Help!

WEEK
3

At one time or another, everybody needs help. When building an application, there are several features that you can add to aid your users in their times of need. This aid is what today's information focuses on. Today you will learn:

- About adding help to your application.
- What the different types of help are.
- Which types of help are more important than others.
- How to add the different types of help to your applications.

Who Needs Help?

Everyone needs help at one time or another. In applications, users don't always understand what is being displayed or what they are expected to do. By adding help to your applications, you provide a means for the users to get aid.

There are specific types of information that should be provided in help. This help should focus on the application being used. In addition, help is not meant to teach a user how to use the application. Help is intended to provide assistance to the user.

What Are the Different Types of Help?

The types of assistance to be provided can vary. Each type of assistance, or form of help, has a specific intent. Following is a list of the different help types commonly found in applications:

- Startup help
- General help
  - Help for help
  - Extended help
  - Help for keys
  - Help index
- About... help
- Context-sensitive help
- Tutorials

Startup Help

Startup help is provided when an application is first started. It can be provided in one of two ways, either automatically or when the user requests it.

Startup help is most common in applications that require command line parameters. Consider the DOS TYPE command. If you enter `TYPE` at the command without the name of a file to be listed, you get the following response:

```
C>TYPE
Required parameter missing
```

This information is provided because you didn't start the program correctly. The information is intended to assist you in beginning the program correctly.

The TYPE command's help was provided automatically. Some programs provide startup help that has to be requested. This request is made when you start the program. To receive the aid, an additional parameter is passed. This parameter is generally a special character such as a dash (-) or a slash (/), followed by the letter H or the word help. An example of a program that uses this form of help is the shareware utility, PKZIP by PKWARE Inc. When running PKZIP, you can include a `-h` on the command line for help information. The following illustrates what is presented when you type `PKZIP -h` and press enter:

```
PKZIP (R)   FAST!   Create/Update Utility   Version 2.04g   02-01-93
Copr. 1989-1993 PKWARE Inc.  All Rights Reserved.  Shareware Version
PKZIP Reg. U.S. Pat. and Tm. Off.   Patent No. 5,051,745


PKZIP /h[1] for basic help   PKZIP /h[2|3|4] for other help screens.


Usage:  PKZIP [options] zipfile [@list] [files...]


       Simple Usage:   PKZIP zipfile file(s)...
                         |      |      |
Program ----------------         |      |
                                 |      |
New zipfile to create ----------        |
                                        |
File(s) you wish to compress ----------

The above usage is only a very basic example of PKZIP's capability.


Press 2 for more options (including spanning & formatting), press 3 for
advanced options, 4 for trouble shooting options, any other key to quit
help.
```

As you can see, the information provided by PKZIP is helpful in showing you how to use the program. This information aids you in the operation of PKZIP. If you don't know this information, you will be unable to operate the program.

Note: Startup information is provided to help the user start an application.

If your program requires command line parameters, you should provide startup help if the user doesn't include the appropriate command line parameters. At a minimum, you should tell the user what is needed to run the program. Listing 16.1 provides a program called PRINTIT.C. This program types a listing with line numbers. This listing provides startup help if command line parameters aren't entered.

Listing 16.1. PRINTIT.C. Listing using startup help.

```
1:  /*  Program: PRINTIT.C
2:   *  Author:  Bradley L. Jones
3:   *  Purpose: This program prints out a listing with
4:   *  line numbers!
5:   *=======================================================*/
6:
7:  #include <stdio.h>
8:  #include <stdlib.h>
9:
10: void do_heading(char *filename);
11:
12: int line, page;
13:
14: main( int argv, char *argc[] )
15: {
16:    char buffer[256];
17:    FILE *fp;
18:
19:    if( argv < 2 )
20:    {
21:       fprintf(stderr, "\nProper Usage is: " );
22:       fprintf(stderr, "\n\nPRINTIT filename.ext\n" );
23:       exit(1);
24:    }
25:
26:    if (( fp = fopen( argc[1], "r" )) == NULL )
27:    {
28:         fprintf( stderr, "Error opening file, %s!", argc[1] );
29:         exit(1);
30:    }
```

```
31:
32:     page = 0;
33:     line = 1;
34:     do_heading( argc[1]);
35:
36:     while( fgets( buffer, 256, fp ) != NULL )
37:     {
38:        if( line % 55 == 0 )
39:           do_heading( argc[1] );
40:
41:        fprintf( stdprn, "%4d:\t%s", line++, buffer );
42:     }
43:
44:     fprintf( stdprn, "\f" );
45:     fclose(fp);
46:     return 0;
47:  }
48:
49:  void do_heading( char *filename )
50:  {
51:      page++;
52:
53:      if ( page > 1)
54:         fprintf( stdprn, "\f" );
55:
56:      fprintf( stdprn, "Page: %d, %s\n\n", page, filename );
57:  }
```

Output

```
C:\PROG\DAY16> Printit

Proper Usage is:

PRINTIT filename.ext
```

Analysis

The output displayed is the result of running PRINTIT without any command line parameters. As you can see from the output, the user is informed of how to run the program. This is just enough information to help the user try again.

This startup help was easy to add in the listing. By adding the `argc` and `argv` parameters to your `main()` function, you can capture command line parameters. The `argv` parameter contains the number of separate commands that were on the command line. For PRINTIT, two arguments are needed, the PRINTIT program and the file to be printed. Line 19 uses an `if` to determine if less than two parameters were entered. If true, then the user is provided with the startup help (lines 20 to 24). If two or more parameters were entered, then the program runs. If additional commands or filenames are passed, they are ignored.

DO	DON'T

DO use startup help when your application doesn't have any screens (for example, in command line utilities).

General Help

General help is provided in the application. Each of the types of general help are usually available in two different ways. The first is via the Help pull-down menu on the action bar. The second is by using an accelerator key. Table 16.1 presents each of the types of general help along with its corresponding accelerator key.

Table 16.1. General help.

Action Bar Text	Accelerator Key
Help for help	Shift+F10
Extended help	F2
Keys help	F9
Help index	F11

Note: Graphical applications will also contain a Help button that can be clicked on with a mouse to access help.

Although there are accelerator keys for each of the types of general help, it isn't mandatory that the accelerator keys be used. They are just suggestions. Each of these keys can be used for other purposes in your application, even if you include the general help options.

Expert Tip: The F2 accelerator key for Extended help is used quite often. The other accelerator keys in Table 16.1 aren't used as often. I suggest that you set up F2 for extended help and leave the other accelerator keys off if they are needed for other functions. It is suggested that you don't use F2 for other tasks. The other accelerator keys in Table 16.1 can be used for other tasks.

What Is Help for Help?

Help for help may seem somewhat rhetorical, but it may well be needed. In some systems, the help that is provided can be more complicated than the system itself. This aid informs the user on how to use the help provided with the application.

Note: Many applications do not provide help for help.

What Is Extended Help?

Extended help is provided in virtually all applications that provide help. It provides information for the portion of the application currently being used.

Extended help should provide an overview of using the current screen. This information can include the purpose of the current screen, specific values for fields presented, and how to accomplish special tasks on the screen. This help should be available via the action bar. In addition, you should allow this information to be available through a function key. If the F1 key isn't being used for context-sensitive help (covered later today), then it should be used for extended help. If the F1 key is unavailable, the F2 function key should be used for extended help.

Expert Tip: You should make it easy for users to access help in your applications.

Expert Tip: Whenever possible, you should include both context-sensitive help and extended help in your applications.

In an application such as *Record of Records!*, there would be different extended help for each of the entry screens. In addition, the menus would have separate extended help. Because each of the screens is functionally different, it should make sense that each extended help would be separate.

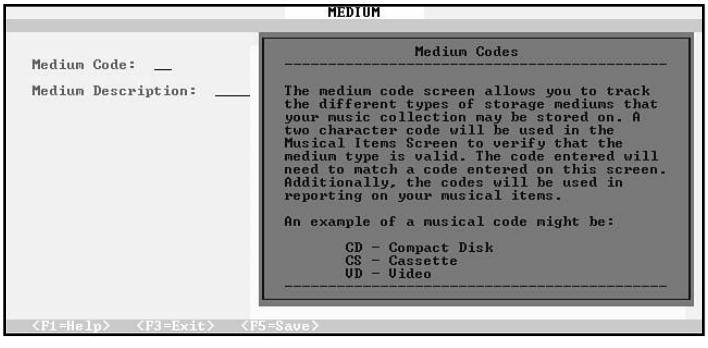
The help boxes that were provided in the *Record of Records!* application would be considered scaled-down versions of extended help. Because the *Record of Records!* application doesn't have context-sensitive help—yet—the F1 function key is used for extended help. At this time, you should modify the *Record of Records!* application to provide better extended help. A replacement for the `display_medium_help()` function for the medium code screen is in Listing 16.2. You'll notice that this version of the function provides much more descriptive help.

Listing 16.2. LIST1602.C. A new `display_medium_help()` function.

```
1:  /*-----------------------------------------------------*
2:   *   display_medium_help()                             *
3:   *-----------------------------------------------------*/
4:
5:  void display_medium_help(void)
6:  {
7:    int  ctr;
8:    char *scrnbuffer;
9:
10:   char helptext[19][45] = {
11:      "              Medium Codes",
12:      "-------------------------------------------",
13:      "",
14:      "The medium code screen allows you to track",
15:      "the different types of storage mediums that",
16:      "your music collection may be stored on. A",
17:      "two character code will be used in the",
18:      "Musical Items Screen to verify that the",
19:      "medium type is valid. The code entered will",
20:      "need to match a code entered on this screen.",
21:      "Additionally, the codes will be used in",
22:      "reporting on your musical items.",
23:      "",
24:      "An example of a musical code might be:",
25:      "",
```

```
26:        "       CD - Compact Disk",
27:        "       CS - Cassette",
28:        "       VD - Video",
29:        "-----------------------------------------" };
30:
31:     scrnbuffer = save_screen_area(2, 23, 28, 78 );
32:
33:
34:     grid( 3, 23, 28, 77, ct.shdw_fcol, ct.bg_bcol, 3 );
35:     box(2, 22, 29, 78, SINGLE_BOX, FILL_BOX,
36:                         ct.help_fcol, ct.help_bcol);
37:
38:     for( ctr = 0; ctr < 19; ctr++ )
39:     {
40:       write_string( helptext[ctr],
41:                     ct.help_fcol, ct.help_bcol, 3+ctr, 32 );
42:     }
43:
44:     cursor(24, 79);
45:     getch();
46:     restore_screen_area(scrnbuffer);
47:  }
```

Output



Analysis You'll notice that the help information has been coded into an array. This is not necessarily the best way of adding help because this uses a lot of data space. You may find that it is better to create a separate disk file that you can read into your application. This is similar to what will be done later when you add context-sensitive help to your application.

Note: By using a separate file, you'll be able to change the help information without recompiling your application.

What Is Keys Help?

Keys help is relevant to an application that uses a lot of accelerator keys. Keys help should provide information on what accelerator keys or function keys do. This information is generally presented in a format similar to the following:

F1	Help
F3	Exit program
F5	Add a record to the database
F10	Access action bar
<Esc>	Cancel current function
<Enter>	Next field

What Is Index Help?

In some applications, the extended help can become very detailed and very long. The help may be broken down into sections similar to the sections in this book. To help users get the specific help that they need, an index may be provided. This index for the extended help provides a means for users to quickly find exactly what they need.

The Index help should provide a list of the topics available. The user should be able to select an item from the list. Upon selection, the corresponding extended help should be presented.

Note: Index help, which may also be referred to as the Help index, is only provided in applications with a large amount of help.

Using an About Box

An about box is the most commonly used help screen. It can also be the least helpful. An about box serves a slightly different purpose from the other types of help. Instead of providing help directly aimed at using the application, an about box provides high-level information. The information included will be the application's name, its

version number, the company that produced/owns the software, and any copyright information. As you can see, this is identifying information rather than help information.

Other information that may be included in an about box would be the registration number, if one exists, a company logo, the software's logo, or some system information. System information that may be included would be the available memory and the total memory.

As you should see, all the information is helpful to you in addition to the user. If a user calls for support, the information in the about box helps you identify which program and which version of the program the user is running. In addition, it helps the user to know who to contact because the company name should be in the about box.

Following is Listing 16.3. This listing contains the code for the *Record of Records!* about box. This code can be linked to the function calls in the *Record of Records!* about boxes.

Type

Listing 16.3. ABOUT.C. The *Record of Records!* about box code.

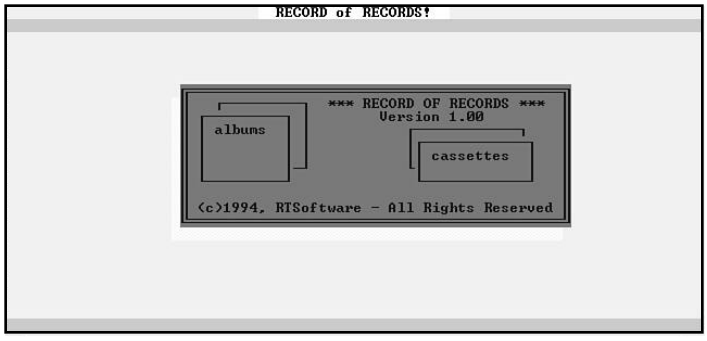
```
1:  /*=========================================================
2:   * Filename: ABOUT.C
3:   *           About box for RECORD OF RECORDS
4:   *
5:   * Author:   Bradley L. Jones
6:   *
7:   * Purpose:  Dispays about box for Record of Records!
8:   *
9:   * Note:     This listing can either be linked with other
10:  *           listing in the Rec of Rec app, or it can be
11:  *           added to the bottom of the RECOFREC.C listing
12:  *
13:  *           Prototype should be added to recofrec.h
14:  *=========================================================*/
15:
16: #include <stdio.h>
17: #include <conio.h>       /* not an ANSI header, used for getch() */
18: #include <string.h>      /* for strlen() */
19: #include <ctype.h>
20: #include "tyac.h"
21: #include "records.h"
22: #include "recofrec.h"
23:
24:
25: void display_about_box(void);
```

continues

Listing 16.3. continued

```
26:
27:  void display_about_box( void )
28:  {
29:     char *scrn_buffer = NULL;
30:
31:     scrn_buffer = save_screen_area( 6, 17, 19, 64);
32:
33:     grid( 7, 17, 19, 63, ct.shdw_fcol, ct.bg_bcol, 3 );
34:     box( 6, 16, 20, 64, DOUBLE_BOX, FILL_BOX,
35:          ct.help_fcol, ct.help_bcol);
36:
37:     /* draw graphic boxes on box */
38:     box( 7, 12, 24, 34, SINGLE_BOX, BORDER_ONLY,
39:                    ct.help_fcol, ct.help_bcol);
40:     box( 8, 13, 22, 32, SINGLE_BOX, FILL_BOX,
41:                    ct.help_fcol, ct.help_bcol);
42:
43:
44:     box(  9, 12, 46, 59, SINGLE_BOX, BORDER_ONLY,
45:                    ct.help_fcol, ct.help_bcol);
46:     box( 10, 13, 47, 60, SINGLE_BOX, FILL_BOX,
47:                    ct.help_fcol, ct.help_bcol);
48:
49:
50:     /* Fill in text */
51:     write_string( "*** RECORD OF RECORDS ***",
52:                    ct.help_fcol, ct.help_bcol, 7, 37 );
53:     write_string( "albums",
54:                    ct.help_fcol, ct.help_bcol, 9, 24 );
55:     write_string( "cassettes",
56:                    ct.help_fcol, ct.help_bcol, 11, 49 );
57:     write_string( "Version 1.00",
58:                    ct.help_fcol, ct.help_bcol, 8, 43 );
59:     write_string( "(c)1994, RTSoftware - All Rights Reserved",
60:                    ct.help_fcol, ct.help_bcol, 15, 22 );
61:
62:     cursor(24, 79);
63:     getch();
64:
65:     restore_screen_area(scrn_buffer);
66:  }
67:
68:  /*=========================================================*
69:   *                     end of listing                      *
70:   *=========================================================*/
```

Output



Analysis

This function will need to be linked with the other programs for the *Record of Records!* application. You can choose to add it to the RECOFREC.C listing, or you can keep it in its own separate listing. You'll need to add the prototype to the RECOFREC.H header file because you will be calling this function from each of the different areas. In fact, you'll want to call this from each of the three entry and edit screens along with the main menu. Each of these should have the `display_about_box()` function called from the `About...` option on the Help pull-down menu.

This function shouldn't offer any surprises. The function saves the screen area that will hold the about box (line 31). A grid and a box are then displayed to hold the about information. Lines 38 to 47 draw the graphic boxes that are to appear in the about box. Lines 51 to 60 display the textual information. In your applications, you can draw and write any textual information you want. In the case of *Record of Records!*, the application name, some graphics that give an idea of what the application does, a version number, and, finally, a copyright with a company name and date are presented. Line 65 restores the screen when the user presses a key. The function then ends and control returns to the calling program.

Context-Sensitive Help

Context-sensitive help is generally the most valuable to a user. It is help that provides information on the task that is currently being executed. For example, if a user is on the Medium Code field in the medium screen, then the context-sensitive help would provide information on what a medium code is or on what values are valid for medium codes.

In Listing 16.4, a function that provides context-sensitive help has been created. This function should be added to the medium code screen that was originally presented on Day 13. A few changes need to be made to integrate this function into the medium code listing. These changes are detailed after Listing 16.4.

Listing 16.4. LIST1604.C. The context-sensitive help.

```
1:  #include <stdlib.h>
2:  /*******************************************************
3:   *   NOTE:   Prototypes for following two functions are  *
4:   *           in the recofrec.h header file.  These same  *
5:   *           functions are usable in other screens.      *
6:   *******************************************************/
7:
8:  /*-------------------------------------------------------*
9:   * Function: display_context_help()                      *
10:  * Purpose:  Display help that is relative to a specific *
11:  *           field on a specific screen.                 *
12:  * Notes:    The first parameter needs to be the         *
13:  *           name of the help file.                      *
14:  *           Each screen should have its own help file   *
15:  *           since offsets in the file are based on      *
16:  *           field position.                             *
17:  *           Each position should have 3 lines in its    *
18:  *           help file. Max line length 64 characters.   *
19:  *-------------------------------------------------------*/
20: void display_context_help( char *file, int position )
21: {
22:    FILE *fp;
23:    int   ctr;
24:    char *rv = NULL;
25:
26:    char *buffer1 = NULL;
27:    char *buffer2 = NULL;
28:    char *buffer3 = NULL;
29:
30:    /* allocate buffers */
31:    buffer1 = (char *) malloc(65 * sizeof(char));
32:    buffer2 = (char *) malloc(65 * sizeof(char));
33:    buffer3 = (char *) malloc(65 * sizeof(char));
34:
35:    /* make sure all the allocations worked */
36:    if( buffer1 == NULL || buffer2 == NULL || buffer3 == NULL)
37:    {
38:        display_msg_box("Error allocating memory...",
39:                        ct.err_fcol, ct.err_bcol );
40:        if( buffer1 != NULL )
41:           free(buffer1);
42:        if( buffer2 != NULL )
43:           free(buffer2);
44:        if( buffer3 != NULL )
45:           free(buffer3);
46:
47:        return;
48:    }
```

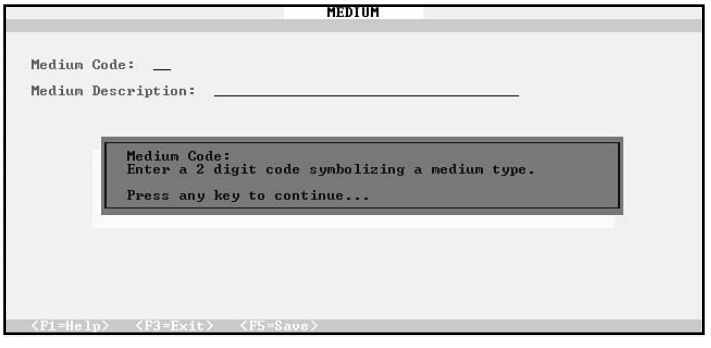
```
49:
50:
51:     fp = fopen( file, “r”);
52:
53:     if( fp == NULL )  /* make sure the file was opened */
54:     {
55:        display_msg_box(“Error opening help file...”,
56:                        ct.err_fcol, ct.err_bcol );
57:     }
58:     else
59:     {
60:        /* spin through to appropriate record */
61:        for( ctr = 0; (ctr < (position * 3)) ; ctr++ )
62:        {
63:           rv = fgets( buffer1, 65, fp );
64:           if( rv == NULL)
65:             break;
66:        }
67:
68:        /* ready to read three lines */
69:        if( rv != NULL || position == 0 )
70:        {
71:           rv = fgets( buffer1, 65, fp );
72:           if( rv != NULL )
73:           {
74:              rv = fgets( buffer2, 65, fp );
75:              if( rv != NULL )
76:                  rv = fgets( buffer3, 65, fp );
77:           }
78:
79:           display_cntxt_help_msg( buffer1, buffer2, buffer3 );
80:        }
81:        else  /* hit end of file too soon */
82:        {
83:           display_msg_box( “Error in message file...”,
84:                        ct.err_fcol, ct.err_bcol );
85:        }
86:
87:        fclose( fp );
88:     }
89:
90:     free( buffer1 );
91:     free( buffer2 );
92:     free( buffer3 );
93:  }
94:
95:  /*-------------------------------------------------------*
96:   *   display_context_help()                              *
97:   *-------------------------------------------------------*/
```

continues

Listing 16.4. continued

```
98:  void display_cntxt_help_msg( char *string1,
99:                               char *string2,
100:                              char *string3 )
101: {
102:    char *scrn_buffer = NULL;
103:    scrn_buffer = save_screen_area( 10, 16, 10, 70 );
104:
105:    grid( 11, 16, 10, 69, ct.shdw_fcol, ct.bg_bcol, 3 );
106:    box(10, 15, 11, 70, SINGLE_BOX, FILL_BOX,
107:                    ct.help_fcol, ct.help_bcol);
108:
109:    write_string( string1, ct.help_fcol, ct.help_bcol, 11, 14 );
110:    write_string( string2, ct.help_fcol, ct.help_bcol, 12, 14 );
111:    write_string( string3, ct.help_fcol, ct.help_bcol, 13, 14 );
112:
113:    write_string( "Press any key to continue...",
114:                  ct.help_fcol, ct.help_bcol, 14, 14 );
115:
116:    cursor(24, 79);
117:    getch();
118:
119:    restore_screen_area(scrn_buffer);
120: }
121:
122: /*=========================================================*
123:  *                     end of listing                      *
124:  *=========================================================*/
```

Output



Analysis

This function can be added to the end of the medium code screen's listing. A few changes need to be made to the medium code screen listing. The F1 case within the `get_medium_input_data()` function (lines 193 to 196 in Listing 13.4) should be modified. The call in the F1 case was used to access extended help. This should now

be moved to an F2 function. The resulting code for the F1 and F2 function keys should be similar to the following listing. This code should replace lines 193 to 196 in Listing 13.4.

Type

Listing 16.5. LIST1605.C. Replacement F1 case and new F2 case.

```
case F1:          /* context sensitive help */
                  display_context_help( HELP_DBF, position );
                  break;

case F2:          /* extended help */
                  display_medium_help();
                  break;
```

In addition to replacing your F1 case with the preceding code, you should also modify the `getline()` calls that set up the exit keys. You need to add the F2 key to the list. This requires that you increase the number of exit keys by one from 12 to 13 keys. The following lines should replace lines 111 to 118 in Day 13's Listing 13.4.

```
/*  Set up exit keys.  */
static char fexit_keys[ 14 ] = { F1, F2, F3, F4, ESC_KEY,
                  PAGE_DN, PAGE_UP, CR_KEY,
                  TAB_KEY, ENTER_KEY, SHIFT_TAB,
                  DN_ARROW, UP_ARROW, NULL };

static char *exit_keys = fexit_keys;
getline( SET_EXIT_KEYS, 0, 13, 0, 0, 0, exit_keys );
```

The output of Listing 16.4 presented the context-sensitive help that is received when F1 is pressed and the cursor is on the Medium Code field. To get the medium code to work, you need to incorporate all of the changes mentioned so far. In addition, you need to create a help file called MEDIUM.HLP. This file needs to contain the context-sensitive help. The following unnumbered listing contains the text in the MEDIUM.HLP help file.

Type

Listing 16.6. MEDIUM.HLP. The context-sensitive help for the medium code screen.

```
Medium Code:
Enter a 2 digit code symbolizing a medium type.

Medium Description:
Enter a description for the Medium Code.
```

The MEDIUM.HLP listing contains the contextual help. You should note that the file contains six lines. The last line is blank.

Looking back at Listing 16.4, you can see how the context-sensitive help works. When F1 is pressed, the context-sensitive help function, `display_context_help()`, is called. The `display_context_help()` function was presented in Listing 16.4 (line 2). The `display_context_help()` function starts by declaring three character buffers and allocating them with the `malloc` function (lines 26 to 33). Line 36 then verifies that all three character buffers were allocated. If one of them was not allocated, then an error message is displayed. Lines 40 to 45 then perform cleanup to ensure that any memory that was allocated is freed before returning to the calling program. Notice that the program reports the problem and keeps going. As long as users are doing functions that don't require allocating more memory, they will be able to continue with the program.

If the allocation was okay, then the function continues in line 51. Here, the function attempts to open the filename that was passed. In this case, the file is MEDIUM.HLP. Because a file name wasn't hard coded, this function will be usable in the other entry and edit screens.

If the open isn't successful, then once again an error message is displayed in line 55. If the file is opened, then the appropriate information needs to be found. The help files should be set up so that there are three lines for each position on the screen. The first three lines of the help file should contain the information that will be displayed for the field in position one. The second set of three lines should contain the information for the second position on the screen, and so on. If a field doesn't have three lines of help, then blank lines should be included in the file.

The `for` statement, in lines 61 to 66, spins through the file to the appropriate set of help lines by doing reads in the file. The `fgets()` function is used because the lines of text are variable length. If the lines were all a set length, then `fseek()` could be used along with `fread()`. Using the `fseek()` and `fread()` functions would be more efficient; however, they require that the help file contain lines that are all the same length. Each line of the file is read from the beginning. What is in the lines is ignored until the offset is reached based on the field position.

Line 69 begins the read for the three lines of information that will be displayed on the screen. If the end of the file has been reached, then you don't want to continue reading the help file. The signal for the end of the file is when `fgets()` returns the value of `NULL`. Line 69 checks to see if the end of the file has been reached. If the position is 0, then the file hasn't been read. Lines 71 to 76 then do three additional reads to obtain the

appropriate help information. With each read, a check is done to ensure that everything went okay. Once completed, line 79 calls a function to actually display the three lines on the screen. This function, `display_cntxt_help_msg()`, is similar to the `display_msg_box()` function. Instead of a single line, three lines of information are displayed (see lines 95 to 120).

After displaying the context help, the program waits for the user to press a key. The help file is then closed (line 87) and the allocated memory is released back to the system (lines 90 to 93). Control is then returned to the calling function.

This function could be sped up a little bit. Instead of opening and closing the file each time the help is needed, you could open it at the beginning of the program and close it when the program ends.

Note: The exercises at the end of today will ask you to create context-sensitive help for the other screens in the *Record of Records!* application.

Tutorials

Tutorials are different from Help. Whereas help provides aid in using an application, tutorials provide training. A tutorial walks the user through the steps of using the applications. These instructions can be presented in several forms. These forms can be either online, in paper documentation, or both.

Online tutorials can either be interactive or non-interactive. An *interactive tutorial* enables the user to interact with what is being presented. For example, a data-entry screen within the tutorial may require the user to actually enter data.

A *non-interactive tutorial* presents the information, but doesn't enable the user to actually perform any of the application's functions. Most non-interactive tutorials are presented in slide show format. One slide containing information is presented with textual information for the user to read. Included are pictures of screens, messages, dialog boxes, and more.

Note: Interactive tutorials are more effective than non-interactive tutorials; however, they can be more difficult to develop.

Tutorials that aren't online are easier to produce. These paper tutorials are generally interactive. They walk the user through using each portion of the application. Included in the tutorial are the same constructs that are included in the non-interactive tutorials. In addition, steps are presented that tell the user how to get into the application and actually perform the different processes.

Note: Tutorials and Help are not the same. A tutorial is not a replacement for help. Help should not be written to be a tutorial.

Documentation and Manuals

An entire book could be written on preparing documentation for computer applications. Documentation comes in several forms and serves several purposes. The main goal of documentation should be to help the users in their utilization of the application.

Documentation can come in the form of several different types of manuals. There are installation or set-up manuals, user guides, reference guides, and tutorials. In addition, several application specific manuals may be provided.

Each of the different manuals has a different purpose. The installation or set-up manuals help the user install and set up the software. Once the software is running, these manuals are generally put aside. The user guide is provided as an overview to the application. It should contain sections on each portion of the application. A reference guide may contain information similar to the user guide, but it will differ in the way it's organized. In addition, the level of detail in a reference will generally be greater than that of a user guide. A reference manual should be designed so that information is easy to access. A user manual will be set up in the order that a user is most likely to use the application.

Tutorials, which were described earlier, provide the user with step-by-step instructions in actually using the application. These instructions have the users actually enter information into the application so that they get hands-on experience.

Note: Tutorials that are provided in manuals are generally hands-on in nature, which means that they have the user actually enter information into the application. Online tutorials may or may not be hands-on.

DO	DON'T

DO include help in your applications.

DON'T confuse tutorials with help.

DO include documentation with your applications if you plan to sell them.

Summary

Today's chapter provides help. Actually, it provides information on adding help to your application. There are several different forms of help that can be added to your applications. Context-sensitive help is the most helpful. It provides information on the action that is being performed at the time help is requested. Extended help is more common. Extended help provides information relative to the current screen being used. Several other types of help are also available. These include help for help, index help, keys help, and about boxes. In addition to help, some applications provide tutorials. To supplement help, documentation may be provided in the form of startup or installation manuals, user guides, or reference manuals.

Q&A

Q Do you have to add help to your applications?

A No, you don't have to add help to your applications; however, most times you should. Virtually all applications that are available for sale have some form of help. At a minimum, they contain an about box. Most include context-sensitive help or extended help.

Q Is the F1 key always used for context-sensitive help?

A No, some applications don't provide context-sensitive help, but do provide extended help. In these applications, the F1 key is used for the extended help. The reason behind this is because most people expect the F1 key to provide some form of help.

Q Should a tutorial be a part of the application?

A No, a tutorial should be a separate application. You can provide access to the tutorial through the Help pull-down menu; however, you shouldn't include the tutorial as a part of the application. The application should be written to serve a function. A tutorial is written to teach the user that function. Your prime objective should be to write the application, not the tutorial.

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. Is help necessary in a computer application?
2. What are the different types of help?
3. Are tutorials considered help?
4. When should startup help be used?
5. If you are only providing extended help in your application, what accelerator key should be assigned to it?
6. If you are providing extended help and context-sensitive help, what accelerator keys should be used?

Exercises

Today's exercises are all on your own. You should model your answers after the material presented today.

1. **ON YOUR OWN:** You were shown code for adding extended help to the medium code screen. Add extended help to the Group Information screen and the Musical Items screen.
2. **ON YOUR OWN:** Create context-sensitive help for the Group Information screen and the Musical Items screen.
3. **ON YOUR OWN:** Add help to the application that you are creating on your own. (See "ON YOUR OWN" exercises from previous days.)