## 15.6 PROGRAM EFFICIENCY

Two critical resources of a computer system are execution time and memory. The efficiency of a program is measured in terms of these two resources. Efficiency can be improved with good design and coding practices.

### Execution Time

The execution time is directly tied to the efficiency of the algorithm selected. However, certain coding techniques can considerably improve the execution efficiency. The following are some of the techniques, which could be applied while coding the program.

1. Select the fastest algorithm possible.
2. Simplify arithmetic and logical expressions.
3. Use fast arithmetic operations, whenever possible.
4. Carefully evaluate loops to avoid any unnecessary calculations within the loops.
5. If possible, avoid the use of multi-dimensional arrays.
6. Use pointers for handling arrays and strings.

However, remember the following, while attempting to improve efficiency.

1. Analyse the algorithm and various parts of the program before attempting any efficiency changes.
2. Make it work before making it faster.
3. Keep it right while trying to make it faster.
4. Do not sacrifice clarity for efficiency.

### Memory Requirement

Memory restrictions in the microcomputer environment is a real concern to the programmer. It is therefore desirable to take all necessary steps to compress memory requirements.

1. Keep the program simple. This is the key to memory efficiency.
2. Use an algorithm that is simple and requires less steps.
3. Declare arrays and strings with correct sizes.
4. When possible, limit the use of multi-dimensional arrays.
5. Try to evaluate and incorporate memory compression features available with the language.

## *REVIEW QUESTIONS*

15.1 Discuss the various aspects of program design.
15.2 How does program design relate to program efficiency?
15.3 Readability is more important than efficiency, Comment.
15.4 Distinguish between the following:
   a. Syntactic errors and semantic errors.
   b. Run-time errors and logical errors.
   c. Run-time errors and latent errors.

d. Debugging and testing.
e. Compiler testing and run-time testing.

15.5 A program has been compiled and linked successfully. When you run this program you face one or more of the following situations.
a. Program is executed but no output.
b. It produces incorrect answers.
c. It does not stop running.

15.6 List five common programming mistakes. Write a small program containing these errors and try to locate them with the help of computer.

15.7 In a program, two values are compared for convergence, using the statement

```
if( (x-y) < 0.00001) ...
```

Does the statement contain any error? If yes, explain the error.

15.8 A program contains the following if statements:

```
... ..
... ..
if(x>1&&y == 0)p = p/x;
if(x == 5|| p > 2) p = p+2;
... ..
... ..
```

Draw a flow chart to illustrate various logic paths for this segment of the program and list test data cases that could be used to test the execution of every path shown.

15.9 Given below is a function to compute the yth power of an integer x.

```
power(int x, int y)
{
  int p;
  p = y;
  while(y > 0)
        x *= y --;
  return(x);
}
```

This function contains some bugs. Write a test procedure to locate the errors with the help of a computer.

15.10 A program reads three values from the terminal, representing the lengths of three sides of a box namely length, width and height and prints a message stating whether the box is a cube, rectangle, or semi-rectangle. Prepare sets of data that you feel would adequately test this program.

# Appendix I

# Bit-Level Programming

## 1. INTRODUCTION

One of the unique features of C language as compared to other high-level languages is that it allows direct manipulation of individual bits within a word. Bit-level manipulations are used in setting a particular bit or group of bits to 1 or 0. They are also used to perform certain numerical computations faster. As pointed out in Chapter 3, C supports the following operators:

1. Bitwise logical operators.
2. Bitwise shift operators.
3. One's complement operator.

All these operators work only on integer type operands.

## 2. BITWISE LOGICAL OPERATORS

There are three logical bitwise operators. They are:

- *Bitwise* AND (&)
- *Bitwise* OR (|)
- *Bitwise exclusive* OR (^)

These are binary operators and require two integer-type operands. These operators work on their operands bit by bit starting from the least significant (i.e. the rightmost) bit, setting each bit in the result as shown in Table 1.

**Table 1** *Result of Logical Bitwise Operations*

| *op1* | *op2* | *op1 & op2* | *op1 \| op2* | *op1 ^ op2* |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |

## Bitwise AND

The bitwise AND operator is represented by a single ampersand (&) and is surrounded on both sides by integer expressions. The result of ANDing operation is 1 if both the bits have a value of 1; otherwise it is 0. Let us consider two variables **x** and **y** whose values are 13 and 25. The binary representation of these two variables are

```
x - - -> 0000 0000 0000 1101
y - - -> 0000 0000 0001 1001
```

If we execute statement

```
z = x & y ;
```

then the result would be:

```
z - - -> 0000 0000 0000 1001
```

Although the resulting bit pattern represents the decimal number 9, there is no apparent connection between the decimal values of these three variables.

Bitwise ANDing is often used to test whether a particular bit is 1 or 0. For example, the following program tests whether the fourth bit of the variable **flag** is 1 or 0.

```
#define TEST 8 /* represents 00.....01000 */
main()
{
        int flag;
        ....
        ....
        if((flag & TEST) != 0) /* test 4th bit */
        {
          printf(" Fourth bit is set \n");
        }
        ....
        ....
```

Note that the bitwise logical operators have lower precedence than the relational operators and therefore additional parentheses are necessary as shown above.

The following program tests whether a given number is odd or even.

```
main()
{
        int test = 1;
        int number;

        printf("Input a number \n");
        scanf("%d", &number);

        while (number != -1)
        {
             if(number & test)
                  print("Number is odd\n\n");
             else
                  printf("Number is even\n\n");

                  printf("Input a number \n");
                  scanf("%d", &number);
        }
}
```

*Output*

```
Input a number
20
Number is even

Input a number
9
Number is odd

Input a number
-1
```

### Bitwise OR

The bitwise OR is represented by the symbol | (vertical bar) and is surrounded by two integer operands. The result of OR operation is 1 if *at least* one of the bits has a value of 1; otherwise it is zero. Consider the variables **x** and **y** discussed above.

```
x - - ->      0000 0000 0000 1101
y - - ->      0000 0000 0001 1001

x|y - - ->    0000 0000 0001 1101
```

```
y = x & mask;
y = x | mask;
```

Masking is used in many different ways.

- To decide bit pattern of an integer variable.
- To copy a portion of a given bit pattern to a new variable, while the remainder of the new variable is filled with 0s (using bitwise AND).
- To copy a portion of a given bit pattern to a new variable, while the remainder of the new variable is filled with 1s (using bitwise OR).
- To copy a portion of a given bit pattern to a new variable, while the remainder of the original bit pattern is inverted within the new variable (using bitwise *exclusive* OR).

The following function uses a mask to display the bit pattern of a variable.

```
void bit_pattern(int u)
{
     int i, x, word;
     unsigned mask;

     mask = 1;
     word = 8 * sizeof(int);
     mask = mask << (word - 1);
            /* shift 1 to the leftmost position */

     for(i = 1; i<= word; i++)
     {
       x = (u & mask) ? 1 : 0; /* identify the bit */
       printf("%d", x); /* print bit value */
       mask >>= 1; /* shift mask by 11 position to right */
```

# Appendix
# II

# ASCII Values of Characters

| *ASCII Value* | *Character* | *ASCII Value* | *Character* | *ASCII Value* | *Character* | *ASCII Value* | *Character* |
|---|---|---|---|---|---|---|---|
| 000 | NUL | 032 | blank | 064 | @ | 096 | ← |
| 001 | SOH | 033 | ! | 065 | A | 097 | a |
| 002 | STX | 034 | " | 066 | B | 098 | b |
| 003 | ETX | 035 | # | 067 | C | 099 | c |
| 004 | EOT | 036 | $ | 068 | D | 100 | d |
| 005 | ENQ | 037 | % | 069 | E | 101 | e |
| 006 | ACK | 038 | & | 070 | F | 102 | f |
| 007 | BEL | 039 | ' | 071 | G | 103 | g |
| 008 | BS | 040 | ( | 072 | H | 104 | h |
| 009 | HT | 041 | ) | 073 | I | 105 | i |
| 010 | LF | 042 | * | 074 | J | 106 | j |
| 011 | VT | 043 | + | 075 | K | 107 | k |
| 012 | FF | 044 | , | 076 | L | 108 | l |
| 013 | CR | 045 | – | 077 | M | 109 | m |
| 014 | SO | 046 | . | 078 | N | 110 | n |
| 015 | SI | 047 | / | 079 | O | 111 | o |
| 016 | DLE | 048 | 0 | 080 | P | 112 | p |
| 017 | DC1 | 049 | 1 | 081 | Q | 113 | q |
| 018 | DC2 | 050 | 2 | 082 | R | 114 | r |
| 019 | DC3 | 051 | 3 | 083 | S | 115 | s |
| 020 | DC4 | 052 | 4 | 084 | T | 116 | t |
| 021 | NAK | 053 | 5 | 085 | U | 117 | u |
| 022 | SYN | 054 | 6 | 086 | V | 118 | v |
| 023 | ETB | 055 | 7 | 087 | W | 119 | w |
| 024 | CAN | 056 | 8 | 088 | X | 120 | x |
| 025 | EM | 057 | 9 | 089 | Y | 121 | y |
| 026 | SUB | 058 | : | 090 | Z | 122 | z |
| 027 | ESC | 059 | ; | 091 | [ | 123 | { |
| 028 | FS | 060 | < | 092 | \ | 124 | \| |
| 029 | GS | 061 | = | 093 | ] | 125 | } |
| 030 | RS | 062 | > | 094 | ↑ | 126 | ~ |
| 031 | US | 063 | ? | 095 | - | 127 | DEL |

**Note:** The first 32 characters and the last character are control characters; they cannot be printed.

| *Function* | *Data type returned* | *Task* |
|---|---|---|
| fmod(d1, d2) | double | Return the remainder of d1/d2 (with same sign as d1). |
| labs(l) | long int | Return the absolute value of 1. |
| log(d) | double | Return the natural logarithm of d. |
| log10(d) | double | Return the logarithm (base 10) of d. |
| pow(d1,d2) | double | Return d1 raised to the d2 power. |
| sin(d) | double | Return the sine of d. |
| sinh(d) | double | Return the hyperbolic sine of d. |
| sqrt(d) | double | Return the square root of d. |
| tan(d) | double | Return the tangent of d. |
| tanh(d) | double | Return the hyperbolic tangent of d. |
| **<stdio.h>** | | |
| fclose(f) | int | Close file f. Return 0 if file is successfully closed. |
| feof(f) | int | Determine if an end-of-file condition has been reached. If so, return a nonzero value; otherwise, return 0. |
| fgetc(f) | int | Enter a single character form file f. |
| fgets(s, i, f) | char* | Enter string s, containing i characters, from file f. |
| fopen(s1,s2) | file* | Open a file named s1 of type s2. Return a pointer to the file. |
| fprint(f,...) | int | Send data items to file f. |
| fputc(c,f) | int | Send a single character to file f. |
| fputs(s,f) | int | Send string s to file f. |
| fread(s,i1,i2,f) | int | Enter i2 data items, each of size i1 bytes, from file f to string s. |
| fscanf(f,...) | int | Enter data items from file f |
| fseek(f,l,i) | int | Move the pointer for file f a distance l bytes from location i. |
| ftell(f) | long int | Return the current pointer position within file f. |
| fwrite(s,i1,i2,f) | int | Send i2 data items, each of size i1 bytes from string s to file f. |
| getc(f) | int | Enter a single character from file f. |
| getchar(void) | int | Enter a single character from the standard input device. |
| gets(s) | char* | Enter string s from the standard input device. |
| printf(...) | int | Send data items to the standard output device. |
| putc(c,f) | int | Send a single character to file f. |
| putchar(c) | int | Send a single character to the standard output device. |
| puts(s) | int | Send string s to the standard output device. |
| rewind(f) | void | Move the pointer to the beginning of file f. |
| scanf(...) | int | Enter data items from the standard input device. |
| **<stdlib.h>** | | |
| abs(i) | int | Return the absolute value of i. |
| atof(s) | double | Convert string s to a double-precision quantity. |
| atoi(s) | int | Convert string s to an integer. |
| atol(s) | long | Convert string s to a long integer. |

| *Function* | *Data type returned* | *Task* |
|---|---|---|
| calloc(u1,u2) | void* | Allocate memory for an array having u1 elements, each of length u2 bytes. Return a pointer to the beginning of the allocated space. |
| exit(u) | void | Close all files and buffers, and terminate the program. (Value of u is assigned by the function, to indicate termination status). |
| free(p) | void | Free a block of allocated memory whose beginning is indicated by p. |
| malloc(u) | void* | Allocate u bytes of memory. Return a pointer to the beginning of the allocated space. |
| rand(void) | int | Return a random positive integer. |
| realloc(p, u) | void* | Allocate u bytes of new memory to the pointer variable p. Return a pointer to the beginning of the new memory space. |
| srand(u) | void | Initialize the random number generator. |
| system(s) | int | Pass command string s to the operating system. Return 0 if the command is successfully executed; otherwise, return a nonzero value typically −1. |
| **<string.h>** | | |
| strcmp(s1, s2) | int | Compare two strings lexicographically. Return a negative value if s1<s2; 0 if s1 and s2 are identical; and a positive value if s1>s2. |
| strcmpi(s1, s2) | int | Compare two strings lexicographically, without regard to case. Return a negative value if s1<s2; 0 if s1 and s2 are identical; and a value of s1 > s2. |
| strcpy(s1, s2) | char* | Copy string s2 to string s1. |
| strlen(s) | int | Return the number of characters in string s. |
| strset(s, c) | char* | Set all characters within s to c(excluding the terminating null character \0). |
| **<time.h>** | | |
| difftime(11,12) | double | Return the time difference 11 ~ 12, where 11 and 12 represent elapsed time beyond a designated base time (see the time function). |
| time(p) | long int | Return the number of seconds elapsed beyond a designated base time. |

```
74        printf("Enter Room  Number[%3d]: ",i+1);
75        gets(iroom);
76
77        if (iroom[0] == '\0' ) /* user hits enter - quits */
78        {  gotoxy(1,25);
             cprintf("You chose to quit: Entry %d was not added to the
79  database.",i+1);
80           getch();
81           break;
82        }
83        printf("Enter Phone Number[%3d]: ",i+1);
84        gets(iphone);
85
86        if (iphone[0] == '\0') /* user hits enter - quits */
87        {  gotoxy(1,25);
             cprintf("You chose to quit: Entry %d was not added to the
88    database.",i+1);
89           getch();
90           break;
91        }
92        /* check the string for valid inputs */
93        error_iroom = chkstrdig(iroom,4);
94        error_iphone = chkstrdig(iphone,8);
95        /* loop's while room input error (out of range/character) */
96        while(error_iroom != 0)
97        { if (error_iroom == -1)
98          {  clrscr();
99             refreshscreen();
100            drawscreen();
101            gotoxy(1,4);
102            printf(">> Add Entry <<");
103            gotoxy(1,25);
               cprintf("Error: Room  Number - out of Range, Your entry was greater
104   than 4 digits. ");
105            gotoxy(1,6);
106            printf("Renter Room  Number[%3d]: ",i+1);
107            gets(iroom);
108         }
109         if (error_iroom == -2)
110         {  clrscr();
111            refreshscreen();
112            drawscreen();
113            gotoxy(1,4);
114            printf("*** Add Entry ***");
115            gotoxy(1,25);
               cprintf("Error: Room  Number - Character(s) detected, character(s)
```

```
116  are not allowed.");
117           gotoxy(1,6);
118           printf("Renter Room  Number[%3d]: ",i+1);
119           gets(iroom);
120        }/* checks string room input if valid */
121        error_iroom = chkstrdig(iroom,4);
122       }/*loop's while phone input error (out of range/character) */
123       while(error_iphone !=0)
124       {if (error_iphone == -1)
125        {  clrscr();
126           refreshscreen();
127           drawscreen();
128           gotoxy(1,4);
129           printf(">> Add Entry <<");
130           gotoxy(1,25);
              cprintf("Error: Phone Number - out of Range, Your entry was greater
131  than 8 digits. ");
132           gotoxy(1,6);
133           printf("Room  Number[%3d] Entry: %s",i+1,iroom);
134           gotoxy(1,7);
135           printf("Renter Phone Number[%3d]: ",i+1);
136           gets(iphone);
137        }
138        if (error_iphone == -2)
139        {  clrscr();
140           refreshscreen();
141           drawscreen();
142           gotoxy(1,4);
143           printf(">> Add Entry <<");
144           gotoxy(1,25);
              cprintf("Error: Phone Number - Character(s) detected, character(s)
145  are not allowed.");
146           gotoxy(1,6);
147           printf("Room  Number[%3d] Entry: %s",i+1,iroom);
148           gotoxy(1,7);
149           printf("Renter Phone Number[%3d]: ",i+1);
150           gets(iphone);
151        }/* checks phone input valid */
152        error_iphone = chkstrdig(iphone,8);
153       }
154       /* no room or phone input error - addentry */
155       if (error_iroom == 0 && error_iphone == 0)
156       {  int_iroom = atoi(iroom); /* converts string to int */
157          longint_iphone = atol(iphone); /* converts string to long int */
158          current_e_add++;
159          AddEntry(int_iroom,longint_iphone);
```

```
                cprintf("Successful: There are currently %d entries in the data
                base,
",add_count);
            /* room_found is globe it counts room no. found in FindRoom
            function */
            cprintf("found %d.",room_found);
            getch();
        }
        if (room_check == -1) /* return = -1 Room was not found */
        {  gotoxy(1,25);
           cprintf("Error: The Room No. Your looking for was Not Found.");
           getch();
        }

       }
       else
       if (option == '5') /* ListAll option */
       {  clrscr();
          refreshscreen();
          drawscreen();
          gotoxy(1,4);
          printf(">> ListAll <<\n\n");

          list_check = ListAll();

          if (list_check == 0) /* return 0 if entries are in database */
          {  gotoxy(1,25);
             cprintf("List Sucuessful");
             getch();
          }
          if (list_check == -1) /* return -1 - emptylist */
          {
              gotoxy(1,25);
              cprintf("Empty List");
              getch();
     }
       }
       else
       if (option == '6') /* Getotalentries option */
       {  total_entries = GeTotalEntries();
gotoxy(1,25);
cprintf("There are currently %d entries stored in the
Database.",total_entries);
        getch();
      }
      else
```

```
290       if (option == '7') /* Sort Option */
291       {  clrscr();
292          refreshscreen();
293          drawscreen();
294          gotoxy(1,4);
295          printf(">> Sort All Entries <<");
296          gotoxy(1,6);
297          printf("Press 'A' to sort database in [A]scending order");
298          gotoxy(1,7);
299          printf("Press 'D' to sort database in [D]escending order.");
300          gotoxy(1,9);
301          printf("Note: Database is sorted by phone no. entries.");
302          sortopt = getch();
303          flushall();
304
305          sort_check = SortAllEntries(sortopt);
306          getch();
307          if (sort_check == 0) /* return = 0 - entries, in db & was sorted */
308          {  gotoxy(1,25);
             cprintf("Database was successfully Sorted.
309  ");
310             getch();
311          }
312          if (sort_check == -1) /* return = -1 - if db is empty */
313          {  gotoxy(1,25);
314             cprintf("Database was not sorted - Database is empty!");
315             getch();
316          }
317       }
318       else
319       if (option == '8') /* Load Database from file option */
320       {  clrscr();
321          refreshscreen();
322          drawscreen();
323          gotoxy(1,4);
324          printf(">> Load Database <<");
325          LoadDB();
326       }
327       else
328       if (option == '9') /* exit option */
329       {  gotoxy(1,25);
           cprintf("Do you really want to exit?, Press 'Y' to confirm, anykey to
330  cancel");
331          exit_opt = getch();
332          flushall();
333          if (exit_opt == 'y' || exit_opt == 'Y')
```

```
406       }
407       if (add_count !=0) /* if database is not empty process with delete */
408       {/* keeps looping while move up position is not = to deleted entry */
409          for (x=0; x < del_entry; x++)
410          {   for (k=0; k < add_count; k++)
411           {/* When -1 is found it moves everything by one */
412              if (room[k] == -1 && phone[k] == -1)
413              {  loop_mov_stop=0;
414              loop_mov =0;
415              count_del++;
416              /* loop_mov_stop calculates moves needed */
417              loop_mov_stop = add_count-(k+1);
418              while (loop_mov_stop != loop_mov)
419              {      room[k+loop_mov] = room[(k+1)+loop_mov];
420                     phone[k+loop_mov] = phone[(k+1)+loop_mov];
421                     loop_mov++; /* counter for move */
422              }
423              }
424           }
425          }
426       }
427       /* Calcalates total entry */
428       add_count = add_count - del_entry;
429
430       if (del_found_flag == 0) /* flag is 0 when delete entry input was found
*/
431       { return(0);  } /* return sucessful */
432       Else
433       { return(-1); } /* return not found */
434    }
```

Let's take a closer look at how the DeleteEntry function works. To make things easier let,
Room =1,2,3,4,5,6,7,8,9,10
Phone=1,2,3,4,5,6,7,8,9,10

Ten entries in the database with the digits from 1 to 10 both having the same values entered. Now if the user requests Room/Phone "4" to be deleted, the delete entry function will find the digit "4" in both Room and Phone matching the user's request.

Find -> is done within a for loop until add_count number is reached, Add_count is the counter for the number of entries added (Line 385). If it finds the digit '4' it asks the user if he/she wants to delete the current entry in the record.

This is what happens when the user selects 'Yes',

1) Copy that current entry to a temp location (Lines 397, 399).
2) Then a '-1' is copied on top of the location where digit '4' was found overwriting it (marking it has been deleted) (Lines 398, 400). Tot_del_entry and del_entry is incremented by one each time this is done (Lines 401,402).

3) Another for loop is nested within the for, used to find '-1's' marked for deleted, it loops for the no. of entries that has been deleted (Lines 409, 412). Calculation of the move up stop position is done on line 417.
4) Then using the while loop (Line 418) everything is moved up by one position. At the end of the while loop (Line 428), the number of records that exist after deletion has been done is calculated.

```
435   /*-------------------------------------------------------------------------
436       FindPhone function
437       -----------------
438       Used to search for a phone number in the database.
439
440       Returns 0 if phone no. was found.
441       Returns -1 if phone no. is not found.
442   -------------------------------------------------------------------------*/
443   int FindPhone(long int p)
444   {
445   int k, phone_found_flag= -1;
446       gotoxy(1,8);
447       for(k=0; k < add_count; k++)
448       {  if (add_count != 0) /* if database is not empty then run a search */

449          {  if (k != 0 && (k%15) == 0)
450        {  gotoxy(1,8); /* moves cursor to beginning when screen filled */
451           getch();
452        }
453        if (p == phone[k])
          {  printf("Phone No. [%-8ld] was found in record No. [%3d  ]\tRoom No.
454   [%-4d]\n",phone[k],k+1,room[k]);
455           phone_found++;
456           phone_found_flag = 0;
457        }
458         }
459       }
460       if (phone_found_flag == 0) /* flag is 0 if record was found */
461       { return(0);  } /* return sucessful */
462       else
463       { return(-1); } /* return not found */
464   }
465   /*-------------------------------------------------------------------------
466       FindRoom function
467       ----------------
468       Used to search for a Room number in the database.
469
470       Returns 0 if room no. was found.
471       Returns -1 if room no. is not found.
472   -------------------------------------------------------------------------*/
```

```
553     add_count -1))
                {   phone_str_tmp = phone[k]; /* stores previous array to
554     phone_str_tmp */
                 phone[k] = phone[k + 1]; /* copys next array to the previous
555     array before it */
                 phone[k + 1] = phone_str_tmp; /* Previous array is copyed to next
556     array */
                 /* same process is done here but with room no. */
557              room_str_tmp = room[k];
558              room[k] = room[k + 1];
559              room[k + 1] = room_str_tmp;
560              sortalldone =1; /* sets to 1 if sort is done */
561             }
562             /* same method used here but sorts in decending order */
563             if ((phone[k] < phone[k + 1])&&(sel == 'd' || sel == 'D')&&(k !=
564     add_count -1))
565             {   phone_str_tmp = phone[k];
566              phone[k] = phone[k + 1];
567              phone[k + 1] = phone_str_tmp;
568              room_str_tmp = room[k];
569              room[k] = room[k + 1];
570              room[k + 1] = room_str_tmp;
571              sortalldone =1;
572             }
573         }
574            }while (sortalldone);
575        }
576
577        if ((sel == 'a' || sel == 'A')&&add_count !=0)
578        {   gotoxy(1,25);
579             printf("You have chosen to sort the database in [A]scending order. ");
580             return(0);    /* sucessfully sorted */
581        }
582        else
583        if ((sel == 'd' || sel == 'D')&&add_count !=0)
584        {  gotoxy(1,25);
585            printf("You have chosen to sort the database in [D]ecending order. ");
586           return(0);     /* sucessfully sorted */
587        }
588        else
589        if ((sel != 'a' || sel != 'A' || sel != 'd' || sel != 'D')&&add_count !=0)
590        {  gotoxy(1,12);
591            printf("Invalid option - database was not sorted!");
592        }
593        else
594        { return(-1); } /* list empty */
595    }
```