de enteros dimensión arbitraria (pero inferior a 10 filas y 10 columnas), 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
```

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 = 40 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 orden máximo 10 (aunque puede ser menor) y un vector de, como máximo, 10 elementos. El programa indicará si la matriz contiene el vector en alguna de sus 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 una cadena de caracteres arbitrariamente larga que finalice con el carácter fin de fichero, EOF (en Linux, Control-D). Sólo 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 se fijarán por medio de constantes. 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 en binario, octal, hexadecimal y decimal. Se permitirá al usuario especificar el sistema de representación previamente a la introducción de los datos. La selección de la operación a realizar se puede solicitar por menú. El resultado se mostrará en el mismo sistema de representación en el que se introdujeron los datos. Es preciso evitar las divisiones por cero. El programa repite continuamente su funcionamiento hasta que el usuario pulse una tecla determinada. Para este ejercicio, utilizar exclusivamente el teclado y la pantalla como entrada y salida estándar, respectivamente.
4. Calcular el número combinatorio $\binom{m}{n}$, con $0 \leq n \leq m$. Sugerencia: se recomienda utilizar recursividad.
5. Contar el número de bits a 1/0 en un número de tipo `long int`. Mostrar únicamente el número de bits a 1.
6. 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).
7. 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. Mostrará la cadena modificada. Si no se ha encontrado el patrón, se mostrará la cadena sin modificar. Opcionalmente, los textos se pueden almacenar en cadenas. En ese caso, cada cadena tendrá una longitud máxima determinada por una constante. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.

Ejercicios de ampliación para el Bloque 3.

8. 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.
9. Ordenar una lista de palabras. Se leerán las palabras de la lista, y a continuación se mostrará la lista ordenada alfabéticamente. Opcionalmente, el texto se puede almacenar en una cadena. En ese caso, la cadena tendrá una longitud máxima determinada por una constante. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.
10. Escribir un filtro: el programa leerá un patrón, y a continuación un texto multilínea. Mostrará sólo aquellas líneas del texto que contengan el patrón. Opcionalmente, los textos se puede almacenar en cadenas. En ese caso, cada cadena tendrá una longitud máxima determinada por una constante. El programa deberá asegurarse de que el número de caracteres leídos no excede el espacio disponible.
11. 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).
12. 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.

13. 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 arbitraria 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.

14. Realizar un programa que lea una frase y, conservando el orden de las palabras, escriba cada una de ellas al revés. 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. Suponer que el código de cada elemento es único.
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×n, se define la submatriz Ap,q como la matriz, de tamaño (n − 1) × (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:

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

en donde $a_{k\,\mathrm{j}}$ 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 la misma siempre, por lo que, en este caso, el valor que se usa como semilla será 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. Evidentemente la lista ya está ordenada, por lo que en este recorrido no se modificará nada, y de esa manera se llega al final del proceso.