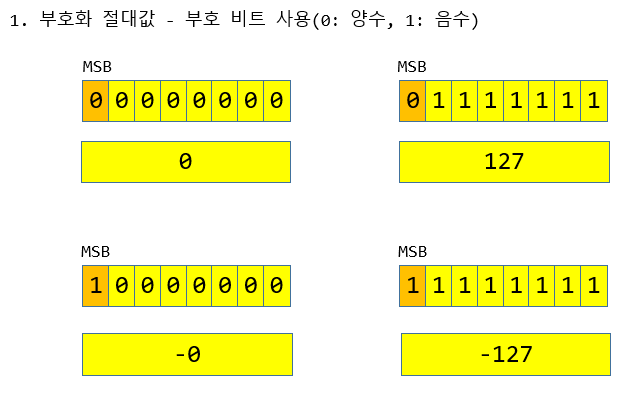
char(1) <= short(2) <= int <= long(4)

cpu가 16비트에서는 int -> 2바이트

cpu가 32비트에서는 int -> 4바이트

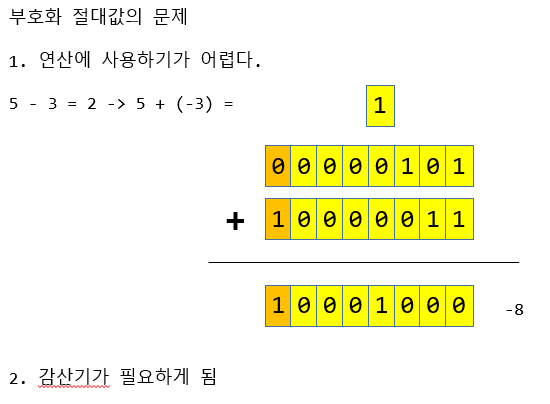
(cpu가 한번에 처리할수 있는양에 따라 16비트/32비트)

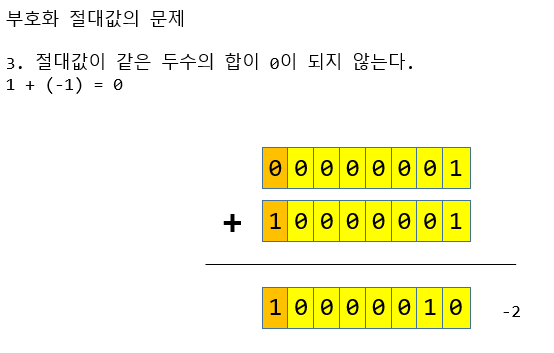
그러나 cpu가 64비트라고 해도 int가 8바이트가 되는 것은 아니다.( 기존에 cpu에서는 int가 4바이트이므로 문제가 생기므로)

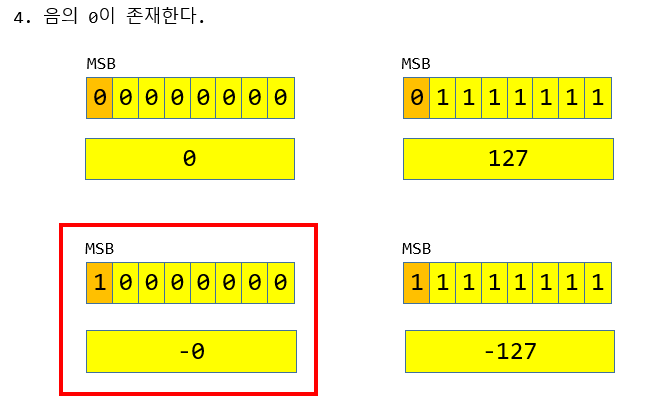


컴퓨터(CPU)가 정수를 처리하는 방법

1. 부호화 절대값
2. 1의 보수
3. 2의 보수



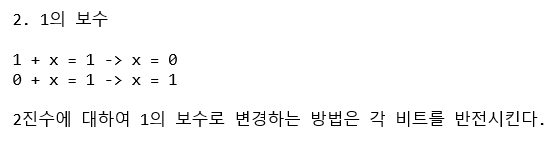


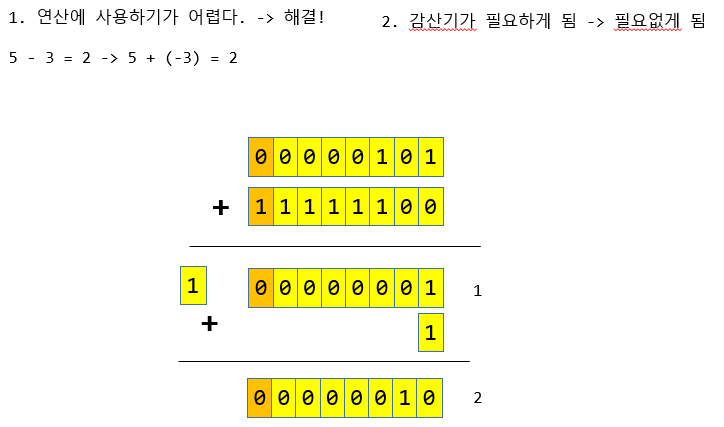


<부호화 절대값의 문제점>

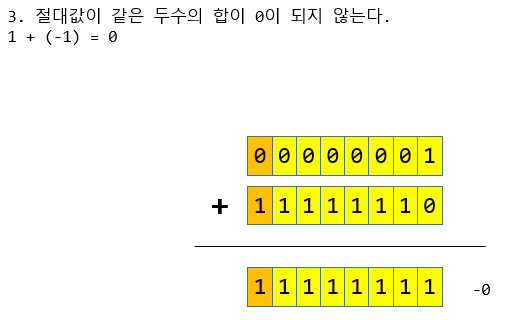
🡪이러한 문제를 해결하기 위해서 ‘보수’를 사용한다.

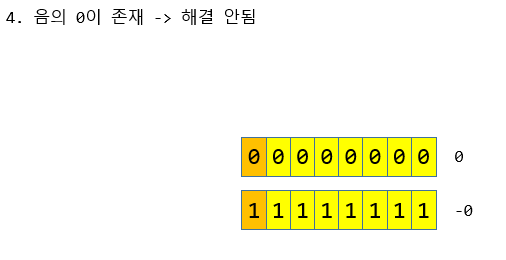
5 – 3 = 5 + (-3) -> -3을 보수로! 즉 5 + 7 = 12 인데 한자리 계산이므로 12에서 1을 떼면 2.



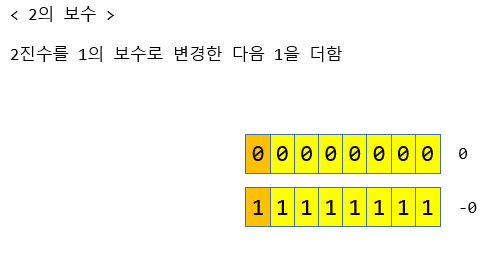


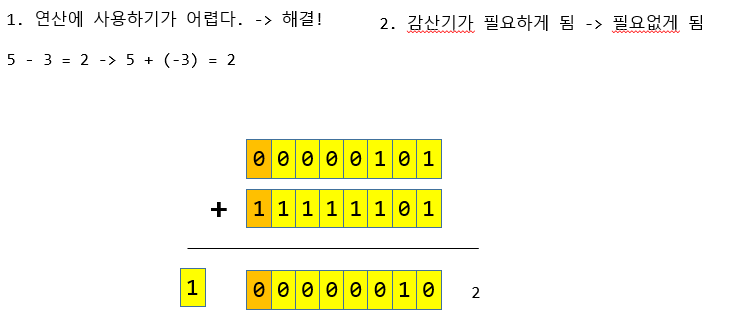
<1의 보수를 통한 해결책>

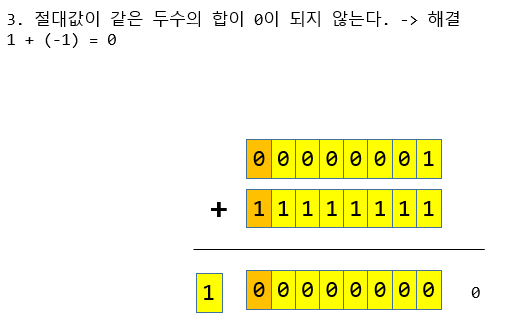


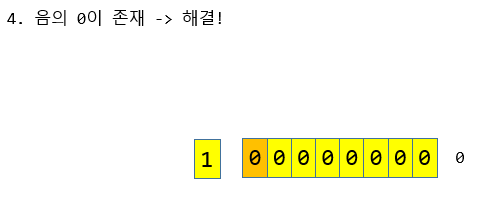


그러나 ‘1의 보수’를 쓴다 하더라도 2가지의 문제가 해결이 안되는데..

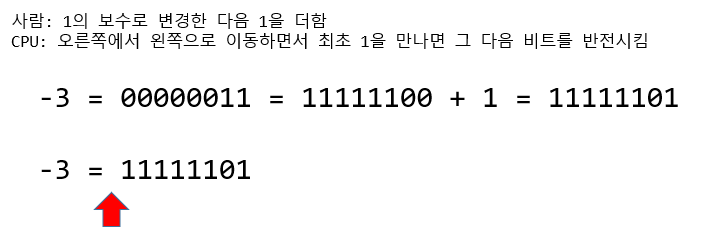




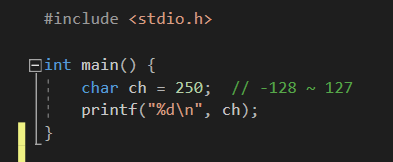


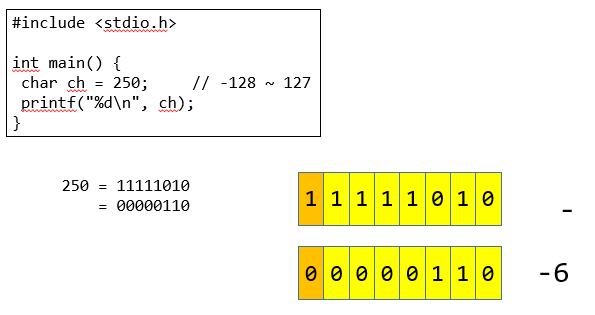


위의 2가지 문제점을 해결하기 위한 방법으로 ‘2의 보수’를 사용한다.



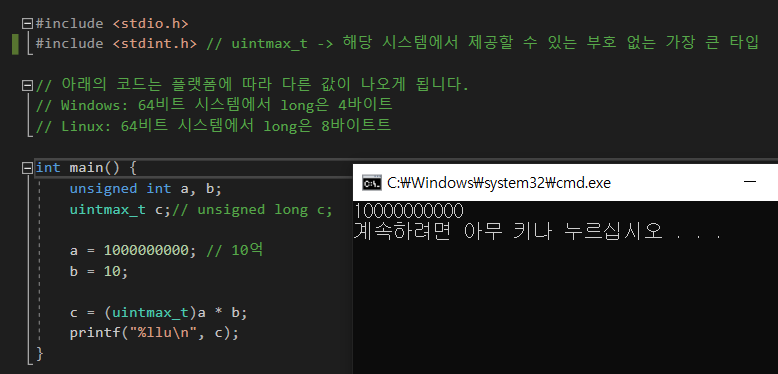
2의 보수시 사람과 cpu의 차이. (cpu의 방식은 다르므로 성능상으로 결코 떨어지지않음)





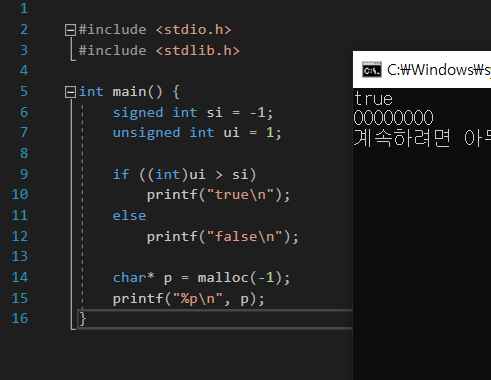
최초 1을 만나면 보수처리를 해준다. -> 00000110으로 바뀜.

답은 -6이 나오게 된다. (2의 보수)

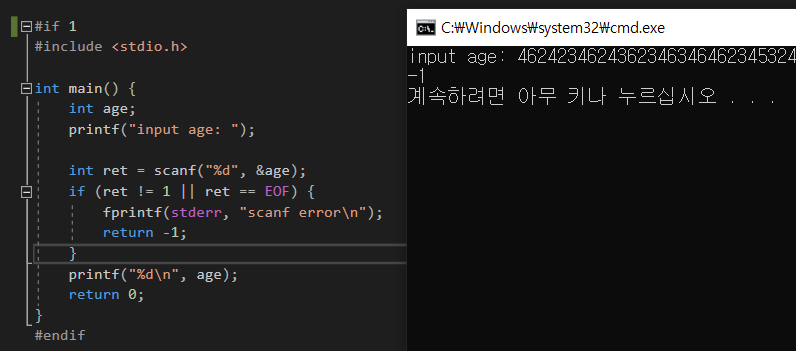


windows에서는 64비트 시스템에서 long은 4바이트

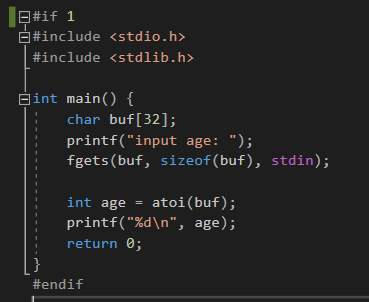
linux에서는 64비트 시스템에서 long은 8바이트로 처리된다.



unsinged int 와 int의 값비교



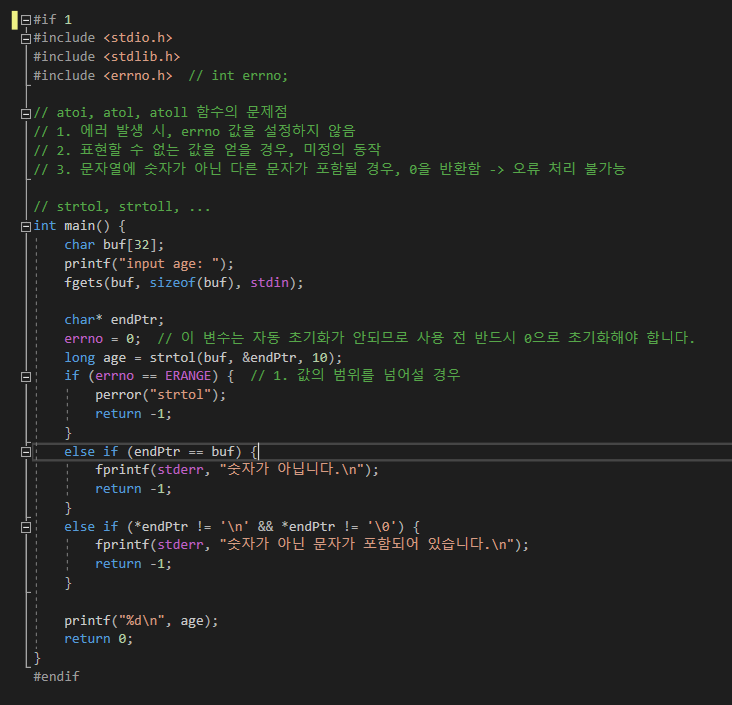
scanf가 문자나 문자열은 탐지 가능하나 엄청큰 숫자를 넣게되면 탐지가 안된다..(그래서 scanf는 위험한 코드이다..)



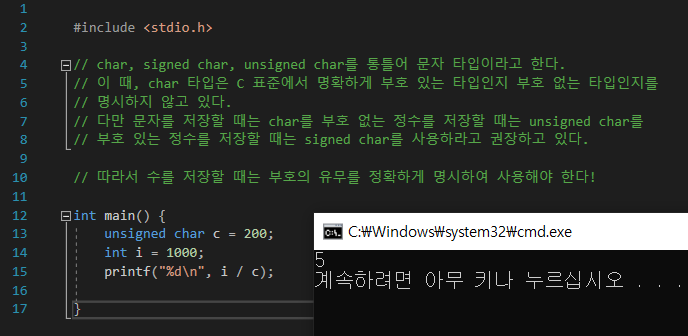
그래서 해결책이 나이를 문자열로 입력을 받은후 atoi로 int로 바꾸는 것인데 이것도 문제가있다..

* atoi / atol / atoll 함수의 문제점

1. 에러 발생 시, errno 값을 설정하지 않음
2. 표현할 수 없는 값을 얻을 경우 미정의 동작 처리됨.
3. 문자열에 숫자가 아닌 다른 문자가 포함될 경우 0을 반환함->오류처리 불가능



위와 같은 문제를 해결하기 위해 사용하는 것이 strtol, strtoll, … 등이 있다.



그냥 char 와 int의 연산을 하게 되면 -17이라는 이상한 값이 나온다.

즉, char / signed char / unsigned char를 통틀어 문자 타입이라고 하는데, 이때 char 타입은 C표준에서 명확하게 부호 있는 타입인지 부호없는 타입인지를 명시하고있지 않다.

**‘문자를 저장할 때는 char’**를

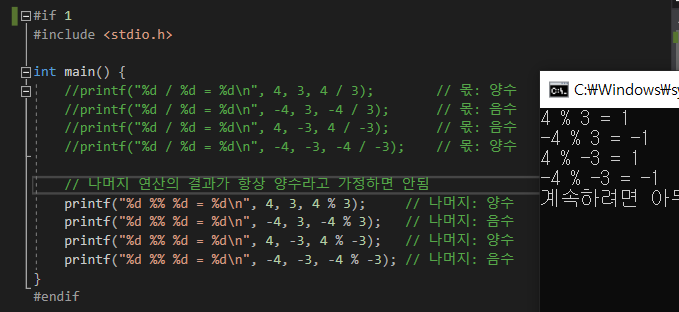
**‘부호 없는 정수를 저장할때는 unsigned char’**를

**‘부호있는 정수를 저장할 때는 signed char’**를

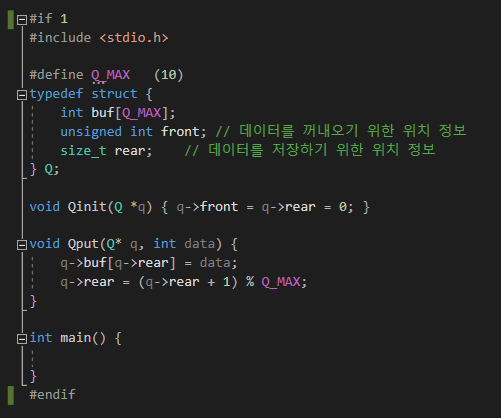
사용하라고 권장하고있다.

따라서 수를 저장할 때는 부호의 유무를 정확하게 명시하여 사용해야한다.

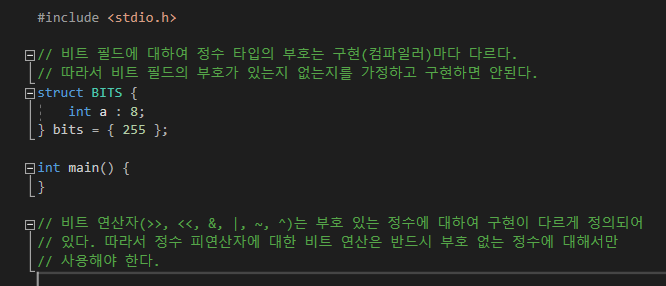
그래서 위 코드에서는 char 앞에 unsigned char를 붙여줘야 정상적인 5가 출력된다.



나머지 연산의 결과가 항상 양수라고 가정하면 안된다.



q->rear + 1 이 음수라면 %는 피제수의 영향을 받으므로 q->rear도 음수가 된다. 그러므로 앞에 unsigned int나 size\_t를 붙여 음수가 될 가능성을 없애주면 된다

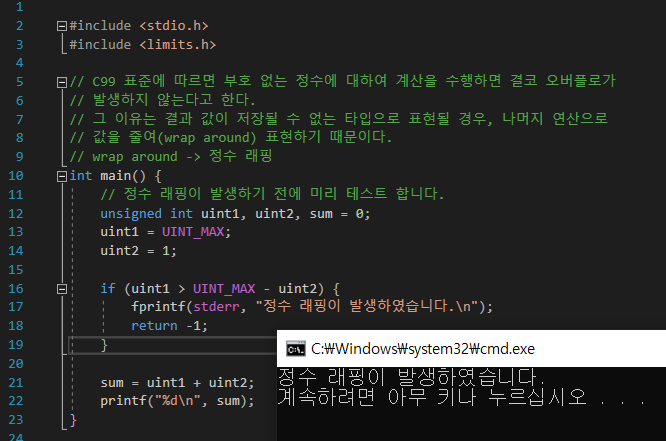


(int a 를 8비트로 쓸 때 부호가 잇는지 없는지를 가정해서는 안된다. 그래서 반드시 앞에 부호를 명시 해줘야한다.) 그렇게 되면 -1이 될수도 255가 될수도 있다..

즉, 비트필드에 대하여 정수타입의 부호는 구현(컴파일러)마다 다르다.

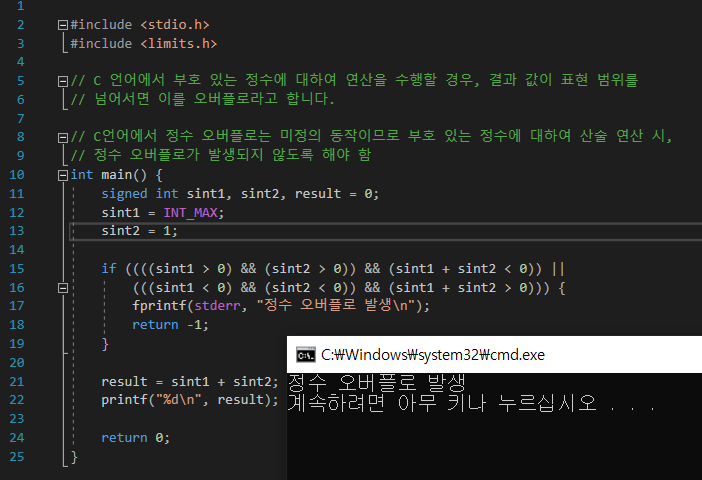
따라서 비트 필드의 부호가 있는지 없는지를 가정하고 구현하면 안된다.

->비트 연산자(>>, <<, &, | , ~, ^)는 부호있는 정수에 대하여 구현이 다르게 정의되어있다. 따라서 정수 피연산자에 대한 비트 연산은 반드시 부호없는 정수에 대해서만 사용해야한다.



값의 범위를 넘어선 정수 오버플로우(부호있는 정수)가 아니다.. 랩어라운드(wrap aroupd 부호없는 정수 -> 정수 랩핑) 라고 하는데 , 부호 없는 정수의 계산은 **예측이 가능**하다. (자동차 계기판처럼 9999에서 0000으로 가는 식으로 예측 가능(모듈러스 연산))

uint1 + uint2 > UINT\_MAX 에서 탐지를 하면되지만 불가능하다.왜냐하면 좌항에서 이미 잘려나가고 없기 때문에 uint2를 우항으로 이항시키면 잘려나가는 것 없이 탐지가 가능할 것이다.



c언어에서 부호있는 정수에 대해 연산을 수행할경우 결과값이 표현범위를 넘어서면 이를 오버플로우 라고한다.

c언어에서 정수 오버플로우는 미정의동작이므로 부호 있는 정수에 대하여 산술연산시, 정수 오버플로우가 발생되지 않도록 해야한다. (컴파일/런타임 둘다 오류가 안난다.. -버그)

cf. INT\_MIN + (-1) 은 언더플로우라고하지않는다. 이것도 값의 표현범위를 벗어난 것이므로 똑같이 오버플로우라고한다. 위 코드에는 이에대한 조건문도 있음