



第8章

数组

内容提要

- 一维数组
- 多维数组
- 变长数组

标量与聚合变量

- 目前为止 (仅基本类型), 变量都是标量
 - 数学中: 只有大小, 没有方向
 - C中: 保存单一数据项。
- C语言也支持构造 (聚合) 类型:
 - 变量为聚合变量, 存储数据项集合。
- 两种聚合类型: 数组和结构。
 - 数组 (**array**): 一组有序、同类数据 (并列) 的集合, eg, 全班C成绩
 - 结构 (**structure**): 一组不尽相同类型数据 (关联) 的集合, eg, 个人信息, height、weight、sex、name、age

一维数组

- 数据项:

- 元素 (**element**)，根据在数组中的位置进行访问。

- 一维数组:

- 最简单的数组，元素一个接一个地编排在单独一行 (或者一列) 内:



数组声明

- 声明数组元素类型、名称和数据数量
 - `int a[10], c_grade[75];`
- 数组名：
 - 命名规则与变量相同
- 数组元素：
 - 任何类型：基本、构造
 - 在内存中连续存储

数组声明

- 数组长度：
 - 常量(整数)表达式
 - 较好方法：宏定义数组长度：

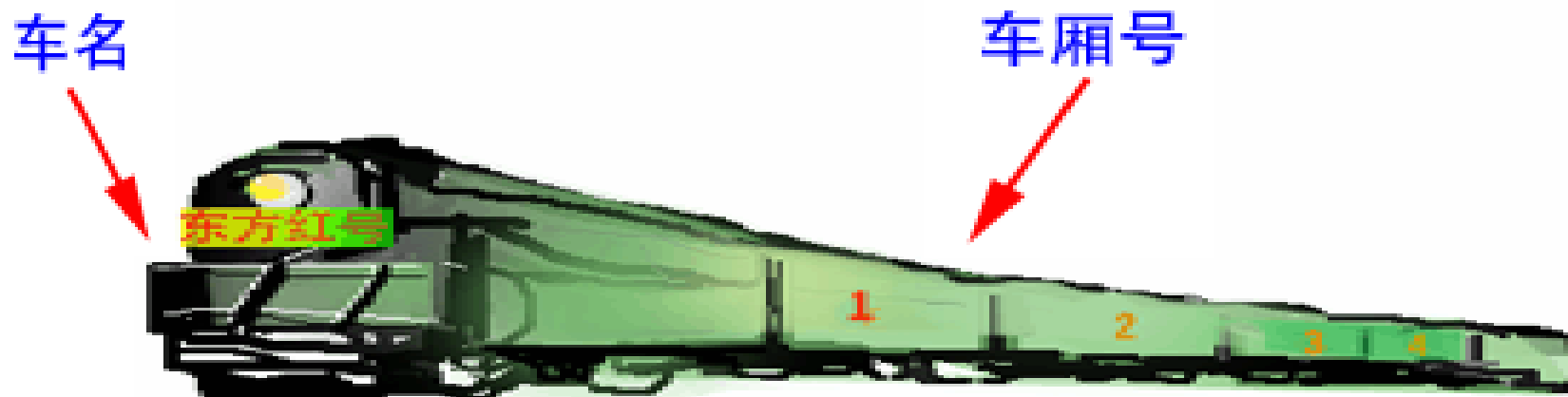
```
#define N 10
```

```
...
```

```
int a[N];
```

数组下标

- (subscripting) 或索引 (indexing)
- 元素在数组中的位置，格式
 - 数组名 [元素位置 (第几个)]
- 存取特定数组元素：



数组元素

- `a[i]` 和普通变量一样使用。
- 类型为数组类型

```
printf("%d\n", a[5]);
```

```
++a[i];
```

- 左值

```
a[0] = 1;
```


数组操作

- 不能整体操作，逐个操作，类似点名
- 循环操作数组每个元素，惯用法**for**循环
- 数组清零

```
int a[N];
```

```
for (i = 0; i < N; i++) a[i] = 0;
```

- 写数组

```
for (i = 0; i < N; i++)
```

```
    scanf("%d", &a[i]);
```

- 累加

```
for (i = 0; i < N; i++) sum +=  
a[i];
```

数组地址

- 首元素地址: `&a[0]`
- 数组名`a`代表: 数组`a`在内存中的首地址

`a == &a[0]`

`scanf ("%d", &a[0]);`

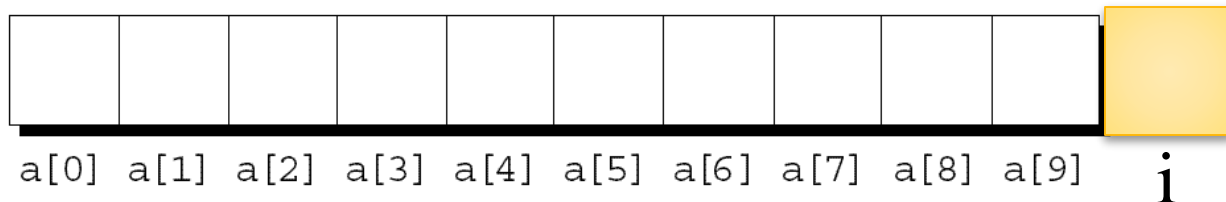
- 等价于

`scanf ("%d", a);`

数组下标问题

- C语言不检查下标范围，造成下标超出（越界）——不可预知的行为。
 - `train t[10]; t[11], t[10]=9`
- 典型错误：**跑偏**——忘记n元素数组下标从0到n-1，而从1到n

```
int a[10], i;  
for (i = 1; i <= 10; i++)  
    a[i] = 0; //可能无限循环。
```



温故而知新

- 数据类型：
 - 标量
 - 聚合（构造）类型：数组，结构
- 数组：一组有序、同类数据（元素）的集合
- 声明（三要素）：

```
type name[length];
```

```
int/float scores[135];
```

```
char name[20];
```

- 长度：常量

温故而知新

- 元素（数组项）
 - 下标访问 $0 \sim N-1$ ，不要跑偏
 - `a[i]`，等同变量，左值
- 地址
 - 数组（首元素）：`name`
 - 元素：`&a[i]`
- 访问：不能整体访问，逐个元素访问
 - `name[i]`， i 为下标（ $0 \sim N-1$ ）；
`for (int i=0; i<N; i++)`
`scanf ("%d", &scores[i]);`

数组下标

- 数组下标可以是任何整数表达式:

- `a[i+j*10] = 0;`

- 表达式可能会有副作用:

- `i = 0; // 下标`

- `while (i < N)`

- `a[i++] = 0;`

- 如果 `a[++i]`, 则跑偏

数组下标

- 注意数组下标副作用:

```
i = 0;
```

```
while (i < N)
```

```
    a[i] = b[i++];
```

- **a[i] = b[i++]:**

- 关乎子表达式运算顺序,
- 可能导致不可预知的行为。

- 避免下标自增自减操作

```
for (i = 0; i < N; i++)
```

```
    a[i] = b[i];
```

程序：数列反向

- 用户录入一串数，然后按反向顺序输出这些数：

Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31

In reverse order: 31 50 11 23 94 7 102 49 82 34

- 思路：
 - 循环读数列
 - 正向存储到数组（下标0到N-1）
 - 反向读数组输出


```
#define N 10
int main(void)
{
    int a[N], i;

    printf("Enter %d numbers: ", N);
    for (i = 0; i < N; i++)
        scanf("%d", &a[i]);

    printf("In reverse order:");
    for (i = N - 1; i >= 0; i--)
        printf(" %d", a[i]);
    return 0;
}
```

数组初始化

- 声明时赋初值
- 通用的格式：
 - **type name[length]={ 常量表达式列表 }**
- 常量表达式列表逗号分隔：

例1: ~~int a[5]={ 2 , 4 , 6 , 8 , 10 } , b[5] ;
b[5]=a[5] ;~~

例2 : ~~int a[5] ;
a[5]={ 2 , 4 , 6 , 8 , 10 } ;~~

数组初始化

- 初始化式短于数组长度，剩余元素赋值为0：

```
int a[10] = {1, 2, 3, 4, 5, 6};  
/*{1, 2, 3, 4, 5, 6, 0, 0, 0, 0}  
*/
```

- 利用该特性，全部数组元素初始化为零

```
int a[10] = {0};  
/* {0, 0, 0, 0, 0, 0, 0, 0, 0,  
0} */
```

- 初始化式长过要初始化的数组：非法。

初始化式确定数组长度

- 给定初始化式，可省略数组长度：

```
int a[] = {1, 2, 3, 4, 5, 6,  
7, 8, 9, 10};
```

- 编译器利用初始化式的长度（常量表达式个数）来确定数组的大小。

指定初始化式(C99)

- 经常有这样的情况：较少元素需显式初始化，其他元素默认赋值。

```
int a[15] =  
{0, 0, 29, 0, 0, 0, 0, 0, 0,  
7, 0, 0, 0, 0, 48};
```

- 对大数组，冗长且易错。

指定初始化式(C99)

- 指定初始化式:

```
int a[15] = { [2] = 29, [9] =  
7, [14] = 48 };
```

- 括号中数字:

- 指示符 (**designator**)——常量表达式

- 赋值简短、易读 (至少对某些数组来说)。赋值顺序任意:

```
int a[15] = { [14] = 48, [9] =  
7, [2] = 29 };
```

程序：检查数是否重复出现数字

- 输入数：28212；程序显示是否有重复
- 思路：
 - 1、逐位获取（遍历）数字
 - 2、记录并检查数字是否出现

```
scanf("%d", &n);  
while (n > 0) {  
    digit = n % 10;//个位数  
    .....;  
    n /= 10;//更新n，抛掉个位}
```



下标（对应数字）： 0 1 2 3 4 5 6 7 8 9

bool digit_seen[10]:

F	T	重	F	F	F	F	F	F	F
---	---	---	---	---	---	---	---	---	---

初始未出现：F；出现：T；重复：break


```
#include <stdbool.h>    /* C99 only */
```

```
//typedef int bool; 非C99
```

```
bool digit_seen[10] = {false};
```

```
int digit; //遍历数位
```

```
long n;
```

```
printf("Enter a number: ");
```

```
scanf("%ld", &n);
```

```
while (n > 0) {
```

```
    digit = n % 10; //个位数
```

```
    if (digit_seen[digit]) break;
```

```
    //重复退出循环
```

```
    digit_seen[digit] = true; //记录数字出现
```

```
    n /= 10; //更新n, 抛掉个位
```

```
}
```

```
if (n > 0) printf("Repeated digit\n");
```

```
else printf("No repeated digit\n");
```

建表

读数

取位

查重

标现

更新n

break退出

程序练习I

- 统计用户输入字符串中各英文字母出现次数，
- 思路：
 - 数组记录26字母出现次数
 - `int letter[26]; letter[0]:` 字母a出现次数.....，初始为零
 - 循环读入字符串
 - 每个字母对号入座，累计出现次数

对数组使用sizeof运算符

- 确定数组及元素的大小 (字节数)。

`sizeof(a)`

`sizeof(a[0])`

`sizeof(a) == sizeof(a[0]) * N`

`int a[10]`

`sizeof(a) = 40;`

对数组使用sizeof运算符

- 求数组长度:

`sizeof(a) = sizeof(a[0]) * N`

`N = sizeof(a) / sizeof(a[0])`

- 计算所得数组长度用于数组操作

- 数组a清零:

`for (i = 0; i < sizeof(a) /
sizeof(a[0]); i++)`

`a[i] = 0;`

- 即使数组长度改变, 也不需要改变循环。

对数组使用sizeof运算符

- `i < sizeof(a) / sizeof(a[0])`: 有些编译器会警告: 危险
 - `sizeof`返回`size_t`(无符号int)
 - 有符号和无符号整数比较产生危险
- 避免警告
 - `(int) (sizeof(a) / sizeof(a[0]))`
 - 注意括号
- 宏定义有帮助:

```
#define SIZE ((int) (sizeof(a) /  
sizeof(a[0])))  
for (i = 0; i < SIZE; i++)  
    a[i] = 0;
```

程序： 计算利息

- 输入起始利率（递增4个利率），年限，显示100美元几年内的价值：

Enter interest rate: 6 //起始利率

Enter number of years: 5

Years	6%	7%	8%	9%	10%
1	106.00	107.00	108.00	109.00	110.00
2	112.36	114.49	116.64	118.81	121.00
3					10
4					41
5					05

数组

输出表头: for(;;)printf ()

for (1-5年)

{

for (6%-10%);

}

```

define NUM_RATES 5//利率数量
define INITIAL_BALANCE 100
.....
int i, low_rate, num_years, year;
double value[5];

scanf("%d%d", &low_rate, &num_years); //起始利率、年数
printf("\nYears");
for (i = 0; i < NUM_RATES; i++) {
    printf("%6d%", low_rate + i);
    value[i] = INITIAL_BALANCE;
} //输出title, 初始化数组
printf("\n");
for (year = 1; year <= num_years; year++)
{ //年循环
    printf("%3d      ", year); //年序号
    for (i = 0; i < NUM_RATES; i++) { //不同利率循环
        value[i] += (low_rate + i) / 100.0 * value[i];
        printf("%7.2f", value[i]);
    }
    printf("\n");
}

```

程序练习3

- 求斐波拉契数列P124-练习题8-5
 - 0, 1, 1, 2, 3, 5, 8.....
 - $f_0=0$; $f_1=1$;
 - $f_n=f_{n-1}+f_{n-2}$;

求前n项——数组实现

```
int fb[N]={0,1},i;  
for (i=2;i<N;i++)  
{  
    fb[i]=fb[i-1]+fb[i-2];  
    printf("%d\n",fb[i]);  
}
```


温故而知新——数组

- 数组初始化

type name[N]={ e_0, e_1, \dots, e_{N-1} } // 常量表达式列表, 不足后面补零

- C99新特性: 指定初始化式

type name[N]={ [i]= e_i , ..., [j]= e_j }

数组应用——记录、统计数据

- 记录字符出现次数：字符、数字转换
- 字母：'a' 'z' → 数组 `letter[26]` 0-25 元素
 - 字母 → 字母表序号：'a' → 0, 'b' → 1 'z' → 25
 - `ch=getchar()` → `i`?
 - `ch-'a'` → `i`

```
While ((ch=getchar()) != '\n') {  
    If (ch>='a' && ch<='z') {  
        i=ch-'a' ; //有用switch实现的  
        letter[i]++;  
    }  
}
```
- 数字字符——数值 '1' → 1
 - `'0'-'0'=0; '1'-'0'=1;`
 - `char n; n-'0' → 数字字符对应数字`

程序练习2——字符格式时间转数值格式时间

- 输入12小时制字符格式时间存入字符数组
- 字符格式转数值格式时间
"11:15pm" → 23:15

char time[10] '1' '1' ':' '1' '5' 'p' 'm'

$h = '1' '1' \rightarrow 23$

'1'(49), '1' → 1, 1 (字符'1'在ASCII表中的序号)

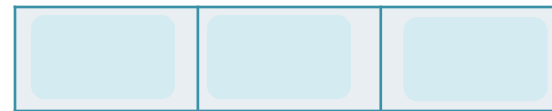
'1' - '0' = 1, $n_{\text{字符}} - '0' = n_{\text{数值}}$

$\rightarrow 1 * 10 + 1 = 11$

$\rightarrow 11 + 12 = 23$

```
int h=0,m=0,i=0;
char time[10]={0},ch;
printf("Enter 12-hours time:");
while((ch=getchar())!='\n') {
    time[i]=ch;
    i++;}
```

```
i=0; //从下标0开始处理小时
while(time[i]!=':') {
    h=h*10+time[i]-'0';
    i++; }
```



```
i++; //跳过':'处理分钟
while(time[i]>='0'&&time[i]<='9') {
    m=m*10+time[i]-'0';
    i++; }
```

```
//处理pm
```

```
if(toupper(time[i])=='P') h+=12;
printf("%d:%d",h,m);
```

求第n项——非数组实现

- 变量版

- $f_0=0$; $f_1=1$;
- $f_n=f_{n-1}+f_{n-2}$;
- 变量: fn 当前项, fm 前一项, fl 前二项

0	1	1	2	3	5	8	
---	---	---	---	---	---	---	-------	--

fl

$fl=fm$

fm

$fm=fn$

fn

$fn=fm+fl$

求第n项——非数组实现

```
int  fn, fm=1, fl=0, i;  
printf("%d\t", fl);  
printf("%d\t", fm);  
for (i=0; i<N; i++)  
{  
    fn=fm+fl;  
    fl=fm;  
    fm=fn;  
    printf("%d\t", fn);  
}
```

程序练习4

- 数组实现P35 3-5
- 任意顺序输入1-16，4*4输出，
- 计算每行、每列和对角线上4数之和
- 输入1-16存入数组value
- 四行和存入数组row
- 四列和存入数组column

冒泡排序（小→大）

6 5 3 1 8 7 2 4

- 思路：

轮次	待排序数个数	比较次数	到最后。
10	N	N-1	
1	N-1	N-2	次（最大
...			
j	N-j	N-j-1	
...			
N-2	2	1	

- 循环控制

```
for (j=0 ; j<N-1 ; j++)
```

```
    for (i=0 ; i<N-j-1 ; i++) //第j轮  
    { 相邻两数比较和对调； }
```



```
int a[N], i, j, t ;
printf("Input %d numbers: \n",N) ;
for(i=0; i<N; i++) scanf("%d",&a[i]) ;
printf("\n") ;
```

```
for(j=0; j<N-1; j++) /*控制N-1轮排序*/
    for(i=0; i<N-j-1; i++) /*每轮比较次数*/
        if(a[i]>a[i+1]){
            t=a[i];
            a[i]=a[i+1];
            a[i+1]=t;
        }
```



```
for(i=0 ; i<N ; i++)printf("%d\t",a[i]) ;
```

多维数组

- 数组可以有任意维数。
- 二维数组 (数学: 矩阵 **matrix**):
int m[5][9]; \\ 声明
- **m** 有 5 行 9 列, 行和列下标从 0 开始:

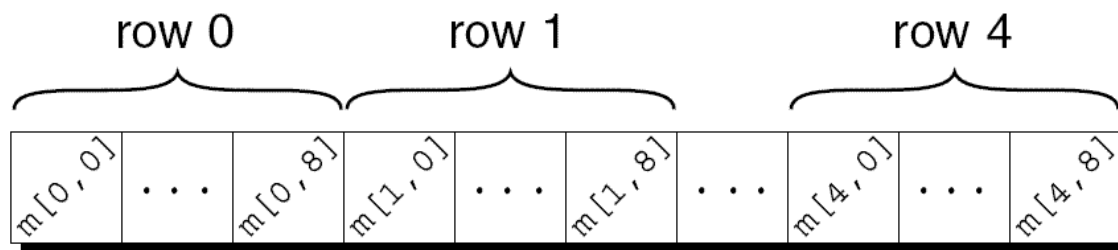
	0	1	2	3	4	5	6	7	8
0									
1									
2									
3									
4									

多维数组元素访问

- $m[i][j]$ 。
 - $[i]$: 第 i 行
 - $[j]$: 第 j 个（列）元素

多维数组存储

- 虽然以表格形式显示二维数组-逻辑结构
- 在计算机内存中按照行主序存储二维数组
 - 从第0行开始，接着第1行，
- 数组 `int m[5][9]` 的存储方式:



多维数组处理

- 逐元素处理，降维（嵌套）处理
- 嵌套for循环：处理多维数组的理想选择
 - 每一维用一个for循环控制，eg二维
- 例，单位矩阵初始化问题

```
#define N 10
double ident[N][N];
int row, col;
for (row = 0; row < N; row++)
    for (col = 0; col < N; col++) { // 进入row行
        if (row == col) // row行、col列元素
            ident[row][col] = 1.0;
        else
            ident[row][col] = 0.0;
        printf.....
    }
```

多维数组初始化

- 给出所有元素初始化式：

```
int m[5][9] = {1, 1, 1, 1, 1, 0, 1, 1, 1,  
               0, 1, 0, 1, 0, 1, 0, 1, 0,  
               0, 1, 0, 1, 1, 0, 0, 1, 0,  
               1, 1, 0, 1, 0, 0, 0, 1, 0,  
               1, 1, 0, 1, 0, 0, 1, 1, 1}
```

- 编译器逐行填充
 - 一行填满后开始填充下一行
- 危险
 - 容易错位：多余（额外）元素或丢失元素

多维数组初始化

- 嵌套一维初始化式:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},  
               {0, 1, 0, 1, 0, 1, 0, 1, 0},  
               {0, 1, 0, 1, 1, 0, 0, 1, 0},  
               {1, 1, 0, 1, 0, 0, 0, 1, 0},  
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

- 高维数组初始化式采用类似的方法。
- C语言为多维数组提供了多种方法来缩写初始化式:

多维数组初始化

- 补0法：初始化少量元素，剩余补0

- 行不足：

- 如，初始化式填满的前三行；后边两行补0：

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},  
               {0, 1, 0, 1, 0, 1, 0, 1, 0},  
               {0, 1, 0, 1, 1, 0, 0, 1, 0}};
```

- 列不足：

- 初始化式不足填满一行，此行其余元素补0：

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},  
               {0, 1, 1},  
               {0, 1, 0, 1},  
               {1, 1, 0, 1, 0, 0, 0, 1},  
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```


多维数组初始化

- C99的指定初始化式:

```
int ident[3][3] = {[0][0] = 1,  
[1][1] = 1, [2][2] = 1};
```

- 没有指定值的元素都默认置为0。

常量数组

- 元素不变的数组
- **const** 数组声明
- 将任何数组（一维或多维）变为“常量”数组

```
const char hex_chars[] =  
{ '0', '1', '2', '3', '4', '5',  
  '6', '7', '8', '9', 'A', 'B',  
  'C', 'D', 'E', 'F' };
```

eg. 学号数组;

常量数组

- 声明数组为**const**的好处：
 - 表明程序不会改变数组。
 - 对编译器发现错误也很有帮助。
- **const**的使用不仅限于数组，但用于定义数组特别有用。

程序：发牌

- 按用户要求数量，给用户随机发牌。

Enter number of cards in hand: 5

Your hand: 7c 2s 5d as 2h

- 扑克表示：
- 花色：梅花clubs、方块diamonds、红桃hearts或黑桃spades
 - `const char suit code[] = {'c', 'd', 'h', 's'};`
- 点数：2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A)
 - `const char rank code[] = {'2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q', 'k', 'a'};`


程序：发牌

- 程序发牌：
 - 一张牌对应两个下标
 - 发一张——产生两个下标
- 需要解决的问题有：
 - 如何随机抽取纸牌（洗牌）？
 - 如何避免两次抽到（产生）相同的牌？

随机抽牌

- 随机选择四种花色
 - `int suit` 随机
- 随机选择13点数
 - `int rank` 随机
- 随机取值
 - `rand()` (`<stdio.h>`) 生成一个随机数。

```
for(i=0;i<20;i++)  
printf("%d\n",rand());
```



```
"D:\C程序设计\..."  
41  
18467  
6334  
26500  
19169  
15724  
11478  
29358  
26962  
24464  
5705  
28145  
23281  
16827  
9961  
491  
2995  
11942  
4827  
5436  
请按任意键继续. . .  
请按任意键继续. . .  
程序第二次  
run
```

随机抽牌——rand()

- 每次产生的随机序列相同——随机数生成器种子相同（默认）
- 设置不同随机数种——洗牌：
 - 撒种： `srand (unsigned int)` 初始化随机数生成器
 - 种子： `time (NULL)`
 - 撒种： `srand ((unsigned) time (NULL))`
 - 一个程序中播种一次即可
- 产生的值大小不一，如何来选花色和点数呢？
- 缩放随机数： `rand () % N`，产生0到N-1
 - **N=4**，选花色
 - **N=13**，选点数

避免重复发一张牌

- 采用 4 x 13 表格记录已发过的牌：
 - 二维bool数组 `in_hand`
 - 行：花色
 - 列：点数

	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'	'10'	'J'	'Q'	'K'	'A'
'c'	F	F	F	T	F	F	...						
'd'	...												
'h'						T							
's'													

1. [2][5]——红桃7——标记为T
2. [0][3]——方块5——标记为T
3. [2][5]——已经标记：已发——另外抽牌

结果输出

- 随机发牌确定牌的花色、点数
 - 随机产生的两个整数作为下标
 - [2], [5]
- 输出结果——两个字符
 - 7h（红桃7）

```
#include <stdbool.h>    /* C99 only */
#include <stdio.h>
#include <stdlib.h>\\随机数处理
#include <time.h>
```

```
#define NUM_SUITS 4//花色数
#define NUM_RANKS 13//每种花色点数
```

```
int main(void)
{
```

```
    bool in_hand[NUM_SUITS][NUM_RANKS] = {false};
```

```
//跟踪发牌情况
```

```
    int num_cards, rank, suit;
```

扑克定义

```
    const char rank_code[] =
    {'2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q',
    'k', 'a'};
```

```
    const char suit_code[] = {'c', 'd', 'h', 's'};
```

```

srand( (unsigned) time(NULL) );
//初始化随机数种，一次就好
printf("Enter number of cards in hand: ");
scanf("%d", &num_cards); //叫牌
printf("Your hand: ");

while (num_cards > 0) { //随机抽牌，for也可
//在此srand(.....) ??
    suit = rand() % NUM_SUITS; /*产生花色*/
    rank = rand() % NUM_RANKS; /*产生点数*/
    if (!in_hand[suit][rank]) { //检查是否发过
        in_hand[suit][rank] = true; //标记
        num_cards--;
        printf("%c%c",
            rank_code[rank], suit_code[suit]);
    }
}
printf("\n");

return 0;
}

```

变长数组(C99)

- C89数组长度必须常量，C99数组长度可变。
 - `int n;`
 - `scanf ("%d", &n);`
 - `int fib[n];`
- 数组长度在程序执行时计算，声明后不变。主要优点：
 - 通过交互或准确计算确定所需元素个数
 - 避免程序员指定长度，过长(浪费内存)或过短(导致程序出错)。

变长数组(C99)

- 数组的长度不一定要用变量来指定，任意表达式都可以：

```
int a[3*i+5];
```

```
int b[j+k];
```

- 像其它数组一样，变长数组也可以是多维的：**int c[m][n];**