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.

2.82 ◆

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`