Number Systems

WEEK
1

Understanding numbers and number systems may seem like a strange topic; however, these are very important for fully understanding the power of C. Today you will learn:

- ☐ Why number systems are so important.
- ☐ Which number systems are important.
- ☐ How to convert from one number system to another.
- ☐ How to work with the number systems.

The Importance of Number Systems

Numbers are the key to computer programming. This can become obvious within the C programming language. The reason they are important is because every element of a computer program breaks down into a numeric value. Even what appears to be letters and symbols are only numbers to the computer. To go one step further, every numeric value within the computer—and hence every letter or symbol—can be represented with the numbers zero and one. For example, to the computer, the letter A is:

```
01000001
```

Review Tip: Why does the computer represent everything with ones and zeros? Simply stated, ones and zeros are equated with on and off. A computer can store information in memory as magnetic charges that are either positive or negative. If a charge is positive, it is on. This can be equated to one. If a charge is negative, it is off. This negative charge can be equated to zero.

Deriving the Numbers the Computer Used

You may be asking, "How does the computer know what numbers to use?" Depending on the computer, the numbers used to represent various characters may be different. In the case of IBM-compatible computers, a set of number representations have been standardized. This set of numbers is represented within an ASCII

Character Table. ASCII stands for American Standard Code for Information Interchange. The ASCII Character Table contains every standard character and its numeric equivalent. Appendix B contains a complete ASCII Character Table.

Listing 6.1 presents a program that displays the values available in the ASCII Character Table. Nothing in this program should be new to you.

Type

Listing 6.1. ASCII values.

```
1:  /* Program:  list0601.c
2:   * Author:   Bradley L. Jones
3:   * Purpose:  Print all the ASCII character values.
4:   *=========================================================*/
5:
6:  #include <stdio.h>
7:
8:  void main(void)
9:  {
10:    unsigned char ch;
11:    char trash[256];
12:
13:    printf("\n\nThe ASCII VALUES:" );
14:
15:    for( ch = 0; ch < 255; ch++ )
16:    {
17:       printf("\n%3.3d:  %c", ch, ch );
18:
19:       if( ((ch % 20) == 0) && (ch != 0) )
20:       {
21:          printf("\nPress <Enter> to continue");
22:          gets(trash);
23:       }
24:    }
25:    printf("\n%3.3d:  %c", ch, ch );
26: }
```

Output

```
The ASCII VALUES:
000:
001:  ☺
002:  ☻
003:  ♥
004:  ♦
005:  ♣
006:  ♠
007:
008:
009:
010:

011:  ♂
012:  ♀
013:
014:  ♫
015:  ☼
016:  ►
017:  ◄
018:  ↕
019:  ‼
020:  ¶
Press enter to continue
```

```
019:  ‼
020:  ¶
Press enter to continue

021:  §
022:  ▬
023:  ↨
024:  ↑
025:  ↓
026:
027:
028:  ∟
029:  ↔
030:  ▲
031:  ▼
032:
033:  !
034:  "
035:  #
036:  $
037:  %
038:  &
039:  '
040:  (
Press enter to continue_
```

```
039:  '
040:  (
Press enter to continue

041:  )
042:  *
043:  +
044:  ,
045:  -
046:  .
047:  /
048:  0
049:  1
050:  2
051:  3
052:  4
053:  5
054:  6
055:  7
056:  8
057:  9
058:  :
059:  ;
060:  <
Press enter to continue_
```

```
059:  ;
060:  <
Press enter to continue

061:  =
062:  >
063:  ?
064:  @
065:  A
066:  B
067:  C
068:  D
069:  E
070:  F
071:  G
072:  H
073:  I
074:  J
075:  K
076:  L
077:  M
078:  N
079:  O
080:  P
Press enter to continue_
```

```
079:  O
080:  P
Press enter to continue

081:  Q
082:  R
083:  S
084:  T
085:  U
086:  V
087:  W
088:  X
089:  Y
090:  Z
091:  [
092:  \
093:  ]
094:  ^
095:  _
096:  `
097:  a
098:  b
099:  c
100:  d
Press enter to continue_
```

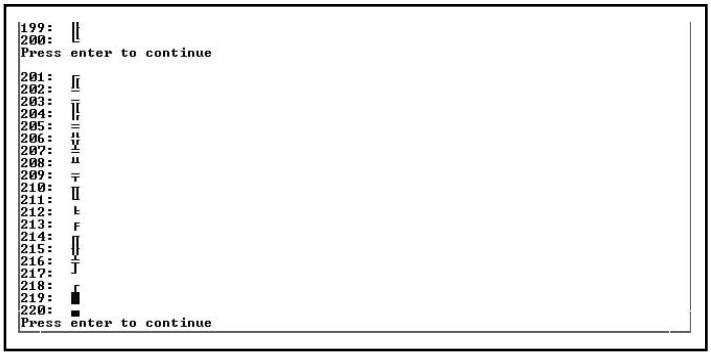
```
099:  c
100:  d
Press enter to continue

101:  e
102:  f
103:  g
104:  h
105:  i
106:  j
107:  k
108:  l
109:  m
110:  n
111:  o
112:  p
113:  q
114:  r
115:  s
116:  t
117:  u
118:  v
119:  w
120:  x
Press enter to continue_
```

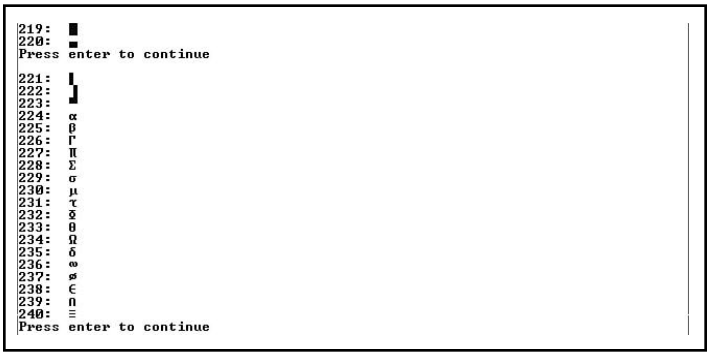
```
119:  w
120:  x
Press enter to continue

121:  y
122:  z
123:  {
124:  |
125:  }
126:  ~
127:  ⌂
128:  Ç
129:  ü
130:  é
131:  â
132:  ä
133:  à
134:  å
135:  ç
136:  ê
137:  ë
138:  è
139:  ï
140:  î
Press enter to continue
```

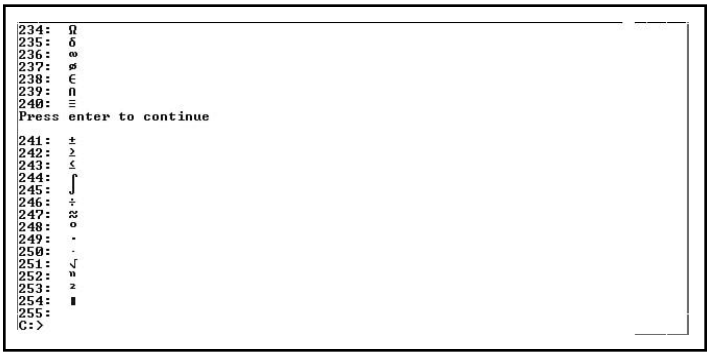
```
139:  ï
140:  î
Press enter to continue

141:  ì
142:  Â
143:  Å
144:  É
145:  æ
146:  Æ
147:  ô
148:  ö
149:  ò
150:  û
151:  ù
152:  ÿ
153:  Ö
154:  Ü
155:  ¢
156:  £
157:  ¥
158:  ₧
159:  ƒ
160:  á
Press enter to continue
```

```
159:  ƒ
160:  á
Press enter to continue

161:  í
162:  ó
163:  ú
164:  ñ
165:  Ñ
166:  ª
167:  º
168:  ¿
169:  ⌐
170:  ¬
171:  ½
172:  ¼
173:  ¡
174:  «
175:  »
176:  ░
177:  ▒
178:  ▓
179:  │
180:  ┤
Press enter to continue
```

```
179:  │
180:  ┤
Press enter to continue

181:  ╡
182:  ╢
183:  ╖
184:  ╕
185:  ╣
186:  ║
187:  ╗
188:  ╝
189:  ╜
190:  ╛
191:  ┐
192:  └
193:  ┴
194:  ┬
195:  ├
196:  ─
197:  ┼
198:  ╞
199:  ╟
200:  ╚
Press enter to continue
```


199: ╟
200: ╚
Press enter to continue
201: ╔
202: ╩
203: ╦
204: ╠
205: ═
206: ╬
207: ╧
208: ╨
209: ╤
210: ╥
211: ╙
212: ╘
213: ╒
214: ╓
215: ╫
216: ╪
217: ┘
218: ┌
219: █
220: ▄
Press enter to continue


219: █
220: ▄
Press enter to continue
221: ▌
222: ▐
223: ▀
224: α
225: ß
226: Γ
227: π
228: Σ
229: σ
230: µ
231: τ
232: Φ
233: Θ
234: Ω
235: δ
236: ∞
237: ø
238: ∈
239: ∩
240: ≡
Press enter to continue


234: Ω
235: δ
236: ∞
237: ø
238: ∈
239: ∩
240: ≡
Press enter to continue
241: ±
242: ≥
243: ≤
244: ⌠
245: ⌡
246: ÷
247: ≈
248: °
249: ∙
250: ·
251: √
252: ⁿ
253: ²
254: ■
255:
C:>

Analysis As stated earlier, nothing in this program should be new to you. Line 10 declares an unsigned character variable, `ch`, which will be used to print the values in the table. A character is the smallest variable type (excluding a bit—covered later). A character can store 256 different values—hence the number of values in the ASCII table. As you can see in line 15, these 256 values are printed starting with 0 and ending with 255. Line 17 does the actual printing. The numeric value is printed first followed by the character value. Both the numeric and character values are of the same variable, `ch`. This line shows that the two are, in essence, equivalent.

Line 19 contains an `if` statement that allows the program to automatically break after printing every 20 values. If your screen can display more lines, you can adjust this number. Line 22 uses the `gets()` function to simply get any information the user may enter on the screen. The variable `trash` was declared to be 256 bytes long because this is the maximum number of characters the keyboard buffer will allow before requiring the enter key to be pressed. Line 25 prints the final value of the table.

Note: If you are tempted to change line 15 in Listing 6.1 to the following so that you can remove line 25, beware! This won't work:

```
for( ch = 0; ch <= 255; ch++ )
```

The maximum value a character can hold is 255. When the variable increments to 256, it actually rolls around so 256 equals 0. Because 0 is less than 255, the loop starts all over!

A few of the values may not print to the screen. This is because values such as a beep (ASCII value 7) cannot be seen. If your computer has a speaker, then when character 7 is printed in the output, you will hear a beep. In addition, you may notice that number 10 of the output precedes a blank line. This value is translated to a line feed. When the line feed is printed by the program, it causes a line to be skipped.

Which Number Systems Are Important?

There are a multitude of number systems available. The number system you should be most familiar with is the decimal, or base 10, system. The decimal system is the

number system that you learn in school. In addition to the decimal system, three other numbers systems are typically referred to when programming. These are binary, octal, and hexadecimal.

Listing 6.2 is a program that enables you to enter a character and then translates the character into its equivalent numeric values. The decimal, hexadecimal, octal, and binary values will all be displayed.

Type

Listing 6.2. A character translation program.

```
1:  /* Program:  list0602.c
2:   * Author:   Bradley L. Jones
3:   * Purpose:  Print numeric values of an entered character.
4:   *=========================================================*/
5:
6:  #include <stdio.h>
7:  #include <stdlib.h>
8:
9:  char *char_to_binary( int );
10:
11: void main(void)
12: {
13:    int ch;
14:    char *rv;
15:
16:    printf("\n\nEnter a character ==>" );
17:    ch = getchar();
18:
19:    printf("\n\nYour character:     %c", ch );
20:    printf("\n\n Decimal value:     %d", ch );
21:    printf("\n Octal value:       %o", ch);
22:    printf("\n Hexadecimal value: %x", ch );
23:
24:    rv = char_to_binary(ch);
25:
26:    printf("\n Binary value:      %s", rv);
27:    printf("\n\nYour character:     %c", ch );
28: }
29:
30: char *char_to_binary( int ch )
31: {
32:   int  ctr;
33:   char *binary_string;
34:   int  bitstatus;
35:
36:   binary_string = (char*) malloc( 9 * sizeof(char) );
37:
38:   for( ctr = 0; ctr < 8; ctr++)
```

continues

Listing 6.2. continued

```
39:    {
40:       switch( ctr )
41:       {
42:         case 0:  bitstatus = ch & 128;
43:                  break;
44:         case 1:  bitstatus = ch & 64;
45:                  break;
46:         case 2:  bitstatus = ch & 32;
47:                  break;
48:         case 3:  bitstatus = ch & 16;
49:                  break;
50:         case 4:  bitstatus = ch & 8;
51:                  break;
52:         case 5:  bitstatus = ch & 4;
53:                  break;
54:         case 6:  bitstatus = ch & 2;
55:                  break;
56:         case 7:  bitstatus = ch & 1;
57:                  break;
58:       }
59:
60:       binary_string[ctr] = (bitstatus) ? '1' : '0';
61:
62:  //     printf( "\nbitstatus = %d, ch = %d, binary_string[%d] = %c",
63:  //              bitstatus, ch, ctr, binary_string[ctr]);
64:    }
65:
66:    binary_string[8] = 0;  /* Null Terminate */
67:
68:    return( binary_string );
69: }
```

Output

```
Enter a character ==>A
Your character:     A

 Decimal value:     65
 Octal value:       101
 Hexadecimal value: 41
 Binary value:      01000001

Your character:     A
```

Analysis

This program uses the conversion parameter within the `printf()` function's string to display the decimal, octal, and hexadecimal values. Lines 16 and 17 prompt the user for a character and store it in the integer variable, `ch`. This could also have been an unsigned character variable; however, when translating characters

to numbers as this program is doing, it is often easier to use an integer. Line 19 uses the `%c` conversion character in the `printf()` to directly print the character. Line 20 uses the `%d` conversion character, which translates the character to a numeric value. Lines 21 and 22 use the `%o` and `%x` conversion values to print the octal and hexadecimal values. Line 26 prints the binary value of the character. The `main()` function ends by once again printing the original character.

Line 24 calls a function to convert the character to its binary value. Some compilers will allow a `%b` conversion character to work. This program has its own conversion function, `char_to_binary()`, in lines 30 through 69. This function converts each digit of the binary number individually. Lines 62 and 63 have been commented out. These lines were added so that the programmer could see each digit being converted. Later today, you will see this conversion function again. At that time, each line will be explained.

DO	DON'T
DON'T be confused by all the number systems. The rest of today will describe and explain each number system in detail.	
DO read the rest of today's material if you do not understand these number systems.	

The Decimal Number System

As stated, the decimal system is the base 10 system that you started learning to count with in kindergarten. Once you realize how the decimal system actually works, the other number systems will be easy to understand.

There are two keys to understanding a number system. The first is to know what number system you are using. Second, you should know the number system's base. In the case of the decimal number system, the base is 10. In fact, the name of the number system will generally stand for the base number. Decimal stands for 10.

The base also states how many different numbers (or characters) are used when representing numbers. In addition, using the base, you can translate numbers from other number systems to the more familiar decimal system.

To aid in the understanding of the different number systems, consider the objects in Figure 6.1. How many objects are in the two pictures?

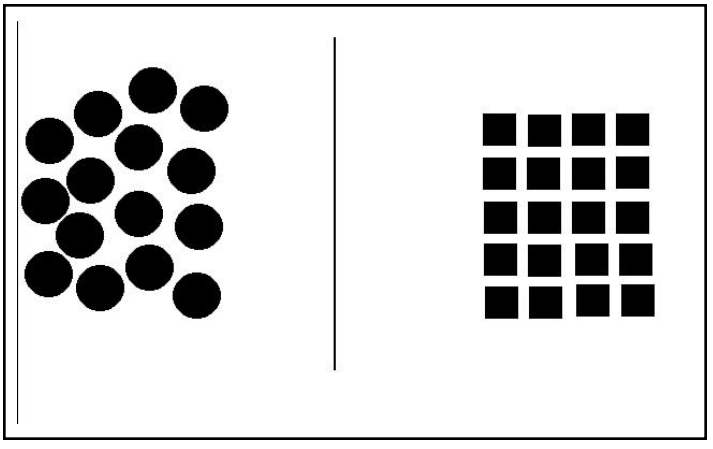


Figure 6.1. *Count the objects on each side.*

You should have answered 15 circles and 20 squares. These are decimal numbers. How did you determine that the answer to the first was a 1 followed by a 5 (fifteen)? You probably just counted and the answer was obvious. However, the logic you may have used to determine this is slightly more complex.

Remembering back, you should know that the right most digit, 5 in the first case, is the "ones" values. The 1 is the "tens" value. Bigger numbers may have "hundreds," "thousands," or more.

As already stated, the decimal number system is base 10. Also stated was that a base 10 system has only 10 digits to use, 0 through 9. The way a number is written is determined by its base. For each digit from right to left, the number is the base, 10 in the case of decimal, to an exponential power. This starts at the right side with the base to the power of 0, and increases by an additional power for each digit to the left. Table 6.1 illustrates this for the decimal number system.

Table 6.1. Decimal digits.

Digit	Base Decimal Value	Digit Equivalent	Name
First	10^0 =	1	Ones
Second	10^1 =	10	Tens

Digit	Base Decimal Value	Digit Equivalent	Name
Third	10^2 =	100	Hundreds
Fourth	10^3 =	1,000	Thousands
Fifth	10^4 =	10,000	Ten-Thousands
Sixth	10^5 =	100,000	Hundred-Thousands
Seventh	10^6 =	1,000,000	Millions

Data in Decimal

As stated earlier, all computer information is represented as numbers. It is possible to view this number in its decimal formats. Listing 6.3 takes a file as a parameter and displays each line in its decimal format instead of characters.

Note: This listing and several of the following use a test file. If you have the disk provided with this book, the test file is called TEST. This file can be created with your text editor. The test file contains the following:

```
1234567890
ABCDEFG
HIJKLMN
OPQRSTU
VWXYZ
abcdefg
hijklmn
opqrstu
vwxyz

!@#$%^&*()_+{}[];:'"<>?/.,‘~
>               <
Once upon a time there were three little pigs...
```

Type

Listing 6.3. Program to type a file in decimal.

```
1:  /* Program:  decdump.c
2:   * Author:   Bradley L. Jones
3:   * Purpose:  This program types out a file. It displays
4:   *           the decimal equivalent of each character.
5:   *=======================================================*/
6:
7:  #include <stdio.h>
8:  #include <string.h>
9:  #include <stdlib.h>
10:
11: main(int argv, char *argc[])
12: {
13:    int ch;
14:    unsigned int  line = 1;
15:
16:    FILE *fp;
17:
18:    if( argv != 2 )
19:    {
20:       printf("\n\nOops!  Proper usage is:");
21:       printf("\n\n%s in_file ", argc[0]);
22:       printf("\n\nOriginal file will be printed in decimal");
23:       return(1);
24:    }
25:
26:    /***  Open the file  ***/
27:    if (( fp = fopen( argc[1], "r" )) == NULL )
28:    {
29:       printf( "\n\nOops!  Error in opening file: %s\n\n", argc[1]);
30:       exit(99);
31:    }
32:
33:    printf("\n%5.5d:  ", line );
34:
35:    while( (ch = fgetc( fp )) != EOF )
36:    {
37:       printf("%d ", ch );
38:
39:       if(ch == '\n')
40:       {
41:         line++;
42:         printf("\n%5.5d:  ", line );
43:       }
44:    }
45:
46:    fclose( fp );
47:
48:    return(0);
49: }
```

Output

```
D:\TYAC>DECDUMP TEST

00001:  49 50 51 52 53 54 55 56 57 48 10
00002:  65 66 67 68 69 70 71 10
00003:  72 73 74 75 76 77 78 10
00004:  79 80 81 82 83 84 85 10
00005:  86 87 88 89 90 10
00006:  97 98 99 100 101 102 103 10
00007:  104 105 106 107 108 109 110 10
00008:  111 112 113 114 115 116 117 10
00009:  118 119 120 121 122 10
00010:  1 2 10
00011:  33 64 35 36 37 94 38 42 40 41 95 43 123 125 91 93 59 58 39
        34 60 62 63 47 46 44 96 126 10
00012:  62 32 32 32 32 32 32 32 32 32 32 32 32 32 32 60 10
00013:  10
00014:  79 110 99 101 32 117 112 111 110 32 97 32 116 105 109 101 32
        116 104 101 114 101 32 119 101 114 101 32 116 104 114 101
        101 32 108 105 116 108 101 32 112 105 103 115 46 46 46 10
00015:  10
00016:
```

Analysis

This program uses a command line argument. Line 18 checks to see if a single parameter was entered along with the program name at the time the user started the program. If not, error messages are printed and the program exits. Line 27 attempts to open the filename entered when the program was started. If the file cannot be opened for reading, an error message is printed and the program exits. The heart of this program is in lines 33 through 44. Line 33 does an initial print of the first line number. Line 35 then reads a character from the file. Line 37 prints the character as a decimal value using the `%d` within the `printf()` function. Line 39 then checks to see if the character is a carriage return. If the character is a carriage return, a new line number is printed. Once the end of the file is reached, the printing of the characters stops, and the file is closed before exiting the program.

This program prints line numbers followed by the data from the file. The output is from using the TEST file described earlier. You should notice that a few extra characters seem to get printed in the output. Most obvious should be the extra number 10 at the end of each line. This is the line feed or carriage return that you would not normally see in a text file. In addition, you should notice that in output line 00012 the spaces are printed as 32s. These values can be seen in the ASCII Character Table.

The Binary Number System

As you can see by the output in Listing 6.3, looking at the decimal representations of characters is not very readable. In addition, these values are really not very helpful. One

of the most descriptive number systems to use with computers is binary. Listing 6.4 is a rewrite of Listing 6.3, except that information is printed in binary.

Type

Listing 6.4. Program to type a file in binary.

```
1:  /* Program:  bindump.c
2:   * Author:   Bradley L. Jones
3:   * Purpose:  This program types a file to the screen.
4:   *           It displays the binary equivalent of each
5:   *           character
6:   *=========================================================*/
7:
8:  #include <stdio.h>
9:  #include <string.h>
10: #include <stdlib.h>
11:
12: char *char_to_binary( int );
13:
14: main(int argv, char *argc[])
15: {
16:    int ch,
17:        letter = 0;
18:
19:    unsigned int line = 1;
20:
21:    FILE *fp;
22:
23:    if( argv != 2 )
24:    {
25:       printf("\n\nOops!  Proper usage is:");
26:       printf("\n\n%s in_file ", argc[0]);
27:       printf("\n\nOriginal file will be printed in Binary.");
28:       exit(1);
29:    }
30:
31:    /***  Open the file  ***/
32:    if (( fp = fopen( argc[1], "r" )) == NULL )
33:    {
34:       printf( "\n\nOops!  Error opening file: %s\n\n", argc[1]);
35:       exit(99);
36:    }
37:
38:    printf("\n%5.5d: ", line );
39:
40:    while( (ch = fgetc( fp )) != EOF )
41:    {
42:
43:       printf("%s ", char_to_binary(ch) );
44:
45:       if(ch == '\n')
```

```
46:        {
47:          line++;
48:          letter = 0;
49:          printf("\n%5.5d: ", line );
50:        }
51:        else
52:        if( ++letter >= 7 )            /* for formatting output */
53:        {
54:           printf("\n       ");
55:           letter = 0;
56:        }
57:
58:     }
59:
60:     fclose( fp );
61:
62:     return(0);
63:  }
64:
65:
66:
67:  char *char_to_binary( int ch )
68:  {
69:    int  ctr;
70:    char *binary_string;
71:    int  bitstatus;
72:
73:    binary_string = (char*) malloc( 9 * sizeof(char) );
74:
75:    for( ctr = 0; ctr < 8; ctr++)
76:    {
77:       switch( ctr )
78:       {
79:         case 0:  bitstatus = ch & 128;
80:                  break;
81:         case 1:  bitstatus = ch & 64;
82:                  break;
83:         case 2:  bitstatus = ch & 32;
84:                  break;
85:         case 3:  bitstatus = ch & 16;
86:                  break;
87:         case 4:  bitstatus = ch & 8;
88:                  break;
89:         case 5:  bitstatus = ch & 4;
90:                  break;
91:         case 6:  bitstatus = ch & 2;
92:                  break;
93:         case 7:  bitstatus = ch & 1;
94:                  break;
95:       }
```

continues

Listing 6.4. continued

```
96:
97:       binary_string[ctr] = (bitstatus) ? '1' : '0';
98:     }
99:
100:    binary_string[8] = 0;  /* Null Terminate */
101:
102:    return( binary_string );
103: }
```

Output

```
D:\TYAC>bindump test

00001: 00110001 00110010 00110011 00110100 00110101 00110110
       00110111 00111000 00111001 00110000 00001010
00002: 01000001 01000010 01000011 01000100 01000101 01000110
       01000111 00001010
00003: 01001000 01001001 01001010 01001011 01001100 01001101
       01001110 00001010
00004: 01001111 01010000 01010001 01010010 01010011 01010100
       01010101 00001010
00005: 01010110 01010111 01011000 01011001 01011010 00001010
00006: 01100001 01100010 01100011 01100100 01100101 01100110
       01100111 00001010
00007: 01101000 01101001 01101010 01101011 01101100 01101101
       01101110 00001010
00008: 01101111 01110000 01110001 01110010 01110011 01110100
       01110101 00001010
00009: 01110110 01110111 01111000 01111001 01111010 00001010
00010: 00000001 00000010 00001010
00011: 00100001 01000000 00100011 00100100 00100101 01011110
       00100110 00101010 00101000 00101001 01011111 00101011
       01111011 01111101 01011011 01011101 00111011 00111010
       00100111 00100010 00111100 00111110 00111111 00101111
       00101110 00101100 01100000 01111110 00001010
00012: 00111110 00100000 00100000 00100000 00100000 00100000
       00100000 00100000 00100000 00100000 00100000 00100000
       00100000 00100000 00100000 00100000 00111100 00001010
00013: 00001010
00014: 01001111 01101110 01100011 01100101 00100000 01110101
       01110000 01101111 01101110 00100000 01100001 00100000
       01110100 01101001 01101101 01100101 00100000 01110100
       01101000 01100101 01110010 01100101 00100000 01110111
       01100101 01110010 01100101 00100000 01110100 01101000
       01110010 01100101 01100101 00100000 01101100 01101001
       01110100 01101100 01100101 00100000 01110000 01101001
       01100111 01110011 00101110 00101110 00101110 00001010
00015: 00001010
00016:
```

Analysis This output was also obtained by running the program with the TEST file that was described earlier in the chapter. Note that there is a lot more data printed out in the output. Because the information is in binary, it is the most accurate representation of what is truly stored.

Looking at the listing, you can see that it is very similar to the Listing 6.3. The `main()` function allows a command line argument to be received (line 14). Lines 23 through 29 verify that one and only one command line parameter was entered. If there were more, or less, then an error message is printed and the program exits. Line 32 attempts to open the file. If the open fails, line 34 prints an error message and exits.

The heart of the program is lines 38 through 63. Line 38 prints the first line number before jumping into a `while` loop. Line 40 begins the loop. Each character is gotten from the file using the `fgetc()` function. The `while` loop continues until the end of the file is reached (EOF). Line 43 prints the binary character using `printf()`. Notice that the string that is printed is the return value from the `char_to_binary()` function. This is the same function used in Listing 6.2 earlier. Line 45 checks to see if the character read—and just printed—was the newline character. If it is, line 47 increments the line number, line 48 resets the letter count, and line 49 prints the new line number on a new line. If the character read is not a newline character, then the `else` condition in lines 51 to 56 is executed. The `else` condition checks to see how many characters have been printed on the line. Because the binary representation of the file can get long, only 7 characters from the file being used are printed on each line. Line 52 checks to see if seven characters have already been printed. If seven characters have been printed, a new line is started that is indented over a few spaces (line 54). The letter count is reset to 0, and the next interaction of the `while` loop occurs.

The `char_to_binary()` function may not be as easy to follow as the rest of the program. Lines 69 to 71 declare three variables that will be used along with the `ch` integer that was passed in. `ch` contains the character that is to be converted. Each character will be translated into a single binary number. Since characters can be any number from 0 to 255, an 8-digit binary number will be needed.

Why eight digits? Consider Figure 6.1 again. This time look at it in the context of binary numbers. How many items are there in the pictures? Instead of the decimal 15 and 20 that you answered before, count the items using the binary system. The answers are 00001111 and 00010100. Just as you had "ones," "tens," "hundreds," and so on. in the decimal system, you have equivalent categories in the binary system. From the word binary you can deduce that there are two different digits that can be used. These are 0 and 1. One object would be 1, two (decimal) objects would be 10.

You should not read this as ten because it is two. The categories for the binary system can be determined using the base just as you did for the decimal system earlier. Table 6.2 illustrates the digit groupings for the binary system.

Table 6.2. Binary digits.

Digit	Base Value	Decimal Equivalent	Digit Name
First	2^0 =	1	Ones
Second	2^1 =	2	Twos
Third	2^2 =	4	Fours
Fourth	2^3 =	8	Eights
Fifth	2^4 =	16	Sixteens
Sixth	2^5 =	32	Thirty-twos
Seventh	2^6 =	64	Sixty-fours
Eighth	2^7 =	128	One-twenty-eights

Only the first eight digits are represented here. This is typically all you will need when converting characters. Eight bits make up a byte. A byte is the amount of space typically used to store a character. Consider the following binary numbers:

```
00000001 equals 1 in decimal
00000010 equals 2 in decimal
00000100 equals 4 in decimal
00000101 equals 4 + 1 or 5 in decimal
11111111 equals 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 or 255 in decimal.
```

To translate the binary numbers to decimal, you simply add the decimal values from Table 6.2 for the corresponding digits that are not zero.

Now look back at the `char_to_binary()` function in Listing 6.4. You can see that lines 79 through 94 have the decimal equivalent values that are listed in Table 6.2. Instead of converting from binary to decimal as the previous examples displayed, the program converts the decimal value to binary. Following the flow of this function, you can see how to convert from decimal to binary. We know that there are only eight digits in the binary number because a character can only be from 0 to 256. The program starts at the left of the eight-digit binary number and determines the value of each digit. Line 75 is a `for` statement that uses the `ctr` variable to keep track of which of the eight digits

is being determined. Line 77 switches to a `case` statement that works with the individual digit. The first time through, the digit being worked on (`ctr`) will be 0. This `case`, in lines 79 and 81, does a binary math statement. The binary AND operator (`&`) is used to determine if the character contains a bit value for 128. This is done by using the binary AND operator with the number you are testing for, in this case 128. If the character does contain the bit for 128, then `bitstatus` will be set to a non-zero number. After doing this, the `switch` statement is left, and the conditional operator is used in line 97. If the value in `bitstatus` does equals zero, then the number did not contain a bit value for 128. If `bitstatus` does not equal zero, you know that the character's decimal value did contain a bit for 128. Because the value of a character cannot be greater than 255, you know that 128 will be divisible at most one time into `ch`. Using the `for` loop, you can then cycle through each bit value for the character's numeric value. This continues through to the eight digits.

Line 100 null-terminates the binary number which is now stored as a string. Line 102 returns this string. Notice that this string is actually a pointer to a character array. Line 73 used the `malloc()` function to allocate the nine characters needed to hold the binary number. This function allocates a string for holding the binary number. If this function were called often, the calling functions should free the binary strings. By not freeing the string, memory is being lost.

Review Tip: You can test if a bit is on or off by using the binary AND operator (`&`). For example, if you AND a character with the the value of 128, then all of the bits will be set to zeros (off) except for the bit in the 128 position. This bit will be left as it is. If it is on, it will remain on.

Note: Listing 6.2 uses the `char_to_binary` function also. It, however, included two extra lines:

```
printf( "\nbitstatus = %d, ch = %d, binary_string[%d] = %c",
          bitstatus, ch, ctr, binary_string[ctr]);
```

These two lines could be added to Listing 6.4 between lines 96 and 98. They print each step of the binary conversion.

The Hexadecimal Number System

As you could see by the previous program, displaying a file in its binary values may be more helpful than looking at its decimal values. However, it is also easy to see that looking at the binary values provides much more information than is really needed. What is needed is a number system that can represent each of the 256 different values of a character and still be easily converted to binary. Actually, using a base 256 number system would provide too many different characters to be useful. The number system that seems to provide the best representation is the hexadecimal, or base 16, system. It takes only two digits to represent all 256 character values.

Table 6.3. Hexadecimal digits.

Digit	Base Value	Decimal Equivalent	Digit Name
First	16^0 =	1	Ones
Second	16^1 =	16	Sixteens
Third	16^2 =	256	Two-hundred fifty-sixes

Looking at Table 6.3, you can see that by the time you get to the third digit of a hexadecimal number, you are already at a number equivalent to 256 in the decimal system. By including 0, you can represent all 256 characters with just two digits! If hexadecimal is new to you, you might be wondering how you can represent 16 characters. Remember the base determines the number of characters that are used in displaying the number. Table 6.4 illustrates the hexadecimal characters and the decimal equivalents.

Table 6.4. The hexadecimal digits.

Hexadecimal Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011

Hexadecimal Digit	Decimal Equivalent	Binary Equivalent
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

The alpha characters in a hexadecimal number can be either upper- or lowercase.

Note: Hexadecimal numbers are generally followed with lowercase h to signify that they are hexadecimal. For example, 10h would be hexadecimal 10, not decimal 10. Its decimal equivalent would be 16.

There is a second reason why hexadecimal numbers are preferred by programmers. It is easy to convert a binary number to and from hexadecimal. Simply convert each of the two digits of the hexadecimal number individually and concatenate the result. Or, if converting from binary to hexadecimal, convert the left four digits to a single hexadecimal number and then convert the right four. Consider the following examples.

Converting from hexadecimal to binary:

- ☐ Hexadecimal value: F1h (or 241 decimal).
- ☐ Converting the first digit, F, to binary yields 1111.
- ☐ Converting the second digit, 1, to binary yields 0001.
- ☐ The total binary equivalent of F1 is 1111 0001 or 11110001.

Converting from binary to hexadecimal:

- ☐ Binary value: 10101001 (or 169 decimal).
- ☐ Converting the first four digits, 1010, yields A in hexadecimal.
- ☐ Converting the second four digits, 1001, yields 9 in hexadecimal.
- ☐ The total hexadecimal equivalent of 10101001 is A9h.

Listing 6.5, HEXDUMP.C, is a rewrite of the programs you have seen before. This program simply prints the hexadecimal values of a file.

Listing 6.5. Program to type a file in hexadecimal.

```
1:   /* Program:  hexdump.c
2:    * Author:   Bradley L. Jones
3:    * Purpose:  This program types a file to the screen.
4:    *           It displays the hexadecimal equivalent of
5:    *           each character
6:    *=====================================================*/
7:
8:   #include <stdio.h>
9:   #include <string.h>
10:  #include <stdlib.h>
11:
12:  main(int argv, char *argc[])
13:  {
14:     int ch;
15:     unsigned int line = 1;
16:
17:     FILE *fp;
18:
19:     if( argv != 2 )
20:     {
21:        printf("\n\nOops!  Proper usage is:");
22:        printf("\n\n%s in_file ", argc[0]);
23:        printf("\n\nOriginal file will be printed in HEX.");
24:        exit(1);
25:     }
26:
```

```
27:     /***  Open the file  ***/
28:     if (( fp = fopen( argc[1], "r" )) == NULL )
29:     {
30:        printf( "\n\nOops!  Error in opening file: %s\n\n",
31:                 argc[1]);
32:        exit(99);
33:     }
34:
35:     printf("\n%5.5d:  ", line );
36:
37:     while( (ch = fgetc( fp )) != EOF )
38:     {
39:        printf("%X ", ch );
40:
41:        if(ch == '\n')
42:        {
43:          line++;
44:          printf("\n%5.5d:  ", line );
45:        }
46:     }
47:
48:     fclose( fp );
49:
50:     return(0);
51:  }
```

Output

```
D:\TYAC>HEXDUMP TEST

00001:  31 32 33 34 35 36 37 38 39 30 A
00002:  41 42 43 44 45 46 47 A
00003:  48 49 4A 4B 4C 4D 4E A
00004:  4F 50 51 52 53 54 55 A
00005:  56 57 58 59 5A A
00006:  61 62 63 64 65 66 67 A
00007:  68 69 6A 6B 6C 6D 6E A
00008:  6F 70 71 72 73 74 75 A
00009:  76 77 78 79 7A A
00010:  1 2 A
00011:  21 40 23 24 25 5E 26 2A 28 29 5F 2B 7B 7D 5B 5D 3B 3A 27 22
        3C 3E 3F 2F
        2E 2C 60 7E A
00012:  3E 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 3C A
00013:  A
00014:  4F 6E 63 65 20 75 70 6F 6E 20 61 20 74 69 6D 65 20 74 68 65
        72 65 20 77
        65 72 65 20 74 68 72 65 65 20 6C 69 74 6C 65 20 70 69 67 73
        2E 2E 2E A
00015:  A
00016:
```

Analysis This program is similar to the binary and decimal dump programs that you have already seen. The big difference is in line 39. In order to print a hexadecimal value in C, you simply need to use the %X specifier. This will automatically print the hexadecimal value.

The Octal Number System

The octal number system is rarely used in C or by C programmers. It is often mentioned because it is easy to convert to using the printf() conversion character, %o. The octal number system is the base 8 number system. As you should conclude from this, there are 8 digits, 0 through 7.

Type **Listing 6.6. Program to type a file in octal.**

```
1:   /* Program:  octdump.c
2:    * Author:   Bradley L. Jones
3:    * Purpose:  This program types a file to the screen.
4:    *           It displays the octal equivalent of each
5:    *           character
6:    *=====================================================*/
7:
8:   #include <stdio.h>
9:   #include <string.h>
10:  #include <stdlib.h>
11:
12:  main(int argv, char *argc[])
13:  {
14:     int ch;
15:     unsigned int line = 1;
16:
17:     FILE *fp;
18:
19:     if( argv != 2 )
20:     {
21:        printf("\n\nOops!  Proper usage is:");
22:        printf("\n\n%s in_file ", argc[0]);
23:        printf("\n\nOriginal file will be printed in Octal.");
24:        exit(1);
25:     }
26:
27:     /***  Open the file  ***/
28:     if (( fp = fopen( argc[1], "r" )) == NULL )
29:     {
30:        printf( "\n\nOops!  Error in opening file: %s\n\n",
31:                 argc[1]);
32:        exit(99);
33:     }
```

```
34:
35:     printf("\n%5.5d:  ", line );
36:
37:     while( (ch = fgetc( fp )) != EOF )
38:     {
39:        printf("%03o ", ch );
40:
41:        if(ch == '\n')
42:        {
43:          line++;
44:          printf("\n%5.5d:  ", line );
45:        }
46:     }
47:
48:     fclose( fp );
49:
50:     return(0);
51: }
```

Output

```
D:\TYAC>OCTDUMP TEST

00001:  061 062 063 064 065 066 067 070 071 060 012
00002:  101 102 103 104 105 106 107 012
00003:  110 111 112 113 114 115 116 012
00004:  117 120 121 122 123 124 125 012
00005:  126 127 130 131 132 012
00006:  141 142 143 144 145 146 147 012
00007:  150 151 152 153 154 155 156 012
00008:  157 160 161 162 163 164 165 012
00009:  166 167 170 171 172 012
00010:  001 002 012
00011:  041 100 043 044 045 136 046 052 050 051 137 053 173 175 133
        135 073 072
        047 042 074 076 077 057 056 054 140 176 012
00012:  076 040 040 040 040 040 040 040 040 040 040 040 040 040 040
        040 074 012
00013:  012
00014:  117 156 143 145 040 165 160 157 156 040 141 040 164 151 155
        145 040 164 150 145 162 145 040 167 145 162 145 040 164 150
        162 145 145 040 154 151 164 154 145 040 160 151 147 163 056
        056 056 012
00015:  012
00016:
```

Analysis

Listing 6.6 is a rewrite of Listing 6.5. Notice that the only difference between these two programs, other than the comments, is the `printf()` conversion specifier in line 39. Instead of using `%x` for hexadecimal, `%03o` is used. The `03` zero pads the corresponding variable three characters. The `o` displays the octal value.

DO	DON'T

DO understand the number systems.

DON'T confuse different number systems. Generally the following rules are used in using numeric constants:

- Octal numbers start with 0, that is, 08 would be octal 8.
- Hexadecimal numbers generally start with x, that means x8 is hexadecimal 8.

  Warning, x08 would be octal!
- Decimal numbers do not start with 0 or x.

A Final Program

Oftentimes, you will find that you want to see a program in more than one format. The programs presented thus far have presented the data using only a single number system. Following is a program that you will find to be more useful. It does have one flaw. It does not print the last few characters of the file.

Listing 6.7. Program to type a file in hexadecimal with the character representations.

```
1:   /* Program:  hex.c
2:    * Author:   Bradley L. Jones
3:    * Purpose:  This program types a file to the screen.
4:    *           The information is presented in its regular
5:    *           form and its hexadecimal equivalent.
6:    * Notes:    This program has an imperfection. The last
7:    *           23 or fewer characters in the file will not
8:    *           be printed.
9:    *=========================================================*/
10:
11:  #include <stdio.h>
12:  #include <string.h>
13:  #include <stdlib.h>
14:
15:  main(int argv, char *argc[])
16:  {
17:     int  ch,
18:          ctr;
```

```
19:     char buffer[24];
20:
21:     FILE *fp;
22:
23:     if( argv != 2 )
24:     {
25:        printf("\n\nOops!  Proper usage is:");
26:        printf("\n\n%s in_file ", argc[0]);
27:        printf("\n\nOriginal file will be printed in HEX.");
28:        return(1);
29:     }
30:
31:     /***  Open the file  ***/
32:     if (( fp = fopen( argc[1], "r" )) == NULL )
33:     {
34:        printf( "\n\nOops!  Error in opening file: %s\n\n",
35:                argc[1]);
36:        exit(99);
37:     }
38:
39:     fread(buffer, 24, sizeof(char), fp );
40:
41:     while( !feof(fp))
42:     {
43:        for( ctr = 0; ctr < 24; ctr++ )
44:        {
45:           if( (ctr % 4) == 0 )
46:              printf(" ");
47:           printf("%02X", buffer[ctr] );
48:        }
49:
50:        printf( "  " );
51:
52:        for( ctr = 0; ctr < 24; ctr++ )
53:        {
54:           if( buffer[ctr] == '\n' )
55:              buffer[ctr] = '.';
56:
57:           printf("%c", buffer[ctr] );
58:        }
59:
60:        printf("\n");
61:
62:        fread(buffer, 24, sizeof(char), fp );
63:     }
64:
65:     fclose( fp );
66:
67:     return(0);
68:  }
```

Output

```
31323334 35363738 39300A41 42434445 46470A48 494A4B4C 1234567890.ABCDEFG.HIJKL
4D4E0A4F 50515253 54550A56 5758595A 0A616263 64656667 MN.OPQRSTU.VWXYZ.abcdefg
0A68696A 6B6C6D6E 0A6F7071 72737475 0A767778 797A0A01 .hijklmn.opqrstu.vwxyz._
020A2140 2324255E 262A2829 5F2B7B7D 5B5D3B3A 27223C3E _.!@#$%^&*()_+{}[];:'"<>
3F2F2E2C 607E0A3E 20202020 20202020 20202020 2020203C ?/.,`~.>               <
0A0A4F6E 63652075 706F6E20 61207469 6D652074 68657265 ..Once upon a time there
20776572 65207468 72656520 6C69746C 65207069 67732E2E  were three little pigs
```

Analysis

As you can see, this program prints the hexadecimal values of the file entered along with the actual character representation. This makes it easy to see what values are actually stored in a file. You also should notice that the new lines and other special characters are printed as single characters with no special processing. This means that the output is an actual representation of what is in the file. The hexadecimal numbers in the output are grouped in sets of four. Every two characters is an individual hexadecimal number. The break between every four is simply for readability.

Summary

This chapter covered many of the number systems that are commonly referred to in programming. While you use the decimal number system every day, it is not the most practical number system to use when dealing with computer data. The binary number system, which consists of two digits, is the most accurate representative for showing the computer's view of data. The hexadecimal system can easily be converted to and from binary. This easy conversion, along with its capability to represent many numbers with just a few digits, makes it the better number system for working with computer data. The octal system is also mentioned since C provides ways of easily converting to it.

Q&A

Q Why would you want to convert characters to decimal values?

A Once you are comfortable with the other number systems, you typically will not use the decimal system. If you are not comfortable with the other number systems, it is easier to add and subtract using decimal. For example, to change an uppercase letter to lowercase, you add 32 (decimal) to it. This makes more sense to most people. If you are looking at a file to determine what is there, the hexadecimal representation is the easiest to read.

Q Is the octal number system important?

A Typically, the octal number system is not used. Most programmers opt to use the hexadecimal system. The binary system is used mainly when doing bit manipulations or when working with binary data files (hence the name binary). Octal is seldom used.

Q What is the difference between the lowercase x specifier and the uppercase X conversion specifier in the `printf()` function?

A The x, or X, specifier prints out the hexadecimal value. If the lowercase x is used, the alpha digits of any hexadecimal numbers will be in lowercase. If the uppercase X is used, then the alpha digits of any hexadecimal numbers will be in uppercase. The difference is simply in the presentation of the hexadecimal numbers.

Workshop

The Workshop provides quiz questions to help you solidify your understanding of the material covered and exercises to provide you with experience using what you've learned.

Quiz

1. Why are numbers important in computing?
2. What determines what numeric value a letter gets translated to?
3. What is meant by decimal value?
4. Why are binary numbers important?
5. Why would you want to use hexadecimal numbers?
6. Why might you want to look at the numeric values of your data?
7. What digits are used in the binary number system?
8. What digits are used in the decimal number system?
9. What digits are used in the octal number system?
10. What digits are used in the hexadecimal number system?

Exercises

1. What are the numeric equivalents of the letter B? Compute for binary, octal, decimal, and hexadecimal. (Don't use the programs from the chapter to answer this!)
2. What are the numeric equivalents of the following character: ☺
3. What are the decimal values of the following ASCII characters?
   a. X
   b. space
   c. x
   d. 1
   e. ♥
4. What characters do the following numbers represent?
   a. 65
   b. 63
   c. 5
   d. 125
   e. 57
5. Rewrite Listing 6.2 to accept a decimal number instead of a character. Have the program print each of the other numeric representations.
6. Write a function that takes advantage of the information provided in the ASCII Character Table. This function should convert a lowercase letter to an uppercase letter, or an uppercase letter to lowercase. Do not use library functions provided by the compiler! Name the function `switch_case()`.
7. **BUG BUSTER:**

```
char x;
for ( x = 'a'; x < 'Z'; x++ )
{
    printf( "%c ", x);
}
```

8. **ON YOUR OWN:** Write a program that reads a file and counts the number of occurrences of each character. Remember to differentiate between characters that may appear the same such as the difference between a null (decimal value 0) and a space (decimal value 32).