ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA ELECTRICA

TESIS DE GRADO

SOFTWARE DIDACTICO PARA EL ANALISIS DE SISTEMAS DE TRANSMISION EN MODO TRANSVERSAL ELECTROMAGNETICO Y EN MODO FUNDAMENTAL

ANEXOS

TESIS PREVIA LA OBTENCION DEL TITULO DE INGENIERO
EN ELECTRONICA Y TELECOMUNICACIONES

DIEGO JAVIER SALAZAR SAETEROS

1993

ANEXOS

ANEXO I

LISTADO DEL PROGRAMA PRINCIPAL Y SUS MODULOS COMPONENTES

ANEXO II

MANUAL DE USO DEL PROGRAMA DESARROLLADO

```
/* ------------------------------------------------------------------
   Nombre:          REFLEX.C
   Tipo:            Programa principal
   Lenguaje:        Microsoft QuickC
   Video:           CGA, EGA, VGA, MCGA o HGC

   Listado:         REFLEX.C
                    MENU.C
                    BOX.C
                    MOUSEFUN.C
                    GETKEY.C
                    FUNCION.C
                    SOUND.C
                    VIDEO.C
                    COMPLEX.C
                    INGRESO.C

   Variables:
                    Zo              Impedancia característica (dato)
                    Zr              Impedancia de carga (dato)
                    Rs              Resistencia de la fuente (dato)
                    Zr_1            Impedancia de carga 1 (resultado/d1)
                    Zr_2            Impedancia de carga 2 (resultado/d2)
                    v               Vector de voltaje de la onda estacionaria
                    x               Vector de posición de la onda estacionaria
                    f               Vector de fase de la onda estacionaria
                    vcc             Vector de voltaje en corto-circuito
                    alfa_dB         Constante de atenuación (dato)
                    frecuencia      Frecuencia de operación (dato)
                    alfa            Constante de atenuación (resultado)
                    beta            Constante de fase (resultado)
                    alfa_beta       Relación Ó/β
                    frec_corte      Frecuencia de corte (modo dominante)
                    lambda          Longitud de onda
                    modulo_rho      Módulo del coeficiente de reflexión
                    angulo_rho      Angulo del coeficiente de reflexión
                    min_v           Valor mínimo normalizado de la onda
                    d1              Distancia al mínimo en CC desde el mínimo
                                    de la onda
                    d2              Distancia al mínimo en CC desde el mínimo
                                    de la onda
                    d               Distancia de los puntos de -3dB
                    S               Relación de onda estacionaria
                    error_s         Error promedio de simulación
                    l               Distancia entre la fuente y la carga
                    z               Distancia de análisis
                    presentar       Definición de carátula inicial
                    presionar       Barra de continuación
                    info_box        Definición de cuadro de información
                    info_equipo     Definición de cuadro de información
                    bar_main        Definición de la barra de menú
                    drop_equipo     Menú de selección de modo
                    drop_simular    Menú de simulación en la distancia
                    drop_grafico    Menú de selección de gráfico
                    drop_calculo    Menú de cálculo de impedancia de carga
                                    normalizada
```

```
                drop_s          Menú de selección y/o calculo de S
                drop_datos      Menú de simulación en el tiempo
                drop_fuente     Menú de selección de fuente de exitación
                drop_fin        Menú para salida
                ingreso_simular  Información de ingreso de datos en modo
                                 TEM
                ing_simular_guia Información de ingreso de datos en modo
                                 fundamental
                ingreso_t_l     Información de ingreso de datos en modo
                                 TEM
                ingreso_t_g     Información de ingreso de datos en modo
                                 fundamental
                calculo         Información de ingreso de datos para
                                 cálculo de zr
                comprobacion    Barra de comprobación de datos correctos
                espera          Mensaje de espera
                mensaje_t       Información de dato erróneo
                mensaje1        Información de dato erróneo
                mensaje2        Información de dato erróneo
                simulacion      Resultados de simulación
                enter           Barra de continuación
                msg_impresion   Mensaje de impresión
                msg_confirm     Mensaje de confirmación de impresión
                n               Primer nivel de selección
                n1              Segundo nivel de selección
                bar_choice      Barra de menú de selección
                equipo_flag     Bandera de selección de modo
                fin_flag        Bandera indicadora de fin
                simular_flag    Bandera indicadora de ingreso de datos
                simular_flag_t  Bandera indicadora de ingreso de datos
                calculo_flag    Bandera indicadora de ingreso de datos
                control_flag    Bandera de control de validez
                selec_flag      Bandera de no finalización
                chequeo         Caracter de control de ingreso de datos
                control         Caracter de control de impresión
                save_info_equipo    Variable de almacenamiento en buffer
                save_info_box       Variable de almacenamiento en buffer
                save_bar_main       Variable de almacenamiento en buffer
                save_drop_simular   Variable de almacenamiento en buffer
                save_drop_t         Variable de almacenamiento en buffer
                save_drop_calculo   Variable de almacenamiento en buffer
                save_ingreso_simular Variable de almacenamiento en buffer
                save_ingreso_calculo Variable de almacenamiento en buffer
                save_comprobar      Variable de almacenamiento en buffer
                save_espera         Variable de almacenamiento en buffer
                save_simulacion     Variable de almacenamiento en buffer
                save_enter          Variable de almacenamiento en buffer
                save_mensaje        Variable de almacenamiento en buffer
                save_msg_imp        Variable de almacenamiento en buffer
                save_msg_con        Variable de almacenamiento en buffer

Tratamiento:    (no command line parameters)

Descripción:    Programa que origina los menús de selección,
                  y realiza la ejecución de la opción deseada.
------------------------------------------------------------------ */
```

```
#include <stdio.h>
#include <graph.h>
#include <math.h>
#include "box.h"
#include "menu.h"
#include "sound.h"
#include "t_colors.h"
#include "video.h"
#include "funcion.h"
#include "ingreso.h"

/* Declaración de variables globales */

struct videoconfig config;

main()
{

/* Declaración de variables locales */

char *presentar[]=
    {
    "",
    "    E S C U E L A   P O L I T E C N I C A   N A C I O N A L",
    "                FACULTAD DE INGENIERIA ELECTRICA",
    "",
    "SOFTWARE DIDACTICO PARA EL ANALISIS DE SISTEMAS DE TRANSMISION",
    "  EN MODO TRANSVERSAL ELECTROMAGNETICO Y EN MODO FUNDAMENTAL",
    "",
    "                    Diego Salazar Saeteros",
    "                            1993",
    "",
    NULL
    };

char *presionar[] =
    {
    "",
    "Presione cualquier tecla para continuar",
    "",
    NULL
    };

char *drop_equipo[] =
    {
    " MODO A ANALIZAR ",
    "Modo TEM",
    "Modo Fundamental",
    " SELECCIONE ",
    NULL
    };

char *info_equipo[] =
    {
    " INSTRUCCIONES DE SELECCION DE MODO ",
    "El menú  permite  seleccionar  el  modo de",
```

```
    "transmisión a simularse.",
    "Para accesar  al menú,  use las teclas del",
    "cursor,  presione  la primera  letra de la",
    "opción  deseada,  o  utilice  su  mouse  y",
    "ubíquese en la opción escogida.  El  ENTER",
    "o la tecla izquierda del mouse seleccionan",
    "la opción.   El ESC o la tecla derecha del",
    "mouse terminan el programa.",
    "",
    NULL
    };

char *info_box[] =
    {
    " INSTRUCCIONES DE USO ",
    "La barra superior permite  seleccionar las",
    "varias opciones que se muestran.",
    "Para accesar  al menú,  use las teclas del",
    "cursor,  presione  la primera  letra de la",
    "opción  deseada,  o  utilice  su  mouse  y",
    "ubíquese en la opción escogida.  El  ENTER",
    "o la tecla izquierda del mouse seleccionan",
    "la opción.  El ESC o la tecla  derecha del",
    "mouse cancelan la opción.",
    "",
    NULL
    };

char *bar_main =  "  Patrón de onda  Señal en el tiempo  Cálculo de zr  Fin
";

char *drop_simular[] =
    {
    " PATRON DE ONDA ",
    "Ingreso de datos",
    "Resultados",
    "Gráficos",
    " SELECCIONE ",
    NULL
    };

char *drop_grafico[] =
    {
    " GRAFICOS ",
    "Magnitud",
    "Fase",
    " SELECCIONE ",
    NULL
    };

char *drop_datos[] =
    {
    " SEÑAL EN EL TIEMPO ",
    "Ingreso de datos",
    "Gráfico",
    " SELECCIONE ",
```

```
    NULL
    };

char *drop_calculo[] =
    {
    " CALCULO DE zr ",
    "Ingreso de datos",
    "Resultados",
    " SELECCIONE ",
    NULL
    };

char *drop_s[] =
    {
    " INGRESO O CALCULO DE S ",
    "Ingreso en forma directa",
    "Cálculo por puntos de -3dB",
    " SELECCIONE ",
    NULL
    };

char *drop_fin[] =
    {
    " FIN ",
    "No",
    "Si",
    " SELECCIONE ",
    NULL
    };

char *ingreso_simular[] =
    {
    " INGRESO DE DATOS - MODO TEM ",
    "Introduzca los valores que a continuación se detallan en",
    "las unidades indicadas  ( para valores infinitos ingrese",
    "la letra I ).  Presione <ENTER> para iniciar ingreso :",
    "",
    "*   Constante de atenuación (Ó) en [dB/m] :",
    "",
    "*    Frecuencia de operación (f) en [MHz] :",
    "",
    "*   Impedancia característica (Zo) en [Û] :",
    "",
    "*         Impedancia de carga (Zr) en [Û]",
    "                              Componente real :",
    "                         Componente imaginaria :",
    "",
    NULL
    };

char *ing_simular_guia[] =
    {
    " INGRESO DE DATOS - MODO FUNDAMENTAL ",
    "Introduzca los valores que a continuación se detallan en",
    "las unidades indicadas  ( para valores infinitos ingrese",
    "la letra I ).  Presione <ENTER> para iniciar ingreso :",
```

```
    "",
    "",
    "",
    "*    Frecuencia de operación (f) en [GHz] :",
    "",
    "*       Frecuencia de corte (fc) en [GHz] :",
    "",
    "*    Impedancia normalizada de carga (zr)",
    "                              Componente real :",
    "                         Componente imaginaria :",
    "",
    NULL
    };

char *drop_fuente[] =
    {
    " FUENTE DE EXITACION ",
    "Sinusoidal",
    "Continua",
    " SELECCIONE ",
    NULL
    };

char *ingreso_t_l[] =
    {
    " INGRESO DE DATOS - MODO TEM ",
    "Introduzca los valores que a continuación se detallan en",
    "las unidades indicadas  ( para valores infinitos ingrese",
    "la letra I ).  Presione <ENTER> para iniciar ingreso :",
    "",
    "* Resistencia interna de la fuente en [Û] :",
    "",
    "*   Impedancia característica (Zo) en [Û] :",
    "",
    "*         Impedancia de carga (Zr) en [Û]",
    "                              Componente real :",
    "                         Componente imaginaria :",
    "",
    "*  Distancia entre fuente y carga en [cm] :",
    "",
    "*       Distancia de análisis (x) en [cm] :",
    "",
    "*    Frecuencia de operación (f) en [MHz] :",
    "",

    NULL
    };

char *ingreso_t_g[] =
    {
    " INGRESO DE DATOS - MODO FUNDAMENTAL ",
    "Introduzca los valores que a continuación se detallan en",
    "las unidades indicadas  ( para valores infinitos ingrese",
    "la letra I ).  Presione <ENTER> para iniciar ingreso :",
    "",
    "*    Impedancia normalizada de carga (zr)",
```

```
    "                              Componente real :",
    "                           Componente imaginaria :",
    "",
    "*  Distancia entre fuente y carga en [cm] :",
    "",
    "*       Distancia de análisis (x) en [cm] :",
    "",
    "*    Frecuencia de operación (f) en [GHz] :",
    "",
    "*       Frecuencia de corte (fc) en [GHz] :",
    "",
    NULL
    };

char *mensaje_t[] =
    {
    " ERROR ",
    "La distancia de  análisis debe ser MENOR",
    "O IGUAL a la distancia entre la fuente y",
    "la impedancia de carga.",
    "",
    "",
    " Presione cualquier tecla para continuar ",
    NULL
    };


char *calculo[] =
    {
    " INGRESO DE DATOS - CALCULO DE zr ",
    "Introduzca los valores que a continuación se detallan en",
    "las unidades indicadas. Presione <ENTER> para ingreso :",
    "",
    "*    Frecuencia de operación (f) en [MHz] :",
    "",
    "*                      Distancia d1 en [cm] :",
    "",
    "*                      Distancia d2 en [cm] :",
    "",
    "*       Relación de onda estacionaria (S) :",
    "",
    "",
    "",
    NULL
    };

char *comprobacion[] =
    {
    "",
    "Todos los datos están correctos? <S/N>",
    "",
    NULL
    };

char *mensaje1[] =
    {
```

```
    " ERROR ",
    "La frecuencia de  operación (f) es  MENOR",
    "a la frecuencia  de corte (fc) de la guía",
    "",
    "           NO EXISTE PROPAGACION          ",
    "",
    "",
    " Presione cualquier tecla para continuar ",
    NULL
    };

char *mensaje2[] =
    {
    " ERROR ",
    "La frecuencia NO SE  ENCUENTRA dentro del",
    "ancho de banda de transmisión.            ",
    "",
    "  ATENUACION EN LA GUIA NO DESPRECIABLE  ",
    "",
    "",
    " Presione cualquier tecla para continuar ",
    NULL
    };

char *espera[] =
    {
    "",
    "Espere un momento por favor ....",
    "",
    NULL
    };

char *simulacion[] =
    {
    " RESULTADOS DE SIMULACION - MODO TEM ",
    "Frecuencia de operación (f) :                    ",
    "Constante de Atenuación (Ó) :                    ",
    "Constante de Fase (β) :                          ",
    "Módulo del Coeficiente de Reflexión :            ",
    "Angulo del Coeficiente de Reflexión :            ",
    "Valor máximo normalizado de señal (Vmax) : 1.000 ",
    "Valor mínimo normalizado de señal (Vmin) :       ",
    "Relación de onda estacionaria (S) :              ",
    "Distancia al mínimo en C.C. :                    ",
    "Distancia d1 :                                   ",
    "Distancia d2 :                                   ",
    "Impedancia de carga (con d1) :                   ",
    "Impedancia de carga (con d2) :                   ",
    "Error promedio de simulación :                   ",
    "",
    NULL
    };

char *resultados_zr[] =
    {
    " RESULTADOS DE CALCULO DE zr ",
```

```
    "Relación de onda estacionaria (S) :                    ",
    "Impedancia normalizada zr (con d1) :                   ",
    "Impedancia normalizada zr (con d2) :                   ",
    "",
    NULL
    };

char *enter[] =
    {
    "",
    "Presione <ENTER> para continuar",
    "",
    NULL
    };

char *msg_impresion[] =
    {
    "",
    "Desea impresión de resultados? <S/N>",
    "",
    NULL
    };

char *msg_confirm[] =
    {
    "",
    "Está la impresora encendida y en ON LINE? <S/N>",
    "",
    NULL
    };

struct complex Zo, Zr, Rs;
struct complex Zr_1, Zr_2;
double v[DATOS1+1], x[DATOS1+1], vcc[DATOS+1], f[DATOS+1];
double alfa_dB, frecuencia, frec_corte;
double alfa, beta, alfa_beta, lambda;
double modulo_rho, angulo_rho;
double min_v, d1, d2, S, d;
double error_s;
double l, z;
int n, n1;
int bar_choice = 1;
int fin_flag = 0;
int simular_flag = 0;
int simular_flag_t = 0;
int calculo_flag = 0;
int control_flag = 0;
int equipo_flag = 0;
int selec_flag = 0;
int far *save_info_equipo;
int far *save_info_box;
int far *save_bar_main;
int far *save_drop_simular;
int far *save_drop_calculo;
int far *save_ingreso_simular;
int far *save_drop_t;
```

```
int far *save_ingreso_calculo;
int far *save_comprobar;
int far *save_espera;
int far *save_mensaje;
int far *save_simulacion;
int far *save_enter;
int far *save_msg_imp;
int far *save_msg_con;
char chequeo = 1, control;

/* Inicialización de video */
setup();
cleanup();
_setvideomode( _TEXTC80 );
_clearscreen( _GCLEARSCREEN );
_displaycursor( _GCURSOROFF );

/* Llenar fondo de pantalla */

box_charfill( 1, 1, 25, 80, 219 );

/* Carátula de presentación */

menu_back_color( BK_RED );
menu_line_color( T_RED );
menu_text_color( T_WHITE | T_BRIGHT );
menu_message( 7, 7, presentar );
delay();
menu_back_color( BK_CYAN );
menu_line_color( T_CYAN );
menu_text_color( T_WHITE | T_BRIGHT );
menu_message( 21, 19, presionar );
getch();
_clearscreen( _GCLEARSCREEN );

/* Llenar fondo de pantalla */

box_charfill( 1, 1, 25, 80, 178 );
menu_back_color( BK_BLUE );
menu_line_color( T_CYAN );
menu_text_color( T_WHITE );
while ( !selec_flag )
{
fin_flag = 0;
_displaycursor( _GCURSORON );

/* Mostrar cuadro de información */

save_info_equipo = menu_message( 12, 25, info_equipo );

/* Selección de modo a simularse */

menu_erase( menu_drop( 5, 10, drop_equipo, &n ) );

if( n == 0 )
      selec_flag = 1;
```

```
else
{
switch( n )
      {
      case 1:
      /* selección de modo TEM */
            equipo_flag = 0;
            break;

      case 2:
      /* selección de modo fundamental */
            equipo_flag = 1;
            break;
      }

menu_erase( save_info_equipo );
box_charfill( 1, 1, 25, 80, 178 );

/* Inicio de lazo principal de control */

while ( !fin_flag )
    {
    box_charfill( 1, 1, 25, 80, 178 );
    chequeo = 1;

    /* Mostrar cuadro de información */
    save_info_box = menu_message( 12, 25, info_box );

    /* Mostrar barra de menú principal */
    save_bar_main = menu_bar( 3, 6, bar_main, &bar_choice );
    switch( bar_choice )
       {
       /* Simulación de Patrón de Onda Estacionaria */
       case 1:
           save_drop_simular =  menu_drop( 5, 5, drop_simular, &n );
           switch( n )
             {
             case 1:
                  /* función para ingreso de datos */
                  menu_erase( save_info_box );
                  menu_erase( save_bar_main );
                  menu_erase( save_drop_simular );

                  while( control_flag == 0 )
                    {
                    box_charfill( 1, 1, 25, 80, 178 );
                    if( equipo_flag == 0 )
                         {
                         save_ingreso_simular = menu_message( 4, 10,
                         ingreso_simular );
                         simular_flag = ingreso( &alfa_dB, &frecuencia,
                         &Zo.x, &Zo.y, &Zr.x, &Zr.y );
                         }
                    else
                         {
                         chequeo = 1;
```

```
            save_ingreso_simular = menu_message( 4, 10,
            ing_simular_guia );
            simular_flag = ingreso_guia( &frecuencia,
            &frec_corte, &Zr.x, &Zr.y );

            if( simular_flag != 0 )
            {
            if( frecuencia <= frec_corte )
               {
               save_mensaje = menu_message(15, 28, mensaje1);
               note( 900, 5 );
               getch();
               menu_erase( save_mensaje );
               chequeo = 2;
               }

            if( ( frecuencia < 1.25 * frec_corte ) ||
            ( frecuencia > 1.9 * frec_corte ) )
               {
               save_mensaje = menu_message(15, 28, mensaje2);
               note( 900, 5 );
               getch();
               menu_erase( save_mensaje );
               chequeo = 2;
               }
            }
            }

         if( simular_flag == 0 )
            {
            control_flag = 1;
            menu_erase( save_ingreso_simular );
            }
         else
         {
         if( chequeo != 2 )
            {
            save_comprobar = menu_message( 21, 19,
            comprobacion );
            chequeo = getch();
            if (chequeo == 'S' || chequeo == 's')
               control_flag = 1;
            else
               control_flag = 0;

            menu_erase( save_ingreso_simular );
            menu_erase( save_comprobar );
            }
         else
            {
            control_flag = 0;
            menu_erase( save_ingreso_simular );
            }
         }
         }
```

```
        control_flag = 0;
        if( simular_flag != 0 )
        {
        menu_back_color( BK_RED );
        menu_line_color( T_RED );
        menu_text_color( T_WHITE | T_BLINK );
        save_espera = menu_message( 12, 23, espera );

        /* Cálculo de resultados */

        if( equipo_flag == 1 )
           {
           Zo.x = 1;
           Zo.y = 0;
           alfa_dB = 0;
           frecuencia = frecuencia*1E3;
           }
        alfa_beta = k_propagacion( alfa_dB, frecuencia, &alfa,
        &beta, &lambda );
        reflexion( Zr, Zo, &modulo_rho, &angulo_rho );

        if( equipo_flag == 1 )
           kg( frecuencia*1E6, frec_corte*1E9, &beta, &lambda );

        onda( DATOS, alfa, beta, lambda, modulo_rho, angulo_rho,
        vcc, v, x, f );

        if( alfa_beta < 0.01 )
           {
           S = resultados( Zr, DATOS, v, x, lambda, &min_v, &d1,
           &d2 );
           Zr_1 = impedancia_carga( 1, d1, lambda, S, Zo );
           Zr_2 = impedancia_carga( 2, d2, lambda, S, Zo );
           error_s = error( Zr, Zr_1, Zr_2 );
           }
        note( 900, 5 );
        menu_back_color( BK_BLUE );
        menu_line_color( T_CYAN );
        menu_text_color( T_WHITE );
        menu_erase( save_espera );
        }
     break;

     case 2:
        /* función para presentación de resultados */
        menu_erase( save_info_box );
        menu_erase( save_bar_main );
        menu_erase( save_drop_simular );
        if( simular_flag == 0 )
          note( 900, 5 );
        else
          {
          if( equipo_flag == 1 )
               {
               simulacion[0] = " RESULTADOS DE SIMULACION - MODO
               FUNDAMENTAL ";
```

```
                alfa = frec_corte;
                }
            result_sim( equipo_flag, simulacion, frecuencia,
          alfa_beta, alfa, beta, modulo_rho, angulo_rho, min_v, S,
          lambda, d1, d2, Zr_1, Zr_2, error_s );

            if( alfa_beta < 0.01 )
                save_simulacion = menu_message(3, 13, simulacion);
            else
                save_simulacion = menu_message(3, 15, simulacion);

            save_msg_imp = menu_message( 21, 21, msg_impresion );
            control = getch();
            if (control == 'S' || control == 's')
                {
                menu_erase( save_msg_imp );
                save_msg_con = menu_message( 21, 16, msg_confirm );
                control = getch();
                if (control == 'S' || control == 's')
                  {
                  menu_erase( save_msg_con );
                  impresion( 16, simulacion );
                  }
                else
                  menu_erase( save_msg_con );
                }
            else
                menu_erase( save_msg_imp );

            save_enter = menu_message( 21, 24, enter );
            enter_key();
            menu_erase( save_simulacion );
            menu_erase( save_enter );
            }
        break;

    case 3:
        /* función de presentación de gráficos */
        if( simular_flag == 0 )
          {
          menu_erase( save_info_box );
          menu_erase( save_bar_main );
          menu_erase( save_drop_simular );
          note( 900, 5 );
          }
        else
          {
          menu_erase( menu_drop( 9, 16, drop_grafico, &n1 ) );
          menu_erase( save_info_box );
          menu_erase( save_bar_main );
          menu_erase( save_drop_simular );
          if( n1 == 0 )

              break;
          else
              {
```

```
                    switch( n1 )
                    {
                    case 1:
                      /* Gráfico de la magnitud del Patrón de Onda
                      Estacionaria */
                      graf_sim( equipo_flag, alfa_beta, x, v, vcc,
                      frecuencia, alfa, min_v, d1, d2, Zr );
                      box_charfill( 1, 1, 25, 80, 178 );
                      break;
                    case 2:
                      /* Gráfico de la fase del Patrón de Onda
                      Estacionaria */
                      graf_fase(equipo_flag, x, f, frecuencia, alfa,
                      Zr);
                      box_charfill( 1, 1, 25, 80, 178 );
                      break;
                    }
                    }
                  }
                break;

          default:
              menu_erase( save_info_box );
              menu_erase( save_bar_main );
              menu_erase( save_drop_simular );
              break;
          }
        break;

case 2:
    /* Simulación de la señal en el tiempo */
    save_drop_t = menu_drop( 5, 21, drop_datos, &n );
    switch( n )
      {
      case 1:
           /* función para ingreso de datos */
           if( equipo_flag == 0)
             menu_erase( menu_drop( 7, 30, drop_fuente, &n1 ) );
           else
             n1 = 1;

           if( n1 == 0 )
           {
           menu_erase( save_info_box );
           menu_erase( save_bar_main );
           menu_erase( save_drop_t );
           break;
           }
           else
           {
           menu_erase( save_info_box );
           menu_erase( save_bar_main );
           menu_erase( save_drop_t );

           while( control_flag == 0 )
             {
```

```
box_charfill( 1, 1, 25, 80, 178 );

if( n1 == 2 )
    ingreso_t_l[17] = "";
else
    ingreso_t_l[17] = "*    Frecuencia de operación (f)
    en [MHz] :";

chequeo = 1;

if( equipo_flag == 0 )
    {
    save_ingreso_simular = menu_message( 2, 10,
    ingreso_t_l );
    simular_flag_t = ing_t_l( n1, &frecuencia, &Rs.x,
    &Rs.y, &Zo.x, &Zo.y, &Zr.x, &Zr.y, &l, &z );
    }
else
    {
    save_ingreso_simular = menu_message( 4, 10,
    ingreso_t_g );
    simular_flag_t = ing_t_g(&frec_corte, &frecuencia,
    &Zr.x, &Zr.y, &l, &z );
    }

if( simular_flag_t == 0 )
    {
    control_flag = 1;
    menu_erase( save_ingreso_simular );
    }
else
    {
    if( l < z )
      {
      save_mensaje = menu_message( 15, 28, mensaje_t );
      note( 900, 5 );
      getch();
      menu_erase( save_mensaje );
      chequeo = 2;
      }
    if( equipo_flag == 1 )
      {
      if( frecuencia <= frec_corte )
          {
          save_mensaje = menu_message(15, 28,
          mensaje1);
          note( 900, 5 );
          getch();
          menu_erase( save_mensaje );
          chequeo = 2;
          }

      if( ( frecuencia < 1.25 * frec_corte ) ||
      ( frecuencia > 1.9 * frec_corte ) )
          {
          save_mensaje = menu_message(15, 28,
```

```
                        mensaje2);
                        note( 900, 5 );
                        getch();
                        menu_erase( save_mensaje );
                        chequeo = 2;
                        }
                    }

                  if( chequeo != 2 )
                    {
                    save_comprobar = menu_message( 22, 19,
                    comprobacion );
                    chequeo = getch();
                    if (chequeo == 'S' || chequeo == 's')
                        control_flag = 1;
                    else
                        control_flag = 0;

                    menu_erase( save_ingreso_simular );
                    menu_erase( save_comprobar );
                    }
                  else
                    {
                    control_flag = 0;
                    menu_erase( save_ingreso_simular );
                    }
                  }
                }
                control_flag = 0;

                if( simular_flag_t != 0 )
                {
                /* Cálculo de resultados */
                menu_back_color( BK_RED );
                menu_line_color( T_RED );
                menu_text_color( T_WHITE | T_BLINK );
                save_espera = menu_message( 12, 23, espera );
                onda_t( equipo_flag, n1, frecuencia, frec_corte, Rs,
                Zo, Zr, l, z, v, x );
                note( 900, 5 );
                menu_back_color( BK_BLUE );
                menu_line_color( T_CYAN );
                menu_text_color( T_WHITE );
                menu_erase( save_espera );
                }
              }
              break;

        case 2:
            /* función de presentación de gráfico en el tiempo */
            if( simular_flag_t == 0 )
              {
              menu_erase( save_info_box );
              menu_erase( save_bar_main );
              menu_erase( save_drop_t );
              note( 900, 5 );
```

```
                }
              else
                {
                menu_erase( save_info_box );
                menu_erase( save_bar_main );
                menu_erase( save_drop_t );
                graf_sim_t(n1, equipo_flag, frecuencia, Zr, l, z, v, x);
                box_charfill( 1, 1, 25, 80, 178 );
                }
              break;

          default:
              menu_erase( save_info_box );
              menu_erase( save_bar_main );
              menu_erase( save_drop_t );
              break;
          }
        break;

    case 3:
        /* Cálculo de la impedancia de carga */
        save_drop_calculo =  menu_drop( 5, 41, drop_calculo, &n );
        switch( n )
          {
          case 1:
               /* función para ingreso de datos */
               menu_erase( menu_drop( 7, 47, drop_s, &n1 ) );
               if( n1 == 0 )
               {
               menu_erase( save_info_box );
               menu_erase( save_bar_main );
               menu_erase( save_drop_calculo );
               break;
               }
               else
               {
               menu_erase( save_info_box );
               menu_erase( save_bar_main );
               menu_erase( save_drop_calculo );

               while( control_flag == 0 )
                 {
                 box_charfill( 1, 1, 25, 80, 178 );
                 if( equipo_flag == 1 )
                     {
                     calculo[12] = "*       Frecuencia de corte (fc) en
                     [GHz] :";
                     calculo[4] = "*    Frecuencia de operación (f) en
                     [GHz] :";
                     calculo[10] = "*       Relación de onda
                     estacionaria (S) :";
                     }
                 else
                     {
                     calculo[12] = " ";
                     calculo[4] = "*    Frecuencia de operación (f) en
```

```
        [MHz] :";
        }

    if( n1 == 2 )
        calculo[10] = "*  Distancia entre puntos de -3dB en
        [cm] :";

    save_ingreso_calculo = menu_message( 4, 10, calculo );
    calculo_flag = ing_calculo( equipo_flag, n1,
    &frec_corte, &frecuencia, &d1, &d2, &d );

    if( calculo_flag == 0 )
        {
        control_flag = 1;
        menu_erase( save_ingreso_calculo );
        }
    else
    {
    if( equipo_flag == 1 )
        {
        if( frecuencia <= frec_corte )
          {
          save_mensaje = menu_message( 15, 28, mensaje1 );
          note( 900, 5 );
          getch();
          menu_erase( save_mensaje );
          chequeo = 2;
          }
        }

    if( chequeo != 2 )
         {
         save_comprobar = menu_message( 21, 19,
         comprobacion );
         chequeo = getch();
         if (chequeo == 'S' || chequeo == 's')
          control_flag = 1;
         else
          control_flag = 0;

         menu_erase( save_ingreso_calculo );
         menu_erase( save_comprobar );
         }
    else
         {
         control_flag = 0;
         chequeo = 1;
         menu_erase( save_ingreso_calculo );
         }
    }
    }
control_flag = 0;

if( calculo_flag != 0 )
{
/* Cálculo de resultados */
```

```
                S = res_zr( equipo_flag, n1, frecuencia, frec_corte, d,
                &lambda );
                Zo.x= 1; Zo.y = 0;
                d1 = d1 * 1E-2;
                d2 = d2 * 1E-2;
                Zr_1 = impedancia_carga( 1, d1, lambda, S, Zo );
                Zr_2 = impedancia_carga( 2, d2, lambda, S, Zo );
                note( 900, 5 );
                }
                break;
                }

        case 2:
                /* función para presentación de resultados */
                menu_erase( save_info_box );
                menu_erase( save_bar_main );
                menu_erase( save_drop_calculo );
                if ( calculo_flag == 0 )
                   note( 900, 5 );
                else
                 {
                 result_zr( resultados_zr, S, Zr_1, Zr_2 );
                 save_simulacion = menu_message( 9, 13, resultados_zr );

                 save_msg_imp = menu_message( 21, 21, msg_impresion );
                 control = getch();
                 if (control == 'S' || control == 's')
                      {
                      menu_erase( save_msg_imp );
                      save_msg_con = menu_message( 21, 16, msg_confirm );
                      control = getch();
                      if (control == 'S' || control == 's')
                        {
                        menu_erase( save_msg_con );
                        impresion( 5, resultados_zr );
                        }
                      else
                        menu_erase( save_msg_con );
                      }
                 else
                      menu_erase( save_msg_imp );


                 save_enter = menu_message( 21, 24, enter );
                 enter_key();
                 menu_erase( save_simulacion );
                 menu_erase( save_enter );
                 }
                break;

        default:
                menu_erase( save_info_box );
                menu_erase( save_bar_main );
                menu_erase( save_drop_calculo );
                break;
        }
```

```
            break;

        case 4:
            /* Selección de fin */
            menu_erase( menu_drop( 5, 53, drop_fin, &n ) );
            switch( n )
              {
              case 1:
                   fin_flag = 1;
                   simular_flag = calculo_flag = simular_flag_t = 0;
                   menu_erase( save_info_box );
                   menu_erase( save_bar_main );
                   break;

              case 2:
                   fin_flag = selec_flag = 1;
                   break;

              default:
                   menu_erase( save_info_box );
                   menu_erase( save_bar_main );
                   break;
              }
            break;

        default:
            menu_erase( save_info_box );
            menu_erase( save_bar_main );
            break;
        }
    }
}
}

/* Limpieza de pantalla antes de salir */
_clearscreen( _GCLEARSCREEN );

}
```

```
/* ---------------------------------------------------------------------
   Nombre:          BOX.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           Modo de texto en color o monocromático
   ---------------------------------------------------------------------
*/

#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:          COMPLEX.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           (ningún requerimiento especial)
   ---------------------------------------------------------------------
*/

#include <math.h>
#include <stdio.h>
#include "complex.h"

/* ---------------------------------------------------------------------
   Function:        cstr()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          Structure of type complex
     (output)       str        String for formatted complex number

   Returned:        The formatted string

   Variables:       (none)

   Description:     Formats a complex number into a string
   ---------------------------------------------------------------------
*/
char *cstr( struct complex a, char *str )
{
    if ( a.y >= 0.0 )
        sprintf( str, "( %g +%gi )", a.x, a.y );
    else
        sprintf( str, "( %g %gi )", a.x, a.y );
    return ( str );
}

/* ---------------------------------------------------------------------
   Function:        strc()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        str        String designation of a complex number
     (output)       a          Structure of type complex

   Returned:        0 if conversion failed
                    2 if both numerical fields converted correctly

   Variables:       (none)

   Description:     Converts a string designation of a complex
                    number into a complex number structure
   ---------------------------------------------------------------------
*/
```

```
int strc( char *str, struct complex *a )
{
    if ( sscanf( str, " ( %lg %lg", &a->x, &a->y ) == 2 )
        return ( 2 );
    else
        return ( 0 );
}

/* ------------------------------------------------------------------
   Function:        cadd()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          First complex number
     (input)        b          Second complex number

   Returned:        Complex number result

   Variables:       c          Complex result

   Description:     Adds two complex numbers
   ------------------------------------------------------------------
*/

struct complex cadd( struct complex a, struct complex b )
{
    struct complex c;

    c.x = a.x + b.x;
    c.y = a.y + b.y;

    return ( c );
}

/* ------------------------------------------------------------------
   Function:        csub()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          First complex number
     (input)        b          Second complex number

   Returned:        Complex number result

   Variables:       c          Complex result

   Description:     Subtracts two complex numbers
   ------------------------------------------------------------------
*/

struct complex csub( struct complex a, struct complex b )
{
```

```
    struct complex c;

    c.x = a.x - b.x;
    c.y = a.y - b.y;

    return ( c );
}


/* ---------------------------------------------------------------
   Function:        cmul()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          First complex number
     (input)        b          Second complex number

   Returned:        Complex number result

   Variables:       c          Complex result

   Description:     Multiplies two complex numbers
   ---------------------------------------------------------------
*/

struct complex cmul( struct complex a, struct complex b )
{
    struct complex c;

    c.x = a.x * b.x - a.y * b.y;
    c.y = a.x * b.y + a.y * b.x;

    return ( c );
}


/* ---------------------------------------------------------------
   Function:        cdiv()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          First complex number
     (input)        b          Second complex number

   Returned:        Complex number result

   Variables:       tmp        Variable for repetitive division
                c          Complex result

   Description:     Divides two complex numbers
   ---------------------------------------------------------------
*/
```

```
struct complex cdiv( struct complex a, struct complex b )
{
    double tmp;
    struct complex c;

    tmp = b.x * b.x + b.y * b.y;

    c.x = ( a.x * b.x + a.y * b.y ) / tmp;
    c.y = ( a.y * b.x - a.x * b.y ) / tmp;

    return ( c );
}


/* ------------------------------------------------------------------
   Function:        cpow()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          First complex number
     (input)        b          Second complex number

   Returned:        Complex number result

   Variables:       t1         Temporary complex number variable
                    t2         Temporary complex number variable
                    c          Complex result

   Description:     Raises complex number a to the power b
   ------------------------------------------------------------------
*/

struct complex cpow( struct complex a, struct complex b )
{
    struct complex t1, t2, c;

    t1 = clog( a );
    t2 = cmul( t1, b );
    c = cexp( t2 );

    return ( c );
}


/* ------------------------------------------------------------------
   Function:        croot()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          First complex number
     (input)        b          Second complex number

   Returned:        Complex number result
```

```
   Variables:      t          Temporary complex number variable
                   c          Complex result

   Description:    Finds the b root of a
   ------------------------------------------------------------------
*/

struct complex croot( struct complex a, struct complex b )
{
    struct complex t, c;

    t = crec( b );
    c = cpow( a, t );

    return ( c );
}


/* ------------------------------------------------------------------
   Function:       cexp()
   Toolbox:        COMPLEX.C
   Demonstrated:   COMPTEST.C

   Parameters:
     (input)       a          Complex number

   Returned:       Complex number result

   Variables:      tmp        Temporary double precision real
                   c          Complex result

   Description:    Finds the exponential function of a complex
                number, or raises e to the power a
   ------------------------------------------------------------------
*/

struct complex cexp( struct complex a )
{
    double tmp;
    struct complex c;

    tmp = exp( a.x );

    c.x = tmp * cos( a.y );
    c.y = tmp * sin( a.y );

    return ( c );
}


/* ------------------------------------------------------------------
   Function:       clog()
   Toolbox:        COMPLEX.C
   Demonstrated:   COMPTEST.C
```

```
   Parameters:
     (input)       a          Complex number

   Returned:       Complex number result

   Variables:      c          Complex result

   Description:    Finds the complex natural logarithm of a complex
                   number
   ---------------------------------------------------------------------
*/

struct complex clog( struct complex a )
{
    struct complex c;

    c.x = log( cabs( a ));
    c.y = atan2( a.y, a.x );

    return ( c );
}

/* ---------------------------------------------------------------------
   Function:       crec()
   Toolbox:        COMPLEX.C
   Demonstrated:   COMPTEST.C

   Parameters:
     (input)       a          Complex number

   Returned:       Complex number result

   Variables:      tmp        Temporary double precision real
                   c          Complex result

   Description:    Calculates 1 / a, or the complex reciprocal of
                   a complex number
   ---------------------------------------------------------------------
*/

struct complex crec( struct complex a )

    double tmp;
    struct complex c;

    tmp = a.x * a.x + a.y * a.y;

    c.x = a.x / tmp;
    c.y = -a.y / tmp;

    return ( c );
```

```
/* ---------------------------------------------------------------------
   Function:        csqr()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          Complex number

   Returned:        Complex number result

   Variables:       tmp1       Temporary double precision real
                    tmp2       Temporary double precision real
                    c          Complex result

   Description:     Calculates the square root of a complex number
   ---------------------------------------------------------------------
*/

struct complex csqr( struct complex a )
{
    double tmp1, tmp2;
    struct complex c;

    tmp1 = sqrt( cabs( a ) );
    tmp2 = atan2( a.y, a.x ) / 2.0;

    c.x = tmp1 * cos( tmp2 );
    c.y = tmp1 * sin( tmp2 );

    return ( c );
}


/* ---------------------------------------------------------------------
   Function:        complex_to_polar()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          Complex number

   Returned:        Polar notation result

   Variables:       x          Temporary double precision real
                    y          Temporary double precision real
                    c          Complex result
              f

   Description:     Converts a complex number to its polar notation
                    equivalent
   ---------------------------------------------------------------------
*/
struct complex complex_to_polar(struct complex a)
{
struct complex c;
double x, y;
```

```
int f;

x = a.x;
y = a.y;

c.x = hypot(x, y);

if (a.y == 0)
      f = 1;
else
      f = a.y / fabs(a.y);
c.y = f * acos(a.x / c.x);

return(c);
}

/* ---------------------------------------------------------------------
   Function:        polar_to_complex()
   Toolbox:         COMPLEX.C
   Demonstrated:    COMPTEST.C

   Parameters:
     (input)        a          Polar notation number

   Returned:        Complex number result

   Variables:       radius     Temporary double precision real
                    angle      Temporary double precision real
                    c          Complex result

   Description:     Converts a polar number to the equivalent
                    complex number
   ---------------------------------------------------------------------
*/

struct complex polar_to_complex( struct complex a )
{
    double radius, angle;
    struct complex c;

    radius = a.x;
    angle = a.y;

    c.x = radius * cos( angle );
    c.y = radius * sin( angle );

    return ( c );
}
```

```
/* ------------------------------------------------------------------
   Nombre:          FUNCION.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           CGA, MCGA, EGA, VGA o HGC
   ------------------------------------------------------------------
*/

#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <graph.h>
#include <conio.h>
#include <string.h>
#include <pgchart.h>
#include <stdlib.h>
#include "complex.h"
#include "video.h"
#include "funcion.h"
#include "sound.h"
#include "getkey.h"

/* ------------------------------------------------------------------
   Función:         k_propagacion()
   Módulo:          FUNCION.C
   Principal:       REFLEX.C

   Parámetros:
   (entrada)        alfa_dB           Constante de atenuación
   (entrada)        frecuencia_MHz    Frecuencia de operación
   (salida)         alfa              Constante de atenuación
   (salida)         beta              Constante de fase
   (salida)         lambda            Longitud de onda

   Retorno:         (Ó/ß)             Relación alfa/beta

   Variables:       (ninguno)

   Descripción:     Rutina para el cálculo del coeficiente de
                    propagación y la longitud de onda
   ------------------------------------------------------------------
*/
double k_propagacion( double alfa_dB, double frecuencia_MHz, double  *alfa,
double *beta, double *lambda )
{
/* Cálculo de la constante de propagación (Ó + jß) */
*alfa = (log(10) / 20) * alfa_dB;
*beta = PI * frecuencia_MHz / 150;

/* Cálculo de la longitud de onda */
*lambda = 300 / frecuencia_MHz;

/* Ó/ß */
return( 5.497 * alfa_dB / frecuencia_MHz );
}
```

```
/* ------------------------------------------------------------------
   Función:        maximo()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
     (entrada)     num      Número de datos a compararse
     (entrada)     a        Vector de datos a compararse

   Retorno:        max_valor   Valor máximo del vector

   Variables:      i        Contador

   Descripción:    Rutina para cálculo del valor máximo
   ------------------------------------------------------------------
*/
double maximo( int num, double *a )
{
int i;
double max_valor = *a;
for ( i=1; i<=num; i++ )
      max_valor = max( *(a+i), max_valor );
return( max_valor );
}

/* ------------------------------------------------------------------
   Función:        minimo()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
     (entrada)     num      Número de datos a compararse
     (entrada)     a        Vector de datos a compararse

   Retorno:        min_valor   Valor máximo del vector

   Variables:      i        Contador

   Descripción:    Rutina para cálculo del valor máximo
   ------------------------------------------------------------------
*/
double minimo( int num, double *a )
{
int i;
double min_valor = *a;
for ( i=1; i<=num/2; i++ )
      min_valor = min( *(a+i), min_valor );
return( min_valor );
}

/* ------------------------------------------------------------------
   Función:        onda()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C
```

```
   Parámetros:
   (entrada)       n             Número de datos a calcularse
   (entrada)       alfa          Constante de atenuación
   (entrada)       beta          Constante de fase
   (entrada)       lambda        Longitud de onda
   (entrada)       modulo_rho    Módulo del coeficiente de reflexión
   (entrada)       angulo_rho    Angulo del coeficiente de reflexión
   (salida)        vcc           Vector de datos con Zcc
   (salida)        v             Vector de datos de voltaje
   (salida)        x             Vector de datos de posición
   (salida)        f             Vector de datos de fase

   Retorno:        (ninguno)

   Variables:      v1            Vector de datos de voltaje
                   max1_v        Valor máximo de voltaje
                   incremento_x  Intervalo de cálculo
                   p             Valor de cálculo
                   q             Valor de cálculo
                   qcc           Valor de cálculo
                   a             Valor de cálculo
                   b             Valor de cálculo
                   i             Contador

   Descripción:    Rutina para el cálculo de la onda estacionaria
   ------------------------------------------------------------------
*/
void onda( int n, double alfa, double beta, double lambda, double
modulo_rho, double angulo_rho, double *vcc, double *v, double *x, double *f
)
{
double v1[DATOS+1];
double max1_v, incremento_x, p, q, qcc, a, b;
int i;

p = log(1 / sqrt(modulo_rho));
q = -(angulo_rho * PI /.180) / 2;
qcc = -PI / 2;
a = tan( -q );

/* Inicialización de los vectores de datos */

for (i=0; i<=n; i++)
      {
      *(x+i) = *(v+i) = *(vcc+i) = *(f+i) = v1[i] = 0;
      }

/* Cálculo de la magnitud y fase de la de onda estacionaria */

incremento_x = lambda / n;

for (i=0; i<=n; i++)
      {
      b = tanh( alfa * *(x+i) + p ) * tan( beta * *(x+i) + q );
      v1[i] = sqrt( pow( sinh( alfa * *(x+i) + p ),2 ) + pow( cos( beta *
      *(x+i) + q ),2 ));
```

```
      *(vcc+i) = sqrt( pow( sinh( alfa * *(x+i) ),2 ) + pow( cos( beta *
      *(x+i) + qcc ),2 ));
      *(f+i) = 180 * atan( ( a + b ) / ( 1 - a * b ) ) / PI;
      *(x+i+1) = *(x+i) + incremento_x;
      }

/* Normalización del patrón de onda estacionaria */

max1_v = maximo( n, v1 );
for (i=0; i<=n; i++)
      if( ( q != 0 ) && ( q != qcc ) )
            *(v+i) = *(v1+i) / max1_v;
      else
            *(v+i) = *(v1+i);
}

/* ------------------------------------------------------------------
   Función:        distancia_1()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       a         Vector de datos de voltaje
   (entrada)       b         Vector de datos de posición
   (entrada)       num       Número de datos
   (entrada)       min_v     Valor mínimo de voltaje
   (salida)        indice    Indicador de posición de valor mínimo

   Retorno:        (d1)      Distancia d1

   Variables:      i         Contador

   Descripción:    Cálculo de d1 para el Método del Doble Mínimo
   ------------------------------------------------------------------
*/
double distancia_1(double *a, double *b, int num, double min_v, int
*indice)
{
double d1;
int i = 0;
do      {
      d1 = b[i];
      i++;
      }
while (a[i] != min_v);
d1 = b[i];
*indice = i;
return(d1);
}

/* ------------------------------------------------------------------
   Función:        impedancia_carga()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
```

```
   (entrada)       i       Toma valores de 1 o 2
   (entrada)       d       d1 para cuando i = 1
                           d2 para cuando i = 2
   (entrada)       lambda  Longitud de onda
   (entrada)       S       Relación de onda estacionaria
   (entrada)       Zo      Impedancia característica

   Retorno:        (Zr)    Impedancia de carga

   Variables:      A       Valor para cálculos
                   B       Valor para cálculos

   Descripción:        Rutina para cálculo de la impedancia de carga
   Método utilizado:   METODO DEL DOBLE MINIMO
                       d1 distancia desde el mínimo en CC hacia el
                       generador hasta el mínimo.
                       d2 distancia desde el mínimo en CC hacia la
                       carga hasta el mínimo.
   ------------------------------------------------------------------
*/
struct complex impedancia_carga(int i,  double d, double lambda, double  S,
struct complex Zo)
{
struct complex Zr, A, B;
A.x = 1;
B.x = S;
if (i == 1)
      B.y = -tan(2*PI*d/lambda);
else
      B.y = tan(2*PI*d/lambda);
A.y = S*B.y;
Zr = cmul(Zo, cdiv(A, B));
return(Zr);
}

/* ------------------------------------------------------------------
   Función:        reflexion()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       Zr          Impedancia de carga
   (entrada)       Zo          Impedancia característica
   (salida)        modulo_rho  Módulo del coeficiente de reflexión
   (salida)        angulo_rho  Angulo del coeficiente de reflexión

   Retorno:        (ninguno)

   Variables:      X           Valor para cálculos
                   Y           Valor para cálculos
                   rho         Coeficiente de reflexión

   Descripción:    Rutina para el cálculo del coeficiente de reflexión
   ------------------------------------------------------------------
*/
void reflexion(  struct complex Zr, struct complex  Zo, double *modulo_rho,
```

```
double *angulo_rho )
{
struct complex X, Y, rho;

if ((Zr.x == Zo.x) && (Zr.y == 0))  /* Para Zr = Zo */
      Zr.y = 0.0001;

X = csub(Zr, Zo);
Y = cadd(Zr, Zo);

if ((Zr.x == 0) && (Zr.y == 0)) /* Para corto-circuito */
      {
      rho.x = -1;
      rho.y = 0;
      }
else
      {
      if ((Zr.x == 1E50) && (Zr.y == 1E50)) /* Para circuito-abierto */
            {
            rho.x = 1;
            rho.y = 0;
            }
      else
            rho = cdiv(X, Y);
      }

*modulo_rho = complex_to_polar(rho).x;
*angulo_rho = (180 / PI) * complex_to_polar(rho).y;
}

/* ------------------------------------------------------------------
   Función:        relacion()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       Zr      Impedancia de carga
   (entrada)       a       Vector de datos de voltaje
   (entrada)       b       Vector de datos de posición
   (entrada)       min_v   Valor mínimo de voltaje
   (entrada)       i       Indice indicador del voltaje mínimo

   Retorno:        (S)     Relación de onda estacionaria

   Variables:      d       Distancia desde el voltaje mínimo hasta
                           ¹2 de min_v
                   d1      Valor para cálculos
                   min_s   Valor de ¹2 de min_v
                   dmin    Valor de distancia para min_v

   Descripción:    Cálculo de S para valores mayores que 10
   ------------------------------------------------------------------
*/
double relacion(  struct complex Zr,  double *a,  double *b,  double min_v,
double lambda, int i )
{
```

```
double d, d1, min_s, dmin, S;
min_s = min_v * sqrt( 2 );
dmin = b[i];
do      {
      d1 = b[i];
      i++;
      }
while (a[i] <= min_s);
d = 2 * ( d1 - dmin );
if( ( ( Zr.x == 1E50 ) || ( Zr.y  == 1E50 ) ) || ( ( Zr.x == 0 ) && (  Zr.y
== 0 ) ) ),
      S = 500;
else
    S = sqrt( 1 / ( pow( sin( PI * d / lambda ), 2 )) + 1 );
return( S );
}

/* ------------------------------------------------------------------
   Función:        resultados()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       Zr      Impedancia de carga
   (entrada)       n       Número de datos
   (entrada)       v       Vector de datos de voltaje
   (entrada)       x       Vector de datos de posición
   (entrada)       lambda  Longitud de onda
   (salida)        min_v   Voltaje mínimo
   (salida)        d1      Distancia d1
   (salida)        d2      Distancia d2

   Retorno:        (S)     Relación de onda estacionaria

   Variables:      indice  Indice indicador de voltaje mínimo

   Descripción:    Rutina para calcular el voltaje mínimo,
                d1, d2 y relación de onda estacionaria
------------------------------------------------------------------
*/
double resultados( struct complex Zr,  int n, double *v, double  *x, double
lambda, double *min_v, double *d1, double *d2 )
{
double S;
int indice;

*min_v = minimo( n, v );

/* Cálculo de la distancia d1 y d2 */

*d1 = distancia_1( v, x, n, *min_v, &indice );
*d2 = lambda / 2 - *d1;

/* Cálculo de la relación de onda estacionaria */

S = 1 / *min_v;
```

```
if( S > 10 )
      S = relacion( Zr, v, x, *min_v, lambda, indice );
return( S );
}

/* ---------------------------------------------------------------------
   Función:          error()
   Módulo:           FUNCION.C
   Principal:        REFLEX.C

   Parámetros:
   (entrada)         Zr    Impedancia de carga verdadera
   (entrada)         Zr_1  Impedancia de carga calculada con d1
   (entrada)         Zr_2  Impedancia de carga calculada con d2

   Retorno:          Error en tanto por ciento

   Variables:        a     Porcentaje de cálculo
                     b     Porcentaje de cálculo
                     c     Porcentaje de cálculo
                     d     Porcentaje de cálculo

   Descripción:      Rutina para el cálculo del error de simulación
   ---------------------------------------------------------------------
*/
double error( struct complex Zr, struct complex Zr_1, struct complex Zr_2 )
{
double a, b, c, d;
a = porcentaje( Zr.x, Zr_1.x );
b = porcentaje( Zr.x, Zr_2.x );
c = porcentaje( Zr.y, Zr_1.y );
d = porcentaje( Zr.y, Zr_2.y );
return( (a + b + c + d) / 4 );
}

/* ---------------------------------------------------------------------
   Función:          porcentaje()
   Módulo:           FUNCION.C
   Principal:        REFLEX.C

   Parámetros:
   (entrada)         vr    Valor verdadero
   (entrada)         vc    Valor calculado

   Retorno:          (p)   Error en tanto por ciento

   Variables:        (ninguno)

   Descripción:      Rutina para el cálculo del error de simulación
   ---------------------------------------------------------------------
*/
double porcentaje( double vr, double vc )
{
double p;
if( vr == 0 )
      p = 0;
```

```
else
      p = fabs( ((vr - vc) / vr) * 100 );
return( p );
}

/* ------------------------------------------------------------------
   Función:          result_sim()
   Módulo:           FUNCION.C
   Principal:        REFLEX.C

   Parámetros:
   (entrada)         equipo          Indicador de modo utilizado
   (entrada)         sim             Arreglo de mensajes de salida
   (entrada)         f               Frecuencia de operación
   (entrada)         alfa_beta       Relación Ó/β
   (entrada)         alfa            Constante de atenuación
   (entrada)         beta            Constante de fase
   (entrada)         modulo_rho      Módulo del coeficiente de reflexión
   (entrada)         angulo_rho      Angulo del coeficiente de reflexión
   (entrada)         min_v           Voltaje mínimo
   (entrada)         S               Relación de onda estacionaria
   (entrada)         lambda          Longitud de onda
   (entrada)         d1              Distancia d1
   (entrada)         d2              Distancia d2
   (entrada)         Zr_1            Impedancia de carga con d1
   (entrada)         Zr_2            Impedancia de carga con d2
   (entrada)         error_s         Error porcentual de simulación

   Retorno:          (ninguno)

   Variables:        msg1 ... msg18  Mensajes de salida
                     frecuenciac     Frecuencia de operación
                     alfac           Constante de atenuación
                     betac           Constante de fase
                     moduloc         Módulo del coeficiente de reflexión
                     anguloc         Angulo del coeficiente de reflexión
                     vminc           Voltaje mínimo
                     Sc              Relación de onda estacionaria
                     dccc            Distancia al mínimo con Zcc
                     d1c             Distancia d1
                     d2c             Distancia d2
                     errorc          Error porcentual de simiulación
                     zr1             Impedancia de carga con d1
                     zr2             Impedancia de carga con d2
                     dcc             Distancia al mínimo con Zcc

   Descripción:      Rutina para presentación de resultados
   ------------------------------------------------------------------
*/
void result_sim( int equipo, char **sim, double f, double alfa_beta, double
alfa, double beta, double modulo_rho, double angulo_rho, double min_v,
double S, double lambda, double d1, double d2, struct complex Zr_1, struct
complex Zr_2, double error_s )
{
char msg1[56] = "Frecuencia de operación (f) : ";
char msg2[56] = "Constante de Atenuación (Ó) : ";
```

```
char msg3[56] = "Constante de Fase (β) : ";
char msg4[56] = "Módulo del Coeficiente de Reflexión : ";
char msg5[56] = "Angulo del Coeficiente de Reflexión : ";
char msg6[51] = "Valor máximo normalizado de señal (Vmax) : 1.000";
char msg7[56] = "Valor mínimo normalizado de señal (Vmin) : ";
char msg8[56] = "Relación de onda estacionaria (S) : ";
char msg9[56] = "Distancia al mínimo en C.C. : ";
char msg10[56] = "Distancia d1 : ";
char msg11[56] = "Distancia d2 : ";
char msg12[56] = "Impedancia de carga (con d1) : ";
char msg13[56] = "Impedancia de carga (con d2) : ";
char msg14[56] = "Error promedio de simulación : ";
char msg15[56] = "  IMPOSIBLE aplicar el Método del Doble Mínimo";
char msg16[56] = " para el cálculo de la impedancia de carga (Zr)";
char msg17[56] = "              Relación Ó/β > 0,01";
char msg18[56] = "";
char msg19[56] = "Frecuencia de corte (fc) : ";
char msg20[56] = "Constante de propagación (kg) : ";
char msg21[56] = "Zr normalizada (con d1) : ";
char msg22[56] = "Zr normalizada (con d2) : ";
char msg23[56] = "Campo eléctrico máximo normalizado (Emax) : 1.000";
char msg24[56] = "Campo eléctrico mínimo normalizado (Emin) : ";

char frecuenciac[15], alfac[19], betac[15], moduloc[9], anguloc[13];
char vminc[7], Sc[7], dccc[11], d1c[11], d2c[11], errorc[11];
char zr1[25], zr2[25];
double dcc;
dcc = lambda / 2;
if( equipo == 1 )
      {
      sprintf( frecuenciac, "%5.2f GHz", ( f / 1E3 ) );
      sprintf( alfac, "%5.2f GHz", ( alfa ) );
      sprintf( betac, "%5.3f rad/m", beta );
      strcat( msg19, alfac );
      strcpy( sim[2], msg19 );
      strcat( msg20, betac );
      strcpy( sim[3], msg20 );
      }
else
      {
      sprintf( frecuenciac, "%6.2f MHz", f );
      sprintf( alfac, "%6.5f nepper/m", alfa );
      sprintf( betac, "%5.3f rad/m", beta );
      strcat( msg2, alfac );
      strcpy( sim[2], msg2 );
      strcat( msg3, betac );
      strcpy( sim[3], msg3 );
      }
sprintf( moduloc, "%6.5f", modulo_rho );
sprintf( anguloc, "%6.2f °", angulo_rho );
strcat( msg1, frecuenciac );
strcat( msg4, moduloc );
strcat( msg5, anguloc );
strcpy( sim[1], msg1 );
strcpy( sim[4], msg4 );
strcpy( sim[5], msg5 );
```

```
if( alfa_beta < 0.01 )
      {
      sprintf( vminc, "%4.3f", min_v );
      sprintf( Sc, "%5.3f", S );
      sprintf( dccc, "%5.4f m", dcc );
      sprintf( d1c, "%5.4f m", d1 );
      sprintf( d2c, "%5.4f m", d2 );

      if( equipo == 1 )
            {
            strcat( msg24, vminc );
            strcpy( sim[7], msg24 );
            strcpy( sim[6], msg23 );
            if( Zr_1.y > 0 )
                  sprintf( zr1, "%5.3f + j %5.3f", Zr_1.x, Zr_1.y );
            else
                  sprintf(zr1, "%5.3f - j %5.3f", Zr_1.x, fabs( Zr_1.y ) );

            if( Zr_2.y > 0 )
                  sprintf( zr2, "%5.3f + j %5.3f", Zr_2.x, Zr_2.y );
            else
                  sprintf(zr2, "%5.3f - j %5.3f", Zr_2.x, fabs( Zr_2.y ) );
            }
      else
            {
            strcat( msg7, vminc );
            strcpy( sim[7], msg7 );
            strcpy( sim[6], msg6 );
            if( Zr_1.y > 0 )
                  sprintf( zr1, "%6.2f + j %6.2f (Û)", Zr_1.x, Zr_1.y );
            else
                  sprintf(zr1,"%6.2f - j %6.2f (Û)",Zr_1.x,fabs(Zr_1.y));

            if( Zr_2.y > 0 )
                  sprintf( zr2, "%6.2f + j %6.2f (Û)", Zr_2.x, Zr_2.y );
            else
                  sprintf(zr2,"%6.2f - j %6.2f (Û)",Zr_2.x,fabs(Zr_2.y));
            }

      sprintf( errorc, "%5.3f %s", error_s, "%" );

      if( S == 500 )
            strcat( msg8, "Valor infinito" );
      else
            strcat( msg8, Sc );

      strcat( msg9, dccc );
      if( angulo_rho == 0 )
            {
            strcat( msg10, d1c );
            strcat( msg11, d2c );
            strcat( msg12, "Valor infinito" );
            strcat( msg13, "Valor infinito" );
            strcat( msg14, "No calculado" );
            strcpy( sim[12], msg12 );
            strcpy( sim[13], msg13 );
```

```
            }
      else
            {
            if( angulo_rho >= 179.9 )
                  {
                  strcat( msg10, "0.0000 m" );
                  strcat( msg11, dccc );
                  strcat( msg12, "0.00 + j 0.00" );
                  strcat( msg13, "0.00 + j 0.00" );
                  strcat( msg14, "No calculado" );
                  strcpy( sim[12], msg12 );
                  strcpy( sim[13], msg13 );
                  }
            else
                  {
                  strcat( msg10, d1c );
                  strcat( msg11, d2c );
                  if( equipo == 1 )
                        {
                        strcat( msg21, zr1 );
                        strcat( msg22, zr1 );
                        strcpy( sim[12], msg21 );
                        strcpy( sim[13], msg22 );
                        }
                  else
                        {
                        strcat( msg12, zr1 );
                        strcat( msg13, zr2 );
                        strcpy( sim[12], msg12 );
                        strcpy( sim[13], msg13 );
                        }
                  strcat( msg14, errorc );
                  }
            }
      strcpy( sim[8], msg8 );
      strcpy( sim[9], msg9 );
      strcpy( sim[10], msg10 );
      strcpy( sim[11], msg11 );
      strcpy( sim[14], msg14 );
      }
else
      {
      strcpy( sim[6], msg18 );
      strcpy( sim[7], msg18 );
      strcpy( sim[8], msg18 );
      strcpy( sim[9], msg15 );
      strcpy( sim[10], msg16 );
      strcpy( sim[11], msg17 );
      strcpy( sim[12], msg18 );
      strcpy( sim[13], msg18 );
      strcpy( sim[14], msg18 );
      }
```

```
/* ------------------------------------------------------------------
   Función:         kg()
   Módulo:          FUNCION.C
   Principal:       REFLEX.C

   Parámetros:
   (entrada)        f          Frecuencia de operación
   (entrada)        fc         Frecuencia de corte
   (salida)         beta       Constante de propagación de la guía
   (salida)         lambda     Longitud de onda de la guía

   Retorno:         (kg)       Constante de propagación

   Variables:       (ninguno)

   Descripción:     Rutina para el cálculo de la cte. de propagación
   ------------------------------------------------------------------
*/
double kg( double f, double fc, double *beta, double *lambda )
{
*beta =  2 * PI / 3E8 * sqrt( pow( f, 2 ) - pow( fc, 2 ) );
*lambda = 3E8 / sqrt( pow( f, 2 ) - pow( fc, 2 ) );
}

/* ------------------------------------------------------------------
   Función:         res_zr()
   Módulo:          FUNCION.C
   Principal:       REFLEX.C

   Parámetrcs:
   (entrada)        flag     Bandera indicadora de modo utilizado
                    n        Bandera de selección de cálculo
   (entrada)        f        Frecuencia de operación
   (entrada)        fc       Frecuencia de corte
   (entrada)        a        Distancia de -3dB
   (salida)         lambda   Longitud de onda

   Retorno:         (S)      Relación de onda estacionaria

   Descripción:     Cálculo de S y longitud de onda
   ------------------------------------------------------------------
*/
double res_zr( double flag, double n, double f, double fc, double a, double
*lambda )
{
double  S;

if( flag == 1 )
      *lambda = 3E8/sqrt( pow( ( f * 1E9 ), 2 ) - pow( ( fc * 1E9 ), 2 ) );
else
      *lambda = 300 / f;

if( n == 2 )
      S = sqrt( 1 / ( pow( sin( PI * ( a * 1E-2 ) / *lambda ), 2 )) + 1 );
else
      S = a;
```

```
return( S );
}

/* ------------------------------------------------------------------
   Función:        result_zr()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       sim             Arreglo de mensajes de salida
   (entrada)       S               Relación de onda estacionaria
   (entrada)       Zr_1            Impedancia normalizada de carga con d1
   (entrada)       Zr_2            Impedancia normalizada de carga con d2

   Retorno:        (ninguno)

   Variables:      msg1 ... msg18  Mensajes de salida
                   Sc              Relación de onda estacionaria
                   zr1             Impedancia de carga con d1
                   zr2             Impedancia de carga con d2

   Descripción:    Rutina para presentación de resultados de zr
   ------------------------------------------------------------------
*/
void result_zr( char **sim, double S,  struct complex Zr_1, struct  complex
Zr_2 )
{
char msg1[56] = "Relación de onda estacionaria (S) : ";
char msg2[56] = "Impedancia normalizada zr (con d1) : ";
char msg3[56] = "Impedancia normalizada zr (con d2) : ";
char Sc[7], zr1[25], zr2[25];

sprintf( Sc, "%5.3f", S );

if( Zr_1.y > 0 )
      sprintf( zr1, "%5.3f + j %5.3f", Zr_1.x, Zr_1.y );
else
      sprintf( zr1, "%5.3f - j %5.3f", Zr_1.x, fabs( Zr_1.y ) );

if( Zr_2.y > 0 )
      sprintf( zr2, "%5.3f + j %5.3f", Zr_2.x, Zr_2.y );
else
      sprintf( zr2, "%5.3f - j %5.3f", Zr_2.x, fabs( Zr_2.y ) );

strcat( msg1, Sc );
strcat( msg2, zr1 );
strcat( msg3, zr2 );
strcpy( sim[1], msg1 );
strcpy( sim[2], msg2 );
strcpy( sim[3], msg3 );
}

/* ------------------------------------------------------------------
   Función:        onda_t()
   Módulo:         FUNCION.C
   Principal:      REFLEX.C
```

```
   Parámetros:
   (entrada)      m_flag    Bandera indicador de modo
   (entrada)      f_flag    Bandera indicadora de fuente de exitación
   (entrada)      f         Frecuencia de trabajo
   (entrada)      fc        Frecuencia de corte
   (entrada)      Rs        Resistencia de la fuente
   (entrada)      Zo        Impedancia característica
   (entrada)      Zr        Impedancia de carga
   (entrada)      l         Distancia entre fuente y carga
   (entrada)      x         Distancia de análisis
   (salida)       v         Vector de datos de voltaje
   (salida)       t         Vector de datos de tiempo

   Retorno:       (ninguno)

   Variables:     T         Período de tiempo
                  rho_s     Coef. de reflexión en la fuente
                  rho_r     Coef. de reflexión en la carga
                  Vo        Voltaje al inicio de la propagación
                  s         Coef. de reflexión en la fuente
                  r         Coef. de reflexión en la carga
                  c         Velocidad de la luz
                  w         Frecuencia de trabajo
                  i         Contador
                  j         Contador
                  e         Vector de módulos de voltaje
                  I         Vector de intervalos de tiempo
                  uno       uno complejo
                  A         Variable de uso temporal
                  B         Variable de uso temporal
                  C         Variable de uso temporal
                  w1        Variable de uso temporal
                  w2        Variable de uso temporal
                  a         Variable de uso temporal

   Descripción:   Rutina para el cálculo de la onda transitoria
                  al inicio de las reflexiones
   ------------------------------------------------------------------
*/
void onda_t(  int m_flag, int f_flag,  double f, double  fc, struct complex
Rs, struct complex  Zo, struct complex Zr,  double l, double x,  double *v,
double *t )
{
double T, Vo;
struct complex rho_s, rho_r, s, r, uno;
struct complex A[7], B[8], C[8];
double c, incremento, a, w, w1, w2;
double e[8], I[10];
int i, j;

f = f * 1E6;
fc = fc * 1E6;
Zo.y = Rs.y = uno.y = 0;

if( m_flag == 1 )
      {
```

```
        Zo.x = 1;
        Rs.x = 0;
        f = f * 1E3;
        fc = fc * 1E3;
        }
else
        fc = 0;

if( f_flag == 2 )
        c = 3E8;
else
        c = 3E8 * ( f / sqrt( ( f * f ) - ( fc * fc ) ) );

l = l * 1E-2;
x = x * 1E-2;
T = l / c;
uno.x = 1;
w = 2 * PI * f;
Vo = Zo.x / ( Zo.x + Rs.x );
reflexion( Rs, Zo, &s.x, &s.y );
reflexion( Zr, Zo, &r.x, &r.y );
s.y = s.y * PI / 180;
r.y = r.y * PI / 180;
rho_s = polar_to_complex( s );
rho_r = polar_to_complex( r );

A[0] = cmul( rho_r, rho_s );
for( i=1; i<=3; i++ )
        {
        A[2 * i - 1] = cmul( A[2 * i - 2], rho_r );
        A[2 * i] = cmul( A[2 * i - 1], rho_s );
        }

B[0] = cadd( uno, rho_r );
for( i=1; i<=7; i++ )
        B[i] = cadd( B[i-1], A[i-1] );

for( i=0; i<=7; i++ )
        {
        C[i] = complex_to_polar( B[i] );
        e[i] = Vo * C[i].x;
        }

I[0] = T * x / l;
for( i=1; i<=4; i++ )
        {
        I[2 * i - 1] = (-1) * I[0] + 2 * i * T;
        I[2 * i] = I[0] + 2 * i * T;
        }
I[9] =  (-1) * I[0] + 10 * T;

/* Inicialización de los vectores de datos */

for (i=0; i<=DATOS1; i++)
        *(v+i) = *(t+i) = 0;
```

```
/* Cálculo de la forma de onda en el tiempo */
switch( f_flag )
      {
       case 1:
      /* Fuente sinusoidal */
      *(t+0) = 0;
      incremento = I[0] * 10 / DATOS1;
      for (i=0; i<(DATOS1 / 10); i++)
            {
            *(v+i) = 0;
            *(t+i+1) = *(t+i) + incremento;
            }

      incremento = (I[1] - I[0]) * 10 / DATOS1;
      for (i=(DATOS1 / 10); i<(2 * DATOS1 / 10); i++)
            {
            if( incremento == 0 )
                  *(v+i) = *(v + ( DATOS1 / 10 - 1 ) );
            else
                  {
                  w1 = ( w * *(t+i) ) / ( 2 * PI );
                  w2 = 2 * modf( w1, &a ) * PI;
                  *(v+i) = Vo * sin( w2 );
                  }
            *(t+i+1) = *(t+i) + incremento;
            }

      for(j=1; j<=8; j++)
            {
            incremento = ( I[j + 1] - I[j] ) * 10 / DATOS1;
            for (i=((j + 1) * DATOS1 / 10); i<( (j + 2) * DATOS1 / 10);
            i++)
                  {
                  if( incremento == 0 )
                        *(v+i) = *(v + ( (j + 1) * DATOS1 / 10 - 1 ) );
                  else
                        {
                        w1 = ( w * *(t+i) ) / ( 2 * PI );
                        w2 = 2 * modf( w1, &a ) * PI;
                        *(v+i) = e[j - 1] * sin( w2 );
                        }
                  *(t+i+1) = *(t+i) + incremento;
                  }
            }
            break;

      case 2:
      /* Fuente continua */

      *(t+0) = 0;
      incremento = I[0] * 10 / DATOS1;
      for (i=0; i<(DATOS1 / 10); i++)
            {
            *(v+i) = 0;
            *(t+i+1) = *(t+i) + incremento;
            }
```

```
        incremento = (I[1] - I[0]) * 10 / DATOS1;
        for (i=(DATOS1 / 10); i<(2 * DATOS1 / 10); i++)
              {
              if( incremento == 0 )
                    *(v+i) = *(v + ( DATOS1 / 10 - 1 ) );
              else
                    *(v+i) = Vo;
              *(t+i+1) = *(t+i) + incremento;
              }

        for(j=1; j<=8; j++)
              {
              incremento = ( I[j + 1] - I[j] ) * 10 / DATOS1;
              for (i=((j + 1) * DATOS1 / 10); i<( (j + 2) * DATOS1 / 10);
              i++)
                    {
                    if( incremento == 0 )
                          *(v+i) = *(v + ( (j + 1) * DATOS1 / 10 - 1 ) );
                    else
                          *(v+i) = e[j - 1];
                    *(t+i+1) = *(t+i) + incremento;
                    }
              }
              break;
        }
}
```

```
/* ---------------------------------------------------------------------
   Nombre:          GETKEY.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           (ningún requerimiento especial)
   ---------------------------------------------------------------------
*/

#include <conio.h>
#include "mousefun.h"
#include "getkey.h"

#define HORZ_COUNTS 30
#define VERT_COUNTS 20

static int mouse_flag = 0;

/* ---------------------------------------------------------------------
   Function:        enter_key()
   Toolbox:         GETKEY.C
   Demonstrated:    REFLEX.C

   Parameters:      (none)

   Returned:        (none)

   Variables:       msg        Character for comparison

   Description:     Continues the program execution when <ENTER>
                       is pressed
   ---------------------------------------------------------------------
*/

void enter_key()
{
int msg;
do
      msg = getch();
while( msg != 13 );  /* ASCII for <ENTER> */
}

/* ---------------------------------------------------------------------
   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_press( LBUTTON, &status, &presses, &horz_pos, &vert_pos );
    mouse_press( 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;
            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 )
```

```
/* ------------------------------------------------------------------
   Nombre:          INGRESO.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           CGA, MCGA, EGA, VGA o HGC
   ------------------------------------------------------------------
*/

#include <stdio.h>
#include <errno.h>
#include <graph.h>
#include <conio.h>
#include <string.h>
#include <pgchart.h>
#include <stdlib.h>
#include <math.h>
#include "sound.h"
#include "getkey.h"
#include "ingreso.h"

/* ------------------------------------------------------------------
   Función:         ingreso()
   Módulo:          INGRESO.C
   Principal:       REFLEX.C

   Parámetros:
   (salida)         alfa_dB          Constante de atenuación
   (salida)         frecuencia_MHz   Frecuencia de operación
   (salida)         Zo               Impedancia característica
   (salida)         Zr               Impedancia de carga

   Retorno:         (1)   Para correcto ingreso de datos

   Variables:       alfa_dBa         Constante de atenuación
                    frecuencia_MHza  Frecuencia de operación
                    Zoa              Impedancia característica
                    Ra               Impedancia de carga
                    Xa               Impedancia de carga
                    dato_erroneo     Bandera de control de ingreso
                    esc              Control de tecla Esc

   Descripción:     Ingreso de datos y comprobación de su validez
   ------------------------------------------------------------------
*/
int  ingreso( double *alfa_dB, double *frecuencia_MHz, double *Zo_x, double
*Zo_y, double *Zr_x, double *Zr_y )
{
char alfa_dBa[20], frecuencia_MHza[20], Zoa[20], Ra[20], Xa[20];
int dato_erroneo = 1, esc;
_displaycursor( _GCURSORON );
_settextcolor( 0 );

dato_erroneo = 1;
while( dato_erroneo == 1 )
      {
```

```
      _settextposition( 9, 57 );
      printf("          ");
      _settextposition( 9, 58 );
      if( getkey_or_mouse() == KEY_ESCAPE )
            {
            esc = 1;
            dato_erroneo = 0;
            }
      else
            {
            fgets( alfa_dBa, sizeof( alfa_dBa ), stdin );
            dato_erroneo = comprobar( alfa_dBa );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( (alfa_dBa[0] == '-') || (alfa_dBa[0] == 'I') )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      }
      *alfa_dB = atof(alfa_dBa);

if( esc == 1 )
      return( 0 );
else
      {
      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition(11,57);
            printf("          ");
            _settextposition(11,58);
            fgets(frecuencia_MHza,sizeof(frecuencia_MHza),stdin);
            dato_erroneo = comprobar(frecuencia_MHza);
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if((frecuencia_MHza[0] == '-') || (frecuencia_MHza[0] == 'I') )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *frecuencia_MHz = atof(frecuencia_MHza);

      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition(13,57);
            printf("          ");
            _settextposition(13,58);
            fgets(Zoa,sizeof(Zoa),stdin);
            dato_erroneo = comprobar(Zoa);
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( (Zoa[0] == '-') || (Zoa[0] == 'I') )
```

```
                {
                dato_erroneo = 1;
                note( 900, 5 );
                }
          }
     *Zo_x = atof(Zoa);
     *Zo_y = 0;

     dato_erroneo = 1;
     while( dato_erroneo == 1 )
          {
          _settextposition(16,57);
          printf("          ");
          _settextposition(16,58);
          fgets(Ra,sizeof(Ra),stdin);
          dato_erroneo = comprobar(Ra);
          if( dato_erroneo == 1 )
               note( 900, 5 );
          if( Ra[0] == '-' )
               {
               dato_erroneo = 1;
               note( 900, 5 );
               }
          }
     *Zr_x = atof(Ra);
     if (Ra[0] == 'I')
          *Zr_x = 1E50;

     dato_erroneo = 1;
     while( dato_erroneo == 1 )
          {
          _settextposition(17,57);
          printf("          ");
          _settextposition(17,58);
          fgets(Xa,sizeof(Xa),stdin);
          dato_erroneo = comprobar(Xa);
          if( dato_erroneo == 1 )
               note( 900, 5 );
          }
     *Zr_y = atof(Xa);
     if (Xa[0] == 'I')
          *Zr_y = 1E50;

     _displaycursor(_GCURSOROFF);
     return(1);
     }
}

/* ------------------------------------------------------------------
   Función:        ingreso_guia()
   Módulo:         INGRESO.C
   Principal:      REFLEX.C

   Parámetros:
   (salida)        frecuencia_GHz   Frecuencia de operación
   (salida)        fc               Frecuencia de corte
```

```
   (salida)        zr                Impedancia de carga normalizada

   Retorno:        (1)   Para correcto ingreso de datos

   Variables:      frecuencia_GHza   Frecuencia de operación
                   fc_a              Frecuencia de corte
                   ra                Impedancia de carga normalizada
                   xa                Impedancia de carga normalizada
                   dato_erroneo      Bandera de control de ingreso
                   esc               Control de tecla Esc

   Descripción:    Ingreso de datos y comprobación de su validez
   ------------------------------------------------------------------
*/
int  ingreso_guia( double *frecuencia_GHz, double *fc, double *zr_x, double
*zr_y )
{
char frecuencia_GHza[20], fc_a[20], ra[20], xa[20];
int dato_erroneo = 1, esc;
_displaycursor(_GCURSORON);
_settextcolor(0);

dato_erroneo = 1;
while( dato_erroneo == 1 )
      {
      _settextposition(11,57);
      printf("          ");
      _settextposition(11,58);
      if( getkey_or_mouse() == KEY_ESCAPE )
            {
            esc = 1;
            dato_erroneo = 0;
            }
      else
            {
            fgets( frecuencia_GHza, sizeof( frecuencia_GHza ), stdin );
            dato_erroneo = comprobar( frecuencia_GHza );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if((frecuencia_GHza[0] == '-') || (frecuencia_GHza[0] == 'I'))
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      }
      *frecuencia_GHz = atof( frecuencia_GHza );

if( esc == 1 )
      return( 0 );
else
      {
      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition(13,57);
```

```
            printf("          ");
            _settextposition(13,58);
            fgets( fc_a, sizeof( fc_a ), stdin );
            dato_erroneo = comprobar( fc_a );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( ( fc_a[0] == '-' ) || ( fc_a[0] == 'I' ) )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *fc = atof( fc_a );

      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition(16,57);
            printf("          ");
            _settextposition(16,58);
            fgets( ra, sizeof( ra ), stdin );
            dato_erroneo = comprobar( ra );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( ra[0] == '-' )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *zr_x = atof( ra );
      if ( ra[0] == 'I' )
            *zr_x = 1E50;

      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition(17,57);
            printf("          ");
            _settextposition(17,58);
            fgets( xa, sizeof( xa ), stdin );
            dato_erroneo = comprobar( xa );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            }
      *zr_y = atof( xa );
      if ( xa[0] == 'I' )
            *zr_y = 1E50;

      _displaycursor(_GCURSOROFF);
      return(1);
      }
```

```
/* ------------------------------------------------------------------
   Función:        ing_t_1()
   Módulo:         INGRESO.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       n               Indicador de fuente
   (salida)        f_MHz           Frecuencia de operación
   (salida)        Rs              Resistencia de la fuente
   (salida)        Zo              Impedancia característica
   (salida)        Zr              Impedancia de carga
   (salida)        l               Distancia entre fuente y carga
   (salida)        x               Distancia de análisis

   Retorno:        (1)   Para correcto ingreso de datos

   Variables:      f_MHza          Frecuencia de operación
                   Rsa             Resistencia de carga
                   Zoa             Impedancia característica
                   Ra              Impedancia de carga
                   Xa              Impedancia de carga
                   la              Tiempo máximo de análisis
                   xda              Distancia de análisis
                   dato_erroneo    Bandera de control de ingreso
                   esc             Control de tecla Esc

   Descripción:    Ingreso de datos y comprobación de su validez
   ------------------------------------------------------------------
*/
int  ing_t_1(  int n,  double *f_MHz,  double *Rs_x,  double  *Rs_y, double
*Zo_x, double *Zo_y, double *Zr_x, double *Zr_y, double *l, double *x )
{
char Rsa[20], f_MHza[20], Zoa[20], Ra[20], Xa[20], la[20], xda[20];
int dato_erroneo = 1, esc;
_displaycursor( _GCURSORON );
_settextcolor( 0 );

dato_erroneo = 1;
while( dato_erroneo == 1 )
      {
      _settextposition( 7, 57 );
      printf("          ");
      _settextposition( 7, 58 );
      if( getkey_or_mouse() == KEY_ESCAPE )
            {
            esc = 1;
            dato_erroneo = 0;
            }
      else
            {
            fgets( Rsa, sizeof( Rsa ), stdin );
            dato_erroneo = comprobar( Rsa );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( (Rsa[0] == '-') || (Rsa[0] == 'I') )
                  {
```

```
                    dato_erroneo = 1;
                    note( 900, 5 );
                    }
              }
        }
        *Rs_x = atof( Rsa );
        *Rs_y = 0;

if( esc == 1 )
        return( 0 );
else
        {
        dato_erroneo = 1;
        while( dato_erroneo == 1 )
              {
              _settextposition( 9, 57 );
              printf( "          " );
              _settextposition( 9, 58 );
              fgets( Zoa, sizeof( Zoa ), stdin );
              dato_erroneo = comprobar( Zoa );
              if( dato_erroneo == 1 )
                    note( 900, 5 );
              if( ( Zoa[0] == '-' ) || ( Zoa[0] == 'I' ) )
                    {
                    dato_erroneo = 1;
                    note( 900, 5 );
                    }
              }
        *Zo_x = atof(Zoa);
        *Zo_y = 0;

        dato_erroneo = 1;
        while( dato_erroneo == 1 )
              {
              _settextposition( 12, 57 );
              printf("          ");
              _settextposition( 12, 58 );
              fgets( Ra, sizeof( Ra ), stdin );
              dato_erroneo = comprobar( Ra );
              if( dato_erroneo == 1 )
                    note( 900, 5 );
              if( Ra[0] == '-' )
                    {
                    dato_erroneo = 1;
                    note( 900, 5 );
                    }
              }
        *Zr_x = atof(Ra);
        if ( Ra[0] == 'I' )
              *Zr_x = 1E50;

        dato_erroneo = 1;
        while( dato_erroneo == 1 )
              {
              _settextposition( 13, 57 );
              printf("          ");
```

```
            _settextposition( 13, 58 );
            fgets( Xa, sizeof( Xa ), stdin );
            dato_erroneo = comprobar( Xa );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            }
      *Zr_y = atof( Xa );
      if ( Xa[0] == 'I' )
            *Zr_y = 1E50;

      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition( 15, 57 );
            printf("          ");
            _settextposition( 15, 58 );
            fgets( la, sizeof( la ), stdin );
            dato_erroneo = comprobar( la );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( ( la[0] == '-' ) || ( la[0] == 'I' ) )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *l = atof( la );

      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition( 17, 57 );
            printf("          ");
            _settextposition( 17, 58 );
            fgets( xda, sizeof( xda ), stdin );
            dato_erroneo = comprobar( xda );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( ( xda[0] == '-' ) || ( xda[0] == 'I' ) )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *x = atof(xda);

      if( n == 1 )
            {
            dato_erroneo = 1;
            while( dato_erroneo == 1 )
                  {
                  _settextposition( 19, 57 );
                  printf("          ");
                  _settextposition( 19, 58 );
                  fgets( f_MHza, sizeof( f_MHza ), stdin );
                  dato_erroneo = comprobar( f_MHza );
```

```
                if( dato_erroneo == 1 )
                      note( 900, 5 );
                if( (f_MHza[0] == '-') || (f_MHza[0] == 'I') )
                      {
                      dato_erroneo = 1;
                      note( 900, 5 );
                      }
                }
            *f_MHz = atof(f_MHza);
            }

      _displaycursor(_GCURSOROFF);
      return(1);
      }
}

/* ------------------------------------------------------------------------
   Función:         ing_t_g()
   Módulo:          INGRESO.C
   Principal:       REFLEX.C

   Parámetros:
   (entrada)        fc                Frecuencia de corte
   (salida)         f_GHz             Frecuencia de operación
   (salida)         zr                Impedancia de carga normalizada
   (salida)         l                 Tiempo máximo de análisis
   (salida)         x                 Distancia de análisis


   Retorno:         (1)   Para correcto ingreso de datos

   Variables:       f_GHza            Frecuencia de operación
                    fca               Frecuencia de corte
                    ra                Impedancia de carga normalizada
                    xa                Impedancia de carga normalizada
                    la                Tiempo máximo de análisis
                    xda               Distancia de análisis
                    dato_erroneo      Bandera de control de ingreso
                    esc               Control de tecla Esc

   Descripción:     Ingreso de datos y comprobación de su validez
   ------------------------------------------------------------------------
*/
int ing_t_g( double *fc, double *f_GHz, double *zr_x,  double *zr_y, double
*l, double *x )
{
char f_GHza[20], fca[20], ra[20], xa[20], la[20], xda[20];
int dato_erroneo = 1, esc;
_displaycursor( _GCURSORON );
_settextcolor( 0 );

dato_erroneo = 1;
while( dato_erroneo == 1 )
      {
      _settextposition( 10, 57 );
      printf("          ");
```

```
        _settextposition( 10, 58 );
        if( getkey_or_mouse() == KEY_ESCAPE )
              {
              esc = 1;
              dato_erroneo = 0;
              }
        else
              {
              fgets( ra, sizeof( ra ), stdin );
              dato_erroneo = comprobar( ra );
              if( dato_erroneo == 1 )
                    note( 900, 5 );
              if( ra[0] == '-' )
                    {
                    dato_erroneo = 1;
                    note( 900, 5 );
                    }
              }
        }
        *zr_x = atof( ra );
        if ( ra[0] == 'I' )
              *zr_x = 1E50;

if( esc == 1 )
        return( 0 );
else
        {
        dato_erroneo = 1;
        while( dato_erroneo == 1 )
              {
              _settextposition( 11, 57 );
              printf("          ");
              _settextposition( 11, 58 );
              fgets( xa, sizeof( xa ), stdin );
              dato_erroneo = comprobar( xa );
              if( dato_erroneo == 1 )
                    note( 900, 5 );
              }
        *zr_y = atof( xa );
        if ( xa[0] == 'I' )
              *zr_y = 1E50;

        dato_erroneo = 1;
        while( dato_erroneo == 1 )
              {
              _settextposition( 13, 57 );
              printf("          ");
              _settextposition( 13, 58 );
              fgets( la, sizeof( la ), stdin );
              dato_erroneo = comprobar( la );
              if( dato_erroneo == 1 )
                    note( 900, 5 );
              if( ( la[0] == '-' ) || ( la[0] == 'I' ) )
                    {
                    dato_erroneo = 1;
                    note( 900, 5 );
```

```
                }
        }
    *l = atof( la );

    dato_erroneo = 1;
    while( dato_erroneo == 1 )
        {
        _settextposition( 15, 57 );
        printf("          ");
        _settextposition( 15, 58 );
        fgets( xda, sizeof( xda ), stdin );
        dato_erroneo = comprobar( xda );
        if( dato_erroneo == 1 )
            note( 900, 5 );
        if( ( xda[0] == '-' ) || ( xda[0] == 'I' ) )
            {
            dato_erroneo = 1;
            note( 900, 5 );
            }
        }
    *x = atof( xda );

    dato_erroneo = 1;
    while( dato_erroneo == 1 )
        {
        _settextposition( 17, 57 );
        printf("          ");
        _settextposition( 17, 58 );
        fgets( f_GHza, sizeof( f_GHza ), stdin );
        dato_erroneo = comprobar( f_GHza );
        if( dato_erroneo == 1 )
            note( 900, 5 );
        if( (f_GHza[0] == '-') || (f_GHza[0] == 'I') )
            {
            dato_erroneo = 1;
            note( 900, 5 );
            }
        }
    *f_GHz = atof( f_GHza );

    dato_erroneo = 1;
    while( dato_erroneo == 1 )
        {
        _settextposition( 19, 57 );
        printf("          ");
        _settextposition( 19, 58 );
        fgets( fca, sizeof( fca ), stdin );
        dato_erroneo = comprobar( fca );
        if( dato_erroneo == 1 )
            note( 900, 5 );
        if( (fca[0] == '-') || (fca[0] == 'I') )
            {
            dato_erroneo = 1;
            note( 900, 5 );
            }
        }
```

```
        *fc = atof( fca );

        _displaycursor(_GCURSOROFF);
        return(1);
        }
}

/* ------------------------------------------------------------------
   Función:         ing_calculo()
   Módulo:          INGRESO.C
   Principal:       REFLEX.C

   Parámetros:
   (entrada)        flag            Bandera de selección de modo
   (entrada)        n               Selección de cálculo de S
   (salida)         fc              Frecuencia de corte
   (salida)         f               Frecuencia de operación
   (salida)         d1              Distancia d1
   (salida)         d2              Distancia d2
   (salida)         d               Distancia puntos de -3dB

   Retorno:         (1)   Para correcto ingreso de datos

   Variables:       fc_a            Frecuencia de corte
                    f_a             Frecuencia de operación
                    d1_a            Distancia d1
                    d2_a            Distancia d2
                    d_a             Distancia puntos de -3dB
                    dato_erroneo    Bandera de control de ingreso
                    esc             Control de tecla Esc

   Descripción:     Ingreso de datos y comprobación de su validez
   ------------------------------------------------------------------
*/
int ing_calculo( int flag, int n, double *fc, double *f, double *d1, double
*d2, double *d )
{
char fc_a[20], f_a[20], d1_a[20], d2_a[20], d_a[20];
int dato_erroneo = 1, esc;
_displaycursor(_GCURSORON);
_settextcolor(0);

dato_erroneo = 1;
while( dato_erroneo == 1 )
      {
      _settextposition( 8, 57 );
      printf("          ");
      _settextposition( 8, 58 );
      if( getkey_or_mouse() == KEY_ESCAPE )
            {
            esc = 1;
            dato_erroneo = 0;
            }
      else
            {
            fgets( f_a, sizeof( f_a ), stdin );
```

```
            dato_erroneo = comprobar( f_a );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( f_a[0] == '-' )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      }
      *f = atof(f_a);

if( esc == 1 )
      return( 0 );
else
      {
      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition( 10, 57 );
            printf("          ");
            _settextposition( 10, 58 );
            fgets( d1_a, sizeof( d1_a ), stdin );
            dato_erroneo = comprobar( d1_a );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( d1_a[0] == '-' )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *d1 = atof( d1_a);

      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition( 12, 57 );
            printf("          ");
            _settextposition( 12, 58 );
            fgets( d2_a, sizeof( d2_a ), stdin );
            dato_erroneo = comprobar( d2_a );
            if( dato_erroneo == 1 )
                  note( 900, 5 );
            if( d2_a[0] == '-' )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *d2 = atof( d2_a );

      dato_erroneo = 1;
      while( dato_erroneo == 1 )
            {
            _settextposition( 14, 57 );
```

```
            printf("          ");
            _settextposition( 14, 58 );
            fgets( d_a, sizeof( d_a ), stdin );
            dato_erroneo = comprobar( d_a );
            if( dato_erroneo == 1 )
            note( 900, 5 );
            if( d_a[0] == '-' )
                  {
                  dato_erroneo = 1;
                  note( 900, 5 );
                  }
            }
      *d = atof( d_a );

      if( flag == 1 )
            {
            dato_erroneo = 1;
            while( dato_erroneo == 1 )
                  {
                  _settextposition( 16, 57 );
                  printf("          ");
                  _settextposition( 16, 58 );
                  fgets( fc_a, sizeof( fc_a ), stdin );
                  dato_erroneo = comprobar( fc_a );
                  if( dato_erroneo == 1 )
                        note( 900, 5 );
                  if( fc_a[0] == '-' )
                        {
                        dato_erroneo = 1;
                        note( 900, 5 );
                        }
                  }
            *fc = atof( fc_a );
            }
      _displaycursor(_GCURSOROFF);
      return(1);
      }
}

/* ------------------------------------------------------------------
   Función:       comprobar()
   Módulo:        INGRESO.C
   Principal:     REFLEX.C

   Parámetros:
   (entrada)      numbera    Dato a comprobarse

   Retorno:       (0)        Dato correcto
                  (1)        Dato incorrecto

   Variables:     i          Contador

   Descripción:   Comprueba la validez de datos ingresados
   ------------------------------------------------------------------
*/
int comprobar(char *numbera)
```

```
{
int i=0;
while (numbera[i] != '\n')
      {
      if ((numbera[i] >= '0') && (numbera[i] <= '9'))
            i++;
      else
            if ((numbera[i] == '.') || (numbera[i] == '-') || (numbera[i]
            == 'I'))
                  i++;
            else
                  return(1);
      }
return(0);
}

/* ---------------------------------------------------------------------
   Función:          impresion()
   Módulo:           FUNCION.C
   Principal:        REFLEX.C

   Parámetros:
   (entrada)         lineas          Número de líneas a imprimirse
   (entrada)         mensaje         Mensaje a imprimirse

   Retorno:          (ninguno)

   Variables:        i               Contador

   Descripción:      Rutina para impresión de resultados
   ---------------------------------------------------------------------
*/
int impresion( int lineas, char **mensaje )
{
int i;
/* Inicialización de impresora */
fprintf( stdprn, "%c@",27 );  /*ESC @*/
fprintf( stdprn, "\n\n\n\n\n\n\n\n" );

/* Espaciamiento entre filas */
fprintf( stdprn, "%cA%c", 27, 8 );  /*ESC A 8*/

/* Calidad de texto NLQ */
fprintf( stdprn, "%cx%c", 27, 1 );  /*ESC x 1*/

/* Tipo de letra Roman */
fprintf( stdprn, "%ck%c", 27, 0 );  /*ESC k 1*/

/* Letra 10 cpi */
fprintf( stdprn, "%cP", 27 );  /*ESC P*/

/* Habilitación negrillas */
fprintf( stdprn, "%cE", 27 );  /*ESC E*/
fprintf( stdprn, "\n\n\n\t\t%s\n\n\n\n", mensaje[0] );
/* Deshabilitación negrillas */
fprintf( stdprn, "%cF", 27 );  /*ESC F*/
```

```
for( i = 1; i < lineas; i++ )
      fprintf( stdprn, "\t\t%s\n\n", mensaje[i] );
fprintf( stdprn, "\f", " " );
}

/* ------------------------------------------------------------------
   Función:         imp_graf()
   Módulo:          FUNCION.C
   Principal:       REFLEX.C

   Parámetros:
   (entrada)        x           Número de filas de pixeles
   (entrada)        y           Número de columnas de pixeles

   Retorno:         (ninguno)

   Variables:       F           Número de filas de 8 pixeles
                    C           Número de columnas
                    A, n1       Número de columnas de pixeles para
                                completar una fila
                    B, n2       Número de grupos de 256 columnas
                    pixel       Valor del color del pixel
                    numero      Caracter ASCII a imprimirse
                    datos       Vector de datos para impresión
                    z           Variable temporal
                    fila        Contador
                    columna     Contador
                    i           Contador

   Descripción:     Rutina para impresión de gráficos
   ------------------------------------------------------------------
*/
void imp_graf( int x, int y )
{
int i, F, C, A, B, fila, columna, pixel, numero;
int datos[8] = { 128, 64, 32, 16, 8, 4, 2, 1 };
double n1, n2, z;

numero = 0;
z = ( double ) x;
n1 = 256 * modf( z / 256, &n2);
A = ( int ) n1;
B = ( int ) n2;
F = ( y + 1 ) / 8;
C = x + 1;

/* Inicialización de impresora */
fprintf( stdprn, "%c@",27 );  /*ESC @*/
fprintf( stdprn, "\n\n\n\n\n\n\n\n\n\n\n\t\t" );

/* Espaciamiento entre filas */
fprintf( stdprn, "%cA%c", 27, 7 );  /*ESC A 7*/

/* Lazos de barrido de pantalla e impresión */
for( fila = 0; fila <= F; fila++ )
      {
```

```
      /* Densidad de impresión */
      fprintf( stdprn, "%cL%c%c", 27, A, B ); /*ESC L n1 n2*/
      for( columna = 0; columna <= C; columna++ )
            {
            for( i = 0; i <= 7; i++ )
                  {
                  pixel =  _getpixel( columna, i + 8 * fila );
                  if( pixel != 0 )
                        if( pixel != 3 )
                              if( pixel != 4)
                                    numero = numero + datos[i];
                  }
            fprintf( stdprn, "%c", numero );
            numero = 0;
            }
      fprintf( stdprn, "\n\t\t" );
      }
fprintf( stdprn, "\f" );
}
```

```
/* ---------------------------------------------------------------------
   Nombre:          MENU.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           (ningún requerimiento especial)
   ---------------------------------------------------------------------
*/

#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_WHITE | T_BRIGHT;
static int c_text = T_WHITE;
static int c_prompt = T_CYAN |T_BRIGHT;
static int c_hitext = T_WHITE | T_BRIGHT;
static int c_hiletter = T_RED | T_BRIGHT;
static long int c_back = BK_BLUE;
static long int c_hiback = BK_BLACK;

/* Default border lines and shadow control */
static int mb_lines = 1;
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
                    lastc           Last character checked
                    thisc           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 )
{
    int len;
    int fore;
    int maxchoice;
    int i, j;
    int cpos;
    int quit_flag = 0;
    int far *savebuf;
    int fstr[81];
    char lastc, thisc;
    long int bstr[81];
    unsigned key;
    long int back;
    struct rccoord oldpos;

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

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

    /* Calculate the string length only once */
    len = strlen( string );

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

    /* Put the menu bar on the screen */
    _settextposition( row, col );
    _outtext( string );

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

    /* Initialize choice if necessary */
    if ( *choice < 1 )
        *choice = 1;

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

        /* 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 + 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 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:          MOUSEFUN.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           CGA, MCGA, EGA, VGA o HGC
   ------------------------------------------------------------------------
*/

#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_settcurs()
   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_settcurs( 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   m4, 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 = 4;

    _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, 34
        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:          SOUND.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           (ningún requerimiento especial)
   ---------------------------------------------------------------
*/

#include <conio.h>
#include <time.h>
#include "sound.h"

static unsigned control;
static int control_flag = 1;

/* ---------------------------------------------------------------
   Function:        speaker_toggle()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:      (none)

   Returned:        (function returns nothing)

   Variables:       (none)

   Description:     Pulses the speaker on or off with each call
   ---------------------------------------------------------------
*/

void speaker_toggle( void )
{
    if ( control_flag )
        {
        control = inp( 0x61 );
        control_flag = 0;
        }
    outp( 0x61, ( inp( 0x61 ) & 0xFE ) ^ 2 );
}


/* ---------------------------------------------------------------
   Function:        sound()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:
     (input)        frequency       Frequency of generated tone

   Returned:        (function returns nothing)

   Variables:       divisor         Timer value for given frequency

   Description:     Sets a tone at a given frequency
   ---------------------------------------------------------------
```

```
*/

void sound( int frequency )
{
    unsigned divisor;

    divisor = (unsigned)( 1193180L / frequency );
    if ( control_flag )
        {
        outp( 0x43, 0xB6 );
        outp( 0x42, divisor % 256 );
        outp( 0x42, divisor / 256 );
        control = inp( 0x61 );
        control_flag = 0;
        }
    else
        {
        divisor = (unsigned)( 1193180L / frequency );
        outp( 0x42, divisor % 256 );
        outp( 0x42, divisor / 256 );
        }
    outp( 0x61, control | 3 );
}


/* ------------------------------------------------------------------
   Function:        silence()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:      (none)

   Returned:        (function returns nothing)

   Variables:       (none)

   Description:     Turns off the tone generator
   ------------------------------------------------------------------
*/

void silence( void )
{
    outp( 0x61, control );
    control_flag = 1;
}


/* ------------------------------------------------------------------
   Function:        wait_ticks()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:
     (input)        ticks           Number of clock ticks

   Returned:        (function returns nothing)
```

```
   Variables:      now             Time as returned by sound()

   Description:    Delays for a given number of clock ticks
   ------------------------------------------------------------------
*/

void wait_ticks( unsigned ticks )
{
    clock_t now;

    do
        {
        now = clock();
        while ( clock() == now )
            {;}
        }
    while( --ticks );
}


/* ------------------------------------------------------------------
   Function:       warble()
   Toolbox:        SOUND.C
   Demonstrated:   SOUNTEST.C

   Parameters:
     (input)       count     Number of warble cycles

   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Creates a three-tone warble
   ------------------------------------------------------------------
*/

void warble( int count )
{
    do
        {
        sound( 500 );
        wait_ticks( 1 );
        sound( 2000 );
        wait_ticks( 1 );
        sound( 1000 );
        wait_ticks( 1 );
        sound( 750 );
        wait_ticks( 1 );
        }
    while ( --count );

    silence();
```

```
/* ---------------------------------------------------------------------
   Function:        weird()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:      count      Number of sound generation cycles

   Returned:        (function returns nothing)

   Variables:       i          Looping index
                    j          Tone frequency

   Description:     Creates a modulated sound
   ---------------------------------------------------------------------
*/

void weird( int count )
{
    int i, j;

    sound( 50 );
    do
        for ( i = 50; i < 1200; i += 100 )
            for ( j = i; j < i + 1200; j += 5 )
                sound( j );
    while ( --count );

    silence();
}


/* ---------------------------------------------------------------------
   Function:        siren()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:      count      Number of sound generation cycles

   Returned:        (function returns nothing)

   Variables:       i          Looping index

   Description:     Creates a sound whose frequency rises and falls
   ---------------------------------------------------------------------
*/

void siren( int count )
{
    int i;

    sound( 50 );
    do
        {
        for ( i = 50; i < 2000; i++ )
            sound( i );
        for ( i = 2000; i > 50; i-- )
```

```
            sound( i );
        }
    while ( --count );

    silence();
}


/* ---------------------------------------------------------------------
   Function:        white_noise()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:      ticks      Number of clock ticks

   Returned:        (function returns nothing)

   Variables:       i          Looping index
                    rndm       Pseudorandom unsigned integer
                    now        Time as returned by clock()

   Description:     Generates white noise, a wide_ranging multifrequency
                    sound
   ---------------------------------------------------------------------
*/

void white_noise( int ticks )
{
    unsigned i, rndm;
    clock_t now;

    do
        {
        now = clock();
        while ( clock() == now )
            {
            speaker_toggle();
            rndm = rndm * 317 + 21317;
            for ( i = rndm & 0xFF; i; i-- )
                {;}
            }
        }
    while( --ticks );

    silence();
}


/* ---------------------------------------------------------------------
   Function:        note()
   Toolbox:         SOUND.C
   Demonstrated:    SOUNTEST.C

   Parameters:      frequency      Frequency of the tone
                    ticks          Number of clock ticks
```

```
   Returned:       (function returns nothing)

   Variables:      (none)

   Description:    Creates a tone given its frequency and duration
   ---------------------------------------------------------------
*/

void note( int frequency, int ticks )
{
    sound( frequency );
    wait_ticks( ticks );
    silence();
}
```

```
/* ---------------------------------------------------------------------
   Nombre:          VIDEO.C
   Tipo:            Módulo
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Video:           CGA, MCGA, EGA, VGA o HGC
   ---------------------------------------------------------------------
*/

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <graph.h>
#include <pgchart.h>
#include <math.h>
#include <stdlib.h>
#include "video.h"
#include "sound.h"
#include "getkey.h"
#include "ingreso.h"

typedef enum {FALSE, TRUE} boolean;

/* ---------------------------------------------------------------------
   Función:         delay()
   Módulo:          VIDEO.C
   Principal:       REFLEX.C

   Parámetros:      (ninguno)

   Retorno:         (ninguno)

   Variables:    n    Contador del lazo de demora

   Descripción:  Tiempo sin ninguna ejecución
   ---------------------------------------------------------------------
*/
void delay()
{
unsigned int i,j;
int n = 500;
for (i=1; i<=n; ++i)
      {
      for (j=1; j<=n; ++j);
      }
}

/* ---------------------------------------------------------------------
   Función:         rotulo()
   Módulo:          VIDEO.C
   Principal:       REFLEX.C

   Parámetros:
   (entrada)        msg         Mensaje a mostrarse

   Retorno:         (ninguno)
```

```
   Variables:      leftcol      Posición izquierda del rótulo
                   rightcol     Posición derecha del rótulo
                   toprow       Posición superior del rótulo
                   bottomrow    Posición inferior del rótulo

   Descripción:    Crea un rótulo con mensaje interior
   ------------------------------------------------------------------
*/
void rotulo(char *msg)
{
int leftcol, rightcol, toprow, bottomrow;
int vline,hline;

/* Inicialización de coordenadas para el rótulo */

leftcol = (80-strlen(msg))/2 - 3;

rightcol = 80 - leftcol;
toprow = 10;
bottomrow = 14;

/* Limpiar pantalla */

CLR_SCREEN;

/* Presentación de mensaje */

/* Dibuja las cuatro esquinas */

CUR_MOV(toprow,leftcol);
puts("\xC9"); /* ASCII 201 */
CUR_MOV(toprow,rightcol);
puts("\xBB"); /* ASCII 187 */
CUR_MOV(bottomrow,leftcol);
puts("\xC8"); /* ASCII 200 */
CUR_MOV(bottomrow,rightcol);
puts("\xBC"); /* ASCII 188 */

/* Dibuja las líneas verticales */

for (vline=toprow+1;vline<=bottomrow-1;++vline)
      {
      CUR_MOV(vline,leftcol);
      puts("\xBA"); /* ASCII 186 */
      CUR_MOV(vline,rightcol);
      puts("\xBA"); /* ASCII 186 */
      }

/* Dibuja las líneas horizontales */

for (hline=leftcol+1;hline<=rightcol-1;++hline)
      {
      CUR_MOV(toprow,hline);
      puts("\xCD"); /* ASCII 205 */
      CUR_MOV(bottomrow,hline);
      puts("\xCD"); /* ASCII 205 */
```

```
      }

/* Muestra mensaje */

CUR_MOV(toprow+2,leftcol+4);
printf("%s\n\n\n\n",msg);
CUR_MOV(25,80);
}

/* ---------------------------------------------------------------------
   Función:         setup()
   Módulo:          VIDEO.C
   Principal:       REFLEX.C

   Parámetros:      (ninguno)

   Retorno:         (ninguno)

   Variables:       config   Estructura de parámetros gráficos

   Descripción:     Comprueba e inicializa el modo gráfico, caso
                 contrario da un mensaje de imposibilidad
   ---------------------------------------------------------------------
*/
void setup()
{
struct videoconfig config;
_getvideoconfig( &config );
switch( config.adapter )
      {
      case _HGC:
            _setvideomode(_HERCMONO);
            break;

      case _VGA:
            _setvideomode(_VRES16COLOR);
            break;

      case _EGA:
            _setvideomode(_ERESCOLOR);
            break;

      case _CGA:
            _setvideomode(_HRESBW);
            break;

      case _MCGA:
            _setvideomode(_VRES16COLOR);
            break;

      default:
            siren( 3 );
            rotulo("IMPOSIBLE utilizar la tarjeta para video instalada");
            delay();
            exit(1);
      }
```

```
_getvideoconfig( &config );
}


/* ---------------------------------------------------------------------
   Función:         cleanup()
   Módulo:          VIDEO.C
   Principal:       REFLEX.C

   Parámetros:      (ninguno)

   Retorno:         (ninguno)

   Variables:       config  Estructura de parámetros gráficos

   Descripción:     Retorna a modo gráfico de texto
   ---------------------------------------------------------------------
*/
void cleanup()
{
_clearscreen(_GCLEARSCREEN);
_setvideomode(_DEFAULTMODE);
}

/* ---------------------------------------------------------------------
   Función:         graf_sim()
   Módulo:          VIDEO.C
   Principal:       REFLEX.C

   Parámetros:
   (entrada)        equipo  Indicador de modo utilizado
   (entrada)        ab      Relación Ó/β
   (entrada)        x       Vector de datos de distancia
   (entrada)        y       Vector de datos de voltaje simulado
   (entrada)        ycc     Vector de datos de voltaje con Zcc
   (entrada)        f       Frecuencia de trabajo en MHz
   (entrada)        alfa    Constante de atenuación
   (entrada)        vmin    Voltaje mínimo
   (entrada)        d1      Distancia d1
   (entrada)        d2      Distancia d2
   (entrada)        Zr      Impedancia de carga

   Retorno:         (ninguno)

   Variables:       config  Estructura de parámetros gráficos
                    env     Estructura del entorno del gráfico
                    datos   Estructura de parámetros gráficos
                    xvalue  Vector de datos de distancia
                    yvalue  Vector de datos de voltaje
                    label   Arreglo de etiquetas de información
                    i       Contador
                    control Caracter de control de flujo

   Descripción:     Presentación de resultados gráfico de magnitud
   ---------------------------------------------------------------------
*/
```

```
void graf_sim(  int equipo, double ab,  double *x, double  *y, double *ycc,
double f, double alfa, double vmin, double d1, double d2, struct complex Zr
)
{
struct videoconfig config;
chartenv env;
palettetype datos;
float xvalue[2*DATOS];
float yvalue[2*DATOS];
int i;
char resultados[80] = "f = ";
char fc[46], alfac[19], vminc[16], d1c[18], d2c[18], zr[35];
char control;
char *label[] =
      {
      "     Impedancia de carga:",
      "                                    ",
      };

_getvideoconfig( &config );

for( i=0; i<DATOS; i++ )
      {
      xvalue[i] = xvalue[DATOS+i] = x[i];
      yvalue[i] = ycc[i];
      yvalue[DATOS+i] = y[i];
      }

if(  Zr.x == 0 && Zr.y == 0 )
      if( equipo == 1 )
            strcpy( zr, "  zr = 0.00 + j 0.00 (ohmios)");
      else
            strcpy( zr, "  Zr = 0.00 + j 0.00 (ohmios)");
else
      {
      if( Zr.x == 1E50 && Zr.y == 1E50 )
            strcpy( zr, "  Zr = Valor infinito" );
      else
            {
            if( equipo == 1 )
                  {
                  if( Zr.y > 0 )
                        sprintf(zr,"  zr = %5.3f + j %5.3f", Zr.x, Zr.y);
                  else
                        sprintf(zr,"zr = %5.3f - j %5.3f",Zr.x,fabs(Zr.y));
                  }
            else
                  {
                  if( Zr.y > 0 )
                        sprintf( zr, "  Zr = %6.2f + j %6.2f (ohmios)",
                        Zr.x, Zr.y );
                  else
                        sprintf( zr, "  Zr = %6.2f - j %6.2f (ohmios)",
                        Zr.x, fabs( Zr.y ) );
                  }
            }
```

```
      }

strcpy( label[1], zr );

if( ab < 0.01 )
      {
      if( equipo == 1 )
            {
            sprintf( fc, "%4.2f GHz, Emax = 1.000, ", ( f / 1E3 ) );
            sprintf( vminc, "Emin = %4.3f, ", vmin );
            if(  Zr.x == 0 && Zr.y == 0 )
                  {
                  sprintf( d1c, "d1 = 0.0000 m, " );
                  sprintf( d2c, "d2 = %5.4f m ", ( d1 + d2 ) );
                  }
            else
                  {
                  sprintf( d1c, "d1 = %5.4f m, ", d1 );
                  sprintf( d2c, "d2 = %5.4f m ", d2 );
                  }
            }
      else
            {
            sprintf( fc, "%6.2f MHz, Vmax = 1.000, ", f );
            sprintf( vminc, "Vmin = %4.3f, ", vmin );
            if(  Zr.x == 0 && Zr.y == 0 )
                  {
                  sprintf( d1c, "d1 = 0.000 m, " );
                  sprintf( d2c, "d2 = %4.3f m ", ( d1 + d2 ) );
                  }
            else
                  {
                  sprintf( d1c, "d1 = %4.3f m, ", d1 );
                  sprintf( d2c, "d2 = %4.3f m ", d2 );
                  }
            }
      strcat( resultados, fc );
      strcat( resultados, vminc );

      if( vmin <= 0.97 )
            {
            strcat( resultados, d1c );
            strcat( resultados, d2c );
            }
      }
else
      {
      sprintf( fc, "%6.2f MHz, cte. atenuación = ", f );
      sprintf( alfac, "%6.5f nepper/m", alfa );
      strcat( resultados, fc );
      strcat( resultados, alfac );
      }

setup();
_pg_initchart();
_pg_getpalette( datos );
```

```
_pg_defaultchart( &env, _PG_SCATTERCHART, _PG_POINTANDLINE );

if( equipo == 1 )
      {
      strcpy( env.maintitle.title, " PATRON DE ONDA ESTACIONARIA - MODO
      FUNDAMENTAL " );
      strcpy( env.yaxis.axistitle.title, "Campo eléctrico normalizado" );
      }
else
      {
      strcpy(env.maintitle.title,"PATRON DE ONDA ESTACIONARIA - MODO TEM");
      strcpy( env.yaxis.axistitle.title, "Voltaje normalizado" );
      }
env.maintitle.justify = _PG_CENTER;
strcpy( env.subtitle.title, resultados );
env.subtitle.justify = _PG_CENTER;
env.yaxis.grid = TRUE;
env.xaxis.grid = TRUE;
strcpy( env.xaxis.axistitle.title, "Distancia desde la carga [metros]");
env.chartwindow.border = FALSE;
env.xaxis.gridstyle = env.yaxis.gridstyle = 4;
env.legend.legend = TRUE;
env.legend.place = _PG_BOTTOM;
env.legend.autosize = TRUE;
datos[1].plotchar = ' ';
datos[2].plotchar = ' ';

if( vmin > 0.97 )
      {
      env.yaxis.autoscale = FALSE;
      env.yaxis.scalemin = 0;
      env.yaxis.scalemax = 1.2;
      env.yaxis.scalefactor = 0.2;
      env.yaxis.ticinterval = 0.2;
      env.yaxis.ticdecimals = 1;
      }

if( config.adapter == _VGA )
      {
      env.maintitle.titlecolor = 15;
      env.subtitle.titlecolor = 1;
      env.chartwindow.background = 5;
      env.datawindow.background = 4;
      }

set_bottom( &env.chartwindow.x2, &env.chartwindow.y2 );

_pg_setpalette( datos );

if( _pg_chartscatterms( &env, xvalue, yvalue, 2, DATOS, DATOS, label ) )
      {
      cleanup();
      rotulo("IMPOSIBLE mostrar gráfico, memoria insuficiente !");
      delay();
      }
else
```

```
      {
      _settextposition(30, 1);
      _outtext("Desea impresión del gráfico? <S/N>");
      control = getch();
      if (control == 'S' || control == 's')
           {
           _settextposition(30, 1);
           _outtext(" ");
           _settextposition(30, 1);
           _outtext("Está la impresora encendida y en ON LINE? <S/N>");
           control = getch();
           if (control == 'S' || control == 's')
               imp_graf( env.chartwindow.x2, env.chartwindow.y2 );
           }
      _settextposition(30, 1);
      _outtext("                                                    ");
      _settextposition(30, 1);
      _outtext("Presione <ENTER> para continuar");
      enter_key();
      cleanup();
      }
}

/* ---------------------------------------------------------------------
   Función:        graf_fase()
   Módulo:         VIDEO.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       equipo  Indicador de modo utilizado
   (entrada)       x       Vector de datos de distancia
   (entrada)       fi      Vector de datos de fase
   (entrada)       f       Frecuencia de trabajo en MHz
   (entrada)       alfa    Constante de atenuación
   (entrada)       Zr      Impedancia de carga

   Retorno:        (ninguno)

   Variables:      config  Estructura de parámetros gráficos
                   env     Estructura del entorno del gráfico
                   datos   Estructura de parámetros gráficos
                   xvalue  Vector de datos de distancia
                   fvalue  Vector de datos de fase
                   label   Arreglo de etiquetas de información
                   i       Contador
                   control Caracter de control de flujo

   Descripción:    Presentación de resultados gráfico de fase
   --------------------------------------------------------------------
*/
void graf_fase( int  equipo, double *x, double *fi,  double f, double alfa,
struct complex Zr )
{
struct videoconfig config;
chartenv env;
palettetype datos;
```

```
float xvalue[2*DATOS];
float fvalue[2*DATOS];
int i;
char resultados[80] = "f = ";
char control;
char fc[46], alfac[19], zr[35];
char *label[] =
      {
      "     Impedancia de carga:",
      "                                    ",
      };

_getvideoconfig( &config );

for( i=0; i<DATOS; i++ )
      {
      xvalue[i] = xvalue[DATOS+i] = x[i];
      fvalue[i] = 0;
      fvalue[DATOS+i] = fi[i];
      }

if(  Zr.x == 0 && Zr.y == 0 )
      if( equipo == 1 )
            strcpy( zr, "  zr = 0.00 + j 0.00 (ohmios)");
      else
            strcpy( zr, "  Zr = 0.00 + j 0.00 (ohmios)");
else
      {
      if( Zr.x == 1E50 && Zr.y == 1E50 )
            strcpy( zr, "  Zr = Valor infinito" );
      else
            {
            if( equipo == 1 )
                  {
                  if( Zr.y > 0 )
                        sprintf(zr,"  zr = %5.3f + j %5.3f", Zr.x, Zr.y);
                  else
                        sprintf(zr,"zr = %5.3f - j %5.3f",Zr.x,fabs(Zr.y));
                  }
            else
                  {
                  if( Zr.y > 0 )
                        sprintf( zr, "  Zr = %6.2f + j %6.2f (ohmios)",
                        Zr.x, Zr.y );
                  else
                        sprintf( zr, "  Zr = %6.2f - j %6.2f (ohmios)",
                        Zr.x, fabs( Zr.y ) );
                  }
            }
      }

strcpy( label[1], zr );

if( equipo == 1 )
       {
       sprintf( fc, "%4.2f GHz, atenuación despreciable", ( f / 1E3 ) );
```

```
        strcat( resultados, fc );
        }
else
        {
        sprintf( fc, "%6.2f MHz, cte. atenuación = ", f );
        sprintf( alfac, "%6.5f nepper/m", alfa );
        strcat( resultados, fc );
        strcat( resultados, alfac );
        }

setup();
_pg_initchart();
_pg_getpalette( datos );
_pg_defaultchart( &env, _PG_SCATTERCHART, _PG_POINTANDLINE );

if( equipo == 1 )
      strcpy(env.maintitle.title, "VARIACION DE FASE - MODO FUNDAMENTAL");
else
      strcpy( env.maintitle.title, " VARIACION DE FASE - MODO TEM " );

env.maintitle.justify = _PG_CENTER;
strcpy( env.subtitle.title, resultados );
env.subtitle.justify = _PG_CENTER;
env.yaxis.grid = TRUE;
env.xaxis.grid = TRUE;
strcpy( env.yaxis.axistitle.title, "Fase [grados]" );
strcpy( env.xaxis.axistitle.title, "Distancia desde la carga [metros]" );
env.chartwindow.border = FALSE;
env.xaxis.gridstyle = env.yaxis.gridstyle = 4;
env.yaxis.autoscale = FALSE;
env.yaxis.scalemin = -90;
env.yaxis.scalemax = 90;
env.yaxis.scalefactor = 15;
env.yaxis.ticinterval = 15;
env.legend.legend = TRUE;
env.legend.place = _PG_BOTTOM;
env.legend.autosize = TRUE;
datos[1].plotchar = ' ';
datos[2].plotchar = ' ';

if(  Zr.x == 0 && Zr.y == 0 )
      {
      env.yaxis.scalemin = 0;
      env.yaxis.scalemax = 100;
      env.yaxis.scalefactor = 10;
      env.yaxis.ticinterval = 10;
      }

if( config.adapter == _VGA )
      {
      env.maintitle.titlecolor = 15;
      env.subtitle.titlecolor = 1;
      env.chartwindow.background = 5;
      env.datawindow.background = 4;
      }
```

```
set_bottom( &env.chartwindow.x2, &env.chartwindow.y2 );
_pg_setpalette( datos );

if( _pg_chartscatterms( &env, xvalue, fvalue, 2, DATOS, DATOS, label ) )
      {
      cleanup();
      rotulo("IMPOSIBLE mostrar gráfico, memoria insuficiente !");
      delay();
      }
else
      {
      _settextposition(30, 1);
      _outtext("Desea impresión del gráfico? <S/N>");
      control = getch();
      if (control == 'S' || control == 's')
           {
           _settextposition(30, 1);
           _outtext(" ");
           _settextposition(30, 1);
           _outtext("Está la impresora encendida y en ON LINE? <S/N>");
           control = getch();
           if (control == 'S' || control == 's')
               imp_graf( env.chartwindow.x2, env.chartwindow.y2 );
           }
      _settextposition(30, 1);
      _outtext("                                                    ");
      _settextposition(30, 1);
      _outtext("Presione <ENTER> para continuar");
      enter_key();
      cleanup();
      }
}

/* ---------------------------------------------------------------------
   Función:        graf_sim_t()
   Módulo:         VIDEO.C
   Principal:      REFLEX.C

   Parámetros:
   (entrada)       fuente  Indicador de fuente simulada
   (entrada)       equipo  Indicador de modo utilizado
   (entrada)       f       Frecuencia de trabajo en MHz
   (entrada)       Zr      Impedancia de carga
   (entrada)       l       Distancia entre fuente y carga
   (entrada)       x       Distancia de análisis
   (entrada)       v       Vector de datos de voltaje
   (entrada)       t       Vector de datos de tiempo

   Retorno:        (ninguno)

   Variables:      config  Estructura de parámetros gráficos
                   env     Estructura del entorno del gráfico
                   datos   Estructura de parámetros gráficos
                   vvalue  Vector de datos de voltaje
                   tvalue  Vector de datos de tiempo
                   label   Arreglo de etiquetas de información
```

```
                    i       Contador
                    control Caracter de control de flujo

   Descripción:     Presentación de resultado gráfico en el tiempo
   ------------------------------------------------------------------------
*/
void  graf_sim_t(  int fuente,  int equipo,  double f,  struct  complex Zr,
double l, double x, double *v, double *t )
{
struct videoconfig config;
chartenv env;
palettetype datos;
float vvalue[2*DATOS1];
float tvalue[2*DATOS1];
int i;
char resultados[80] = " ";
char fc[46], lc[18], xc[18], zr[35];
char control;
char *label[] =
      {
      "     Impedancia de carga:",
      "                                    ",
      };
_getvideoconfig( &config );

for( i=0; i<DATOS1; i++ )
      {
      tvalue[i] = tvalue[DATOS1+i] = (float) t[i] / 1E-9;
      vvalue[i] = 0;
      vvalue[DATOS1+i] = (float) v[i];
      }

if(  Zr.x == 0 && Zr.y == 0 )
      if( equipo == 1 )
            strcpy( zr, "  zr = 0.00 + j 0.00 (ohmios)");
      else
            strcpy( zr, "  Zr = 0.00 + j 0.00 (ohmios)");
else
      {
      if( Zr.x == 1E50 && Zr.y == 1E50 )
            strcpy( zr, "  Zr = Valor infinito" );
      else
            {
            if( equipo == 1 )
                  {
                  if( Zr.y > 0 )
                        sprintf(zr,"  zr = %5.3f + j %5.3f", Zr.x, Zr.y);
                  else
                        sprintf(zr,"zr = %5.3f-j %5.3f",Zr.x,fabs(Zr.y ));
                  }
            else
                  {
                  if( Zr.y > 0 )
                        sprintf( zr, "  Zr = %6.2f + j %6.2f (ohmios)",
                        Zr.x, Zr.y );
                  else
```

```
                        sprintf( zr, "  Zr = %6.2f - j %6.2f (ohmios)",
                        Zr.x, fabs( Zr.y ) );
                    }
                }
            }

strcpy( label[1], zr );

if( fuente == 2 )
      sprintf( fc, "Fuente continua, ", f );
else
      if( equipo == 1 )
            sprintf( fc, "f = %4.2f GHz, ", f );
      else
            sprintf( fc, "f = %6.2f MHz, ", f );

sprintf( lc, "l = %4.1f cm, ", l );
sprintf( xc, "x = %4.1f cm ", x );

strcat( resultados, fc );
strcat( resultados, lc );
strcat( resultados, xc );

setup();
_pg_initchart();
_pg_getpalette( datos );
_pg_defaultchart( &env, _PG_SCATTERCHART, _PG_POINTANDLINE );

if( equipo == 1 )
      {
      strcpy( env.maintitle.title, " SEÑAL TRANSITORIA EN EL TIEMPO - MODO
      FUNDAMENTAL " );
      strcpy( env.yaxis.axistitle.title, "Campo eléctrico normalizado" );
      }
else
      {
      strcpy( env.maintitle.title, " SEÑAL TRANSITORIA EN EL TIEMPO - MODO
      TEM " );
      strcpy( env.yaxis.axistitle.title, "Voltaje normalizado" );
      }

env.maintitle.justify = _PG_CENTER;
strcpy( env.subtitle.title, resultados );
env.subtitle.justify = _PG_CENTER;
env.yaxis.grid = TRUE;
env.xaxis.grid = TRUE;
strcpy( env.xaxis.axistitle.title, "Tiempo de análisis [nanosegundos]");
env.chartwindow.border = FALSE;
env.xaxis.gridstyle = env.yaxis.gridstyle = 4;
env.legend.legend = TRUE;
env.legend.place = _PG_BOTTOM;
env.legend.autosize = TRUE;
datos[1].plotchar = ' ';
datos[2].plotchar = ' ';

if( Zr.x == 0 && Zr.y == 0 && fuente == 2 )
```

```
      {
      env.yaxis.autoscale = FALSE;
      env.yaxis.scalemin = 0;
      env.yaxis.scalemax = 1.2;
      env.yaxis.scalefactor = 0.2;
      env.yaxis.ticinterval = 0.2;
      env.yaxis.ticdecimals = 1;
      }

/*if( Zr.y == 0 && fuente == 2 )
      {
      env.yaxis.autoscale = FALSE;
      env.yaxis.scalemin = 0;
      env.yaxis.scalemax = 1.2;
      env.yaxis.scalefactor = 0.2;
      env.yaxis.ticinterval = 0.2;
      env.yaxis.ticdecimals = 1;
      }                          */

if( config.adapter == _VGA )
      {
      env.maintitle.titlecolor = 15;
      env.subtitle.titlecolor = 1;
      env.chartwindow.background = 5;
      env.datawindow.background = 4;
      }

set_bottom( &env.chartwindow.x2, &env.chartwindow.y2 );

_pg_setpalette( datos );

if( _pg_chartscatterms( &env, tvalue, vvalue, 2, DATOS1, DATOS1, label ) )
      {
      cleanup();
      rotulo("IMPOSIBLE mostrar gráfico, memoria insuficiente !");
      delay();
      }
else
      {
      _settextposition(30, 1);
      _outtext("Desea impresión del gráfico? <S/N>");
      control = getch();
      if (control == 'S' || control == 's')
           {
           _settextposition(30, 1);
           _outtext(" ");
           _settextposition(30, 1);
           _outtext("Está la impresora encendida y en ON LINE? <S/N>");
           control = getch();
           if (control == 'S' || control == 's')
               imp_graf( env.chartwindow.x2, env.chartwindow.y2 );
           }
      _settextposition(30, 1);
      _outtext("                                                    ");
      _settextposition(30, 1);
      _outtext("Presione <ENTER> para continuar");
```

```
        enter_key();
        cleanup();
        }
}

/* ---------------------------------------------------------------------
   Función:         set_bottom()
   Módulo:          VIDEO.C
   Principal:       REFLEX.C

   Parámetros:
   (salida)         a        Coordenada derecha de la ventana
   (salida)         b        Coordenada inferior de la ventana

   Retorno:         (ninguno)

   Variables:       config   Estructura de parámetros gráficos

   Descripción:     De acuerdo al tipo de monitor, fija la posición
                    borde inferior de la ventana principal del gráfico
   --------------------------------------------------------------------
*/
void set_bottom( short *a, short *b )
{
struct videoconfig config;
_getvideoconfig( &config );

switch( config.adapter )
        {
        case _HGC:
                *a = 719;
                *b = 15 * 348 / 16;
                break;

        case _VGA:
                *a = 639;
                *b = 15 * 480 / 16;
                break;

        case _EGA:
                *a = 639;
                *b = 15 * 350 / 16;
                break;

        case _CGA:
                *a = 639;
                *b = 15 * 200 / 16;
                break;

        case _MCGA:
                *a = 639;
                *b = 15 * 480 / 16;
                break;
        }
_getvideoconfig( &config );
}
```

```
/* ---------------------------------------------------------------------
   Nombre:          BOX.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para 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:          COMPLEX.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para COMPLEX.C
   ------------------------------------------------------------------
*/

#ifndef COMPLEX_DEFINED

char *cstr( struct complex, char * );
int strc( char *, struct complex * );
struct complex cadd( struct complex, struct complex );
struct complex csub( struct complex, struct complex );
struct complex cmul( struct complex, struct complex );
struct complex cdiv( struct complex, struct complex );
struct complex cpow( struct complex, struct complex );
struct complex croot( struct complex, struct complex );
struct complex cexp( struct complex );
struct complex clog( struct complex );
struct complex crec( struct complex );
struct complex csqr( struct complex );
struct complex complex_to_polar( struct complex );
struct complex polar_to_complex( struct complex );

#define COMPLEX_DEFINED
#endif
```

```
/* ------------------------------------------------------------------
   Nombre:          FUNCION.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para FUNCION.C
   ------------------------------------------------------------------
*/

#define PI 3.141592654
#define DATOS 750
#define DATOS1 800

struct complex impedancia_carga( int, double, double, double, struct
complex );
void reflexion( struct complex, struct complex, double *, double * );
void onda( int, double, double, double, double, double, double *, double *,
double *, double * );
void onda_t( int, int, double, double, struct complex, struct complex,
struct complex, double, double, double *, double * );
void result_sim( int, char **, double, double, double, double, double,
double, double, double, double, double, double, struct complex, struct
complex, double );
void result_zr( char **, double, struct complex, struct complex );
double maximo( int, double * );
double minimo( int, double * );
double distancia_1( double *, double *, int, double, int * );
double relacion( struct complex, double *, double *, double, double, int );
double k_propagacion( double, double, double *, double *, double * );
double resultados( struct complex, int, double *, double *, double, double
*, double *, double * );
double error( struct complex, struct complex, struct complex );
double porcentaje( double, double );
double kg( double, double, double *, double * );
double res_zr( double, double, double, double, double, double * );
```

```
/* ------------------------------------------------------------------
   Nombre:          GETKEY.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para 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

void enter_key();
unsigned int getkey( void );
unsigned int getkey_or_mouse( void );

#define GETKEY_DEFINED
#endif
```

```
/* ------------------------------------------------------------------
   Nombre:          INGRESO.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para INGRESO.C
   ------------------------------------------------------------------
*/
int ingreso( double *, double *, double *, double *, double *, double * );
int ingreso_guia( double *, double *, double *, double * );
int ing_t_l( int, double *, double *, double *,  double *, double *, double
*, double *, double *, double * );
int ing_t_g( double *, double *, double *, double *, double *, double * );
int ing_calculo( int, int, double *, double *, double *, double *, double *
);
int comprobar( char * );
int impresion( int, char ** );
void imp_graf( int, int );
```

```
/* ---------------------------------------------------------------------
   Nombre:          MENU.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para 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:          MOUSEFUN.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para 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_settcurs( 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 */

/* Default graphics mode cursor */
static struct graphics_cursor far gcursor_default =
    {

    /*  screen mask  */
    0xCFFF,     /* 1100111111111111 */
    0xC7FF,     /* 1100011111111111 */
    0xC3FF,     /* 1100001111111111 */
    0xC1FF,     /* 1100000111111111 */
    0xC0FF,     /* 1100000011111111 */
    0xC07F,     /* 1100000001111111 */
    0xC03F,     /* 1100000000111111 */
    0xC01F,     /* 1100000000011111 */
    0xC00F,     /* 1100000000001111 */
    0xC007,     /* 1100000000000111 */
    0xC07F,     /* 1100000001111111 */
    0xC43F,     /* 1100010000111111 */
    0xCC3F,     /* 1100110000111111 */
    0xFE1F,     /* 1111111000011111 */
    0xFE1F,     /* 1111111000011111 */
    0xFF1F,     /* 1111111100011111 */

    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x1000,     /* 0001000000000000 */
    0x1800,     /* 0001100000000000 */
    0x1C00,     /* 0001110000000000 */
    0x1E00,     /* 0001111000000000 */
    0x1F00,     /* 0001111100000000 */
    0x1F80,     /* 0001111110000000 */
    0x1FC0,     /* 0001111111000000 */
    0x1FE0,     /* 0001111111100000 */
    0x1F00,     /* 0001111100000000 */
    0x1B00,     /* 0001101100000000 */
    0x1180,     /* 0001000110000000 */
    0x0180,     /* 0000000110000000 */
    0x00C0,     /* 0000000011000000 */
    0x00C0,     /* 0000000011000000 */
    0x0000,     /* 0000000000000000 */
```

```
    /* hot spot x,y */
    02, 00
    };


/* Graphics mode cursor, pointing hand */
static struct graphics_cursor far gcursor_hand =
    {

    /*  screen mask  */
    0xE1FF,     /* 11100h0111111111 */
    0xE1FF,     /* 1110000111111111 */
    0xE1FF,     /* 1110000111111111 */
    0xE1FF,     /* 1110000111111111 */
    0xE1FF,     /* 1110000111111111 */
    0xE000,     /* 1110000000000000 */
    0xE000,     /* 1110000000000000 */
    0xE000,     /* 1110000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */


    /*  cursor mask  */
    0x1E00,     /* 00011H1000000000 */
    0x1200,     /* 0001001000000000 */
    0x1200,     /* 0001001000000000 */
    0x1200,     /* 0001001000000000 */
    0x1200,     /* 0001001000000000 */
    0x13FF,     /* 0001001111111111 */
    0x1249,     /* 0001001001001001 */
    0x1249,     /* 0001001001001001 */
    0xF249,     /* 1111001001001001 */
    0x9001,     /* 1001000000000001 */
    0x9001,     /* 1001000000000001 */
    0x9001,     /* 1001000000000001 */
    0x8001,     /* 1000000000000001 */
    0x8001,     /* 1000000000000001 */
    0x8001,     /* 1000000000000001 */
    0xFFFF,     /* 1111111111111111 */

    /* hot spot x,y */
    05, 00
    };
```

```
/* Graphics mode cursor, check mark */
static struct graphics_cursor far gcursor_check =
    {

    /*  screen mask  */
    0xFFF0,     /* 1111111111110000 */
    0xFFE0,     /* 1111111111100000 */
    0xFFC0,     /* 1111111111000000 */
    0xFF81,     /* 1111111110000001 */
    0xFF03,     /* 1111111100000011 */
    0x0607,     /* 0000011000000111 */
    0x000F,     /* 0000000000001111 */
    0x001F,     /* 0000000000011111 */
    0xC03F,     /* 1100000000111111 */
    0xF07F,     /* 1111000001111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */

    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x0006,     /* 0000000000000110 */
    0x000C,     /* 0000000000001100 */
    0x0018,     /* 0000000000011000 */
    0x0030,     /* 0000000000110000 */
    0x0060,     /* 0000000001100000 */
    0x70C0,     /* 0111000011000000 */
    0x1D80,     /* 0001110110000000 */
    0x0700,     /* 0000011100000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */

    /* hot spot x,y */
    06, 07
    };


/* Graphics mode cursor, hour glass */
static struct graphics_cursor far gcursor_hour =
    {

    /*  screen mask  */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x8001,     /* 1000000000000001 */
    0xC003,     /* 1100000000000011 */
    0xE007,     /* 1110000000000111 */
```

```
    0xF00F,     /* 1111000000001111 */
    0xE007,     /* 1110000000000111 */
    0xC003,     /* 1100000000000011 */
    0x8001,     /* 1000000000000001 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */

    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x7FFE,     /* 0111111111111110 */
    0x6006,     /* 0110000000000110 */
    0x300C,     /* 0011000000001100 */
    0x1818,     /* 0001100000011000 */
    0x0C30,     /* 0000110000110000 */
    0x0660,     /* 0000011001100000 */
    0x03C0,     /* 0000001111000000 */
    0x0660,     /* 0000011001100000 */
    0x0C30,     /* 0000110000110000 */
    0x1998,     /* 0001100110011000 */
    0x33CC,     /* 0011001111001100 */
    0x67E6,     /* 0110011111100110 */
    0x7FFE,     /* 0111111111111110 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */

    /* hot spot x,y */
    07, 07
    };


/* Graphics mode cursor, jet aircraft */
static struct graphics_cursor far gcursor_jet =
    {
        :
    /*  screen mask  */
    0xFFFF,     /* 1111111111111111 */
    0xFEFF,     /* 1111111011111111 */
    0xFC7F,     /* 1111110001111111 */
    0xF83F,     /* 1111100000111111 */
    0xF83F,     /* 1111100000111111 */
    0xF83F,     /* 1111100000111111 */
    0xF01F,     /* 1111000000011111 */
    0xE00F,     /* 1110000000001111 */
    0xC007,     /* 1100000000000111 */
    0x8003,     /* 1000000000000011 */
    0x8003,     /* 1000000000000011 */
    0xF83F,     /* 1111100000111111 */
    0xF83F,     /* 1111100000111111 */
    0xF01F,     /* 1111000000011111 */
    0xE00F,     /* 1110000000001111 */
    0xFFFF,     /* 1111111111111111 */
```

```
    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0100,     /* 0000000100000000 */
    0x0380,     /* 0000001110000000 */
    0x0380,     /* 0000001110000000 */
    0x0380,     /* 0000001110000000 */
    0x07C0,     /* 0000011111000000 */
    0x0FE0,     /* 0000111111100000 */
    0x1FF0,     /* 0001111111110000 */
    0x3FF8,     /* 0011111111111000 */
    0x638C,     /* 0110001110001100 */
    0x0380,     /* 0000001110000000 */
    0x0380,     /* 0000001110000000 */
    0x07C0,     /* 0000011111000000 */
    0x0C60,     /* 0000110001100000 */
    0x0000,     /* 0000000000000000 */

    /* hot spot x,y */
    07, 01
    };


/* Graphics mode cursor, left pointing arrow */
static struct graphics_cursor far gcursor_left =
    {

    /*  screen mask  */
    0xFE1F,     /* 1111111000011111 */
    0xF01F,     /* 1111000000011111 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0xF01F,     /* 1111000000011111 */
    0xFE1F,     /* 1111111000011111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */

    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x00C0,     /* 0000000011000000 */
    0x07C0,     /* 0000011111000000 */
    0x7FFE,     /* 0111111111111110 */
    0x07C0,     /* 0000011111000000 */
    0x00C0,     /* 0000000011000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
```

```
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */

    /* hot spot x,y */
    00, 03
    };


/* Graphics mode cursor, plus sign */
static struct graphics_cursor far gcursor_plus =
    {

    /*  screen mask  */
    0xFC3F,     /* 1111110000111111 */
    0xFC3F,     /* 1111110000111111 */
    0xFC3F,     /* 1111110000111111 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0xFC3F,     /* 1111110000111111 */
    0xFC3F,     /* 1111110000111111 */
    0xFC3F,     /* 1111110000111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */

    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x0180,     /* 0000000110000000 */
    0x0180,     /* 0000000110000000 */
    0x0180,     /* 0000000110000000 */
    0x7FFE,     /* 0111111111111110 */
    0x0180,     /* 0000000110000000 */
    0x0180,     /* 0000000110000000 */
    0x0180,     /* 0000000110000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */

    /* hot spot x,y */
    07, 04
    };
```

```
/* Graphics mode cursor, up pointing arrow */
static struct graphics_cursor far gcursor_up =
    {

    /*  screen mask  */
    0xF9FF,     /* 1111100111111111 */
    0xF0FF,     /* 1111000011111111 */
    0xE07F,     /* 1110000001111111 */
    0xE07F,     /* 1110000001111111 */
    0xC03F,     /* 1100000000111111 */
    0xC03F,     /* 1100000000111111 */
    0x801F,     /* 1000000000011111 */
    0x801F,     /* 1000000000011111 */
    0x000F,     /* 0000000000001111 */
    0x000F,     /* 0000000000001111 */
    0xF0FF,     /* 1111000011111111 */
    0xF0FF,     /* 1111000011111111 */
    0xF0FF,     /* 1111000011111111 */
    0xF0FF,     /* 1111000011111111 */
    0xF0FF,     /* 1111000011111111 */
    0xF0FF,     /* 1111000011111111 */

    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x0600,     /* 0000011000000000 */
    0x0F00,     /* 0000111100000000 */
    0x0F00,     /* 0000111100000000 */
    0x1F80,     /* 0001111110000000 */
    0x1F80,     /* 0001111110000000 */
    0x3FC0,     /* 0011111111000000 */
    0x3FC0,     /* 0011111111000000 */
    0x7FE0,     /* 0111111111100000 */
    0x0600,     /* 0000011000000000 */
    0x0600,     /* 0000011000000000 */
    0x0600,     /* 0000011000000000 */
    0x0600,     /* 0000011000000000 */
    0x0600,     /* 0000011000000000 */
    0x0600,     /* 0000011000000000 */
    0x0000,     /* 0000000000000000 */

    /* hot spot x,y */
    05, 00
    };


/* Graphics mode cursor, X mark */
static struct graphics_cursor far gcursor_x =
    {

    /*  screen mask  */
    0x07E0,     /* 0000011111100000 */
    0x0180,     /* 0000000110000000 */
    0x0000,     /* 0000000000000000 */
    0xC003,     /* 1100000000000011 */
    0xF00F,     /* 1111000000001111 */
    0xC003,     /* 1100000000000011 */
```

```
    0x0000,     /* 0000000000000000 */
    0x0180,     /* 0000000110000000 */
    0x03C0,     /* 0000001111000000 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */
    0xFFFF,     /* 1111111111111111 */

    /*  cursor mask  */
    0x0000,     /* 0000000000000000 */
    0x700E,     /* 0111000000001110 */
    0x1C38,     /* 0001110000111000 */
    0x0660,     /* 0000011001100000 */
    0x03C0,     /* 0000001111000000 */
    0x0660,     /* 0000011001100000 */
    0x1C38,     /* 0001110000111000 */
    0x700E,     /* 0111000000001110 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */
    0x0000,     /* 0000000000000000 */

    /* hot spot x,y */
    07, 04
    };

#define MOUSEFUN_DEFINED
#endif
```

```
/* ------------------------------------------------------------------------
   Nombre:          SOUND.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para SOUND.C
   ------------------------------------------------------------------------
*/

#ifndef SOUND_DEFINED

void sound( int );
void silence( void );
void speaker_toggle( void );
void wait_ticks( unsigned int );
void warble( int );
void weird( int );
void siren( int );
void white_noise( int );
void note( int, int );

#define SOUND_DEFINED
#endif
```

```
/* ---------------------------------------------------------------------
   Nombre:          T_COLORS.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Definición de colores para modo de texto
   ---------------------------------------------------------------------
*/

#ifndef T_COLORS_DEFINED

/* Standard text mode colors */
#define T_BLACK 0
#define T_BLUE 1
#define T_GREEN 2
#define T_CYAN 3
#define T_RED 4
#define T_MAGENTA 5
#define T_BROWN 6
#define T_WHITE 7

/* Modifiers that can be added to the text mode color constants */
#define T_BRIGHT 8
#define T_BLINK 16

/* Common combinations */
#define T_GRAY ( T_BLACK | T_BRIGHT )
#define T_YELLOW ( T_BROWN | T_BRIGHT )

/* Background text mode color constants */
#define BK_BLACK 0L
#define BK_BLUE 1L
#define BK_GREEN 2L
#define BK_CYAN 3L
#define BK_RED 4L
#define BK_MAGENTA 5L
#define BK_BROWN 6L
#define BK_WHITE 7L

#define T_COLORS_DEFINED
#endif
```

```
/* ------------------------------------------------------------------
   Nombre:          VIDEO.H
   Tipo:            Include
   Lenguaje:        Microsoft QuickC
   Principal:       REFLEX.C
   Descripción:     Prototipos y definiciones para VIDEO.C
   ------------------------------------------------------------------
*/

#define CUR_MOV(row,col) printf("\033[%d;%dH",row,col)
#define CLR_SCREEN printf("\033[2J")
#define DATOS 750
#define DATOS1 800

void rotulo( char * );
void demora();
void setup();
void cleanup();
void graf_sim( int, double, double  *, double *, double *,  double, double,
double, double, double, struct complex );
void graf_sim_t( int, int,  double, struct complex, double, double,  double
*, double * );
void graf_fase( int, double *, double *, double, double, struct complex );
void set_bottom( short *, short * );
```

MANUAL DE USO DEL PROGRAMA DESARROLLADO

El archivo ejecutable del programa tiene el nombre de REFLEX.EXE. A continuación se detallará los requerimientos tanto de hardware como de software para la ejecución correcta del programa, así como se explicará la ejecución del mismo.

1. REQUERIMIENTOS DE HARDWARE PARA LA EJECUCION DE REFLEX.EXE.

Para poder ejecutar el programa REFLEX.EXE, se necesita:

- Un computador IBM o compatible, con un microprocesador 80286, o superior.
- 640 Kb. o más de memoria RAM.
- Tarjeta gráfica de video (CGA, EGA, MCGA, VGA o Hércules).
- Un mouse (no necesario pero si recomendable).
- Una impresora paralela de matriz de puntos, inicializada en modo gráfico.

2. REQUERIMIENTOS DE SOFTWARE PARA LA EJECUCION DE REFLEX.EXE.

Se requiere para la ejecución del programa el Sistema Operativo MS-DOS versión 3.1 o superior (se recomienda usar MS-DOS 6.0).

Antes de ejecutar el programa REFLEX.EXE, se debe asegurar que los archivos AUTOEXEC.BAT y CONFIG.SYS contengan los archivos necesarios para que los gráficos y textos se presenten correctamente.

Se debe utilizar el archivo MSHERC.COM con al finalidad de emular la tarjeta gráfica cuando se use un monitor Hércules. Si la ejecución se la realiza desde un diskette, es necesario crear los archivos anteriormente mencionados.

2.1 Ejecución del programa REFLEX.EXE desde diskette.

El diskette debe contener los siguientes archivos para la ejecución correcta del programa:

AUTOEXEC.BAT

CONFIG.SYS

MSHERC.COM

GRAPHICS.COM

GRAPHICS.PRO

REFLEX.EXE

El archivo AUTOEXEC.BAT debe contener:

```
@ECHO OFF
PROMPT $p$g
GRAPHICS.COM
MSHERC.COM
REFLEX.EXE
CLS
```

El archivo CONFIG.SYS debe ser formado como se indica:

```
BUFFERS=30
FILES=30
DEVICE=ANSI.SYS
```

2.2 Ejecución del programa REFLEX.EXE desde el disco duro.

El archivo AUTOEXEC.BAT debe contener las siguientes sentencias:

```
C:\DOS\GRAPHICS.COM
C:\DOS\MSHERC.COM
```

En el archivo CONFIG.SYS se debe añadir lo siguiente:

```
DEVICE=C:\DOS\ANSI.SYS
```

Después de incluír en los archivos mencionados las sentencias indicadas, se debe copiar el archivo REFLEX.EXE en el disco duro y ejecutarlo como sigue:

```
C:\REFLEX  ( <——┘ )
```

3. MANEJO DEL PROGRAMA REFLEX.EXE.

En la Figura A.1 se indica la carátula de presentación del programa. Para seguir con la ejecución del mismo, se debe seguir la instrucción indicada (presione una tecla para continuar).

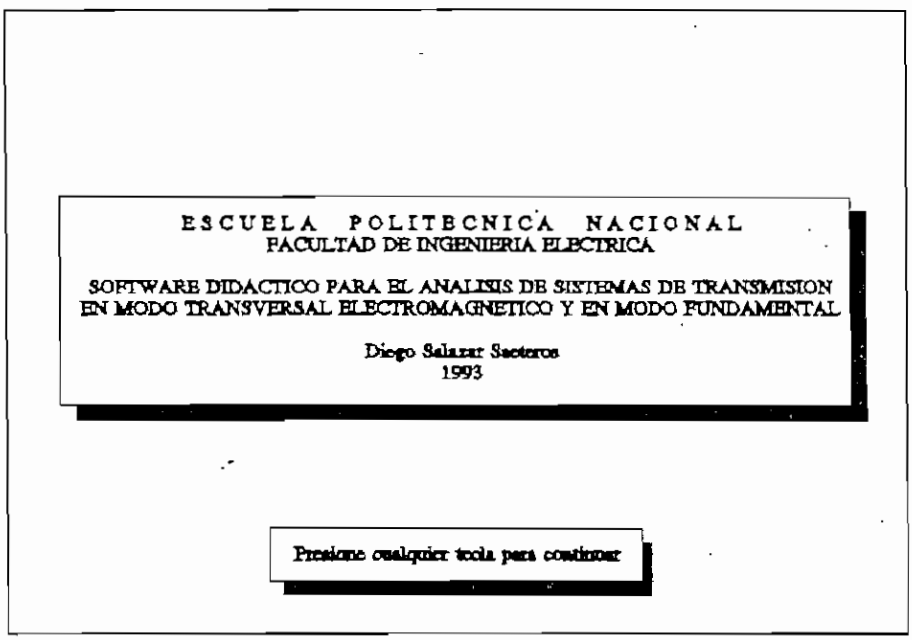

Figura A.1. Carátula de presentación.

A continuación, el programa presenta una pantalla donde se puede seleccionar el modo de propagación que se desea analizar. La Figura A.2 representa la pantalla antes mencionada.

Al seleccionar cualquiera de los modos de propagación, inmediatamente se presenta la pantalla de selección general, con los distintos items del programa que se analizarán más adelante.

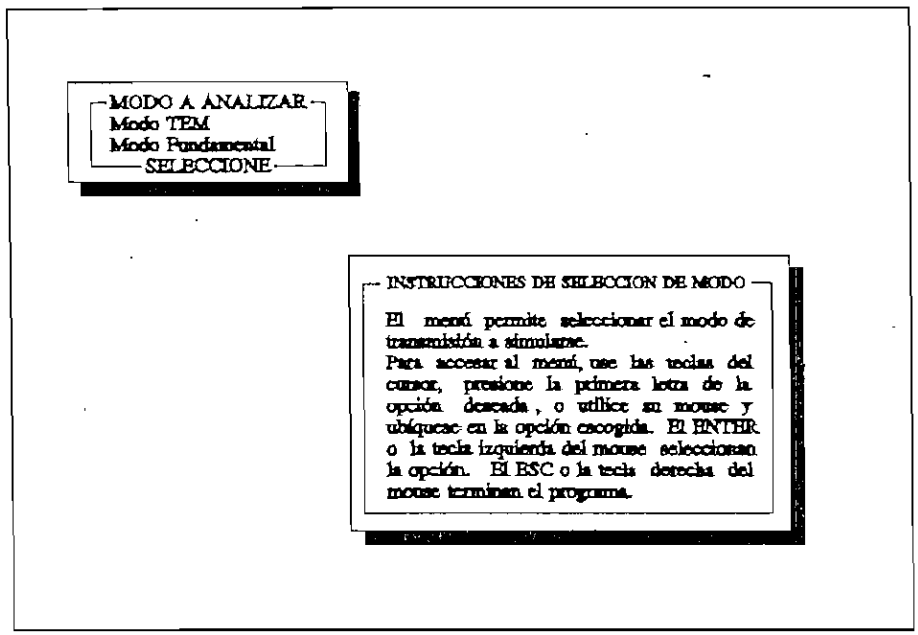

Figura A.2. Pantalla de selección de modo de propagación.

En la Figura A.3 se puede observar la pantalla de selección general.

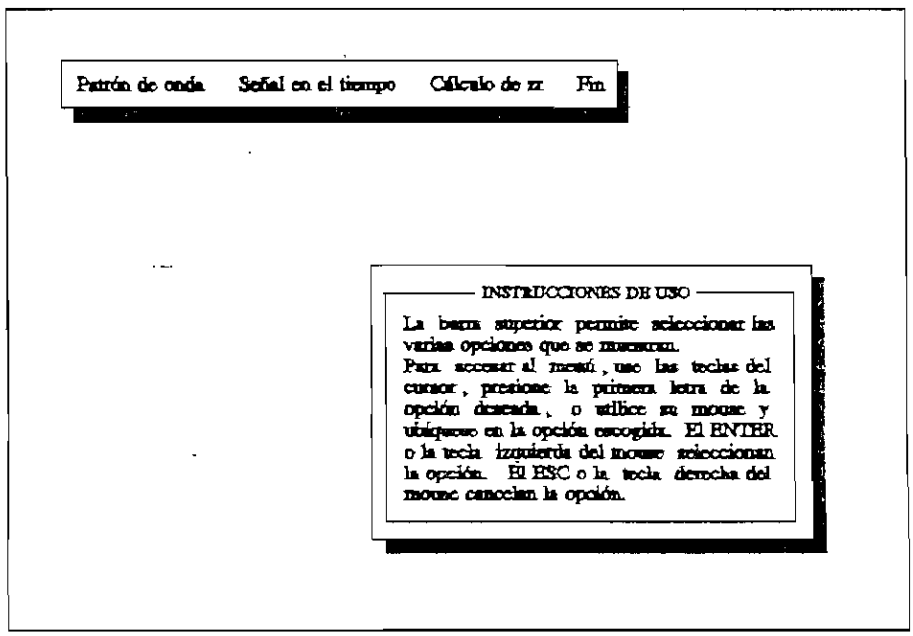

Figura A.3. Pantalla de selección general.

3.1 Patrón de onda estacionaria.

Dentro de la pantalla de selección general, se puede escoger la opción *Patrón de onda*, la misma que ofrece los items *Ingreso de datos*, los cuales se analizan en el programa para de esta manera permitir el acceso a las opciones *Resultados* y *Gráficos*. En la Figura A.4 se observa la pantalla que resume lo anteriormente mencionado.

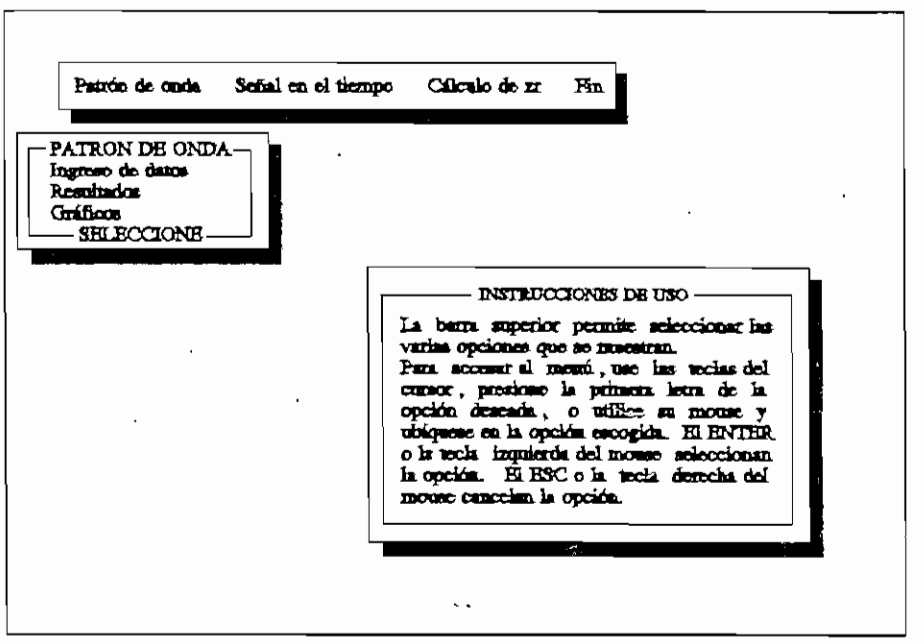

Figura A.4. Pantalla de selección para el patrón de onda.

En el ingreso de datos, se presenta la pantalla correspondiente a los datos necesarios para la simulación del patrón de onda estacionaria para el modo ya escogido. Las Figuras A.5 y A.6 muestran las pantallas de ingreso de datos para la simulación del patrón de onda estacionaria para el modo TEM y para el modo fundamental respectivamente.

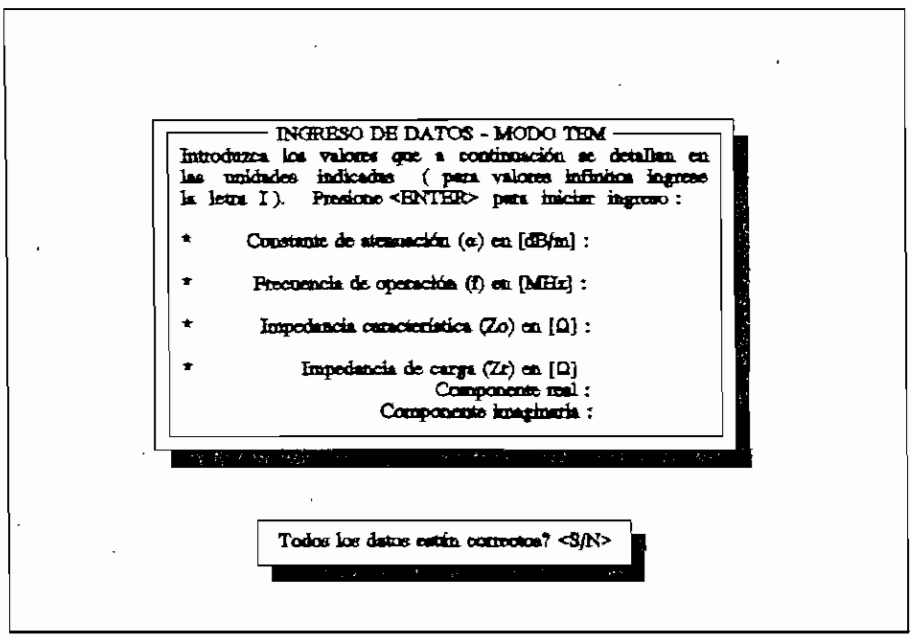

Figura A.5. Pantalla de ingreso para el modo TEM.

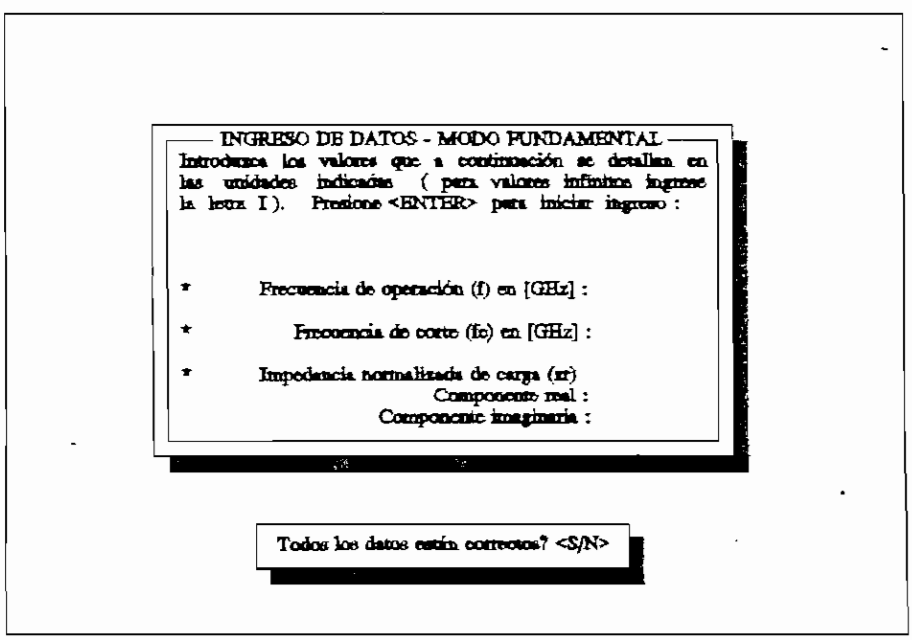

Figura A.6. Pantalla de ingreso de datos para el modo fundamental.

Después del ingreso de datos, las opciones de *Resultados* y *Gráficos* se activan. La opción *Gráficos* permite obtener las ilustraciones para la magnitud del patrón de onda estacionaria y para la variación de fase, como se indica en la Figura A.7.

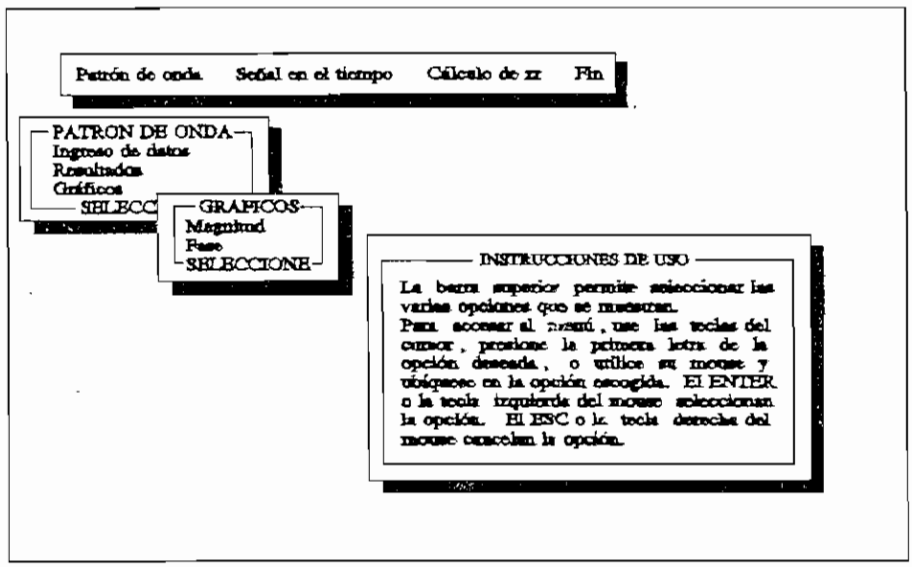

Figura A.7. Pantalla de selección de gráficos.

3.2 Señal transitoria en el tiempo.

La Figura A.8 presenta la pantalla para la selección de *Ingreso de datos* y la obtención del *Gráfico* de la señal transitoria.

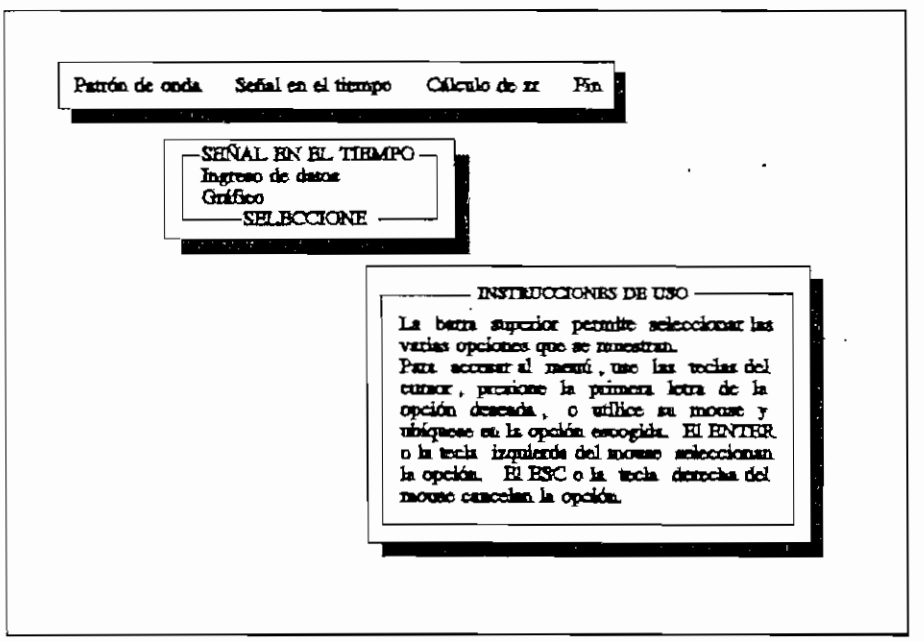

Figura A.8. Pantalla de selección para la señal en el tiempo.

Al escoger *Ingreso de datos*, inmediatamente se presenta un menú de selección de fuente de exitación, sólo para el modo TEM, como se aprecia en la Figura A.9.

Al escoger la opción *Ingreso de datos*, se presenta la pantalla correspondiente al modo de propagación y a la fuente de exitación escogidos, tal como se puede analizar en las Figuras A.10 a A.12.

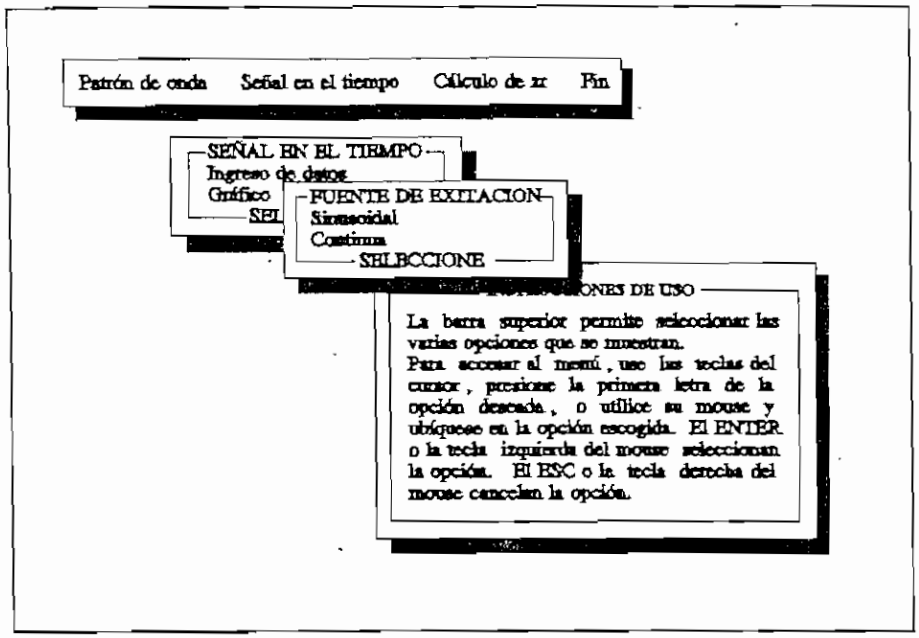

Fig. A.9. Pantalla de selección de fuente de exitación.

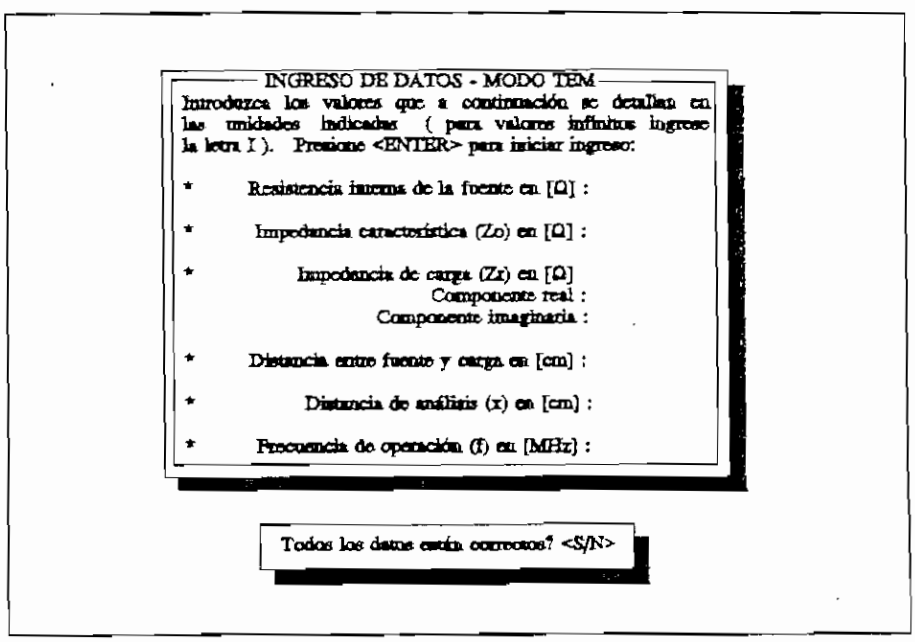

Figura A.10. Pantalla de ingreso de datos para fuente sinusoidal.

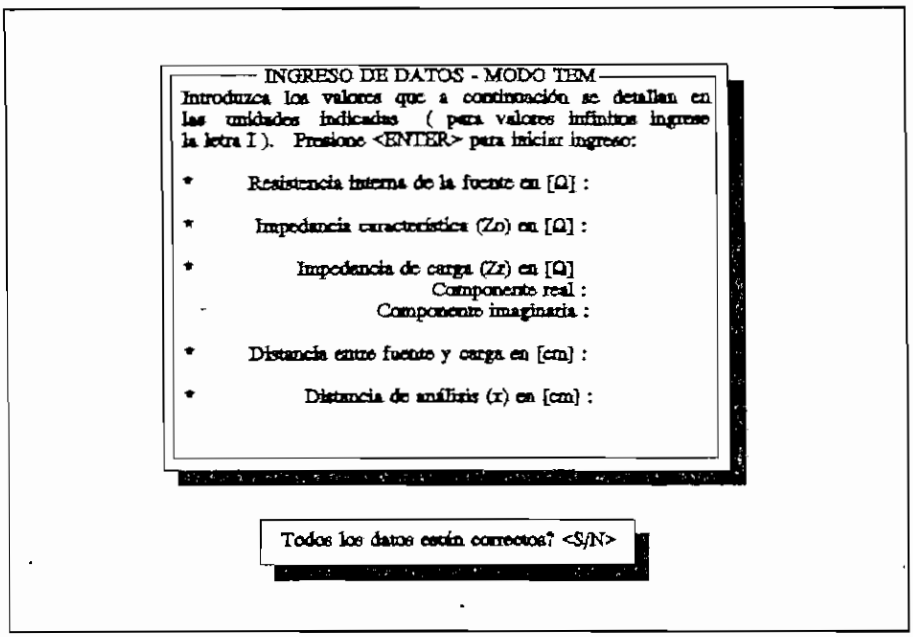

Figura A.11. Pantalla de ingreso de datos para fuente continua.

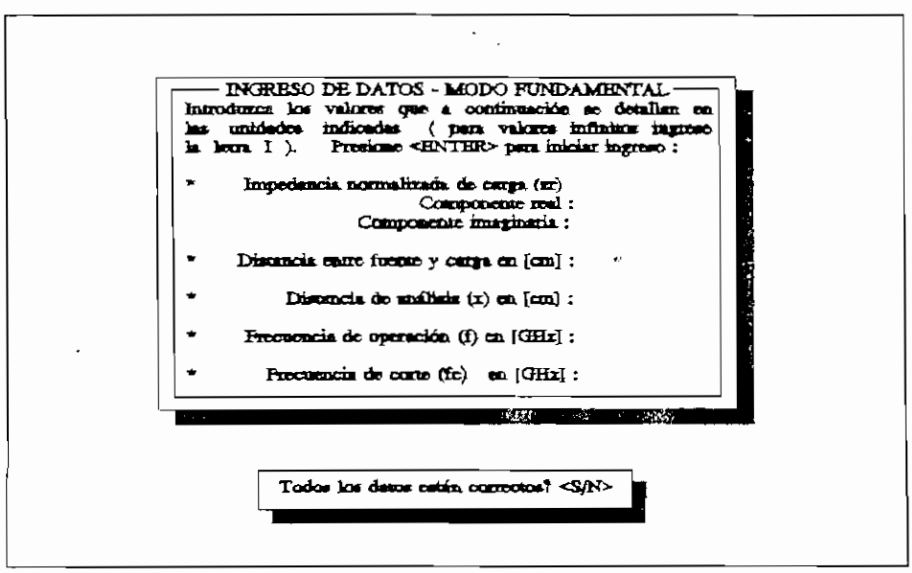

Figura A.12. Pantalla de selección de ingreso de datos para el modo fundamental.

3.3 Cálculo de la impedancia de carga.

El item *Cálculo de zr*, presenta dos opciones: *Ingreso de datos* y *Resultados*, tal como se puede ver en la Figura A.13.

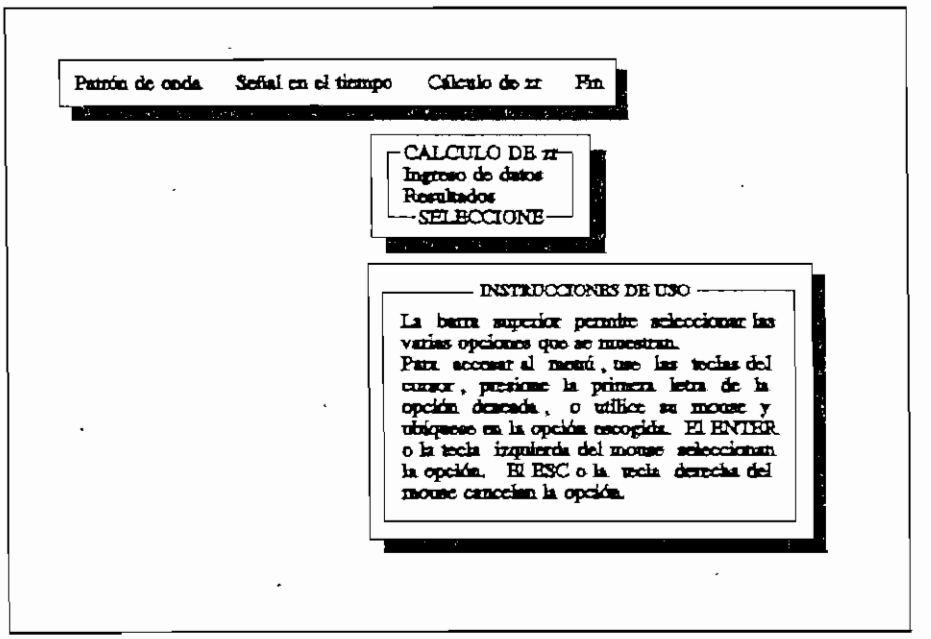

Figura A.13. Pantalla de opciones para cálculo de la impedancia de carga.

La opción *Ingreso de datos* permite escoger el cálculo de la relación de onda estacionaria por medio de la distancia entre los puntos de media potencia, o el ingreso directo de S. La Figura A.14 indica lo anteriormente dicho.

Si se escoge la opción *Ingreso en forma directa*, se presenta la pantalla correspondiente al modo de propagación analizado, como se puede observar en las Figuras A.15 y A.16.

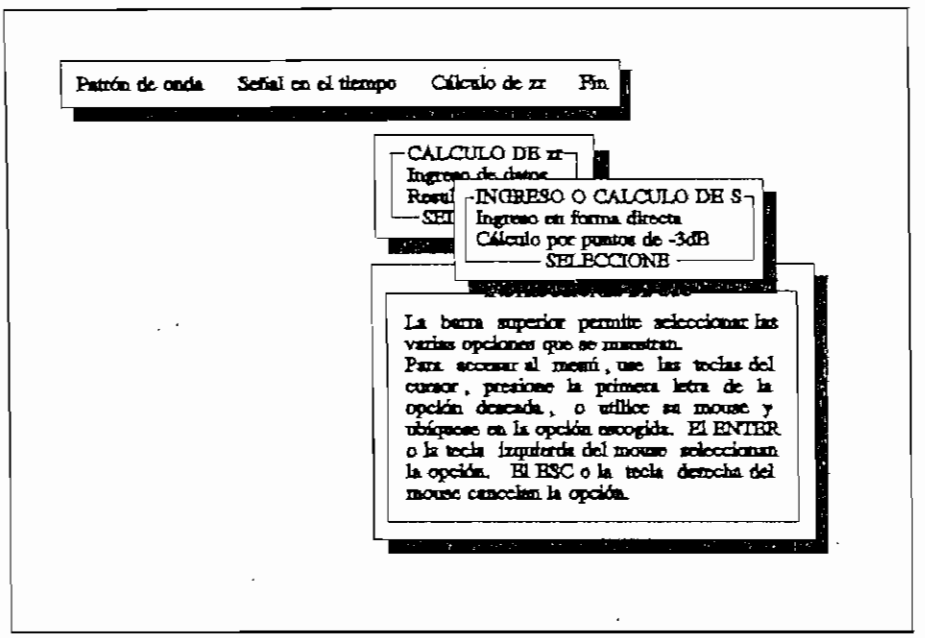

Figura A.14. Pantalla de selección de ingreso o cálculo de S.

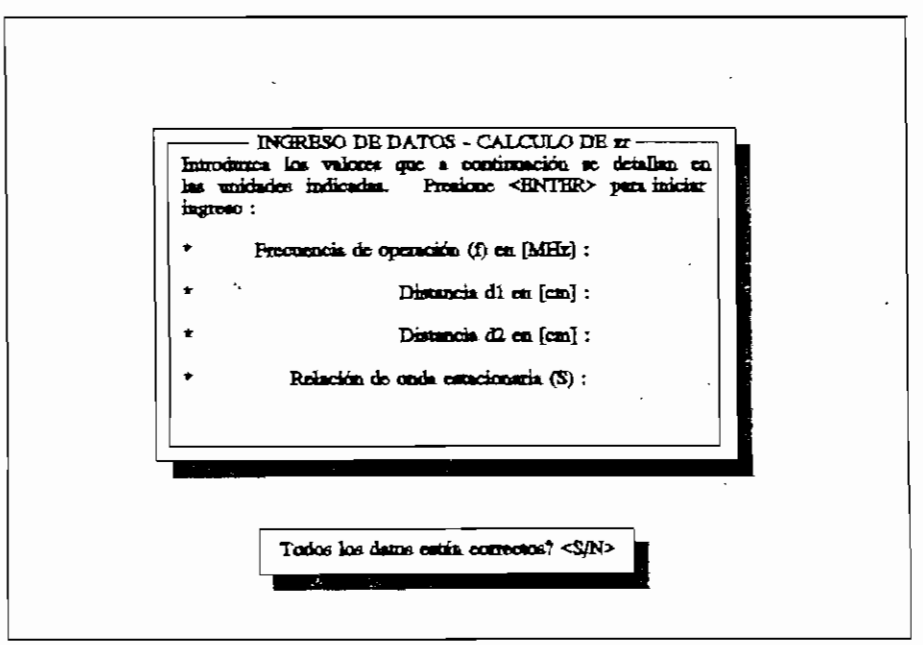

Figura A.15. Pantalla de ingreso directo de S para el modo TEM.

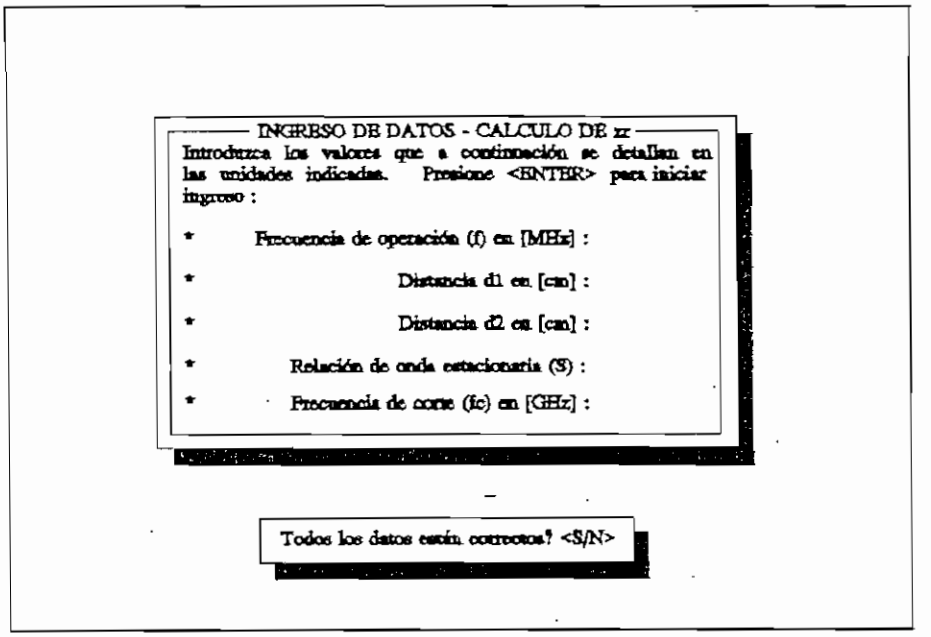

Figura A.16. Pantalla de ingreso en forma directa para el modo fundamental.

Si la opción escogida es *Cálculo por puntos de -3dB*, para el ingreso de datos se presenta la pantalla correspondiente al modo de propagación en análisis, tal como se observa en las Figuras A.17 y A.18.

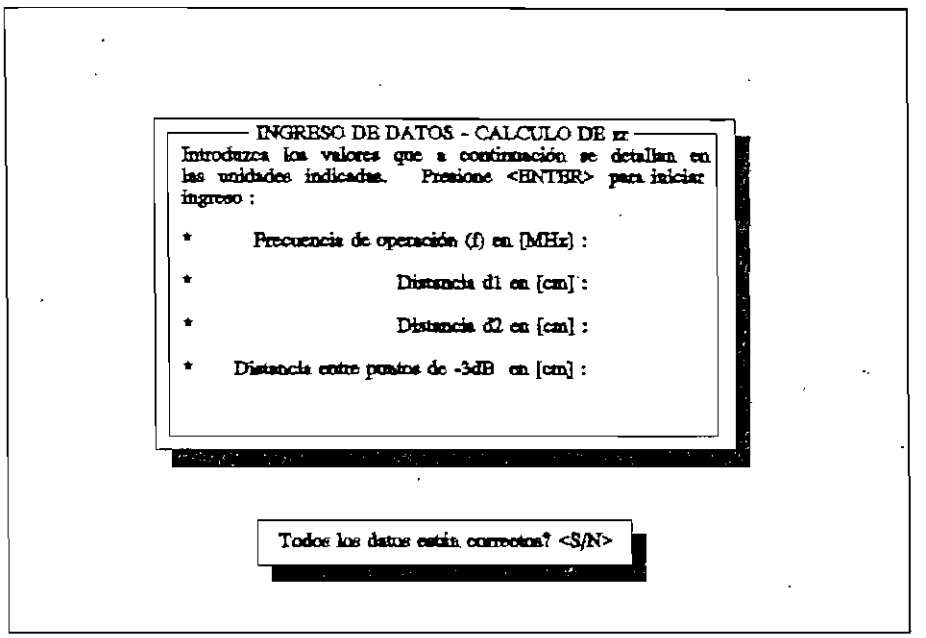

Figura A.17. Pantalla de ingreso de datos para el modo TEM.

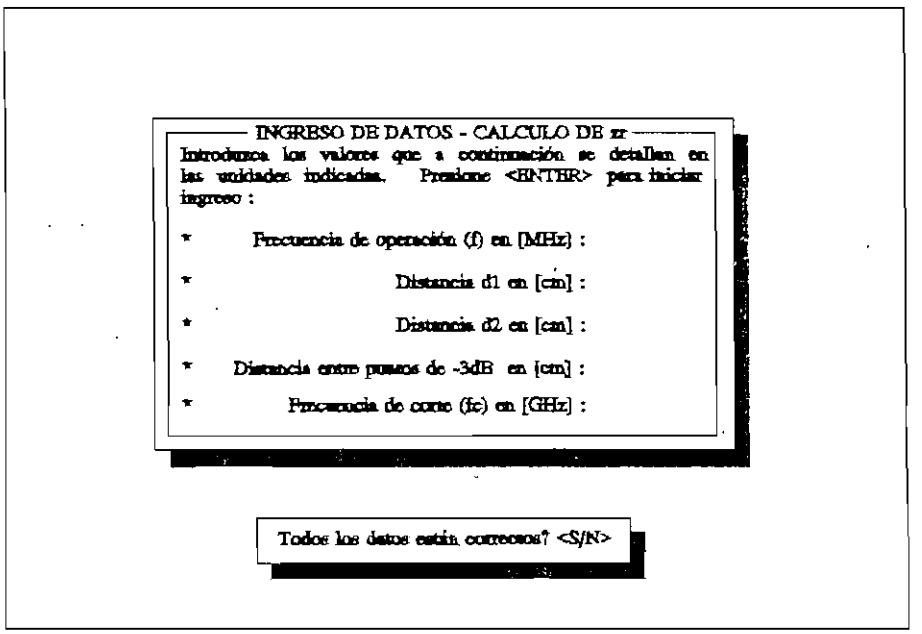

Figura A.18. Pantalla de ingreso de datos para el modo fundamental.

3.4 Fin del programa.

En la Figura A.19, se observa la pantalla correspondiente a la opción *Fin*. Al seleccionar *No* el programa retorna a la pantalla mostrada en la Figura A.2.

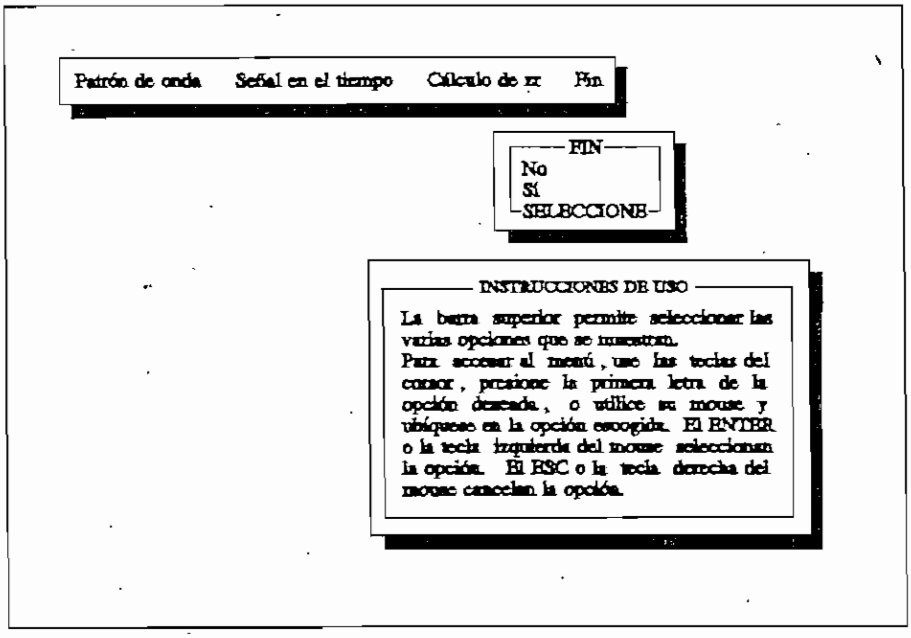

Figura A.19. Pantalla de selección de Fin del programa.