ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

IMPLANTACION DE UN CONTROLADOR EXPERTO

MANUAL DE USO DEL PROGRAMA

TESIS PREVIA A LA OBTENCION DEL TITULO DE

INGENIERO ELECTRONICO EN LA ESPECIALIZACION DE

ELECTRONICA Y CONTROL

ANA ELENA ESTEVEZ BARRERA

OCTUBRE DE 1992

INDICE

PAGINA

MANUAL DE USO DEL PROGRAMA

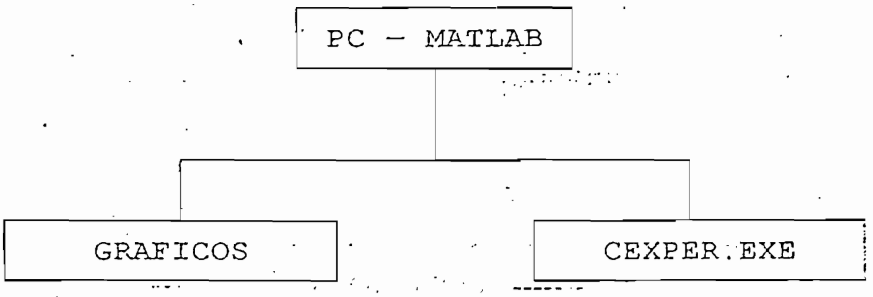
1) REQUISITOS DE SOFTWARE Y HARDWARE.

El programa desarrollado requiere el siguiente software y hardware:

- IBM PC, PC/XT, PC/AT o computador compatible.
- Mínimo 320K de memoria.
- Coprocesador matemático 8087, 80287 ó 80387.
- Una unidad de disco duro y una de disco flexible.
- Sistema operativo MS-DOS.
- Programa PC-MATLAB.
- Programa CEXPER.EXE.

2) INSTALACION DEL PROGRAMA.

Para instalar el programa, se recomienda utilizar la siguiente estructura que permite ejecutar el programa CEXPER, generar datos y observar los resultados obtenidos a través de gráficos.



El programa del controlador experto desarrollado se denomina CEXPER.EXE, el que se incorpora al directorio raíz del programa PC-MATLAB. Además se debe crear el directorio GRAFICOS, el cual contiene la lista de archivos que ejecutan las rutinas de gráficos. Estos archivos son: GRAF.M, GRAFLIST.M, TESIS.M, GABIERTO.M, GCERRADO.M, GPID.M, GDEADBEA.M, GSELF.M, GEXPERT1.M, GEXPERT2.M, GTODOS1.M, GTODOS2.M, GERRORS1.M, GERRORS2.M, GCONTRO1.M, y GCONTRO2.M.

Finalmente es necesario actualizar el archivo MATLAB.BAT, al que se debe añadir la localización del directorio GRAFICOS y unidad A en la sentencia PATH.

3) INGRESO AL PROGRAMA.

Considerando la organización del software mencionado anteriormente, el procedimiento de llamada al programa CEXPER es el siguiente:

- Ingresar al programa PC-MATLAB.

Digite GRAF y presione ENTER para desplegar el menú de gráficos, que se indica a continuación:

```
**********************************************************************
*          ------- MENU DE GRAFICOS -------                           *
*                                                                    *
*    1)    Ejecutar tesis: CEXPER.EXE                                *
*    2)    Respuesta en lazo abierto.                                *
*    3)    Respuesta en lazo cerrado.                                *
*    4)    Respuesta con el control PID.                             *
*    5)    Respuesta con el control DEADBEAT.                        *
*    6)    Respuesta con el control SELF_TUNING.                     *
*    7)    Respuesta con el control EXPERTO 1.                       *
*    8)    Respuesta con el control EXPERTO 2.                       *
*    9)    Salidas del sistema y control EXPERTO 1.                  *
*   10)    Salidas del sistema y control EXPERTO 2.                  *
*   11)    Errores del sistema y control EXPERTO 1.                  *
*   12)    Errores del sistema y control EXPERTO 2.                  *
*   13)    Señal de control EXPERTO 1.                               *
*   14)    Señal de control EXPERTO 2.                               *
*                                                                    *
*    0)    SALIR.                                                    *
**********************************************************************
```

La primera opción permite llamar al programa CEXPER y generar resultados que serán almacenados en la unidad A del computador.

Una vez calculados los datos con la primera opción, es posible observar el comportamiento del sistema en estudio. Seleccionar las opciones del 2 al 14, para observar el tipo de gráfico deseado.

Las opciones 2 y 3 permiten observar la respuesta del proceso en lazo abierto y cerrado respectivamente. Para mostrar la respuesta de la planta con los controles PID,

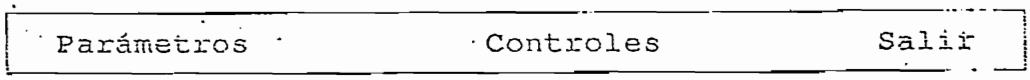
DEADBEAT, SELF TUNING, EXPERTO 1 y EXPERTO 2, seleccione las opciones 4, 5, 6, 7 y 8, según sea el caso. Un gráfico comparativo de todos los tipos de respuestas obtenidas del sistema al aplicar cada algoritmo de control y controlador experto, se obtiene al seleccionar las opciones 9 y 10. Las señales de error del sistema son graficadas al considerar las opciones 11 y 12. Finalmente, la acción de control EXPERTO 1 y EXPERTO 2 es mostrada en pantalla al digitar las últimas opciones del menú de gráficos 13 y 14 respectivamente. Si desea finalizar la sesión de trabajo, seleccionar la opción 0.

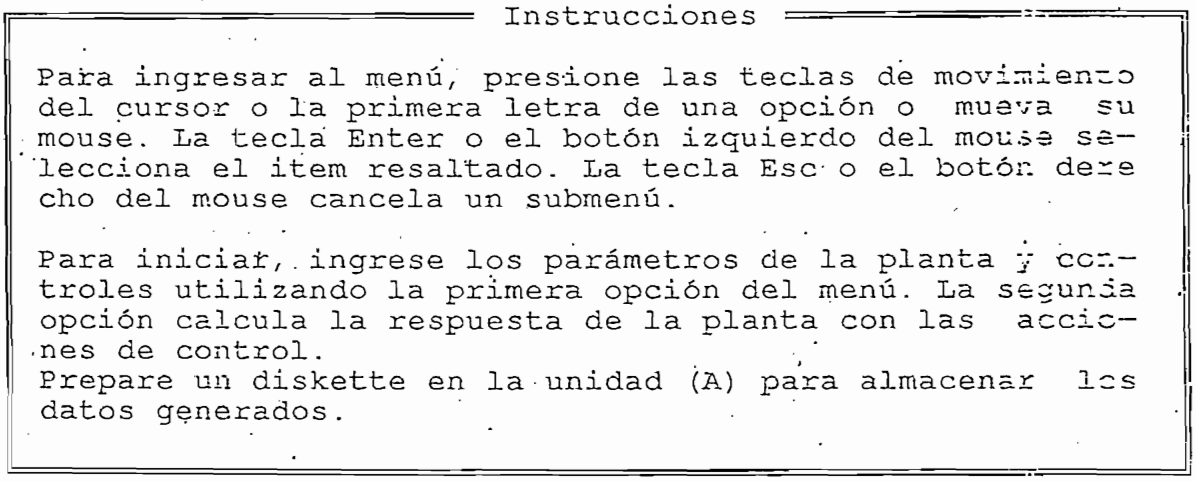
Cada opción de gráficos permite realizar una impresión del mismo, para ello una vez que termina de generar el gráfico presionar cualquier tecla para continuar. El programa despliega un mensaje en pantalla como el que se indica a continuación:

DESEA IMPRIMIR EL GRAFICO? (1) SI (2) NO

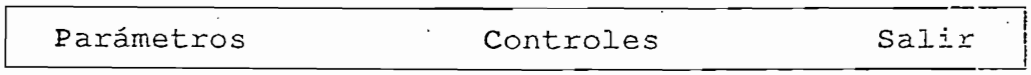
Digitar (1) si su respuesta es afirmativa y prepare la impresora, o (2) para regresar al menú de gráficos.

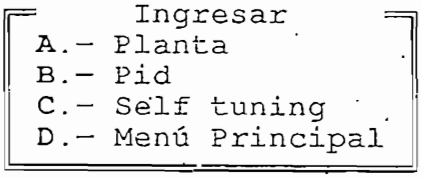
Como se indicó anteriormente, es necesario primero ejecutar el programa del conrolador experto, para proceder a observar los resultados obtenidos. Por lo tanto escoger la primera opción del menú de gráficos, de esta forma se carga en memoria el programa CEXPER, el que despliega en pantalla el menú principal como el que se muestra a continuación:





Como se menciona en el mensaje de instrucciones, para iniciar la simulación es necesario escoger la primera opción del menú Parámetros, mostrándose en pantalla el siguiente submenú:





Seleccionar la opción A para ingresar los parámetros de la planta. El programa CEXPER envía un mensaje de ayuda, en el cual se indica los datos necesarios para la simulación. Este mensaje es:

Ingreso de los parámetros de la planta

FUNCION DE TRANSFERENCIA:

$$G(S) = \frac{b(m)*S^m + b(m-1)*S^{(m-1)} + .. + b(0)}{a(n)*S^n + a(n-1)*S^{(n-1)} + .. + a(0)} * exp(-Tr*S)$$

d= entero (Tr / Tmuestreo)

$n \geq 1$; $m < n$; $2n + d \leq 11$

Ingrese los siguientes datos:

m =
b[0] =
n =
a[0] =
Tr =
Límite actuador =
Set point =
Variar parámetros S/N ? =

ESTAN CORRECTOS LOS DATOS S/N ?

Contestar adecuadamente a cada mensaje, donde m, n, b[ ], a[ ] y Tr pertenecen a la función de transferencia G(S), que describe la dinámica del proceso en estudio. Para ingresar Tr se puede considerar su valor absoluto.

Es importante considerar la restricción : $2n + d \leq 11$, donde d es el resultado de obtener la parte entera del cociente que resulta al dividir el tiempo de retardo del sistema (si existe) y el período de muestreo.

Para el valor del set point digitar 1, lo que representa una entrada paso unitaria. Si se desea analizar el comportamiento del sistema al variar el coeficiente a(n-1) de la función de transferencia contestar al mensaje Variar parámetros S/N ? afirmativamente. Si ese es el caso se pide ingresar el valor del factor fac, por el que se multiplicará el

valor original de a(n-1) para simular un cambio de parámetros.

En caso de error, contestar al mensaje ESTAN CORRECTOS LOS DATOS S/N ? negativamente y volver a ingresar toda la información de la planta. Caso contrario presionar ENTER o contestar afirmativamente.

A continuación se muestra en pantalla las raíces del polinomio característico, a través del cual se determina las constantes de tiempo del sistema que permiten calcular el período de muestreo apropiado para la simulación del proceso.

```
CALCULO DEL PERIODO DE MUESTREO
-------------------------------

RAICES DEL POLINOMIO CARACTERISTICO

No.      Parte Real      Parte Imaginaria

La menor constante de tiempo del sistema es :
Se sugiere un Tmuestreo menor o igual a :
Ingrese : Tmuestreo =

ESTAN CORRECTOS LOS DATOS S/N ?
```

En este caso el programa espera que se ingrese un valor diferente a cero, caso contrario seguirá mostrando el mismo mensaje. Una vez que se digita el valor del Tmuestreo correctamente, contestar afirmativamente al mensaje ESTAN CORRECTOS LOS DATOS S/N ? para retornar al menú principal.

Posteriormente es necesario especificar los parámetros que definen los controles PID y SELF TUNING, para lo cual se vuelve a escoger la primera opción del menú principal y la opción B para el caso del control PID o la opción C para el control SELF TUNING.

Al considerar la opción B, se muestra en pantalla el mensaje correspondiente al ingreso de parámetros del control PID no interactuante como se indica a continuación:

```
Ingreso de parámetros del controlador PID no interactuante

                                    1                      Td
   U(S) = Kp * [ ( 1 + ---) * E(S) + -------- * Y(S) ]
                       STi                 1 + αSTd

 Kp= Ganancia   Ti= Tiempo integral  Td= tiempo derivativo
```

Ingrese los siguientes datos:

Kp = Ti = Td =

ESTAN CORRECTOS LOS DATOS S/N ?

Para el cálculo de los parámetros de Kp, Ti y Td, se recomienda utilizar las técnicas de ajuste de Ziegler Nichols, de margen de fase de Åstrom o respuesta de frecuencia.

Ingresados correctamente los parámetros del controlador se contesta afirmativamente al último mensaje, el que confirma si

los datos ingresados son correctos o no. Se presiona ENTER o responde afirmativamente para retornar al menú principal.

Para realizar el ingreso de datos del control SELF TUNING, se debe escoger la opción C del submenú Parámetros. En este momento se presenta en pantalla el siguiente mensaje:

```
═══ Ingreso de polos deseados para el control SELF-TUNING ═══

                                        -1          -1
1.- Respuesta de primer orden: T(Z  ) = 1 + Z   * t1
    t1 = - exp(-Tmuestreo/τ)

                                     -1        -1        -2
2.- Respuesta de segundo orden:T(Z  )=1 + Z  *t1 + Z  *t2

                                                      ½
   t1= -2*exp(-ε*Wn*Tmuestreo) * cos(Wn*(1-ε²) *Tmuestreo
   t2= exp(-2*ε*Wn*Tmuestreo)

τ = Constante de tiempo deseada      Tmuestreo= Período de
                                                muestreo

ε = Coeficiente de amortiguamiento          Wn= Frecuencia
                                                natural
```

Seleccione el tipo de respuesta deseada :

(1) Primer orden. (2) Segundo orden.

ESTAN CORRECTOS LOS DATOS S/N ?

Si el proceso por analizar está descrito por una función de transferencia de primer orden, se recomienda seleccionar la opción (1). Para sistemas de segundo u orden superior escoger la opción (2).

Al seleccionar la primera opción es necesario ingresar el valor de la constante de tiempo con la que se espera responda el sistema compensado. Se presenta en pantalla el siguiente mensaje:

```
Se sugiere escoger   τ   ≤
Ingrese   τ =
```

Donde el programa está diseñado para sugerir una constante de tiempo de acuerdo al tiempo de observación del proceso. El cual se obtiene de multiplicar el período de muestreo por el número de iteraciones (200). De todas formas el usuario puede ingresar el valor que a su criterio es más conveniente. Utilizar la siguiente fórmula:

$$t_s = 4 * \tau$$

Donde se asigna el valor esperado de ts y se calcula τ.

Si se selecciona el tipo de respuesta de segundo orden, se debe ingresar los valores correspondientes al coeficiente de amortiguamiento ε y frecuencia natural Wn. De igual forma el programa sugiere los valores que se debe asignar a estos parámetros, mas el usuario puede calcularlos a partir de las siguiente expresiones:

$$M_p = 100 * e^{\frac{-\varepsilon\pi}{\sqrt{1-\varepsilon^2}}}$$

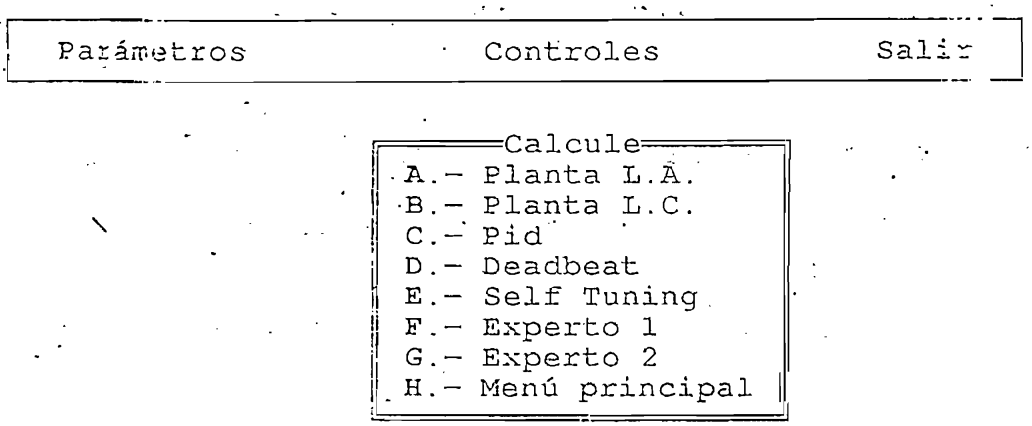
$$t_s = \frac{4}{\varepsilon Wn}$$

Si el proceso que se está analizando es un sistema de primer orden sin retardo de transporte, el controlador SELF TUNING se limita a calcular unicamente una respuesta tipo primer orden. En pantalla se muestra el siguiente mensaje de advertencia:

```
Seleccione el tipo de respuesta deseada :

(1) Primer orden.          (2) Segundo orden.

SOLO PUEDE ESCOGER UNA RESPUESTA DE PRIMER ORDEN:

                ESTAN CORRECTOS LOS DATOS S/N ?
```

Una vez que se han ingresado los valores correspondientes y si no están incorrectos se procede de la misma forma que en los casos anteriores para regresar al menú principal.

En este momento el sistema está listo para proceder con la simulación del proceso y el cálculo de las respuestas del sistema a las respectivas acciones de control, para el efecto seleccione la segunda opción del menú principal Controles, a partir de la cual se despliega en pantalla el siguiente submenú:



Donde las opciones A y B, sirven para calcular la respuesta del sistema en lazo abierto y cerrado respectivamente, sin utilizar ningún controlador. Estas opciones permiten simular el comportamiento de los procesos cuando trabajan individualmente.

Las opciones C, D,y E realizan el cálculo de la salida del sistema cuando se aplican al mismo los controles PID, DEADBEAT y SELF TUNING. La señal de salida de los controladores EXPERTO 1 y EXPERTO 2 son obtenidas al seleccionar las opciones F y G.

En todos los casos, los cálculos realizados son almacenados en un diskette de la unidad A. En éste se generan archivos que recogen la información del sistema que se ha simulado como son: tiempo, señal de error, salida del controlador, respuesta del sistema compensado y la señal de referencia o set point. Estos archivos son recuperados a la memoria del computador, al seleccionar cualquier opción (2 - 11) del menú de gráficos descrito anteriormente.

Se finaliza los cálculos en el CEXPER, siempre que se haya ejecutado todas las opciones del submenú Controles, sólo en este momento se puede proceder a salir del programa, para lo cual se escoge la última opción del menú principal Salir.

En pantalla se muestra el submenú de salida, digitar S para retornar al menú de gráficos o N para permanecer en el programa.

LISTADO DEL PROGRAMA

```
/* ------------------------------------------------------------------------
```

Nombre: CEXPER.C
Tipo: Programa Principal
Lenguaje: Microsoft QuickC version 2

Lista del programa:

CEXPER.C
MENU.C
BOX.C
MOUSEFUN.C
GETKEY.C
RAICES.C
SIMULAR.C
CONTROL.C
IDENTIFI.C
DISCRETO.C
BASE.C
GRAPHICS.LIB

Variables:

info_pres	Mensaje de presentación
info_box	Mensaje de ayuda
bar_main	Menú principal
drop_datos	Menú para ingreso de parámetros
drop_control	Menú para cálculo de controles
drop_grafico	Menú para graficar respuestas
drop_quit	Menú para salida
info_func_trans	Mensaje de G(S)
info_func_disc	Mensaje de G(Z)
info_pid	Mensaje del PID
info_calculo	Mensaje de espera para el cálculo
info_archivo	Mensaje de advertencia para preparar la unidad A.
n	Primer nivel de selección del submenu
n1	Segundo nivel de selección del submenu
bar_choice	Selección del menu principal
first_time	Bandera del menú principal
quit_flag	Bandera de salida de la función
save_info_box	Buffer para grabar la pantalla
save_bar_main	Buffer para grabar la pantalla

save_drop_colors	Buffer para grabar la pantalla
save_drop_menu	Buffer para grabar la pantalla
GRAD_NUM	Grado del numerador de G(S)
GRAD_DEN	Grado del denominador de G(S)
NUMER	Coeficientes del numerador de G(S)
DENOM	Coeficientes del denominador de G(S)
T_MUES	Tiempo de muestreo
T_RETAR	Tiempo de retardo
LIMITE	Límite del actuador
SET_POINT	Punto de trabajo
DENOM_D	Coeficientes del numerador de G(Z)
NUMER_D	Coeficientes del denominador de G(Z)
POL_CAR[]	Polinomio característico de la planta.
fac	Factor de variación de a(n-1)
KP	Ganancia Kp del control PID
TI	Tiempo integral del control PID
TD	Tiempo derivativo del control PID
tpolos[]	Asignación de polos del sistema
Tao	Constante de tiempo, planta 1er orden
EP	Coeficiente de amortiguamiento, 2do orden
Wn	Frecuencia natural, planta 2do orden
H	Ganancia de compensación para el control SELF-TUNING
PUNTOS	Máximo número de cálculos
REF[]	Referencia del sistema
ERROR[]	Error del sistema
UL	Respuesta del limitador
U_PID[]	Respuesta del controlador PID
U_DEAD[]	Respuesta del controlador DEADBEAT
U_SELF[]	Respuesta del controlador SELF-TUNING
U_EXPERT[]	Respuesta del controlador EXPERTO 1 y 2
Yout[]	Respuesta de la planta analizada

---

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include "menu.h"
#include "box.h"
#include "t_colors.h"
#include "funcion.h"

double T_MUES;
```

```
main()

{
    extern double T_MUES;
    FILE *archivo;
    int n, n1;
    long int esp;
    int presenta = 1;
    int bar_choice = 1;
    int first_time = 1;
    int quit_flag = 0;
    int far *save_info_box;
    int far *save_bar_main;
    int far *save_drop_colors;
    int far *save_drop_menu;
    int far *save_drop_menu1;

    int ic, jc, col, fil;
    double fac;
    int ban1 = 1;
    double sum, Tao, EP, Wn;
    char DATO[30], DATO2[2], VARIAR[2];
    int GRAD_NUM, GRAD_DEN, T_RETAR, CAM;
    double DENOM[MAX+1], NUMER[MAX+1], DENOM_D[MAX+1], NUMER_D[MAX+1];
    double REAL[MAX+1], IMAG[MAX+1], POL_CAR[MAX+1];
    double TMIN, SET_POINT, LIMITE;
    double tpolos[MAX+1];
    float  T_RETAR1;
    double KP, TI, TD;

    double Yout[PUNTOS+1];
    double ERROR[PUNTOS+1];
    double ERRORs[PUNTOS+1];
    double U_PID[PUNTOS+1];
    double U_DEAD[PUNTOS+1];
    double U_SELF[PUNTOS+1];
    double U_EXPERT[PUNTOS+1];
    double REF[PUNTOS+1];
    double U, UL;
    double DERIV_E;
    double II = 1;

char *info_pres[] =
    {
    "",
    " ",
    "            ESCUELA POLITECNICA NACIONAL                ",
    "          FACULTAD DE INGENIERIA ELECTRICA              ",
```

```
    " ",
    "        DEPARTAMENTO DE ELECTRONICA Y CONTROL        ",
    " ",
    " ",
    "                    TESIS                    ",
    " ",
    "        IMPLANTACION DE UN CONTROLADOR EXPERTO        ",
    " ",
    " ",
    "        Realizado por:    ELENA ESTEVEZ B.        ",
    "        Dirigido por:    ING. RAFAEL FIERRO        ",
    " ",
    " ",
    "              Octubre - 1992              ",
    " ",
    " Presione cualquier tecla para continuar",
    NULL
    };

char *info_box[] =
    {
    " Instrucciones ",
    " ",
    "Para ingresar al menú, presione las teclas de movimiento del",
    "cursor o la primera letra de una opción o mueva su mouse. La ",
    "tecla Enter o el botón izquierdo del mouse selecciona el item",
    "resaltado. La tecla Esc o el botón derecho del mouse cancela ",
    "un submenú.                                    ",
    " ",
    "Para iniciar, ingrese los parametros de la planta y controles",
    "utilizando la primera opción del menú. La segunda opción cal-",
    "cula la respuesta de la planta con las acciones de control. ",
    "Prepare un diskette en la unidad (A) para almacenar los datos",
    "generados.",
    " ",
    "",
    NULL
    };

char *bar_main = " Parámetros  Controles  Salir ";

char *drop_datos[] =
    {
    " Ingresar ",
    "A.- Planta",
    "B.- Pid",
    "C.- Self tuning",
    "D.- Menú principal",
```

```
    "",
    NULL
    };

char *drop_control[] =
    {
    " Calcule ",
    "A.- Planta L.A.",
    "B.- Planta L.C.",
    "C.- Pid",
    "D.- Deadbeat",
    "E.- Self tuning",
    "F.- Experto 1",
    "G.- Experto 2",
    "H.- Menú principal",
    "",
    NULL
    };

char *drop_quit[] =
    {
    "",
    "No",
    "Si",
    "",
    NULL
    };

char *info_func_trans[] =
    {
    " Ingreso de los parámetros de la planta ",
    " ",
    "   FUNCION DE TRANSFERENCIA :",
    " ",
    " ",
    "              b(m)*S^m + b(m-1)*S^(m-1) + .. + b(0)",
    "   G(S) = ---------------------------------------------- * exp(-Tr * S)   ",
    "              a(n)*S^n.+ a(n-1)*S^(n-1) + .. + a(0)",
    " ",
    "   d = entero (Tr / Tmuestreo)",
    " ",
    "              n ≥ 1   :   m < n   ;   2n + d ≤ 11 ",
    "",
    NULL
    };

char *info_pid[] =
    {
```

```
    " Ingreso de parámetros del controlador PID no interactuante ",
    " ",
    "                          1                  Td                  ",
    "    U(S) = Kp * [ (1 + ---- ) * E(S) + ------------ * Y(s) ]",
    "                         STi             1 + αSTd          ",
    " ",
    " Kp = Ganancia    Ti = Tiempo integral   Td = Tiempo derivativo",
    "",
    NULL
    };

char *info_self[] =
    {
    " Ingreso de polos deseados para el control SELF-TUNING ",
    "                                        -1        -1",
    " 1.- Respuesta de primer orden: T(Z  ) = 1 + Z  * t1 ",
    "     t1 = -exp(-Tmuestreo/τ) ",
    "                                       -1        -1        -2 ",
    " 2.- Respuesta de segundo orden: T(Z  ) = 1 + Z  * t1 + Z   * t2 ",
    "                                                    2  ½  ",
    "     t1 = -2*exp(-ε*Wn*Tmuestreo) * cos (Wn*(1 - ε  )  * Tmuestreo)",
    "     t2 = exp(-2*ε*Wn*Tmuestreo)",
    " ",
    " τ = Constante de tiempo deseada       Tmuestreo = Período de muestreo",
    " ε = Coeficiente de amortiguamiento          Wn = Frecuencia natural ",
    "",
    NULL
    };

char *info_calculo1[] =
    {
    "",
    " ",
    " CALCULANDO LA RESPUESTA DE LA PLANTA EN LAZO ABIERTO ",
    " ",
    " ",
    "                    No.    ",
    " ",
    "",
    NULL
    };

char *info_calculo2[] =
    {
    "",
    " ",
    " CALCULANDO LA RESPUESTA DE LA PLANTA EN LAZO CERRADO ",
    " "
```

```
    " ",
    "                    No.     ",
    " ",
    "",
    NULL
    };

char *info_calculo3[] =
    {
    "",
    " ",
    " CALCULANDO LA RESPUESTA DE LA PLANTA CON EL CONTROL PID ";
    " ",
    " ",
    "                    No.     ",
    " ",
    "",
    NULL
    };

char *info_calculo4[] =
    {
    "",
    " ",
    " CALCULANDO LA RESPUESTA DE LA PLANTA CON EL CONTROL DEADBEAT
";
    " ",
    " ",
    "                    No.     ",
    " ",
    "",
    NULL
    };

char *info_calculo5[] =
    {
    "",
    " ",
    " CALCULANDO LA RESPUESTA DE LA PLANTA CON EL CONTROL SELF-TUNING
";
    " ",
    " ",
    "                    No.     ",
    " ",
    "",
    NULL
    };
```

```
char *info_calculo6[] =
    {
    "",
    " ",
    " CALCULANDO LA RESPUESTA DE LA PLANTA CON EL CONTROL EXPERTO 1
",
    " ",
    " ",
    "                    No.    ",
    " ",
    "",
    NULL
    };

char *info_calculo7[] =
    {
    "",
    " ",
    " CALCULANDO LA RESPUESTA DE LA PLANTA CON EL CONTROL EXPERTO 2
",
    " ",
    " ",
    "                    No.    ",
    " ",
    "",
    NULL
    };

char *info_archivo[] =
    {
    "",
    " ",
    "INSERTE UN DISKETTE EN LA UNIDAD A PARA ALMACENAR LOS DATOS
GENERADOS",
    " ",
    " ",
    " Presione cualquier tecla para continuar ",
    NULL
    };

    /* Inicializa video */
    _setvideomode( _TEXTC80 );
    _clearscreen( _GCLEARSCREEN );

    /* Llena el background */
    box_charfill( 1, 1, 25, 80, [redacted]);

    /* Lazo principal inicia aquí */
```

```
while (!quit_flag )
   {
    /* Borrar la información de mensajes y menú principal si no es
       la primera vez. */

   if ( !first_time )
      {
      menu_erase( save_info_box );
      menu_erase( save_bar_main );
       }
    else
       first_time = 0;

    /* Presentación de la tesis */
    if ( presenta )
    {
       save_info_box = menu_message( 3, 8, info_pres );
       getch();
       presenta = 0;
    }
    menu_erase( save_info_box );

    /* Muestra la información de ayuda */
    save_info_box = menu_message( 9, 8, info_box );

    /* Barra de menú principal */
    save_bar_main = menu_bar( 3, 25, bar_main, &bar_choice );
    switch( bar_choice )
    {
      /* ..........................................................
         Ingreso de parámetros de la planta y controles
       ..........................................................
      */
      case 1:
       save_drop_menu = menu_drop( 5, 22,drop_datos, &n );
       switch( n )
        {
        case 1:

          /* Ingreso de la Función de Transferencia de la planta
             ..................................................
          */

          do
          {
           _clearscreen( _GCLEARSCREEN );
           menu_message(1, 8, info_func_trans);
           _settextposition(16,8);
```

```
printf("Ingrese los siguientes datos : ");

_settextposition(18,8);
printf("m = ");
fgets( DATO, sizeof(DATO), stdin);
GRAD_NUM = atoi(DATO);

_settextposition(18,30);
printf("n = ");
fgets( DATO, sizeof(DATO), stdin);
GRAD_DEN = atoi(DATO);

/* Inicializar arreglos */
for(ic=0; ic <= MAX; ic++)
  {
    NUMER[ic] = 0;
    DENOM[ic] = 0;
    tpolos[ic] = 0;
    T_MUES = 0;
  }

fil = 19;
col = 8;
for(ic = 1; ic <= GRAD_NUM + 1; ic++)
  {
  _settextposition(fil++, col);
  printf("b[%d] = ",GRAD_NUM-ic+1);
  fgets( DATO, sizeof(DATO), stdin);
  NUMER[ic] = atof(DATO);
  }

fil = 19;
col = 30;
for(ic = 1; ic <= GRAD_DEN + 1; ic++)
  {
  _settextposition(fil++, col);
  printf("a[%d] = ",GRAD_DEN-ic+1);
  fgets( DATO, sizeof(DATO), stdin);
  DENOM[ic] = atof(DATO);
  }

_settextposition(18,50);
printf("Tr = ");
fgets( DATO, sizeof(DATO), stdin);
T_RETAR1 = fabs(atof(DATO));

_settextposition(19,50);
printf("Límite actuador = ");
```

```
fgets( DATO, sizeof(DATO), stdin);
LIMITE = atof(DATO);

_settextposition(20,50);
printf("Set point = ");
fgets( DATO, sizeof(DATO), stdin);
SET_POINT = atof(DATO);

_settextposition(21,50);
printf("Variar parámetros S/N ? = ");
gets(VARIAR);

/* Setear bandera para cambio de parámetros */
if( !strcmp(VARIAR,"S") || !strcmp(VARIAR,"s"))
   CAM = 1;
else
   CAM = 0;

if( CAM == 1)
  {
   _settextposition(22,50);
   printf("Factor de variación = ");
   fgets( DATO, sizeof(DATO), stdin);
   fac = atof(DATO);
  }

_settextposition(24,23);
printf("ESTAN CORRECTOS LOS DATOS S/N ? ");
gets(DATO);
}

while( !strcmp(DATO,"n") || !strcmp(DATO,"N"));

/* Cálculo del Período de muestreo */

do
{
_clearscreen( _GCLEARSCREEN );
_settextposition(3,22);
printf("CALCULO DEL PERIODO DE MUESTREO");
_settextposition(4,22);
printf("-----------------------------");

/* Obtención del Polinomio Característico */
ic = GRAD_DEN - GRAD_NUM;
for(ic=GRAD_DEN+1; ic > 0; ic--)
  {
    if ((ic - jc) < 0)
```

```
        NUMER[ic - jc] = 0;

      POL_CAR[ic] = DENOM[ic] + NUMER[ic - jc];
    }

  /* Cálculo de raíces del Polinomio Característico */
  raizpoli2(GRAD_DEN, POL_CAR, REAL, IMAG);

  /* Salida en pantalla de los valores obtenidos */
  _settextposition(7,20);
  printf("RAICES DEL POLINOMIO CARACTERISTICO");
  _settextposition(9,15);
  printf("No.        Parte Real          Parte Imaginaria");
  for(ic=1; ic <= GRAD_DEN; ic++)
    {
    _settextposition(10+ic,15);
    printf("%2d %19.9f %22.9f", ic, REAL[ic], IMAG[ic]);
    }

  TMIN = tmin(GRAD_DEN, REAL, IMAG);

  _settextposition(13 + GRAD_DEN,15);
  printf("La menor constante de tiempo del sistema es : %3.9f ",TMIN);

  _settextposition(15 + GRAD_DEN,15);
  printf("Se sugiere un Tmuestreo menor o igual a: %3.9f",TMIN/10);

  _settextposition(17 + GRAD_DEN,15);
  printf("Ingrese :   Tmuestreo = ");
  fgets(DATO, sizeof(DATO), stdin);
  T_MUES = atof(DATO);

  if ( T_MUES == 0 )
    {
     _settextposition(20 + GRAD_DEN,23);
     printf("NO INGRESO EL VALOR DE Tmuestreo !  ");
    }

  _settextposition(21,25);
  printf("ESTA CORRECTO EL DATO S/N ? ");
  gets(DATO);
}

while( !strcmp(DATO,"n") || !strcmp(DATO,"N") || (T_MUES ==0));

/* Ajuste del Tiempo de retardo para la simulación */
T_RETAR1 = T_RETAR1/T_MUES;
T_RETAR = T_RETAR1;
```

```
/* Cálculo de la Función de Transferencia discreta */
DISC( GRAD_NUM, GRAD_DEN, NUMER, DENOM, NUMER_D,
      DENOM_D);

break;

case 2:

/* Ingreso de parámetros para el control pid
   ..........
*/

do
{
_clearscreen( _GCLEARSCREEN );
menu_message(3, 8, info_pid);

_settextposition(15,8);
printf("Ingrese los siguientes datos : ");

_settextposition(17, 8);
printf("Kp = ");
fgets( DATO, sizeof(DATO), stdin);
KP = atof(DATO);

_settextposition(17, 25);
printf("Ti = ");
fgets( DATO, sizeof(DATO), stdin);
TI = atof(DATO);

_settextposition(17, 42);
printf("Td = ");
fgets( DATO, sizeof(DATO), stdin);
TD = atof(DATO);

_settextposition(21,23);
printf("ESTAN CORRECTOS LOS DATOS S/N ? ");
gets(DATO);
}

while( !stremp(DATO,"n") || !stremp(DATO,"N"));
break;

case 3:

/* Ingreso de polos deseados para el control self-tuning
   ..........
*/
```

```
do
{
 _clearscreen( _GCLEARSCREEN );
 menu_message(1, 5, info_self);

 _settextposition(18,8);
 printf("Seleccione el tipo de respuesta deseada : ");

 _settextposition(20,8);
 printf("(1) Primer orden.   (2) Segundo orden.      ");

 if((GRAD_DEN == 1) && (T_RETAR == 0))
 {
 _settextposition(22, 8);
 printf("SOLO PUEDE ESCOGER UNA-RESPUESTA DE PRIMER ORDEN: ");
 strcpy(DATO2,"1");
 }
 gets(DATO2);

 _settextposition(24,15);
 printf("SELECCIONO CORRECTAMENTE EL TIPO DE RESPUESTA S/N ?");
 gets(DATO);
}
while( !strcmp(DATO,"n") || !strcmp(DATO,"N"));

/* Ingreso de datos para una respuesta: tipo primer orden */
if( !strcmp(DATO2,"1"))
 {
   do
   {
    _clearscreen( _GCLEARSCREEN );
    menu_message(1, 5, info_self);

    /* Cálculo de la constante de tiempo recomendada */
    Tao = T_MUES*200/10/4;

    _settextposition(18,8);
    printf("Se sugiere escoger τ <= %3.6f ",Tao);

    _settextposition(20,8);
    printf("Ingrese τ = ");
    fgets( DATO, sizeof(DATO), stdin);
    Tao = atof(DATO);

    /* Cálculo de t1 */
    tpolos[1] = -exp(-T_MUES/Tao);

    _settextposition(24,23);
```

```
  printf("ESTAN CORRECTOS LOS DATOS S/N ? ");
  gets(DATO);
 }
 while( !stremp(DATO,"n") || !stremp(DATO,"N"));
}

/* Ingreso de datos para una respuesta tipo segundo orden */
if( !stremp(DATO2,"2"))
{
 do
 {
  _clearscreen( _GCLEARSCREEN );
  menu_message(1, 5, info_self);

  /* Valor del coeficiente de amortiguamiento recomendado
     para obtener una respuesta con un sobreimpulso aceptable
  */
  _settextposition(18,8);
  printf("Se sugiere escoger  ε = 0.7 a 0.8 ");

  _settextposition(18,50);
  printf("Ingrese ε  =  ");
  fgets( DATO, sizeof(DATO), stdin);
  EP = atof(DATO);

  /* Cálculo de la frecuencia natural recomendada para obtener
     una respuesta con tiempo de establecimiento conveniente
  */
  Wn = 6*4/(200*T_MUES*EP);

  _settextposition(20,8);
  printf("Se sugiere escoger Wn  >  %3.6f",Wn);

  _settextposition(20,50);
  printf("Ingrese Wn  =  ");
  fgets( DATO, sizeof(DATO), stdin);
  Wn = atof(DATO);

  /* Cálculo de t1 y t2 del polinomio T(Z) */
  tpolos[1]=-2*exp(-EP*Wn*T_MUES)*cos(Wn*sqrt(1-EP*EP)*T_MUES);
  tpolos[2] = exp(-2*EP*Wn*T_MUES);

  _settextposition(24,23);
  printf("ESTAN CORRECTOS LOS DATOS S/N ? ");
  gets(DATO);
 }
 while( !stremp(DATO,"n") || !stremp(DATO,"N"));
```

```
    break;

  case 4:

    /* Regresar al menu principal
       ..........
    */

    menu_erase(save_drop_menu );
    break;
  }
  _clearscreen( _GCLEARSCREEN );
  box_charfill( 1, 1, 25, 80, '▒');
  menu_message( 9, 8, info_box );
  break;

/*.....
  Cálculo de las señales de control
  .....
*/

case 2:

 save_drop_menu = menu_drop( 5, 33, drop_control, &n );
 switch( n )
  {
  case 1:

    /* Cálculo de la respuesta del sistema en lazo abierto
       ..........
    */

    _clearscreen( _GCLEARSCREEN );
    menu_message(8, 12, info_calculo1);

    /* Inicialización de la respuesta en lazo abierto */
    for(ic = 0; ic <= PUNTOS; ic++)
      Yout[ic] = 0;

    for(ic = 1; ic <= PUNTOS; ic++)
       {
        U = SET_POINT;

        if (CAM == 1)
         {
          if ((ic >= 80) && (ic < 120))
            U = 0;
         }
```

```
        REF[ic] = U;

        Yout[ic] = RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER,
                         T_RETAR, REF[ic], 0, ic, CAM, fac, &ban1);

        _settextposition(13,42);
        printf("%d ",ic);
       }

    /* Reset de las Condiciones Iniciales y punteros para una
       nueva simulación de la planta */
    RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER, T_RETAR, U, 1, ic,
          CAM, fac, &ban1);

    /* Almacenar los datos calculados en un archivo ABIERTO.DAT */
    _clearscreen( _GCLEARSCREEN );
    menu_message(8, 4, info_archivo);
    getch();

    if (NULL == (archivo=fopen("A:ABIERTO.DAT","w")))
       {
        _clearscreen(_GCLEARSCREEN);
        printf("ERROR AL ABRIR EL ARCHIVO !\n");
        printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
        getch();
        exit(0);
       }

    for(ic=1; ic <= PUNTOS; ic++)
       fprintf(archivo,"%3.4f  %3.4f  %3.4f\n", ic*T_MUES,
               Yout[ic], REF[ic]);

    if (fclose(archivo) != NULL)
       {
        printf("ERROR AL CERRAR EL ARCHIVO.\n");
        printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
        getch();
        exit(0);
       }
    break;

case 2:

 /* Cálculo de la respuesta del sistema en lazo cerrado
 ..........
 */

 _clearscreen( _GCLEARSCREEN );
```

```
menu_message(8, 12, info_calculo2);

/* Inicialización de la respuesta en lazo cerrado */
for(ic = 0; ic <= PUNTOS; ic++)
 {
   Yout[ic] = 0;
   ERROR[ic] = 0;
 }

for(ic = 1; ic <= PUNTOS; ic++)
   {
    U = SET_POINT;

    if (CAM == 1)
     {
      if ((ic >= 80) && (ic < 120))
        U = 0;
     }

     REF[ic] = U;

     ERROR[ic] = REF[ic] - Yout[ic-1];

     Yout[ic] = RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER,
                      T_RETAR, ERROR[ic], 0, ic, CAM, fac, &ban1);

     _settextposition(13,42);
     printf("%d ",ic);
   }

/* Reset de las Condiciones Iniciales y punteros para una
   nueva simulación de la planta */
RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER, T_RETAR, U, 1, ic,
        CAM, fac, &ban1);

/* Almacenar los datos calculados en un archivo CERRADO.DAT */
_clearscreen( _GCLEARSCREEN );
menu_message(8, 4, info_archivo);
getch();

if (NULL == (archivo=fopen("A:CERRADO.DAT","w")))
   {
    _clearscreen(_GCLEARSCREEN);
    printf("ERROR AL ABRIR EL ARCHIVO !\n");
    printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
    getch();
    exit(0);
   }
```

```
for(ic=1; ic <= PUNTOS; ic++)
   fprintf(archivo,"%3.4f  %3.4f  %3.4f  %3.4f\n",
           ic*T_MUES, ERROR[ic], Yout[ic], REF[ic]);

if (fclose(archivo) != NULL)
   {
    printf("ERROR AL CERRAR EL ARCHIVO.\n");
    printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
    getch();
    exit(0);
   }

break;

case 3:

 /* Cálculo del controlador PID no interactuante
    ..........................................................
 */

 _clearscreen( _GCLEARSCREEN );
 menu_message(8, 12, info_calculo3);

 /* Inicialización de las condiciones iniciales */
 for(ic = 0; ic <= PUNTOS; ic++)
    {
     U_PID[ic] = 0;
     ERROR[ic] = 0;
     Yout[ic] = 0;
    }

 for(ic = 1; ic <= PUNTOS; ic++)
 {
  U = SET_POINT;

  if (CAM == 1)
    {
     if ((ic >= 80) && (ic < 120))
        U = 0;
    }

  REF[ic] = U;

  ERROR[ic] = REF[ic] - Yout[ic-1];

  U_PID[ic] = PID(KP,TI,TD,Yout[ic-1],ERROR[ic],U_PID[ic-1],0);

  Yout[ic] = RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER,
```

```
                T_RETAR, U_PID[ic], 0, ic, CAM, fac, &ban1);

    _settextposition(13,42);
    printf("%d ",ic);
  }

  /* Reset de las Condiciones Iniciales y punteros para una
     nueva simulación y compensación de la planta */
  PID(KP, TI, TD, Yout[0], ERROR[0], U_PID[0], 1);

  RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER, T_RETAR, U, 1, ic,
        CAM, fac, &ban1);

  /* Almacenar los datos calculados en un archivo PID.DAT */
  _clearscreen( _GCLEARSCREEN );
  menu_message(8, 4, info_archivo);
  getch();

  if (NULL == (archivo=fopen("A:PID.DAT","w")))
     {
      _clearscreen(_GCLEARSCREEN);
      printf("ERROR AL ABRIR EL ARCHIVO !\n");
      printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
      getch();
      exit(0);
     }

  for(ic=1; ic <= PUNTOS; ic++)
   fprintf(archivo,"%3.4f  %3.4f  %3.4f  %3.4f  %3.4f\n",
          ic*T_MUES, ERROR[ic], U_PID[ic], Yout[ic], REF[ic]);

  if (fclose(archivo) != NULL)
     {
      printf("ERROR AL CERRAR EL ARCHIVO.\n");
      printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
      getch();
      exit(0);
     }

  break;

 case 1:

  /* Cálculo del controlador DEADBEAT
     ..............................................
  */

  _clearscreen( _GCLEARSCREEN );
```

```
menu_message(8, 9, info_calculo4);

/* Inicialización de las condiciones iniciales */
for(ic = 0; ic <= PUNTOS; ic++)
   {
    U_DEAD[ic] = 0;
    ERROR[ic] = 0;
    Yout[ic] = 0;
   }

for(ic = 1; ic <= PUNTOS; ic++)
{
 U = SET_POINT;

 if (CAM == 1)
   {
    if ((ic >= 80) && (ic < 120))
       U = 0;
   }

 REF[ic] = U;

 ERROR[ic] = REF[ic] - Yout[ic-1];

 U_DEAD[ic] = DEADBEAT( GRAD_DEN, NUMER_D, DENOM_D,ERROR[ic],
                        U_DEAD[ic-1], T_RETAR, ic, 0);

 Yout[ic] = RUNGE( GRAD_NUM, GRAD_DEN, DENOM, NUMER,
                   T_RETAR, U_DEAD[ic], 0, ic, CAM, fac, &ban1);

 _settextposition(13,42);
 printf("%d ",ic);
}

/* Reset de las Condiciones Iniciales y punteros para una
   nueva simulación y compensación de la planta */

DEADBEAT( GRAD_DEN, NUMER_D, DENOM_D, ERROR[0], U_DEAD[0],
          T_RETAR, ic, 1);

RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER, T_RETAR, U, 1, ic,
      CAM, fac, &ban1);

/* Almacenar los datos calculados en un archivo DEADBEAT.DAT */
_clearscreen( _GCLEARSCREEN );
menu_message(8, 4, info_archivo);
getch();
```

```
if (NULL == (archivo=fopen("A:DEADBEAT.DAT","w")))
   {
    _clearscreen(_GCLEARSCREEN);
    printf("ERROR AL ABRIR EL ARCHIVO !\n");
    printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
    getch();
    exit(0);
   }

for(ic=1; ic <= PUNTOS; ic++)
 fprintf(archivo,"%3.4f  %3.4f  %3.4f  %3.4f  %3.4f\n",
        ic*T_MUES, ERROR[ic], U_DEAD[ic], Yout[ic], REF[ic]);

if (fclose(archivo) != NULL)
   {
    printf("ERROR AL CERRAR EL ARCHIVO.\n");
    printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
    getch();
    exit(0);
   }

break;

case 5:

/* Cálculo del controlador SELF-TUNING
   ...................................
*/

_clearscreen( _GCLEARSCREEN );
menu_message(8, 6, info_calculo5);

/* Inicialización de las condiciones iniciales */
for(ic = 0; ic <= PUNTOS; ic++)
   {
    U_SELF[ic] = 0;
    ERROR[ic] = 0;
    Yout[ic] = 0;
   }

for(ic = 1; ic <= PUNTOS; ic++)
{
 U = SET_POINT;

 if (CAM == 1)
   {
    if ((ic >= 80) && (ic < 120))
       U = 0;
```

```
    }

    REF[ic] = U;

    ERROR[ic] = REF[ic] - Yout[ic-1]/H;

    U_SELF[ic] = SELF( GRAD_NUM, GRAD_DEN ,NUMER_D, DENOM_D,
                       ERROR[ic], U_SELF[ic-1], T_RETAR, ic,
                       tpolos, &H, 0, NUMER, DENOM, fac,CAM,&ban1);

    Yout[ic] = RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER,
                       T_RETAR, U_SELF[ic], 0, ic, CAM, fac, &ban1);

    _settextposition(13,42);
    printf("%d ",ic);
  }

  _clearscreen( _GCLEARSCREEN );
  /* Reset de las Condiciones Iniciales y punteros para una
     nueva simulación y compensación de la planta */

  SELF(GRAD_NUM, GRAD_DEN, NUMER_D, DENOM_D, ERROR[0],
       U_SELF[0], T_RETAR, ic, tpolos, &H, 1, NUMER, DENOM, fac, CAM,
       &ban1);

  RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER, T_RETAR, U, 1, ic,
        CAM, fac, &ban1);

  /* Almacenar los datos calculados en un archivo SELF-TUNING.DAT */
  _clearscreen( _GCLEARSCREEN );
  menu_message(8, 4, info_archivo);
  getch();

  if (NULL == (archivo=fopen("A:SELF.DAT","w")))
     {
       _clearscreen(_GCLEARSCREEN);
       printf("ERROR AL ABRIR EL ARCHIVO !\n");
       printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
       getch();
       exit(0);
     }

  for(ic=1; ic <= PUNTOS; ic++)
   fprintf(archivo,"%3.4f  %3.4f  %3.4f  %3.4f  %3.4f\n",
          ic*T_MUES, ERROR[ic], U_SELF[ic], Yout[ic], REF[ic]);

  if (fclose(archivo) != NULL)
     {
```

```
        printf("ERROR AL CERRAR EL ARCHIVO.\n");
        printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
        getch();
        exit(0);
       }

    break;

    case 6:

    /* Controlador Experto 1: Conecta a la planta la mejor acción
       de control en cada período de muestreo.
       ..........
    */

    _clearscreen( _GCLEARSCREEN );
    menu_message(8, 9, info_calculo6);

    /* Inicialización de las condiciones iniciales */
    for(ic = 0; ic <= PUNTOS; ic++)
        {
         U_PID[ic] = 0;
         U_DEAD[ic] = 0;
         U_SELF[ic] = 0;
         U_EXPERT[ic] = 0;
         ERROR[ic] = 0;
         ERRORs[ic] = 0;
         Yout[ic] = 0;
        }

    for(ic = 1; ic <= PUNTOS; ic++)
    {
     U = SET_POINT;

     if (CAM == 1)
       {
        if ((ic >= 80) && (ic < 120))
           U = 0;
       }

     REF[ic] = U;

     ERROR[ic] = REF[ic] - Yout[ic-1];

     ERRORs[ic] = REF[ic] - Yout[ic-1]/11;

     DERIV_E = ERROR[ic] - ERROR[ic-1];
```

```
U_PID[ic] = PID(KP, TI, TD, Yout[ic-1], ERROR[ic],
                U_EXPERT[ic-1], 0);

U_DEAD[ic] = DEADBEAT( GRAD_DEN, NUMER_D, DENOM_D,
                       ERROR[ic], U_EXPERT[ic-1], T_RETAR, ic, 0);

U_SELF[ic] = SELF( GRAD_NUM, GRAD_DEN ,NUMER_D, DENOM_D,
                   ERRORs[ic], U_EXPERT[ic-1], T_RETAR, ic,
                   tpolos, &H, 0, NUMER, DENOM, fac, CAM,&ban1);

U = EXPERTO1( ERROR[ic], DERIV_E, U_PID[ic], U_DEAD[ic],
              U_SELF[ic], SET_POINT, ic, GRAD_DEN, REF[ic]);

U_EXPERT[ic] = ACTUADOR( LIMITE, U);

Yout[ic] = RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER,T_RETAR,
                 U_EXPERT[ic], 0, ic, CAM, fac, &ban1);

_settextposition(13,42);
printf("%d ",ic);
}

/* Reset de las Condiciones Iniciales y punteros para una
   nueva simulación y compensación de la planta */

PID( KP, TI, TD, Yout[0], ERROR[0], U_EXPERT[0], 1);

DEADBEAT( GRAD_DEN, NUMER_D, DENOM_D, ERROR[0], U_EXPERT[0],
          T_RETAR, ic, 1);

SELF( GRAD_NUM, GRAD_DEN, NUMER_D, DENOM_D, ERRORs[0],
      U_EXPERT[0], T_RETAR, ic, tpolos, &H, 1, NUMER, DENOM,
      fac, CAM, &ban1);

RUNGE( GRAD_NUM, GRAD_DEN, DENOM, NUMER, T_RETAR, U, 1, ic,
       CAM, fac, &ban1);

/* Almacenar los datos calculados en un archivo EXPERTO1.DAT */
_clearscreen( _GCLEARSCREEN );
menu_message(8, 4, info_archivo);
getch();

if (NULL == (archivo=fopen("A:EXPERTO1.DAT","w")))
   {
     _clearscreen(_GCLEARSCREEN);
     printf("ERROR AL ABRIR EL ARCHIVO !\n");
     printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
     getch();
```

```
        exit(0);
      }

  for(ic=1; ic <= PUNTOS; ic++)
   fprintf(archivo,"%3.4f  %3.4f  %3.4f  %3.4f  %3.4f\n",
          ic*T_MUES, ERROR[ic], U_EXPERT[ic], Yout[ic], REF[ic]);

  if (fclose(archivo) != NULL)
     {
      printf("ERROR AL CERRAR EL ARCHIVO.\n");
      printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
      getch();
      exit(0);
     }

  break;

case 7:

  /* Controlador Experto 2: Conecta a la planta una acción
     de control resultante, en cada período de muestreo.
     ..........
  */

  _clearscreen( _GCLEARSCREEN );
  menu_message(8, 9, info_calculo7);

  /* Inicialización de las condiciones iniciales */
  for(ic = 0; ic <= PUNTOS; ic++)
     {
      U_PID[ic] = 0;
      U_DEAD[ic] = 0;
      U_SELF[ic] = 0;
      U_EXPERT[ic] = 0;
      ERROR[ic] = 0;
      ERRORs[ic] = 0;
      Yout[ic] = 0;
     }

  for(ic = 1; ic <= PUNTOS; ic++)
  {
   U = SET_POINT;

   if (CAM == 1)
     {
      if ((ic >= 80) && (ic < 120))
         U = 0;
     }
```

```
REF[ic] = U;

ERROR[ic] = REF[ic] - Yout[ic-1];

ERRORs[ic] = REF[ic] - Yout[ic-1]/H;

DERIV_E = ERROR[ic] - ERROR[ic-1];

U_PID[ic] = PID(KP, TI, TD, Yout[ic-1], ERROR[ic],
                U_EXPERT[ic-1], 0);

U_DEAD[ic] = DEADBEAT( GRAD_DEN, NUMER_D, DENOM_D,
                       ERROR[ic], U_EXPERT[ic-1], T_RETAR, ic, 0);

U_SELF[ic] = SELF( GRAD_NUM, GRAD_DEN ,NUMER_D, DENOM_D,
                   ERRORs[ic], U_EXPERT[ic-1], T_RETAR, ic,
                   tpolos, &H, 0, NUMER, DENOM, fac, CAM,&ban1);

U = EXPERTO2( ERROR[ic], DERIV_E, U_PID[ic], U_DEAD[ic],
              U_SELF[ic], SET_POINT, ic, GRAD_DEN, REF[ic]);

U_EXPERT[ic] = ACTUADOR( LIMITE, U);

Yout[ic] = RUNGE(GRAD_NUM, GRAD_DEN, DENOM, NUMER,T_RETAR,
                 U_EXPERT[ic], 0, ic, CAM, fac, &ban1);

_settextposition(13,42);
printf("%d ",ic);
}

/* Reset de las Condiciones Iniciales y punteros para una
   nueva simulación y compensación de la planta */

PID( KP, TI, TD, Yout[0], ERROR[0], U_EXPERT[0], 1);

DEADBEAT( GRAD_DEN, NUMER_D, DENOM_D, ERROR[0], U_EXPERT[0],
          T_RETAR, ic, 1);

SELF( GRAD_NUM, GRAD_DEN, NUMER_D, DENOM_D, ERRORs[0],
      U_EXPERT[0], T_RETAR, ic, tpolos, &H, 1, NUMER, DENOM,
      fac, CAM, &ban1);

RUNGE( GRAD_NUM, GRAD_DEN, DENOM, NUMER, T_RETAR, U, 1, ic,
       CAM, fac, &ban1);

/* Almacenar los datos calculados en un archivo EXPERT02.DAT */
_clearscreen( _GCLEARSCREEN );
menu_message(8, 1, info_archivo);
```

```
    getch();

    if (NULL == (archivo=fopen("A:EXPERTO2.DAT","w")))
      {
        _clearscreen(_GCLEARSCREEN);
        printf("ERROR AL ABRIR EL ARCHIVO !\n");
        printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
        getch();
        exit(0);
      }

    for(ic=1; ic <= PUNTOS; ic++)
     fprintf(archivo,"%3.4f   %3.4f   %3.4f   %3.4f   %3.4f\n",
             ic*T_MUES, ERROR[ic], U_EXPERT[ic], Yout[ic], REF[ic]);

    if (fclose(archivo) != NULL)
       {
        printf("ERROR AL CERRAR EL ARCHIVO.\n");
        printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR: \n");
        getch();
        exit(0);
       }

    break;

  case 8:

    /* Regresar al menu principal
       ..........................................
    */

    menu_erase(save_drop_menu );
    break;

  }

  _clearscreen( _GCLEARSCREEN );
  box_charfill( 1, 1, 25, 80, ' ');
  menu_message( 9, 8, info_box );

  break;


/* Salida del programa
   ...................
*/

case 3:
```

```
            menu_erase( menu_drop( 5, 49, drop_quit, &n ));
            if ( n == 2 )
               quit_flag = 1;
            break;

           default:
            /* (Se ingnorará la tecla Esc en este nivel) */

            break;
           }

      }

   /* Borrar la pantalla antes de salir */
   _clearscreen( _GCLEARSCREEN );

}
```

```
/*-------------------------------------------------------------------
  Nombre      :   FUNCION.H
  Tipo        :   Include
  Descripción :   Prototipos y Definiciones para los módulos: CEXPER.C,
                  RAICES.C, SIMULAR.C, CONTROL.C, IDENTIFI.C, DISCRETO.C
                  y BASE.C, funciones: raizpoli2(), fp(), fz(), tmin()
-------------------------------------------------------------------
*/
#define MAX 10
#define PUNTOS 200
#define getrandom( min, max ) ((rand() % (int)((max) - (min))) + (min) + 1)

void raizpoli2( int, double[], double[], double[]);
void fp( int, double[], double, double, double*, double*);
void fz( int, double[], double, double, double*, double*);
double tmin( int, double[], double[]);
double mayor( int, double[]);

double RUNGE( int, int, double[], double[], int, double, int, int, int,
              double, int*);
double RETARDO( double, int, int);
double ACTUADOR( double, double);
void EVALUA( int, double[], double, double[], double[]);

double PID( double, double, double, double, double, double, int);
double DEADBEAT( int, double[], double[], double, double, int, int, int);
double SELF( int, int, double[], double[], double, double, int, int, double[],
             double*, int, double[], double[], double, int, int*);

double ECUNR( double[][MAX+1], double[], double[], int);
void SUMMAT( double[][MAX+1], double[][MAX+1], double[][MAX+1], int, int,
             int);
void MULTMAT( double[][MAX+1], double[][MAX+1], double[][MAX+1], int, int,
              int, int);
void MULTFAC( double[][MAX+1], double, double[][MAX+1], int, int);
void COPIAM( double[][MAX+1], double[][MAX+1], int, int);
void IDENT( int, int, double[], double[], double[][MAX+1]);

void DISC( int, int, double[], double[], double[], double[]);

double EXPERTO1( double, double, double, double, double, double, int, int,
                 double);

double EXPERTO2( double, double, double, double, double, double, int, int,
                 double);
```

```
/* ------------------------------------------------------------------
   Nombre      :   RAICES.C
   Tipo        :   Módulo de CEXPER.C
   Descripción:    Utiliza el método de Newton Raphson para calcular las
                   raíces del polinomio característico. Calcula además la
                   menor constante de tiempo del sistema.
                   Incluye las funciones: raizpoli2( ), fp( ), fz( ), tmin( )
                   mayor( ).
   ------------------------------------------------------------------
*/


/* ..................................................................
   Función para cálculo de las raíces del polinomio. Utiliza el método
   de descenso más pronunciado con escalamiento de raíces.
   n          - grado del polinomio.
   coef       - arreglo con los coeficientes en orden descendente de
                potencias.
   preal      - arreglo con partes reales de las raíces.
   pimag      - arreglo con partes imaginarias de las raíces.
   ..................................................................
*/

#include <math.h>
#include "funcion.h"

void raizpoli2(int n, double coef[], double preal[], double pimag[])
{
  double C[MAX], A1[MAX];
  double eps1 = .00000001, eps2 = eps1*eps1;
  double X, F1, F2, H, DX, DY, R, S, T, Y, U, V, U1, V1;
  int j, ngrado = n, numraiz = 0;

  for (j=1; j<=ngrado; j++)
       {
         preal[j] = 0;
         pimag[j] = 0;
         A1[j] = coef[j+1] / coef[1];
       }
  while (ngrado>0)
       {
        int jj;
        if (fabs(A1[ngrado]) < eps1)
              {
               ++numraiz;
               --ngrado;
               goto final;
              }
```

```
if (ngrado == 1)
        {
         ++numraiz;
         preal[numraiz] = -A1[ngrado];
         ngrado = 0;
        }
else
 if (ngrado == 2)
        {
              X = -A1[1] / 2;
              ++numraiz;
              T = X * X - A1[2];
              if (T >= 0)
              {
                T = sqrt(T);
                preal[numraiz] = X + T;
                preal[numraiz+1] = X - T;
               }
              else
               {
                preal[numraiz] = X;
                preal[numraiz+1] = preal[numraiz];
                pimag[numraiz] = sqrt(-T);
                pimag[numraiz+1] = -pimag[numraiz];
               }
              ngrado = 0;
        }
else

 * Grado es mayor que 2, utiliza método de descenso
   más pronunciado.
   Escalar las raíces del polinomio de modo que  el
   producto de los valores absolutos de las mismas
   sea 1.
*/
{
T = fabs(A1[ngrado]);
if (fabs(T-1) > eps1)
{
  H = pow(T,(1/ngrado));
 T = 1;
 for (j=1; j<=ngrado; j++)
  {
   T = T * H;
    C[j] = A1[j] / T;
  }
}
else
```

```
{
 for (jj=1; jj<=ngrado; jj++)
  C[jj]= A1[jj];
 H = 1;
 }

/* Valores iniciales de arranque dentro del círculo unitario */
X = .7;
Y = .6;

fz(ngrado, C, X, Y, &U, &V);

/* Valor absoluto del polinomio */
F1 = U * U + V * V;

/* delta x */
DX = 1;

/* delta y */
DY = 1;

while (F1 > eps2 || (fabs(DX) > eps1 && fabs(DY) > eps1))
{
comienzo:
 /* Calcular DX y DY */
 fp(ngrado, C, X, Y, &U1, &V1);
 T = U1 * U1 + V1 * V1;
 DX = -(U * U1 + V * V1) / T;
 DY=  (U * V1 - V * U1) / T;
paso:

 /* Avanzar un paso */
 X += DX;
 Y += DY;

 /* Calcular polinomio en punto nuevo */
 fz(ngrado, C, X, Y, &U, &V);

 F2 = U * U + V * V;
 if (F2 < eps2 && fabs(DX) < eps1 && fabs(DY) < eps1)
  break;
 else
  if (F2 < F1)
   {
   F1 = F2;
   goto comienzo;
   }
  else
```

```
    {
     X -= DX;
     Y -= DY;
     DX = .8*DX;
     DY = .8*DY;
     goto paso;
     }
   }


  /* Multiplicar raíz encontrada por factor de escalamiento */
  X *= H;
  Y *= H;

  /*  Si la raíz encontrada es real, dividir el polinomio para
      z - raíz y disminuir el grado en 1, o si la raíz encontrada
      es compleja, dividir el polinomio para (z-raíz)(z-conjugada)
      y disminir el grado en 2. */

  if (fabs(Y) <= eps1) Y=0;
 ++numraiz;
 preal[numraiz] = X;
 pimag[numraiz] = Y;
  if (fabs(Y) > eps1)
  {
   ++numraiz;
   preal[numraiz] = X;
    pimag[numraiz] = -Y;
    R = -2 * X;
   S = X * X + Y * Y;
    A1[1] = A1[1] - R;
    A1[2] = A1[2] - R * A1[1] - S;
   if (ngrado >= 5)
     for (j=3; j <= ngrado - 2; j++)
     A1[j] = A1[j] - R * A1[j-1] - S * A1[j-2];
   ngrado -= 2;
  }
 else
  {
   T = 1;
    for (j=1; j <= ngrado-1; j++)
    {
     T = T * X + A1[j];
     A1[j] = T;
    }
   --ngrado;
  }
}
```

```
final: ;
 }
}


/*.....
  Función para la evaluación de la derivada F'(Z) = U1 + jV1
  .....
*/

void fp(int N0, double C[], double X, double Y, double *U1, double *V1)
{
  int j;
  double T;

  *U1=N0;
  *V1=0;
  for (j=1; j<=N0-1; j++)
   {
    T = *U1 * X - *V1 * Y + (N0 - j) * C[j];
    *V1 = *V1 * X + *U1 * Y;
    *U1 = T;
   }
}



/* .....
   Función para evaluación del polinomio F(Z) = U + jV
   Utiliza el método de Horner.
   .....
*/

void fz(int N0, double C[], double X, double Y, double *U, double *V)
{
  double T;
  int j;
  *U = 1;
  *V = 0;
  for (j=1; j<=N0; j++)
        {
          T = *U * X - *V * Y + C[j];
          *V = *V * X + *U * Y;
          *U = T;
        }
}
```

```
/* ..........
   Función para calcular la menor constante de tiempo de la planta
   ..........
*/

double tmin(int n, double R[], double I[])

{
  double Rmay, Imay, Tmin;
  double TRmin, TImin;

  /* Obtiene la raíz real de mayor valor */
  Rmay = mayor(n, R);

  /* Obtiene la raíz imaginaria de mayor valor */
  Imay = mayor(n, I);

  if (Rmay == 0)
    TRmin = 0;
  else
    /* Cálculo de la contante de tiempo del eje real */
    TRmin = 1./Rmay;

  if (Imay == 0)
    TImin = 0;
  else
    /* Cálculo de la constante de tiempo del eje imaginario */
    TImin = 2*3.1416/Imay;

  if((TRmin) && (TImin) != 0)
   {
    /* Se obtiene la menor constante de tiempo del sistema */
    if (TImin < TRmin)
     Tmin = TImin;
    else
     Tmin = TRmin;
   }

  if (TRmin == 0)
    Tmin = TImin;

  if (TImin == 0)
    Tmin = TRmin;

  return(Tmin);
}
```

```
/* ..........
   Función para obtener el mayor elemento de un arreglo
   ..........
*/

double mayor(int n, double B[])
{
  double may;
  int i;

  /* Considera el primer elemento como mayor */
  may = fabs(B[1]);
  for (i=2; i<=n; i++)
    {
     /* Compara el primer elemento con el siguiente en caso de ser
          menor lo cambia */
      if (may < fabs(B[i]))
          may = fabs(B[i]);
    }

  /* Retorna el mayor elemento del arreglo */
  return(may);
}
```

```
/* ------------------------------------------------------------------
   Nombre      : SIMULAR.C
   Tipo        : Módulo de CEXPER.C
   Descripción : Realiza la simulación de la planta.
                 Incluye las funciones: RUNGE( ), RETARDO( ) y
                 ACTUADOR( ).
   ------------------------------------------------------------------
*/

#include "funcion.h"


/* ..................................................................
   Función para la integración de las variables de estado del sistema
   Se emplea el método de integración de Runge Kutta de cuarto orden.
   ..................................................................
*/

double RUNGE(int m, int n, double DENOM1[], double NUMER1[], int Tret,
             double Uk, int reset, int ic, int CAM, double fac, int *ban)
{

  extern double T_MUES;
  double K1[MAX+1], K2[MAX+1], K3[MAX+1], K4[MAX+1], XS[MAX+1], DXI[MAX+1];
  static double XI[MAX+1];
  double Uretar;
  double h = T_MUES/10;
  double Y, Ysalida;
  double NUEVO1, VIEJO1;
  int i,j;


  /* Reset de los arreglos utilizados
     ............................................
  */
  if (reset)
     {
       for (i=0; i <= MAX; i++)
            XI[i] = 0;
       RETARDO(Uk, Tret, reset);
       if ((CAM == 1) && (*ban == 1))
         {
          DENOM1[2] = VIEJO1;
         }
      *ban = 1;
       return(0);
     }
```

```
/* Si el proceso tiene tiempo de retardo, se procede a simular el mismo
   ..................................................................
*/
if (Tret)
  {
    Uretar = RETARDO(Uk, Tret, reset);
    Uk = Uretar;
  }

/* Cambio del coeficiente a(n-1) de la F.T. G(S):  à(n-1) = a(n-1) * fac
   ..................................................................
*/
  if ((CAM == 1) && (ic == 120) && (*ban == 1))
    {
      VIEJO1 = DENOM1[2];
      NUEVO1 = fac * DENOM1[2];
      DENOM1[2] = NUEVO1;
    }

/* Integración de las ecuaciones de estado
   ..........................................

   Ecuaciones de Runge Kutta de cuarto orden

   El paso de integración es T_MUES/10. Por lo tanto se realiza 10
   iteracciones en cada período de muestreo
*/

/* Inicio del lazo para la integración */
for(j = 1; j <= 10; j++)
  {

    /* Evaluación de las variables de estado */

    /* f(tk, xk) */
    EVALUA(n, DENOM1, Uk, XI, DXDT);

    /* Cálculo de K1 */
    for(i = 1; i <= n; i++)
       {
         * K1 = h.f(tk, xk) */
         K1[i] = h*DXDT[i];

         /* xk + K1/2 */
         XS[i] = XI[i] + K1[i]/2;
       }

    /* Evaluacion de las variables de estado */
```

```
/* f(tk+h/2, xk+K1/2) */
EVALUA(n, DENOM1, Uk, XS, DXDT);

/* Cálculo de K2 */
for(i = 1; i <= n; i++)
   {
     /* K2 = h.f(tk+h/2, xk+K1/2) */
     K2[i] = h*DXDT[i];

     /* xk + K2/2 */
     XS[i] = XI[i] + K2[i]/2;
   }

/* Evaluación de las variables de estado */

/* f(tk+h/2, xk+K2/2) */
EVALUA(n, DENOM1, Uk, XS, DXDT);

/* Cálculo de K3 */
for(i = 1; i <= n; i++)
   {
     /* K3 = h.f(tk+h/2, xk+K2/2) */
     K3[i] = h*DXDT[i];

     /* xk + K3 */
     XS[i] = XI[i] + K3[i];
   }

/* Evaluación de las variables de estado */

/* f(tk+h, xk+K3) */
EVALUA(n, DENOM1, Uk, XS, DXDT);

/* Cálculo de K4 */
for(i = 1; i <= n; i++)
     /* K4 = h.f(tk+h, xk+K3) */
     K4[i] = h*DXDT[i];

  /* Obtención de las variables de estado integradas */
  for(i = 1; i <= n; i++)
     XI[i] = XI[i] + (K1[i] + 2*K2[i] + 2*K3[i] + K4[i])/6;

  /* Cálculo de la respuesta de la planta */
  Y = 0;
  for(i = 1; i <= m + 1; i++)
     Y = Y + NUMER1[m - i + 2]*XI[i];
  Ysalida = Y;
}
```

```
    return(Ysalida);
}

/* ..........
   Función para simular el retardo en la planta
   ..........
*/

double RETARDO(double Uo, int Tr, int reset)
{
   int i;

   /* Se asigna el valor del retardo al indicador de un arreglo */
   static int Ip = 0;

   /* Se define un arreglo que almacena los valores de Uo */
   static double URET[PUNTOS + 100];

   if (reset)
     {
      Ip = 0;
      for (i=0; i <= PUNTOS; i++)
          URET[i] = 0;
      return(0);
     }
   if (Ip == 0)
      Ip = Tr;


   /* Se almacena el valor de Uo en una localidad desplazada Tr veces en
      el arreglo */
   URET[Ip++] = Uo;

   /* Retorna el valor de Uo retardada Tr veces */
   return(URET[Ip - Tr -1]);

}


/* ..........
   Función para evaluar las variables de estado de la planta.
   ..........
*/

void EVALUA(int n, double DENOM2[], double Uk1, double XI[], double DXDT[])
{
   int i;
```

```
  double SUM = 0;

  /* Evaluación de las variables de estado de i=1 a i=n-1 */
  for (i = 1; i <= n-1; i++)
     DXDT1[i] = XI1[i+1];

  /* Evaluación de la variable i=n */
  for (i = 1; i <= n; i++)
     SUM = SUM + (DENOM2[n - i +2]/DENOM2[1])*XI1[i];
  DXDT1[n] = U[k] - DENOM2[1] - SUM;
}


/* ..........................................................................
   Función para simular el límite del actuador de la planta
   ..........................................................................
*/

double ACTUADOR(double L, double X)
{
  double YL;

  /* Si el valor medido es menor al límite inferior, fije el valor
     de la salida en -L */
  if (X <= -L)
     YL = -L;

  /* Si el valor medido es mayor al límite superior, fije el valor
     de la salida en L */
  else if (X >= L)
     YL = L;

  /* Caso contrario el valor medido se encuentra dentro del rango
     permitido */
  else
     YL = X;

  return(YL);
}
```

```
/* ---------------------------------------------------------------------
   Nombre          : CONTROL.C
   Tipo            : Módulo de CEXPER.C
   Descripción     : Realiza los cálculo de las acciones de control.
                     Incluye las funciones: PID( ), DEADBEAT( ), SELF( ).
   ---------------------------------------------------------------------
*/

#include <graph.h>
#include "funcion.h"

/* ..................................................................
   Función para el cálculo del controlador PID no interactuante
   ..................................................................
*/

double PID(double Kp, double Ti, double Td, double Yin, double error,
           double Uin, int reset)

{
   extern double T_MUES;
   double BETA, KP2;
   static double INTEGRAL;
   static double DERIVATIVO;
   static double Yold;
   double Upi,Upid;
   double D1, D2, GAMA;
   int i;

   if (reset)
     {
      INTEGRAL = 0;
      DERIVATIVO = 0;
      Yold = 0;
      return(0);
     }

   /* Cálculo del término Proporcional Integral: PI */

   KP2 = Kp * (1 + T_MUES / (2 * Ti));
   BETA= (2*Ti - T_MUES) / (2*Ti + T_MUES);
   INTEGRAL = BETA*INTEGRAL + (1-BETA)*Uin;
   Upi = KP2 * error + INTEGRAL;

   /* Cálculo del término Proporcional Derivativo: PD */

   GAMA = Td / T_MUES;
   D1 = (0.2*GAMA - 1) / (1 + 0.2*GAMA);
```

```
  D2 = 2*GAMA / (1 + 0.2*GAMA);
  DERIVATIVO = D1*DERIVATIVO + D2*(Yin - Yold);

  /*  Cálculo de la acción PID */

  Upid = Upi - KP2*DERIVATIVO;
  Yold = Yin;
  return (Upid);
}


/* ..........................................................................
   Función para el cálculo del controlador DEADBEAT
   ..........................................................................
*/

double DEADBEAT( int n, double B[], double A[], double error, double Uin,
                 int d, int k, int reset)
{
  int i;
  double SUM = 0;
  static double E[PUNTOS +1];
  static double U[PUNTOS +1];
  double q[MAX + 1], p[MAX + 1];
  double Uc;

  if (reset)
    {
      for (i = 0; i <= PUNTOS; i++)
      {
        E[i] = 0;
        U[i] = 0;
      }
      return(0);
    }


  /* Cálculo de los coeficientes del controlador DEADBEAT */

  for (i = 1; i <= n; i++)
    SUM = SUM + B[i];

  q[0] = 1 / SUM;

  for (i = 1; i <= n; i++)
  {
    q[i] = A[i]*q[0];
    p[i] = B[i]*q[0];
```

```
}

/* En caso que el puntero trate de direccionar localidades inferiores a
   cero, se asegura que estos valores se los considere cero.
*/

if ( d )
{
  if ((k - n - d) < 0)
      U[k - n - d] = 0;
}

/* Inicialización de valores para el cálculo de la ley de control */

U[k - 1] = Uin;
E[k] = error;
SUM = 0;

/* Cálculo de la respuesta del controlador DEADBEAT */

for (i = 1; i <= n; i++)
{
  SUM = SUM + p[i]*U[k - i - d] + q[i]*E[k -i];
}

U[k] = SUM + q[0]*E[k];
Uc = U[k];
return (Uc);
}


/*
  ..........................................................................
  Función para el cálculo del controlador SELF-TUNING
  ..........................................................................
*/

double SELF( int m, int n, double B[], double A[], double error, double Uin,
             int d, int k, double t[], double *S, int reset, double NUM[],
             double DEN[], double fac, int CAM, int *ban)
{
  int i, j;
  int Mm, nf, ng;
  double SUM1 = 0, SUM2 = 0;
  static double PS[PUNTOS +1];
  static double tS[PUNTOS +1];
  double PARAM[MAX+1][MAX+1];
  double Aold[MAX+1], Bold[MAX+1];
```

```
double Ac[MAX+1], Be[MAX+1];
double Mc[MAX + 1][MAX + 1];
double Mi[MAX + 1];
double CONT[MAX+1];
double F[MAX+1], G[MAX+1];
double aux1, aux2, aux3, aux4;
double DETR, Uc;
double VIEJO1, NUEVO1;

if (reset)
  {
    for (i = 0; i <= PUNTOS; i++)
    {
       ES[i] = 0;
       US[i] = 0;
    }
    *S = 1;
    if ((CAM == 1) && (*ban == 0))
      {
        DEN[2] = VIEJO1;
        memcpy( B, Bold, sizeof(double)*(MAX+1));
        memcpy( A, Aold, sizeof(double)*(MAX+1));
      }
    return(0);
  }

 * Inicialización de valores para el cálculo de la ley de control

   Control por asignación de polos: U(k) =  G(Z^-1)/[1 + F(Z^-1)]

   G(Z^-1) = G0 + G1(Z^-1)+.....+ Gng(Z^-ng)
   F(Z^-1) = F1(Z^-1) + F2(Z^-2)+.....+ Fnf(Z^-nf)

   ng = n - 1;     nf = n + d - 1

   n = orden del sistema
   d = retardo

*/

 * Orden del controlador */
nf = n + d -1;
ng = n - 1;
Mm = nf + ng + 1;

 * Cambiar el parámetro a(n-1) de G(S) */
if((k == 120) && (CAM == 1))
 {
```

```
  *ban = 0;
  VIEJO1 = DEN[2];
  NUEVO1 = DEN[2] * fac;
  DEN[2] = NUEVO1;

  memcpy( Bold, B, sizeof(double)*(MAX+1));
  memcpy( Aold, A, sizeof(double)*(MAX+1));

  /* Recálculo de la función de transferencia discreta */
  DISC ( m, n, NUM, DEN, B, A);
}

/* Identificar los parámetros de la planta */
IDENT( n, d, A, B, PARAM);

/* Inicializar arreglos por utilizar */
for(i=0; i <= MAX; i++)
 {
  Ae[i] = 0;
  Be[i] = 0;
 }

/* Arreglo con los valores estimados de A[] */
for(i = 1; i <= n; i++)
  Ae[i] = -PARAM[i][1];

/* Arreglo con los valores estimados de B[] */
for(i = 1; i <= n; i++)
  Be[i] = PARAM[i+n][1];

/* Cálculo de los coeficientes del controlador

  Si la planta es de primer orden y no existe retardo
  nf = 0,  ng = 0 => Solamente existe G0 */

if ((n == 1) && (d == 0))
  G[0] = (t[1] - Ae[1]) / Be[1];

else
 {
   /* Formar la matriz de coeficientes y términos independientes */
   MATRIZ (Mm, nf, ng, n, d, Mc, Mi, Ae, Be, t);

   /* Resolver el sistema de ecuaciones lineales para obtener los
      coeficientes del controlador */
   DETR = ECLINR( Mc, Mi, CONT, Mm);

   if ( DETR == 0)
```

```
    *ban = 0;
    VIEJO1 = DEN[2];
    NUEVO1 = DEN[2] * fac;
    DEN[2] = NUEVO1;

    memcpy( Bold, B, sizeof(double)*(MAX+1));
    memcpy( Aold, A, sizeof(double)*(MAX+1));

    /* Recálculo de la función de transferencia discreta */
    DISC ( m, n, NUM, DEN, B, A);
  }

/* Identificar los parámetros de la planta */
IDENT( n, d, A, B, PARAM);

/* Inicializar arreglos por utilizar */
for(i=0; i <= MAX; i++)
  {
   Ac[i] = 0;
   Bc[i] = 0;
  }

/* Arreglo con los valores estimados de A[] */
for(i = 1; i <= n; i++)
   Ac[i] = -PARAM[i][1];

/* Arreglo con los valores estimados de B[] */
for(i = 1; i <= n; i++)
   Bc[i] = PARAM[i+n][1];

/* Cálculo de los coeficientes del controlador

   Si la planta es de primer orden y no existe retardo
   nf = 0,  ng = 0 => Solamente existe G0 */

if ((n == 1) && (d == 0))
   G[0] = (a[1] - Ac[1]) / Bc[1];

else
   {
     /* Formar la matriz de coeficientes y términos independientes */
     MATRIZ (Mm, nf, ng, n, d, Mc, Mi, Ac, Bc, 0);

      * Resolver el sistema de ecuaciones lineales para obtener los
        coeficientes del controlador */
     DETR = ECLINR( Mc, Mi, CONT, Mm);

     if ( DETR == 0)
```

```
    {
      _clearscreen(_GCLEARSCREEN);
      printf("ERROR AL RESOLVER EL SISTEMA. LA MATRIZ ES SINGULAR !\n");
      printf("PRESIONE CUALQUIER TECLA PARA CONTINUAR. \n");
      getch();
      exit(0);
    }
    else
    {
      /* Arreglo con los coeficientes de F(Z^-1) */
      for(i=1; i <= nf; i++)
        F[i] = CONT[i];

      /* Arreglo con los coeficientes de G(Z^-1) */
      for(i=1; i <= ng+1; i++)
        G[i-1] = CONT[nf+i];
    }
  }

/* Cálculo del compensador proporcional para obtener error en estado
   estable igual a cero:

        (1+F1+F2+..+Fnf)*(1+a1+a2+..+an)+(G0+G1+..+Gng)*(b1+b2+..+bn)
   S = ------------------------------------------------------------------------------
                          (G0+G1+..+Gng)*(b1+b2+..+bn)
*/

  aux1=0;
  aux2=0;
  aux3=0;
  aux4=0;

  for(i=1; i<=nf; i++)
    aux1=aux1 + F[i];

  for(i=1; i<=n; i++)
    aux2=aux2 + Ae[i];

  for(i=0; i<=ng; i++)
    aux3=aux3 + G[i];

  for(i=1; i<=n; i++)
    aux4=aux4 + Be[i];

  *S = ((1+aux1)*(1+aux2)+aux3*aux4)/(aux4*aux3);


  /* Inicializar datos */
```

```
  US[k - 1] = Uin;
  ES[k] = *S * error;

  /* Cálculo del valor de la acción de control SELF-TUNIG */
  for(i=1; i <= nf; i++)
    {
      if ((k - i) < 0)
          US[k - i] = 0;
      SUM1 = SUM1 - F[i]*US[k - i];

    }
  for(i=0; i <= ng; i++)
    {
      if ((k - i) < 0)
          ES[k - i] = 0;
      SUM2 = SUM2 + G[i]*ES[k - i];
    }

  US[k] = SUM1+SUM2;
  Uc = US[k];
  return (Uc);
}
```

```
/*-------------------------------------------------------------------------
   Nombre          : IDENTIFI.C
   Tipo            : Módulo de CEXPER.C
   Descripción     : Permite realizar la identificación del sistema.
                     Incluye las funciones: MATRIZ( ), ECLINR( ), SUMMAT( ),
                     MULTMAT( ), MULTFAC( ), COPIAM( ) e IDENT( ).
-------------------------------------------------------------------------
*/

#include "funcion.h"
#include <math.h>

/* ..........................................................................
   Función para formar la matriz de coeficientes y términos independientes
   qúe representan el sistema de ecuaciones lineales por resolver.
   ..........................................................................
*/

void MATRIZ(int M, int NF, int NG, int n, int D, double MA[][MAX+1],
            double MB[], double Ad[], double Bd[], double td[])
{
   int i, j;
   int fil, col;

   /* Inicializar la matriz de coeficientes y términos independientes
      con cero */

   for (i=0; i <= M; i++)
    {
     for ( j=0; j <= M; j++)
       MA[i][j] = 0;
     MB[i] = 0;
    }

   /* Formar la matriz de coeficientes */
   fil = 1;
   for (col=1; col <= NF; col++)
   {
     MA[fil++][col] = 1;
     for (i=1; i <= n; i++)
       MA[fil++][col] = Ad[i];
     fil = col + 1;
   }
   fil = D + 1;
   for (col=1; col <= NG+1; col++)
   {
     for (i=1; i <= n; i++)
```

```
      MA[fil++][NP+col] = Bd[i];
    fil = D + 1 + col;
  }

  /* Formar la matriz de términos independientes */
  for (i=1; i <= M; i++)
    MB[i] = td[i] - Ad[i];

}

/* ..........
   Función para resolver un sistema de ecuaciones algebraicas lineales con
   coeficientes reales.
   Método de reducción de Gauss con pivotaje de filas y columnas
   ..........
*/

double ECLINR ( double AC[][MAX+1], double BC[], double XC[], int N)

{
   double KK[MAX+1];
   double EPS = 0.00000001;
   double DET = 1;
   double T, aux, FACTOR, SUMA;
   int i, k, j;
   int IFIL, ICOL;

   for (i=1; i <= N; i++)
     KK[i] = i;

   /* Reducción de la matriz AC a la forma triangular superior */

   for (i=1; i < N; i++)
    {
      IFIL = i;
      ICOL = i;
      T = fabs(AC[i][i]);
      for (k=i; k <= N; k++)
       {
          for (j=i; j <= N; j++)
           {
             if (fabs(AC[k][j]) > T)
                {
                   IFIL = k;
                   ICOL = j;
                   T = fabs(AC[k][j]);
                }
```

```
    }
  }

  if (T < EPS)
  {
     DET = 0;
     return (DET);
  }

  /* Intercambio de filas IFIL con i */
  if (IFIL != i)
  {
     for (j=i; j <= N; j++)
     {
       aux = AC[IFIL][j];
       AC[IFIL][j] = AC[i][j];
       AC[i][j] = aux;
     }
     aux = BC[IFIL];
     BC[IFIL] = BC[i];
     BC[i] = aux;
     DET = -DET;
  }

  /* Intercambio de columnas ICOL con i */
  if (ICOL != i)
  {
     for (j=1; j <= N; j++)
     {
       aux = AC[j][ICOL];
       AC[j][ICOL] = AC[j][i];
       AC[j][i] = aux;
     }
     aux = KK[ICOL];
     KK[ICOL] = KK[i];
     KK[i] = aux;
     DET = -DET;
  }

  /* Pivotaje finalizado, continuar con reducción */
  for (k=i+1; k <= N; k++)
     if (fabs(AC[k][i]) > EPS)
     {
       FACTOR = -AC[k][i]/AC[i][i];

       /*Multiplicar fila i por FACTOR y sumar a fila k para
         obtener cero */
       for(j=i+1; j <= N; j++)
```

```
            AC[k][j] = AC[k][j] + FACTOR * AC[i][j];
          BC[k] = BC[k] + FACTOR * BC[i];
        }

    /* Actualizar el valor del determinante */
    DET = DET * AC[i][i];
  }

  DET = DET * AC[N][N];
  if (fabs(AC[N][N]) < EPS)
    {
      DET = 0;
      return (DET);
    }

  /* Fin de reducción. Comienzo de evaluación de incógnitas.
     Se evaluan en BC. */
  BC[N] = BC[N] / AC[N][N];
  for (i=N-1; i > 0; i--)
    {
      SUMA = 0;
      for (k=i+1; k <= N; k++)
        SUMA = SUMA + AC[i][k] * BC[k];
      BC[i] = (BC[i] - SUMA) / AC[i][i];
    }

  /* Intercambio de elementos de BC de acuerdo a vector auxiliar KK
     Resultado se almacena en XC. */
  for (i=1; i <= N; i++)
    {
      j = KK[i];
      XC[j] = BC[i];
    }
  return (DET);
}

/* ..........................................................................
   Función para sumar o restar matrices : AM[f][c] +/- BM[f][c] = R[f][c]
   ..........................................................................
*/

void SUMMAT( double AM[][MAX+1], double BM[][MAX+1], double R[][MAX+1],
             int f, int c, int op)

{
  int i, j;
```

```
  for(i=1; i <= f; i++)
    {
     for (j=1; j <= c; j++)
        {
          if (op == 0)
             R[i][j] = AM[i][j] + BM[i][j];
          else
             R[i][j] = AM[i][j] - BM[i][j];
        }
    }
}


/* ..........................................................................
   Función para multiplicar matrices: AM[f1][c1] * BM[f2][c2] = R[f1][c2]
   ..........................................................................
*/

void MULTMAT( double AM[][MAX+1], double BM[][MAX+1], double R[][MAX+1],
                    int f1, int c1, int f2, int c2)
{
   int i, j, k;

   for(i=1; i <= f1; i++)
     {
       for (j=1; j <= c2; j++)
          {
            R[i][j] = 0;
            for(k=1; k <= c1; k++)
               R[i][j] = R[i][j] + AM[i][k] * BM[k][j];

          }
     }
}


/* ..........................................................................
   Función para multiplicar una matriz por un factor : a*AM[f][c] = R[f][c]
   ..........................................................................
*/

void MULTFAC( double AM[][MAX+1], double a, double R[][MAX+1], int f, int c)
{
  int i, j;
  for(i=1; i <= f; i++)
    {
      for(j=1; j <= c; j++)
          R[i][j] = a * AM[i][j];
```

```
    }
}


/* ..........................................................................
   Función para copiar una matriz en otra: AM[f][c] = BM[f][c]
   ..........................................................................
*/

void COPIAM( double AM[][MAX+1], double BM[][MAX+1], int f, int c)
{
   int i, j;

   for(i=1; i <= f; i++)
     {
       for (j=1; j <= c; j++)
            BM[i][j] = AM[i][j];
     }
}


/* ..........................................................................
   Función para Identificación de los parámetros de un sistema.

   Método de Identificación: MINIMOS CUADRADOS RECURSIVO.
   ..........................................................................
*/

void IDENT( int n, int d, double AD[], double BD[], double PARAM[][MAX+1])
{
   int i, j, k;
   int n2;
   double aux, sum;
   double P[MAX+1][MAX+1], IDEN[MAX+1][MAX+1];
   double PXT[MAX+1][MAX+1], XPXT[MAX+1][MAX+1];
   double Ym[30+1], Um[30+1];
   double X[MAX+1][MAX+1], XT[MAX+1][MAX+1];
   double L[MAX+1][MAX+1], XPA[MAX+1][MAX+1];
   double Ei, Em;
   double LE[MAX+1][MAX+1], PARAM1[MAX+1][MAX+1];
   double LN[MAX+1][MAX+1], P1[MAX+1][MAX+1], P2[MAX+1][MAX+1];
   double FAC_OLV = 1 / 0.98;

   /* Inicializar la matriz de covarianza P[][] = α * I[][].
      Se considera una matriz diagonal de orden 2nx2n.
      Valor de α = 100000.
   */
   n2 = 2*n;
```

```
for (i=1; i <= n2; i++)
 {
  for (j=1; j <= n2; j++)
   {
    if ( j == i)
      {
        P[i][j] = 100000;
        IDEN[i][j] = 1;
      }
    else
      {
        P[i][j] = 0;
        IDEN[i][j] = 0;
      }
   }
 }
/* Inicializar el arreglo de los parámetros estimados */
for (i=0; i <= MAX; i++)
   PARAM[i][1] = 0;

/* Inicializar arreglos para 30 iteraciones */
for (i=0; i <= 30; i++)
  {
    Ym[i] = 0;
    Um[i] = 0;
  }

/* Identificación del sistema */
for (i=1; i <= 30; i++)
 {
   /* Inicialización para la generación del número randómico */
   srand( 12 );
   aux = getrandom(0,10);
   Um[i] = aux/10 - 0.5;
   sum = 0;

   /* Cálculo del valor de salida */
   for(j=1; j <= n; j++)
    {
      if ( d )
      {
        if ((i-j-d) < 0)
          Um[i-j-d] = 0;
      }

      if ((i - j) < 0)
        {
          Ym[i - j] = 0;
```

```
        Um[i - j] = 0;
      }
    sum = sum - AD[j]*Ym[i - j] + BD[j]*Um[i - j - d];
 }
Ym[i] = sum;

/* Formación de la matriz X(i)=[Y(i-1) .. Y(i-n) U(i-1) .. U(i-n)]
   se ha considerado el orden de identificación igual al orden del
   sistema en análisis => n
*/

k = i - 1;
for(j=1; j <= n; j++)
 {
    if (k < 0)
       Ym[k] = 0;
    X[1][j] = Ym[k];
    XT[j][1] = Ym[k];
    k--;
 }

k = i - 1 - d;
for(j=n+1; j <= n2; j++)
 {
    if (k < 0)
       Um[k] = 0;
    X[1][j] = Um[k];
    XT[j][1] = Um[k];
    k--;
 }

/* Aplicación del algoritmo de los MINIMOS CUADRADOS RECURSIVO */

/* P(i-1)*XT(i) => PXT(i) */
MULTMAT( P, XT, PXT, n2, n2, n2, 1);

/* X(i)*P(i-1)*XT(i) => XPXT(i) */
MULTMAT( X, PXT, XPXT, 1, n2, n2, 1);

/* P(i-1)*XT(i) [1 + X(i)*P(i-1)*XT(i)] => L(i) */
L1 = 1 / ( 1 + XPXT[1][1]);
MULTFAC( PXT, L1, L, n2, 1);

/* Em = Ym(i) - X(i)*Θ(i-1) */
MULTMAT( X, PARAM, XPA, 1, n2, n2, 1);
Em = Ym[i] - XPA[1][1];

/* Θest(i) = Θest(i-1) + L(i)*Em */
```

```
    MULTFAC( L, Em, LE, n2, 1);
    SUMMAT( PARAM, LE, PARAM1, n2, 1, 0);
    for(j=1; j <= n2; j++)
        PARAM[j][1] = PARAM1[j][1];

    /* I - L(i)*X(i) => P1(i) */
    MULTMAT( L, X, LX, n2, 1, 1, n2);
    SUMMAT( IDEN, LX, P1, n2, n2, 1);

    /* [I - L(i)*X(i)]*P(i-1) => P2 */
    MULTMAT( P1, P, P2, n2, n2, n2, n2);

    /* Actualización de la matriz de covarianza:
        P(i) = P(i-1) / 0.98
    */
    MULTFAC( P2, FAC_OLV, P, n2, n2);
  }
}
```

```
/* ---------------------------------------------------------------
   Nombre      : DISCRETO.C
   Tipo        : Módulo de CEXPER.C
   Descripción : Realiza la discretización de la planta.
                 Incluye la función: DISC( ).
   ---------------------------------------------------------------
*/

#include "funcion.h"


/* ..............................................................................
   Función para discretizar sistemas continuos en base de sus ecuaciones
   de estado (se incluye Z.O.H.) para luego generar la función de trans-
   ferencia discreta (G(Z)).
   ..............................................................................
 */

void DISC(int m, int n, double NUMERC[], double DENOMC[], double NUMERD[],
          double DENOMD[])
{
  extern double T_MUES;
  int i,j,k;
  double Qk, sum;
  double AC[MAX+1][MAX+1], BC[MAX+1][MAX+1], C[MAX+1][MAX+1];
  double AD[MAX+1][MAX+1], BD[MAX+1][MAX+1];
  double ID[MAX+1][MAX+1], FI[MAX+1][MAX+1];
  double Aaux1[MAX+1][MAX+1], Aaux2[MAX+1][MAX+1];
  double AR[MAX+1][MAX+1], Fk[MAX+1][MAX+1];

  /* Inicializar arreglos */
  for (i=0; i <= MAX; i++)
   {
    for (j=0; j <= MAX; j++)
     {
       AC[i][j] = 0;
       BC[i][j] = 0;
       C[i][j]  = 0;
       AD[i][j] = 0;
       BD[i][j] = 0;
       ID[i][j] = 0;
       FI[i][j] = 0;
       Fk[i][j] = 0;
       AR[i][j] = 0;
       Aaux1[i][j] = 0;
       Aaux2[i][j] = 0;
     }
    NUMERD[i] = 0;
```

```
 DENOMD[i] = 0;
}

/* Matriz Identidad */
for (i=1; i <= n; i++)
 {
  for (j=1; j <= n; j++)
   {
    if (i == j)
     {
       ID[i][j] = 1;
       FI[i][j] = 1;
       Fk[i][j] = 1;
     }
   }
 }

/* Representación del sistema a variables de estado de la forma canónica
   controlable.
                  X' = AC * X + BC * U
                  Y  = C  * X
*/

/* Matriz A */
for (i=1; i <= n - 1; i++)
  AC[i][i+1] = 1;
for (j=1; j <= n; j++)
  AC[n][j] = -DENOMC[n+2-j] / DENOMC[1];

/* Matriz B */
BC[n][1] = 1    DENOMC[1];

 * Matriz C *
for (j=1; j <= n; j++)
  C[1][j] = NUMERC[m+2-j];

/* Inicio de la discretización para obtener:
                  X(k+1) = AD * X(k) + B * U(k)
                  Y(k)   = C  * X(k)
   Formulas:

 =>  AD = ID + AC * T_MUES * FI
     BD = FI * T_MUES * BC
     CD = C
*/
for (k=15; k > 1; k--)
 {
  MULTFAC( AC, T_MUES/k, Aaux1, n, n);
```

```
  MULTMAT( Aaux1, FI, AR, n, n, n, n);
  SUMMAT( ID, AR, AR, n, n, 0);
  COPIAM( AR, FI, n, n);
}

/* Obtención de la matriz AD */
MULTFAC( AC, T_MUES, Aaux1, n, n);
MULTMAT( Aaux1, FI, AR, n, n, n, n);
SUMMAT( ID, AR, AD, n, n, 0);

/* Obtención de la matriz BD */
MULTFAC( FI, T_MUES, Aaux1, n, n);
MULTMAT( Aaux1, BC, BD, n, n, n, 1);


/* Formar la función de transferencia discreta G(Z)
                          -1
  G(Z) = C * [Z*I - AD]    * BD

            -1
  [Z*I - AD]  =  Adj ( Z*I - AD) / |Z*I - AD|
                        -1      -2              -n
  Adj (Z*I - AD) = F1*Z + F2*Z   +.....+ Fn*Z
                     .    -1              -n
  |Z*I - AD| = 1 + Q1*Z   +.....+ Qn*Z

  F1 = I

  Fk = AD*Fk-1 + Qk-1*I

  Qk = - tr(AD*Fk) / k ;  tr = suma de los elementos de la diagonal.
*/

for (k=1; k <= n; k++)
 {
  MULTMAT( AD, Fk, AR, n, n, n, n);

  /* Suma de los elementos de la diagonal de la matriz AR */
  sum = 0;
  for (i=1; i <= n; i++)
   {
    for (j=1; j <= n; j++)
     {
      if (i == j)
        sum = sum + AR[i][j];
     }
   }
```

```
    Qk = -sum / k;
    DENOMD[k] = Qk;

    MULTMAT( Fk, BD, Aaux1, n, n, n, 1);
    MULTMAT( C, Aaux1, Aaux2, 1, n, n, 1);
    NUMERD[k] = Aaux2[1][1];

    MULTFAC( ID, Qk, Aaux1, n, n);
    SUMMAT( AR, Aaux1, Fk, n, n, 0);
  }
}
```

```
/* ------------------------------------------------------------------------
   Nombre          : BASE.C
   Tipo            : Módulo de CEXPER.C
   Descripción     : Contiene la Base de Conocimiento del sistema. Incorpora
                     la información del funcionamiento de la planta y los
                     controladores, representada en reglas de producción.
                     Incluye las funciones: EXPERTO1( ) y EXPERTO2( ).
   ------------------------------------------------------------------------
*/

#include "funcion.h"
#include <math.h>
#include <stdlib.h>

/* ........................................................................

   SISTEMA EXPERTO POR CONMUTACION:  EXPERTO1
   ------------------------------------------

   Función para seleccionar la mejor acción de control por conectar
   a la planta.
   ........................................................................
*/

double EXPERTO1(double e, double de, double Upid, double Udead, double Uself,
                double SP, int ik, int n, double Up)

{
  double Uexp1;

  /* 1. Reglas para un sistema de primer orden
     ..................................................
  */

  if ( n == 1)
   {

    /* 1.1 Reglas para cuando la referencia es positiva */
    if (Up > 0)

    {
     /* 1.1.1  Cambio del set point */
     if ((ik == 1) || ((ik == 120) && (e > 0.9*SP)))
      Uexp1 = Udead;

     /* 1.1.3  Error mayor al 10% del set point y su derivada negativa */
     if ((e > 0.1*SP) && (de < 0))
      Uexp1 = Uself;
```

```
/* 1.1.6  Error mayor al 10% del set point y su derivada positiva */
if ((ik > 1) && (ik != 120) && (e > 0.1*SP) && (de > 0))
 Uexp1 = max(Upid, Uself);

/* 1.1.2  Error mayor al 99% del set point y existe retardo */
if ((ik > 1) && (ik < 80) && (e > 0.99*SP) && (fabs(de) < 0.1))
 Uexp1 = Uself;

if ((ik > 120) && (e > 0.99*SP) && (fabs(de) < 0.1))
 Uexp1 = Uself;

/* 1.1.4  Valor absoluto del error menor o igual al 10% del set point */
if (fabs(e) <= 0.1*SP)
 Uexp1 = Uself;

/* 1.1.5  Error negativo y menor al 10% del set point */
if (e < -0.1*SP)
 Uexp1 = min(Upid, Uself);
}

/* 1.2  Reglas para cuando la referencia es cero */
if (Up == 0)

{
 /* 1.2.1  Cambio del set point */
 if (ik == 80)
  Uexp1 = Udead;

 /* 1.2.3  Error menor al -10% del set point y su derivada positiva.
           Reflejo de la regla 1.1.3, pero para referencia igual a cero */
 if ((e < -0.1*SP) && (de > 0))
  Uexp1 = Uself;

 /* 1.2.6  Error menor al -10% del set point y su derivada negativa.
           Reflejo de la regla 1.1.6, pero para referencia igual a cero */
 if ((ik > 80) && (e < -0.1*SP) && (de < 0))
  Uexp1 = min(Upid, Uself);

 /* 1.2.2  Error menor al -99% del set point y existe retardo */
 if ((ik > 80) && (e < -0.99*SP) && (fabs(de) < 0.1))
  Uexp1 = Uself;

 /* 1.2.4  Valor absoluto del error o igual al 10% del set point */
 if (fabs(e) <= 0.1*SP)
  Uexp1 = Uself;

 /* 1.2.5  Error positivo y mayor al 10% del set point.
           Reflejo de la regla 1.1.5, pero para referencia igual a cero */
```

```
  if (e > 0.1*SP)
   Uexp1 = max(Upid, Uself);
 }

}

/* 2.  Para sistemas de segundo orden y orden superior
   ..........................................................................
*/

else

{
 /* 2.1  Reglas para cuando la referencia es positiva */
 if (Up > 0)

 {
  /* 2.1.1  Cambio del set point */
  if ((ik == 1) || ((ik == 120) && (e > 0.9*SP)))
   Uexp1 = Udead;

  /* 2.1.3  Error mayor al 10% del set point y su derivada negativa */
  if ((e > 0.1*SP) && (de < 0))
   Uexp1 = Uself;

  /* 2.1.7  Error mayor al 10% del set point y su derivada positiva */
  if ((ik > 1) && (e > 0.1*SP) && (de > 0))
   Uexp1 = max(Upid, Uself);

  /* 2.1.2  Error mayor al 99% del set point y existe retardo */
  if ((ik > 1) && (ik < 80) && (e > 0.99*SP) && (fabs(de) < 0.1))
   Uexp1 = Uself;

  if ((ik > 120) && (e > 0.99*SP) && (fabs(de) < 0.1))
   Uexp1 = Uself;

  /* 2.1.4  Valor absoluto del error menor o igual al 10% del set point */
  if (fabs(e) <= 0.1*SP)
   {
    if (n > 2)
      Uexp1 = Uself;
    else
      Uexp1 = Upid;
   }

   * 2.1.6  Error negativo y menor al 10% del set point */
  if (e < -0.1*SP)
   Uexp1 = min(Upid, Uself);
```

```
}

/* 2.2 Reglas para cuando la referencia es cero */
if (Up == 0)
{
 /* 2.2.1 Cambio del set point */
 if ((ik == 80))
  Uexp1 = Udead;

 /* 2.2.3 Error menor al -10% del set point y su derivada positiva.
          Reflejo de la regla 2.1.3, pero para referencia igual a cero */
 if ((e < -0.1*SP) && (de > 0))
  Uexp1 = Uself;

 /* 2.2.7 Error menor al -10% del set point y su derivada negativa.
          Reflejo de la regla 2.1.7, pero para referencia igual a cero */
 if ((ik > 80) && (e < -0.1*SP) && (de < 0))
  Uexp1 = min(Upid, Uself);

 /* 2.2.2 Error menor al -99% del set point y existe retardo
          Reflejo de la regla 2.1.2, pero para referencia igual a cero */
 if ((e < -0.99*SP) && (fabs(de) < 0.1))
  Uexp1 = Uself;

 /* 2.2.4 Valor absoluto del error o igual al 10% del set point */
 if (fabs(e) <= 0.1*SP)
  {
   if (n > 2)
     Uexp1 = Uself;
   else
     Uexp1 = Upid;
  }

 /* 2.2.6 Error positivo y mayor al 10% del set point.
          Reflejo de la regla 2.1.6, pero para referencia igual a cero */
 if (e > 0.1*SP)
  Uexp1 = max(Upid, Uself);
}

return (Uexp1);
}
```

```
/* ..........................................................................

   SISTEMA EXPERTO POR PONDERACION : EXPERTO2

   Función para obtener una combinación lineal de todos los controladores.
   ..........................................................................
*/

double EXPERTO2(double e, double de, double Upid, double Udead, double Uself,
                double SP, int ik, int n, double Up)

{
  double Uexp2;

 /* 1.  Reglas para un sistema de primer orden
    ..........................................
 */

 if ( n == 1)
  {

   /* 1.1  Reglas para cuando la referencia es positiva */
   if (Up > 0)

   {
    /* 1.1.1  Cambio del set point */
    if ((ik == 1) || ((ik == 120) && (e > 0.9*SP)))
     Uexp2 = 0.7*Udead + 0.2*Uself + 0.1*Upid;

    /* 1.1.3  Error mayor al 10% del set point y su derivada negativa */
    if ((e > 0.1*SP) && (de < 0))
     Uexp2 = 0.9*Uself + 0.1*Udead;

    /* 1.1.6  Error mayor al 10% del set point y su derivada positiva */
    if ((ik > 1) && (ik != 120) && (e > 0.1*SP) && (de > 0))
     Uexp2 = 0.9*max(Upid,Uself) + 0.1*min(Upid,Uself);

    /* 1.1.2  Error mayor al 99% del set point y existe retardo */
    if ((ik > 1) && (ik < 80) && (e > 0.99*SP) && (fabs(de) < 0.1))
     Uexp2 = 0.8*Uself + 0.2*Udead;

    if ((ik > 120) && (e > 0.99*SP) && (fabs(de) < 0.1))
     Uexp2 = 0.8*Uself + 0.2*Udead;

    /* 1.1.4  Valor absoluto del error menor o igual al 10% del set point */
    if (fabs(e) <= 0.1*SP)
     Uexp2 = 0.8*Uself + 0.2*Upid;
```

```
 /* 1.1.5  Error negativo,y menor al 10% del set point */
 if (e < -0.1*SP)
  Uexp2 = 0.9*min(Upid,Uself) + 0.1*max(Upid,Uself);
}

/* 1.2  Reglas para cuando la referencia es cero */
if (Up == 0)

{
 /* 1.2.1  Cambio del set point */
 if (ik == 80)
  Uexp2 = 0.7*Udead + 0.2*Uself + 0.1*Upid;

 /* 1.2.3  Error menor al -10% del set point y derivada positiva.
           Reflejo de la regla 1.1.3, pero para referencia igual a cero */
 if ((e < -0.1*SP) && (de > 0))
  Uexp2 = 0.9*Uself + 0.1*Udead;

 /* 1.2.6  Error menor al -10% del set point y derivada negativa.
           Reflejo de la regla 1.1.6, pero para referencia igual a cero */
 if ((ik > 80) && (e < -0.1*SP) && (de < 0))
  Uexp2 = 0.9*min(Upid,Uself) + 0.1*max(Upid,Uself);

 /* 1.2.2  Error menor al -99% del set point y existe retardo */
 if ((e < -0.99*SP) && (fabs(de) < 0.1))
  Uexp2 = 0.8*Uself + 0.2*Udead;

 /* 1.2.4  Valor absoluto del error o igual al 10% del set point */
 if (fabs(e) <= 0.1*SP)
  Uexp2 = 0.8*Uself + 0.2*Upid;

 /* 1.2.5  Error positivo y mayor al 10% del set point.
           Reflejo de la regla 1.1.5, pero para referencia igual a cero */
 if (e > 0.1*SP)
  Uexp2 = 0.9*max(Upid,Uself) + 0.1*min(Upid,Uself);
}

}

/* 2.  Para sistemas de segundo orden y orden superior
   ..........
*/

else

{

 /* 2.1  Reglas para cuando la referencia es positiva */
```

```
if (Up > 0)

{
 /* 2.1.1  Cambio del set point */
 if ((ik == 1 ) || ((ik == 120) && (e > 0.9*SP)))
  Uexp2 = 0.7*Udead + 0.2*Uself + 0.1*Upid;

 /* 2.1.3  Error mayor al 10% del set point y la derivada es negativa */
 if ((e > 0.1*SP) && (de < 0))
  Uexp2 = 0.8*Uself + 0.2*Upid;

 /* 2.1.7  Error mayor al 10% del set point y la derivada es positiva */
 if ((ik > 1) && (e > 0.1*SP) && (de > 0))
  Uexp2 = 0.9*max(Uself,Upid) + 0.1*min(Uself,Upid);

 /* 2.1.2  Error mayor al 99% del set point y existe retardo */
 if ((ik > 1) && (ik < 80) && (e > 0.99*SP) && (fabs(de) < 0.1))
  Uexp2 = 0.8*Uself + 0.2*Upid;

 if ((ik > 120) && (e > 0.99*SP) && (fabs(de) < 0.1))
  Uexp2 = 0.8*Uself + 0.2*Upid;

 /* 2.1.4  Valor absoluto del error menor o igual al 10% del set point */
 if (fabs(e) <= 0.1*SP)
 {
  if (n > 2)
      Uexp2 = 0.9*Uself + 0.1*Upid;
  else
      Uexp2 = 0.9*Upid + 0.1*Uself;
 }

 /* 2.1.6  Error negativo y menor al 10% del set point */
 if (e < -0.1*SP)
  Uexp2 = 0.9*min(Uself,Upid) + 0.1*max(Uself,Upid);
}

/* 2.2  Reglas para cuando la referencia es cero */
if (Up == 0)

{
 /* 2.2.1  Cambio del set point */
 if (ik == 80)
  Uexp2 = 0.7*Udead + 0.2*Uself + 0.1*Upid;

 /* 2.2.3  Error menor al -10% del set point y derivada positiva.
           Reflejo de la regla 2.1.3, pero para referencia igual a cero */
 if ((e < -0.1*SP) && (de > 0))
  Uexp2 = 0.8*Uself + 0.2*Upid;
```

```
/* 2.2.7  Error menor al -10% del set point y derivada negativa.
          Reflejo de la regla 2.1.7, pero para referencia igual a cero *
if ((ik > 80) && (e < -0.1*SP) && (de < 0))
 Uexp2 = 0.9*min(Uself,Upid) + 0.1*max(Uself,Upid);

/* 2.2.2  Error menor al -99% del set point y existe retardo
          Reflejo de la regla 2.1.2, pero para referencia igual a cero *.
if ((e < -0.99*SP) && (fabs(de) < 0.1))
 Uexp2 = 0.8*Uself + 0.2*Upid;

/* 2.2.4  Valor absoluto del error o igual al 10% del set point */
if (fabs(e) <= 0.1*SP)
{
 if (n > 2)
    Uexp2 = 0.9*Uself + 0.1*Upid;
 else
    Uexp2 = 0.9*Upid + 0.1*Uself;
}

/* 2.2.6  Error positivo y mayor al 10% del set point.
          Reflejo de la regla 2.1.6, pero para referencia igual a cero *
if (e > 0.1*SP)
 Uexp2 = 0.9*max(Uself,Upid) + 0.1*min(Uself,Upid);
}

}

return (Uexp2);
}
```

```
/* ---------------------------------------------------------
   Nombre:          MENU.C
   Tipo:            Módulo Toolbox de CEXPER.C
   Descripción:     Genera menúes en pantalla.
   ---------------------------------------------------------
*/

#include <graph.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <malloc.h>
#include "box.h"
#include "mousefun.h"
#include "getkey.h"
#include "t_colors.h"
#include "menu.h"

/* Default menu colors */
static int c_lines = T_CYAN;
static int c_title = T_CYAN;
static int c_text = T_WHITE;
static int c_prompt = T_RED;
static int c_hitext = T_BLACK;
static int c_hiletter = T_RED;
static long int c_back = BK_BLUE;
static long int c_hiback = BK_WHITE;

/* Default border lines and shadow control */
static int mb_lines = 2;
static int mb_shadow = 1;


/* ---------------------------------------------------------
   Function:        menu_box_lines()
   Toolbox:         MENU.C
   Demonstrated:    MENUTEST.C

   Parameters:
     (input)        line_type          0, 1, or 2 (outline)

   Returned:        (function returns nothing)

   Variables:       (none)

   Description:     Sets the box outline type.  Selects single-line or
                    double-line border (or none)
   ---------------------------------------------------------
```

```
*/

void menu_box_lines( int line_type )
{
    mb_lines = line_type;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       menu_box_shadow()
   Toolbox:        MENU.C
   Demonstrated:   MENUTEST.C

   Parameters:
     (input)       on_off          Shadow control

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the menu box shadow control to on or off.
                   0 = off, non-zero = on
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

void menu_box_shadow( int on_off )
{
    mb_shadow = on_off;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       menu_back_color()
   Toolbox:        MENU.C
   Demonstrated:   MENUTEST.C

   Parameters:
     (input)       back            Background color

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the background color for boxes
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

void menu_back_color( long back )
```

```
{
   c_back = back;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       menu_line_color()
   Toolbox:        MENU.C
   Demonstrated:   MENUTEST.C

   Parameters:
    (input)        lines          Border line color

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the box outline color
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

void menu_line_color( int lines )
{
   c_lines = lines;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       menu_title_color()
   Toolbox:        MENU.C
   Demonstrated:   MENUTEST.C

   Parameters:
    (input)        title          Title text color

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the text color for the title
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

void menu_title_color( int title )
{
   c_title = title;
}
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:      menu_text_color()
   Toolbox:       MENU.C
   Demonstrated:  MENUTEST.C

   Parameters:
     (input)      text            Menu text color

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Sets the menu box text color
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

void menu_text_color( int text )
{
    c_text = text;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:      menu_prompt_color()
   Toolbox:       MENU.C
   Demonstrated:  MENUTEST.C

   Parameters:
     (input)      prompt          Menu prompt line color

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Sets the menu box prompt line text color
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

void menu_prompt_color( int prompt )
{
    c_prompt = prompt;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:      menu_hilight_letter()
   Toolbox:       MENU.C
   Demonstrated:  MENUTEST.C
```

```
   Parameters:
     (input)      hiletter          Highlighted letter color

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Sets highlighted character color for menu options
   ---------------------------------------------
*/

void menu_hilight_letter( int hiletter )
{
    c_hiletter = hiletter;
}


/* ---------------------------------------------
   Function:      menu_hilight_text()
   Toolbox:       MENU.C
   Demonstrated:  MENUTEST.C

   Parameters:
     (input)      hitext            Highlighted text color

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Sets highlighted text color for menu options
   ---------------------------------------------
*/

void menu_hilight_text( int hitext )
{
    c_hitext = hitext;
}


/* ---------------------------------------------
   Function:      menu_hilight_back()
   Toolbox:       MENU.C
   Demonstrated:  MENUTEST.C

   Parameters:
     (input)      hiback            Highlighted line background

   Returned:      (function returns nothing)
```

```
   Variables:      (none)

   Description:    Sets the background color for the highlighted line
                   in the menu box
   ---------------------------------------------------------
*/

void menu_hilight_back( long hiback )
{
   c_hiback = hiback;
}


/* ---------------------------------------------------------
   Function:       menu_bar()
   Toolbox:        MENU.C
   Demonstrated:   MENUTEST.C

   Parameters:
     (input)       row       Screen row to locate menu bar
     (input)       col       Screen column to locate menu bar
     (input)       string    String of menu bar selections
     (output)      choice    Number of item selected by user

   Returned:       Buffer used to restore the background

   Variables:      len           Length of menu string
                   fore          Saves current foreground color
                   maxchoice     Number of choices
                   i             Looping index
                   j             Looping index
                   cpos          Current position in the menu
                   quit_flag     Signals to exit function
                   savebuf       Buffer containing background
                   fstr          Foreground color attributes
                   laste         Last character checked
                   thise         Current character checked
                   bstr          Background color attributes
                   key           Key code from getkey_or_mouse()
                   back          Saves current background color
                   oldpos        Saves the cursor position

   Description:    Creates a pop-up menu bar
   ---------------------------------------------------------
*/

int far *menu_bar ( int row, int col, char *string, int *choice )
{
```

```
{

/* Determine the color attributes */
j = 0;
maxchoice = 0;
lastc = 0;
for ( i = 0; i < len; i++ )
    {
    thisc = string[i];
    if ( lastc == ' ' && thisc == ' ' && i < len - 1 )
        {
        j++;
        maxchoice++;
        }
    if ( j == *choice && i < len - 1 )
        {
        fstr[i] = c_hitext;
        bstr[i] = c_hiback;
        }
    else
        {
        fstr[i] = c_text;
        bstr[i] = c_back;
        }
    if ( isupper( thisc ))
        {
        fstr[i] = c_hiletter;
        if ( j == *choice )
            cpos = i;
        }
    lastc = thisc;
    }

 * Put the attributes to video */
for ( i = 0; i < len; i++ )
    {
    _settextcolor( fstr[i] );
    _setbkcolor( bstr[i] );
    box_color( row, col + i, row, col + i );
    }

 * Put cursor at appropriate position */
_settextposition( row, col + cpos );

key = getkey_or_mouse();

 * Convert to upper case */
if ( key >= 'a' && key <= 'z' )
```

```
        key -= 32;

    /* Check for alpha key */
    if ( key >= 'A' && key <= 'Z' )
        {
        for ( i = 0; i < len; i++ )
            {
            if ( ++cpos >= len )
                {
                cpos = 0;
                *choice = 0;
                }
            if ( isupper( string[cpos] ))
                *choice += 1;
            if ( string[cpos] == (char)key )
                break;
            }
        }

    /* Check for control keys */
    switch( key )
        {
        case KEY_LEFT:
            if ( *choice > 1 )
                *choice -= 1;
            break;
        case KEY_RIGHT:
            if ( *choice < maxchoice )
                *choice += 1;
            break;
        case KEY_HOME:
            *choice = 1;
            break;
        case KEY_END:
            *choice = maxchoice;
            break;
        case KEY_ESCAPE:
        case KEY_UP:
            *choice = 0;
            quit_flag = 1;
            break;
        case KEY_ENTER:
        case KEY_DOWN:
            quit_flag = 1;
            break;
        }
    }
```

```
    /* Restore original conditions */
    _settextposition( oldpos.row, oldpos.col );
    _settextcolor( fore );
    _setbkcolor( back );
    return ( savebuf );
}

/* ------------------------------------------------
   Function:       menu_drop()
   Toolbox:        MENU.C
   Demonstrated:   MENUTEST.C

   Parameters:
    (input)     row       Screen row to locate menu bar
    (input)     col       Screen column to locate menu bar
    (input)     strary    String array of menu selections
    (output)    choice    Number of item selected by user

   Returned:       Buffer used to restore the background

   Variables:      n             Number of strings in menu
                   len           Length of menu string
                   fore          Saves current foreground color
                   tmpcol        Column to start title and prompt
                   maxchoice     Number of choices
                   i             Looping index
                   quit_flag     Signals to exit function
                   savebuf       Buffer containing background
                   key           Key code from getkey_or_mouse()
                   back          Saves current background color
                   oldpos        Saves the cursor position

   Description:    Creates a popup drop down menu
   ------------------------------------------------
*/

int far *menu_drop( int row, int col, char **strary, int *choice )
{
    int n = 0;
    int len = 0;
    int fore;
    int tmpcol;
    int maxchoice;
    int i;
    int quit_flag = 0;
    int far *savebuf;
    unsigned key;
    long int back;
```

```
struct rccoord oldpos;

/* Save the current color settings */
fore = _gettextcolor();
back = _getbkcolor();

/* Save the current cursor position */
oldpos = _gettextposition();

/* Determine the number of strings in the menu */
while ( strary[n] != NULL )
    n++;

/* Set the maximum choice number */
maxchoice = n - 2;

/* Determine the maximum menu string length */
for ( i = 0; i < n; i++ )
    if ( strlen( strary[i] ) > len )
        len = strlen( strary[i] );

/* Save the menu background */
if ( mb_shadow )
    savebuf = box_get( row, col, row + n, col + len + 5 );
else
    savebuf = box_get( row, col, row + n - 1, col + len + 3 );

/* Create the menu box */
_settextcolor( c_lines );
_setbkcolor( c_back );
box_erase( row, col, row + n - 1, col + len + 3 );
box_draw( row, col, row + n - 1, col + len + 3, mb_lines );

/* Cast a shadow */
if ( mb_shadow )
    {
    _settextcolor( T_GRAY );
    _setbkcolor( BK_BLACK );
    box_color( row + n, col + 2, row + n, col + len + 3 );
    box_color( row + 1, col + len + 1, row + n, col + len + 5 );
    }

/* Put the title at the top */
tmpcol = col + ( len - strlen( strary[0] ) + 4 ) / 2;
_settextposition( row, tmpcol );
_settextcolor( c_title );
_setbkcolor( c_back );
_outtext( strary[0] );
```

```
/* Print the choices */
_settextcolor( c_text );
for ( i = 1; i <= maxchoice; i++ )
    {
    _settextposition( row + i, col + 2 );
    _outtext( strary[i] );
    }

/* Put the prompt at the bottom */
tmpcol = col + ( len - strlen( strary[n - 1] ) + 4 ) / 2;
_settextposition( row + n - 1, tmpcol );
_settextcolor( c_prompt );
_outtext( strary[n - 1] );

/* Initialize choice */
*choice = 1;

/* Process each key press */
while ( !quit_flag )
    {

    /* Determine and set the color attributes */
    for ( i = 1; i <= maxchoice; i++ )
        {
        if ( i == *choice )
            {
            _setbkcolor( c_hiback );
            _settextcolor( c_hiletter );
            box_color( row + i, col + 1, row + i, col + 2 );
            _settextcolor( c_hitext );
            box_color( row + i, col + 3, row + i, col + len + 2 );
            }
        else
            {
            _setbkcolor( c_back );
            _settextcolor( c_hiletter );
            box_color( row + i, col + 1, row + i, col + 2 );
            _settextcolor( c_text );
            box_color( row + i, col + 3, row + i, col + len + 2 );
            }
        }

    /* Put cursor at appropriate position */
    _settextposition( row + *choice, col + 2 );

    key = getkey_or_mouse();

    /* Convert to upper case */
```

```
    if ( key >= 'a' && key <= 'z' )
        key -= 32;

    /* Check for alpha key */
    if ( key >= 'A' && key <= 'Z' )
        {
        for ( i = 1; i <= maxchoice; i++ )
            {
            *choice += 1;
            if ( *choice > maxchoice )
                *choice = 1;
            if ( strary[*choice][0] == (char)key )
                break;
            }
        }

    /* Check for control keys */
    switch ( key )
        {
        case KEY_UP:
            if ( *choice > 1 )
                *choice -= 1;
            break;
        case KEY_DOWN:
            if ( *choice < maxchoice )
                *choice += 1;
            break;
        case KEY_HOME:
            *choice = 1;
            break;
        case KEY_END:
            *choice = maxchoice;
            break;
        case KEY_ESCAPE:
            *choice = 0;
            quit_flag = 1;
            break;
        case KEY_ENTER:
            quit_flag = 1;
            break;
        }
    }

/* Restore original conditions */
_settextposition( oldpos.row, oldpos.col );
_settextcolor( fore );
_setbkcolor( back );
return ( savebuf );
```

```
}

/*---------------------------------------------------------
   Function:      menu_message()
   Toolbox:       MENU.C
   Demonstrated:  MENUTEST.C

   Parameters:
     (input)      row       Screen row to locate message box
     (input)      col       Screen column to locate message box
     (input)      strary    String array of message text

   Returned:      Buffer used to restore the background

   Variables:     n         Number of strings in message
                  len       Length of longest menu string
                  fore      Saves current foreground color
                  tmpcol    Column to start title and prompt
                  i         Looping index
                  savebuf   Buffer containing background
                  key       Key code from getkey_or_mouse()
                  back      Saves current background color
                  oldpos    Saves the cursor position

   Description:   Creates a pop-up message box
   ---------------------------------------------------------
*/

int far *menu_message( int row, int col, char **strary )
{
    int n = 0;
    int len = 0;
    int fore;
    int tmpcol;
    int i;
    int far *savebuf;
    unsigned key;
    long int back;
    struct rccoord oldpos;

    /* Save the current color settings */
    fore = _gettextcolor();
    back = _getbkcolor();

    /* Save the current cursor position */
    oldpos = _gettextposition();
```

```
/* Determine the number of strings in the message */
while ( strary[n] != NULL )
    n++;

/* Determine the maximum message string length */
for ( i = 0; i < n; i++ )
    if ( strlen( strary[i] ) > len )
        len = strlen( strary[i] );

/* Save the message background */
if ( mb_shadow )
    savebuf = box_get( row, col, row + n, col + len + 5 );
else
    savebuf = box_get( row, col, row + n - 1, col + len + 3 );

/* Create the information box */
_settextcolor( c_lines );
_setbkcolor( c_back );
box_erase( row, col, row + n - 1, col + len + 3 );
box_draw( row, col, row + n - 1, col + len + 3, mb_lines );

/* Cast a shadow */
if ( mb_shadow )
    {
    _settextcolor( T_GRAY );
    _setbkcolor( BK_BLACK );
    box_color( row + n, col + 2, row + n, col + len + 3 );
    box_color( row + 1, col + len + 4, row + n, col + len + 5 );
    }

/* Put the title at the top */
tmpcol = col + ( len - strlen( strary[0] ) + 4 ) / 2;
_settextposition( row, tmpcol );
_settextcolor( c_title );
_setbkcolor( c_back );
_outtext( strary[0] );

/* Print the text */
_settextcolor( c_text );
for ( i = 1; i < n - 1; i++ )
    {
    _settextposition( row + i, col + 2 );
    _outtext( strary[i] );
    }

/* Put the prompt at the bottom */
tmpcol = col + ( len - strlen( strary[n - 1] ) + 4 ) / 2;
_settextposition( row + n - 1, tmpcol );
```

```
    _settextcolor( c_prompt );
    _outtext( strary[n - 1] );

    /* Restore original conditions */
    _settextposition( oldpos.row, oldpos.col );
    _settextcolor( fore );
    _setbkcolor( back );
    return ( savebuf );
}

/*---------------------------------------------------
   Function:       menu_erase()
   Toolbox:        MENU.C
   Demonstrated:   MENUTEST.C

   Parameters:
     (input)       buf       Buffer for restoring background

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Restores the background behind a bar menu,
                   pull-down menu, or message box
   ---------------------------------------------------
*/

void menu_erase( int far *buf )
{
    box_put( buf );
    _ffree( buf );
}
```

```
/*-------------------------------------------------------------------
   Nombre:          MENU.H
   Tipo:            Include
   Descripción:     Prototipos y definiciones del módulo MENU.C
-------------------------------------------------------------------
*/

#ifndef MENU_DEFINED

void menu_box_lines( int );
void menu_box_shadow( int );
void menu_back_color( long int );
void menu_line_color( int );
void menu_title_color( int );
void menu_text_color( int );
void menu_prompt_color( int );
void menu_hilight_letter( int );
void menu_hilight_text( int );
void menu_hilight_back( long int );
int far *menu_bar( int, int, char *, int * );
int far *menu_drop( int, int, char **, int * );
int far *menu_message( int, int, char ** );
void menu_erase( int far * );

#define MENU_DEFINED
#endif
```

```
/* ---------------------------------------------------------
   Nombre:        BOX.C
   Tipo:          Módulo Toolbox de CEXPER.C
   Descripción:   Permite generar cuadros en pantalla.
   ---------------------------------------------------------
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <dos.h>
#include <string.h>
#include <graph.h>
#include "box.h"

static void determine_video ( void );
static unsigned video_seg = 0;
static char far *videoptr;
static int columns;

/* ---------------------------------------------------------------------
   Function:      box_get()
   Toolbox:       BOX.C
   Demonstrated:  BOXTEST.C MENU.C

   Parameters:
     (input)      row1     Upper left corner of box
     (input)      col1     Upper left corner of box
     (input)      row2     Lower right corner of box
     (input)      col2     Lower right corner of box

   Returned:      Address of far integer buffer containing data
                  saved from the rectangular area of screen

   Variables:     i        Looping index for lines in box
                  width    Width of box area
                  height   Height of box area
                  bytes    Total number bytes to store box data
                  buf      Address of far buffer for storage
                  bufptr   Index into storage buffer memory
                  video_off Offset of video address for box data

   Description:   Saves contents of a rectangular area of the
                  screen in a dynamically allocated buffer
   ---------------------------------------------------------------------
*/

unsigned far *box_get( unsigned row1, unsigned col1,
```

```
                    unsigned row2, unsigned col2 )
{
    unsigned i, width, height, bytes;
    unsigned far *buf, far *bufptr;
    unsigned video_off;

    /* Calculate the dimensions in bytes */
    width = ( col2 - col1 + 1 ) * 2;
    height = row2 - row1 + 1;
    bytes = height * width + 8;

    /* Allocate storage space */
    if (( buf = (unsigned far *)malloc((size_t)bytes )) == NULL)
        {
        printf( "box_get(): malloc() failed\n" );
        exit( 0 );
        }

    /* Save the box coordinates in the buffer */
    bufptr = buf;
    *bufptr++ = row1;
    *bufptr++ = col1;
    *bufptr++ = row2;
    *bufptr++ = col2;

    /* Determine the text mode video segment and number of columns */
    determine_video();

    /* Calculate starting location in video memory */
    video_off = (unsigned)(( columns * ( row1 - 1 ) +
                ( col1 - 1 ))* 2 );

    /* Grab each line of the video */
    for ( i = 0; i < height; i++ )
        {
        movedata( video_seg, video_off,
                FP_SEG( bufptr ), FP_OFF( bufptr ), width );
        bufptr += width / 2;
        video_off += columns * 2;
        }

    /* Return the buffer */
    return ( buf );
}

/* ---------- ---------------------------------------------------------------
 Function:      box_put()
```

```
   Toolbox:       BOX.C
   Demonstrated:  BOXTEST.C MENU.C

   Parameters:
     (input)      buf       Far integer buffer previously created
                            by the function box_get()

   Returned:      (function returns nothing)

   Variables:     row1      Upper left corner of box
                  col1      Upper left corner of box
                  row2      Lower right corner of box
                  col2      Lower right corner of box
                  i         Loop index for each line of the box
                  width     Width of the box
                  height    Height of the box
                  bytes     Total number of bytes in the box
                  video_off Offset of video address for box data
                  workbuf   Index into the buffer

   Description:   Restores screen contents that were saved in a
                  buffer by a previous call to box_get()
   ------------------------------------------------------------------------
*/

void box_put( unsigned far * buf )
{
    unsigned row1, col1, row2, col2;
    unsigned i, width, height, bytes;
    unsigned video_off;
    unsigned far *workbuf;

    /* Get the box coordinates */
    workbuf = buf;
    row1 = *workbuf++;
    col1 = *workbuf++;
    row2 = *workbuf++;
    col2 = *workbuf++;

    /* Calculate the dimensions in bytes */
    width = ( col2 - col1 + 1 ) * 2;
    height = row2 - row1 + 1;
    bytes = height * width;

    /* Determine the text mode video segment and number of columns */
    determine_video();

    /* Calculate starting location in video memory */
```

```
    video_off = ( columns * ( row1 - 1 ) + ( col1 - 1 )) * 2;

    /* Put each line out to video */
    for ( i = 0; i < height; i++ )
        {
        movedata( FP_SEG( workbuf ), FP_OFF( workbuf ),
                video_seg, video_off, width );
        workbuf += width / 2;
        video_off += columns * 2;
        }
}


/* ---------------------------------------------------------------------------
   Function:       box_color()
   Toolbox:        BOX.C
   Demonstrated:   BOXTEST.C MENU.C

   Parameters:
     (input)       row1      Upper left corner of box
     (input)       col1      Upper left corner of box
     (input)       row2      Lower right corner of box
     (input)       col2      Lower right corner of box

   Returned:       (function returns nothing)

   Variables:      x         Looping index for each row of box
                   y         Looping index for each column of box
                   fore      Current foreground text color
                   back      Current background text color
                   attr      Attribute byte combining fore and back

   Description:    Sets the foreground and background colors for
                   all characters in a box to the current colors.
                   Characters in the box are unaffected
   ---------------------------------------------------------------------------
*/

void box_color( unsigned row1, unsigned col1,
                unsigned row2, unsigned col2 )
{
    unsigned x, y;
    unsigned fore;
    unsigned long back;
    unsigned char attr;

    /* Determine the text mode video segment and number of columns */
    determine_video();
```

```
    /* Build the attribute byte */
    fore = _gettextcolor();
    back = _getbkcolor();
    attr = (unsigned char)(( fore & 0xF ) |
          ((( fore & 0x10 ) >> 1 ) | back ) << 4 ));

    /* Work through the box */
    for ( x = row1 - 1; x < row2; x++ )
        for ( y = col1 - 1; y < col2; y++ )
            *( videoptr + ( columns * x + y ) * 2 + 1 ) = attr;
}


/* ---------------------------------------------------------------------------
   Function:       box_charfill()
   Toolbox:        BOX.C
   Demonstrated:   BOXTEST.C MENUTEST.C

   Parameters:
     (input)       row1      Upper left corner of box
     (input)       col1      Upper left corner of box
     (input)       row2      Lower right corner of box
     (input)       col2      Lower right corner of box
     (input)       c         Character used to fill the box

   Returned:       (function returns nothing)

   Variables:      x         Looping index for each row of box
                   y         Looping index for each column of box

   Description:    Fills a rectangular area of the screen with a
                   character.  Attributes are unaffected
   ---------------------------------------------------------------------------
*/

void box_charfill ( unsigned row1, unsigned col1,
                    unsigned row2, unsigned col2, char c )
{
    unsigned x, y;

    /* Determine the text mode video segment and number of columns */
    determine_video();

    /* Work through the box */
    for ( x = row1 - 1; x < row2; x++ )
        for ( y = col1 - 1; y < col2; y++ )
            *( videoptr + ( columns * x + y ) * 2 ) = c;
}
```

```
/* ---------------------------------------------------------------------
   Function:       box_draw()
   Toolbox:        BOX.C
   Demonstrated:   BOXTEST.C MENU.C

   Parameters:
     (input)       row1      upper left corner of box
     (input)       col1      upper left corner of box
     (input)       row2      lower right corner of box
     (input)       col2      lower right corner of box
     (input)       line_type Indicates single-line or double-
                             line box border (or none)

   Returned:       (function returns nothing)

   Variables:      x         Keeps track of horizontal position
                   y         Keeps track of vertical position
                   dx        Horizontal motion increment
                   dy        Vertical motion increment
                   c         Character for each part of the border

   Description:    Draws a single-line or double-line box border
                   around a box.  Does not affect attributes
   ---------------------------------------------------------------------
*/

void box_draw( unsigned row1, unsigned col1,
               unsigned row2, unsigned col2, unsigned line_type )
{
   unsigned x, y, dx, dy;
   unsigned c;

   /* Determine the text mode video segment and number of columns */
   determine_video();

   /* Work around the box */
   x = col1;
   y = row1;
   dx = 1;
   dy = 0;
   do
      {

      /* Set the default character for unbordered boxes */
      c = ' ';

      /* Set the single-line drawing character */
```

```
    if ( line_type == 1 )
        if ( dx )
            c = 196;
        else
            c = 179;

    /* Set the double-line drawing character */
    else if ( line_type == 2 )
        if ( dx )
            c = 205;
        else
            c = 186;

    /* Change direction at top right corner */
    if ( dx == 1 && x == col2 )
        {
        dx = 0;
        dy = 1;
        if ( line_type == 1 )
            c = 191;
        else if ( line_type == 2 )
            c = 187;
        }

    /* Change direction at bottom right corner */
    if ( dy == 1 && y == row2 )
        {
        dx = -1;
        dy = 0;
        if ( line_type == 1 )
            c = 217;
        else if ( line_type == 2 )
            c = 188;
        }

    /* Change direction at bottom left corner */
    if ( dx == -1 && x == col1 )
        {
        dx = 0;
        dy = -1;
        if ( line_type == 1 )
            c = 192;
        else if ( line_type == 2 )
            c = 200;
        }

    /* Check for top left corner */
    if ( dy == -1 && y == row1 )
```

```
        {
        if ( line_type == 1 )
            c = 218;
        else if ( line_type == 2 )
            c = 201;
        }

      /* Put new character to video */
      *( videoptr + ( columns * ( y - 1 ) + ( x - 1 )) * 2 ) = (char)c;

      /* Move to next position */
      x += dx;
      y += dy;

      }
   while ( dy != -1 || y >= row1 );
}


/* ---------------------------------------------------------------------------
   Function:       box_erase()
   Toolbox:        BOX.C
   Demonstrated:   BOXTEST.C MENU.C

   Parameters:
     (input)       row1      Upper left corner of box
     (input)       col1      Upper left corner of box
     (input)       row2      Lower right corner of box
     (input)       col2      Lower right corner of box

   Returned:       (function returns nothing)

   Variables:      i         Looping index for each row of the box
                   buf       String of spaces for each row

   Description:    Fills a box with spaces.  Uses the current color
                   attributes
   ---------------------------------------------------------------------------
*/

void box_erase( unsigned row1, unsigned col1,
                unsigned row2, unsigned col2 )
{
   unsigned i;
   char buf[81];

   /* Fill the buffer with spaces */
   sprintf( buf, "%*s", col2 - col1 + 1, "" );
```

```
    /* Put each line out to video */
    for ( i = row1; i <= row2; i++ )
        {
        _settextposition( i, col1 );
        _outtext( buf );
        }
}

/* ---------------------------------------------------------------------------
   Function:       determine_video()
   Note:           STATIC FUNCTION AVAILABLE ONLY TO THIS MODULE
   Language:       Microsoft QuickC
   Toolbox:        BOX.C

   Parameters:     (none)

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Determines the text mode video segment and the
                   number of character columns currently set.
                   Fills in static variables that are
                   available only to the functions in this module.
   ---------------------------------------------------------------------------
*/

static void determine_video( void )
{
    if ( !video_seg )
        {
        /* Determine the text mode video segment */
        switch ( *((char far *)0x449) )
            {
            case 0:
            case 1:
            case 2:
            case 3:
                video_seg = 0xB800;
                videoptr = (char far *)0xB8000000;
                break;
            case 7:
                video_seg = 0xB000;
                videoptr = (char far *)0xB0000000;
                break;
            default:
                printf( "BOX.C: not in text mode\n" );
```

```
        exit( 0 );
    }
    /* Determine number of columns for current text mode */
    columns = *( (int far *)0x44A );
  }
}
```

```
/* ---------------------------------------------------------------
   Nombre:          BOX.H
   Tipo:            Include
   Descripción:     Definiciones del módulo BOX.C
   ---------------------------------------------------------------
*/

#ifndef BOX_DEFINED

unsigned far *box_get( unsigned, unsigned, unsigned, unsigned );
void box_put( unsigned far * );
void box_color( unsigned, unsigned, unsigned, unsigned );
void box_charfill( unsigned, unsigned, unsigned, unsigned, char );
void box_draw( unsigned, unsigned, unsigned, unsigned, unsigned );
void box_erase( unsigned, unsigned, unsigned, unsigned );

#define BOX_DEFINED
#endif
```

```
/* ---------------------------------------------------------
   Nombre:          MOUSEFUN.C
   Tipo:            Módulo Toolbox de CEXPER.C
   Descripción:     Permite realizar el manejo del mouse.
   ---------------------------------------------------------
*/

#include <dos.h>
#include "mousefun.h"

/* ---------------------------------------------------------
   Function:        mouse_reset()
   Toolbox:         MOUSEFUN.C
   Demonstrated:    MOUSTEST.C

   Parameters:
     (output)       status    Status of the mouse
     (output)       buttons   Number of mouse buttons

   Returned:        (function returns nothing)

   Variables:       m1        Local variable for register ax
                    m2        Local variable for register bx

   Description:     Resets the mouse and verifies its existence
   ---------------------------------------------------------
*/

void mouse_reset( int *status, int *buttons )
{
    int m1, m2;

    _asm
       {
       xor   ax, ax
       int   33h
       mov   m1, ax
       mov   m2, bx
       }

    *status = m1;
    *buttons = m2;
}


/* ---------------------------------------------------------
   Function:        mouse_show()
   Toolbox:         MOUSEFUN.C
```

```
   Demonstrated:   MOUSTEST.C

   Parameters:     (none)

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Makes the mouse cursor visible
   ---------------------------------------------------
*/

void mouse_show( void )
{
    _asm
      {
      mov   ax, 1
      int   33h
      }
}


/*-----------------------------------------------------
   Function:       mouse_hide()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:     (none)

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Makes the mouse cursor invisible
   ---------------------------------------------------
*/

void mouse_hide( void )
{
    _asm
      {
      mov   ax, 2
      int   33h
      }
}

/*-----------------------------------------------------
```

```
   Function:       mouse_status().
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (output)      left_button    State of the left button
     (output)      right_button   State of the right button
     (output)      horz_pos       Horizontal position of the mouse
     (output)      vert_pos       Vertical position of the mouse

   Returned:       (function returns nothing)

   Variables:      m2        Local variable for register bx
                   m3        Local variable for register cx
                   m4        Local variable for register dx

   Description:    Gets the current state of the mouse buttons and
                   the mouse cursor position
   ---------------------------------------------------------
*/

void mouse_status( int *left_button, int *right_button,
                   int *horz_pos, int *vert_pos )
{
   int m2, m3, m4;

   _asm
      {
      mov   ax, 3
      int   33h
      mov   m2, bx
      mov   m3, cx
      mov   m4, dx
      }

   *left_button = m2 & 1;
   *right_button = ( m2 >> 1 ) & 1;
   *horz_pos = m3;
   *vert_pos = m4;
}


/* ---------------------------------------------------------
   Function:       mouse_setpos()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
```

```
   (input)     horizontal    Horizontal position
   (input)     vertical      Vertical position

  Returned:    (function returns nothing)

  Variables:   (none)

  Description: Sets the mouse cursor to the indicated position
  ------------------------------------------------------------
*/

void mouse_setpos( int horizontal, int vertical )
{
   _asm
      {
      mov   ax, 4
      mov   cx, horizontal
      mov   dx, vertical
      int   33h
      }
}


/* ------------------------------------------------------------
   Function:     mouse_press()
   Toolbox:      MOUSEFUN.C
   Demonstrated: MOUSTEST.C

   Parameters:
    (input)      button    Left or right button
    (output)     status    Status of the button
    (output)     presses   Number of button presses
    (output)     horz_pos  Horizontal position at last press
    (output)     vert_pos  Vertical position at last press

   Returned:     (function returns nothing)

   Variables:    m1        Local variable for register ax
                 m2        Local variable for register bx
                 m3        Local variable for register cx
                 m4        Local variable for register dx

   Description:  Gets button press information
   ------------------------------------------------------------
*/

void mouse_press( int button, int *status, int *presses,
                  int *horz_pos, int *vert_pos )
```

```
{
   int m1, m2, m3, m4;

   _asm
      {
      mov   ax, 5
      mov   bx, button
      int   33h
      mov   m1, ax
      mov   m2, bx
      mov   m3, cx
      mov   m4, dx
      }

   if ( button == LBUTTON )
      *status = m1 & 1;
   else
      *status = ( m1 >> 1 ) & 1;

   *presses = m2;
   *horz_pos = m3;
   *vert_pos = m4;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:      mouse_release()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
     (input)      button    Left or right button
     (output)     status    Status of the button
     (output)     presses   Number of button releases
     (output)     horz_pos  Horizontal position at last release
     (output)     vert_pos  Vertical position at last release

   Returned:      (function returns nothing)

   Variables:     m1        Local variable for register ax
                  m2        Local variable for register bx
                  m3        Local variable for register cx
                  m4        Local variable for register dx

   Description:   Gets button release information
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
void mouse_release ( int button, int *status, int *releases,
                     int *horz_pos, int *vert_pos )
{
   int m1, m2, m3, m4;

   _asm
      {
      mov   ax, 6
      mov   bx, button
      int   33h
      mov   m1, ax
      mov   m2, bx
      mov   m3, cx
      mov   m4, dx
      }

   if ( button == LBUTTON )
      *status = m1 & 1;
   else
      *status = ( m1 >> 1 ) & 1;

   *releases = m2;
   *horz_pos = m3;
   *vert_pos = m4;
}


/* ---------------------------------------------
   Function:       mouse_sethorz()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (input)       horz_min  Minimum horizontal cursor position
     (input)       horz_max  Maximum horizontal cursor position

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets minimum and maximum horizontal mouse
                   cursor positions
   ---------------------------------------------
*/

void mouse_sethorz( int horz_min, int horz_max )
{
   _asm
```

```
    {
      mov   ax, 7
      mov   cx, horz_min
      mov   dx, horz_max
      int   33h
    }
}

/*-----------------------------------------------------------------
   Function:       mouse_setvert()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (input)       vert_min  Minimum vertical cursor position
     (input)       vert_max. Maximum vertical cursor position

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets minimum and maximum vertical mouse cursor
                   positions
   -----------------------------------------------------------------
*/

void mouse_setvert( int vert_min, int vert_max )
{
   _asm
     {
      mov   ax, 8
      mov   cx, vert_min
      mov   dx, vert_max
      int   33h
     }
}

/*-----------------------------------------------------------------
   Function:       mouse_setgcurs()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (input)       cursor    Structure defining a graphics cursor

   Returned:       (function returns nothing)
```

```
   Variables:      cursor_seg     Segment of the cursor structure
                   cursor_off     Offset of the cursor structure
                   hotx           Hot spot x value
                   hoty           Hot spot y value

   Description:    Creates a graphics mode mouse cursor
   ---------------------------------------------------
*/

void mouse_setgcurs( struct graphics_cursor far *cursor )
{
    unsigned cursor_seg = FP_SEG( cursor );
    unsigned cursor_off = FP_OFF( cursor );
    int hotx = cursor->hot_spot_x;
    int hoty = cursor->hot_spot_y;

    _asm
        {
        mov   ax, 9
        mov   bx, hotx
        mov   cx, hoty
        mov   es, cursor_seg
        mov   dx, cursor_off
        int   33h
        }
}


/* ---------------------------------------------------
   Function:       mouse_setcurs()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (input)       cursor_select  Hardware or software cursor
     (input)       screen_mask    Screen mask (or start scan line)
     (input)       cursor_mask    Cursor mask (or end scan line)

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the text mode hardware or software cursor
   ---------------------------------------------------
*/

void mouse_setcurs( int cursor_select, int screen_mask, int cursor_mask )
{
```

```
    _asm
        {
        mov   ax, 10
        mov   bx, cursor_select
        mov   cx, screen_mask
        mov   dx, cursor_mask
        int   33h
        }
}


/* ---------------------------------------------------------
   Function:       mouse_motion()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (output)      horz_mickeys   Horizontal mickeys
     (output)      vert_mickeys   Vertical mickeys

   Returned:       (function returns nothing)

   Variables:      m3      Local variable for register cx
                   m4      Local variable for register dx

   Description:    Gets the accumulated mouse motion counts
                   (mickeys) since the last call to this function
   ---------------------------------------------------------
*/

void mouse_motion( int *horz_mickeys, int *vert_mickeys )
{
    int m3, m4;

    _asm
        {
        mov   ax, 11
        int   33h
        mov   m3, cx
        mov   m4, dx
        }

    *horz_mickeys = m3;
    *vert_mickeys = m4;
}


/* ---------------------------------------------------------
```

```
   Function:       mouse_setratios()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (output)      horizontal     Horizontal mickey/pixel ratio
     (output)      vertical       Vertical mickey/pixel ratio

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the mickey/pixel ratios for mouse motion
   ---------------------------------------------------------
*/

void mouse_setratios( int horizontal, int vertical )
{
   _asm
     {
     mov   ax, 15
     mov   cx, horizontal
     mov   dx, vertical
     int   33h
     }
}

/* ---------------------------------------------------------
   Function:       mouse_condoff()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (input)       x1        Upper left corner of region
     (input)       y1        Upper left corner of region
     (input)       x2        Lower right corner of region
     (input)       y2        Lower right corner of region

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets a region for conditionally turning off the
                   mouse cursor
   ---------------------------------------------------------
*/
```

```
void mouse_condoff( int x1, int y1, int x2, int y2 )
{
   _asm
      {
      mov   ax, 16
      mov   cx, x1
      mov   dx, y1
      mov   si, x2
      mov   di, y2
      int   33h
      }
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       mouse_setdouble()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
    (input)        mickeys_per_second  Double speed threshold

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the mouse double speed threshold
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

void mouse_setdouble( int mickeys_per_second )
{
    _asm
       {
       mov   ax, 19
       mov   dx, mickeys_per_second
       int   33h
       }
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       mouse_storage()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
    (output)       buffer_size    Bytes for saving mouse state
```

```
   Returned:      (function returns nothing)

   Variables:     m2        Local variable for register bx

   Description:   Determines the number of bytes required for
                  saving the current state of the mouse
   ---------------------------------------------------------
*/

void mouse_storage( int *buffer_size )
{
   int m2;

   _asm
      {
      mov   ax, 21
      int   33h
      mov   m2, bx
      }

   *buffer_size = m2;
}


/* ---------------------------------------------------------
   Function:      mouse_save()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
     (in/out)     buffer    Buffer for saving mouse state

   Returned:      (function returns nothing)

   Variables:     buffer_seg     Segment of the buffer
                  buffer_off     Offset of the buffer

   Description:   Saves the current state of the mouse
   ---------------------------------------------------------
*/

void mouse_save( char far *buffer )
{
   unsigned buffer_seg = FP_SEG( buffer );
   unsigned buffer_off = FP_OFF( buffer );

   _asm
      {
```

```
        mov   ax, 22
        mov   es, buffer_seg
        mov   dx, buffer_off
        int   33h
        }
}

/*-----------------------------------------------------
   Function:       mouse_restore()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
    (input)        buffer    Buffer for restoring the mouse state

   Returned:       (function returns nothing)

   Variables:      buffer_seg     Segment of the buffer
                   buffer_off     Offset of the buffer

   Description:    Restores the current state of the mouse
   -----------------------------------------------------
*/

void mouse_restore( char far *buffer )
{
    unsigned buffer_seg = FP_SEG( buffer );
    unsigned buffer_off = FP_OFF( buffer );

    _asm
        {
        mov   ax, 23
        mov   es, buffer_seg
        mov   dx, buffer_off
        int   33h
        }
}

/*-----------------------------------------------------
   Function:       mouse_setsensitivity()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
    (input)        horz      Relative horizontal sensitivity
```

```
   (input)      vert      Relative vertical sensitivity
   (input)      threshold Relative double speed threshold

  Returned:     (function returns nothing)

  Variables:    (none)

  Description:  Sets the mouse sensitivity and double speed
                threshold
  ---------------------------------------------------
*/

void mouse_setsensitivity( int horz, int vert, int threshold )
{
   _asm
    {
      mov   ax, 26
      mov   bx, horz
      mov   cx, vert
      mov   dx, threshold
      int   33h
      }
}


/* ---------------------------------------------------
   Function:      mouse_getsensitivity()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
    (output)    horz      Relative horizontal sensitivity
    (output)    vert      Relative vertical sensitivity
    (output)    threshold Relative double speed threshold

   Returned:    (function returns nothing)

   Variables:   (none)

   Description: Gets the mouse sensitivity and double speed
                threshold
   ---------------------------------------------------
*/

void mouse_getsensitivity( int *horz, int *vert, int *threshold )
{
   int m2, m3, m4;
```

```
    _asm
        {
        mov   ax, 27
        int   33h
        mov   m2, bx
        mov   m3, cx
        mov   m1, dx
        }

    *horz = m2;
    *vert = m3;
    *threshold = m4;
}


/* ---------------------------------------------
   Function:       mouse_setmaxrate()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
     (input)       interrupts_per_second - Interrupt rate

   Returned:       (function returns nothing)

   Variables:      rate      Number for range of interrupt rates

   Description:    Sets the interrupt rate (InPort mouse only)
   ---------------------------------------------
*/

void mouse_setmaxrate( int interrupts_per_second )
{
    int rate;

    if ( interrupts_per_second <= 0 )
        rate = 0;
    else if ( interrupts_per_second > 0 && interrupts_per_second <= 30 )
        rate = 1;
    else if ( interrupts_per_second > 30 && interrupts_per_second <= 50 )
        rate = 2;
    else if ( interrupts_per_second > 50 && interrupts_per_second <= 100 )
        rate = 3;
    else
        rate = 1;

    _asm
        {
```

```
        mov   ax, 28
        mov   bx, rate
        int   33h
        }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       mouse_setpage()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
    (input)        crt_page  Video page for mouse cursor

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Sets the video page where mouse cursor appears
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

void mouse_setpage( int crt_page )
{
    _asm
        {
        mov   ax, 29
        mov   bx, crt_page
        int   33h
        }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:       mouse_getpage()
   Toolbox:        MOUSEFUN.C
   Demonstrated:   MOUSTEST.C

   Parameters:
    (output)       crt_page  Video page for mouse cursor

   Returned:       (function returns nothing)

   Variables:      m2        Local variable for register bx

   Description:    Gets the video page in which mouse cursor appears
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
*/

void mouse_getpage( int *crt_page )
{
   int m2;

   _asm
   {
      mov   ax, 30
      int   33h
      mov   m2, bx
   }

   *crt_page = m2;
}


/* ---------------------------------------------------------
   Function:      mouse_setlang()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
    (input)       language  Language number

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Sets the language for mouse driver messages
   --------------------------------------------------------- */

void mouse_setlang( int language )
{
   _asm
   {
      mov   ax, 31
      mov   bx, language
      int   33h
   }
}


/* ---------------------------------------------------------
   Function:      mouse_getlang()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C
```

```
  Parameters:
    (output)     language  Language number

  Returned:      (function returns nothing)

  Variables:     (none)

  Description:   Gets the language for mouse driver messages
  ------------------------------------------------
*/

void mouse_getlang( int *language )
{
   int m2;

   _asm
      {
      mov   ax, 35
      int   33h
      mov   m2, bx
      }

   *language = m2;
}


/* ------------------------------------------------
   Function:      mouse_getversion()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
     (output)     version      Mouse driver version number
     (output)     mouse_type   Type of mouse
     (output)     irq_num      Interrupt request type

   Returned:      (function returns nothing)

   Variables:     m2           Local variable for register bx
                  m3           Local variable for register cx
                  maj          Major part of version number
                  min          Minor part of version number

   Description:   Gets the mouse driver version number, mouse type,
                  and interrupt request type
   ------------------------------------------------
*/
```

```
void mouse_getversion( double *version, int *mouse_type, int *irq_num )
{
    int m2, m3;
    int maj, min;

    _asm
        {
        mov   ax, 36
        int   33h
        mov   m2, bx
        mov   m3, cx
        }

    maj = ( m2 >> 12 ) * 10 + (( m2 >> 8 ) & 0xf );
    min = (( m2 >> 4 ) & 0xf ) * 10 + ( m2 & 0xf );
    *version = maj + min / 100.0;
    *mouse_type = m3 >> 8;
    *irq_num = m3 & 0xff;
}
```

```
/* ---------------------------------------------------------------------
   Nombre:        MOUSEFUN.H
   Tipo:          Include
   Descripción:   Prototipos y definiciones para el módulo MOUSEFUN.C
   ---------------------------------------------------------------------
*/

#ifndef MOUSEFUN_DEFINED

#define LBUTTON 0
#define RBUTTON 1

#define SOFT_TEXT_CURSOR 0
#define HARD_TEXT_CURSOR 1

#define ENGLISH 0
#define FRENCH 1
#define DUTCH 2
#define GERMAN 3
#define SWEDISH 4
#define FINNISH 5
#define SPANISH 6
#define PORTUGESE 7
#define ITALIAN 8

#define MOUSE_BUS 1
#define MOUSE_SERIAL 2
#define MOUSE_INPORT 3
#define MOUSE_PS2 4
#define MOUSE_HP 5

#define IRQ_PS2 0

/* Structure definition for graphics mode mouse cursors */
struct graphics_cursor
    {
    int screen_mask[16];
    int cursor_mask[16];
    int hot_spot_x;
    int hot_spot_y;
    };

void mouse_reset( int *, int * );                          /* Function 0 */
void mouse_show( void );                                   /* Function 1 */
void mouse_hide( void );                                   /* Function 2 */
void mouse_status( int *, int *, int *, int * );           /* Function 3 */
void mouse_setpos( int, int );                             /* Function 4 */
void mouse_press( int, int *, int *, int *, int * );       /* Function 5 */
```

```
void mouse_release( int, int *, int *, int *, int * );   /* Function  6 */
void mouse_sethorz( int, int );                          /* Function  7 */
void mouse_setvert( int, int );                          /* Function  8 */
void mouse_setgcurs( struct graphics_cursor far * );     /* Function  9 */
void mouse_setcurs( int, int, int );                     /* Function 10 */
void mouse_motion( int *, int * );                       /* Function 11 */
void mouse_setratios( int, int );                        /* Function 15 */
void mouse_condoff( int, int, int, int );                /* Function 16 */
void mouse_setdouble( int );                             /* Function 19 */
void mouse_storage( int * );                             /* Function 21 */
void mouse_save( char far * );                           /* Function 22 */
void mouse_restore( char far * );                        /* Function 23 */
void mouse_setsensitivity( int, int, int );              /* Function 26 */
void mouse_getsensitivity( int *, int *, int * );        /* Function 27 */
void mouse_setmaxrate( int );                            /* Function 28 */
void mouse_setpage( int );                               /* Function 29 */
void mouse_getpage( int * );                             /* Function 30 */
void mouse_setlang( int );                               /* Function 34 */
void mouse_getlang( int * );                             /* Function 35 */
void mouse_getversion( double *, int *, int * );         /* Function 36 */

#define MOUSEFUN_DEFINED
#endif
```

```
/* ---------------------------------------------------------
   Nombre:          GETKEY.C
   Tipo:            Móculo Toolbox de CEXPER.C
   Descripción:     Permite realizar el manejo del teclado.
   ---------------------------------------------------------
*/

#include <conio.h>
#include "mousefun.h"
#include "getkey.h"

#define HORZ_COUNTS 30
#define VERT_COUNTS 20

static int mouse_flag = 0;

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:        getkey()
   Toolbox:         GETKEY.C
   Demonstrated:    GETKTEST.C

   Parameters:      (none)

   Returned:        Unsigned integer key code

   Variables:       key      Value returned by getch()

   Description:     Returns an unsigned integer that corresponds to
                    a keypress
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

unsigned getkey( void )
{
    unsigned key;

    if (( key = getch()) == 0 )
        key = getch() << 8;

    return ( key );
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Function:        getkey_or_mouse()
   Toolbox:         GETKEY.C
   Demonstrated:    GETKTEST.C
```

```
Parameters:     (none)

Returned:       Unsigned integer key code

Variables:      key         Value returned by getch()
                status      Mouse status
                buttons     Number of mouse buttons
                horz        Horizontal mouse position
                vert        Vertical mouse position
                presses     Number of mouse button presses
                horz_pos    Mouse position at last button press
                vert_pos    Mouse position at last button press
                tot_horz    Accumulated horizontal mouse motion
                tot_vert    Accumulated vertical mouse motion

Description:    Returns an unsigned integer that corresponds to a
                keypress; also detects mouse motion and converts
                it to equivalent keypresses
---------------------------------------------------------------
*/

unsigned getkey_or_mouse( void )
{
    unsigned key;
    int status, buttons;
    int horz, vert;
    int presses, horz_pos, vert_pos;
    int tot_horz, tot_vert;

    /* Set the mouse motion counters to 0 */
    tot_horz = tot_vert = 0;

    /* Clear out the mouse button press counts */
    mouse_pressed( LBUTTON, &status, &presses, &horz_pos, &vert_pos );
    mouse_pressed( RBUTTON, &status, &presses, &horz_pos, &vert_pos );

    /* Loop starts here, watches for keypress or mouse activity */
    while ( 1 )
        {

        switch ( mouse_flag )
            {

            /* If this is first iteration, check for existence of mouse */
            case 0:
                mouse_reset( &status, &buttons );
                if ( status == 0 )
                    mouse_flag = -1;
```

```
    return ( key );
    }
}
```

```
        else
            mouse_flag = 1;
        break;

    /* If mouse does not exist, ignore monitoring functions */
    case -1:
        break;

    /* Check for mouse activity */
    case 1:

        /* Accumulate mouse motion counts */
        mouse_motion( &horz, &vert );
        tot_horz += horz;
        tot_vert += vert;

        /* Check for enough horizontal motion */
        if ( tot_horz < -HORZ_COUNTS )
            return ( KEY_LEFT );
        if ( tot_horz > HORZ_COUNTS )
            return ( KEY_RIGHT );

        /* Check for enough vertical motion */
        if ( tot_vert <= -VERT_COUNTS )
            return ( KEY_UP );
        if ( tot_vert > VERT_COUNTS )
            return ( KEY_DOWN );

        /* Check for mouse left button presses */
        mouse_press( LBUTTON, &status, &presses,
                     &horz_pos, &vert_pos );
        if ( presses )
            return ( KEY_ENTER );

        /* Check for mouse-right button presses */
        mouse_press( RBUTTON, &status, &presses,
                     &horz_pos, &vert_pos );
        if ( presses )
            return ( KEY_ESCAPE );
        break;
    }

/* Check for keyboard input */
if ( kbhit() )
    {
    if (( key = getch()) == 0 )
        key = getch() << 8;
```

```
/* ---------------------------------------------------------------------
   Nombre:          GETKEY.H
   Tipo:            Include
   Description:     Prototipos y definiciones del módulo GETKEY.C
   ---------------------------------------------------------------------
*/

#ifndef GETKEY_DEFINED

#define KEY_F1          15104
#define KEY_F2          15360
#define KEY_F3          15616
#define KEY_F4          15872
#define KEY_F5          16128
#define KEY_F6          16384
#define KEY_F7          16640
#define KEY_F8          16896
#define KEY_F9          17152
#define KEY_F10         17408
#define KEY_SHIFT_F1    21504
#define KEY_SHIFT_F2    21760
#define KEY_SHIFT_F3    22016
#define KEY_SHIFT_F4    22272
#define KEY_SHIFT_F5    22528
#define KEY_SHIFT_F6    22784
#define KEY_SHIFT_F7    23040
#define KEY_SHIFT_F8    23296
#define KEY_SHIFT_F9    23552
#define KEY_SHIFT_F10   23808
#define KEY_CTRL_F1     24064
#define KEY_CTRL_F2     24320
#define KEY_CTRL_F3     24576
#define KEY_CTRL_F4     24832
#define KEY_CTRL_F5     25088
#define KEY_CTRL_F6     25344
#define KEY_CTRL_F7     25600
#define KEY_CTRL_F8     25856
#define KEY_CTRL_F9     26112
#define KEY_CTRL_F10    26368
#define KEY_ALT_F1      26624
#define KEY_ALT_F2      26880
#define KEY_ALT_F3      27136
#define KEY_ALT_F4      27392
#define KEY_ALT_F5      27648
#define KEY_ALT_F6      27904
#define KEY_ALT_F7      28160
#define KEY_ALT_F8      28416
#define KEY_ALT_F9      28672
```

```
#define KEY_ALT_F10    28928
#define KEY_INSERT     20992
#define KEY_HOME       18176
#define KEY_PGUP       18688
#define KEY_DELETE     21248
#define KEY_END        20224
#define KEY_PGDN       20736
#define KEY_UP         18432
#define KEY_LEFT       19200
#define KEY_DOWN       20480
#define KEY_RIGHT      19712
#define KEY_ENTER      13
#define KEY_ESCAPE     27
#define KEY_BACKSPACE  8
#define KEY_TAB        9
#define KEY_SHIFT_TAB  3840
#define KEY_CTRL_LEFT  29440
#define KEY_CTRL_RIGHT 29696
#define KEY_CTRL_HOME  30464
#define KEY_CTRL_PGUP  33792
#define KEY_CTRL_PGDN  30208
#define KEY_CTRL_END   29952
#define KEY_CTRL_ENTER 10

unsigned int getkey( void );
unsigned int getkey_or_mouse( void );

#define GETKEY_DEFINED
#endif
```

LISTADO DE FUNCIONES PARA GRAFICAR EN PC-MATLAB

```
echo off
%       ------------------------------------------------------------
%       NOMBRE          :  GRAF.M
%       DESCRIPCION     :  Grafica la respuesta del sistema escogiendo las
%                          diferentes opciones del menu de gráficos.
%       ------------------------------------------------------------

clear

while 1

        opcion= ['tesis     '
                 'gabierto  '
                 'gcerrado  '
                 'gpid      '
                 'gdeadbea  '
                 'gself     '
                 'gexpert1  '
                 'gexpert2  '
                 'gtodos1   '
                 'gtodos2   '
                 'gerrors1  '
                 'gerrors2  '
                 'gcontro1  '
                 'gcontro2  '];

        clc
        help graflist
        n = input('          SELECCIONE UNA OPCION DEL MENU: ');
        if ((n <= 0) | (n > 14))
                break
        end
        opcion = opcion(n,:);
        eval(opcion)
        clear

end
clc
```

```
%  ---------------------------------------------------------------
%  NOMBRE          : GRAFLIST.M
%  DESCRIPCION     : Menú de opciones
%  ---------------------------------------------------------------

%  ***************************************************************
%  *            ------- MENU DE GRAFICOS ------                  *
%  *                                                             *
%  *    1)    Ejecutar tesis: CEXPER.EXE                         *
%  *    2)    Respuesta en lazo abierto.                         *
%  *    3)    Respuesta en lazo cerrado.                         *
%  *    4)    Respuesta con el control PID.                      *
%  *    5)    Respuesta con el control DEADBEAT.                 *
%  *    6)    Respuesta con el control SELF_TUNING.              *
%  *    7)    Respuesta con el control EXPERTO 1.                *
%  *    8)    Respuesta con el control EXPERTO 2.                *
%  *    9)    Salidas del sistema y control EXPERTO 1.           *
%  *    10)   Salidas del sistema y control EXPERTO 2.           *
%  *    11)   Errores del sistema y control EXPERTO 1.           *
%  *    12)   Errores del sistema y control EXPERTO 2.           *
%  *    13)   Señal de control EXPERTO 1.                        *
%  *    14)   Señal de control EXPERTO 2.                        *
%  *                                                             *
%  *    0)    SALIR.                                             *
%  ***************************************************************

echo off
clc

% ---------------------------------------------------------------
% NOMBRE          : TESIS.M
% DESCRIPCION     : Función para ejecutar el programa principal CEXPER.EXE
%                   que genera los datos a ser analizados.
% ---------------------------------------------------------------

!cexper
clc


echo off
clc
clear
```

```
%  ------------------------------------------------------------
%  NOMBRE                : GABIERTO.M
%  DESCRIPCION           : Función para graficar la respuesta del sistema
%                          en lazo abierto.
%  ------------------------------------------------------------

load abierto.dat
plot(abierto(:,1),abierto(:,2),'-',abierto(:,1),abierto(:,3),'--')
title('RESPUESTA DE LA PLANTA EN LAZO ABIERTO')
xlabel(' - SALIDA      -- REFERENCIA          Tiempo(seg)')
ylabel('MAGNITUD')
grid
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO?  (1) SI    (2) NO ');
if (p == 1)
   print
end
clc



echo off
clc
clear

%  ------------------------------------------------------------
%  NOMBRE                : GCERRADO.M
%  DESCRIPCION           : Función para graficar la respuesta del sistema
%                          en lazo cerrado.
%  ------------------------------------------------------------

load cerrado.dat
plot(cerrado(:,1),cerrado(:,3),'-',cerrado(:,1),cerrado(:,1),'--')
title('RESPUESTA DE LA PLANTA EN LAZO CERRADO')
xlabel(' - SALIDA      -- REFERENCIA          Tiempo(seg)')
ylabel('MAGNITUD')
grid
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO?  (1) SI    (2) NO ');
if (p == 1)
   print
end
clc
```

```
echo off
clc
clear

%  ------------------------------------------------------------------
%  NOMBRE              : GPID.M
%  DESCRIPCION         : Función para graficar la respuesta del sistema
%                        con el control PID.
%  ------------------------------------------------------------------

load pid.dat
plot(pid(:,1),pid(:,4),'-',pid(:,1),pid(:,5),'--')
title('RESPUESTA DE LA PLANTA CON EL CONTROL PID')
xlabel(' - SALIDA        -- REFERENCIA          Tiempo(seg)')
ylabel('MAGNITUD')
grid
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO?  (1) SI    (2) NO ');
if (p == 1)
   print
end
clc



echo off
clc
clear

%  ------------------------------------------------------------------
%  NOMBRE              : GDEADBEA.M
%  DESCRIPCION         : Función para graficar la respuesta del sistema
%                        con el control DEADBEAT.
%  ------------------------------------------------------------------

load deadbeat.dat
plot(deadbeat(:,1),deadbeat(:,4),'-',deadbeat(:,1),deadbeat(:,5),'--')
title('RESPUESTA DE LA PLANTA CON EL CONTROL DEADBEAT')
xlabel(' - SALIDA        -- REFERENCIA          Tiempo(seg)')
ylabel('MAGNITUD')
grid
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO?  (1) SI    (2) NO ');
if (p == 1)
   print
end
clc
```

```
echo off
clc
clear

%  ------------------------------------------------------------
%  NOMBRE              : GSELF.M
%  DESCRIPCION         : Función para graficar la respuesta del sistema
%                        con el control SELF_TUNING.
%  ------------------------------------------------------------

load self.dat
plot(self(:,1),self(:,4),'-',self(:,1),self(:,5),'--')
title('RESPUESTA DE LA PLANTA CON EL CONTROL SELF_TUNING')
xlabel(' - SALIDA        -- REFERENCIA          Tiempo(seg)')
ylabel('MAGNITUD')
grid
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO?  (1) SI    (2) NO ');
if (p == 1)
   print
end
clc



echo off
clc
clear

%  ------------------------------------------------------------
%  NOMBRE         : GEXPERT1.M
%  DESCRIPCION    : Función para graficar la respuesta del sistema
%                   con el control EXPERTO 1.
%  ------------------------------------------------------------

load experto1.dat
plot(experto1(:,1),experto1(:,4),'-',experto1(:,1),experto1(:,5),'--')
title('RESPUESTA DE LA PLANTA CON EL CONTROL EXPERTO 1')
xlabel(' - SALIDA        -- REFERENCIA          Tiempo(seg)')
ylabel('MAGNITUD')
grid
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO?  (1) SI    (2) NO ');
if (p == 1)
   print
end
clc
```

```
echo off
clc
clear

%  ------------------------------------------------------------
%  NOMBRE          : GEXPERT2.M
%  DESCRIPCION     : Función para graficar la respuesta del sistema
%                    con el control EXPERTO 2.
%  ------------------------------------------------------------

load experto2.dat
plot(experto2(:,1),experto2(:,4),'-',experto2(:,1),experto2(:,5),'--')
title('RESPUESTA DE LA PLANTA CON EL CONTROL EXPERTO 2')
xlabel(' - SALIDA        -- REFERENCIA                Tiempo(seg)'),
ylabel('MAGNITUD')
grid,
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO?  (1) SI     (2) NO ');
if (p == 1)
   print
end
clc



echo off
clc
clear

%  ------------------------------------------------------------
%  NOMBRE          : GTODOS1.M
%  DESCRIPCION     : Función para graficar la respuesta del sistema
%                    con todos los controles: PID, DEADBEAT, SELF_TUNING,
%                    EXPERTO 1.
%  ------------------------------------------------------------

load pid.dat
load deadbeat.dat
load self.dat
load experto1.dat
plot(experto1(:,1),experto1(:,4),'-',deadbeat(:,1),deadbeat(:,4),'--',pid(:,1),pid(:,4),'-
.',self(:,1),self(:,4),':')
title('RESPUESTAS DE LA PLANTA Y CONTROL EXPERTO 1')
xlabel('-.- PID   -- DEADBEAT   .. SELF-TUNING   - EXPERTO 1     Tiempo(seg)')
ylabel('MAGNITUD')
grid
pause % PRESIONE CUALQUIER TECLA PARA CONTINUAR
p = input('DESEA IMPRIMIR EL GRAFICO? (1) SI     (2) NO ');
```