Exercises

Section 20.1

*1. Show the output produced by each of the following program fragments. Assume that i, j, and k are unsigned short variables.

```
(a) i = 8; j = 9;
    printf("%d", i >> 1 + j >> 1);
(b) i = 1;
    printf("%d", i & ~i);
(c) i = 2; j = 1; k = 0;
    printf("%d", ~i & j ^ k);
(d) i = 7; j = 8; k = 9;
    printf("%d", i ^ j & k);
```

2. Describe a simple way to "toggle" a bit (change it from 0 to 1 or from 1 to 0). Illustrate the technique by writing a statement that toggles bit 4 of the variable i.

*3. Explain what effect the following macro has on its arguments. You may assume that the arguments have the same type.

```
#define M(x,y)  ((x)^=(y), (y)^=(x), (x)^=(y))
```

4. In computer graphics, colors are often stored as three numbers, representing red, green, and blue intensities. Suppose that each number requires eight bits, and we'd like to store all three values in a single long integer. Write a macro named MK_COLOR with three parameters (the red, green, and blue intensities). MK_COLOR should return a long in which the last three bytes contain the red, green, and blue intensities, with the red value as the last byte and the green value as the next-to-last byte.

5. Write macros named GET_RED, GET_GREEN, and GET_BLUE that, when given a color as an argument (see Exercise 4), return its 8-bit red, green, and blue intensities.

6. (a) Use the bitwise operators to write the following function:

```
unsigned short swap_bytes(unsigned short i);
```

swap_bytes should return the number that results from swapping the two bytes in i. (Short integers occupy two bytes on most computers.) For example, if i has the value 0x1234 (00010010 00110100 in binary), then swap_bytes should return 0x3412 (00110100 00010010 in binary). Test your function by writing a program that reads a number in hexadecimal, then writes the number with its bytes swapped:

```
Enter a hexadecimal number (up to four digits): 1234
Number with bytes swapped: 3412
```

Hint: Use the %hx conversion to read and write the hex numbers.

(b) Condense the swap_bytes function so that its body is a single statement.

7. Write the following functions:

```
unsigned int rotate_left(unsigned int i, int n);
unsigned int rotate_right(unsigned int i, int n);
```

rotate_left should return the result of shifting the bits in i to the left by n places, with the bits that were "shifted off" moved to the right end of i. (For example, the call `rotate_left(0x12345678, 4)` should return `0x23456781` if integers are 32 bits long.) `rotate_right` is similar, but it should "rotate" bits to the right instead of the left.

Ⓦ 8. Let `f` be the following function:

```
unsigned int f(unsigned int i, int m, int n)
{
  return (i >> (m + 1 - n)) & ~(~0 << n);
}
```

(a) What is the value of `~(~0 << n)`?
(b) What does this function do?

9. (a) Write the following function:

```
int count_ones(unsigned char ch);
```

`count_ones` should return the number of 1 bits in `ch`.
(b) Write the function in part (a) without using a loop.

10. Write the following function:

```
unsigned int reverse_bits(unsigned int n);
```

`reverse_bits` should return an unsigned integer whose bits are the same as those in `n` but in reverse order.

11. Each of the following macros defines the position of a single bit within an integer:

```
#define SHIFT_BIT 1
#define CTRL_BIT  2
#define ALT_BIT   4
```

The following statement is supposed to test whether any of the three bits have been set, but it never displays the specified message. Explain why the statement doesn't work and show how to fix it. Assume that `key_code` is an `int` variable.

```
if (key_code & (SHIFT_BIT | CTRL_BIT | ALT_BIT) == 0)
  printf("No modifier keys pressed\n");
```

12. The following function supposedly combines two bytes to form an unsigned short integer. Explain why the function doesn't work and show how to fix it.

```
unsigned short create_short(unsigned char high_byte,
                            unsigned char low_byte)
{
  return high_byte << 8 + low_byte;
}
```

*13. If `n` is an `unsigned int` variable, what effect does the following statement have on the bits in `n`?

```
n &= n - 1;
```

Hint: Consider the effect on `n` if this statement is executed more than once.

Section 20.2 Ⓦ 14. When stored according to the IEEE floating-point standard, a `float` value consists of a 1-bit sign (the leftmost—or most significant—bit), an 8-bit exponent, and a 23-bit fraction, in that order. Design a structure type that occupies 32 bits, with bit-field members corresponding to the sign, exponent, and fraction. Declare the bit-fields to have type `unsigned int`. Check the manual for your compiler to determine the order of the bit-fields.

*15. (a) Assume that the variable s has been declared as follows:

```
struct {
  int flag: 1;
} s;
```

With some compilers, executing the following statements causes 1 to be displayed, but with other compilers, the output is –1. Explain the reason for this behavior.

```
s.flag = 1;
printf("%d\n", s.flag);
```

(b) How can this problem be avoided?

Section 20.3

16. Starting with the 386 processor, x86 CPUs have 32-bit registers named EAX, EBX, ECX, and EDX. The second half (the least significant bits) of these registers is the same as AX, BX, CX, and DX, respectively. Modify the `regs` union so that it includes these registers as well as the older ones. Your union should be set up so that modifying EAX changes AX and modifying AX changes the second half of EAX. (The other new registers will work in a similar fashion.) You'll need to add some "dummy" members to the `word` and `byte` structures, corresponding to the other half of EAX, EBX, ECX, and EDX. Declare the type of the new registers to be `DWORD` (double word), which should be defined as `unsigned long`. Don't forget that the x86 architecture is little-endian.

Programming Projects

1. Design a union that makes it possible to view a 32-bit value as either a `float` or the structure described in Exercise 14. Write a program that stores 1 in the structure's sign field, 128 in the exponent field, and 0 in the fraction field, then prints the `float` value stored in the union. (The answer should be –2.0 if you've set up the bit-fields correctly.)