DESIGN AND IMPLEMENTATION OF LEAVE MANAGEMENT SYSTEM FOR SIBCOM TECHNOLOGIES PVT. LTD.,BANGALORE

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT
FOR THE DEGREE OF

MASTER OF COMPUTER APPLICATIONS

BY:

SHALINI SHARMA

MCA 26/90

VOLUME II

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY, PATIALA
(DEEMED UNIVERSITY)

1993

INCLUDE.H

```
/*                          Filename ........Include.H                    *
/*                          ---------------------------                   *


  #include <malloc.h>
  #include <bios.h>
  #include <conio.h>
  #include <ctype.h>
  #include <dos.h>
  #include <graph.h>
  #include <io.h>
  #include <memory.h>
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
```

DEFINE.H

DEFINE.H

```
/*                          Filename .........Define.H                          */
/*                          -------------------------                           */


/* ------------------------------------------------------------------------------*/
/*  Definitions of constants used in Dbfile.H and Form.H                         */

  #define FIELD_TERMINATOR 0x04
  #define MAX_FIELDS         30
  #define FIELD_NAME_WIDTH   11
  #define MAX_FIELD_WIDTH   100
  #define MAX_REC_LENGTH   1000
  #define TRUE                1
  #define FALSE               0
  #define NUM_FIELDS         20


/* ------------------------------------------------------------------------------*/
/*  Definitions of constants used in Printer.H                                   */

  #define PRINTER_OFF              200
  #define PRINTER_READY            144
  #define PRINTER_OFFLINE_PAPER      8
  #define PRINTER_OFFLINE_NO_PAPER  40
  #define PRINTER_NOT_CONNECTED     48


/* ------------------------------------------------------------------------------*/
/*  Definitions of constants used in Utility.C                                   */

  #define h_line   '\xC4'
  #define v_line   '\xB3'
  #define dh_line  '\xCD'
  #define dv_line  '\xBA'
  #define tlc      '\xDA'
  #define trc      '\xBF'
  #define blc      '\xC0'
  #define brc      '\xD9'
  #define dtlc     '\xC9'
  #define dtrc     '\xBB'
  #define dblc     '\xC8'
  #define dbrc     '\xBC'
  #define dlc      '\xC7'
  #define drc      '\xB6'
```

```
/* ------------------------------------------------------------------------*/
/*  Definitions of Keyboard keys used in Form.C and Menu.C                 */

  #define ENTER    13
  #define ESC      27
  #define HOME     199
  #define UP_ARR   200
  #define PGUP     201
  #define LT_ARR   203
  #define RT_ARR   205
  #define END      207
  #define DN_ARR   208
  #define PGDN     209
  #define INS      210
  #define DEL      211

  #define CAPS_ON  1
  #define CAPS_OFF 0


/* ------------------------------------------------------------------------*/
/*  Structure for getting cursor position                                  */

  typedef struct rc_cord
  {
      unsigned char row;
      unsigned char col;
  }RC_CORD;


  #define DONE      1
  #define VIDEO_ADDRESS 0xb8000000
```

UTILITY.C

```
/*                      Filename ........Utility.C                          */
/*                      -------------------------                           */


/* ------------------------------------------------------------------------*/
/*  Prototypes  of  functions  used  in  Utility.C                         */

union REGS r;

int getcharacter( void);
int confirm( char *, unsigned char, int, int);
int is_space( char *);

void hor_line( unsigned char, unsigned char, unsigned char, char);
void ver_line( unsigned char, unsigned char, unsigned char, char);
void box( unsigned char, unsigned char, unsigned char, unsigned char, char);
void clearscreen( unsigned char, unsigned char, unsigned char, unsigned char);
void setposition( unsigned char, unsigned char);
void put_char( char, unsigned char, unsigned char);
void restorescr( char *);
void message( char *, unsigned char, unsigned char );
void reversevideo( char *);
void normalvideo( char *);
void put_str( char *, unsigned char , unsigned char, unsigned char);
void install( void);
char *savescr( char *);
char whatchar( unsigned char , unsigned char );

RC_CORD getposition( void);


/* ------------------------------------------------------------------------*/
/*  Module ..........Getcharacter()                                        */
/*  Function ........Gets character from keyboard, echoes to screen        */
/*  Parameters ......Nil                                                   */
/*  Returns .........The character                                         */

int getcharacter()
{
    int ch;

    while(1)
    {
       r.h.ah = 1;
       int86( 0x16, &r, &r);
       r.h.ah = 0;
       int86( 0x16, &r, &r);
```

```
    if( r.h.al == 0)
        ch = r.h.ah | 128;
    else
        ch = r.h.al;

    return (ch);
}



* -----------------------------------------------------------------------*/
*  Module ..........Clearscreen()                                        */
*  Function ........Clears the screen within specifed rows and columns   */
*  Parameters ......Starting row, starting column, ending row, ending col */
*  Returns .........Nil                                                  */

oid clearscreen( unsigned char row1, unsigned char col1, unsigned char row2,
                unsigned char col2)

  r.h.ah = 6;
  r.h.al = 0;
  r.h.ch = row1;
  r.h.cl = col1;
  r.h.dh = row2;
  r.h.dl = col2;
  r.h.bh = 7;
  int86( 0x10, &r, &r);



* -----------------------------------------------------------------------*/
*  Module ..........Setposition()                                        */
*  Function ........Moves cursor to specified row and column position on */
*                    console                                             */
*  Parameters ......Row position, column position                       */
*  Returns .........Nil                                                  */

oid setposition( unsigned char row, unsigned char col )

  r.h.ah = 0x0f;
  int86(0x10,&r, &r);
  r.h.ah = 0x02;
  r.h.dh = row;
  r.h.dl = col;
  int86(0x10, &r, &r);
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Box()                                                 */
/*  Function ........Draws a box, given end coordinates of diagonal and    */
/*                    line type                                            */
/*  Parameters ......Startinng row of diagonal, starting column of diagonal,*/
/*                    ending row of diagonal, ending column of diagonal,   */
/*                    line type                                            */

void box( unsigned char x1, unsigned char y1, unsigned char x2,
          unsigned char y2, char boundary)
{
    char hline_type, vline_type, topl, topr, botl, botr;

    if( boundary != 's')
    {
        hline_type = 'd';
        vline_type = 'd';
        topl = dtlc;
        topr = dtrc;
        botl = dblc;
        botr = dbrc;
    }
    if( boundary == 's')
    {
        hline_type = 's';
        vline_type = 's';
        topl = tlc;
        topr = trc;
        botl = blc;
        botr = brc;
    }
    setposition( x1, y1);
    putch( topl);
    setposition( x1, y2);
    putch( topr);
    setposition( x2, y1);
    putch( botl);
    setposition( x2, y2);
    putch( botr);
    hor_line( x1, y1+1, y2-1, hline_type);
    hor_line( x2, y1+1, y2-1, hline_type);
    ver_line( x1+1, x2-1, y1 ,vline_type);
    ver_line( x1+1, x2-1, y2 ,vline_type);
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Hor_line()                                            */
/*  Function ........Draws a horizontal line of specified line type        */
/*  Parameters ......Row position, starting column position,               */
/*                     ending column position, line type                   */
/*  Returns .........Nil                                                   */

void hor_line( unsigned char row, unsigned char col1, unsigned char col2,
               char hline_type)
{
    int i;

    if( hline_type == 's')
       hline_type = h_line;
    else
       hline_type = dh_line;
    for( i = col1; i <= col2; i++)
    {
        setposition( row, i);
        putch( hline_type);
    }
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Ver_line()                                            */
/*  Function ........Draws a vertical line of specified line type          */
/*  Parameters ......Starting row position, ending row position,           */
/*                     column position, line type                          */
/*  Returns .........Nil                                                   */

void ver_line( unsigned char row1, unsigned char row2, unsigned char col,
               char vline_type)
{
    int i;
    if( vline_type == 's')
        vline_type = v_line;
    else
        vline_type = dv_line;
    for ( i = row1; i <= row2; i++)
    {
        setposition( i, col);
        putch( vline_type);
    }
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Getposition()                                         */
/*  Function ........Gets current cursor position                          */
/*  Parameters ......Nil                                                   */
/*  Returns .........Row position and column position                      */

RC_CORD getposition( void)
{
    RC_CORD position;

    r.h.ah = 0x0f;
    int86( 0x10, &r, &r);
    r.h.ah = 0x03;
    position.row = r.h.dh;
    position.col = r.h.dl;

    return( position);
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Savescr()                                             */
/*  Function ........Stores the current screen characters and their        */
/*                   attributes                                            */
/*  Parameters ......The screen attributes                                 */

char *savescr( char *buff)
{
   int srow, scol, i;

   for( srow = 1; srow < 26; srow++ )
        for( scol = 1; scol < 161; scol++ )
        {
           buff[ (srow - 1) * 160 + (scol - 1) ] = whatchar( srow, scol);
        }
   buff[4000] = '\0';

   return( buff);
}


char whatchar( unsigned char row, unsigned char col)
{
   char far *vidptr;

   vidptr = (char far *) VIDEO_ADDRESS;
   return( *(vidptr + (row - 1) * 160 + (col - 1)) );
}
```

```
/* ------------------------------------------------------------------------*/
/* Module ..........Restorescr()                                           */
/* Function ........Restores a screen                                      */

void restorescr( char *buff)
{
   int srow, scol;

   for( srow = 1; srow < 26; srow++ )
        for( scol = 1; scol < 161; scol++ )
             put_char( buff[ (srow - 1) * 160 + (scol - 1) ], srow, scol);
}


void put_char( char ch, unsigned char row, unsigned char col)
{
   char far *vidptr;

   vidptr = (char far *)VIDEO_ADDRESS;
   *( vidptr + ( row - 1) * 160 + (col - 1)) = ch;
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Confirm()                                             */
/*  Function ........Displays message on console, centered w.r.t the screen,*/
/*                    at specified  row and accepts two keystrokes to       */
/*                    continue the ongoing process                          */
/*  Parameters ......Message string, row position, keystroke 1, keystroke 2*/
/*  Returns .........Any one of the specified keystrokes                   */

int confirm( char *message, unsigned char row, int value1, int value2 )
{
    unsigned char col;
    int ch;

    col = 40 - strlen(message)/2;
    setposition(row, col);
    printf("%s", message);
    setposition(row, col + strlen(message));
    while(1)
    {
        ch = getcharacter();
        putchar(ch);
        if( toupper(ch) == value1 || toupper(ch) == value2 )
            return ( toupper(ch));
        setposition(row, col + strlen(message));
    }
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Is_space()                                            */
/*  Function ........Checks for a blank string                             */
/*  Parameters ......String                                                */
/*  Returns .........Value indicating status of string ( Blank = 1)        */

int is_space( char *str)
{
    unsigned char i, status;
    status = 1;

    for( i = 0; str[i] != '\0'; i++ )
    {
        if( str[i] != 32 )
        {
            status = 0;
            break;
        }
    }
    return status;
}


/* ------------------------------------------------------------------------*/
/*  Module ..........message()                                             */
/*  Function ........Prints a message String at row, with Attrib           */
/*  Parameters ......String, Row, Attrib                                   */
/*  Returns .........None                                                  */

void message( char *str, unsigned char attrib, unsigned char row )
 {
    unsigned char col;

    col = 40 - strlen(str)/2;
    setposition(row, col);
    printf("%s", str);
 }



/* ------------------------------------------------------------------------*/
/* Module ..........Put_str()                                              */
/* Function ........Displays on the screen, a string with it's attributes  */

void put_str( char *str, unsigned char row, unsigned char col, unsigned char attrib
{
   char far *vidptr;
   unsigned char i;

   vidptr = (char far *)VIDEO_ADDRESS;
```

```
   for( i = 0; str[i] != '\0'; )
   {
       if( ((row - 1) * 160 + (col - 1)) % 2 == 0 )
        {
           *( vidptr + (row - 1) * 160 + (col - 1)) = str[i];
           i++;
        }
       else
           *( vidptr + (row - 1) * 160 + (col - 1)) = attrib;
       col++;
       if( col > 79 )
           row++;
       if( str[i+1] == '\0' )
           *( vidptr + (row - 1) * 160 + (col - 1)) = attrib;

   }
}



/* ---------------------------------------------------------------------------*/
/* Module ..........Install()                                                 */
/* Function ........Installs the package                                      */

void install( void)
{
    FILE *ini_file;
    unsigned char i, ch, len;
    struct dosdate_t date;
    char *p_buff, *d_buff, *v_buff;
    struct passwd
    {
        unsigned char set;
        char password[15];
        char date_last_accessed[10];
        char filler[8];
    } init;

   clearscreen(0,0,24,79);

   ch = len = 0;
   init.set = 0;
   strcpy( init.password, "               ");
   strcpy( init.date_last_accessed, "          " );

   if( !access("lms.ini", 0) )
   {
       if( (ini_file = fopen( "lms.ini", "rb+")) != NULL )
       {
           if( fread( &init, sizeof(init), 1, ini_file ) == 1)
```

```
    {
        printf("\nWarning : LMS.INI file already exists.");
        ch = confirm("Data files would be erased!! Continue (Y/N) ? ", 24, 13, 27);
        if( ch == ESC )
        {
            fclose(ini_file);
            exit(0);
        }
    }
}
else
{
     message("Unable to open file LMS.INI for reading", 0, 23);
     confirm("Press Enter to Exit", 24, 13, 27);
     exit(0);
}

se

if( (ini_file = fopen( "lms.ini", "wb+")) != NULL )
{
    while(1)
    {
        message("Enter System Password : ", 10, 10 );
        while( ch != 13)
        {
            ch = getcharacter();
            p_buff[ len] = ch;
            len++;
            if( len == 15)
                putchar( 7);
        }
        p_buff[ len] = '\0';
        len = 0;
        message("Re enter System Password : ", 11, 10 );
        while( ch != 13)
        {
            ch = getcharacter();
            v_buff[ len] = ch;
            len++;
            if( len == 15)
               exit(0);
        }
        v_buff[ len] = '\0';
        if( !strcmp(v_buff, p_buff) )
            break;
        else
        {
            message("Passwords do not match!!", 0, 23);
            ch = confirm("Press Esc to Abort (or) Space to Continue", 24, 32,
```

```
                    if( ch == ESC )
                    {
                        fclose(ini_file);
                        return;
                    }
                }
            }
            _dos_getdate( &date);
            sprintf( d_buff, "%02d/%02d/%02d", date.day, date.month, date.year);
            strcpy( init.password, p_buff);
            strcpy( init.date_last_accessed, d_buff);
            fwrite( &init, sizeof( init), 1, ini_file);
        }
        else
        {
             message("Unable to create file LMS.INI", 0, 23);
             confirm("Press Esc to Exit", 24, 27, 27);
        }
    }
    fclose(ini_file);
}
```

PRINTER.H

```
/*                          Filename ........Printer.H                      */
/*                          --------------------------                      */


/*  This file contains the definitions of structure types used in Printer.C  */

typedef struct printer
{
   char buffer[245];
   char pause;
   unsigned char buffer_index;
   unsigned char row;
   unsigned char col;
   int printer_status;
   unsigned char exit_code;
   unsigned current_page;
   unsigned last_page;
   unsigned char printer_type;
} PRINTER;
```

PRINTER.C

```
/*                          Filename ........Printer.C                          */
/*                          --------------------------                          */


/* ------------------------------------------------------------------------------*/
/*  Prototypes of functions used in Printer.C                                    */


PRINTER init_printer(PRINTER );
PRINTER print_str( PRINTER, char *, unsigned char, unsigned char);
void clear_buff( void);
int print_char( unsigned char );
int get_status( void);


/* ------------------------------------------------------------------------------*/
/* Module ..........Init_printer()                                              */
/* Function .........Initializes the structure components of type printer       */

PRINTER init_printer( PRINTER printer)
{
  biosprint( 1, 0, 0);
  delay( 1000);
  printer.buffer_index = 0;
  printer.row = 0;
  printer.col = 0;
  printer.current_page = 0;
}


/* ------------------------------------------------------------------------------*/
/* Module .........Print_str()                                                   */
/* Function .......Sends report to the printer to be printed                    */

PRINTER print_str( PRINTER printer, char *str, unsigned char row, unsigned char col)
{
    int c, r;
    unsigned char buff_len;

    r = c = 0;
    strcpy( printer.buffer, str);
    buff_len = strlen( printer.buffer);
    printer.row = row;
    printer.col = col;
    printer.printer_status = biosprint( 2, 0, 0);
    while(1)
    {
```

```
  if( r >= printer.row && c >= printer.col)
    break;
  printer.printer_status = biosprint( 2, 0, 0);
  if( printer.printer_status == PRINTER_READY)
  {
       for( r = 0; r < printer.row; r++)
         biosprint( 0, 10, 0);
      biosprint( 0, 13, 0);
      for( c = 0; c < printer.col; c++)
         biosprint( 0, 32, 0);
      biosprint( 0, 13, 0);
   }
   else
   {
      switch( printer.printer_status)
      {
          case  PRINTER_OFF              : setposition( 22, 5);
                                           printf("Printer  Off !!");
                                           printf("\nESC to abort ; ENTER to retry");
                                           break;

           case PRINTER_OFFLINE_PAPER    : setposition( 22, 5);
                                           printf("Printer Offline !!");
                                           printf("\nESC to abort ; ENTER to retry");
                                           break;

          case PRINTER_OFFLINE_NO_PAPER:setposition( 22, 5);
                                           printf("No  Paper !!");
                                           printf("\nESC to abort ; ENTER to retry");
                                           break;
      }
      while(1)
      {
         printer.pause = getcharacter();
         if( printer.pause == ENTER || printer.pause == ESC)
            break;
      }
      if( printer.pause == ESC)
      {
          clear_buff();
          return( printer);
      }
  }


rinter.buffer_index = 0;
hile(printer.buffer_index <= buff_len)

   printer.printer_status = biosprint( 2, 0, 0);
```

```
if( printer.printer_status == PRINTER_READY )
{
    biosprint( 0, printer.buffer[printer.buffer_index], 0 );
    printer.buffer_index++;
    delay( 3 );
}
else
{
    switch( printer.printer_status)
    {
        case  PRINTER_OFF                 : setposition( 22, 5);
                                              printf("Printer  Off !!");
                                            printf("\nESC to abort ; ENTER to retry");
                                            break;

        case PRINTER_OFFLINE_PAPER        : setposition( 22, 5);
                                            printf("Printer Offline !!");
                                            printf("\nESC to abort ; ENTER to retry");
                                            break;

        case PRINTER_OFFLINE_NO_PAPER     : setposition( 22, 5);
                                            printf("No  Paper !!");
                                            printf("\nESC to abort ; ENTER to retry");
                                            break;
    }
    while(1)
    {
        printer.pause = getcharacter();
        if( printer.pause == ENTER || printer.pause == ESC)
            break;

     }
     if( printer.pause == ESC)
     {
         clear_buff();
         return( printer);
     }
}

sprint( 0, 13, 0);
sprint( 0, 10, 0);
```

```
/* ------------------------------------------------------------------------*/
/* Module ..........Get_status()                                           */
/* Function ........Gets the printer status                                */

int get_status( void)
{
   return( biosprint( 2, 0, 0) );
}


/* ------------------------------------------------------------------------*/
/* Module ..........Clear_buff()                                           */
/* Function ........Clears the printer buffer                              */

void clear_buff( void)
{
   biosprint( 0, 24, 0);
}
```

MENU.H

```
/*                              Filename ........Menu.H                    */
/*                              -----------------------                    */

/*  This file contains the structure types of sttructure components used in  */
/*  Menu.C                                                                  */

typedef struct menu_item
{
   char item[30];
   unsigned char ver_item;
   unsigned char hor_item;
   unsigned char hor_item_pos;
} MENU_ITEM;

typedef struct box_item
{
   char item[30];
   unsigned char max_ver_items;
   unsigned char ver_item_pos;
   unsigned char hor_item_pos;
} BOX_ITEM;

typedef struct menu
{
   MENU_ITEM menu_items[30];
   BOX_ITEM box_items[10];
   unsigned char max_hor_items;
   unsigned char max_ver_items;
   unsigned char hor_menu_pos;
   unsigned char max_menu_items;
} MENU;

typedef struct menu_choice
{
   unsigned char hor_choice;
   unsigned char ver_choice;
} MENU_CHOICE;
```

MENU.C

```
/*                          Filename .......Menu.C                          */
/*                          ----------------------                          */


#include "menu.h"


/* ------------------------------------------------------------------------*/
/*  Prototypes of Functions used in Menu.C                                 */

MENU *menubox( MENU *, MENU_CHOICE);
MENU *init_main_menu( MENU *, unsigned char, unsigned char, unsigned char);
MENU *print_main_menu( MENU *);
MENU *init_menu_items( MENU *, char *, unsigned char,
                       unsigned char );
MENU_CHOICE read_main_menu( MENU *, MENU_CHOICE );

int mainbar( char **, unsigned char *, char *[][]);
int read_mainmenu( char **, int *, int);

void title1( unsigned char);
void bot_scr( void);
void init_menu(char **, int, int);



/* ------------------------------------------------------------------------*/
/*  Module ..........Title()                                               */
/*  Function ........Displays name of package and current date on the console */

void title1( unsigned char row)
{
   struct date today;
    char *title = "Leave  Management  System";

    setposition( row - 1, 60);
    getdate( &today);
    printf("Date : %02d/%02d/%4d", today.da_day, today.da_mon, today.da_year); */
    setposition( row, 40 - strlen( title) / 2);
    _settextcolor( (long) 15);
    cputs( title);
    hor_line( row + 1, 40 - (strlen( title) / 2) - 1, (40 - (strlen( title) / 2)
              - 1) + strlen( title) + 1, 's');
    _settextcolor( (long) 7);
    hor_line( 21, 0, 79, 's');
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Init_main_menu()                                      */
/*  Function ........Initializes components of the menu structure          */

MENU *init_main_menu( MENU *main_menu, unsigned char max_hor_items,
                      unsigned char max_ver_items, unsigned char hor_menu_pos )

{
    unsigned char i, j;

    for( i = 1; i <= 30; i++)
    {
        for( j = 1; j <= 30; j++)
           main_menu->menu_items[i].item[j] = ' ';
        main_menu->menu_items[i].ver_item = 0;
        main_menu->menu_items[i].hor_item = 0;
        main_menu->menu_items[i].hor_item_pos = 0;
    }
    main_menu->max_hor_items = max_hor_items;
    main_menu->max_ver_items = max_ver_items;
    main_menu->hor_menu_pos  = hor_menu_pos;
    main_menu->max_menu_items = 1;

    return( main_menu);
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Init_menu_items()                                     */
/*  Function ........Initializes each menu option and sub - option         */

MENU *init_menu_items( MENU *main_menu, char *prompt, unsigned char posx,
                       unsigned char posy)

{
    strcpy( main_menu->menu_items[main_menu->max_menu_items].item, prompt);
    main_menu->menu_items[main_menu->max_menu_items].hor_item = posx;
    main_menu->menu_items[main_menu->max_menu_items].ver_item = posy;
    main_menu->max_menu_items++;

    return( main_menu);
}
```

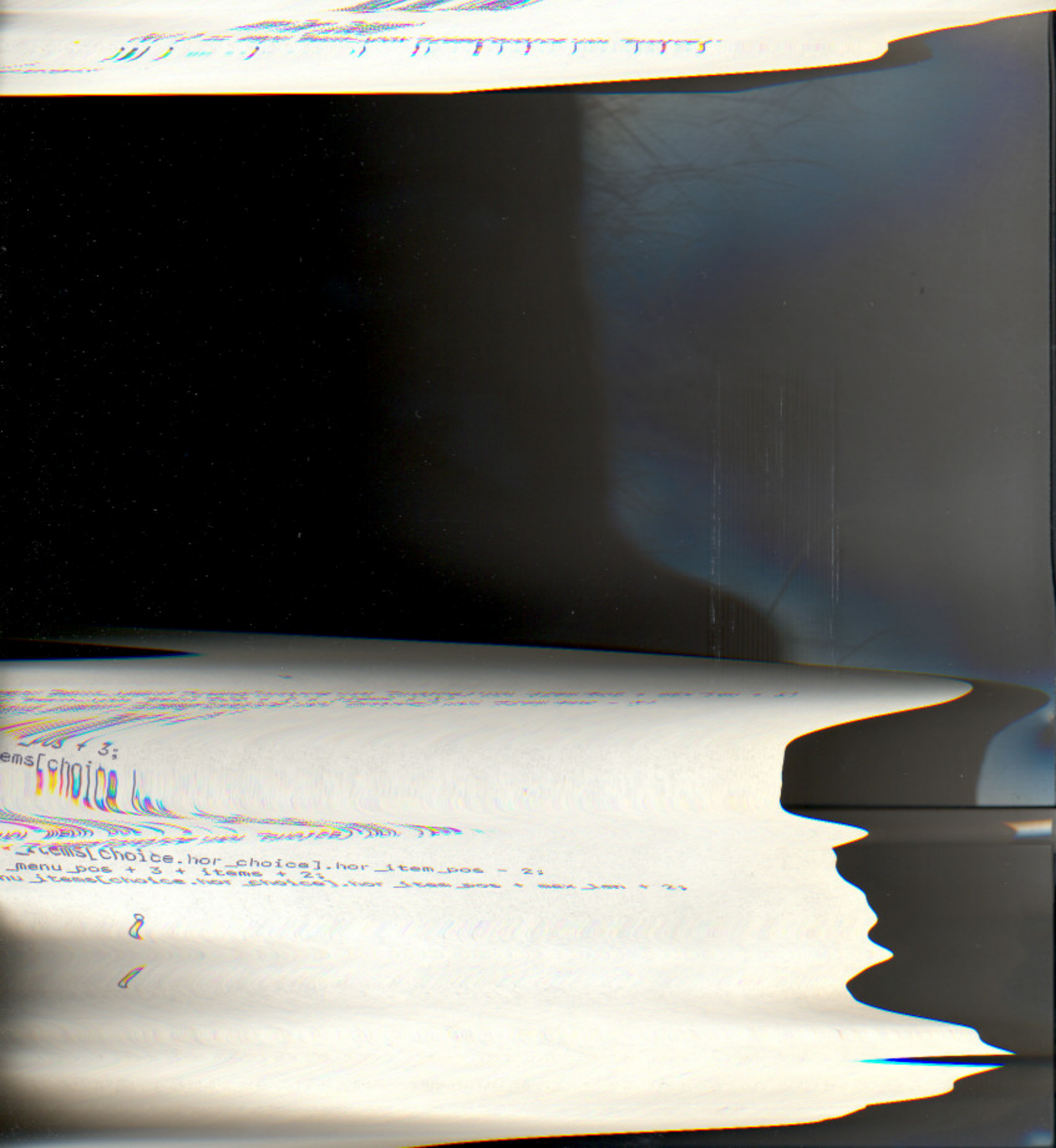
```
/* ------------------------------------------------------------------------*/
/*  Module ..........Print_main_menu()                                     */
/*  Function ........Displays on the console horizontally, the main menu   */

MENU *print_main_menu( MENU *main_menu)
{
    unsigned char i, j, len;

    for( i = 0; i <= 79; i++)
    {
        setposition( main_menu->hor_menu_pos, i);
        reversevideo(" ");
    }

    len = 5;
    for( i = 1; i < main_menu->max_menu_items; i++)
    {
        if( main_menu->menu_items[i].ver_item == 0 )
        {
            main_menu->menu_items[i].hor_item_pos = len;
            setposition( main_menu->hor_menu_pos, len);
            len = len + 5 + strlen( main_menu->menu_items[i].item);
            reversevideo( main_menu->menu_items[i].item);
        }
    }
    _settextcolor((long) 7 );
    hor_line( main_menu->hor_menu_pos + 1, 0, 79, 's');

    return( main_menu);
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Read_main_menu()                                      */
/*  Function ........Reads the highlighted option of the horizontal main menu */
/*                   and, displays the corresponding sub - options in a    */
/*                   vertical box  directly beneath the main menu option.  */
/*                   Allows movement both horizontally and vertically between */
/*                   the options, highlighting the current option          */

MENU_CHOICE read_main_menu( MENU *main_menu, MENU_CHOICE choice)
{
    unsigned char status, check, i, j, pt;
    int key;

    j = 1;
    check = 1;
    status = 0;
    pt = choice.hor_choice;
```

```
setposition( main_menu->hor_menu_pos, main_menu->menu_items[choice.hor_choice].
             hor_item_pos);
normalvideo( main_menu->menu_items[choice.hor_choice].item);
setposition( main_menu->hor_menu_pos, main_menu->menu_items[choice.hor_choice].
             hor_item_pos);
while( check)
{
    key = getcharacter();
    switch( key)
    {
        case LT_ARR : if( pt == 1)
                      {
                          setposition( main_menu->hor_menu_pos, main_menu->
                                       menu_items[pt].hor_item_pos);
                          reversevideo( main_menu->menu_items[pt].item);
                          pt = main_menu->max_hor_items;
                          setposition( main_menu->hor_menu_pos, main_menu->
                                       menu_items[pt].hor_item_pos);
                          normalvideo( main_menu->menu_items[pt].item);
                          if( status)
                          {
                              j = 1;
                              choice.hor_choice = pt;
                              main_menu = menubox( main_menu, choice);
                          }
                      }
                      else
                      {
                          setposition( main_menu->hor_menu_pos, main_menu->
                                       menu_items[pt].hor_item_pos);
                          reversevideo( main_menu->menu_items[pt].item);
                          pt--;
                          setposition( main_menu->hor_menu_pos, main_menu->
                                       menu_items[pt].hor_item_pos);
                          normalvideo( main_menu->menu_items[pt].item);
                          if( status)
                          {
                              j = 1;
                              choice.hor_choice = pt;
                              main_menu = menubox( main_menu, choice);
                          }
                      }
                      setposition( main_menu->hor_menu_pos, main_menu->
                                   menu_items[pt].hor_item_pos);
                      break;

      case RT_ARR   : if( pt == main_menu->max_hor_items)
                      {
                          setposition( main_menu->hor_menu_pos, main_menu->
                                       menu_items[pt].hor_item_pos);
```

```
                    reversevideo( main_menu->menu_items[pt].item);
                    pt = 1;
                    setposition( main_menu->hor_menu_pos, main_menu->
                                 menu_items[pt].hor_item_pos);
                    normalvideo( main_menu->menu_items[pt].item);
                    if( status)
                    {
                        j = 1;
                        choice.hor_choice = pt;
                        main_menu = menubox( main_menu, choice);
                    }
                }
                else
                {
                    setposition( main_menu->hor_menu_pos, main_menu->
                                 menu_items[pt].hor_item_pos);
                    reversevideo( main_menu->menu_items[pt].item);
                    pt++;
                    setposition( main_menu->hor_menu_pos, main_menu->
                                 menu_items[pt].hor_item_pos);
                    normalvideo( main_menu->menu_items[pt].item);
                    if( status)
                    {
                        j = 1;
                        choice.hor_choice = pt;
                        main_menu = menubox( main_menu, choice);
                    }
                }
                setposition( main_menu->hor_menu_pos, main_menu->
                             menu_items[pt].hor_item_pos);
                break;

case HOME       : setposition( main_menu->hor_menu_pos, main_menu->
                               menu_items[pt].hor_item_pos);
                reversevideo( main_menu->menu_items[pt].item);
                pt = 1;
                setposition( main_menu->hor_menu_pos, main_menu->
                             menu_items[pt].hor_item_pos);
                normalvideo( main_menu->menu_items[pt].item);
                if( status)
                {
                     choice.hor_choice = pt;
                     main_menu = menubox( main_menu, choice);
                }
                setposition( main_menu->hor_menu_pos, main_menu->
                             menu_items[pt].hor_item_pos);
                break;

case END        : setposition( main_menu->hor_menu_pos, main_menu->
                               menu_items[pt].hor_item_pos);
```

```
                    reversevideo( main_menu->menu_items[pt].item);
                    pt = main_menu->max_hor_items;
                    setposition( main_menu->hor_menu_pos, main_menu->
                                 menu_items[pt].hor_item_pos);
                    normalvideo( main_menu->menu_items[pt].item);
                    if( status)
                    {
                         choice.hor_choice = pt;
                         main_menu = menubox( main_menu, choice);
                    }
                    setposition( main_menu->hor_menu_pos, main_menu->
                                 menu_items[pt].hor_item_pos);
                    break;

case UP_ARR     : if( status)
                    {
                        if( j == 1)
                        {
                             setposition( main_menu->box_items[choice.
                             hor_choice].ver_item_pos + j, main_menu->
                             menu_items[choice.hor_choice].hor_item_pos);
                             normalvideo( main_menu->box_items[j].item);
                             j = main_menu->box_items[choice.hor_choice].
                                 max_ver_items;
                             setposition( main_menu->box_items[choice.
                             hor_choice].ver_item_pos + j, main_menu->
                             menu_items[choice.hor_choice].hor_item_pos);
                             reversevideo( main_menu->box_items[j].item);
                        }
                        else
                        {
                             setposition( main_menu->box_items[choice.
                             hor_choice].ver_item_pos + j, main_menu->
                             menu_items[choice.hor_choice].hor_item_pos);
                             normalvideo( main_menu->box_items[j].item);
                             j--;
                             setposition( main_menu->box_items[choice.
                             hor_choice].ver_item_pos + j, main_menu->
                             menu_items[choice.hor_choice].hor_item_pos);
                             reversevideo( main_menu->box_items[j].item);
                        }
                        setposition( main_menu->box_items[choice.
                        hor_choice].ver_item_pos + j, main_menu->menu_items
                        [choice.hor_choice].hor_item_pos);
                    }
                    else
                        putchar( 7);
                    break;
```

```
case DN_ARR     : if( status)
                  {
                      if( j == main_menu->box_items[choice.hor_choice].
```



```
        case ESC        : if( status)
                          {
                              status = 0;
                              clearscreen( 6, 0, 20, 79);
                              choice.ver_choice = 0;
                              break;
                          }
                          else
                          {
                              check = 0;
                              clearscreen( 6, 0, 20, 79);
                              choice.hor_choice = 0;
                              choice.ver_choice = 0;
                              return( choice);
                          }

    }
}
```

```
* ------------------------------------------------------------------------------*/
*                                                                                */
*  Module ..........Menubox()                                                    */
*  Function ........For the main menu option draws a box with the suboptions     */
*                   directly beneath the main menu option location,              */
*                   highlighting the first sub - option                          */

ENU *menubox( MENU *main_menu, MENU_CHOICE choice)

    unsigned char i, j, curr_len, max_len, items;
    unsigned char row1, col1, row2, col2;

    curr_len = max_len = items = 0;
    j = 1;

    clearscreen( 6, 0, 20, 79);
    for( i = 1; i <= main_menu->max_menu_items; i++)
       if( main_menu->menu_items[i].hor_item == choice.hor_choice)
       {
           items++;
           curr_len = strlen( main_menu->menu_items[i].item);
           if(curr_len > max_len)
               max_len = curr_len;
       }

    row1 = main_menu->hor_menu_pos + 3;
    col1 = main_menu->menu_items[choice.hor_choice].hor_item_pos - 2;
    row2 = main_menu->hor_menu_pos + 3 + items + 2;
    col2 = main_menu->menu_items[choice.hor_choice].hor_item_pos + max_len + 2;
```

```
box( row1, col1, row2, col2, 'd');

for( i = 1; i <= main_menu->max_menu_items; i++)
   if( main_menu->menu_items[i].hor_item == choice.hor_choice &&
        main_menu->menu_items[i].ver_item != 0)
   {
        strcpy( main_menu->box_items[j].item, main_menu->menu_items[i].item);
        j++;
   }

main_menu->box_items[choice.hor_choice].max_ver_items = j - 1;
main_menu->box_items[choice.hor_choice].ver_item_pos = row1 + 1;

for( i = 1; i <= main_menu->box_items[choice.hor_choice].max_ver_items; i++)
{
    setposition( row1 + 1 + i, col1 + 2);
    normalvideo( main_menu->box_items[i].item);
}

setposition( main_menu->box_items[choice.hor_choice].ver_item_pos + 1, col1 + 2);
reversevideo( main_menu->box_items[1].item);

return( main_menu);
```

DBFILE.H

```
/*                      Filename ........Dbfile.H                          */
/*                      --------------------------                         */


/*       This file contains the structure definitions for DBF files. The   */
/*    structures are those of the File Header, the Field Header and user -  */
/*    defined types for file handling in C, depending on the application    */
/*    and it's size. It also includes a Signature for Field Terminator.     */



typedef struct file_header
{
    char dbf_id;
    char last_update[3];
    long last_rec;
    unsigned data_offset;
    unsigned rec_size;
    char filler[20];
} FILE_HEAD;


typedef struct field_length
{
    union
    {
        unsigned char_len;
        struct
        {
            char len;
            char dec;
        } num_size;
     } len_info;
} FIELD_LENGTH;


typedef struct field_header
 {
   char field_name[11];
   char field_type;
   char dummy[4];
   FIELD_LENGTH field_length;
   char filler[14];
 } FIELD_HEAD;
```

```
typedef struct fields
{
    char buffer[FIELD_NAME_WIDTH];
    char data_type;
    FIELD_LENGTH length;
    char field_data[MAX_FIELD_WIDTH];
    union
    {
        struct dates
        {
            int day;
            int month;
            int year;
        } type_date;
        char type_string[ MAX_FIELD_WIDTH];
        int  type_numeric;
        char type_logical[1];
     } field_value;
} FIELDS;


typedef struct data_file
{
    FILE *file_pointer;
    char filename[12];
    FILE_HEAD file_header;
    FIELD_HEAD field_header;
    FIELDS fields[MAX_FIELDS];
    char record_buffer[MAX_REC_LENGTH];
    unsigned char no_of_fields;
    unsigned current_record;
    char eof_status;
    char delete_status;
    char locate_status;
} DBFILE;


typedef struct pack
{
    FILE *pack_ptr;
    long undel_rec;
} PKFILE;
```

DBFILE.C

```
/*                      Filename.......Dbfile.C                              */
/*                      -----------------------                              */


#include "dbfile.h"

/* ---------------------------------------------------------------------------- */
/*  Prototypes of functions used in Dbfile.C                                    */

DBFILE *usedbf( DBFILE *, char *);
DBFILE *init_data( DBFILE * );
DBFILE *null_data( DBFILE * );
DBFILE *skip_rec( DBFILE *, int);
DBFILE *goto_rec( DBFILE *, int);
DBFILE *gotop_file( DBFILE * );
DBFILE *gobottom_file( DBFILE * );
DBFILE *append_blank( DBFILE * );
DBFILE *replace_rec( DBFILE *, char *);
DBFILE *add_rec( DBFILE *, char *);
DBFILE *delete_rec( DBFILE *);
DBFILE *undelete_rec( DBFILE *);
DBFILE *get_rec( DBFILE *, char *, char *);

PKFILE *usepack( PKFILE *);

void pack1( DBFILE *, PKFILE *);
void closedbf( DBFILE * );
void closepack( PKFILE *);

char *strstrcpy( char *, int, int);


/* ---------------------------------------------------------------------------- */
/*  Module ..........Usedbf()                                                   */
/*  Function ........Opens a file                                               */

DBFILE *usedbf( DBFILE *dbfile, char *filename)
{
    unsigned char i;

    dbfile = (DBFILE *)malloc( sizeof(DBFILE) + 20);

    if(( dbfile->file_pointer = fopen( filename, "rb+" )) == NULL )
       return NULL;
    strcpy( dbfile->filename, filename);
    if( fread ( &dbfile->file_header, sizeof( FILE_HEAD ), 1,
                dbfile->file_pointer ) == 1 )
```

```
    {
        dbfile->no_of_fields  = (unsigned char)((  dbfile->file_header.
                        data_offset- sizeof(FILE_HEAD) ) / sizeof(FIELD_HEAD) );
        for( i = 0; i < dbfile->no_of_fields; i++ )
        {
            fread( &dbfile->field_header, sizeof( FIELD_HEAD), 1,
                    dbfile->file_pointer);
            strcpy( dbfile->fields[i].buffer, dbfile->field_header.field_name);
            dbfile->fields[i].data_type = dbfile->field_header.field_type;
            dbfile->fields[i].length = dbfile->field_header.field_length;
        }

       if( dbfile->file_header.last_rec )
        {
            fseek( dbfile->file_pointer, ( long)dbfile->file_header.data_offset,
                    SEEK_SET);
            fgets( dbfile->record_buffer, (int)dbfile->file_header.rec_size + 1,
                    dbfile->file_pointer);
            dbfile->current_record = 1;
            dbfile->eof_status = FALSE;
            dbfile = init_data( dbfile);
        }
        else
        {
            dbfile->current_record = 0;
            dbfile->eof_status = TRUE;
            dbfile = null_data( dbfile);
        }
    }

    return( dbfile);
}


/* ------------------------------------------------------------------------*/
/*                                                                         */
/*  Module ..........Init_data()                                           */
/*  Function .........Moves data from the buffer to the respective fields  */

DBFILE *init_data( DBFILE *dbfile)
{
    int i, j;
    char *cut, *dcut;

    for( i = 0, j = 1; i < dbfile->no_of_fields; i++ )
    {
        switch( dbfile->fields[i].data_type )
        {
            case 'C' :  cut = strstrcpy( dbfile->record_buffer, j ,
                        dbfile->fields[i].length.len_info.num_size.len +j- 1 );
```

```
                    strcpy( dbfile->fields[i].field_data, cut );
                    strcpy( dbfile->fields[i].field_value.type_string,
                            dbfile->fields[i].field_data);
                    break;

            case 'N'  : cut = strstrcpy( dbfile->record_buffer, j ,
                    dbfile->fields[i].length.len_info.num_size.len+ j - 1);
                    strcpy( dbfile->fields[i].field_data, cut );
                    sscanf(  dbfile->fields[i].field_data,  "%d",
                             &dbfile->fields[i].field_value.type_numeric);
                    break;

            case 'D'  : cut = strstrcpy( dbfile->record_buffer, j , j + 7);
                    strcpy( dbfile->fields[i].field_data, cut );
                    dcut = strstrcpy( dbfile->fields[i].field_data, 0, 3);
                    dbfile->fields[i].field_value.type_date.year=atoi(dcut);
                    dcut = strstrcpy( dbfile->fields[i].field_data, 4, 5);
                    dbfile->fields[i].field_value.type_date.month =
                            atoi(dcut);
                    dcut = strstrcpy( dbfile->fields[i].field_data, 6, 7);
                    dbfile->fields[i].field_value.type_date.day=atoi(dcut);
                    break;

            case 'L'  : cut = strstrcpy( dbfile->record_buffer, j , j );
                    strcpy( dbfile->fields[i].field_data, cut );
                    strcpy( dbfile->fields[i].field_value.type_logical,cut);
                    break;
        }
        j += dbfile->fields[i].length.len_info.num_size.len;
    }
    if( dbfile->record_buffer[0] == '*')
        dbfile->delete_status = TRUE;
    else
        dbfile->delete_status = FALSE;

    return ( dbfile );
}


/* ------------------------------------------------------------------------*/
/*                                                                         */
/*  Module ..........Null_data()                                           */
/*  Function ........Appends null to each field and the record buffer      */

DBFILE *null_data( DBFILE *dbfile)
{
    int i;

    for( i = 0; i < dbfile->no_of_fields; i++ )
    {
```

```
        dbfile->fields[i].field_data[0] = '\0';
    }
    dbfile->record_buffer[0] = '\0';

    return ( dbfile );
}



/*-------------------------------------------------------------------------*/
/*  Module ..........Skip_rec()                                            */
/*  Function ........File pointer skips specified number of records in the */
/*                   file                                                  */

DBFILE *skip_rec( DBFILE *dbfile, int val)
{
    if( (dbfile->current_record + (val)) <= dbfile->file_header.last_rec )
    {
        fseek(dbfile->file_pointer, ( long)dbfile->file_header.data_offset +
             (dbfile->current_record + (val-1)) * dbfile->file_header.rec_size,
              SEEK_SET);
        fgets( dbfile->record_buffer, (int)dbfile->file_header.rec_size + 1,
               dbfile->file_pointer);
        dbfile = init_data( dbfile);
        dbfile->current_record += (val);
    }
    else
    {
        dbfile->eof_status = TRUE;
        dbfile->current_record = 0;
        dbfile = null_data( dbfile);
    }

    return( dbfile);
}


/*-------------------------------------------------------------------------*/
/*  Module ..........Goto_rec()                                            */
/* Function .........Moves file pointer to the specified record number     */

DBFILE *goto_rec( DBFILE *dbfile, int val)
{
    if( val <= dbfile->file_header.last_rec)
    {
        fseek(dbfile->file_pointer, ( long)dbfile->file_header.data_offset +
             dbfile->file_header.rec_size * (val - 1), SEEK_SET);
        fgets( dbfile->record_buffer, (int)dbfile->file_header.rec_size + 1,
               dbfile->file_pointer);
```

```
        dbfile = init_data( dbfile);
        dbfile->current_record = val;
    }
    else
    {
        dbfile->eof_status = TRUE;
        dbfile->current_record = 0;
        dbfile = null_data( dbfile);
    }

    return( dbfile);
}

/* ------------------------------------------------------------------------*/
/*  Module ..........Gotop_file()                                          */
/*  Function ........Moves file pointer to the first record in the file    */

DBFILE *gotop_file( DBFILE *dbfile)
{
    fseek(dbfile->file_pointer, ( long)dbfile->file_header.data_offset,
          SEEK_SET);
    fgets( dbfile->record_buffer, (int)dbfile->file_header.rec_size + 1,
           dbfile->file_pointer);
    dbfile = init_data( dbfile);
    dbfile->current_record = 1;
    dbfile->eof_status = FALSE;

    return( dbfile);
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Gobottom_file()                                       */
/*  Function ........Moves file pointer to the last record in the file     */

DBFILE *gobottom_file( DBFILE *dbfile)
{
    fseek(dbfile->file_pointer, dbfile->file_header.data_offset +
         (( dbfile->file_header.last_rec - 1) * dbfile->file_header.rec_size),
          SEEK_SET);
    fgets( dbfile->record_buffer, (int)dbfile->file_header.rec_size + 1,
           dbfile->file_pointer);
    dbfile = init_data( dbfile);
    dbfile->current_record = dbfile->file_header.last_rec;
    dbfile->eof_status = FALSE;

    return( dbfile);
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Append_blank()                                        */
/*  Function ........Appends a blank record to the file                    */

DBFILE *append_blank( DBFILE *dbfile)
{
    int i;

    dbfile->file_header.last_rec++ ;
    rewind( dbfile->file_pointer);
    fwrite ( &dbfile->file_header, sizeof( FILE_HEAD ), 1,
              dbfile->file_pointer);
    fseek( dbfile->file_pointer, dbfile->file_header.data_offset +
         ((dbfile->file_header.last_rec) * dbfile->file_header.rec_size),
         SEEK_SET);
    for( i = 0; i < dbfile->file_header.rec_size; i++)
        dbfile->record_buffer[i] = ' ';
    dbfile->record_buffer[i] = '\0';
    fputs( dbfile->record_buffer, dbfile->file_pointer);

    dbfile->current_record = dbfile->file_header.last_rec + 1;

    return( dbfile);
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Replace_rec()                                         */
/*  Function ........Replaces the current record by the specified record   */

DBFILE *replace_rec( DBFILE *dbfile, char *record)
{
    int size;

    size = dbfile->file_header.rec_size;
    fseek( dbfile->file_pointer, ( long)dbfile->file_header.data_offset +
         ((dbfile->current_record  - 1) * dbfile->file_header.rec_size),
          SEEK_SET);
    fwrite( record, size, 1, dbfile->file_pointer);
    dbfile = init_data( dbfile);

    return( dbfile);
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Add_rec()                                             */
/*  Function ........Appends a record to the file                          */

DBFILE *add_rec( DBFILE *dbfile, char *record)
{
    dbfile = append_blank( dbfile);
    fseek( dbfile->file_pointer, dbfile->file_header.data_offset +
         ((dbfile->file_header.last_rec - 1) *  dbfile->file_header.rec_size),
           SEEK_SET);
    fputs( record, dbfile->file_pointer);
    dbfile = init_data( dbfile);

    return( dbfile);
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Delete_rec()                                          */
/*  Function ........Marks the current record for deletion                 */

DBFILE *delete_rec( DBFILE *dbfile)
{
    dbfile->record_buffer[0] = '*';
    dbfile = init_data( dbfile);
    dbfile = replace_rec( dbfile, dbfile->record_buffer);

    return( dbfile);
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Undelete_rec()
/*  Function ........Reactivates current record marked for deletion

DBFILE *undelete_rec( DBFILE *dbfile)
{
    dbfile->record_buffer[0] = ' ';
    dbfile = init_data( dbfile);
    dbfile = replace_rec(dbfile, dbfile->record_buffer);

    return( dbfile);
}
```

```
/* ------------------------------------------------------------------------*/
/* Module ..........Usepack()                                              */
/* Function ........Opens a file for packing records                       */

PKFILE *usepack( PKFILE *pkfile)
{
   pkfile = (PKFILE *)malloc( sizeof(PKFILE) + 10);

   if( (pkfile->pack_ptr = fopen( "packfile", "wb+")) == NULL)
       return NULL;

   pkfile->undel_rec = 0;

   return( pkfile);
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Pack1()                                               */
/*  Function.........Removes physically from file , all records marked for */
/*                   deletion                                              */

void pack1( DBFILE *dbfile, PKFILE *pkfile)
{
   char buff[MAX_REC_LENGTH];
   char *name_dbf;
   int i;

   strcpy( name_dbf, dbfile->filename);
   pkfile = usepack( pkfile);
   rewind( dbfile->file_pointer);
   while( (ftell( dbfile->file_pointer)) < dbfile->file_header.data_offset)
   {
     i = fgetc( dbfile->file_pointer);
     fputc( i, pkfile->pack_ptr);
   }

   dbfile = gotop_file( dbfile);
   while( !dbfile->eof_status)
   {
     if( !dbfile->delete_status)
      {
        fwrite( dbfile->record_buffer, dbfile->file_header.rec_size, 1,
                pkfile->pack_ptr);
        pkfile->undel_rec++;
      }
     dbfile = skip_rec( dbfile, 1);
   }
```

```
    rewind( pkfile->pack_ptr);
    dbfile->file_header.last_rec = pkfile->undel_rec++;
    fwrite( &dbfile->file_header, sizeof( FILE_HEAD), 1, pkfile->pack_ptr);
    closedbf( dbfile);
    closepack( pkfile);

    remove( name_dbf);
    rename( "packfile",name_dbf);
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Get_rec()                                             */
/*  Function ........Gets from file, record corresponding to a key field and  */
/*                   it's data value                                       */

DBFILE *get_rec( DBFILE *dbfile, char *key_field, char *data_string)
{
    unsigned char j;

    dbfile->locate_status = 0;
    dbfile = gotop_file( dbfile );
    while( !dbfile->eof_status )
    {
        for( j = 0; j < dbfile->no_of_fields; j++ )
        {
            if( !strcmp(dbfile->fields[j].buffer, strupr(key_field)) )
              if( !strcmp(dbfile->fields[j].field_data, data_string) )
              {
                  dbfile->locate_status = 1;
                  break;
              }
        }
        if( dbfile->locate_status )
            break;
        else
            skip_rec( dbfile, 1 );
    }

    return (dbfile);
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Closedbf()                                            */
/*  Function ........Closes an open file                                   */

void closedbf( DBFILE *dbfile)
{
    fclose( dbfile->file_pointer );
    free( dbfile);
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Closepack()                                           */
/*  Function ........Closes the file opened for the pack operation         */

void closepack(  PKFILE *pkfile)
{
    fclose( pkfile->pack_ptr);
    free( pkfile);
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Strstrcpy()                                           */
/*  Function ........Returns a substring of the given string               */

char *strstrcpy( char *str, int start, int end)
{
    int i, j;
    char *temp = "";

    if( start <= strlen( str))
       for( i = start, j = 0; i <= strlen( str) && i <= end; i++, j++ )
          temp[j] = str[i];
    temp[j] = '\0';

    return (temp);
}
```

FORM.H

```
/*                          Filename ........Form.H                          */
/*                          -----------------------                          */

/*   This file contains definitions of all structure types used in Form.C     */

typedef struct key_field

    char key_prompt[30];
    char key_field_name[FIELD_NAME_WIDTH];
    unsigned char key_field_width;
    char key_data_type;
    union
    {
        char key_string[ MAX_FIELD_WIDTH];
        char key_date[8];
        char key_logical[1];
        char key_numeric[10];
    } key_data_buffer;
    unsigned char key_row;
    unsigned char key_col;
    char key_format[10];
    char key_caps_status;
    int key_field_value;
} KEY_FIELD;


typedef struct key_form
{
    KEY_FIELD data_entry[NUM_FIELDS];
    unsigned char current_field;
    unsigned char last_field;
    unsigned char exit_status;
    struct coordinates
    {
        unsigned char form_row1;
        unsigned char form_row2;
        unsigned char form_col1;
        unsigned char form_col2;
    } form_coordinates;
    struct form_header
    {
        unsigned char header_row;
        unsigned char header_col;
        char header_string[40];
     } form_header;
     char form_prompt[10][10];
     unsigned char form_key_value;
     unsigned char read_only_status;
} KEY_FORM;
```

FORM. C

```
/*                                  Filename ........Form.C                  */
/*                                  ------------------------                 */


#include "form.h"


/* ------------------------------------------------------------------------- */
/*  Prototypes of Functions used in Form.C                                   */


KEY_FIELD init_field( KEY_FIELD , char *, unsigned char, char, unsigned char,
                      unsigned char, char *, char, char *);
KEY_FIELD read_field( KEY_FIELD );
KEY_FORM *mem_form_alloc( KEY_FORM *);
KEY_FORM *init_key_form( KEY_FORM *, char *, unsigned char , unsigned char ,
                         unsigned char ,unsigned char , unsigned char);
KEY_FORM *form_process( KEY_FORM *);
KEY_FORM *read_key_form( KEY_FORM *, unsigned char , unsigned char,
                         unsigned char , unsigned char);

void title( void);
void clear_form( KEY_FORM *);
void insert( char *, unsigned char, int);
void delete( char *, unsigned char );



/* ------------------------------------------------------------------------- */
/*  Module ..........Init_field()                                            */
/*  Function ........Initializes each field of the screen entry form and     */
/*                   displays the prompt on the screen                       */

KEY_FIELD init_field( KEY_FIELD key_field, char *prompt, unsigned char width,
                      char data_type, unsigned char row, unsigned char col,
                      char *format, char caps, char *field_name)
{
    unsigned char i;

    strcpy( key_field.key_prompt, prompt);
    key_field.key_field_width = width;
    key_field.key_data_type = data_type;
    key_field.key_row = row;
    key_field.key_col = col;
    strcpy( key_field.key_format, format);
    key_field.key_caps_status = caps;
```

```
    key_field.key_field_value = 0;
    strcpy( key_field.key_field_name, strupr( field_name));

    setposition( key_field.key_row, key_field.key_col);
    printf( "%s", key_field.key_prompt );

    switch( key_field.key_data_type)
    {
        case 'C' : for( i = 0; i < width; i++)
                       key_field.key_data_buffer.key_string[i] = ' ';
                   key_field.key_data_buffer.key_string[i] = '\0';
                   break;

        case 'D' : strcpy( key_field.key_data_buffer.key_date, "        ");
                   break;

        case 'N' : for( i = 0; i < width; i++ )
                      key_field.key_data_buffer.key_numeric[i] = ' ';
                   key_field.key_data_buffer.key_numeric[i] = '\0';
                   break;

        case 'L' : strcpy( key_field.key_data_buffer.key_logical, "N");
                   break;
    }

    return( key_field);
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Read_field()                                          */
/*  Function ........Reads the data entered by the user on the screen form and*/
/*                   moves it to the respective fields                     */

KEY_FIELD read_field( KEY_FIELD key_field)
{
    int ch, i;
    unsigned char col, flag, k, ins_status, del_status;

    ins_status = del_status = 0;
    col = key_field.key_col + strlen( key_field.key_prompt);
    i = 0;
    flag = 0;

    setposition( key_field.key_row, key_field.key_col +
                 strlen( key_field.key_prompt));
    while(1)
    {
        ch = getcharacter();
```

```
switch( ch)
{
    case RT_ARR : if( col < (key_field.key_col +
                      strlen( key_field.key_prompt) +
                      key_field.key_field_width))
                  {
                      col +=1;
                      i += 1;
                      setposition( key_field.key_row, col);
                  }
                  else
                      putchar(7);
                  break;

    case LT_ARR : if( col > key_field.key_col +
                      strlen( key_field.key_prompt))
                  {
                      col -=1;
                      i -= 1;
                      setposition( key_field.key_row, col);
                  }
                  else
                      putchar(7);
                  break;

    case ESC    : switch( key_field.key_data_type)
                  {
                      case 'C' : key_field.key_field_value = ch;
                                 return( key_field);

                      case 'D' : key_field.key_field_value = ch;
                                 return( key_field);

                      case 'N' : key_field.key_field_value = ch;
                                 return( key_field);

                      case 'L' : key_field.key_field_value = ch;
                                 return( key_field);
                  }
                  break;

   case ENTER   : switch( key_field.key_data_type)
                  {
                      case 'C' : key_field.key_data_buffer.key_string[key_
                                 key_field_width] = '\0';
                                 key_field.key_field_value = ch;
                                 return( key_field);

                      case 'D' : key_field.key_data_buffer.key_date[key_fi
                                 key_field_width] = '\0';
```

```
                                key_field.key_field_value = ch;
                                return( key_field);

                        case 'N' : key_field.key_data_buffer.key_numeric[key_field.
                                key_field_width] = '\0';
                                key_field.key_field_value = ch;
                                return( key_field);

                        case 'L' : key_field.key_data_buffer.key_logical[key_field.
                                key_field_width] = '\0';
                                key_field.key_field_value = ch;
                                return( key_field);
                    }
                    break;

    case UP_ARR     : key_field.key_field_value = ch;
                      return( key_field);

    case DN_ARR     : key_field.key_field_value = ch;
                      return( key_field);

    case INS        : ins_status ^= 1;
                      break;

    case DEL        : flag = 1;
                      del_status = 1;
                      delete( key_field.key_data_buffer.key_string, i);
                      break;

    default         : switch( key_field.key_data_type)
                      {
                          case 'C' : if( isprint( ch) && col < (key_field.
                                     key_col + strlen( key_field.key_prompt)
                                     + key_field.key_field_width))
                                     {
                                         flag = 1;
                                         if( !ins_status )
                                             key_field.key_data_buffer.
                                             key_string[i] = ch;
                                         else
                                             insert( key_field.key_data_buffer.
                                                     key_string, i, ch);
                                     }
                                     else
                                         putchar(7);
                                     break;

                           case 'D' : if( isdigit( ch) && col < (key_field.
                                      key_col + strlen( key_field.key_prompt)
                                      + key_field.key_field_width))
```

```
                                {
                                    flag = 1;
                                    if( !ins_status )
                                        key_field.key_data_buffer.
                                        key_date[i] = ch;
                                    else
                                        insert( key_field.key_data_buffer.
                                                key_date, i, ch);
                                }
                                else
                                    putchar(7);
                                break;

                        case 'N'  : if( isdigit( ch) && col < (key_field.
                                    key_col + strlen( key_field.key_prompt)
                                    + key_field.key_field_width))
                                    {
                                        flag = 1;
                                        if( !ins_status )
                                            key_field.key_data_buffer.
                                            key_numeric[i] = ch;
                                        else
                                            insert( key_field.key_data_buffer.
                                                    key_numeric, i, ch);
                                    }
                                    else
                                        putchar(7);
                                    break;

                        case 'L '  : if( ch == 'N' || ch == 'n' || ch == 'Y'
                                    || ch == 'y' && col < (key_field.key_col
                                     + strlen( key_field.key_prompt) + key_field.
                                    {
                                        flag = 1;
                                        key_field.key_data_buffer.
                                        key_logical[i] = ch;
                                    }
                                    else
                                        putchar(7);
                                    break;
                    }
                    break;
}
if( flag )
{
    if( !ins_status && !del_status)
    {
        col++;
        i = col - (key_field.key_col + strlen( key_field.key_prompt));
        putchar( ch);
```

```
            }
            if( ins_status)
            {
                col++;
                i = col - (key_field.key_col + strlen( key_field.key_prompt));
                setposition( key_field.key_row, key_field.key_col +
                             strlen( key_field.key_prompt) );
                printf("%s", key_field.key_data_buffer.key_string);
                setposition( key_field.key_row, key_field.key_col +
                             strlen( key_field.key_prompt) + i );
            }
            if( del_status)
            {
                setposition( key_field.key_row, key_field.key_col +
                             strlen( key_field.key_prompt) );
                printf("%s", key_field.key_data_buffer.key_string);
                setposition( key_field.key_row, key_field.key_col +
                             strlen( key_field.key_prompt) + i);
                del_status = 0;
            }
        }
        flag = 0;
    }
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Mem_form_alloc()                                      */
/*  Function ........Allocates memory ( = sizeof( KEY_FORM) ) dynamically from*/
/*                   the heap                                              */

KEY_FORM *mem_form_alloc( KEY_FORM *key_form)
{
    if( (key_form = malloc( sizeof( KEY_FORM))) != NULL)
        return( key_form);
}


/*-------------------------------------------------------------------------*/
/*  Module ..........Init_key_form()                                       */
/*  Function ........Initializes the screen entry form, draws the form on the  */
/*                   console and displays the menu prompts                 */

KEY_FORM  *init_key_form( KEY_FORM *key_form, char *title,
                          unsigned char last_field, unsigned char row1,
                          unsigned char col1, unsigned char row2,
                          unsigned char col2)

{
```

```
unsigned char i, len, j, curr_col;



int ch;
struct date today;

key_form->current_field = 1;
key_form->read_only_status = 0;
key_form->last_field = last_field;
strcpy( key_form->form_prompt[0], "");
strcpy( key_form->form_prompt[1], " Add  ");
strcpy( key_form->form_prompt[2], " Edit ");
strcpy( key_form->form_prompt[3], " Del  ");
strcpy( key_form->form_prompt[4], " View ");
strcpy( key_form->form_prompt[5], " Find ");
strcpy( key_form->form_prompt[6], " Next ");
strcpy( key_form->form_prompt[7], " Top  ");
strcpy( key_form->form_prompt[8], " Last ");
strcpy( key_form->form_prompt[9], "");

_settextcolor((short) 7);
box( 1, 1, 4, 79, 'd');
_settextcolor((short) 15);
setposition( 2, 41 - strlen("Leave Management System   ")/2);
cputs("Leave Management System");
_settextcolor((long) 7);
setposition( 3, 41 - strlen("Sibcom Technologies Pvt. Ltd.   ")/2);
printf("Sibcom Technologies Pvt. Ltd.");
getdate( &today);
setposition( 2, 69);

printf("%02d/%02d/%02d", today.da_day, today.da_mon, today.da_year);

len = ((col2 - col1) - strlen( title))/2;
setposition( row1, len);
_settextcolor((short) 15);
cputs( title);
_settextcolor((short) 7);

ver_line( row1 + 1, row2 - 1, col1, 'd');
ver_line( row1 + 1, row2 - 1, col2, 'd');
hor_line( row2, col1 + 1, col2 - 1, 'd');
hor_line( row1, col1 + 1, len - 2, 'd');
hor_line( row1, len + strlen( title) + 1, col2 - 1, 'd');
setposition( row1, col1);
putch(dtlc);
setposition( row1, col2 );
putch(dtrc);
setposition( row2, col1);
putch(dblc);
setposition( row2, col2);
putch(dbrc);
```

```
    setposition( row1, len - 1);



    putch( dtrc);
    setposition( row1, len + strlen( title));
    putch( dtlc);

    setposition( row2 - 2, col1);
    putch( dlc);
    setposition( row2 - 2, col2);
    putch( drc);
    hor_line( row2 - 2, col1 + 1, col2 - 1, 's');

    setposition( row2 - 1, 8);
    for( i = 1; i < 9; i++)
    {
        reversevideo( key_form->form_prompt[i]);
        printf(" ");
    }
    key_form->form_key_value = 0;

    return( key_form);
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Read_key_form()                                       */
/*  Function ........Reads menu options on the screen form and stores the  */
/*                   option selected                                       */

KEY_FORM *read_key_form( KEY_FORM *key_form,
                         unsigned char row1, unsigned char col1,
                         unsigned char row2, unsigned char col2)
{
     unsigned char i, len, j, curr_col;
     int ch;

     setposition( row2 - 1, (key_form->form_key_value * 7) + 1);
     normalvideo( key_form->form_prompt[ key_form->form_key_value]);
     curr_col = (key_form->form_key_value * 7) + 1;

     j = 1;
     i = 1;
     while( ch != ESC)
     {
         ch = getcharacter();
         setposition( row2 - 1, curr_col);
         switch( ch)
         {
             case RT_ARR : if( curr_col > 56)
```

```
                {

                     reversevideo( key_form->form_prompt[j]);
                     j = 1;
                     curr_col = 8;
                }
                else
                {
                     reversevideo( key_form->form_prompt[j]);
                     curr_col = curr_col + 7;
                     j++;
                }
                setposition( row2 - 1, curr_col);
                normalvideo( key_form->form_prompt[j]);
                break;

case LT_ARR :   if( curr_col < 9)
                {
                     reversevideo( key_form->form_prompt[j]);
                     j = 8;
                     curr_col = 57;
                }
                else
                {
                     reversevideo( key_form->form_prompt[j]);
                     j --;
                     curr_col = curr_col - 7;
                }
                setposition( row2 - 1, curr_col);
                normalvideo( key_form->form_prompt[j]);
                break;

case ENTER  :   key_form->form_key_value = j;
                return( key_form);

case HOME   :   if( curr_col == 8)
                    putchar(7);
                else
                {
                    reversevideo( key_form->form_prompt[j]);
                    j = 1;
                    curr_col = 8;
                    setposition( row2 - 1, curr_col);
                    normalvideo( key_form->form_prompt[j]);
                }
                break;

 case END   :   if( curr_col == 57)
                    putchar(7);
                else
                {
```

```
                        reversevideo( key_form->form_prompt[j]);



                        j = 8;
                        curr_col = 57;
                        setposition( row2 - 1, curr_col);
                        normalvideo( key_form->form_prompt[j]);
                    }
                    break;
            }
        }
        key_form->form_key_value = 0;
        return( key_form);
}


/* ------------------------------------------------------------------------*/
/*  Module ..........Form_process()                                        */
/*  Function ........Call upon various functions to carry out processing on an*/
/*                   input form                                            */

KEY_FORM *form_process( KEY_FORM *form_lev_mas)
{
   unsigned char i;
   int ch, ch1;


   while( 1)
   {
      ch = ch1 = 0;
      form_lev_mas->form_key_value = 1;
      form_lev_mas->current_field = 0;

      form_lev_mas->data_entry[form_lev_mas->last_field].key_field_value = 0;

      while( form_lev_mas->data_entry[form_lev_mas->current_field].
             key_field_value != ESC)
      {
          if( form_lev_mas->data_entry[form_lev_mas->last_field].
              key_field_value == ENTER )
            break;
          switch( form_lev_mas->data_entry[form_lev_mas->current_field].
                  key_field_value )
          {
             case UP_ARR : if( form_lev_mas->current_field == 1 )
                             putchar(7);
                           else
                             form_lev_mas->current_field--;
                           break;

             case DN_ARR : if( form_lev_mas->current_field ==
```

```
                        form_lev_mas->last_field )

                          putchar(7);
                        else
                          form_lev_mas->current_field++;
                        break;

          case ENTER  : if( form_lev_mas->current_field <
                            form_lev_mas->last_field )
                          form_lev_mas->current_field++;
                        else
                          form_lev_mas->data_entry[form_lev_mas->current_field]
                          key_field_value = DONE;
                        break;
        }
        if( !form_lev_mas->read_only_status)
            form_lev_mas->data_entry[form_lev_mas->current_field] =
            read_field( form_lev_mas->data_entry[form_lev_mas->current_field]);
        else
        {
            confirm( "Press ESC to Exit", 24, 27, 27);
            return( form_lev_mas);
        }
    }
    if( form_lev_mas->data_entry[form_lev_mas->current_field].key_field_value
        == ESC)
    {
       ch = confirm( "Do You want to Exit ( Y/N ) :", 24, 78, 89 );
       if( ch == 89)
       {
          clearscreen( 24, 0, 24, 79);
          form_lev_mas->exit_status = 0;
          return( form_lev_mas);
       }
    }
    else
    {
       ch1 = confirm( "Are the Entries O.K. ( Y/N ) :", 24, 78, 89 );
       if( ch1 == 89)
       {
          clearscreen( 24, 0, 24, 79);
          form_lev_mas->exit_status = 1;
          return( form_lev_mas);
       }
    }
  }
}
```

```
/* ------------------------------------------------------------------------*/
/*  Module ..........Clear_form()                                          */
/*  Function ........Clears memory allocated for form                      */

void clear_form( KEY_FORM *key_form)
{
    free( key_form);
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Insert()                                              */
/*  Function ........Inserts a character in a string on the console        */

void insert( char *str, unsigned char position, int ch)
{
    unsigned char i, j, len;
    char *buffer;

    len = strlen( str);
    strcpy( buffer, str);
    str[position] = ch;
    for( i = position + 1, j = position; buffer[j] != '\0' && i != len; i++, j++)
        str[i] = buffer[j];
    str[i+1] = '\0';
}



/* ------------------------------------------------------------------------*/
/*  Module ..........Delete()                                              */
/*  Function ........Deletes a character in a string on the console        */

void delete( char *str, unsigned char position)
{
    unsigned char i, j, len;
    char *buffer, *temp;

    len = strlen( str);
    strcpy( buffer, str);
    for( i = 0; i < len; i++)
        temp[i] = ' ';
    temp[i + 1] = '\0';

    for( i = 0; i < position; i++)
        temp[i] = buffer[i];
```

```
    for( i = position, j = position + 1; buffer[j] != '\0'; i++, j++)
        temp[i] = buffer[j];
    temp[i+1] = '\0';
    for( i = 0; temp[i] != '\0'; i++)
        str[i] = temp[i];
    str[i + 1] = '\0';
}
```

```
/*                              Filename ........Lms.C                          */
/*                              ----------------------                          */


/* This is the main program file of the Leave Management System. The actual    */
/* processing according to the requirements of the package is done here, which*/
/* calls upon the libraries of functions developed.                            */


#include "include.h"
#include "define.h"
#include "utility.c"
#include "menu.c"
#include "form.c"
#include "dbfile.c"

/* ---------------------------------------------------------------------------*/
/* Prototypes of functions used in Lms.C                                       */

DBFILE *emp_mas(  DBFILE *);
DBFILE *lev_mas(  DBFILE *);
DBFILE *process(  DBFILE *);

KEY_FORM *lev_mas_prompts( KEY_FORM *);
KEY_FORM *emp_mas_prompts( KEY_FORM *);
KEY_FORM *process_prompts( KEY_FORM *);
KEY_FORM *moveto_form( DBFILE *, KEY_FORM * );

char *moveto_dbfile( KEY_FORM *);


/* ---------------------------------------------------------------------------*/
/* Module ..........Main()                                                     */
/* Function ........Main body of the program for the Leave Management System  */

main()
{
    unsigned char i;
    int ch;

    FILE *ini_file;
    DBFILE *dbf_emp_mas, *dbf_lev_mas, *dbf_process;
    PKFILE *pkfile;
    MENU *main_menu;
    MENU_CHOICE choice;
    char *scr;
```

```
char *p_buff, *d_buff;
unsigned char len;
struct dosdate_t date;


struct passwd
{
    unsigned char set;
    char password[15];
    char date_last_accessed[10];
    char filler[8];
}init;


main_menu = ( MENU *)malloc( sizeof( MENU ) + 20);
scr = (char *)malloc(4020);
clearscreen( 0, 0, 24, 80);

choice.hor_choice = 1;
choice.ver_choice = 0;

title1(1);
main_menu = init_main_menu( main_menu, 6, 16, 4);


main_menu = init_menu_items( main_menu, "Master", 1, 0);
main_menu = init_menu_items( main_menu, "Processing", 2, 0);
main_menu = init_menu_items( main_menu, "Query", 3, 0);
main_menu = init_menu_items( main_menu, "Reports", 4, 0);
main_menu = init_menu_items( main_menu, "Utilities", 5, 0);
main_menu = init_menu_items( main_menu, "Quit", 6, 0);

main_menu = init_menu_items( main_menu, "Leave Master Record      ", 1, 1);
main_menu = init_menu_items( main_menu, "Employee Personal Record ", 1, 2);
main_menu = init_menu_items( main_menu, "Employee Leave Record    ", 2, 1);
main_menu = init_menu_items( main_menu, "LWP Employees ", 3, 1);
main_menu = init_menu_items( main_menu, "Employee Leave Details    ", 4, 1);
main_menu = init_menu_items( main_menu, "Employee Personal Details ", 4, 2);
main_menu = init_menu_items( main_menu, "LWP Employees             ", 4, 3);
main_menu = init_menu_items( main_menu, "Remove Deleted Records", 5, 1);
main_menu = init_menu_items( main_menu, "Undel Personal Records", 5, 2);
main_menu = init_menu_items( main_menu, "Undel Emp-Lev Records ", 5, 3);
main_menu = init_menu_items( main_menu, "Undel Lev-Mas Records ", 5, 4);
main_menu = init_menu_items( main_menu, "Change Password       ", 5, 5);
main_menu = init_menu_items( main_menu, "Take Backups          ", 5, 6);
main_menu = init_menu_items( main_menu, "  Yes  ", 6, 1);
main_menu = init_menu_items( main_menu, "  No   ", 6, 2);

main_menu = print_main_menu( main_menu);
```

```
while( choice.hor_choice != 0)
{
    choice = read_main_menu( main_menu, choice);
    scr = savescr( scr);
    clearscreen( 0, 0, 24, 79);
    switch( choice.hor_choice)
    {
        case 1  : switch( choice.ver_choice)
                  {
                      case 1  :  dbf_lev_mas = usedbf( dbf_lev_mas,
                                                "lev_mas.dbf");
                                 dbf_lev_mas = lev_mas(  dbf_lev_mas);
                                 closedbf( dbf_lev_mas);
                                 restorescr( scr);
                                 break;

                      case 2  :  dbf_emp_mas = usedbf( dbf_emp_mas,
                                                "emp_mas.dbf");
                                 dbf_emp_mas = emp_mas(  dbf_emp_mas);
                                 closedbf( dbf_emp_mas);
                                 restorescr( scr);
                                 break;
                  }
                  break;

        case 2  : switch( choice.ver_choice)
                  {
                      case 1  :  dbf_process = usedbf( dbf_process,
                                                "process.dbf");
                                 dbf_process = process( dbf_process);
                                 closedbf( dbf_process);
                                 restorescr( scr);
                                 break;
                  }
                  break;

        case 3  : switch( choice.ver_choice)
                  {
                      case 1  :  query1();
                                 break;

                      case 2  :  query2();
                                 break;
                  }
                  break;

        case 4  : switch( choice.ver_choice)
                  {
                      case 1  :  rep_emp_leave();
                                 break;
```

```
                case 2  :   rep_emp_person();
                            break;
            }
            break;

case 5  : switch( choice.ver_choice)
            {
                 case 1  : restorescr( scr);
                           confirm( "Are you sure ( Y/N ) : ", 18,
                                    89, 89);
                           clearscreen( 18, 18, 0, 79);
                           setposition( 18, 40 - ( strlen(
                                       "  Working ...")/2));
                           printf("Working ...");
                           dbf_emp_mas = usedbf( dbf_emp_mas,
                                                 "emp_mas.dbf");
                           pack1( dbf_emp_mas, pkfile);
                           dbf_lev_mas = usedbf( dbf_lev_mas,
                                                 "lev_mas.dbf");
                           pack1( dbf_lev_mas, pkfile);
                           restorescr( scr);
                           break;

                 case 2  : restorescr( scr);
                           dbf_emp_mas = usedbf( dbf_emp_mas,
                                                 "emp_mas.dbf");
                           while( !dbf_emp_mas->eof_status)
                           {
                               if( dbf_emp_mas->delete_status)
                                   dbf_emp_mas = undelete_rec(
                                                 dbf_emp_mas);
                               dbf_emp_mas = skip_rec( dbf_emp_mas,1);
                           }
                           closedbf( dbf_emp_mas);
                           break;

                 case 3  : dbf_process = usedbf( dbf_process,
                                                 "process.dbf");
                           while( !dbfprocess->eof_status)
                           {
                               if( dbf_process->delete_status)
                                   dbf_process = undelete_rec(
                                                  dbf_process);
                               dbf_process = skip_rec( dbf_process, 1);
                           }
                           closedbf( dbf_process);
                           restorescr( scr);
                           break;
```

```
case 4  : restorescr( scr);
          dbf_lev_mas = usedbf( dbf_lev_mas,
                                "lev_mas.dbf");
          while( !dbf_lev_mas->eof_status)
          {
              if( dbf_lev_mas->delete_status)
                  dbf_lev_mas = undelete_rec(
                                dbf_lev_mas);
              dbf_lev_mas = skip_rec( dbf_lev_mas, 1);
          }
          closedbf( dbf_lev_mas);
          break;

case 5  : restorescr( scr);
          init.set = 99;
          if(( ini_file = fopen( "prj.ini", "wb+"))
              == NULL)
          {
              confirm( "Error in creating Password",
                       18, 13, 27);
              clearscreen( 18, 0, 18, 79);
              break;
          }
          else
          {
              setposition( 18, 25);
              printf("Enter  System  Password");
              len = 0;
              ch = 0;
              while( ch != 13)
              {
                   ch = getcharacter();
                   p_buff[ len] = ch;
                   len++;
                   if( len == 15)
                       putchar( 7);
              }
              p_buff[ len] = '\0';
              clearscreen( 18, 0, 18, 79);
              ch = confirm( "Do you wish to continue
                            ( Y/N ) : ", 18, 78, 89);
              clearscreen( 18, 0, 18, 79);
                     if( ch == 78)
                  break;
              setposition( 18, 35);
              printf("Working ...");
              _dos_getdate( &date);
              sprintf( d_buff, "%02d/%02d/%02d",
                    date.day, date.month, date.year);
              strcpy( init.password, p_buff);
```

```
                    strcpy( init.date_last_accessed, d_buff);
                    fwrite( &init, sizeof( init), 1, ini_file);
                    fclose( ini_file);
                    clearscreen( 18, 0, 18, 79);
                }
                break;

case 5  : restorescr( scr);
                if(( ini_file = fopen("prj.ini", "rb+"))
                    == NULL)
                {
                    confirm( "Error in searching for old
                              password", 18, 13, 27);
                    clearscreen( 18, 0, 18, 79);
                    break;
                }
                if( !( fread( &init, sizeof( init), 1,
                   ini_file)))
                {
                    confirm( "Error in searching for old
                              password", 18, 13, 27);
                    clearscreen( 18, 0, 18, 79);
                    break;
                }
                ch = 0;
                len = 0;
                setposition( 18, 25);
                printf("Enter  Old  Password");
                while( ch != 13)
                {
                     ch = getcharacter();
                     p_buff[ len] = ch;
                     len++;
                     if( len == 15)
                        putchar( 7);
                }
                p_buff[ len] = '\0';
                clearscreen( 18, 0, 18, 79);
                ch = confirm( "Do you wish to continue
                                ( Y/N ) : ", 18, 78, 89);
                       if( ch == 78)
                   break;
                clearscreen( 18, 0, 18, 79);
                setposition( 18, 35);
                printf("Checking Old Password ...");
                if( strcmp( p_buff, init.password))
                {
                    clearscreen( 18, 0, 18, 79);
                    confirm( "Invalid Password", 18, 13, 27);
                          clearscreen( 18, 0, 18, 79);
```

```
                    break;
                }
                fclose( ini_file);
                if(( ini_file = fopen( "prj.ini", "wb+"))
                    == NULL)
                {
                    confirm( "Error in creating Password",
                             18, 13, 27);
                    clearscreen( 18, 0, 18, 79);
                    break;
                }
                else
                {
                    setposition( 18, 25);
                    printf("Enter  New  Password");
                    len = 0;
                    ch = 0;
                    while( ch != 13)
                    {
                         ch = getcharacter();
                         p_buff[ len] = ch;
                         len++;
                         if( len == 15)
                             putchar( 7);
                    }
                    p_buff[ len] = '\0';
                    clearscreen( 18, 0, 18, 79);
                    ch = confirm( "Do you wish to continue
                                   ( Y/N ) : ", 18, 78, 89);

                   if( ch == 78)
                       break;
                    clearscreen( 18, 0, 79, 18);
                    setposition( 18, 35);
                    printf("Working ...");

                    _dos_getdate( &date);
                    sprintf( d_buff, "%02d/%02d/%02d",
                             date.day, date.month, date.year);
                    strcpy( init.password, p_buff);
                    strcpy( init.date_last_accessed, d_buff);
                    fwrite( &init, sizeof( init), 1, ini_file);
                    fclose( ini_file);
                    clearscreen( 18, 0, 18, 79);
                }
                break;
 }
break;
```

```
            case 6  : switch( choice.ver_choice)
                      {
                           case 1  : restorescr( scr);
                                     ch = confirm( "Are you sure ( Y/N ) : ",
                                                   18, 78, 89);
                                     clearscreen( 18, 0, 18, 79);
                                     if( ch == 89)
                                     {
                                         clearscreen( 0, 0, 24, 79);
                                         choice.hor_choice = 0;
                                     }
                                     break;

                           case 2  : restorescr( scr);
                                     break;
                      }
                      break;
        }
    }

    free( scr);
}


/* ------------------------------------------------------------------------*/
/* Module ..........Lev_mas()                                              */
/* Function ........Performs operations on the Leave Master File           */

DBFILE *lev_mas( DBFILE *dbfile)
{
    KEY_FORM *form_lev_mas;
    int ch;
    unsigned char prompt_status;
    char *str;

    prompt_status = 0;
    form_lev_mas->form_key_value = 1;
    form_lev_mas = init_key_form( form_lev_mas, " Master Leave Record ", 11, 5, 1,
                                  23, 79);
    form_lev_mas = lev_mas_prompts( form_lev_mas);
    while( form_lev_mas->form_key_value != 0)
    {
        if( prompt_status )
           form_lev_mas = lev_mas_prompts( form_lev_mas);
        prompt_status = 0;
        form_lev_mas = read_key_form( form_lev_mas, 5, 1, 23, 79);
        if( form_lev_mas->form_key_value < 6)
        {
            while(1)
```

```
{
    if( prompt_status )
        form_lev_mas = lev_mas_prompts( form_lev_mas);
    prompt_status = 0;
    if( !form_lev_mas->form_key_value )
        break;

    form_lev_mas->data_entry[0] = read_field( form_lev_mas->
                                              data_entry[0]);
    if( form_lev_mas->data_entry[0].key_field_value == ESC)
        break;

    if( is_space( form_lev_mas->data_entry[0].key_data_buffer.
                  key_string ))
        continue;

    else
    {
         prompt_status = 1;
         switch( form_lev_mas->form_key_value)
         {
             case 1  : dbfile = get_rec( dbfile, "LEV_CODE",
                                form_lev_mas->data_entry[0].
                                key_data_buffer.key_string );
                       if( dbfile->locate_status)
                       {
                           confirm( "Record  already  exists", 24,
                                    13, 27);
                           clearscreen( 24, 0, 24, 79);
                           prompt_status = 0;
                           break;
                       }
                       form_lev_mas->read_only_status = 0;
                       form_lev_mas = form_process( form_lev_mas);
                       if( form_lev_mas->exit_status )
                       {
                           str = moveto_dbfile( form_lev_mas);
                           dbfile = add_rec( dbfile, str);
                       }
                       break;


             case 2  : dbfile = get_rec( dbfile, "LEV_CODE",
                                form_lev_mas->data_entry[0].
                                key_data_buffer.key_string );
                       if( !dbfile->locate_status)
                       {
                           confirm( "Record  does  not  exist", 24,
                                    13, 27);
                           clearscreen( 24, 0, 24, 79);
```

```
                    break;
                }
                if( dbfile->delete_status)
                {
                    confirm( "Record has been Deleted", 24,
                             13, 27);
                    clearscreen( 24, 0, 24, 79);
                    break;
                }
                form_lev_mas = moveto_form( dbfile, form_lev_mas);
                form_lev_mas->read_only_status = 0;
                form_lev_mas = form_process( form_lev_mas);
                if( form_lev_mas->exit_status )
                {
                    str = moveto_dbfile( form_lev_mas);
                    dbfile = replace_rec( dbfile, str);
                }
                break;

        case 3  : dbfile = get_rec( dbfile, "LEV_CODE",
                            form_lev_mas->data_entry[0].
                            key_data_buffer.key_string);
                if( !dbfile->locate_status)
                {
                    confirm( "Record  does  not  exist", 24,
                             13, 27);
                    clearscreen( 24, 0, 24, 79);
                    break;
                }
                if( dbfile->delete_status)
                {
                    confirm( "Record has been Deleted", 24,
                             13, 27);
                    clearscreen( 24, 0, 24, 79);
                    break;
                }
                form_lev_mas = moveto_form( dbfile, form_lev_mas);
                ch = confirm( "Press DEL to delete, ESC to abort",
                              24, 211, 27);
                clearscreen( 24, 0, 24, 79);
                if( ch == DEL)
                    dbfile = delete_rec( dbfile);
                break;

        case 4  : dbfile = get_rec( dbfile, "LEV_CODE",
                            form_lev_mas->data_entry[0].
                            key_data_buffer.key_string);
                if( !dbfile->locate_status)
                {
                    confirm( "Record  does  not  exist", 24,
```

```
                                        13, 27);
                                clearscreen( 24, 0, 24, 79);
                                break;
                            }
                            if( dbfile->delete_status)
                            {
                                confirm( "Record has been Deleted", 24,
                                        13, 27);
                                clearscreen( 24, 0, 24, 79);
                                break;
                            }
                            form_lev_mas = moveto_form( dbfile, form_lev_mas);
                            form_lev_mas->read_only_status = 1;
                            form_lev_mas = form_process( form_lev_mas);
                            break;

                    case 5  : dbfile = get_rec( dbfile, "LEV_CODE",
                                        form_lev_mas->data_entry[0].
                                        key_data_buffer.key_string);
                            if( !dbfile->locate_status)
                            {
                                confirm( "Record  does  not  exist", 24,
                                        13, 27);
                                clearscreen( 24, 0, 24, 79);
                                break;
                            }
                            form_lev_mas = moveto_form( dbfile, form_lev_mas);
                            form_lev_mas->read_only_status = 1;
                            form_lev_mas = form_process( form_lev_mas);
                            break;
                }
            }
        }
    }
}
else
{
    switch( form_lev_mas->form_key_value)
    {
        case 6  : if( !dbfile->eof_status)
                  {
                    dbfile = skip_rec( dbfile, 1);
                    strcpy( form_lev_mas->data_entry[0].key_data_buffer.
                            key_string, dbfile->fields[0].field_data );
                    setposition( form_lev_mas->data_entry[0].key_row,
                            form_lev_mas->data_entry[0].key_col +
                            strlen( form_lev_mas->data_entry[0].key_prompt)
                        printf("%s", form_lev_mas->data_entry[0].
                            key_data_buffer.key_string);
                    form_lev_mas = moveto_form( dbfile, form_lev_mas);
                    form_lev_mas->read_only_status = 1;
```

```
                    form_lev_mas = form_process( form_lev_mas);
                }
                else
                    confirm( "There are no more Records", 24, 13, 27);
                break;

            case 7  : dbfile = gotop_file( dbfile);
                strcpy( form_lev_mas->data_entry[0].key_data_buffer.
                        key_string, dbfile->fields[0].field_data );
                setposition( form_lev_mas->data_entry[0].key_row,
                        form_lev_mas->data_entry[0].key_col +
                        strlen( form_lev_mas->data_entry[0].key_prompt) );
                printf("%s", form_lev_mas->data_entry[0].key_data_buffer.
                        key_string);
                    form_lev_mas = moveto_form( dbfile, form_lev_mas);
                form_lev_mas->read_only_status = 1;
                form_lev_mas = form_process( form_lev_mas);
                break;

        case 8  : dbfile = gobottom_file( dbfile);
                   strcpy( form_lev_mas->data_entry[0].key_data_buffer.
                        key_string, dbfile->fields[0].field_data );
                setposition( form_lev_mas->data_entry[0].key_row,
                        form_lev_mas->data_entry[0].key_col +
                        strlen( form_lev_mas->data_entry[0].key_prompt) );
                 printf("%s", form_lev_mas->data_entry[0].key_data_buffer.
                        key_string);
                form_lev_mas = moveto_form( dbfile, form_lev_mas);
                form_lev_mas->read_only_status = 1;
                form_lev_mas = form_process( form_lev_mas);
                break;
    }
    prompt_status = 1;
}


turn( dbfile);
```

```
/* ------------------------------------------------------------------------*/
/* Module ..........Lev_mas_prompts()                                      */
/* Function ........Displays on the console, the prompts for the entry form of*/
/*                  Leave Master File                                      */

KEY_FORM *lev_mas_prompts( KEY_FORM *form_lev_mas)
{
   clearscreen( 6, 2, 20, 78);

   form_lev_mas->data_entry[0] = init_field( form_lev_mas->data_entry[0],
                                 "Leave Code : ", 5, 'C', 7, 3, "", CAPS_ON,
                                 "lev_code");
   form_lev_mas->data_entry[1] = init_field( form_lev_mas->data_entry[1],
                                 "Leave Name : ", 30, 'C', 7, 32, "", CAPS_ON,
                                 "lev_name");
   form_lev_mas->data_entry[2] = init_field( form_lev_mas->data_entry[2],
                                 "Leave Description : ", 50, 'C', 9, 3, "", CAPS_ON
                                 "lev_descr");
   form_lev_mas->data_entry[3] = init_field( form_lev_mas->data_entry[3],
                                 "Max. Duration : ", 3, 'N', 11, 3, "", CAPS_ON,
                                 "max_durn");
   form_lev_mas->data_entry[4] = init_field( form_lev_mas->data_entry[4],
                                 "Sex (M/F/B) : ", 1, 'C', 11, 32, "", CAPS_ON,
                                 "sex");
   form_lev_mas->data_entry[5] = init_field( form_lev_mas->data_entry[5],
                                 "On Confirmation : ", 1, 'L', 13, 3, "", CAPS_ON,
                                 "confirm");
   form_lev_mas->data_entry[6] = init_field( form_lev_mas->data_entry[6],
                                 "Accumulation Type : ", 1, 'L', 15, 3, "", CAPS_ON
                                 "accum_type");
   form_lev_mas->data_entry[7] = init_field( form_lev_mas->data_entry[7],
                                 "Max. Accumulation : ", 3, 'N', 15, 32, "", CAPS_O
                                 "max_accum");
   form_lev_mas->data_entry[8] = init_field( form_lev_mas->data_entry[8],
                                 "Borrow Type : ", 1, 'L', 17, 3, "", CAPS_ON,
                                 "borr_type");
   form_lev_mas->data_entry[9] = init_field( form_lev_mas->data_entry[9],
                                 "Max. Borrow : ", 3, 'N', 17, 32, "", CAPS_ON,
                                 "max_borr");
   form_lev_mas->data_entry[10] = init_field( form_lev_mas->data_entry[10],
                                  "Encashment Type : ", 1, 'L', 19, 3, "", CAPS_ON,
                                  "encash_type");
   form_lev_mas->data_entry[11] = init_field( form_lev_mas->data_entry[11],
                                  "Details : ", 30, 'C', 19, 28, "", CAPS_ON,
                                  "details");

   return( form_lev_mas);
}
```

```
/* ------------------------------------------------------------------------*/
/* Module ..........Emp_mas()                                              */
/* Function ........Performs operations on the Employee Personal Details File */

DBFILE *emp_mas( DBFILE *dbfile)
{
    KEY_FORM *form_emp_mas;
    int ch;
    unsigned char prompt_status;
    char *str;

    prompt_status = 0;
    form_emp_mas->form_key_value = 1;
    form_emp_mas = init_key_form( form_emp_mas, " Employee Personal Record ",
                   11, 5, 1, 23, 79);
    form_emp_mas = emp_mas_prompts( form_emp_mas);
    while( form_emp_mas->form_key_value != 0)
    {
        if( prompt_status )
           form_emp_mas = emp_mas_prompts( form_emp_mas);
        prompt_status = 0;
        form_emp_mas = read_key_form( form_emp_mas, 5, 1, 23, 79);
        if( form_emp_mas->form_key_value < 6)
        {
            while(1)
            {
                if( prompt_status )
                    form_emp_mas = emp_mas_prompts( form_emp_mas);
                prompt_status = 0;
                if( !form_emp_mas->form_key_value )
                    break;

                form_emp_mas->data_entry[0] = read_field( form_emp_mas->
                                                          data_entry[0]);
                if( form_emp_mas->data_entry[0].key_field_value == ESC)
                    break;

                if( is_space( form_emp_mas->data_entry[0].key_data_buffer.
                              key_string ))
                    continue;

                else
                {
                     prompt_status = 1;
                     switch( form_emp_mas->form_key_value)
                     {
                         case 1  : dbfile = get_rec( dbfile, "EMP_CODE",
                                                     form_emp_mas->data_entry[0].
                                                     key_data_buffer.key_string );
                                   if( dbfile->locate_status)
```

```
                {
                    confirm( "Record  already  exists",
                          24, 13, 27);
                    clearscreen( 24, 0, 24, 79);
                    prompt_status = 0;
                    break;
                }
                form_emp_mas->read_only_status = 0;
                form_emp_mas = form_process( form_emp_mas);
                if( form_emp_mas->exit_status )
                {
                    str = moveto_dbfile( form_emp_mas);
                    dbfile = add_rec( dbfile, str);
                }
                break;

        case 2  : dbfile = get_rec( dbfile, "EMP_CODE",
                          form_emp_mas->data_entry[0].
                          key_data_buffer.key_string );
                if( !dbfile->locate_status)
                {
                    confirm( "Record  does  not  exist",
                             24, 13, 27);
                    clearscreen( 24, 0, 24, 79);
                    break;
                }
                if( dbfile->delete_status)
                {
                    confirm( "Record has been Deleted",
                             24, 13, 27);
                    clearscreen( 24, 0, 24, 79);
                    break;
                }
                form_emp_mas = moveto_form( dbfile, form_emp_mas);
                form_emp_mas->read_only_status = 0;
                form_emp_mas = form_process( form_emp_mas);
                if( form_emp_mas->exit_status )
                {
                    str = moveto_dbfile( form_emp_mas);
                    dbfile = replace_rec( dbfile, str);
                }
                break;

        case 3  : dbfile = get_rec( dbfile, "EMP_CODE",
                          form_emp_mas->data_entry[0].
                          key_data_buffer.key_string);
                if( !dbfile->locate_status)
                {
                    confirm( "Record  does  not  exist",
                             24, 13, 27);
```

```
                clearscreen( 24, 0, 24, 79);
                break;
            }
            if( dbfile->delete_status)
            {
                confirm( "Record has been Deleted",
                         24, 13, 27);
                clearscreen( 24, 0, 24, 79);
                break;
            }
            form_emp_mas = moveto_form( dbfile, form_emp_mas);
            ch = confirm( "Press DEL to delete, ESC to abort",
                          24, 211, 27);
            clearscreen( 24, 0, 24, 79);
            if( ch == DEL)
                dbfile = delete_rec( dbfile);
            break;

    case 4  : dbfile = get_rec( dbfile, "EMP_CODE",
                        form_emp_mas->data_entry[0].
                        key_data_buffer.key_string);
            if( !dbfile->locate_status)
            {
                confirm( "Record  does  not  exist",
                         24, 13, 27);
                clearscreen( 24, 0, 24, 79);
                break;
            }
            if( dbfile->delete_status)
            {
                confirm( "Record has been Deleted",
                         24, 13, 27);
                clearscreen( 24, 0, 24, 79);
                break;
            }
            form_emp_mas = moveto_form( dbfile, form_emp_mas);
            form_emp_mas->read_only_status = 1;
            form_emp_mas = form_process( form_emp_mas);
            break;

    case 5  : dbfile = get_rec( dbfile, "EMP_CODE",
                        form_emp_mas->data_entry[0].
                        key_data_buffer.key_string);
            if( !dbfile->locate_status)
            {
                confirm( "Record  does  not  exist",
                         24, 13, 27);
                clearscreen( 24, 0, 24, 79);
                break;
            }
```

```
                        form_emp_mas = moveto_form( dbfile, form_emp_mas);
                        form_emp_mas->read_only_status = 1;
                        form_emp_mas = form_process( form_emp_mas);
                        break;
                }
        }
    }
}
else
{
    switch( form_emp_mas->form_key_value)
    {
        case 6  : if( !dbfile->eof_status)
                  {
                    dbfile = skip_rec( dbfile, 1);
                    strcpy( form_emp_mas->data_entry[0].key_data_buffer.
                            key_string, dbfile->fields[0].field_data );
                    setposition( form_emp_mas->data_entry[0].key_row,
                            form_emp_mas->data_entry[0].key_col +
                            strlen( form_emp_mas->data_entry[0].key_prompt) )
                        printf("%s", form_emp_mas->data_entry[0].
                            key_data_buffer.key_string);
                    form_emp_mas = moveto_form( dbfile, form_emp_mas);
                    form_emp_mas->read_only_status = 1;
                    form_emp_mas = form_process( form_emp_mas);
                  }
                  else
                      confirm( "There are no more Records", 24, 13, 27);
                  break;

          case 7  : dbfile = gotop_file( dbfile);
                  strcpy( form_emp_mas->data_entry[0].key_data_buffer.
                            key_string, dbfile->fields[0].field_data );
                  setposition( form_emp_mas->data_entry[0].key_row,
                            form_emp_mas->data_entry[0].key_col +
                            strlen( form_emp_mas->data_entry[0].key_prompt) );
                  printf("%s", form_emp_mas->data_entry[0].key_data_buffer.
                            key_string);
                      form_emp_mas = moveto_form( dbfile, form_emp_mas);
                  form_emp_mas->read_only_status = 1;
                  form_emp_mas = form_process( form_emp_mas);
                  break;

        case 8  : dbfile = gobottom_file( dbfile);
                     strcpy( form_emp_mas->data_entry[0].key_data_buffer.
                            key_string, dbfile->fields[0].field_data );
                  setposition( form_emp_mas->data_entry[0].key_row,
                            form_emp_mas->data_entry[0].key_col +
                            strlen( form_emp_mas->data_entry[0].key_prompt) );
                   printf("%s", form_emp_mas->data_entry[0].key_data_buffer.
```

```
                                    key_string);
                        form_emp_mas = moveto_form( dbfile, form_emp_mas);
                        form_emp_mas->read_only_status = 1;
                        form_emp_mas = form_process( form_emp_mas);
                        break;
                }
                prompt_status = 1;
            }
        }

    return( dbfile);




/* ------------------------------------------------------------------------*/
/* Module ..........Emp_mas_prompts()                                      */
/* Function ........Displays on the console, prompts for the entry form of */
/*                  Employee Personal details                              */

KEY_FORM *emp_mas_prompts( KEY_FORM *form_emp_mas)

    clearscreen( 6, 2, 18, 78);
    form_emp_mas->data_entry[0] = init_field( form_emp_mas->data_entry[0],
                                  "Employee Code : ", 5, 'C', 7, 3, "",
                                   CAPS_ON, "emp_code");
    form_emp_mas->data_entry[1] = init_field( form_emp_mas->data_entry[1],
                                  "Employee Name : ", 30, 'C', 7, 32, "",
                                   CAPS_ON, "emp_name" );
    form_emp_mas->data_entry[2] = init_field( form_emp_mas->data_entry[2],
                                  "Address : (Temp) ", 50, 'C', 9, 3, "",
                                   CAPS_ON, "addr_temp" );
    form_emp_mas->data_entry[3] = init_field( form_emp_mas->data_entry[3],
                                  "          (Perm) ", 50, 'C', 10, 3, "",
                                   CAPS_ON, "addr_perm");
    form_emp_mas->data_entry[4] = init_field( form_emp_mas->data_entry[4],
                                  "Sex : ", 1, 'C', 12, 3, "", CAPS_ON, "sex");
    form_emp_mas->data_entry[5] = init_field( form_emp_mas->data_entry[5],
                                  "Marital Status : ", 1, 'C', 12, 19, "",
                                   CAPS_ON, "mar_status");
    form_emp_mas->data_entry[6] = init_field( form_emp_mas->data_entry[6],
                                  "Date of Birth : ", 8, 'D', 12, 47, "  /  /  ",
                                   CAPS_ON, "d_birth");
    form_emp_mas->data_entry[7] = init_field( form_emp_mas->data_entry[7],
                                  "Date of Joining : ", 8, 'D', 14, 3, "  /  /  ",
                                   CAPS_ON, "d_join");
    form_emp_mas->data_entry[8] = init_field( form_emp_mas->data_entry[8],
                                  "Date of Confirmation : ", 8, 'D', 14, 41,
                                  "  /  /  ", CAPS_ON, "d_conf");
    form_emp_mas->data_entry[9] = init_field( form_emp_mas->data_entry[9],
                                  "Designation : ", 20, 'C', 16, 3, "", CAPS_ON,
                                  "designatn");
    form_emp_mas->data_entry[10] = init_field( form_emp_mas->data_entry[10],
                                   "Location : ", 15, 'C', 16, 46, "", CAPS_ON,
                                   "location");
```

```
 form_emp_mas->data_entry[11] = init_field( form_emp_mas->data_entry[11],
                                 "Passport No. : ", 10, 'C', 18, 3, "",
                                  CAPS_ON, "passport");

 return( form_emp_mas);



-------------------------------------------------------------------------------*/
Module ..........Process()                                                     */
Function ........Performs operations on the Employee Leave Details File       */

FILE *process( DBFILE *dbf_process)

 KEY_FORM *kform_process;
 DBFILE *dbf_emp_mas, *dbf_lev_mas;
 unsigned char prompt_status, i;
 char *str, catstr[10], date[8], day[2], month[2], year[4];
 int ch;
 struct dosdate_t today;


 kform_process = mem_form_alloc( kform_process );
 memset( (KEY_FORM *)kform_process, NULL, sizeof(KEY_FORM) );

 prompt_status = 0;
 for( i = 0; i <= 10; i++)
     catstr[i] = ' ';
 catstr[i] = '\0';
 for( i = 0; i <= 8; i++)
     date[i] = ' ';
 date[i] = '\0';
 kform_process->form_key_value = 1;
 kform_process = init_key_form( kform_process, " Employee Leave Record ",
                  11, 5, 1, 23, 79);
 kform_process = process_prompts( kform_process);

 _dos_getdate( &today);
 itoa( today.day, day, 10);
 itoa( today.month, month, 10);
 itoa( today.year, year, 10);
 strcat( date, day);
 strcat( date, month);
 strcat( date, year);
 strcpy( kform_process->data_entry[0].key_data_buffer.key_date, date);

 while( kform_process->form_key_value != 0)
 {
     if( prompt_status )
```

```
   kform_process = process_prompts( kform_process);
prompt_status = 0;
kform_process = read_key_form( kform_process, 5, 1, 23, 79);
while(1)
{
    if( prompt_status )
       kform_process = process_prompts( kform_process);
    prompt_status = 0;
    if( !kform_process->form_key_value )
       break;

    kform_process->data_entry[1] = read_field( kform_process->data_entry[1]);
    if( kform_process->data_entry[1].key_field_value == ESC)
       break;

    if( is_space( kform_process->data_entry[1].key_data_buffer.key_string ))
       continue;
    else
    {
        switch( kform_process->form_key_value)
        {
            case 1 : dbf_emp_mas = usedbf( dbf_emp_mas, "emp_mas.dbf");
                     dbf_emp_mas = get_rec( dbf_emp_mas, "EMP_CODE",
                                  kform_process->data_entry[1].
                                  key_data_buffer.key_string);
                     if( !dbf_emp_mas->locate_status)
                     {
                         confirm( "No record found", 24, 13, 27);
                         clearscreen( 24, 0, 24, 79);
                         break;
                     }
                        setposition( kform_process->data_entry[2].key_row,
                                  kform_process->data_entry[2].key_col +
                                  strlen( kform_process->data_entry[1].key_promp
                     printf( dbf_emp_mas->fields[1].field_data);
                     kform_process->data_entry[2] = read_field
                                  ( kform_process->data_entry[2]);
                     kform_process->data_entry[3] = read_field
                                  ( kform_process->data_entry[3]);
                     dbf_lev_mas = usedbf( dbf_lev_mas, "lev_mas.dbf");
                     dbf_lev_mas = get_rec( dbf_lev_mas, "LEV_CODE",
                                  kform_process->data_entry[3].
                                  key_data_buffer.key_string);
                     if( !dbf_lev_mas->locate_status)
                     {
                          confirm( "No record found", 24, 13, 27);
                          clearscreen( 24, 0, 24, 79);
                          break;
                     }
                     strcpy( catstr, kform_process->data_entry[1].
```

```
                    key_data_buffer.key_string);
strcat( catstr, kform_process->data_entry[3].
                    key_data_buffer.key_string);
strcpy( kform_process->data_entry[13].
                    key_data_buffer.key_string, catstr);
dbf_process = get_rec( dbf_process, "CURR_REC", catstr);
if( !dbf_process->locate_status)
    dbf_process->fields[8].field_value.type_numeric = 0;
setposition( kform_process->data_entry[4].key_row,
             kform_process->data_entry[4].key_col +
             strlen( kform_process->data_entry[4].key_prompt)
printf( dbf_lev_mas->fields[1].field_data);
if( strcmp( dbf_lev_mas->fields[4].field_data, "b" )
            && strcmp( dbf_emp_mas->fields[4].field_data,
            dbf_lev_mas->fields[4].field_data) )
{
     confirm( "Employee is not entitled to this leave",
              24, 13, 27);
     clearscreen( 24, 0, 24, 79);
     break;
}
kform_process->data_entry[5] = read_field
              ( kform_process->data_entry[5]);
kform_process->data_entry[6] = read_field
              ( kform_process->data_entry[6]);
kform_process->data_entry[7] = read_field
              ( kform_process->data_entry[7]);
dbf_process->fields[8].field_value.type_numeric =
    dbf_process->fields[8].field_value.type_numeric +
    atoi( kform_process->data_entry[5].key_data_buffer.
    key_numeric);
itoa(  dbf_process->fields[8].field_value.type_numeric,
     kform_process->data_entry[8].key_data_buffer.
     key_numeric, 10);
setposition( kform_process->data_entry[8].key_row,
     kform_process->data_entry[8].key_col +
     strlen( kform_process->data_entry[8].key_prompt));
printf( kform_process->data_entry[8].key_data_buffer.
     key_numeric);
if( dbf_process->fields[8].field_value.type_numeric >
    dbf_lev_mas->fields[3].field_value.type_numeric)
{
     confirm( "Total Leave exceeds Maximum Leave",
              24, 13, 27);
    clearscreen( 24, 0, 24, 79);
    while(1)
    {
        kform_process->data_entry[9] = read_field
              ( kform_process->data_entry[9]);
        if( kform_process->data_entry[9].key_data_buffer.
```

```
                    key_logical == "y")
                {
                    kform_process->data_entry[10] = read_field
                        ( kform_process->data_entry[10]);
                    break;
                }
                else
                {
                    kform_process->data_entry[11] = read_field
                        ( kform_process->data_entry[11]);
                    if( kform_process->data_entry[11].
                        key_data_buffer.key_logical == "y")
                    {
                        kform_process->data_entry[12] =
                        read_field( kform_process->
                                    data_entry[12]);
                        break;
                    }
                    else
                    {
                        confirm( "Extra days leave has not
                                  been allocated", 24, 13, 27);
                        clearscreen( 24, 0, 24, 79);
                        ch = confirm( "Do you wish to
                             continue ( Y/N ) :", 24, 78, 89);
                        if( ch == 89)
                            continue;
                        else
                            break;
                    }
                }
            }
        }
        ch = confirm( "Are the Entries O.K. ( Y/N) :",
                      24, 78, 89);
        clearscreen( 24, 0, 24, 79);
        if( ch == 89)
            str = moveto_dbfile( kform_process);
        closedbf( dbf_emp_mas);
        closedbf( dbf_lev_mas);

case 2 : confirm( "This option facility is unavailable",
                  24, 13, 27);
         clearscreen( 24, 0, 24, 79);
         break;

case 3 : dbf_emp_mas = usedbf( dbf_emp_mas, "emp_mas.dbf");
         dbf_emp_mas = get_rec( dbf_emp_mas, "EMP_CODE",
                       kform_process->data_entry[1].
                       key_data_buffer.key_string);
```

```
if( !dbf_emp_mas->locate_status)
{
    confirm( "No record found", 24, 13, 27);
    clearscreen( 24, 0, 24, 79);
    break;
}
   setposition( kform_process->data_entry[2].key_row,
     kform_process->data_entry[2].key_col +
     strlen( kform_process->data_entry[1].key_prompt));
printf( dbf_emp_mas->fields[1].field_data);
kform_process->data_entry[2] = read_field
     ( kform_process->data_entry[2]);
kform_process->data_entry[3] = read_field
     ( kform_process->data_entry[3]);
dbf_lev_mas = usedbf( dbf_lev_mas, "lev_mas.dbf");
dbf_lev_mas = get_rec( dbf_lev_mas, "LEV_CODE",
       kform_process->data_entry[3].key_data_buffer.
       key_string);
if( !dbf_lev_mas->locate_status)
{
     confirm( "No record found", 24, 13, 27);
     clearscreen( 24, 0, 24, 79);
     break;
}
strcpy( catstr, kform_process->data_entry[1].
                key_data_buffer.key_string);
strcat( catstr, kform_process->data_entry[3].
                key_data_buffer.key_string);
dbf_process = get_rec( dbf_process, "CURR_REC", catstr);
if( !dbf_process->locate_status)
{
    confirm( "No record found", 24, 13, 27);
    clearscreen( 24, 0, 24, 79);
    break;
}
while(1)
{
    if( !dbf_process->delete_status)
    {
          kform_process = moveto_form( dbf_process,
                        kform_process);
        kform_process->read_only_status = 1;
        kform_process = form_process( kform_process);
        ch = confirm( "Is this the record ? (Y/N) ",
                      24, 78, 89);
        clearscreen( 24, 0, 24, 79);
    }
    if( ch == 78)
    {
        if( dbf_process->eof_status)
```

```
                    {
                        confirm( "No record exists", 24, 13, 27);
                        clearscreen( 24, 0, 24, 79);
                        break;
                    }
                    get_next_rec( dbf_process, "CURR_REC", catstr);
                    continue;
                }
                else
                    break;
            }

f( !dbf_process->delete_status)
            {
                ch = confirm( "Press DEL to delete, ESC to abort",
                              24, 211, 27);
                clearscreen( 24, 0, 24, 79);
                if( ch == DEL)
                    dbf_process = delete_rec( dbf_process);
            }
            closedbf( dbf_emp_mas);
            closedbf( dbf_lev_mas);
            break;

        case 4 : dbf_emp_mas = usedbf( dbf_emp_mas, "emp_mas.dbf");
                 dbf_emp_mas = get_rec( dbf_emp_mas, "EMP_CODE",
                               kform_process->data_entry[1].
                               key_data_buffer.key_string);
                 if( !dbf_emp_mas->locate_status)
                 {
                     confirm( "No record found", 24, 13, 27);
                     clearscreen( 24, 0, 24, 79);
                     break;
                 }
                    setposition( kform_process->data_entry[2].key_row,
                                 kform_process->data_entry[2].key_col +
                                 strlen( kform_process->data_entry[1].key_prompt));
                 printf( dbf_emp_mas->fields[1].field_data);
                 kform_process->data_entry[2] = read_field
                               ( kform_process->data_entry[2]);
                 kform_process->data_entry[3] = read_field
                               ( kform_process->data_entry[3]);
                 dbf_lev_mas = usedbf( dbf_lev_mas, "lev_mas.dbf");
                 dbf_lev_mas = get_rec( dbf_lev_mas, "LEV_CODE",
                               kform_process->data_entry[3].
                               key_data_buffer.key_string);
                 if( !dbf_lev_mas->locate_status)
                 {
                      confirm( "No record found", 24, 13, 27);
                      clearscreen( 24, 0, 24, 79);
```

```
            break;
        }
        strcpy( catstr, kform_process->data_entry[1].
                        key_data_buffer.key_string);
        strcat( catstr, kform_process->data_entry[3].
                        key_data_buffer.key_string);
        dbf_process = get_rec( dbf_process, "CURR_REC", catstr);
        if( !dbf_process->locate_status)
        {
            confirm( "No record found", 24, 13, 27);
            clearscreen( 24, 0, 24, 79);
            break;
        }
        while(1)
        {
            if( !dbf_process->delete_status)
            {
                kform_process = moveto_form( dbf_process,
                                kform_process);
                kform_process->read_only_status = 1;
                kform_process = form_process( kform_process);
                ch = confirm( "Is this the record ? (Y/N) ",
                              24, 78, 89);
                clearscreen( 24, 0, 24, 79);
            }
            if( ch == 78)
            {
                if( dbf_process->eof_status)
                {
                    confirm( "No record exists", 24, 13, 27);
                    clearscreen( 24, 0, 24, 79);
                    break;
                }
                get_next_rec( dbf_process, "CURR_REC", catstr);
                continue;
            }
            else
                break;
        }
        closedbf( dbf_emp_mas);
        closedbf( dbf_lev_mas);
        break;

case 5 : dbf_emp_mas = usedbf( dbf_emp_mas, "emp_mas.dbf");
         dbf_emp_mas = get_rec( dbf_emp_mas, "EMP_CODE",
                       kform_process->data_entry[1].
                       key_data_buffer.key_string);
         if( !dbf_emp_mas->locate_status)
         {
             confirm( "No record found", 24, 13, 27);
```

```
    clearscreen( 24, 0, 24, 79);
    break;
}
   setposition( kform_process->data_entry[2].key_row,
              kform_process->data_entry[2].key_col +
              strlen( kform_process->data_entry[1].key_prompt));
printf( dbf_emp_mas->fields[1].field_data);
kform_process->data_entry[2] = read_field
              ( kform_process->data_entry[2]);
kform_process->data_entry[3] = read_field
              ( kform_process->data_entry[3]);
dbf_lev_mas = usedbf( dbf_lev_mas, "lev_mas.dbf");
dbf_lev_mas = get_rec( dbf_lev_mas, "LEV_CODE",
              kform_process->data_entry[3].
              key_data_buffer.key_string);
if( !dbf_lev_mas->locate_status)
{
     confirm( "No record found", 24, 13, 27);
     clearscreen( 24, 0, 24, 79);
     break;
}
strcpy( catstr, kform_process->data_entry[1].
        key_data_buffer.key_string);
strcat( catstr, kform_process->data_entry[3].
        key_data_buffer.key_string);
dbf_process = get_rec( dbf_process, "CURR_REC", catstr);
if( !dbf_process->locate_status)
{
    confirm( "No record found", 24, 13, 27);
    clearscreen( 24, 0, 24, 79);
    break;
}
while(1)
{
    if( !dbf_process->delete_status)
    {
          kform_process = moveto_form( dbf_process,
                        kform_process);
        kform_process->read_only_status = 1;
        kform_process = form_process( kform_process);
        ch = confirm( "Is this the record ? (Y/N) ",
                      24, 78, 89);
        clearscreen( 24, 0, 24, 79);
    }
    if( ch == 78)
    {
        if( dbf_process->eof_status)
        {
            confirm( "No record exists", 24, 13, 27);
            clearscreen( 24, 0, 24, 79);
```

```
                    break;
                }
                get_next_rec( dbf_process, "CURR_REC", catstr);
                continue;
            }
            else
                break;
        }
        closedbf( dbf_emp_mas);
        closedbf( dbf_lev_mas);
        break;

case 6 : if( dbf_process->eof_status)
        {
            confirm( "No record exists", 24, 13, 27);
            clearscreen( 24, 0, 24, 79);
            break;
        }
        get_next_rec( dbf_process, "CURR_REC",
                      kform_process->data_entry[13].
                      key_data_buffer.key_string);
        if( !dbf_process->delete_status)
        {
            kform_process = moveto_form( dbf_process,
                            kform_process);
            kform_process->read_only_status = 1;
            kform_process = form_process( kform_process);
        }
        break;

case 7 : dbf_emp_mas = usedbf( dbf_emp_mas, "emp_mas.dbf");
            dbf_emp_mas = get_rec( dbf_emp_mas, "EMP_CODE",
                      kform_process->data_entry[1].
                      key_data_buffer.key_string);
        if( !dbf_emp_mas->locate_status)
        {
            confirm( "No record found", 24, 13, 27);
            clearscreen( 24, 0, 24, 79);
            break;
        }
           setposition( kform_process->data_entry[2].key_row,
                      kform_process->data_entry[2].key_col +
                      strlen( kform_process->data_entry[1].
                      key_prompt));
        printf( dbf_emp_mas->fields[1].field_data);
        kform_process->data_entry[2] = read_field
                       ( kform_process->data_entry[2]);
        kform_process->data_entry[3] = read_field
                       ( kform_process->data_entry[3]);
        dbf_lev_mas = usedbf( dbf_lev_mas, "lev_mas.dbf");
```

```
            dbf_lev_mas = get_rec( dbf_lev_mas, "LEV_CODE",
                          kform_process->data_entry[3].
                          key_data_buffer.key_string);
            if( !dbf_lev_mas->locate_status)
            {
                 confirm( "No record found", 24, 13, 27);
                 clearscreen( 24, 0, 24, 79);
                 break;
            }
            strcpy( catstr, kform_process->data_entry[1].
                            key_data_buffer.key_string);
            strcat( catstr, kform_process->data_entry[3].
                            key_data_buffer.key_string);
            dbf_process = get_rec( dbf_process, "CURR_REC", catstr);
            if( !dbf_process->locate_status)
            {
                confirm( "No record found", 24, 13, 27);
                clearscreen( 24, 0, 24, 79);
                break;
            }
            if( !dbf_process->delete_status)
            {
                    kform_process = moveto_form( dbf_process,
                                  kform_process);
                 kform_process->read_only_status = 1;
                 kform_process = form_process( kform_process);
            }
            closedbf( dbf_emp_mas);
            closedbf( dbf_lev_mas);
            break;

case 8 :    dbf_emp_mas = usedbf( dbf_emp_mas, "emp_mas.dbf");
                 dbf_emp_mas = get_rec( dbf_emp_mas, "EMP_CODE",
                          kform_process->data_entry[1].
                          key_data_buffer.key_string);
            if( !dbf_emp_mas->locate_status)
            {
                confirm( "No record found", 24, 13, 27);
                clearscreen( 24, 0, 24, 79);
                break;
            }
               setposition( kform_process->data_entry[2].key_row,
                         kform_process->data_entry[2].key_col +
                         strlen( kform_process->data_entry[1].key_prompt));
            printf( dbf_emp_mas->fields[1].field_data);
            kform_process->data_entry[2] = read_field
                          ( kform_process->data_entry[2]);
            kform_process->data_entry[3] = read_field
                          ( kform_process->data_entry[3]);
            dbf_lev_mas = usedbf( dbf_lev_mas, "lev_mas.dbf");
```

```
                    dbf_lev_mas = get_rec( dbf_lev_mas, "LEV_CODE",
                                  kform_process->data_entry[3].
                                  key_data_buffer.key_string);
                    if( !dbf_lev_mas->locate_status)
                    {
                         confirm( "No record found", 24, 13, 27);
                         clearscreen( 24, 0, 24, 79);
                         break;
                    }
                    strcpy( catstr, kform_process->data_entry[1].
                                    key_data_buffer.key_string);
                    strcat( catstr, kform_process->data_entry[3].
                                    key_data_buffer.key_string);
                    dbf_process = gotop_file( dbf_process);
                    while( !dbf_process->eof_status)
                    {
                         dbf_process = get_next_rec( dbf_process,
                                       "CURR_REC", catstr);
                    }
                    if( !dbf_process->locate_status)
                    {
                        confirm( "Record does not exist", 24, 13, 27);
                        clearscreen( 24, 0, 24, 79);
                        break;
                    }
                    kform_process = moveto_form( dbf_process, kform_process);
                    kform_process->read_only_status = 1;
                    kform_process = form_process( kform_process);
                    closedbf( dbf_emp_mas);
                    closedbf( dbf_lev_mas);
                    break;
          }

          prompt_status = 1;
          }
     }

ee(kform_process);
curn( dbf_process);
```

```
/* ------------------------------------------------------------------------*/
/* Module ..........Process_prompts()                                      */
/* Function ........Displays on the console, prompts for the entry form of */
/*                  Employee Leave Details                                 */

KEY_FORM *process_prompts( KEY_FORM *form_process)
{
    clearscreen( 6, 2, 18, 78);
    form_process->data_entry[1] = init_field( form_process->data_entry[1],
                                  "Employee Code : ", 5, 'C', 7, 3, "", CAPS_ON,
                                  "emp_code");
    form_process->data_entry[2] = init_field( form_process->data_entry[2],
                                  "Employee Name : ", 30, 'C', 7, 32, "", CAPS_ON,
                                  "emp_name");
    form_process->data_entry[3] = init_field( form_process->data_entry[3],
                                  "Leave Code : ", 5, 'C', 9, 3, "", CAPS_ON,
                                  "lev_code");
    form_process->data_entry[4] = init_field( form_process->data_entry[4],
                                  "Leave Name : ", 30, 'C', 9, 32, "", CAPS_ON,
                                  "lev_code");
    form_process->data_entry[5] = init_field( form_process->data_entry[5],
                                  "Duration : ", 3, 'N', 11, 3, "", CAPS_ON,
                                  "lev_durn");
    form_process->data_entry[6] = init_field( form_process->data_entry[6],
                                  "From : ", 8, 'D', 11, 25, "  /  /  ", CAPS_ON,
                                  "st_date");
    form_process->data_entry[7] = init_field( form_process->data_entry[7],
                                  "To : ", 8, 'D', 11, 50, "  /  /  ", CAPS_ON,
                                  "end_date");
    form_process->data_entry[8] = init_field( form_process->data_entry[8],
                                  "Total Leave Taken : ", 3, 'N', 13, 3, "", CAPS_ON
                                  "total_lev");
    form_process->data_entry[9] = init_field( form_process->data_entry[9],
                                  "Leave without Pay : ", 1, 'L', 15, 3, "", CAPS_ON
                                  "lwp");
    form_process->data_entry[10] = init_field( form_process->data_entry[10],
                                   "Days of Leave without Pay : ", 3, 'N', 15, 32, "
                                    CAPS_ON, "lwp_days");
    form_process->data_entry[11] = init_field( form_process->data_entry[11],
                                   "Privileged Leave : ", 1, 'L', 17, 3, "",
                                    CAPS_ON, "privileged");
    form_process->data_entry[12] = init_field( form_process->data_entry[12],
                                   "Days of Privileged Leave : ", 3, 'N', 17, 32, ""
                                    CAPS_ON, "privi_days");

    return( form_process);
}
```

```
/* ------------------------------------------------------------------------*/
/* Module ..........Moveto_file()                                          */
/* Function ........Moves data items from the form to the file             */

char *moveto_dbfile( KEY_FORM *form)
{
   char *buff, concat[MAX_REC_LENGTH];
   unsigned char i;

   buff = (char *) malloc( MAX_REC_LENGTH + 50 );
   buff[0] = ' ';
   buff[1] = '\0';
   for( i = 0; i <= form->last_field; i++ )
    {
        switch( form->data_entry[i].key_data_type )
         {
             case 'C' :  strcat( buff, form->data_entry[i].key_data_buffer.
                         key_string);
                         break;

             case 'D' :  strcat( buff, form->data_entry[i].key_data_buffer.
                         key_date);
                         break;

             case 'L' :  strcat( buff, form->data_entry[i].key_data_buffer.
                         key_logical);
                         break;

             case 'N' :  strcat( buff, form->data_entry[i].key_data_buffer.
                         key_numeric);
                         break;

         }
    }
   strcpy( concat, buff);
   free( buff);
   return ( concat);
}
```

97-5-94

```
/* ------------------------------------------------------------------------*/
/* Module .........Moveto_form()                                           */
/* Function .......Moves data items from file to form                      */

KEY_FORM *moveto_form( DBFILE *dbfile, KEY_FORM *key_form )
{
   unsigned char i;

   for( i = 1; i <= dbfile->no_of_fields; i++ )
   {
      switch( dbfile->fields[i].data_type )
       {
          case 'C' : strcpy( key_form->data_entry[i].key_data_buffer.
                             key_string, dbfile->fields[i].field_data );
                     setposition( key_form->data_entry[i].key_row,
                                  key_form->data_entry[i].key_col +
                                  strlen(key_form->data_entry[i].key_prompt) );
                     printf("%s", key_form->data_entry[i].key_data_buffer.
                                  key_string);
                     break;

          case 'D' : strcpy( key_form->data_entry[i].key_data_buffer.key_date,
                             dbfile->fields[i].field_data );
                     setposition( key_form->data_entry[i].key_row,
                                  key_form->data_entry[i].key_col +
                                  strlen(key_form->data_entry[i].key_prompt) );
                     printf("%s", key_form->data_entry[i].key_data_buffer.key_date);
                     break;

          case 'L' : strcpy( key_form->data_entry[i].key_data_buffer.key_logical,
                             dbfile->fields[i].field_data );
                     setposition( key_form->data_entry[i].key_row,
                                  key_form->data_entry[i].key_col +
                                  strlen(key_form->data_entry[i].key_prompt) );
                     printf("%s", key_form->data_entry[i].key_data_buffer.
                                  key_logical);
                     break;

          case 'N' : strcpy( key_form->data_entry[i].key_data_buffer.key_numeric,
                             dbfile->fields[i].field_data );
                     setposition( key_form->data_entry[i].key_row,
                                  key_form->data_entry[i].key_col +
                                  strlen(key_form->data_entry[i].key_prompt) );
                     printf("%s", key_form->data_entry[i].key_data_buffer.
                                  key_numeric);
                     break;
       }
   }
  return (key_form);
}
```