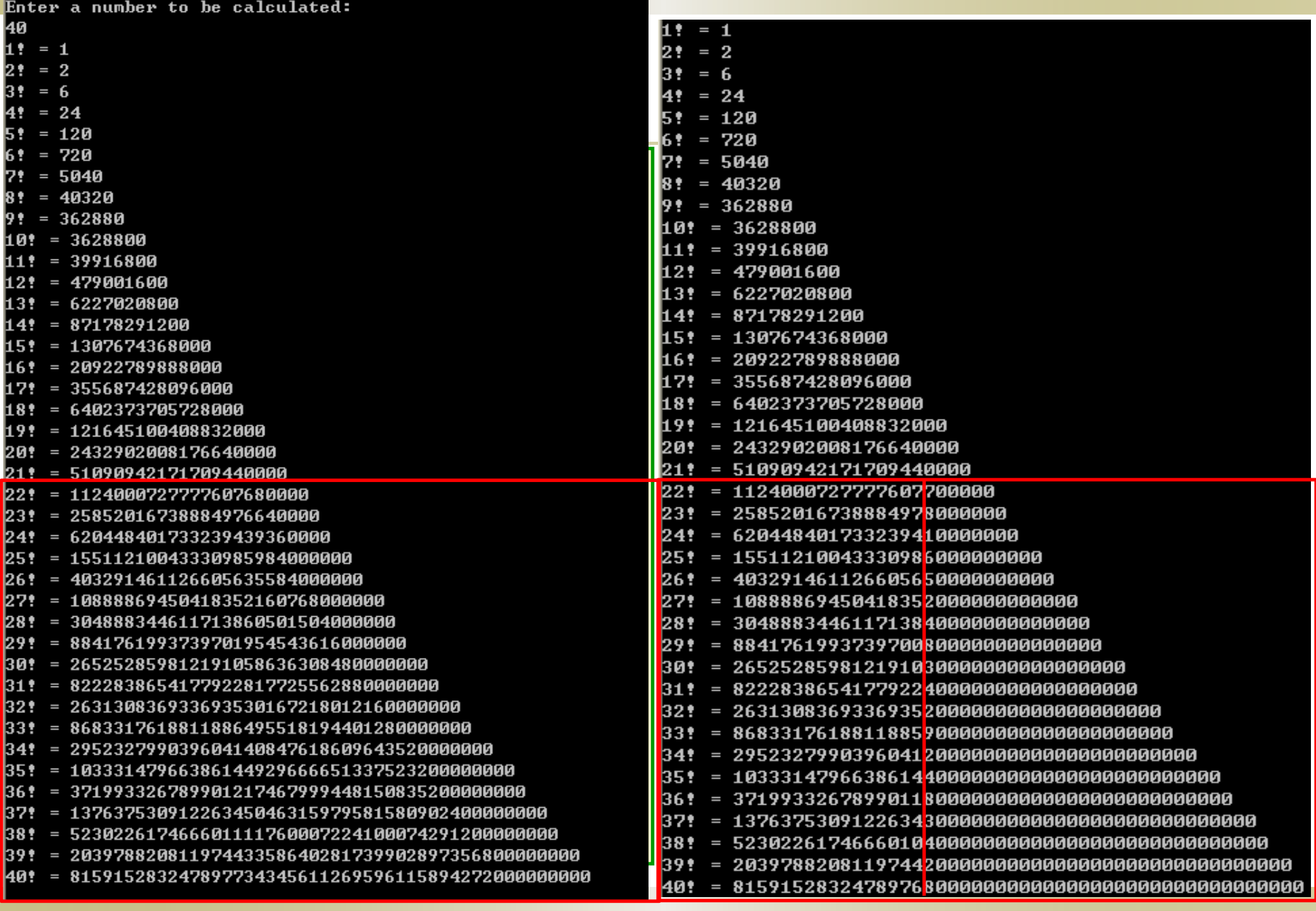
■ 为什么结果会这样？

```
#include <stdio.h>
double Fact(unsigned int n);
int main()
{
    int i;
    for (i=1; i<=40; i++)
    {
        printf("%d! = %.0f\n",i,Fact(i));
    }
    return 0;
}
double Fact(unsigned int n)
{
    unsigned int i;
    double result = 1;
    for (i=1; i<=n; i++)
    {
          result = result * i;
    }
    return result;
}
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605650322784000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000
31! = 8222838654177922817725562880000000
32! = 263130836933693530167218012160000000
33! = 8683317618811886495518194401280000000
34! = 295232799039604140847618609643520000000
35! = 10333147966386144929666651337523200000000
36! = 371993326789901217467999448150835200000000
37! = 13763753091226345046315979581580902400000000
38! = 523022617466601904792627223700074291200000000
39! = 20397882081197443358640281739902897356800000000
40! = 815915283247897734345611269596115894272000000000
```


Enter a number to be calculated:
40
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000
31! = 8222838654177922817725562880000000
32! = 263130836933693530167218012160000000
33! = 8683317618811886495518194401280000000
34! = 295232799039604140847618609643520000000
35! = 10333147966386144929666513375232000000000
36! = 371993326789901217467999448150835200000000
37! = 13763753091226345046315979581580902400000000
38! = 523022617466601111760007224100074291200000000
39! = 20397882081197443358640281739902897356800000000
40! = 815915283247897734345611269596115894272000000000
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607700000
23! = 25852016738884978000000
24! = 620448401733239410000000
25! = 15511210043330986000000000
26! = 403291461126605650000000000
27! = 10888869450418352000000000000
28! = 304888344611713840000000000000
29! = 8841761993739700800000000000000
30! = 265252859812191030000000000000000
31! = 8222838654177922400000000000000000
32! = 263130836933693520000000000000000000
33! = 8683317618811885900000000000000000000
34! = 295232799039604120000000000000000000000
35! = 10333147966386144000000000000000000000000
36! = 3719933267899011800000000000000000000000000
37! = 13763753091226343000000000000000000000000000
38! = 523022617466601040000000000000000000000000000
39! = 20397882081197442000000000000000000000000000000
40! = 815915283247897680000000000000000000000000000000

第7讲 数组

教材8.3~8.5节

MOOC第8周

哈尔滨工业大学

苏小红

sxh@hit.edu.cn

“路见不平一声吼，该出手时就出手”

- 梁山好汉一百单八将
  - 梁山好汉[第一位]----宋江
  - 梁山好汉[第二位]----卢俊义
  - 。。。

图片来自百度

- 数组名
- 有108个元素的一维数组

四问一维数组

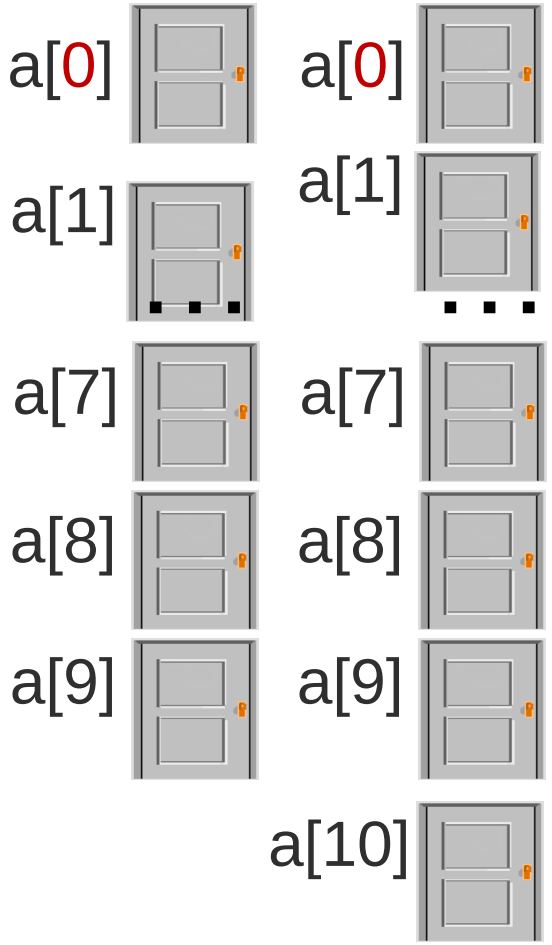
- 1. 如何定义？
  - 定义一个有10个int型元素的一维数组

```
int a[10];
```

```
#define N 10
int a[N];
```

```
#define N 10
int a[N+1];
```

- 2. 在内存中是如何存储的？
  - 顺序存储，随机访问



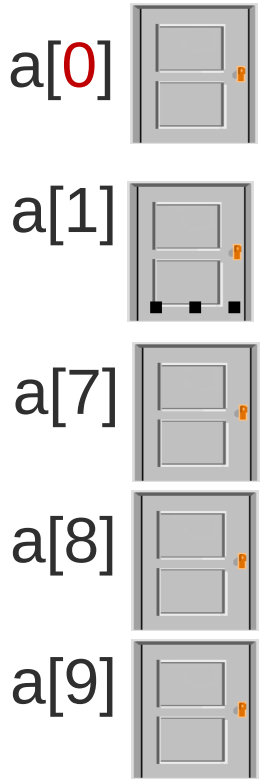
四问一维数组

- 用1个下标（Subscript）表示一维数组元素
- 3. 如何确定元素的位置？
  - 数组名——数组的首地址 `&a[0]`
    - 下标代表数组元素相对于首地址的偏移量

`a[i]`

`&a[i]`

`a+i`



图片来自百度

四问一维数组

- 4. 未初始化的元素值是否都是随机数？

short a[10]

a[0]	2字节
a[1]	2字节
…	…
a[9]	2字节

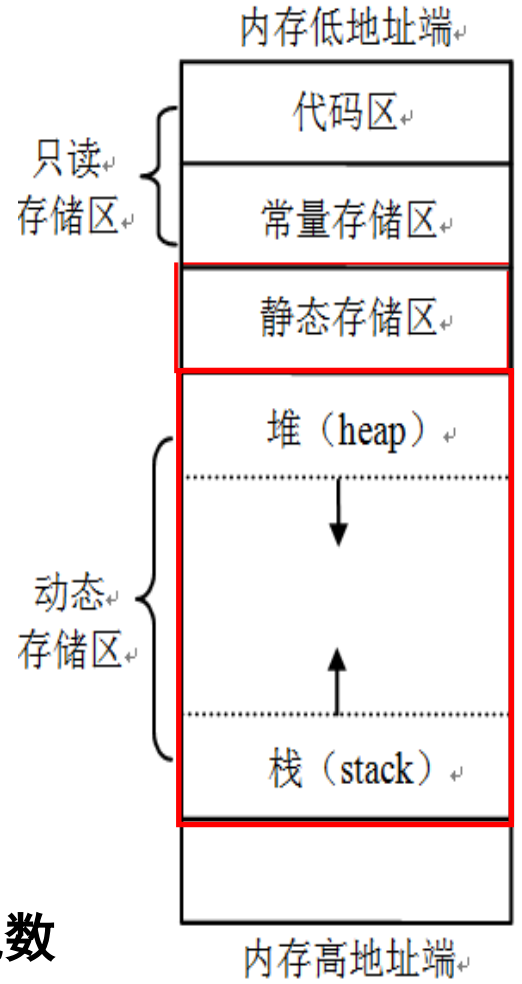
char name[8]

name[0]	1字节
name[1]	1字节
…	…
name[7]	1字节

static float x[8]

x[0]	4字节
x[1]	4字节
…	…
x[7]	4字节

动态存储区　　静态存储区



- 数组的**数据类型**——每个元素在内存中占多少个**字节**
- 数组的**存储类型**——在**动态**还是**静态**存储区分配内存
  - **静态**数组和**全局**数组**自动初始化为0值**，否则，是随机数

三问二维数组

- 1. 如何定义？
  - 定义有4行5列int型元素的二维数组：

```
int b[4][5];
```

```
#define  M  4
#define  N  5
int  a[M][N];
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

- 2. 在内存中是如何存储的？

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

三问二维数组

- 用两个下标表示二维数组元素
- 3. 如何确定元素的位置？
  - 第一个数组元素的首地址
    - b[i][j]相对于首地址的偏移量？

b[i][j]

&b[0][0]

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

b[3][2]

&b[i][j]　　&b[0][0] + i*N + j

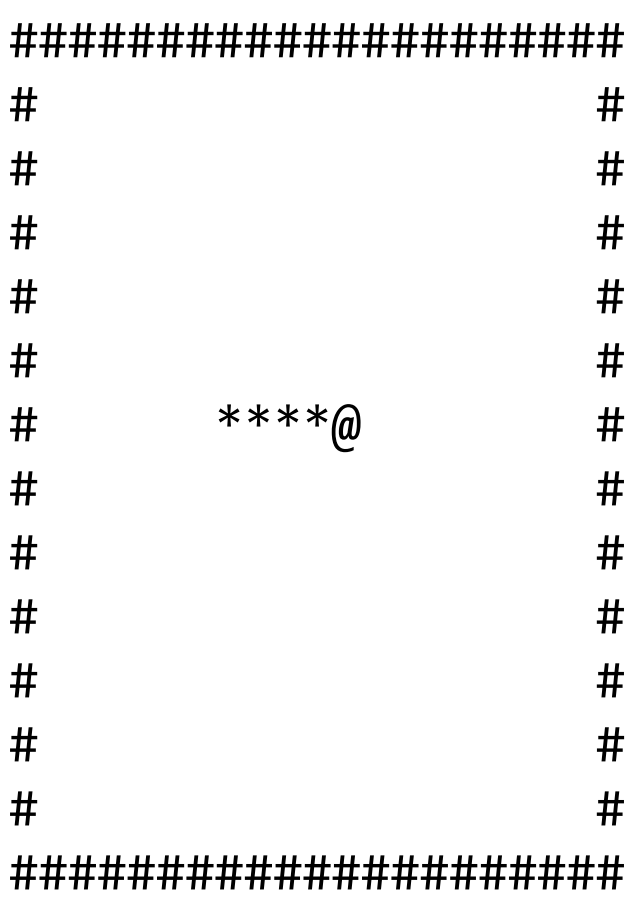
数组有什么用？

- 贪吃蛇游戏
- 二维数组存储画布中的元素
  - 0：空格；-1：边框#；1：蛇头@；>1的正数：蛇身*
  - int canvas[HIGH][WIDTH] = {0};

```
########################
#                      #
#                      #
#                      #
#                      #
#                      #
#       ****@          #
#                      #
#                      #
#                      #
#                      #
#                      #
#                      #
#                      #
########################
```

贪吃蛇游戏

- 二维数组存储画布中的元素
  - 0：空格；-1：边框#；1：蛇头@；>1的正数：蛇身*
  - `int canvas[HIGH][WIDTH] = {0};`
- 小蛇的移动



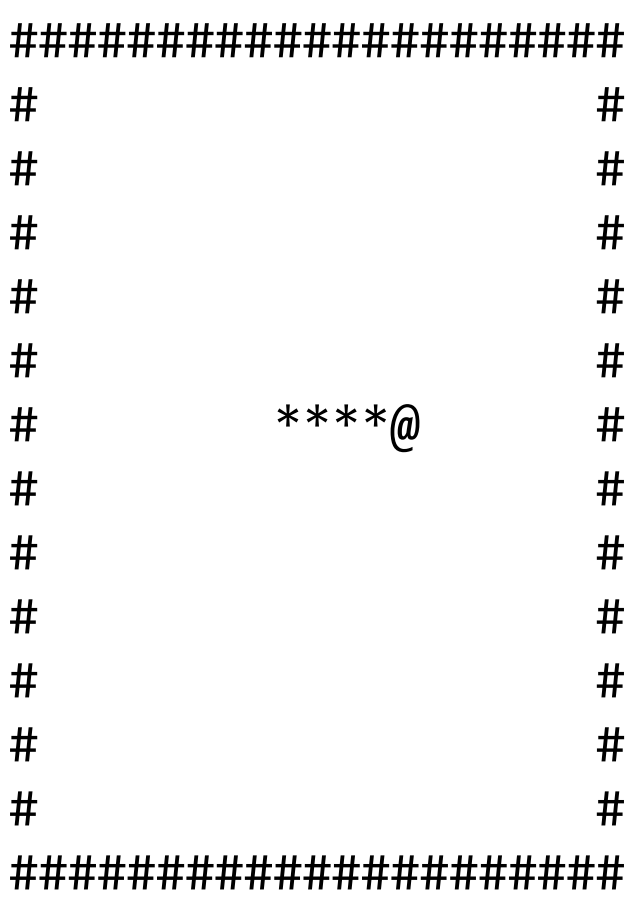
贪吃蛇游戏

- 二维数组存储画布中的元素
  - 0：空格；-1：边框#；1：蛇头@；>1的正数：蛇身*
  - `int canvas[HIGH][WIDTH] = {0};`
- 小蛇的移动

```
##########################
#                        #
#                        #
#                        #
#                        #
#                        #
#        ****@           #
#                        #
#                        #
#                        #
#                        #
#                        #
#                        #
#                        #
##########################
```

贪吃蛇游戏

- 二维数组存储画布中的元素
  - 0：空格；-1：边框#；1：蛇头@；>1的正数：蛇身*
  - `int canvas[HIGH][WIDTH] = {0};`
- 小蛇的移动



贪吃蛇游戏

- 二维数组存储画布中的元素
  - 0：空格；-1：边框#；1：蛇头@；>1的正数：蛇身*
  - `int canvas[HIGH][WIDTH] = {0};`
- 小蛇的移动

```
##########################
#                        #
#                        #
#                        #
#                        #
#                        #
#                        #
#            ****@       #
#                        #
#                        #
#                        #
#                        #
#                        #
#                        #
#                        #
#                        #
##########################
```

贪吃蛇游戏

- **小蛇的移动**

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	5	4	3	2	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

所有大于0的元素加1

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	6	5	4	3	2	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

去尾加头
最大值变为0，2右方/上方/下方的元素变为1

右移

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

上移

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	5	4	3	2	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

下移

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	5	4	3	2	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0

贪吃蛇游戏

- 小蛇吃食物增加长度

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	5	4	3	2	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

所有大于0的元素加1

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	6	5	4	3	2	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

2右方/上方/下方的元素变为1

右移

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

上移

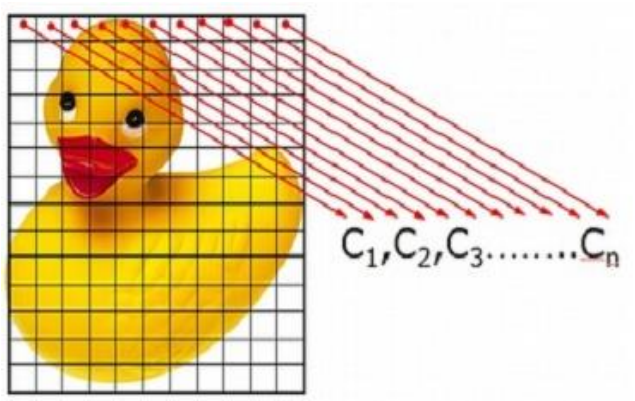
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0
0	0	6	5	4	3	2	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

下移

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	6	5	4	3	2	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0

数组还有什么用？

- 矩阵运算
- 表格数据处理
- 保存图像等批量数据
- 向函数传递批量数据
- ……



图片来自百度

● 图像的最小单位是像素

数组有什么用？

返回Fibonacci数列的**第n项**

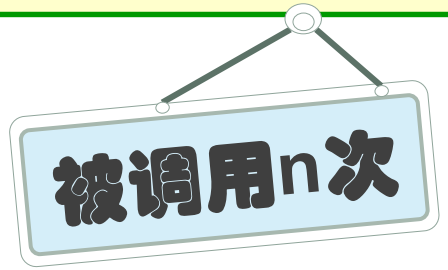
```
long Fib(int n)
{
    int i;
    long f[N]={0,1,1};
    for (i=3; i<=n; i++)
    {
      f[i] = f[i-1] + f[i-2];
    }
    return f[n];
}
```


```
long Fib(int n)
{
  if (n == 1)
     return 1;
  else if (n == 2)
     return 1;
  else
     return Fib(n-1) + Fib(n-2);
}
```

数组有什么用？

返回Fibonacci数列的**第n项**

```
long Fib(int n)
{
    int i;
    long f[N]={0,1,1};
    for (i=3; i<=n; i++)
    {
      f[i] = f[i-1] + f[i-2];
    }
    return f[n];
}
```



若要求出**前n项**？

```
#include <stdio.h>
#define N 20
long Fib(int n);
int main()
{
   int n, i;
   long x;
   printf("Input n:");
   scanf("%d", &n);
   for (i=1; i<=n; i++)
   {
      x = Fib(i);
      printf("Fib(%d)=%ld\n",i,x);
   }
   return 0;
}
```

数组有什么用？

返回Fibonacci数列的第n项

```
long Fib(int n)
{
    int i;
    long f[N]={0,1,1};
    for (i=3; i<=n; i++)
    {
      f[i] = f[i-1] + f[i-2];
    }
    return f[n];
}
```

图片来自百度

返回Fibonacci数列的前n项

```
void Fib(long f[], int n)
{
    int i;
    f[1]=1;
    f[2]=1;
    for (i=3; i<=n; i++)
    {
      f[i] = f[i-1] + f[i-2];
    }
}
```

图片来自百度

向函数传递一维数组

```
void Fib(long f[], int n)
{
    int i;
    f[1]=1;
    f[2]=1;
    for (i=3; i<=n; i++)
    {
      f[i] = f[i-1] + f[i-2];
    }
}
```

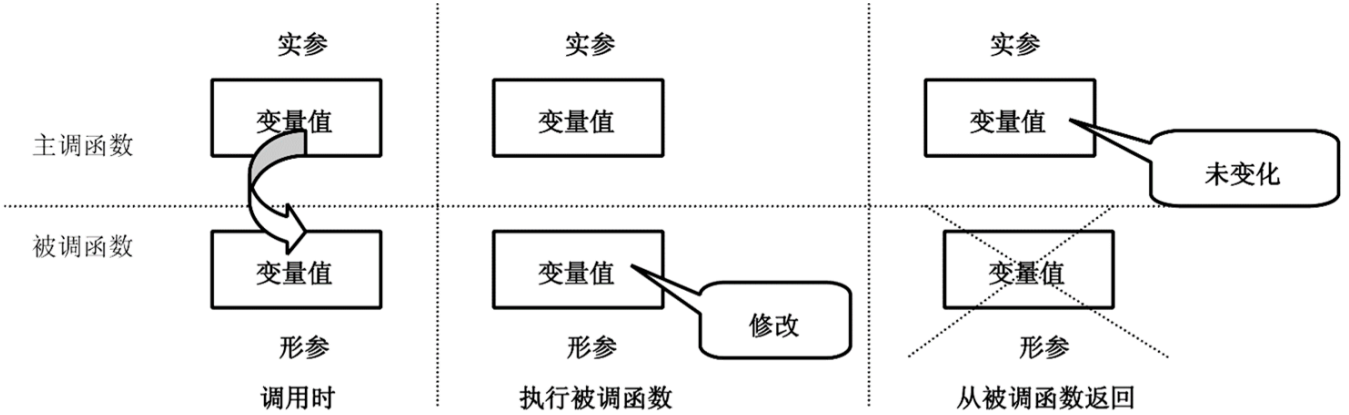
通常不指定一维数组的长度
用一个形参指定数组的大小

用不带下标的数组名
做函数实参

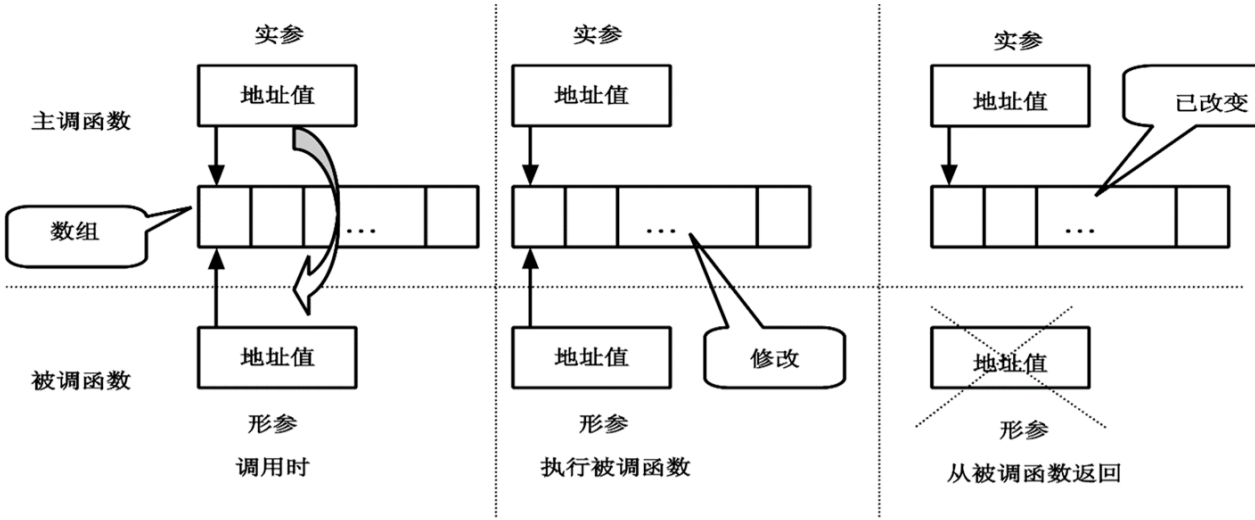
```
#include <stdio.h>
#define N 20
void Fib(long f[], int n);
int main()
{
    int n, i;
    long f[N];
    printf("Input n:");
    scanf("%d", &n);

    Fib(f, n);

    for (i=1; i<=n; i++)
    {
        printf("Fib(%d)=%ld\n",i,f[i]);
    }
    return 0;
}
```

为什么数组参数可以返回元素值？

Call-by-Value



Simulating Call-by-Reference



奇思妙想

- 如何计算Fibonacci数列各项（1，1，2，3，5，8，…不算0）的平方和?

$$fib^2(1) + fib^2(2) + \ldots + fib^2(n)$$

奇思妙想

- 如何计算Fibonacci数列各项（1，1，2，3，5，8，…不算0）的平方和?

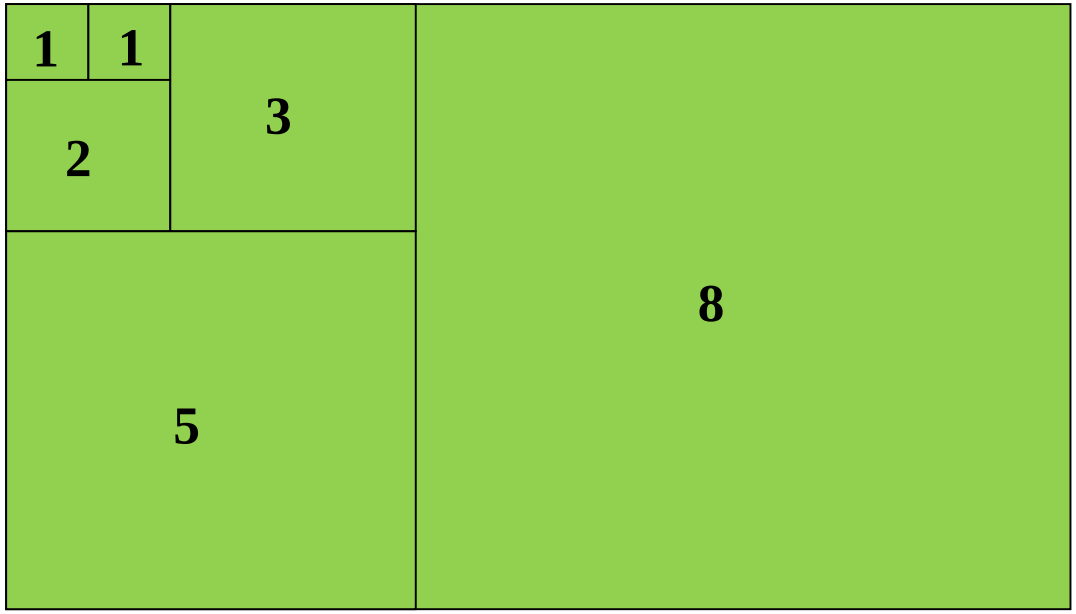
$$fib^2(1)+fib^2(2)+\ldots+fib^2(n)$$

```
int main()
{
    int n, i;
    long f[N], sum = 0;
    printf("Input n:");
    scanf("%d", &n);
    Fib(f, n);
    for (i=1; i<=n; i++)
    {
        sum = sum + f[i]*f[i];
    }
    printf("sum=%ld\n", sum);
    return 0;
}
```

奇思妙想

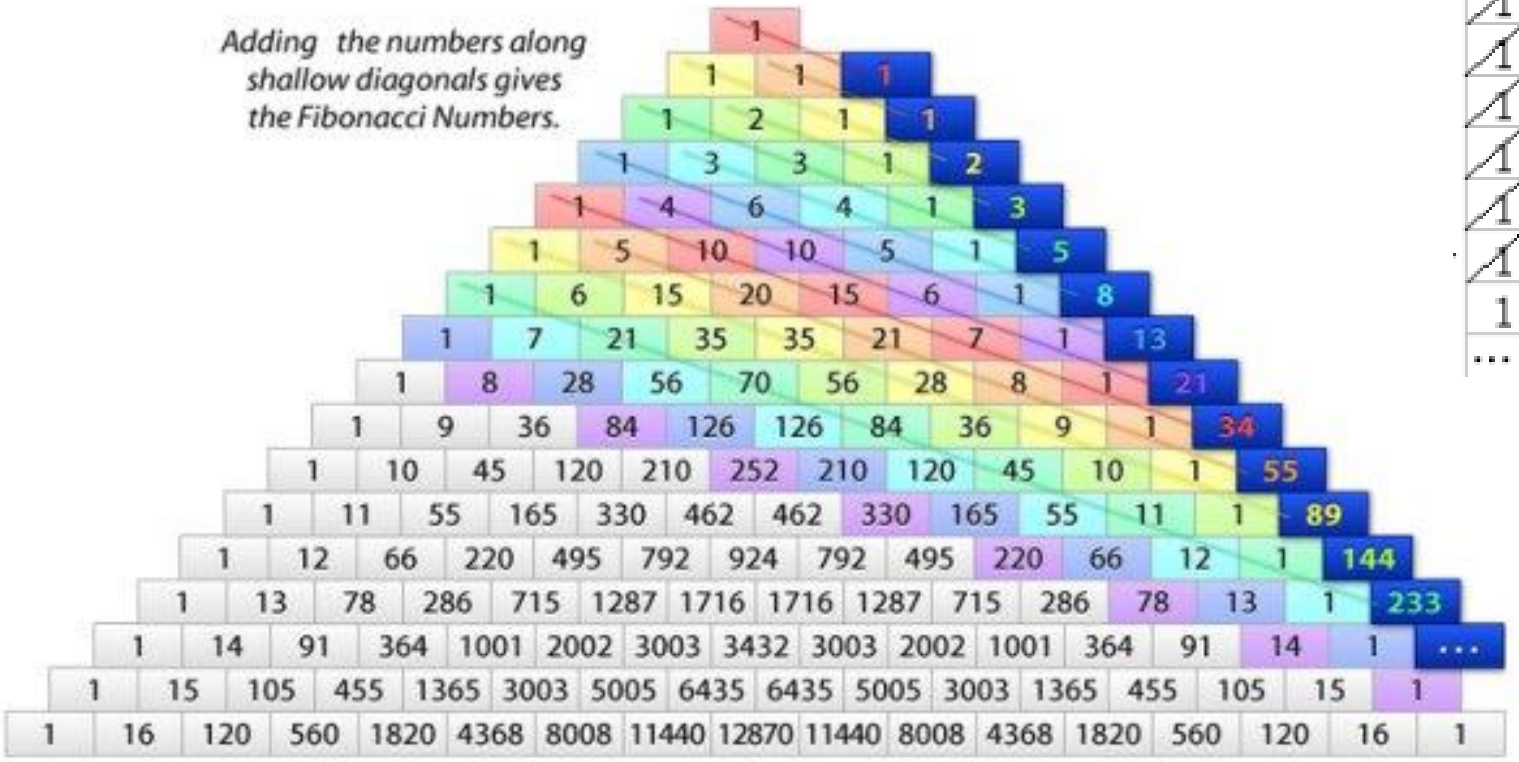
- 如何快速计算Fibonacci数列各项的平方和？

$$F_1^2 + F_2^2 + \ldots + F_n^2 = F_n F_{n+1}$$



奇思妙想

- Fibonacci数列与杨辉三角之间的神秘关系？



F	1	1	2	3	5	8
1						
1	1					
1	2	1				
1	3	3	1			
1	4	6	4	1		
1	5	10	10	5		
1	6	15	20	15		
…						

按线求和即得裴氏数列

奇思妙想

- 你知道Fibonacci数列为什么有名吗？

$$\frac{F_{n+1}}{F_n}$$

$\frac{1}{1} = 1.000000000$

$\frac{2}{1} = 2.000000000$

$\frac{3}{2} = 1.500000000$

$\frac{5}{3} = 1.666666667$

$\frac{8}{5} = 1.600000000$

$\frac{13}{8} = 1.625000000$

$\frac{21}{13} = 1.615384615$

$\frac{34}{21} = 1.619047619$

$\frac{55}{34} = 1.617647059$

$\frac{89}{55} = 1.6182181618$

$\frac{144}{89} = 1.617977528$

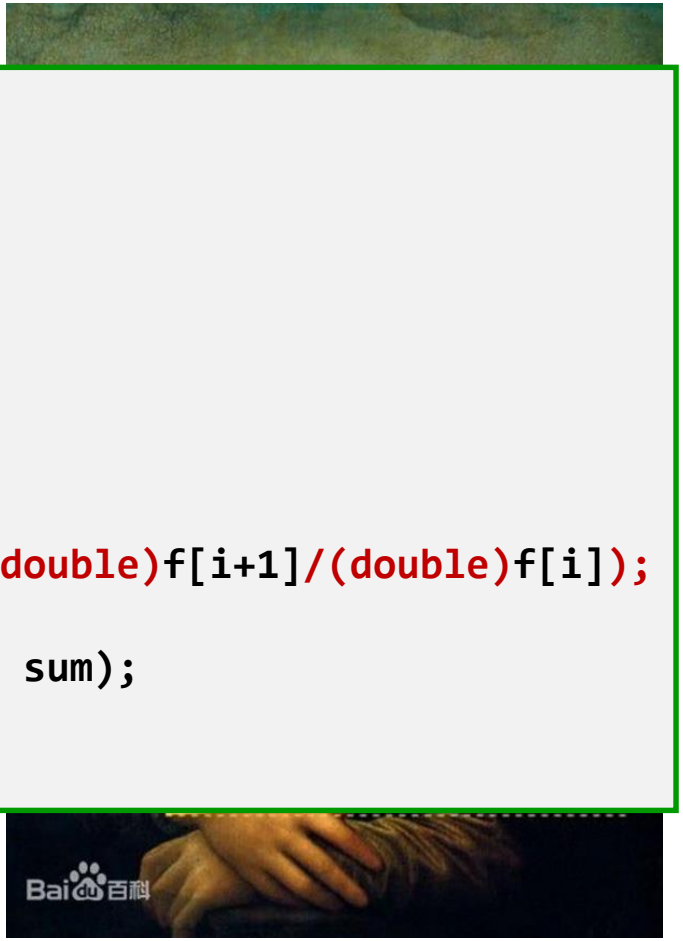
$\frac{233}{144} = 1.618055556$

$\frac{377}{233} = 1.618025751$

$\frac{610}{377} = 1.618037135$

$\frac{987}{610} = 1.618032787$

```
int main()
{
    int n, i;
    long f[N];
    printf("Input n:");
    scanf("%d", &n);
    Fib(f, n);
    for (i=1; i<n; i++)
    {
        printf("%f\n", (double)f[i+1]/(double)f[i]);
    }
    printf("sum=%ld\n", sum);
    return 0;
}
```



黄金分割比与黄金分割数列

图片来自百度

设置

单选题

下列说法错误的是（）。

A 简单变量做函数参数时，是将实参的值传给形参，实参和形参在内存中占用不同的存储单元，因此形参值的改变不会影响实参。

B 数组做函数参数时，是将实参数组的首地址传给形参，形参数组和实参数组在内存中共享相同的存储单元，因此对形参数组元素值的修改也就相当于是对实参数组元素值的修改。

C 在声明函数的二维数组形参时，可省略数组第二维的长度，但不能省略数组第一维的长度。

D 在声明函数的一维数组形参时，通常不指定数组的大小，而用另一个形参来指定数组的大小。

提交

敢来和我比拼一下速度吗？

- 计算阿姆斯特朗数（ Armstrong number）
  - 是一个n位数，本身等于各位数的n次方加和。
  - 从键盘输入数据的位数n（n<=9），编程输出n位的阿姆斯特朗数

```
unsigned long ArmstrongNum(
        unsigned long number,
        unsigned int n)
{
   unsigned int digit;
   unsigned long m=number, sum=0;
   while (number != 0)
   {
        digit = number % 10;
        sum = sum + pow(digit, n);
        number = number / 10;
   }
   if (m == sum)  return sum;
   else return 0;
}
```

```
int main()
{
    unsigned int n;
    unsigned long head, tail, digit;
    printf("Input digit bits:");
    scanf("%u", &n);
    head = pow(10, n-1);
    tail = pow(10, n) - 1;
    for(; head<tail; head++)
    {
       digit = ArmstrongNum(head, n);
       if (digit!=0) printf("%lu\n",digit);
    }
    return 0;
}
```

F:\c\test\bin\Debug\test.exe

```
Input digit bits:7
2767074
4210818
```

F:\c\test\bin\Debug\test.exe

```
Input digit bits:8
88593477
```

F:\c\test\bin\Debug\test.exe

```
Input digit bits:7
1741725
4210818
9800817
9926315
```

F:\c\test\bin\Debug\test.exe

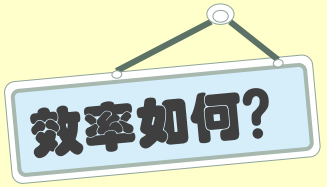
```
Input digit bits:8
24678050
24678051
88593477
```

当n=7时double赋值给long，数值溢出

```
unsigned long ArmstrongNum(
      unsigned long number,
      unsigned int n)
{
  unsigned int digit;
  unsigned long m=number, sum=0;
  while (number != 0)
  {
      digit = number % 10;
      sum = sum + pow(digit, n);
      number = number / 10;
  }
  if (m == sum)  return sum;
  else return 0;
}
```

```
int main()
{
    unsigned int n;
    unsigned long head, tail, digit;
    printf("Input digit bits:");
    scanf("%u", &n);
    head = pow(10, n-1);
    tail = pow(10, n) - 1;
    for(; head<tail; head++)
    {
       digit = ArmstrongNum(head, n);
       if (digit!=0) printf("%lu\n",digit);
    }
    return 0;
}
```

```
unsigned long ArmstrongNum(unsigned long number, unsigned int n)
{
  unsigned int digit;
  unsigned long m = number;
  double sum = 0;
  while (number != 0)
  {
      digit = number % 10;
      sum = sum + pow(digit, n);
      number = number / 10;
  }
  if (m == (unsigned long)sum)  return sum;
  else return 0;
}
```



```c
#include <stdio.h>
unsigned long ArmstrongNum(unsigned long number, unsigned int n);
unsigned long pw[10][10]= {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                           {1, 1, 1 ,1, 1, 1, 1, 1, 1, 1},
                           {1, 2, 4, 8, 16, 32, 64, 128, 256, 512},
                           {1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683},
                           {1, 4, 16, 64, 256, 1024, 4096, 16384, 65536, 262144},
                           {1, 5, 25, 125, 625, 3125, 15625, 78125, 390625, 1953125},
                           {1, 6, 36, 216, 1296, 7776, 46656, 279936, 1679616, 10077696},
                           {1, 7, 49, 343, 2401, 16807, 117649, 823543, 5764801, 40353607},
                           {1, 8, 64, 512, 4096, 32768, 262144, 2097152, 16777216, 134217728},
                           {1, 9, 81, 729, 6561, 59049, 531441, 4782969, 43046721, 387420489}
                          };
unsigned long pw10[10]={1,10,100,1000,10000,100000,1000000,10000000,100000000,1000000000};
```

```c
unsigned long ArmstrongNum(
      unsigned long number,
      unsigned int n)
{
  unsigned int digit;
  unsigned long m=number, sum=0;
  while (number != 0)
  {
      digit = number % 10;
      sum = sum + pw[digit][n];
      number = number / 10;
  }
  if (m == sum)  return sum;
  else return 0;
}
```

```c
int main()
{
    unsigned int n;
    unsigned long head, tail, digit;
    printf("Input digit bits:");
    scanf("%u", &n);
    head = pw10[n-1];
    tail = pw10[n] - 1;
    for(; head<tail; head++)
    {
       digit = ArmstrongNum(head, n);
       if (digit!=0) printf("%lu\n",digit);
    }
    return 0;
}
```

程序优化技巧

- 构造一个查找表，以空间换时间，简化和优化程序

图片来自百度

优化三天打鱼两天晒网程序

- 中国有句俗语叫“三天打鱼两天晒网”，某人从1990年1月1日起开始“三天打鱼两天晒网”，即工作三天，然后再休息两天。从键盘任意输入某年某月某天，编程判断他是在工作还是在休息。
  - 如果是在工作，则输出“He is working”
  - 如果是在休息，则输出“He is having a rest”
  - 如果输入非法字符或者输入的年份小于1990或者输入的月份和日期不合法，则输出"Input error!" 。
  - 输入提示信息： "Input year,month,day:"

```c
#include <stdio.h>
#include <stdlib.h>
int WorkORrest(int year, int month, int day);
int IsLeapYear(int y);
int IsLegalDate(int year, int month, int day);
int main()
{
    int year, month, day, n, ret;
    do{
        printf("Input year,month,day:");
        n = scanf("%d,%d,%d", &year, &month, &day);
        if (n != 3) while (getchar() != '\n');
    }while (n != 3);//处理非法字符输入
    if (!IsLegalDate(year, month, day))
    {
        printf("Input error!\n");
        exit(0);
    }
    ret = WorkORrest(year, month, day);
    if (ret == 1)
        printf("He is working\n");
    else
        printf("He is having a rest\n");
    return 0;
}
```

```
//函数功能：判断日期year,month,day是否合法，若合法，则返回1，否则返回0
int IsLegalDate(int year, int month, int day)
{
    int dayofmonth;
    if (year<1 || month<1 || month>12 || day<1)  return 0;
    switch (month)
    {
      case 1: case 3: case 5: case 7: case 8: case 10: case 12:
              dayofmonth = 31;
              break;
      case 2: if (IsLeapYear(year))
                  dayofmonth = 29; //闰年29天
              else
                  dayofmonth = 28; //平年28天
              break;
      case 4: case 6: case 9: case 11:
              dayofmonth = 30;
              break;
    }
    return day > dayofmonth ? 0 : 1;
}
```

```
int IsLegalDate(int year, int month, int day)
{
  int dayofmonth0[12]={31,28,31,30,31,30,31,31,30,31,30,31};
  int dayofmonth1[12]={31,29,31,30,31,30,31,31,30,31,30,31};
  if (year<1 || month<1 || month>12 || day<1)  return 0;
  if (IsLeapYear(year))
  {
     return day > dayofmonth1[month-1] ? 0 : 1;
  }
  else
  {
     return day > dayofmonth0[month-1] ? 0 : 1;
  }
}
```

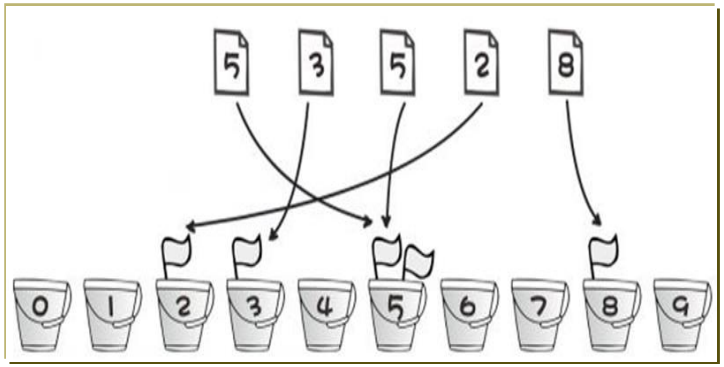
```
int IsLegalDate(int year, int month, int day)
{
  int leap;
  int dayofmonth[2][12]={{31,28,31,30,31,30,31,31,30,31,30,31},
                         {31,29,31,30,31,30,31,31,30,31,30,31}
                        };
  if (year<1 || month<1 || month>12 || day<1)  return 0;
  leap = IsLeapYear(year) ? 1 : 0;
  return day > dayofmonth[leap][month-1] ? 0 : 1;
}
```

```
//函数功能：某人三天打鱼两天晒网，判断year年month月day日是工作还是休息
//函数参数：year,month,day，分别代表年、月、日
//函数返回值：返回1，表示工作，返回-1，表示休息，返回0表示参数错误
int WorkORrest(int year, int month, int day)
{
    int i, sum = 0, leap;
    int dayofmonth[2][12]={{31,28,31,30,31,30,31,31,30,31,30,31},
                           {31,29,31,30,31,30,31,31,30,31,30,31}
                          };
    //累计前面年份的天数
    for (i=1990; i<year; i++)
    {
        sum = sum + IsLeapYear(i) ? 366 : 365;
    }
    //累计当年前面月份的天数
    for (i=1; i<month; i++)
    {
        leap = IsLeapYear(year) ? 1 : 0;
        sum = sum + dayofmonth[leap][i-1];
    }
    //累计当月的天数
    sum = sum + day;

    sum = sum % 5;

    return sum == 0 || sum == 4 ? -1 : 1;
}
```

简单而又奢侈的数据统计方法

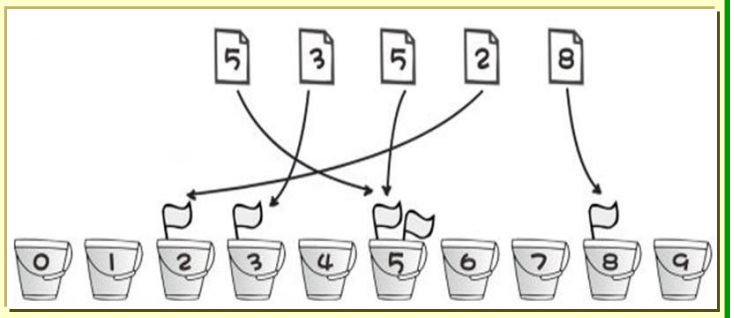
- 从键盘输入10种商品的编号（0~9之间的数），输出购买次数最多的商品编号-众数
  - 例如，输入 5 3 5 2 8，输出5（而非2）

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	0	2	0	0	0	0
0	0	1	1	0	2	0	0	0	0
0	0	1	1	0	2	0	0	1	0



一个萝卜一个坑

```
#define N 10
//函数功能：计算n个数的众数，不考虑两个或两个以上出现次数相同的情况
int ModeVal(int a[], int n)
{
   int  i, j, max, modeValue, count[N+1] = {0};
   for (i=0; i<n; i++)
   {
        count[a[i]]++;  //统计每个等级的出现次数
   }
   //统计出现次数最多的数
   for (max=count[0], modeValue=0, j=1; j<N; j++)
   {
        if (count[j] > max)
        {
            max = count[j]; //记录出现次数的最大值
            modeValue = j;  //记录出现次数最多的数，即最大值所在下标位置
        }
   }
   return modeValue; //返回出现次数最多的数，而非出现最多的次数
}
```



素数探求——求100以内的所有素数

```
#include <stdio.h>
#include <math.h>
int IsPrime(int x);
int main()
{
   int i;
   for (i=1; i<=100; i++)
   {
      if (IsPrime(i))
      {
         printf("%d\t", i);
      }
   }
   printf("\n");
   return 0;
}
```

```
int IsPrime(int x)
{
  int i, squareRoot;
  if (x <= 1) return 0;
  squareRoot = (int)sqrt(x);
  for (i=2; i<=squareRoot; i++)
  {
     if (x%i == 0)  return 0;
  }
  return 1;
}
```

筛法——求100以内的所有素数

`int a[N+1];` (N值为100)

初始化：令`a[2]=2,a[3]=3,.....,a[N]=N`

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

筛2的倍数

2	3	0	5	0	7	0	9	0	11	0	13	0	15	0	17

筛3的倍数

2	3	0	5	0	7	0	0	0	11	0	13	0	0	0	17

筛5的倍数

2	3	0	5	0	7	0	0	0	11	0	13	0	0	0	17

依次从a中筛掉2的倍数，3的倍数，5的倍数，.....，sqrt(N)的倍数；

既筛掉所有素数的倍数，直到a中仅剩下素数为止（剩下的不为0的数不是任何数的倍数）

```
void SiftPrime(int a[], int n)
{
  int i, j;
  for (i=2; i<=n; i++)
  {
       a[i] = i;
  }
  for (i=2; i<=sqrt(n); i++)
  {
     for (j=i+1; j<=n; j++)
     {
       if (a[i] != 0 && a[j] != 0 && a[j] % a[i] == 0)
       {
           a[j] = 0;
       }
     }
  }
}
```

```
void PrintPrime(int a[], int n)
{
  int i;
  for (i=2; i<=n; i++)
  {
     if (a[i] != 0)
     {
        printf("%d\t", a[i]);
     }
   }
   printf("\n");
}
```

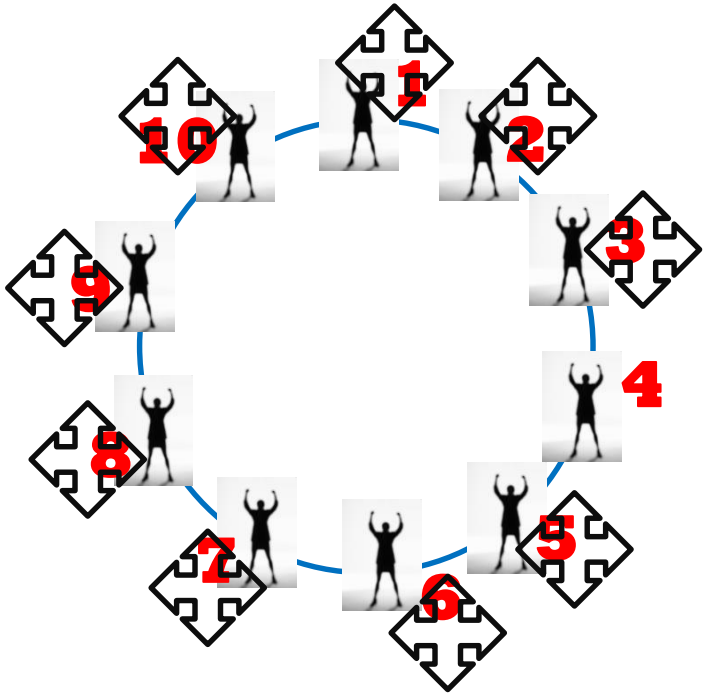
```
#include<stdio.h>
#include<math.h>
#define N  100
void PrintPrime(int a[], int n);
void SiftPrime(int a[], int n);
int main()
{
   int a[N+1];
   SiftPrime(a, N);
   PrintPrime(a, N);
   return 0;
}
```

筛法的推广应用

- **约瑟夫问题：Live or die, it’s a problem!**
  - 据说著名犹太历史学家 Josephus有过以下的故事：在罗马人占领乔塔帕特后，39个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式：
  - 41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。
  - Josephus 和他的朋友不想死，问其站在什么位置可以逃过这场死亡游戏。
- **华为2016校园招聘上机笔试题：**
  - 有一个整型数组a[n]顺序存放0 ~ n-1，要求每隔两个数删掉一个数，到末尾时循环至开头继续进行，求最后一个被删掉的数的原始下标位置。以 8 个数(n=8)为例：{0，1，2，3，4，5，6，7}，0->1->2(删除)->3->4->5(删除)->6->7->0(删除)，如此循环直到最后一个数被删除。

课后思考题：循环报数问题

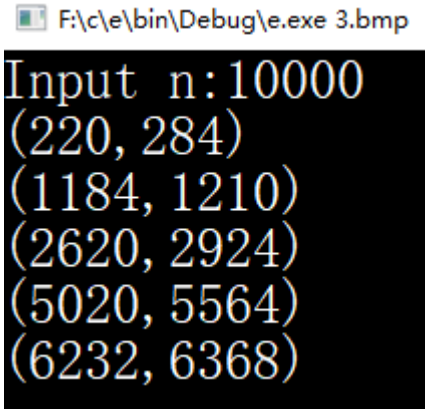
- 循环报数问题：有n个人围成一圈，顺序编号。从第一个人开始从1到m报数，凡报到m的人退出圈子，问最后留下的那个人的初始编号是什么？



```
//函数功能：用数组求解循环报数问题
//函数参数：n为参与报数的总人数，m表示每隔m人有一人退出圈子
//函数返回值：返回剩下的最后一个人的编号
int Countoff(int n, int m)
{
    int i, c = 0, quit = 0, a[N];
    //按从1到n的顺序给每个人编号并记录在数组a中（a[0]不用）
    do{
        for (i=1; i<=n; i++)
        {
            if (a[i] != 0) //尚未退出的人才报数
            {
                //用计数器c记录已报数的人数
                //若第m个报数的人应退出，则标记为退出，并用quit记录退出的人数
                }
            }
        }
    }while (quit != n-1);  //退出圈子的人数达到n-1人时不再继续报数
    for (i=1; i<=n; i++)
    {
        if (a[i] != 0) return i;
    }
    return 0;
}
```

速度比拼——求亲密数

- 2500年前数学大师毕达哥拉斯就发现，220与284两数之间存在着奇妙的联系：
  - 220的真因数之和为：1+2+4+5+10+11+20+22+44+55+110=284
  - 284的真因数之和为：1+2+4+71+142=220
- 毕达哥拉斯把这样的数对称为相亲数，也称亲密数。
  - 如果整数A的全部因子（包括1，不包括A本身）之和等于B，且整数B的全部因子（包括1，不包括B本身）之和等于A，则将整数A和B称为亲密数。
- 从键盘任意输入一个整数n，编程计算并输出n以内的全部亲密数。



速度比拼——求亲密数

```
//函数功能：返回x的所有因子之和
int FactorSum(int x)
{
    int i;
    int sum = 0;
    for (i=1; i<x; i++)
    {
        if (x%i == 0)
        {
            sum = sum + i;
        }
    }
    return sum;
}
```

```
int main()
{
  int  n, i, j;
  printf("Input n:");
  scanf("%d", &n);
  for (i=1; i<n; i++)
  {
    for (j=i+1; j<n; j++)
    {
       if (FactorSum(i)==j && FactorSum(j)==i)
       {
          printf("(%d,%d)\n", i, j);
       }
    }
  }
  return 0;
}
```

220的真因数之和为：1+2+4+5+10+11+20+22+44+55+110=284

284的真因数之和为：1+2+4+71+142=220

影响效率的瓶颈在哪里？

改进因子之和的计算

挑战速度：输出n（<1000000）以内的所有完数？

```
int IsPerfect(int x)
{
    int i;
    int sum = 1;
    int k = (int)sqrt(x);
    for (i=2; i<=k; i++)
    {
        if (x%i == 0)
        {
            sum += i;
            sum += x/i;
        }
    }
    return sum==x ? 1 : 0;
}
```

220的真因数之和为：1+2+4+5+10+11+20+22+44+55+110=284

284的真因数之和为：1+2+4+71+142=220

```
// 函数功能：返回x的所有真因子之和
int FactorSum(int x)
{
    int i;
    int sum = 1;
    int k = (int)sqrt(x);
    for (i=2; i<=k; i++)
    {
        if (x%i == 0)
        {
            sum += i;
            sum += x/i;
        }
    }
    return sum;
}
```

用数组一次性求出所有数的真因子之和

```
void FactorSum(int a[], int n)
{
    int i, j, k;
    for (i=0; i<n; i++)
    {
        a[i] = 1;
        k = (int)sqrt(i);
        for (j=2; j<=k; j++)
        {
            if (i%j == 0)
            {
                a[i] += j;
                a[i] += i/j;
            }
        }
    }
}
```

220的真因数之和为：1+2+4+5+10+11+20+22+44+55+110=284

284的真因数之和为：1+2+4+71+142=220

```
#include <stdio.h>
#include <math.h>
#define N 30000
void FactorSum(int a[],int n);
int main()
{
    int a[N];
    int i, j, n;
    scanf("%d", &n);
    FactorSum(a, n);
    for (i=0; i<n; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (a[i]==j && a[j]==i)
            {
                printf("(%d,%d)\n", i, j);
            }
        }
    }
    return 0;
}
```

优化主函数——减少循环次数

```
int main()
{
  int  n, i, j;
  printf("Input n:");
  scanf("%d", &n);
  for (i=1; i<n; i++)
  {
     for (j=i+1; j<n; j++)
     {
        if (FactorSum(i)==j &&
            FactorSum(j)==i)
        {
          printf("(%d,%d)\n", i, j);
        }
     }
   }
   return 0;
}
```

220的真因数之和为：1+2+4+5+10+11+20+22+44+55+110=284

284的真因数之和为：1+2+4+71+142=220

```
int main()
{
    int  n, k, i, j;
    printf("Input n:");
    scanf("%d", &n);
    for (i=1; i<n; i++)
    {
        j = FactorSum(i); //计算i的所有因子之和
        k = FactorSum(j); //计算m的所有因子之和
        if (i==k && i<j)  //若m和i是亲密数
        {
            printf("(%d,%d)\n", i, j);
        }
    }
    return 0;
}
```

优化主函数——减少循环次数

```
int main()
{
  int  n, i, j;
  printf("Input n:");
  scanf("%d", &n);
  for (i=1; i<n; i++)
  {
     for (j=i+1; j<n; j++)
     {
        if (FactorSum(i)==j &&
            FactorSum(j)==i)
        {
          printf("(%d,%d)\n", i, j);
        }
     }
   }
   return 0;
}
```

220的真因数之和为：1+2+4+5+10+11+20+22+44+55+110=284

284的真因数之和为：1+2+4+71+142=220

```
int main()
{
    int  n, k, i, j;
    printf("Input n:");
    scanf("%d", &n);
    for (i=1; i<n; i++)
    {
        j = FactorSum(i);
        if (i < j)
        {
            k = FactorSum(j);
            if (i == k)  //若m和i是亲密数
            {
               printf("(%d,%d)\n", i, j);
            }
        }
    }
    return 0;
}
```

同时优化

```
void FactorSum(int a[], int n)
{
    int i, j, k;
    for (i=0; i<n; i++)
    {
        a[i] = 1;
        k = (int)sqrt(i);
        for (j=2; j<=k; j++)
        {
            if (i%j == 0)
            {
                a[i] += j;
                a[i] += i/j;
            }
        }
    }
}
```

220的真因数之和为：1+2+4+5+10+11+20+22+44+55+110=284

284的真因数之和为：1+2+4+71+142=220

```
int main()
{
    int  n, k, i, j, a[N];
    printf("Input n:");
    scanf("%d", &n);
    FactorSum(a, n);
    for (i=1; i<n; i++)
    {
        j = a[i]; //计算i的所有因子之和
        k = a[j]; //计算m的所有因子之和
        if (i==k && i<j)  //若m和i是亲密数
        {
            printf("(%d,%d)\n", i, j);
        }
    }
    return 0;
}
```