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

10 bits.

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

Fixed Width Numbers

<table>
<thead>
<tr>
<th></th>
<th>Unsigned</th>
<th>Sign and Magnitude</th>
<th>One's Complement</th>
<th>Two's Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def:</td>
<td>Standard binary numbers.</td>
<td>First bit is sign, rest is unsigned magnitude</td>
<td>First bit is sign, rest is magnitude (but inverted if negative)</td>
<td>First bit is sign, rest is magnitude. (Add one after inverting negatives)</td>
</tr>
<tr>
<td>Cons:</td>
<td>No negatives!</td>
<td>Two zeros. Complicated HW required. Weird overflow loop.</td>
<td>Two zeros. Still somewhat complicated HW.</td>
<td>Can’t negate most negative number.</td>
</tr>
</tbody>
</table>

What is the value of the following bytes if interpreted as each of the above types?

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0010 0010</td>
<td>0b1111 1111</td>
<td>0b1001 0111</td>
<td></td>
</tr>
<tr>
<td>38 unsigned</td>
<td>255 unsigned</td>
<td>151 unsigned</td>
<td></td>
</tr>
<tr>
<td>38 SM</td>
<td>-127 SM</td>
<td>-23 SM</td>
<td></td>
</tr>
<tr>
<td>38 One’s</td>
<td>-0 One’s</td>
<td>-104 One’s</td>
<td></td>
</tr>
<tr>
<td>38 Two’s</td>
<td>-1 Two’s</td>
<td>-105 Two’s</td>
<td></td>
</tr>
</tbody>
</table>

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){
    if (signMag>>31){ // easy way to grab a specific bit
        return -1*(signMag&0x7fffffff);//teach them what masking is
    }
    else return signMag;
}
```

Klingon Word of the Week: ghun (gargle sound-oon) - to program (a computer)
Pointers

<table>
<thead>
<tr>
<th>Pointer Type</th>
<th>The Address Operator (&amp;)</th>
<th>The Dereference Operator (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *x;</td>
<td>x = &amp;y;</td>
<td>*x = 5;</td>
</tr>
</tbody>
</table>

Tells the compiler to interpret the variable’s value as an address. Returns the address of the variable provided. Used only for pointer assignment. Essentially “follows” the pointer to allow for access to the referenced data.

Write a function that swaps the value of two integers:

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

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 |
Initial Value     | 15  | 19  | -5  | 171 | 0   | 255 |
                   | 3   | 144 | 170 | 171 | 0   | 4   |
                   | 144 | 656 | -12 | 171 | 172 |
```

```
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

Klingon Word of the Week: ghun (gargle sound-oon) - to program (a computer)