

學號:41247001S

輸出的結果和我原本想的不太一樣。不過後來查document之後發現原因了。

uint32\_t 的 -1 = 4294967295 (2倍的 $2^{32}-1$ )  
這是因為 int32\_t 儲存 -1 的方式在二進位制  
跟uint32\_t 的 4294967295 一樣所導致的

當 uint32\_t + int32\_t 時, 運算上面編譯會把int32\_t先轉換成 uint32\_t, 因此就會變成  $0 + 4294967295 = 4294967295$

而為甚麼當 uint16\_t + int32\_t 時就不會呢?  
這是因為 int32\_t 所使用的記憶體比uint16\_t大。  
(兩個變數的記憶體大小不同)  
所以這時編譯就會變成把 uint16\_t 轉成 int32\_t。  
變成  $0 + (-1) = -1$ 。  
而不會像前面的uint32\_t 一樣變成 4294967295

此外, 我也有找到會自動發生強制轉換的情形。例如兩者的資料型態不同、兩者所使用的記憶體大小不同等。

```
#include <stdint.h>
#include <stdio.h>
```

```
int main()
{
    uint32_t a = 0;
    uint16_t b = 0;
    int32_t e = 1;
    int64_t c = a - e;
    printf("%ld ", c);

    int64_t d = b - e;
    printf("%ld\n", d);
}
```

```
andy.lu4146@ub20:~/CFile/hw02$ ./test
4294967295 -1
andy.lu4146@ub20:~/CFile/hw02$
```

