

## C++ Gossip: 複製建構函式、物件的指定

當您宣告一個物件時，您可以使用另一個物件將之初始化，例如：

```
SomeClass s1;
SomeClass s2 = s1;
```

這麼作的話，s1的屬性會一一的「複製」至s2的每一個屬性，下面這個程式是個簡單的示範，您進行物件的指定，而最後用&運算子取出物件的記憶體位址，您可以看到兩個物件所佔的位址並不相同：

```
#include <iostream>
using namespace std;

class Employee {
public:
    Employee() {
        _num = 0;
        _years = 0.0;
    }

    Employee(int num, double years) {
        _num = num;
        _years = years;
    }

    int num() {
        return _num;
    }

    double years() {
        return _years;
    }

private:
    int _num;
    double _years;
};

int main() {
    Employee p1(101, 3.5);
    Employee p2 = p1;

    cout << "p1 addr:\t" << &p1 << endl;
    cout << "p1.num: \t" << p1.num() << endl;
    cout << "p1.years:\t" << p1.years() << endl;

    cout << "p2 addr:\t" << &p2 << endl;
    cout << "p2.num: \t" << p2.num() << endl;
    cout << "p2.years:\t" << p2.years() << endl;

    return 0;
}
```

執行結果：

```
p1 addr: 0x22ff60
p1.num: 101
p1.years: 3.5
p2 addr: 0x22ff50
p2.num: 101
p2.years: 3.5
```

然而這中間潛藏著一個危機，尤其是在屬性成員包括指標時，以建構函式、解構函式 中的SafeArray類別來說，看看下面的程式問題會出在哪邊：


```
SafeArray arr1(10);
SafeArray arr2 = arr1;
```

表面上看起來沒有問題，但記得\_array是int型態指標，而在解構函式是這麼寫的：

```
SafeArray::~SafeArray() {
    delete [] _array;
}
```


arr2複製了arr1的屬性，當然也包括了\_array指標，如果arr1資源先被回收了，但arr2的\_array仍然參考至一個已被回收資源的位址，這時再存取該位址的資料就有危險，例如下面這段程式就可能造成程式不可預期的結果：

```
SafeArray *arr1 = new SafeArray(10);
SafeArray arr2 = *arr1;
delete arr1;
```



**Gjun 巨匠電腦**  
**新學期 就業專班**

☐ 10月最新課表
 ☐ 11月最新課表
 ☐ 12月最新課表
 ☐ 1月最新課表



**免費索取簡章 送好禮5選1>>**



**六都市長選舉 2014**

**立即參與 # 市民發聲 !**

城市選擇 ▼
市政類別 ▼
議題選擇 ▼

在此填寫你的意見，限制60字內，(可略過)

☐ 同意市民相關服務、隱私權及內容條款

遞交

Google
replay()

爲了避免這樣的錯誤，您可以定義一個複製建構函式，當初始化時如果有提供複製建構函式，則會使用您所定義的複製建構函式，您可以在定義複製建構函式時，當遇到指標成員時，產生一個新的資源並指定位址給該成員，例如：

- SafeArray.h

```
class SafeArray {
public:
    int length;

    // 複製建構函式
    SafeArray(const SafeArray&);
    // 建構函式
    SafeArray(int);
    // 解構函式
    ~SafeArray();

    int get(int);
    void set(int, int);

private:
    int *_array;

    bool isSafe(int i);
};
```

- SafeArray.cpp

```
#include "SafeArray.h"

// 複製建構函式
SafeArray::SafeArray(const SafeArray &safeArray)
    : length(safeArray.length) {
    _array = new int[safeArray.length];

    for(int i = 0; i < safeArray.length; i++) {
        _array[i] = safeArray._array[i];
    }
}

// 動態配置陣列
SafeArray::SafeArray(int len) {
    length = len;
    _array = new int[length];
}

// 測試是否超出陣列長度
bool SafeArray::isSafe(int i) {
    if(i > length || i < 0) {
        return false;
    }
    else {
        return true;
    }
}

// 取得陣列元素值
int SafeArray::get(int i) {
    if(isSafe(i)) {
        return _array[i];
    }

    return 0;
}

// 設定陣列元素值
void SafeArray::set(int i, int value) {
    if(isSafe(i)) {
        _array[i] = value;
    }
}

// 刪除動態配置的資源
SafeArray::~SafeArray() {
    delete [] _array;
}
```

如果屬性成員中有指標型態，除了爲物件始化撰寫複製建構函式之外，最好再重載=指定運算子，因爲=指定運算子預設也是將物件的屬性值一一複製過去，您應該重載=指定運算子，在遇到指標成員時，產生位址上的資源複本，例如：

- SafeArray.h

```
class SafeArray {
public:
    int length;
```



凱擘 大寬頻

**120M**  
只要 **\$879** /月

還享四大貼心優惠  
學生專屬

- ▶ 暫停上網免付費
- ▶ 免過戶費
- ▶ Wi-Fi免費裝
- ▶ 免移機費

**立即  
申裝**

```

// 複製建構函式
SafeArray(const SafeArray&);
// 建構函式
SafeArray(int);
// 解構函式
~SafeArray();

int get(int);
void set(int, int);

// 重載=運算子
SafeArray& operator=(const SafeArray&);

private:
    int *_array;

    bool isSafe(int i);
};

```

#### • SafeArray.cpp

```

#include "SafeArray.h"

// 複製建構函式
SafeArray::SafeArray(const SafeArray &safeArray)
    : length(safeArray.length) {
    _array = new int[safeArray.length];

    for(int i = 0; i < safeArray.length; i++) {
        _array[i] = safeArray._array[i];
    }
}

// 重載=指定運算子
SafeArray& SafeArray::operator=(const SafeArray &safeArray) {
    if(this != &safeArray) {
        length = safeArray.length;

        // 先清除原有的資源
        delete [] _array;

        _array = new int[safeArray.length];
        for(int i = 0; i < safeArray.length; i++) {
            _array[i] = safeArray._array[i];
        }

        return *this;
    }
}

// 動態配置陣列
SafeArray::SafeArray(int len) {
    length = len;
    _array = new int[length];
}

// 測試是否超出陣列長度
bool SafeArray::isSafe(int i) {
    if(i > length || i < 0) {
        return false;
    }
    else {
        return true;
    }
}

// 取得陣列元素值
int SafeArray::get(int i) {
    if(isSafe(i)) {
        return _array[i];
    }

    return 0;
}

// 設定陣列元素值
void SafeArray::set(int i, int value) {
    if(isSafe(i)) {
        _array[i] = value;
    }
}

```

```
}  
  
// 刪除動態配置的資源  
SafeArray::~SafeArray() {  
    delete [] _array;  
}
```