Format A		Format B	
Bits	Value	Bits	Value
1 01111 001	$\frac{-9}{8}$	1 0111 0010	$\frac{-9}{8}$
0 10110 011	______	______	______
1 00111 010	______	______	______
0 00000 111	______	______	______
1 11100 000	______	______	______
0 10111 100	______	______	______

2.89 ◆

We are running programs on a machine where values of type `int` have a 32-bit two's-complement representation. Values of type `float` use the 32-bit IEEE format, and values of type `double` use the 64-bit IEEE format.

We generate arbitrary integer values `x`, `y`, and `z`, and convert them to values of type `double` as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to double */
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

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. Note that you cannot use an IA32 machine running GCC to test your answers, since it would use the 80-bit extended-precision representation for both `float` and `double`.

A. `(float) x == (float) dx`

B. `dx - dy == (double) (x-y)`

C. `(dx + dy) + dz == dx + (dy + dz)`

D. `(dx * dy) * dz == dx * (dy * dz)`

E. `dx / dx == dz / dz`

2.90 ◆

You have been assigned the task of writing a C function to compute a floating-point representation of 2^x. You decide that the best way to do this is to directly construct the IEEE single-precision representation of the result. When x is too small, your routine will return 0.0. When x is too large, it will return $+\infty$. Fill in the blank portions of the code that follows to compute the correct result. Assume the

```
    movq    %rax, 184(%rdi)
    ret
```

What are the values of A and B? (The solution is unique.)

3.69 ◆◆◆

You are charged with maintaining a large C program, and you come across the following code:

```
typedef struct {
    int first;
    a_struct a[CNT];
    int last;
} b_struct;

void test(long i, b_struct *bp)
{
    int n = bp->first + bp->last;
    a_struct *ap = &bp->a[i];
    ap->x[ap->idx] = n;
}
```

The declarations of the compile-time constant `CNT` and the structure `a_struct` are in a file for which you do not have the necessary access privilege. Fortunately, you have a copy of the `.o` version of code, which you are able to disassemble with the OBJDUMP program, yielding the following disassembly:

```
   void test(long i, b_struct *bp)
   i in %rdi, bp in %rsi
0000000000000000 <test>:
   0:   8b 8e 20 01 00 00       mov    0x120(%rsi),%ecx
   6:   03 0e                   add    (%rsi),%ecx
   8:   48 8d 04 bf             lea    (%rdi,%rdi,4),%rax
   c:   48 8d 04 c6             lea    (%rsi,%rax,8),%rax
  10:   48 8b 50 08             mov    0x8(%rax),%rdx
  14:   48 63 c9                movslq %ecx,%rcx
  17:   48 89 4c d0 10          mov    %rcx,0x10(%rax,%rdx,8)
  1c:   c3                      retq
```

Using your reverse engineering skills, deduce the following:

A. The value of `CNT`.

B. A complete declaration of structure `a_struct`. Assume that the only fields in this structure are `idx` and `x`, and that both of these contain signed values.

3.70 ◆◆◆

Consider the following union declaration:

```
union ele {
    struct {
        long *p;
        long y;
    } e1;
    struct {
        long x;
        union ele *next;
    } e2;
};
```

This declaration illustrates that structures can be embedded within unions.

The following function (with some expressions omitted) operates on a linked list having these unions as list elements:

```
void proc (union ele *up) {
    up->_________ = *(_________) - _________;
}
```

A. What are the offsets (in bytes) of the following fields:

```
e1.p       ________
e1.y       ________
e2.x       ________
e2.next    ________
```

B. How many total bytes does the structure require?

C. The compiler generates the following assembly code for `proc`:

```
void proc (union ele *up)
up in %rdi
proc:
  movq    8(%rdi), %rax
  movq    (%rax), %rdx
  movq    (%rdx), %rdx
  subq    8(%rax), %rdx
  movq    %rdx, (%rdi)
  ret
```

On the basis of this information, fill in the missing expressions in the code for `proc`. *Hint:* Some union references can have ambiguous interpretations. These ambiguities get resolved as you see where the references lead. There