PLAYBOOK NOTES: BIT MANIPULATION – PART 2

Bitwise OR (|):

The first thing you should notice is that the *bitwise OR-operator* is a single vertical bar (|) which is different from a *logical OR-operator* (||).

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

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

A Bitwise OR 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	0	0	0	1	0	1	1
Byte 2:	0	0	1	0	1	1	0	0
Byte 1 | Byte 2	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**

Starting with bit 0, you can see that bit 0 for byte 1 is a 1 and byte 2 is a 0. (1 OR 0 = 1)

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

Looking at bit 3, you can see that bit 3 for byte 1 and byte 2, the bit is a 1 (1 OR 1 = 1)

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

Thus, the following code:

```
int byteOne = 139; // Binary: 1000 1011
int byteTwo = 44; //  Binary: 0010 1100

int answer;

// Answer will equal 175, Binary: 1010 1111
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 OR operator in a more meaningful way:

We can use the OR operator to set the bit of a variable (we will call this variable code).

If you want to set the 6th bit as active (equal to 1), we would use the bitwise OR-operator with a number where *only* the 6th bit is active (set to 1). This number is often referred to as a mask.

Assume we have the following byte of information:

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

We will set the fifth bit of the code using a mask called MASK_5 that has only the 5th bit as a 1.

The following is a picture of MASK_5.

BIT:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MASK_5:	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:	0	0	1	0	0	0	0	0
Byte 1 | Byte 2	1	0	1	0	1	1	1	1

```
// 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;

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

// Set the fifth bit of the code to 1:
code = code | MASK_5;

//
// Code is now equal to 175, Binary: 1010 111
//
```

In this example, we are setting the 5th bit of the code to 1. Because we are using the OR operator, the value of the 5th bit of code is not relevant.

Any number OR 1 is always equal to 1. The 5th bit is guaranteed to be set to 1after the OR operation.