```c
#include <stdio.h>
int main() {
    /*Question 1*/
    int output = 0;
    unsigned long value = 58178824237 26;

    do {
        output = output ^ (value & 0x1);
        value = value >> 1;
    } while (value != 0x0);
    printf("ans %d\n", output);
    return 0;
}
```

```c
/*Question 2*/
int getParityFlag(unsigned int input) {
    int flag;
    /*we do right shifts of 16, 8, 4, 2, and 1 because we want to account for each
    individual xor so we want to shift this by powers of 2.
    take for instance the 4 bit of arbitrary "abcd".
    if we do an xor of this onto itself shifted by 2, we are doing:

    00ab (this can also be 11ab by arithmetic shift, but does not matter in this context)
    ^
    abcd

    here, we have the following:
    (0^a)(0^b)(a^c)(b^d) as our 4 bit value. Now, we do this again by a shift of 1:
    (0)(a)(b)(acbd)
    at the end, we have considered the ^ of b and d, the ^ of a and c, and the ^ of both
    these values. This gives us the consideration of all of the potential 1's since
    0 ^ 1 will net us 1 and 1 ^ 1 will net us 0. At the end, we want to apply a ! on
    the result since all even number of 1's should give us parity flag of 1
    and odd number of 1's should give us parity flag of 0.*/
    input ^= (input >> 16);
    input ^= (input >> 8);
    input ^= (input >> 4);
    input ^= (input >> 2);
    input ^= (input >> 1);
    input = input & 1;
    flag = !(input);
    return flag;
}
```