Quick Review

N bits represent 2^N things:

How many bits do you need to represent 768 things?

10 bits. $2^9 < 768 < 2^{10}$.

Kind men give terminal pets extra zebra yolk:

2^{67} = 128 Ei

With 8 bits, what are the bit patterns for the following? For the last row, what is the decimal value of the given bit pattern?

	Unsigned	Sign & Magnitude	One's Complement	Two's Complement
`-1`	N/A	0b1000 0001	0b1111 1110	0b1111 1111
`MAX`	0b1111 1111	0b0111 1111	0b0111 1111	0b0111 1111
`MIN`	0b0000 0000	0b1111 1111	0b1000 0000	0b1000 0000
`0x83`	131	-3	-124	-125

In general, with N bits the max/min for unsigned is ___$2^{(N-1)}$ / 0___, and for two's complement the max/min is ___$2^{(N-1)}$ - 1 / $-2^{(N-1)}$___.

What are the advantages and disadvantages of each integer representation?

Unsigned
- Pros: can represent large positive values
- Cons: no negatives

Sign & Mag.
- Pros: negatives
- Cons: lots - complex hardware, 2 zeroes, weird binary odometer behavior

One's Complement
- Pros: negatives, fixed binary odometer
- Cons: 2 zeroes, hardware still a little complex

Two's Complement
- Pros: negatives, fixed binary odometer, 1 zero
- Cons: none?

Complete the following function `convert()` that takes an unsigned integer as an argument, and returns it's value when interpreted as a sign and magnitude number:

```
int convert(unsigned int signMag){
  // This is only one way to do it, many others valid
  int sign = signMag & (0x80000000);
  int mag = signMag & (~0x80000000);
  if(sign)
    return -mag;
  else
    return mag;


}
```

C details

```
int* p1, p2, p3, p4;
```

Did I just declare four pointers?

No, this declares one pointer (p1), and 3 ints (p2, p3, and p4).

```
if ((5/4) * 100 == 125) printf("C can do math!\n");
```

Did it print?

No. 5/4 is done as integer division, and so the result of 1.25 is truncated to 1, and 100 is not equal to 125. To get this to work, cast the 5 or the 4 to a double before dividing, or write (5.0/4)

Pointers

Writing the function swap and complete its call.

```
int foo = 5;
int baz = 42;
swap(&foo, &baz);
printf("foo is %d, baz is %d\n", foo, baz);
/* foo is 42, baz is 5 */

void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

What is the output of the following program given this snapshot of memory?

Variable (if any)		a	b	c	p					x	y	
Address	...	171	172	173	174	175	176	177	...	655	656	...
Initial Value		15	19	-5	171	0	255	4		-1	8	
		3	144	170	176							
		144	656	-12								

```
int main(int argc, char * argv[]){
      int a = 3, b = 144, c = 170;
      int *p;
      printf("%d, %d, %d\n", *p, p, &p);
      p = (int *) foo(a,&c);
      printf("%d, %d, %d\n", *p, p, &p);
      bar(&a, &b);
      printf("%d, %d, %d\n", a, b, c);
      return 0;
}
```

```
int foo (int x, int * y){
      *y = -12;
      return x + (int) y;
}

void bar (int * x, int * y){
      *x = *y;
      *y = (int) &y;
}
```

```
3, 171, 174
255, 176, 174
144, 656, -12
```

Bonus Question

What does this function do?

```
int bitcount (unsigned int n)   {
   int count = 8 * sizeof(int) ;
   n ^= (unsigned int) - 1 ;
   while (n)  {
      count-- ;
      n &= (n - 1) ;
   }
   return count ;
}
```