SOLUCIONES COMENTADAS AL EXAMEN DE LABORATORIO ESTRUCTURAS DE LOS COMPUTADORES SEPTIEMBRE 1.997

```
1)
;
;
*********************************************************************************
;
; A continuación definimos el segmento de datos.
;
;
*********************************************************************************
;
DATOS SEGMENT
	Terminar		EQU 4C00h
	LeerCadena		EQU 0Ah		; Función de la INT 21h
	EscribirCaracter	EQU 02h		; Función de la INT 21h
	EscribirCadena		EQU 09h		; Función de la INT 21h
	CLAVE			DB "APROBARECUANDOLOSSAPOSBAILENFLAMENCO"
	CADENA			DB 81, 0	; Estructura requerida por la función 0Ah de la INT 21h
				DB 81 DUP (0)	; para dejar los datos de la cadena
	MsgPedir		DB "Introduzca una frase, por favor: ", 10, 13, '$'
	MsgSalida		DB "La frase codificada queda: ", 10, 13, '$'
	OrdenARestar		EQU 40h
	LongitudClave		EQU 36
	ENTER			DB 10, 13, '$'	; Cadena de texto que simula un ENTER
	ESPACIO			EQU 20h
DATOS ENDS
;
;
*********************************************************************************
;
; A continuación definimos el segmento de pila.
;
;
*********************************************************************************
;
PILA SEGMENT STACK
	DB 1024 DUP (" ")
PILA ENDS
;
;
*********************************************************************************
;
; A continuación definimos el segmento de datos.
;
;
*********************************************************************************
;
CODIGO SEGMENT
	ASSUME CS:CODIGO, SS:PILA, DS:DATOS
;
;
*********************************************************************************
;
; A continuación definimos el procedimiento principal que llama a los siguientes procedimientos:
;		LeerFrase:	Solicita y lee por teclado una cadena máxima de 80 caracteres.
;		Codificar:	Codifica la frase que se ha leído.
;		DarEnter:	Produce un salto de línea por pantalla.
;		EscribirFrase:	Escribe la frase codificada por pantalla.
; Parámetros que pasa:	NINGUNO.
; Parámetros que recibe:	NINGUNO.
```

```
;
;
*************************************************************************************
;
Principal PROC FAR
      MOV AX, DATOS
      MOV DS, AX
      CALL LeerFrase      ; Llamamos al  procedimiento para leer la cadena de caracteres.
      CALL Codificar      ; Llamamos al  procedimiento para codificar la frase.
      CALL DarEnter       ; Damos un ENTER.
      CALL EscribirFrase  ; Llamamos al  procecimiento de escribir la frase codificada.
      MOV AX, Terminar    ; Indicamos al DOS que deseamos terminar el programa.
      INT 21h
Principal ENDP
;
;
*************************************************************************************
;
; PROCEDIMIENTO:     LeerFrase.
; OBJETIVOS:         Leer una cadena de 80 caracteres como máximo Deja la cadena leída en
;                    la estructura CADENA. Emplea la función 0Ah de la INT 21h.
; Parámetros que pasa:    NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
*************************************************************************************
;
LeerFrase PROC NEAR
      PUSH DX             ; Guardamos en la pila el contenido de los registros que vamos a
      PUSH AX             ; emplear en el procedimiento
      LEA DX, MsgPedir    ; Imprimimos la frase que le solicita la frase al usuario. La dirección en DS:DX
      MOV AH, EscribirCadena      ; La función 09 de la INT 21h
      INT 21h
      LEA DX, CADENA              ; Cargamos en DS:DX la dirección de la Cadena a leer por la función
      MOV AH, LeerCadena          ; 0Ah de la INT 21h
      INT 21h
      POP AX              ; Recuperamos el contenido que tenían los registros antes de llamar al
      POP DX              ; procedimiento.
      RET
LeerFrase ENDP
;
;
*************************************************************************************
;
; PROCEDIMIENTO:     Codificar.
; OBJETIVOS:         Codifica la frase leída almacenada en CADENA con la clace almacenada en
;                    CLAVE.
; Parámetros que pasa:    NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
*************************************************************************************
;
Codificar PROC NEAR
      PUSH AX             ; Guardamos el contenido de los registros que se van a emplear en
      PUSH BX             ; el procedimiento.
      PUSH CX
      PUSH DX
      PUSH SI
      LEA BX, CADENA  ; Cargamos en DS:BX  la dirección de la CADENA leída.
```

```
    LEA SI, CLAVE       ; Cargamos en DS:SI la dirección de la CLAVE almacenada.
    INC BX              ; Incrementamos el puntero BX de forma que podamos
    XOR CX, CX          ; almacenar en el registro CX el número de caracteres que se
    MOV CL, [BX]        ; han leído realmente y que se encuentra en la segunda posición de CADENA.
    MOV DH, LongitudClave       ; Indicamos en DH la longitud de la clave (36 caracteres).
    INC BX              ; Incrementamos BX para acceder al primer caracter leído.
  BucleCodificar:
    MOV DL, [BX]        ; Guardamos en DL el contenido del caracter que se desea codificar.
    MOV AH, [SI]        ; Guardamos en AH el caracter de la clave.
    SUB AH, OrdenARestar        ; Calculamos el  núemro de orden del caracter de la clave de la
                                ; forma: Código_ASCII_Caracter - 40h
    ADD DL, AH          ; Sumamos al caracter que deseamos codificar el número de orden calculado.
    MOV [BX], DL        ; Dejamos el resultado de nuevo en CADENA.
    INC BX              ; Incrementamos BX para apuntar al siguiente caracter a codificar.
    DEC DH              ; Decrementamos el contador de caracteres de la CLAVE.
    JZ VuelveAClave     ; Si DH = 0. He llegado al final de la clave, tengo que iniciarla de nuevo.
    INC SI              ; En otro caso incremento SI para apuntar al siguiente caracter de la CLAVE,
    JMP FinBucle        ; Salto al final del bucle.
  VuelveAclave:
    LEA SI, CLAVE       ; Si se ha terminado la CLAVE, vuelvo a poner en DS:SI la dirección de
                        ; comienzo de la CLAVE.
    MOV DH, LongitudClave       ; Pongo en DH la longitud de la CLAVE (36 caracteres)
  FinBucle:
    LOOP BucleCodificar         ; Si CX  no es 0 salto a BucleCodificar.
    POP SI                      ; Recupero el contenido de los registros que
    POP DX                      ; teníamos antes de llamar al procedimiento.
    POP CX
    POP BX
    POP AX
    RET
Codificar ENDP
;
;
********************************************************************************
;
; PROCEDIMIENTO:    EscribirFrase.
; OBJETIVOS:        Escribe la frase codificada por pantalla.
; Parámetros que pasa:    NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
********************************************************************************
;
EscribirFrase PROC NEAR
    PUSH AX             ; Guardamos los registros que vamos a emplear.
    PUSH CX
    PUSH DX
    LEA DX, MsgSalida           ; Cargamos en DS:DX la dirección del mensaje de salida que
    MOV AH, EscribirCadena      ; imprimiremos por pantalla mediante la función 09 de la INT 21h.
    INT 21h
    LEA BX, CADENA              ; Cargo en DS:BX la dirección de la CADENA ya codificada.
    INC BX                      ; Incrementamos BX para acceder al número de caracteres
    XOR CX, CX                  ; que tiene la cadena realmente y de esa forma inicializar el
    MOV CL, [BX]                ; contador de CX
    MOV AH, EscribirCaracter    ; Imprimiremos la cadena codificada caracter a caracter mediante el
                                ; uso de la función 02 de la INT 21h
  BucleEscribir:
    INC BX                      ; Accedemos al primer caracter de la cadena codificada.
    MOV DL, [BX]                ; Llevamos ese caracter a DL (necesario para poder imprimirse)
    INT 21h
```

```
        MOV DL, ESPACIO; Ponemos en DL el código del espacio en blanco.
        INT 21h                 ; y lo imprimimos.
        LOOP BucleEscribir      ; Si CX  no es 0 saltamos a BucleEscribir.
        POP DX                  ; Recuperamos el contenido que tenían los registros antes
        POP CX                  ; de la llamada al procedimiento.
        POP AX
        RET
EscribirFrase ENDP
;
;
********************************************************************************
;
; PROCEDIMIENTO:    DarEnter.
; OBJETIVOS:        Produce un salto de línea por pantalla.
; Parámetros que pasa:  NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
********************************************************************************
;
DarEnter PROC NEAR
        PUSH AX         ; Guardamos los registros que vamos a emplear
        PUSH DX
        MOV AH, EscribirCadena  ; Indicamos en AH que deseamos emplear la función 09 de la
                                ; INT 21H para imprimir una cadena de caracteres terminada en $.
        LEA DX, ENTER           ; Ponemos en DS:DX  la dirección de la cadena a imprimir.
        INT 21h
        POP DX          ; Recuperamos el contenido que tenían los registros antes de la
        POP AX          ; llamada al procedimiento
        RET
DarEnter ENDP
CODIGO ENDS
END Principal

2)
;
;
********************************************************************************
;
; A continuación definimos el segmento de datos.
;
;
********************************************************************************
;
DATOS SEGMENT
        Terminar        EQU 4C00h
        LeerCaracter    EQU 01h         ; Función de la INT 21h
        LeerCadena      EQU 0Ah         ; Función de la INT 21h
        EscribirCaracter EQU 02h        ; Función de la INT 21h
        EscribirCadena  EQU 09h         ; Función de la INT 21h
        ENTER           DB  10, 13, '$'     ; Cadena de texto que simula un ENTER
        MsgPedir        DB "Introduce un número de dos cifras hexadecimales: ", 10, 13, '$'
        NumeroLeido     DW  0000
        NumeroTemp      DB  00
        NumeroDec       DW  0000
        SerieFibonacci  DW 0, 1         ; Estructura de memoria para almacenar el máximo número
                        DW 12 DUP (0)   ; de términos de la serie de Fibonacci inferiores a 255 (FFh).
        NumTerminos     DW 14
        CERO            EQU 30h
        UNO             EQU 31h
```

```
        ESPACIO         EQU 20h
        EsLetra         EQU 3Ah
        EsALetra        EQU 0Ah
        BitsInferiores  EQU 0Fh
DATOS ENDS
;
;
;*****************************************************************************************
;
; A continuación definimos el segmento de pila.
;
;
;*****************************************************************************************
;
PILA SEGMENT STACK
        DB 1024 DUP (" ")
PILA ENDS
;
;
;*****************************************************************************************
;
; A continuación definimos el procedimiento principal que llama a los siguientes procedimientos:
;       LeerNumero:             Solicita y lee por teclado un número de dos cifras hexadecimales.
;       DarEnter:               Produce un salto de línea por pantalla.
;       Fibinacci:              Calcula la serie de Fibonacci para el número leído.
;       ConvertirADecimal:      Convierte los terminos de la serie de Fibonacci en números en
                                base diez, es decir, decimales. (NO SE PEDÍA EN EL EXAMEN).
;       EscribirSerire:         Escribe la serie de Fibonacci por pantalla.
; Parámetros que pasa:  NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
;*****************************************************************************************
;
CODIGO SEGMENT
        ASSUME CS:CODIGO, SS:PILA, DS:DATOS
Principal PROC FAR
        MOV AX, DATOS
        MOV DS, AX
        CALL LeerNumero
        CALL DarEnter
        CALL Fibonacci
        CALL ConvertirADecimal
        CALL EscribirSerie
        MOV AX, Terminar
        INT 21h
Principal ENDP
;
;
;*****************************************************************************************
;
; PROCEDIMIENTO:    LeerNúmero.
; OBJETIVOS:        Solicitar y leer un número hexadecimal de dos cifras por teclado.
; LLAMA A:          ConvertirANumero
; Parámetros que pasa:  AL. Código ASCII del número leído por teclado.
; Parámetros que recibe:  NINGUNO.
;
;
;*****************************************************************************************
;
```

```
LeerNumero PROC NEAR
      PUSH AX             ; Guardamos el valor de los registros que vamos a emplear.
      PUSH DX
      MOV AH, EscribirCadena    ; Indicamos mediante la función 09 de la INT 21h que
      LEA DX, MsgPedir          ; se introduzca una cifra de dos dígitos hexadecimales.
      INT 21h
      MOV AH, LeerCaracter      ; Leemos la cifra mediante la función 01 de la INT 21h
      INT 21h
      CALL ConvertirANumero     ; Solicitamos la conversión de ASCII a número
      MOV DH, AL
      INT 21h                   ; Leemos la segunda cifra.
      CALL ConvertirANumero     ; Solicitamos la conversión de ASCII a número
      MOV DL, AL
      SHL DH, 1                 ; Desplazamos la primera cifra 4 veces a la izquierda.
      SHL DH, 1
      SHL DH, 1
      SHL DH, 1
      ADD DH, DL                ; Juntamos las dos cifras en una sola, con lo que se obtiene el
                                ; número leído.
      MOV DL, DH
      XOR DH, DH
      MOV NumeroLeido, DX       ; Guardamos en Nuemro leído la cifra leída.
      POP DX                    ; Recuperamos el contenido que tenían los registros antes de
      POP AX                    ; llamar al procedimiento.
      RET
LeerNumero ENDP
;
;
********************************************************************************************
;
; PROCEDIMIENTO:    DarEnter.
; OBJETIVOS:        Produce un salto de línea por pantalla.
; Parámetros que pasa:    NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
********************************************************************************************
;
DarEnter PROC NEAR
      PUSH AX             ; Guardamos los registros que vamos a emplear
      PUSH DX
      MOV AH, EscribirCadena    ; Indicamos en AH que deseamos emplear la función 09 de la
                                ; INT 21H para imprimir una cadena de caracteres terminada en $.
      LEA DX, ENTER             ; Ponemos en DS:DX  la dirección de la cadena a imprimir.
      INT 21h
      POP DX              ; Recuperamos el contenido que tenían los registros antes de la
      POP AX              ; llamada al procedimiento
      RET
DarEnter ENDP
;
;
********************************************************************************************
;
; PROCEDIMIENTO:    Fibonacci.
; OBJETIVOS:        Calcular la serie de Fibonacci para el valor leído.
; Parámetros que pasa:    NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
********************************************************************************************
```

```
;
Fibonacci PROC NEAR
      PUSH AX             ; Guardamos los registros que vamos a emplear.
      PUSH BX
      PUSH CX
      PUSH DX
      XOR BX, BX
      XOR CX, CX
      INC CL              ; En CX indiaremos de cuantos términos se compone la serie.
  BucleFibonacci:
      XOR DX, DX
      XOR AX, AX
      INC CL              ; Indicamos que la serie consta de un término más.
      MOV DX, SerieFibonacci[BX]          ; Guardamos en DX el término n-1
      INC BX
      INC BX
      MOV AX , SerieFibonacci[BX]         ; Guardamos en AX el termino n
      ADD DX, AX                          ; Calculamos el término n+1
      CMP DX, NumeroLeido                 ; Miramos si el valor obtenido es mayor que el
      JA Final                            ; número que hemos introducido por teclado.
      INC BX                              ; Actualizamnos BX  hasta que apunte a la dirección
      INC BX                              ; donde se almacenará el término n+1
      MOV SerieFibonacci[BX], DX          ; Guardamos el término n+1 de la serie.
      DEC BX                              ; Decrementamos BX  hasta apuntar al término que en
      DEC BX                              ; siguiente iteración del bucle será el n-1.
      JMP BucleFibonacci                  ; Saltamos al BucleFibonacci
  Final:
      MOV NumTerminos, CX                 ; Guardamos el número de términos de la serie.
      POP DX                              ; Recuperamos el contenido que tenían los registros
      POP CX                              ; antes de llamar al procedimiento.
      POP BX
      POP AX
      RET
Fibonacci ENDP
```

```
;
;
*********************************************************************************
;
; PROCEDIMIENTO:     ConvertirADecimal.  (NO SE PEDÍA EN EL EXAMEN)
; OBJETIVOS:         Convierte el valor de cada término de la serie calculado antes en su
;                    correspondiente valor decimal
; LLAMA A:           Tratar2.
; Parámetros que pasa:   AL.
; Parámetros que recibe: NINGUNO.
;
;
*********************************************************************************
;
ConvertirADecimal PROC NEAR
    PUSH AX                 ; Guardamos el contenido de los registros que vamos a emplear
    PUSH BX
    PUSH CX
    PUSH DX
    XOR AX, AX              ; Inicializamos a cero los registros que vamos a usar.
    XOR BX, BX
    XOR CX, CX
    XOR DX, DX
    MOV CX, NumTerminos     ; Ponemos en el CX  número de términos de la serie calculada.
    LEA BX, SerieFibonacci  ; Cargamos en DS:BX la dirección donde se encuentra la serie.
  BucleSerie:
    XOR DX, DX              ; Inicializamos DX y AX a cero.
    XOR AX, AX
    MOV AX,[BX]             ; Llevamos el primer término de la serie al registro AX
    MOV DL, 0Ah             ; y dividimos por 10 (0Ah). En realidad estamos aplicando la
    DIV DL                  ; regla de Horner para el cambio entre bases.
    CMP AL, 0Ah             ; Si el cociente es mayor que 10 (0Ah) vuelvo a dividir.
    JB Tr2                  ; en otro caso ya tengo las dos cifras del número.
    XOR DX, DX              ; Inicializamos DX a cero.
    MOV DL, AH              ; Guardamos el resto de la división en DH.
    PUSH DX                 ; Guardamos el resto en la pila.
    MOV DL, 0Ah             ; Dividimos el cociente obtenido antes (AL)
    XOR AH, AH              ; Por 10 (0Ah)
    DIV DL
    CALL Tratar2            ; Llamamos al procedimiento Tratar2 para juntar las dos cifras.
    POP DX                  ; Recuperamos el resto de la primera división.
    SHL AX, 1               ; Desplazamos cuatro veces AX con lo que tendremos las
    SHL AX, 1               ; centenas y las decenas del número.
    SHL AX, 1
    SHL AX, 1
    ADD AX, DX              ; Sumamos las unidades al número anterior.
    JMP FinBucleSerie       ; Saltamos al final del bucle.
  Tr2:
    CALL Tratar2            ; Llamamos a Tratar2 para juntar las cifras del número.
  FinBucleSerie:
    MOV [BX], AX            ; Guardamos el número convertido a decimal en la posición
    INC BX                  ; de memoria correspondiente al término de la serie.
    INC BX                  ; Incrementamos BX dos veces ya que cada término son 16 bits.
    LOOP BucleSerie         ; CX = CX - 1 y si CX no es cero salto a BucleSerie
```

```
      POP DX                    ; Recuperamos el contenido que tenían los registros antes de
      POP CX                    ; llamar al procedimiento.
      POP BX
      POP AX
      RET
ConvertirADecimal ENDP
;
;
*************************************************************************************
;
; PROCEDIMIENTO:    Tratar2
; OBJETIVOS:        Junta dos cifras separadas en un único número
; Parámetros que pasa:    AL y  AH.
; Parámetros que recibe:  AL.
;
*************************************************************************************
;
Tratar2 PROC NEAR
      SHL AL, 1         ; Desplazo cuatro veces a la izquierda AL
      SHL AL, 1
      SHL AL, 1
      SHL AL, 1
      ADD AL, AH        ; AL = AL + AH se juntan las dos cifras
      XOR AH, AH        ; Ponemos a cero AH
      RET
Tratar2 ENDP
;
;
*************************************************************************************
;
; PROCEDIMIENTO:    ConvertirANumero
; OBJETIVOS:        Realiza la corrección del valor de Código ASCII al valor numérico.
; Parámetros que pasa:    AL.
; Parámetros que recibe:  AL.
;
*************************************************************************************
;
ConvertirANumero PROC NEAR
      CMP AL, EsLetra   ; Comprobamos si AL es un dígito entre la A y la F.
      JB EsNumero
      SUB AL, 07h       ; Si es letra debo restar 37h (primero resto 7 y luego 30h)
  EsNumero:
      SUB AL, 30h
      RET
ConvertirANumero ENDP
;
;
*************************************************************************************
;
; PROCEDIMIENTO:    ConvertirAACII
; OBJETIVOS:        Realiza la corrección del valor numérico al del Código ASCII.
; Parámetros que pasa:    AL.
; Parámetros que recibe:  AL.
;
*************************************************************************************
;
ConvertirAASCII PROC NEAR
      CMP AL, EsALetra  ; Comparamos AL con un dígito hexadecimal comprendido entre la A y la F.
      JB EsANumero
      ADD AL, 07h       ; Si está entre A y F se le suma 37h (primero 07 y leugo 30h)
```

```
  EsANumero:
      ADD AL, 30h          ; Si es un dígito entre 0 y 9 se le suma 30h.
      RET
ConvertirAASCII ENDP
;
;
*******************************************************************************************
;
; PROCEDIMIENTO:     EscribirSerie
; OBJETIVOS:         Realiza la impresión de los caracteres ASCII
; Parámetros que pasa:    NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
*******************************************************************************************
;
EscribirSerie PROC NEAR
      PUSH AX                  ; Salvamos los registros que vamos a utilizar.
      PUSH BX
      PUSH CX
      PUSH DX
      XOR CX, CX
      MOV CX, NumTerminos      ; Ponemos en CX cuantos términos componen nuestra serie.
      LEA BX, SerieFibonacci   ; Ponemos en BX la direción de comienzo de la serie.
   BucleEscribir:
      MOV AX, [BX]             ; Llevamos a AX los 16 bits a los que apunta BX, ya que podemos
                               ; tener los términos 144 y 233 que son menores qie 255 (FFh).
      CMP AX, 0100h            ; Comparamos AX con 0100h para ver si son el 144 o el 233.
      JB SigueEscribiendo      ; Si es menor trato las dos cifras.
      PUSH AX                  ; Guardo AX en la pila.
      MOV DL, AH               ; Llevo a DL número de la cifra superior (1**4**4 ó 2**3**3)
      ADD DL, 30h              ; Le sumo la corrección para convertirlo en Código ASCI.
                               ; Por ser número le sumo 30h.
      MOV AH, EscribirCaracter ; imprimiremos por pantalla mediante la función 02 de la INT 21h.
      INT 21h
      POP AX                   ; Recupero la cifra.
   SigueEscribiendo:
      SHR AL, 1                ; Trato los dos dígitos inferiores.
      SHR AL, 1                ; Desplaxo cuatro veces a la derecha el segundo dígito 1**4**4
      SHR AL, 1                ; (primero si el número es de dos cifras: **8**9)
      SHR AL, 1                ; rellenando con ceros por la izquierda.
      CALL ConvertirAASCII     ; Convierto el dígito en su Código ASCII.
      CMP AL, CERO             ; Comparo con el cero para evitar escribir la serie en la forma:
      JE Seguir                ; 0000 0001 0001 0002 0003 …
      MOV DL, AL               ; Llevo a DL el Código ASCII de la cifra.
      MOV AH, EscribirCaracter ; imprimiremos por pantalla mediante la función 02 de la INT 21h.
      INT 21h
   Seguir:
      MOV AX, [BX]             ; Recupero el valor de la cifra.
      AND AL, BitsInferiores   ; Me quedo con el tercer dígito (segundo si es de dos cifras 8**9**) 14**4**
      CALL ConvertirAASCII     ; Convierto el dígito en su Código ASCII.
      MOV DL, AL               ; Llevo a DL el Código ASCII de la cifra.
      MOV AH, EscribirCaracter ; imprimiremos por pantalla mediante la función 02 de la INT 21h.
      INT 21h
      INC BX                   ; Incremento BX para que apunte al siguiente término (dos bytes)
      INC BX
      CALL EscribirEspacio     ; Escribo un espacio en blanco
      LOOP BucleEscribir       ; CX = CX - 1 y si CX no es cero salto a Bucle Escribir.
      POP DX                   ; Recupero el contenido que tenían antes los registros.
      POP CX
      POP BX
```

```
        POP AX
        RET
EscribirSerie ENDP
;
;
*************************************************************************************
;
; PROCEDIMIENTO:      EscribirEspacio
; OBJETIVOS:          Produce la impresión de un espacio en blanco.
; Parámetros que pasa:    NINGUNO.
; Parámetros que recibe:  NINGUNO.
;
;
*************************************************************************************
;
EscribirEspacio PROC NEAR
        PUSH AX
        PUSH DX
        MOV DL, ESPACIO
        MOV AH, EscribirCaracter
        INT 21h
        POP DX
        POP AX
        RET
EscribirEspacio ENDP

CODIGO ENDS
END Principal
```

3) A continuación se muestra el código corregido.

```
DOSSEG
.MODEL SMALL
.STACK 1024
.DATA
PULSA   EQU 1
DATO    DB ?, 10, 13, '$'
ADIOS   DB "Senores,"
        DB 10, 13, 36
        DB "Se acabo. $"
PITOS   DB 20   ; Num. pitidos
        DB 11111111b   ; Retardo

.CODE
INIC:   MOV DX, @DATA
        MOV DS, DX
;Lee del puerto
TESTEA:   MOV DX, 332h
        IN AX, DX
        XOR AX, AX
        AND AX, PULSA
        CMP AX, 0
        JNE TESTEA
;Lee dato del puerto
        MOV DX, 200H
        IN AL, DX
        MOV DATO, AL
;Escribe dato pantalla
        ADD AL, 30
        LEA BX, DATO
```

```
        MOV AH, 09
        INT 21h
;Multiplica dato X 10
        LEA AX, DATO
        MOV BL, 10
        MUL BL
;Si dato > 100, pita
        CMP DATO, 100
        JL VISOR
        CALL PITAR
;Saca dato por puerto
VISOR:  MOV DX, 300h
        OUT DX, AL
;Se acabo
        LEA DX, ADIOS
        MOV AH, 09h
        INT 21h
        MOV AX, 04Ch
        INT 21h
PITAR PROC
        PUSH AX
        PUSHF
BUCLE:  LEA CX, PITOS
;Se pita
        MOV AX, 0E07h
        INT 10h
;Retardo (1000 veces)
        LEA BX, PITOS+1
MAS:    DEC BX
        JNZ MAS
        LOOP BUCLE
        POP AX
        POPF
PITAR ENDP
        END INIC
```

4)

```
REM     Se pone ECHO a OFF
@echo off
REM     Comprobamos que se le pase al menos un parámetro y en caso contrario se saca la ayuda.
IF "%1"=="" GOTO Error
REM     Comprobamos que se le pasen dos parámetros.
IF "%2"=="" GOTO MsgFaltan
REM     Se definen las variables:
REM             Parametro1 con el primer parámetro para buscar luego en resto.
REM             Resultado igual a PAR
REM             Etiqueta igual a EsPar
SET Parametro1=%1
SET Resultado=PAR
SET Etiqueta=EsPar
```

```
REM     Desplazamos una posición a la izquierda los parámetros.
SHIFT
REM     Entramos en el bucle del programa
:Bucle
REM     Si el parámetro es igual al que tengo salto a la etiqueta %ETIQUETA% (EsPar)
    IF %Parametro1%==%1 GOTO %Etiqueta%
:FinBucle
REM     Desplazamos una posición a la izquierda los parámetros.
REM     Si no hay más terminamos.
REM     Si quedan parámetros, saltamos al Bucle.
    SHIFT
    IF "%1"=="" GOTO Fin
    GOTO Bucle
REM     Si llegamos aquí, cambiamos el resultado por PAR y la etiqueta por EsPar
REM     Y saltamos a FinBucle
:EsImpar
     SET RESULTADO=PAR
     SET Etiqueta=EsPar
     GOTO FinBucle
REM     Si llegamos aquí, cambiamos el resultado por IMPAR y la etiqueta por EsImPar
REM     Y saltamos a FinBucle
:EsPar
     SET RESULTADO=IMPAR
     SET Etiqueta=EsImPar
     GOTO FinBucle
REM     Sacamos la sintáxis del comando.
REM     Y terminamos el programa.
:Error
    ECHO SINTAXIS: PARIMPAR P1 [P2 [P3 [P4 ...]]]
    GOTO Final
REM     Indicamos que faltan parámetros
REM     Y terminamos el programa.
:MsgFaltan
    ECHO FALTAN Parametros
:Fin
     ECHO %RESULTADO%
:Final
```

5)

```
REM     Ponemos ECHO a OFF
@ECHO OFF
REM     Comprobamos que se ha pasado al menos un parámetro y en otro
REM     caso sacamos la ayuda del comando.
IF "%1"=="" GOTO Error
REM     La orden CHKDSK  %2 /V saca el contenido de todos los ficheros y directorios
REM     por pantalla. Al redirigirla a un FIND /I “%1” solamente se mostrarán las líneas
REM     que contienen el fichero. Y a su vez al redirigirla a un fichero.bat tendremos un
REM     fichero BAT con los directorios y ficheros que se desean ejecutar
CHKDSK %2 /V | FIND /I "%1" >>TEMP.BAT
REM     Llamamos al fichero BAT creado mediante CALL, para luego seguir ejecutando el nuestro.
CALL TEMP.BAT
REM     Borramos el fichero BAT creado y evitamos mensajes mediante >NUL
DEL TEMP.BAT >NUL
REM     Saltamos al final del programa
GOTO Fin
```

```
REM     Mostramos la ayuda del comando en pantalla.
:Error
        ECHO SINTAXIS: BUS_EJEC Fichero Disco
:Fin
```

6)

```
REM     Ponemos ECHO a OFF
@ECHO OFF
REM     Comprobamos que se ha pasado al menos un parámetro y en otro
REM     caso sacamos la ayuda del comando.
IF "%1"=="" GOTO Error
REM     Guardamos en la variable Parametro1 el nombre del fichero a concatenar.
SET Parametro1=%1
REM     Comenzamos el bucle
:BUCLE
REM     Desplazamos los parámetros una posición a la izquierda.
SHIFT
REM     Si no hay más hemos terminado
IF "%1"=="" GOTO Fin
REM     Sacamos el contenido del fichero origen y lo añadimos al actual mediante la
REM     redirección >> (añadir al final)
TYPE %Parametro1%>>%1
REM     Saltamos al bucle.
GOTO BUCLE
REM     Sacamos la ayuda del comando.
:Error
        ECHO SINTAXIS PEGAR2 origen destino
:Fin
```