A. In what way does our code fail to comply with the C standard?

B. Modify the code to run properly on any machine for which data type `int` is at least 32 bits.

C. Modify the code to run properly on any machine for which data type `int` is at least 16 bits.

2.68 ◆◆

Write code for a function with the following prototype:

```
/*
 * Mask with least signficant n bits set to 1
 * Examples: n = 6 --> 0x3F, n = 17 --> 0x1FFFF
 * Assume 1 <= n <= w
 */
int lower_one_mask(int n);
```

Your function should follow the bit-level integer coding rules (page 164). Be careful of the case $n = w$.

2.69 ◆◆◆

Write code for a function with the following prototype:

```
/*
 * Do rotating left shift.  Assume 0 <= n < w
 * Examples when x = 0x12345678 and w = 32:
 *    n=4 -> 0x23456781, n=20 -> 0x67812345
 */
unsigned rotate_left(unsigned x, int n);
```

Your function should follow the bit-level integer coding rules (page 164). Be careful of the case $n = 0$.

2.70 ◆◆

Write code for the function with the following prototype:

```
/*
 * Return 1 when x can be represented as an n-bit, 2's-complement
 * number; 0 otherwise
 * Assume 1 <= n <= w
 */
int fits_bits(int x, int n);
```

Your function should follow the bit-level integer coding rules (page 164).

2.71 ◆

You just started working for a company that is implementing a set of procedures to operate on a data structure where 4 signed bytes are packed into a 32-bit `unsigned`. Bytes within the word are numbered from 0 (least significant) to 3 (most significant). You have been assigned the task of implementing a function for a machine using two's-complement arithmetic and arithmetic right shifts with the following prototype:

```
/* Declaration of data type where 4 bytes are packed
   into an unsigned */
typedef unsigned packed_t;

/* Extract byte from word.  Return as signed integer */
int xbyte(packed_t word, int bytenum);
```

That is, the function will extract the designated byte and sign extend it to be a 32-bit `int`.

Your predecessor (who was fired for incompetence) wrote the following code:

```
/* Failed attempt at xbyte */
int xbyte(packed_t word, int bytenum)
{
    return (word >> (bytenum << 3)) & 0xFF;
}
```

A. What is wrong with this code?

B. Give a correct implementation of the function that uses only left and right shifts, along with one subtraction.

2.72 ◆◆

You are given the task of writing a function that will copy an integer `val` into a buffer `buf`, but it should do so only if enough space is available in the buffer.

Here is the code you write:

```
/* Copy integer into buffer if space is available */
/* WARNING: The following code is buggy */
void copy_int(int val, void *buf, int maxbytes) {
    if (maxbytes-sizeof(val) >= 0)
            memcpy(buf, (void *) &val, sizeof(val));
}
```

This code makes use of the library function `memcpy`. Although its use is a bit artificial here, where we simply want to copy an `int`, it illustrates an approach commonly used to copy larger data structures.

You carefully test the code and discover that it *always* copies the value to the buffer, even when `maxbytes` is too small.

A. Explain why the conditional test in the code always succeeds. *Hint:* The `sizeof` operator returns a value of type `size_t`.

B. Show how you can rewrite the conditional test to make it work properly.

```
void *malloc(size_t size);
void *memset(void *s, int c, size_t n);
```

2.77 ◆◆

Suppose we are given the task of generating code to multiply integer variable `x` by various different constant factors K. To be efficient, we want to use only the operations `+`, `-`, and `<<`. For the following values of K, write C expressions to perform the multiplication using at most three operations per expression.

A. $K = 17$

B. $K = -7$

C. $K = 60$

D. $K = -112$

2.78 ◆◆

Write code for a function with the following prototype:

```
/* Divide by power of 2. Assume 0 <= k < w-1 */
int divide_power2(int x, int k);
```

The function should compute $x/2^k$ with correct rounding, and it should follow the bit-level integer coding rules (page 164).

2.79 ◆◆

Write code for a function `mul3div4` that, for integer argument `x`, computes $3 *$ `x`/4 but follows the bit-level integer coding rules (page 164). Your code should replicate the fact that the computation `3*x` can cause overflow.

2.80 ◆◆◆

Write code for a function `threefourths` that, for integer argument `x`, computes the value of $\frac{3}{4}$`x`, rounded toward zero. It should not overflow. Your function should follow the bit-level integer coding rules (page 164).

2.81 ◆◆

Write C expressions to generate the bit patterns that follow, where a^k represents k repetitions of symbol a. Assume a w-bit data type. Your code may contain references to parameters `j` and `k`, representing the values of j and k, but not a parameter representing w.

A. $1^{w-k}0^k$

B. $0^{w-k-j}1^k0^j$

2.82 ◆

We are running programs where values of type `int` are 32 bits. They are represented in two's complement, and they are right shifted arithmetically. Values of type `unsigned` are also 32 bits.

We generate arbitrary values x and y, and convert them to unsigned values as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
/* Convert to unsigned */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
```

For each of the following C expressions, you are to indicate whether or not the expression *always* yields 1. If it always yields 1, describe the underlying mathematical principles. Otherwise, give an example of arguments that make it yield 0.

A. `(x<y) == (-x>-y)`

B. `((x+y)<<4) + y-x == 17*y+15*x`

C. `~x+~y+1 == ~(x+y)`

D. `(ux-uy) == -(unsigned)(y-x)`

E. `((x >> 2) << 2) <= x`

2.83 ◆◆

Consider numbers having a binary representation consisting of an infinite string of the form $0.y\,y\,y\,y\,y\,y \cdots$, where y is a k-bit sequence. For example, the binary representation of $\frac{1}{3}$ is $0.01010101 \cdots$ ($y = 01$), while the representation of $\frac{1}{5}$ is $0.001100110011 \cdots$ ($y = 0011$).

A. Let $Y = B2U_k(y)$, that is, the number having binary representation y. Give a formula in terms of Y and k for the value represented by the infinite string. *Hint:* Consider the effect of shifting the binary point k positions to the right.

B. What is the numeric value of the string for the following values of y?
   (a) 101
   (b) 0110
   (c) 010011

2.84 ◆

Fill in the return value for the following procedure, which tests whether its first argument is less than or equal to its second. Assume the function `f2u` returns an unsigned 32-bit number having the same bit representation as its floating-point argument. You can assume that neither argument is *NaN*. The two flavors of zero, $+0$ and -0, are considered equal.

```
int float_le(float x, float y) {
    unsigned ux = f2u(x);
    unsigned uy = f2u(y);
```