2018 시스템 프로그래밍
- Lab 03 -

제출일자	2018.10.12
분 반	01
이 름	함지희
학 번	201702087

Datalab bit.c

1	bitAnd

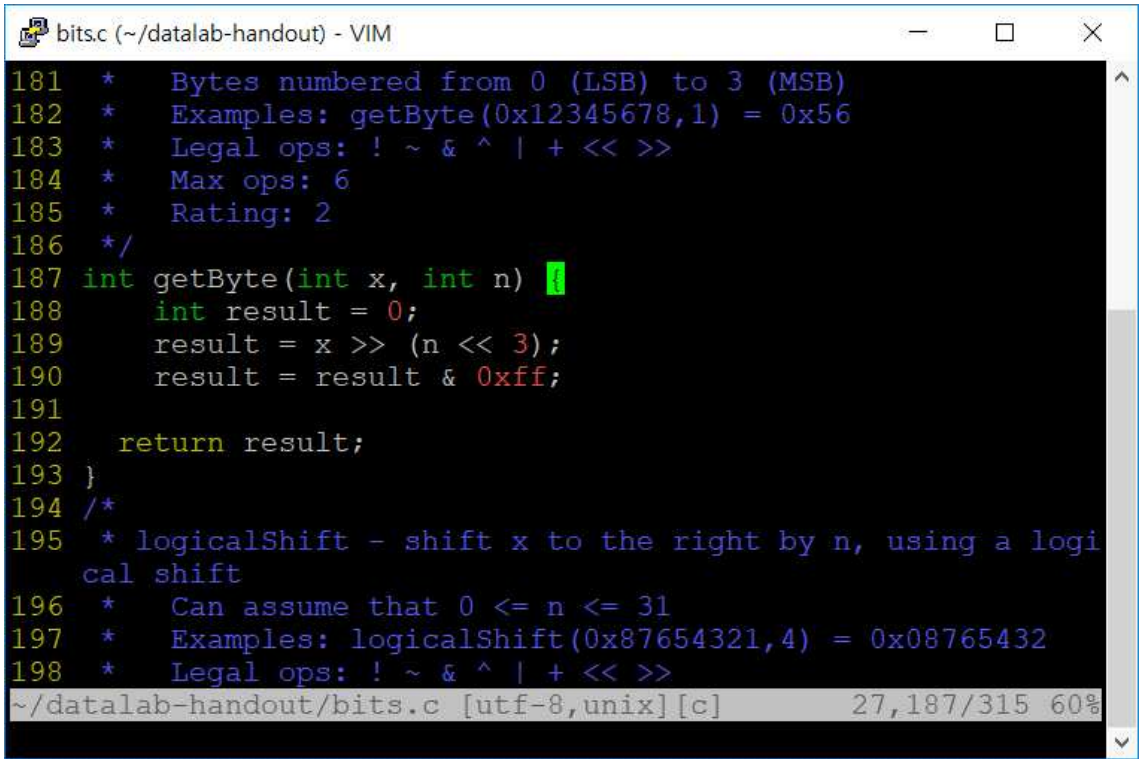
bits.c (~/datalab-handout) - VIM

```
166 /* We do not support C11 <threads.h>.  */
167 /*
168  * bitAnd - x&y using only ~ and |
169  *   Example: bitAnd(6, 5) = 4
170  *   Legal ops: ~ |
171  *   Max ops: 8
172  *   Rating: 1
173  */
174 int bitAnd(int x, int y) {
175     int result= 0;
176     result = ~(~x | ~y);
177     return result;
178 }
179 /*
180  * getByte - Extract byte n from word x
181  *   Bytes numbered from 0 (LSB) to 3 (MSB)
182  *   Examples: getByte(0x12345678,1) = 0x56
183  *   Legal ops: ! ~ & ^ | + << >>
184  *   Max ops: 6
~/datalab-handout/bits.c [utf-8,unix][c]          1,178/315 55%
```

드모르간의 법칙을 사용한다.

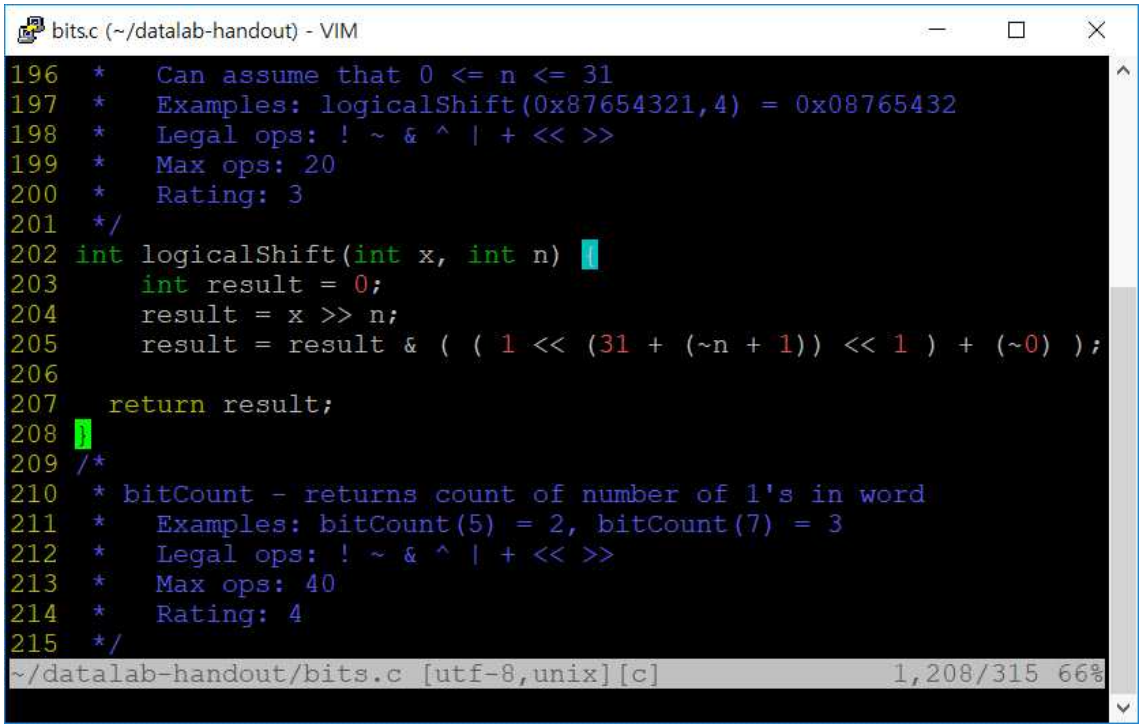
~(X|Y) = (~X&~Y) 이므로, ~(~X|~Y) = X&Y 이다.

2	getByte



1Byte는 8bit이기 때문에 8n만큼 오른쪽으로 쉬프트해준다.
(왼쪽으로 쉬프트하면 2가 곱해지므로 3번 쉬프트해서 n에 8을 곱해줌)
그러면 내가 얻고싶은 byte가 오른쪽끝 8bit에 위치하게 된다.
0xff와 &를 하면 오른쪽끝 8bit를 제외하고 다 0이 되기 때문에 내가 얻고싶은 byte만 살릴 수 있다.

3	logicalShift



logicalShift는 오른쪽으로 쉬프트해주면 빈자리를 다 0으로 채운다.
음수는 최상위비트가 1이기 때문에 그냥 쉬프트해주면 arithmetishift로 빈자리를 1로 채우게 된다. 그래서 이런 경우에 대비해서 빈자리를 0으로 만들어 줘야한다.
일단 n만큼 쉬프트를 해준다 (x에 따라 빈자리가 n개의 1로 채워지거나 0으로 채워짐)
총 32비트이므로 n만큼의 빈자리를 0으로 만들기 위해 맨앞 비트부터 n개의 0 , 32 - n 개의 1로 이루어진 수와 &를 해주어야한다.
2의 n제곱에 1을 빼면 앞은 다 0이고 n번째bit부터 다 1인 이진수가 나옴
(1을 31-n 만큼 왼쪽으로 쉬프트 해준 후에 한번 더 왼쪽으로 쉬프트 해준다. 거기에다 1을 빼주면 위에서 말한 수가 나온다. 이 수와 결과값을 &하면 됨)

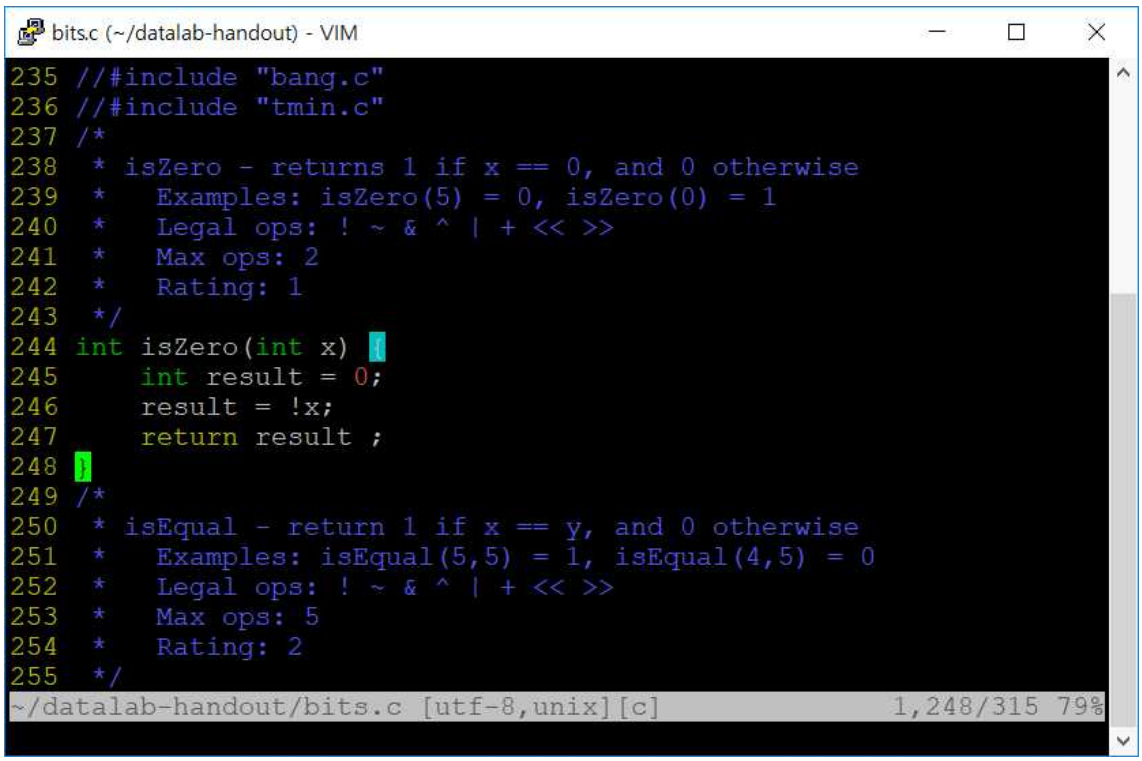
4	bitCount

bits.c (~/datalab-handout) - VIM

```
214  *   Rating: 4
215  */
216 int bitCount(int x) {
217     int result = 0;
218     int filter = 0x1;
219     filter = filter | filter << 8;
220     filter = filter | filter <<16;
221
222     result = x & filter;
223     result += x >> 1 & filter;
224     result += x >> 2 & filter;
225     result += x >> 3 & filter;
226     result += x >> 4 & filter;
227     result += x >> 5 & filter;
228     result += x >> 6 & filter;
229     result += x >> 7 & filter;
230     result = result + ( result >> 16 );
231     result = ( result + (result>>8) ) & 0xff;
232
233   return result;
234 }
~/datalab-handout/bits.c [utf-8,unix][c]                1,234/315 72%
```

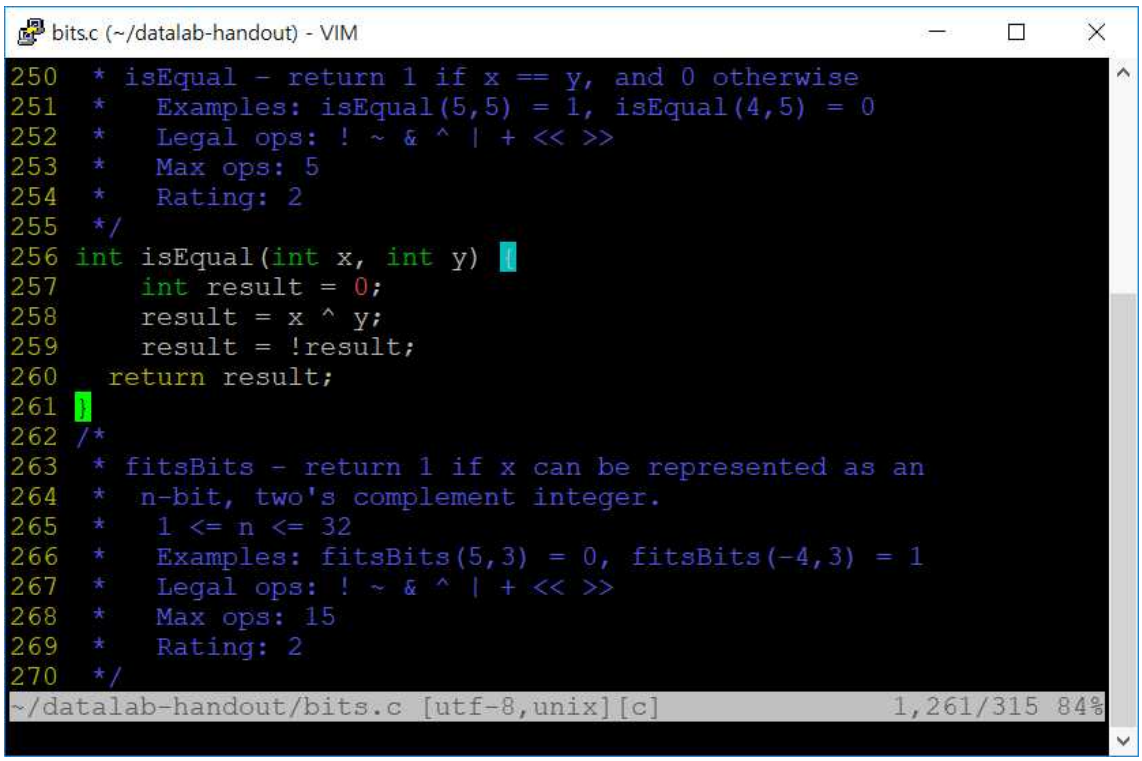
필터를 설정하고 x를 오른쪽으로 1비트씩 움직여서 필터와 &를 해줘서 필터가 1 인부분에 걸리면 1이 나오게하여 1의 개수를 세는 것이다.
필터는 0000 0001//0000 0001//0000 0001//0000 0001 이 되게한다. 1byte씩 나눠서 검사하면 비트를 7번만 움직이고도 1을 알아낼 수 있기 때문이다.
X&filter를 하면 x의 각 바이트의 첫 번째 자리가 1인곳은 결과값의 각 바이트의 첫 번째자리에 그대로 1로 저장된다. 그 다음에 x를 1만큼 오른쪽으로 쉬프트하고 위 과정을 반복하면 x의 각 바이트의 두 번째 자리가 1 인 곳이 결과값의 각 바이트의 첫 번째 자리에 1로 나오게 된다. 이런식으로 x를 총 7번 비트이동시키면서 세고 결과값에 더해준다면 결과값에는...
(3번 byte의 1개수)//(2번 byte의 1개수)//(1번 byte의 1개수)//(0번 byte의 1개수) 가 된다.
이 결과값을 오른쪽으로 16이동하여 3,2번 byte의 1의 개수와 1,0번 byte의 1의 개수를 더해주고 그걸 다시 결과값에 저장한 후 그 값을 또 8만큼 오른쪽으로 이동해서 1번 byte와 0번 byte를 더해준다. 더해주었기 때문에 0번byte자리 외의 자리에도 1들이 남아 있을 것이므로 0xff와 &를 해주어 0번 byte만 나오게끔한다. 그러면 1의 개수를 셀 수 있다.

5	isZero



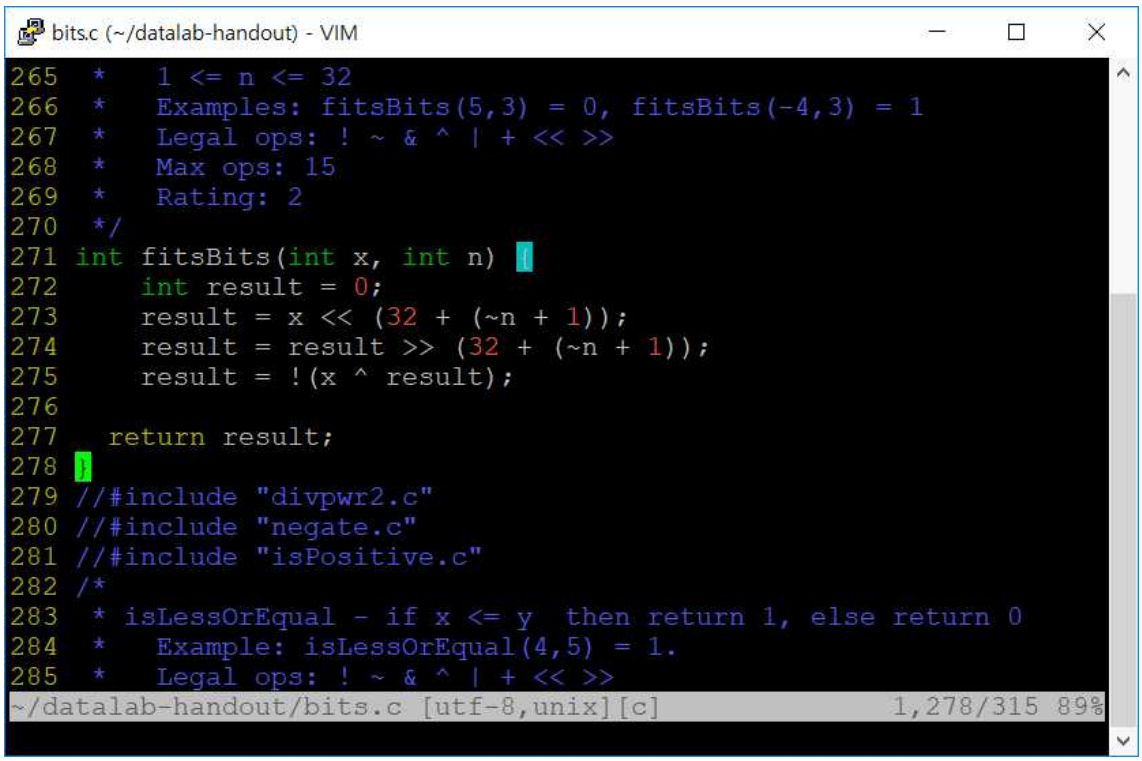
!는 0일때만 1을 반환한다.
이걸 이용하여, !x를 해주면 x가 0일때만 1을 반환하게 된다.

6	isEqual



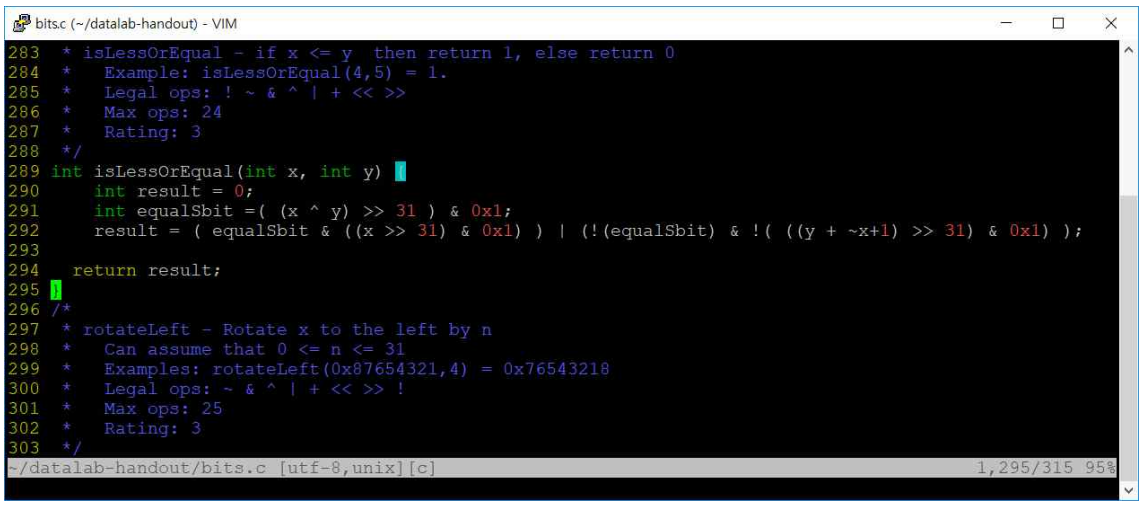
^는 같으면 0을 반환하고 다르면 1을 반환한다.
이걸로 x ^ y를 해주면 x와 y가 같으면 0을 반환을 할 것이다. 여기선 같으면 1을 반환해야하므로 !를 이용하여 0일 때 1이 반환되게끔 한다.

7	fitsBits



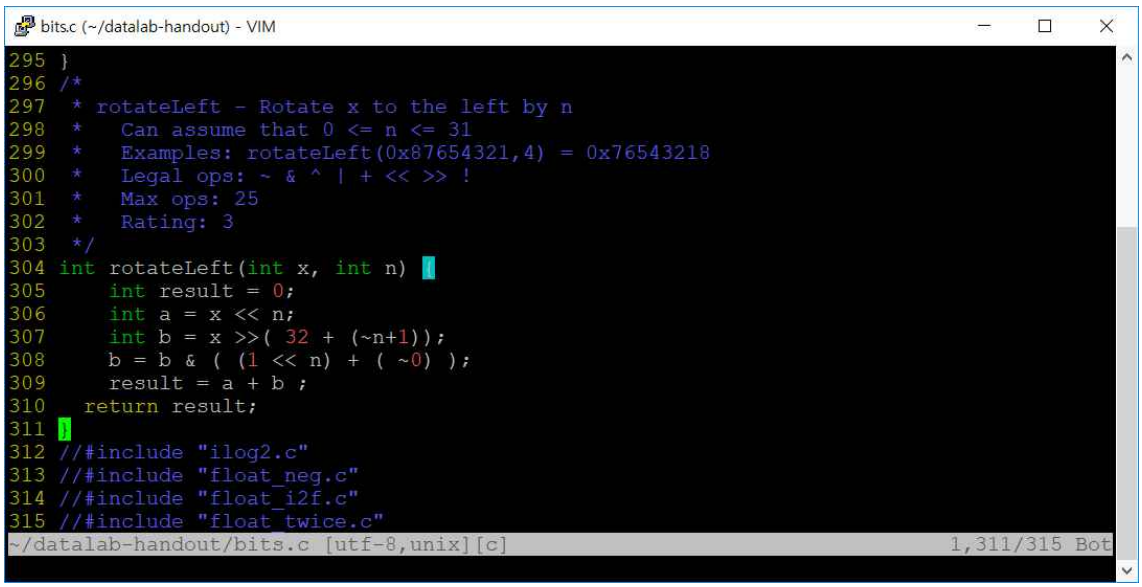
2의 보수 진법에서 x가 n비트로 표현가능한가를 묻는 것이다.
X가 만약에 n비트만으로도 잘 표현이 된다면 32 - n만큼 왼쪽으로 쉬프트하고 다시 오른쪽으로 쉬프트해서 원래대로 돌아와도 원래 x와 값이 달라지지 않을 것이다.
!(x ^ result)로 원래 x와 x를 왼쪽오른쪽으로 쉬프트한 result를 비교하여 같으면 1 , 다르면 0을 반환하게끔 한다.

8	isLessOrEqual



x와 y의 부호가 같을 땐 y-x가 양수면 1이고 음수면 0이다.
그런데 만약 x와 y의 부호가 다를 때는 y-x를 해주면 오버플로우가 발생할 수 있으므로 하지않고 x의 부호를 확인하여 x가 음수면 y가 양수이므로 1이고 x가 양수면 y가 음수이므로 0을 반환하게끔 한다.
여기서 equalSbit변수는 x와 y가 부호가 같은지 다른지를 나타내는데 부호가 같으면 ^를 했을 때 최상위비트 (31만큼 오른쪽이동 & 0x1) 가 같으니까 0을 반환한다.
(서로 부호가 다를때 & x가 음수) | (서로 부호가 같을 때 & y-x가 양수) 를 표현해주었다. 서로 부호가 달라서 equalSbit가 1이고, x의 최상위 비트가 1이라면(x는 음수) &연산으로 1을 반환하고 or뒤의 결과와 상관없이 결과값으로 1을 반환한다.
서로 부호가 같다면 equalSbit가 0 이지만 !를 해주어서 1이고 y-x의 최상위 비트가 0이면 (y-x가 양수) !로 1이된다 이 둘을 &해주면 1인데 이럴 경우엔 or앞의 결과와 상관없이 결과값에 1을 반환한다. 조건에 안맞으면 & 연산에서 다 걸러진다.

9	rotateLeft



일단 n비트만큼 왼쪽으로 이동해준다. 그러면 맨 오른쪽 비트에 n개의 0이 생긴다.
이 값을 a에 저장을 한다.
x의 맨 왼쪽부터 n개의 비트가 x의 맨 뒤 비트 뒤에 붙어야하기 때문에 32-n 만큼
오른쪽으로 이동해주고 b에 저장을 한다.
만약 b가 양수였다면 오른쪽으로 이동하고 빈자리가 0이기 때문에 그대로 a와 더해서
반환하면 그만이지만
b가 음수일 수도 있기 때문에 b의 이동해서 생긴 빈자리를 확실하게 0으로
채워줘야한다. logicalShift와 동일한 원리로 2의 n제곱에 1을 빼서 앞은 다 0이고
n번째bit부터 다 1인 이진수와 &연산을 해주어서 빈자리에 0을 채워서 b에 다시
저장한다.
이제 a와 b를 더한 수를 반환해주면 된다.