位元運算
Bit Manipulations

位元運算

- C 語言提供了一些 operators，讓我們能用更低階的方式存取和修改資料。這些 operators 包括

&	AND
|	OR
^	XOR
>>	Right shift
<<	Left shift
~	One's compliment

位元運算

- 有時候我們需要儲存的資訊可能只有 1 和 0 兩種值，譬如記錄某種狀態 "有" 或 "無" (所謂的 flag)。當我們需要記錄大量這一類的資料時，若使用 **int** 來記錄會太浪費空間，這種情況就適合使用 bit 運算，只需用到原本的空間的 1/32
- 範例

練習 WD_04

- 用圖示來解釋下面的運作原理

```
unsigned getBits(unsigned x, int p, int n)
{
   return ( x >> (p-n) ) & ~( ~0 << n );
   /* 取出 x 的第 p 位置起 n 個 bits */
}
```

- 寫出 **unsigned invert(x, p, n)**把 **x** 第 **p** 位置起 **n** 個 bits 由 0 變 1，1 則變為 0
- 寫出 **unsigned rightRotate(x, n)**傳回 **x** 向右 rotate **n** bits 之後的結果

練習 WD_04 講解

- 以 **getBits(100, 6, 4)**為例，會從右邊數來第 6 個 bit 開始, 取出 4 個 bits

- 十進位 100 用二進位表示為：

  00000000000000000000000001100100

  紅色是我們要取出的 bits

- **dispBits(getBits(100, 6, 4));** 會得到

  00000000000000000000000000001001

練習 WD_04 講解

運作原理：

- **~0** 會得到 **111...11111111** 三十二個 1
- **~0 << n** 會得到 **111...11110000** 在這個例子 **n = 4**，所以最右邊四個 bits 因為向左 shift 的關係會補零
- **~(~0<<n)** 會得到 **000...00001111** 反轉的結果，這個結果會被當作 mask
- **x>>(p-n)** 把 **000...01100100** 向右 shift 兩個 bits (6-4)變成 **000...00011001** 左邊多出來的空位會補零
- **(x>>(p-n)) & ~(~0<<n)** 相當於 **000...00011001 & 000...00001111**

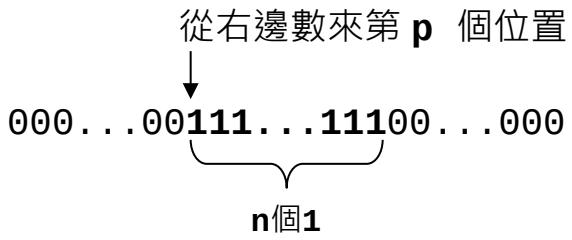
最後得到我們要的結果 **000...00001001**

invert()可能的寫法之一

```
unsigned invert( unsigned x, int p, int n )
{
   return  x ^ (~(~0 << n) << (p-n));
}
```

想辦法產生一個 mask, 能讓 p 位置後接 n 個 1, 其餘位置都是零:

1. 這樣的 mask 可以用 **~(~0 << n) << (p-n)** 產生
2. 有了這個 mask, 只要把它和 x 做 XOR 就可得到 **invert()**要做的效果
3. 因為任何 bit 和 0 做 XOR 的結果會維持不變，
4. 而任何 bit 和 1 做 XOR 則會變成 complement



rightRotate() 可能的寫法之一

```
unsigned rightRotate(unsigned x, int n)
{
   return ((x & ~(~0 << n)) << (sizeof(x)*8 - n)) |
(x >> n) ;
}
```

- 第一部份用 **(x & ~(~0 << n))** 取出最右邊 n 個 bits，然後向左 shift **(sizeof(x)*8 - n)** bits
- 第二部份用 **(x >> n)** 把 x 向右 shift **n** bits，把兩個部份做 OR 組合起來就可以做出向右 rotate 的效果