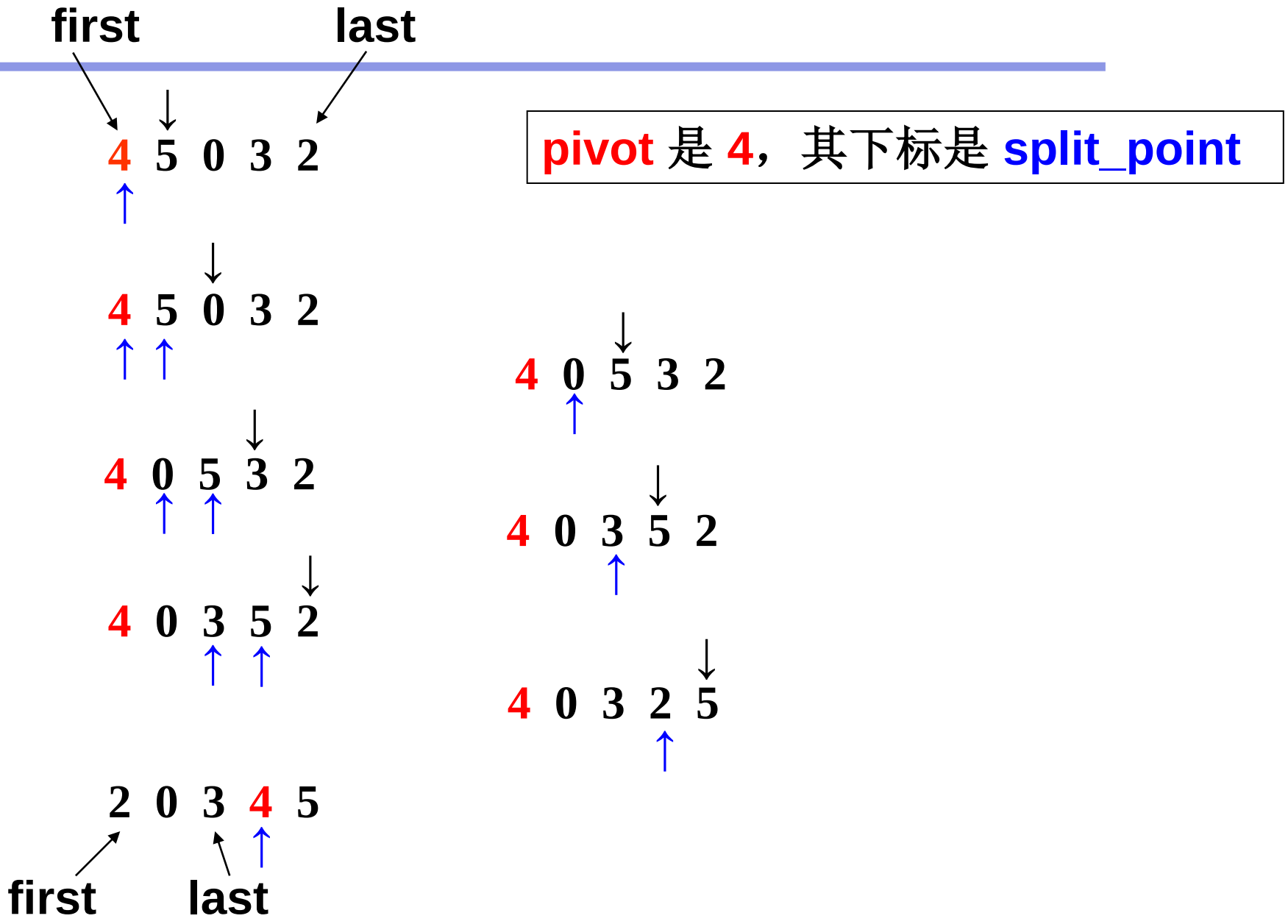
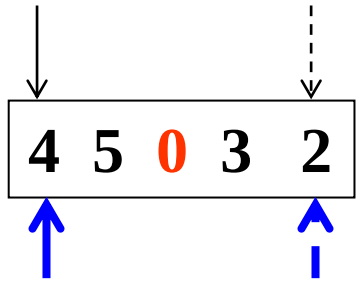
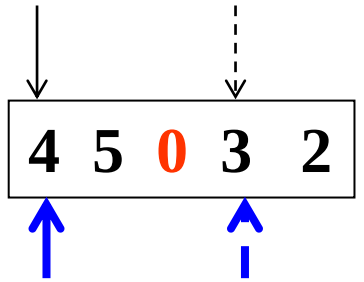
- **关于数组的长度**
  - C++ 标准规定 必须为常量
  - C标准（新版）允许为变量
- **关于初始化**
  - 指的是定义的时候“赋值”，不是赋值操作，是用赋值操作的形式**指定**初始值
    - `int a[5] = {0};`
  - 赋值不是初始化，但程序员们常常不严谨地称第一次赋值为“初始化”
    - `for(…) a[i] = i;`
- **关于快速排序**

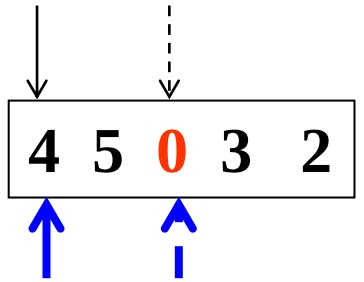


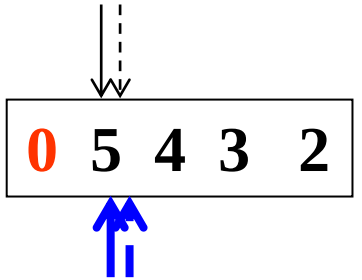
pivot 是 4，其下标是 split_point

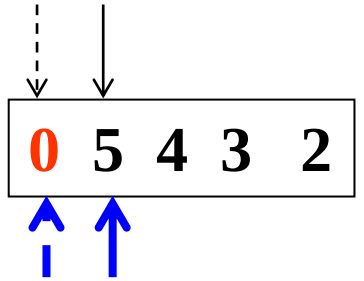
令中间那个元素为 pivot，左右分别与之比较

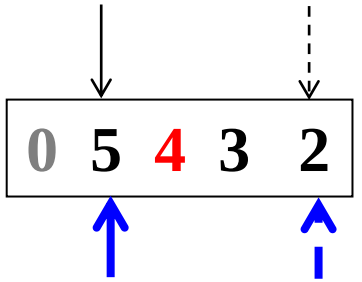


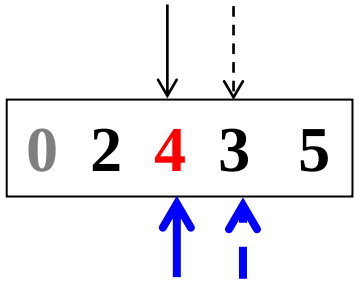

4 5 0 3 2

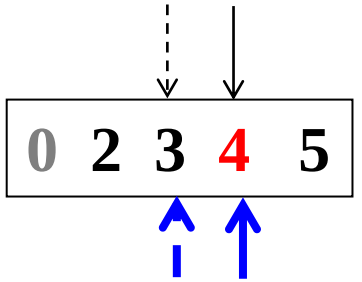

4 5 0 3 2


0 5 4 3 2


0 5 4 3 2

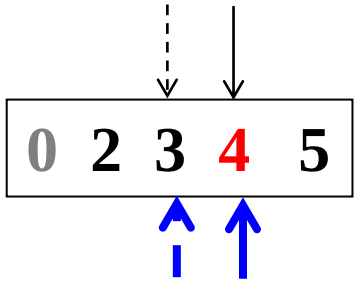

0 5 4 3 2


0 2 4 3 5




```
int i = first, j = last, pivot = a[(last + first)/2];
while(i <= j)
{     while(a[i] < pivot) ++i;
      while(a[j] > pivot) --j;
      if(i <= j)
            swap(a[i++], a[j--]);
}
return i-1;
```

```
void QuickSort(int sdata[ ], int first, int last)
{     if(first < last)
      {     int split_point = Split(sdata, first, last);
            QuickSort(sdata, first, split_point);
            QuickSort(sdata, split_point+1, last);
      }
}
```




```
int i = first, j = last, pivot = a[(last + first)/2];
while(i <= j)
{     while(a[i] < pivot) ++i;
      while(a[j] > pivot) --j;
      if(i <= j)
            swap(a[i++], a[j--]);
}
return i;
```

```
void QuickSort(int sdata[ ], int first, int last)
{     if(first < last)
      {     int split_point = Split(sdata, first, last);
            QuickSort(sdata, first, split_point-1);
            QuickSort(sdata, split_point, last);
      }
}
```

作业7

- **Ex1.编程求一个一维int型数组的第k大的数。**

```
int a[N] = {2, 6, 1, 4, 8};
cout << kth_element(a, N, 3);
```

```
int kth_element(int a[], int n, int k)
{
    for(int i = n; i > 1; --i)
    {
        int min = 0;
        for(int j = 1; j < i; ++j)
            if(a[min] >= a[j])
                min = j;
        if(min != i-1)
        {
            int temp = a[min];
            a[min] = a[i-1];
            a[i-1] = temp;
        }
    }
    return a[k-1];
}//从大到小排序
```

若从小到大排序：a[N – k]

```
for(int j = 0; j < k; ++j)
{
      int max = 0;
      for(int i = 0; i < n-1-j; ++i)
      {     if(x[i+1] >= x[i])
                  max = i+1;
      }
      if(max != n-1-j)
      {
            int temp = x[n-1-j];
            x[n-1-j] = x[max];
            x[max] = temp;
      }
}
return x[N-k];
```

提前结束排序

```
for(int j=0; j < N; ++j)
{
        t=0;
        for(int m=0; m < N; ++m)
        {
                if(a[j] >= a[m])
                        ++t;
        } //a[j]能超过几个数
        if(t == N-k+1)
                return a[j];
}
```

不排序

- Ex2. 假设一维int型数组中的N个元素按从小到大的顺序排列，编写函数，“删除”其中的重复元素，返回重复元素个数n，并在主调函数中输出前N-n个元素。例如，对于一维数组{1, 1, 2, 3, 3, 3, 4, 5, 6, 6}，函数执行后，数组为：{1, 2, 3, 4, 5, 6, 6, 6, 6, 6}，返回值是4，输出1, 2, 3, 4, 5, 6。

```
int a[N] = {2, 3, 1, 4, 8, 6, 7, 9, 0, 5};
int count = delDuplicate(a, N);
for(int i=0; i < N-count; ++i)
      cout << a[i] << " ";
```

```
int delDuplicate(int a[], int n)
{     int count = 0;
      for(int i=0; i<n-1;  )
      {     if(a[i] == a[i+1])
            {     for(int j=i+1; j < n; ++j)
                        a[j-1] = a[j];
                  --n,++count;
            }
            else ++i;
      }
      return count;
}
```

- Ex3．假设有一个数组x[]，它有n个元素，每一个都大于0；称x[0]+x[1]+…+x[i]为前置和，而x[j]+x[j+1]+…+x[n-1]为后置和。试编写一个程序，求出x[]中有多少组相同的前置和与后置和。例如，如果x[]的元素为3、6、2、1、4、5、2，则x[]的前置和有：3、9、11、12、16、21、23；后置和有：2、7、1、12、14、20、23；11、12、23这3对就是值相同的前置和与后置和，因为：11=3+6+2（前置和）=2+5+4（后置和），12=3+6+2+1（前置和）=2+5+4+1（后置和），23是整个数组元素的和。

```
int  head_tail(int  x[], int n)
{
     int  count = 0;
     int  prefix = 0, suffix = 0;
     int  prefix_idx = 0, suffix_idx = n-1;  //下标
     while(suffix_idx >= 0 && prefix_idx <= n-1)
          if(prefix < suffix)       // prefix too small
               prefix += x[prefix_idx++];
          else if(prefix > suffix) // suffix too small
               suffix += x[suffix_idx--];
          else
          {                         // get an equal pair:
               ++count;             // increase count and
               prefix += x[prefix_idx++];     // advance pref
               suffix += x[suffix_idx--];     // and suffix
          }
     return count;  //起初两个0多算一次 抵消最后少算一次
}
```

- **Ex4**. 编程实现用筛法求素数（首先假设所有的数从小到大排列在筛子中，然后，从最小的数开始依次将每个数的倍数从筛子中筛掉，最终筛子中剩下的数均为素数）。

- 筛法求20之前的素数

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

2 3 5 7 11 13 17 19

```
#define MAX 21
bool s[MAX];

for(int i=2; i < MAX; ++i)
      s[i] = true;              // 初始化，所有数都在筛子中
for(int i=2; i<MAX; ++i)
      if(s[i])                  // 值为true的元素下标为素数
      {
            cout << i << '\t'; //打印素数i
            for(int m = i+1; m < MAX; ++m)
                  if(m%i == 0)
                        s[m] = false;
                        // 从筛子中筛去当前素数的倍数
      }
```

```
#define MAX 21
bool s[MAX];

for(int i=2; i < MAX; ++i)
      s[i] = true;              // 初始化，所有数都在筛子中
for(int i=2; i<MAX; ++i)
      if(s[i])                  // 值为true的元素下标为素数
      {
            cout << i << '\t'; //打印素数i
            for(int m = 2*i; m < MAX; m += i)
                  if(s[m])
                        s[m] = false;
                        // 从筛子中筛去当前素数的倍数
      }
```

第11周自主训练任务

1．验证一维数组和二维数组的定义、初始化与输出方法。

2．编程分别用数组和递归函数实现：输入一个正整数，输出其各位数字。例如，输入89532，输出8，9，5，3，2。

```
void EchoPrint(int n)
{
    if( n >= 10 )
        EchoPrint( n/10 );
    printf("%d, ", n%10);
}
```

```
int main( )
{
      int n;
      cin >> n;
      int a[10];
      int count = NumDigit(n);
      for(int i = count-1; i >= 0; --i)
      {    a[i] = n%10;
           n /= 10;
      }
      for(int i=0; i < count; ++i)
       cout << a[i] << ", ";
      return 0;
}
```

```
int NumDigit(int n)
{
      int count = 0;
      while(n != 0)
      {     n = n/10;
            ++count;
      }
      return count;
}
```

3. 分析程序执行结果，并上机验证（注意下标变化及赋值操作的作用）。

程序如下:

```
int b[][3] = {0, 2, 1, 1, 0, 2, 1, 2, 0};
for(int i=0; i <= 2; ++i)
    for(int j=0; j <= 2; ++j)
    {
        b[i][j] = b[b[i][j]][b[j][i]];
        printf("%d, ", b[i][j]);
    }
```

0	2	1
1	0	2
1	2	0

0	2	0
2	0	0
2	2	0

b[0][0]	b[2][1]	b[1][1]
b[1][2?]	b[0][0]?	b[2][2]
b[1][1?]	b[2][2?]	b[0][0]

b[1][0]　　b[2][0]

4. 编程实现从键盘输入一个N×N的矩阵，把它转置后输出。（说明：对矩阵进行转置就是交换二维数组中a[i][j]与a[j][i]的值。）

```
for(int i=0; i<N; ++i)
    for(int j=i+1; j<N; ++j)
    {
        int temp=a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = temp;
    }
```

```
a[i][j]^=a[j][i];
a[j][i]^=a[i][j];
a[i][j]^=a[j][i];
```

```
i == j ?
```

转置输出？

用另一个数组?

5. 调试、运行课件中的排序程序，并修改程序，实现数组的降序排列。

```
void BubbleSort(int sdata[], int count)
{
     for(int i = 0; i < count-1; ++i)
          for(int j = 0; j < count-1-i; ++j)
               if(sdata[j] < sdata[j+1])
               {
                    int temp = sdata[j];
                    sdata[j] = sdata[j+1];
                    sdata[j+1] = temp;
               }
}
```

- 若课件中选择法排序程序中外循环为“for(int i = 1; i < count; ++i)”，程序中的其他代码应做哪些相应修改，可以实现原功能？

```
void SelSort(int sdata[], int count)
{                        1          count  ++
    for(int i = count; i > 1; --i)//控制外循环次数
    {
        int max = 0;
        for(int j = 1; j < i; ++j) //count-i+1
            if(sdata[max] < sdata[j])
                max = j;
        if(max != i-1) //count-i
        {
            int temp = sdata[max];
            sdata[max] = sdata[i-1]; //count-i
            sdata[i-1] = temp; //count-i
        }
    }
}
```

6. 编程用一维数组实现求第n个费波那契数。

```
int myFib(int f[], int n)
{
      f[0] = f[1] = 1;

      for(int i=2; i < n; ++i)
            f[i] = f[i-1] + f[i-2];

      return f[n-1];
}
```

```
#define N 10

int fibs[N];
cout << myFib(fibs, N);
```

7. 不引入新的数组，实现数组前m个元素与后n个元素交换位置的函数，其中m+n等于数组的长度，例如，设m为3，n为4，a中的数据为1 2 3 4 5 6 7，函数执行后，a中的数据为4 5 6 7 1 2 3。函数原型为：

void ArrSwap(int a[], int m, int n);

```
void swapArray(int a[], int m, int n)
{
      for(int i = 0; i < m; ++i)
      {
            int j, tmp = a[0];
            for(j = 0; j < m+n-1; ++j)
                  a[j] = a[j + 1];
            a[j] = tmp;
      }
}
```

依次做轮转

做三次整体翻转也可以

交换输出？

```
int main()
{
    const int N = 7;
    const int M = 1;

    int a[N] = {1,2,3,4,5,6,7};
    …
    swapArray(a, M, N-M);

    for(int i=0; i<N; ++i)
          cout << a[i] << ",";
    cout << endl;

    return 0;
}
```

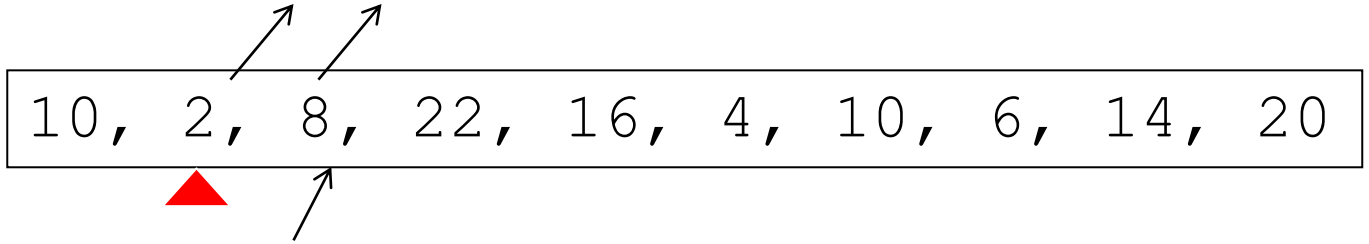
8. 十个小孩围成一圈分糖果，每个小孩分得的糖果数依次为：10、2、8、22、16、4、10、6、14、20。然后所有的小孩同时将自己手中的糖果分一半给右边的小孩；此时块数为奇数的人可向老师要一块糖果。问需要几次这样的调整后，小孩手中糖果的块数就都一样了？每人有多少块糖果？编程求解上述问题。

```
int i, count=0, a[10] = {10,2,8,22,16,4,10,6,14,20};
while(1)
{     …                              //分糖
      …                              //调整

      ++count;                       //计数

      for(i=0; i<9; ++i)   //判断
            if(a[i] != a[i+1])
                  break;
      if(i == 9)                     //全部相等，则结束   （事件型循环）
            break;
}
cout << "count=" << count << ", number=" << a[0] << endl;
```

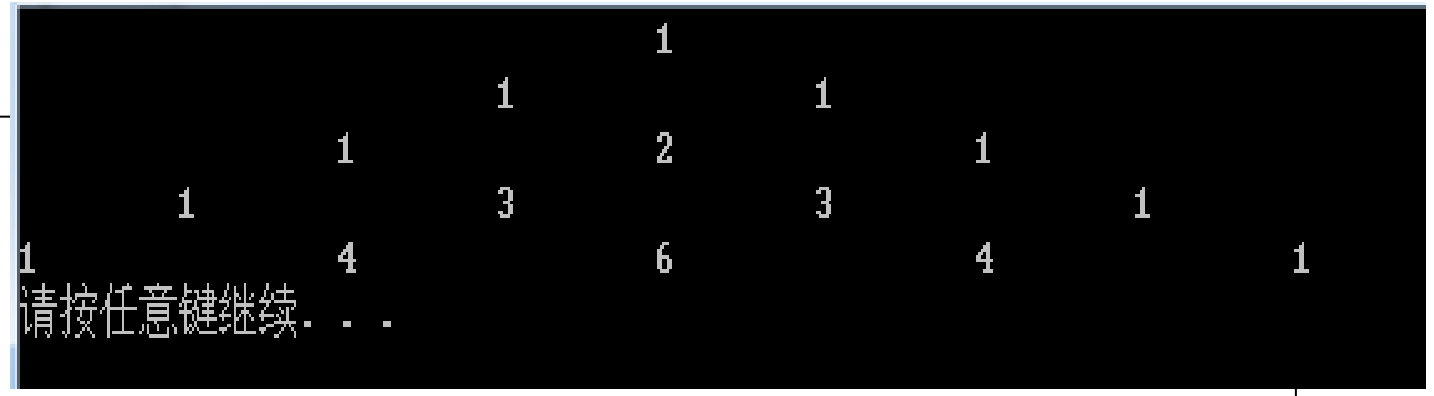
另开一个数组b，存储原数组a的一半 ^_^




```
int temp1, temp2;
while(1)
{
      temp2 = a[0];
      for(i=0; i < 9; ++i)
      {
            temp1 = temp2;
            temp2 = a[i+1];
            a[i+1] = temp1/2 + temp2/2; //右边 = 自己/2 + 右边/2
      }
      a[0] = a[0]/2 + temp2/2;
                                        //分糖

      for(i=0; i < 10; ++i)             //调整，即处理奇数
            if(a[i]%2 == 1)
                  ++a[i];
      …
}
```

9. 利用数组编程实现杨辉三角显示功能。




```
#define N 5
int i, j, a[N][2*N] = {0};

for( i = 0; i < N; ++i )
{
     a[i][N-1 - i] = 1;
     a[i][N-1 + i] = 1;
}

for( i = 1; i < N; ++i)
     for( j = 1; j < 2*N-1; ++j )
          a[i][j] = a[i -1][j - 1] + a[i - 1][j + 1];
```

```
for( i = 0; i < N; ++i)
{
     for( j = 0; j < 2*N-1; ++j )
          if(a[i][j])
               cout << a[i][j] << '\t';
          else
               cout << '\t';
     cout << endl;
}
```

10. 编程输出一个N阶（N为大于1的奇数）幻方。即在一个由N×N个方格组成的方阵中，填入1、2、3、...、N2各个数，使得每一行、每一列及两个对角线上的数之和均相等。例如，一个3阶幻方：

三阶幻方

8　　1　　6

3　　5　　7

4　　9　　2

（填数方法提示：把1填在第一行最中间的格子中，然后依次将下一个数填在上一个数的右上角格子中。如果目标格子在第一行的上方，则填入该列的最后一行格子中；如果目标格子在最后一列的右方，则填入该行的第一列格子中；如果目标格子已经被占用，则填入当前格子的下方格子中。）

```
const int M=15;
```

```
void magicsm(int a[][M], int n)
{
     int i, j;

     for(i = 0; i < n; ++i)              //数组a置0
          for(j = 0; j < n; ++j)
               a[i][j] = 0;

     i = 1;                              //安排1
     j = (n+1) / 2;
     a[i-1][j-1] = 1;                    //注意目标位置下标为i-1、j-1

     for(int k = 2; k <= n * n; ++k)     //安排 2, 3, ..., n*n
     {…}
}
```

```
for(int k = 2; k <= n * n; ++k)      //安排 2, 3, ..., n*n
{
//先计算下标
      if(i == 1 && j == n)           //前数占右上角格子,
            i = 2;                                 //放其下方
      else                                         //放前数右上方
      {     i = i - 1 + n ;                        //防止下标为负数
            i = i > n ? i - n : i;     //防止越界
            j = j % n + 1;                         //防止越界
      }
      if(a[i-1][j-1] != 0)           //目标格子被占用
      {     i += 2;                                //放其下方,
            j -= 1;                                //即从右上方调到下方
      }
//再安排数
      a[i-1][j-1] = k;
}
```

或

```
void magicsm(int a[][M], int n)
{
     int i, j;

     for(i = 0; i < n; ++i)              //数组a置0
          for(j = 0; j < n; ++j)
               a[i][j] = 0;

     i = 0;                              //安排1
     j = M / 2;
     a[i][j] = 1;
```

```
for(int k = 2; k <= n * n; ++k)      //安排2, 3, ..., n*n
{
      --i, ++j;
      if(i == -1 && j == n)          //右上角
            i = 1, j-=1;
      else if(i == -1)               //上方
            i = n-1;
      else if(j == n)                //右方
            j = 0;

      if(a[i][j] != 0)               //目标格子被占用
            i += 2, --j;

      a[i][j] = k;
}
```