File Routines: Using Dynamic Data

WEEK
3

On Day 17, you added file routines to the medium code and the group information screens in your *Record of Record!* applications. While you added these functions to the screen, you still need to complete the action bar routines. Additionally, while these routines worked for those screens, they do not work for the Musical Items screen. The number of Titles with each musical item is a variable. Today, you'll use what you learned on Day 17 as a starting point; however, you will do much more. Today you will:

- ☐ Add the file routines to the action bar.
- ☐ Create a function to find a specific record.
- ☐ Understand why the routines from Day 17 won't work for the Musical Items screen.
- ☐ Look at what the differences are for the file routines needed for the Musical Items screen.
- ☐ Create file routines for the Musical Items screen.

File Functions and the Action Bar

On Day 17, you added all the functions to the Medium Code screen; however, you didn't add them to the action bar. Each of the file functions needs to be added to the action bar. Listing 18.1 contains the new MEDMABAR.C code.

Listing 18.1. MEDMABAR.C. The medium action bar function.

```
1:  /*=========================================================
2:   * Filename: medmabar.c
3:   *           RECORD OF RECORDS - Version 1.0
4:   *
5:   * Author:   Bradley L. Jones
6:   *
7:   * Purpose:  Action bar for medium screen.  This will
8:   *           contain multiple menus. The functions called
9:   *           by the menu selections may not be available
10:  *           until later days.
11:  *
12:  * Return:   Return value is either key used to exit a menu
13:  *           with the exception of F10 which is returned
14:  *           as Enter to redisplay main menu.  In each menu
15:  *           the left and right keys will exit and move
16:  *           control to the right or left menu.
17:  *=========================================================*/
```

```
18:
19:  #include <stdio.h>
20:  #include <stdlib.h>    /* for exit() */
21:  #include <string.h>
22:
23:  #include “tyac.h”
24:  #include “records.h”
25:
26:  /* Global variables/functions for MEDIUM */
27:  extern MEDIUM_REC medium;   /* Record structure with data */
28:  extern MEDIUM_REC prev;
29:  extern FILE *idx_fp;        /* Main File ptr to data file */
30:  extern FILE *db_fp;         /* Data file */
31:  extern int nbr_records;     /* Ttl # of index rec for mediums */
32:  extern int start_rec;
33:  extern int disp_rec;
34:
35:  /*--------------------*
36:   *     prototypes     *
37:   *--------------------*/
38:
39:  #include “recofrec.h”
40:
41:  int do_medium_menu1( void );
42:  int do_medium_menu2( void );
43:  int do_medium_menu3( void );
44:  int do_medium_menu4( void );
45:
46:  void draw_medium_screen( void );
47:  void draw_medium_prompts( void );
48:  void display_medium_fields( void );
49:  int  search_med_rec(char *);
50:
51:
52:
53:  /*----------------------------*
54:   *  medium screen action bar  *
55:   *----------------------------*/
56:
57:  int do_medium_actionbar( void )
58:  {
59:    int  menu  = 1;
60:    int  cont  = TRUE;
61:    int  rv    = 0;
62:    char *abar_text ={“ File   Edit   Search   Help “};
63:
64:
65:    while( cont == TRUE )
66:    {
67:       write_string(abar_text, ct.abar_fcol, ct.abar_bcol, 1, 2);
68:
```

continues

Listing 18.1. continued

```
69:       switch( menu )
70:       {
71:
72:          case 1:  /* file menu */
73:                   write_string( " File ", ct.menu_high_fcol,
74:                        ct.menu_high_bcol, 1, 2);
75:
76:                   rv = do_medium_menu1();
77:                   break;
78:
79:          case 2:  /* edit menu */
80:                   write_string( " Edit ", ct.menu_high_fcol,
81:                        ct.menu_high_bcol, 1, 9);
82:
83:                   rv = do_medium_menu2();
84:                   break;
85:
86:          case 3:  /* search menu */
87:                   write_string( " Search ", ct.menu_high_fcol,
88:                        ct.menu_high_bcol, 1, 16);
89:
90:                   rv = do_medium_menu3();
91:                   break;
92:
93:          case 4:  /* Help menu */
94:                   write_string( " Help ", ct.menu_high_fcol,
95:                        ct.menu_high_bcol, 1, 25);
96:
97:                   rv = do_medium_menu4();
98:                   break;
99:
100:         default: /* error */
101:                  cont = FALSE;
102:                  break;
103:      }
104:
105:      switch( rv )
106:      {
107:         case LT_ARROW:  menu--;
108:                         if( menu < 1 )
109:                             menu = 4;
110:                         break;
111:
112:         case RT_ARROW:  menu++;
113:                         if( menu > 4 )
114:                             menu = 1;
115:                         break;
116:
117:         default:        cont = FALSE;
118:                         break;
```

```
119:       }
120:    }
121:    write_string(abar_text, ct.abar_fcol, ct.abar_bcol, 1, 2);
122:    cursor_on();
123:    return(rv);
124: }
125:
126:
127:
128: /*--------------------*
129:  *  do_menu 1  (File) *
130:  *--------------------*/
131:
132: int do_medium_menu1( void )
133: {
134:    int   rv       = 0;
135:    int   menu_sel = 0;
136:    char *saved_screen = NULL;
137:
138:    char *file_menu[2] = { " Exit  <F3> ", "1Ee" };
139:    char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
140:
141:    saved_screen = save_screen_area( 0, 10, 0, 40 );
142:
143:    rv = display_menu( 3, 4, SINGLE_BOX, file_menu, 2,
144:                       exit_keys, &menu_sel, LR_ARROW, SHADOW);
145:
146:    switch( rv )
147:    {
148:       case ENTER_KEY: /* accept selection */
149:       case CR:
150:                        rv = F3;
151:                        break;
152:
153:       case F3:        /* exiting */
154:       case ESC_KEY:
155:       case LT_ARROW:  /* arrow keys */
156:       case RT_ARROW:
157:                        break;
158:
159:       case F10:       /* exit action bar */
160:                        rv = ENTER_KEY;
161:                        break;
162:
163:       default:         boop();
164:                        break;
165:    }
166:    restore_screen_area( saved_screen );
167:
168:    return(rv);
```

continues

Listing 18.1. continued

```
169: }
170:
171: /*--------------------*
172:  *  do_menu 2  (Edit) *
173:  *--------------------*/
174:
175: int do_medium_menu2( void )
176: {
177:   int   rv       = 0;
178:   int   menu_sel = 0;
179:   char *saved_screen = NULL;
180:
181:   char *edit_menu[8] = {
182:           " New          ", "1Nn",
183:           " Add     <F4> ", "2Aa",
184:           " Change  <F5> ", "3Cc",
185:           " Delete  <F6> ", "4Dd" };
186:
187:   char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
188:
189:   saved_screen = save_screen_area( 1, 10, 8, 40 );
190:
191:   rv = display_menu( 3, 11, SINGLE_BOX, edit_menu, 8,
192:                      exit_keys, &menu_sel, LR_ARROW, SHADOW);
193:
194:   switch( rv )
195:   {
196:      case ENTER_KEY: /* accept selection */
197:      case CR:
198:               switch( menu_sel )
199:               {
200:                 case 1:  /* Clear the screen */
201:                          restore_screen_area( saved_screen );
202:                          disp_rec = 0;
203:                          clear_medium_fields();
204:                          draw_medium_screen();
205:                          break;
206:
207:                 case 2:  /* Add a record */
208:                          restore_screen_area( saved_screen );
209:                          rv = add_mdata();
210:                          if ( rv == NO_ERROR )
211:                          {
212:                            /* Reset display counter */
213:                            display_msg_box("Added record!",
214:                                      ct.db_fcol, ct.db_bcol);
215:                            disp_rec = 0;
216:                            clear_medium_fields();
217:                            draw_medium_screen();
```

```
218:                        }
219:                        else  /* Only do if File I/O */
220:                        if ( rv > NO_ERROR)
221:                        {
222:                          display_msg_box("Fatal Error writing
                                              data...",
223:                               ct.err_fcol, ct.err_bcol );
224:                          exit(1);
225:                        }
226:                        break;
227:
228:                case 3:  /* Update the current record */
229:                        restore_screen_area( saved_screen );
230:                        rv = add_mdata();   /* updates record */
231:
232:                        if ( rv == NO_ERROR )
233:                        {
234:                          /* Reset display counter */
235:
236:                          display_msg_box("Record Updated!",
237:                                    ct.db_fcol, ct.db_bcol);
238:                        }
239:                        else
240:                        if ( rv > NO_ERROR )
241:                        {
242:                          display_msg_box("Fatal Error writing
                                              data...",
243:                               ct.err_fcol, ct.err_bcol );
244:
245:                          exit(1);
246:                        }
247:                        break;
248:
249:                case 4:  /* Deleting the current record */
250:                        restore_screen_area( saved_screen );
251:                        if( (yes_no_box( "Delete record ?",
252:                             ct.db_fcol, ct.db_bcol )) == 'Y' )
253:                        {
254:                          rv = del_med_rec();
255:                          if (rv == NO_ERROR)
256:                          {
257:                            disp_rec = 0;
258:                            clear_medium_fields();
259:                            draw_medium_screen();
260:                          }
261:                          else
262:                          {
263:                            display_msg_box("Fatal Error deleting
                                                data...",
264:                               ct.err_fcol, ct.err_bcol );
```

continues

Listing 18.1. continued

```
265:                                exit(1);
266:                              }
267:                            }
268:                            break;
269:
270:                    default: /* continue looping */
271:                            boop();
272:                            break;
273:                }
274:
275:                rv = ENTER_KEY;
276:                break;
277:
278:        case F3:        /* exiting */
279:        case ESC_KEY:
280:        case LT_ARROW: /* arrow keys */
281:        case RT_ARROW:
282:                        restore_screen_area( saved_screen );
283:                        break;
284:
285:        case F10:       /* action bar */
286:                        restore_screen_area( saved_screen );
287:                        rv = ENTER_KEY;
288:                        break;
289:
290:        default:        boop();
291:                        break;
292:    }
293:
294:    return(rv);
295: }
296:
297: /*----------------------*
298:  *  do menu 3  (Search) *
299:  *----------------------*/
300:
301: int do_medium_menu3( void )
302: {
303:    int   rv       = 0;
304:    int   menu_sel = 0;
305:    char *saved_screen = NULL;
306:
307:    char *search_menu[6] = {
308:             " Find...         ", "1Ff",
309:             " Next      <F7> ", "2Nn",
310:             " Previous  <F8> ", "3Pp" };
311:
312:    char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
313:
314:    saved_screen = save_screen_area( 1, 10, 0, 60 );
```

```
315:
316:    rv = display_menu( 3, 18, SINGLE_BOX, search_menu, 6,
317:                       exit_keys, &menu_sel, LR_ARROW, SHADOW);
318:
319:    switch( rv )
320:    {
321:       case ENTER_KEY: /* accept selection */
322:       case CR:
323:                 switch( menu_sel )
324:                 {
325:                  case 1:  /* Do find dialog */
326:                           restore_screen_area( saved_screen );
327:
328:                           rv = search_med_rec(medium.code);
329:
330:                           if( rv == NO_ERROR )
331:                           {
332:                              /* record found */
333:                              draw_medium_screen();
334:                              display_msg_box("Record Found!",
335:                                       ct.db_fcol, ct.db_bcol);
336:                           }
337:                           else
338:                           if( rv < 0 )
339:                           {
340:                              display_msg_box("Record Not Found!",
341:                                       ct.err_fcol, ct.err_bcol);
342:                           }
343:                           else
344:                           {
345:                             display_msg_box("Fatal Error processing
                                                 data...",
346:                                    ct.err_fcol, ct.err_bcol );
347:                             exit(1);
348:                           }
349:                           break;
350:
351:                  case 2:  /* Next Record */
352:                           restore_screen_area( saved_screen );
353:                           rv = proc_med_rec(NEXT_REC);
354:                           if ( rv == NO_ERROR )
355:                           {
356:                              draw_medium_screen();
357:                           }
358:                           else
359:                           {
360:                             display_msg_box("Fatal Error processing
                                                 data...",
361:                                    ct.err_fcol, ct.err_bcol );
362:                             exit(1);
```

continues

Listing 18.1. continued

```
363:                          }
364:                          break;
365:
366:                case 3:   /* Prev record */
367:                          restore_screen_area( saved_screen );
368:                          rv = proc_med_rec(PREVIOUS_REC);
369:                          if ( rv == NO_ERROR )
370:                          {
371:                            draw_medium_screen();
372:                          }
373:                          else
374:                          {
375:                            display_msg_box("Fatal Error processing
                                                 data...",
376:                                ct.err_fcol, ct.err_bcol );
377:                            exit(1);
378:                          }
379:                          break;
380:
381:                default:  /* shouldn't happen */
382:                          boop();
383:                          break;
384:              }
385:
386:                        rv = ENTER_KEY;
387:                        break;
388:
389:       case F3:         /* exiting */
390:       case ESC_KEY:
391:       case LT_ARROW:   /* arrow keys */
392:       case RT_ARROW:
393:                        restore_screen_area( saved_screen );
394:                        break;
395:
396:       case F10:        /* action bar */
397:                        rv = ENTER_KEY;
398:                        restore_screen_area( saved_screen );
399:                        break;
400:
401:       default:         boop();
402:                        break;
403:   }
404:
405:   return(rv);
406: }
407:
408: /*--------------------*
409:  *  do menu 4  (Help) *
410:  *--------------------*/
```

```
411:
412: int do_medium_menu4( void )
413: {
414:   int   rv       = 0;
415:   int   menu_sel = 0;
416:   char *saved_screen = NULL;
417:
418:   char *help_menu[4] = {
419:           " Help  <F2> ", "1Hh",
420:           " About      ", "2Ee" };
421:
422:   char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
423:
424:   saved_screen = save_screen_area( 1, 10, 0, 60 );
425:
426:   rv = display_menu( 3, 27, SINGLE_BOX, help_menu, 4,
427:                      exit_keys, &menu_sel, LR_ARROW, SHADOW);
428:
429:   switch( rv )
430:   {
431:      case ENTER_KEY: /* accept selection */
432:      case CR:
433:                      switch( menu_sel )
434:                      {
435:                        case 1:  /* Extended Help */
436:                                 display_medium_help();
437:
438:                                 break;
439:
440:                        case 2:  /* About box */
441:                                 display_about_box();
442:
443:                                 break;
444:
445:                        default: /* continue looping */
446:                                 boop();
447:                                 break;
448:                      }
449:
450:                      break;
451:
452:       case F3:       /* exiting */
453:       case ESC_KEY:
454:       case LT_ARROW: /* arrow keys */
455:       case RT_ARROW:
456:                      break;
457:
458:       case F10:      /* action bar */
459:                      rv = ENTER_KEY;
460:                      break;
```

continues

Listing 18.1. continued

```
461:
462:       default:       boop();
463:                      break;
464:   }
465:   restore_screen_area( saved_screen );
466:
467:   return(rv);
468: }
469:
470: /*=========================================================
471:  * Function: search_med_rec()
472:  *
473:  * Author:   Bradley L. Jones
474:  *           Gregory L. Guntle
475:  *
476:  * Purpose: Searches for the key to locate
477:  *
478:  * Returns:  0 - No errors - key found
479:  *          <0 - Key not found
480:  *          >0 - File I/O Error
481:  *=========================================================*/
482:
483: int search_med_rec(char *key)
484: {
485:   int rv = NO_ERROR;
486:   int done = FALSE;
487:   int srch_rec;
488:   int result;
489:
490:   MEDIUM_INDEX temp;
491:
492:   /* Start at top of chain */
493:   /* Stop when either
494:      1 - The key passed matches   or
495:      2 - The key passed is < rec read
496:      The latter indicates the key is not in the file  */
497:
498:   srch_rec = start_rec;
499:   while (!done)
500:   {
501:     /* Get record */
502:     rv = get_rec(srch_rec, idx_fp, sizeof(temp),
503:           sizeof(int)*2, (char *)&temp);
504:     if (rv == 0)
505:     {
506:       result = stricmp(key, temp.code);
507:       if (result == 0)   /* Found match */
508:       {
509:         /* Now get data */
```

```
510:          done = TRUE;
511:          rv = get_rec(temp.data, db_fp, sizeof(medium),
512:                0, (char *)&medium);
513:        }
514:        else
515:        if (result < 0)
516:        {
517:          /* Key < next rec in DB - key doesn't exist */
518:          done = TRUE;
519:          rv = -1;              /* Key not found */
520:        }
521:        else
522:        {
523:          srch_rec = temp.next;
524:          if (srch_rec == 0)    /* At end of list ? */
525:          {
526:            done = TRUE;
527:            rv = -1;            /* Key not found */
528:          }
529:        }
530:      }
531:    }
532:
533:    return(rv);
534:
535: }
```


Analysis A lot of this code hasn't changed from Day 15 when you initially created the MEDMABAR.C source. The new code begins with the addition of the STRING.H header file in line 21. Line 38 contains a prototype for a new function, `search_med_rec()`. Additional new code begins in line 26 with the declaration of several external variables. These external variables were all covered on Day 17. The rest of the changes are in the `do_medium_menu2()` and `do_medium_menu3()` functions.

The `do_medium_menu2()` function begins on line 175. This action bar menu contained the EDIT options for creating a new record, adding a new record, changing an existing record, and deleting a record. Creating a new record means to clear the screen so that a new record can be entered. This is a function that wasn't presented on Day 17; however, the logic behind this was. Adding a new record, changing an existing record, and deleting a record were all covered on Day 17. Today, you will see them added to the action bar.

Clearing for a new record is done within case 1 in lines 200 to 205. This case starts by restoring the saved screen area. The current record, `disp_rec`, is then set to zero

because you won't be editing a preexisting record. Line 203 then calls the `clear_medium_fields()` function to clear any data that may be in the medium code structure. Once the screen is redrawn in line 204, you are ready for a new record.

The second EDIT action bar menu option is to add a record. This is accomplished in case 2 (lines 207 to 226). Again, this starts with the restoring of the screen. Line 209 calls the `add_mdata()` function that was presented on Day 17. This function adds a record to the files. If this addition of the records was successful, then line 213 displays a message so that users will know they were successful. Lines 215 to 217 then repeat the process of clearing the screen for a new record. If the writing of the record failed, then a fatal error is displayed in line 221 and the program exits.

The third case handles updating or changing a record. This case is presented in lines 228 to 250. This function follows a flow similar to adding a record. As shown on Day 17, the `add_mdata()` function handles both the updating and the adding of records. If the `add_mdata()` function successfully updates the current record, then an appropriate message is displayed in line 236. If the addition wasn't successful, then a fatal error message is displayed in line 242 and the program exits.

Deleting a record can be accomplished with the fourth case in lines 249 to 268. The process of deleting should always start with asking the users if they are sure they want to remove a record. This is done in line 251. If the user says yes, then the `del_med_rec()` function is called (Line 254). If the delete is successful, lines 256 to 258 refresh the screen so a new record can be added. If there is a problem in deleting, then an error message is displayed and the program exits. With this, the functions in the EDIT action bar menu are complete.

The SEARCH menu also contains three functions that work with file I/O. These are handled in the `do_medium_menu3()`. These are the `Find...`, the next search, and the previous search.

The `Find...` case is new. This is presented in lines 325 to 349. This function starts with the restoration of the screen in the same manner as all the other functions. Line 328 then does the actual finding with the `search_med_rec()` function. This function attempts to find the corresponding record for the value in the code field on the screen. If a matching record is found, then the medium screen is redrawn (line 333) and a message is displayed. Line 338 checks to see if the record wasn't found. If `rv` is less than zero, then the provided code wasn't found. If `rv` is greater than zero, then a fatal error is displayed in line 345.

The `search_med_rec()` function is presented in lines 470 to 535. The search starts at the first alphabetic record in the index and works its way until it finds a record that either matches or is greater than its own value. Line 498 starts the process by

initializing the counter, `srch_rec`, to the first index record, `start_rec`. A `while` loop then begins the processing of each index record in sorted (next) order. Line 502 retrieves the index for the current search number. If there isn't a problem, then line 506 compares the current index's code to the code that is being searched for. If the values are equal, then line 510 sets the `done` flag to `TRUE` and the data record is retrieved. Otherwise, if the result is less than zero (line 515), then there isn't a need to continue because the current search code is greater than the code being looked for. Negative one is returned so that the calling program will know that the code was not found—not that there was an error.

If the search code was less than the code being looked for, then lines 523 to 528 are executed. The `srch_rec` is set to the next index record number, `temp.next`. If this number is zero, then the end of the index file has been reached without finding the record. This process continues until the `done` flag is set to `TRUE`. As you just saw, this happens when either the record is found, the end of the file is reached, or a code greater than the current code is found.

In the `do_medium_menu3()` function, you can see the next and previous records are found in lines 351 to 379. Finding the next or finding the previous record are completed in the same way with one difference. The `proc_med_rec()` function is used to get the record. If successful, the screen is redrawn; otherwise, a fatal error message is displayed. The difference in the two cases is that the next record case passes `NEXT_REC` to `proc_med_rec()` in line 353 and the previous record case passes `PREVIOUS_REC` to `proc_med_rec()` in line 368.

With this, you have completed the action bar routines. The `do_something()` case that had been added as a filler can be removed because all of the action bar items are complete. In fact, at this point, the entire Medium Codes screen is complete!

Note: At this point the Medium Codes entry and edit screen is complete!

DO	DON'T

DO always ask the users to ensure they want to delete a record.

DO display messages to let the user know when he or she has done something such as adding, deleting, or changing a record.

The Musical Items Screen

At this point, you have completed the Medium Code screen. The Group Information screen has been left for you to do as exercises. This leaves the Musical Items screen.

The Musical Items screen is a little more complex than the Medium Code and the Group Information screens. The complexity stems from the addition of the structure for holding songs. In the following sections, you'll see the code for working with the Musical Items screen. The code presented will show you how to process the medium screen information using the index structure, the data structure for the main medium information, and an array of song structures.

Preliminary Changes for the Albums Screen

Before adding the processes of manipulating the song records, you need to make a few additional modifications to several of your existing files. Changes need to be made to RECOFREC.C, RECOFREC.H, and RECORDS.H.

The RECOFREC.C source file needs several additional global variables. These variables are needed by many of the musical item functions. You should add the following global declarations to the beginning of the RECOFREC.C source file:

```
FILE *song_fp;          /* For tracking songs */
int total_songs;        /* Nbr of songs in the song file */
```

The additional FILE pointer, `song_fp`, will be used to point to the data file that will hold the song records. The `total_songs` value will keep track of the total number of songs in the song file. The importance of `total_songs` will be seen later in the Musical Items screen's code.

As with the Medium Code screen, the RECOFREC.H header file should receive a few modifications also. To this header file you should add the prototypes for the new functions you will be adding. The following are the function prototypes that should be included in RECOFREC.H for the Musical Items screen:

```
/*-----------------------------*
 * Prototypes for albums screen *
 *-----------------------------*/

int  do_albums_actionbar(void);
void display_albums_help(void);

int  clear_albums_fields(void);
```

```
void draw_albums_screen(void);

int  verify_adata(void);
int  add_adata(void);
int  add_album_rec(void);
int  del_alb_rec(void);
int  proc_alb_rec(int);

int  open_songs(void);
```

Each of these new prototypes will be covered as the listings for the Musical Items screen are presented. In addition to adding the preceding prototypes to the RECOFREC.H header file, you should also ensure that you have all the necessary structures in the RECORDS.H header file. There should be three separate structures for the albums screen. These are:

```
typedef struct
{
   char title[ 30+1 ];
   char group[ 25+1];
   char medium_code[2+1];
   DATE date_purch;
   char cost[ 5+1 ];
   char value[ 5+1 ];
   int  nbr_songs;
   int  del_flag;

} ALBUM_REC;

typedef struct
{
   int  del_flag;
   char title[40+1];
   char minutes[2+1];
   char seconds[2+1];
   int  next;          /* next song record */

} SONG_REC;

typedef struct
{
   char title[ 30+1 ];
   int  nbr_songs;
   int  del_flag;
   unsigned short prev;
   unsigned short next;
   unsigned short data;
   unsigned short song;

} ALBUM_INDEX;
```

The ALBUM_REC structure and the SONG_REC structure should be familiar for the most part. The ALBUM_REC structure has one additional field called del_flag that will be used to signal if the record has been marked as deleted. This will be covered later.

The ALBUM_INDEX structure is new. This structure is similar to the index file presented for the Medium Code screen; however, there are three differences. First, you'll notice that this structure also has a deleted flag, del_flag. The structure also has a nbr_songs field and a song field. The nbr_songs field will be used to hold the total number of songs saved with this album. The song field will be used to hold the record number in the song data file where the first song is stored. This is simply an index for the song file. Figure 18.1 presents a representation of these structures. Figure 18.2 shows portions of these structures with data.

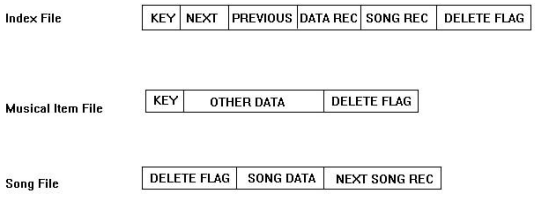

Figure 18.1. *A representation of the musical item structures.*

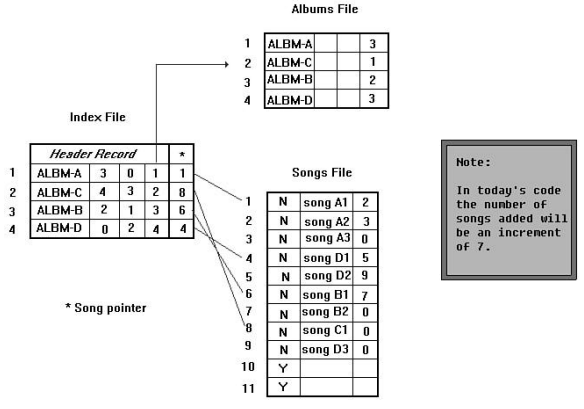

Figure 18.2. *The musical item structures with data.*

The Musical Items Screen

In addition to the RECORDS.H header file, the RECOFREC.C file, and the RECOFREC.H header file, the ALBUMS.C file also will need several changes.

Because you haven't had many listings involving the Musical Items screen, the entire ALBUMS.C listing is presented in Listing 18.2.

Warning: Only the differences from the medium code listing presented on Day 17 will be analyzed in the following listings.

Type

Listing 18.2. ALBUMS.C. The Musical Items screen's main file.

```
1:  /*=========================================================
2:   * Filename: albums.c
3:   *
4:   * Author:   Bradley L. Jones
5:   *           Gregory L. Guntle
6:   *
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 <stdlib.h>
16:  #include <string.h>
17:  #include <conio.h>        /* for getch() */
18:  #include "tyac.h"
19:  #include "records.h"
20:
21:  /*--------------------*
22:   *     prototypes     *
23:   *--------------------*/
24:  #include "recofrec.h"
25:
26:  void draw_albums_screen( void );
27:  void draw_albums_prompts( void );
28:  void display_albums_fields( void );
29:
30:  int  clear_albums_fields(void);
31:  int  get_albums_data( int row );
32:  int  get_albums_input_data( void );
33:  void display_albums_help(void);
34:
35:  /*------------------*
36:   * Global Variables  *
```

continues

Listing 18.2. continued

```
37:   *------------------*/
38:
39:  extern FILE *idx_fp;      /* Main File ptr to data file */
40:  extern FILE *db_fp;       /* Data file */
41:  extern FILE *song_fp;     /* Pointer for songs */
42:  extern nbr_records;       /* Total # of rec for albums */
43:  extern int start_rec;
44:  extern int disp_rec;
45:  extern int total_songs;
46:
47:  /*------------------*
48:   * Defined constants*
49:   *------------------*/
50:
51:  #define ALBUMS_DBF   "ALBUMS.DBF"
52:  #define ALBUMS_IDX   "ALBUMS.IDX"
53:  #define SONGS_DBF    "SONGS.DBF"
54:  #define HELP_DBF     "ALBUMS.HLP"
55:
56:  /*------------------------*
57:   * structure declarations *
58:   *------------------------*/
59:
60:  ALBUM_REC albums;
61:  ALBUM_REC *p_albums = &albums;
62:  ALBUM_REC alb_prev;
63:  SONG_REC  songs[7];
64:
65:  int song_pg;        /* Tracks page of songs being displayed */
66:  int max_song_pg;    /* Max number of pages allocated to     */
67:                      /*   hold the songs                     */
68:  char *song_ptr;     /* Ptr to dynamic mem where songs are   */
69:  char *temp_ptr;     /* Used for reallocating memory */
70:
71:  #define ONE_SONG    sizeof(songs[0])
72:  #define SEVEN_SONGS sizeof(songs)
73:
74:  /*=======================================================*
75:   *   do_albums_screen()                                  *
76:   *=======================================================*/
77:
78:  int do_albums_screen(void)
79:  {
80:    int rv;
81:
82:    /* Open both Index and DBF file */
83:    if ( (rv = open_files(ALBUMS_IDX, ALBUMS_DBF)) == 0 )
84:    {
85:      if ( (rv = open_songs()) == NO_ERROR)
86:      {
```

```
87:        /* Try and get enough mem for
88:           holding 7 songs (1 screen) */
89:        song_ptr = (char *)malloc(SEVEN_SONGS);
90:        /* Check for mem error */
91:        if (song_ptr != NULL)   /* No error - continue */
92:        {
93:          /* Setup for adding new records at beginning */
94:          memset(&alb_prev, '\0', sizeof(alb_prev));
95:          max_song_pg = 1;      /* Only one pg of mem */
96:          song_pg = 1;          /* Displaying 1 pg of songs */
97:
98:          disp_rec = 0;         /* Initialize displaying rec # */
99:
100:         clear_albums_fields();
101:         draw_albums_screen();
102:
103:         get_albums_input_data();
104:
105:         fclose(song_fp);         /* Close song DBF */
106:         rv = close_files();      /* Close IDX and DBF file */
107:         free(song_ptr);          /* Release memory */
108:       }
109:       else
110:       {
111:         display_msg_box(
112:               "Fatal Error - Unable to allocate memory!",
113:               ct.err_fcol, ct.err_bcol );
114:
115:         fclose(song_fp);         /* Close song DBF */
116:         rv = close_files();      /* Close IDX and DBF file */
117:         exit(1);
118:       }
119:     }
120:   }
121:   else
122:   {
123:      display_msg_box("Error opening ALBUMS files...",
124:                      ct.err_fcol, ct.err_bcol );
125:   }
126:   return(rv);
127: }
128:
129: /*------------------------*
130:  *   draw_albums_screen() *
131:  *------------------------*/
132:
133: void draw_albums_screen( void )
134: {
135:    char rec_disp[6];
136:
```

continues

Listing 18.2. continued

```
137:    draw_borders("  Musical Items  "); /* draw screen background */
138:
139:    write_string("<F1=Help>   <F3=Exit>   <PgUp/PgDn=More Songs>",
140:                  ct.abar_fcol, ct.abar_bcol, 24, 3);
141:
142:    write_string( " File   Edit   Search   Help",
143:          ct.abar_fcol, ct.abar_bcol, 1, 2);
144:
145:    draw_albums_prompts();
146:    display_albums_fields();
147: }
148:
149: /*------------------------*
150:  *   draw_albums_prompts()*
151:  *------------------------*/
152:
153: void draw_albums_prompts( void )
154: {
155:    int  ctr;
156:    char tmp[10];
157:
158:    write_string("Title:",
159:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 4, 3 );
160:    write_string("Group:",
161:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 5, 3 );
162:    write_string("Medium:",
163:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 7, 3 );
164:    write_string("Date Purchased:    /  /",
165:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 9, 3 );
166:    write_string("Cost:  $   .",
167:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 9, 33 );
168:    write_string("Value:  $   .",
169:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 9, 52 );
170:    write_string("Track   Song Title",
171:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 11, 7 );
172:    write_string("Time",
173:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 11, 59 );
174:    for( ctr = 0; ctr < 7; ctr++ )
175:    {
176:        sprintf(tmp, "%02d:",((song_pg-1)*7+ctr+1) );
177:        write_string( tmp,
178:              ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 13+ctr, 8 );
179:        write_string(":",
180:              ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 13+ctr, 61 );
181:    }
182:
183:    write_string("Total Album Time:   :  :",
184:                  ct.fld_prmpt_fcol, ct.fld_prmpt_bcol, 21, 39 );
185:
186:    /* Track information */
```

```
187:
188: }
189:
190: /*------------------------*
191:  *   draw_albums_fields()  *
192:  *------------------------*/
193:
194: void display_albums_fields( void )
195: {
196:    int ctr;
197:    char tmp[7] = { 0, 0, 0, 0, 0, 0, 0 }; /* set to nulls */
198:    char under_40[41] =
199:                {"_____________________________________"};
200:
201:    write_string(under_40+10,   /* 30 underscores */
202:                 ct.fld_fcol, ct.fld_bcol, 4, 12 );
203:    write_string(under_40+15,     /* 25 underlines */
204:                 ct.fld_fcol, ct.fld_bcol, 5, 12 );
205:    write_string("__", ct.fld_fcol, ct.fld_bcol, 7, 13 );
206:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 20 );
207:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 23 );
208:    write_string("__", ct.fld_fcol, ct.fld_bcol, 9, 26 );
209:
210:    write_string("___", ct.fld_fcol, ct.fld_bcol, 9, 41 );
211:    write_string("__",  ct.fld_fcol, ct.fld_bcol, 9, 45 );
212:
213:    write_string("___", ct.fld_fcol, ct.fld_bcol, 9, 61 );
214:    write_string("__",  ct.fld_fcol, ct.fld_bcol, 9, 65 );
215:
216:    for( ctr = 0; ctr < 7; ctr++)
217:    {
218:       write_string(under_40,
219:                    ct.fld_fcol, ct.fld_bcol, 13+ctr, 16 );
220:       write_string("__",
221:                    ct.fld_fcol, ct.fld_bcol, 13+ctr, 59 );
222:       write_string("__",
223:                    ct.fld_fcol, ct.fld_bcol, 13+ctr, 62 );
224:    }
225:
226:    write_string("__", ct.fld_fcol, ct.fld_bcol, 21, 57);
227:    write_string("__", ct.fld_fcol, ct.fld_bcol, 21, 60);
228:    write_string("__", ct.fld_fcol, ct.fld_bcol, 21, 63);
229:
230:    /*** display data, if exists ***/
231:
232:    write_string(albums.title, ct.fld_fcol, ct.fld_bcol, 4, 12);
233:    write_string(albums.group, ct.fld_fcol, ct.fld_bcol, 5, 12);
234:    write_string(albums.medium_code,
235:                  ct.fld_fcol, ct.fld_bcol, 7, 13 );
236:
```

continues

Listing 18.2. continued

```
237:    strncpy( tmp, albums.date_purch.month, 2 );
238:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 20 );
239:    strncpy( tmp+4, albums.date_purch.day, 2 );
240:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 23 );
241:    strncpy( tmp, albums.date_purch.year, 2 );
242:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 26 );
243:
244:    strncpy(tmp, albums.cost, 3);
245:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 41 );
246:    strncpy(tmp, albums.cost+3, 2 );
247:    tmp[2] = NULL;
248:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 45 );
249:
250:    strncpy(tmp, albums.value, 3);
251:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 61 );
252:    strncpy(tmp, albums.value+3, 2 );
253:    tmp[2] = NULL;
254:    write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 65 );
255:
256:    /* Get songs from appropriate memory location and */
257:    /*    move into the structured array */
258:    memcpy(&songs,
259:           song_ptr+((song_pg-1)*SEVEN_SONGS), SEVEN_SONGS);
260:
261:    /* song title information */
262:    for( ctr = 0; ctr < 7; ctr++ )
263:    {
264:       write_string(songs[ctr].title,
265:            ct.fld_fcol, ct.fld_bcol, 13+ctr, 16 );
266:       write_string(songs[ctr].minutes,
267:            ct.fld_fcol, ct.fld_bcol, 13+ctr, 59 );
268:       write_string(songs[ctr].seconds,
269:            ct.fld_fcol, ct.fld_bcol, 13+ctr, 62 );
270:      /* calc total here. */
271:    }
272:    /* finish total count and print here */
273: }
274:
275: /*-----------------------------------------------------*
276:  *   get_albums_input_data()                           *
277:  *-----------------------------------------------------*/
278: int get_albums_input_data( void )
279: {
280:     int   position,
281:           rv,
282:           okay,               /* used with edits */
283:           loop = TRUE;
284:
285:     /*  Set up exit keys.  */
286:      static char fexit_keys[ 19 ] = { F1, F2, F3, F4, F5, F6,
```

```
287:                        F7, F8, F10,
288:                        ESC_KEY, PAGE_DN, PAGE_UP, CR_KEY,
289:                        TAB_KEY, ENTER_KEY, SHIFT_TAB,
290:                        DN_ARROW, UP_ARROW, NULL };
291:
292:     static char *exit_keys = fexit_keys;
293:     getline( SET_EXIT_KEYS, 0, 18, 0, 0, 0, exit_keys );
294:
295:     /*** setup colors and default keys ***/
296:     getline( SET_DEFAULTS, 0, 0, 0, 0, 0, 0 );
297:     getline( SET_NORMAL, 0, ct.fld_fcol, ct.fld_bcol,
298:                             ct.fld_high_fcol, ct.fld_high_bcol, 0 );
299:     getline( SET_UNDERLINE, 0, ct.fld_fcol, ct.fld_bcol,
300:                             ct.fld_high_fcol, ct.fld_high_bcol, 0 );
301:     getline( SET_INS, 0, ct.abar_fcol, ct.abar_bcol, 24, 76, 0 );
302:
303:     position = 0;
304:
305:     while( loop == TRUE )      /**  get data for top fields  **/
306:     {
307:        switch( (rv = get_albums_data( position )) )
308:        {
309:           case CR_KEY    :
310:           case TAB_KEY   :
311:           case ENTER_KEY :
312:           case DN_ARROW  : /* go down a field */
313:                            ( position == 30 ) ?
314:                                  (position = 0 ) : position++;
315:                            break;
316:
317:           case SHIFT_TAB :
318:           case UP_ARROW  : /* go up a field */
319:                            ( position == 0 ) ?
320:                                  ( position = 30 ) : position--;
321:                            break;
322:
323:           case ESC_KEY :
324:           case F3      :   /* exit back to main menu */
325:                    if( (yes_no_box( "Do you want to exit?",
326:                           ct.db_fcol, ct.db_bcol )) == 'Y' )
327:                    {
328:                        loop = FALSE;
329:                    }
330:                    break;
331:
332:           case F4: /* add data */
333:                    /* Save songs back into memory location */
334:                    memcpy(song_ptr+((song_pg-1)*SEVEN_SONGS),
```

continues

Listing 18.2. continued

```
335:                          &songs, SEVEN_SONGS);
336:                    rv = add_adata();
337:                    if ( rv == NO_ERROR )
338:                    {
339:                      disp_rec = 0;
340:                      reset_memory();
341:                      clear_albums_fields();
342:                      draw_albums_screen();
343:                      position = 0;
344:                    }
345:                    else  /* Only do next part if File I/O  */
346:                    if ( rv > NO_ERROR )
347:                    {
348:                      display_msg_box(
349:                           "Fatal Error writing data...",
350:                           ct.err_fcol, ct.err_bcol );
351:                      exit(1);
352:                    }
353:                    break;
354:
355:            case F5: /* Change Record */
356:                    /* Save songs back into memory location */
357:                    memcpy(song_ptr+((song_pg-1)*SEVEN_SONGS),
358:                           &songs, SEVEN_SONGS);
359:                    rv = add_adata();
360:                    if ( rv == NO_ERROR )
361:                    {
362:                      disp_rec = 0;
363:                      reset_memory();
364:                      clear_albums_fields();
365:                      draw_albums_screen();
366:                      position = 0;
367:                    }
368:                    else  /* Only do next part if File I/O  */
369:                    if ( rv > NO_ERROR )
370:                    {
371:                      display_msg_box(
372:                           "Fatal Error changing data...",
373:                           ct.err_fcol, ct.err_bcol );
374:                      exit(1);
375:                    }
376:                    break;
377:
378:            case F6: /* Delete data */
379:                    /* Make sure rec is on screen */
380:                    if( (yes_no_box( "Delete record ?",
381:                          ct.db_fcol, ct.db_bcol )) == 'Y' )
382:                    {
383:                      rv = del_alb_rec();
384:                      if (rv == NO_ERROR)
```

```
385:                     {
386:                       disp_rec = 0;
387:                       clear_albums_fields();
388:                       draw_albums_screen();
389:                       position = 0;
390:                     }
391:                     else
392:                     {
393:                       display_msg_box(
394:                           "Fatal Error deleting data...",
395:                            ct.err_fcol, ct.err_bcol );
396:                       exit(1);
397:                     }
398:                   }
399:                   break;
400:
401:          case F7: /* Next record */
402:                     rv = proc_alb_rec(NEXT_REC);
403:                     if ( rv == NO_ERROR )
404:                     {
405:                        draw_albums_screen();
406:                        position = 0;
407:                     }
408:                     else
409:                     {
410:                       display_msg_box(
411:                            "Fatal Error processing data...",
412:                            ct.err_fcol, ct.err_bcol );
413:                       exit(1);
414:                     }
415:                     break;
416:
417:          case F8: /* Prev record */
418:                     rv = proc_alb_rec(PREVIOUS_REC);
419:                     if ( rv == NO_ERROR )
420:                     {
421:                        draw_albums_screen();
422:                        position = 0;
423:                     }
424:                     else
425:                     {
426:                       display_msg_box(
427:                            "Fatal Error processing data...",
428:                            ct.err_fcol, ct.err_bcol );
429:                       exit(1);
430:                     }
431:                     break;
432:
433:       case F10: /* action bar */
434:                 rv = do_albums_actionbar();
```

continues

Listing 18.2. continued

```
435:
436:                 if( rv == F3 )
437:                 {
438:                    if( (yes_no_box( “Do you want to exit?”,
439:                          ct.db_fcol, ct.db_bcol )) == ‘Y’ )
440:                    {
441:                        loop = FALSE;
442:                    }
443:                 }
444:
445:                 position = 0;
446:                 break;
447:
448:       case PAGE_DN :   /* Used for display 7 songs @ time */
449:                 /* First resave - structure to memory */
450:                 memcpy(song_ptr+((song_pg-1)*SEVEN_SONGS),
451:                        &songs, SEVEN_SONGS);
452:                 song_pg++;     /* Get next page  */
453:                 /* Only allow 13 pages (91 songs) */
454:                 /* because displaying only 2 digit track # */
455:                 if (song_pg > 13)
456:                 {
457:                   boop();
458:                   song_pg--;
459:                 }
460:                 else
461:                 {
462:                   /* Do we need to allocate more memory */
463:                   if (song_pg > max_song_pg )
464:                   {
465:                     temp_ptr = song_ptr;   /* Save original */
466:                     song_ptr = (char *)realloc(song_ptr,
467:                                   song_pg*SEVEN_SONGS);
468:
469:                     /* check to see if memory was obtained. */
470:                     if (song_ptr == NULL)
471:                     {
472:                       display_msg_box(“Memory not available!”,
473:                                ct.db_fcol, ct.db_bcol);
474:                       song_ptr = temp_ptr; /* restore orig. */
475:                       song_pg--;
476:                     }
477:                     else
478:                     {
479:                       memset(song_ptr+((song_pg-1)*SEVEN_SONGS),
480:                              ‘\0’, SEVEN_SONGS);
481:                       max_song_pg++;         /* Up the pages */
482:                     }
483:                   }
484:                 }
```

```
485:
486:                 position = 10;  /* Start at top of song */
487:                 draw_albums_screen();  /* Redraw screen */
488:                 break;
489:
490:       case PAGE_UP :   /* Go up a page of songs */
491:                 /* First resave - structure to memory */
492:                 memcpy(song_ptr+((song_pg-1)*SEVEN_SONGS),
493:                        &songs, SEVEN_SONGS);
494:                 song_pg--;
495:                 if (song_pg < 1)
496:                 {
497:                   boop();
498:                   song_pg++;
499:                 }
500:                 else
501:                 {
502:                   position = 10;  /* Start at top of song */
503:                   draw_albums_screen();  /* Redraw screen */
504:                 }
505:                 break;
506:
507:       case F1:  /* context sensitive help */
508:                 display_context_help( HELP_DBF, position );
509:                 break;
510:
511:       case F2:  /* extended help */
512:                 display_medium_help();
513:                 break;
514:
515:       default:  /* error */
516:                 display_msg_box( " Error ",
517:                    ct.err_fcol, ct.err_bcol );
518:                 break;
519:
520:         }     /* end of switch */
521:     }         /* end of while  */
522:
523:     return( rv );
524: }
525:
526: /*-------------------------------------------------------*
527:  *   get_albums_data()                                   *
528:  *-------------------------------------------------------*/
529:
530: int get_albums_data( int row )
531: {
532:     int  rv;
533:     char tmp[4] = { 0, 0, 0, 0 }; /* set to null values */
534:
```

continues

Listing 18.2. continued

```
535:     switch( row )
536:     {
537:        case 0 :
538:           rv = getline( GET_ALPHA, 0,  4, 12, 0, 30,
539:                         albums.title);
540:           break;
541:        case 1 :
542:           rv = getline( GET_ALPHA, 0,  5, 12, 0, 25,
543:                         albums.group);
544:           break;
545:        case 2 :
546:           rv = getline( GET_ALPHA, 0,  7, 13, 0,  2,
547:                         albums.medium_code);
548:           break;
549:        case 3 :
550:           strncpy( tmp, albums.date_purch.month, 2 );
551:           rv = getline( GET_NUM,   0,  9, 20, 0,  2, tmp );
552:           zero_fill_field(tmp, 2);
553:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 20);
554:           strncpy( albums.date_purch.month, tmp, 2);
555:           break;
556:        case 4 :
557:           strncpy( tmp, albums.date_purch.day, 2 );
558:           rv = getline( GET_NUM,   0,  9, 23, 0,  2, tmp );
559:           zero_fill_field(tmp, 2);
560:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 23);
561:           strncpy( albums.date_purch.day, tmp, 2 );
562:           break;
563:        case 5 :
564:           strncpy( tmp, albums.date_purch.year, 2 );
565:           rv = getline( GET_NUM,   0,  9, 26, 0,  2, tmp );
566:           zero_fill_field(tmp, 2);
567:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 26);
568:           strncpy( albums.date_purch.year, tmp, 2 );
569:           break;
570:        case 6 :
571:           strncpy( tmp, albums.cost, 3 );
572:           rv = getline( GET_NUM,   0,  9, 41, 0,  3, tmp );
573:           zero_fill_field(tmp, 3);
574:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 41);
575:           strncpy( albums.cost, tmp, 3 );
576:           break;
577:        case 7 :
578:           strncpy( tmp, albums.cost+3, 2 );
579:           rv = getline( GET_NUM,   0,  9, 45, 0,  2, tmp );
580:           zero_fill_field(tmp, 2);
581:           write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 45);
582:           strncpy( albums.cost+3, tmp, 2 );
583:           break;
```

```
584:         case 8 :
585:            strncpy( tmp, albums.value, 3 );
586:            rv = getline( GET_NUM,   0,  9, 61, 0,  3, tmp );
587:            zero_fill_field(tmp, 3);
588:            write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 61);
589:            strncpy( albums.value, tmp, 3 );
590:            break;
591:         case 9 :
592:            strncpy( tmp, albums.value+3, 2 );
593:            rv = getline( GET_NUM,   0,  9, 65, 0,  2, tmp );
594:            zero_fill_field(tmp, 2);
595:            write_string(tmp, ct.fld_fcol, ct.fld_bcol, 9, 65);
596:            strncpy( albums.value+3, tmp, 2 );
597:            break;
598:         case 10 :
599:            rv = getline( GET_ALPHA, 0,  13, 16, 0, 40,
600:                                                songs[0].title);
601:            break;
602:         case 11 :
603:            rv = getline( GET_NUM, 0, 13, 59, 0, 2,
604:                                                songs[0].minutes);
605:            zero_fill_field(songs[0].minutes, 2);
606:            write_string(songs[0].minutes,
607:                         ct.fld_fcol, ct.fld_bcol, 13, 59);
608:            break;
609:         case 12 :
610:            rv = getline( GET_NUM, 0, 13, 62, 0, 2,
611:                                                songs[0].seconds);
612:            zero_fill_field(songs[0].seconds, 2);
613:            write_string(songs[0].seconds,
614:                         ct.fld_fcol, ct.fld_bcol, 13, 62);
615:            break;
616:         case 13 :
617:            rv = getline( GET_ALPHA, 0, 14, 16, 0, 40,
618:                                                songs[1].title);
619:            break;
620:         case 14 :
621:            rv = getline( GET_NUM, 0, 14, 59, 0, 2,
622:                                                songs[1].minutes);
623:            zero_fill_field(songs[1].minutes, 2);
624:            write_string(songs[1].minutes,
625:                         ct.fld_fcol, ct.fld_bcol, 14, 59);
626:            break;
627:         case 15 :
628:            rv = getline( GET_NUM, 0, 14, 62, 0, 2,
629:                                                songs[1].seconds);
630:            zero_fill_field(songs[1].seconds, 2);
631:            write_string(songs[1].seconds,
632:                         ct.fld_fcol, ct.fld_bcol, 14, 62);
633:            break;
```

continues

Listing 18.2. continued

```
634:         case 16 :
635:            rv = getline( GET_ALPHA, 0, 15, 16, 0, 40,
636:                                                songs[2].title);
637:            break;
638:         case 17 :
639:            rv = getline( GET_NUM, 0, 15, 59, 0, 2,
640:                                                songs[2].minutes);
641:            zero_fill_field(songs[2].minutes, 2);
642:            write_string(songs[2].minutes,
643:                         ct.fld_fcol, ct.fld_bcol, 15, 59);
644:            break;
645:         case 18 :
646:            rv = getline( GET_NUM, 0, 15, 62, 0, 2,
647:                                                songs[2].seconds);
648:            zero_fill_field(songs[2].seconds, 2);
649:            write_string(songs[2].seconds,
650:                         ct.fld_fcol, ct.fld_bcol, 15, 62);
651:            break;
652:         case 19 :
653:            rv = getline( GET_ALPHA, 0, 16, 16, 0, 40,
654:                                                songs[3].title);
655:            break;
656:         case 20 :
657:            rv = getline( GET_NUM, 0, 16, 59, 0, 2,
658:                                                songs[3].minutes);
659:            zero_fill_field(songs[3].minutes, 2);
660:            write_string(songs[3].minutes,
661:                         ct.fld_fcol, ct.fld_bcol, 16, 59);
662:            break;
663:         case 21 :
664:            rv = getline( GET_NUM, 0, 16, 62, 0, 2,
665:                                                songs[3].seconds);
666:            zero_fill_field(songs[3].seconds, 2);
667:            write_string(songs[3].seconds,
668:                         ct.fld_fcol, ct.fld_bcol, 16, 62);
669:            break;
670:         case 22 :
671:            rv = getline( GET_ALPHA, 0, 17, 16, 0, 40,
672:                                                songs[4].title);
673:            break;
674:         case 23 :
675:            rv = getline( GET_NUM, 0, 17, 59, 0, 2,
676:                                                songs[4].minutes);
677:            zero_fill_field(songs[4].minutes, 2);
678:            write_string(songs[4].minutes,
679:                         ct.fld_fcol, ct.fld_bcol, 17, 59);
680:            break;
681:         case 24 :
682:            rv = getline( GET_NUM, 0, 17, 62, 0, 2,
683:                                                songs[4].seconds);
```

```
684:            zero_fill_field(songs[4].seconds, 2);
685:            write_string(songs[4].seconds,
686:                         ct.fld_fcol, ct.fld_bcol, 17, 62);
687:            break;
688:         case 25 :
689:            rv = getline( GET_ALPHA, 0, 18, 16, 0, 40,
690:                                              songs[5].title);
691:            break;
692:         case 26 :
693:            rv = getline( GET_NUM, 0, 18, 59, 0, 2,
694:                                              songs[5].minutes);
695:            zero_fill_field(songs[5].minutes, 2);
696:            write_string(songs[5].minutes,
697:                         ct.fld_fcol, ct.fld_bcol, 18, 59);
698:            break;
699:         case 27 :
700:            rv = getline( GET_NUM, 0, 18, 62, 0, 2,
701:                                              songs[5].seconds);
702:            zero_fill_field(songs[5].seconds, 2);
703:            write_string(songs[5].seconds,
704:                         ct.fld_fcol, ct.fld_bcol, 18, 62);
705:            break;
706:         case 28 :
707:            rv = getline( GET_ALPHA, 0, 19, 16, 0, 40,
708:                                              songs[6].title);
709:            break;
710:         case 29 :
711:            rv = getline( GET_NUM, 0, 19, 59, 0, 2,
712:                                              songs[6].minutes);
713:            zero_fill_field(songs[6].minutes, 2);
714:            write_string(songs[6].minutes,
715:                         ct.fld_fcol, ct.fld_bcol, 19, 59);
716:            break;
717:         case 30 :
718:            rv = getline( GET_NUM, 0, 19, 62, 0, 2,
719:                                              songs[6].seconds);
720:            zero_fill_field(songs[6].seconds, 2);
721:            write_string(songs[6].seconds,
722:                         ct.fld_fcol, ct.fld_bcol, 19, 62);
723:            break;
724:      }
725:      return( rv );
726: }
727:
728: /*-------------------------------------------------------*
729:  *   clear_albums_fields()                               *
730:  *-------------------------------------------------------*/
731:
732: int clear_albums_fields(void)
733: {
```

continues

Listing 18.2. continued

```
734:    int ctr;
735:
736:    getline(CLEAR_FIELD, 0, 31, 0, 0, 0, albums.title);
737:    getline(CLEAR_FIELD, 0, 26, 0, 0, 0, albums.group);
738:    getline(CLEAR_FIELD, 0,  3, 0, 0, 0, albums.medium_code);
739:    getline(CLEAR_FIELD, 0,  2, 0, 0, 0, albums.date_purch.month);
740:    getline(CLEAR_FIELD, 0,  2, 0, 0, 0, albums.date_purch.day);
741:    getline(CLEAR_FIELD, 0,  2, 0, 0, 0, albums.date_purch.year);
742:    getline(CLEAR_FIELD, 0,  6, 0, 0, 0, albums.cost);
743:    getline(CLEAR_FIELD, 0,  6, 0, 0, 0, albums.value);
744:
745:    for( ctr = 0; ctr < 7; ctr++ )
746:    {
747:       getline(CLEAR_FIELD, 0, 41, 0, 0, 0, songs[ctr].title);
748:       getline(CLEAR_FIELD, 0,  3, 0, 0, 0, songs[ctr].minutes);
749:       getline(CLEAR_FIELD, 0,  3, 0, 0, 0, songs[ctr].seconds);
750:       songs[ctr].del_flag = FALSE;
751:       songs[ctr].next = 0;
752:    }
753:    /* Save to memory location */
754:    memcpy(song_ptr+((song_pg-1)*SEVEN_SONGS),
755:           &songs, SEVEN_SONGS);
756:    return(0);
757: }
758:
759: /*-----------------------------------------------------*
760:  *   add_adata()                                       *
761:  *                                                     *
762:  *   Returns  0 - No Errors                            *
763:  *           >0 - File I/O Error                       *
764:  *           <0 - Missing info before can store        *
765:  *-----------------------------------------------------*/
766:
767: int add_adata()
768: {
769:    int rv = NO_ERROR;
770:
771:    /* Verify data fields */
772:    rv = verify_adata();
773:    if ( rv == NO_ERROR )
774:    {
775:      /* Check to see if matches old rec */
776:      /* If match - then update db record only */
777:      if ( stricmp(alb_prev.title, albums.title) == 0)
778:      {
779:         /* Rewrite ALBUMS DATA */
780:         rv = put_rec(disp_rec, db_fp,
781:              sizeof(albums), 0, (char *)&albums);
782:
783:         /* Now Update SONGS */
```

```
784:          if (rv == NO_ERROR)
785:             rv = update_songs();
786:
787:          display_msg_box("Record updated!",
788:               ct.db_fcol, ct.db_bcol);
789:       }
790:       else
791:       {
792:         /* Keys no longer match - need to
793:            add this key as a new one */
794:         rv = add_album_rec();
795:         display_msg_box("Record added!",
796:               ct.db_fcol, ct.db_bcol);
797:       }
798:    }
799:
800:    return(rv);
801: }
802:
803: /*-------------------------------------------------------*
804:  *   Verify data fields                                  *
805:  *-------------------------------------------------------*/
806: int verify_adata()
807: {
808:    int rv = NO_ERROR;
809:
810:    if( strlen( albums.title ) == 0 )
811:    {
812:       display_msg_box("Must enter a Title",
813:          ct.err_fcol, ct.err_bcol);
814:       rv = -1;
815:    }
816:    else
817:    {
818:       /* edit date and any other fields */
819:
820:    }
821:
822:    return(rv);
823: }
824:
825: /*-------------------------------------------------------*
826:  *   Opens SONGS DBF                                     *
827:  *-------------------------------------------------------*/
828: int open_songs()
829: {
830:
831:   int rv = NO_ERROR;
832:
833:   if( (song_fp = fopen(SONGS_DBF, "r+b" )) == NULL )
```

continues

Listing 18.2. continued

```
834:   {
835:     /* Doesn't exist - create it */
836:     if( (song_fp = fopen(SONGS_DBF, "w+b")) == NULL )
837:     {
838:       rv = CREATE_ERROR;
839:     }
840:     else
841:     {
842:       total_songs = 0;  /* First time file opened */
843:     }
844:   }
845:   else    /* Get total songs from song file */
846:   {
847:     rv = get_rec(0, song_fp, sizeof(int), 0,
848:                (char *)&total_songs);
849:   }
850:
851:   return(rv);
852: }
853:
854: /*-------------------------------------------------------*
855:  *   display_albums_help()                               *
856:  *-------------------------------------------------------*/
857:
858: void display_albums_help(void)
859: {
860:   int  ctr;
861:   char *scrnbuffer;
862:
863:   char helptext[19][45] = {
864:      "                 Musical Items",
865:      "-------------------------------------------",
866:      "",
867:      "",
868:      "This is a screen for entering musical",
869:      "items such as albums.                ",
870:      "To exit screen, press <F3> or <ESC>. ",
871:      "",
872:      "",
873:      "Up to 91 songs can be entered for each",
874:      "album.  Songs are entered 7 per page. Use",
875:      "The Page Up and Page Down keys to add or",
876:      "view additional songs.",
877:      "",
878:      "",
879:      "",
880:      "     ***  Press any key to continue ***",
881:      "",
882:      "-------------------------------------------" };
```

```
883:
884:    scrnbuffer = save_screen_area(2, 23, 28, 78 );
885:    cursor_off();
886:
887:    grid( 3, 23, 28, 77, ct.shdw_fcol, ct.bg_bcol, 3 );
888:    box(2, 22, 29, 78, SINGLE_BOX, FILL_BOX,
889:                         ct.help_fcol, ct.help_bcol);
890:
891:    for( ctr = 0; ctr < 19; ctr++ )
892:    {
893:      write_string( helptext[ctr],
894:                    ct.help_fcol, ct.help_bcol, 3+ctr, 32 );
895:    }
896:
897:    getch();
898:    cursor_on();
899:    restore_screen_area(scrnbuffer);
900: }
901:
902: /*=========================================================*
903:  *                      end of listing                     *
904:  *=========================================================*/
```


Note: The *Record of Records!* application should have several source files not including the TYAC.LIB library. The files that you should be compiling include:

RECOFREC.C	ABOUT.C	
ALBMABAR.C	GRPSABAR.C	MEDMABAR.C
MMNUABAR.C		
MEDIUM.C	ALBUMS.C	GROUPS.C
ADDMREC.C	DELMREC.C	PROCMREC.C
ADDGREC.C	DELGREC.C	PROCGREC.C

To these files, you should link the TYAC.LIB library, which should contain all the functions learned in this book including those from earlier today. Following, you will be presented with four more files that will need to be compiled and linked with these. These are:

ADDAREC.C DELAREC.C PROCAREC.C SONGS.C

Warning: If you compile the *Record of Records!* application with the preceding ALBUMS.C listing, you may get unresolved externals for `del_alb_rec()`, `proc_alb_rec()`, `add_album_rec()`, `reset_memory()`, `update_songs()`, `add_new_songs()`, and `calc_nbr_songs()`. These new functions will all be covered later today.

While the ALBUMS.C listing is much longer than the other screen listings, this is mainly a result of the number of fields on the screen. Listing 18.2 presents a nearly complete source file. You'll still need to create the musical item I/O functions to eliminate the unresolved externals. In addition, you'll need to update the action bar to also contain the I/O functions. Because the functionality of adding songs is unique, you'll also need to create several additional functions specifically for them.

The beginning of this listing is nearly identical to the medium code and the group information listings. The first real difference comes in line 53. A second data file is defined for holding the songs. Lines 60 to 63 also offer some differences. Line 60 declares an album structure, `ALBUM_REC`. Line 61 declares a pointer to this structure. Line 62 declares a second album structure, `alb_prev`. This is similar to what you have seen before.

Starting in line 63 are several new statements that will be used for holding songs. Line 63 declares an array of `SONG_REC` structures that will be used to hold seven songs. This is the number of songs that are presented on the screen. Two variables, `song_pg` and `max_song_pg`, are declared in lines 65 and 66 to keep track of song pages. Because seven songs can be displayed on the screen at a time, they are grouped together as a page. If the screen is displaying song tracks one to seven, then the first page is displayed. Song tracks eight to 14 make up the second page, 15 to 21 make up the third page, and so on. The `song_pg` variable keeps track of which page is currently displayed. The `max_song_pg` variable keeps track of what the highest page for the current musical item record is.

Because you don't know how many songs there will be, you'll need to dynamically allocate storage space for them. Lines 68 and 69 declare two pointers, `song_ptr` and `temp_ptr`, that will be used for setting up the storage areas. To help in allocating the appropriate amount of memory, lines 71 and 72 contain two defined constants, `ONE_SONG` and `SEVEN_SONGS`. These will be used in allocating the appropriate amounts of memory later in the listing.

The processing of the Musical Items screen begins in the same manner as the other entry and edit program. Line 83 calls the `open_files()` function to open the main data and index files. This is followed in line 85 by a call to `open_songs()`. Because `open_files()` only opens one data file and one index file, the song file must be opened separately.

The `open_songs()` function is presented in lines 825 to 852. This function uses the `open()` function to open the file for reading. If the file doesn't exist, then it is created in line 836. If the file is created, then the total number of songs, `total_songs`, is set to zero in line 842. If the file already existed, then line 847 gets the number of songs from the beginning of the file.

Once all the files are opened, memory is allocated to hold the first song page (line 89). If the allocation was successful, then processing continues. In line 94, `memset()` is used to clear the `alb_prev` structure to `NULL` values. Lines 95 and 96 set the `max_song_pg` and `song_pg` variables to one because a new record only has one page. Because a record is not yet displayed, `disp_rec` is set to zero in line 98. At this point, processing begins to follow the same flow as the other screens.

When the files are closed, the song data file must be closed separately because the `close_files()` function only closes the main data file and index file. Line 105 uses `fclose()` to close the song file. In addition to closing the song file, you need to free the dynamic memory that was allocated to hold the song. This is accomplished in line 107.

Expert Tip: When using dynamic memory allocation functions such as `malloc()`, always verify that memory was allocated.

Most of the other functions in the listing operate like their counterparts in the medium code screen. The `clear_albums_fields()` function in lines 728 to 757 uses `getline()` to clear the fields. The `clear_albums_fields()` function also clears out the dynamic memory by copying the cleared song structure to the dynamic memory area (lines 754 to 755). The `draw_albums_screen()` function in lines 129 to 147 also operates in a familiar manner.

The *get_albums_input_data()* Function

The `get_albums_input_data()` function is presented in lines 275 to 524. The overall structure of this function is the same as the Medium Code screen. A few of the cases

have differences that should be reviewed one at a time. The `switch` statement in line 307 calls the `get_albums_data()` function. The `get_albums_data()` is presented in lines 526 to 726. It doesn't offer anything that hasn't already been presented.

The first real difference can be seen in the F4 case, which is used for adding a record. In line 334, the songs that are currently displayed on the screen are copied to the memory area that has been dynamically allocated. This is followed by line 336, which calls the `add_adata()` function to do the actual adding. If the add is successful, then lines 339 to 353 are executed. Line 339 resets the display record to zero, and line 340 calls `reset_memory()` to free the dynamic memory that had been allocated. The `reset_memory()` function is presented in the SONGS.C listing later today. The fields are then cleared and the screen is redrawn.

The F5 case in lines 355 to 376 is used for updating information. This code follows the same flow as the add case. The only difference is that the fatal error message in line 372 is more specific to changing data instead of writing data.

The next few cases aren't really different from the Medium Code screens. The delete case is activated by F6. This function calls `del_alb_rec()` in line 383. The `del_alb_rec()` function will be covered later today. The F7 and F8 cases call `proc_alb_rec()`, which will also be covered later. Both of these functions will have similarities to their medium code screen counterparts.

The Page Up and Page Down Functions

The page up and page down functions are different for the Musical Items screen than they are for the Medium Code or Group Information screens. The page down function displays the next seven songs. The page up function displays the previous seven. If there isn't a previous page, or if the maximum number of pages has been reached, then the system will beep.

Note: Line 455 sets the maximum number of pages that can be displayed to 13.

A structure was declared earlier called `songs`. This structure holds seven songs because that is the number of songs displayed on the screen. The user of the application can enter more than seven songs by pressing page down. Each time page down is pressed, an additional area of dynamic memory is allocated. Each of these areas can be referred to as a page. Each page is declared a size that can hold seven songs. Figure 18.3 illustrates the structure and the dynamic area for pages.

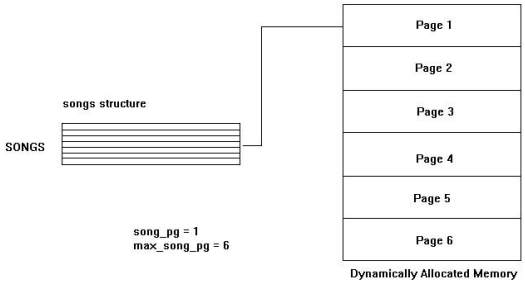

Figure 18.3. *The song structure and the dynamic song pages.*

As you page down, additional pages may be added to the end of the dynamic memory area. The information for the current page is copied to the songs structure. As you page up and page down, information is swapped to and from the `songs` structure. The song information that is currently displayed on the screen is kept in the structure.

Note: You could work directly from the dynamic memory and not with the songs structure; however, the code becomes more complicated.

Page down is presented in lines 448 to 488. The first step is to copy the current `song` structure information to the dynamic memory. This is done in line 450 with the `memcpy()` function. Line 452 then increments the current page, `song_pg`, to the next page.

Review Tip: The `memcpy()` function is an ANSI C function that copies from one memory location to another.

In line 455, a check is done to see if more than 13 pages have been allocated. The limit of 13 pages is subjective. You can raise or lower this limit. If they were already on the 13th page, then the computer beeps in line 457 and the `song_pg` is reset to 13.

If `song_pg` is less than 13, then the next page can be displayed. This is handled in lines 461 to 483. In line 463, a check is done to see if the current song page is within the pages that have already been allocated. This is done by checking the `max_song_pg` value. If the new `song_pg` is greater, then a new page must be allocated. This is done in line 466 using the `realloc()` function to increase the overall size of the allocated memory. If the `realloc()` fails, then line 472 displays an error message, the pointer to the song pages is reset to its original value (line 474), and the song page, `song_pg`, is set back to the original page.

If the allocation was successful, then line 479 is called. This line uses the `memset()` function to clear out the newly-allocated page of memory. Once done, the maximum page counter, `max_song_pg`, is incremented to include the new page.

Review Tip: The `memset()` is an ANSI function that fills a specified memory location with a character value.

The page down logic ends with resetting the screen with the field position being set to 10, the first song field. The Musical Items screen is then redrawn. With this, the page down functionality is complete.

The page up functionality is much simpler. With page up, you don't have to worry about allocating additional pages. Line 492 starts by copying the current song information back to the dynamic memory area. The song page, `song_pg`, is then decremented (line 494). If the line number is less than one, then you are at the beginning of the dynamic memory area. The computer beeps (line 497) and the song page is set back to the original page (line 498). If a previous page is available, then the screen position is set to 10 and the screen is redrawn.

Supportive Functions: The SONGS.C Listing

Due to some of the extra complexity in the Musical Items screen, several additional functions are needed. These are functions that were not present with the medium code listings. Listing 18.3 contains the SONGS.C file. This file contains four functions that are called by other portions of the musical items code. These are `add_new_songs()`, `update_songs()`, `calc_nbr_songs()`, and `reset_memory()`.

Listing 18.3. SONGS.C. Functions needed for the Musical Items screen.

```
1:  /*=======================================================
2:   * Filename: SONGS.C
3:   *
4:   * Author:   Bradley L. Jones
5:   *           Gregory L. Guntle
6:   *
7:   * Purpose: Routines for add/changing and calculating
8:   *          songs for working with I/O.
9:   *
10:  *=======================================================*/
```

```
11:
12:  #include <stdio.h>
13:  #include <stdlib.h>
14:  #include <string.h>
15:  #include "records.h"
16:  #include "tyac.h"
17:
18:  /*------------------*
19:   * Global Variables *
20:   *------------------*/
21:
22:  extern FILE *idx_fp;      /* Main File ptr to data file */
23:  extern FILE *db_fp;       /* Data file */
24:  extern FILE *song_fp;     /* Points to Songs */
25:  extern nbr_records;       /* Total # of rec for albums */
26:  extern int start_rec;
27:  extern int disp_rec;
28:  extern int total_songs;   /* Total songs in the song file */
29:
30:  extern ALBUM_REC albums;
31:  extern ALBUM_REC alb_prev;
32:  extern SONG_REC songs[7];
33:
34:  extern int song_pg;     /* Tracks pg of songs being displayed */
35:  extern int max_song_pg; /* Max number of pages allocated */
36:                          /*     to hold the songs         */
37:  extern char *song_ptr;  /* Ptr to dynamic mem where songs are */
38:
39:  #define ONE_SONG         sizeof(songs[0])
40:  #define SEVEN_SONGS      sizeof(songs)
41:
42:  /*--------------------*
43:   *     prototypes     *
44:   *--------------------*/
45:
46:  #include "recofrec.h"
47:
48:  /*-------------------------------------------------------*
49:   *  add_new_songs();                                     *
50:   *  Returns:  0 - No Error                               *
51:   *            >0 Error - see defines at top of file      *
52:   *-------------------------------------------------------*/
53:  int add_new_songs()
54:  {
55:    int rv = NO_ERROR;
56:    int recnbr;
57:    int pgs, sngs;
58:    int hold_start;
59:
60:    /* Starting point to add record */
61:    hold_start = total_songs+1;
```

continues

Listing 18.3. continued

```
62:    recnbr = hold_start;
63:
64:    /* Loop through the all the pages */
65:    /* Start at first page of songs - 1 page = 7 songs*/
66:    pgs = 0;
67:    while ( pgs < max_song_pg && rv == NO_ERROR )
68:    {
69:      sngs = 0;  /* Reset songs to start at top */
70:      /* Load structure with appropriate memory location */
71:      memcpy(&songs, song_ptr + (pgs*SEVEN_SONGS),
72:               SEVEN_SONGS);
73:
74:      /* Now loop through individual songs */
75:      while ( sngs < 7 && rv == NO_ERROR )
76:      {
77:        /* Set the next pointer to point to the next
78:           possible record */
79:        /* Are we writing last record ? */
80:        if (sngs == 6 && pgs == max_song_pg-1)
81:          songs[sngs].next = 0;
82:        else
83:          songs[sngs].next = recnbr+1;
84:        rv = put_rec(recnbr, song_fp, ONE_SONG, sizeof(int),
85:                          (char *)&songs[sngs]);
86:        sngs++;       /* Next song in structure */
87:        recnbr++;     /* Next record location */
88:      }
89:      pgs++;     /* Next page in memory */
90:    }
91:
92:    if (rv == NO_ERROR)
93:    {
94:       total_songs += (recnbr-hold_start); /* Update # of songs */
95:       /* Save the total to the file */
96:       rv = put_rec(0, song_fp, sizeof(int), 0,
97:                (char *)&total_songs);
98:    }
99:
100:   return(rv);
101: }
102:
103:
104: /*-----------------------------------------------------*
105:  *  update_songs();                                    *
106:  *  Returns:  0 - No Error                             *
107:  *            >0 Error - see defines at top of file    *
108:  *-----------------------------------------------------*/
109: int update_songs()
110: {
111:   int rv=NO_ERROR;
112:   int recnbr;
```

```
113:   SONG_REC one_song;      /* For holding one song */
114:   int pgs, sngs;
115:   ALBUM_INDEX alb_idx;
116:   int next_rec;          /* Holds ptr to next song rec */
117:   int new_nbr_songs;
118:   int adding_new = FALSE;      /* Flag for adding new recs */
119:
120:   /* Calc nbr songs - mostly needed to eliminate
121:      any blank pages - resets max_song_pg to be accurate */
122:   new_nbr_songs = calc_nbr_songs();
123:
124:   /* Get starting record number */
125:   rv = get_rec(disp_rec, idx_fp, sizeof(alb_idx),
126:                sizeof(int)*2, (char *)&alb_idx);
127:   if (rv == NO_ERROR)
128:   {
129:     recnbr = alb_idx.song;     /* Get first record from index */
130:
131:     /* Loop through all the pages */
132:     /* Start at first page of songs - 1 page = 7 songs*/
133:     pgs = 0;
134:     while ( pgs < max_song_pg && rv == NO_ERROR )
135:     {
136:       sngs = 0;  /* Reset songs to start at top */
137:       /* Load structure with appropriate memory location */
138:       memcpy(&songs, song_ptr + (pgs*SEVEN_SONGS),
139:              SEVEN_SONGS);
140:
141:       /* Now loop through individual songs - skip last one */
142:       while ( sngs < 7 && rv == NO_ERROR)
143:       {
144:         if (!adding_new)
145:         {
146:           rv = get_rec(recnbr, song_fp, ONE_SONG,
147:                 sizeof(int), (char *)&one_song);
148:           next_rec = one_song.next;
149:         }
150:
151:         /* Are we at end of chain & there is still more
152:            to come - another page                        */
153:         if (next_rec == 0 && pgs+1<max_song_pg)
154:         {
155:           adding_new = TRUE;
156:         }
157:
158:         if (adding_new)
159:         {
160:           total_songs++;         /* Increase global counter */
161:           next_rec = total_songs;    /* Next rec to put */
162:           /* Ck to see if at end of records to store */
163:           if (pgs+1 == max_song_pg && sngs == 6)
```

continues

Listing 18.3. continued

```
164:           {
165:             next_rec = 0;     /* Mark as end of song */
166:           }
167:         }
168:
169:         if (rv == NO_ERROR)
170:         {
171:           songs[sngs].next = next_rec;
172:           rv = put_rec(recnbr, song_fp, ONE_SONG, sizeof(int),
173:                     (char *)&songs[sngs]);
174:           sngs++;  /* Next song in structure */
175:           recnbr = next_rec;      /* Follow chain */
176:         }
177:       }
178:       pgs++;     /* Next page in memory */
179:     }
180:
181:     /* Have we added new records ? */
182:     /* If so - let's adjust total records */
183:     if (adding_new)
184:     {
185:        total_songs--;    /* Adjust for true number */
186:     }
187:
188:     /* Are we adding more records than previously there ? */
189:     /* If so update index record as well */
190:     if (new_nbr_songs > alb_idx.nbr_songs)
191:     {
192:       alb_idx.nbr_songs = new_nbr_songs;
193:       /* Resave index */
194:       rv = put_rec(disp_rec, idx_fp, sizeof(alb_idx),
195:              sizeof(int)*2, (char *)&alb_idx);
196:     }
197:
198:     /* Resave total number of songs in file */
199:     if (rv == NO_ERROR)
200:     {
201:        rv = put_rec(0, song_fp, sizeof(int), 0,
202:               (char *)&total_songs);
203:     }
204:
205:     fflush(song_fp);
206:   }
207:
208:   return(rv);
209: }
210:
211: /*-------------------------------------------------------*
212:  *  calc_nbr_songs();                                    *
213:  *                                                       *
```

```
214:  *  This calculates the number of songs.  It will       *
215:  *  return a multiple of 7.  It is mostly used to       *
216:  *  determine the true number of pages - so a lot of    *
217:  *  blank song records don't get written to disk.       *
218:  *                                                      *
219:  *  Returns: number of songs (multiple of 7)            *
220:  *------------------------------------------------------*/
221: int calc_nbr_songs()
222: {
223:   int fnd;
224:   int pgs, sngs;
225:   int nbrsongs = 0;
226:
227:   pgs = 0;
228:   while (pgs < max_song_pg)
229:   {
230:     sngs=0;
231:     fnd = FALSE;
232:     /* Load structure with proper mem location */
233:     memcpy(&songs, song_ptr + (pgs*SEVEN_SONGS), SEVEN_SONGS);
234:
235:     while (sngs<7 && !fnd)
236:     {
237:       /* Is there anything in title - if find one
238:          skip rest - use entire block of 7        */
239:       if ( strlen(songs[sngs].title) > 0 )
240:          fnd = TRUE;
241:       else
242:         sngs++;
243:     }
244:
245:     if (fnd)
246:     {
247:        nbrsongs+=7;
248:     }
249:     pgs++;     /* Next page in memory */
250:   }
251:
252:   /* Adjust true number of pages */
253:   /* Make sure at least writing first seven songs */
254:   if (nbrsongs == 0)
255:   {
256:     nbrsongs = 7;
257:   }
258:   max_song_pg = nbrsongs/7;
259:   /* Readjust memory locations as well */
260:   song_ptr = (char *)realloc(song_ptr, max_song_pg*SEVEN_SONGS);
261:   if (song_ptr == NULL)
262:   {
263:     display_msg_box("Memory error !",
264:             ct.err_fcol, ct.err_bcol);
```

continues

Listing 18.3. continued

```
265:     exit(1);
266:   }
267:
268:   return(nbrsongs);
269: }
270:
271: /*-------------------------------------------------------*
272:  *  reset_memory();                                      *
273:  *                                                       *
274:  *  Resets memory - clears it out, adjusts it back to    *
275:  *  a single page, etc.                                  *
276:  *-------------------------------------------------------*/
277: void reset_memory()
278: {
279:   /* Null out current memory block */
280:   memset(song_ptr, '\0', max_song_pg*SEVEN_SONGS);
281:
282:   /* Adjust global numbers */
283:   max_song_pg = 1;
284:   song_pg = 1;
285:
286:   /* reduce memory to hold one page - 7 songs */
287:   song_ptr = (char *)realloc(song_ptr, SEVEN_SONGS);
288:   if (song_ptr == NULL)
289:   {
290:     display_msg_box("Memory error - resizing !",
291:             ct.err_fcol, ct.err_bcol);
292:     exit(1);
293:   }
294:
295: }
```

Analysis The `add_new_songs()` function is presented in lines 48 to 101. This function is used to add songs when a musical item record is being added. This function adds songs to the song data file. Songs are added in groups of seven. Remember, each group of seven is considered a page. For each page, the songs are added one at a time. The `while` statement starting in line 67 controls the looping of the pages.

A second `while` statement in line 75 controls the adding of each of the seven records on each page. As each song is added, it is provided with a pointer to the next song. If the song being added is the last song, then the pointer to the next record is set to zero to signal the end of the song linked list (line 81). If it isn't the last song, then the pointer to the next record is set to the next record number (line 83). Once the next pointer is set, line 84 adds the record using the `put_rec()` function. After adding the record,

the song counter, `sngs`, and the record counter, `recnbr`, are both incremented for the next song.

Once all the songs on each page are added, the processing continues to line 92. It is here that a check is done to ensure that there hasn't been an error. If there hasn't been an error, then the total number of songs is updated. Line 96 updates the first position of the song file with the new total number of songs.

Updating song records is more complicated than adding songs. When you add songs, you are always working from the end of the file. Updating requires reading each record already in the database. Once you reach the last song record, you may need to add new songs to the end. The `update_songs()` function in lines 104 to 209 does just this.

The update function starts off with a call to the `calc_nbr_songs()` function. This function is presented in lines 211 to 269. The `calc_nbr_songs()` loops through each page to see if a song has been entered. This function uses a flag, `fnd`, to signal if there is a record on a page. If a record is found on a page, then the page must be added, so the number of songs, `nbrsongs`, is incremented by seven.

After checking each page for songs, line 254 checks to see if there were any songs. If there weren't any songs, then line 256 sets the number of songs to seven so that at least one page of songs is added. Line 258 determines how many pages of songs there are. This is done by dividing the number of songs by seven. The global variable, `max_song_pg`, is then reset to this number. The allocated memory is then adjusted accordingly in line 260. The final result for the number of songs is returned to the calling function in line 268.

The final function in this listing is the `reset_memory()` function in lines 271 to 295. This function frees the memory above the first page. When you are ready to start a new record, you need to reset the amount of memory that is being used. This function starts by clearing out the first page of memory. Lines 283 and 284 reset the maximum song page to 1 and the current song page to 1. The final step to resetting the memory is to reset the allocated amount down to a single page. This is done with the `realloc()` function in line 287. As should always be done, a check is performed to ensure that the allocation was competed without an error.

Adding a Record: The *add_album()_rec* Function

The `add_album_rec()` function is used to add a new record to the Musical Items screen's data files. This function is presented in Listing 18.4.

Type

Listing 18.4. ADDAREC.C. Adding a medium code.

```
1:  /*=========================================================
2:   * Filename: ADDAREC.C
3:   *
4:   * Author:   Bradley L. Jones
5:   *           Gregory L. Guntle
6:   *
7:   * Purpose: Adds a record to the ALBUMS DBF
8:   *
9:   *=========================================================*/
10:
11: #include <stdio.h>
12: #include <string.h>
13: #include “records.h”
14: #include “tyac.h”
15:
16: /*------------------*
17:  * Global Variables *
18:  *------------------*/
19:
20: extern FILE *idx_fp;      /* Main File ptr to data file */
21: extern FILE *db_fp;       /* Data file */
22: extern FILE *song_fp;     /* Points to Songs */
23: extern nbr_records;       /* Total # of rec for albums */
24: extern int start_rec;
25: extern int disp_rec;
26: extern int total_songs;   /* Total songs in the song file */
27:
28: extern ALBUM_REC albums;
29: extern ALBUM_REC alb_prev;
30: extern SONG_REC songs[7];
31:
32: /*--------------------*
33:  *     prototypes     *
34:  *--------------------*/
35:
36: #include “recofrec.h”
37: int add_album_rec(void);
38:
39: /*-------------------------------------------------------*
40:  *  add_album_rec();                                     *
41:  *  Returns:  0 - No Error                               *
42:  *            >0 Error - see defines at top of file      *
43:  *-------------------------------------------------------*/
44:
45: int add_album_rec()
46: {
47:  int result;
48:  ALBUM_INDEX temp, newrec;
49:  int rv = NO_ERROR;
```

```
50:   int found;
51:   int srch_rec;
52:
53:   nbr_records++;
54:   if (nbr_records == 1)    /* Is this first record */
55:   {
56:     temp.prev = 0;
57:     temp.next = 0;
58:     temp.data = 1;         /* First rec in data file */
59:     temp.song = total_songs+1;
60:     temp.del_flag = FALSE;
61:     temp.nbr_songs = calc_nbr_songs();
62:     strcpy(temp.title, albums.title);
63:
64:     /* Write Index record */
65:     rv = put_rec(nbr_records, idx_fp, sizeof(temp),
66:                  sizeof(int)*2, (char *) &temp);
67:     if (rv == NO_ERROR)     /* No Error from prior */
68:     {
69:       /* Store Album data */
70:       rv = put_rec(nbr_records, db_fp, sizeof(albums),
71:                    0, (char *) &albums);
72:
73:       /* Store songs */
74:       if (rv == NO_ERROR)
75:         rv = add_new_songs();
76:     }
77:
78:     /* Update Alpha starting pointer */
79:     if (rv == NO_ERROR)
80:     {
81:       start_rec = nbr_records; /* Update global starting point */
82:       rv = update_header();
83:       fflush(song_fp);
84:     }
85:
86:   }
87:   /* Need to search for appropriate place to hold rec */
88:   else
89:   {
90:     found = FALSE;
91:     srch_rec = start_rec;
92:     while (!found)
93:     {
94:       rv = get_rec(srch_rec, idx_fp, sizeof(temp),
95:                    sizeof(int)*2, (char *)&temp);
96:       /* Proceed only if no errors */
97:       if (rv == NO_ERROR)
98:       {
99:         /* Compare two keys - ignoring CASE of keys */
```

continues

Listing 18.4. continued

```
100:        result = stricmp(albums.title, temp.title);
101:        if (result < 0)   /* New key is < this rec key */
102:        {
103:          /* Found place to put it - store info */
104:          found = TRUE;
105:          /* See if this new rec is < start rec  */
106:          /* If so - need to adjust starting ptr */
107:          if (srch_rec == start_rec)
108:             start_rec = nbr_records;
109:
110:          /* First build new Index rec & store new rec */
111:          newrec.prev = temp.prev;  /* Previous rec    */
112:          newrec.next = srch_rec;   /* Point to rec just read */
113:          newrec.data = nbr_records;  /* Pt to data */
114:          newrec.song = total_songs+1;
115:          newrec.del_flag = FALSE;
116:          newrec.nbr_songs = calc_nbr_songs();
117:          strcpy(newrec.title, albums.title);
118:
119:          rv = put_rec(nbr_records, idx_fp, sizeof(newrec),
120:                       sizeof(int)*2, (char *)&newrec);
121:          if (rv == NO_ERROR)
122:          {
123:            /* Update previous rec */
124:            temp.prev = nbr_records;
125:            rv = put_rec(srch_rec, idx_fp, sizeof(temp),
126:                         sizeof(int)*2, (char *)&temp);
127:
128:            /* Now write data - only if no errors */
129:            if (rv == NO_ERROR)
130:            {
131:              /* Now write Album data */
132:              rv = put_rec(nbr_records, db_fp, sizeof(albums),
133:                           0, (char *)&albums);
134:
135:              /* Now write Songs */
136:              if (rv == NO_ERROR)
137:                 rv = add_new_songs();
138:            }
139:
140:            /* Now check on updating Next pointer */
141:            /* Is there a ptr pointing to this new rec ?*/
142:            if (rv == NO_ERROR)
143:            {
144:              if (newrec.prev !=0 )
145:              {
146:                rv = get_rec(newrec.prev, idx_fp, sizeof(temp),
147:                             sizeof(int)*2, (char *)&temp);
148:                if ( rv == NO_ERROR)
```

```
149:                {
150:                  temp.next = nbr_records;
151:                  rv = put_rec(newrec.prev, idx_fp, sizeof(temp),
152:                               sizeof(int)*2, (char *)&temp);
153:                }
154:              }
155:            }
156:
157:          }
158:        }
159:        else    /* new rec >= alpha, adjust ptr */
160:        {
161:          if (temp.next == 0) /* End of chain - add to end */
162:          {
163:            found = TRUE;
164:
165:            /* Build Index record */
166:            /* Point backwards to prev rec */
167:            newrec.prev = srch_rec;
168:            newrec.next = 0;        /* There is no next rec */
169:            newrec.data = nbr_records;
170:            newrec.song = total_songs+1;
171:            newrec.del_flag = FALSE;
172:            newrec.nbr_songs = calc_nbr_songs();
173:            strcpy(newrec.title, albums.title);
174:
175:            rv = put_rec(nbr_records, idx_fp, sizeof(newrec),
176:                         sizeof(int)*2, (char *)&newrec);
177:            if (rv == NO_ERROR)
178:            {
179:              /* Update previous rec */
180:              temp.next = nbr_records;
181:              rv = put_rec(srch_rec, idx_fp, sizeof(temp),
182:                           sizeof(int)*2, (char *)&temp);
183:              if (rv == NO_ERROR)
184:              {
185:                /* Now write data */
186:                rv = put_rec(nbr_records, db_fp, sizeof(albums),
187:                             0, (char *)&albums);
188:                if (rv == NO_ERROR)
189:                {
190:                  /* Store songs */
191:                  rv = add_new_songs();
192:                }
193:              }
194:            }
195:          }
196:          else  /* Not at end - get next rec ptr */
197:             srch_rec = temp.next;
198:        }
```

continues

Listing 18.4. continued

```
199:       }
200:       else
201:         found = TRUE;  /* Exit because of error */
202:     }   /* End of While */
203:
204:     /* Update starting alpha ptr in hdr */
205:     if (rv == NO_ERROR)
206:     {
207:       rv = update_header();
208:     }
209:
210:     /* Makes sure file gets updated */
211:     fflush(idx_fp);
212:     fflush(db_fp);
213:     fflush(song_fp);
214:
215:   }  /* End else */
216:
217:   return(rv);
218: }
```

Analysis This function is very similar to the `add_medium_rec()` function presented on Day 17. The main difference is that after adding to the album data file and the album index file, you must also add the individual songs to the song file. You were already presented with a function that does the adding, `add_new_songs()`.

There is a second difference that needs to be covered. This is the use of a delete flag. Rather than actually deleting records from the files, a delete flag is used. A delete flag is used to state whether a record is active or not. Instead of physically removing a record from the database, it is simply removed from the indexes. By removing the record from the indexes, you cause it to be inaccessible.

There are reasons for using a delete flag. It is used in the musical instrument files to make the logic easier. You don't have to move around the records in the same way that you did for the medium screen. Additionally, if you use a delete flag, it gives you the ability to undelete an item. The functionality to undelete isn't presented in this book. Undeleting is just a matter of switching the delete flag from TRUE back to FALSE. You'll see in the delete code that deleting is just a matter of setting all the delete flags to TRUE.

Deleting a Record: The *del_alb_rec()* Function

The `del_alb_rec()` function is used to delete musical items. Listing 18.5 presents the DELAREC.C source file, which contains this function. Deleting records in the musical items file is done differently than the medium code and group files.

Listing 18.5. DELAREC.C. Deleting a musical item.

```
1:  /*=========================================================
2:   * Filename: DELAREC.C
3:   *
4:   * Author:   Bradley L. Jones
5:   *           Gregory L. Guntle
6:   *
7:   * Purpose: Deletes a record from the DB
8:   *          For albums - this simply marks the record
9:   *          as being deleted.  It also marks the data record
10:  *          as being deleted.  Then it spins through all the
11:  *          songs marking them as deleted.  It leaves
12:  *          all pointers alone.  This allows the reader the
13:  *          ability to write an undelete function.
14:  *
15:  *=========================================================*/
16:
17:  #include <stdio.h>
18:
19:  #include “records.h”
20:  #include “tyac.h”
21:
22:  /*--------------------*
23:   *  Global Variables  *
24:   *--------------------*/
25:
26:  extern FILE *idx_fp;      /* Main File ptr to data file */
27:  extern FILE *db_fp;       /* Data file */
28:  extern FILE *song_fp;     /* Pointer for songs */
29:  extern nbr_records;       /* Total # of rec for mediums */
30:  extern int start_rec;
31:  extern int disp_rec;
32:  extern int total_songs;
33:
34:  /*------------------*
35:   * Defined constants*
36:   *------------------*/
37:
38:  #define ALBUMS_DBF   “ALBUMS.DBF”
39:  #define ALBUMS_IDX   “ALBUMS.IDX”
```

continues

Listing 18.5. continued

```
40:  #define SONGS_DBF    "SONGS.DBF"
41:  #define HELP_DBF     "ALBUMS.HLP"
42:
43:  /*------------------------*
44:   * structure declarations *
45:   *------------------------*/
46:  extern ALBUM_REC albums;
47:  extern ALBUM_REC alb_prev;
48:  extern SONG_REC songs[7];
49:
50:  extern int song_pg;     /* Tracks pg of songs being displayed */
51:  extern int max_song_pg; /* Max number of pages allocated to   */
52:                          /* hold the songs                     */
53:  extern char *song_ptr;  /* Ptr to dynamic mem where songs are */
54:
55:  #define ONE_SONG         sizeof(songs[0])
56:  #define SEVEN_SONGS      sizeof(songs)
57:
58:
59:  int del_alb_rec()
60:  {
61:    int rv = NO_ERROR;
62:    ALBUM_INDEX temp, delrec;
63:    ALBUM_REC hold_data;
64:    SONG_REC hold_song;
65:    int chg;
66:    int srch_rec;
67:    long fpos;
68:    int true_rec;  /* Includes deleted records as well */
69:    int rec;
70:
71:    /* Are we trying to delete a blank record */
72:    if (disp_rec != 0)  /* No - then proceed */
73:    {
74:      /* Get the index info for this rec */
75:      rv = get_rec(disp_rec, idx_fp, sizeof(delrec),
76:             sizeof(int)*2, (char *)&delrec);
77:      if (rv == NO_ERROR)
78:      {
79:        /* Are there any pointers in this rec
80:           If both pointers are 0 - then this must
81:           be the only record in the DB           */
82:        if (delrec.prev == 0 && delrec.next == 0)
83:        {
84:          nbr_records = 0;
85:          disp_rec = 0;
86:          start_rec = 1;
87:          total_songs = 0;
88:        }
89:        else
```

```
90:         {
91:           chg = FALSE;
92:           srch_rec = 1;   /* Start at first */
93:           fseek(idx_fp, 0, SEEK_END); /* Go to end of Index file */
94:           fpos = ftell(idx_fp);       /* Get nbr of bytes in file*/
95:           fpos -= sizeof(int)*2;      /* Adjust for header info  */
96:           true_rec = fpos/(sizeof(temp)); /* Gets number of recs */
97:
98:           /* Are we deleting the starting alpha record ? */
99:           if (disp_rec == start_rec)
100:             start_rec = delrec.next;     /* Reset pointer */
101:
102:          /* Go until all the ptrs have been adjusted */
103:          while ( (srch_rec <= true_rec) && (rv == NO_ERROR) )
104:          {
105:            rv = get_rec(srch_rec, idx_fp, sizeof(temp),
106:                  sizeof(int)*2, (char *)&temp);
107:            if (rv == NO_ERROR)
108:            {
109:              /* Only do it - rec read has not been deleted */
110:              if (!temp.del_flag)
111:              {
112:                /* There is no need to update pointers for
113:                   the record being deleted - so only do the
114:                   logic below if they don't match          */
115:                if (srch_rec != disp_rec)
116:                {
117:
118:                  /* Get record */
119:                  rv = get_rec(srch_rec, idx_fp, sizeof(temp),
120:                        sizeof(int)*2, (char *)&temp);
121:                  if (rv == NO_ERROR)
122:                  {
123:                    /* Does this rec prev pointer need to
124:                       be adjusted */
125:                    if (temp.prev == disp_rec)
126:                    {
127:                      chg = TRUE;
128:                      temp.prev = delrec.prev;
129:                    }
130:
131:                    if (temp.next == disp_rec)
132:                    {
133:                      chg = TRUE;
134:                      temp.next = delrec.next;
135:                    }
136:
137:                    /* Since not moving last record up -
138:                       the code to do this is not necessary
139:                       for the previous and next pointers   */
```

continues

Listing 18.5. continued

```
140:
141:                    /* Not moving data - so don't adjust
142:                       pointer to data - will leave so
143:                       reader can write an undelete routine */
144:
145:                    /* Only update rec if change has been made */
146:                    if (chg)
147:                    {
148:                      /* Rewrite index back to file in same pos */
149:                      rv = put_rec(srch_rec, idx_fp, sizeof(temp),
150:                             sizeof(int)*2, (char *)&temp);
151:                      chg = FALSE;
152:                    }
153:
154:                    if (rv == NO_ERROR)
155:                    {
156:                      /* Go to next record */
157:                      srch_rec++;
158:                    }
159:                  }   /* If NO_ERROR  */
160:                }     /* If SRCH != DISP_REC */
161:                else
162:                      srch_rec++;
163:              }       /* checking for DEL record */
164:              else
165:                srch_rec++;
166:            }         /* NO ERROR  */
167:          }           /* WHILE */
168:
169:          /* Clean out NEXT/PREV Ptrs in the deleted rec
170:             Index - but leave rec there  and leave data ptrs
171:             alone - they can be used for purging the old recs */
172:          if (rv == NO_ERROR)
173:          {
174:            delrec.prev = 0;
175:            delrec.next = 0;
176:            delrec.nbr_songs = 0;
177:            delrec.del_flag = TRUE;
178:            /* Overwrite data to delete w/Last one */
179:            rv = put_rec(disp_rec, idx_fp, sizeof(delrec),
180:                 sizeof(int)*2, (char *)&delrec);
181:          }
182:
183:          /* DATA    - ALBUMS */
184:          /* Mark the del_flag component of the record   */
185:          /* leave data alone.                           */
```

```
186:         rv = get_rec(delrec.data, db_fp, sizeof(hold_data),
187:                    0, (char *)&hold_data);
188:         if (rv == NO_ERROR)
189:         {
190:           /* Mark flag and rewrite data */
191:           hold_data.del_flag = TRUE;
192:           rv = put_rec(delrec.data, db_fp, sizeof(hold_data),
193:                    0, (char *)&hold_data);
194:         }
195:
196:         /* DATA    - SONGS */
197:         /* Spin through all songs - mark them as deleted */
198:         rec = delrec.song;    /* Starting point */
199:         while (rec !=0 && rv == NO_ERROR)
200:         {
201:           rv = get_rec(rec, song_fp, sizeof(hold_song),
202:                     sizeof(int), (char *)&hold_song);
203:           if (rv == NO_ERROR)
204:           {
205:             hold_song.del_flag = TRUE;
206:             rv = put_rec(rec, song_fp, sizeof(hold_song),
207:                      sizeof(int), (char *)&hold_song);
208:             if (rv == NO_ERROR)
209:               rec = hold_song.next;
210:           }
211:         }
212:       }                  /* end of ELSE  */
213:
214:       if (rv == NO_ERROR)
215:       {
216:         rv = update_header();  /* Update header info */
217:         if (rv == NO_ERROR)
218:            rv = put_rec(0, song_fp, sizeof(int), 0,
219:                 (char *)&total_songs);
220:       }
221:
222:       fflush(idx_fp);   /* Make sure all items written to disk*/
223:       fflush(db_fp);
224:       fflush(song_fp);  /* Update song DB as well */
225:
226:     }    /* If NO_ERROR from 1st time */
227:   }      /* If disp_rec != 0  */
228:   else
229:     boop();  /* Let them know - can't delete blank rec */
230:
231:   return(rv);
232: }
```

Analysis As stated earlier, the deleting of records in the `del_alb_rec()` operates differently than the `del_med_rec()` function. Rather than physically removing the records from the databases, the `del_flag` field in each structure is set to `TRUE`. This will give you the ability to undelete. As you can see by the listing, there are several comments that point out the differences between this listing and the `del_med_rec()` listing that was detailed on Day 17.

Processing a Record (Next/Previous)

The last function needed before you'll be able to compile the musical items code without external errors is the `proc_alb_rec()` function. This function is presented in Listing 18.6, which contains the PROCAREC.C source file.

Type **Listing 18.6. PROCAREC.C. Processing the next and previous records.**

```
1:  /*============================================================
2:   * Filename: PROCAREC.C
3:   *
4:   * Author:   Bradley L. Jones
5:   *           Gregory L. Guntle
6:   *
7:   * Purpose: Process the requests for getting next/prev
8:   *          records from the ALBUMS DBF
9:   *
10:  *============================================================*/
11:
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <string.h>
15: #include "records.h"
16: #include "tyac.h"
17:
18: #include "recofrec.h"
19:
20: /*------------------*
21:  * Global Variables *
22:  *------------------*/
23:
24: extern FILE *idx_fp;      /* Main File ptr to data file */
25: extern FILE *db_fp;       /* Data file */
26: extern FILE *song_fp;     /* Pointer for songs */
27: extern nbr_records;       /* Total # of rec for albums */
28: extern int start_rec;
29: extern int disp_rec;
30: extern int total_songs;
31:
```

```
32:  extern ALBUM_REC albums;
33:  extern ALBUM_REC alb_prev;
34:  extern SONG_REC songs[7];
35:
36:  extern int song_pg;     /* Tracks page of songs displayed */
37:  extern int max_song_pg; /* Max number of pages allocated  */
38:                         /*  to allocated to hold the songs */
39:  extern char *song_ptr;  /* Ptr to dynamic mem where songs are */
40:
41:  #define ONE_SONG         sizeof(songs[0])
42:  #define SEVEN_SONGS      sizeof(songs)
43:
44:
45:  int proc_alb_rec(int);
46:  int get_alb_info(void);
47:
48:  /*-------------------------------------------------------*
49:   *   proc_alb_rec                                        *
50:   *-------------------------------------------------------*/
51:
52:  int proc_alb_rec(int direct)
53:  {
54:    ALBUM_INDEX temp;
55:    int rv = NO_ERROR;
56:
57:    /* Only do - if there are records in the file */
58:    if (nbr_records != 0)
59:    {
60:      /* Do we just need to display the very first rec */
61:      if (disp_rec == 0)
62:      {
63:         disp_rec = start_rec;
64:         rv = get_alb_info();
65:      }
66:      else
67:      {
68:        /* Get Index ptrs for record on screen */
69:        rv = get_rec(disp_rec, idx_fp, sizeof(temp),
70:               sizeof(int)*2, (char *)&temp);
71:        if (rv == NO_ERROR)
72:        {
73:          if( direct == NEXT_REC )
74:          {
75:            if (temp.next == 0)     /* There is no other rec */
76:            {
77:              boop();               /* No more records */
78:            }
79:            else
80:            {
81:              disp_rec = temp.next;
```

continues

Listing 18.6. continued

```
82:              rv = get_alb_info();
83:            }
84:          }
85:          else  /* Need to go backwards */
86:          {
87:            if (temp.prev == 0)     /* There is no other rec */
88:            {
89:               boop();               /* No more records */
90:            }
91:            else
92:            {
93:              disp_rec = temp.prev;
94:              rv = get_alb_info();
95:            }
96:          }
97:        }
98:      }
99:    }
100:   else
101:   {
102:     boop();
103:   }
104:
105:   return(rv);
106: }
107:
108: /* ----------------------------------------------  *
109:  *  get_alb_info()                                 *
110:  *                                                 *
111:  *  Handles getting the Index rec, getting the     *
112:  *  data pointer from the index and then getting   *
113:  *  the appropriate data rec.  It also gets the    *
114:  *  songs from the SONGS.DBF file.                 *
115:  *                                                 *
116:  * ----------------------------------------------  */
117:
118: int get_alb_info()
119: {
120:   ALBUM_INDEX temp;
121:   int rv;
122:   int recnbr;
123:   int pgs, sngs;
124:
125:   /* Get Index record for this Request */
126:   rv = get_rec(disp_rec, idx_fp, sizeof(temp),
127:                sizeof(int)*2, (char *)&temp);
128:   if (rv == NO_ERROR)
129:   {
130:
131:     /* Now get the actual data record to display */
```

```
132:     rv = get_rec(temp.data, db_fp, sizeof(albums),
133:                  0, (char *)&albums);
134:     if (rv == NO_ERROR)
135:     {
136:       /* Setup previous record */
137:       memcpy(&alb_prev, &albums, sizeof(albums));
138:
139:       /* Clear out any items from memory location */
140:       reset_memory();
141:
142:       /* Adjust new max song pages to meet the needs */
143:       /* of the title */
144:       max_song_pg = temp.nbr_songs/7;
145:
146:       /* Does it need more than 1 page ?  */
147:       if (max_song_pg > 1)
148:       {
149:          song_ptr = (char *)realloc(song_ptr,
150:                      max_song_pg*SEVEN_SONGS);
151:          if (song_ptr == NULL)  /* Not able to get mem */
152:          {
153:            display_msg_box("Memory not available!",
154:                      ct.err_fcol, ct.err_bcol );
155:            exit(1);
156:          }
157:       }
158:
159:       /* Get all the songs */
160:       recnbr = temp.song;  /* Starting point */
161:
162:       /* Loop through all the pages */
163:       /* Start at first page of songs - 1 page = 7 songs*/
164:       pgs = 0;
165:       while ( pgs < max_song_pg && rv == NO_ERROR )
166:       {
167:         sngs = 0;
168:         /* Now loop through individual songs */
169:         while ( sngs < 7 && rv == NO_ERROR )
170:         {
171:           /* Get the record and place into memory */
172:           rv = get_rec(recnbr, song_fp, ONE_SONG,
173:                sizeof(int),(char *)&songs[sngs]);
174:           if ( rv == NO_ERROR )
175:           {
176:             recnbr = songs[sngs].next;
177:             sngs++;
178:           }
179:         }
180:
181:         /* Move the information into memory */
```

continues

Listing 18.6. continued

```
182:          if (rv == NO_ERROR)
183:          {
184:            memcpy(song_ptr+(pgs*SEVEN_SONGS),
185:                   &songs, SEVEN_SONGS);
186:          }
187:          pgs++;   /* Get Next page worth of songs */
188:        }
189:      }
190:    }
191:    return(rv);
192: }
```

Analysis The `proc_alb_rec()` function is called with a value of either `NEXT_REC` or `PREVIOUS_REC`, which is stored in the direct parameter. As with the medium code function, this determines in which direction the reading should occur. Most of this function follows what was presented in `proc_med_rec()` on Day 17.

The one difference that deserves mentioning is in lines 164 to 188 in the `get_alb_info()` function. These lines operate similarly to the code seen earlier today. Line 165 uses a `while` loop to read a page of songs. In each page, seven songs are read using the `while` loop that begins in line 169. After each page of records is read into the songs structure, it is copied into the dynamic memory area. This is done in line 184. The rest of the code presented in this listing doesn't present anything you haven't already seen earlier today.

The Musical Items Screen's Action Bar

To help complete the Musical Item code, Listing 18.7 is included. This contains the action bar routines for the Musical Items screen. With the addition of this listing, your Musical Items screen should be complete.

Type **Listing 18.7. ALBMABAR.C. The Musical Items screen action bar.**

```
1:   /*============================================================
2:    * Filename: albmabar.c
3:    *           RECORD OF RECORDS - Version 1.0
4:    *
5:    * Author:   Bradley L. Jones
```

```
6:    *
7:    * Purpose:  Action bar for musical items screen.  This will
8:    *           contain multiple menus. The functions called
9:    *           by the menu selections may not be available
10:   *           until later days.
11:   *
12:   * Return:   Return value is either key used to exit a menu
13:   *           with the exception of F10 which is returned
14:   *           as Enter to redisplay main menu.  In each menu
15:   *           the left and right keys will exit and move
16:   *           control to the right or left menu.
17:   *=========================================================*/
18:
19:  #include <string.h>
20:  #include <stdio.h>
21:  #include <stdlib.h>
22:  #include "tyac.h"
23:  #include "records.h"
24:
25:
26:  /*--------------------*
27:   *     prototypes     *
28:   *--------------------*/
29:
30:  #include "recofrec.h"
31:
32:  int do_albums_menu1( void );
33:  int do_albums_menu2( void );
34:  int do_albums_menu3( void );
35:  int do_albums_menu4( void );
36:
37:  static void do_something( char * );
38:
39:  /*--------------------*
40:   *  Global Variables  *
41:   *--------------------*/
42:
43:  extern FILE *idx_fp;      /* Main File ptr to data file */
44:  extern FILE *db_fp;       /* Data file */
45:  extern FILE *song_fp;     /* Pointer for songs */
46:  extern nbr_records;       /* Total # of rec for mediums */
47:  extern int start_rec;
48:  extern int disp_rec;
49:  extern int total_songs;
50:
51:  /*------------------*
52:   * Defined constants*
53:   *------------------*/
54:
55:  #define ALBUMS_DBF   "ALBUMS.DBF"
```

continues

Listing 18.7. continued

```
56:  #define ALBUMS_IDX   "ALBUMS.IDX"
57:  #define SONGS_DBF    "SONGS.DBF"
58:  #define HELP_DBF     "ALBUMS.HLP"
59:
60:  /*------------------------*
61:   * structure declarations *
62:   *------------------------*/
63:  extern ALBUM_REC albums;
64:  extern ALBUM_REC alb_prev;
65:  extern SONG_REC songs[7];
66:
67:  extern int song_pg;       /* Tracks page of songs being displayed */
68:  extern int max_song_pg;   /* Max number of pages allocated to
69:                               hold the songs */
70:  extern char *song_ptr;    /* Ptr to dynamic mem where songs are
                                  kept */
71:
72:  #define ONE_SONG         sizeof(songs[0])
73:  #define SEVEN_SONGS      sizeof(songs)
74:
75:  /*----------------------------*
76:   *  albums screen action bar  *
77:   *----------------------------*/
78:
79:  int do_albums_actionbar( void )
80:  {
81:    int  menu  = 1;
82:    int  cont  = TRUE;
83:    int  rv    = 0;
84:    char *abar_text ={" File   Edit   Search   Help "};
85:
86:
87:    while( cont == TRUE )
88:    {
89:       write_string(abar_text, ct.abar_fcol, ct.abar_bcol, 1, 2);
90:
91:       switch( menu )
92:       {
93:
94:          case 1:  /* file menu */
95:                   write_string( " File ", ct.menu_high_fcol,
96:                        ct.menu_high_bcol, 1, 2);
97:
98:                   rv = do_albums_menu1();
99:                   break;
100:
101:         case 2:  /* edit menu */
102:                  write_string( " Edit ", ct.menu_high_fcol,
```

```
103:                        ct.menu_high_bcol, 1, 9);
104:
105:                  rv = do_albums_menu2();
106:                  break;
107:
108:         case 3:  /* search menu */
109:                  write_string( " Search ", ct.menu_high_fcol,
110:                        ct.menu_high_bcol, 1, 16);
111:
112:                  rv = do_albums_menu3();
113:                  break;
114:
115:         case 4:  /* Help menu */
116:                  write_string( " Help ", ct.menu_high_fcol,
117:                        ct.menu_high_bcol, 1, 25);
118:
119:                  rv = do_albums_menu4();
120:                  break;
121:
122:         default: /* error */
123:                  cont = FALSE;
124:                  break;
125:      }
126:
127:      switch( rv )
128:      {
129:         case LT_ARROW:  menu--;
130:                         if( menu < 1 )
131:                             menu = 4;
132:                         break;
133:
134:         case RT_ARROW:  menu++;
135:                         if( menu > 4 )
136:                             menu = 1;
137:                         break;
138:
139:         default:        cont = FALSE;
140:                         break;
141:      }
142:    }
143:    write_string(abar_text, ct.abar_fcol, ct.abar_bcol, 1, 2);
144:    cursor_on();
145:    return(rv);
146: }
147:
148:
149: /*--------------------*
150:  *  do_menu 1  (File) *
151:  *--------------------*/
152:
```

continues

Listing 18.7. continued

```
153: int do_albums_menu1( void )
154: {
155:   int   rv       = 0;
156:   int   menu_sel = 0;
157:   char *saved_screen = NULL;
158:
159:   char *file_menu[2] = { " Exit  <F3> ", "1Ee" };
160:   char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
161:
162:   saved_screen = save_screen_area( 0, 10, 0, 40 );
163:
164:   rv = display_menu( 3, 4, SINGLE_BOX, file_menu, 2,
165:                      exit_keys, &menu_sel, LR_ARROW, SHADOW);
166:
167:   restore_screen_area( saved_screen );
168:
169:   switch( rv )
170:   {
171:      case ENTER_KEY: /* accept selection */
172:      case CR:
173:                      rv = F3;
174:                      break;
175:
176:      case F3:       /* exiting */
177:      case ESC_KEY:
178:      case LT_ARROW: /* arrow keys */
179:      case RT_ARROW:
180:                      break;
181:
182:      case F10:      /* exit action bar */
183:                      rv = ENTER_KEY;
184:                      break;
185:
186:      default:        boop();
187:                      break;
188:   }
189:   return(rv);
190: }
191:
192: /*--------------------*
193:  *  do_menu 2  (Edit) *
194:  *--------------------*/
195:
196: int do_albums_menu2( void )
197: {
198:   int   rv       = 0;
199:   int   menu_sel = 0;
200:   char *saved_screen = NULL;
201:
202:   char *edit_menu[8] = {
```

```
203:           " New          ", "1Nn",
204:           " Add     <F4> ", "2Aa",
205:           " Change  <F5> ", "3Cc",
206:           " Delete  <F6> ", "4Dd" };
207:
208:   char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
209:
210:   saved_screen = save_screen_area( 1, 10, 8, 40 );
211:
212:   rv = display_menu( 3, 11, SINGLE_BOX, edit_menu, 8,
213:                      exit_keys, &menu_sel, LR_ARROW, SHADOW);
214:
215:   restore_screen_area( saved_screen );
216:
217:   switch( rv )
218:   {
219:      case ENTER_KEY: /* accept selection */
220:      case CR:
221:               switch( menu_sel )
222:               {
223:                 case 1: /* Clear the screen */
224:                         disp_rec = 0;
225:                         reset_memory();
226:                         clear_albums_fields();
227:                         draw_albums_screen();
228:                         break;
229:
230:                 case 2:  /* Add a record */
231:                          /* Save songs back into memory location */
232:                          memcpy(song_ptr+((song_pg-1)*SEVEN_SONGS),
233:                                 &songs, SEVEN_SONGS);
234:                          rv = add_adata();
235:                          if ( rv == NO_ERROR )
236:                          {
237:                            disp_rec = 0;
238:                            reset_memory();
239:                            clear_albums_fields();
240:                            draw_albums_screen();
241:                          }
242:                          else  /* Only do next part if File I/O  */
243:                          if ( rv > NO_ERROR )
244:                          {
245:                            display_msg_box("Fatal Error adding...",
246:                                 ct.err_fcol, ct.err_bcol );
247:                            exit(1);
248:                          }
249:                          break;
250:
251:                 case 3:  /* Update the current record */
252:                          /* Save songs back into memory location */
```

continues

Listing 18.7. continued

```
253:                          memcpy(song_ptr+((song_pg-1)*SEVEN_SONGS),
254:                                &songs, SEVEN_SONGS);
255:                          rv = add_adata();
256:                          if ( rv == NO_ERROR )
257:                          {
258:                            disp_rec = 0;
259:                            reset_memory();
260:                            clear_albums_fields();
261:                            draw_albums_screen();
262:                          }
263:                          else  /* Only do next part if File I/O  */
264:                          if ( rv > NO_ERROR )
265:                          {
266:                            display_msg_box("Fatal Error updating...",
267:                                 ct.err_fcol, ct.err_bcol );
268:                            exit(1);
269:                          }
270:                          break;
271:
272:                case 4:   /* Deleting the current record */
273:                          if( (yes_no_box( "Delete record ?",
274:                                ct.db_fcol, ct.db_bcol )) == 'Y' )
275:                          {
276:                            rv = del_alb_rec();
277:                            if (rv == NO_ERROR)
278:                            {
279:                              disp_rec = 0;
280:                              clear_albums_fields();
281:                              draw_albums_screen();
282:                            }
283:                            else
284:                            {
285:                              display_msg_box("Fatal Error
                                                  deleting...",
286:                                 ct.err_fcol, ct.err_bcol );
287:                              exit(1);
288:                            }
289:                          }
290:                          break;
291:
292:                default:  /* continue looping */
293:                          boop();
294:                          break;
295:              }
296:
297:              rv = ENTER_KEY;
298:              break;
299:
300:       case F3:       /* exiting */
```

```
301:       case ESC_KEY:
302:       case LT_ARROW: /* arrow keys */
303:       case RT_ARROW:
304:                      break;
305:
306:       case F10:      /* action bar */
307:                      rv = ENTER_KEY;
308:                      break;
309:
310:       default:       boop();
311:                      break;
312:   }
313:   return(rv);
314: }
315:
316: /*----------------------*
317:  *  do menu 3  (Search) *
318:  *----------------------*/
319:
320: int do_albums_menu3( void )
321: {
322:   int   rv       = 0;
323:   int   menu_sel = 0;
324:   char *saved_screen = NULL;
325:
326:   char *search_menu[6] = {
327:           " Find...         ", "1Ff",
328:           " Next      <F7> ", "2Nn",
329:           " Previous  <F8> ", "3Pp" };
330:
331:   char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
332:
333:   saved_screen = save_screen_area( 1, 10, 0, 60 );
334:
335:   rv = display_menu( 3, 18, SINGLE_BOX, search_menu, 6,
336:                      exit_keys, &menu_sel, LR_ARROW, SHADOW);
337:
338:   restore_screen_area( saved_screen );
339:
340:   switch( rv )
341:   {
342:      case ENTER_KEY: /* accept selection */
343:      case CR:
344:               switch( menu_sel )
345:               {
346:                 case 1:  /* Do find dialog */
347:                          rv = search_alb_rec(albums.title);
348:
349:                          if( rv == NO_ERROR )
350:                          {
```

continues

Listing 18.7. continued

```
351:                              /* record found */
352:                              draw_albums_screen();
353:                              display_msg_box("Record Found!",
354:                                        ct.db_fcol, ct.db_bcol);
355:                           }
356:                           else
357:                           if( rv < 0 )
358:                           {
359:                              display_msg_box("Record Not Found!",
360:                                        ct.err_fcol, ct.err_bcol);
361:                           }
362:                           else
363:                           {
364:                             display_msg_box(
365:                                   "Fatal Error processing data...",
366:                                   ct.err_fcol, ct.err_bcol );
367:                             exit(1);
368:                           }
369:                           break;
370:
371:                  case 2:  /* Next Record */
372:                        rv = proc_alb_rec(NEXT_REC);
373:                        if ( rv == NO_ERROR )
374:                           draw_albums_screen();
375:                        else
376:                        {
377:                          display_msg_box("Fatal Error processing
                                        data...",
378:                               ct.err_fcol, ct.err_bcol );
379:                          exit(1);
380:                        }
381:                        break;
382:
383:                  case 3:  /* Previous Record */
384:                        rv = proc_alb_rec(PREVIOUS_REC);
385:                        if ( rv == NO_ERROR )
386:                        {
387:                           draw_albums_screen();
388:                        }
389:                        else
390:                        {
391:                          display_msg_box("Fatal Error processing
                                        data...",
392:                               ct.err_fcol, ct.err_bcol );
393:                          exit(1);
394:                        }
395:                        break;
396:
397:                  default: /* shouldn't happen */
398:                           boop();
```

```
399:                          break;
400:               }
401:               rv = ENTER_KEY;
402:                    break;
403:
404:       case F3:       /* exiting */
405:       case ESC_KEY:
406:       case LT_ARROW: /* arrow keys */
407:       case RT_ARROW:
408:                      break;
409:
410:       case F10:      /* action bar */
411:                      rv = ENTER_KEY;
412:                      break;
413:
414:       default:       boop();
415:                      break;
416:   }
417:   return(rv);
418: }
419:
420: /*--------------------*
421:  *  do menu 4  (Help) *
422:  *--------------------*/
423:
424: int do_albums_menu4( void )
425: {
426:   int   rv       = 0;
427:   int   menu_sel = 0;
428:   char *saved_screen = NULL;
429:
430:   char *help_menu[4] = {
431:           " Help  <F2> ", "1Hh",
432:           " About      ", "2Ee" };
433:
434:   char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};
435:
436:   saved_screen = save_screen_area( 1, 10, 0, 60 );
437:
438:   rv = display_menu( 3, 27, SINGLE_BOX, help_menu, 4,
439:                      exit_keys, &menu_sel, LR_ARROW, SHADOW);
440:
441:   restore_screen_area( saved_screen );
442:
443:   switch( rv )
444:   {
445:      case ENTER_KEY: /* accept selection */
446:      case CR:
447:                      switch( menu_sel )
448:                      {
```

continues

Listing 18.7. continued

```
449:                          case 1:  /* Extended Help */
450:                                   display_albums_help();
451:
452:                                   break;
453:
454:                          case 2:  /* About box */
455:                                   display_about_box();
456:
457:                                   break;
458:
459:                          default: /* continue looping */
460:                                   boop();
461:                                   break;
462:                       }
463:
464:                       break;
465:
466:       case F3:        /* exiting */
467:       case ESC_KEY:
468:       case LT_ARROW: /* arrow keys */
469:       case RT_ARROW:
470:                       break;
471:
472:       case F10:       /* action bar */
473:                       rv = ENTER_KEY;
474:                       break;
475:
476:       default:        boop();
477:                       break;
478:   }
479:   return(rv);
480: }
481:
482: /*=========================================================
483:  * Function: search_alb_rec()
484:  *
485:  * Author:   Bradley L. Jones
486:  *           Gregory L. Guntle
487:  *
488:  * Purpose: Searches for the key to locate
489:  *
490:  * Returns:  0 - No errors - key found
491:  *          <0 - Key not found
492:  *          >0 - File I/O Error
493:  *=========================================================*/
494:
495: int search_alb_rec(char *key)
496: {
497:   int rv = NO_ERROR;
498:   int done = FALSE;
```

```
499:   int srch_rec;
500:   int result;
501:
502:   ALBUM_INDEX temp;
503:
504:   /* Start at top of chain */
505:   /* Stop when either
506:      1 - The key passed matches   or
507:      2 - The key passed is < rec read
508:      The latter indicates the key is not in the file  */
509:
510:   srch_rec = start_rec;
511:   while (!done)
512:   {
513:     /* Get record */
514:     rv = get_rec(srch_rec, idx_fp, sizeof(temp),
515:           sizeof(int)*2, (char *)&temp);
516:     if (rv == 0)
517:     {
518:       result = stricmp(key, temp.title);
519:       if (result == 0)   /* Found match */
520:       {
521:         /* Now get data */
522:         done = TRUE;
523:         rv = get_rec(temp.data, db_fp, sizeof(albums),
524:                0, (char *)&albums);
525:       }
526:       else
527:       if (result < 0)
528:       {
529:         /* Key < next rec in DBF - key doesn't exist */
530:         done = TRUE;
531:         rv = -1;              /* Key not found */
532:       }
533:       else
534:       {
535:         srch_rec = temp.next;
536:         if (srch_rec == 0)    /* At end of list ? */
537:         {
538:           done = TRUE;
539:           rv = -1;            /* Key not found */
540:         }
541:       }
542:     }
543:   }
544:
545:   return(rv);
546:
547: }
```

Summary

Today, you were presented with a multitude of code. The day started by completing the code necessary for the Medium Code screen. You were presented with the completed action bar routines. Once everything for the Medium Code screen was presented, the chapter moved into the Musical Items screen. The Musical Items screen offers a higher degree of complexity due to the variable number of songs that can be entered. Whereas most of the Musical Items screen was presented with little analysis, all of the major differences were detailed. This includes the page up and page down functions that are used to display multiple sets of song records.

Q&A

Q Is it better to use an undelete flag or to physically remove records?

A The answer to this question is dependent upon the application. If storage needs to be kept at a minimum, then you should physically remove data from the file. If there is a chance that the user will want to restore a deleted record, then you should use a delete flag.

Q What can be done if the musical items' song file gets too big?

A You can physically remove the records that are marked for deletion. With the functions you have created, it should be easy to write a program that reads each record and writes them to new files. Because only nondeleted records will be read, the new file won't include these. Once you have created all of the records in the album files, you can delete them. The temporary files that you created can then be renamed to the album file names.

Q Can records marked as deleted be reused when adding new records?

A If you don't plan to undelete, then records marked as deleted could be filled with new records. You could change the logic of the add function to read each record in the file in physical order. If the record read is marked as deleted, then the record being added could use the space. If the record isn't marked as deleted, then it would be skipped.

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 two file functions on the action bar are not accessible with a function key?
2. What do each of the functions from Question 1 do?
3. What should you always do before deleting a record?
4. What is a delete flag used for?
5. What do page up and page down do in the Medium Code screen?
6. What do page up and page down do in the Musical Items screen?
7. What is the most songs that can be stored in the Musical Items screen?

Exercises

1. Complete the *Record of Records!* entry and edit screens. You should now have everything you need to complete all three screens. Tomorrow, you'll add the reporting features.
2. Write a program that prints the information in the Musical Items screen's index file, ALBUMS.IDX. A file such as this can be used for debugging purposes.
3. **ON YOUR OWN:** Write a program that uses a file with a variable number of records such as the songs in the album screen. An example of such a file could include a contact database. You could store the addresses in a separate file and include a billing address, a shipping address, and a home address.