일반적인 논리 연산자 vs 비트 단위 논리 연산자

Regular Logical Operators : &&, ||, and !

```
bool have_apple = true;
bool like_apple = true;

if (have_apple && like_apple)
    eat_apple();
```

Bitwise Logical Operators :
- Bitwise NOT ~ Tilde
- Bitwise AND & Ampersand
- Bitwise OR | Vertical bar
- Bitwise EXCLUSIVE OR ^ Caret

비트 단위 논리 연산자가 필요한 이유

8 bytes
(64 bits)

```
unsigned char has_sword         = 1;
unsigned char has_shield        = 0;
unsigned char has_magic_potion  = 0;
unsigned char has_shoes         = 1;
unsigned char has_gun           = 0;
unsigned char has_pet           = 1;
unsigned char has_guntlet       = 0;
unsigned char has_arrow         = 0;
```

vs

8 bits

```
unsigned char items = 148; // Binary 10010100
```

$$148 = 2^7 + 2^4 + 2^2$$

- 위와 같이 여러 옵션들을 메모리를 아끼면서 사용할 수 있음

# Bitwise AND &

```
unsigned char a = 6;
unsigned char b = 5;

printf("%hhu", a & b);
```

| Decimal | Binary |
|---|---|
| $6 = 2^2 + 2^1$ | 00000110 |
| $5 = 2^2 + 2^0$ | 00000101 |
| $4 = 2^2$ | 00000100 |

- binary에서 둘 다 1이면 1

# Bitwise OR |

```
unsigned char a = 6;
unsigned char b = 5;

printf("%hhu", a | b);
```

| Decimal | Binary |
|---|---|
| $6 = 2^2 + 2^1$ | 00000110 |
| $5 = 2^2 + 2^0$ | 00000101 |
| $7 = 2^2 + 2^1 + 2^0$ | 00000111 |

- binary에서 둘 중 하나만 1이면 1

# Bitwise Exclusive OR ^

```
unsigned char a = 6;
unsigned char b = 5;

printf("%hhu", a ^ b);
```

| Decimal | Binary |
|---|---|
| $6 = 2^2 + 2^1$ | 00000110 |
| $5 = 2^2 + 2^0$ | 00000101 |
| $3 = 2^1 + 2^0$ | 00000011 |

- binary에서 둘 다 1이면 0

# Bitwise Not ~

```
unsigned char a = 6;

printf("%hhu", ~a);
```

| Decimal | Binary |
|---|---|
| $6 = 2^2 + 2^1$ | 00000110 |
| 249 | 11111001 |

$$249 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^0$$

- binary에서 0은 1로, 1은 0으로 변환