Bonus:

<< is left shift.

表全部的位數都向左移一位, 且右側補0進來。

>> is right shift.

表全部的位數都向右移一位, 但最左邊怎麼補則根據編譯器而不同。

若是不管怎樣都補0, 我們稱為邏輯位移。

舉例:100010 >>1 後會得到010001, 010010>>1 後會得到001001。

但算術位移是符號位為何就補那個值進去。

舉例: 100010 >>1 後會得到110001, 但如果是010010 >> 1 後會得到001001。

釐清這些之後就可以解題了。

以下的結果可以知道這個編譯器是使用算術位移, 但這個不是我們要的。

```
1 #include <stdio.h
 2 #include <stdint.h>
 4 int main()
                                                                     un@yun-VirtualBox:~/Documents$ ./test
             int32_t number = 0;
scanf( "%d", & number );
int32_t bit = 1;
bit = bit << 31;
                                                                    un@yun-VirtualBox:~/Documents$ gcc test.c -o test
un@yun-VirtualBox:~/Documents$ ./test
             for( int i = 0 ; i < 32 ; i++ )
10
                                                                    00000000000000000000000000001111
                       if( bit & number )printf( "1" );
else printf( "0" );
11
12
13
14
                       bit = bit >> 1;
15
16
17
18
             printf("\n");
return 0;
```

所以如果我們要避免這個狀況,就乾脆使用uint32_t的型別,因為這樣就不會存有符號位。 像是下圖這樣子。

```
I #include <stdio.h>
2 #include <stdint.h>
                                                                yun@yun-VirtualBox: ~/Docume...
int main(){
                                                         un@yun-VirtualBox:~/Documents$ gcc test.c -o test
         int32_t number = 0;
                                                         un@yun-VirtualBox:~/Documents$ ./test
         scanf( "%d", & number );
uint32_t bit = 1;
                                                        00000000000000000000000000001100
         bit = bit << 31;
                                                         yun@yun-VirtualBox:~/Documents$
          for( int i = 0 ; i < 32 ; i++ )</pre>
          {
                  if( bit & number )printf( "1" );
else printf( "0" );
                  bit = bit >> 1;
         printf("\n");
         return 0;
3 }
```