

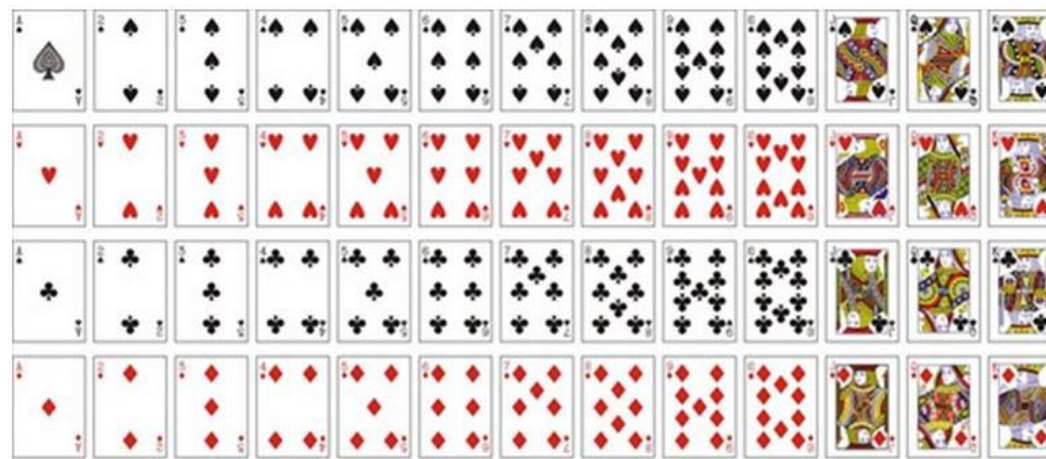
第10章 结构体和共用体

——典型实例：洗发牌模拟

如何表示52张扑克牌？



- 每张牌分为4种花色 (Suit)
 - * 黑桃 (Spades)、红桃 (Hearts)、草花 (Clubs)、方块 (Diamonds)
- 每种花色有13张牌面 (Face)
 - * A, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King
- 第一种方法
 - * 用二维数组
 - * `deck[4][13];`



	Ace	Two	Three	Four	Five	Six	Seven	Eight	Nine	Ten	Jack	Queen	King
	0	1	2	3	4	5	6	7	8	9	10	11	12
Spades	0												
Hearts	1												
Clubs	2												
Diamonds	3												

`deck[2][12]` represents the King of Clubs

Clubs King

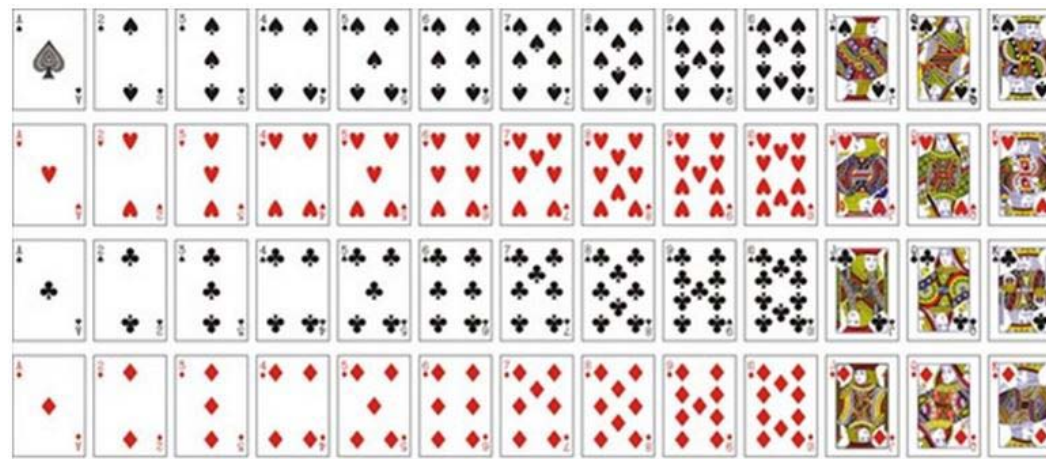
如何表示52张扑克牌？



■ 第二种方法

* 用结构体数组

```
typedef struct
{
    char suit[10];    /*花色*/
    char face[10];    /*牌面*/
}CARD;
CARD card[52];
```



问题：如何表示第1张牌的花色和牌面？

```
card[0].suit
card[0].face
```

如何保存一副扑克牌？



```
typedef struct
{
    char suit[10];    /*花色*/
    char face[10];    /*牌面*/
}CARD;
CARD card[52];
```



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33...
i/13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2...
i%13	0	1	2	3	4	5	6	7	8	9	10	11	12	0	1	2	3	4	5	6	7	8	9	10	11	12	0	1	2	3	4	5	6	7...

```
int i;
for (i=0; i<52; i++)
{
    strcpy(card[i].suit, pSuit[i/13]);
    strcpy(card[i].face, pFace[i%13]);
}
```

```
char *pSuit[] = {"Spades", "Hearts",
                 "Clubs", "Diamonds"};

char *pFace[] = {"A", "2", "3", "4", "5",
                 "6", "7", "8", "9", "10",
                 "Jack", "Queen", "King"};
```

如何保存一副扑克牌？



```
void FillCard(CARD card[])
{
    char *pSuit[]={"Spades", "Hearts", "Clubs", "Diamonds"};
    char *pFace[]={"A","2","3","4","5","6","7","8","9","10","Jack","Queen","King"};
    int i;
    for (i=0; i<52; i++)
    {
        strcpy(card[i].suit, pSuit[i/13]);
        strcpy(card[i].face, pFace[i%13]);
    }
}
```

```
int i;
for (i=0; i<52; i++)
{
    strcpy(card[i].suit, pSuit[i/13]);
    strcpy(card[i].face, pFace[i%13]);
}
```

```
char *pSuit[] = {"Spades","Hearts",
                  "Clubs","Diamonds"};
char *pFace[] = {"A","2","3","4","5",
                  "6","7","8","9","10",
                  "Jack","Queen","King"};
```

如何发牌?



```
typedef struct
{
    char suit[10];    /*花色*/
    char face[10];    /*牌面*/
}CARD;
CARD card[52];
```



```
//发牌
void Deal(CARD card[])
{
    int i;

    for (i=0; i<52; i++)
    {
        printf("%10s%7s\n", card[i].suit, card[i].face);
    }
}
```



如何设计主函数？

```
#include <stdio.h>
#include <string.h>
typedef struct card
{
    char suit[10];    /*花色*/
    char face[10];    /*牌面*/
}CARD;
void FillCard(CARD card[]);
void Deal(CARD card[]);
int main()
{
    CARD card[52];
    FillCard(card); /*按序保存一副扑克牌*/
    Deal(card);      /*输出未经洗牌的结果*/
    return 0;
}
```

如何模拟洗牌?



■ 洗牌的基本思路

- * 将52张牌顺序编号
- * 然后随机生成一个编号的序列
 - * 例如, 3, 9, 10, 21, 34, 0, 1, ...
- * **result**的下标代表发牌的顺序号

```
//发牌
void Deal(CARD card[], int result[])
{
    int i;
    for (i=0; i<52; i++)
    {
        printf("%10s%7s\n", card[result[i]].suit,
                card[result[i]].face);
    }
}
```

洗牌前, `result[0]=0`, 表示发的第1张牌是`card[0]`: 花色Spades, 牌面A

洗牌后, `result[0]`不一定为0



```
//洗牌前
int i, result[52];
for (i=0; i<52; i++)
{
    result[i] = i;
}
```




如何模拟洗牌？

■ 打乱编号的算法步骤：

- * step1: 令 $i = 0$
- * step2: 产生 $0 \sim 51$ 的随机数，将其放于 $result[i]$ 内
- * step3: $i = i + 1$
- * step4: 如果 $i < 52$ ，则重复 step2~3，否则，结束循环
- * step5: 输出结果

```
result[0] = 3  
result[1] = 5  
result[2] = 13  
result[3] = 3
```

0	1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	32	33	34	35	36	37	38
39	40	41	42	43	44	45	46	47	48	49	50	51

■ 存在一个致命的问题：

- * 重复 step2 时，产生的随机数可能与以前产生的随机数相同，意味着发出去的 52 张牌中将出现 2 张以上相同的牌



如何模拟洗牌？

■ 解决方法

- * 增加一步，判断新产生的随机数以前是否出现过
- * 如果出现过，则重新生成一个随机数；
- * 如果以前未出现过，则 $i=i+1$

■ 算法步骤：

- * step1: 令 $i = 0$
- * step2: 产生 $0 \sim 51$ 的随机数，将其放于`result[i]`内
- * step3: 判断`result[i]`之前(`result[0]~result[i-1]`)是否出现过
 - 若之前没有出现过，则 $i=i+1$ ，否则直接回到step2
- * step4: 如果 $i < 52$ ，则重复step2~3，否则，结束循环
- * step4: 输出结果



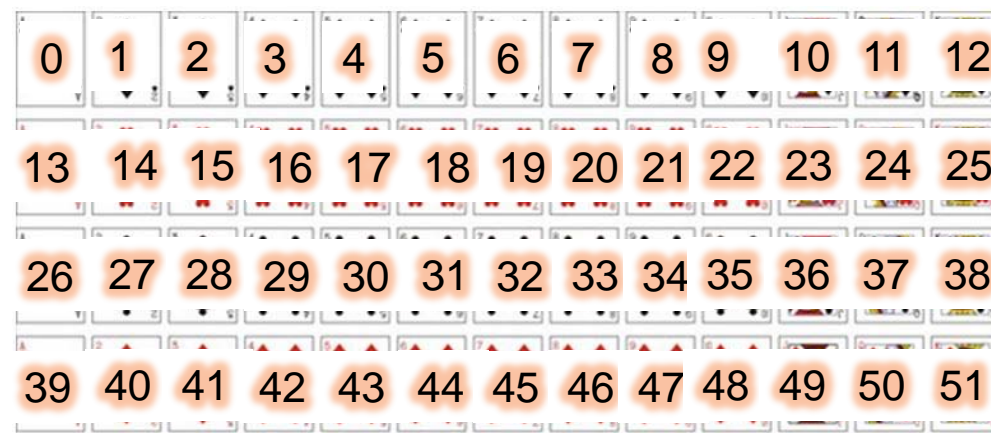
如何模拟洗牌？

■ 算法缺陷

- * 随着生成随机数数量的增加，新的随机数与已产生的随机数相同的可能性越来越大，有可能出现算法延迟问题

■ 高效算法

- * 先将52张牌按照花色与牌面顺序存放
- * 再将其随机打乱
- * 每次循环生成一个0~51之间的随机数 j
- * 然后将 `result` 中的发牌序号 `result[i]` 与随机选出的 `result[j]` 进行交换





如何模拟洗牌？

■ 洗牌算法

- * 每次循环生成一个0~51之间的随机数 j
- * 然后将 `result` 中的发牌序号 `result[i]` 与随机选出的 `result[j]` 进行交换

```
//按新牌顺序编号
void Initialize(int result[])
{
    int i;
    for (i=0; i<52; i++)
    {
        result[i] = i;
    }
}
```

```
//通过对编号随机置乱，实现洗牌
void Shuffle(int result[])
{
    int i, j, temp;
    srand(time(NULL));
    for (i=0; i<52; i++)
    {
        j = rand() % 52;
        temp      = result[i];
        result[i] = result[j];
        result[j] = temp;
    }
}
```

```
#include <string.h>
#include <stdlib.h>
#include <time.h>
typedef struct card
{
    char suit[10];    /*花色*/
    char face[10];    /*牌面*/
}CARD;
void FillCard(CARD card[]);
void Deal(CARD card[], int result[]);
void Initialize(int result[]);
void Shuffle(int result[]);
int main()
{
    CARD card[52];
    int result[52];
    FillCard(card);    //将牌面和花色顺序存放于结构体数组card
    Initialize(result); //将牌编号顺序存放于数组result
    Deal(card, result); //输出洗牌前的结果
    Shuffle(result);    //洗牌
    printf("After Shuffle:\n");
    Deal(card, result); //输出洗牌后的结果
    return 0;
}
```

讨论

- 本节介绍的洗牌程序，只是修改了数组`result`中保存的发牌序号，通过对发牌序号随机乱序实现了洗牌的效果，并没有实际修改结构体数组`card`中按序存放的牌面和花色。
- 请问如果要将结构体数组中的牌面和花色也进行乱序存放，那么程序应该如何修改？请分析对比这两种方法各自的优缺点。

