



C 程序中的数组

北京大学 信息科学技术学院

数组的定义

■ 数组的定义

类型

float

int

数组名 [常量表达式]

sheep[10];

a2001[1000];

■ 再次强调：数组下标从0开始

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

1	2	3	4	5	6	7	8	9	10
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

关于数组的定义 (1/3)

```
#include<iostream>
using namespace std;
int main()
{
    int n = 10;
    int a[n] = { 0 };
    for (int i = 0; i < 10; i++)
        cout << a[i];
    return 0;
}
```

**此处应为
常量表达式**

关于数组的定义 (2/3)

```
#include <iostream>
using namespace std;
int main()
{
    const int i = 4;
    int a[i] = { 1, 2, 3, 4 };
    cout << "a[0]=" << a[0] <<endl
         << "a[1]=" << a[1] <<endl
         << "a[2]=" << a[2] <<endl
         << "a[3]=" << a[3] <<endl;
    return 0;
}
```

关于数组的定义 (3/3)

```
#include <iostream>
using namespace std;
#define N 4
int main()
{
    int a[N] = { 1, 2, 3, 4 };
    cout << "a[0]=" << a[0] << endl
         << "a[1]=" << a[1] << endl
         << "a[2]=" << a[2] << endl
         << "a[3]=" << a[3] << endl;
    return 0;
}
```

数组的初始化

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

1	2	3	4	5	6	7	8	9	10
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

数组的初始化 (1/6)

```
#include <iostream>
using namespace std;
int main()
{
    int a[4];
    cout << a[0] << a[1] << a[2] << a[3] << endl;
    return 0;
}
```

-858993460	-858993460	-858993460	-858993460
a[0]	a[1]	a[2]	a[3]

数组的初始化 (2/6)

```
#include <iostream>
using namespace std;
int main()
{
    int a[ ] = { 1, 2, 3, 4 };
    cout << a[0] << a[1] << a[2] << a[3] << endl;
    return 0;
}
```

1	2	3	4
a[0]	a[1]	a[2]	a[3]

数组的初始化 (3/6)

```
#include <iostream>
using namespace std;
int main()
{
    int a[4] = { 1, 2 };
    cout << a[0] << a[1] << a[2] << a[3] << endl;
    return 0;
}
```

1	2	0	0
a[0]	a[1]	a[2]	a[3]

数组的初始化 (4/6)

```
#include <iostream>
using namespace std;
int main()
{
    int a[4] = { 0 };
    cout << a[0] << a[1] << a[2] << a[3] << endl;
    return 0;
}
```

0	0	0	0
a[0]	a[1]	a[2]	a[3]

数组的初始化 (5/6)

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

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

```
    cout << a[0] << a[1] << a[2] << a[3] << endl;
```

```
    return 0;
```

```
}
```

数组的初始化 (6/6)

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int a[4] = { 1, 2, 3, d };
```

```
    cout << a[0] << a[1] << a[2] << a[3] << endl;
```

```
    return 0;
```

```
}
```



从一维数组 到 二维数组

int a[12] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

1	2	3	4	5	6	7	8	9	10	11	12
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]

二维数组

	0列	1列	2列	3列
0行	1 a[0][0]	2 a[0][1]	3 a[0][2]	4 a[0][3]
1行	5 a[1][0]	6 a[1][1]	7 a[1][2]	8 a[1][3]
2行	9 a[2][0]	10 a[2][1]	11 a[2][2]	12 a[2][3]

```
int a[3][4];
```

二维数组的初始化 (1/6)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
```

```
{
```

```
    int a[3][4] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };
```

```
    for (int i = 0; i < 3; i++)
```

```
    {
```

```
        for (int j = 0; j < 4; j++)
```

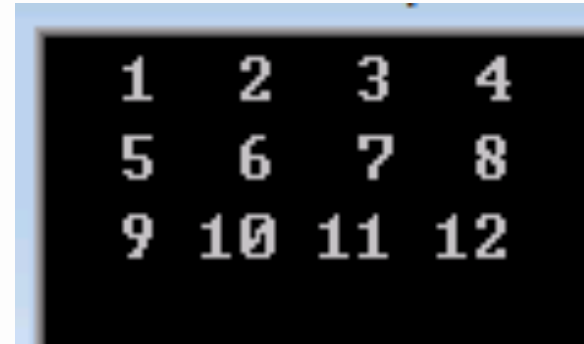
```
            cout << setw(3) << a[i][j];
```

```
        cout << endl;
```

```
    }
```

```
    return 0;
```

```
}
```



1	2	3	4
5	6	7	8
9	10	11	12

二维数组的初始化 (2/6)

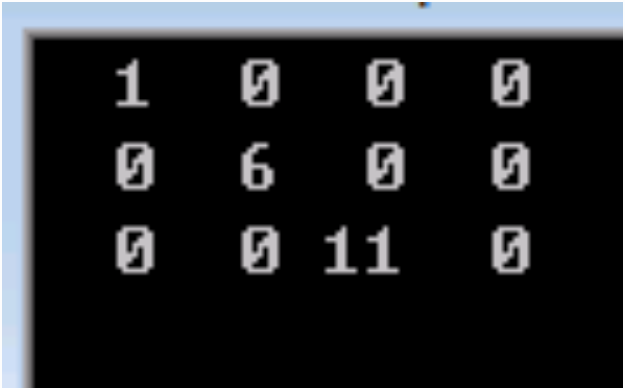
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
            cout << setw(3) << a[i][j];
        cout << endl;
    }
    return 0;
}
```


二维数组的初始化 (3/6)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a[ ][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
            cout << setw(3) << a[i][j];
        cout << endl;
    }
    return 0;
}
```

二维数组的初始化 (4/6)

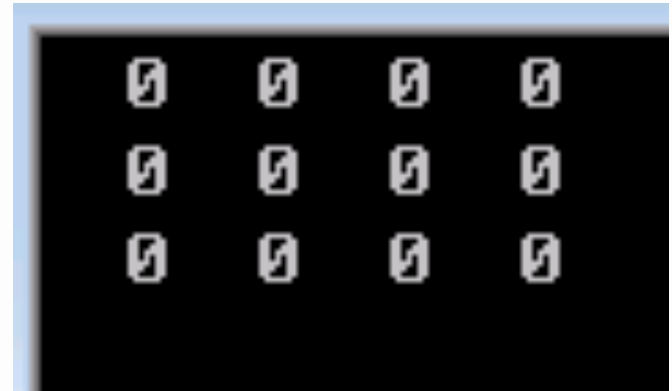
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a[3][4] = {{1}, {0, 6}, {0, 0, 11}};
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
            cout << setw(3) << a[i][j];
        cout << endl;
    }
    return 0;
}
```



1	0	0	0
0	6	0	0
0	0	11	0

二维数组的初始化 (5/6)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a[3][4] = {0};
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
            cout << setw(3) << a[i][j];
        cout << endl;
    }
    return 0;
}
```



二维数组的初始化 (6/6)

```
int main()
```

```
{
```

```
    int a[3][4] = { 0 };
```

```
    for (int i = 0; i < 3; i++)
```

```
        for (int j = 0; j < 4; j++)
```

```
            a[i][j] = 4 * i + j + 1;
```

```
    for (int i = 0; i < 3; i++)
```

```
    {
```

```
        for (int j = 0; j < 4; j++)
```

```
            cout << setw(3) << a[i][j];
```

```
        cout << endl;
```

```
    }
```

```
    return 0;
```

```
}
```

	0列	1列	2列	3列
0行	1 a[0][0]	2 a[0][1]	3 a[0][2]	4 a[0][3]
1行	5 a[1][0]	6 a[1][1]	7 a[1][2]	8 a[1][3]
2行	9 a[2][0]	10 a[2][1]	11 a[2][2]	12 a[2][3]

二维数组

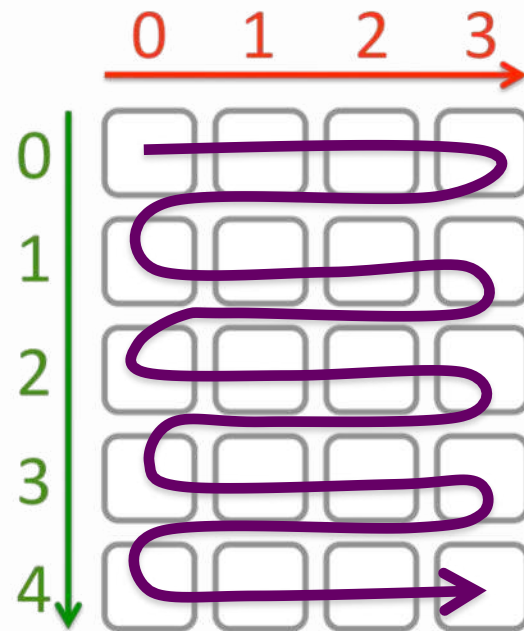
```
int a[3][4];
```

	0列	1列	2列	3列
0行	1 a[0][0]	2 a[0][1]	3 a[0][2]	4 a[0][3]
1行	5 a[1][0]	6 a[1][1]	7 a[1][2]	8 a[1][3]
2行	9 a[2][0]	10 a[2][1]	11 a[2][2]	12 a[2][3]

...
a[0][0]
a[0][1]
a[0][2]
a[0][3]
a[1][0]
a[1][1]
a[1][2]
a[1][3]
a[2][0]
a[2][1]
a[2][2]
a[2][3]
...

正向 按行 遍历

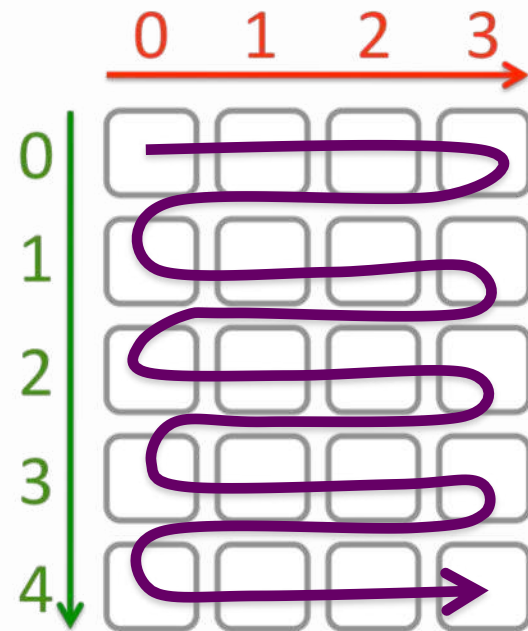
```
#define ROW 5  
#define COL 4  
int sz[ROW][COL];
```



```
for(int row = 0; row < ROW; row++){  
    for(int col = 0; col < COL; col++){  
        ... sz[row][col] ...  
    }  
}
```

正向 按行 遍历

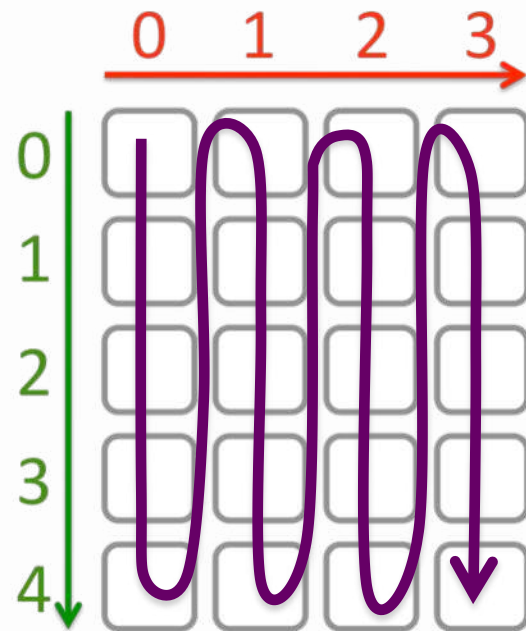
```
#define ROW 5  
#define COL 4  
int sz[ROW][COL];
```



```
for(int row = 0; row < ROW; row++){  
    for(int col = 0; col < COL; col++){  
        printf("%d ", sz[row][col]);  
    }  
}
```

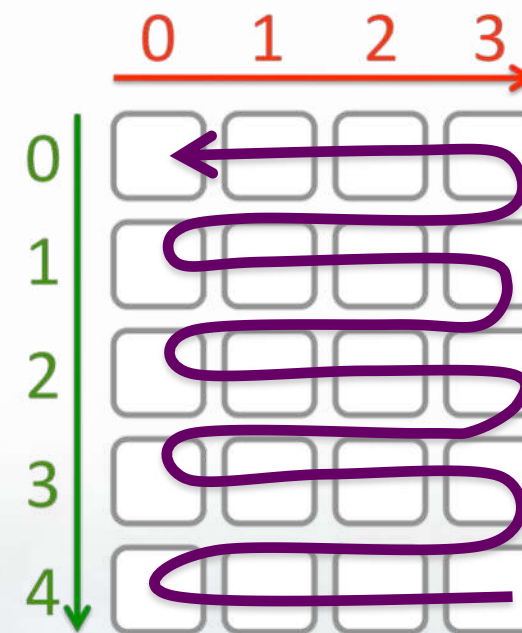
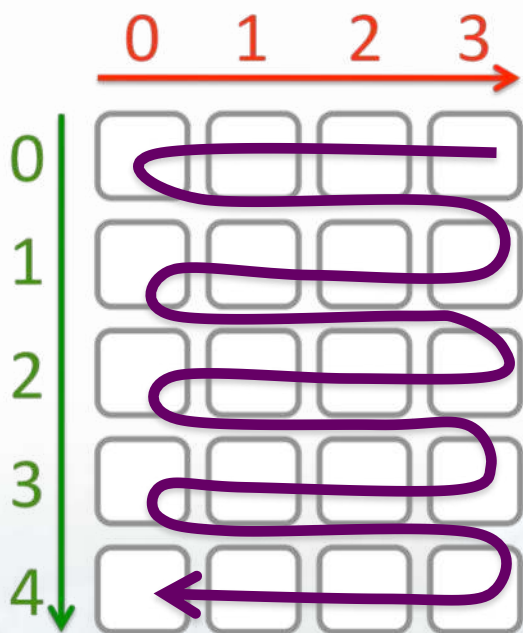
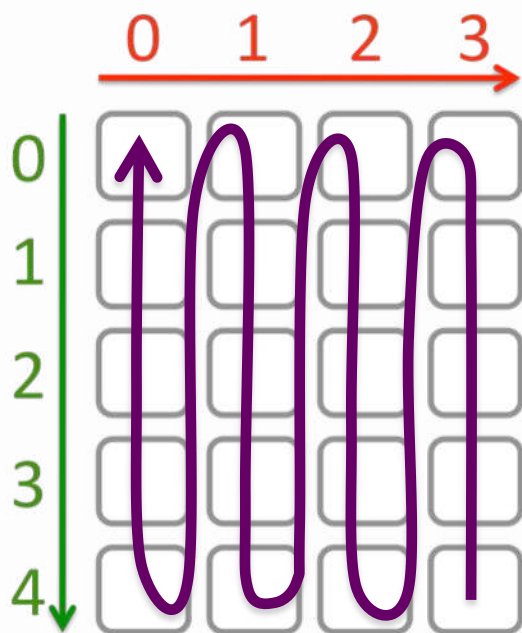
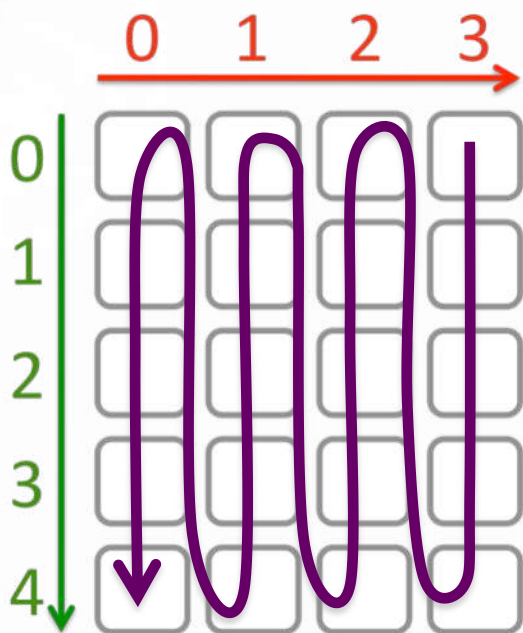
正向 按列 遍历

```
#define ROW 5  
#define COL 4  
int sz[ROW][COL];
```



```
for(int col = 0; col < COL; col++){  
    for(int row = 0; row < ROW; row++){  
        ... sz[row][col] ...  
    }  
}
```


其它遍历顺序

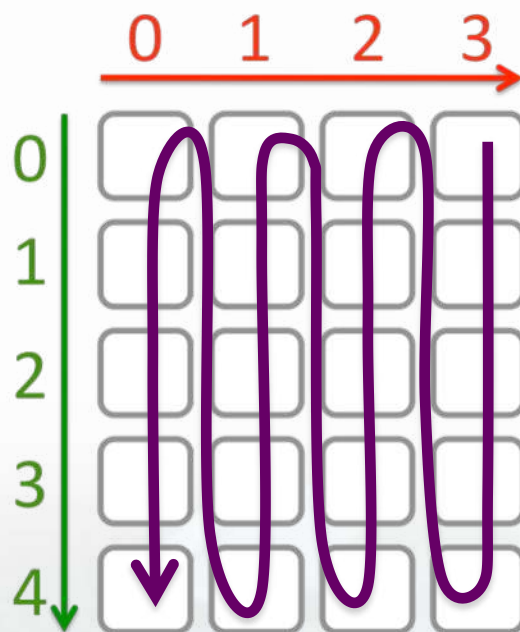


课堂参与

请一位同学上台，在黑板上写一段代码，实现下述功能：

按照下图所示的顺序，

遍历数组 `int sz[ROW][COL]`



寻找山顶

- 描述

在一个 $m \times n$ 的山地上，已知每个地块的平均高程，请求出所有山顶所在的地块（所谓山顶，就是其地块平均高程不比其上下左右相邻的四个地块每个地块的平均高程小的地方）。

- 关于输入

第一行是两个整数，表示山地的长 m （ $5 \leq m \leq 20$ ）和宽 n （ $5 \leq n \leq 20$ ）。

其后 m 行为一个 $m \times n$ 的整数矩阵，表示每个地块的平均高程。每行的整数间用一个空格分隔。

- 关于输出

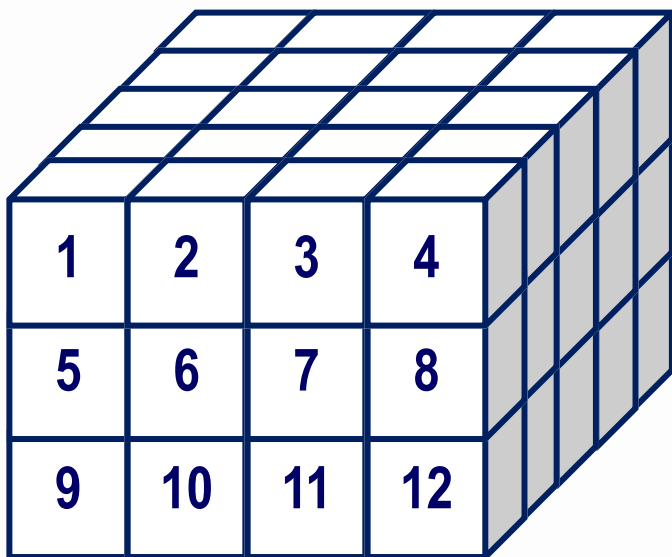
输出所有山顶所在地块的位置。每行一个。按先 m 值从小到大，再 n 值从小到大的顺序输出。

```

#include <stdio.h>
#define MAX    20                /* 山地最大长宽值 */
int main()
{
    int a[MAX+2][MAX+2] = {0};    /* 定义一个比地块大一圈的二维数组 */
    int m, n, i, j;
    scanf("%d%d", &m, &n);
    /* 数组内部元素值对应于相应地块位置的高程 */
    for (i = 1; i <= m; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    /* 遍历每一个地块位置，判断它的高程是否大于等于上下左右4个位置的值 */
    for (i = 1; i <= m; i++) {
        for (j = 1; j <= n; j++) {
            if (a[i][j] >= a[i - 1][j] && a[i][j] >= a[i + 1][j] &&
                a[i][j] >= a[i][j - 1] && a[i][j] >= a[i][j + 1]) {
                printf("%d %d\n", i - 1, j - 1); /* 找到山顶，输出 */
            }
        }
    }
    return 0;
}

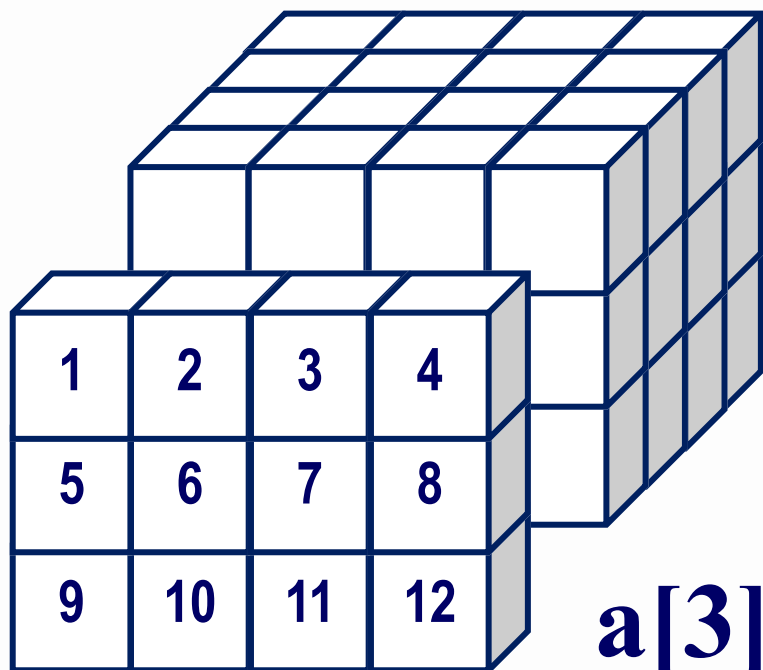
```

三维数组



```
int a[5][3][4];
```

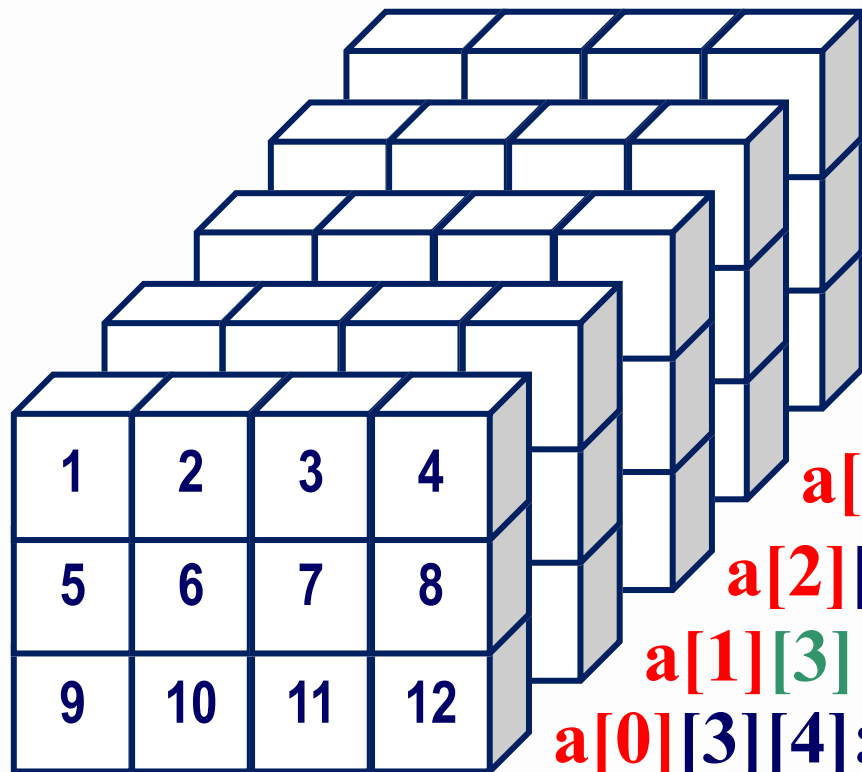
三维数组



```
int a[5][3][4];
```

```
a[3][4];
```

三维数组



$a[4][3][4];$
 $a[3][3][4];$
 $a[2][3][4];$
 $a[1][3][4];$
 $a[0][3][4];$

...
$a[0][0][0]$
$a[0][0][1]$
$a[0][0][2]$
$a[0][0][3]$
$a[0][1][0]$
$a[0][1][1]$
$a[0][1][2]$
$a[0][1][3]$
$a[0][2][0]$
$a[0][2][1]$
$a[0][2][2]$
$a[0][2][3]$
$a[0][3][0]$
$a[0][3][1]$
$a[0][3][2]$
$a[0][3][3]$
$a[1][0][0]$
$a[1][0][1]$
.....
$a[1][3][3]$
$a[2][0][0]$
$a[2][0][1]$
.....
$a[2][3][3]$
$a[3][0][0]$
.....
$a[3][3][3]$
$a[4][0][0]$
.....
$a[4][3][3]$
...

三维数组

```
int main()
```

```
{
```

```
    int a[5][3][4] = { 0 };
```

```
    for (int i = 0; i < 5; i++)
```

```
    for (int j = 0; j < 3; j++)
```

```
    for (int k = 0; k < 4; k++)
```

```
        a[i][j][k] = 12 * i + 4 * j + k + 1;
```

```
    for (int i = 0; i < 5; i++)
```

```
    {
```

```
        for (int j = 0; j < 3; j++)
```

```
        {
```

```
            {
```

```
                for (int k = 0; k < 4; k++)
```

```
                cout << setw(3) << a[i][j][k];
```

```
            }
```

```
            cout << endl;
```

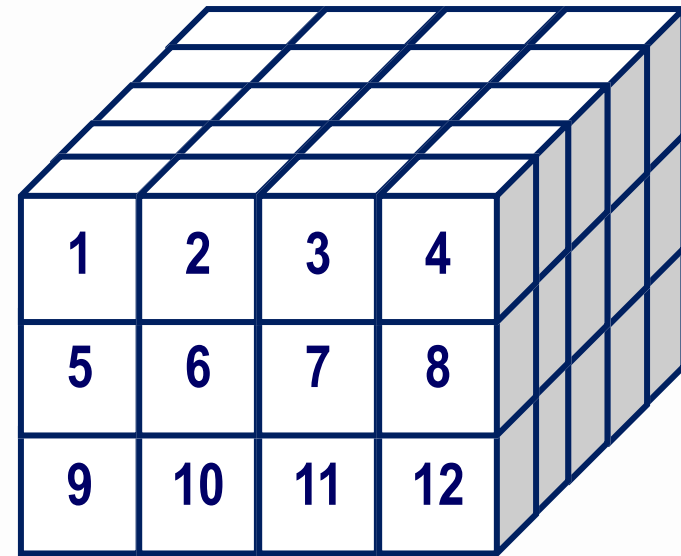
```
        }
```

```
        cout << endl;
```

```
    }
```

```
    return 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	52
53	54	55	56
57	58	59	60



数组的作用

```
int a[12] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

1	2	3	4	5	6	7	8	9	10	11	12
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]

例1 数字统计

- 输入20个0~9之间的整数，请你统计每个数在输入数列中出现的次数。

4 5 7 4 3 2 4 5 7 7 7 4 3 2 2 4 6 7 8 4

2输入了3次
3输入了2次
4输入了6次

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

例1 数字统计

```
for (i = 0; i < 20; i++)  
{  
    cin >> num;  
    for (j = 0; j < 10; j++)  
    {  
        if (num==j)  
            count[j]++;  
    }  
}
```

例1 数字统计

```
#include <iostream>
using namespace std;
int main()
{
    int num, count[10] = {0};
    for (int i = 0; i < 20; i++)
    {
        cin >> num;
        for (int j = 0; j < 10; j++)
        {
            if (num == j)    count[j]++;
        }
    }
    for (int i = 0; i < 10; i++)
    {
        if (count[i] != 0)
            cout << i << "输入了" << count[i] << "次" << endl;
    }
    return 0;
}
```

例1 数字统计

```
for (i=1;i<=1000;i++)  
{  
    cin >> num;  
    switch(num)  
    {  
        case 0 : count[0]++;break;  
        case 1 : count[1]++;break;  
        case 2 : count[2]++;break;  
        ...  
    }  
}
```

例1 数字统计

```
#include <iostream>
using namespace std;
int main()
{
    int num, count[10] = {0};
    for (int i = 1; i <= 1000; i++)
    {
        cin >> num;
        switch (num)
        {
            case 0: count[0]++; break;
            case 1: count[1]++; break;
            case 2: count[2]++; break;
            case 3: count[3]++; break;
            case 4: count[4]++; break;
            case 5: count[5]++; break;
            case 6: count[6]++; break;
            case 7: count[7]++; break;
            case 8: count[8]++; break;
            case 9: count[9]++; break;
        }
    }
    for (int i = 0; i < 10; i++)
    {
        if (count[i] != 0)
            cout << i << "输入了" << count[i] << "次" << endl;
    }
    return 0;
}
```

例1 数字统计

■ 更简单的办法：

```
for (i = 0; i < 20; i++)  
{  
    cin >> num;  
    count[num]++;  
}
```

例1 数字统计

```
#include <iostream>
using namespace std;
int main()
{
    int num, count[10] = {0};
    for (int i = 0; i < 20; i++)
    {
        cin >> num;
        count[num]++;
    }
    for (int i = 0; i < 10; i++)
    {
        if (count[i] != 0)
            cout << i << "输入了" << count[i] << "次" << endl;
    }
    return 0;
}
```




例2 数字统计

■ 问题

- ◆ 某学校有1000位老师，分布在20个不同的学院中，每个学院最多有12个系，请你编写一个程序，输入每位老师的所在院、系的编号（院编号1-20，系编号1-12），打印出各个系老师的数量。

■ 分析

- ◆ 你的解决方案？

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{ int teacher[21][13];
  int school, department;
  int i,j;
  char name[30];

  for(i=0;i<1000;i++)
  {
    cin>>name>>school>>department;
    teacher[school][department]++;
  }

  for (i=1;i<21;i++)
    for(j=1;j<13;j++)
      cout << setw(4) << teacher[i][j] ;
  cout << endl;
  return 0;
}
```

例3 找出素数

■ 问题：

◆ 请编写程序，输出100以内的所有素数；

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]	a[16]	a[17]	a[18]



例3 找出素数

■ 问题：

- ◆ 请编写程序，输出100以内的所有素数；

■ 典型的解决方法：

- ◆ 循环（ i 从 2 至 100）

- 设置一个标识 $p = 0$ ；

- 循环（ j 从 2 至 $i-1$ ）

- ◆ 如果（ $i \text{ mode } j == 0$ ） $p = 1$

- 如果 $p = 0$ ，输出 i

例1 数字统计

```
#include <iostream>
using namespace std;
int main()
{
    bool prime = true;
    for (int i = 0; i < 100; i++)
    {
        prime = true;
        for (int j = 2; j < i; j++)
        {
            if (i % j == 0)
                prime = false;
        }
        if (prime == true)
            cout << i << endl;
    }
    return 0;
}
```

例3 找出素数

■ 问题：

◆ 请编写程序，输出100以内的所有素数；

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]	a[16]	a[17]	a[18]

■ 思路：

◆ 将数组中 1的倍数、2的倍数、3的倍数、.....
100倍数全部划掉；

◆ 那么，剩下的数就是素数；



例3 找出素数

■ 解题思路：

- ◆ 将数组所有元素设置为0；
- ◆ 筛出所有合数：
 - 分别计算2, 3, 4, 5, ..., 99自我相加多次的数值；
 - ◆ 每次计算得到的结果都是一个合数，在数组中标记该数字被“筛掉”；
 - ◆ 每次计算过程中，只要相加结果没有达到100就继续自我相加；
- ◆ 根据标记输出所有没有被筛掉的数字；

```
#include<iostream>
using namespace std;
int main()
{
    int sum = 0, a[100] = { 0 };
    for (int i = 2; i < 100; i++)
    {
        sum = i;
        while (sum < 100)
        {
            sum = sum + i;
            if (sum < 100) a[sum] = 1;
        }
    }
    for (int i = 2; i < 100; i++)
    {
        if (a[i] == 0) cout << i << " ";
    }
    return 0;
}
```


例3 找出素数

■ 筛法求素数：

- ◆ 埃拉托斯特尼（Eratosthenes，约公元前274～194年）发明，又称埃拉托斯特尼筛子。
- ◆ 基本思路
 - 不是挑选出所有素数，而是筛掉所有的合数；

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]	a[16]	a[17]	a[18]

例3 找出素数

■ 稍作优化：

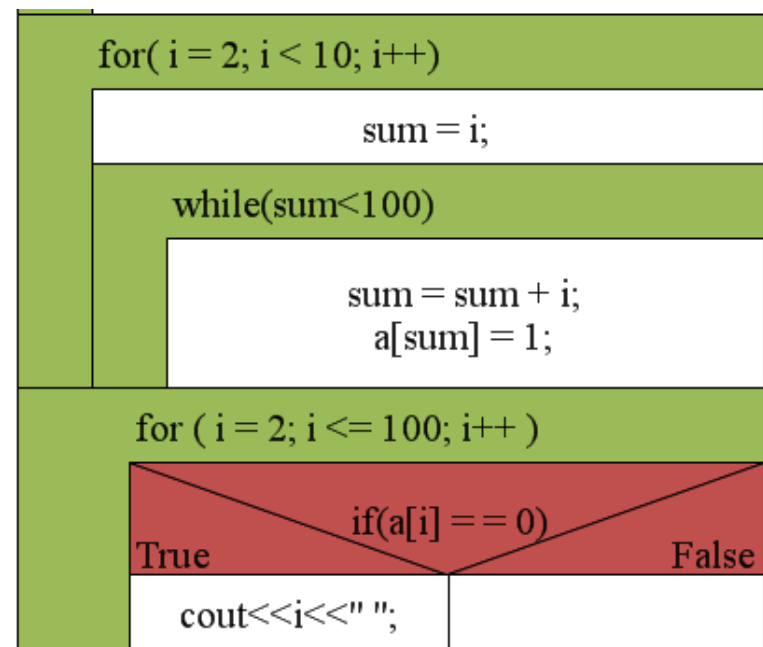
- ◆ 可以让 $2, 3, 4, 5, \dots, c$ 中的每个数自我相加多次，来获得100之内的所有合数；
- ◆ c 如何确定？
 - 根据初等数论，若 n 为合数，则 n 的最小正因数 c 满足：

$$1 < c \leq \sqrt{n}$$

```

#include<iostream>
#include<cmath>
using namespace std;
int main()
{
    int sum = 0, a[100] = { 0 };
    for (int i = 2; i < sqrt(100.0); i++)
    {
        sum = i;
        while (sum < 100)
        {
            sum = sum + i;
            if (sum < 100) a[sum] = 1;
        }
    }
    for (int i = 2; i < 100; i++)
    {
        if (a[i] == 0) cout << i << " ";
    }
    return 0;
}

```

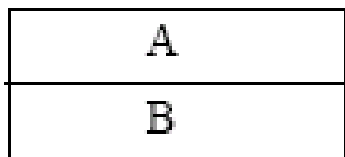


N-S图

用N-S图描述算法

■ N-S图

- ◆ 一种算法表示法，由美国人I. Nassi和B. Shneiderman共同提出，是算法的一种结构化描述方法。



顺序结构

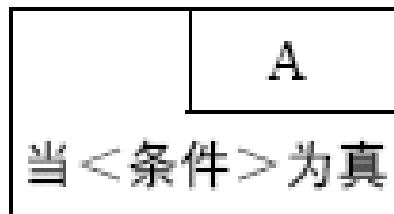
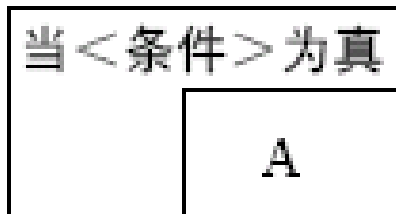


a)



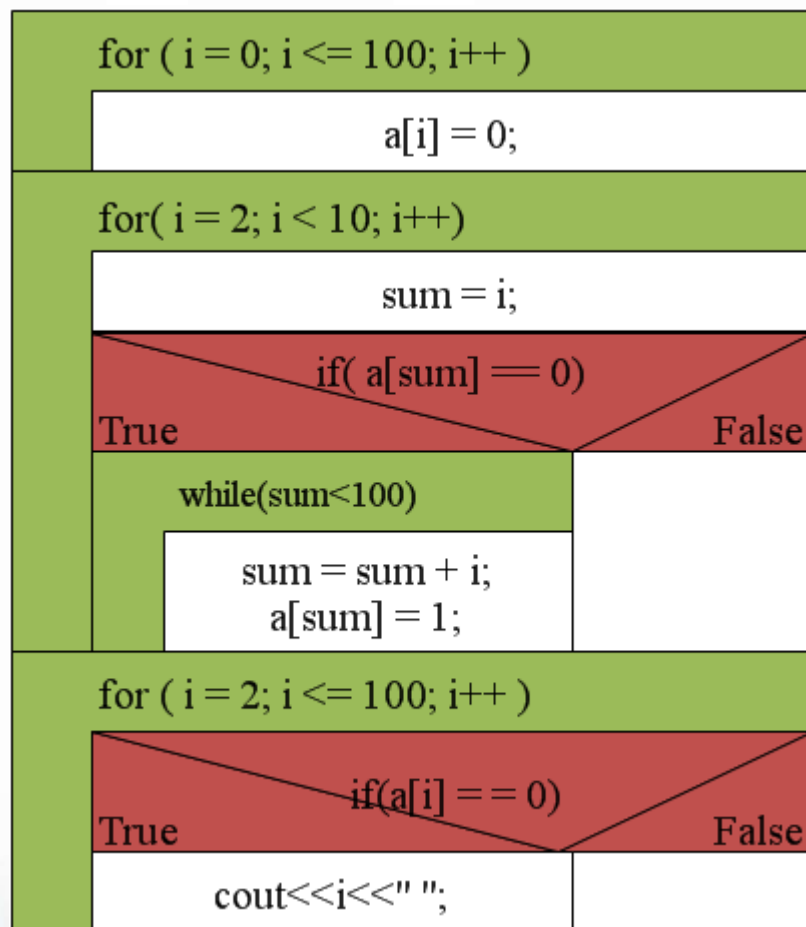
b)

选择结构的N-S图



循环结构的N-S图

优化的程序



```

#include<iostream>
#include<cmath>
using namespace std;
int main( )
{
    bool a[100] = {0};
    int sum = 0;
    for(int i = 0; i < 100; i++)
        a[i] = 0;
    for(int i = 2; i < sqrt(100.0); i++)
    {
        sum = i;
        if(a[sum]==0)
        {
            while(sum<100)
            {
                sum = sum + i;
                a[sum] = 1;
            }
        }
    }
    for(int i = 0; i < 100; i++)
        if(a[i] == 0)
            cout<<i<<" ";

    return 0;
}

```



数组的作用

■ 不仅

◆ 当你有一些数据要进行存储时：

- 用于存放一系列数据类型相同的数据；

■ 还能

◆ 当你的处理对象是连续的整数时：

- 利用数据与下标间的对应关系，解决问题；



谢 谢 ！



北京大学



统计单词数

■ 问题：

- ◆ 输入一个英文句子，统计其中有多少个单词，单词之间用空格分开。





统计单词数

■ 分析:

- ◆ 读入一整句话；
- ◆ 用什么区分单词？空格
- ◆ 每次读入一个字符时，如果是字母，就要判断前一个是不是空格，是则新单词开始，不是则继续
 - 需要记住前一个字符的状态:



北京大学



统计单词数

```
#include <iostream>
using namespace std;
int main()
{
    char str[80];
    int i, num = 0, flag = 0;
    cin.getline(str, 80);
    for(i = 0; str[i] != '\0'; i++)
        if (str[i]==' ') flag=0;
        else if (flag==0)
            { flag = 1; num++;}
    cout<<"字符串中有"<<num<<"个单词"<<endl;
    return 0;
}
```



北京大学



数组 的 应用示例

——排序



北京大学



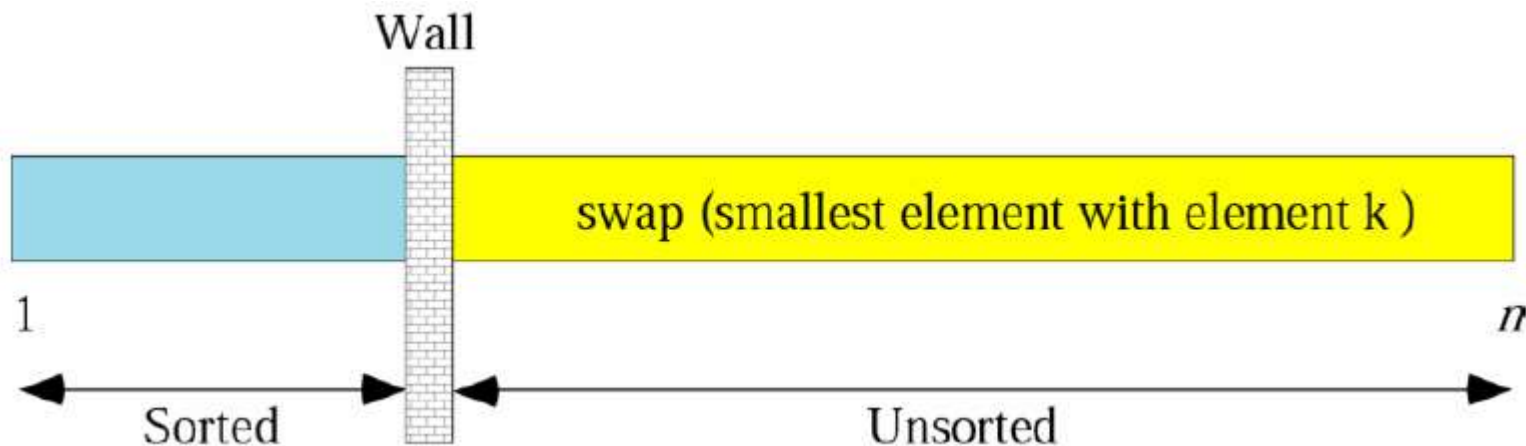
什么是排序

- **排序（Sort）**是指对一些数据的重新组织，使得数据由大到小（降序）或者由小到大（升序）存储。排序是很一种基本的要求。我们所感兴趣的是如何寻找到一个好（计算速度快或占用内存少）的办法来进行排序。
 - ◆ **选择排序**
 - ◆ **冒泡排序**

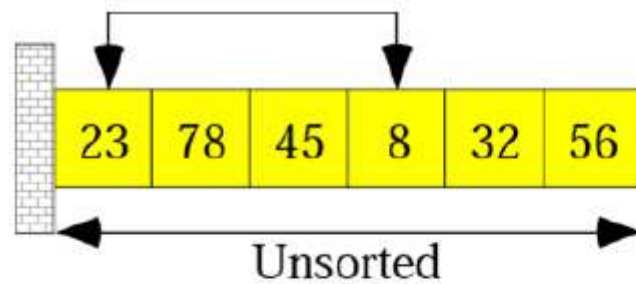


选择排序 Selection sort

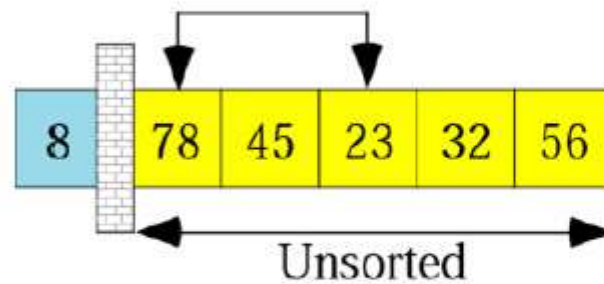
- ◆ 选身高，第一第二。。。，从余下的选最高
- ◆ 数据列表被分为两个子列表：已排序和未排序。找到未排序列表中值最小（或最大）的元素，并把它和未排序列表中的第一个元素进行交换。



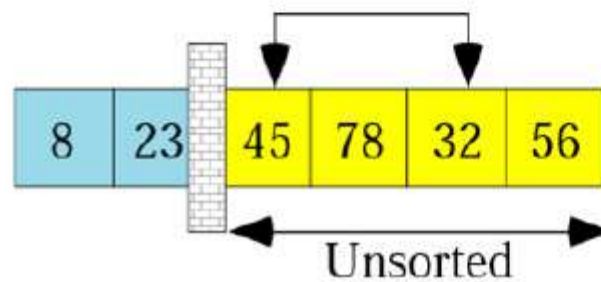
选择排序示例



Original list



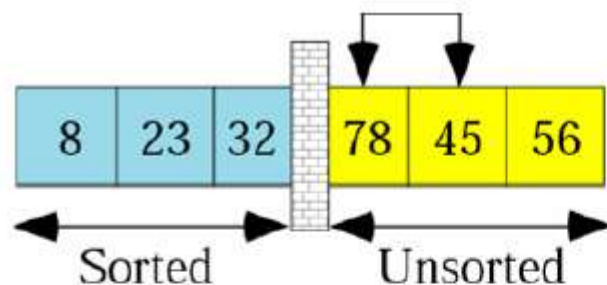
After pass 1



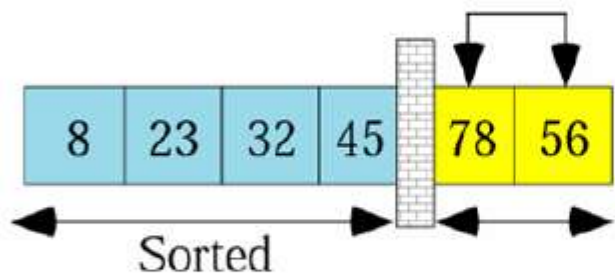
After pass 2



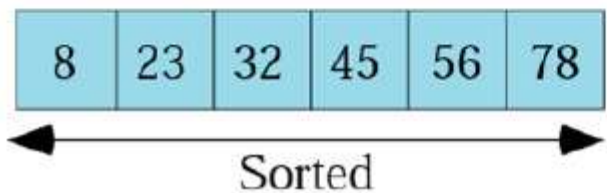
选择排序示例(续)



After pass 3

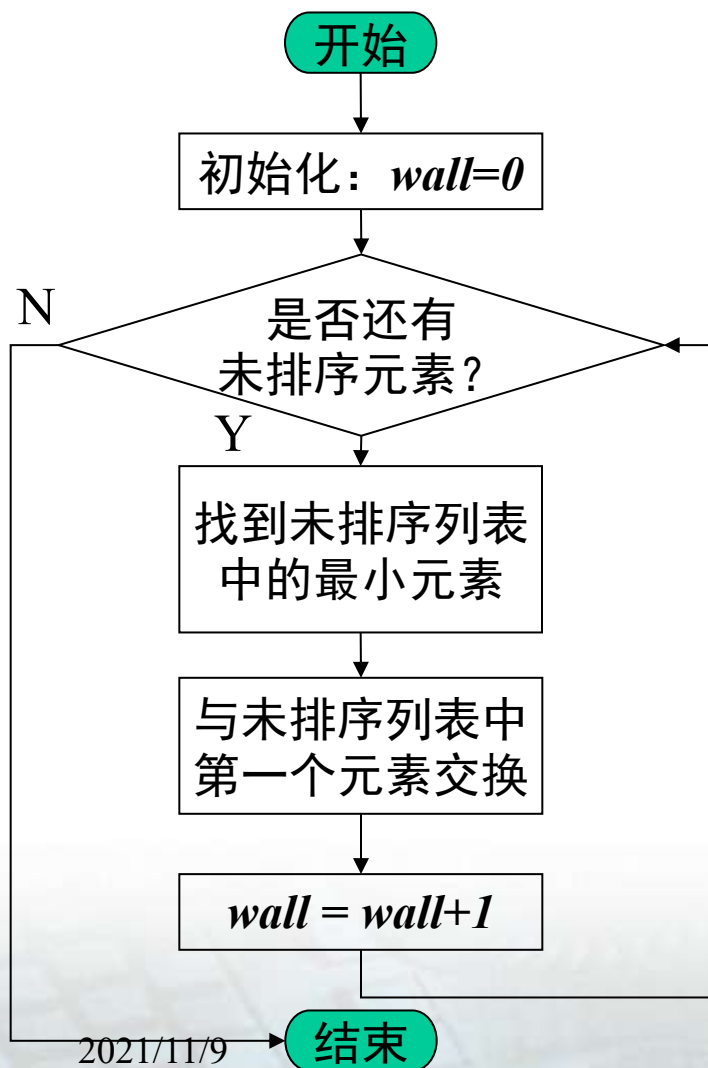


After pass 4



After pass 5

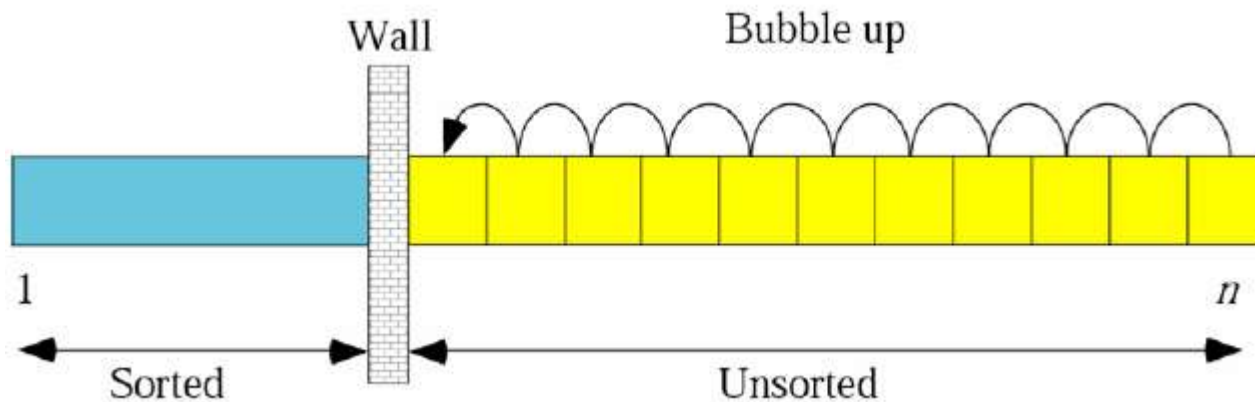
选择排序——算法、程序



```
void selectionSort(int A[], int n)
{
    int i, j, k, t;
    for ( i = 0; i < n; i++ )
    {
        k = i;
        /* 查找最小的元素 */
        for ( j = i + 1; j < n; j++ )
        {
            if (A[j] < A[k])
                k = j;
        }
        /* 交互i和k两个元素 */
        if ( k != i )
        {
            t = A[k];
            A[k] = A[i];
            A[i] = t;
        }
    }
}
```

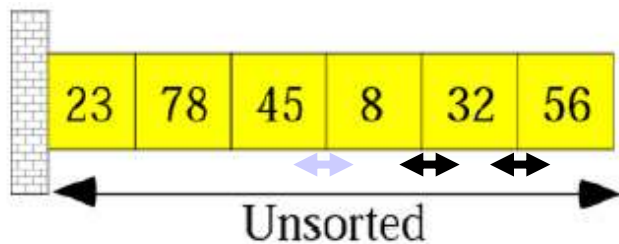

冒泡排序 Bubble sort

- ◆ 数据列表被分为两个子列表：已排序和未排序。未排序列表中最小（或最大）的元素通过冒泡的形式（**从后往前冒泡**）从未排序列表中交换到已排序列表中。

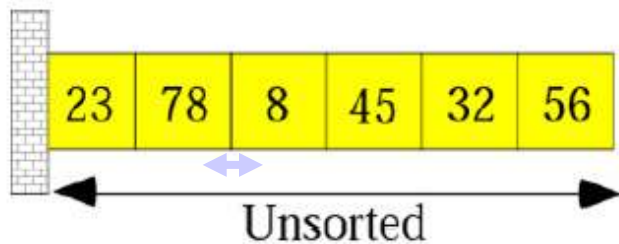


冒泡排序示例

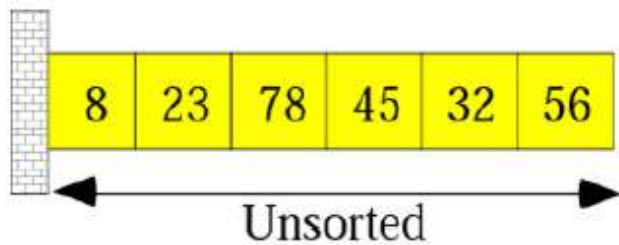
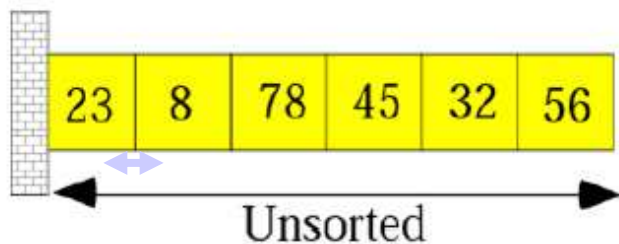
冒泡的过程



↔ 比较

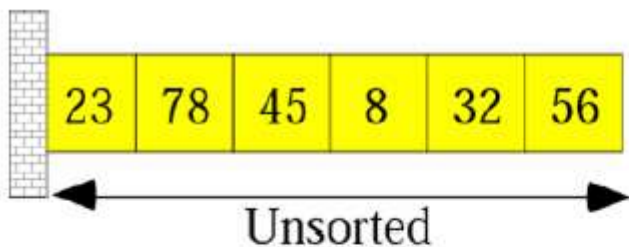


↔ 比较并交换

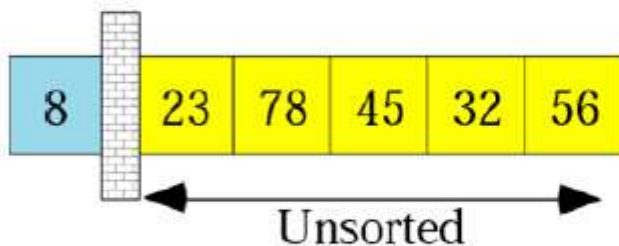




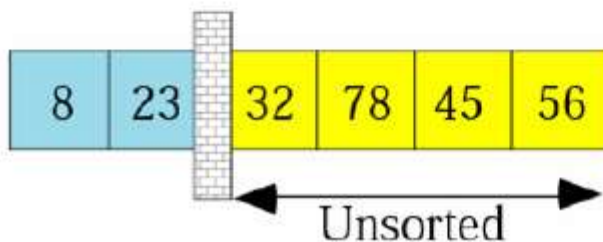
冒泡排序示例（续）



Original list



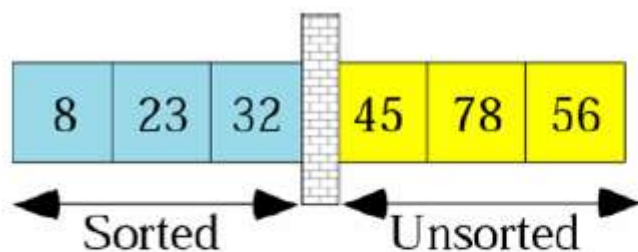
After pass 1



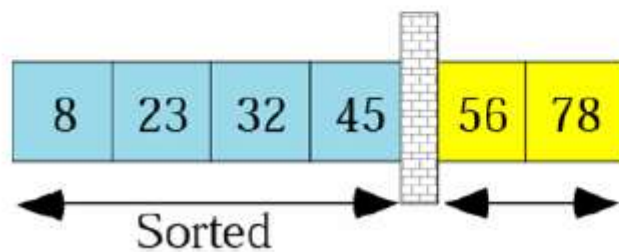
After pass 2



冒泡排序示例（续）

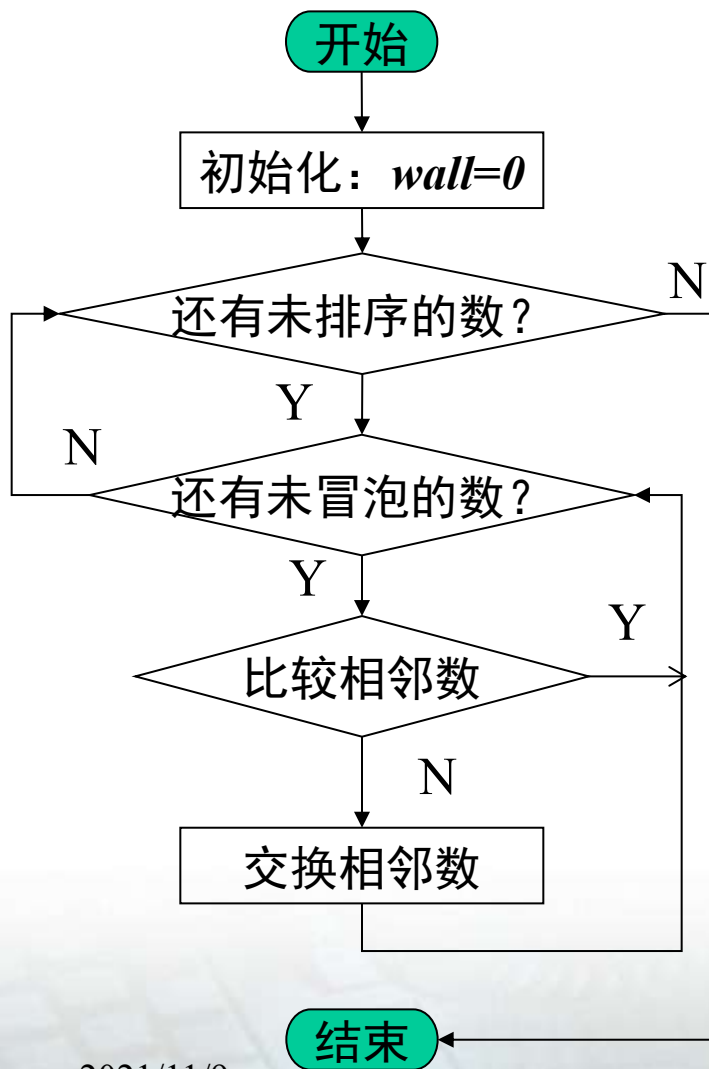


After pass 3



After pass 4
Sorted

冒泡排序——算法、程序



```
void bubbleSort(int A[], int n)
{
    int i, j, t;
    for ( i = 0; i < n; i++ )
    {
        for (j = n - 1; j > i; j--)
        {
            if (A[j] < A[j-1])
            {
                t = A[j];
                A[j] = A[j-1];
                A[j-1] = t;
            }
        }
    }
}
```



排序算法简介

■ 就地排序算法（不增加新的存储空间）

1、插入排序法（Insert Sort）

将一个数插入到序列中的合适位置。

2、选择排序法（Selection Sort）

每次把最小（大）的元素交换到最前面。

3、冒泡排序法（Bubble Sort）

比较并交换相邻的元素，直到所有元素都被放到合适的位置。

4、堆排序（Heap Sort）

一种效率非常高，但原理较复杂的排序方法。

5、快速排序算法（Quick Sort）