13 Network Address

13.1 Bit Masking

C provides several bit operations including the bitwise AND (`&`), the bitwise OR (`|`), and the bitwise XOR (`^`). The bitwise shift operations (<< and >>) are also supported by C. The following code shows how to count the number of 1 bits in the binary representations of integers:

```c
#include <stdio.h>

int bits(unsigned int n) {
    int cnt = 0;
    while (n > 0) {
        cnt += n & 1;
        n >>= 1;
    }
    return cnt;
}

int main()
{
    struct {
        unsigned int a: 4, b: 4;
    } n = {0xA, 0xB};

    printf("ones(%X) = %d\n", n.a, bits(n.a));
    printf("ones(%X) = %d\n", n.b, bits(n.b));

    return 0;
}
```

To represent two nibbles, `0xA` and `0xB`, a structure with two bitfields is used. Since the bitfield occupies just four bits, it is promoted to an unsigned integer when passed to the function `bits`.

13.2 Programming Lab 13: subnet.c

An IP address in the IPv4 scheme is represented by 32 bits, i.e. by four bytes. Some bytes in the IP address are designated for the network address, which can be obtained using the subnet mask. In short, the network address A_{net} can be calculated by masking bits of the IP address I using the subnet mask S: $A_{net} = I \& S$. Given the subnet mask 255.255.0.0 and the IP address 164.125.252.57, for example, the network address 164.125.0.0.

Write a program calculating the number of 1 bits in the network address given the IP address and the subnet mask. Define and use the type `IP` consisting of four bitfields where every field has eight bits. The IP address, the subnet mask, and the network address should be represented by the type `IP`. For the IP address and the subnet mask in the above example, your program should print 9.

Your program is to read from standard input. Input consists of two lines. The first line of the input contains the IP address. The second line of the input contains the subnet mask. Your program is to write to standard output. The output consists of a single line containing the number of 1 bits in the network address.

Additional requirements for bonus points

- Define and use the function `bits` accepting an `IP` pointer argument and returning the number of bits in the `IP` variable pointed to.
- Count the number of bits using a recursion rather than a loop.

Input	**Output**

`164.125.252.57`
`255.255.0.0`
	`9`

`125.252.57.164`
`255.0.0.0`
	`6`