EJERCICIOS DE FUNDAMENTOS DE INFORMÁTICA

TESTS Y EJERCICIOS RESUELTOS

José María Rodríguez Corral • José Galindo Gómez • María José Ferreiro Ramos
María del Carmen Aranda Garrido • José Luis Isla Montes

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE CÁDIZ

Ejercicios de Fundamentos de Informática

Tests y ejercicios resueltos.

Autores:

José María Rodríguez Corral.
José Galindo Gómez.
María José Ferreiro Ramos.
María del Carmen Aranda Garrido.
José Luís Isla Montes.

UNIVERSIDAD DE CÁDIZ
SERVICIO DE PUBLICACIONES
1997

José María Rodríguez Corral.
José Galindo Gómez.
María José Ferreiro Ramos.
María del Carmen Aranda Garrido.
José Luís Isla Montes.

I.S.B.N.: 84-7786-464-0
Diseño de Portada: Creasur, S.C.
Imprime: Servicio de Autoedición e Impresión
Universidad de Cádiz

- A los alumnos, que son quienes le dan todo el sentido a mi profesión y constituyen la motivación principal para realizar día a día mi trabajo. También quiero dedicar el libro a mis compañeros autores y, por supuesto, al autor del Prólogo.

  * José María Rodríguez Corral.

- A todos los que trabajan por un mundo mejor lejos de sus intereses personales y en especial a Ken Saro-Wiwa, escritor y ecologista, y a otros ocho Ogonis que fueron ahorcados en Nigeria, en 1995, por defender las tierras de los Ogoni de la devastación causada por las actividades petrolíferas de la empresa Shell.

  Greenpeace y Amnistía Internacional no pudieron evitar la ejecución.

  * José Galindo Gómez.

- A los estudiantes y lectores, porque para ellos y por ellos se siguen editando libros.

  * María José Ferreiro Ramos.

- A toda la gente que quiero y muy especialmente a José María, María José y Pp Galindo, aunque ya los incluí antes.

  * José Luís Isla Montes.

Índice General

Prólogo

Aprender es como remar contra corriente; si no se avanza, se retrocede.

Proverbio chino.

Cuando un autor científico escribe un libro de problemas, se mueve entre dos razones principales. La primera es trascender de la teoría, trascender en el sentido puramente pedagógico, esto es ir más allá de la teoría para mostrar otro acontecer de la materia, ofrecer otra visión complementaria, ampliar las perspectivas sobre la disciplina en cuestión. Trascender implica también profundizar en lo teórico, aclarar matices y sugerir nuevos caminos. La segunda razón, más prosaica, se inscribe en lo puramente pragmático, en el "sirve para...", es la razón de la operatividad, es casi la justificación vergonzante de que la teoría existe "para algo", esta visión práctica se pretende erróneamente situar, como fundamento de la teoría. D. Miguel de Unamuno dijo una vez -en contra de esto-, que: "Cuando una teoría es buena no se ha de fundamentar. Si fuera preciso hacerlo se podría inventar cualquier fundamento". La teoría no debe ser una servidumbre obligada de la praxis, sino que, a nuestro juicio, la práctica debe conjugar las razones, la de trascendencia y la pragmática creando un andamiaje pedagógico que ayude a alcanzar el objetivo de enseñar -y de aprender-.

Alguien decía que el principal secreto de la enseñanza era simplificar; simplificar no sólo en el sentido de clarificar, llevando cada cosa a sus esencias o a sus principios, sino, en todo; elegir lo que está al alcance de los alumnos, lo que por ellos es asimilable, lo que cubre su necesidad de saber y al mismo tiempo da una idea fiel de la verdad... Simplificar eliminando aquello que por preciso que sea, excede del espíritu, el tiempo y los medios. Posiblemente, enseñar poco, a fin de enseñar bien.

El libro, de mis queridos compañeros -que tengo la satisfacción y el honor de prologar- se enmarca en estas líneas; síntesis y simplificación, logrando una completa armonía entre la razón trascendente y la razón pragmática. En suma, un excelente libro que completa la serie de sus publicaciones y cierra un círculo, que espero, sea para abrir otros.

IGNACIO PÉREZ BLANQUER
Prof. Titular del Dpto. de Lenguajes y
Sistemas Informáticos de la UCA.

Introducción al Libro

Tras la publicación de los libros **Aprendiendo C** y **Fundamentos Informáticos**, cierto amigo nuestro nos sugirió la idea de llevar a buen término el trabajo emprendido. Según él, se trataría de escribir un libro de ejercicios resueltos para las asignaturas de Introducción a la Informática que se imparten en los distintos Campus de la Universidad de Cádiz.

A comienzo del curso 96/97, nada más comenzar el trabajo nos dimos cuenta de que el objetivo debía ser más ambicioso. No podía quedarse en un libro destinado exclusivamente a nuestros alumnos. Decidimos, por tanto, ampliar nuestros horizontes y escribir un libro que realmente fuera **útil** y **práctico**, no sólo para nuestros alumnos, sino para los de otras Universidades, de Enseñanzas Medias e incluso personas que desearan prepararse una oposición en cuyos contenidos se incluyeran temas de *Informática Básica*.

Ejercicios de Fundamentos de Informática se compone de cuatro capítulos, cada uno de los cuales consta de un conjunto de problemas resueltos y una serie de preguntas tipo test. A continuación vamos a explicar de forma sucinta el contenido de cada capítulo:

El Ordenador.

En este capítulo, tanto los ejercicios como las preguntas de test giran en torno a la estructura interna del ordenador, las unidades funcionales en que éste se descompone (Memoria Principal y Unidad Central de Proceso) y los periféricos, mediante los cuales el ordenador se comunica con el exterior.

Representando la Información.

Los diferentes sistemas de numeración, las operaciones básicas aritméticas y lógicas, los códigos de Entrada Salida y los códigos de detección de errores son los temas sobre los cuales versan los distintos ejercicios y preguntas del segundo capítulo.

Sistemas Operativos.

Un sistema operativo o software funcional es un programa o conjunto de programas que pretende facilitar el uso del ordenador y conseguir que los distintos programas que en él se ejecutan utilicen los recursos de éste lo más eficientemente posible. Los ejercicios y las preguntas del capítulo pretenden afianzar los conocimiento teóricos ya adquiridos sobre los conceptos fundamentales y las funciones básicas de un sistema operativo. La última parte del capítulo trata los principales aspectos de dos sistemas operativos actuales muy conocidos: MS-DOS y UNIX.

Metodología de la Programación y Lenguaje C.

En primer lugar, pretendemos afianzar y poner en práctica los conceptos básicos y técnicas relacionadas con el diseño de algoritmos y programas bien modularizados y estructurados. Por otra parte, a lo largo del capítulo se profundiza gradualmente en los distintos aspectos correspondientes al análisis y desarrollo de funciones y programas en lenguaje C.

Ejercicios de Fundamentos de Informática no es una obra acabada, sino que esta abierta a sugerencias para futuras modificaciones y mejoras. Por ello, desde aquí animamos a los lectores a que nos hagan llegar todas las sugerencias que consideren oportunas, para que entre todos hagamos un libro cada vez más útil y mejor adaptado a nuestras necesidades.

LOS AUTORES

Capítulo 1

El ordenador.

1. Si un disco tiene una capacidad de 1.4 MB menos 10.1 KB

   a) ¿Cuál es su capacidad expresada en bytes?
   b) ¿De qué tipo puede ser dicho disco?

   SOLUCIÓN:

   a) Simplemente hay que realizar un cambio de unidades.

      $1.4 * 2^{20}$ bytes = 14 * 1048576 bytes = 1468006.4 bytes.
      $10.1 * 2^{10}$ bytes = 10.1 * 1024 bytes = 10342.4 bytes

      Capacidad total: 1.468006.4 bytes - 10342.4 bytes = 1.457.664 bytes.

   b) Dada su escasa capacidad, lo más probable es que se trate de un disco flexible.

      En particular, esta descripción corresponde a un disquete de tres pulgadas y media de *alta densidad* (HD), con formato MS-DOS.

2. Un disco duro formado por 5 discos tiene 250 cilindros y 80 sectores/pista. Si el tamaño de cada sector es 512 bytes ¿Cuántos Kilobytes tiene el disco duro? ¿Y Megabytes?

   SOLUCIÓN:

   Para saber el número total de sectores que tiene el disco, antes hay que conocer el número total de pistas. Éste se obtiene mediante la siguiente operación:

   250 cilindros * 5 discos = 1250 pistas

   Una vez conocido el número de pistas, basta multiplicar éste por el número de sectores por pista para averiguar el número total de sectores:

   1250 pistas * 80 sectores/pista = 100000 sectores

   Por último, conocido el número total de sectores, la capacidad total del disco es la siguiente:

   100000 sectores * 512 bytes/sector = 51200000 bytes = 50000 Kilobytes = 48.82 Megabytes

3. Supóngase que tenemos un fichero que almacena 3666 bytes de información. Calcular el espacio real que ocupará ese fichero si se almacena en un disco magnético que tiene el siguiente tamaño por unidad de asignación (*cluster*):

   a) 512 bytes / cluster.
   b) 2 Kilobytes / cluster.
   c) 8 Kilobytes / cluster.

SOLUCIÓN:

Todo fichero ocupa un determinado número de clusters, agrupaciones o unidades de asignación. Cada cluster es asignado exclusivamente a un fichero y no puede ser compartido. El número de clusters que se asignan a un fichero es el menor número de ellos que puedan contenerlo.

Si tenemos un fichero con B bytes y lo queremos almacenar en un disco que tiene C bytes por cluster, dicho fichero ocupará en el disco el siguiente número de clusters:

$$(B + C - 1) \text{ div } C$$

donde *div* es el operador de división entera.

Por tanto, el tamaño que realmente está reservado para un fichero de B bytes viene dado por la siguiente expresión:

$$C * ((B + C - 1) \text{ div } C) \text{ bytes}$$

En nuestro caso, el fichero tiene 3666 bytes y ocupará una cantidad de espacio distinta según el tamaño del cluster del disco en el que se guarde:

a) 512 bytes / cluster.

$$(3666 + 512 - 1) \text{ div } 512 =$$
$$4177 \text{ div } 512 = 8 \text{ clusters}$$

El espacio ocupado en bytes es: 8 clusters * 512 bytes / cluster = 4096 bytes = 4 Kilobytes.

b) 2 Kilobytes / cluster = 2048 bytes / cluster.

$$(3666 + 2048 - 1) \text{ div } 2048 =$$
$$5713 \text{ div } 2048 = 2 \text{ clusters}$$

El espacio ocupado en bytes es: 2 clusters * 2048 bytes / cluster = 4096 bytes = 4 Kilobytes.

Obsérvese que en este caso el número de bytes reservados para ese fichero coincide con el caso anterior, pero no es necesario que coincida. Véase el siguiente caso:

c) 8 kilobytes / cluster = 8192 bytes / cluster:

$$(3666 + 8192 - 1) \text{ div } 8192 =$$
$$11857 \text{ div } 8192 = 1 \text{ cluster}$$

El espacio ocupado en bytes es: 1 cluster * 8192 bytes / cluster = 8192 bytes = 8 Kilobytes.

Esta es la razón por la cual en un disco que tiene una capacidad de N bytes no se pueda almacenar normalmente un número de ficheros en el que la suma de capacidades sea próxima a N. Puede comprobarse fácilmente cuando se intenta grabar un fichero en un disco y se produce el error "Disk Full" (Disco Lleno). En ese caso, es muy probable que la suma de los tamaños de todos los ficheros del disco, incluido el fichero que queremos grabar, sea menor que la capacidad global del disco.

4. Se sabe que la capacidad máxima de un CD-ROM es de 660 MB. Si se tiene en cuenta que la velocidad que necesita un CD-A (Compact Disk de Audio) para su reproducción es la denominada *simple velocidad* de un lector de CD-ROM, ¿Cuántos minutos de audio pueden almacenarse en este soporte?

SOLUCIÓN:

En primer lugar conviene recordar que la velocidad de transferencia de un lector de CD-ROM de *simple velocidad* es 150 KB/seg.

De este modo, si dividimos 660 MB (675840 KB) entre 150 KB/seg obtenemos un tiempo de 4505,6 segundos, que equivale aproximadamente a 75 minutos. Este es el número máximo de minutos que puede durar la información de audio almacenable en un CD-ROM y es otro modo de expresar la capacidad máxima de dicho soporte.

Notas:

- En la información que se almacena en un CD-A no existen códigos de detección o corrección de errores.
- En general se aconseja no almacenar más de 650 MB (aproximadamente 74 minutos de audio) en un CD-ROM con el fin de evitar posibles errores de lectura.

5. Indíquense los periféricos que manejan, de algún modo, tanto señales analógicas como digitales.

SOLUCIÓN:

Módem:	Modula la información digital procedente del ordenador para transmitirla a través de una línea de transmisión analógica. También recibe la información procedente de la línea analógica y la demodula para transmitírsela al ordenador receptor.
Tarjeta de Sonido:	Captura el sonido, lo digitaliza y trata en formato digital. También realiza el proceso inverso y genera sonido mediante la conversión analógica del mismo almacenado en formato digital.
Scanner:	Analiza las señales luminosas (analógicas) procedentes de una imagen y digitaliza sus colores. Una vez almacenada, la imagen puede procesarse en su formato digital y posteriormente imprimirse.
Conversor A/D:	Digitaliza una señal analógica, la cual puede proceder, por ejemplo, de un sensor, un micrófono o una cámara de vídeo.
Conversor D/A:	Transforma una señal digital en analógica. Esta última puede tratarse, por ejemplo, de una señal de audio que haga sonar un altavoz, una señal de vídeo dirigida a un monitor o una tensión que active un motor eléctrico.

6. Supongamos que tenemos una tarjeta de sonido que puede digitalizar sonidos o música con la calidad de un Compact Disk de audio normal (CD-A), con 44.1 Khz. de frecuencia de muestreo y 12 bits de tamaño de muestra. ¿Cuánto espacio necesitaríamos para almacenar la canción de Héroes del Silencio titulada "Sal" de su disco "Senderos de Traición" que dura cuatro segundos?

SOLUCIÓN:

$$4\ segs. * 44100\ \frac{muestras}{seg.} * 12\ \frac{bits}{muestra} = 2116800\ bits = 264000\ bytes$$

7. Algunas de las características más importantes de un monitor son el tamaño, la relación de aspecto (*aspect ratio*), la resolución y la densidad de puntos de imagen. Expliquemos brevemente estas características:

- El tamaño se indica, al igual que en las pantallas convencionales de televisión, mediante el número de pulgadas que mide la diagonal de la pantalla.

- La relación de aspecto es la que existe entre el ancho y el alto de la pantalla, usualmente 4 / 3.

- La resolución mide el número de puntos de una imagen y viene dada por el número de pixeles que ocupa una fila y una columna de dicha imagen. Una resolución típica es 800 x 600, lo cual indica que cada fila tiene 800 pixeles y cada columna 600 pixeles.

- La densidad de puntos viene dada por el número de pixeles que hay en una pulgada de pantalla. Lógicamente, a mayor densidad los puntos estarán más cercanos y los trazos de la imagen se verán más continuos. La densidad se mide en puntos por pulgada o dpi (dots per inche). Como nota curiosa diremos que el ojo humano es capaz de distinguir aproximadamente hasta los 380 dpi en una imagen vista a una distancia de 45 cm. Un objetivo consiste en conseguir monitores de al menos 300 dpi.

Visto todo lo anterior, supóngase que se tiene un monitor de 21 pulgadas con una relación de aspecto de 4 / 3. ¿Cuál es la resolución de dicha pantalla si su densidad es de 300 dpi?

SOLUCIÓN:

Si al número de pixeles por cada fila le llamamos **a** (anchura) y al número de pixeles por cada columna le llamamos **h** (altura), tenemos que:

$$a / h = 4 / 3$$

de donde obtenemos que:

$$a = 4\,h / 3$$

Como el monitor tiene 21 pulgadas, por el teorema de Pitágoras podemos averiguar el número de pulgadas de la altura y la anchura:

$$a^2 + h^2 = 21^2$$

$$(4\,h / 3)^2 + h^2 = 21^2$$

$$(16 / 9 + 1)\,h^2 = 21^2$$

$$h^2 = 441 / (25 / 9) = 158.76$$

$$h = 12.6 \text{ pulgadas}$$

de donde podemos sacar el tamaño de la anchura:

$$a = 4\,h / 3 = 16.8 \text{ pulgadas}$$

Conocidos esos tamaños y la densidad de puntos podemos obtener el resultado final:

Resolución horizontal: 16.8 pulgadas * 300 dpi = 5040 pixeles
Resolución vertical: 12.6 pulgadas * 300 dpi = 3780 pixeles

Por lo tanto, la resolución de la pantalla es 5040 x 3780 pixeles.

8. Existe una propuesta para mejorar la definición de la televisión actual. Es la llamada TV de alta definición, la cual incluye los siguientes dos requisitos básicos:

1.- Resolución: 1280 x 1024 pixeles.
2.- Color real (*true color*).

Averiguar:

a) Relación de aspecto (*aspect ratio*) de esta definición.
b) Número total de pixeles de la misma.
c) Cantidad de memoria mínima que se necesita para ella.

SOLUCIÓN:

a) La relación de aspecto es el cociente entre el ancho y el alto de la pantalla:

1280 pixeles / 1024 pixeles = 1.25
(relación de 4 / 3)

b) El número total de pixeles se obtiene multiplicando el número de pixeles de cada fila por el de cada columna:

1280 * 1024 = 1310720 pixeles

c) Como se trata de color real, para cada píxel se guardará la información del color considerando 256 niveles por cada uno de los tres colores básicos: rojo, verde y azul (*rgb*, red-green-blue). Esto implica almacenar ocho bits por cada color, ya que $2^8 = 256$. Por tanto, para cada píxel se guardará la información del color en tres bytes (24 bits).

Como tenemos 1280 * 1024 pixeles distintos y tenemos que guardar la información del color de todos ellos, la cantidad de memoria mínima necesaria viene dada por la siguiente operación:

1280 * 1024 * 3 bytes =
3840 * 1024 bytes =
3840 KB (Kilobytes) =
3.75 MB (Megabytes)

9. Se dispone de un monitor de 800x600 pixeles de resolución, con 256 colores posibles y 0.28 mm de separación entre pixeles. Este monitor está conectado a un ordenador cuyo microprocesador trabaja a una frecuencia de reloj de 200 Mhz. Además, el tiempo que tarda en dibujarse un pixel es 25 ns. (nanosegundos). Hallar razonadamente:

a) Frecuencia de barrido vertical del monitor.
b) Frecuencia de barrido horizontal.
c) Cantidad mínima de memoria de vídeo necesaria.

SOLUCIÓN:

a) La pantalla tiene una resolución de 800x600 pixeles, es decir, 480000 pixeles en total. Como cada pixel se dibuja en 25 ns, el tiempo de dibujo de la pantalla completa es:

480000 pixeles * 25 ns = 12 * 10^6 ns = 12 ms.

Por tanto, la frecuencia de barrido vertical es:

$$\frac{1}{12 * 10^{-3} segs} = \frac{1000}{12\ segs} = 83.\hat{3}\ Hz.$$

Este resultado indica que la pantalla se dibuja (refresca) 83.33 veces en un segundo.

b) Para obtener la frecuencia de barrido horizontal, dado que cada línea tiene 800 pixeles, una línea tardará en dibujarse 800 * 25 ns. Por lo tanto, la frecuencia de barrido horizontal es:

$$\frac{1}{800 pixels * 25ns} = \frac{1}{20000\ ns} = 50\ Khz$$

También podía calcularse a partir de la frecuencia de barrido vertical, obtenida en el apartado anterior:

$$\frac{83.\hat{3}\ pantallas}{segundo} * \frac{600\ lineas}{pantalla} = 50\ Khz$$

c) Para almacenar un pixel de la pantalla, dado que su color puede elegirse entre 256 colores distintos, se necesitan ocho bits, ya que $2^8 = 256$. Como la pantalla tiene una resolución de 800x600 pixeles, el total de memoria necesaria (si un byte son ocho bits) viene dado por la siguiente expresión:

800 * 600 pixeles * 8 bits = 480000 pixeles * 8 bits = 480000 bytes.

10. Entre dos ordenadores, cuyos microprocesadores poseen ambos una frecuencia de reloj de 200 Mhz., existe una línea de comunicación a 18 Kbps (kilobits por segundo). Si queremos transmitir un fichero de texto en código ASCII extendido (8 bits por carácter) usando un bit de paridad, ¿Cuántos CPS (caracteres por segundo) se podrán transmitir si suponemos que ambos microprocesadores tienen un bus de datos de 64 bits?

NOTA: Suponemos que no se envía ningún otro tipo de trama: cabecera, sincronismo, etc.

SOLUCIÓN:

En realidad, tanto el tamaño del bus de datos de los microprocesadores como su frecuencia de reloj, son datos totalmente irrelevantes para hallar la solución del problema. Tan sólo se trata de hacer un cambio de unidades en la velocidad de transmisión conociendo el tamaño del carácter a transmitir.

$$Num\ CPS = \frac{18\ kbps}{9\ bits\ /\ caracter} = \frac{18\ *\ 1024\ bps}{9\ bits\ /\ caracter} = 2048\ CPS$$

Preguntas de Test

1. El uso de transistores en los ordenadores supuso que éstos tuvieran un menor tamaño y mayor velocidad que sus antecesores, construidos a base de circuitos integrados.

   a) Verdadero.
   b) Falso.

2. El concepto de *programa almacenado*, es decir, la existencia de un programa almacenado en una memoria y una Unidad Central de Proceso que ejecute una a una todas sus instrucciones, se debe a

   a) Isaac Newton.
   b) Confucio.
   c) Peter Norton.
   d) Blaise Pascal.
   e) Nicklaus Wirth.
   f) Bill Gates.
   g) Von Neumann.

3. En la memoria de una máquina de Von Neumann

   a) se almacenan exclusivamente los datos que se desea procesar.
   b) se almacena exclusivamente el programa (conjunto de instrucciones) a ejecutar.
   c) datos y programas se almacenan en memorias distintas claramente diferenciadas.
   d) datos y programas comparten la misma memoria.
   e) Ninguna de las afirmaciones anteriores es correcta.

4. La máquina de Von Neumann es un modelo de máquina de procesamiento digital que consta de los siguientes módulos básicos:

   a) circuitos conversores de señales analógicas a digitales y viceversa.
   b) CPU (unidad central de proceso), memoria y periféricos de Entrada/Salida.
   c) unos circuitos especiales dependientes de la utilidad que se le vaya a dar a la máquina, según el programa que se desee ejecutar.
   d) Ninguna de las respuestas anteriores es correcta.

5. Un kilobyte (KB) es

   a) medio megabyte (MB).
   b) 1024 bytes.
   c) 1024 MB.
   d) 1000 bytes.
   e) Su tamaño depende del tipo de memoria RAM.

6. Un Gigabyte (GB) equivale a

   a) 1024 bytes.
   b) 2^{20} Megabytes.
   c) 2^{20} Kilobytes.
   d) 2^{30} bits.

7. ¿Cuántos terabytes (TB) hay en un gigabyte (GB)?

   a) Ninguno completo (sólo hay parte).
   b) Medio terabyte exacto.
   c) Dos terabytes.
   d) 1024.
   e) Ninguna de las respuestas anteriores es correcta.

8. Si disponemos de una memoria con 1024 palabras y cada una de ellas tiene un tamaño de dos bytes, la memoria tiene una capacidad global de

   a) un kilobyte.
   b) dos kilobytes.
   c) mil veinticuatro kilobytes.
   d) dos mil cuarenta y ocho bits.
   e) dos mil cuarenta y ocho kilobytes.
   f) Ninguna de las respuestas anteriores es correcta.

9. La memoria principal es de acceso

   a) directo.
   b) secuencial.
   c) volátil.
   d) de un tipo de acceso propio de este tipo de memoria.

10. La memoria principal de un ordenador

   a) es hardware.
   b) es de dos tipos: RAM y ROM.
   c) es de acceso directo.
   d) almacena información en código binario.
   e) Todas las respuestas anteriores son correctas.
   f) Ninguna de las respuestas anteriores es correcta.

11. Una dirección de memoria es

   a) un número de bits (ceros o unos) aleatorios.
   b) un conjunto de bits que indican si los datos están en memoria RAM o ROM.
   c) una secuencia de bits que indican la posición en la que comienza un programa situado en la memoria RAM.
   d) un conjunto de bits que indican una posición concreta de una palabra de memoria.
   e) la localización de la memoria en la CPU.

12. RAM significa

   a) Ramdon Access Memory.
   b) Real Access Memory.
   c) Reliability Access Memory.
   d) Memoria Alta de almacenamiento Rápido.
   e) Almacenamiento Real en Memoria.

13. La memoria RAM es

a) memoria no volátil de acceso aleatorio.
b) memoria volátil de acceso directo.
c) memoria de sólo lectura.
d) el conjunto de buffers de un ordenador.
e) memoria de acceso real.

14. La memoria RAM sirve para almacenar

a) los registros de la unidad central de proceso (CPU).
b) los registros de una Base de Datos.
c) datos e instrucciones de forma temporal.
d) información que necesitemos almacenar por mucho tiempo.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

15. Una palabra de memoria es

a) el número de bytes que necesitamos para realizar una operación.
b) el mínimo número de bytes que podemos leer o escribir de una vez en la memoria principal.
c) un conjunto de bits que indican una posición concreta en la memoria.
d) el número de bits que forman un byte.
e) el número de bytes que forman un kilobyte.
f) el número de transferencias a memoria por segundo.

16. Un buffer es

a) hardware.
b) software.
c) una parte común a todos los periféricos.
d) un programa controlador de periféricos o dispositivos (driver).
e) Son correctas las respuestas a) y b).

17. La memoria caché es

a) una parte de la memoria ROM.
b) chips (circuitos integrados) de memoria rápida, que se colocan entre un dispositivo rápido y otro lento, para acelerar los accesos al dispositivo lento.
c) un chip coprocesador que acelera las operaciones matemáticas complejas de la unidad aritmético-lógica (ALU).
d) la parte de la memoria RAM que almacena el S.O. para que éste se ejecute con rapidez.
e) una parte de la CPU encargada de acelerar todo el sistema.
f) memoria usada para gestionar el sistema de vídeo.

18. La BIOS es

a) un circuito para controlar los Buses.
b) el sistema básico de Entrada/Salida en memoria ROM.
c) una parte de la memoria RAM.
d) la memoria EEPROM.
e) el Bus de Instrucciones Operativas del Sistema.

19. La CPU es

a) la Unidad Central de control de Periféricos.
b) el dispositivo encargado de gestionar la memoria.
c) el hardware encargado de procesar la información.
d) el proceso asignado a un sistema operativo (S.O.) de disco (DOS).
e) la parte del S.O. que controla el hardware del ordenador.

20. La Unidad Central de Proceso (CPU) del computador la constituyen

a) el ordenador central y sus periféricos.
b) los periféricos y la memoria principal.
c) la memoria RAM, la memoria ROM y la Unidad de Control (UC).
d) la memoria RAM y la Unidad de Control (UC).
e) la Unidad Aritmético-lógica (ALU) y la Unidad de Control (UC).
f) Ninguna de las respuestas anteriores es correcta.

21. El reloj de la CPU sirve para

a) saber la hora según el huso horario del país en el que se esté.
b) para indicar a la memoria cuando queremos leer datos o instrucciones.
c) para sincronizar las operaciones de la máquina.
d) para conocer el tiempo de ejecución de cualquier programa.

22. Los registros de la CPU son

a) unidades de memoria rápida del tamaño de una palabra de memoria.
b) unidades de memoria intermedia entre la CPU y los periféricos.
c) el registro de instrucción en ejecución y el registro contador de instrucciones.
d) de direcciones y datos.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

23. Los registros de la CPU son

a) unidades de memoria que sólo existen dentro de la Unidad de Control.
b) pequeñas unidades de memoria destinadas a almacenar información de control.
c) decodificadores del código de las instrucciones de un programa.
d) más rápidos que la memoria principal.

24. Al elemento de un ordenador que se encarga de buscar la instrucción máquina, interpretarla y controlar su ejecución se le llama

a) Memoria Principal.
b) Unidad de Control.
c) C.P.U. (Unidad Central de Proceso).
d) A.L.U. (Unidad Aritmético-lógica).
e) Memoria RAM.

25. La Unidad de Control (UC) decide qué componente del ordenador debe actuar en cada momento:

a) Verdadero.
b) Falso.

26. La Unidad de Control (UC) se encarga de

a) controlar el flujo de Entrada/Salida (E/S) del ordenador.
b) asegurar que la ALU realiza bien sus operaciones y que el resto del sistema funciona sincronizadamente con el reloj.
c) controlar que el programa en ejecución hace lo que realmente queremos que haga.
d) determinar la instrucción a ejecutar, decodificarla e interpretarla.
e) supervisar el contenido de los registros de la CPU.

27. El Registro Contador de Instrucciones de la Unidad de Control contiene

a) un contador de pulsos de reloj.
b) el número de instrucciones ejecutadas durante un período de tiempo.
c) el número de instrucciones del programa en ejecución.
d) la siguiente instrucción a ejecutar.
e) la dirección de memoria de la siguiente instrucción a ejecutar.
f) el estado de la instrucción ejecutada.
g) Ninguna de las respuestas anteriores.

28. El Registro de Instrucción de la Unidad de Control contiene

a) el código de operación del conjunto de instrucciones del programa.
b) el código de operación de una instrucción del programa.
c) la instrucción a ejecutar en un lenguaje de programación de alto nivel.
d) la dirección del código de operación de la instrucción a ejecutar.

29. La Unidad de Control realizará las siguientes fases para ejecutar cada instrucción del programa:

a) Fase de captación de instrucción y fase de realización de operaciones aritmético-lógicas.
b) Fase de captación de instrucción y fase de ejecución de instrucción.
c) Fase de decodificación de instrucción y fase de ejecución de instrucción.
d) Fase de compilación de instrucción y fase de ejecución de instrucción.

30. Las fases de ejecución de instrucciones en la Unidad Central de Proceso (CPU) son

a) fase de búsqueda de instrucción y de escritura de resultados.
b) fase de búsqueda de instrucción y de ejecución de la misma.
c) fase de captación de instrucción y de envío a la Unidad Aritmético-lógica (ALU).
d) fase de carga en memoria y de envío a la Unidad de Control (UC).
e) fase de lectura de memoria y de escritura en los registros.
f) Ninguna de las respuestas anteriores es correcta.

31. Los registros de la unidad aritmético-lógica se dividen en

a) registro acumulador, registros de operandos y registro de estado.
b) registro acumulador, registros operacionales y registro de estado.
c) registro contador de programa y registro de instrucción en ejecución.
d) Ninguna de las respuestas anteriores es correcta.

32. Indicar cuál es el orden correcto, de mayor a menor velocidad de acceso, para lectura de información.

a) Memoria principal, memoria caché y registros de la CPU.
b) Registros de la CPU, memoria caché, memoria principal, disco flexible, disco duro (HD) y cinta magnética.
c) Memoria caché, registros de la CPU, memoria RAM y memoria ROM.
d) Memoria caché, RAM, ROM, registros de la CPU, HD, discos flexibles y cinta magnética.
e) Registros de la CPU, memoria caché, RAM, ROM, HD, discos flexibles y cinta magnética.
f) Registros de la CPU, memoria caché, HD, ROM, RAM, discos flexibles y cinta magnética.

33. Por el tipo de información que manejan se distinguen los siguientes tipos de Buses:

a) Bus de datos y control.
b) Bus de datos serie y paralelo.
c) Bus de datos, control y direcciones.
d) Bus de sistema y bus local.
e) Todos los tipos de buses nombrados anteriormente.

34. Las siglas D.M.A. significan

a) Director Management Access, que es un chip encargado de dirigir y gestionar los accesos.
b) Directional Memory Address, que es un bus especial por el que se comunican las direcciones de memoria y la información sobre la que se desea actuar.
c) Directed Memory ALU, que es un método usado para acelerar la ejecución de aquellas instrucciones que hacen uso de la ALU y además requieren datos de memoria principal.
d) Direct Memory Access, que es un chip encargado de controlar el acceso directo a memoria, a fin de que los datos puedan transferirse entre memoria y periféricos sin tener que pasar por los registros de la CPU.
e) Ninguna de las afirmaciones anteriores es correcta.

35. En un ordenador denominamos *puerto*

a) al lugar donde atracan los barcos de datos en la CPU.
b) al lugar del cual leen los datos los programas.
c) al lugar en el cual escriben los datos los programas.
d) al lugar del que leen y en el que escriben los datos los programas.
e) a la conexión eléctrica a través de la cual el ordenador envía y recibe datos de otros dispositivos.
f) Todas las respuestas anteriores son incorrectas.

36. ¿Por cuál de los puertos circulan los bits secuencialmente a través de una única línea de transmisión?

a) Serie.
b) Paralelo.
c) Impresora.
d) ESDI.
e) IDE.

37. La información en un disco duro se organiza en

a) pistas y cilindros.
b) superficies, pistas y cilindros.
c) superficies, pistas y sectores.
d) pistas, sectores y unidades de asignación.

38. Un disco duro tiene

a) tantos cilindros como pistas hay en cada superficie.
b) tantos sectores como pistas hay en el disco.
c) tantas pistas como cilindros hay en el disco.
d) tantas pistas como superficies tiene el disco.

39. Indicar la afirmación correcta de las siguientes, referidas a memoria masiva:

a) Un disco duro (HD) tiene 2 cabezas de lectura/escritura (una para cada cara del disco).
b) Los sectores se dividen en pistas y las pistas en cilindros.
c) Los cilindros se dividen en pistas y las pistas en sectores.
d) Un disco duro, al tener más capacidad, suele ser más lento que un disco flexible.

40. Si tenemos un disco duro con 1 terabyte (TB) de capacidad, ¿Cuántos gigabytes (GB) tiene?

a) Ninguno completo (sólo hay parte).
b) Medio gigabyte exacto.
c) Dos gigabytes.
d) 1024 gigabytes.
e) Ninguna de las respuestas anteriores es correcta.

41. La velocidad de transferencia de un disco duro se expresa como el número de bytes que se transfieren entre éste y la memoria, en cualesquiera de los dos sentidos, por segundo. Esta velocidad depende, entre otros factores, del tipo de disco duro y de su interfaz (los SCSI son los más rápidos) y actualmente oscila entre

a) tres bytes / segundo.
b) tres kilobytes / segundo.
c) tres megabytes / segundo.
d) trescientos treinta y tres megabytes / segundo.
e) tres gigabytes / segundo.

42. Los discos flexibles

a) son removibles y reutilizables.
b) son ideales para copias de seguridad.
c) son robustos y se pueden doblar sin riesgos.
d) no necesitan que se les dé formato previamente a su utilización.
e) son rápidos y de tecnología magnética.

43. Los discos Bernouilli

a) datan del siglo XVIII.
b) son de tecnología híbrida.
c) son como los discos duros pero muchos más veloces.
d) son muy seguros en caso de corte eléctrico.
e) utilizan el teorema de Jacopini-Bohn en su funcionamiento.
f) Todas las respuestas anteriores son correctas.
g) Ninguna de las respuestas anteriores es correcta.

44. La ventaja del disco Bernoulli frente al disco duro consiste en que

a) en caso de producirse un corte eléctrico, éste no deja al disco Bernouilli inutilizable.
b) en caso de golpes fortuitos, no se daña la superficie del disco Bernouilli ni la cabeza de lectura/escritura.
c) el primero tiene mucha mayor capacidad, la cual suele expresarse en Terabytes.
d) Ninguna de las respuestas anteriores es correcta.

45. La ventaja de los discos Bernoulli frente a discos duros consiste en que

a) tienen menor tiempo de acceso.
b) son fácilmente trasladables de un ordenador a otro.
c) tienen mayor velocidad de transferencia.
d) Todas las respuestas anteriores son correctas
e) Ninguna de las respuestas anteriores es correcta.

46. ¿Cuántas pistas tiene un CD-ROM?

a) Depende de su capacidad.
b) Depende de la cantidad de información que almacene.
c) Mil veinticuatro pistas.
d) Más de mil veinticuatro pistas.
e) Una pista.
f) Ninguna de las respuestas anteriores es correcta.

47. ¿Cuántos kilómetros mide aproximadamente la pista de un CD-ROM?

a) 0,5 km.
b) 5 km.
c) 9 km.
d) 34 km.
e) 100 km.

48. La velocidad de transferencia de un CD-ROM

a) es menor que la de un disco duro.
b) es mayor que la de disco duro.
c) depende del modelo, pues existen algunos cuya velocidad es mayor que la de un disco duro.
d) varía según estemos leyendo o escribiendo en él.
e) Ninguna de las respuestas anteriores es correcta.

49. Los primeros lectores de CD-ROM tenían una velocidad de transferencia que pasó a llamarse *simple velocidad*, que es la necesaria para leer los CD de música o CD-A. La velocidad de los modernos lectores se mide como múltiplos de esta simple velocidad, que es igual a

a) 150 bits / segundo.
b) 150 bytes / segundo.
c) 150 KB / segundo.
d) 150 MB / segundo.
e) 150 GB / segundo.

50. Cuando escribimos estas líneas (Octubre de 1996), acaban de salir al mercado los lectores de CD-ROM de velocidad 12X. Esto indica que son 12 veces más rápidos que los primitivos lectores de *simple velocidad*. ¿Cuántas veces se tiene que multiplicar la velocidad de transferencia de los lectores de simple velocidad para equipararla aproximadamente con la de un disco duro?

a) 15X.
b) 20X.
c) 150X.
d) 512X.
e) 1024X.

51. Los soportes de información con tecnología de Cambio de Fase son de tecnología

a) magnética.
b) óptica.
c) híbrida (Magneto-óptica).
d) Ninguna de las anteriores.

52. Los soportes de información con tecnología de Cambio de Fase son

a) soportes ópticos de lectura/escritura.
b) soportes magnéticos de lectura/escritura.
c) soportes ópticos de sólo lectura.
d) soportes en los que sólo se puede escribir una vez.
e) Ninguna de las respuestas anteriores es correcta.

53. Las unidades de los soportes de información con tecnología de Cambio de Fase

a) también pueden leer discos CD-ROM.
b) también pueden leer discos magneto-ópticos.
c) tienen una cabeza de lectura escritura que detecta las variaciones del campo magnético.
d) Todas las respuestas anteriores son correctas.
e) Ninguna de las respuestas anteriores es correcta.

54. Los dispositivos floptical son

a) periféricos de entrada (lectura) exclusivamente.
b) soportes de información magnéticos.
c) soportes de información ópticos.
d) soportes de información magneto-ópticos.
e) Ninguno de las anteriores.

55. Una señal analógica es lo contrario de una señal digital, esto es

a) una señal variable en el tiempo.
b) una señal continua, la cual puede tomar infinitos valores dentro de un intervalo.
c) una señal modular, término utilizado en la codificación de la información digital.
d) una señal discreta, la cual sólo puede tomar un conjunto finito de valores dentro de un intervalo.

56. Indicar las opciones donde aparecen en primer lugar un periférico de Entrada y después otro de Salida.

a) Plotter y ratón.
b) Scanner y plotter.
c) Ratón y joystick.
d) Ratón e impresora.
e) Scanner y trackball.
f) Lápiz óptico y pantalla.
g) Todas las opciones anteriores son correctas.
h) Ninguna de las opciones anteriores es correcta.

57. La VRAM es

a) un sistema de memoria RAM virtual.
b) un tipo de memoria para aplicaciones gráficas.
c) una memoria RAM que necesita batería (volátil).
d) una memoria de acceso secuencial.
e) una memoria usada en pantallas LCD y CRT.

58. Los monitores con tecnología CRT poseen una pantalla de cristal líquido o cuarzo líquido, la cual es muy utilizada debido a su pequeño grosor.

a) Verdadero.
b) Falso.

59. Las impresoras LÁSER:

a) usan tinta líquida para el proceso de impresión.
b) usan ozono para la fijación de la tinta.
c) cargan la tinta (toner) eléctricamente para imprimir.
d) Todas las respuestas anteriores son correctas.
e) Ninguna de las respuestas anteriores es correcta.

60. Un componente clave de una impresora LÁSER es

a) la célula fotoeléctrica.
b) el tambor fotosensible.
c) el arco voltaico.
d) el puente de diodos.
e) el diodo LED.
f) el conversor D/A (Digital/Analógico).
g) Todos los anteriores.
h) Ninguno de los anteriores.

61. ¿Qué es un plotter?

a) Una impresora preparada para dibujar gráficos.
b) Un periférico que sirve para realizar dibujos sobre papel.
c) Un periférico utilizado para digitalizar imágenes.
d) Un soporte de Información.
e) Ninguna de las respuestas anteriores es correcta.

62. El módem se encarga de modular y demodular señales para posibilitar la comunicación de datos analógicos por las líneas digitales ya existentes.

a) Verdadero.
b) Falso.

63. Un módem sirve para

a) comunicar ordenadores por líneas digitales.
b) comunicar ordenadores por líneas analógicas.
c) comunicar ordenadores por cualquier tipo de línea.
d) comunicar ordenadores modulando la red.
e) comunicar ordenadores mediante líneas de fax.
f) modular y demodular las líneas de comunicaciones.
g) Ninguna de las respuestas anteriores es correcta.

64. Si un sistema de comunicación digital transmite a 32 bps (bits por segundo), está transmitiendo a

a) doscientos cincuenta y seis bytes por segundo.
b) cuatro caracteres por segundo.
c) Las dos respuestas anteriores son correctas.
d) Ninguna de las respuestas anteriores es correcta.

65. Los sensores

a) son sensibles a una magnitud física y generan una señal eléctrica proporcional a su valor.
b) digitalizan las señales analógicas que leen.
c) transforman una señal digital en otra analógica.
d) No realizan ninguna de las funciones anteriores.

66. Un scanner nos permite

a) trazar gráficas.
b) digitalizar imágenes.
c) Ninguna de las respuestas anteriores es correcta.

67. La calidad de un scanner se mide por

a) la velocidad con que transmite la información, expresada en BPS (bits por segundo).
b) la resolución de las imágenes que es capaz de mostrar: EGA, VGA, XGA y SVGA.
c) la posibilidad de digitalizar las imágenes en color y la velocidad de transmisión de las mismas en CPS (caracteres por segundo).
d) la posibilidad de digitalizar las imágenes en color y el número de puntos por pulgada (dpi) que es capaz de distinguir en éstas.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

68. Un tablero gráfico es

a) un dispositivo apuntador.
b) una impresora de calidad.
c) un dispositivo para dibujar en papel de cualquier tamaño.
d) un dispositivo para comunicar gráficos entre ordenadores.
e) Ninguna de las respuestas anteriores es correcta.

69. El lápiz óptico es un periférico que sirve para

a) dibujar en un tablero y digitalizar el dibujo.
b) dibujar en una pantalla especialmente sensible.
c) para marcar posiciones en la pantalla.
d) para aplicaciones de CAM (fabricación asistida por ordenador).
e) Ninguna de las respuestas anteriores es correcta.

70. Indicar cuáles de los siguientes dispositivos son apuntadores:

a) Plotter.
b) Scanner.
c) Ratón.
d) Trackball.
e) Impresora.
f) Lápiz óptico.
g) Joystick.

71. Indicar cuál es la afirmación verdadera:

a) La memoria es siempre de tipo magnético u óptico: discos, cintas, CD-ROMs, ...
b) Un plotter es un dispositivo de entrada.
c) Existen pantallas que nos permiten introducir datos.
d) Ninguna de las afirmaciones anteriores es verdadera.

72. Los siguientes ordenadores están ordenados de mayor a menor tamaño, capacidad de proceso, fiabilidad y precio:

a) Ordenador personal (PC), miniordenador y microordenador.
b) Mainframe, PC y miniordenador.
c) Mainframe, Pentium, PC y miniordenador.
d) Miniordenador, mainframe y PC.
e) Mainframe, miniordenador y microordenador.

73. Relacionar los siguientes términos:

milisegundos	frecuencia de reloj
bits por segundo	capacidad de proceso
caracteres por segundo	velocidad de impresión
megaherzios	capacidad de memoria
instrucciones por segundo	tiempo de acceso
kilobytes	velocidad de transmisión

Respuestas

1. b)
2. g)
3. d)
4. b)
5. b)
6. c)
7. a)
8. b)
9. a)
10. e)
11. d)
12. a)
13. b)
14. c)
15. b)
16. a)
17. b)
18. b)
19. c)
20. e)
21. c)
22. a)
23. d)
24. b)
25. a)
26. d)
27. e)
28. b)
29. b)
30. b)
31. a)
32. e)
33. c)
34. d)
35. e)
36. a)
37. c)
38. a)
39. c)
40. d)
41. c)
42. a)
43. d)
44. b)
45. b)
46. e)
47. d)
48. a)
49. c)
50. b)
51. b)
52. a)

53. a)

54. d)

55. b)

56. b), d) y f)

57. b)

58. b)

59. c)

60. b)

61. b)

62. b)

63. b)

64. b)

65. a)

66. b)

67. d)

68. a)

69. c)

70. c), d), f) y g)

71. c)

72. e)

73.

milisegundos	tiempo de acceso
bits por segundo	velocidad de transmisión
caracteres por segundo	velocidad de impresión
megaherzios	frecuencia de reloj
instrucciones por segundo	capacidad de proceso
kilobytes	capacidad de memoria

Capítulo 2

Representando la Información.

1. ¿Cuántos bits como mínimo son necesarios para codificar 260 símbolos?

   SOLUCIÓN:

   Con n bits podemos codificar hasta 2^n símbolos. Se trata entonces de encontrar el menor valor entero de n tal que 2^n sea mayor o igual que 260.

   Este valor es **9**, ya que $2^8 = 256 < 260 < 2^9 = 512$. Este número también se puede calcular como el menor valor entero que sea mayor igual al logaritmo en base 2 del número de símbolos.

   $$n = \lceil \log_2 num\_simbolos \rceil$$

2. Pasar a *base 2* el número en base decimal 9.125.

   SOLUCIÓN:

   Para realizar el cambio de base conviene trabajar con la parte entera y la parte fraccionaria por separado.

   En lo que respecta a la parte entera, basta con realizar sucesivas divisiones enteras entre dos y posteriormente escribir los restos y el último cociente en orden inverso a como se han obtenido:

9 / 2 = 4;	9 % 2 = **1**;
4 / 2 = 2;	4 % 2 = **0**;
2 / 2 = **1**;	2 % 2 = **0**;

   Por lo tanto, $9_{10)} = 1001_{2)}$.

   Sin embargo, para pasar a binario la parte fraccionaria hay que multiplicar ésta por dos sucesivamente e ir anotando las partes enteras que se obtienen:

   0.125 * 2 = **0**.25;
   0.25 * 2 = **0**.50;
   0.50 * 2 = **1**.00;

   Por lo tanto, $0.125_{10)} = 001_{2)}$.

   De este modo, el número $9.125_{10)}$ expresado en binario es $1001.001_{2)}$.

3. Los ordenadores utilizan las notaciones de *complemento a uno* y *complemento a dos* para llevar a cabo unas acciones concretas. Justificar por qué los ordenadores utilizan esas notaciones para realizar dichas acciones.

SOLUCIÓN:

Los ordenadores utilizan las notaciones de complemento a uno y complemento a dos para realizar la resta de dos números binarios.

De este modo, para obtener la diferencia entre minuendo y sustraendo podemos calcular el complemento a uno del minuendo, sumárselo al sustraendo y añadirle al resultado una unidad, o bien sumarle al minuendo el complemento a dos del sustraendo. Ambos métodos proporcionan el mismo resultado.

Por lo tanto, si realizamos las restas exclusivamente como sumas e inversiones evitamos tener que utilizar unidades aritmético-lógicas que incluyan la resta entre sus posibles operaciones, ya que estas unidades son más complejas y costosas.

De hecho, los microprocesadores realizan la resta binaria utilizando la notación de complemento a dos, pues sus unidades aritmético-lógicas resultan más simples, y como consecuencia de ello, su diseño global. Sin embargo, el microcódigo correspondiente a la instrucción máquina de resta se complica mínimamente, debido a lo cual la duración de dicha instrucción es ligeramente mayor. Esto es debido a que la operación de resta se sustituye por una inversión bit a bit y dos operaciones de suma.

4. Escribir todos los números enteros que pueden representarse en un ordenador de *tres bits*, indicando el valor de éstos en base 10, si usa una representación en complemento a dos. ¿Y si usa una representación en complemento a uno?. Razonar qué modo de representación es mejor. ¿Por qué en un modo existen dos representaciones para el cero?

SOLUCIÓN:

Base 10	C_2	C_1
3	011	011
2	010	010
1	001	001
0	000	000
- 0	---	111
- 1	111	110
- 2	110	101
- 3	101	100
- 4	100	---

A la vista de los resultados, se puede concluir que la representación en complemento a dos es mejor ya que no tiene dos representaciones para el cero, positivo y negativo, como ocurre con la representación en complemento a uno, y permite representar un número más (-4), que no se puede representar en complemento a uno con sólo tres bits.

En complemento a uno existen dos representaciones para el cero:

Cero Positivo: 000 Cero Negativo: 111

El cero negativo se obtiene de hallar el complemento a uno del cero positivo. De este modo, los circuitos aritméticos deberían tener en cuenta las dos representaciones del cero para su correcto funcionamiento y, por lo tanto, tendrían que realizar dos comprobaciones para el cero. En complemento a dos no ocurre ese problema ya que al sumar uno al complemento a uno del cero (111) se produce un desbordamiento y se pierde el bit más significativo, que no cabe si sólo se utilizan tres bits (1000). Así se volvería a obtener el cero (000).

5. Suponer que tenemos una máquina digital que trabaja con números enteros de *n bits*. Indicar el valor más grande y más pequeño que podemos representar con esos n bits usando las siguientes representaciones:

1. Sin signo.
2. Con signo, representando signo y magnitud.
3. Con signo, usando representación en complemento a uno.
4. Con signo, usando representación en complemento a dos.

Poner un ejemplo para cada caso suponiendo n = 3 e indicar la representación binaria de los valores extremos.

SOLUCIÓN:

1. Sin signo.

   En este caso sólo podemos representar números positivos, desde cero hasta 2^n - 1. Con tres bits podemos representar desde el cero (000) hasta el siete (111).

2. Con signo, representando signo y magnitud.

   En este caso, consideramos el primer bit, que es el bit más significativo (MSB), como bit de signo: Cero para números positivos y uno para números negativos. Los n - 1 bits restantes se utilizarán para representar la magnitud del número. Por lo tanto, podemos representar desde el $-(2^{n-1} - 1)$ hasta el $+(2^{n-1} - 1)$.

   Con tres bits, podemos representar desde el -3 (111) hasta el +3 (011). Obsérvese que con esta modalidad tenemos dos representaciones posibles para el cero, la negativa (100) y la positiva (000).

3. Con signo, usando representación en Complemento a uno.

   Al igual que en el caso anterior, podemos representar desde el $-(2^{n-1} - 1)$ hasta el $+(2^{n-1} - 1)$. La única diferencia es que ahora los números negativos (y sólo ellos) se representarán en complemento a uno.

   Con tres bits podemos representar desde el -3 (100) hasta el +3 (011). Obsérvese que con esta modalidad volvemos a tener dos representaciones posibles para el cero, la negativa (111) y la positiva (000).

4. Con signo, usando representación en complemento a dos.

   Esta es la representación más ventajosa, ya que nos permite representar un rango mayor de números y no tenemos dos representaciones para el cero. Podemos representar desde $-(2^{n-1})$ hasta el $+(2^{n-1} - 1)$. Al igual que en el complemento a uno, la representación en complemento a dos está reservada exclusivamente para los números negativos.

   Con tres bits podemos representar desde el -4 (100) hasta el +3 (011).

6. En los ordenadores se emplea siempre una representación en binario y normalmente se usa la representación en complemento a la base (complemento a dos) para los números negativos. Sin embargo, los mismos procedimientos empleados para realizar las operaciones en base dos pueden utilizarse en cualquier otra base. Aquí pretendemos comprobarlo usando una representación en base veinte (vigésima).

En base veinte tenemos que utilizar 20 símbolos, que son: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I y J. Estos símbolos corresponden a los valores comprendidos entre el cero y el diecinueve en base diez respectivamente.

Suponer una hipotética computadora que emplee representación en base veinte usando complemento a la base (complemento a veinte) para representar los números negativos. Se pide:

a) Representar el número $5018_{10)}$.
b) Representar el número $2319_{10)}$.
c) Representar el número negativo $-2319_{10)}$.
d) Realizar la siguiente resta utilizando la representación en complemento a la base: $5018_{10)} - 2319_{10)}$.

SOLUCIÓN:

a) Para representar un número decimal en cualquier otra base hay que realizar las divisiones sucesivas entre la base y luego tomar los restos en orden inverso al que se han obtenido. Si representamos con el símbolo "/" la operación división entera y por con el símbolo "%" la operación módulo (resto de la división entera) tenemos que:

5018 / 20 = 250;	5018 % 20 = 18;
250 / 20 = 12;	250 % 20 = 10;
12 / 20 = 0;	12 % 20 = 12;

Los restos sucesivos en orden inverso son 12, 10 y 18, los cuales se corresponden con los símbolos en base veinte C, A e I respectivamente.

Por tanto el número entero $5018_{10)}$ en base diez corresponde al número CAI en base veinte. Podemos comprobarlo fácilmente:

$$CAI = C * 20^2 + A * 20^1 + I * 20^0 =$$
$$12 * 400 + 10 * 20 + 18 * 1 =$$
$$4800 + 200 + 18 = 5018$$

b) La representación en base veinte del número 2319 se obtiene de manera similar:

2319 / 20 = 115;	2319 % 20 = 19;
115 / 20 = 5;	115 % 20 = 15;
5 / 20 = 0;	5 % 20 = 5;

Los restos sucesivos en orden inverso son 5, 15 y 19, los cuales se corresponden con los símbolos en base veinte 5, F e J respectivamente.

Por tanto, el número entero $2319_{10)}$ corresponde al número 5FJ en base veinte, lo cual puede comprobarse fácilmente:

$$5FJ = 5 * 20^2 + F * 20^1 + J * 20^0 =$$
$$5 * 400 + 15 * 20 + 19 * 1 =$$
$$2000 + 300 + 19 = 2319$$

c) Para representar el número negativo -2319 hay que hallar el complemento a la base de su valor absoluto expresado en base veinte. Como ya sabemos, éste último corresponde al número $5FJ_{20)}$. Así pues, para obtener el complemento a veinte hay que realizar la siguiente resta para calcular primero el complemento a la base menos uno (complemento a diecinueve):

$$JJJ - 5FJ = E40$$

Esta resta es simple, ya que se hace dígito a dígito y no puede haber acarreos. La letra J representa al número $19_{10)}$, la letra F al $15_{10)}$ y la E al $14_{10)}$.

Por último, para obtener la representación en complemento a veinte hay que sumar la unidad al resultado de la conversión en complemento a diecinueve:

$$E40 + 1 = E41$$

d) Al igual que ocurre en un ordenador binario, para realizar una resta se suma al minuendo el complemento a la base del sustraendo y posteriormente se desprecia el acarreo si existe. Así pues, para efectuar la resta 5018 - 2319 hay que sumar al número $5018_{10)}$ expresado en base veinte (CAI) el complemento a veinte del número $2319_{10)}$ (E41) y despreciar el acarreo de esa suma, que es el dígito más significativo del resultado:

$$CAI + E41 = 16EJ$$

Tras despreciar el acarreo obtenemos el resultado final, que se corresponde con el número en base veinte 6EJ.

Podemos verificar la operación efectuada representando su resultado en base diez y comprobando que coincide con el resultado de la resta 5018 - 2319, que es el número $2699_{10)}$.

$$6EJ = 6 * 20^2 + E * 20^1 + J * 20^0 =$$
$$6 * 400 + 14 * 20 + 19 * 1 =$$
$$2400 + 280 + 19 = 2699$$

7. Pasar a hexadecimal y octal (sin pasar a binario en ningún momento) el número $-5435_{10)}$ utilizando la notación de *complemento a la base*.

SOLUCIÓN:

a) Para obtener la representación en notación hexadecimal, el primer paso consiste en pasar a base dieciséis el valor absoluto del número.

Para representar un número decimal en cualquier otra base hay que realizar las divisiones sucesivas entre la base y luego tomar los restos en orden inverso al que se han obtenido. Si representamos con el símbolo "/" la operación división entera y por con el símbolo "%" la operación módulo (resto de la división entera) tenemos que:

5435 / 16 = 339;	5435 % 16 = 11;
339 / 16 = 21;	339 % 16 = 3;
21 / 16 = 1;	21 % 16 = 5;

El último cociente y los restos sucesivos, en orden inverso, son respectivamente 1, 5, 3 y 11.

Por tanto el número entero $5435_{10)}$ en base diez corresponde al número 153B en notación hexadecimal. Podemos comprobarlo fácilmente:

$$153B = 1 * 16^3 + 5 * 16^2 + 3 * 16^1 + 11 * 16^0 =$$
$$1 * 4096 + 5 * 256 + 3 * 16 + 11 * 1 =$$
$$4096 + 1280 + 48 + 11 = 5435$$

Para representar el número negativo -5435 hay que hallar el complemento a la base de su valor absoluto expresado en base dieciséis. Como ya sabemos, éste último corresponde al número $153B_{16)}$. Así pues, para obtener el complemento a dieciséis hay que realizar la siguiente resta para calcular primero el complemento a la base menos uno (complemento a quince):

$$FFFF - 153B = EAC4$$

Esta resta es simple, ya que se hace dígito a dígito y no puede haber acarreos. La letra F representa al número $15_{10)}$, la letra B al $11_{10)}$, la letra E al $14_{16)}$, la letra A al $10_{10)}$ y la C al $12_{10)}$.

Por último, para obtener la representación en complemento a dieciséis hay que sumar la unidad al resultado de la conversión en complemento a quince:

$$EAC4 + 1 = EAC5$$

b) La representación en base ocho del número 5435 se obtiene de manera similar:

5435 / 8 = 679;	5435 % 8 = 3;
679 / 8 = 84;	679 % 8 = 7;
84 / 8 = 10;	84 % 8 = 4;
10 / 8 = 1;	10 % 8 = 2;

El último cociente y los restos sucesivos, en orden inverso, son respectivamente 1, 2, 4, 7 y 3.

Por tanto el número entero $5435_{10)}$ en base diez corresponde al número 12473 en notación octal. Podemos comprobarlo fácilmente:

$$12473 = 1 * 8^4 + 2 * 8^3 + 4 * 8^2 + 7 * 8^1 + 3 * 8^0 =$$
$$1 * 4096 + 2 * 512 + 4 * 64 + 7 * 8 + 3 * 1 =$$
$$4096 + 1024 + 256 + 56 + 3 = 5435$$

Para representar el número negativo -5435 hay que hallar el complemento a la base de su valor absoluto expresado en base ocho. Como ya sabemos, éste último corresponde al número $12473_{8)}$. Así pues, para obtener el complemento a ocho hay que realizar la siguiente resta para convertirlo primero en complemento a la base menos uno (complemento a siete):

$$77777 - 12473 = 65304$$

Esta resta es simple, ya que se hace dígito a dígito y no puede haber acarreos.

Por último, para obtener la representación en complemento a ocho hay que sumar la unidad al resultado de la conversión en complemento a siete:

$$65304 + 1 = 65305$$

8. Supongamos que tenemos una máquina de *32 bits* que usa representación interna en complemento a 2. ¿Cómo representaría esta máquina el número 13? ¿Y el número -3? Expresar la solución en binario y hexadecimal.

SOLUCIÓN:

a) Tan sólo se trata de pasar el número 13 a binario y realizar una extensión en signo para obtener una representación en treinta y dos bits. El paso de binario a hexadecimal es inmediato.

En binario: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{2)}$
En hexadecimal: $0000\ 000D_{16)}$

b) El número 3 expresado en binario con treinta y dos bits es:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{2)}$$

De este modo, para obtener la representación del número -3 hay que obtener el complemento a dos del resultado anterior:

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{2)}$$

El paso de binario a hexadecimal también es inmediato: $FFFF\ FFF3_{16)}$

9. Dado un ordenador de *dieciséis bits* que representa internamente las cantidades en complemento a 2, indicar cuál será la representación interna de los siguientes números en dicho ordenador:

a) 32768
b) -32769
c) 32767
d) -32768

SOLUCIÓN:

a) y b) Ambos números no pueden representarse en complemento a 2 utilizando exclusivamente dieciséis bits. Con dieciséis bits y usando la notación de complemento a 2 sólo se puede representar el siguiente rango:

$$[-2^{16-1}, 2^{16-1} - 1] = [-32768, 32767]$$

c) Su representación interna es $0111\ 1111\ 1111\ 1111_{2)}$

d) Para representar un número en complemento a 2, el primer paso consiste en pasar el número expresado en base diez a binario natural prescindiendo del signo. Así se obtiene el número $1000\ 0000\ 0000\ 0000_{2)}$. Posteriormente, para obtener la representación en complemento a 1, se realiza la siguiente operación:

$$1\ 1111\ 1111\ 1111\ 1111_{2)} - 0\ 1000\ 0000\ 0000\ 0000_{2)} = 1\ 0111\ 1111\ 1111\ 1111_{2)}$$

Nótese como se añade provisionalmente un bit a la izquierda para no perder el signo negativo. Por último, para obtener la representación del número en complemento a 2 tan sólo hay que sumar la unidad a la representación en complemento a 1. Así se obtiene el número $1\ 1000\ 0000\ 0000\ 0000_{2)}$. No obstante, dado que el ordenador es de dieciséis bits debe prescindirse del bit más a la izquierda. Esto es posible debido a que el nuevo número aún conserva el signo negativo y sigue siendo una representación válida en complemento a 2. Por tanto, el bit quince pasa ahora a ser el más significativo y el resultado final es $1000\ 0000\ 0000\ 0000_{2)}$.

10. Contestar las siguientes cuestiones:

a) En un ordenador cuyo procesador es de dieciséis bits y utilizando complemento a dos, ¿Qué números en base diez tienen las siguientes representaciones? ¿Y si se usa complemento a uno?

1010 1011 1100 1101	1111 1111 1111 1111
1000 0000 0000 0000	0111 0111 0111 0111

b) Representar en BCD Empaquetado y Desempaquetado el siguiente número:

32038313

Establecer una método para representar el mismo número si es negativo y representarlo utilizando dicho método.

SOLUCIÓN:

a) En primer lugar debemos tener en cuenta el valor del bit de signo, ya que si el número es positivo tan sólo tenemos que pasarlo a base diez. Por el contrario, si el número es negativo, deberemos deshacer antes la operación de complemento a uno o complemento a dos según el caso.

En principio vamos a suponer que se ha utilizado la notación de complemento a dos. Entonces, si el número es negativo bastará con restarle la unidad e invertirlo bit a bit para obtener su valor absoluto. Finalmente deberemos pasar este último a base diez.

1010 1011 1100 1101 - 1 = 1010 1011 1100 1100
~ 1010 1011 1100 1100 = 0101 0100 0011 0011
$0101\ 0100\ 0011\ 0011_{2)} = 2^{14} + 2^{12} + 2^{10} + 2^5 + 2^4 + 2^1 + 2^0 = 21555_{10)}$
Por lo tanto, la representación del número $1010\ 1011\ 1100\ 1101_{2)}$ en base diez es $-21555_{10)}$.

1111 1111 1111 1111 - 1 = 1111 1111 1111 1110
~ 1111 1111 1111 1110 = 0000 0000 0000 0001
$0000\ 0000\ 0000\ 0001_{2)} = 2^0 = 1_{10)}$
Por lo tanto, la representación del número $1111\ 1111\ 1111\ 1111_{2)}$ en base diez es $-1_{10)}$.

1000 0000 0000 0000 - 1 = 0111 1111 1111 1111
~ 0111 1111 1111 1111 = 1000 0000 0000 0000
$1000\ 0000\ 0000\ 0000_{2)} = 2^{15} = 32768_{10)}$
Por lo tanto, la representación del número $1000\ 0000\ 0000\ 0000_{2)}$ en base diez es $-32768_{10)}$.

0111 0111 0111 0111 es un número positivo, pues su bit de signo vale cero. Por lo tanto, simplemente tenemos que pasarlo a base diez:
$0111\ 0111\ 0111\ 0111 = 2^{14} + 2^{13} + 2^{12} + 2^{10} + 2^9 + 2^8 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1 + 2^0 =$
$30583_{10)}$.

Supongamos ahora que se ha utilizado la notación de complemento a uno. En este caso, si el número es negativo bastará invertirlo bit a bit para obtener su valor absoluto. Finalmente deberemos pasar este último a base diez.

~ 1010 1011 1100 1101 = 0101 0100 0011 0010
$0101\ 0100\ 0011\ 0010_{2)} = 2^{14} + 2^{12} + 2^{10} + 2^5 + 2^4 + 2^1 = 21554_{10)}$
Por lo tanto, la representación del número $1010\ 1011\ 1100\ 1101_{2)}$ en base diez es $-21554_{10)}$.

~ 1111 1111 1111 1111 = 0000 0000 0000 0000
$0000\ 0000\ 0000\ 0000_{2)} = 0_{10)}$
Por lo tanto, la representación del número $1111\ 1111\ 1111\ 1111_{2)}$ en base diez es $0_{10)}$.

~ 1000 0000 0000 0000 = 0111 1111 1111 1111
$0111\ 1111\ 1111\ 1111_{2)} = 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 32767_{10)}$
Por lo tanto, la representación del número $1000\ 0000\ 0000\ 0000_{2)}$ en base diez es $-32767_{10)}$.

b) La representación del número 32038313 en BCD desempaquetado es la siguiente:

00000011 00000010 00000000 00000011 00001000 00000011 00000001 00000011

En BCD empaquetado se almacenan dos dígitos en cada byte:

00110010 00000011 10000011 00010011

Para representar en BCD un número negativo, bastará con usar una combinación de ceros y unos que no corresponda a ningún dígito codificado en BCD. Por ejemplo, podemos usar la combinación 00001111. No obstante, si el número en notación decimal tiene un número impar de cifras y queremos representarlo en BCD empaquetado, el byte más significativo puede tener la forma 1111XXXX, donde XXXX es la representación BCD de la cifra más significativa del número.

De este modo, la representación del número -32038313 en BCD desempaquetado es:

00001111 00000011 00000010 00000000 00000011 00001000 00000011 00000001 00000011

y en BCD empaquetado:

00001111 00110010 00000011 10000011 00010011

11. En un ordenador de *dieciséis bits* que usa representación interna en complemento a dos para los enteros, indicar qué números tienen las siguientes representaciones:

a) 1010 1011 1100 1101
b) 1000 0000 0000 0000

SOLUCIÓN:

a) Al pasar el número dado a hexadecimal se obtiene el número $ABCD_{16)}$. Como el primer bit de la representación binaria es uno, el número es negativo y por lo tanto hay que deshacer la operación de complemento a dos (o volver a complementar):

FFFF - (ABCD - 1) = FFFF - ABCD + 1 = 5432 + 1 = 5433

El resultado es 5433 en hexadecimal. Para pasarlo a decimal hay que multiplicar por las potencias relativas en base 16:

$5433)_{16} = 5 * 16^3 + 4 * 16^2 + 3 * 16^1 + 3 = 21555_{10)}$

Como el número era negativo, el resultado final es: **- 21555**.

b) Al pasar el número dado a hexadecimal se obtiene el número $8000_{16)}$. Como el primer bit de la representación binaria es uno, este número también es negativo y por lo tanto hay que deshacer la operación de complemento a 2 (o volver a complementar):

FFFF - (8000 - 1) = FFFF - 8000 + 1 = 7FFF + 1 = 8000

El resultado es 8000 en hexadecimal. Para pasarlo a decimal hay que multiplicar por las potencias relativas en base 16:

$$8000)_{16} = 8 * 16^3 + 0 * 16^2 + 0 * 16 + 0 = 32768_{10)}$$

Como el número era negativo, el resultado final es: **- 32768**.

12. Efectuar las siguientes operaciones matemáticas en base dos:

a) Multiplicar por $4_{10)}$ el número binario 110101001001.

b) Dividir por $8_{10)}$ el número binario 101001110010.

SOLUCIÓN:

a) El número decimal 4 corresponde al número 100 en binario. Multiplicar por la unidad seguida de ceros (potencias de dos en binario) es tan fácil como añadir tantos ceros como tenga este operando. En este caso basta con añadir dos ceros: 11010100100100.

b) Dado que el número decimal 8 corresponde al número binario 1000, para dividir por la unidad seguida de ceros basta con desplazar hacia la izquierda la coma decimal tantos lugares como ceros tenga el divisor. En este caso hay que desplazarla tres lugares: 101001110.01.

13. Suponer que tenemos un conjunto B con n elementos dotado con dos operaciones algebraicas que representaremos por + y *, denominadas respectivamente *suma* y *producto*, las cuales son distintas de la suma y producto de números enteros o reales.

a) Indicar las propiedades que debe cumplir dicho conjunto para que sea considerado un *álgebra de Boole*.

b) Poner un ejemplo de álgebra de Boole.

SOLUCIÓN:

a) Para que el conjunto B con las operaciones algebraicas (+, *) sea un álgebra de Boole debe verificar las siguientes cuatro propiedades:

Supongamos que tenemos tres elementos cualesquiera a, b y c pertenecientes al conjunto B.

1. Las operaciones son *conmutativas* para cualquier elemento de B.

$$a + b = b + a$$
$$a * b = b * a$$

2. Existen en B dos elementos distintos, denominados *elementos identidades* para cada una de las operaciones.

Identidad de la suma: Elemento 0.
Identidad del producto: Elemento 1.

$$a + 0 = 0 + a = a$$
$$a * 1 = 1 * a = a$$

Además:

$$a * 0 = 0 * a = 0$$
$$a + 1 = 1 + a = 1$$

3. Cada operación es distributiva respecto de la otra.

$$a + (b * c) = (a + b) * (a + c)$$
$$a * (b + c) = (a * b) + (a * c)$$

4. Todo elemento a perteneciente a B tiene su complementario a', tal que:

$$a + a' = 1$$
$$a * a' = 0$$

Nota: Es muy frecuente denotar el complementario de un elemento situando encima del elemento una raya horizontal que lo cubre.

b) El conjunto formado por dos elementos B = {0, 1} con las operaciones + y * (suma y producto) definidas en las siguientes tablas constituye un ejemplo válido de álgebra de Boole.

+	0	1
0	0	1
1	1	1

*	0	1
0	0	0
1	0	1

A este álgebra de Boole se le conoce con el nombre de *álgebra de conmutación* y es fundamental para el análisis y diseño de circuitos lógicos. En ellos cada elemento de este álgebra se representa por un nivel de tensión distinto, según la tecnología empleada.

En este contexto, la operación + se denomina *suma lógica* u operación binaria *OR* y la operación * es llamada *producto lógico* u operación binaria *AND*. El complemento se denomina a su vez *negación lógica* y se puede representar mediante la operación unaria *NOT*.

Existen otras operaciones lógicas que proceden de la composición de esas tres operaciones básicas. Por ejemplo, la operación *NAND* (NOT AND), la operación *NOR* (NOT OR) y la operación *XOR* (eXclusive OR).

14. Todo álgebra de Boole cumple la propiedad de *idempotencia*. Indíquese en qué consiste y compruébese para el álgebra de conmutación definido en el ejercicio anterior.

SOLUCIÓN:

Según la propiedad de idempotencia, para todo elemento x perteneciente a un álgebra de Boole B, se cumple que:

$$x + x = x$$
$$x * x = x$$

En particular, para el álgebra de conmutación el elemento x sólo puede tomar los valores cero o uno:

$$0 + 0 = 0$$
$$0 * 0 = 0$$

$$1 + 1 = 1$$
$$1 * 1 = 1$$

15. Todo álgebra de Boole cumple el *teorema de Morgan*. Indíquese en qué consiste y compruébese para el álgebra de conmutación definido anteriormente.

SOLUCIÓN:

Según el teorema de Morgan, para todo elemento b_i perteneciente a un álgebra de Boole con i = {1, 2, ..., n}, se cumple que:

1. $(b_1 + b_2 + ... + b_n)' = b_1' * b_2' * ... * b_n'$
2. $(b_1 * b_2 * ... * b_n)' = b_1' + b_2' + ... + b_n'$

Nota: El complementario de un elemento b_i lo hemos representado como b_i'.

Para comprobar el teorema basta con considerar todos los casos posibles. La comprobación de la primera parte es como sigue:

La suma de los n elementos será cero si todos los elementos son cero. El complementario de ese resultado es uno. Por otra parte, si todos los elementos son cero, el producto de sus complementarios valdrá uno, con lo que se cumple la igualdad

$$(0 + 0 + ... + 0)' = 1 * 1 * ... * 1 = 1$$

Otro caso es que no todos los elementos sean cero. Si un elemento vale uno, la suma total valdrá uno y su complementario cero. En la parte derecha de la igualdad tenemos que uno de los elementos del producto es el complementario de ese elemento. Si en un producto hay un cero el resultado total valdrá cero, con lo que se cumple también la igualdad

$$(b_1 + b_2 + ... + 1 + ... + b_n)' = b_1' * b_2' * ...* 0 * ... * b_n' = 0$$

La comprobación de la segunda parte se hace de modo similar:

El producto de los n elementos será uno si todos los elementos valen uno. El complementario de ese resultado es cero. En ese caso, en la parte derecha de la igualdad tenemos que sólo si todos los elementos valen uno la suma de sus complementarios valdrá cero, con lo que se cumple la igualdad

$$(1 * 1 * ... * 1)' = 0 + 0 + ... + 0 = 0$$

Por otra parte, si no todos los elementos valen uno es que existe al menos un elemento que vale cero. Así, el producto de la parte izquierda valdrá cero y su complementario uno. En la parte derecha tendremos el complementario de ese elemento que vale cero, que es uno. Por tanto, tendremos una suma en la que un operando vale uno, por lo que la suma valdrá uno independientemente del resto de operandos. Tenemos, por tanto, que se cumple la igualdad

$$(b_1 * b_2 * ... * 0 * ... * b_n)' = b_1' + b_2' + ... + 1 + ... + b_n' = 1$$

16. Realizar las siguientes operaciones:

a) Multiplicar por $4_{)10}$ el número binario 110101010111.

b) Dividir por $8_{)10}$ el número binario 1110101011111.

c) Hallar el valor de la siguiente expresión lógica:

```
1 AND ((0 OR NOT (1)) AND (1 OR 0))
```

SOLUCIÓN:

a) El número decimal 4 corresponde al número 100 en binario. Multiplicar por la unidad seguida de ceros (potencias de dos en binario) es tan fácil como añadir tantos ceros como tenga este operando. En este caso basta con añadir dos ceros: 11010101011100.

b) Dado que el número decimal 8 corresponde al número binario 1000, para dividir por la unidad seguida de ceros basta con desplazar hacia la izquierda la coma decimal tantos lugares como ceros tenga el divisor. En este caso hay que desplazarla tres lugares: 1110101011.111.

c) Para obtener el valor de la expresión conviene comenzar por los paréntesis más internos e ir resolviendo sucesivamente los de mayor nivel hasta hallar el resultado final.

```
1 AND ((0 OR NOT (1)) AND (1 OR 0)) =
    1 AND ((0 OR 0) AND (1 OR 0)) =
          1 AND (0 AND 1) =
             1 AND 0 = 0
```

17. Resolver la operación lógica `(a XOR a) OR b` para todos los posibles valores de `a` y `b`, sabiendo que $a, b \in \{0, 1\}$.

SOLUCIÓN:

La expresión `(a XOR a)` siempre tomará el valor cero, ya que tanto `(0 XOR 0)` como `(1 XOR 1)` valen cero. Además, si en una operación `OR` uno de los operandos es cero el resultado será igual al valor del otro operando. Por lo tanto se tiene que:

```
(a XOR a) OR b = (0 OR b) = b
```

18. Indicar cuál es el resultado de la siguiente expresión lógica, indicando la secuencia de operaciones realizadas y los resultados parciales.

```
((1 AND 0) OR (1 AND (NOT 0))) AND NOT (0 AND (NOT 1))
```

SOLUCIÓN:

Al igual que en ejercicios anteriores, para obtener el valor de la expresión conviene comenzar por los paréntesis más internos e ir resolviendo sucesivamente los de mayor nivel hasta hallar el resultado final.

```
((1 AND 0) OR (1 AND (NOT 0))) AND NOT (0 AND (NOT 1)) =
          (0 OR (1 AND 1)) AND NOT (0 AND 0) =
                (0 OR 1) AND NOT (0) =
                     1 AND 1 = 1
```

19. Hallar el valor de la siguiente expresión lógica:

```
(1 XOR 1) AND
( ((1 OR 0) AND (0 XOR 1)) XOR
  ( ((NOT 1) AND (0 XOR 0)) AND
    ( (1 XOR (0 OR 1)) OR
      (0 XOR (1 AND (0 XOR 1)))
    )
  )
)
```

SOLUCIÓN:

La solución es muy simple. La expresión anterior se puede escribir de este modo:

```
(1 XOR 1) AND ?
```

Como `(1 XOR 1)` es cero se obtiene la expresión `(0 AND ?)`, y una operación `AND` de un cero con cualquier otro valor lógico es siempre cero, por lo que el valor de la expresión global es cero.

Como se ve, la solución es más fácil de lo que puede parecer a simple vista, ya que no es necesario efectuar todas las operaciones que se indican.

20. Como es sabido, los operadores básicos de la lógica binaria son tres: `AND`, `OR` y `NOT`. Existen también otros operadores que se obtienen por composición de los anteriores, como son el operador `NAND`, que consiste en negar una operación `AND`, o el operador `NOR`, que es la negación de una operación `OR`. Otro es el operador `OR` exclusivo (`XOR`). El resultado de una operación `XOR` es uno si sus dos operandos son distintos y cero si son iguales. El ejercicio que se propone ahora consiste en expresar la operación lógica `(a XOR b)` en función de los tres operadores lógicos básicos.

SOLUCIÓN:

```
(a XOR b) = (a AND (NOT b)) OR ((NOT a) AND b)
```

21. Resolver las siguientes cuestiones:

a) Realizar la siguiente operación lógica indicando los resultados parciales:

```
(NOT ((1 AND 0) AND ((NOT 1) OR 0))) XOR (NOT 1)
```

b) Dada una computadora cuya palabra es de treinta y dos bits y que utiliza la notación de complemento a dos para realizar las operaciones aritméticas, se pide obtener la representación interna del número $-32768_{)10}$ para dicha computadora.

22. Contestar las siguientes cuestiones:

a) ¿Cómo afecta el *bit de paridad* a la *Eficiencia* y a la *Redundancia*?
b) ¿Para qué puede servir incluir un *bit de paridad* par en un *código de cuenta fija*?

SOLUCIÓN:

a) La eficiencia de un código (τ) es la relación que existe entre el número de símbolos que efectivamente se representan (m) y el número de símbolos que en total se pueden representar (2^n). El valor *n* es el número de bits que tiene el código.

El bit de paridad, ya sea ésta par o impar, es función del número de unos que posea la palabra del código. Se trata pues de un bit redundante. Si añadimos al código un bit de paridad, el número de palabras del mismo (m) no habrá variado. No obstante, el número total de símbolos que se pueden representar pasa a ser 2^{n+1}.

Por lo tanto, la nueva eficiencia, con relación a la antigua, tiene la siguiente expresión:

$$\tau' = \frac{m}{2^{n+1}}$$

De este modo puede decirse que τ es igual a $2 * \tau'$, con lo que la eficiencia original se ha visto reducida a la mitad.

La redundancia se define en base a la eficiencia mediante la expresión $r = 1 - \tau$. Dado que τ es estrictamente mayor que τ', tenemos que la redundancia del código con bit de paridad (r') resulta ser mayor que la original (r). Este resultado es lógico, pues el bit de paridad se define como un bit redundante, ya que es función de los restantes bits de la palabra del código y no aporta nada a la información a grabar o transmitir.

b) Introducir un bit de paridad en un código de cuenta fija no tiene ningún sentido. Los bits de paridad sirven para detectar posibles errores en la transmisión o grabación de información e indican si el número de unos en cada palabra del código es par o impar, según el tipo de paridad. En un código de cuenta fija el número de unos es constante por lo que es absurdo añadir un bit que indique si dicho número de unos es par o impar. Por lo tanto, en todas las palabras del código el valor del bit de paridad será siempre el mismo y no nos ayudará a detectar posibles errores.

23. Escribir un código de cuenta fija de cuatro bits para representar cinco símbolos distintos.

a) Hallar su eficiencia y redundancia.

b) Hallar la eficiencia y redundancia del mismo código introduciendo un bit de paridad. Razonar los resultados y obtener conclusiones.

SOLUCIÓN:

a) Los códigos de cuenta fija son aquéllos que tienen un número fijo de ceros y de unos. Si tenemos que construir un código de cuenta fija para representar cinco símbolos, es obvio que dicho código tendrá siempre dos unos y dos ceros. Otras posibilidades consisten en construir el código con tres unos y un cero o viceversa, pero éstas quedan descartadas ya que el código de cuenta fija así construido sólo permitiría representar cuatro símbolos y nosotros necesitamos cinco.

Con un código con las características deseadas podemos representar hasta seis símbolos distintos:

```
0011
1100
1001
0110
1010
0101
```

Si sólo necesitamos cinco tendremos que descartar alguna combinación de las anteriores. Por ejemplo la última:

```
0011
1100
1001
0110
1010
```

La eficiencia de un código (τ) es el cociente resultante de dividir el número de símbolos que se representan entre el número de símbolos que en total se pueden representar. En este caso, el número de símbolos que se representan es cinco y el número de símbolos que se pueden representar es, cómo en todos los códigos binarios, 2 elevado al número de bits del código, que en este caso es 2^4. Por tanto, la eficiencia es:

$$\tau = \frac{5}{2^4} = \frac{5}{16} = 0.3125$$

Esta cantidad está, como debe ser, comprendida entre cero y uno, y nos indica que es un código poco eficiente, por lo que será bastante redundante. La redundancia es:

$$R = 100\,(1-\tau) = 100\,(1-0.3125) = 68.75\%$$

b) Introducir un bit de paridad en un código de cuenta fija no tiene ningún sentido. Los bits de paridad sirven para detectar posibles errores en la transmisión o grabación de información e indican si el número de unos en cada palabra del código es par o impar. En un código de cuenta fija el número de unos es constante por lo que es absurdo añadir un bit que indique si dicho número de unos es par o impar. En todos los símbolos el valor del bit de paridad será siempre el mismo y por lo tanto no nos ayudará a detectar errores. Por ejemplo, si en el código anterior añadimos al final un bit de paridad siguiendo el criterio impar, resultará que todos los bits de paridad son iguales a uno:

```
0011 1
1100 1
1001 1
0110 1
1010 1
```

Al añadir el bit de paridad siempre introducimos redundancias en un código, pero a cambio de detectar algunos posibles errores. En este caso no tiene sentido añadir un bit de paridad por tratarse de un código de cuenta fija. Si calculamos la eficiencia y redundancia del código anterior vemos que es menos eficiente y por tanto más redundante:

$$\tau = \frac{5}{2^5} = \frac{5}{32} = 0.15625$$

$$R = 100\,(1-\tau) = 100\,(1-0.15625) = 84.375\%$$

Preguntas de Test

1. Un bit es

   a) la unidad mínima utilizada para medir la información.
   b) el número de bytes necesarios para almacenar un carácter.
   c) ocho bytes.
   d) Ninguna de las respuestas anteriores.

2. Un bit es

   a) una unidad para medir la velocidad de transmisión de la información en un ordenador.
   b) la unidad más pequeña de información que se puede manejar en un ordenador digital.
   c) un dígito binario.
   d) Son correctas las respuestas b) y c).

3. Un byte es

   a) la octava parte de un bit.
   b) la octava parte de un kilobyte.
   c) un conjunto aleatorio de bits.
   d) el hardware necesario para controlar el flujo de información en un sistema digital binario.
   e) el número de bits necesarios para almacenar un carácter.

4. Una palabra de n bits permite representar

   a) n cantidades distintas.
   b) 2n cantidades distintas.
   c) 2^n cantidades distintas.
   d) 2^{n-1} cantidades distintas.

5. El valor decimal del número binario 00000010 es

   a) uno en decimal.
   b) dos en decimal.
   c) diez en decimal.
   d) Depende de la base concreta que usemos.
   e) Ninguna de las respuestas anteriores es correcta.

6. El valor decimal del número binario 11010110 es

   a) 246.
   b) 214.
   c) 192.
   d) 65535.
   e) Ninguno de los anteriores.

7. El resultado de dividir $1101100100_{2)}$ entre $100_{2)}$ es

   a) 0011011001.
   b) 1101101011.
   c) 1101100111.
   d) 942310622.
   e) Ninguno de los anteriores.

8. El resultado de dividir el número binario 111000 entre el número en base diez 16 es

   a) 11100.
   b) 1110.
   c) 111.
   d) 11.1.
   e) Ninguna de las respuestas anteriores es correcta.

9. Suponer que en una determinada máquina tenemos almacenado en binario un número entero positivo. Duplicar ese número, suponiendo que no se produce desbordamiento, es equivalente a

   a) desplazar sus bits una posición hacia la izquierda, introduciendo un cero por la derecha como bit menos significativo.
   b) desplazar sus bits una posición hacia la derecha introduciendo un cero por la izquierda como bit más significativo.
   c) desplazar sus bits dos posiciones hacia la izquierda introduciendo dos ceros por la derecha como bits menos significativos.
   d) desplazar sus bits dos posiciones hacia la derecha introduciendo dos ceros por la izquierda como bits más significativos.
   e) efectuar una operación AND de dicho número consigo mismo.
   f) efectuar una operación OR de dicho número consigo mismo.
   g) Ninguna de las respuestas anteriores es correcta.

10. Suponer que en una determinada máquina tenemos almacenado en binario un número entero positivo. Cuadruplicar ese número, suponiendo que no se produce desbordamiento, es equivalente a

    a) desplazar sus bits una posición hacia la izquierda, introduciendo un cero por la derecha como bit menos significativo.
    b) desplazar sus bits una posición hacia la derecha introduciendo un cero por la izquierda como bit más significativo.
    c) desplazar sus bits dos posiciones hacia la izquierda introduciendo dos ceros por la derecha como bits menos significativos.
    d) desplazar sus bits dos posiciones hacia la derecha introduciendo dos ceros por la izquierda como bits más significativos.
    e) efectuar una operación AND de dicho número consigo mismo.
    f) efectuar una operación OR de dicho número consigo mismo.
    g) Ninguna de las respuestas anteriores es correcta.

11. Suponer que en una determinada máquina tenemos almacenado en binario un número entero positivo. Dividir ese número por dos, despreciando la parte decimal, es equivalente a

a) desplazar sus bits una posición hacia la izquierda, introduciendo un cero por la derecha como bit menos significativo.
b) desplazar sus bits una posición hacia la derecha introduciendo un cero por la izquierda como bit más significativo.
c) desplazar sus bits dos posiciones hacia la izquierda introduciendo dos ceros por la derecha como bits menos significativos.
d) desplazar sus bits dos posiciones hacia la derecha introduciendo dos ceros por la izquierda como bits más significativos.
e) efectuar una operación AND de dicho número consigo mismo.
f) efectuar una operación OR de dicho número consigo mismo.
g) Ninguna de las respuestas anteriores es correcta.

12. Suponer que en una determinada máquina tenemos almacenado en binario un número entero positivo. Dividir ese número por cuatro, despreciando la parte decimal, es equivalente a

a) desplazar sus bits una posición hacia la izquierda, introduciendo un cero por la derecha como bit menos significativo.
b) desplazar sus bits una posición hacia la derecha introduciendo un cero por la izquierda como bit más significativo.
c) desplazar sus bits dos posiciones hacia la izquierda introduciendo dos ceros por la derecha como bits menos significativos.
d) desplazar sus bits dos posiciones hacia la derecha introduciendo dos ceros por la izquierda como bits más significativos.
e) efectuar una operación AND de dicho número consigo mismo.
f) efectuar una operación OR de dicho número consigo mismo.
g) Ninguna de las respuestas anteriores es correcta.

13. El complemento a dos de un número

a) sólo tiene sentido en números binarios.
b) es una transformación usada para representar números negativos en un circuito digital.
c) permite representar un rango mayor que el complemento a uno.
d) Todas las respuestas anteriores son correctas.
e) Ninguna de las respuestas anteriores es correcta.

14. En una máquina de 3 bits, usando una representación en complemento a uno se pueden representar

a) 10 números distintos.
b) 9 números distintos.
c) 8 números distintos.
d) 7 números distintos.
e) Ninguna de las respuestas anteriores es correcta.

15. El BCD es

a) un sistema de numeración.
b) un sistema de codificación.
c) un código de detección de errores.
d) un código de corrección de errores.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

16. La representación del número 9758 en BCD empaquetado es

a) 1001011001011000.
b) 1010011101101000.
c) 1010011101010100.
d) 261E.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

17. La operación lógica OR es también conocida como

a) Suma Lógica.
b) Resta Lógica.
c) Producto Lógico.
d) División Lógica.
e) Operación Lógica Exclusiva.
f) Ninguna de las respuestas anteriores es correcta.

18. La operación lógica AND es también conocida como

a) Suma Lógica.
b) Resta Lógica.
c) Producto Lógico.
d) División Lógica.
e) Operación Lógica Exclusiva.
f) Ninguna de las respuestas anteriores es correcta.

19. Si suponemos que $X_1X_2X_3X_4$ es un dato de cuatro bits (cada dígito X_i equivale a un bit), ¿Qué resultado se obtiene si realizamos una operación AND bit a bit de este dato con el dato 1111?

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

20. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **AND 0000** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

21. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **AND** $\mathbf{X_1X_2X_3X_4}$ es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

22. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **OR 1111** es

a) $X_1X_2X_3X_4$
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$
f) Ninguna de las respuestas anteriores es correcta.

23. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **OR 0000** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

24. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **OR** $\mathbf{X_1X_2X_3X_4}$ es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

25. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **XOR 1111** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

26. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **XOR 0000** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

27. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **XOR** $\mathbf{X_1X_2X_3X_4}$ es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

28. La operación NAND (NOT AND) consiste en realizar en primer lugar la operación AND y después negar el resultado obtenido. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **NAND 1111** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

29. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **NAND 0000** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

30. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **NAND** $\mathbf{X_1X_2X_3X_4}$ es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

31. La operación NOR (NOT OR) consiste en realizar en primer lugar la operación OR y después negar el resultado obtenido. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **NOR 1111** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

32. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **NOR 0000** es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

33. Si $X_1X_2X_3X_4$ es un dato de cuatro bits, el resultado de la operación bit a bit $\mathbf{X_1X_2X_3X_4}$ **NOR** $\mathbf{X_1X_2X_3X_4}$ es

a) $X_1X_2X_3X_4$.
b) 1111.
c) 1010.
d) 0000.
e) El complemento a uno de $X_1X_2X_3X_4$.
f) Ninguna de las respuestas anteriores es correcta.

34. Si $X_1X_2X_3X_4$ es un dato de cuatro bits y deseamos obtener su complemento a uno, ¿Cuál de las siguientes operaciones nos permiten conseguir nuestro propósito?

a) $X_1X_2X_3X_4$ OR $X_1X_2X_3X_4$.
b) $X_1X_2X_3X_4$ XOR 0000.
c) $X_1X_2X_3X_4$ XOR 1111.
d) $X_1X_2X_3X_4$ AND 1111.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

35. En la programación en ensamblador es muy frecuente querer dejar a valor cero un determinado registro del microprocesador. Supongamos que queremos dejar a cero el registro AX. Esto se puede conseguir de muchas formas (por ejemplo mediante la instrucción MOV AX, 0). Sin embargo, la manera más rápida y eficiente consiste en efectuar una de las operaciones siguientes. Indicar de cuál se trata.

a) AX AND AX
b) AX OR AX
c) AX XOR AX
d) AX NAND AX
e) AX NOR AX
f) Todas las operaciones anteriores obtienen el resultado deseado.
g) Ninguna de las operaciones anteriores obtiene el resultado deseado.

36. La eficiencia de un código es

a) el número de símbolos que se pueden representar.
b) el número de símbolos que realmente se representan.
c) la relación entre el número de símbolos que se pueden representar y el número de símbolos que se representan.
d) un número real comprendido entre cero y uno.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

37. Dado un código de n bits utilizado para representar m símbolos, su eficiencia es

a) $m / 2^m$.
b) $m / 2^n$.
c) $m * 2^n$.
d) 2^n.
e) 2^m.
f) Ninguna de las respuestas anteriores es correcta.

38. Si un código tiene una eficiencia e, su redundancia R se define mediante la expresión

a) R = (1 - e) %.
b) R = (e - 1) %.
c) R = 100 (1 - e) %.
d) R = 100 (e - 1) %.
e) R = 100 e %.
f) Ninguna de las respuestas anteriores es correcta.

39. El código de paridad

a) es un código eficiente.
b) es un código de detección y corrección de errores.
c) incrementa la velocidad de transmisión de datos.
d) introduce redundancias para detectar errores en la transmisión o grabación de la información.

40. El código de paridad, usado en algunas memorias RAM o en transmisiones, sirve para

a) verificar la verdad o falsedad de la información almacenada o transmitida.
b) verificar si funcionan correctamente la memoria y la CPU.
c) comprobar el estado de los periféricos conectados.
d) detectar los posibles errores en la transmisión o almacenamiento de la información.

41. El código de paridad se utiliza para

a) averiguar si un número es par.
b) corregir errores en la transmisión de datos.
c) detectar errores en la transmisión de datos.
d) detectar y corregir errores en la transmisión de datos.
e) Ninguna de las funciones anteriores.

42. El código de paridad con criterio par

a) suma uno a los números impares para convertirlos en pares.
b) añade un bit a los números impares dejando intactos los pares.
c) añade un bit de forma que el número de unos del dato a transmitir sea par.
d) añade un bit de forma que el número de ceros del dato a transmitir sea par.
e) añade un bit de forma que el número de unos más el de ceros del dato sea par.
f) Ninguna de las respuestas anteriores es correcta.

43. El código ASCII es

a) un sistema operativo de uso extendido.
b) un hardware controlador de dispositivos.
c) un estándar americano para codificar caracteres y poder comunicar e intercambiar información.
d) un software controlador del ratón que permite codificar los caracteres a los que éste apunta.
e) el código utilizado para almacenar información en el almacenamiento intermedio, que es la memoria caché.
f) Ninguna de las respuestas anteriores es correcta.

44. El ASCII es un código de

a) corrección de errores.
b) detección de errores.
c) entrada/salida.
d) carga y almacenamiento.
e) Ninguna de las respuestas anteriores es correcta.

Respuestas

1. a)
2. d)
3. e)
4. c)
5. b)
6. b)
7. a)
8. d)
9. a)
10. c)
11. b)
12. d)
13. d)
14. d)
15. b)
16. c)
17. a)
18. c)
19. a)
20. d)
21. a)
22. b)
23. a)
24. a)
25. e)
26. a)
27. d)
28. e)
29. b)
30. e)
31. d)
32. e)
33. e)
34. c)
35. c)
36. d)
37. b)
38. c)
39. d)
40. d)
41. c)
42. c)
43. c)
44. c)

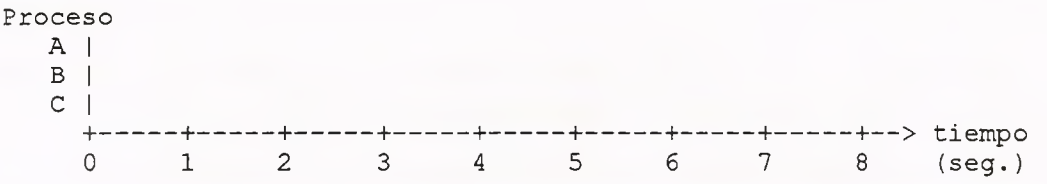
Capítulo 3

Sistemas Operativos.

1. Supongamos un microprocesador de 64 bits a 100 Mhz, bajo un *sistema operativo a tiempo compartido*, con un quantum de 1 seg. Teniendo en cuenta la tabla de procesos adjunta, explicar y detallar qué proceso se ejecuta en cada instante desde el segundo cero.

Proceso	Tiempo de llegada	Tiempo de CPU requerido
A	0.5 seg.	2 seg.
B	1 seg.	1.5 seg.
C	1.25 seg.	2 seg.

Supondremos además que el proceso A necesita efectuar una operación de Entrada/Salida de 2 seg. de duración cuando lleva ejecutándose 1.5 seg.



Nota: Los tres procesos A, B y C son igual de prioritarios. No obstante, a la hora de la ejecución de éstos, se otorgará preferencia al último en llegar.

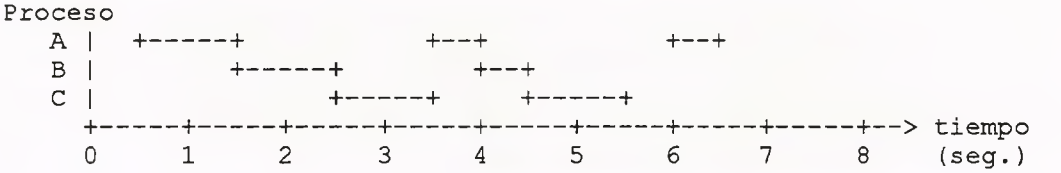
SOLUCIÓN:

En primer lugar, a los 0.5 segundos comienza a ejecutarse el proceso A, que es el primero en llegar. Una vez agotado su quantum, comienza a ejecutarse el proceso B a los 1.5 segundos. Concluido su quantum, comienza a ejecutarse a los 2.5 segundos el proceso C, aunque éste ya había llegado al sistema a los 1.25 segundos y fue el último en llegar.

Acabado el quantum del proceso C, el proceso A vuelve al estado de ejecución. Sin embargo, cuando lleva 1.5 segundos ejecutándose, debe realizar una operación de entrada/salida. Dicha operación, por tanto, tiene lugar cuando se alcanza el segundo 4.

Como A se encuentra bloqueado, el sistema operativo reactiva al proceso B, el cual a los 0.5 segundos finaliza su ejecución. Esto ocurre en el instante 4.5 seg. Entonces el proceso C vuelve al estado de ejecución, la cual finaliza transcurrido un segundo, en el instante 5.5 seg.

Llegado ese instante sólo queda el proceso A por finalizar, pero todavía le queda medio segundo para terminar su operación de entrada/salida y desbloquearse, pues ésta dura dos segundos. Esto ocurre cuando se alcanza el segundo 6. Entonces el proceso A continúa su ejecución y termina transcurrido medio segundo, en el instante 6.5 seg.

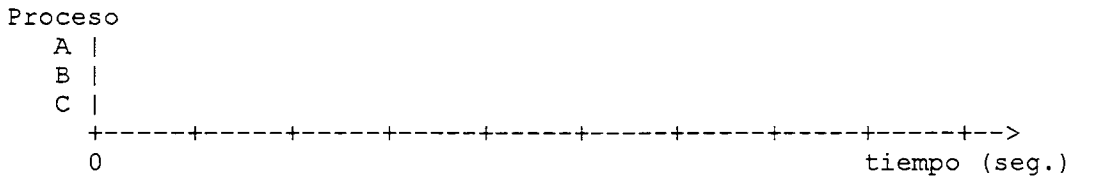


2. Supongamos un ordenador con un microprocesador de 64 bits a 150 Mhz. que funciona bajo un *sistema operativo a tiempo compartido* con un quantum de 1 seg.

a) Teniendo en cuenta la tabla siguiente, explicar y detallar qué proceso se ejecuta en cada instante (a partir del segundo cero).

Proceso	Tp llegada	Tp CPU	E/S a los	Durac. E/S
A	0.5 seg.	2 seg.	1.5 seg.	3 seg.
B	1.25 seg.	4 seg.	1.5 seg.	2 seg.
C	2 seg.	1 seg.	0.5 seg.	1.5 seg.

La primera columna indica el nombre del proceso, la segunda el tiempo de llegada al sistema. El tiempo de CPU indica el que necesita el proceso para ejecutarse totalmente. Además, cada proceso requiere efectuar un operación de Entrada/Salida en el tiempo de ejecución (relativo al comienzo de ésta) que se indica en la cuarta columna, con la duración que se indica en la quinta columna.



NOTA: Los tres procesos A, B y C son igual de prioritarios. Si un proceso ha llegado al sistema, pero no ha comenzado su ejecución, el S.O. le otorga la prioridad máxima.

b) ¿Puede haber tiempos muertos de CPU en este sistema? Razonar la respuesta.

SOLUCIÓN:

a) En el segundo 0.5 llega el primer proceso al sistema, que es A. Agotado su quantum, el proceso B comienza su ejecución en el instante 1.5 seg., aunque ya había llegado al sistema un cuarto de segundo antes. Cuando éste último agota su quantum, en el instante 2.5 seg., el sistema operativo activa el proceso C, que llegó al sistema medio segundo antes.

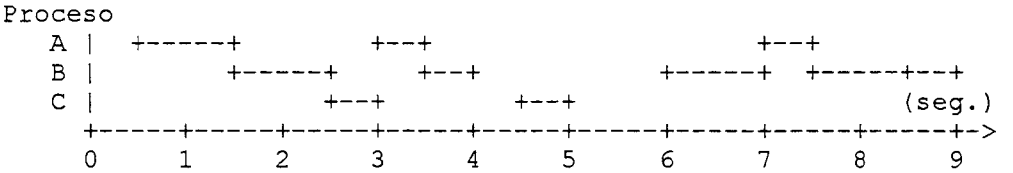
A los 0.5 seg. el proceso C comienza una operación de entrada/salida y el sistema operativo reactiva entonces al proceso A, el cual continúa ejecutándose durante 0.5 seg. En ese instante (3.5 seg.) el proceso A precisa realizar otra operación de entrada/salida. Entonces, el sistema operativo reactiva al proceso B, el cual también necesita realizar también una operación de entrada/salida a los 0.5 seg. de su reactivación.

Por lo tanto, durante un intervalo de medio segundo a partir del instante 4 seg., el sistema operativo no podrá activar ningún proceso, pues todos se encuentran *bloqueados* a la espera de que finalice su operación de entrada/salida.

En el instante 4.5 seg. el proceso C concluye su operación de entrada/salida y el sistema operativo lo reactiva. Sin embargo, éste sólo utiliza medio segundo de su quantum, pues acaba su ejecución en el segundo 5. De nuevo, el sistema operativo no puede reactivar ninguno de los dos procesos restantes, pues tanto A como B continúan realizando su operación de entrada/salida.

En el segundo 6, el proceso B concluye su operación de entrada/salida y el sistema operativo puede reactivarlo. Cuando acaba su quantum, en el instante 7 seg., el proceso A se encuentra en condiciones de continuar su ejecución, dado que terminó su operación de entrada/salida en el instante 6.5 seg. Por tanto, el proceso A continúa ejecutándose durante medio segundo, pues en ese instante acaba su ejecución.

Por último, el sistema operativo reactiva el proceso B en el instante 7.5 seg., pues es el único que queda por acabar. Sin embargo, cuando concluye su quantum, el sistema operativo no lo deja en estado *suspendido*, puesto que es el único proceso que se está ejecutando. Por lo tanto, continúa su ejecución durante medio segundo más hasta el instante 9 seg., que acaba por fin su ejecución.



b) Sí, los tiempos muertos de CPU tienen lugar cuando *todos* los procesos existentes en el sistema se encuentran bloqueados, ya sea debido a un interbloqueo, o bien por encontrarse realizando operaciones de entrada/salida.

En principio, este sistema es muy simple y no se contempla la asignación de recursos ni, por supuesto, el interbloqueo. Sin embargo, la aparición de tiempos muertos debido a operaciones simultáneas de entrada/salida si puede ocurrir, y de hecho tiene lugar en el sistema. Concretamente, en la ejecución anterior los tiempos muertos corresponden a los intervalos [4 seg., 4.5 seg.] y [5 seg., 6 seg.].

3 Supongamos un ordenador con 3 MB de memoria RAM gestionada mediante un S.O. por Particiones Fijas de la siguiente forma:

- Una partición de 1 MB.
- Dos particiones de 512 KB.
- Una partición de 624 KB.
- Una partición de 400 KB.

Los siguientes programas se ejecutan en los tiempos y con los requisitos de memoria que a continuación se indican:

Programa	Tiempo de llegada	Tamaño	Duración
A	0 seg.	500 KB	10 seg.
B	1 seg.	550 KB	2 seg.
C	2 seg.	300 KB	7 seg.
D	3 seg.	550 KB	5 seg.
E	4 seg.	500 KB	4 seg.
F	5 seg.	100 KB	3 seg.
G	7 seg.	25 KB	3 seg.
H	8 seg.	550 KB	1 seg.

Especificar de forma gráfica qué partición está ocupada en cada momento y por cuál programa. Según el ejemplo propuesto, discutir la eficiencia del Método de Gestión de Memoria utilizado.

SOLUCIÓN:

Representaremos la memoria a gestionar del siguiente modo:

1 MB
512 KB
512 KB
624 KB
400 KB

Para simplificar, no se tendrá en cuenta el tamaño de las particiones en la representación.

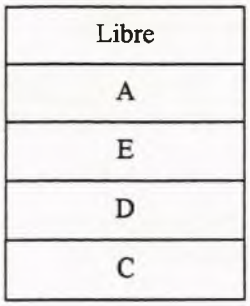
Inicialmente la memoria está vacía. De este modo, a la hora de ejecutar el programa A de 500 KB el S.O. le asignará la partición más pequeña que pueda contenerlo, es decir, una de las dos de 512 KB:

Libre
A
Libre
Libre
Libre

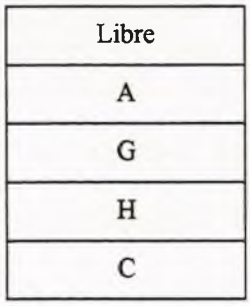
En el segundo 1 llega el proceso B y en el 2 llega el C, a los cuales el S.O. les asigna las particiones de 624 KB y 400 KB respectivamente:

Libre
A
Libre
B
C

En el segundo 3 se libera la memoria que ocupaba el proceso B, ya que éste termina a los 2 segundos de su ejecución. En ese momento llega el proceso D, que se mete en la misma partición que libera el proceso B. Un segundo después llega el proceso E, el cual se introduce en la partición de 512 KB que está libre:



En el segundo 5 llega el proceso F y se introduce en la partición más grande, por ser la única que está libre, desperdiciando así mucha memoria de esa partición de 1 MB. En el segundo 7 llega el proceso G, que a pesar de su escaso tamaño y de haber suficiente memoria libre para él, no se puede ejecutar por estar todas las particiones ocupadas. En el segundo 8 se liberan 3 particiones, de manera que el proceso G pasa a ejecutarse en la de 512 KB y el recién llegado H pasa a la partición de 624 KB:



En el segundo 9 se liberan las particiones de los procesos H y C. En el segundo 10 se libera la partición del proceso A y en el segundo 11 la del proceso G.

Vamos a representar la ocupación de la memoria de otra forma, considerando que cada fila es una partición y que cada columna es un segundo, en el cual cada partición está ocupada por el proceso que se indica con su letra correspondiente. Las particiones libres están en blanco.

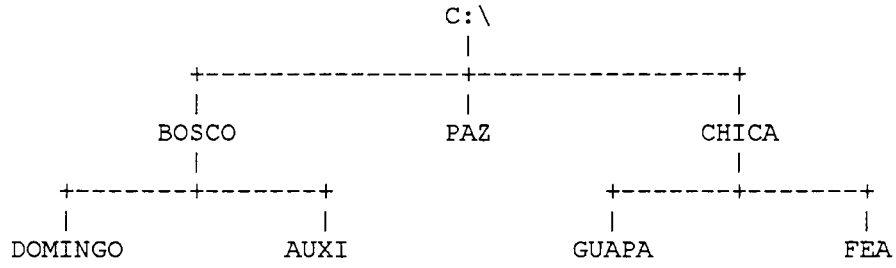
```
Partición
---------
  1 MB                        F  F  F
512 KB     A  A  A  A  A  A  A  A  A  A
512 KB                    E  E  E  E  G  G  G
624 KB        B  B  D  D  D  D  D  H
400 KB           C  C  C  C  C  C  C
-------------------------------------------
Segundo    0  1  2  3  4  5  6  7  8  9  10 11
```

El método de asignación de memoria por Particiones Fijas no se utiliza por ser poco eficiente, pues desperdicia muchísima memoria, lo cual puede comprobarse observando la gestión de memoria realizada en el ejemplo propuesto. En la actualidad se utilizan otros métodos más avanzados, como la gestión por Montón o Poll de Memoria. En este método la memoria se mantiene sin particionar, y a cada proceso el Sistema Operativo le asigna la cantidad de memoria necesaria para su ejecución. Por lo tanto no se desperdicia memoria, pero los mecanismos de protección de las zonas de memoria ocupadas por los procesos son más complejos, ya que los límites de las particiones varían con el tiempo.

3. Dar la secuencia de órdenes necesaria para conseguir que un Ordenador Personal bajo el Sistema Operativo MS-DOS ejecute las siguientes acciones:

Nota: Suponemos que acabamos de arrancar el ordenador y que el indicador de MS-DOS es `C:\`. Igualmente se supone que las órdenes serán emitidas una detrás de otra, teniendo en cuenta los efectos de las anteriores.

a) Crear la siguiente *estructura de directorios*: (omitimos el nombre completo de cada subdirectorio con para evitar redundancias)



b) Copiar todos los ficheros del directorio `FEA` (suponiendo que alguien haya creado archivos en dicho directorio), que empiecen por `FEA`, les siga un sólo carácter alfanumérico y cuya extensión termine por `XT` al directorio `AUXI`.

c) Mover todos los ficheros del subdirectorio `PAZ` al subdirectorio `BOSCO`.

d) Borrar el subdirectorio `CHICA` completamente, es decir, borrar todo su contenido y el propio subdirectorio.

SOLUCIÓN:

El ejercicio propuesto puede resolverse de muchas maneras, aquí tan sólo proponemos algunos ejemplos:

a) Utilizando rutas de acceso absolutas:

```
md \bosco
md \bosco\domingo
md \bosco\auxi
md \paz
md \chica
md \chica\guapa
md \chica\fea
```

Usando rutas de acceso relativas:

```
md bosco
cd bosco
md domingo
md auxi
cd ..

md paz

md chica
cd chica
md guapa
md fea
cd ..
```

b) Utilizando rutas de acceso absolutas:

```
copy \chica\fea\fea?.?xt \bosco\auxi
```

Usando rutas de acceso relativas:

```
cd bosco\auxi
copy ..\..\chica\fea\fea?.?xt
cd ..\..
```

c) Utilizando rutas de acceso absolutas:

```
move \paz\*.* \bosco
```

Usando rutas de acceso relativas:

```
cd bosco
move ..\paz\*.* .
cd ..
```

d) Utilizando rutas de acceso absolutas:

```
del \chica\guapa\*.*
rd \chica\guapa

del \chica\fea\*.*
rd \chica\fea

del \chica\*.*
rd chica
```

Usando rutas de acceso relativas:

```
cd chica

del guapa\*.*
rd guapa

del fea\*.*
rd fea

del *.*
cd ..
rd chica
```

4. Explicar las acciones que lleva a cabo el ordenador al ejecutar la siguiente orden:

```
TYPE LISTA.TXT | SORT > PRN
```

SOLUCIÓN:

La orden anterior imprime el contenido del fichero de texto `lista.txt` ordenado alfabéticamente.

Expliquemos como funciona la orden paso a paso:

a) La primera parte de la orden (`type lista.txt`) muestra el contenido del fichero por la salida estándar, que suele ser la pantalla.

b) El filtro `sort` muestra ordenadamente la salida producida por el comando antepuesto a la *tubería*. Así, las líneas del fichero `lista.txt` aparecerán ordenadas alfabéticamente por la salida estándar.

c) El operador de *redirección* > establece como nueva salida estándar la impresora. Es decir, redirige la salida al dispositivo de impresión (`prn`).

5. Contestar de forma breve pero completa las siguientes cuestiones:

a) Explicar cómo funcionaría, en caso de que fuese correcta, la siguiente orden en MS-DOS.

```
c:\system\dos> type ..\unix.doc | sort > prn
```

"`c:\system\dos>`" es el símbolo de atención de MS-DOS.

b) Explicar cómo funcionaría, en caso de que fuese correcta, la siguiente orden en MS-DOS.

```
diskcopy a: c:\windows\*.*
```

c) Explicar cómo funcionaría, en caso de que fuese correcta, la siguiente orden en MS-DOS.

```
xcopy a:\oswarp\*.* c:\windows /s
```

SOLUCIÓN:

a) Esta orden imprime las líneas del fichero de texto `unix.doc`, el cual se encuentra en el subdirectorio `\system` de la unidad `c:`, ordenadas alfabéticamente.

Mediante una *tubería* se conecta la salida de la suborden `type ..\unix.doc` con la entrada del filtro `sort`. De este modo, el filtro muestra las líneas del fichero `unix.doc` en orden alfabético por la salida estándar. Por último, ésta es redirigida al archivo de dispositivo `prn`, que hace referencia a la impresora.

b) El comando `diskcopy` copia dos disquetes de idéntico formato pista a pista. Por lo tanto la orden de este apartado no es correcta, ya que el segundo parámetro debe ser una letra de unidad de disquete seguida del carácter dos puntos, y no los ficheros pertenecientes a una ruta de acceso del disco duro `c:`.

c) La orden copia todos los ficheros pertenecientes al subdirectorio `oswarp` de la unidad `a:` al subdirectorio `windows` de la unidad `c:`.

Además, en virtud del parámetro `/s` se copia la jerarquía de subdirectorios descendientes de `oswarp` dentro del subdirectorio `windows`, exceptuando aquellos directorios que están vacíos.

6. Supongamos que tenemos un disco duro muy lento y necesitamos ejecutar un programa que accede mucho (en lectura y escritura) al disco. ¿Qué soluciones se le ocurren para conseguir acelerar la ejecución del programa? ¿Existe alguna manera de aplicar estas soluciones en un ordenador que funcione bajo el Sistema Operativo MS-DOS? En caso afirmativo, ¿Cuáles son?

SOLUCIÓN:

La solución consiste en sustituir, en la medida de lo posible, la mayor parte de los accesos a discos por accesos a memoria RAM, pues estos últimos son del orden de nanosegundos, mientras que los primeros tardan milisegundos. Esta solución general se puede poner en práctica de dos modos:

a) Utilizando un *disco RAM* o *disco virtual*. Se trata de un disco simulado en la memoria RAM por el Sistema Operativo. El programa a ejecutar junto con los ficheros de datos que utilice puede copiarse en el disco RAM para su ejecución, y posteriormente, los ficheros de salida generados deberán copiarse en el disco duro en el lugar adecuado, ya que la información contenida en el disco RAM desaparece tras apagar el ordenador. Recuérdese que la memoria RAM es volátil.

b) Haciendo uso de una *caché de disco*, que es una zona de memoria RAM gestionada por un programa denominado *controlador de caché*. Dicho programa funciona del siguiente modo:

   Cuando se realiza un acceso en lectura, el controlador de caché busca el dato en la memoria RAM. Si no lo encuentra allí decimos que se ha producido un *fallo de caché*. Entonces el controlador busca el dato en el disco duro, y una vez localizado transfiere a memoria un bloque completo de información, en el cual se encuentra dicho dato. Es probable que el siguiente dato a leer se encuentre localmente próximo al anterior, y por lo tanto esté dentro del bloque de información que se transfirió antes a la memoria RAM. En ese caso, como el controlador de caché busca primero el dato en la memoria RAM, tendrá lugar un *acierto de caché* y nos ahorraremos el acceso al disco duro.

   En los accesos a escritura, el controlador de caché escribe el dato generado por el programa en la memoria RAM, y a su vez indica a éste último que su dato ya ha sido escrito. De este modo, el programa puede seguir su ejecución y en el momento oportuno el controlador transferirá el dato de la memoria RAM al disco duro.

El Sistema Operativo MS-DOS suministra un controlador de disco RAM cuyo nombre es `RAMDRIVE.SYS` y un controlador de caché de disco denominado `SMARTDRIVE.EXE`. El disco RAM puede crearse en memoria convencional o extendida, pero la caché de disco sólo puede crearse en la memoria extendida. La caché puede funcionar tanto con discos duros como con discos flexibles.

7. Imaginemos que tenemos mil disquetes vírgenes, en los cuales queremos copiar en su directorio raíz el contenido del subdirectorio `C:\GREEN`. Escribir un archivo de ejecución por lotes (batch) para el sistema operativo MS-DOS el cual, de forma sucesiva, vaya dando formato a cada disquete en la unidad `A:` y copie en ellos dicho directorio, de forma que el usuario sólo tenga que reemplazar el disquete cuando lo indique el programa.

   Además existe el directorio `C:\GREEN\PEACE`, cuyos archivos también se deben copiar a un subdirectorio de cada disco flexible, cuyo nombre (una palabra) se suministrará al programa como parámetro.

   El archivo batch debe indicar en cada momento qué operación se está realizando a través de mensajes por pantalla y debe incluir comentarios que clarifiquen el programa. Además se debe indicar el nombre de dicho archivo.

   Nota: Podemos suponer que usamos disquetes de la densidad adecuada para el tipo de unidad.

   SOLUCIÓN:

   Los archivos batch, para que se comporten como tales, deben tener siempre la extensión `.BAT`. El nombre para este archivo de ejecución por lotes puede ser, por ejemplo, `COPYDISK.BAT`.

```
@echo off
rem Archivo Batch: COPYDISK.BAT
rem Autores: José María, María José y Pp.
rem Fecha: 28-10-1996
rem Programa para copiar determinados archivos en un
rem disquete en la unidad A: dándole formato previamente.
rem Formato de la llamada: COPIDISK direct
rem donde "direct" es el nombre de un directorio que se
rem creará en A:\ y donde se copiarán determinados archivos.
echo Este programa copia archivos en disquetes dándoles formato.
:INICIO
echo Pulsa Ctrl-C para acabar o
echo Introduce un disquete en A: para dar formato y copiar archivos.
pause
echo Dado formato al disco...
FORMAT A: /V:COPIDISK
echo Disco con formato. Copiando archivos de C:\GREEN a A:\
COPY C:\GREEN\*.* A:\
echo Copiando archivos de C:\GREEN\PEACE a A:\%1
MD A:\%1
COPY C:\GREEN\PEACE\*.* A:\%1
echo ***** Disquete procesado. *****
GOTO INICIO
```

Con la opción `/V` del comando `format` podemos asignar una etiqueta de volumen al disquete. De esta forma evitamos que el usuario tenga que introducir una etiqueta cada vez que se le da formato a un disquete.

El archivo batch se ejecutará un número indeterminado de veces. Cuando ya no deseemos dar formato y copiar más archivos tendremos que pulsar la combinación de teclas `<Ctrl-C>` para acabar.

El primer y único argumento de este archivo batch debe ser el nombre del directorio que deseamos crear en `A:\`. A este nombre nos referimos en el archivo batch como `%1`, ya que es el primer argumento. Dejamos como ejercicio propuesto modificar el archivo anterior para que presente un mensaje de error si este argumento no se incluye en la línea de comandos de MS-DOS.

8. Escribir un archivo de ejecución por lotes para MS-DOS que nos permita borrar ficheros de forma selectiva. Es decir, que nos permita borrar todos los ficheros cuyos nombre cumplen un determinado patrón (con caracteres comodín) excepto los que cumplan otro patrón concreto. Ejemplo: Imaginemos que se desea borrar de un directorio determinado todos los ficheros menos los que tengan la extensión TXT.

El archivo batch tendrá dos argumentos, que serán sendos patrones de nombres de ficheros. Para simplificar supondremos que no incluyen la ruta de acceso (path). El archivo batch deberá borrar del directorio actual todos los ficheros indicados en el primer argumento menos los que se indiquen en el segundo. En el ejemplo anterior, el primer argumento será `*.*` y el segundo `*.TXT`, borrando así todos los ficheros excepto los que tengan la extensión TXT.

En cada paso, el archivo batch deberá presentar por pantalla un mensaje indicando lo que está haciendo cada comando. Antes de efectuar el borrado, el programa indicará su cometido y pedirá confirmación para continuar. Se deberá desactivar el eco al principio y se deben incluir comentarios indicando autor, fecha y nombre completo del archivo batch en cuestión.

Indicar además cuál es la línea de comando que hay que suministrar al sistema operativo para que el archivo batch borre del directorio actual todos los ficheros menos aquéllos que tienen extensión TXT (ejemplo anterior).

SOLUCIÓN:

```
@echo off
rem Archivo Batch: MIBORRA.BAT
rem Autores: José María, María José y Pp.
rem Fecha: 28-10-1996
echo Programa para borrado selectivo:
echo Borra del directorio actual los archivos indicados en el primer
echo argumento (%1) menos los indicados en el segundo argumento (%2).
echo Pulse <Ctrl-C> para acabar u otra tecla para continuar.
Pause
echo Creando directorio auxiliar...
MD \MIBORRA
echo Moviendo ficheros con patrón %1 al directorio auxiliar...
MOVE %1 \MIBORRA
echo Moviendo ficheros con patrón %2 al directorio actual...
MOVE \MIBORRA\%2 .
echo Borrando directorio auxiliar y sus ficheros...
DELTREE \MIBORRA /Y
echo ***** Borrado selectivo completado *****
```

El archivo batch crea un directorio temporal a donde mueve los ficheros cuyo patrón se especifica en el primer argumento. A continuación, de todos los ficheros anteriores vuelve a mover al directorio actual aquéllos cuyo patrón se especifica en el segundo argumento. Por último, mediante el comando `DELTREE` borra el directorio temporal junto con todos los ficheros que contiene. La opción `/Y` se usa para impedir que este comando solicite confirmación antes de comenzar el proceso de borrado, aunque no funciona (un fallo más de MS-DOS).

Obsérvese que el directorio temporal se crea en el directorio raíz de la unidad actual. Por lo tanto estamos suponiendo que:

a) Podemos escribir en la unidad actual, lo cual es lógico ya que pretendemos borrar ficheros de dicha unidad.
b) Que hay espacio suficiente para crear un directorio, lo cual es bastante probable debido a que un directorio consume muy poco espacio.

Otro modo de resolver el problema consiste en mover al directorio temporal los ficheros que no queremos borrar (aquéllos cuyo patrón se especifica en el segundo argumento), borrar a continuación los especificados mediante primer argumento y finalmente volver a mover los archivos del directorio temporal al actual:

```
MD \MIBORRA
MOVE %2 \MIBORRA
DEL %1
MOVE \MIBORRA\%2 .
RD \MIBORRA
DELTREE \MIBORRA /Y
```

Obsérvese que cuando queremos mover algún fichero al directorio actual escribimos un punto como segundo argumento del comando `MOVE`. Este punto hace referencia al directorio actual, al igual que los dos puntos (`..`) simbolizan el directorio padre del actual.

El problema puede resolverse además de una tercera forma, pero esta última no es recomendable pues modificamos los atributos de los ficheros y al final no restauramos dichos atributos. No obstante, indicaremos esta tercera forma por interés didáctico, aunque repetimos que no es aconsejable. Esta tercera forma consiste en activar el atributo de sólo lectura (atributo R) a los ficheros especificados mediante el segundo argumento. A continuación borramos los ficheros especificados mediante el primer argumento (los anteriores no se borrarán) y, por último, desactivamos el atributo de sólo lectura a aquéllos a los que se les había activado anteriormente:

```
ATTRIB +R %2
DEL %1
ATTRIB -R %2
```

En determinadas circunstancias este último método puede ser el mejor, ya que no es necesario crear directorios intermedios y el funcionamiento del fichero batch no depende de que en la unidad actual exista suficiente espacio libre para contener un nuevo directorio.

Para borrar todos los ficheros del directorio actual exceptuando los que posean extensión TXT debe escribirse la siguiente línea de comando:

```
MIBORRA  *.*  *.TXT
```

9. Escribir un archivo de ejecución por lotes que muestre por pantalla los nombres de todos los ficheros que tengan un determinado tamaño o bien hayan sido creados o modificados por última vez en una determinada fecha u hora. El tamaño, la fecha o la hora se indicará como primer y único argumento del archivo batch. Éste buscará en todos los subdirectorios de la unidad actual comenzando por el directorio raíz.

SOLUCIÓN:

```
@echo off
rem Archivo BUSCA.BAT: Muestra los ficheros/subdirectorios
rem que tienen un determinado tamaño, fecha u hora.
dir \ /s | find "%1"
```

La explicación es simple: Mediante el comando `dir` y la opción `/s` se muestran todos los ficheros existentes en todos los subdirectorios a partir del directorio raíz. Se trata después de seleccionar aquéllos que cumplen el requisito especificado mediante el argumento del fichero batch. A continuación se muestran algunos ejemplos que ilustran el uso de este archivo. Obsérvese el formato en el que se indica el tamaño, la fecha o la hora:

`BUSCA 7.223`	Muestra los ficheros que tengan ese tamaño. Hay que incluir el punto de los millares, ya que la orden `dir` de la versión 6.2 los muestra así.
`BUSCA 06/01/93`	Muestra los ficheros que fueron creados o modificados por última vez en esa magnífica y gloriosa fecha.
`BUSCA 1:37`	Muestra los ficheros creados o modificados por última vez en esa hora concreta.

10. Escribir un archivo de ejecución por lotes para MS-DOS que muestre por orden alfabético todos los subdirectorios de la unidad actual que son hijos de otro subdirectorio, el cual se pasa como primer y único argumento del archivo batch.

Tan sólo deben mostrarse los subdirectorios, de modo que aparezca en una línea cada uno, tal como los muestra el comando `dir`. Si el número de subdirectorios excede el total de líneas de la pantalla, éstos deberán mostrarse pantalla a pantalla.

SOLUCIÓN:

Existen varias formas de solucionar este problema. La más simple consiste en usar el comando `dir` indicándole que muestre sólo los items que tienen activado el atributo *d*, el cual indica que se trata de directorios. Para ello hay que utilizar la opción `/a` del comando `dir` seguida del atributo que deberán tener los items seleccionados. Además se hará uso del filtro `sort`, el cual lee a través de una tubería la salida del comando `dir` y ordena alfabéticamente dicha información. La salida del filtro `sort` se conecta con la entrada del filtro `more`, el cual mostrará pantalla a pantalla los directorios si éstos ocuparan más de una.

```
@echo off
rem Muestra los subdirectorios del primer argumento.
dir %1 /ad | sort | more
```

El archivo anterior consigue parcialmente el objetivo, ya que el comando `dir` muestra información extra, como la etiqueta de volumen del disco o disquete y el nombre del subdirectorio padre. Para evitar que se muestren esas líneas y lograr que aparezcan exclusivamente los subdirectorios del directorio indicado como primer argumento, puede usarse la orden `dir` conectando su salida a la entrada de la orden `find` mediante otra tubería.

```
@echo off
rem Muestra los subdirectorios del primer argumento.
dir %1 | find "<" | sort | more
```

Así sólo se seleccionan las líneas que contienen el carácter "<", las cuales corresponden a los subdirectorios, ya que éstos van etiquetados con la secuencia de caracteres "<DIR>". No obstante, de esta forma también se muestran los directorios actual (`.`) y padre (`..`). Para impedir que eso ocurra se puede utilizar de nuevo la orden `find` con objeto de eliminar esas líneas.

```
@echo off
rem Muestra los subdirectorios del primer argumento.
dir %1 | find "<" | find /v "." | sort | more
```

Obsérvese que la orden anterior usa cuatro tuberías para conseguir el objetivo.

Preguntas de Test

1. Un sistema operativo es

   a) hardware.
   b) software.
   c) un programa para controlar el hardware por lo que no es ni hardware ni software.
   d) la parte de la memoria que controla los periféricos.
   e) uno de los programas que almacena la ROM.

2. Un sistema operativo es

   a) un lenguaje de programación.
   b) un código de representación de instrucciones.
   c) un programa gestor de compilación.
   d) un traductor de lenguaje natural.
   e) Todas las respuestas anteriores son correctas.
   f) Ninguna de las respuestas anteriores es correcta.

3. Los sistemas operativos

   a) planifican y supervisan la ejecución de los programas.
   b) controlan las operaciones de entrada y salida de datos.
   c) administran los recursos del ordenador.
   d) Todas las respuestas anteriores son correctas.
   e) Ninguna de las respuestas anteriores es correcta.

4. Un sistema distribuido es

   a) un conjunto de ordenadores bien colocados.
   b) una red de ordenadores.
   c) un conjunto de ordenadores con el mismo sistema operativo.
   d) Ninguna de las respuestas anteriores.

5. La multiprogramación consiste en

   a) la existencia de más de un programa en memoria principal.
   b) la existencia de más de un programa en memoria secundaria.
   c) la existencia de más de un programa en la c.p.u.
   d) realizar muchos programas a la vez.
   e) Ninguna de las respuestas anteriores es correcta.

6. La multiprogramación consiste en

   a) realizar múltiples programas a la vez.
   b) un sistema de tiempo compartido.
   c) tener múltiples programas cargados en memoria principal simultáneamente.
   d) distribuir y ejecutar un programa en varios procesadores.
   e) Ninguna de las respuestas anteriores es correcta.

7. Indicar la afirmación correcta:

   a) Los sistemas operativos de procesamiento por lotes incorporaban la técnica SPOOL para acelerar las Entradas y Salidas de datos.
   b) La técnica de *tiempo compartido* todavía se usa en algunos sistemas operativos actuales.
   c) La multiprogramación consiste en lograr que el programador pueda codificar varios programas a la vez.
   d) El primer ordenador tenía un sistema operativo que le permitía gestionar mejor sus escasos recursos (memoria, CPU, lectoras y perforadoras de tarjetas).

8. ¿En qué modo de ejecución pueden los programas ejecutar todas las instrucciones máquina del microprocesador, acceder a cualquier posición de la memoria y modificar cualquier registro?

   a) Modus operandi.
   b) Modo privilegiado o supervisor.
   c) Modo usuario.
   d) Modo normal.
   e) Modo sistema operativo.
   f) Ninguna de las respuestas anteriores es correcta.

9. La técnica de SPOOL

   a) sirve para optimizar sistemas operativos interactivos.
   b) es una forma de administrar la memoria (gestión de recursos).
   c) es una forma de administrar la CPU (gestión de procesos).
   d) se usa para optimizar la gestión de Entradas/Salidas.
   e) Todas las respuestas anteriores son correctas.
   f) Ninguna de las respuestas anteriores es correcta.

10. En un sistema operativo un *proceso* es

    a) un programa.
    b) un programa almacenado en un fichero o lo que es lo mismo, un fichero que contiene el código de un programa.
    c) un programa que ya ha empezado a ejecutarse y aún no ha terminado.
    d) cualquiera de las funciones básicas del sistema operativo.
    e) un subsistema de gestión de programas y ficheros en el sistema operativo.

11. En un S.O., principalmente tipo UNIX, las llamadas relacionadas con el manejo de procesos son

    a) crear y matar (destruir un proceso).
    b) leer y escribir.
    c) pedir memoria y liberar memoria.
    d) crear, matar, esperar a la finalización del proceso hijo, pedir memoria, liberar memoria y enviar señales a otros procesos.
    e) Ninguna de las respuestas anteriores es correcta.

12. Un fichero informático es

    a) un conjunto de información, al que nos podemos referir con un nombre, almacenado en un soporte informático.
    b) un buffer de memoria para almacenar datos de comunicación entre la CPU y un dispositivo de almacenamiento, usualmente lento.
    c) un almacenamiento temporal de datos.
    d) un conjunto de información que contiene toda la necesaria para ejecutar un programa.
    e) Todas las respuestas anteriores son correctas.
    f) Ninguna de las respuestas anteriores es correcta.

13. En un sistema operativo las llamadas al sistema típicas relacionadas con el manejo de ficheros son

a) crear, borrar, copiar y mover.
b) leer y escribir.
c) abrir y cerrar.
d) crear, borrar, abrir, cerrar, leer y escribir.
e) Ninguna de las respuestas anteriores es correcta.

14. En un sistema operativo denominamos *ruta de acceso*, camino o path a

a) el camino que siguen los datos desde la memoria principal a la CPU.
b) el bus de datos, por el que circulan todos los datos.
c) el camino que siguen los datos utilizados por un programa en ejecución.
d) la lista de nombres de directorios que indican la ubicación de un fichero o directorio determinado.
e) Ninguna de las respuestas anteriores es correcta.

15. En un sistema operativo, la salida estándar es

a) la pantalla o monitor, que es donde escriben normalmente los programas.
b) el dispositivo donde escriben siempre todos los programas.
c) el dispositivo de donde leen sus datos por defecto todos los programas.
d) un dispositivo en el cual los programas escriben normalmente los resultados de salida y que puede redirigirse a un fichero u otro dispositivo. Si no se redirige suele ser la pantalla.
e) Ninguna de las respuestas anteriores es correcta.

16. MS-DOS es

a) un sistema operativo multiusuario y multiplataforma.
b) un sistema operativo multiusuario para PCs.
c) un sistema operativo tipo UNIX para PCs.
d) un sistema operativo potente, el cual utiliza la técnica de tiempo compartido para ejecutar los programas, pero carece de interfaz gráfico.
e) Todas las respuestas anteriores son correctas.
f) Ninguna de las respuestas anteriores es correcta.

17. En MS-DOS los comandos externos

a) permanecen siempre en memoria principal.
b) están almacenados en ROM y se ejecutan cuando son requeridos.
c) forman parte de la BIOS.
d) están almacenados en ficheros.
e) no existen, sólo existen los comandos internos.

18. Desde el *prompt* de un sistema operativo tipo DOS se pueden ejecutar los ficheros con extensión

a) `.EXE`.
b) `.EXE` y `.COM`.
c) `.EXE`, `.COM` y `.BAT`.
d) `.EXE`, `.COM`, `.BAT` y `.SYS`.
e) `.TXT` y `.BAT`.
f) Ninguno de los anteriores.

19. Los atributos de ficheros y directorios en MS-DOS son:

a) Oculto y de sólo lectura.
b) Lectura, escritura y ejecución (rwx).
c) A, H, R, S y D.
d) A, H, R y S.
e) Ninguno de los anteriores.

20. En MS-DOS se impone una longitud máxima para el nombre y la extensión respectivamente ¿Cuáles son dichas longitudes?

a) 7 y 3.
b) Once repartidos entre nombre y extensión.
c) 3 y 8.
d) 8 y 3.
e) 8 y 2.
f) 9 y 3.

21. La F.A.T. es

a) el Sistema de Transferencia de Atributos (File Attribute Transfer).
b) la Tabla de Ficheros borrados Recuperables.
c) la Tabla de Ficheros borrados Irrecuperables.
d) la Tabla de Asignación de Procesos.
e) la Tabla de Asignación de Ficheros.
f) Ninguna de las respuestas anteriores es correcta.

22. Indicar todos los ficheros (puede haber más de uno) que se verán afectados de alguna forma por la siguiente orden, suponiendo que estén situados en el directorio actual: `COPY PRAC??.T??`

a) PRACTICA.TXT
b) TUPRAC.TXT
c) PRAC1.TXT
d) PRAC12.TXT
e) PRAC.T
f) PRACTI.T

23. Linux es

a) un programa para dibujar gráficos.
b) un sistema operativo tipo UNIX gratuito y muy potente, el cual se puede instalar en PCs.
c) el autor de un sistema operativo tipo UNIX.
d) un módulo de gestión de ficheros que pertenece al sistema operativo UNIX.
e) el primer sistema operativo multiusuario que existió.
f) Ninguna de las respuestas anteriores es correcta.

24. En un sistema operativo tipo UNIX el *pid* es

a) un número identificativo que posee cada proceso.
b) un número identificativo que posee cada usuario.
c) un número que indica la prioridad de cada proceso.
d) un número que indica el tiempo estimado de ejecución de cada proceso.
e) Ninguna de las respuestas anteriores es correcta.

25. En un sistema operativo tipo UNIX el *uid* es

a) un número identificativo que posee cada proceso.
b) un número identificativo que posee cada usuario.
c) un número que indica la prioridad de cada proceso.
d) un número que indica el tiempo estimado de ejecución de cada proceso.
e) Ninguna de las respuestas anteriores es correcta.

26. En un sistema operativo tipo DOS o UNIX, ¿Cómo podemos referenciar de forma rápida el directorio padre del directorio actual?

a) Mediante el directorio especial `"c:\Padre"`.
b) Mediante el directorio especial `"c:\.."`.
c) Con un punto (`.`).
d) Con dos puntos (`..`).
e) Con tres puntos (`...`).
f) Ninguna de las respuestas anteriores es correcta, ya que depende del directorio actual en el que nos encontremos.

27. En un sistema operativo tipo DOS o UNIX, ¿Cómo podemos referenciar de forma rápida el directorio actual?

a) Mediante las barras "\" y "/", según el caso.
b) Mediante el directorio especial `"c:\.."`.
c) Con un punto (`.`).
d) Con dos puntos (`..`).
e) Con tres puntos (`...`).
f) Ninguna de las respuestas anteriores es correcta, ya que depende del directorio actual en el que nos encontremos.

28. El concepto de *redirección* en un sistema operativo tipo DOS o UNIX consiste en

a) comunicar dos procesos mediante la desviación o redirección de la salida estándar del primer proceso a la entrada estándar del segundo.
b) redirigir la salida estándar de un proceso a la entrada de un fichero de dispositivo (impresora, monitor, teclado, etc).
c) redirigir la entrada estándar de un proceso a un fichero para que el primero lea sus datos desde ese fichero.
d) redirigir la entrada y la salida estándar de un proceso al mismo fichero.
e) redirigir la entrada estándar, la salida estándar o ambas de un proceso determinado a ficheros distintos o al mismo fichero.

29. Los símbolos utilizados para representar los operadores de redirección, ya sea ésta a un fichero o desde un fichero son

a) |, \ y /.
b) >, < y >>.
c) >, < y <<.
d) >, <, >> y <<.
e) |, > y <.
f) |, >, < y >>.

30. El operador de redirección ">"

a) redirige la salida estándar a un fichero cuyo nombre se debe indicar a la derecha del operador.
b) si el fichero que se indica a su derecha no existe lo crea.
c) si el fichero ya existe lo borra completamente antes de comenzar a escribir en él.
d) Todas las afirmaciones anteriores son ciertas.
e) Ninguna de las afirmaciones anteriores es cierta.

31. El operador de redirección "<"

a) redirige la entrada estándar desde un fichero cuyo nombre se debe indicar a la derecha del operador.
b) redirige la salida estándar a un fichero cuyo nombre se debe indicar a la derecha del operador.
c) si el fichero que se indica a su derecha no existe lo crea.
d) si el fichero ya existe lo borra completamente antes de comenzar a escribir en él.
e) Todas las afirmaciones anteriores son ciertas.
f) Ninguna de las afirmaciones anteriores es cierta.

32. El operador de redirección ">>"

a) redirige la salida estándar a un fichero cuyo nombre se debe indicar a la derecha del operador.
b) si el fichero que se indica a su derecha no existe lo crea.
c) si el fichero ya existe no lo borra, sino que añade la nueva información al final del mismo.
d) Todas las afirmaciones anteriores son ciertas.
e) Ninguna de las afirmaciones anteriores es cierta.

33. Si queremos redirigir la salida estándar de un programa denominado `"dir"` a un fichero de nombre `"loco.txt"` tendremos que ejecutar la orden

a) `dir | loco.txt.`
b) `dir < loco.txt.`
c) `dir > loco.txt.`
d) `dir \loco.txt.`
e) Ninguna de las órdenes anteriores ejecuta la acción deseada.

34. Si queremos redirigir la entrada estándar de un programa denominado `"sort"` desde un fichero de nombre `"loco.txt"` tendremos que ejecutar la orden

a) `sort | loco.txt.`
b) `sort < loco.txt.`
c) `sort > loco.txt.`
d) `sort \loco.txt.`
e) Ninguna de las órdenes anteriores ejecuta la acción deseada.

35. Si deseamos redirigir la entrada estándar de un programa denominado `"sort"` para que éste lea sus datos desde un fichero de nombre `"mundo.txt"` y que los resultados se almacenen en otro fichero llamado `"ordenado.txt"` en lugar de mostrarse por la salida estándar, deberemos ejecutar la orden

a) `mundo.txt > sort > ordenado.txt.`
b) `ordenado.txt < sort < mundo.txt.`
c) `ordenado.txt < mundo.txt > sort.`
d) `sort < ordenado.txt > mundo.txt.`
e) `sort < mundo.txt > ordenado.txt.`
f) Todas las respuestas anteriores consiguen el objetivo planteado.
g) Ninguna de las respuestas anteriores consigue el objetivo planteado.

36. ¿Qué efecto tiene en el sistema operativo MS-DOS la orden "`dir > sort`" ?

a) Visualiza la lista de ficheros y directorios ordenados alfabéticamente.
b) Almacena en un fichero determinado por el sistema operativo la lista de ficheros y directorios ordenados alfabéticamente.
c) Almacena en un fichero denominado "`sort`" la lista de ficheros y directorios ordenados alfabéticamente.
d) Almacena en un fichero denominado "`sort`" la lista de ficheros y directorios.
e) Ninguna de las respuestas anteriores es correcta.

37. ¿Qué efecto tiene en el sistema operativo MS-DOS la orden "`sort < dir > prn`" ?

a) Imprime la lista de ficheros y directorios ordenados alfabéticamente.
b) Imprime el contenido de un fichero denominado "`dir`" ordenado alfabéticamente.
c) Ordena alfabéticamente la lista de ficheros y directorios y almacena el resultado en otro fichero de nombre "`prn`".
d) La orden anterior es incorrecta.
e) Ninguna de las respuestas anteriores es correcta.

38. El concepto de *tubería* (pipe) en un sistema operativo tipo DOS o UNIX consiste en

a) comunicar dos procesos mediante la desviación o redirección de la salida estándar del primer proceso a la entrada estándar del segundo.
b) redirigir la salida estándar de un proceso a la entrada de un fichero de dispositivo (impresora, monitor, teclado, etc).
c) redirigir la entrada estándar de un proceso a un fichero para que el primero lea sus datos desde ese fichero.
d) redirigir la entrada y la salida estándar de un proceso al mismo fichero.
e) redirigir la entrada estándar, la salida estándar o ambas de un proceso determinado a ficheros distintos o al mismo fichero.

39. El símbolo usado para representar una tubería es

a) `|`
b) `\`
c) `/`
d) `>`
e) `<`
f) Depende del tipo de tubería que deseemos utilizar.

40. Si deseamos redirigir la salida estándar de un programa llamado "`dir`" a la entrada estándar de otro denominado "`sort`" tendremos que ejecutar la orden

a) `dir | sort.`
b) `dir > sort.`
c) `dir < sort.`
d) `dir >> sort.`
e) `dir, sort.`
f) `dir & sort.`
g) `dir` y `sort` deberán ejecutarse por separado.

41. ¿Qué efecto tiene en el sistema operativo MS-DOS la orden "`dir | sort`" ?

a) Visualiza la lista de ficheros y directorios ordenados alfabéticamente.
b) Almacena en un fichero determinado por el sistema operativo la lista de ficheros y directorios ordenados alfabéticamente.
c) Almacena en un fichero denominado "`sort`" la lista de ficheros y directorios ordenados alfabéticamente.
d) Almacena en un fichero denominado "`sort`" la lista de ficheros y directorios.
e) Ninguna de las respuestas anteriores es correcta.

42. ¿Qué efecto tiene en el sistema operativo MS-DOS la orden "`dir | sort > prn`" ?

a) Imprime la lista de ficheros y directorios ordenados alfabéticamente.
b) Imprime el contenido de un fichero denominado "`dir`" ordenado alfabéticamente.
c) Ordena alfabéticamente el contenido de un fichero denominado "`dir`" y almacena el resultado en otro fichero de nombre "`prn`".
d) La orden anterior es incorrecta.
e) Ninguna de las respuestas anteriores es correcta.

43. ¿Qué efecto tiene en el sistema operativo MS-DOS la orden "`sort | dir > prn` ?

a) Imprime la lista de ficheros y directorios ordenados alfabéticamente.
b) Imprime el contenido de un fichero denominado "`dir`" ordenado alfabéticamente.
c) Ordena alfabéticamente el contenido de un fichero denominado "`dir`" y almacena el resultado en otro fichero de nombre "`prn`".
d) La orden anterior es incorrecta.
e) Ninguna de las respuestas anteriores es correcta.

44. ¿A cuál de las siguientes órdenes equivale la orden de MS-DOS "`type dir | sort`"?

a) `sort < dir`
b) `sort | type dir`
c) `sort <| dir`
d) `dir | sort`
e) `dir type | sort`
f) Esa orden es incorrecta.
g) Ninguna de las respuestas anteriores es correcta.

45. ¿A cuál de las siguientes órdenes equivale la orden de MS-DOS "`type dir | more`"?

a) `more < dir`
b) `more | type dir`
c) `more <| dir`
d) `dir | more`
e) `dir type | more`
f) Esa orden es incorrecta.
g) Ninguna de las respuestas anteriores es correcta.

Respuestas

1. b)
2. f)
3. d)
4. d)
5. a)
6. c)
7. b)
8. b)
9. d)
10. c)
11. d)
12. a)
13. d)
14. d)
15. d)
16. f)
17. d)
18. c)
19. c)
20. d)
21. e)
22. c), d), e) y f)
23. b)
24. a)
25. b)
26. d)
27. c)
28. e)
29. b)
30. d)
31. a)
32. d)
33. c)
34. b)
35. e)
36. d)
37. b)
38. a)
39. a)
40. a)
41. a)
42. a)
43. d)
44. a)
45. a)

Capítulo 4

Metodología de la Programación y Lenguaje C.

1. Enumerar las ventajas e inconvenientes más destacados del uso de un lenguaje de alto nivel con respecto al lenguaje máquina.

   SOLUCIÓN:

   Ambos tipos de lenguajes son bastante diferentes y las ventajas de uno son los inconvenientes del otro.

   Ventajas:

   - Los programas codificados en un lenguaje de alto nivel son más fáciles de entender, escribir y modificar.
   - Sus instrucciones están orientadas al problema a resolver y no al ordenador, y además son muy potentes.
   - No es necesario que el programador conozca la arquitectura interna del ordenador (características del microprocesador y funcionamiento de los dispositivos de E/S) en el que se va a ejecutar el programa.
   - Los lenguajes de alto nivel ofrecen una mayor facilidad para detectar errores (sintácticos y semánticos).
   - Permiten utilizar sentencias declarativas para asociar identificadores simbólicos a los distintos elementos del programa (variables, constantes, subprogramas, etc.).
   - Se pueden incluir comentarios con el fin de clarificar cualquier aspecto del programa.

   Inconvenientes:

   - Los programas fuentes deben ser traducidos (compilados o interpretados) para poder ser ejecutados.
   - En general, el lenguaje máquina permite escribir programas más eficientes en cuanto a tiempo de ejecución y memoria utilizada, aunque ello depende en gran medida de la habilidad del programador.

2. Dadas las siguientes variables en C, razonar cuánto espacio requieren en memoria.

```
struct cosa {
  int a, b;
  char c, d[20];
} cosa 1;

struct objeto {
  char a, b;
  int c;
  struct cosa d;
};

struct objeto peace[2];
```

SOLUCIÓN:

La variable `cosa1` es de tipo `struct cosa`. Su declaración ocupa en memoria veinticinco bytes.

2 bytes + 2 bytes + 1 byte + 20 bytes = 25 bytes.

La variable `peace` es un array de dos elementos de tipo `struct objeto`, y ocupa cincuenta y ocho bytes.

2 * (tamaño de `struct objeto`) = 2 * (1 byte + 1 byte + 2 bytes + 25 bytes) = 58 bytes.

En total ambas variables requieren un espacio en memoria de 25 bytes + 58 bytes = 83 bytes.

Recordemos que cuando declaramos el identificador `objeto` estamos declarando simplemente un patrón de estructura y no una variable, por lo que esa declaración no ocupa ninguna memoria. Al declarar el identificador `peace` declaramos una variable de tipo array en la que cada elemento es de tipo `struct objeto`. Por lo tanto, en el fragmento de código anterior solamente se declaran dos variables: `cosa1` y `peace`.

3. Indicar en el programa fuente dónde se encuentran los siguientes fragmentos de código:

a) La llamada.
b) La declaración.
c) La definición (el código).
d) Los parámetros, su tipo y cómo se transfieren.

correspondientes a la función `printf()`.

```
#include <stdio.h>

int main() {

  printf("NUM: %d", 4);
  return 0x01;
}
```

SOLUCIÓN:

a) La llamada a la función es la primera sentencia del programa:

```
printf("NUM: %d", 4);
```

b) La declaración o prototipo de la función `printf()` se encuentra en el fichero de encabezamiento `stdio.h`.

c) El subprograma `printf()` es una función de biblioteca. Su código no aparece en el programa fuente, pues ya ha sido escrito por el fabricante del compilador y se encuentra en una biblioteca de funciones. Dicho código se unirá al del programa fuente tras la fase de compilación de éste último, durante el proceso conocido como *edición de enlaces* (linkage).

d) Los parámetros están dentro de los paréntesis de la expresión de llamada a la función. Son dos expresiones constantes y ambos se pasan por valor:

- Una cadena de caracteres (`char *`) cuyo valor es `"NUM: %d"`.
- Un entero (`int`) con valor cuatro.

4. Indicar la salida del siguiente programa en Lenguaje C:

```
#define UNO  printf("1")
#define DOS  UNO; UNO
#define TRES DOS; printf("3")

void main() {

  UNO;
  DOS;
  TRES;
}
```

SOLUCIÓN:

El programa anterior escribe en la salida estándar el siguiente número:

111113

5. Dado el siguiente fragmento de código en lenguaje C:

```
int x = ALGO;
printf ("%d; %o; %x;", x, x, x);
```

Indicar el resultado de la llamada a la función si previamente se ha definido ALGO como una macro (o constante simbólica) como sigue:

a) `#define ALGO 20`
b) `#define ALGO 020`
c) `#define ALGO 0x20`

SOLUCIÓN:

a) La macro se ha definido como el valor entero 20. El resultado consistirá en imprimir ese número en base diez, posteriormente en notación octal y, por último, en notación hexadecimal:

```
20; 24; 14;
```

b) Como la constante empieza por cero, ésta se ha escrito en notación octal. Dicha constante también deberá expresarse en base diez y en notación hexadecimal:

```
16; 20; 10;
```

c) Por último, se ha escrito la constante en notación hexadecimal, que la función también deberá expresar en base diez y en notación octal:

```
32; 40; 20;
```

6. Suponemos que en un programa escrito en lenguaje C se tiene declarada una variable mediante la siguiente sentencia:

```
int a = 3;
```

Indicar, razonando la respuesta, el resultado que se obtiene al realizar la siguiente llamada a función:

```
printf("%i,%i", 9 &a, 9|a);
```

SOLUCIÓN:

Dado que justo antes del símbolo `&` aparece un número entero, se deduce rápidamente que éste no es el operador de dirección. Por lo tanto, sólo puede tratarse del operador AND a nivel de bits. Asimismo, el símbolo `|` es el operador OR a nivel de bits.

El entero 9 se representa en binario como 1001 (con tantos ceros por delante como hagan falta para ocupar el número de bytes que se estén usando para representar los enteros). El entero 3 se representa como 0011. Tras realizar las operaciones indicadas se obtienen los siguientes resultados:

```
1001 & 0011 = 0001
1001 | 0011 = 1011
```

El número binario 0001 corresponde al uno en base diez y el número binario 1011 corresponde al once en base diez. De este modo, la salida generada por la llamada a función es:

1,11

7. Indicar razonadamente el valor de las variables enteras `a`, `b` y `c` después de ejecutarse el siguiente fragmento de código en lenguaje C:

```
c = 5;
a = c++;
b = c;
c = ++a;
```

SOLUCIÓN:

En cada una de las cuatro sentencias anteriores se efectúan una serie de acciones que detallamos a continuación por orden:

1. En la primera sentencia se asigna el valor entero 5 a la variable entera `c`. Las otras variables, en principio, tienen un valor indefinido.

2. En esta segunda sentencia tiene lugar un post-incremento, por lo cual la variable `c` sólo podrá incrementarse después de la asignación. Por lo tanto, `a` tomará el valor 5 y `c` el valor 6.

3. En la tercera sentencia simplemente se asigna el valor de `c` a la variable `b`. Por tanto `b` tomará el valor 6.

4. En la última instrucción tiene lugar un pre-incremento, debido a lo cual la variable `a` se incrementa justo antes de la asignación. De este modo, la variable `a` pasa a valer 6 y ese valor se asigna posteriormente a la variable `c`. Esta última asignación es innecesaria, ya que esa variable ya tenía el valor 6.

Por consiguiente, las tres variables toman el mismo valor entero: 6.

8. Indicar *justificadamente* cuáles son los resultados de los siguientes programas:

```
#include <stdio.h>

void main() {
  int a, b;

  a = 4;
  b = a++;

  printf("a: %d, b: %d.\n", a, b);
}
```

```
#include <stdio.h>

void main() {
  int a, b;

  a = 4;
  b = ++a;

  printf("a: %d, b: %d.\n", a, b);
}
```

SOLUCIÓN:

En el primer programa la variable `a` está afectada por el operador de post-incremento. Por lo tanto, dicha variable se incrementa en una unidad *después* de la asignación. Por lo tanto, la salida de éste programa es la siguiente:

```
a: 5, b: 4.
```

En el segundo programa la variable `a` está afectada por el operador de pre-incremento. Por lo tanto, dicha variable se incrementa en una unidad *antes* de la asignación. Por lo tanto, la salida de éste programa es la siguiente:

```
a: 5, b: 5.
```

9. Sea el siguiente fragmento de un programa codificado en lenguaje C, en el que la variable `x` es de tipo entero:

```
switch (x) {
  case 5: printf ("5");
  case 2: printf ("2");
          break;
  case 3: printf ("3");
  default: printf ("Def Leppard");
  case 1: printf ("1");
}
```

Indicar cuál es la salida del fragmento anterior si justo antes de su ejecución la variable `x` toma los valores 1, 2, 3, 4 y 5.

SOLUCIÓN:

El comportamiento de la sentencia `switch` es simple: Primero comprueba el valor de la variable `x` con todos los de las etiquetas que acompañan a la palabra reservada `case`, independientemente del orden de éstas. Si encuentra alguna etiqueta cuyo valor coincide con el de `x` ejecuta las sentencias que aparezcan a partir de ese lugar hasta que encuentre una sentencia `break;` o hasta que termine la sentencia `switch`. En caso de que la variable `x` no coincida con ninguna etiqueta se ejecutarán entonces las sentencias que aparecen tras la palabra clave `default` (si existe, ya que es opcional) hasta que aparezca una sentencia `break;` o finalice la sentencia `switch`.

x	Salida
1	1
2	2
3	3Def Leppard1
4	Def Leppard1
5	52

10. Simular la ejecución de la siguiente sentencia `switch` mediante sentencias `if` sin utilizar sentencias compuestas o bloques.

```
switch (x) {
  case 5: printf("5");
  case 6: printf("6");
          break;
  case 1:
  case 2: printf("2");
  default: printf("FIN");
}
```

SOLUCIÓN:

```
if (x == 5)
   printf ("56");
else if (x == 6)
        printf ("6");
     else if (x == 1 || x == 2)
             printf ("2FIN");
          else
             printf ("FIN");
```

11. Indicar de manera razonada qué se obtendría de la ejecución del siguiente fragmento de programa en lenguaje C. Se supone que no se producen errores al escribir en la salida estándar.

```
for (putchar('1'); putchar('2'); putchar('3'))
    putchar('4');
```

SOLUCIÓN:

La función `putchar()` escribe en la salida estándar el carácter que recibe como único argumento y devuelve dicho carácter si la escritura ha sido efectuada correctamente.

En el bucle `for` se ejecuta en primer lugar la primera expresión (inicialización) y posteriormente se evalúa la segunda expresión (condición). Si ésta resulta ser distinta de cero se ejecuta el cuerpo del bucle, se evalúa la tercera expresión del mismo (actualización) y se vuelve a evaluar la condición. En caso contrario se abandona el bucle.

En este caso la condición resulta ser siempre verdadera, por lo que se trata un bucle infinito. En dicho bucle se escribe primero el carácter '1' y posteriormente se repite indefinidamente la secuencia de caracteres '243'.

1243243243243243243243243...

12. ¿Qué acciones realiza el siguiente fragmento de código escrito en lenguaje C?

```
c = 12;
while (x = --c) {
      printf("%i; ", x);
      c = x--;
      printf("%i\n", x);
      }
```

SOLUCIÓN:

En principio, si el fragmento de programa es difícil de leer se aconseja reescribirlo de forma más clara y legible, aunque resulte menos compacta y optimizada que la versión original. De este modo resulta más sencillo realizar la traza del fragmento de código.

```
c = 12;
c = c - 1;
x = c;
while (c != 0) {
      printf("%i; ", x);
      c = x;
      x = x - 1;
      printf("%i\n", x);
      c = c - 1;
      x = c;
      }
```

Aunque no es necesario se puede escribir una nueva versión del código, que elimine las sentencias y variables redundantes:

```
c = 11;
while (c != 0) {
      printf("%i; %i\n", c, c - 1);
      c = c - 1;
      }
```

El fragmento de código escribe en la salida estándar la siguiente lista de números:

```
11; 10
10; 9
9; 8
8; 7
7; 6
6; 5
5; 4
4; 3
3; 2
2; 1
1; 0
```

13. Indicar razonadamente cuál es la salida del siguiente fragmento de programa en lenguaje C:

```
unsigned int x = 010;   /* Representación octal */
unsigned int y;

do { y = x-- + 2 * 8 + 16;
     --x;
     printf("\n %x.", y);
   } while (x);
```

SOLUCIÓN:

A la variable x se le asigna un dato en notación octal, pues en C las constantes numéricas que empiezan por cero se consideran expresadas en base octal. Para operar se puede pasar la constante a base diez, que es lo más cómodo. Así la constante octal 010 equivale al número ocho en base diez.

En la primera expresión del bucle `do-while` se decrementa en una unidad la variable x, pero como se trata de un post-decremento, antes de decrementarse dicha variable se debe evaluar la expresión.

```
y = x-- + 2 * 8 + 16
```

Esta es equivalente a la siguiente expresión, teniendo en cuenta la precedencia de operadores:

```
y = x-- + 32
```

y sustituyendo el valor inicial de la variable x se tiene que:

```
y = 8 + 32
```

por lo que el valor final de y es cuarenta (en base diez). Tras esa asignación se decrementa la variable `x`, que pasa a valer siete.

La siguiente expresión es un pre-decremento. En este caso da igual que se trate de un pre-decremento o un post-decremento, ya que no forma parte de otra expresión más compleja. Así la variable x se decrementa de nuevo pasando a valer seis.

Mediante la función `printf()` se imprime en la salida estándar el valor de la variable y en hexadecimal debido al uso del modificador `%x`. Por lo tanto, se imprimirá el siguiente valor:

28.

ya que este valor corresponde al número cuarenta expresado en notación hexadecimal:

$$28)_h = 2 * 16 + 8 = 32 + 8 = 40$$

El siguiente paso consiste en comprobar la condición de salida del bucle. Esta condición verifica si la variable x es distinta de cero, ya que para el lenguaje C es verdad todo lo que es distinto de cero, y es falso todo lo que vale cero. Por ello, el siguiente fragmento de código:

```
do { .
     .
     .
} while (x);
```

es equivalente a este otro:

```
do { .
     .
     .
} while (x != 0);
```

En esta primera iteración la variable x pasa a valer seis. Como ese valor es distinto de cero la condición es verdadera y se pasa a ejecutar de nuevo el bucle:

```
y = 6 + 32
```

A continuación x se decrementa dos veces, pasando a valer cuatro. Por último, se imprime el valor 38 en notación hexadecimal (que es lo mismo que restar dos unidades a la representación anterior):

26.

Como el valor final de la variable x no es cero, el bucle se ejecuta de nuevo: Se asigna a la variable y el valor treinta y seis (4 + 32). A continuación x se decrementa dos veces pasando a valer dos y se imprime el valor treinta y seis expresado en notación hexadecimal:

24.

De nuevo, x no es cero por lo que tiene lugar una iteración más: Se asigna el valor treinta y cuatro a la variable y, se decrementa dos veces la variable x, que pasa a valer cero, y se imprime el valor treinta y cuatro en notación hexadecimal:

22.

Ahora la variable x vale cero, por lo que la condición del bucle es falsa y se termina la ejecución de dicho bucle.

Resumiendo, el resultado del anterior fragmento de código en lenguaje C es la impresión a través de la salida estándar de las cuatro líneas siguientes:

28.
26.
24.
22.

14. La función `strcmp()`, cuyo prototipo se encuentra en el fichero `string.h`, acepta dos cadenas de caracteres como argumentos. Dicha función compara carácter a carácter ambas cadenas, devolviendo cero si son iguales y un valor distinto de cero si son distintas. En ese caso, el valor devuelto es el resultado de restar los primeros dos caracteres que son distintos en cada cadena: El de la primera cadena menos el de la segunda. Indicar el valor que devuelven las siguientes llamadas a dicha función, sabiendo que el valor ASCII del carácter 'A' es 65 y del carácter 'B' es 66:

a) `strcmp("HOLA", "HOLA")`
b) `strcmp("HOLA FELIX", "HOLB FÉLIX")`
c) `strcmp("HOLBA", "HOLA")`
d) `strcmp("HOLAB", "HOL")`
e) `strcmp("HOL", "HOLA AMIGO FÉLIX RODRÍGUEZ")`

SOLUCIÓN:

a) Devuelve 0, ya que ambas cadenas son iguales.
b) Devuelve -1, que es el resultado de restar 66 ('B') al número 65 ('A').
c) Devuelve 1, que es el resultado de la operación 66 - 65.
d) Devuelve 65, ya que restamos a 65 el valor ASCII del carácter fin de cadena ('\0'), cuyo valor es cero.
e) Devuelve -65, pues al carácter fin de cadena hay que restarle el número 65.

15. Indíquese si existen fallos en el siguiente programa en lenguaje C. Razónese la respuesta y en caso de que haya fallos expóngase alguna posible solución:

```
#include <stdio.h>

void main()
{ int i, *p:

  i = 3;
  *p = i;

  printf ("%i", *p);
}
```

SOLUCIÓN:

Efectivamente el programa anterior tiene un fallo muy grave que puede ocasionar que el programa no funcione bien algunas veces.

En principio, la declaración de variables es correcta y el programa reserva memoria para almacenar un entero y un puntero. Justo después de la declaración, los valores de las variables no están definidos y éstas contienen lo que se llama "basura". En la primera asignación se coloca el entero 3 en la posición de memoria que había reservada para un entero. En la siguiente asignación es donde está el error. Véase que el valor de la variable `i` se coloca en la posición de memoria cuya dirección está indicada por el puntero `p`, el cual contiene inicialmente "basura".

Es decir: En la declaración de variables el programa ha asignado memoria para almacenar el puntero `p`, pero no ha reservado memoria para colocar un entero en la dirección que contiene aleatoriamente `p` (apuntada por `p`), que inicialmente puede ser cualquiera.

Una solución posible consiste en asignar al puntero `p` la dirección de memoria de la variable `i`. Eso es correcto porque la asignación se está efectuando al puntero, cuyo espacio en memoria está reservado para tal fin.

```
#include <stdio.h>

void main()
{ int i, *p:

  i = 3;
  p = &i;

  printf ("%i", *p);
}
```

16. Indíquese si hay fallos en el siguiente programa en lenguaje C. Razónese la respuesta y en caso de que haya fallos expóngase alguna posible solución:

```
#include <stdio.h>

void main() {
  int i, *p:

  i = 3;
  p = i;
  printf ("%i", *p);
}
```

SOLUCIÓN:

El fallo del programa anterior hace que el resultado no sea el esperado.

Al asignar el valor de la variable `i` (3) al puntero p, se está suponiendo que ese número es la dirección de una variable de tipo entero, lo cual no es cierto. Al intentar escribir como valor entero el contenido de la dirección de memoria indicada (apuntada) por `p`, el programa muestra el contenido de la dirección 3, que puede ser cualquier cosa. De este modo, el resultado es indefinido y puede variar de una ejecución a otra.

Una posible solución consiste en asignar al puntero `p` la dirección de memoria de la variable `i`. Eso es correcto porque se asigna al puntero una dirección de memoria donde hay almacenado un número entero.

```
#include <stdio.h>

void main()
{ int i, *p:

  i = 3;
  p = &i;
  printf ("%i", *p);
}
```

17. Indicar la salida del siguiente programa en lenguaje C:

```
#include <stdio.h>

void swap (int *, int *);
void swap2 (int *, int *);

int asumar = 3; /* variable global */

void main ()
{
  int x = 3, y = 14;

  printf ("\nx = %i; y = %i;", x, y);
  swap(&x, &y);
  printf("\nx = %i; y = %i;", x, y);
  swap2(&x, &y);
  printf("\nx = %i; y = %i;", x, y);

  asumar = 1;
  swap2(&x, &y);
  printf("\nx = %i; y = %i;", x, y);
}
```

```
void swap (int *a, int *b)
{
  int aux = *a;
  *a = *b;
  *b = aux;
}

void swap2 (int *i, int *j)
{
  swap (i, j);

  if (*i < 10) *i += asumar;
  if (*j < 10) *j += asumar;
}
```

SOLUCIÓN:

Cada llamada a la sentencia `printf()` muestra el valor que van tomando en cada momento las variables `x` e `y`. La función `swap()` intercambia el valor de dos variables enteras a través de sus direcciones respectivas.

La función `swap2()` también intercambia el valor de dos variables enteras, pero además si el valor de alguna variable es menor que diez, a esa variable se le añade el valor de la variable global `asumar`. Obsérvese que cuando se realiza la llamada a la función `swap()` dentro de `swap2()` no se usa el operador de dirección (`&`), ya que tanto `i` como `j` son punteros que guardan direcciones de memoria de variables de tipo entero.

Por todo lo anterior, concluimos que las cuatro llamadas a `printf()` escriben en la salida estándar las siguientes cuatro líneas:

```
x = 3; y = 14;
x = 14; y = 3;
x = 6; y = 14;
x = 14; y = 7;
```

18. Explicar cómo funciona y qué acciones realiza el siguiente programa en Lenguaje C.

```
#include <stdio.h>

#define MAXLONG  200
#define NUMBLANCOS 4

int misterio(char *, int *);

void main() {

  char a[MAXLONG];
  int p = 0, j = 0;

  printf("-Dame la frase");
  gets(a);
  j = misterio(a, &p); /* Cuidado con la p */

  if (j) {
     printf("\n-SALIDA: Longitud %i: ", p); /* %i == %d */
     puts(a);
     }
  else puts("\nNa de na.");
}
```

```
int misterio(x, l)
char x[];
int *l; {

  int i = 0, blancos;

  for (blancos = 0; (blancos < NUMBLANCOS) && (x[i] != '\0'); i++)
      if(x[i] == ' ') blancos++;

  if (blancos == NUMBLANCOS) {
     x[i-1] = '\0'; /* fin de cadena */
     *l = i - 1;
     return 1;
     }
  return 0;
}
```

SOLUCIÓN:

La función `misterio()` comprueba si la cadena suministrada como primer argumento contiene al menos cuatro espacios, y en caso afirmativo recorta dicha cadena justo antes del cuarto espacio y devuelve por referencia la longitud de la cadena ya truncada.

Juego de Ensayo 1.

```
/* función misterio() */

 i  blancos  l  x
 0     0     0  este es mi primer ejemplo
 4     1     0  este es mi primer ejemplo
 7     2     0  este es mi primer ejemplo
10     3     0  este es mi primer ejemplo
17     4    17  este es mi primer

/* función main() */

 p   j  a
 0   0  este es mi primer ejemplo
 17  1  este es mi primer

-SALIDA: Longitud: 17
```

Juego de Ensayo 2.

```
/* función misterio() */

 i  blancos  l  x
 0     0     0  otra prueba
 4     1     0  otra prueba
11     1     0  otra prueba

/* función main() */

 p   j  a
 0   0  otra prueba
 0   0  otra prueba

Na de na.
```

19. Explicar cómo funciona y qué acciones realiza el siguiente programa en Lenguaje C.

```
#include <stdio.h>
#include <string.h>


#define MAXLONG 200
#define DIM       3


void main() {

  char s[MAXLONG];
  int cuenta[DIM], i = 0, j = 0;

  printf("- Dame la frase: ");

  gets(s);

  for (; i < DIM; i++) cuenta[i] = 0;

  paz(s, cuenta, &j);

  for (i = 0; i < DIM; i++)
      printf("Con %i letras: %i.\n", i + 1, cuenta[i]);

  printf("\nTotal: %i.\n", j + cuenta[0] + cuenta[1] + cuenta[2]);
}


void paz(cadena, cuen, nopaz)
char cadena[];
int cuen[], *nopaz; {

  int i = strlen(cadena) - 1;
  /* strlen(): longitud de la cadena */

  while (i >= 0) {
        if (cadena[i] == 'z')
           if (cadena[i-1] == 'a' && i > 0)
              if (cadena[i-2] == 'p' && i > 1)
                 cuen[2]++;
              else cuen[1]++;
           else cuen[0]++;
        else (*nopaz)++;
        i--;
        }
}
```

SOLUCIÓN:

La función `paz()` cuenta el número de ocurrencias de la palabra *paz* en la frase, el número de ocurrencias de la cadena *az* que no forman parte de la palabra *paz* y el número de veces que aparece la letra *z* sin formar parte de la palabra *paz*. La variable `j` contabiliza todas aquellas letras que no son *z*.

De este modo, el programa muestra el número de ocurrencias las cadenas *paz*, *az* y *z*. Además indica la longitud total de la cadena de caracteres excluyendo el carácter de fin de cadena.

Juego de Ensayo 1.

```
cadena: pazazz
```

i	nopaz	cuen[0]	cuen[1]	cuen[2]
5	0	1	0	0
4	0	1	1	0
3	1	1	1	0
2	1	1	1	1
1	2	1	1	1
0	3	1	1	1

```
Con 1 letras: 1.
Con 2 letras: 1.
Con 3 letras: 1.

Total: 6.
```

Juego de Ensayo 2.

```
cadena: haya paz
```

i	nopaz	cuen[0]	cuen[1]	cuen[2]
7	0	0	0	1
6	1	0	0	1
5	2	0	0	1
4	3	0	0	1
3	4	0	0	1
2	5	0	0	1
1	6	0	0	1
0	7	0	0	1

```
Con 1 letras: 0.
Con 2 letras: 0.
Con 3 letras: 1.

Total: 8
```

20. Explicar la función que realiza el siguiente programa.

```
#include <stdio.h>
#define NULO '\0'

char lia(int *, int, char *[]);

void main(argc, argv) {
int argc;
char *argv[]; {

  int s = 0;
  char c;

  if (argc > 1) {
     c = lia(&s, --argc, argv);
     printf("\n- Resultado: %f.", ((float) s) / argc);

     if (c != NULO)
        printf("\n- Bien!, es '%c'.", c);
     else
        printf("\n- Pues nooooo!");
     }
}
```

```
char lia(x, cuantos, lista) /* Función llamada desde main() */
int *x, cuantos;
char *lista[]; {

  int r = 1, ok = 0;
  int j;
  char primen = lista[1][0];

  for (; r <= cuantos; r++) {
      for (j = 0; lista[r][j] != NULO; j++);
      *x += j;
      if (lista[r][0] == primen) ok++;
      }

  if (ok == cuantos)
     return primen;
  else return NULO;
}
```

SOLUCIÓN:

El programa indica si todos los argumentos pasados al programa en la línea de ejecución, a excepción del nombre del mismo (que se almacena en `argv[0]`), comienzan por el mismo carácter, y en caso afirmativo cuál es dicho carácter. Además, calcula la longitud media de los argumentos que preceden al nombre del programa.

Juego de Ensayo 1.

```
lia uno una dos usted
```

Función lia

primen	lista	cuantos	r	j	x	ok
u	uno	4	1	3	3	1
	una	4	2	3	6	2
	dos	4	3	3	9	2
	usted	4	4	5	14	3

Principal

s	c	argc
0		5
14	0	4

```
- Resultado: 3.500000.
- Pues noooooo!
```

Juego de Ensayo 2.

```
lia uno una usar usted útil
```

Función lia

primen	lista	cuantos	r	j	x	ok
u	uno	5	1	3	3	1
	una	5	2	3	6	2
	usar	5	3	4	10	3
	usted	5	4	5	15	4
	útil	5	5	4	19	5

Principal

s	c	argc
0		6
19	'u'	5

```
- Resultado: 3.800000.
- Bien!, es 'u'.
```

21. Escribir una sentencia de llamada a la función `printf()` que escriba en la salida estándar, al principio de una nueva línea, la siguiente frase:

```
M.K. Gandhi, que fue llamado 'Mahatma', dijo: "No hay camino
para la paz; la paz es el camino".
```

Obsérvese que debe incluirse un tabulador al principio del texto. Por comodidad y para hacer un programa más legible se aconseja no utilizar más de ochenta caracteres en cada línea de programa.

SOLUCIÓN:

```
printf ("\n\tM.K. Gandhi, que fue llamado \'Mahatma\', dijo: \
\"No hay camino para la paz; la paz es el camino\".");
```

Observaciones:

1. Para empezar una nueva línea se usa el carácter `\n`.
2. Para colocar un tabulador se usa el carácter `\t`.
3. Para incluir un apóstrofo se utiliza la secuencia de escape `\'`.
4. Para colocar dobles comillas se usa la secuencia de escape `\"`.
5. Para romper una sentencia que no cabe entera en una línea y hacer que ésta continúe al principio de la línea siguiente se utiliza el carácter barra invertida (\).

22. Escribir una expresión que se pueda usar en un programa para calcular el número de elementos de un array de nombre `Libertad`.

SOLUCIÓN:

El número de elementos de un array será igual al resultado de dividir el tamaño del array completo entre el tamaño de uno de sus elementos. Para calcular esos tamaños en bytes se puede usar el operador de tiempo de compilación `sizeof`:

```
sizeof(Libertad) / sizeof(Libertad[0])
```

NOTA: Esta expresión no funcionará si el array se ha pasado como argumento en una función, ya que el tamaño global del array no se podrá calcular correctamente. Recuérdese que el operador `sizeof` funciona en tiempo de compilación y no de ejecución.

23. Escribir un fragmento de código en lenguaje C usando únicamente un bucle `for`, el cual lea caracteres de la entrada estándar e imprima por cada carácter leído la cadena de caracteres "Paz". El bucle deberá terminar cuando se lea el carácter 'P'.

SOLUCIÓN:

```
for (; getchar() != 'P'; )
    puts("Paz");
```

24. Escribir tres fragmentos de código en lenguaje C que sean equivalentes y que cada uno use una de las siguientes sentencias de control: `while`, `do-while` y `for`.

SOLUCIÓN:

Se trata de aplicar los tres tipos de bucles existentes en el lenguaje C para realizar una misma operación. A continuación se presentan los tres bucles en tres columnas:

```
i = 1;              i = 1;              for (i = 1; i < 5; i++);
while (i < 5)       do {
      i++;                 i++
                    } while (i < 5);
```

También se podría haber resuelto el ejercicio de una manera trivial, aunque poco elegante, utilizando bucles infinitos:

```
while (1);          do {                for (; ; ; );
                    } while (1);
```

25. Se va a realizar un programa que opera con vectores de dimensión tres (tridimensionales) en un espacio Euclídeo. Cada componente del vector es un número real (`double`). Las operaciones a realizar son:

a) Suma de vectores.
b) Resta de vectores.
c) Módulo de un vector.
d) Producto por un escalar real (double).
e) Producto escalar de dos vectores.

Se pide codificar en lenguaje C las funciones, con su encabezamiento y cuerpo correspondiente, que realizan cada una de estas operaciones. Para soportar la estructura de datos del vector se debe utilizar el tipo abstracto de datos ARRAY.

SOLUCIÓN:

```
#include <math.h>
#define DIM 3


void suma(double v1[], double v2[], double v3[]) {

  int i = 0;

  for (; i < DIM; i++)
      v3[i] = v1[i] + v2[i];
}

void resta(double v1[], double v2[], double v3[]) {

  int i = 0;

  for (; i < DIM; i++)
      v3[i] = v1[i] - v2[i];
}

double modulo(double v[]) {

  int i = 0;
  double suma = 0;

  for (; i < DIM; i++)
      suma += v[i] * v[i];

  return sqrt(suma);
}
```

```
void por_escalar(double v[], double n) {

  int i = 0;

  for (; i < DIM; i++)
      v[i] *= n;
}

float producto_escalar(double v1[], double v2[]) {

  int i = 0;
  double suma = 0;

  for (; i < DIM; i++)
      suma += v1[i] * v2[i];

  return suma;
}
```

26. El N.I.F. (Número de Identificación Fiscal) es exclusivo para cada persona y está formado por el número del D.N.I. (Documento Nacional de Identidad) y una letra añadida, la cual depende directamente del D.N.I.

La utilidad de esta letra consiste en permitir la detección de posibles errores al introducir un determinado D.N.I. Es simplemente un dígito de control de errores. Por ejemplo, al D.N.I. número 12.345.678 le corresponde la letra Z. Un programa que necesite almacenar este tipo de información deberá pedir el N.I.F. (D.N.I. y su letra asociada). Si al introducir este número nos equivocamos en un dígito introduciendo, por ejemplo, el número 12.645.678, nos daremos cuenta de que hemos cometido un error, ya que a éste último le corresponde la letra W y no la Z.

El algoritmo para obtener esta letra a partir del D.N.I. es el siguiente: Se calcula el resto de dividir el D.N.I. entre 23. Dicho resto está comprendido entre 0 y 22. Entonces, a cada uno de esos posibles restos se le asigna respectivamente las siguientes letras: T, R, W, A, G, M, Y, U, P, D, X, B, N, J, Z, S, Q, V, H, L, C, K y E.

Se ha escogido el número 23 por ser el número primo más grande que es menor que el número de letras en el abecedario. En español las letras son 28 y el siguiente número primo después del 23 es el 29. Además hay letras que no pueden asignarse por estar formadas por dos caracteres.

Se pide realizar un programa que lea del teclado un número de D.N.I. y escriba la letra del N.I.F. que le corresponde. Téngase en cuenta que el número del D.N.I. es muy grande y no puede almacenarse en una variable de tipo `int`.

SOLUCIÓN:

```
/************************************************/
/* Objetivo: Calcular la letra de control del NIF */
/* Entrada : Se pide el número de DNI             */
/* Salida  : La letra asociada a ese DNI          */
/************************************************Pp*/

#include <stdio.h>

void main()
{ char NIF;                  /* Letra a calcular */
  unsigned long int DNI;     /* DNI de entrada   */

  puts("\n-----> SEPA su N.I.F. <-----\n");
  printf("\t- Dime tu número de D.N.I.: ");
  scanf("%lu",&DNI);
```

```
  switch (DNI%23) {
         case 0:  NIF='T'; break;
         case 1:  NIF='R'; break;
         case 2:  NIF='W'; break;
         case 3:  NIF='A'; break;
         case 4:  NIF='G'; break;
         case 5:  NIF='M'; break;
         case 6:  NIF='Y'; break;
         case 7:  NIF='U'; break;
         case 8:  NIF='P'; break;
         case 9:  NIF='D'; break;
         case 10: NIF='X'; break;
         case 11: NIF='B'; break;
         case 12: NIF='N'; break;
         case 13: NIF='J'; break;
         case 14: NIF='Z'; break;
         case 15: NIF='S'; break;
         case 16: NIF='Q'; break;
         case 17: NIF='V'; break;
         case 18: NIF='H'; break;
         case 19: NIF='L'; break;
         case 20: NIF='C'; break;
         case 21: NIF='K'; break;
         case 22: NIF='E'; break;
         };
  printf("\n\t- La letra de su N.I.F. es: %c \n", NIF);
}
```

27. Diseñar un algoritmo mediante Pseudocódigo para multiplicar dos matrices cuyas dimensiones son (M x K) y (K x N). El resultado será una matriz de dimensión (M x N).

NOTA: Se supone que los datos de las matrices y sus dimensiones ya se han leído y están almacenados en las variables correspondientes.

SOLUCIÓN:

Para hallar los elementos de la matriz resultante tendremos que recorrer todas sus posiciones calculando el valor del elemento correspondiente a cada posición.

```
ENTRADA:     Matriz A[M][K] de Enteros
             Matriz B[K][N] de Enteros
             M, K, N: Enteros

SALIDA:      Matriz C[M][N] de Enteros

VARIABLES:   x, y, z, suma: Enteras

PARA x = 1 HASTA M CON PASO 1 HACER
     PARA y = 1 HASTA N CON PASO 1 HACER
          suma = 0;
          PARA z = 1 HASTA K CON PASO 1 HACER
               suma = suma + A[x][z] * B[z][y];
          FIN PARA
          C[x][y] = suma;
     FIN PARA
FIN PARA
```

28. Codificar en lenguaje C un programa que lea una palabra a través de la entrada estándar y la presente en pantalla con el formato a continuación se indica:

```
palabra
a     r
l     b
a     a
b     l
r     a
arbalap
```

También se pide el pseudocódigo del algoritmo correspondiente.

SOLUCIÓN:

Pseudocódigo:

```
ENTRADA:    Vector palabra[30] de Caracteres
SALIDA:     Vector palabra[30] de Caracteres
VARIABLES:  long, i, j: Enteras

ESCRIBIR <salto de línea>
ESCRIBIR "Teclea la palabra: "

LEER palabra

long = longitud(palabra)

ESCRIBIR palabra
ESCRIBIR <salto de línea>

PARA i = 1 HASTA (long - 2) CON PASO 1 HACER
     ESCRIBIR palabra[i]
     PARA j = 1 HASTA (long - 2) CON PASO 1 HACER
          ESCRIBIR " "
     FIN PARA
     ESCRIBIR palabra[long - i - 1]
FIN PARA

PARA i = long HASTA 0 CON PASO -1 HACER
     ESCRIBIR palabra[i]
FIN PARA
```

Codificación en lenguaje C:

```
/*************************************************/
/* Objetivo: Escribir una palabra según el formato */
/*           anterior.                           */
/* Entrada : Una cadena de caracteres leída por la */
/*           entrada estándar.                   */
/* Salida  : La palabra escrita con el formato     */
/*           deseado.                            */
/*************************************************/

#include <stdio.h>
#include <string.h>

void main () {

    char palabra[30];
    int long, i, j; /* Las variables i y j son contadores. */
```

```
    printf("\nTeclea la palabra: ");
    gets(palabra);
    long = strlen(palabra);

    /* Escribe el lado superior. */
    printf(palabra);
    putchar('\n');

    /* Escribe el lado izquierdo y derecho. */
    for (i = 1; i <= long - 2; i++) {
        printf("%c", palabra[i]);
        for (j = 1; j <= long - 2; j++)
            putchar(' ');
        printf("%c\n", palabra[long - i - 1]);
        }

    /* Escribe el lado inferior. */
    for (i = long - 1; i >= 0; i--)
        printf("%c", palabra[i]);
}
```

29. Escribir un función en C que determine si una matriz cuadrada de orden 3x3 es simétrica. Una matriz simétrica es una matriz cuadrada (tiene igual número de filas que de columnas) que coincide con su traspuesta. Se supone que los elementos de la matriz son números enteros. Además de la codificación en C, se pide la representación del algoritmo mediante Pseudocódigo. La función se realizará usando estructuras de bucle.

Para codificar la función se sugiere el siguiente prototipo:

```
int simetrica(int matriz[][3]);
```

SOLUCIÓN:

Pseudocódigo:

```
DEFINIR_MACRO TAM 3

Función SIMÉTRICA (M): Indica si la matriz M es simétrica.
M es una matriz de enteros de orden 3x3.

      VARIABLES:  i, j: Enteras
                  s: Booleana

      s = VERDADERO
      i = 0

      MIENTRAS (i < TAM) Y (s = VERDADERO)
           j = 0
           MIENTRAS (j < i) Y (s = VERDADERO)
                SI M[i][j] <> M[j][i] ENTONCES
                   s = FALSO
                FIN SI
                j = j + 1
           FIN MIENTRAS
           i = i + 1
      FIN MIENTRAS

      devolver s

FIN SIMÉTRICA
```

Codificación en lenguaje C:

```
/*********************************************/
/* Objetivo: Indicar si una matriz cuadrada de */
/*           orden 3x3 es simétrica.           */
/* Entrada : Una matriz cuadrada de orden 3x3. */
/* Salida  : Indicar si es simétrica.          */
/*********************************************/

#define TAM 3

int simetrica(int m[][TAM]) {

  int i, j;

  for (i = 0; i < TAM; i++)
      for (j = 0; j < i; j++)
          if (m[i][j] != m[j][i])
             return 0;
  return 1;
}
```

30. Diseñar un subalgoritmo en pseudocódigo que devuelva como resultado el factorial de un número que se le suministra como argumento. Dicho subalgoritmo deberá ser recursivo y basarse en la siguiente definición:

$$N! = 1 \qquad \text{si } N = 0$$
$$N! = N * (N - 1)! \qquad \text{si } N > 0$$

También se pide la codificación del subalgoritmo mediante una función escrita en lenguaje C.

SOLUCIÓN:

Pseudocódigo:

```
Función FACTORIAL (N): Calcula N! recursivamente.
N es un valor entero no negativo.

      SI N = 0
         ENTONCES devolver 1
      SI NO
         devolver (N * FACTORIAL(N - 1))
      FIN SI

FIN FACTORIAL
```

Codificación en lenguaje C:

```
/*************************************************/
/* Objetivo: Calcular el factorial de un número. */
/* Entrada : Un número N entero positivo.        */
/* Salida  : Factorial de N (N!).                */
/* Observaciones: Algoritmo recursivo.           */
/*************************************************/

unsigned long factorial (unsigned long N)
{
  if (N == 0)
     return 1;

  return (N * factorial(N - 1));
}
```

Obsérvese que la sentencia `if` no tiene sentencia `else` asociada, ya que en este caso no es necesario, aunque si se incluye la función también es válida. Recuérdese que cuando se ejecuta la sentencia `return` termina completamente la ejecución de la función, por lo que si se ejecuta la primera sentencia `return` no se ejecutará la segunda y viceversa.

31. Escribir un subalgoritmo en pseudocódigo que calcule el término N-simo de la sucesión de Fibonacci. Esta función deberá ser recursiva, basándose en que el primer término de la sucesión es cero, el segundo uno y cada término sucesivo es el resultado de la suma de los dos anteriores. Así, el principio de la sucesión es:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Si llamamos fibo (N) al término N-simo de la sucesión, éste puede definirse de este modo:

fibo (0) = 0	si N = 1
fibo (1) = 1	si N = 2
fibo (N) = fibo (N) + fibo (N - 1)	si N > 2

Codificar también una función en lenguaje C que implemente el anterior subalgoritmo recursivo.

SOLUCIÓN:

Pseudocódigo:

```
Función FIBO (N): Calcula el término N-simo de la sucesión
de Fibonacci recursivamente.
N es valor un entero mayor o igual a 1.

      SI N = 1
         ENTONCES devolver 0
      SI NO
         SI N = 2
            ENTONCES devolver 1
         SI NO
            devolver (FIBO(N - 1) + FIBO(N - 2))
         FIN SI
      FIN SI

FIN FIBO
```

Codificación en lenguaje C:

```
/***********************************************/
/* Objetivo: Calcular el término N-simo de la  */
/*           sucesión de Fibonacci.            */
/* Entrada : Un número N entero mayor que cero. */
/* Salida  : El término N-simo.                */
/* Observaciones: Algoritmo recursivo.         */
/***********************************************/

unsigned long fibo(unsigned long N) {

if (N == 1) return 0;
else if (N == 2)
        return 1;

return (fibo(N - 1) + fibo(N - 2));
}
```

32. Confeccionar el Pseudocódigo de un algoritmo que acepte un número por su entrada estándar e indique si es o no capicúa. El programa principal llamará a la función `capicua()`, a la que se le pasará como parámetro el número en cuestión y devolverá 1 si éste es capicúa y 0 si no lo es. También se pide la codificación en Lenguaje C.

Nota: Un número es capicúa si resulta indiferente el sentido en que éste se lea. Ej: 71922917.

SOLUCIÓN:

Pseudocódigo:

```
MÓDULO PRINCIPAL

      VARIABLES: num: Entera

      ESCRIBIR ("Introduce un número: ")
      LEER (num)

      SI CAPICÚA (VALOR_ABSOLUTO (num)) ENTONCES
           ESCRIBIR (num, " es capicúa.")
      SINO
           ESCRIBIR (num, " no es capicúa.")
      FIN SI

FIN PRINCIPAL


Función CAPICÚA (num): Indica si num es o no capicúa.
num es valor un entero.

      VARIABLES: dividendo, aux, resto: Enteras

      dividendo = num
      aux = 0

      MIENTRAS dividendo > 0
           resto = módulo(dividendo / 10)
           aux = aux * 10 + resto
           dividendo = dividendo / 10
      FIN MIENTRAS

      SI aux = num ENTONCES
           devolver VERDADERO
      SINO
           devolver FALSO
      FIN SI

FIN CAPICÚA
```

Codificación en lenguaje C:

```
/*****************************************************/
/* Objetivo: Indicar si el número que se suministra */
/*           como parámetro es o no capicúa.         */
/* Entrada : Un número entero.                       */
/* Salida  : Indicar si es capicúa.                  */
/*****************************************************/

#include <stdio.h>
#include <conio.h>
#include <math.h>
```

```
int capicua(int); /* prototipo */


void main(){

  int num;

  clrscr();
  printf("Introduce un número: ");
  scanf("%d", &num);
  putchar('\n');

  if (capicua(abs(num)))
     printf("%d es capicúa.\n", num);
  else
     printf("%d no es capicúa.\n", num);
}


int capicua(int num){

  int resto, dividendo, aux;

  dividendo = num;
  aux = 0;

  while (dividendo > 0) {
        resto = dividendo % 10;
        aux = aux * 10 + resto;
        dividendo /= 10;
        }

  if (aux == num)
     return 1;
  else
     return 0;
}
```

33. Escribir un programa en C que lea por teclado un máximo de cincuenta frases y posteriormente las muestre centradas en la pantalla. Cada frase tendrá una longitud máxima de ochenta caracteres incluido el carácter de fin de cadena. La lectura finalizará cuando se lea la frase número cincuenta o se introduzca el carácter <RETURN> como único carácter de una frase. El programa deberá usar las tres estructuras repetitivas existentes en C. También se pide el desarrollo del algoritmo en pseudocódigo.

SOLUCIÓN:

Pseudocódigo:

```
MÓDULO PRINCIPAL

      VARIABLES:  Matriz frases[50][80] de Caracteres
                  contador, i: Enteras

      contador = 0

      REPETIR
          contador = contador + 1
          ESCRIBIR contador, ": "
          LEER frases[contador - 1]
      MIENTRAS (contador <= 50) Y
               (frases[contador - 1][0] <> <RETURN>)
```

```
        PARA i = 0 HASTA (contador - 1) CON PASO 1 HACER
             ESCRIBIR_CENTRO (frases[i])
        FIN PARA

FIN PRINCIPAL


Función ESCRIBIR_CENTRO (cadena): Muestra centrada en la
pantalla la cadena de caracteres que se suministra como
argumento a la función.

        VARIABLES: long, espacios: Enteras

        long = longitud(cadena)
        espacios = (80 - long) / 2

        MIENTRAS espacios > 0
            ESCRIBIR " "
            espacios = espacios - 1
        FIN MIENTRAS

        ESCRIBIR cadena
        ESCRIBIR <salto de línea>

FIN ESCRIBIR_CENTRO
```

Codificación en lenguaje C:

```
/*******************************************************/
/* Objetivo: Leer un conjunto de frases a través del */
/*           teclado y mostrarlas centradas en la    */
/*           pantalla.                               */
/* Observación: Usa las tres estructuras repetitivas */
/*              existentes en C.                     */
/*******************************************************/

#include <stdio.h>
#include <string.h>

void escribir_centro(char *cadena);

void main() {

  int contador = 0, i = 0;
  char frases[50][80];
       /* matriz donde se guarda cada frase */

  /* uso del bucle do-while */
  /* el primer índice de un array siempre es cero. */

  do {
     contador++;
     printf ("%d: ", contador);
     gets(frases[contador - 1]);
     }
  while (contador <= 50 && frases[contador - 1][0]);

  /* uso del bucle for */
  for (; i < contador; i++)
      escribir_centro(frases[i]);
}
```

```
void escribir_centro (char *cadena) {

  int long = strlen(cadena),
      espacios = (80 - long) / 2;
      /* espacios que deben escribirse antes de la frase para
         que quede centrada */

  /* uso del bucle while */
  while (espacios > 0) {
        putchar(' ');
        espacios--;
        }

  printf("%s\n", cadena);
}
```

34. Escribir una función en C que indique si una cadena de caracteres es un palíndromo. Una frase es un palíndromo si se lee igual hacia delante que hacia atrás. Deben ignorarse los espacios en blanco. La función aceptará un único argumento que será la cadena de caracteres y devolverá uno si dicha cadena es un palíndromo y cero si no lo es.

Ejemplo: "dábale arroz a la zorra el abad"

Representar también el algoritmo en Pseudocódigo.

SOLUCIÓN:

Pseudocódigo:

```
Función PALÍNDROMO (cad): Indica si la cadena de caracteres cad
es un palíndromo.

      VARIABLES: i, len: Enteras
                 p: Booleana
      i = 0
      p = VERDADERO

      < eliminar los espacios blancos de cad >
      len = < lugar del último elemento de cad >

      MIENTRAS (p = VERDADERO) y (i < len - i)
           SI cad[i] <> cad[len-i]
              p = FALSO
           FIN SI
           i = i + 1
      FIN MIENTRAS

      devolver p

FIN PALÍNDROMO
```

Codificación en lenguaje C:

```
/************************************************/
/* Objetivo: Indicar si una cadena de caracteres */
/*           es un palíndromo.                   */
/* Entrada : Una cadena de caracteres.           */
/* Salida  : Indicar si es un palíndromo.        */
/************************************************/
```

```
#include <stdio.h>
#include <string.h>

#define TAM 25


int palindromo(char cad[]) {

  int len = 0, i = 0;
  char c[TAM]; /* guarda la cadena original sin los espacios */

  /* quitamos los espacios blancos */
  for (; i < strlen(cad); i++)
      if (cad[i] != ' ')
         c[len++] = cad[i];

  c[len--]= '\0';

  /* comprobar si son iguales los elementos */
  for (i = 0; i < len - i; i++)
      if (c[i] != c[len-i])
         return 0;

  return 1;
}
```

35. Codificar una función que acepte dos cadenas de caracteres como argumentos y copie el contenido de la segunda en la primera invirtiendo el orden de los caracteres. Por ejemplo, si la segunda cadena vale

```
"Lo esencial es invisible a los ojos."
```

tras la llamada a la función, la primera cadena valdrá

```
".sojo sol a elbisivni se laicnese oL"
```

SOLUCIÓN:

```
void invierte(char dest[], char orig[]) {
  int i = strlen(orig) - 1, j = 0;

  for (; i >= 0; j++, i--)
      dest[j] = orig[i];

  dest[j] = '\0';
}
```

La función `strlen()`, cuyo prototipo se encuentra en el fichero de encabezamiento `string.h`, devuelve la longitud de una cadena de caracteres excluyendo el carácter fin de cadena (`'\0'`).

Obsérvese que la asignación que aparece en el cuerpo del bucle `for` es también válida si se invierten los índices:

```
  for (; i >= 0; j++, i--)
      dest[i] = orig[j];
```

Esta función, junto con la función estándar de biblioteca para comparar cadenas de caracteres denominada `strcmp()`, se puede utilizar para averiguar si una cadena es un palíndromo. Obsérvese, por ejemplo, que si la segunda cadena es

```
"reconocer"
```

tras la llamada a la función, la primera cadena también tendrá como contenido

```
"reconocer"
```

Evidentemente, la función `strcmp()` indicará, mediante la devolución del valor cero, que ambas cadenas son iguales. Por lo tanto se trata un palíndromo.

36. La función C `strcmp()` (*String Comparison*) compara dos cadenas de caracteres (str1 y str2), cuyos punteros se suministran como argumentos. El resultado de la comparación es:

- cero si las cadenas son idénticas.
- menor que cero si la diferencia entre los dos primeros caracteres que son distintos es negativa.
- mayor que cero si la diferencia entre los dos primeros caracteres que son distintos es positiva.

Se pide, en primer lugar, diseñar el algoritmo correspondiente a dicha función y representarlo mediante Pseudocódigo. Finalmente deberá codificarse la función en Lenguaje C.

Ejemplos:

```
strcmp("abc", "abc")    ->    0
strcmp("", "")          ->    0
strcmp("Abc", "Abd")    ->    -1
strcmp("Abc", "Abb")    ->    1
strcmp("ABC", "AB")     ->    67
strcmp("AB", "ABC")     ->    -67
```

SOLUCIÓN:

a) Pseudocódigo.

```
Función COMPARAR (o[], d[]): Compara dos cadenas de caracteres
carácter a carácter.
o y d son cadenas de caracteres.

      VARIABLES: i, leno, lend: Enteras

      leno = longitud(o)
      lend = longitud(d)
      i = 0

      MIENTRAS (o[i] = d[i]) Y (i < leno) Y (i < lend)
           i = i + 1
      FIN MIENTRAS

      devolver o[i] - d[i]

FIN COMPARAR
```

b) Codificación en Lenguaje C.

```
/************************************************/
/* Objetivo: Comparar dos cadenas de caracteres. */
/* Entrada : Dos cadenas de caracteres.          */
/* Salida  : Indicar si son iguales o distintas. */
/************************************************/


#include <string.h>
```

```
int comparar(char o[], char d[]){

  int i, leno, lend;

  leno = strlen(o);
  lend = strlen(d);

  for (i = 0; o[i] == d[i] && i < leno && i < lend; i++);

  return(o[i] - d[i]);
}
```

37. Codificar una función en lenguaje C que acepte dos cadenas de caracteres como parámetros (`s1` y `s2`) y compruebe si la segunda cadena `s2` es una subcadena de la primera cadena `s1`. La función devolverá un entero que indicará la posición del primer carácter de `s2` dentro de `s1`, o bien devolverá -1 si `s2` no está contenida en `s1`.

El encabezamiento de la función es el siguiente:

```
int busca(char s1[], char s2[]);
```

Ej: s1: "Imagina que hay una GUERRA y no vamos NADIE"
s2: "que"

La función devolverá el entero 9.

SOLUCIÓN:

```
/*************************************************/
/* Objetivo: Comprobar si s2 es una subcadena de  */
/*           s1.                                  */
/* Entrada : Dos cadenas de caracteres.           */
/* Salida  : Un número entero.                    */
/*************************************************/


#include <stdio.h>
#include <conio.h>
#include <string.h>


int busca(char s1[], char s2[]) {

  int i, j, pos, long_s1, long_s2;
  /* i es el contador de s1 y j es el contador de s2 */

  i = 0;
  pos = -1;
  long_s1 = strlen(s1);
  long_s2 = strlen(s2);

  /* s2 debe tener menor o igual longitud que s1 */
  if (long_s2 > long_s1) return -1;

  while (pos == -1 && i < long_s1) {
        j = 0;
```

```
        /* localizar la primera letra coincidente */
        while (i < long_s1 && s1[i] != s2[j])
              i++;

        if (i < long_s1) { /* coincide la primera letra */
           pos = i;
           /* comprobar si coinciden todas las letras de s2 */
           while (s1[i] == s2[j] && i < long_s1 && j < long_s2) {
                 i++;
                 j++;
                 }
           /* s2 no contenida en s1 */
           if (j < long_s2) pos = -1;
           }
        }

  return pos;
}
```

38. Escribir una función en lenguaje C que reciba como parámetros una cadena de caracteres que representa el nombre de una entidad, otra cadena de caracteres donde debe devolver como resultado el acrónimo de la anterior y un tercer parámetro que indique la longitud máxima de la cadena de salida. Si la longitud del acrónimo resultante es superior a la de la cadena de salida, se truncará el resultado. También debe representarse la lógica del algoritmo mediante Pseudocódigo.

Ejemplo: Cadena 1: `Sociedad Española de Transportistas Autónomos`
Cadena 2: `S.E.D.T.A.`
Longitud de la Cadena 2: 15 caracteres.

Nota: Para la conversión de caracteres de mayúsculas a minúsculas puede utilizarse la función `toupper(c)`, cuyo prototipo se encuentra en `ctype.h`. El argumento `c` es el carácter alfabético a convertir. La función devuelve el carácter en mayúsculas.

SOLUCIÓN:

a) Pseudocódigo.

```
Función ACRÓNIMO (s[], acr[], long_acr): Obtiene en acr el
acrónimo de la entidad cuyo nombre está en s.
s y acr son cadenas de caracteres.
n es un entero positivo.

      VARIABLES: i, j: Enteras

      i = 0
      j = 0

      MIENTRAS (i < longitud (s)) y (j < long_acr - 1)
           < Localizar primera letra de palabra >

           acr[j] = s[i]
           j = j + 1
           acr[j] = '.'
           j = j + 1

           < Buscar fin de palabra >
      FIN MIENTRAS

FIN ACRÓNIMO
```

b) Codificación en Lenguaje C.

```
/************************************************/
/* Objetivo: Obtener el acrónimo de una entidad.  */
/* Entrada : Una cadena de caracteres y un entero */
/*           positivo.                            */
/* Salida  : Una cadena de caracteres.            */
/************************************************/

#include <string.h>
#include <ctype.h>

void acronimo(char s[], char acr[], int long_acr) {

  int i = 0, j = 0;

  while (i < strlen(s) && j < long_acr - 1) {
        /* letra inicial de palabra */
        while (i < strlen(s) && s[i] == ' ')
              i++;
        if (i < strlen(s)) {
           acr[j++] = toupper(s[i]);
           acr[j++] = '.';
           }
        /* en busca de otra palabra */
        while (i < strlen(s) && s[i] != ' ')
              i++;
        }
  acr[j] = '\0';
}
```

39. Escribir una función en C que admita dos argumentos: un entero positivo en base diez y un puntero a una cadena de caracteres (string). Dicha función convertirá el número entero positivo en su equivalente en notación hexadecimal.

El encabezamiento de la función es: `char *hexadec(int, char[]);`

La función devolverá como resultado el puntero a la cadena de caracteres donde se almacena el valor hexadecimal. También debe representarse el algoritmo mediante Pseudocódigo.

SOLUCIÓN:

a) Pseudocódigo.

```
Función HEXADEC (n, cad[]): Convierte en hexadecimal un entero
positivo.
n es un entero positivo y cad una cadena de caracteres.

      VARIABLES: i: Entera

      i = 0

      MIENTRAS n > 15
           SI módulo (n, 16) > 9 ENTONCES
              cad[i] = 'A' + módulo (n, 16) - 10
           SINO
              cad[i] = '0' + módulo (n, 16)
           FIN SI
           i = i + 1
           n = n / 16
      FIN MIENTRAS
```

```
        SI n > 9 ENTONCES
           cad[i] = 'A' + n - 10
        SINO
           cad[i] = '0' + n
        FIN SI

        <Invertir la cadena>

FIN HEXADEC
```

b) Codificación en Lenguaje C.

```
/*****************************************************/
/* Objetivo: Pasar un entero positivo a hexadecimal. */
/* Entrada : Un entero positivo.                     */
/* Salida  : Una cadena de caracteres.               */
/*****************************************************/

#include <string.h>

char *hexadec(int n, char cad[]) {

  int j = 0, i = 0;
  char aux;

  /* Introducir los restos */
  while (n > 15) {
        cad[i++] = (n % 16) > 9 ? 'A' + (n % 16) - 10:
                                  '0' + (n % 16);
        n /= 16;
        }

  /* Introducir el cociente */
  cad[i] = n > 9 ? 'A' + n - 10: '0' + n;
  cad[i + 1] = '\0';

  /* Invertir la cadena */
  for (; j < i; j++, i--) {
      aux = cad[j];
      cad[j] = cad[i];
      cad[i] = aux;
      }
  return cad;
}
```

40. Codificar en lenguaje C un programa al cual se le pase el nombre de un fichero de texto como argumento en línea de ejecución. El programa deberá mostrar por pantalla el contenido del fichero.

SOLUCIÓN:

```
/*****************************************************/
/* Objetivo: Mostrar por pantalla el contenido de un */
/*           fichero de texto.                       */
/* Entrada : El nombre de un fichero de texto.       */
/* Salida  : El contenido del fichero.               */
/*****************************************************/


#include <stdio.h>
#include <stdlib.h>
```

```
void main(int cant_arg, char *vector_arg[]) {

    FILE *fichero;
    char caracter;

    if (cant_arg != 2) {
        printf("CANTIDAD DE ARGUMENTOS ERRÓNEA");
        exit(1);
       }

    /* Apertura del fichero y comprobación de errores. */
    fichero = fopen(vector_arg[1], "r");
    if (fichero == NULL) {
       printf("ERROR AL ABRIR FICHERO");
       exit(1);
       }

    /* Proceso de lectura y escritura por pantalla. */
    caracter = getc(fichero);
    while (caracter != EOF) {
          putchar(caracter);
          caracter = getc(fichero);
          }

    /* La siguiente función comprueba si se ha producido algún
       error. En la lectura anterior, si se produce un error se
       genera un carácter EOF. */
    if (ferror(fichero))
       printf("ERROR AL LEER FICHERO");

    /* Cierre del fichero. */
    fclose(fichero);
}
```

41. Escribir un programa en lenguaje C que calcule la suma, resta, multiplicación o división de dos fracciones cuyos numeradores y denominadores se introduzcan por teclado.

SOLUCIÓN:

```
/*******************************************************/
/* Objetivo: Realizar una de las cuatro operaciones  */
/*           aritméticas básicas con dos fracciones. */
/* Entrada : Los numeradores y denominadores de las  */
/*           fracciones operandos.                   */
/* Salida  : El numerador y el denominador de la     */
/*           fracción resultado.                     */
/*******************************************************/


#include <stdio.h>


typedef struct { /* definición del tipo de dato FRACCIÓN */
  int numerador;
  int denominador;
} FRACCION;

void presentar_menu(void);
void calcular(int seleccion);
/* función que realiza el cálculo según la operación seleccionada */
```

```
void main() {

  int opcion;

  do {
     presentar_menu();
     scanf("%d", &opcion);
     calcular(opcion);
     }
     while (opcion != 5);
}


void presentar_menu(void) {

  printf("\nOperaciones con fracciones:\n\n");
  printf("1: SUMAR\n");
  printf("2: RESTAR\n");
  printf("3: MULTIPLICAR\n");
  printf("4: DIVIDIR\n");
  printf("5: Salir del programa.\n\n");
  printf("Introduzca el número de opción --> ");
}


void calcular(int seleccion) {

  FRACCION f1, f2, f3;

  if (seleccion < 5 && seleccion >= 1) {
     printf("\nFracción 1¦ (numerador denominador): ");
     scanf("%d %d", &f1.numerador, &f1.denominador);
     printf("\nFracción 2¦ (numerador denominador): ");
     scanf("%d %d", &f2.numerador, &f2.denominador);

     switch (seleccion) {
            case 1: f3.numerador = f1.numerador * f2.denominador +
                                   f1.denominador * f2.numerador;
                    f3.denominador = f1.denominador * f2.denominador;
                    break;
            case 2: f3.numerador = f1.numerador * f2.denominador -
                                   f1.denominador * f2.numerador;
                    f3.denominador = f1.denominador * f2.denominador;
                    break;
            case 3: f3.numerador = f1.numerador * f2.numerador;
                    f3.denominador = f1.denominador * f2.denominador;
                    break;
            case 4: f3.numerador = f1.numerador * f2.denominador;
                    f3.denominador = f1.denominador * f2.numerador;
                    break;
            }

     printf("\nResultado = %d / %d", f3.numerador, f3.denominador);
     }
}
```

42. Escribir un programa que calcule un *cuadrado mágico* de orden impar. Un cuadrado mágico es una matriz de dimensión n x n, cuyos elementos son los números naturales comprendidos entre 1 y n sin repetirse ninguno y cuyas filas, columnas y diagonales suman la misma cantidad.

Ejemplo:

```
n = 3)
                    8     1     6
                    3     5     7
                    4     9     2
```

SOLUCIÓN:

El método consiste en colocar el número uno en la primera fila y columna central, el siguiente en la fila superior y columna derecha con respecto a la posición anterior y así sucesivamente. Si la nueva posición estuviera ya ocupada, el nuevo número se colocaría en la fila inferior e igual columna con respecto a la posición ocupada.

a) Pseudocódigo.

```
Programa CUADRADO: Construye un cuadrado mágico de dimensión n.

      VARIABLES: f, c, k: Enteras

      leer (n)

      < Inicializar matriz >
      < Posicionarse en la primera fila y columna central >

      PARA k = 1 HASTA n*n CON PASO 1 HACER
           m[f][c] = k
           < Decrementar f >
           < Incrementar c >
           SI < m[f][c] = ocupado > ENTONCES
              < Deshacer incrementos y decrementos anteriores >
              < Incrementar f >
           FIN SI
      FIN PARA

      < Escribir matriz >

FIN CUADRADO
```

b) Codificación en Lenguaje C.

```
/************************************************/
/* Objetivo: Construir un cuadrado mágico de    */
/*           dimensión n.                       */
/* Entrada : Un entero positivo impar.          */
/* Salida  : Imprimir la matriz cuadrada.       */
/************************************************/


#include <stdio.h>
#include <conio.h>


#define MAX 16


int leerdim(void); /* prototipos */

void inic(int m[][MAX], int),
     escribir(int m[][MAX], int);
```

```
void main() {

  unsigned n, f, c, k;
  unsigned m[MAX][MAX];

  clrscr();

  n = leerdim();

  inic(m, n);

  for  (k = 1, f = 0, c = n/2; k <= n*n; k++) {
       m[f][c] = k;

       if (f == 0) /* decrementar fila */
          f = n - 1;
       else f--;

       if (c == n - 1) /* incrementar columna */
          c = 0;
       else c++;

       if (m[f][c] != 0) { /* posición ocupada */
          if (f == n - 1) /* deshacer incr. y decr. */
             f = 0;
          else f++;

          if (c == 0)
             c = n - 1;
          else c--;

          if (f == n - 1) /* fila inferior */
             f = 0;
          else f++;
          }
       }

  escribir(m, n);
}


int leerdim() {

  int n = MAX;

  while (n <= 0 || n >= MAX || n % 2 == 0) {
        printf("Dimensión impar positiva menor que %d: ", MAX);
        scanf("%d", &n);
        }

  return n;
}


void inic(int m[][MAX], int n) {

  int f, c;

  for (f = 0; f < n; f++) /* inicializar la matriz */
      for (c = 0; c < n; c++)
          m[f][c] = 0;
}
```

```
void escribir(int m[][MAX], int n) {

  int f, c;

  for (f = 0; f < n; f++) { /* mostrar la matriz */
      for (c = 0; c < n; c++)
          printf ("%d ", m[f][c]);
      putchar ('\n');
      }
}
```

43. **El Triángulo de Tartaglia:** Niccolò Fontana (1499-1557) fue un matemático italiano más conocido como Tartaglia (el tartamudo). Siendo niño recibió una herida durante el saqueo de su ciudad natal (Brescia) por las tropas de Gastón de Foix (1512). Esta herida le impidió articular bien las palabras durante toda su vida y de ahí viene su famoso sobrenombre de Tartaglia. Entre sus estudios más destacados se encuentra uno sobre la teoría de las ecuaciones de tercer grado. Pero quizás el trabajo más famoso ha sido el denominado *Triángulo de Tartaglia.*

El Triángulo de Tartaglia es una tabla triangular de números enteros positivos dispuestos en filas horizontales. La fila 1 tiene 2 elementos que son ambos igual a 1. La fila N tiene un elemento más que la N-1 y se consigue poniendo como primer y último elemento la unidad y como elementos intermedios los números que se obtienen de la suma de los dos números contiguos de la línea anterior, de forma ordenada:

```
Fila
----
1       1    1
2       1    2     1
3       1    3     3     1
4       1    4     6     4     1
5       1    5    10    10     5     1
6       1    6    15    20    15     6     1
7       1    7    21    35    35    21     7     1
8       1    8    28    56    70    56    28     8     1
        ...       ...         ...         ...         ...
```

A veces, se considera que existe la fila cero que tiene un único elemento que es la unidad.

La utilidad de esta peculiar tabla radica en que representa los coeficientes de las potencias sucesivas de (a + b). Esto es, los elementos de la fila N son ordenadamente los coeficientes del desarrollo de $(a + b)^N$. El orden de los coeficientes es muy importante y se debe mantener asociado a las potencias sucesivas de a y de b. Con unos ejemplos se verá más claro:

$$(a + b)^1 = a + b$$

$$(a + b)^2 = a^2 + 2\,a\,b + b^2$$

$$(a + b)^3 = a^3 + 3\,a^2\,b + 3\,a\,b^2 + b^3$$

$$(a + b)^4 = a^4 + 4\,a^3 b + 6\,a^2 b^2 + 4\,a\,b^3 + b^4$$

$$(a + b)^5 = a^5 + 5\,a^4 b + 10\,a^3 b^2 + 10\,a^2 b^3 + 5\,a\,b^4 + b^5$$

$$(a + b)^6 = a^6 + 6\,a^5 b + 15\,a^4 b^2 + 20\,a^3 b^3 + 15\,a^2 b^4 + 6\,a\,b^5 + b^6$$

Podemos concluir, por tanto, que si (x_0, x_1, ... x_N) son los N+1 coeficientes de la fila **N** del Triángulo de Tartaglia:

$$x_0 = x_N = 1$$
$$x_1 = x_{N-1} = N$$
$$x_i = x_{N-i}$$

Además, sabemos que el desarrollo de $(a + b)^N$ tiene N+1 sumandos y que en cada uno de ellos la potencia de a sumada a la potencia de b es siempre igual a N. O sea, que $(a + b)^N$ es igual a la sumatoria desde i = 0 hasta i = N de $x_i a^{N-i} b^i$:

$$(a+b)^N = a^N + N\,a^{N-1}\,b + x_2\,a^{N-2}\,b^2 + \ldots + x_i\,a^{N-i}\,b^i + \ldots + \\ + x_{N-2}\,a^2\,b^{N-2} + N\,a\,b^{N-1} + b^N$$

El problema consiste en hacer un programa en lenguaje C que nos calcule la potencia $(a + b)^N$ para un N que leerá de la entrada estándar. El programa leerá N, calculará la fila N-ésima del triángulo de Tartaglia y después mostrará el desarrollo de $(a + b)^N$ con los coeficientes obtenidos y las potencias de a y b adecuadas.

El programa estará leyendo valores de N y mostrando los resultados hasta que se introduzca un valor de N negativo. Además aceptará el valor de N=0, sabiendo que cualquier número elevado a cero vale uno:

$$(a+b)^0 = 1$$

Para simplificar, podemos suponer un máximo para la potencia N. Así declararemos una constante simbólica (macro) con ese valor máximo. Por ejemplo:

```
#define MAXPOTENCIA 34
```

que indicará que como máximo el programa podrá realizar hasta el desarrollo de $(a + b)^{34}$.

SOLUCIÓN:

```
/*********************************************************/
/* Objetivo: Dar la solución general para una potencia  */
/*       de (a + b): (a + b)^N  (N-ésima potencia)      */
/* Observaciones: Se empleará el método del Triángulo   */
/*           de Tartaglia. Se empleará un único array.  */
/*           El array contendrá los valores de los      */
/*           elementos de la fila N-ésima del Triángulo */
/* Requisitos: Existe un número máximo para N, definido */
/*           como una macro: MAXPOTENCIA.               */
/* Fecha: 10 de Noviembre de 1996.                      */
/*********************************************************Pp*/


#include <stdio.h>

#define MAXPOTENCIA 34 /* Máxima potencia N admitida */


void Imprime_Resultado(unsigned long int R[], int N);
```

```
void main()
{ unsigned long int coef[MAXPOTENCIA+1];
                  /* Coeficientes de las potencias */
  int N,          /* Potencia de (a + b) a calcular  */
      Ncalculado, /* Potencia ya calculada          */
      Mitad,      /* El Triángulo de Tartaglia es simétrico,
                     por lo que basta calcular la mitad de
                     cada fila */
      ind;        /* Índice que indica el elemento que
                     estamos calculando en cada momento en
                     una fila del Triángulo */
  unsigned long int aux1, aux2;
                  /* Elementos consecutivos de la fila i que
                     se suman para hallar un elemento de la
                     fila i+1 del Triángulo */

  puts ("\n*****************************************");
  puts ("*     Cálculo de potencias de (a + b)     *");
  puts ("*                (a + b)^N                *");
  puts ("*     Método: Triángulo de TARTAGLIA.     *");
  puts ("*****************************************Pp*");

  while (1) { /* Termina cuando se lee un N negativo */
        printf ("\n* Rango de potencias (N) entre 0 y %u.\n",
                 MAXPOTENCIA);
        printf ("* Dame la potencia N (-1 para fin): ");
                 scanf ("%i", &N);

        if (N < 0) {
           puts ("\n*****  FIN  *****");
           return; /* Fin del programa!!! */
           }
        if (N > MAXPOTENCIA) continue;

        printf ("\n (a+b)^%i = ", N);

        if (N == 0) {
           puts ("1");
           continue;
           }
        if (N == 1) {
           puts ("a + b");
           continue;
           }

        coef[0] = coef[1] = 1; /* Fila 1 */
        Ncalculado = 1;

        /* Este bucle calcula los coeficientes de (a+b)^N */
        while (Ncalculado < N) {
              /* Calcular coeficientes de Ncalculado + 1 */
              Mitad = ++Ncalculado / 2;
              coef[Ncalculado] = aux1 = 1;
              for (ind = 1; ind <= Mitad; ind++) {
                  aux2 = coef[ind];
                  coef[ind] = coef[Ncalculado-ind] = aux1 +
                              aux2;
                  aux1 = aux2;
                  }
              }
        Imprime_Resultado(coef, N);
        }
} /* main */
```

```
/*********************************************/
/* Objetivo: Imprime los resultados de la po-  */
/*           tencia de (a+b) calculada, cuyos  */
/*           coeficientes se dan en un array   */
/* Entrada: R: Array de enteros positivos con  */
/*             los coeficientes de (a+b)^N     */
/*             calculados por el triángulo de  */
/*             Tartaglia.                      */
/*          N: Entero, con la potencia hallada */
/* Requisitos: El array R tendrá como mínimo   */
/*          N elementos (el 0 y el N-1 serán 1)*/
/*********************************************/

void Imprime_Resultado (unsigned long int R[], int N)
{ int i;

  printf ("a^%i", N);
  for (i=1; i < N; i++) {
      printf (" + %lua", R[i]);
      if (N-i != 1)
         printf ("^%i ", N - i);
      printf ("b");
      if (i != 1)
         printf ("^%i", i);
      }
  printf (" + b^%i\n", N);
}
```

El algoritmo propuesto tiene una limitación bastante significativa, ya que limita el número máximo de la potencia N al valor `MAXPOTENCIA`. Al estar este límite establecido, podemos definir un array de `MAXPOTENCIA + 1` elementos. En dicho array almacenaremos sucesivamente el valor de las filas del Triángulo de Tartaglia.

Para solucionar esa limitación proponemos como ejercicio resolver este mismo problema empleando estructuras dinámicas de datos (una lista) en vez de un array.

Téngase en cuenta que pueden producirse problemas de desbordamiento en los tipos de datos empleados. Así, para el tipo de dato `unsigned long int`, se puede calcular como máximo $(a+b)^{34}$, ya que para N=35 se supera el valor máximo para este tipo, que es 4294967296 (suponiendo treinta y dos bits para este tipo de dato). Este problema se puede solventar en parte utilizando tipos de datos que admitan un rango mayor, como el tipo `long double`, que suele emplear 128 bits.

No obstante, cualquier tipo de dato estándar se quedará pequeño en cuanto aumentemos la potencia N lo suficiente. Para solventar ese problema podremos usar otras estructuras dinámicas de datos para almacenar cada elemento del Triángulo de Tartaglia.

Por esto no tendría demasiado sentido usar una estructura dinámica de datos para almacenar una fila del Triángulo y no hacerlo para representar los enteros de cada uno de los elementos de esa fila. Es decir, no tiene sentido que el programa no nos limite el número de filas que podamos calcular si el tipo de dato estándar que empleamos no nos va a permitir almacenar los coeficientes resultantes. Por ejemplo, nos permitirá intentar calcular la fila N-ésima con, N=15000, pero no nos permitirá almacenar los coeficientes más grandes de $(a+b)^{15000}$.

Aclaraciones al programa anterior:

- El programa termina cuando lee un número N negativo. Al ejecutarse la sentencia `return` en el programa principal el programa termina.

- Si N vale cero o uno el resultado se imprime directamente para simplificar el algoritmo.

- El Triángulo de Tartaglia es simétrico, por lo que no es necesario calcular todos los elementos de una fila, sino que basta con calcular la mitad. La variable con el nombre `Mitad` indica el número de elementos que tenemos que calcular cada vez. Obsérvese que cuando se le asigna un valor a esta variable se usa el operador de división entera.

- Se podría haber optimizado aún más el algoritmo anterior, ya que los unos de los extremos de cada fila no es necesario almacenarlos.

Preguntas de Test

1. Un algoritmo es

   a) un programa.
   b) el código fuente de un programa escrito en un lenguaje de programación específico.
   c) un conjunto finito de operaciones no ambiguas para resolver un problema determinado.
   d) un conjunto de instrucciones codificadas en un lenguaje de alto nivel para que puedan ser compiladas posteriormente.
   e) Ninguna de las respuestas anteriores es correcta.

2. Un Lenguaje de Programación es

   a) un conjunto de símbolos y reglas utilizados para construir un programa.
   b) un lenguaje en el que sólo existen dos valores posibles: cero y uno.
   c) un método usado para gestionar los recursos del ordenador.
   d) un programa que hay que compilar para ser ejecutado.
   e) Ninguna de las respuestas anteriores es correcta.

3. Los lenguajes de programación pueden clasificarse en

   a) lenguajes máquina y lenguajes simbólicos.
   b) intérpretes y compiladores.
   c) lenguajes formales y de alto nivel.
   d) lenguajes ensambladores y de alto nivel.
   e) Ninguna de las respuestas anteriores es correcta.

4. Un lenguaje de programación consta de

   a) compilador e intérprete.
   b) programa fuente y programa objeto.
   c) sentencias declarativas e imperativas.
   d) sentencias iterativas y recursivas.

5. Los tipos de sentencias de un lenguaje de programación de alto nivel pueden ser

   a) funcionales y descriptivas.
   b) aritméticas y lógicas.
   c) secuenciales, selectivas e iterativas.
   d) funcionales, de asignación y operacionales.
   e) entero, real y carácter.
   f) imperativas y declarativas.
   g) Ninguna de las respuestas anteriores es correcta.

6. Una sentencia declarativa es

   a) una sentencia de asignación.
   b) una sentencia de selección.
   c) una sentencia de Entrada/Salida.
   d) un bucle.
   e) Ninguna de las respuestas anteriores es correcta.

7. ¿Qué es un programa? (Se entiende informático, no de TV)

a) La parte de la unidad de control (UC) que ejecuta las instrucciones.
b) Una secuencia de instrucciones escritas para que las ejecute el ordenador.
c) Una secuencia de pasos para obtener un fin concreto.
d) El software para controlar y procesar textos.
e) El software que nos ayuda a gestionar y administrar la memoria.
f) Todo lo que se almacena en la memoria RAM.
g) Todas las respuestas anteriores son correctas.
h) Ninguna de las respuestas anteriores es correcta.

8. Si queremos es que un determinado programa se ejecute rápidamente será mejor

a) traducirlo con un traductor.
b) interpretarlo con un intérprete.
c) compilarlo con un compilador.
d) ejecutarlo en un sistema operativo.
e) programarlo en un lenguaje de alto nivel.
g) Ninguna de las respuestas anteriores es correcta.

9. Cuando hablamos de compiladores, intérpretes o aplicaciones nos estamos refiriendo a

a) programas de la memoria ROM.
b) las fases de la ejecución de los programas en la CPU.
c) etapas en la construcción de los programas.
d) hardware.
e) software.
f) redes de computadoras.
g) procesamiento paralelo.

10. Indicar la afirmación correcta:

a) Los traductores pueden ser compiladores o intérpretes.
b) Los compiladores pueden ser traductores o intérpretes.
c) Los intérpretes pueden ser traductores o compiladores.
d) Los compiladores pueden ser ensambladores o máquina.
e) Los compiladores generan código fuente y objeto.
f) Todas las respuestas anteriores son correctas.
g) Ninguna de las respuestas anteriores es correcta.

11. La diferencia fundamental entre compiladores e intérpretes es

a) que el compilador genera código objeto.
b) que el intérprete genera código objeto.
c) que el compilador depende del Sistema Operativo.
d) no hay diferencias ya que ambos son traductores.
e) Ninguna de las respuestas anteriores es correcta.

12. El código máquina es

a) el código que usan los ordenadores para controlar los programas en ejecución.
b) el conjunto de ceros y unos necesarios para almacenar un fichero en disco.
c) el código ASCII.
d) un código de control de la CPU.
e) un código bivaluado que utilizan los ordenadores para procesar la información.

13. Un programa creado con un lenguaje máquina

a) normalmente será menos eficiente (en ocupación de memoria y tiempo de ejecución) que si se escribe con un lenguaje de alto nivel.
b) si lo modificamos, aunque sea una sola instrucción, hay que compilarlo de nuevo.
c) tendrá instrucciones declarativas para indicar el tipo de las variables que se usan.
d) Todas las respuestas anteriores son correctas.
e) Ninguna de las respuestas anteriores es correcta.

14. Se dice que un programa es eficiente

a) cuando se ejecuta en un intervalo de tiempo pequeño.
b) cuando requiere poca memoria para ser ejecutado.
c) cuando optimiza el algoritmo utilizado para ser programado en lenguaje máquina con mayor facilidad.
d) cuando el algoritmo utilizado emplea el menor número de variables posible.
e) cuando el programa tiene el menor número de módulos posible.
f) cuando aprovecha lo mejor posible los recursos disponibles, minimizando el uso de la memoria y de la CPU.

15. Una variable es

a) un objeto que posee un valor y es conocido en un programa o un algoritmo por un identificador.
b) el identificador de un valor constante en un programa.
c) un valor entero que puede variar a lo largo de la ejecución de un programa.
d) un dato desconocido en un programa.
e) Ninguna de las respuestas anteriores es correcta.

16. Una estructura estática de datos es aquélla en la que

a) sus datos no varían.
b) se define su posición en la memoria antes de la ejecución y ésta no varía durante la misma.
c) se define su tamaño en memoria antes de la ejecución con una sentencia declarativa.
d) se define un conjunto variable de elementos de tamaño fijo.
e) se define un conjunto fijo de N elementos de tamaño variable.
f) Todas las respuestas anteriores son correctas.
g) Ninguna de las respuestas anteriores es correcta.

17. Una estructura dinámica de datos es aquélla en la que

a) el espacio ocupado en memoria puede variar a lo largo de la ejecución del programa.
b) se define un conjunto variable de elementos que podemos crear o destruir pidiendo y devolviendo memoria al sistema.
c) no se declara su tamaño en tiempo de compilación.
d) Todas las respuestas anteriores son correctas.
e) Ninguna de las respuestas anteriores es correcta.

18. El teorema de Bomh-Jacopini se basa en

a) programar primero lo más general y luego descender poco a poco a más detalles.
b) escribir primero el algoritmo (en pseudocódigo o diagrama de flujo) y luego codificar el programa en un lenguaje de programación.
c) utilizar sólo tipos de datos simples: entero, real, carácter, etc.
d) escribir un programa utilizando sólo las estructuras de control secuencial, selectiva e iterativa.
e) Todo lo anterior.
f) Ninguna de las respuestas anteriores es correcta.

19. Los tipos de datos básicos en el estándar ANSI C son

a) `int` y `char`.
b) `int`, `char` y `float`.
c) `int`, `char`, `float` y `double`.
d) `int`, `char`, `float`, `double` y `void`.
e) `unsigned`, `signed`, `long` y `short`.
f) `auto`, `extern`, `static` y `register`.

20. Los especificadores de clase de almacenamiento utilizados en C en la declaración de variables son

a) `int` y `char`.
b) `int`, `char` y `float`.
c) `int`, `char`, `float` y `double`.
d) `int`, `char`, `float`, `double` y `void`.
e) `unsigned`, `signed`, `long` y `short`.
f) `auto`, `extern`, `static` y `register`.

21. Los modificadores de los tipos de datos básicos en lenguaje C son

a) `int` y `char`.
b) `int`, `char` y `float`.
c) `int`, `char`, `float` y `double`.
d) `int`, `char`, `float`, `double` y `void`.
e) `unsigned`, `signed`, `long` y `short`.
f) `auto`, `extern`, `static` y `register`.

22. En un programa escrito en C el resultado de la operación `5/2` es

a) 25.
b) $25 * 10^{-1}$.
c) 2.0.
d) 2.
e) Ninguna de las respuestas anteriores es correcta.

23. En un programa escrito en C el resultado de la operación `5./2` es

a) 2.5.
b) $25 * 10^{-2}$.
c) 2.0.
d) 2.
e) Ninguna de las respuestas anteriores es correcta.

24. Las variables globales en un programa

a) se deben usar siempre que se pueda, ya que así se evita el paso de parámetros entre funciones.
b) se deben usar siempre, salvo cuando sea necesario usar variables locales.
c) es preferible usarlas sólo cuando sea necesario, ya que resulta más difícil controlar que sus valores no se modifiquen erróneamente.
d) no se pueden usar en un programa en C, ya que crean conflictos irresolubles con las variables locales.
e) Ninguna de las respuestas anteriores es correcta.

25. Si dentro de una función en lenguaje C tenemos declarada una variable con el mismo identificador que una variable global y en una expresión perteneciente a la función anterior usamos dicho identificador, ¿Qué variable se usará en tal expresión?

a) la global, ya que las variables globales tienen preferencia.
b) la local, ya que las variables locales tienen preferencia.
c) depende de la expresión en la que se use y del tipo de ambas variables.
d) Ninguna de las respuestas anteriores es correcta.

26. ¿Cuál es la salida del siguiente programa?

```
#include <stdio.h>

int a = 1;

void main() {

  int a = 0;

  printf("\nLa variable \"a\" vale: %d. ", a);
  return;
}
```

a) Uno, porque las variables globales siempre tienen preferencia ante las locales.
b) Cero, porque al ser la variable `a` local a la función `main()`, dentro de dicha función la declaración local prevalece sobre la global.
c) Se trata de un error, ya que no pueden existir dos variables con el mismo identificador, aunque tengan distinto ámbito.
d) Ninguna de las respuestas anteriores es correcta.

27. La expresión en lenguaje C: `a = b++;`

a) asigna a la variable `a` el valor de la variable `b` dejando ésta última intacta.
b) asigna a la variable `a` el valor de `b + 1` dejando la variable `b` intacta.
c) asigna a la variable `a` el valor de `b + 1` y modifica también con ese valor la variable `b`.
d) asigna a la variable `a` el valor de la variable `b` y luego incrementa el valor de `b` en una unidad.
e) Ninguna de las respuestas anteriores es correcta.

28. La expresión en lenguaje C: `a = ++b;`

a) asigna a la variable a el valor de la variable `b` dejando ésta última intacta.
b) asigna a la variable `a` el valor de `b + 1` dejando la variable `b` intacta.
c) asigna a la variable `a` el valor de `b + 1` y modifica también con ese valor la variable `b`.
d) asigna a la variable `a` el valor de la variable `b` y luego incrementa el valor de `b` en una unidad.
e) Ninguna de las respuestas anteriores es correcta.

29. En la sentencia `switch` del lenguaje C

a) las opciones precedidas por la palabra reservada `case` deben estar ordenadas y la opción por defecto (`default`) debe estar en último lugar.
b) las opciones precedidas por la palabra reservada `case` no tienen que estar ordenadas pero la opción por defecto (`default`) debe estar forzosamente en último lugar.
c) las opciones precedidas por la palabra reservada `case` deben estar ordenadas aunque la opción por defecto (`default`) no tiene que estar forzosamente en último lugar.
d) las opciones precedidas por la palabra reservada `case` no tienen porqué estar ordenadas y la opción por defecto (`default`) no tiene que estar forzosamente en último lugar.

30. Si en el cuerpo de un bucle de tipo:

```
while (test) {
  ...
      }
```

se usa la sentencia de control `continue;`

a) se ejecutará la siguiente sentencia dentro del cuerpo del bucle.
b) se ejecutará la siguiente sentencia que haya después de este bucle.
c) se evaluará la expresión de test para continuar o no ejecutando el bucle.
d) se ejecutará de nuevo el bucle independientemente del valor de la expresión de test.
e) se ejecutará la última instrucción que haya dentro del cuerpo del bucle.
f) Ninguna de las respuestas anteriores es correcta.

31. Si en el cuerpo de un bucle de tipo:

```
while (test) {
  ...
      }
```

se usa la sentencia de control `break;`

a) se ejecutará la siguiente sentencia dentro del cuerpo del bucle.
b) se ejecutará la siguiente sentencia que haya después de este bucle.
c) se evaluará la expresión de test para continuar o no ejecutando el bucle.
d) se ejecutará de nuevo el bucle independientemente del valor de la expresión de test.
e) se ejecutará la última instrucción que haya dentro del cuerpo del bucle.
f) Ninguna de las respuestas anteriores es correcta.

32. Si en el cuerpo de un bucle de tipo:

```
for (inicialización; test; actualización) {
  ...
    }
```

se usa la sentencia de control `continue;`

a) se ejecutará la siguiente sentencia dentro del cuerpo del bucle.
b) se ejecutará la siguiente sentencia que haya después de este bucle.
c) se evaluará la expresión de test para continuar o no ejecutando el bucle.
d) se ejecutará de nuevo el bucle independientemente del valor de la expresión de test.
e) se evaluará la expresión de actualización y después se evaluará la expresión de test.
f) Ninguna de las respuestas anteriores es correcta.

33. Si en el cuerpo de un bucle de tipo:

```
for (inicialización; test; actualización) {
  ...
    }
```

se usa la sentencia de control `break;`

a) se ejecutará la siguiente sentencia dentro del cuerpo del bucle.
b) se ejecutará la siguiente sentencia que haya después de este bucle.
c) se evaluará la expresión de test para continuar o no ejecutando el bucle.
d) se ejecutará de nuevo el bucle independientemente del valor de la expresión de test.
e) se evaluará la expresión de actualización y después se evaluará la expresión de test.
f) Ninguna de las respuestas anteriores es correcta.

34. El símbolo & en el lenguaje C

a) sirve para modificar el valor de una variable.
b) es el operador de dirección.
c) indica el paso de argumentos a una función.
d) Todas las respuestas anteriores son correctas.
e) Ninguna de las respuestas anteriores es correcta.

35. En un programa en C

a) los arrays se pasan siempre por valor.
b) se pasa siempre por valor la dirección de memoria del primer elemento del array.
c) se pasan por valor todos los elementos del array.
d) se pasan por referencia las direcciones de memoria de todos los elementos del array.
e) Ninguna de las respuestas anteriores es correcta.

36. Si en un programa en C tenemos declarado un array de 10 enteros cuyo identificador es `Leo`, la expresión `(*Leo + 3)` hace referencia a

a) el valor del cuarto elemento del array.
b) el resultado de sumar 3 al puntero al primer elemento del array.
c) el resultado de sumar 3 al puntero al cuarto elemento del array.
d) el resultado de sumar 3 al valor del primer elemento del array.
e) el resultado de sumar 3 al valor del cuarto elemento del array.
f) Ninguna de las respuestas anteriores es correcta.

37. Si en un programa en C tenemos declarado un array de 10 enteros cuyo identificador es `Leo`, la expresión `*(Leo + 3)` hace referencia a

a) el valor del cuarto elemento del array.
b) el resultado de sumar 3 al puntero al primer elemento del array.
c) el resultado de sumar 3 al puntero al cuarto elemento del array.
d) el resultado de sumar 3 al valor del primer elemento del array.
e) el resultado de sumar 3 al valor del cuarto elemento del array.
f) Ninguna de las respuestas anteriores es correcta.

38. Si en un programa en C tenemos declarado un array llamado `Fender` de n posiciones

a) el valor de su identificador `Fender` es igual a `Fender[1]`.
b) el valor de su identificador `Fender` es igual a `Fender[0]`.
c) el valor de `*Fender` es igual a `Fender[1]`.
d) el valor de `*Fender` es igual a `Fender[0]`.
e) Ninguna de las respuestas anteriores es correcta.

39. Si en un programa en C tenemos declarado un array llamado `Fender` de n posiciones

a) el valor de `(Fender + n)` es igual a `Fender[n]`.
b) el valor de `(Fender + n)` es igual a `Fender[n-1]`.
c) el valor de `*(Fender + n - 1)` es igual a `Fender[n]`.
d) el valor de `*(Fender + n)` es igual a `Fender[n-1]`.
e) el valor de `*(Fender + n - 1)` es igual a `Fender[n-1]`.
f) Ninguna de las respuestas anteriores es correcta.

40. Una variable declarada en C como `char *argv[];` como parámetro formal de la función `main()` es

a) un puntero a un array de un carácter.
b) una estructura de punteros.
c) una cadena de caracteres.
d) un array sin ningún elemento.
e) el puntero al primer elemento de un array de caracteres.
f) Ninguna de las respuestas anteriores es correcta.

41. En un programa en lenguaje C, la lectura de un fichero

a) depende del soporte donde esté situado éste, ya que C es muy potente y permite optimizar el acceso a cada dispositivo según su tipo.
b) no depende del soporte donde esté situado éste, salvo que dicho soporte sea un CD-ROM, en cuyo caso hay que leer el fichero de manera especial.
c) a la función de lectura de ficheros utilizada hay que indicarle explícitamente el tipo de soporte donde se encuentra el fichero a leer.
d) no depende del soporte donde esté situado. El S.O. se encarga de ocultar el modo de lectura propio de cada tipo de dispositivo.
e) Ninguna de las respuestas anteriores es correcta.

Respuestas

1. c)
2. a)
3. a)
4. c)
5. f)
6. e)
7. b)
8. c)
9. e)
10. a)
11. a)
12. e)
13. e)
14. f)
15. a)
16. c)
17. d)
18. d)
19. d)
20. f)
21. e)
22. d)
23. a)
24. c)
25. b)
26. b)
27. d)
28. c)
29. d)
30. c)
31. b)
32. e)
33. b)
34. b)
35. b)
36. d)
37. a)
38. d)
39. e)
40. f)
41. d)

Bibliografía

[1] J. Galindo, P. Sánchez, A. Yáñez, M.J. del Jesus, J.J. Aguilera, J.M. Rodríguez, A. Sánchez, J.F. Argudo. *Fundamentos Informáticos*. Servicio de Publicaciones. Universidad de Cádiz, 1996.

[2] P. Norton. *Introducción a la Computación*. McGraw-Hill, 1995.

[3] A. Prieto, A. Lloris, J.C. Torres. *Introducción a la Informática*. Segunda Edición. McGraw-Hill, 1995.

[4] J.M. Rodríguez, J. Galindo. *Aprendiendo C*. Servicio de Publicaciones. Universidad de Cádiz, 1996.

[5] H. Schildt. *C: Manual de Referencia*. Tercera Edición. McGraw-Hill, 1988.

[6] A.S. Tanembaum. *Sistemas Operativos. Diseño e Implementación*. Prentice Hall Internacional Ltd., 1991.

[7] M. Waite, S. Prata, D. Martin. *Programación en C. Introducción y conceptos avanzados*. Segunda Edición. Anaya Multimedia, S.A., 1989.

SERVICIO•DE•PUBLICACIONES
UNIVERSIDAD•DE•CADIZ
1997

ISBN 84-7786-464-0