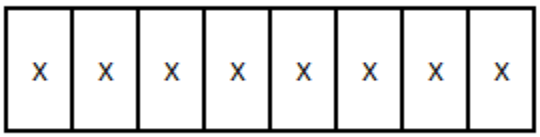
PLAYBOOK NOTES: BIT MANIPULATION – PART 1

Let's review what we know about a few simple data types:

A character (char) is one byte of information. We also know that every byte has 8-bits and every bit is a 1 or a 0.

So, a character in memory (which is just all 1's and 0's) would look like this:



We can see from the picture that there are 8 X's that each represent a 1 or a 0.

Since there are 8 bits in a byte, we know that there are 2^8(256) possibilities with a range of 0 to 255.

Another data type would be an integer (int). We know that an integer is 4 bytes or 32 bits.

We could look at that picture as well, but I think your getting the idea.

Why is this important? Because bit manipulators allow us to make adjustments to the data at the bit level.

We will look at four operators:

1) Bitwise AND (&)
2) Bitwise OR (|)
3) Shift left (<<)
4) Shift right (>>)

Bitwise AND (&):

The first thing you should notice is that the *bitwise AND-operator* is a single ampersand (&) which is different from a *logical AND-operator* (&&).

With a logical AND-operator, we look at two variables:

Variable 1	Variable 2	Variable 1 && Variable 2
F	F	False
F	T	False
T	F	False
T	T	True

A Bitwise AND is similar, except, now we will compare the bits of each variable.

Examine the picture below:

BIT:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1:	1	1	1	1	0	0	0	0
Byte 2:	1	0	1	0	1	0	1	0
Byte 1 & Byte 2	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**

Starting with bit 0, you can see that bit 0 for byte 1 and byte 2 are 0. (0 AND 0 = 0)

Looking at bit 3, you can see that bit 3 for byte 1 is 0 and byte 2 is a 1 (O AND 1 = 0)

Looking at bit 4, you can see that bit 4 for byte 1 is 1 and byte 2 is a 0 (1 AND 0 = 0)

Looking at bit 5, you can see that bit 5 for byte 1 is 1 and byte 2 is a 1 (1 AND 1 = 1)

Thus, the following code:

```
int byteOne = 240; // Binary: 1111 0000
int byteTwo = 170; // Binary: 1010 1010

int answer;

// Answer will equal 160, Binary: 1010 0000
answer = byteOne & byteTwo;
```

Notice we used integers instead of char. We started with a char because it's only one byte of data.

Even though an integer is 32 bits, we are only using the first byte to keep the example simple.

Using the bitwise AND operator in a more meaningful way:

We can use the AND-operator to determine if a bit is a 1 or 0.

Assume we have the following byte of information:

BIT:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1:	1	0	0	0	1	1	1	1

If want to determine if the 5th bit is active (equal to 1) or not active (equal to 0), we would use a bitwise AND with a number where *only* the 5th bit is active. This number is often referred to as a mask.

The following is a picture of MASK_5.

BIT:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1:	0	0	1	0	0	0	0	0

BIT:	7	6	5	4	3	2	1	0
Byte 1:	1	0	0	0	1	1	1	1
Byte 2:	1	0	1	0	1	0	1	0
Byte 1 & Byte 2	0	0	0	0	0	0	0	0

```
// Set macro MASK_5 to 32 because 32 in binary is 0010 0000
// (Notice the 5th bit is active in the binary number)
#define MASK_5 32

int code;
int bit_5;

// Set code to the value in the example
code = 143 // Binary: 1000 1111

// Determine if the fifth bit is active in the code:
bit_5 = code & MASK_5;
```

If the 5th bit is active in the code, then the answer to the AND operation will be equal to MASK_5. (Remember, all the other bits are AND 0 any anything AND 0 will always be 0.

In this example, the 5th bit was equal to 0, so the 5th bit of the answer is also equal to 0.

Putting this in an if statement might look like this:

```
if ((code & MASK_5) == MASK_5)
{
```

One might wonder if there is a way to move the 5th bit to bit 0. The answer is yes and we will cover that in part 3 of the Playbook Notes.