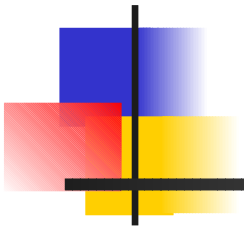
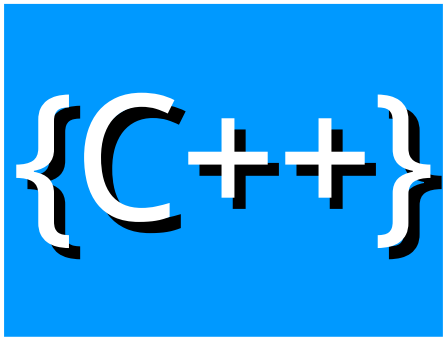
Classes: A Deeper Look



Week 3



Yang-Cheng Chang
Yuan-Ze University
yczhang@saturn.yzu.edu.tw

成員的初始化順序

- 成員的初始化順序往往並不重要
- 但是如果某個成員相依於另一個成員，那麼初始化順序就變得很重要

```
class x {
        int i;
        int j;
public:
        // 執行時期錯誤，因爲 i 在 j 之前被初始化
        x(int val): j(val), i(j) {}
```

- 按成員變數的宣告順序來編寫建構式初值器
- 儘可能避免以某個成員初始化另一個成員

```
        x(int val): i(val), j(val) {}
```

delete

- "delete p" 會刪去 "p" 指標，還是它指到的資料，"*p" ？
  - 該指標指到的資料。"delete" 眞正的意思是：「刪去指標所指到的東西」
- 該怎樣配置／釋放陣列？

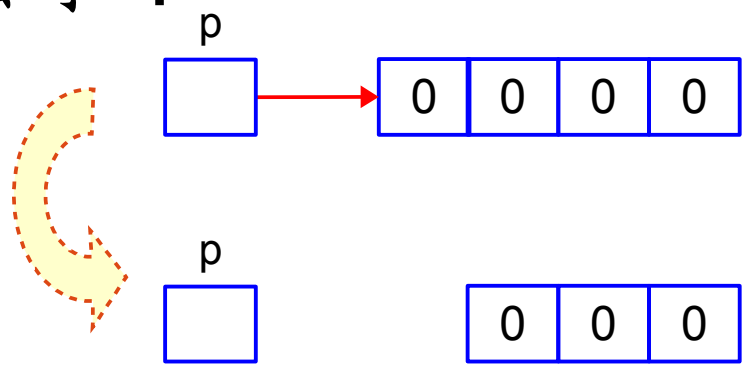
```
int* p = new int[4];
 //...
delete [] p;
```

  - 這語法是必要的，因爲「指向單一元素的指標」與「指向一個陣列的指標」在語法上並無法區分開來。

delete

- 萬一我忘了將 "[]" 用在 "delete" 由 "new int[n]" 配置到的陣列，會發生什麼事？

```
int* p = new int[4];
 ...
delete p;       // 錯誤
```



  - 災難。若你弄錯了，編譯器不會產生任何編譯期或執行期的錯誤訊息。堆積（heap）被破壞是最可能的結局，或是更糟的，你的程式會當掉
  - 確保 new[] 與 delete[] 的正確配對是程式者的責任

delete

- delete 指標後，需要將值設定爲 NULL 嗎？
  - delete 一個指標並不會改變指標的值（不會在 delete 後將指標設爲 NULL），所以使用了 delete 之後最好自己把指標設爲 NULL

```
delete p;
p = nullptr;
```

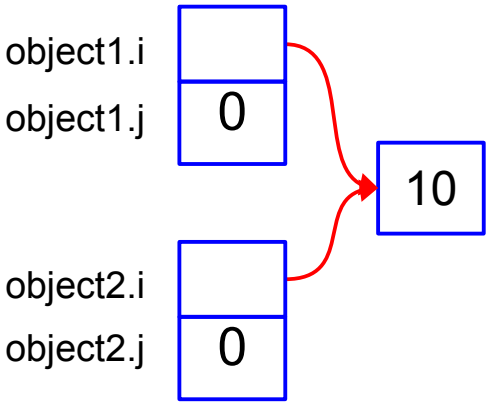
- delete 指標之前是否要檢查指標爲 NULL?
  - 不必，delete 會自動檢查

```
if(p==nullptr){          // 不需要檢查
    delete p;
}
```

靜態成員

- 使用時機
  - 物件之間需要共用變數時

```
class myclass {
  static int i;
  int j
public:
  void setInt(int n) {
     i = n;
  }
  int getInt() {
     return i;
  }
};
int myclass::i;       // Definition of myclass::i. i is still private to myclass.
int main()
{
  myclass object1, object2;
  object1.setInt(10);
  cout << "object1.i: " << object1.getInt() << '\n'; // displays 10
  cout << "object2.i: " << object2.getInt() << '\n'; // also displays 10
  return 0;
}
```



Assignment 3

- 撰寫一個程式能夠處理撲克牌的洗牌與發牌
  - 由 class Card 與 class DeckOfCards 構成
  - class Card 要求
    - 兩個成員變數 face 與 suit ，型別爲整數，用來表示這張牌的花色 (suit) 與人頭牌或數字排 (face)
    - 建構子的參數傳遞兩個整數，分別用來初始化 face,suit
    - 兩個靜態型別的 string 陣列 faces 、 suits ，用來儲存花色與數字的名稱

```
const string Card::suits[4]={" 紅心 ", " 方塊 ", " 黑桃 ", " 梅花 "};
const string Card::faces[13]={"A", "2", "3", "4", "5", "6", "7", "8", "9", "10","J", "Q", "K"};
```

    - 成員函式 toString 傳回 string ，型式爲 " 花色 數字 " ，範例 " 方塊 2"

```
int face=1;
int suit=1;
```

```
string CardName=suits[suit]+” “+faces[face]
```

Assignment 3

- class DeckOfCards 要求
  - 用 vector 儲存 Card 物件，變數名稱爲 deck
  - currentCard 代表目前發出的 Card 在 deck 中的位置
  - 建構子需要初始化每張牌，初始化的 Card 使用 vector 的 push_back 加入到 deck
  - 成員函式 suffle 用來洗牌，洗牌演算法爲從頭到尾走遍 deck 中的每張牌，隨機挑選另一張牌與 currentCard 所代表的牌交換
  - 成員函式 dealCard 從 deck 傳回目前發出的 currentCard 所代表 Card 物件
  - 成員函式 moreCards 傳回 bool 值，用來決定是否還有更多的牌
- 主程式需要生成 DeckOfCards 的物件，並且洗牌後，將 52 張牌依序發出，並且顯示出來

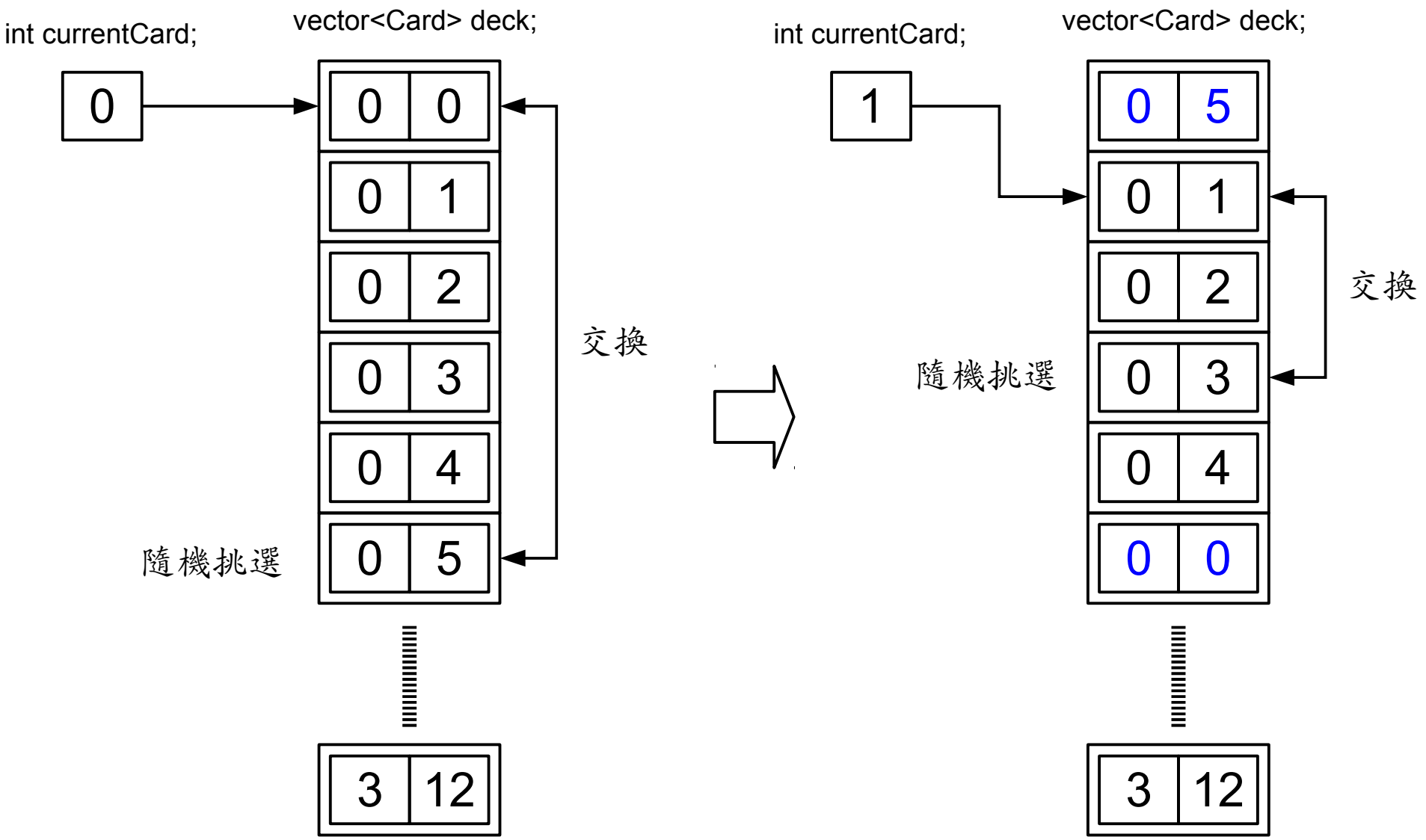
Assignment 3

```
const int NumSuits = 4;
const int NumFaces = 13;

class Card
{
    public:
        Card(int f,int s): face(f),suit(s){}
        string toString();
    private:
        int face,suit;
        static const string suits[NumSuits];
        static const string faces[NumFaces];
};

class DeckOfCards
{
    private:
        vector<Card> deck;
        int currentCard;

    public:
        DeckOfCards();
        void shuffle();
        Card& dealCard();
        bool moreCards();
};
```

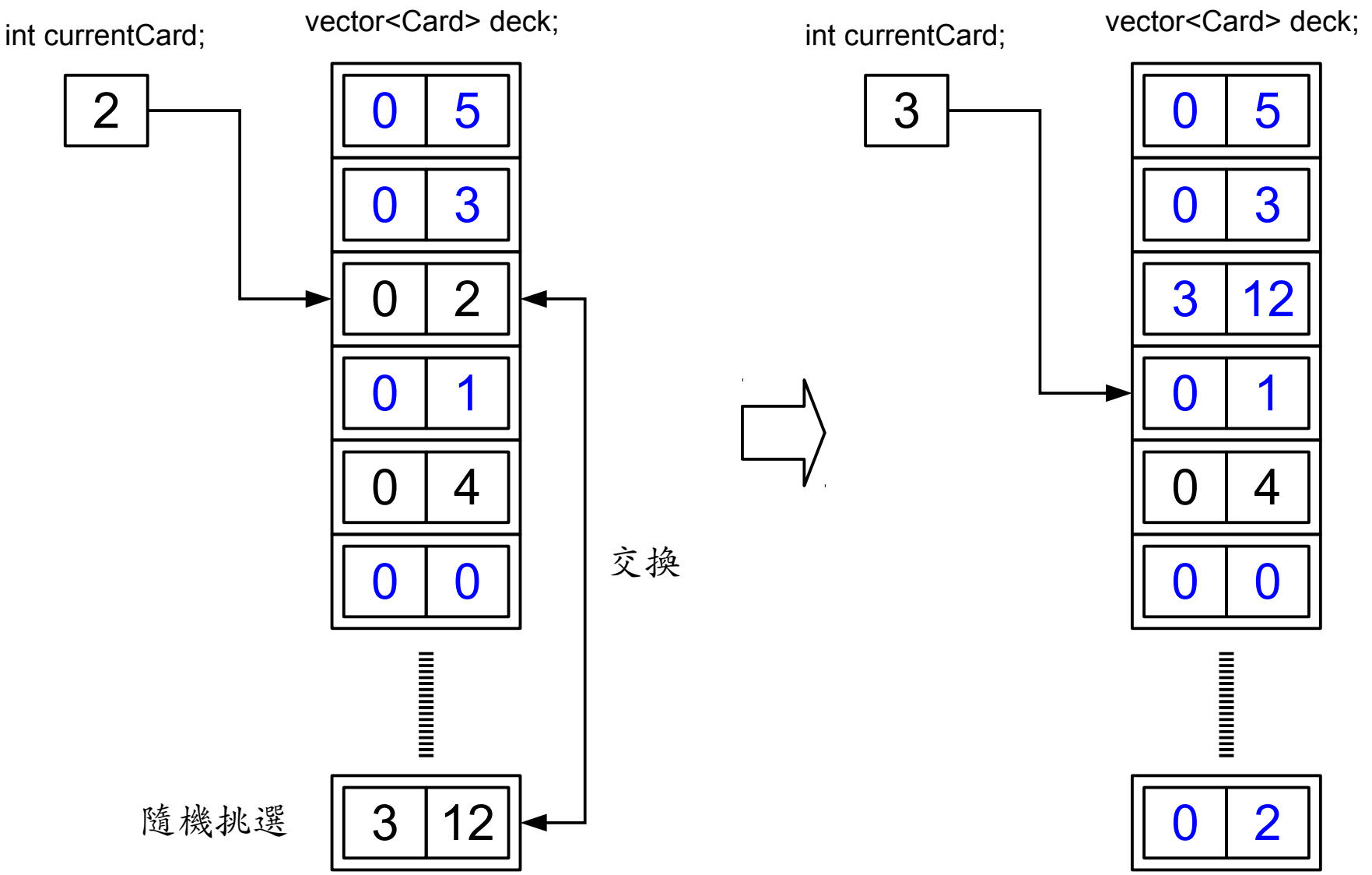
```
const string Card::suits[NumSuits]={" 紅心 ",
" 方塊 ", " 黑桃 ", " 梅花 "};
const string Card::faces[NumFaces]={"A", "2",
"3", "4", "5", "6", "7", "8", "9", "10","J", "Q",
"K"};
```

```
#include <DeckCard.h>
using namespace std;
int main ()
{
    DeckOfCards deck;
    deck.shuffle();
    while(deck.moreCards()){
        Card& c = deck.dealCard();
        cout << c.toString() << endl;
    }
}
```

洗牌演算法


int currentCard;
vector<Card> deck;
0
0 0
0 1
0 2
0 3
0 4
0 5
3 12
交換
隨機挑選
int currentCard;
vector<Card> deck;
1
0 5
0 1
0 2
0 3
0 4
0 0
3 12
交換
隨機挑選

洗牌演算法


int currentCard;
vector<Card> deck;
2
0 5
0 3
0 2
0 1
0 4
0 0
隨機挑選
3 12
交換
int currentCard;
vector<Card> deck;
3
0 5
0 3
3 12
0 1
0 4
0 0
0 2