Quick Review
N bits represent $2^N$ things:
How many bits do you need to represent 768 things?

Kind men give terminal pets extra zebra yolk:
$2^{67} = \ldots$

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?

<table>
<thead>
<tr>
<th></th>
<th>Unsigned</th>
<th>Sign &amp; Magnitude</th>
<th>One’s Complement</th>
<th>Two’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0x83$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In general, with N bits the max/min for unsigned is ____________, and for two’s complement the max/min is ____________.

What are the advantages and disadvantages of each integer representation?

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:

```c
int convert(unsigned int signMag){
}
```

C details
```c
int* p1, p2, p3, p4;
Did I just declare four pointers?

if ((5/4) * 100 == 125) printf("C can do math!\n");
Did it print?
Pointers
Writing the function swap and complete its call.

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

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

<table>
<thead>
<tr>
<th>Variable (if any)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>p</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>...</td>
<td>171</td>
<td>172</td>
<td>173</td>
<td>174</td>
<td>175</td>
</tr>
<tr>
<td>Initial Value</td>
<td>15</td>
<td>19</td>
<td>-5</td>
<td>171</td>
<td>0</td>
<td>255</td>
</tr>
</tbody>
</table>

```c
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;
}
```

Bonus Question
What does this function do?

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