hw0406

題目

```
int p(int i, int N)
{
  return (i <= N && printf("%d ", i) && printf(" %d\n", N) && !p(i + 1, N - 1));
}
```

說明 `printf` 為何可以加在 `return` 後

`printf`

`&&` 通常用來銜接判斷式，但這裡的 `printf` 表面上不太算是判斷式，但實際上printf也有回傳值，若""裡面有東西，為bool結果會是1，沒有(要印的)東西就會是0

```
bool b = 0;
b = (bool)(printf("-1"));
printf("b: %d", b);
```

```
bool b = 0;
b = (bool)(printf(""));
printf("b: %d", b);
```

執行結果：

⚠️ -1b: 1

⚠️ b: 0

所以 `printf` 在題目中都是 1

說明遞迴

回傳函式本身算是遞迴，所以 `i ≤ N` 會一直被判斷，如果 `i > N` 就不會繼續回傳值，也就是不會繼續執行 `printf` 的內容，所以最後才會只印出 i ~ N 的整數

```
return (i <= N && printf("%d ", i) && printf(" %d\n", N) && !p(i + 1, N - 1));
```

說明在 function 前加上 `!`

測式不加 `!`

使用變數a找出函式p的回傳值，題目中的回傳值是1，若把 `!p(i + 1, N - 1)` 改成 `p(i + 1, N - 1)`，就會是0，因為整個回傳值是布林值，所以在function前加上 `!` 會使整個布林值相反

```
int p(int i, int N)
{
  return (i <= N && printf("%d ", i) && printf(" %d\n", N) && p(i + 1, N - 1));
}
int main()
{
    int a = 0;
    a = p(11, 15);  //11 15 是測試用整數
    printf("a: %d\n", a);
    return 0;
}
```

執行結果：

加上 `!`

⚠️

```
11  15
12  14
13  13
a: 1
```

不加 `!`

⚠️

```
11  15
12  14
13  13
a: 0
```

結論: 加與不加只差在最終回傳的值，不影響函式p的功能性

最後 int p 可以改成 bool p 功能應該會是一樣的，因為回傳的事實上是布林值。