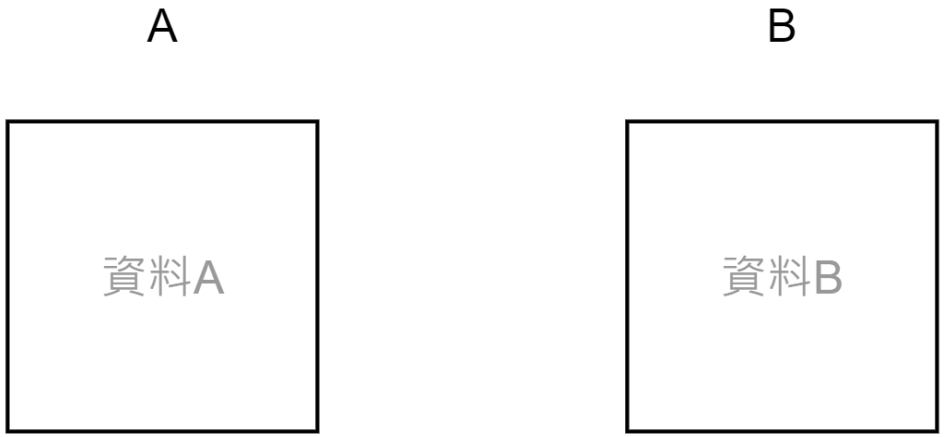
Value Type & Reference Type

C#的變數是如何存放在記憶體裡面?

為什麼很重要?

- 每個程式都是由不同的資料或是物件來完成各項功能
- 如果不了解這些資料是如何存放在記憶體中，就可能會…

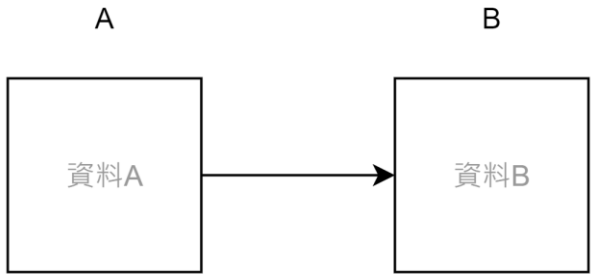
假設有A變數與B變數
各自存放自己的資料
彼此互不相干

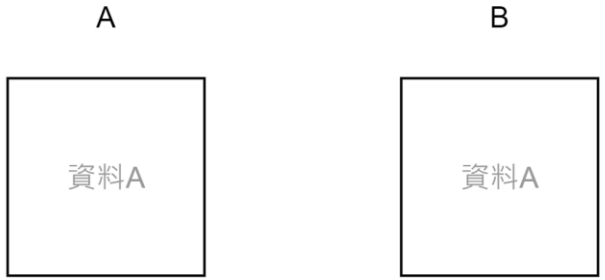


如果執行B=A;

將A變數的值賦予給B變數

此時A、B變數的值會變成一樣





- 正常來說A、B變數的內容是各自獨立的
- 任意修改其中一個變數都不會影響到另外一個的值

但是！

- 如果今天A、B是Reference Type 參考型別的變數
- 只要任意修改其中一個值，另外一個變數也會被影響

大家可能覺得很奇怪?
直接看範例！

1. 我們先宣告一個struct的結構Car
2. 裡面有兩個資料ID、Age

```
23  public struct Car
24  {
25      public int ID;
26      public int Age;
27
        1 個參考
28      public Car(int id, int age)
29      {
30          this.ID = id;
31          this.Age = age;
32      }
33  }
```

1. 接著在Start()裡面宣告carA利用建構子設定ID為111，Age為5
2. 再宣告carB變數，並把carA變數值賦予給carB
3. 然後修改carB的值
4. 接著輸出到Console視窗

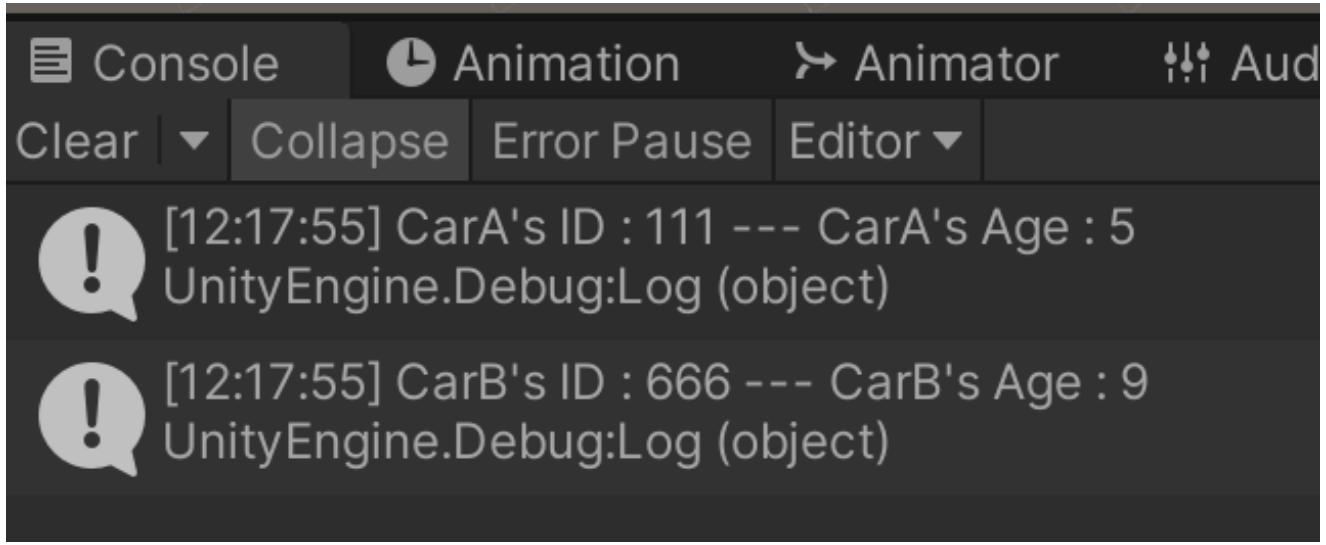
```
Unity 指令碼 (1 個資產參考)|0 個參考
public class Test : MonoBehaviour
{
    Unity Message|0 個參考
    private void Start()
    {
        Car carA = new Car(111, 5);

        Car carB = carA;

        carB.ID = 666;
        carB.Age = 9;

        Debug.Log($"CarA's ID : {carA.ID} --- CarA's Age : {carA.Age}");
        Debug.Log($"CarB's ID : {carB.ID} --- CarB's Age : {carB.Age}");
    }
}
```

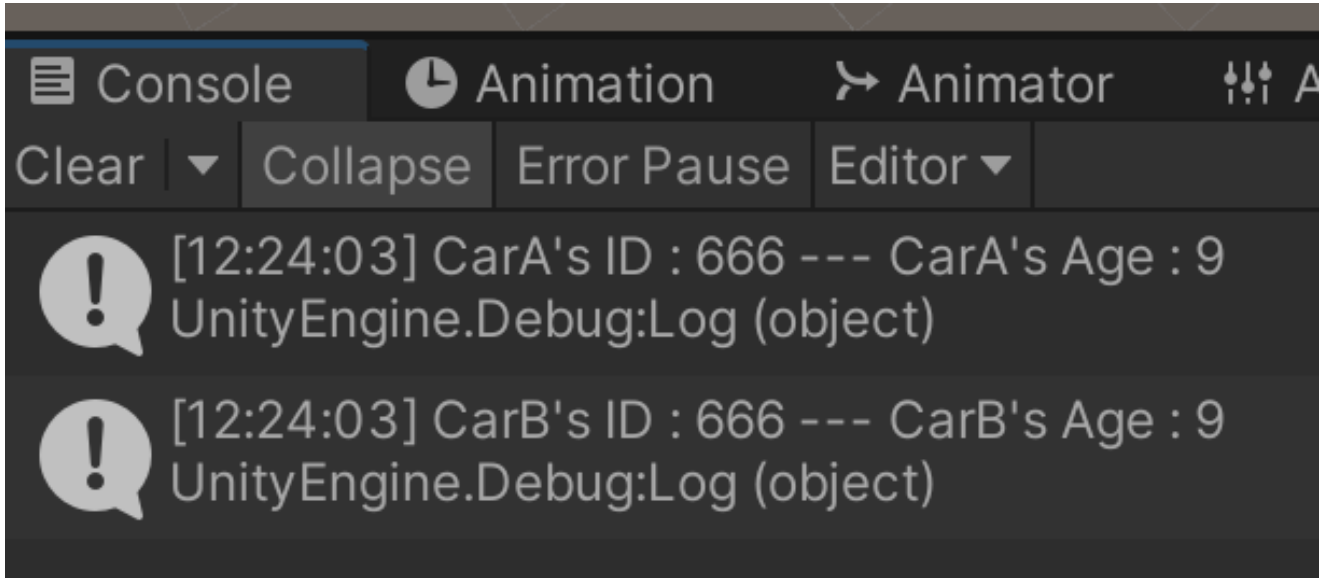
- 回到Unity後執行
- 結果非常符合預期，carA與carB變數的值各自獨立，修改後互不影響



但如果把Car從struct改成class再試試看呢？

```
23  public class Car
24  {
25      public int ID;
26      public int Age;
27
        1 個參考
28      public Car(int id, int age)
29      {
30          this.ID = id;
31          this.Age = age;
32      }
33  }
```

- 回到Unity後執行
- 結果竟然修改carB變數的值後，連carA的值也被影響了，變得與carB一樣



怎麼會這樣呢?

這是因為變數在C#中有幾種不同的型別
它們在記憶體中存放的狀態不同導致的

C#變數可以分為

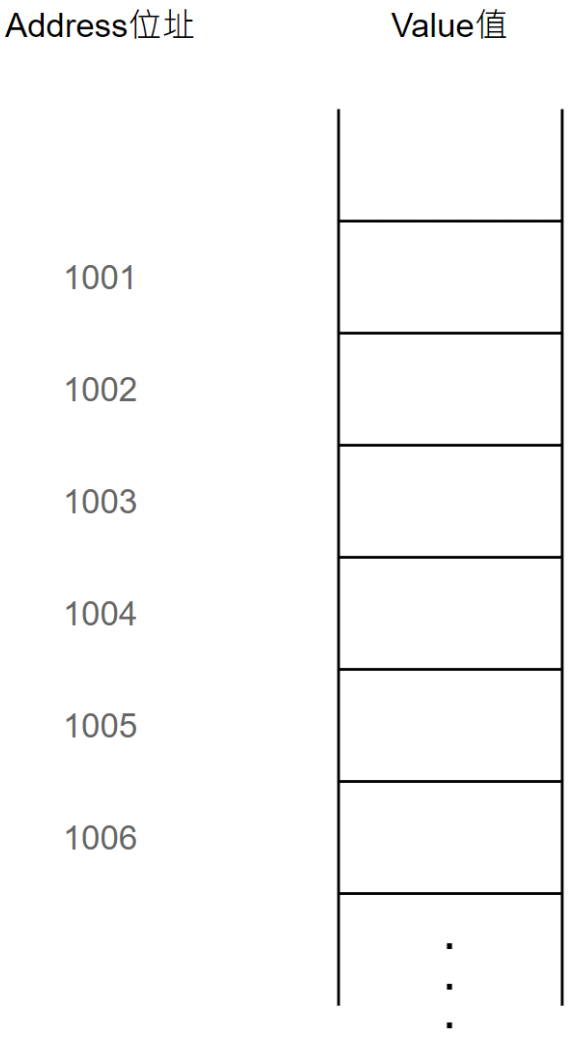
- Value Type 值型別
- Reference Type 參考型別
- Pointer Type 指標型別

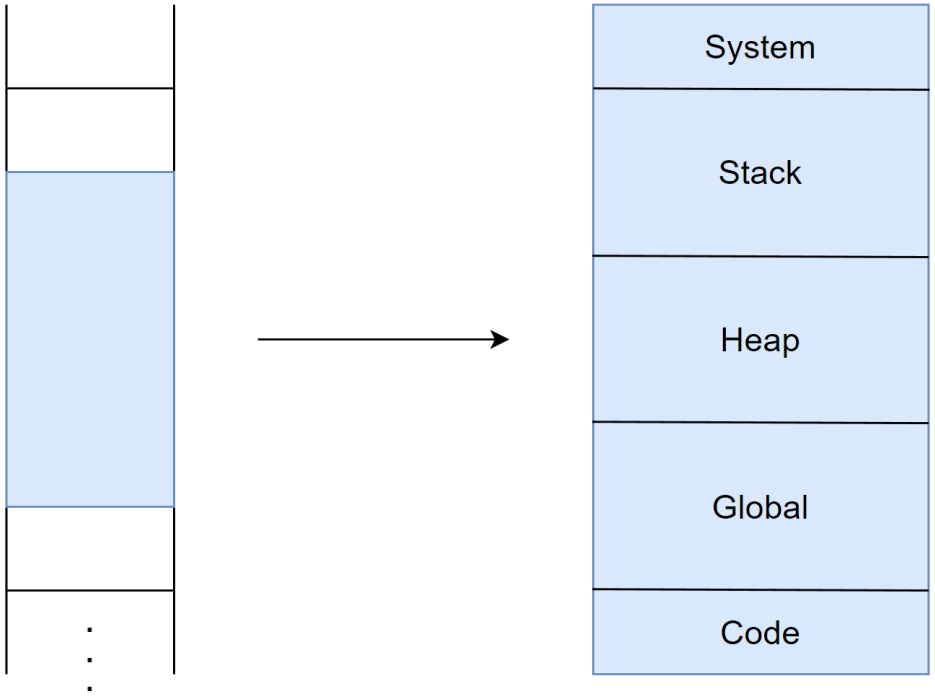
其中Pointer Type指標型別，大部分的時候都不會用到，因此我們不會去討論他

Value Type 值型別	Reference Type 參考型別
變數存放資料的實際值	變數存放資料的記憶體位址
int float bool char enum struct ...	string Array class interface delegate object ...

記憶體介紹

- 記憶體就是一條長長的空間
- 裡面有很多格子，存放我們的變數資料
- 每個格子都有自己的記憶體位址，來方便去找到格子裡的內容



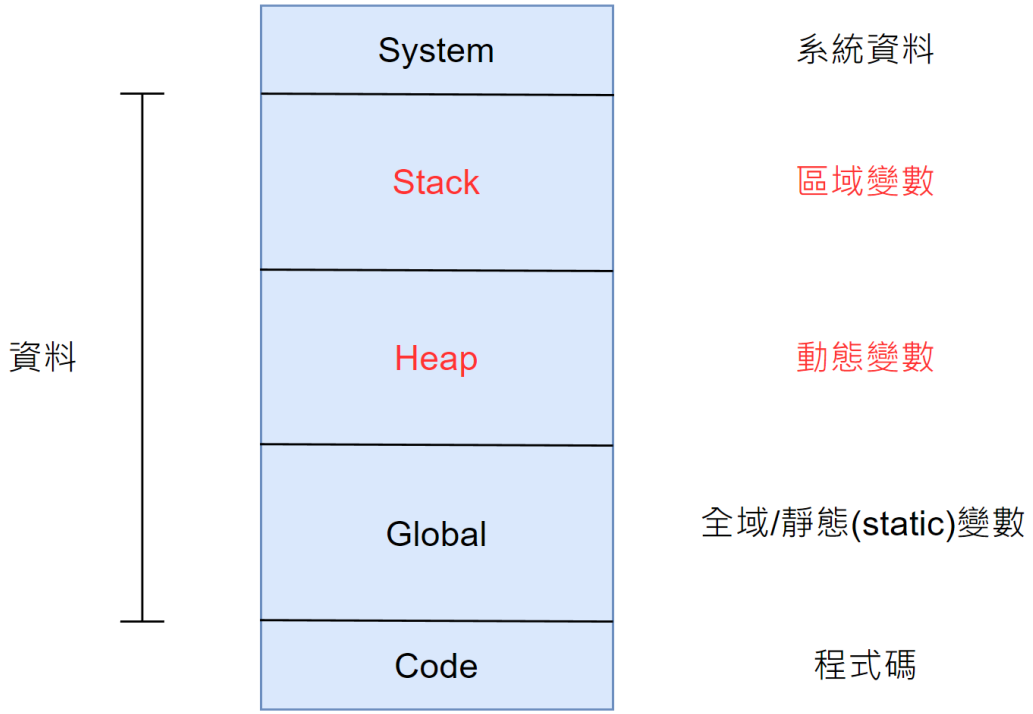


每當我們開啟一個新的應用程式或是遊戲的時候

系統就會分配一塊記憶體空間，用來存放我們剛開啟的程式

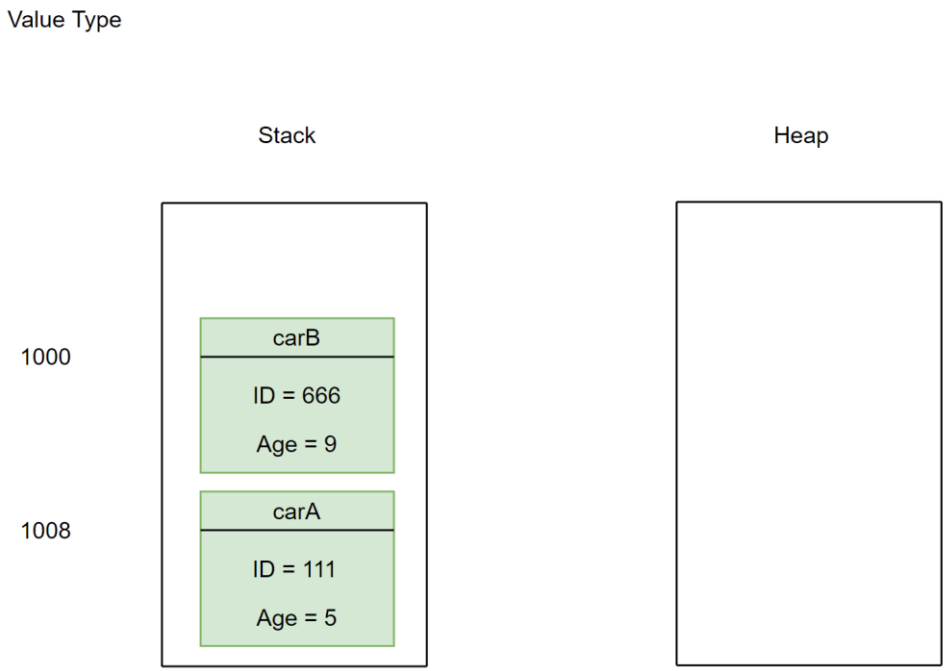
這個區塊的記憶體又會被劃分成幾個部分

- System：系統資料
- Stack(堆疊)：區域變數
- Heap(堆積)：動態變數
- Global：全域/靜態(static)變數
- Code：程式碼

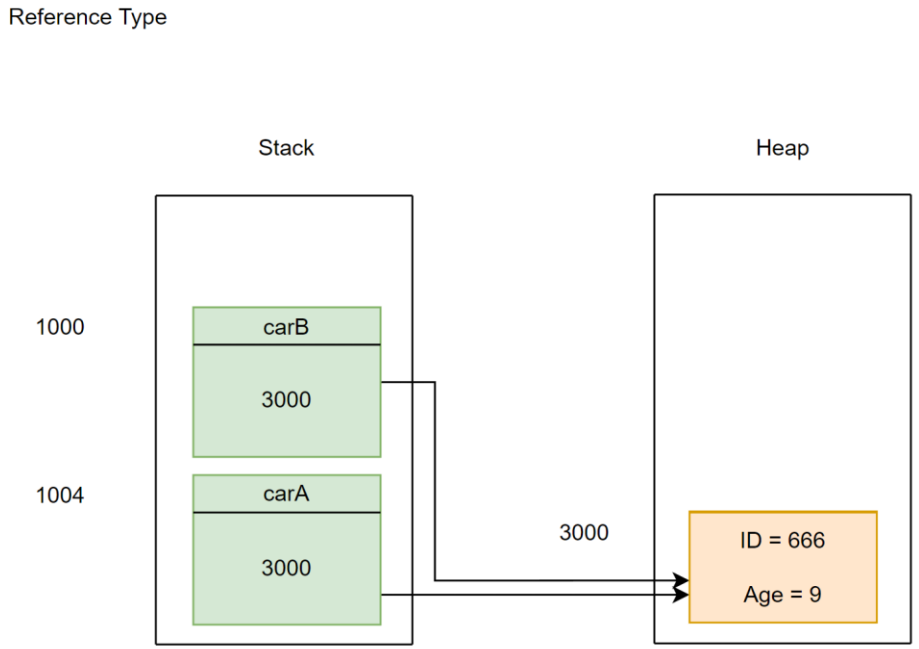


- Stack堆疊 存放所有區域變數的資料，不論是Value或是Reference Type
- Heap堆積 存放所有動態變數配置的資料，通常透過new()方法配置

- Struct因為是Value Type，因此他的變數實際的值都會被存放在Stack中
- 如果這個時候宣告carB，然後把carA的值賦予給他，會直接在Stack中開一個新的空間，並把carA的值複製一份給carB
- 因此兩個變數的值各自獨立，修改後也互不影響



- 而class因為是Reference Type，因此他的變數實際的值被new()出來後都會存放在Heap中
- Stack則是存放該實際物件在Heap中的記憶體位址，稱為carA變數指向記憶體位址3000



- 如果這個時候宣告carB，然後把carA的值賦予給他，會在Stack中開一個新的空間，並把carA指向的記憶體位址複製一份給carB
- 因此兩變數都指向同一個物件，所以修改後會互相影響