CS240 Supplemental Practice – Bits

1) Conversions – Convert the following numbers to the base given in the question mark
   a. 10001111_2 = 143

   b. 127_{10} = 7F

   c. 0xFA = 250

   d. 10001111 01010000 10101011_2 = 0x8F50AB

   e. 10111111_2 = 191

   f. 0x23 = 35

2) In class, we discussed an example called shift and mask that extracted the 2nd most significant byte from a 32-bit integer. Write a function in C that can extract any of the four bytes where the most significant byte is byte 3 and the least significant byte is byte 0. The function header has been provided to you where `number` is a 32-bit integer and `byteNum` is the byte to be extracted:

```
int getByte(int number, int byteNum){

      return (number >> (byteNum * 8)) & 0xFF;

}
```

3) Provide an example number in hex where this expression to extract the most significant byte could lead to an erroneous result: `(number && 0xFF000000) >> 24`. Explain how such an error could occur and whether it matters whether the shift is logical or arithmetic.

   Any integer whose most significant byte begins with a 1. Such an example might be 0xFFFFFFFF. The issue is that a right shift could potentially fill the upper slots with ones if the shift is arithmetic. In fact, it will in this instance because `number` is signed. We

haven’t discussed signed versus unsigned numbers yet but it turns out the type of shift is determined by the type of the number. Logical shift won’t produce an error because the upper slots are always filled with zeroes. In general, it is usually better to shift and then mask so you do not run into these sorts of errors.