Enhancing the User Interface: Menuing

WEEK
2

On Day 13, you created a temporary menu that was somewhat cryptic in regard to functionality. While the menu worked, it was prone to having the user enter information that was not valid. Today, you'll learn how to create and use a much better menu. Today you will:

- ☐ Learn about adding menus to your application.
- ☐ Learn some suggestions for creating menus.
- ☐ Add menus to your application's front end.
- ☐ Add a menu to the entry and edit portion of your application.

What Is Considered a Menu?

A menu is a list of choices available for selection. These choices can usually be made in one of several ways. Figure 14.1 presents a menu along with tags showing various ways to select an item.

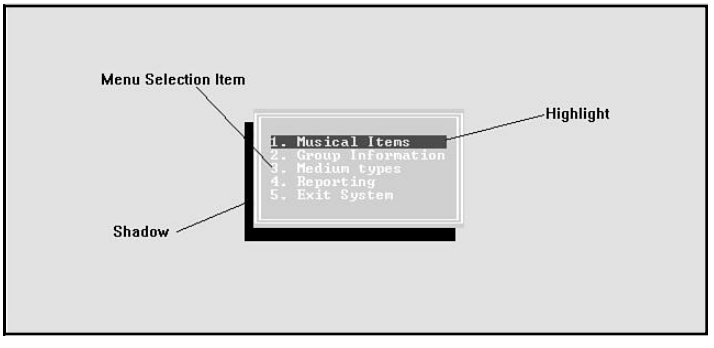


Figure 14.1. *A menu.*

The menu in Figure 14.1 is a menu that you will actually create today. This menu is like many menus in that it offers several ways of selecting an item. The most obvious way of selection is to use the cursor keys to scroll the highlight bar up and down. Additionally, the Home and End keys place the cursor on the first or last menu item. Each row of the menu, or each selection item, has several other ways that it can be selected. The numeric value to the left of each option can be used. For example, if you press 4, the Exit program option will be highlighted. In addition to the obvious numeric value, a mnemonic character can be used. For example, you can type E or e to also select Exit program. The mnemonic key used is generally the first letter, or the

most descriptive letter of the selection item. The actual selection of an item is made by pressing Enter. Additionally, other keys may apply when you are on a menu. In the menu presented in Figure 14.1, if the Escape key is pressed, it may act in the same manner as if menu selection 4 were selected.

Expert Tip: In most cases, the mnemonic key is the first letter of the menu item. Although it starts with E, when Exit is a menu item, it is a common practice to use X as the mnemonic.

Displaying a Menu

To display a menu is not a quick process; however, taken in stride, it isn't too difficult. Listing 14.1 presents a function called `display_menu()`. In this listing, you'll notice a lot of similarities to the `getline()` function that was presented on Day 10. This listing will be explained throughout the sections immediately following it.

Listing 14.1. MENU.C. The `display_menu()` routine.

```
1:   /* ----------------------------------------------------------
2:    * Program: MENU.C
3:    * Authors: Bradley L. Jones
4:    *          Gregory L. Guntle
5:    * Purpose: Manages a menu on the screen
6:    *
7:    * Enter with: * row, col - for positioning menu
8:    *             * box_type - SINGLE_BOX, DOUBLE_BOX or NOBOX
9:    *             * array of ptrs to the menu items to display
10:   *               array is setup this way:
11:   *                   char *array_name[10] = { "menu item",
12:   *                                            "keys" };  etc.
13:   *             * nbr_items - number of menu items (incl keys)
14:   *             * exit_keys - special keys for exit menu
15:   *                           (F3, etc)
16:   *             * address to return menu item selected in
17:   *             * arrow flag - should R/L arrows be used
18:   *             * do_shadow  - should shadow be displayed?
19:   *
20:   * Returns: The exit key and modifies one parm to hold the
21:   *          selected item.
22:   *
23:   * Note(s): The cursor is turned off. Calling function
```

continues

Listing 14.1. continued

```
24:   *          must turn it back on if it is needed.
25:   *=================================================*/
26:
27:  #include <string.h>
28:  #include <conio.h>
29:  #include "tyac.h"
30:  #include "colortbl.h"
31:
32:  /* ================================================ *
33:   * External Global variables defined in MAIN       *
34:   * ================================================ */
35:
36:  extern struct color_table ct;
37:
38:  /* ================================================ *
39:   * Global variables for other functions in this file *
40:   * ================================================ */
41:
42:  int  row, col;
43:  int  nbr;
44:  char **menu_ptr;
45:  char *EXIT_KEYS;
46:  int  nbr_exit_keys;
47:
48:  /* ================================================ *
49:   * Function declarations used in this file         *
50:   * ================================================ */
51:
52:  int display_menu(int, int,    int,   char **,
53:                   int, char *, int *, int, int );
54:
55:  void rewrite_menu_items(int, int);
56:  int check_menu_keys(char);
57:
58:  /* ------------------------------------------------ *
59:   * DISPLAY_MENU:                                    *
60:   *                                                  *
61:   * This function does the real chore in handling    *
62:   * menus.                                           *
63:   * ------------------------------------------------ */
64:
65:  int display_menu(int  srow,   int scol,      int  box_type,
66:                   char **menu, int nbr_items, char *exit_keys,
67:                   int *sel,    int arr_flg,   int  do_shadow )
68:  {
69:    int  i=0;
70:    int  menu_pos;          /* Maintaining menu selections */
71:    int  old_menu_pos;
72:    int  loop_exit = FALSE;
```

```
    int  ch;                /* Character pressed */
    int  max_len  = 0;      /* max string length in array */
    int  temp_len = 0;
    int  key_found;         /* Indicate if key is exit keys */
    char *ptr_to_key;       /* Holds comp val for matching keys */

    /*-------------------------------------------------*
     * Set up global variables - other functions to use *
     *-------------------------------------------------*/

        /* Set Global ptr to menu items */
    menu_ptr = menu;
    row = srow;
    col = scol;
    EXIT_KEYS = exit_keys;
    nbr = nbr_items;
        /* number of exit keys */
    nbr_exit_keys = strlen(EXIT_KEYS);

  /* --------------------------- */
  /*   Calculate string lengths   */
  /* --------------------------- */

    while( i < (nbr/2) )
    {
       temp_len = strlen( *(menu+(i*2)) );
       if(temp_len > max_len )
       {
           max_len = temp_len;
       }
       i++;
    }

    nbr = nbr_items/2;        /* Exclude keys of selection */

  /* ----------------------------- */
  /*   If Box is needed Draw now    */
  /* ----------------------------- */

    if (box_type != 0)
    {
       if( do_shadow == SHADOW )
       {
          grid(row, row+nbr+1, col-3, col+max_len,
               ct.shdw_fcol, ct.bg_bcol, 2);
       }

       box( row-1, row+nbr, col-2, col+max_len+1,
            box_type, FILL_BOX, ct.abar_fcol, ct.abar_bcol);
    }
```

continues

Listing 14.1. continued

```
123:
124: /* ----------------------------- */
125: /*         Display Menu          */
126: /* ----------------------------- */
127:
128:   /* Display menu */
129:   for (i=0; i<nbr; i++)
130:   {
131:     write_string( *(menu+(i*2)),
132:                   ct.menu_fcol, ct.menu_bcol, row+i, col);
133:   }
134:
135:   /*  Highlight first menu item */
136:   write_string( *(menu), ct.menu_high_fcol,
137:                          ct.menu_high_bcol, row, col);
138:
139: /* ------------------------------------------------- */
140:
141:    cursor_off();         /* Turn off cursor */
142:    old_menu_pos = 1;     /* Track selection prior */
143:    menu_pos = 1;         /* Track current sel */
144:
145:    while (loop_exit == FALSE)
146:    {
147:      if ( ( ch=getch() ) == 0 )
148:      {
149:        /* Scan code so read next byte */
150:        ch = getch();
151:        switch ( ch )
152:        {
153:           case HOME: /* goto to TOP of menu */
154:                      menu_pos = 1;
155:                      rewrite_menu_items(menu_pos,
156:                                         old_menu_pos);
157:                      old_menu_pos = menu_pos;
158:                      break;
159:
160:           case END:  /*  goto LAST menu item */
161:                      menu_pos = nbr;
162:                      rewrite_menu_items(menu_pos,
163:                                         old_menu_pos);
164:                      old_menu_pos = menu_pos;
165:                      break;
166:
167:           case RT_ARROW: /* Is LR Arrow movement allowed ?*/
168:                          if ( arr_flg == NO_LR_ARROW)
169:                          {
170:                             /* No - treat like DN_ARROW */
171:                             menu_pos++;
```

```
172:                            if (menu_pos > nbr)
173:                                menu_pos = 1;
174:                            rewrite_menu_items(menu_pos,
175:                                               old_menu_pos);
176:                            old_menu_pos = menu_pos;
177:                         }
178:                         else
179:                         {
180:                            /* LR movement allowed */
181:                            loop_exit = TRUE;
182:                         }
183:                         break;
184:
185:            case LT_ARROW: /* Is LR Arrow movement allowed ?*/
186:                         if ( arr_flg == NO_LR_ARROW)
187:                         {
188:                            menu_pos--;
189:                            if (menu_pos < 1)   /* At end ? */
190:                                menu_pos = nbr;
191:                            rewrite_menu_items(menu_pos,
192:                                               old_menu_pos);
193:                            old_menu_pos = menu_pos;
194:                         }
195:                         else
196:                         {
197:                            /* LR movement allowed */
198:                            loop_exit = TRUE;
199:                         }
200:                         break;
201:
202:            case DN_ARROW: /* Move DOWN one menu selection */
203:                         menu_pos++;
204:                         if (menu_pos > nbr)
205:                             menu_pos = 1;
206:                         rewrite_menu_items(menu_pos,
207:                                            old_menu_pos);
208:                         old_menu_pos = menu_pos;
209:                         break;
210:
211:            case UP_ARROW:  /* Move UP one menu selection */
212:                          menu_pos--;
213:                          if (menu_pos < 1)   /* At end ? */
214:                              menu_pos = nbr;
215:                          rewrite_menu_items(menu_pos,
216:                                             old_menu_pos);
217:                          old_menu_pos = menu_pos;
218:                          break;
219:
220:            default:    loop_exit = check_menu_keys(ch);
221:                        if ( loop_exit == FALSE )
```

continues

Listing 14.1. continued

```
                    {
                       /* key a valid exit key ?*/
                       boop();
                    }
                    break;
      }   /* end of switch  */
   }     /* end of if  */
   else
   {
      switch ( ch )    /* test for other special keys */
      {
         case CR_KEY:  loop_exit = TRUE;
                       break;

         case ESC_KEY: /* is ESC_KEY an exit key? */

                       i = 0;
                       while(i < nbr_exit_keys && !loop_exit)
                       {
                         if ( ch == EXIT_KEYS[i++])
                         {
                            loop_exit=TRUE;
                         }
                       }

                       if( !loop_exit )
                       {
                          boop();
                       }

                       break;

         default: /* Search thru valid keys on Menu items */
                  i=0;
                  key_found = FALSE;
                  while (i<nbr && !key_found)
                  {
                     ptr_to_key = strchr( *(menu+(i*2)+1), ch );
                     if (!ptr_to_key)
                     {
                        /* Not found - look at next one */
                        i++;
                     }
                     else
                     {
                        /* found key - exit */
                        key_found=TRUE;
                     }
                  }
```

```
271:
272:                     if (!key_found)
273:                     {
274:                        boop();
275:                     }
276:                     else
277:                     {
278:                        /* Found letter - position menu sel */
279:                        menu_pos = i+1;
280:                        rewrite_menu_items(menu_pos,
281:                                           old_menu_pos);
282:                        old_menu_pos = menu_pos;
283:                     }
284:                     break;
285:
286:          }   /* end of switch */
287:       }     /* end of else */
288:
289:    }       /* end of while loop */
290:
291:    *sel = menu_pos;
292:    return(ch);
293:
294: }  /* end of subroutine display_menu */
295:
296:
297: /* ------------------------------------------- *
298:  * function: rewrite_menu_items()              *
299:  *                                             *
300:  * This subroutine redisplays the menu items.  *
301:  * The previous selection in NORMAL colors     *
302:  * and the new selections in HIGHLIGHTED colors.  *
303:  * ------------------------------------------- */
304:
305: void rewrite_menu_items( int new_pos, int old_pos )
306: {
307:    /* rewrite last selection - normal colors */
308:    write_string( *(menu_ptr+((old_pos-1)*2)),
309:                  ct.menu_fcol, ct.menu_bcol,
310:                  row+old_pos-1, col);
311:
312:    /* Now rewrite new one w/selections color */
313:    write_string( *(menu_ptr+((new_pos-1)*2)),
314:                  ct.menu_high_fcol, ct.menu_high_bcol,
315:                  row+new_pos-1, col);
316: }
317:
318: /* ------------------------------------------- *
319:  * function: check_menu_keys()                 *
320:  *                                             *
```

continues

Listing 14.1. continued

```
321:  * This subroutine checks the key pressed against *
322:  * a list of keys that can end the procedure.     *
323:  * It receives the key pressed and returns TRUE   *
324:  * if key is in the list, else FALSE if not in    *
325:  * list.                                          *
326:  * ---------------------------------------------- */
327:
328: int check_menu_keys(char key_pressed)
329: {
330:       /* return a true or false to return_code */
331:   int return_code=FALSE;
332:   int loop_ctr = 0;
333:
334:   while ( loop_ctr < nbr_exit_keys && !return_code)
335:   {
336:     if ( key_pressed == EXIT_KEYS[loop_ctr++])
337:     {
338:        return_code=TRUE;
339:     }
340:   }
341:
342:   return(return_code);
343: }
```

Analysis The `display_menu()` function includes several headers. You should notice that two local header files are included in the listing rather than just the TYAC.H header that you have seen in most of the functions presented so far. The new header is COLORTBL.H. This is a header file that contains a color table structure that is almost identical to the structure in RECORD.H from Day 13. Listing 14.2 presents the COLORTBL.H header file.

Type

Listing 14.2. COLORTBL.H. The color table.

```
1:   /*============================================================
2:    * Filename: COLORTBL.H
3:    *
4:    * Author:   Bradley L. Jones & Gregory L. Guntle
5:    *
6:    * Purpose:  Header file for color table definition
7:    *============================================================*/
8:
9:   #ifndef __COLORTBL
10:  #define __COLORTBL
11:
12:  /*--------------------*
```

```
 * color table        *
 *--------------------*/

struct color_table
{
    int  bg_fcol;           /* background */
    int  bg_bcol;

    int  fld_prmpt_fcol;    /* field prompt */
    int  fld_prmpt_bcol;

    int  fld_fcol;          /* input field */
    int  fld_bcol;

    int  fld_high_fcol;     /* highlight character */
    int  fld_high_bcol;

    int  ttl_fcol;          /* screen title */
    int  ttl_bcol;

    int  abar_fcol;         /* action bar & bottom */
    int  abar_bcol;

    int  menu_fcol;         /* menu text */
    int  menu_bcol;

    int  menu_high_fcol;    /* Highlighted menu line */
    int  menu_high_bcol;

    int  err_fcol;          /* error */
    int  err_bcol;

    int  db_fcol;           /* dialog box & msg box */
    int  db_bcol;

    int  help_fcol;         /* help box colors */
    int  help_bcol;

    int  shdw_fcol;         /* shadow color */
};

/*----------------------------------------*
 * extern declarations for global variables *
 *----------------------------------------*/

extern struct color_table ct;

#endif
/*========================================================*
 *                    end of header                       *
 *========================================================*/
```

As you can see, this listing is just the color table. The inclusion of four more color variables has been made to the color table. These are in lines 36 to 40. The values stored in these variables will be used in drawing the menu (`menu_fcol` and `menu_bcol`) and highlighting the currently selected menu item (`menu_high_fcol` and `menu_high_bcol`).

You may wonder why the RECORD.H header file was not included instead of the new COLORTBL.H header file. While the RECORD.H header file already had a color table structure, it also contains structures that are specific to the *Record of Records!* application. The `display_menu()` function will be added to the library. In any applications that use the `display_menu()` function, the color table will need to be included. The other structures that are included in the RECORD.H header file will not be needed by other applications; however, the color table will. By moving the color table into its own file, it can be included wherever needed.

Tip: Remove the color table structure from your RECORD.H file and include the COLORTBL.H header instead. You can include the color table header file in the RECORD.H header file.

The Parameters for the *display_menu()* Function

The `display_menu()` function can be reviewed starting at the top. Like most of the functions presented in this book, the `display_menu()` function's listing starts with several lines of comments. You should pay special attention to lines 7 to 18, which display information on each of the parameters that will be received by the function.

The first two values that will be received by the `display_menu()` function compose the position for the menu. As you will see later, these are the coordinates where the text in the first menu selection item will be positioned. The surrounding box will be offset from this position. The box types have already been defined in your TYAC.H header file with the exception of `NOBOX`. You should add a type definition for `NOBOX`. Set it to a value other than those already used by the other box type definitions.

The fourth parameter is a pointer to a pointer called `menu`. You can cross-reference this variable to the beginning of the function in line 65. The `menu` pointer is used to point to an array of strings. These strings should contain the selection items that will be presented in the menu. Every other item in the array should contain a string with the

escape keys. Following is an example of an array that has been created to be passed in the menu parameter:

```
char *menu[10] = {
          "1. Musical Items     ", "1Mm",
          "2. Group Information", "2Gg",
          "3. Medium types      ", "3Tt",
          "4. Reporting         ", "4Rr",
          "5. Exit System       ", "5Ee"  };
```

As you can see, the first item is a menu selection item option. The second is the selection keys, or mnemonics, that will place the highlight on the given option. In this example, the menu will have five different options, each with its own set of mnemonics.

The fifth option contains a number stating how many items are in the menu array. In the previous example, there were 10 items in the array—five menu items and five sets of selection mnemonics. This number is needed by the menu program to know how large the menu array is. If the menu program didn't receive this number, it wouldn't know where the end of the menu array is.

DO	DON'T
DON'T forget the mnemonic strings.	
DO remember to pass the total of both the menu selection items and the mnemonic strings in parameter five.	

The next item is a character array similar to one seen in the `getline()` function. This is the `exit_keys` array that contains the keys that will cause the menu program to end. An example of an exit key array would be:

```
char MENU_EXIT_KEYS[MAX_KEYS] = {F3, F10, ESC_KEY};
```

For the menu that uses this array, the F3 key, F10 key, and Escape keys will all cause the menu to return to the calling program. In addition, the Enter key will also work. In fact, the Enter key is defaulted to always work.

The next parameter is a pointer to an integer. This integer, called `sel` in line 67, will be used to store the number of the final menu selection. Each selection item is given a number starting with one for the top item. When a selection is made or when an exit key is pressed, the value of `sel` will be filled with the currently highlighted item.

The last two parameters are flags. First is the arrow flag, `arr_flg`, which informs the

display_menu() program whether the left and right arrows are to be used as exit keys. When working with action bars on Day 15, you'll find that it is advantageous to treat the arrow keys as special exit keys. If a single menu is being used, then this flag should be turned off so that the arrow keys won't exit the menu. Defined constants should be added to your TYAC.H header file to be used for this parameter:

```
#define NO_LR_ARROW       0   /* Are Left/Right allowed to exit menu */
#define LR_ARROW          1
```

The last parameter, do_shadow, takes a similar value. It, however, is used to determine whether a shadow should be placed on the box containing the menu. Again, defined constants can be added to your TYAC.H header that can be used for this parameter:

```
/* Menu shadow options */
#define SHADOW            1
#define NO_SHADOW         0
```

Warning: The listings presented later assume that you have added the previous type definitions to your TYAC.H header file.

In addition to these parameters, there are several other values that are declared globally for the display_menu() function to use. Line 36 has a definition for the external color table. You will need to set up a color table in any program that is going to use the display menu function. Lines 42 to 46 declare global variables that will be used to hold copies of some of the parameters to avoid losing their values while processing the menu. In addition, these global variables will give all the functions in the listing access to necessary information.

Warning: You must set up a global color table structure called ct in any programs that call display_menu().

Several local variables are also declared within the display_menu() function in lines 69 to 77. The comments included in the code detail what these do. If no comment is included, then they will be covered as they are used in the function.

The *display_menu()* Function

The actual display_menu() function begins in line 65 of Listing 14.1. The function's

processes begin in lines 84 to 90 with some of the parameter variables being copied to local variables. Lines 96 through 104 determine the length of the longest menu selection item. This number, which will be stored in `max_len`, will be used to determine how wide to draw the menu box. In line 98, the length of a menu selection item is determined. The menu selection items are retrieved using the parameter `menu`. Don't forget that `menu` is a pointer to an array of strings. This means that dereferencing the value of `menu` produces a string. If you add 0 to `menu` and then dereference it, you get the first string in the array. If you add 1 to `menu` and dereference, you get the second string. Because `menu` contains both selection items for the menu and mnemonic strings, you will only want to check the length of every other item. The multiplication of `i` by 2 does just this.

Line 106 resets the value stored in `nbr` to the number of selection items on the menu instead of the number of items in the menu array that was passed in. The `nbr` variable will be used throughout the rest of the listing.

Lines 108 though 138 draw the menu on the screen. Line 112 checks to see if a box is to be drawn. If so, then line 114 determines if a shadow should also be drawn. If a shadow is to be drawn, then the `grid()` function is used. Line 120 then draws the box with the appropriate boarder. The menu box is drawn in the same color as the action bar. You should note that the row and column values passed to `display_menu()` are used to position the text. Because of this, the box is drawn outside of this position. In line 120, you should be able to see that the box is drawn with one row above and below the selection items. In addition, the box is drawn two columns to the left of the position provided. This gives sufficient border room within the menu. Lines 128 to 133 display each of the menu selection items that were provided in the menu array. Again, notice that every other element of the `menu` array is printed by adding an offset times two to `menu`. The final step to drawing the initial menu is to highlight the first item on the menu. This is done in line 136 by rewriting the first item in the menu highlight colors.

Note: You'll need to compensate for the box position when passing row and column values to the `display_menu()` function if you are displaying a box. If you opt not to display a box, then the text will be in the row and column position.

Now that the menu is drawn, you are almost ready to give control to the user just as you did with `getline()` on Day 10. Because a cursor is not needed for menuing, line

141 turns the cursor off. Line 142 saves the current menu position and the prior menu position. Because the menu has not been used yet, both values are set to the first menu item, 1. With this completed, a `while` loop is started that will process until the user exits the menu.

Line 147 gets a key hit from the user with the `getch()` function. If the value retrieved is equal to 0, then a scan code was entered. Lines 148 to 228 process the scan codes. If a non-scan code key was entered, then the `else` in lines 229 to 287 is called.

Processing the Menu Scan Codes

If a scan code was entered, then `getch()` must be called a second time to see what the scan code is. This is done in line 150. A `switch` statement is then used to determine the appropriate processing for the scan code (lines 151 to 227). Several scan keys have special processing. If one of the special scan keys is not pressed, then a default case, in lines 220 to 226, is called that checks to see if the scan key is a valid exit key. This is done using the `check_menu_keys()` function in lines 318 to 343. This function is virtually identical to the exit key function that was in `getline()`.

The Menu and the Home Key

The Home key positions the highlight on the first menu item. This is done by setting `menu_pos` to 1, then calling the `rewrite_menu_items()` function, and finally setting the old menu position, `old_menu_pos`, to the current menu position, which is 1.

The `rewrite_menu_items()` is used by many of the scan codes. This function is presented in lines 297 to 316. The `rewrite_menu_items()` function takes the new position, `new_pos`, and the old position, `old_pos`. The purpose of this function is to remove the highlight from the old position and highlight the new position. This is done by calling `write_string()` for each. Line 308 calls `write_string()` to remove the highlight from the old option. The old option is rewritten on the menu in the normal menu colors, `menu_fcol` and `menu_bcol`. Don't be confused by the string that is passed to `write_string()` as the first parameter. This is the same information that you have seen previously in drawing the menu:

```
*(menu_ptr+((old_pos-1) * 2))
```

This is simply an offset into the menu items array that was mentioned earlier. `menu_ptr` is a global variable set to point to the array of strings. Starting from the middle, this code is easy to follow. The old menu position is converted to a 0 offset by subtracting 1. This is then multiplied by 2 because only every other item in the menu array is a menu selection item—don't forget you need to skip the mnemonics. This determines the offset that needs to be added to `menu_ptr` and then dereferenced to get the

appropriate menu selection item string.

The second `write_string()` in line 313 works in the same manner. Instead of writing the selection item in the normal menu colors, the highlight colors are used.

The Menu and the End Key

The End key works just the opposite of the Home key. The new menu position is set to the last menu item, which is now stored in `nbr` because it was divided by 2 in line 106. This new position is then passed along with the old menu position to the `rewrite_menu_items()` function to update the menu. The old position is set to the new position, and the menu processing continues.

The Menu and the Arrow Keys

The right and left arrow keys may either exit the menu program or work in the same manner as the up and down arrow keys. This is based on the `arr_flg` argument that was passed to the `display_menu()` function. In line 168, the `arr_flg` is checked to determine if the arrows are supposed to exit the program. If the arrows are disabled in regard to exiting, then the `if` is executed. If the right arrow is used, the menu position will be incremented by one. The new position will be checked to ensure that it has not scrolled off the bottom of the menu. If the new position is off the menu, then the position will be reset to the first menu item (lines 171 and 173). For the left arrow key, the menu position is decremented and set to the last position if needed (lines 188 to 190). The process is completed by calling the `rewrite_menu_items()` function and updating the old menu position.

If the right and left arrow keys are set to exit, then the `else` statements are executed in lines 178 to 182 for the right arrow, or lines 195 to 199 for the left arrow. In these cases, the flag to exit the loop, `loop_exit`, is set to `TRUE` so that the `while` statement processing the menu will end.

The up and down arrows work in a manner identical to what was described for the right and left arrow keys. The only difference is that the up and down arrows do not have the exit options. They simply increment or decrement the current menu position, check to see that the new position is still on the menu, call `rewrite_menu_items()`, save the old menu position, and exit back to the menu.

Processing the Non-Scan Code Menu Keys

A non-scan key can be one of only a few keys. Either the key will be the Enter key, the Escape key, or an alphanumeric character. If the Enter key was pressed, then line 233

will set the looping flag, `loop_exit`, to `TRUE` so that the `while` loop will end.

If the key pressed was the Escape key, then a check is done to see if it is an exit key (lines 238 to 244). If the Escape key is an exit key, then the `loop_exit` flag is set to `TRUE`. If the Escape key is not an exit key, then the `loop_exit` key will still be `FALSE`. This will cause line 248 to execute the `boop()` function.

In the case of any other non-scan code value, the default case in line 254 is executed. Lines 256 to 270 are similar to the search done for an exit key. The difference is that instead of searching the exit key array, the mnemonic strings in the menu selection items array are searched. Each string of mnemonic characters is checked using the ANSI `strchr()` function. This function checks to see if the character the user entered, `ch`, is in the mnemonics string for each menu selection item. If it isn't, `strchr()` returns a `NULL` value. Line 260 checks to see if the character was found. If it wasn't, the next mnemonic string is checked. If it was, then the `while` loop is ended by setting a flag, `key_found`, to `TRUE`.

Once all the mnemonic strings are checked, or when the entered character is found, the program continues on. If the character wasn't found, then line 274 beeps the computer using the `boop()` function. If the character was found, then the menu position is set and the menu items are rewritten using `rewrite_menu_items()`. Control is then returned to the menu.

Final Notes on *display_menu()*

Once the user ends the menu, lines 291 and 292 are performed. Line 291 sets the value in the `display_menu()` selection parameter to the current menu position. Line 292 then returns the last key that was pressed to the calling program.

Using the *display_menu()* Function

With all of the previous description, you should be raring to use the `display_menu()` function. Before adding it to the *Record of Records!* program, you should look at it being used in a simpler program. Following in Listing 14.3 is a program showing exactly how to use the `display_menu()` function.

Listing 14.3. TESTMENU.C. A test program for `display_menu()`.

```
1:   /* Program:  testmenu.c
2:    * Author:   Bradley L. Jones
3:    *           Gregory L. Guntle
4:    * Purpose:  Demonstrate the menu function.
```

```
5:    *=======================================================*/
6:
7:   #include <stdio.h>
8:   #include "tyac.h"
9:   #include "colortbl.h"
10:
11:  char *main_menu[10] = {
12:            "1. Musical Items    ", "1Mm",
13:            "2. Group Information", "2Gg",
14:            "3. Medium types     ", "3Tt",
15:            "4. Reporting        ", "4Rr",
16:            "5. Exit System      ", "5Ee"  };
17:
18:  char MENU_EXIT_KEYS[MAX_KEYS] = {F3, F10, ESC_KEY};
19:
20:  struct color_table ct;
21:
22:  void initialize_color_table(void);
23:
24:  int main()
25:  {
26:     int rv;
27:     int menu_sel=0;
28:
29:     initialize_color_table();
30:     clear_screen( ct.bg_fcol, ct.bg_bcol );
31:
32:     rv = display_menu(10, 30, DOUBLE_BOX, main_menu, 10,
33:                       MENU_EXIT_KEYS, &menu_sel, NO_LR_ARROW,
34:                       SHADOW);
35:
36:     cursor( 20, 0 );
37:     printf("\nSelection = %d \n", menu_sel);
38:     printf("Char to exit = %x (hex)\n", rv);
39:     cursor_on();
40:     return(0);
41:  }
42:
43:  /*-------------------------------------------------------*
44:   * initialize_color_table()                              *
45:   *                                                       *
46:   * Set up global color table for rest of application     *
47:   *-------------------------------------------------------*/
48:
49:  void initialize_color_table( void )
50:  {
51:     ct.bg_fcol = YELLOW;
52:     ct.bg_bcol = BLUE;
53:
54:     ct.fld_prmpt_fcol = CYAN;
```

continues

Listing 14.3. continued

```
55:     ct.fld_prmpt_bcol = BLUE;
56:
57:     ct.fld_fcol = BRIGHTWHITE;
58:     ct.fld_bcol = BLUE;
59:
60:     ct.fld_high_fcol = YELLOW;
61:     ct.fld_high_bcol = BLUE;
62:
63:     ct.ttl_fcol = BRIGHTWHITE;
64:     ct.ttl_bcol = BLACK;
65:
66:     ct.abar_fcol = BLACK;
67:     ct.abar_bcol = WHITE;
68:
69:     ct.menu_fcol = BLACK;
70:     ct.menu_bcol = WHITE;
71:
72:     ct.menu_high_fcol = BLUE;
73:     ct.menu_high_bcol = CYAN;
74:
75:     ct.err_fcol = YELLOW;
76:     ct.err_bcol = RED;
77:
78:     ct.db_fcol = WHITE;
79:     ct.db_bcol = BROWN;
80:
81:     ct.help_fcol = YELLOW;
82:     ct.help_bcol = GREEN;
83:
84:     ct.shdw_fcol = BLACK;
85: }
```

Output

Analysis

While the `display_menu()` function was complex and long, using it isn't too difficult. As you can see, half of the TESTMENU.C listing is used to set up the color table with the colors to be used. This is done in a function called

`initialize_color_table()` in lines 43 to 85. This function is almost identical to the function presented on Day 13 in the RECOFREC.C listing.

Expert Tip: You should notice that the menu foreground color, `menu_fcol`, and the menu background color, `menu_bcol`, are set to the same colors as the action bar. This is a common practice.

In lines 8 and 9, you see the TYAC.H header file being included for your library functions, and you see the COLORTBL.H header file being included for the color table structure. Lines 11 to 16 contain the menu structure with the mnemonic keys. This structure will be passed to the display menu function. Line 18 sets up the exit keys for the menu. The last couple of lines before the program start declare a color table, `ct`, and provide a prototype for the function that will be used to initialize the color table.

The `main()` function begins in line 24. This function sets up the color table, calls the menu, prints a couple of lines, restores the cursor, and ends. This is a small amount of code when you consider all the program does. The reason for the small amount of code is because of the power of the `display_menu()` function that you have created.

You should review the call to the `display_menu()` function in line 32. As you can see, this menu will be placed with the text starting in row 10 and column 20 (the first two parameters). The menu will be in a box with a double-lined border (third parameter). The `main_menu` pointer, declared in lines 11 to 16, will be passed in the fourth parameter. Because this array has 10 elements, the number 10 is the fifth parameter. This is followed by the name of the exit key array, which was declared in line 19. The selection that is chosen will be filled into the next parameter, which is the local integer, `menu_sel`. Notice that the address of the `menu_sel` variable is passed because `display_menu()` is expecting an address. The last two parameters are `NO_LR_ARROW` and `SHADOW`. These tell `display_menu()` that the left and right arrows shouldn't exit the menu. Additionally, the menu should have a shadow. When you run the program, you should use the left and right arrow keys to see what they do.

Tip: Change the `NO_LR_ARROW` parameter to `LR_ARROW` and rerun the program to see the effect. Change `SHADOW` to `NO_SHADOW` and see what effect that has.

Lines 37 and 38 print information that can be determined after a call to `display_menu()`. Line 38 prints the returned value from the function. This is printed in hexadecimal; however, you may choose to print it in decimal by changing the `x` to `d` in the `printf()`. You will find that this value should match one of the exit keys that you set up or the value of Enter. Line 37 prints the value that `display_menu()` placed into the seventh parameter, `menu_sel`. This will be the number of the option that was highlighted when the user exited the menu.

Adding a Menu to *Record of Records!*

Now that you have seen a menu in use, you should be ready to use them in your applications. The *Record of Records!* application would be a good place to begin. Following is Listing 14.4. This is a replacement for the RECOFREC.C listing presented in Day 13. Instead of the cryptic menu provided on Day 13, this listing uses the `display_menu()` function.

Note: You should replace your RECOFREC.C listing with Listing 14.4.

Listing 14.4. LIST1404.C. A new *Record of Records!* listing.

```
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:  void initialize_color_table( void );
26:
27:  /*--------------------------*
28:   * define global variables  *
29:   *--------------------------*/
30:
31:  struct color_table ct;          /* color table */
32:
33:  /*==========================================================*
34:   *                     main()                               *
35:   *==========================================================*/
36:
37:  main()
38:  {
39:    int rv       = 0;
40:    int cont     = TRUE;
41:    int menu_sel = 0;
42:
43:    char *main_menu[10] = {
44:            “1. Enter Musical Items     “, “1Mm”,
45:            “2. Enter Medium Codes      “, “2Cc”,
46:            “3. Enter Group Information “, “3Gg”,
47:            “4. Reporting               “, “4Rr”,
48:            “5. Exit System             “, “5EeQqXx”  };
49:
50:    char MENU_EXIT_KEYS[MAX_KEYS] = {F3, F10, ESC_KEY};
51:
52:    initialize_color_table();
53:
54:    while( cont == TRUE )               /* loop in temp menu */
55:    {
56:
57:      draw_borders(“  RECORD of RECORDS!  “ );
58:      write_string( “Help”, ct.abar_fcol, ct.abar_bcol, 1, 3);
59:
60:
61:      rv = display_menu(10, 27, DOUBLE_BOX, main_menu, 10,
62:                        MENU_EXIT_KEYS, &menu_sel, NO_LR_ARROW,
63:                        SHADOW);
64:
65:      switch( rv )
66:      {
67:         case ENTER_KEY: /* accept selection */
68:         case CR:
69:                       switch( menu_sel )
70:                       {
```

continues

Listing 14.4. continued

```
71:                          case 1:  /* Menu option 1 */
72:                                   cursor_on();
73:                                   do_albums_screen();
74:                                   break;
75:
76:                          case 2:  /* Menu option 2 */
77:                                   cursor_on();
78:                                   do_medium_screen();
79:                                   break;
80:
81:                          case 3:  /* Menu option 3 */
82:                                   cursor_on();
83:                                   do_groups_screen();
84:                                   break;
85:
86:                          case 4:  /* Reporting */
87:                                   boop();
88:                                   boop();
89:                                   break;
90:
91:                          case 5:  /* exit */
92:                                   cont = FALSE;
93:                                   break;
94:
95:                          default: /* continue looping */
96:                                   boop();
97:                                   break;
98:                      }
99:                      break;
100:
101:       case F3:      /* exiting */
102:       case ESC_KEY: /* could display 'Are you sure message' */
103:                     cont = FALSE;
104:                     break;
105:
106:       case F10:     /* action bar */
107:  //                   rv = do_main_actionbar();
108:
109:  //                   if( rv == F3 )
110:  //                       cont = FALSE;
111:
112:                     break;
113:
114:       default:      boop();
115:                     break;
116:    }
117:  }
118:
119:    /* clean up screen for exit */
```

```
120:   clear_screen( BRIGHTWHITE, BLACK );
121:   cursor_on();
122:   cursor(0,0);
123:   repeat_char(' ', 80, YELLOW, BLUE );
124:   write_string( "Thank you for using RECORD OF RECORDS!",
125:                 YELLOW, BLUE, 0, 21 );
126:   return 0;
127: }
128:
129: /*-------------------------------------------------------*
130:  *   draw_borders()                                       *
131:  *-------------------------------------------------------*/
132: void draw_borders(char *title)
133: {
134:    int col=0;    /* used to center title */
135:
136:    clear_screen( ct.bg_fcol, ct.bg_bcol );
137:
138:    col = (( 80 - strlen(title)) / 2 );
139:
140:    write_string( title, ct.ttl_fcol, ct.ttl_bcol, 0, col );
141:
142:    cursor(1,0);
143:    repeat_char(' ', 80, ct.abar_fcol, ct.abar_bcol );
144:    cursor(24,0);
145:    repeat_char(' ', 80, ct.abar_fcol, ct.abar_bcol );
146: }
147:
148: /*-------------------------------------------------------*
149:  *   display_msg_box()                                   *
150:  *-------------------------------------------------------*/
151: void display_msg_box(char *msg, int fcol, int bcol )
152: {
153:    char *saved_screen = NULL;
154:    saved_screen = save_screen_area( 11, 15, 19, 60 );
155:
156:    grid( 12, 15, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
157:    box(11, 14, 20, 60, SINGLE_BOX, FILL_BOX, fcol, bcol);
158:
159:    write_string( msg, fcol, bcol, 12, 23 );
160:    write_string( "Press any key to continue...",
161:                  fcol, bcol, 13, 23 );
162:
163:    cursor_off();
164:    getch();
165:    cursor_on();
166:
167:    restore_screen_area(saved_screen);
168: }
169:
```

continues

Listing 14.4. continued

```
170: /*-----------------------------------------------------*
171:  *   yes_no_box()                                       *
172:  *-----------------------------------------------------*/
173: char yes_no_box(char *msg, int fcol, int bcol )
174: {
175:    char ch;
176:    char *saved_screen = NULL;
177:    saved_screen = save_screen_area( 11, 15, 19, 60 );
178:
179:    grid( 12, 15, 19, 59, ct.shdw_fcol, ct.bg_bcol, 3 );
180:    box(11, 14, 20, 60, SINGLE_BOX, FILL_BOX, fcol, bcol);
181:
182:    write_string( msg, fcol, bcol, 12, 23 );
183:    write_string( "Enter (Y) or (N)", fcol, bcol, 13, 23 );
184:
185:    cursor_off();
186:    ch = getch();
187:    ch = toupper(ch);
188:
189:    while( ch != 'Y' && ch != 'N' )
190:    {
191:       ch = toupper( getch() );
192:    }
193:
194:    cursor_on();
195:    restore_screen_area(saved_screen);
196:    return(ch);
197: }
198:
199: /*-----------------------------------------------------*
200:  *  Function: zero_fill_field();                       *
201:  *  Purpose:  Right justifies a character array and then*
202:  *            pads the left side with zeros. (Assumes   *
203:  *            that the field is NOT null terminated.)   *
204:  *  Returns:  # of zeros used to pad field              *
205:  *            -1 if field too large (longer than 20 )   *
206:  *            0  if field is blank (not padded)         *
207:  *-----------------------------------------------------*/
208:
209: int zero_fill_field( char *field, int size )
210: {
211:    int  ctr,
212:         pads = 0;
213:
214:    char tmp[20];
215:
216:    if( size > 20 )
217:    {
218:       pads = -1;   /* field too long */
```

```
219:    }
220:    else
221:    if( strlen(field) == 0 )
222:    {
223:       pads = 0;    /* leave blank fields blank. */
224:    }
225:    else
226:    {
227:       pads = size - (strlen(field));    /* How many 0s? */
228:
229:       for( ctr = 0; ctr < pads; ctr++ ) /* pad tmp field */
230:          tmp[ctr] = '0';
231:
232:           /* copy original info to end of tmp field */
233:       strncpy( tmp+pads, field, strlen(field));
234:           /* replace original field with padded tmp */
235:       strncpy(field, tmp, size);
236:    }
237:
238:    return(pads);
239: }
240:
241: /*-------------------------------------------------------*
242:  * initialize_color_table()                              *
243:  *                                                       *
244:  * Set up global color table for rest of application     *
245:  *-------------------------------------------------------*/
246:
247: void initialize_color_table( void )
248: {
249:    ct.bg_fcol = YELLOW;
250:    ct.bg_bcol = BLUE;
251:
252:    ct.fld_prmpt_fcol = CYAN;
253:    ct.fld_prmpt_bcol = BLUE;
254:
255:    ct.fld_fcol = BRIGHTWHITE;
256:    ct.fld_bcol = BLUE;
257:
258:    ct.fld_high_fcol = YELLOW;
259:    ct.fld_high_bcol = BLUE;
260:
261:    ct.ttl_fcol = BRIGHTWHITE;
262:    ct.ttl_bcol = BLACK;
263:
264:    ct.abar_fcol = BLACK;
265:    ct.abar_bcol = WHITE;
266:
267:    ct.menu_fcol = BLACK;
268:    ct.menu_bcol = WHITE;
```

continues

Listing 14.4. continued

```
269:
270:    ct.menu_high_fcol = BLUE;
271:    ct.menu_high_bcol = CYAN;
272:
273:    ct.err_fcol = YELLOW;
274:    ct.err_bcol = RED;
275:
276:    ct.db_fcol = WHITE;
277:    ct.db_bcol = BROWN;
278:
279:    ct.help_fcol = YELLOW;
280:    ct.help_bcol = GREEN;
281:
282:    ct.shdw_fcol = BLACK;
283: }
284:
285:
286: /*=========================================================*
287:  *                    end of listing                       *
288:  *=========================================================*/
```

Output

Analysis

Much of this listing is the same as the original RECOFREC.C listing. The changes start in the `main()` function. Line 41 sets up a `menu_sel` variable to accept the return value from the menu that will be displayed. Lines 43 to 48 set up the menu items that will be displayed along with their mnemonic values. Line 50 sets up the exit keys. For this main menu, the F3, F10, and Escape keys will all cause the menu to exit. By default, the Enter key will also cause the menu to exit.

Line 52 calls the `initialize_color_table()` function that is presented in lines 241 to 285. As you can see, lines 262 to 266 include the color variables for the menu.

Lines 54 to 114 contain a `while` loop that will keep the menu displayed until the user signals to exit. The screen is drawn in line 57 with a call to `draw_borders()`. The `draw_borders()` function has not changed.

Line 61 begins the actual menu process with a call to `display_menu()`. The menu is set up to be in a box with a shadow. In addition, the box will have a double-lined border. The left and right arrow keys have been turned off because you don't want this menu to end if they are pressed. The menu will be displayed until a user presses one of the exit keys or makes a selection.

Lines 65 to 117 react to the result of the call to `display_menu()`. A `switch` statement routes control based on what key was returned from the menu. If F3 or the Escape key was pressed, then the `cont` flag will be set to `FALSE` so that the loop will end. If the F10 key is pressed, then the action bar function will be called. Because this function will be presented on Day 15, it should be commented out for now.

Warning: The action bar function, `do_main_actionbar()`, is included in the listing because the F10 key is an exit key for the menu. This function will be filled in on Day 15.

If Enter was pressed to exit the menu, then the user made a selection. A `switch` statement, in lines 69 to 98, routes the appropriate action based on the value of `menu_sel`. Remember that `menu_sel` contains the number of the selection made. For selections 1 through 3, the processing is almost identical to what it was before adding the new menu. Each of the corresponding screen functions is called. The only change is that the `cursor_on()` function is called first. Because the menu turned the cursor off, you need to turn it back on.

The other cases are different. Case 4 calls the `boop()` function twice. This is done because reporting won't be covered until Day 20. If the menu selection was 5, then the user selected to exit. The `cont` flag is set to `FALSE` so that the program will exit. A `default` case is also included even though there should never be a value in `menu_sel` other than 1 through 5. It's good programming practice to include a `default` case in every `switch`.

The last few lines of `main()` are almost identical to what was presented before. The big change is the addition of the `cursor_on()` function in line 121. Without this, the cursor would remain off even after you exit the program. The rest of the listing is the same as it was before.

Using a Menu with Entry and Edit

On Day 13, you allowed the user to enter data into entry and edit screens. To retrieve the data from the user, you used the `getline()` function. There are times when only a certain number of choices are valid and these choices will never change. For example, on the Groups screen, you may decide that there are only a certain number of Types of Music that the user will be able to enter into the system. In this situation, you can choose to let the user select a value from a menu instead of typing the value.

You already have all the information you need to accept the type of music from a menu instead of having the user enter the information. In the GROUPS.C listing (originally presented on Day 13 in Listing 13.5), you included a case for each field on the screen. Each case did a call to `getline()` along with any edits that may have been required. The case for the Type of Music field was as follows:

```
293:         case 4 :
294:            rv = getline( GET_ALPHA, 0,  8, 19, 0, 20,
                              groups.music_type);
295:            break;
```

Instead of calling `getline()`, this case will now set up to use a menu. To make it easier to add the menu to the GROUPS.C listing, it will be created in a separate function that the case will call. Replace the `getline()` function with the following:

```
        case 4 :
          rv = do_type_of_music_menu( groups.music_type);
          write_string( groups.music_type,
                        ct.fld_fcol, ct.fld_bcol, 8, 19);
          break;
```

You will need to add a new prototype at the top of the GROUPS.C file for this function. The prototype will be as follows:

```
int do_type_of_music_menu( char * );
```

Once you have made these changes, you are ready to create the menu in the `do_type_of_music_menu()` function. The code for this function is presented in Listing 14.5. You will want to add this to the end of your GROUPS.C listing.

Type

Listing 14.5. The Type of Music entry and edit menu.

```
1:   /*=======================================================
2:    * Filename: LIST1405.c
3:    *
4:    * Author:   Bradley L. Jones
```

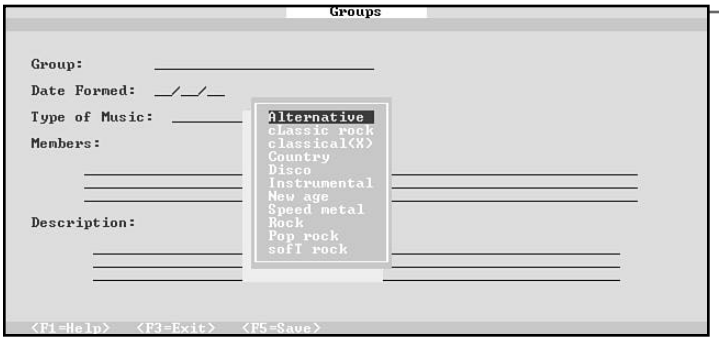
```
5:    *           Gregory L. Guntle
6:    *
7:    * Purpose:  Information to be added to GROUPS.C. This allows
8:    *           for a menu to be created for the TYPE OF MUSIC
9:    *           field.
10:   *=========================================================*/
11:
12:
13:  /*---------------------------------------------------------*
14:   *  do_type_of_music_menu();                               *
15:   *  Returns:  key used to exit menu                        *
16:   *---------------------------------------------------------*/
17:
18:  int do_type_of_music_menu( char *field )
19:  {
20:     char *saved_screen = NULL;
21:     int  menu_sel = 0;
22:     int  rv;
23:
24:     char *menu_data[22] = {
25:          "Alternative ", "1Aa",
26:          "cLassic rock", "2Ll",
27:          "classical(X)", "3Xx",
28:          "Country     ", "4Cc",
29:          "Disco       ", "5Dd",
30:          "Instrumental", "6li",
31:          "New age     ", "7Nn",
32:          "Speed metal ", "8Ss",
33:          "Rock        ", "9Rr",
34:          "Pop rock    ", "Pp",
35:          "sofT rock   ", "Tt"  };
36:
37:     char exit_keys[MAX_KEYS] = { F3, F10, ESC_KEY, SHIFT_TAB };
38:
39:
40:     saved_screen = save_screen_area( 6, 20, 27, 60 );
41:
42:     rv = display_menu( 8, 30, SINGLE_BOX, menu_data, 22,
43:                        exit_keys, &menu_sel, LR_ARROW,
44:                        SHADOW);
45:     cursor_on();
46:
47:     switch( rv )
48:     {
49:        case F3:
50:        case F10:
51:        case ESC_KEY:
52:        case SHIFT_TAB: break;
53:
54:        case LT_ARROW:  rv = SHIFT_TAB;
```

continues

Listing 14.5. continued

```
55:                         break;
56:
57:        case RT_ARROW:   rv = TAB;
58:                         break;
59:
60:        default:  /* item selected */
61:                  strcpy( field, *(menu_data+((menu_sel-1)*2)));
62:                  break;
63:     }
64:
65:     restore_screen_area( saved_screen );
66:
67:     return( rv );
68:  }
69:
70:  /*=========================================================*
71:   *                    end of listing                       *
72:   *=========================================================*/
```

Output



Analysis

Once you add this to the GROUP.C listing, you will be prompted with the menu presented in the output whenever you enter the Type of Music field. In looking at the listing, you should see that the menu is done in the same manner as the menus presented earlier today. Line 21 declares the integer variable to hold the menu selection. Lines 24 to 35 contain the `menu_data` array. This is the values that will be displayed in the menu along with their mnemonics. Line 37 declares the exit keys that the menu will use.

Tip: You should notice that in lines 25 to 35 one letter in each of the menu items is capitalized. This is the mnemonic letter that will move the highlight. You should also notice that even though numbers are

not presented on the menu, numeric mnemonics are still used. This is to give the menu added flexibility.

Before displaying the menu, the `saved_screen` pointer declared in line 20 is used to save off the area of the screen where the menu will be positioned. Line 40 then saves the screen using the `save_screen_area()` function from Day 11. The screen is restored before the function returns control to the entry and edit screen.

The menu is displaced in line 43. As you can see, a single-bordered box is used along with a shadow. In addition, the left and right arrows are enabled. Once the menu has been displayed and returns controls, line 45 turns the cursor back on. (Remember the menu turns it off.)

Lines 47 to 63 react to the returned value from the call to `display_menu()` in line 42. If the F3 key, F10 key, Escape key, or Shift+Tab is pressed, then the key value is returned to the calling screen to process. If the left or right arrow key is pressed, then they are translated to `SHIFT_TAB` and `TAB` values and returned to the screen to process. This causes the left and right arrows to place the cursor on the next or previous field.

If the return value is anything else, then the data should be accepted. The corresponding menu item is copied into the screen field (line 61). Don't let the math in this line confuse you. You are copying an element from the `menu_data` array. The value of `menu_sel` is from 1 to 11, depending on which menu item was selected. The offset into the `menu_data` array will be 2 times the result of the `menu_item` selected minus 1 (so the offsets begin with 0). The reason you multiply by 2 is because only every other item in the `menu_data` array is a menu item. You want to skip the mnemonic key strings.

DO	DON'T
DON'T forget to turn the cursor back on after using the menu.	
DO remember to set up the exit keys before calling `display_menu()`.	
DO compare `display_menu()` to the `getline()` function so that you can understand the similarities.	

Summary

Today, you covered the topic of menus. You were presented with a function called `display_menu()`, which enables you to display menus in your applications. Once you create the `display_menu()` function, you have the ability to add menus to your applications with very little work. You are able to display menus at virtually any location on the screen. In addition, you are able to state which keys you want to work within the menu. These keys include mnemonics that will take the highlight within the menu to a specific menu item. Your menus can be displayed in a box, with shadows, with single- or double-sided borders, and more. Today's information will be followed with action bars tomorrow.

Q&A

Q Can menus call other menus?

A Yes, you can have menus call other menus. In fact, action bars on Day 15 will do just that.

Q Is there a maximum number of menu items that can be listed in a menu?

A Yes. Common sense should tell you not to list more items in your menu than what will fit on the screen.

Q What changes should be made to the TYAC.H header file after creating today's programs?

A The following information should be added to the TYAC.H header file to accommodate the `display_menu()` function:

```
/* Menu shadow options */
#define SHADOW           1
#define NO_SHADOW        0

/* ----------------------------------- *
 *          Menu Items                 *
 * ----------------------------------- */
#define NOBOX             0
#define MAX_MENU_NBR     10
#define MAX_MENU_LEN     30
#define MAX_KEYS         17
#define NO_LR_ARROW       0   /* Are Left/Right allowed to exit
```

```
menu */
#define LR_ARROW          1

int  display_menu(int,   int, int, char **, int, char*,
                  int *, int, int );
```

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 mnemonic characters used for?
2. Why is it best to keep the color table structure in its own listing?
3. What does the Home key do within a menu?
4. What does the Page Down key do within a menu?
5. What is the most selection items that you can have on a menu?

Exercises

1. **ON YOUR OWN:** Add the `display_menu()` function to your TYAC.LIB library.
2. **ON YOUR OWN:** Be sure to create the COLORTBL.H listing presented today. Modify your RECORDS.H header to include this listing instead of containing the color table structure.
3. Write a program that displays a menu with the following options:
    1. RED
    2. YELLOW
    3. BLUE
    4. GREEN

    Include mnemonics.

4. **ON YOUR OWN:** Experiment with the `NO_LR_ARROW`, `LR_ARROW`, `SHADOW`, and `NO_SHADOW` menu parameters.
5. **ON YOUR OWN:** Create menus in your own applications.
6. What would be appropriate mnemonic keys for the menu selection item, "5. Exit"?