
# The User Interface: Screen Design

WEEK
2

Now that you have a specification, your direction is set. You are ready to begin constructing your application. When developing a large-scale application, you should develop in steps. The specification should be your first step. Today, you'll begin the second step. Today you will:

- ☐ Learn about creating entry and edit screens.
- ☐ Learn some of the standards for screen design.
- ☐ Create entry and edit screens for the application presented on Day 12's specification.
- ☐ Create a screen for the musical items (albums).
- ☐ Create a screen for the medium codes.
- ☐ Create a screen for the groups.

# User Interfaces

The most fun that you'll have when developing an application is creating the user interfaces. The *user interface* is the part of an application that the user sees. These interfaces are generally screens that allow the entry and editing of information; however, they may also include menus, pop-up dialog boxes, message boxes, and more. Figures 13.1 through 13.4 show several different user interfaces.


**Figure 13.1.** *A menu interface from a DOS application.*

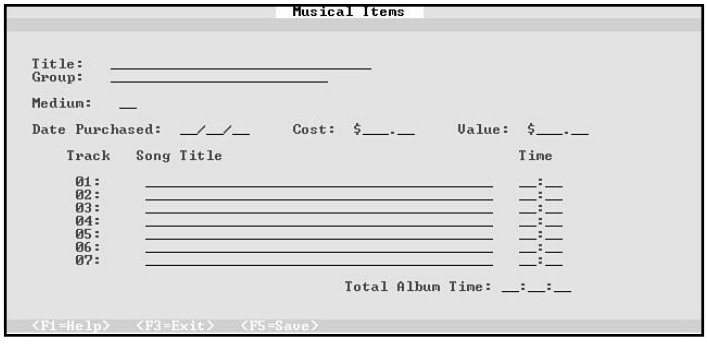



**Figure 13.2.** *An entry and edit screen.*

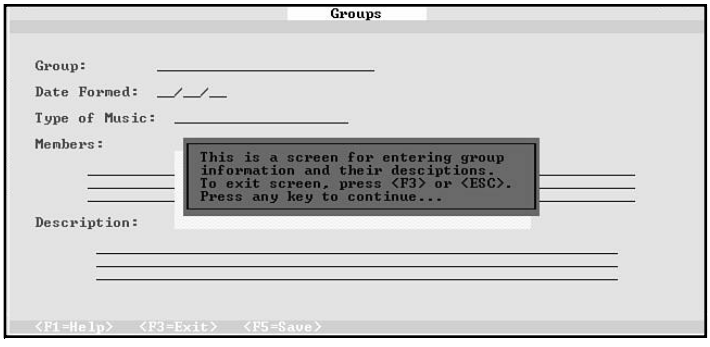



**Figure 13.3.** *An informational interface.*

All of these interfaces use the text graphics that were discussed on Day 8. There are also Graphical User Interfaces (GUIs), which take advantage of pixel graphics. Programs such as Microsoft's Windows (see Figure 13.4), IBM's OS/2, and Geoworks all use graphical user interfaces.

Today, you will learn about entry and edit screens such as the one in Figure 13.2. Before presenting the code, you'll be introduced to the standards on creating entry and edit screens. On Day 14, you'll learn how to create menuing interfaces such as the one in Figure 13.1. On Day 14, you'll also cover action bar items. The days following will help to build the internal functions and reports.

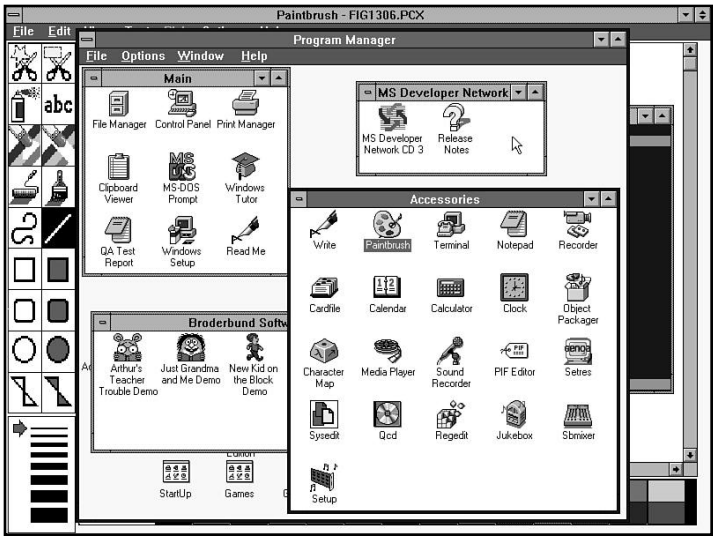



**Figure 13.4.** *A graphical user interface (Microsoft Windows 3.1).*

**Tip:** You should concentrate on building your application one day at a time. Today, you should concentrate on creating the entry and edit portion of your application.

# Creating a User Interface

Many new programmers have a tendency to create screens based solely on what they believe looks neat. To them, the screen actually may look neat; however, they may not have considered the audience that will be using the screen. If you are developing a system that will be given to others, you should consider some standards in screen design. While there are few published standards, there are informal standards that have been accepted by most developers.

**Note:** Standards are usually operating-system specific. For example, DOS applications follow one set of standards, but OS/2 applications have a different set of standards. Several of the standards may cross operating systems.

# The Benefit of Following Standards

There are several possible benefits that can be gained by following a few standards when developing your applications. These benefits include:

- ☐ Lower learning curve for people using the application.
- ☐ Higher comfort level for users in regard to the application.
- ☐ Consistency among your applications.
- ☐ Reduced cost in developing.
- ☐ Increased productivity for you.

The most commonly touted benefit is a consistency in look and feel. Because your application will be designed similar to other programs, users won't feel lost when they use it (look at it). Because parts of your application will look familiar, users will be able to concentrate on learning what the heart of the application does, and not the little things that make the application work. This will help your users increase their productivity with the application.

Additionally, because many functions are predetermined, you won't spend time re-thinking how to accomplish a lot of tasks. This cuts the time you spend on developing applications, which in turn can increase your productivity and reduce your cost of development.

The benefits of standards can also be seen with an experiment. If you have several applications, try running them. In each application, press the F1 function key. In most of your applications, this should provide you with help information.

# Creating a Common User Interface

You should work to create your entry and edit screens similar to others that are available. To do this, you should understand the parts of a screen. Figures 13.5 and 13.6 present the different parts of an entry and edit screen. You'll see each of these parts as you develop an application over the next few days.

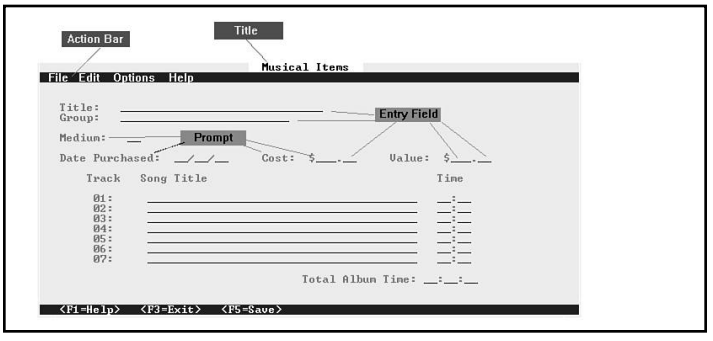



**Figure 13.5.** *Some of the parts of an entry and edit screen.*

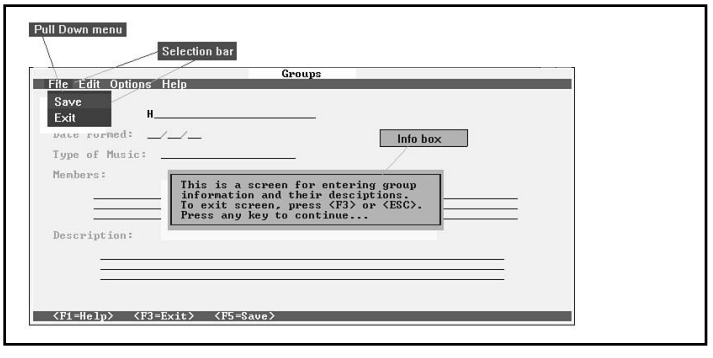



**Figure 13.6.** *Additional parts of an entry and edit screen.*

These figures present an entry and edit screen. There are also standards for working with menus and function keys.

# The Entry and Edit Standards

In the entry and edit portions of an application, there are several areas that you should concentrate on providing standards. These areas include the interface itself and its navigation, the function or special keys, and the feedback to the user.

The feedback to the user is the most straightforward area. An application should provide predictable, consistent feedback. If the user does something that is wrong, either a warning message should be displayed or the computer should beep. A beep should not be used to signal that something was done right. Additionally, if a user does something drastically wrong, a message should be displayed explaining what the error is. Errors can be displayed in message boxes that are red with yellow text to stand out. In addition, the color red is associated with stopping.

The navigation in an entry and edit window should also follow some standards. When entering fields on a screen, the user should start at the top left and work toward the bottom right. When possible, the fields should be presented in the same order as they would be read on a book page. In addition to this navigation, the keys used should follow the functionality presented in Table 13.1.

**Table 13.1. Standards for navigating an entry and edit window.**

| Key | Function |
|---|---|
| Enter | Accept information on-screen and process. Move to next entry field. |
| Page Up | Scroll the screen to the previous page, or to the previous set of information. Go to the first field on the screen. |
| Page Down | Scroll the screen to the next page, or to the next set of information. Go to the last field on a screen. |
| up arrow | Move the cursor to the previous field. If you reach the top of the screen, go to the bottom. Move the cursor to the field directly above the cursor's current location. If you reach the top of the screen, go to the bottom. |
| down arrow | Move the cursor to the next field. If you reach the bottom of the screen, go back to the top. Move the cursor to the field directly below the cursor's current position. If you reach the bottom of the screen, go back to the top. |
| right arrow | Move the cursor to the next position on the current field. |
| left arrow | Move the cursor to the previous position on the current field. |
| Tab | Move the cursor to the next field on the screen. If on the last field, go to the first field. |

*continues*

**Table 13.1. continued**

| Key | Function |
|---|---|
| Shift+Tab | Move the cursor to the previous field on the screen. If on the first field, go to the last field. |
| Home | Put the cursor on the first position of a field. Put the cursor on the first field on the screen. |
| End | Put the cursor on the last filled position of a field. Put the cursor on the last field on the screen. |
| Ctrl+Home | Move the cursor to the first field on the screen. |
| Ctrl+End | Move the cursor to the last field on the screen. |
| Ctrl+Page Down | Move the cursor to the last page. |
| Ctrl+Page Up | Move the cursor to the first page. |
| Alt | Access the action bar if it exists. |
| Backspace | Move the cursor one position to the left if in a field. Remove the character. Move the cursor to the previous field. |
| Delete | Delete the character at the current cursor position. |
| Insert | Toggle whether characters are inserted or overwritten in an entry field. |
| Ctrl+left arrow | Show the page or screen to the left. |
| Ctrl+right arrow | Show the page or screen to the right. |

**Note:** These are informal standards. In some cases, more than one usage is given. This is due to the informality of these uses. You should use these as guidelines.

# The Function Key Standards

The standards for the function keys are probably more important than those for navigation in Table 13.1. Many people get used to the functionality generally

provided by the function keys and other special keys. Table 13.2 presents the functionality usually associated with these keys. Additionally, the table shows which keys should not be used for other functions.

**Table 13.2. The function keys and other special keys.**

| Key | Reusable | Function |
|---|---|---|
| F1 | No | Help (context sensitive) |
| F2 | ** | Extended Help |
| F3 | No | Exit |
| F4 | | |
| F5 | | |
| F6 | | |
| F7 | | Previous |
| F8 | | Next |
| F9 | | Help on key assignments |
| F10 | ** | Access action bar |
| F11 | | Provide alphabetical listing of help topics |
| F12 | | |
| Escape | No | Cancel current task |

No means you should not reuse the key.

** Means you can reuse it, but it is not advised.

**Note:** You should avoid the F11 and F12 keys because not all computers have them.

**Warning:** Do not change the function of the keys in Table 13.2 that are marked. Keys such as F1 and F3 have been used in too many applications to be given different functions unless you have no choice. Most people will assume F1 will provide help and F3 will provide a way of exiting.

**Note:** While standards for menuing exist, they won't be covered here. Day 14 covers menuing in detail.

# Creating an Entry and Edit Screen

The three entry and edit screens that make up the *Record of Records!* application specified on Day 12 will be created today. These entry and edit screens include the Medium Codes screen, the Group Information screen, and the Musical Items screen. Each screen will be modeled after the screens in the prototypes of Day 12's specification. Day 14 will present a menuing program that will tie all three of these entry and edit screens together into a single application.

Because menuing won't be covered until Day 14, you'll need a temporary way of accessing the entry and edit screens. In addition to providing a cryptic menu, Listing 13.1—the RECOFREC.C listing—provides several functions. Most importantly, Listing 13.1 contains a `main()` function. The rest of the listings will be linked with RECOFREC.C to create a single executable program. Because of this, they won't contain `main()` functions. RECOFREC.C also contains several functions that will be used in the other listings presented today.

Listing 13.2 contains the header file called RECOFREC.H. This header file contains the prototypes for the functions in Listing 13.1, RECOFREC.C. Listing 13.3 contains RECORDS.H. This file contains the structures that will be used in the *Record of Records!* application.

**Note:** Today's listings are very long; however, they should be easy to follow. The analyses following each listing describes them. Now may be a good time to consider using the diskette that accompanies this book.

Type

**Listing 13.1. RECOFREC.C. A temporary *Record of Records!* menu.**

```
1:  /*=========================================================
2:   * Filename: RECofREC.c
3:   *           RECORD OF RECORDS - Version 1.0
4:   *
5:   * Author:   Bradley L. Jones
6:   *           Gregory L. Guntle
7:   *
8:   * Purpose:  Allow entry and edit of medium codes.
9:   *
10:  * Note:     Assumes 80 columns by 25 columns for screen.
11:  *=========================================================*/
12:
13: #include <stdio.h>
14: #include <conio.h>         /* not an ANSI header, used for getch()
*/
15: #include <string.h>        /* for strlen() */
16: #include <ctype.h>
17: #include “tyac.h”
18: #include “records.h”
19:
20: /*--------------------*
21:  *     prototypes     *
22:  *--------------------*/
23: #include “recofrec.h”
24:
25: int  do_main_menu(void);
26: void initialize_color_table( void );
27:
28: /*-------------------------*
29:  * define global variables  *
30:  *-------------------------*/
31:
32: struct color_table ct;          /* color table */
33:
34: /*=========================================================*
35:  *                    main()                               *
36:  *=========================================================*/
37:
38: main()
39: {
40:   int rv = 0;
41:
42:   initialize_color_table();
43:
44:   while( rv != 4 )               /* loop in temp menu */
45:   {
46:     rv = do_main_menu();
```

*continues*

### Listing 13.1. continued

```
47:      switch( rv )
48:      {
49:          case '1':  /* Menu option 1 */
50:  //                   do_albums_screen();
51:                     break;
52:
53:          case '2':  /* Menu option 2 */
54:  //                   do_medium_screen();
55:                     break;
56:
57:          case '3':  /* Menu option 3 */
58:  //                   do_groups_screen();
59:                     break;
60:
61:          case 'q':  /* exit */
62:          case 'Q':  rv = 4;
63:                     break;
64:
65:          default:   /* continue looping */
66:                     boop();
67:                     break;
68:      }
69:    }
70:
71:    /* clean up screen for exit */
72:    clear_screen( BRIGHTWHITE, BLACK );
73:    cursor(0,0);
74:    repeat_char(' ', 80, YELLOW, BLUE );
75:    write_string( "Thank you for using RECORD OF RECORDS!",
76:                  YELLOW, BLUE, 0, 21 );
77:    return 0;
78:  }
79:
80:  /*--------------------------------------------------------*
81:   *   do_main_menu                                         *
82:   *--------------------------------------------------------*/
83:  int do_main_menu(void)
84:  {
85:     int rv;
86:
87:     draw_borders("  RECORD of RECORDS!  " );
88:
89:     grid( 11, 16, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
90:     box(10, 15, 20, 60, SINGLE_BOX, FILL_BOX, ct.help_fcol,
          ct.help_bcol);
91:
92:     write_string( "1 - Enter Musical Items", YELLOW, GREEN, 11, 23 );
93:     write_string( "2 - Enter Medium Codes", YELLOW, GREEN, 12, 23 );
94:     write_string( "3 - Enter Group Information", YELLOW, GREEN, 13, 23
                   );
```

```
95:     write_string( "Q - Quit", YELLOW, GREEN, 14, 23 );
96:
97:     cursor(24,79);
98:     rv = getch();
99:
100:    return( rv );
101: }
102:
103: /*-------------------------------------------------------*
104:  *   draw_borders()                                       *
105:  *-------------------------------------------------------*/
106: void draw_borders(char *title)
107: {
108:    int col=0;    /* used to center title */
109:
110:    clear_screen( ct.bg_fcol, ct.bg_bcol );
111:
112:    col = (( 80 - strlen(title)) / 2 );
113:
114:    write_string( title, ct.ttl_fcol, ct.ttl_bcol, 0, col );
115:
116:    cursor(1,0);
117:    repeat_char(' ', 80, ct.abar_fcol, ct.abar_bcol );
118:    cursor(24,0);
119:    repeat_char(' ', 80, ct.abar_fcol, ct.abar_bcol );
120: }
121:
122: /*-------------------------------------------------------*
123:  *   display_msg_box()                                   *
124:  *-------------------------------------------------------*/
125: void display_msg_box(char *msg, int fcol, int bcol )
126: {
127:    char *scrn_buffer = NULL;
128:    scrn_buffer = save_screen_area( 11, 15, 19, 60 );
129:
130:    grid( 12, 15, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
131:    box(11, 14, 20, 60, SINGLE_BOX, FILL_BOX, fcol, bcol);
132:
133:    write_string( msg, fcol, bcol, 12, 23 );
134:    write_string( "Press any key to continue...",
135:                  fcol, bcol, 13, 23 );
136:
137:    cursor(24, 79);
138:    getch();
139:
140:    restore_screen_area( scrn_buffer );
141: }
142:
143: /*-------------------------------------------------------*
```

*continues*

### Listing 13.1. continued

```
144:  *   yes_no_box()                                          *
145:  *------------------------------------------------------*/
146: char yes_no_box(char *msg, int fcol, int bcol )
147: {
148:    char ch;
149:    char *scrn_buffer = NULL;
150:    scrn_buffer = save_screen_area( 11, 15, 19, 60 );
151:
152:    grid( 12, 15, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
153:    box(11, 14, 20, 60, SINGLE_BOX, FILL_BOX, fcol, bcol);
154:
155:    write_string( msg, fcol, bcol, 12, 23 );
156:    write_string( "Enter (Y) or (N)", fcol, bcol, 13, 23 );
157:
158:    ch = getch();
159:    ch = toupper(ch);
160:
161:    cursor(24, 79);
162:    while( ch != 'Y' && ch != 'N' )
163:    {
164:       ch = toupper( getch() );
165:    }
166:
167:    restore_screen_area( scrn_buffer );
168:    return(ch);
169: }
170:
171: /*------------------------------------------------------*
172:  *  Function: zero_fill_field();                        *
173:  *  Purpose:  Right justifies a character array and then*
174:  *            pads the left side with zeros. (Assumes   *
175:  *            that the field is NOT null terminated.)   *
176:  *  Returns:  # of zeros used to pad field              *
177:  *            -1 if field too large (longer than 20 )   *
178:  *            0  if field is blank (not padded)         *
179:  *------------------------------------------------------*/
180:
181: int zero_fill_field( char *field, int size )
182: {
183:    int  ctr,
184:         pads = 0;
185:
186:    char tmp[20];
187:
188:    if( size > 20 )
189:    {
190:       pads = -1;   /* field too long */
191:    }
192:    else
193:    if( strlen(field) == 0 )
```

```
194:    {
195:       pads = 0;     /* leave blank fields blank. */
196:    }
197:    else
198:    {
199:       pads = size - (strlen(field));     /* How many 0s? */
200:
201:       for( ctr = 0; ctr < pads; ctr++ ) /* pad tmp field */
202:          tmp[ctr] = '0';
203:
204:           /* copy original info to end of tmp field */
205:       strncpy( tmp+pads, field, strlen(field));
206:           /* replace original field with padded tmp */
207:       strncpy(field, tmp, size);
208:    }
209:
210:    return(pads);
211: }
212:
213: /*-------------------------------------------------------*
214:  * initialize_color_table()                              *
215:  *                                                       *
216:  * Set up global color table for rest of application     *
217:  *-------------------------------------------------------*/
218:
219: void initialize_color_table( void )
220: {
221:    ct.bg_fcol = YELLOW;
222:    ct.bg_bcol = BLUE;
223:
224:    ct.fld_prmpt_fcol = CYAN;
225:    ct.fld_prmpt_bcol = BLUE;
226:
227:    ct.fld_fcol = BRIGHTWHITE;
228:    ct.fld_bcol = BLUE;
229:
230:    ct.fld_high_fcol = YELLOW;
231:    ct.fld_high_bcol = BLUE;
232:
233:    ct.ttl_fcol = BRIGHTWHITE;
234:    ct.ttl_bcol = BLACK;
235:
236:    ct.abar_fcol = BLACK;
237:    ct.abar_bcol = WHITE;
238:
239:    ct.err_fcol = YELLOW;
240:    ct.err_bcol = RED;
241:
242:    ct.db_fcol = WHITE;
243:    ct.db_bcol = BROWN;
```

*continues*

### Listing 13.1. continued

```
244:
245:    ct.help_fcol = YELLOW;
246:    ct.help_bcol = GREEN;
247:
248:    ct.shdw_fcol = BLACK;
249: }
250:
251: /*=========================================================*
252:  *                    end of listing                       *
253:  *=========================================================*/
```

**Note:** Lines 50, 54, and 58 have been commented out. These lines should be uncommented when the listings presented later in this chapter have been included. You'll be told when each line should be uncommented.

### Listing 13.2. RECOFREC.H. The *Record of Records!* program header.

```
1:  /*=========================================================
2:   * Filename: RECofREC.H
3:   *
4:   * Author:   Bradley L. Jones & Gregory L. Guntle
5:   *
6:   * Purpose:  Header file for RECORD of RECORDS! application
7:   *           This contains the function prototypes needed
8:   *           by more than one source file.
9:   *=========================================================*/
10:
11: #ifndef __RECOFREC_H
12: #define __RECOFREC_H
13:
14: /*----------------------------*
15:  * Prototypes from recofrec.c *
16:  *----------------------------*/
17:
18: void draw_borders(char *);
19:
20: int  do_medium_screen(void);
21: int  do_albums_screen(void);
22: int  do_groups_screen(void);
23: void display_msg_box( char *, int, int );
24: char yes_no_box( char *, int, int );
```

```
25:  int  zero_fill_field( char *, int );
26:
27:  #endif
28:  /*=========================================================*
29:   *                     end of header                       *
30:   *=========================================================*/
```

**Listing 13.3. RECORDS.H. The *Record of Records!* program header containing the structures for the record layouts.**

Type

```
1:   /*=========================================================
2:    * Filename: RECORDS.H
3:    *
4:    * Author:   Bradley L. Jones & Gregory L. Guntle
5:    *
6:    * Purpose:  Header file for RECORD of RECORDS! application
7:    *=========================================================*/
8:
9:   #ifndef __RECORDS_H
10:  #define __RECORDS_H
11:
12:  /*---------------------------*
13:   * File structures definitions*
14:   *---------------------------*/
15:
16:  typedef struct
17:  {
18:     char year[2];
19:     char month[2];
20:     char day[2];
21:
22:  } DATE;
23:
24:  typedef struct
25:  {
26:     char title[ 30+1 ];
27:     char group[ 25+1];
28:     char medium_code[2+1];
29:     DATE date_purch;
30:     char cost[ 5+1 ];
31:     char value[ 5+1 ];
32:     int  nbr_songs;
33:
34:  } ALBUM_REC;
35:
36:  typedef struct
37:  {
```

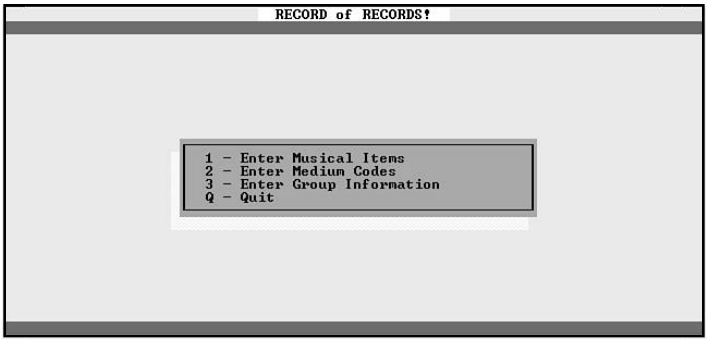
*continues*

**Listing 13.3. continued**

```
38:     char title[40+1];
39:     char minutes[2+1];
40:     char seconds[2+1];
41:
42:  } SONG_REC;
43:
44:  typedef struct
45:  {
46:     char code[2+1];
47:     char desc[35+1];
48:
49:  } MEDIUM_REC;
50:
51:  typedef struct
52:  {
53:     char group[25+1];
54:     DATE date_formed;
55:     char music_type[20+1];
56:     char member[6][30+1];
57:     char info[3][60+1];
58:
59:  } GROUP_REC;
60:
61:  /*--------------------*
62:   * color table        *
63:   *--------------------*/
64:
65:  struct color_table
66:  {
67:     int  bg_fcol;           /* background */
68:     int  bg_bcol;
69:
70:     int  fld_prmpt_fcol;    /* field prompt */
71:     int  fld_prmpt_bcol;
72:
73:     int  fld_fcol;          /* input field */
74:     int  fld_bcol;
75:
76:     int  fld_high_fcol;     /* highlight character */
77:     int  fld_high_fcol;
78:
79:     int  ttl_fcol;          /* screen title */
80:     int  ttl_bcol;
81:
82:     int  abar_fcol;         /* action bar & bottom */
83:     int  abar_bcol;
84:
85:     int  err_fcol;          /* error */
86:     int  err_bcol;
```

```
87:
88:     int  db_fcol;           /* dialog box & msg box */
89:     int  db_bcol;
90:
91:     int  help_fcol;         /* help box colors */
92:     int  help_bcol;
93:
94:     int  shdw_fcol;         /* shadow color */
95: };
96:
97: /*----------------------------------------*
98:  * extern declarations for global variables *
99:  *----------------------------------------*/
100:
101: extern struct color_table ct;
102:
103: #endif
104: /*=========================================================*
105:  *                    end of header                        *
106:  *=========================================================*/
```

**Output**




**Analysis**

Listing 13.1 presents a shadowed box on a colorful screen with a simple menu. This can be seen in the output previously presented. In addition to this menu, there are several functions that will be used by the rest of the application. On Day 14, this menu will be replaced with a full functioning menu.

In lines 13 to 23, several header files are included. All of these header files, except for CONIO.H, are ANSI-compatible. While CONIO.H is not defined by the ANSI standards, it should not limit the portability of the program. Lines 17, 18, and 23 include header files that you have created. The TYAC.H header file is used with your TYAC.LIB library, which needs to be linked with these listings. Line 18 contains the header file that is in Listing 13.3. The RECOFREC.H header file is in Listing 13.2.

The RECOFREC.H header file is a small listing. It contains function prototypes for the functions that are used in more than one of the source files in the *Record of Records!* application. You will see this file included in each of the screen files presented later today.

**Expert Tip:** Many programmers consolidate prototypes for all the functions in an application into a single header file. This consolidated header file is then included in all of the source files for the given application.

The RECORDS.H header file, which is included in line 18 of RECOFREC.C, is in Listing 13.3. This header file contains all of the record structures that are used in the *Record of Records!* application. The `typedef` command has been used to create constant data types with each of these structures. Later, in the screen source files, these constants will be used to declare records.

Lines 16 to 22 of RECOFREC.H begin the structure declarations with a date structure called `DATE`. This structure will become part of the other structures in the rest of the listing. Lines 24 to 34 contain the `ALBUM_REC` structure. This is the structure that will be used with the Musical Items screen. If you review the specification from Day 12, you'll see that the information presented in the data matrix matches the members in the structure. The `SONG_REC` structure follows the `ALBUM_REC` structure. This structure will be used to hold each song title on the Musical Items screen. This will be used in conjunction with the `ALBUM_REC` structure.

Lines 44 to 49 contain the `MEDIUM_REC` structure. This is a much simpler structure to follow because it contains only two items. This structure is used in the first screen presented. Lines 51 to 59 contain the `GROUP_REC` structure, which is used to hold and save the group information.

The final structure in this header is the `color_table` structure. Because only a single color table will be created, there is no reason to create a type defined constant. Instead, a color table structure is declared in line 32 of the RECOFREC.C source file. Looking down the color table, you can see that there are variables defined for each of the different type of colors that could be used in your application (lines 67 to 94). As you review the source code presented over the next few days, you'll see that these structure members are used instead of the color constants presented in the TYAC.H header file.

A `color_table` structure is actually declared and initialized in RECOFREC.C (Listing 13.1). This structure is declared in line 32 as `ct`. This structure is then filled

with values when the `initialize_color_table()` function (lines 213 to 249) is called in line 42 of `main()`. This function is simple to follow. Each member of the color table that will be used in the application is initialized to an appropriate value. Because this function is called at the beginning of the application and because the color table, `ct`, is declared global, the color table can be used by the entire application.

The `main()` function in RECOFREC.C is easy to follow. A `while` loop in lines 44 to 69 puts the program into a loop that displays a menu and then waits for the user to enter a menu option of 1, 2, 3, or Q. The program continues to loop until the letter Q is entered. Once the user enters Q, the program performs some cleanup, displays a message, and then exits. While the program is looping, only three other values are valid, 1, 2, and 3. If the user enters any value other than these, the `boop()` function will cause the program to beep (line 66). If 1, 2, or 3 is called, then the function for the appropriate entry and edit screen is called. Each of these functions is presented in listings later today. Until you create these listings, you should leave these functions commented out.

As was stated, the main menu is displayed with a function call in line 46, `do_main_menu()`. This function is declared in lines 80 to 101. As you will see, this function doesn't present any features that you haven't already seen on earlier days. The `write_string()`, `grid()`, and `box()` functions should all be familiar now. The `draw_borders()` function, which is called in line 87, is defined in lines 103 to 120. This function is broken out from the `do_main_menu()` because the screen entry and edit applications will also use it. Again, the functions shouldn't present anything new.

You should note that in the `do_main_menu()` and the `draw_borders()` functions, color constants aren't used. Instead, the `ct` structure members are used. While this may seem a little cryptic at first, you'll find that using a structure like this will make your program more functional in the long run.



This leaves three final functions in RECOFREC.C. These are `display_msg_box()`, `yes_no_box()`, and `zero_fill_field()`. They are application-specific functions that are used in multiple source files. Because they are used in multiple places, they have been declared in the RECOFREC.C listing along with the `main()` function. Each of these has its own purpose. The `display_msg_box()` function displays a one line message on the screen in a shadowed box. It also displays a "Press any key to continue..." message before waiting for a key to be pressed by the user. You'll see this function used throughout the application. The `yes_no_box()` function in lines 143 to 165 is nearly identical to the `display_msg_box()`. It also displays a message; however, instead of waiting for any key to be pressed, it waits for a Y or an N to be pressed. This value, Y or N, is then passed back to the calling program. The final function, `zero_fill_field()`, may seem a little confusing. This function pads a field with zeros.

It pads the zeros on the left. For example, if a date of 1/1/94 were to be entered, you would use this function to add zeros to the day and month, thus making the date 01/01/94. This function contains a limitation in that it cannot pad a field that is larger than 20 characters in size. This limitation can be changed by modifying line 189. If the size is increased, the temporary character array, `tmp`, would also need to be increased in size (line 186). The comments in the listing should help you understand the rest of this function.

## Overview of Creating Entry and Edit Screens

Now that you have a temporary menu and a table full of colors, you are ready to create your first full-fledged entry and edit screen. The next three listings contain each of the three entry and edit screens for the *Record of Records!* application. Each of these three listings is presented in the same format. In fact, the Medium Codes screen's code was used as a template to create the other two listings. You'll find that this listing makes a good template for creating your own entry and edit screens.

Figure 13.7 presents a pseudo-flow chart of the flow used by an entry and edit screen. This is a simplistic representation of what will occur in the entry and edit screen; however, it should help you understand the overall flow.

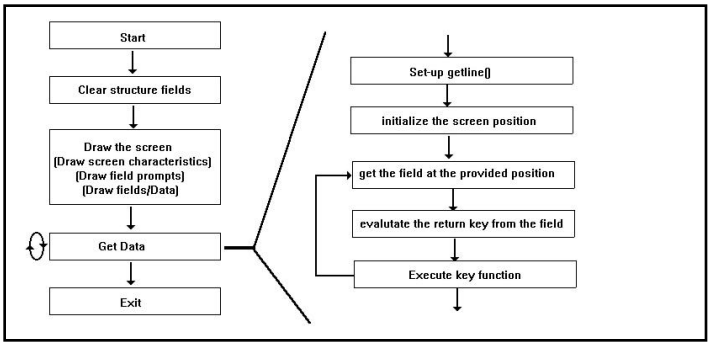



**Figure 13.7.** *The flow of an entry and edit screen.*

The analysis of the medium screen will follow the same order as in this figure.

# The Medium Code Entry and Edit Screen

The medium screen is used to capture the medium code and a description of what the code represents. For example, the code "CD" could be used to represent a compact disc. On a later day, you'll use the medium codes entered in this screen to validate the medium codes entered on the Musical Items screen.

## The Code Behind the Medium Screen

Most of the code for the medium screen is presented all at once in Listing 13.4. This will help you more easily understand the code. The analysis for this listing will be covered in the next few sections. This listing should be compiled along with Listing 13.1, RECOFREC.C. In addition, it will need to be linked with your TYAC.LIB library.

**Note:** Don't forget to uncomment line 54 of RECOFREC.C (Listing 13.1).

**Listing 13.4. MEDIUMS.C. The medium screen.**

```
1:  /*=========================================================
2:   * Filename: medium.c
3:   *
4:   * Author:   Bradley L. Jones
5:   *           Gregory L. Guntle
6:   *
7:   * Purpose:  Allow entry and edit of medium codes.
8:   *
9:   * Note:     This listing is linked with RECofREC.c
10:  *           (There isn't a main() in this listing!)
11:  *=========================================================*/
12:
13:  #include <stdio.h>
14:  #include <string.h>
15:  #include <conio.h>        /* for getch() */
16:  #include "tyac.h"
17:  #include "records.h"
18:
19:  /*--------------------*
20:   *     prototypes     *
21:   *--------------------*/
```

*continues*

### Listing 13.4. continued

```
22:  #include “recofrec.h”
23:
24:  void draw_medium_screen( void );
25:  void draw_medium_prompts( void );
26:  void display_medium_fields( void );
27:
28:  int  clear_medium_fields(void);
29:  int  get_medium_data( int row );
30:  int  get_medium_input_data( void );
31:  void display_medium_help(void);
32:  int  add_medium_data( void );
33:
34:  /*------------------*
35:   * Defined constants *
36:   *------------------*/
37:
38:  #define MEDIUM_DBF   “MEDIUMS.DBF”
39:
40:  /*------------------------*
41:   * structure declarations *
42:   *------------------------*/
43:
44:  MEDIUM_REC medium;
45:  MEDIUM_REC *p_medium = &medium;
46:
47:  /*========================================================*
48:   *   do_medium_screen()                                   *
49:   *========================================================*/
50:
51:  int do_medium_screen(void)
52:  {
53:      clear_medium_fields();
54:      draw_medium_screen();
55:      get_medium_input_data();
56:      return 0;
57:  }
58:
59:  /*------------------------*
60:   *   draw_medium_screen()  *
61:   *------------------------*/
62:
63:  void draw_medium_screen( void )
64:  {
65:     draw_borders(“     MEDIUM     “);  /* draw screen bckgrnd */
66:
67:     write_string( “<F1=Help>   <F3=Exit>   <F4=Save>”,
68:           ct.abar_fcol, ct.abar_bcol, 24, 3);
69:
70:     draw_medium_prompts();
71:     display_medium_fields();
```

```
72:  }
73:
74:  /*------------------------*
75:   *   draw_medium_prompts() *
76:   *------------------------*/
77:
78:  void draw_medium_prompts( void )
79:  {
80:     write_string("Medium Code:",
81:           ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 4, 3 );
82:     write_string("Medium Description:",
83:           ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 6, 3 );
84:  }
85:
86:  /*------------------------*
87:   *   draw_medium_fields()  *
88:   *------------------------*/
89:
90:  void display_medium_fields( void )
91:  {
92:     write_string("__", ct.fld_fcol, ct.fld_bcol, 4, 17 );
93:     write_string("_____________________________________",
94:           ct.fld_fcol, ct.fld_bcol, 6, 24 );
95:
96:     /* display data, if exists */
97:
98:     write_string(medium.code, ct.fld_fcol, ct.fld_bcol, 4, 17 );
99:     write_string(medium.desc, ct.fld_fcol, ct.fld_bcol, 6, 24 );
100: }
101:
102: /*-------------------------------------------------------*
103:  *   get_medium_input_data()                             *
104:  *-------------------------------------------------------*/
105: int get_medium_input_data( void )
106: {
107:     int   position,
108:       rv,
109:       loop = TRUE;
110:
111:     /*  Set up exit keys.  */
112:     static char fexit_keys[ 13 ] = { F1, F3, F4, ESC_KEY,
113:                     PAGE_DN, PAGE_UP, CR_KEY,
114:                     TAB_KEY, ENTER_KEY, SHIFT_TAB,
115:                     DN_ARROW, UP_ARROW, NULL };
116:
117:     static char *exit_keys = fexit_keys;
118:     getline( SET_EXIT_KEYS, 0, 12, 0, 0, 0, exit_keys );
119:
120:     /*** setup colors and default keys ***/
121:     getline( SET_DEFAULTS, 0, 0, 0, 0, 0, 0 );
```

*continues*

### Listing 13.4. continued

```
122:     getline( SET_NORMAL, 0, ct.fld_fcol, ct.fld_bcol,
123:                 ct.fld_high_fcol, ct.fld_high_bcol, 0 );
124:     getline( SET_UNDERLINE, 0, ct.fld_fcol, ct.fld_bcol,
125:                 ct.fld_high_fcol, ct.fld_high_bcol, 0 );
126:     getline( SET_INS, 0, ct.abar_fcol, ct.abar_bcol, 24, 76, 0 );
127:
128:
129:
130:
131:
132:     position = 0;
133:
134:     while( loop == TRUE )      /**  get data for top fields  **/
135:     {
136:        switch( (rv = get_medium_data( position )) )
137:        {
138:           case CR_KEY    :
139:           case TAB_KEY   :
140:           case ENTER_KEY :
141:           case DN_ARROW  : /* go down a field */
142:                            ( position == 1 ) ? ( position = 0 ) :
                                   position++;
143:                            break;
144:
145:           case SHIFT_TAB :
146:           case UP_ARROW  : /* go up a field */
147:                            ( position == 0 ) ? ( position = 1 ) :
                                   position--;
148:                            break;
149:
150:           case ESC_KEY :
151:           case F3      :   /* exit back to main menu */
152:                            if( (yes_no_box( “Do you want to exit?”,
153:                                    ct.db_fcol, ct.db_bcol )) ==‘Y’)

154:                            {
155:                                loop = FALSE;
156:                            }
157:
158:                            break;
159:
160:           case F4  :       /* add data */
161:
162:                            if( strlen( medium.code ) == 0 )
163:                            {
164:                               display_msg_box(“Must enter a medium
                                                  code”,
165:                                       ct.err_fcol, ct.err_bcol);
166:
```

```
167:                            }
168:                            else
169:                            if( strlen( medium.desc ) == 0 )
170:                            {
171:                               display_msg_box("Must enter a"
                                                  "description",
172:                                        ct.err_fcol, ct.err_bcol);
173:                            }
174:                            else /* all okay, so add data */
175:                            {
176:                               add_medium_data();
177:
178:                               clear_medium_fields();
179:                               draw_medium_screen();
180:                               position = 0;
181:                            }
182:
183:                            break;
184:
185:          case PAGE_DN :    /* go to last data entry field */
186:                            position = 1;
187:                            break;
188:
189:          case PAGE_UP :    /* go to first data entry field */
190:                            position = 0;
191:                            break;
192:
193:          case F1:          /* help */
194:                            display_medium_help();
195:                            draw_medium_screen();
196:                            break;
197:
198:          default:          /* error */
199:                            display_msg_box( " Error ",
200:                                  ct.err_fcol, ct.err_bcol );
201:                            break;
202:
203:       }     /* end of switch */
204:    }        /* end of while  */
205:
206:    return( rv );
207: }
208:
209: /*-------------------------------------------------------*
210:  *   get_medium_data()                                   *
211:  *-------------------------------------------------------*/
212:
213: int get_medium_data( int row )
214: {
215:    int rv;
```

*continues*

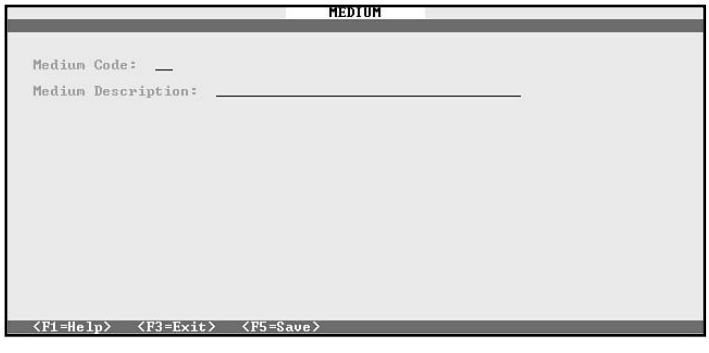
### Listing 13.4. continued

```
216:
217:     switch( row )
218:     {
219:        case 0 :
220:           rv = getline( GET_ALPHA, 0,  4, 17, 0,  2, medium.code);
221:           break;
222:        case 1 :
223:           rv = getline( GET_ALPHA, 0,  6, 24, 0, 35, medium.desc);
224:           break;
225:     }
226:     return( rv );
227: }
228:
229: /*-------------------------------------------------------*
230:  *   clear_medium_fields()                               *
231:  *-------------------------------------------------------*/
232:
233: int clear_medium_fields(void)
234: {
235:   getline( CLEAR_FIELD, 0,  3, 0, 0, 0, medium.code );
236:   getline( CLEAR_FIELD, 0, 36, 0, 0, 0, medium.desc );
237:
238:   return(0);
239: }
240:
241: /*-------------------------------------------------------*
242:  *   display_medium_help()                               *
243:  *-------------------------------------------------------*/
244:
245: void display_medium_help(void)
246: {
247:   grid( 11, 16, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
248:   box(10, 15, 20, 60, SINGLE_BOX, FILL_BOX,
249:                       ct.help_fcol, ct.help_bcol);
250:
251:   write_string( "This is a screen for entering medium",
252:                 ct.help_fcol, ct.help_bcol, 11, 22 );
253:   write_string( "codes and their descriptions. To exit",
254:                 ct.help_fcol, ct.help_bcol, 12, 22 );
255:
256:   write_string( "this screen, press <F3> or <ESC>.",
257:         ct.help_fcol, ct.help_bcol, 13, 22 );
258:   write_string( "Press any key to continue...",
259:                 ct.help_fcol, ct.help_bcol, 14, 22 );
260:
261:   cursor(24, 79);
262:   getch();
263: }
264:
```

```
265: /*-------------------------------------------------------*
266:  *  add_medium_data();                                    *
267:  *  Returns:  1 - if all okay                            *
268:  *            0 - if not all okay                        *
269:  *-------------------------------------------------------*/
270:
271: int add_medium_data( void )
272: {
273:    int   rv = 0;
274:    FILE *fp;
275:
276:    if( (fp = fopen( MEDIUM_DBF, "a" )) == NULL )
277:    {
278:          display_msg_box("Error opening file...",
279:                          ct.err_fcol, ct.err_bcol );
280:    }
281:    else
282:    {
283:       if( (fwrite( p_medium, sizeof( medium ), 1, fp )) == 0 )
284:       {
285:          display_msg_box("Error writing data...",
286:                          ct.err_fcol, ct.err_bcol );
287:       }
288:       else
289:       {
290:          display_msg_box("Record added", ct.db_fcol, ct.db_bcol);
291:          rv = 1;
292:       }
293:
294:       fclose( fp );
295:    }
296:    return( rv );
297: }
298: /*=======================================================*
299:  *                    end of listing                     *
300:  *=======================================================*/
```



Output



Analysis This is a long listing; however, the results are worth all the code. Throughout this listing, you can see many of the functions that you created in the previous chapters. One thing you won't find is a `main()` function. This is because the medium screen will be called by the menu presented in Listing 13.1, RECOFREC.C.

## The *do_medium_screen()* Function

The `do_medium_screen()` function is called from line 54 of the RECOFREC.C source file. This function calls three other functions and then returns back to the menu. These three functions follow the flow presented in Figure 13.5. First a function is called to clear the fields, then the screen is drawn, and finally the data is retrieved.

Before MEDIUM.C begins the `do_medium_screen()` function, several things occur. In lines 13 to 17, various header files are included. In lines 22 through 32, function prototypes are declared. The RECOFREC.H header file is then included. RECOFREC.H is included in the prototype section because it contains mostly prototypes. Line 38 contains a defined constant, `MEDIUM_DBF`. This contains the name of the disk file that will be used to store the medium file. By using a defined constant, it will be easy for you to change the name of the database file. Lines 44 and 45 declare a `MEDIUM_REC` structure called `medium` and a pointer to the structure called `p_medium`. This structure is declared globally so that it is easy to use.

## Clearing the Structure Fields

Line 53 in `do_medium_screen()` calls the function `clear_medium_fields()`, which clears the fields in the structure declared in line 44. The `clear_medium_fields()` function in lines 229 to 239 uses `getline()` to clear each structure member to Nulls. In the case of the `medium` structure, this is the `code` and the `desc` fields. By clearing the fields in this manner, you can be assured that they do not contain any erroneous data.

## Setting Up the Screen

The next step presented in Figure 13.5 was drawing or setting up the screen. The `do_medium_screen()` calls the `draw_medium_screen()` function in line 54. The `draw_medium_screen()` function is declared in lines 59 to 72. Line 65 of this function calls the `draw_borders()` function that was defined in RECOFREC.C (Listing 13.1). The keys that are valid for special functions are then written on the bottom line of the screen (line 67). The next line, line 70, then calls the `draw_medium_prompts()`, which is presented in lines 74 to 84. This function draws each of the field prompts on the screen.

Line 71 of `draw_medium_screen()` calls the `display_medium_fields()` function. This function is important. It draws underlines on the screen in the locations where

getline() will retrieve data. If you leave this function out, getline() will draw the underlines one field at a time. It looks better to draw all the underlines on the screen up front. The second half of display_medium_fields() writes the actual field values on top of the underscores. Although they will be blank the first time into the screen, later functionality may require that the data be displayed too. Once the underlines and data have been drawn, the screen is complete and data entry is ready to begin.

## Capturing the Data

Capturing the data takes up most of the listing. This capturing of the data starts when the do_medium_screen() function calls the get_medium_input_data() in line 55. The get_medium_input_data() function is presented in lines 102 to 207. This function starts by setting up the getline() function. Line 112 sets up the exit keys. As you can see, several exit keys are being set up. If you review the specification on Day 12, you will find that not all of the exit keys have been declared. The rest of the keys will be declared as they are needed on later days. Line 117 sets up a static character array pointing to the exit keys. Line 118 then sets up the exit keys. Lines 121 to 126 use the getline() function to set up the default colors. You should notice that the color table values are being used. These color table values were set up in the RECOFREC.C listing.

**Tip:** If the getline() commands seem unfamiliar, you should review Day 10.

Once set up, the screen position is initialized to 0 in line 132. The first field on the screen is given position 0. Each field on the screen should then be given a sequential number following this one. The order in which the fields will be retrieved is based on their respective position numbers. For the Medium Codes screen, the Medium Codes field is considered position 0 and the description field is considered position 1.

Line 134 begins the loop for getting the data. This loop continues until a key action causes the looping flag, loop, to be chained to FALSE. For each iteration of the loop, a single data item is retrieved. This is done in line 136 using the get_medium_data() function. The current position is passed to the get_medium_data() function so that it knows which field on the screen to retrieve.

Once a field is retrieved, the key returned is evaluated within a switch statement. This return value will be one of the exit keys set up in line 112. Looking at lines 138 to 201, you can see what each exit key does. If the CR_KEY (carriage-return key), TAB_KEY (tab key), ENTER_KEY (enter key), or DN_ARROW (down arrow key) is pressed, then the

position is incremented so that the next field is retrieved. If the field is the last field on the screen (in this case, field number 1), then the position is reset to the first field, 0. The SHIFT_TAB (shift+tab) and the UP_ARROW (up arrow key) do just the opposite. They decrement the position. If the position is the first position on the screen, then the position is reset to the last position on the screen, 1.

The ESC_KEY and the F3 keys function in the same manner. This functionality is presented in lines 151 to 158. Both keys offer the user a way of leaving the entry and edit screen. This follows the standards mentioned previously. The `yes_no_box()` function provided in the RECOFREC.C listing is used to ask the user if he is sure he wishes to exit. If he is not, then the program returns control to the entry screen and the user is left on the field he was previously on. If the user is sure, then `loop` is set to FALSE so that it will end.

The F4 key, lines 160 to 183, is used to add data. Before adding the data, edits may be performed. In this case, the user cannot add the data until he has entered a medium code (lines 162 to 167). If a medium code has not been entered, then a message is displayed explaining the situation to the user. If the code is okay, then the `add_medium_data()` function is called. This function, presented in lines 265 to 297 opens a file, writes the structure, displays a message, and returns. Line 178 then clears the structure fields and redraws the medium screen. This effectively clears the data entry screen for the next record. The position is then reset to 0 and the program resumes accepting data. On Day 15, the `add_medium_data()` function will be replaced as better file handling capabilities are added to the application.

The PAGE_DN and PAGE_UP keys operate in similar ways. PAGE_DN puts the cursor on the last field on the entry screen by changing the value of the position. This is done in line 186 by setting the position to 1. PAGE_UP sets the position to the first field on the screen. This will invariably be the value of 0.

**Note:** There are times when you don't want to set the page down and page up values to the first or last field. This will be seen in the group screen presented later today.

The last key worked with is the F1 key. This key has been defined as the help key. Line 194 calls a function called `draw_medium_screen()`, which is presented in lines 241 to 263. This function creates a box that contains a little bit of help information. On Day 16, you'll see how to create context-sensitive help in your application.

The `default` case in lines 198 to 201 is a safety feature. The `getline()` function should only return those values set up in the exit keys; however, it's always a good programming tactic to include a `default` case in all of your `switch` calls. This can help catch potential problems.

## Getting Individual Data Fields

The `get_medium_input_data()` function called the `get_medium_data()` function in line 136. In doing so, the position of the field that should be retrieved is passed. The `get_medium_data()` function is defined in lines 209 to 227. This function is basically a single `switch` statement that has a case for each field on the screen. Each case has a call to `getline()` that gets the individual field. The return value from `getline()` is captured in `rv` and then returned back to the `get_medium_input_data()` function. This returned value will only be one of the valid exit keys that were set up for `getline()`.

**Note:** Edits on fields can be incorporated in this area of the program. If you want an edit on a field as it is being entered, it could be placed right after the call to `getline()`. In the analysis for the group screen, you will see an example of this.

**Tip:** Review the Medium Codes screen before trying to understand the Group Screen's code. Each of these screens is progressively harder than the previous.



## Everything Else

This is everything in the medium screen! As you should have noticed, this doesn't contain all the features that were mentioned in the specification. In addition, it doesn't have very good file control. Each record is written to a file; however, they can never be accessed. Over the next few days, these holes in the medium screen's functionality will be filled so that your application is complete.

# The Group Information Screen

The Group screen is presented in Listing 13.5. As you will see, this listing is a little more complex than the Medium Codes screen. Some of the differences are caused by the additional number of fields in the `group` structure; however, there are a few other differences. Following is Listing 13.5, which contains the code for the Group Information screen.

**Note:** This listing should be compiled and linked along with Listing 13.1 (RECOFREC.C), Listing 13.4 (MEDIUM.C), and your TYAC.LIB library. You should uncomment line 58 in Listing 13.1 before compiling and linking.

**Listing 13.5. The Group Information screen.**

```
1:  /*=========================================================
2:   * Filename: groups.c
3:   *
4:   * Author:   Bradley L. Jones
5:   *
6:   *
7:   * Purpose:  Allow entry and edit of group information.
8:   *
9:   * Note:     This listing is linked with RECofREC.c
10:  *           (There isn't a main() in this listing!)
11:  *=========================================================*/
12:
13: #include <stdio.h>
14: #include <string.h>
15: #include <conio.h>        /* for getch() */
16: #include "tyac.h"
17: #include "records.h"
18:
19: /*--------------------*
20:  *     prototypes     *
21:  *--------------------*/
22: #include "recofrec.h"
23:
24: void draw_groups_screen( void );
25: void draw_groups_prompts( void );
26: void display_groups_fields( void );
27:
28: int  clear_groups_fields(void);
29: int  get_groups_data( int row );
```

```
30:  int  get_groups_input_data( void );
31:  void display_groups_help(void);
32:  int  add_groups_data( void );
33:
34:  /*------------------*
35:   * Defined constants*
36:   *------------------*/
37:
38:  #define GROUPS_DBF   "GROUPS.DBF"
39:
40:  /*------------------------*
41:   * structure declarations *
42:   *------------------------*/
43:
44:  GROUP_REC groups;
45:  GROUP_REC *p_groups = &groups;
46:
47:  /*========================================================*
48:   *   do_groups_screen()                                   *
49:   *========================================================*/
50:
51:  int do_groups_screen(void)
52:  {
53:      clear_groups_fields();
54:      draw_groups_screen();
55:      get_groups_input_data();
56:      return 0;
57:  }
58:
59:  /*------------------------*
60:   *   draw_groups_screen() *
61:   *------------------------*/
62:
63:  void draw_groups_screen( void )
64:  {
65:     draw_borders("     Groups     ");  /* draw screen bckgrnd */
66:
67:     write_string( "<F1=Help>   <F3=Exit>   <F4=Save>",
68:                   ct.abar_fcol, ct.abar_bcol, 24, 3);
69:
70:     draw_groups_prompts();
71:     display_groups_fields();
72:  }
73:
74:  /*------------------------*
75:   *   draw_groups_prompts()*
76:   *------------------------*/
77:
78:  void draw_groups_prompts( void )
79:  {
```

*continues*

### Listing 13.5. continued

```
80:     write_string("Group:",
81:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 4, 3 );
82:     write_string("Date Formed:    /  /",
83:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 6, 3 );
84:     write_string("Type of Music:",
85:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 8, 3 );
86:     write_string("Members:",
87:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 10, 3 );
88:     write_string("Description:",
89:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 16, 3 );
90: }
91:
92: /*------------------------*
93:  *   draw_groups_fields() *
94:  *------------------------*/
95:
96: void display_groups_fields( void )
97: {
98:     char tmp[3] = { 0, 0, 0 };   /* initialize to null values */
99:     char under_30[31] = {"______________________________"};
100:
101:    write_string(under_30+5, /* 25 underlines */
102:          ct.fld_fcol, ct.fld_bcol, 4, 17 );
103:    write_string("__", ct.fld_fcol, ct.fld_bcol, 6, 17 );
104:    write_string("__", ct.fld_fcol, ct.fld_bcol, 6, 20 );
105:    write_string("__", ct.fld_fcol, ct.fld_bcol, 6, 23 );
106:    write_string(under_30+10, /* 20 underlines */
107:          ct.fld_fcol, ct.fld_bcol, 8, 19 );
108:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 12,  9 );
109:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 12, 42 );
110:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 13,  9 );
111:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 13, 42 );
112:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 14,  9 );
113:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 14, 42 );
114:    /* write the groups.info underlines in two parts */
115:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 18, 10 );
116:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 18, 40 );
117:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 19, 10 );
118:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 19, 40 );
119:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 20, 10 );
120:    write_string(under_30, ct.fld_fcol, ct.fld_bcol, 20, 40 );
121:
122:    /* display data, if exists */
123:
124:    write_string(groups.group,
125:                 ct.fld_fcol, ct.fld_bcol, 4, 17 );
126:    write_string(groups.music_type,
127:                 ct.fld_fcol, ct.fld_bcol, 8, 19 );
128:    write_string(groups.member[0],
129:                 ct.fld_fcol, ct.fld_bcol, 12,  9 );
```

```
130:    write_string(groups.member[1],
131:                 ct.fld_fcol, ct.fld_bcol, 12, 42 );
132:    write_string(groups.member[2],
133:                 ct.fld_fcol, ct.fld_bcol, 13,  9 );
134:    write_string(groups.member[3],
135:                 ct.fld_fcol, ct.fld_bcol, 13, 42 );
136:    write_string(groups.member[4],
137:                 ct.fld_fcol, ct.fld_bcol, 14,  9 );
138:    write_string(groups.member[5],
139:                 ct.fld_fcol, ct.fld_bcol, 14, 42 );
140:    write_string(groups.info[0],
141:                 ct.fld_fcol, ct.fld_bcol, 18, 10 );
142:    write_string(groups.info[1],
143:                 ct.fld_fcol, ct.fld_bcol, 19, 10 );
144:    write_string(groups.info[2],
145:                 ct.fld_fcol, ct.fld_bcol, 20, 10 );
146:
147:    strncpy( tmp, groups.date_formed.month, 2 );
148:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 6, 17 );
149:    strncpy( tmp+4, groups.date_formed.day, 2 );
150:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 6, 20 );
151:    strncpy( tmp, groups.date_formed.year, 2 );
152:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 6, 23 );
153: }
154:
155: /*-------------------------------------------------------*
156:  *   get_groups_input_data()                              *
157:  *-------------------------------------------------------*/
158: int get_groups_input_data( void )
159: {
160:     int   position,
161:           rv,
162:           okay,               /* used with edits */
163:           loop = TRUE;
164:
165:     /*  Set up exit keys.  */
166:     static char fexit_keys[ 13 ] = { F1, F3, F4, ESC_KEY, PAGE_DN,
                                        PAGE_UP, CR_KEY, TAB_KEY,
167:                                     ENTER_KEY, SHIFT_TAB, DN_ARROW,
                                        UP_ARROW, NULL };
168:
169:     static char *exit_keys = fexit_keys;
170:     getline( SET_EXIT_KEYS, 0, 12, 0, 0, 0, exit_keys );
171:
172:     /*** setup colors and default keys ***/
173:     getline( SET_DEFAULTS, 0, 0, 0, 0, 0, 0 );
174:     getline( SET_NORMAL, 0, ct.fld_fcol, ct.fld_bcol,
175:                             ct.fld_high_fcol, ct.fld_high_bcol, 0 );
176:     getline( SET_UNDERLINE, 0, ct.fld_fcol, ct.fld_bcol,
177:                             ct.fld_high_fcol, ct.fld_high_bcol, 0 );
```

*continues*

**Listing 13.5. continued**

```
178:     getline( SET_INS, 0, ct.abar_fcol, ct.abar_bcol, 24, 76, 0 );
179:
180:     position = 0;
181:
182:     while( loop == TRUE )      /**  get data for top fields  **/
183:     {
184:        switch( (rv = get_groups_data( position )) )
185:        {
186:       case CR_KEY    :
187:       case TAB_KEY   :
188:       case ENTER_KEY :
189:       case DN_ARROW  : /* go down a field */
190:                ( position == 13 ) ? ( position = 0 ) : position++;
191:                break;
192:
193:       case SHIFT_TAB :
194:       case UP_ARROW  : /* go up a field */
195:                ( position == 0 ) ? ( position = 13 ) : position--;
196:                break;
197:
198:       case ESC_KEY :
199:       case F3      :   /* exit back to main menu */
200:                if( (yes_no_box( "Do you want to exit?",
201:                       ct.db_fcol, ct.db_bcol )) == 'Y' )
202:                {
203:                    loop = FALSE;
204:                }
205:                break;
206:
207:       case F4  :       /* add data */
208:
209:                okay = TRUE;
210:
211:                if( strlen( groups.group ) == 0 )
212:                {
213:                   display_msg_box("Must enter a group name",
214:                      ct.err_fcol, ct.err_bcol);
215:                   position = 0;
216:                   okay = FALSE;
217:                }
218:                else
219:                {
220:                   /* rest of edits. (i.e. edit date) */
221:                }
222:
223:                if( okay == TRUE )
224:                {
225:                   add_groups_data();
226:
227:                   clear_groups_fields();
```

```
228:                    draw_groups_screen();
229:                    position = 0;
230:                }
231:
232:                break;
233:
234:        case PAGE_DN :   /* go to last data entry field */
235:                position = 11;
236:                break;
237:
238:        case PAGE_UP :   /* go to first data entry field */
239:                position = 0;
240:                break;
241:
242:        case F1:         /* help */
243:                display_groups_help();
244:                draw_groups_screen();
245:                break;
246:
247:        default:         /* error */
248:                display_msg_box( " Error ",
249:                   ct.err_fcol, ct.err_bcol );
250:                break;
251:
252:         }     /* end of switch */
253:     }         /* end of while  */
254:
255:     return( rv );
256: }
257:
258: /*-------------------------------------------------------*
259:  *   get_groups_data()                                   *
260:  *-------------------------------------------------------*/
261:
262: int get_groups_data( int row )
263: {
264:     int  rv;
265:     char tmp[3] = { 0, 0, 0 }; /* initialize to null values */
266:
267:     switch( row )
268:     {
269:        case 0 :
270:           rv = getline( GET_ALPHA, 0,  4, 17, 0, 25, groups.group);
271:           break;
272:        case 1 :
273:           strncpy( tmp, groups.date_formed.month, 2 );
274:           rv = getline( GET_NUM,   0,  6, 17, 0,  2, tmp );
275:           zero_fill_field(tmp, 2);
276:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 6, 17);
```

*continues*

### Listing 13.5. continued

```
277:          strncpy( groups.date_formed.month, tmp, 2);
278:          break;
279:       case 2 :
280:          strncpy( tmp, groups.date_formed.day, 2 );
281:          rv = getline( GET_NUM,   0,  6, 20, 0,  2, tmp );
282:          zero_fill_field(tmp, 2);
283:          write_string(tmp, ct.fld_fcol, ct.fld_bcol, 6, 20);
284:          strncpy( groups.date_formed.day, tmp, 2 );
285:          break;
286:       case 3 :
287:          strncpy( tmp, groups.date_formed.year, 2 );
288:          rv = getline( GET_NUM,   0,  6, 23, 0,  2, tmp );
289:          zero_fill_field(tmp, 2);
290:          write_string(tmp, ct.fld_fcol, ct.fld_bcol, 6, 23);
291:          strncpy( groups.date_formed.year, tmp, 2 );
292:          break;
293:       case 4 :
294:          rv = getline( GET_ALPHA, 0,  8, 19, 0, 20,
                           groups.music_type);
295:          break;
296:       case 5 :
297:          rv = getline( GET_ALPHA, 0, 12,  9, 0, 30,
                           groups.member[0]);
298:          break;
299:       case 6 :
300:          rv = getline( GET_ALPHA, 0, 12, 42, 0, 30,
                           groups.member[1]);
301:          break;
302:       case 7 :
303:          rv = getline( GET_ALPHA, 0, 13,  9, 0, 30,
                           groups.member[2]);
304:          break;
305:       case 8 :
306:          rv = getline( GET_ALPHA, 0, 13, 42, 0, 30,
                           groups.member[3]);
307:          break;
308:       case 9 :
309:          rv = getline( GET_ALPHA, 0, 14,  9, 0, 30,
                           groups.member[4]);
310:          break;
311:       case 10 :
312:          rv = getline( GET_ALPHA, 0, 14, 42, 0, 30,
                           groups.member[5]);
313:          break;
314:       case 11 :
315:          rv = getline( GET_ALPHA, 0, 18, 10, 0, 60,
                           groups.info[0]);
316:          break;
317:       case 12 :
```

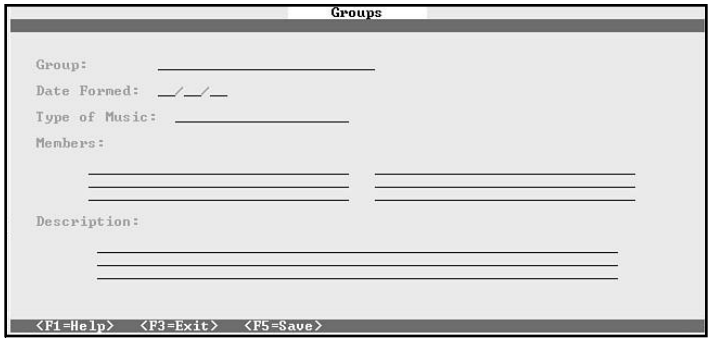
```
318:           rv = getline( GET_ALPHA, 0, 19, 10, 0, 60,
                             groups.info[1]);
319:           break;
320:        case 13 :
321:           rv = getline( GET_ALPHA, 0, 20, 10, 0, 60,
                             groups.info[2]);
322:           break;
323:     }
324:     return( rv );
325: }
326:
327: /*-------------------------------------------------------*
328:  *   clear_groups_fields()                               *
329:  *-------------------------------------------------------*/
330:
331: int clear_groups_fields(void)
332: {
333:   getline( CLEAR_FIELD, 0, 26, 0, 0, 0, groups.group );
334:   getline( CLEAR_FIELD, 0,  2, 0, 0, 0, groups.date_formed.year );
335:   getline( CLEAR_FIELD, 0,  2, 0, 0, 0, groups.date_formed.month );
336:   getline( CLEAR_FIELD, 0,  2, 0, 0, 0, groups.date_formed.day );
337:   getline( CLEAR_FIELD, 0, 21, 0, 0, 0, groups.music_type );
338:   getline( CLEAR_FIELD, 0, 31, 0, 0, 0, groups.member[0] );
339:   getline( CLEAR_FIELD, 0, 31, 0, 0, 0, groups.member[1] );
340:   getline( CLEAR_FIELD, 0, 31, 0, 0, 0, groups.member[2] );
341:   getline( CLEAR_FIELD, 0, 31, 0, 0, 0, groups.member[3] );
342:   getline( CLEAR_FIELD, 0, 31, 0, 0, 0, groups.member[4] );
343:   getline( CLEAR_FIELD, 0, 31, 0, 0, 0, groups.member[5] );
344:   getline( CLEAR_FIELD, 0, 61, 0, 0, 0, groups.info[0] );
345:   getline( CLEAR_FIELD, 0, 61, 0, 0, 0, groups.info[1] );
346:   getline( CLEAR_FIELD, 0, 61, 0, 0, 0, groups.info[2] );
347:
348:   return(0);
349: }
350:
351: /*-------------------------------------------------------*
352:  *   display_groups_help()                               *
353:  *-------------------------------------------------------*/
354:
355: void display_groups_help(void)
356: {
357:   grid( 11, 16, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
358:   box(10, 15, 20, 60, SINGLE_BOX, FILL_BOX,
359:                       ct.help_fcol, ct.help_bcol);
360:
361:   write_string( “This is a screen for entering group”,
362:                 ct.help_fcol, ct.help_bcol, 11, 22 );
363:   write_string( “information and their descriptions.”,
364:                 ct.help_fcol, ct.help_bcol, 12, 22 );
365:
```

*continues*

### Listing 13.5. continued

```
366:   write_string( “To exit screen, press <F3> or <ESC>.”,
367:                 ct.help_fcol, ct.help_bcol, 13, 22 );
368:   write_string( “Press any key to continue...”,
369:                 ct.help_fcol, ct.help_bcol, 14, 22 );
370:
371:   cursor(24, 79);
372:   getch();
373: }
374:
375: /*-------------------------------------------------------*
376:  *  add_groups_data();                                    *
377:  *  Returns:  1 - if all okay                            *
378:  *            0 - if not all okay                        *
379:  *-------------------------------------------------------*/
380:
381: int add_groups_data( void )
382: {
383:    int   rv = 0;
384:    FILE *fp;
385:
386:    if( (fp = fopen( GROUPS_DBF, “a” )) == NULL )
387:    {
388:          display_msg_box(“Error opening file...”,
389:                          ct.err_fcol, ct.err_bcol );
390:    }
391:    else
392:    {
393:       if( (fwrite( p_groups, sizeof( groups ), 1, fp )) == 0 )
394:       {
395:          display_msg_box(“Error writing data...”,
396:                          ct.err_fcol, ct.err_bcol );
397:       }
398:       else
399:       {
400:          display_msg_box(“Record added”, ct.db_fcol, ct.db_bcol);
401:          rv = 1;
402:       }
403:
404:       fclose( fp );
405:    }
406:    return( rv );
407: }
408: /*=======================================================*
409:  *                    end of listing                     *
410:  *=======================================================*/
```

**Output**




**Analysis**

At first glance, this listing may not seen too different from the MEDIUM.C listing (Listing 13.4). The MEDIUM.C listing was actually used as a template to create this listing. All of the medium-specific information was then changed to the corresponding group information. Because a detailed analysis was made on the medium screen, only the major differences need to be covered.

The first change can be seen in line 38. The name of the file being used is more appropriately named `GROUPS.DBF`. Lines 44 and 45 declare a group structure and a pointer using the type-defined constants from the RECORDS.H structure. This structure and the corresponding pointer are used throughout the rest of the listing to hold the data being entered by the user.

Several changes have been made in setting up the screen. In line 65, the `draw_borders()` function is called using the header for groups. In the `draw_groups_prompts()` function in lines 74 to 90, different prompts are displayed. These prompts are more appropriate for the group's information. In line 82, a prompt for a date is displayed. Because only the numeric part of the date will be retrieved, the separators are displayed along with the prompts.

The function to display the fields, `display_groups_fields()`, has some subtle changes also. To conserve on a little bit of storage area, a character array of 30 underlines has been created. This array is used to write several of the field underlines. Line 101 might be a little confusing at first. This is writing the field underlines for the group name. At first glance, the `under_30` character array may seem too long. To get around this, only the last 25 characters are printed. This is accomplished by printing the character array starting at the sixth position—by adding five to its starting address. Line 106 prints 20 underlines in the same manner.

An additional difference in `display_groups_fields()` is the use of the `tmp` character array. This is a temporary character array used to display the individual date members—`month`, `day`, `year`. Because the date fields are stored without null termina-

tors, they can't be displayed like the other null terminated character arrays. To get around this, the date fields are copied into the temporary, `tmp`, field and then the `tmp` field is displayed (lines 147 to 152). When the data is retrieved using `getline()` in the `get_groups_data()` function, this same approach is used. An example of this can be seen in lines 284 to 287.

## Getting the Group Data: The *get_groups_input_data()* Function

Getting data is nearly identical to what was presented in the MEDIUM.C listing. Lines 190 and 195 are changed to reflect that the group's screen has 14 fields. In addition to the other cursor movement functions, the page down function is also modified to set the screen's last position to 13.

The case for the `F4` key in lines 207 to 233 is modified only slightly. This is the function to add the groups record. Before adding the record, the entered data should be verified to ensure that it is accurate. In this listing, the only edit performed is a check on the group name, `group.group`. This name can't be blank. Additional edits, such as an edit on the date, could be added in the `else` statement. A flag called `okay` is used to see if all the edits passed. If they did, the record is added and data entry continues. If the edits didn't pass, a descriptive message is displayed and control is returned to the screen.

The `get_groups_data()` has the actual `getline()` cases for retrieving each field. You should notice that both numbers and characters are retrieved using either `GET_NUM` or `GET_ALPHA` with `getline()`. Several of the `getline()` cases could be expanded to include edits. For example, if the user enters a `month` of 13 in case 1 (lines 272 to 278), then there is an error. You could capture this error at this time, rather than waiting for the user to add the record. One of today's exercises will ask you to add this edit to the group's listing.

**Note:** You can add edits to the `getline()` functions; however, you shouldn't make them mandatory. If the escape key or the `F3` key (for exiting) are pressed, then the user should still be able to leave the screen. It would be up to you to determine if the other keys would enable them to continue.

One last function needs to be reviewed. This is the `zero_fill_field()` function, which is used in several of the `case` statements for getting data. After the call to `getline()`, the field is padded on the left with zeros with this function. The next line

uses `write_string()` to rewrite the field on the screen with the zeros. The `zero_fill_field()` was a part of the RECOFREC.C listing (Listing 13.1).

| DO | DON'T |
|---|---|
| **DO** use edits in your programs to ensure the data entered is valid. | |
| | **DON'T** forget to initialize the data elements in your structures. This way you can be assured that they don't contain garbage. |
| **DO** use Listing 13.4, MEDIUMS.C, as a template for creating your own data entry screens. "Real" programmers only write one program of any given type. They then copy it for a starting point of the next program. | |

# The Musical Items Entry and Edit Screen

Having created the Medium Codes and the Group Information screens, you now only need the Musical Items screen. Listing 13.6 presents the first cut of this screen. The functionality is not complete; however, the frame work is. Several of the advanced features of this listing will be covered on Days 15 through 19. Today, we concentrate solely on the basic data entry functions.

**Note:** This listing should be compiled and linked along with Listings 13.1, 13.2, 13.3, and your TYAC.LIB library. You should uncomment line 58 in Listing 13.1 before compiling and linking.



**Listing 13.6. ALBUMS.C. The Musical Items entry and edit screen.**

```
1:   /*=======================================================
2:    * Filename: albums.c
3:    *
4:    * Author:   Bradley L. Jones
5:    *           Gregory L. Guntle
6:    *
```

*continues*

### Listing 13.6. continued

```
7:    * Purpose:  Allow entry and edit of information on musical
8:    *           items.
9:    *
10:   * Note:     This listing is linked with RECofREC.c
11:   *           (There isn't a main() in this listing!)
12:   *=========================================================*/
13:
14:  #include <stdio.h>
15:  #include <string.h>
16:  #include <conio.h>        /* for getch() */
17:  #include "tyac.h"
18:  #include "records.h"
19:
20:  /*--------------------*
21:   *     prototypes     *
22:   *--------------------*/
23:  #include "recofrec.h"
24:
25:  void draw_albums_screen( void );
26:  void draw_albums_prompts( void );
27:  void display_albums_fields( void );
28:
29:  int  clear_albums_fields(void);
30:  int  get_albums_data( int row );
31:  int  get_albums_input_data( void );
32:  void display_albums_help(void);
33:  int  add_albums_data( void );
34:
35:  /*------------------*
36:   * Defined constants*
37:   *------------------*/
38:
39:  #define ALBUMS_DBF   "ALBUMS.DBF"
40:
41:  /*-----------------------*
42:   * structure declarations *
43:   *-----------------------*/
44:
45:  ALBUM_REC albums;
46:  ALBUM_REC *p_albums = &albums;
47:  SONG_REC  songs[7];
48:
49:  /*=======================================================*
50:   *   do_albums_screen()                                  *
51:   *=======================================================*/
52:
53:  int do_albums_screen(void)
54:  {
55:      clear_albums_fields();
```

```
56:      draw_albums_screen();
57:      get_albums_input_data();
58:      return 0;
59:  }
60:
61:  /*------------------------*
62:   *   draw_albums_screen() *
63:   *------------------------*/
64:
65:  void draw_albums_screen( void )
66:  {
67:     draw_borders("  Musical Items  ");  /* draw screen bckgrnd */
68:
69:     write_string( "<F1=Help>   <F3=Exit>   <F4=Save>",
70:                   ct.abar_fcol, ct.abar_bcol, 24, 3);
71:
72:     draw_albums_prompts();
73:     display_albums_fields();
74:  }
75:
76:  /*------------------------*
77:   *   draw_albums_prompts()*
78:   *------------------------*/
79:
80:  void draw_albums_prompts( void )
81:  {
82:     int  ctr;
83:     char tmp[10];
84:
85:     write_string("Title:",
86:                   ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 4, 3 );
87:     write_string("Group:",
88:                   ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 5, 3 );
89:     write_string("Medium:",
90:                   ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 7, 3 );
91:     write_string("Date Purchased:    /  /",
92:                   ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 9, 3 );
93:     write_string("Cost:  $   .",
94:                   ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 9, 33 );
95:     write_string("Value:  $   .",
96:                   ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 9, 52 );
97:     write_string("Track   Song Title",
98:                   ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 11, 7 );
99:     write_string("Time",
100:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 11, 59 );
101:    for( ctr = 0; ctr < 7; ctr++ )
102:    {
103:        sprintf(tmp, "%02d:",ctr+1);
104:        write_string( tmp,
105:              ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 13+ctr, 8 );
```

*continues*

### Listing 13.6. continued

```
106:        write_string(":",
107:              ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 13+ctr, 61 );
108:    }
109:
110:    write_string("Total Album Time:   :  :",
111:                 ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 21, 39 );
112:
113:    /* Track information */
114:
115: }
116:
117: /*------------------------*
118:  *   draw_albums_fields() *
119:  *------------------------*/
120:
121: void display_albums_fields( void )
122: {
123:    int ctr;
124:    char tmp[4] = { 0, 0, 0, 0 };   /* set to null values */
125:    char under_40[41] =
126:                {"_______________________________"};
127:
128:    write_string(under_40+10,   /* 30 underscores */
129:                 ct.fld_fcol, ct.fld_bcol, 4, 12 );
130:    write_string(under_40+15,     /* 25 underlines */
131:                 ct.fld_fcol, ct.fld_bcol, 5, 12 );
132:    write_string("__", ct.fld_fcol, ct.fld_bcol, 7, 13 );
133:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 20 );
134:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 23 );
135:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 26 );
136:
137:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 41 );
138:    write_string("__",  ct.fld_fcol, ct.fld_bcol, 9, 45 );
139:
140:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 61 );
141:    write_string("__",  ct.fld_fcol, ct.fld_bcol, 9, 65 );
142:
143:    for( ctr = 0; ctr < 7; ctr++)
144:    {
145:       write_string(under_40,
146:                    ct.fld_fcol, ct.fld_bcol, 13+ctr, 16 );
147:       write_string("__",
148:                    ct.fld_fcol, ct.fld_bcol, 13+ctr, 59 );
149:       write_string("__",
150:                    ct.fld_fcol, ct.fld_bcol, 13+ctr, 62 );
151:    }
152:
153:    write_string("__", ct.fld_fcol, ct.fld_bcol, 21, 57);
154:    write_string("__", ct.fld_fcol, ct.fld_bcol, 21, 60);
```

```
155:    write_string("__", ct.fld_fcol, ct.fld_bcol, 21, 63);
156:
157:    /*** display data, if exists ***/
158:
159:    write_string(albums.title, ct.fld_fcol, ct.fld_bcol, 4, 12);
160:    write_string(albums.group, ct.fld_fcol, ct.fld_bcol, 5, 12);
161:    write_string(albums.medium_code,
162:                 ct.fld_fcol, ct.fld_bcol, 7, 13 );
163:
164:    strncpy( tmp, albums.date_purch.month, 2 );
165:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 20 );
166:    strncpy( tmp+4, albums.date_purch.day, 2 );
167:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 23 );
168:    strncpy( tmp, albums.date_purch.year, 2 );
169:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 26 );
170:
171:    strncpy(tmp, albums.cost, 3);
172:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 41 );
173:    strncpy(tmp, albums.cost+3, 2 );
174:    tmp[3] = NULL;
175:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 45 );
176:
177:    strncpy(tmp, albums.value, 3);
178:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 61 );
179:    strncpy(tmp, albums.value+3, 2 );
180:    tmp[3] = NULL;
181:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 65 );
182:
183:    /* song title information */
184:    for( ctr = 0; ctr < 7; ctr++ )
185:    {
186:       write_string(songs[ctr].title,
187:            ct.fld_fcol, ct.fld_bcol, 13+ctr, 16 );
188:       write_string(songs[ctr].minutes,
189:            ct.fld_fcol, ct.fld_bcol, 13+ctr, 69 );
190:       write_string(songs[ctr].seconds,
191:            ct.fld_fcol, ct.fld_bcol, 13+ctr, 62 );
192:      /* calc total here. */
193:    }
194:    /* finish total count and print here */
195: }
196:
197: /*-----------------------------------------------------*
198:  *   get_albums_input_data()                           *
199:  *-----------------------------------------------------*/
200: int get_albums_input_data( void )
201: {
202:     int   position,
203:           rv,
204:           okay,               /* used with edits */
```

*continues*

### Listing 13.6. continued

```
205:           loop = TRUE;
206:
207:     /*  Set up exit keys.  */
208:     static char fexit_keys[ 13 ] = { F1, F3, F4, ESC_KEY, PAGE_DN,
                                          PAGE_UP, CR_KEY, TAB_KEY,
209:                                      ENTER_KEY, SHIFT_TAB,
                                          DN_ARROW, UP_ARROW, NULL };
210:
211:     static char *exit_keys = fexit_keys;
212:     getline( SET_EXIT_KEYS, 0, 12, 0, 0, 0, exit_keys );
213:
214:     /*** setup colors and default keys ***/
215:     getline( SET_DEFAULTS, 0, 0, 0, 0, 0, 0 );
216:     getline( SET_NORMAL, 0, ct.fld_fcol, ct.fld_bcol,
217:                             ct.fld_high_fcol, ct.fld_high_bcol, 0 );
218:     getline( SET_UNDERLINE, 0, ct.fld_fcol, ct.fld_bcol,
219:                             ct.fld_high_fcol, ct.fld_high_bcol, 0 );
220:     getline( SET_INS, 0, ct.abar_fcol, ct.abar_bcol, 24, 76, 0 );
221:
222:     position = 0;
223:
224:     while( loop == TRUE )      /**  get data for top fields  **/
225:     {
226:        switch( (rv = get_albums_data( position )) )
227:        {
228:       case CR_KEY    :
229:       case TAB_KEY   :
230:       case ENTER_KEY :
231:       case DN_ARROW  : /* go down a field */
232:                ( position == 30 ) ? ( position = 0 ) : position++;
233:                break;
234:
235:       case SHIFT_TAB :
236:       case UP_ARROW  : /* go up a field */
237:                ( position == 0 ) ? ( position = 30 ) : position--;
238:                break;
239:
240:       case ESC_KEY :
241:       case F3      :  /* exit back to main menu */
242:                if( (yes_no_box( "Do you want to exit?",
243:                       ct.db_fcol, ct.db_bcol )) == 'Y' )
244:                {
245:                    loop = FALSE;
246:                }
247:                break;
248:
249:       case F4  :   /* add data */
250:
251:                okay = TRUE;
252:
```

```
253:                if( strlen( albums.title ) == 0 )
254:                {
255:                   display_msg_box("Must enter a Title",
256:                      ct.err_fcol, ct.err_bcol);
257:                   position = 0;
258:                   okay = FALSE;
259:                }
260:                else
261:                {
262:                   /* edit date */
263:
264:                }
265:
266:                if( okay == TRUE )
267:                {
268:                   add_albums_data();
269:
270:                   clear_albums_fields();
271:                   draw_albums_screen();
272:                   position = 0;
273:                }
274:
275:                break;
276:
277:       case PAGE_DN :   /* go to last data entry field */
278:                position = 30;
279:                break;
280:
281:       case PAGE_UP :   /* go to first data entry field */
282:                position = 0;
283:                break;
284:
285:       case F1:         /* help */
286:                display_albums_help();
287:                draw_albums_screen();
288:                break;
289:
290:       default:         /* error */
291:                display_msg_box( " Error ",
292:                   ct.err_fcol, ct.err_bcol );
293:                break;
294:
295:         }     /* end of switch */
296:     }         /* end of while  */
297:
298:     return( rv );
299: }
300:
```

*continues*

### Listing 13.6. continued

```
301: /*-----------------------------------------------------*
302:  *   get_albums_data()                                  *
303:  *-----------------------------------------------------*/
304:
305: int get_albums_data( int row )
306: {
307:     int  rv;
308:     char tmp[4] = { 0, 0, 0, 0 }; /* set to null values */
309:
310:     switch( row )
311:     {
312:        case 0 :
313:           rv = getline( GET_ALPHA, 0,  4, 12, 0, 30, albums.title);
314:           break;
315:        case 1 :
316:           rv = getline( GET_ALPHA, 0,  5, 12, 0, 25, albums.group);
317:           break;
318:        case 2 :
319:           rv = getline( GET_ALPHA, 0,  7, 13, 0,  2,
                           albums.medium_code);
320:           break;
321:        case 3 :
322:           strncpy( tmp, albums.date_purch.month, 2 );
323:           rv = getline( GET_NUM,   0,  9, 20, 0,  2, tmp );
324:           zero_fill_field(tmp, 2);
325:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 20);
326:           strncpy( albums.date_purch.month, tmp, 2);
327:           break;
328:        case 4 :
329:           strncpy( tmp, albums.date_purch.day, 2 );
330:           rv = getline( GET_NUM,   0,  9, 23, 0,  2, tmp );
331:           zero_fill_field(tmp, 2);
332:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 23);
333:           strncpy( albums.date_purch.day, tmp, 2 );
334:           break;
335:        case 5 :
336:           strncpy( tmp, albums.date_purch.year, 2 );
337:           rv = getline( GET_NUM,   0,  9, 26, 0,  2, tmp );
338:           zero_fill_field(tmp, 2);
339:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 26);
340:           strncpy( albums.date_purch.year, tmp, 2 );
341:           break;
342:        case 6 :
343:           strncpy( tmp, albums.cost, 3 );
344:           rv = getline( GET_NUM,   0,  9, 41, 0,  3, tmp );
345:           zero_fill_field(tmp, 3);
346:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 41);
347:           strncpy( albums.cost, tmp, 3 );
348:           break;
```

```
349:         case 7 :
350:            strncpy( tmp, albums.cost+3, 2 );
351:            rv = getline( GET_NUM,   0,  9, 45, 0,  2, tmp );
352:            zero_fill_field(tmp, 2);
353:            write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 45);
354:            strncpy( albums.cost+3, tmp, 2 );
355:            break;
356:         case 8 :
357:            strncpy( tmp, albums.value, 3 );
358:            rv = getline( GET_NUM,   0,  9, 61, 0,  3, tmp );
359:            zero_fill_field(tmp, 3);
360:            write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 61);
361:            strncpy( albums.value, tmp, 3 );
362:            break;
363:         case 9 :
364:            strncpy( tmp, albums.value+3, 2 );
365:            rv = getline( GET_NUM,   0,  9, 65, 0,  2, tmp );
366:            zero_fill_field(tmp, 2);
367:            write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 65);
368:            strncpy( albums.value+3, tmp, 2 );
369:            break;
370:         case 10 :
371:            rv = getline( GET_ALPHA, 0,  13, 16, 0, 40,
372:                                              songs[0].title);
373:            break;
374:         case 11 :
375:            rv = getline( GET_NUM, 0, 13, 59, 0, 2,
376:                                              songs[0].minutes);
377:            zero_fill_field(songs[0].minutes, 2);
378:            write_string(songs[0].minutes,
379:                         ct.fld_fcol, ct.fld_bcol, 13, 59);
380:            break;
381:         case 12 :
382:            rv = getline( GET_NUM, 0, 13, 62, 0, 2,
383:                                              songs[0].seconds);
384:            zero_fill_field(songs[0].seconds, 2);
385:            write_string(songs[0].seconds,
386:                         ct.fld_fcol, ct.fld_bcol, 13, 62);
387:            break;
388:         case 13 :
389:            rv = getline( GET_ALPHA, 0, 14, 16, 0, 40,
390:                                              songs[1].title);
391:            break;
392:         case 14 :
393:            rv = getline( GET_NUM, 0, 14, 59, 0, 2,
394:                                              songs[1].minutes);
395:            zero_fill_field(songs[1].minutes, 2);
396:            write_string(songs[1].minutes,
397:                         ct.fld_fcol, ct.fld_bcol, 14, 59);
398:            break;
```

*continues*

### Listing 13.6. continued

```
399:         case 15 :
400:            rv = getline( GET_NUM, 0, 14, 62, 0, 2,
401:                                                songs[1].seconds);
402:            zero_fill_field(songs[1].seconds, 2);
403:            write_string(songs[1].seconds,
404:                         ct.fld_fcol, ct.fld_bcol, 14, 62);
405:            break;
406:         case 16 :
407:            rv = getline( GET_ALPHA, 0, 15, 16, 0, 40,
408:                                                songs[2].title);
409:            break;
410:         case 17 :
411:            rv = getline( GET_NUM, 0, 15, 59, 0, 2,
412:                                                songs[2].minutes);
413:            zero_fill_field(songs[2].minutes, 2);
414:            write_string(songs[2].minutes,
415:                         ct.fld_fcol, ct.fld_bcol, 15, 59);
416:            break;
417:         case 18 :
418:            rv = getline( GET_NUM, 0, 15, 62, 0, 2,
419:                                                songs[2].seconds);
420:            zero_fill_field(songs[2].seconds, 2);
421:            write_string(songs[2].seconds,
422:                         ct.fld_fcol, ct.fld_bcol, 15, 62);
423:            break;
424:         case 19 :
425:            rv = getline( GET_ALPHA, 0, 16, 16, 0, 40,
426:                                                songs[3].title);
427:            break;
428:         case 20 :
429:            rv = getline( GET_NUM, 0, 16, 59, 0, 2,
430:                                                songs[3].minutes);
431:            zero_fill_field(songs[3].minutes, 2);
432:            write_string(songs[3].minutes,
433:                         ct.fld_fcol, ct.fld_bcol, 16, 59);
434:            break;
435:         case 21 :
436:            rv = getline( GET_NUM, 0, 16, 62, 0, 2,
437:                                                songs[3].seconds);
438:            zero_fill_field(songs[3].seconds, 2);
439:            write_string(songs[3].seconds,
440:                         ct.fld_fcol, ct.fld_bcol, 16, 62);
441:            break;
442:         case 22 :
443:            rv = getline( GET_ALPHA, 0, 17, 16, 0, 40,
444:                                                songs[4].title);
445:            break;
446:         case 23 :
447:            rv = getline( GET_NUM, 0, 17, 59, 0, 2,
448:                                                songs[4].minutes);
```

```
449:            zero_fill_field(songs[4].minutes, 2);
450:            write_string(songs[4].minutes,
451:                         ct.fld_fcol, ct.fld_bcol, 17, 59);
452:            break;
453:         case 24 :
454:            rv = getline( GET_NUM, 0, 17, 62, 0, 2,
455:                                                songs[4].seconds);
456:            zero_fill_field(songs[4].seconds, 2);
457:            write_string(songs[4].seconds,
458:                         ct.fld_fcol, ct.fld_bcol, 17, 62);
459:            break;
460:         case 25 :
461:            rv = getline( GET_ALPHA, 0, 18, 16, 0, 40,
462:                                                songs[5].title);
463:            break;
464:         case 26 :
465:            rv = getline( GET_NUM, 0, 18, 59, 0, 2,
466:                                                songs[5].minutes);
467:            zero_fill_field(songs[5].minutes, 2);
468:            write_string(songs[5].minutes,
469:                         ct.fld_fcol, ct.fld_bcol, 18, 59);
470:            break;
471:         case 27 :
472:            rv = getline( GET_NUM, 0, 18, 62, 0, 2,
473:                                                songs[5].seconds);
474:            zero_fill_field(songs[5].seconds, 2);
475:            write_string(songs[5].seconds,
476:                         ct.fld_fcol, ct.fld_bcol, 18, 62);
477:            break;
478:         case 28 :
479:            rv = getline( GET_ALPHA, 0, 19, 16, 0, 40,
480:                                                songs[6].title);
481:            break;
482:         case 29 :
483:            rv = getline( GET_NUM, 0, 19, 59, 0, 2,
484:                                                songs[6].minutes);
485:            zero_fill_field(songs[6].minutes, 2);
486:            write_string(songs[6].minutes,
487:                         ct.fld_fcol, ct.fld_bcol, 19, 59);
488:            break;
489:         case 30 :
490:            rv = getline( GET_NUM, 0, 19, 62, 0, 2,
491:                                                songs[6].seconds);
492:            zero_fill_field(songs[6].seconds, 2);
493:            write_string(songs[6].seconds,
494:                         ct.fld_fcol, ct.fld_bcol, 19, 62);
495:            break;
496:      }
497:      return( rv );
```

*continues*

### Listing 13.6. continued

```
498: }
499:
500: /*-----------------------------------------------------*
501:  *   clear_albums_fields()                             *
502:  *-----------------------------------------------------*/
503:
504: int clear_albums_fields(void)
505: {
506:   int ctr;
507:
508:   getline( CLEAR_FIELD, 0, 31, 0, 0, 0, albums.title );
509:   getline( CLEAR_FIELD, 0, 26, 0, 0, 0, albums.group );
510:   getline( CLEAR_FIELD, 0,  3, 0, 0, 0, albums.medium_code );
511:   getline( CLEAR_FIELD, 0,  2, 0, 0, 0, albums.date_purch.month );
512:   getline( CLEAR_FIELD, 0,  2, 0, 0, 0, albums.date_purch.day );
513:   getline( CLEAR_FIELD, 0,  2, 0, 0, 0, albums.date_purch.year );
514:   getline( CLEAR_FIELD, 0,  6, 0, 0, 0, albums.cost );
515:   getline( CLEAR_FIELD, 0,  6, 0, 0, 0, albums.value );
516:   albums.nbr_songs = 0;
517:   for( ctr = 0; ctr < 7; ctr++ )
518:   {
519:      getline( CLEAR_FIELD, 0, 41, 0, 0, 0, songs[ctr].title );
520:      getline( CLEAR_FIELD, 0,  3, 0, 0, 0, songs[ctr].minutes );
521:      getline( CLEAR_FIELD, 0,  3, 0, 0, 0, songs[ctr].seconds );
522:   }
523:   return(0);
524: }
525:
526: /*-----------------------------------------------------*
527:  *   display_albums_help()               *
528:  *-----------------------------------------------------*/
529:
530: void display_albums_help(void)
531: {
532:   grid( 11, 16, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
533:   box(10, 15, 20, 60, SINGLE_BOX, FILL_BOX,
534:                       ct.help_fcol, ct.help_bcol);
535:
536:   write_string( "This is a screen for entering musical",
537:                 ct.help_fcol, ct.help_bcol, 11, 22 );
538:   write_string( "items such as albums.",
539:                 ct.help_fcol, ct.help_bcol, 12, 22 );
540:
541:   write_string( "To exit screen, press <F3> or <ESC>.",
542:                 ct.help_fcol, ct.help_bcol, 13, 22 );
543:   write_string( "Press any key to continue...",
544:                 ct.help_fcol, ct.help_bcol, 14, 22 );
545:
```

```
546:   cursor(24, 79);
547:   getch();
548: }
549:
550: /*-------------------------------------------------------*
551:  *  add_albums_data();                                    *
552:  *  Returns:  1 - if all okay                            *
553:  *            0 - if not all okay                        *
554:  *  Note:     Titles are not written to file. These      *
555:  *            will be covered on a later day.            *
556:  *-------------------------------------------------------*/
557:
558: int add_albums_data( void )
559: {
560:    int   rv = 0;
561:    FILE *fp;
562:
563:    if( (fp = fopen( ALBUMS_DBF, "a" )) == NULL )
564:    {
565:          display_msg_box("Error opening file...",
566:                          ct.err_fcol, ct.err_bcol );
567:    }
568:    else
569:    {
570:       if( (fwrite( p_albums, sizeof( albums ), 1, fp )) == 0 )
571:       {
572:          display_msg_box("Error writing data...",
573:                          ct.err_fcol, ct.err_bcol );
574:       }
575:       else
576:       {
577:          display_msg_box("Record added", ct.db_fcol, ct.db_bcol);
578:          rv = 1;
579:       }
580:
581:       fclose( fp );
582:    }
583:    return( rv );
584: }
585:
586: /*=======================================================*
587:  *                    end of listing                     *
588:  *=======================================================*/
```

Output

```
                              Musical Items

Title:  _____________________________
Group:  _____________________

Medium:  __

Date Purchased:  __/__/__     Cost:  $___.__     Value:  $___.__

    Track   Song Title                                    Time

     01:    _____________________________________        __:__
     02:    _____________________________________        __:__
     03:    _____________________________________        __:__
     04:    _____________________________________        __:__
     05:    _____________________________________        __:__
     06:    _____________________________________        __:__
     07:    _____________________________________        __:__

                                        Total Album Time:  __:__:__

<F1=Help>   <F3=Exit>   <F5=Save>
```

This listing is almost identical in functionality to the Group Information listing. There is only one subtle difference in the display of the title fields. When the file access controls are covered on Day 19, this listing will be equipped with the capability to add a variable number of record titles. In this version of the listing, only seven titles can be entered. When adding the data with the F4 key, only the album information is added; the individual titles are not. Again, this will be rectified on Day 19 when file access is covered.

You should note each of the areas that are affected by the titles. The first area is the additional structure in line 47. Because only seven titles are worked with at this time, an array is created that will hold only seven. In lines 101 to 108, the prompts for the titles are displayed on the screen. Rather than display each individually, a loop is used. Similar loops are used in lines 143 to 151 to display the field underlines and in lines 184 to 193 to display the field values.

In lines 342 to 354, cases are presented to retrieve the cost of the album. Because the cost can be a decimal number, the input is retrieved in two pieces. First, the numbers to the left of the decimal are retrieved. This is followed by retrieving the numbers to the right. While this works, it isn't an optimal solution. You should consider modifying your `getline()` function to accept decimal numbers using one of the unused option numbers.

# Synopsis: Where Do You Go from Here?

Today, you have covered a great deal of code. While your application is beginning to do quite a bit, it isn't complete. In fact, you have just begun. On Day 14, you will cover menuing, which will help complete your application's user interface. Day 15 will give

your application the capability to work with the data files. On this day your application will be almost fully functional; there will still be more to do. Days 16 and 17 will help bring the application to near completion. Day 16 will aid you in adding help to your program. This will include an easy way of providing context sensitive help. Day 17 will add the little features that were mentioned in the specification. In addition, Day 17 will help you create a screen that will enable the user of a program to change the application's colors. When you complete Day 17, you will have completed your application with the exception of testing and reporting.

# Summary

Today is what can be considered an exciting day. Using all of the functions that you have created on the previous days, along with the specification from Day 12, you began your application. Before starting, some of the standards for creating an entry and edit screen were covered. With these in mind, the code was then presented. First, a simplistic menu was presented, which will be replaced on Day 14. After this, you were presented with three listings that perform each of the entry and edit windows in the specification from Day 12. Detailed analysis helped you understand exactly what is going on.

# Q&A

**Q Are there different kinds of edits?**

**A** Yes, three different types of edits are generally performed: pre-edits, post-edits, and process edits.

**Q What are the differences among pre-edits, post-edits, and process edits?**

**A** The pre-edits occur when a field is first entered. A post-edit occurs when leaving a field. A process edit occurs when the user is ready to process the information on the entire screen. Common pre-edits involve initializations and calculations. Post-edits are a more popular edit. They generally validate information entered into a field. Process edits ensure all the necessary fields are entered in addition to performing edits involving more than one field.

**Q Should I use my creativity to reuse the available function keys?**

**A** You should not try to make up your own uses for the keys on the keyboard. At the beginning of today's material, some of the standard uses for keys—and the benefits of using them—were presented. If you were to use the F1 key to delete a record, many people would inadvertently delete records when they tried to get help. (F1 is generally used for help.)

**Q Are there published standards for creating applications?**

**A** IBM has created standards for creating applications. These standards are referred to in its Systems Application Architecture (SAA) documents. IBM's standards for screen interfaces are referred to as Common User Access (CUA) standards. Microsoft has also developed standards for developing Windows applications.

# Workshop

The Workshop provides quiz questions to help you solidify your understanding of the material covered and exercises to provide you with experience in using what you've learned.

## Quiz

1. What are some of the benefits that can be gained by following standards—even if the standards are informal?
2. What should the F1 function key do?
3. What should the F3 function key do?
4. What should the F5 function key do?
5. If you were given a message box that was red with yellow letters, should you be concerned?
6. True or false: A beep can serve several purposes in an application. You can have it beep when the application starts, when a record is added, when an error occurs, and when the user exits.
7. What is an easy approach to creating your own entry and edit screens?

# Exercises

1. Add an edit to the Group Information screen. This should be a post-edit for the month field. If a value is entered, it should be from 1 to 12. Any other value should receive an error message. (Keep in mind, if a key such as the Escape key or F3 is entered, it should still be processed.)
2. **ON YOUR OWN:** Add additional edits to the preceding listings. Use your own judgment on what edits should be added. Edits can include edits on cost, dates, times, and more.
3. **ON YOUR OWN:** Create an application of your own. Create a contact system that contains an entry screen for entering the names, addresses, and phone numbers of each person.

**Note:** Because Exercise 3 will take you some time, only three exercises are presented. On following days, you will only be presented with a few exercises. Some of these exercises will ask you to expand on the application you developed today. This application is from Exercise 3.