Homework6-2與Homework5-7的比較

班級：電子一乙
學號：113360236
姓名：吳劉軒

(homework6-2流程圖)

主程式

(Homework6-2)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct card {
    const char* face;
    const char* suit;
};
typedef struct card Card;

void fillDeck(Card *const wDeck, const char* wFace[], const char* wSuit[]);
void shuffle(Card* const wDeck);
void deal(const Card* const wDeck);

int main() {
    Card deck[52];
    const char* face[] = { "Ace","Deuce","Three","Four","Five","Six","Seven","Eight","Nine","Ten","Jack","Queen","King" };
    const char* suit[] = { "Hearts","Diamonds","Clubs","Spades" };
    srand(time(NULL));
    fillDeck(deck, face, suit);
    shuffle(deck);
    deal(deck);
    system("pause");
    return 0;
}
```

1. 以結構的方式定義一張牌的屬性(數字、花色)
2. 需要fillDeck函式來初始化卡牌
3. 使用shuffle函式洗牌
4. 使用deal函式顯示洗牌後的結果

(Homework5-7)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void shuffle(int wDeck[][13]);
void deal(const int wDeck[][13], const char* wFace[], const char* wSuit[]);

int main(void){
    const char* suit[4] = { "Hearts", "Diamonds", "Clubs", "Spades" };
    const char* face[13] = { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
    int deck[4][13] = { 0 };
    srand(time(0));
    shuffle(deck);
    deal(deck, face, suit);
    system("pause");
    return 0;
}
```

1. 以二維陣列將所有的牌列出來[4種花色][13個數字]
2. 直接將陣列的數值都設置為0來初始化
   `int deck[4][13] = { 0 };`
3. 使用shuffle函式洗牌
4. 使用deal函式顯示洗牌後的結果

主程式與初始化卡牌比較

一共有兩個差別:

1. Homework6-2以結構的方式定義一張牌的屬性

優點: 程式碼比較直觀，方便讀。
較容易發展出其他應用
較容易維護

缺點: 撰寫過程較麻煩。
需要儲存字串指標，所以記憶體需求較高。

Homework5-7以二維陣列將所有的牌列出來

優點: 轉寫程式的過程比較輕鬆。
因為僅需儲存整數值，所以記憶體需求較低。

缺點: 程式較不易讀懂。
發展其他應用較麻煩(可能想新增一個功能，要改變中間的程式，但是一個改了其他也要跟著改，牽一髮而動全身)。

2. Homework6-2將卡牌初始化的方法是將所有卡牌依照順序先擺好，在進行洗牌。

優點: 非常直觀，一張牌的屬性是甚麼都很清楚。

缺點: 初始化的程式較Homework5-7長

Homework5-7是直接將整個卡牌陣列設成0

優點: 寫得很輕鬆。

缺點: 程式發生邏輯錯誤的時候結果錯的較明顯(可能整疊牌都還是0，使用觀感不佳)。

```
27  void fillDeck(Card* const wDeck, const char* wFace[], const char* wSuit[]) {
28      int i;
29      for (i = 0; i <= 51; i++) {
30          wDeck[i].face = wFace[i % 13];
31          wDeck[i].suit = wSuit[i / 13];
32      }
33  }
34
```

(Homework6-2初始化副程式)

洗牌副程式

```
35  void shuffle(Card* const wDeck){
36      int i, j;
37      Card temp;
38      for (i = 0; i <= 51; i++) {
39          j = rand() % 52;
40          temp = wDeck[i];
41          wDeck[i] = wDeck[j];
42          wDeck[j] = temp;
43      }
44  }
```

(Homework6-2)

以隨機”交換卡牌位置”的方式洗牌

```
19  void shuffle(int wDeck[][13]) {
20      int row;
21      int column;
22      int card;
23      for (card = 1; card <= 52; card++){
24          do{
25              row = rand() % 4;
26              column = rand() % 13;
27          } while (wDeck[row][column] != 0);
28          wDeck[row][column] = card;
29      }
30  }
```

(Homework5-7)

以隨機”將卡牌填入陣列”的方式洗牌

洗牌副程式比較

Homework6-2以隨機”交換卡牌位置”的方式洗牌

優點：與真實世界的洗牌方式雷同，十分直觀。
程式碼較短，迴圈可預測，較無錯誤可能。

缺點：需要多宣告一個結構card來當暫存，所以記憶體需求較高。

Homework5-7以隨機”將卡牌填入陣列”的方式洗牌

優點：僅需宣告三個int，所以記憶體需求較低。

缺點：程式較難想到
程式不易讀懂
wlile迴圈較難預測，可能會有意想不到的錯誤。

顯示結果副程式

(Homework6-2)

```
46  void deal(const Card* const wDeck){
47      int i;
48      for (i = 0; i <= 51; i++) {
49          printf("%5s of %-8s%s",wDeck[i].face,wDeck[i].suit,(i+1)%4 ?" ":"\n");
50      }
51  }
```

可以直接使用一維陣列和迴圈，將被隨機換位置的牌的屬性顯示出來。

(Homework5-7)

```
32  void deal(const int wDeck[][13], const char* wFace[], const char* wSuit[]) {
33      int card;
34      int row;
35      int column;
36
37      for (card = 1; card <= 52; card++) {
38          for (row = 0; row <= 3; row++) {
39              for (column = 0; column <= 12; column++) {
40                  if (wDeck[row][column] == card) {
41                      printf("%5s of %-8s%c", wFace[column], wSuit[row], card % 2 == 0 ? '\n' : '\t');
42                  }
43              }
44          }
45      }
46  }
```

需要使用三層for迴圈加一層if來顯示已經洗好的卡牌。

顯示結果副程式比較

Homework6-2以一維陣列和迴圈來顯示結果

這應該是使用結構的優點，可以將一個物件的所有屬性寫在一起，定義成一個東西，所以可以使用一維陣列再使用一層迴圈來顯示結果。

Homework5-7以使用三層for迴圈加一層if來顯示已經洗好的卡牌

這應該是前面偷懶的報應，雖然使用二維陣列宣告牌組較方便；洗牌的程式簡單好寫，但是需要顯示時就需要考慮二維陣列的行跟列，所以就需要兩個變數以上，程式反而會變複雜且不易讀。

結束