C程序设计基础

Introduction to C programming
Lecture 8: Array & String II

张振国 zhangzg@sustech.edu.cn

南方科技大学/理学院/地球与空间科学系

Review on L7 Array I

array

2-D and N-D array

String

Row-major or column-major order

1-D array

C provides a data structure called **array**. It stores a fixed-size collection of elements of the same type.

```
type name[size] = {…};

type name[] = {…};
```

int array

3	2	1	5	…	8

float array

1.2	4.5	-1.9	3.4	…	8.8

char array

H	R	O	Y	…	P

1-D array

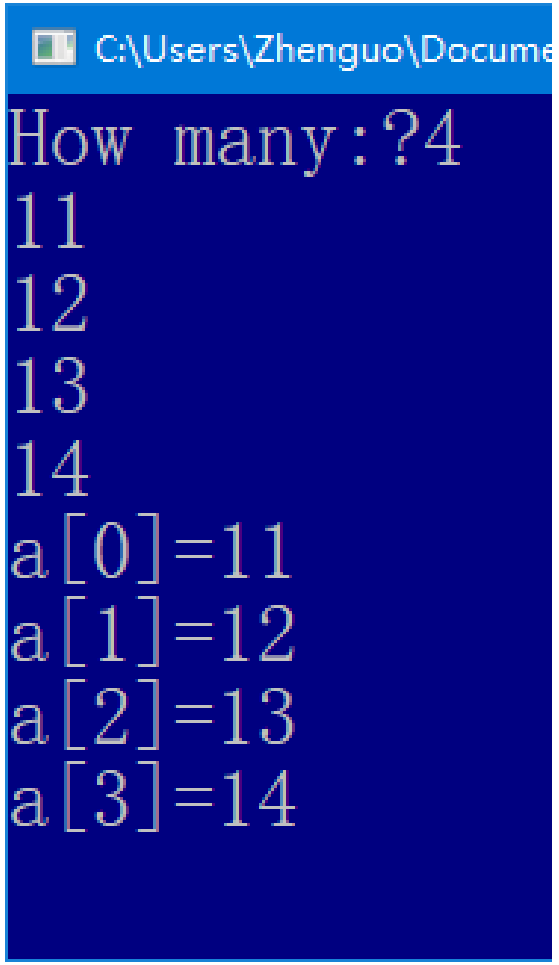
Declare, initialize and access an int array:

- int a[10]; **// declare**
- a[0] = 3, a[1] = 2, …., a[9] = 7; **// initialize**
- int a[10] = {3, 2, 1, 5, 6, 8, 9, 2, 0, 7}; **// declare and initialize**
- int a[] = {3, 2, 1, 5, 6, 8, 9, 2, 0, 7}; **// declare and initialize**
- printf("a[5] = %d", a[5]); **// access the array**

Declare

```
#include <stdio.h>
main() {
     int i, n;
     printf("How many:?");
     scanf("%d", &n);

     int a[n];
     for (i = 0; i < n; i++)
          scanf("%d", &a[i]);

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

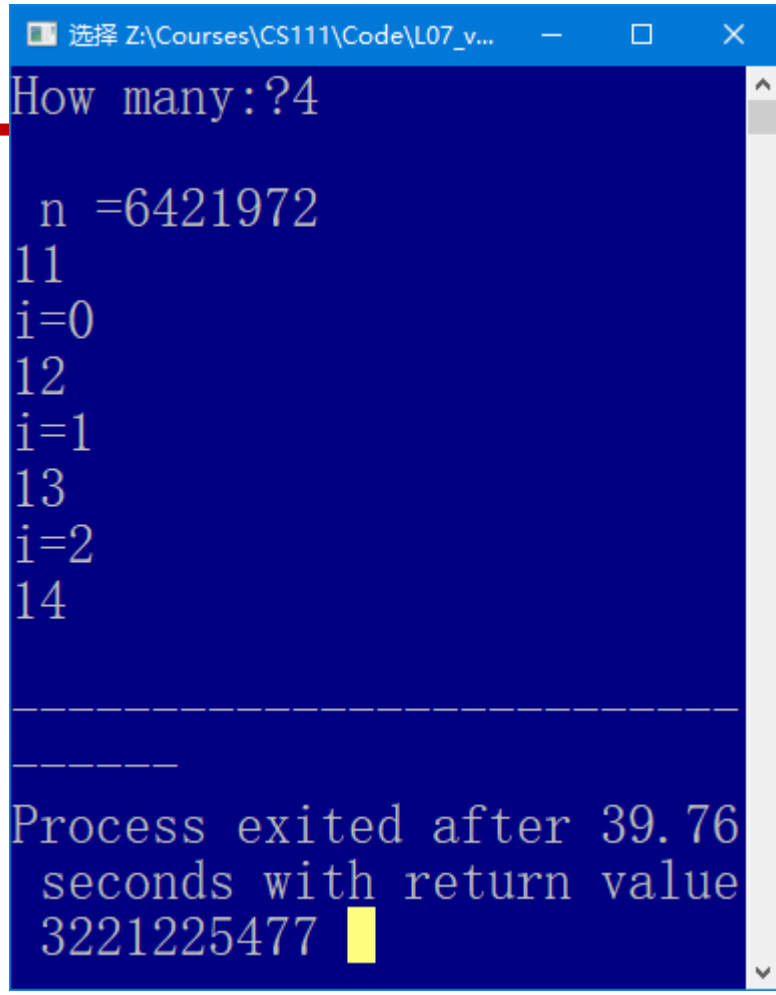
C99可实现变长数组，在函数调用中有诸多好处…



Declare

```
#include <stdio.h>
main() {
     int i, n;
     printf("How many:?");
     int a[n];
     scanf("%d", &n);
     printf("\n n=%d\n", &n);
     for (i = 0; i < n; i++){
          scanf("%d", &a[i]);
          printf("i=%d\n", i);
     }
     for (i = 0; i < n; i++) {
          printf("a[%d]=%d\n", i, a[i]);
     }
     return 0;}
```

C99可实现变长数组，在函数调用中有诸多好处，但是定义须在给定n之后



1-D array

char array: char c[5] = {‘h’, ‘e’, **2, 2.3**, ‘o’}; **// Wrong! Must be in same type!**

int array: int c[5] = {0, 1, 2, **2.5**, 5}; **// Wrong! Must be in same type!**

注意：
与机器、编译器有关有关，
尽量写成标准形式

1-D array

与循环结合的易错点：

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

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

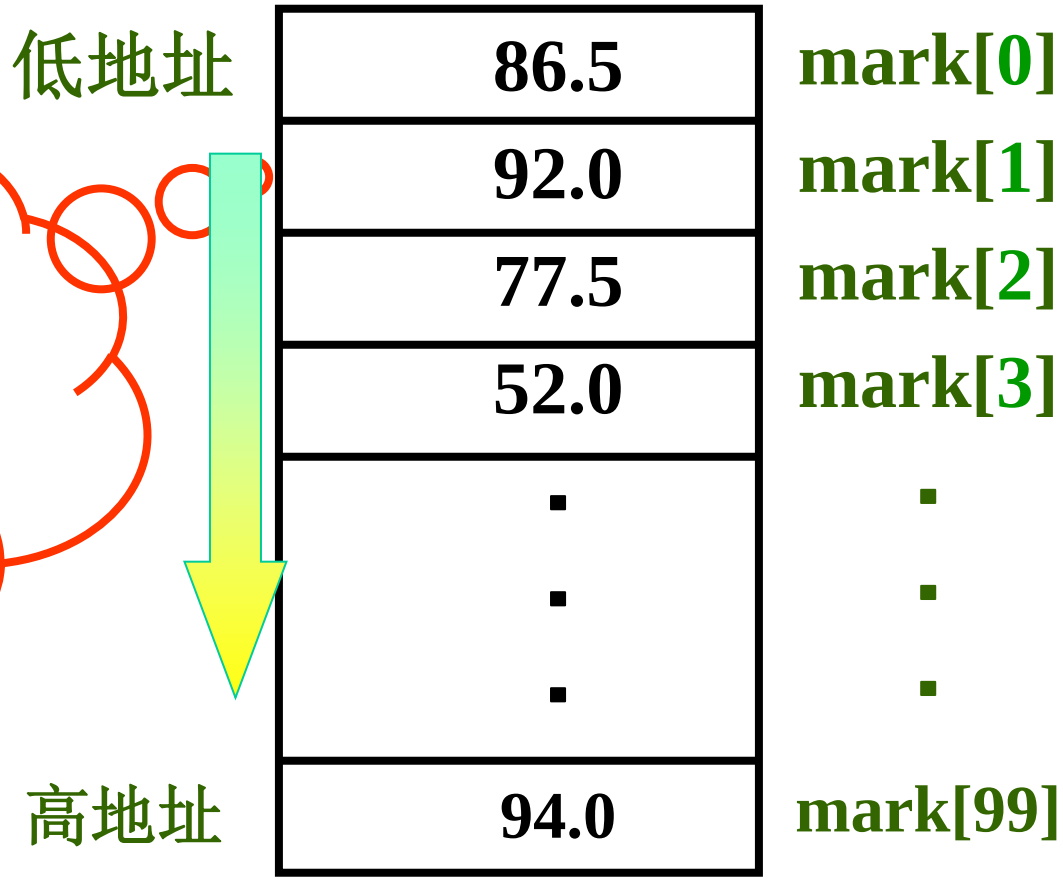
```
for(i=0; i<N; i++)
  a[i]=b[i];
```

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

1-D array

每个数据元素占用的字节数，就是基类型的字节数

一个元素占4个字节



Operations of 1-D array

int a[10] = {3, 2, 1, 5, 6, 8, 9, 2, 0, 7};

3	2	1	5	6	8	9	2	0	7

```
b = a
```

```
int b[] = a;
```

数组变量本身不能被赋值

```
for(i=0;i<n;i++)
{
  b[i] = a[i];
}
```

Operations of 1-D array: sorting

int a[10] = {3, 2, 1, 5, 6, 8, 9, 2, 0, 7};

3	44	38	5	47	15	36	26	27	2	46	4	19	50	48

How the sort the array?

2	3	4	5	15	19	26	27	36	38	44	46	47	48	50

Bubble sort

```c
#include<stdio.h>
int main(void) {
    int arr[] = { 22, 34, 3, 32, 82, 55, 89, 50, 37, 5, 64, 35, 9, 70 };
    int len = (int)sizeof(arr) / sizeof(arr[0]);

    for (int i = 0; i < len - 1; i++) // for each element
        for (int j = 0; j < len - 1 - i; j++) // compare with rest
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }

    for (int i = 0; i < len; i++)
        printf("%d \n", arr[i]);
}
```

```
C:\U...
22
34
3
32
82
55
89
50
37
5
64
35
9
70
```

```
Microso
3
5
9
22
32
34
35
37
50
55
64
70
82
89
```

Selection sort

```c
#include<stdio.h>
int main(void) {
        int arr[] = { 22, 34, 3, 32, 82, 55, 89, 50, 37, 5, 64, 35, 9, 70 };
        int len = (int)sizeof(arr) / sizeof(*arr);
        for (int i = 0; i < len - 1; i++) {
                int min = i;
                for (int j = i + 1; j < len; j++) {
                        if (arr[j] < arr[min])
                                min = j;
                }
                int temp = arr[min];
                arr[min] = arr[i];
                arr[i] = temp;
        }
        int i;
        for (i = 0; i < len; i++)
                printf("%d\n", arr[i]);
}
```

C:\U...

```
22
34
3
32
82
55
89
50
37
5
64
35
9
70
```

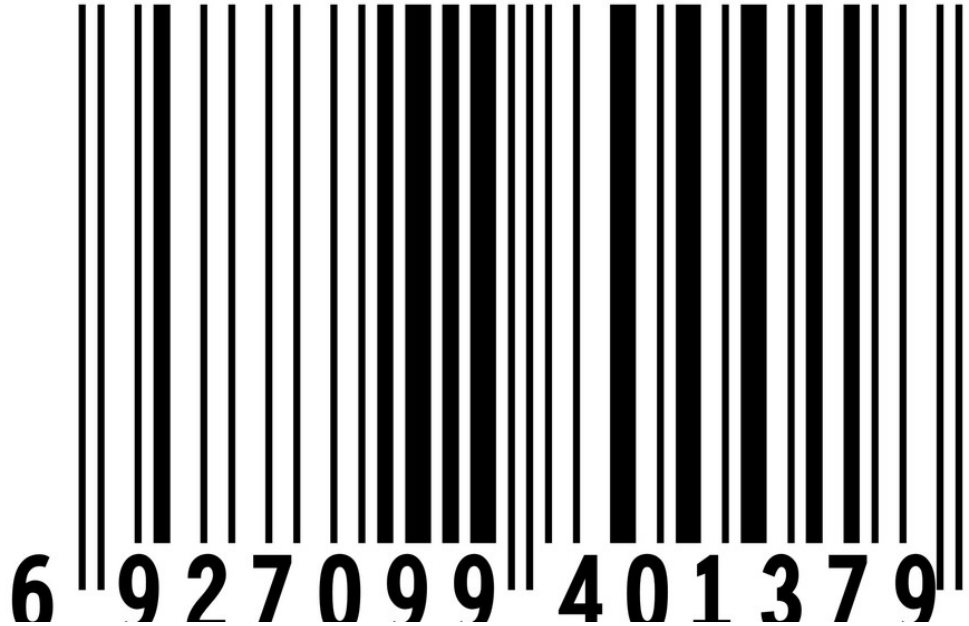
Microso

```
3
5
9
22
32
34
35
37
50
55
64
70
82
89
```

Content

1. 1-D array
2. 2-D and N-D array
3. String
4. Row-major or column-major order

2-D array in life



2-D array

1D-array can be extended to **2D structure**, with (X, Y) indexing the element.

```
type name[size][size];

type name[size][size] = {{…}, {…},…, {…}};

type name[][] = {{…}, {…},…, {…}};
```

2-D array

Declare and initialize a 2D int array

3	2	5
1	7	6

- int a[2][3]; ***// 2 rows x 3 columns***
- a[0][0] = 3; a[0][1] = 2; a[0][2] = 5;
- a[1][0] = 1; a[1][1] = 7; a[1][2] = 6;

Access array: printf("a[1][1] = %d", a[1][1]);

1	0	0	2
0	1	0	0
0	2	1	4

- int a[3][4]; ***// 3 rows x 4 columns***
- a[0][0] = 1; a[0][1] = 0; a[0][2] = 0; a[0][3] = 2;
- a[1][0] = 0; a[1][1] = 1; a[1][2] = 0; a[1][3] = 0;
- a[2][0] = 0; a[2][1] = 2; a[2][2] = 1; a[2][3] = 4;

Access array: printf("a[2][3] = %d", a[2][3]);

2-D array

Declare and initialize a 2D int array

- int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
- int a[2][3] = {1, 2, 3, 4, 5, 6}; // **preferred!**
- int a[ ][3] = {1, 2, 3, 4, 5, 6}; // 2 x 3 mat
- int a[3][4] ={{1}, {5, 6}}; // 3 x 4 mat

1	2	3
4	5	6

1	0	0	0
5	6	0	0
0	0	0	0

Initialize

1．分行给二维数组赋初值。

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

1st row　　2nd row　　3rd row

2．可以将所有数据写在一个花括号内，按数组排列的顺序对各元素赋初值。

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

Initialize

3 . 可以对部分元素赋初值。

例如： int a[3][4]={{1}，{5}，{9}};

也可以对各行中的某一元素赋初值，如

```
int a[3][4]={{1}, {0, 6}, {0, 0, 0, 11}};
```

1	0	0	0
5	6	0	0
0	**0**	**0**	**0**

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

1	0	0	0
5	0	0	0
9	**0**	**0**	**0**

也可以只对某几行元素赋初值。如：

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

Initialize

4．如果对全部元素都赋初值，则定义数组时对第一维的长度可以不指定，但第二维的长度不能省。

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

等价于

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

在定义时也可以只对部分元素赋初值而省略第一维的长度，但应分行赋初值。

例如：`int a[][4] ={{0, 0, 3}, {}, {0, 10}};`

0	0	3	0
0	0	0	0
0	10	0	0

2-D array

1. 下标可以是整型表达式:

a[2-1][2*2-1] √

但是，不要写成 a[2，3]，a[2-1，2*2-1]！！！！

2. 数组元素可以出现在表达式中，也可以被赋值

b[1][2]=a[2][3] /2

2-D array

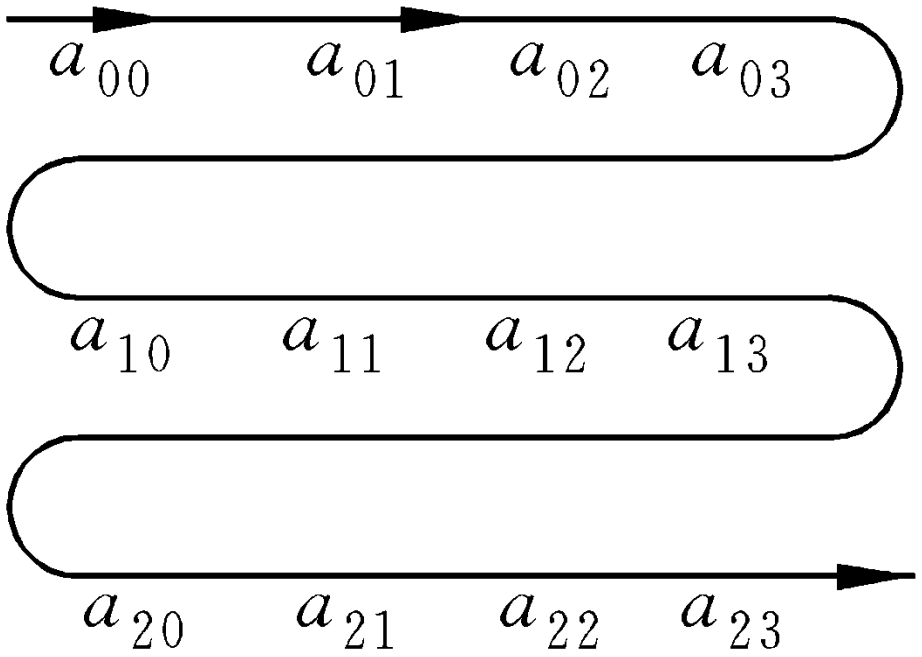
3. 在使用数组元素时，应该注意下标值应在已定义的数组大小的范围内。

```
int a[3][4];    /* 定义a为3×4的数组 */
        ⋮
a[3][4]  =3;
```

×

2-D array

二维数组中的元素在内存中的排列顺序是：按行存放，即先顺序存放第一行的元素，再存放第二行的元素……



2-D array

整型数组 b[3][3]={ {1,2,3}, {4,5,6}, {7,8,9} };

地址	值	数组元素
3000H	1	b[0][0]
3002H	2	b[0][1]
3004H	3	b[0][2]
3006H	4	b[1][0]
3008H	5	b[1][1]
300AH	6	b[1][2]
300CH	7	b[2][0]
300EH	8	b[2][1]
3010H	9	b[2][2]

3-D/N-D array

Declare and initialize a 3-D/N-D int array

- int a[2][3][4];
- a[0][0][0] = 1; a[0][1][2] = 3; a[1][0][3] = 2; // **preferred!**
- int a[2][3][4]= {{{1, 2, 3}, {4, 5, 6}},{{2, 4, 5}, {2, 4, 2}}…};

- int a[2][3][4][2];
- a[0][0][0][0] = 1; a[0][1][2][0] = 3; a[1][0][3][1] = 2;

3-D/N-D array

定义三维数组： `float a[2][3][4];`

注意： 多维数组元素在内存中的排列顺序：

第一维的下标变化最慢，最右边的下标变化最快。

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

Use for loop to define 2D/3D array

2D array

```
int n[4][5];
for (int x = 0; x < 4; x++)
{
    for (int y = 0; y < 5; y++)
    {
        n[x][y] = x+y;
    }
}
```

3D array

```
int n[2][2][3];
for (int x = 0; x < 2; x++)
{
    for (int y = 0; y < 2; y++)
    {
        for (int z = 0; z < 3; z++)
        {
            n[x][y][z] = x+y+z;
        }
    }
}
```

Case study: 2-D array

Case: how to print a 2D float array and char array

```
#include <stdio.h>
int main ()
{
float a[5][2] = { {0.5,1.5},
{1.2,2.1},{2.4,4.2},
{3.4,6.4},{4.4,8.5}};
for ( int i = 0; i < 5; i++ )
{
  for ( int j = 0; j < 2; j++ )
  {
   printf("%f ", a[i][j] );
  }
  printf("\n");
}
return 0;
}
```

```
0.500000 1.500000
1.200000 2.100000
2.400000 4.200000
3.400000 6.400000
4.400000 8.500000
```

```
#include <stdio.h>
int main()
{
char a[5][2] = { {'A','B'}, {'C','D'},
{'E','F'}, {'G','H'},{'I','J'}};
for (int i = 0; i < 5; i++)
   {
      for (int j = 0; j < 2; j++)
      {
         printf("%c", a[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

```
AB
CD
EF
GH
IJ
```

Operations of 2-D array: matrix

Definition of matrix: **A matrix is a collection of numbers arranged into a fixed number of rows and columns.** 

$$\begin{pmatrix} 2 & 5 & 4 \\ 1 & 3 & 6 \\ 7 & 2 & 3 \end{pmatrix}$$

Operations of 2-D array: matrix

Most decisions can be expressed as a linear equation!

$$a \cdot X_1 + b \cdot X_2 + c \cdot X_3 = Y$$

a, b, c are system coefficients

X is observation or measurement

Y is decision

Operations of 2-D array: matrix

$$a \cdot X_1 + b \cdot X_2 + c \cdot X_3 = Y$$

$$\begin{pmatrix} a & b & c \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = Y$$

system coefficients

observation or measurement

decision

Operations of 2-D array: matrix

Multiple measurements build up a matrix

$$\begin{pmatrix} a & b & c \end{pmatrix} \cdot \begin{pmatrix} X_{11} & X_{21} & X_{31} \\ X_{12} & X_{22} & X_{32} \\ X_{13} & X_{23} & X_{33} \end{pmatrix} = \begin{pmatrix} Y_1 & Y_2 & Y_3 \end{pmatrix}$$

system coefficients

observation or measurement

decision

Basic matrix operations

+, -, .*, ./, *, /

$$A = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{21} \\ b_{12} & b_{22} \end{pmatrix}$$

Basic matrix operations

Matrix adding and subtraction

$$A \pm B = \begin{bmatrix} a_{00} \pm b_{00}, & a_{01} \pm b_{01}, & \cdots & a_{0j} \pm b_{0j} \\ a_{10} \pm b_{10}, & a_{11} \pm b_{11}, & \cdots & a_{1j} \pm b_{1j} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i0} \pm b_{i0}, & a_{i1} \pm b_{i1}, & \cdots & a_{ij} \pm b_{ij} \end{bmatrix}$$

Matrix dot product

$$A \cdot B = \begin{bmatrix} a_{00} \cdot b_{00}, & a_{01} \cdot b_{01}, & \cdots & a_{0j} \cdot b_{0j} \\ a_{10} \cdot b_{10}, & a_{11} \cdot b_{11}, & \cdots & a_{1j} \cdot b_{1j} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i0} \cdot b_{i0}, & a_{i1} \cdot b_{i1}, & \cdots & a_{ij} \cdot b_{ij} \end{bmatrix}$$

Matrix cross product

$$A_{23} \times B_{32} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} \cdot b_{00} + a_{01} \cdot b_{10} + a_{02} \cdot b_{20}, & a_{00} \cdot b_{01} + a_{01} \cdot b_{11} + a_{02} \cdot b_{21} \\ a_{10} \cdot b_{00} + a_{11} \cdot b_{10} + a_{12} \cdot b_{20}, & a_{10} \cdot b_{01} + a_{11} \cdot b_{11} + a_{12} \cdot b_{21} \end{bmatrix}$$

Basic matrix operations

How to transpose a matrix?

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

1	2	3
4	5	6

Rotate 90°

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

1	4
2	5
3	6

Case study: 2-D array

Case: how to transpose a 2D matrix?

```
#include <stdio.h>
main()
{
    int a[2][3] = {{1, 2, 4}, {4, 5, 2}};
    int a_trans[3][2];
    for (int i = 0; i < 2; i++){
        for (int j = 0; j < 3; j++){
            a_trans[j][i] = a[i][j];
        }
    }
    printf("\nMatrix A:\n");
    for (int i = 0; i < 2; i++){
        for (int j = 0;j < 3; j++){
           printf("%d ", a[i][j]);}
        printf("\n");}
    printf("\nTranspose of matrix A:\n");
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 2; j++){
            printf("%d ", a_trans[i][j]);}
        printf("\n");}
}
```

```
Matrix A:
1 2 4
4 5 2

Transpose of matrix A:
1 4
2 5
4 2
```

Basic matrix operations

How to turn a matrix upside down?

3	6
2	5
1	4

upside down

1	4
2	5
3	6

Case study: 2-D array

Case: how to turn a 2D matrix upside down?

```
#include <stdio.h>
int main(){
    int a[][2] = {1, 2, 3, 4, 5, 6};
    int i, j, m, n, tmp;
    m = 3; n = 2;
    printf("Original matrix:\n");
    for(i = 0; i < m; i ++)
        for(j = 0; j < n; j ++)
            if(j == 1) printf("%d\n", a[i][j]);
            else printf("%d ", a[i][j]);
    for(i = 0; i < m/2; i ++)
        for(j = 0; j < n; j ++){
            tmp = a[i][j];
            a[i][j] = a[m-1-i][j];
            a[m-1-i][j] = tmp;
        }
    printf("Upside down matrix:\n");
    for(i = 0; i < 3; i ++)
        for(j = 0; j < 2; j ++)
            if(j == 1) printf("%d\n", a[i][j]);
            else printf("%d ", a[i][j]);
    return 0;
}
```

```
Original matrix:
1 2
3 4
5 6
Upside down matrix:
5 6
3 4
1 2
```

Case study: 2-D array

Case: how to reverse the left and right?

```c
#include <stdio.h>
int main(){
    int a[][2] = {1, 2, 3, 4, 5, 6};
    int i, j, m, n, tmp;
    m = 3; n = 2;
    printf("Original matrix:\n");
    for(i = 0; i < m; i ++)
        for(j = 0; j < n; j ++)
            if(j == 1) printf("%d\n", a[i][j]);
            else printf("%d ", a[i][j]);
    for(i = 0; i < m; i ++)
        for(j = 0; j < n/2; j ++){
            tmp = a[i][j];
            a[i][j] = a[i][n-1-j];
            a[i][n-1-j] = tmp;
        }
    printf("Upside down matrix:\n");
    for(i = 0; i < 3; i ++)
        for(j = 0; j < 2; j ++)
            if(j == 1) printf("%d\n", a[i][j]);
            else printf("%d ", a[i][j]);
    return 0;
}
```

```
Original matrix:
1 2
3 4
5 6
Upside down matrix:
2 1
4 3
6 5
```

Case study: subtract 2 matrices

Case: how to subtract 2 matrices?

```
#include <stdio.h>
main()
{
    int a[2][2] = {{1, 2},{4, 5}};
    int b[2][2] = {{2, 2},{1, 3}};
    int c[2][2];
    for (int i = 0; i < 2;i++){
        for (int j = 0; j < 2; j++){
            c[i][j] = a[i][j] - b[i][j];
        }
    }
    printf("Matrix A-B:\n");
    for(int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
}
```

```
Matrix A:
1 2
4 5

Matrix B:
2 2
1 3
```

```
Matrix A-B:
-1 0
3 2
```

Case study: dot multiplication

Case: how to dot multiply 2 matrices?

```
#include <stdio.h>
main()
{
    int a[2][2] = {{1, 2},{4, 5}};
    int b[2][2] = {{2, 2},{1, 3}};
    int c[2][2];
    for (int i = 0; i < 2;i++){
        for (int j = 0; j < 2; j++){
            c[i][j] = a[i][j] * b[i][j];
        }
    }
    printf("Hadamard product of A and
    B:\n");
    for(int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
}
```

```
Matrix A:
1 2
4 5

Matrix B:
2 2
1 3
```

```
Hadamard product of A and B:
2 4
4 15
```

Case study: cross multiplication

Case: how to cross multiply 2 matrices?

```
#include <stdio.h>
main()
{
    int a[2][2] = {{1, 2},{4, 5}};
    int b[2][3] = {{2, 2, 1},{1, 3, 2}};
    int c[2][3];
    for (int i = 0; i < 2;i++){
        for (int j = 0; j < 3; j++){
            for (int k = 0; k < 2; k++){
                if(k==0)
                    c[i][j]=a[i][k]*b[k][j];
                else
                    c[i][j]+=a[i][k]*b[k][j];
        }}}
    printf("Cross product of A and B:\n");
    for(int i = 0; i < 2; i++){
        for (int j = 0; j < 3;j++){
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
}
```

怎么改进该片段？

```
Matrix A:
1 2
4 5

Matrix B:
2 2 1
1 3 2

Cross product of A and B:
4 8 5
13 23 14
```

Case study: cross multiplication

Case: how to cross multiply 2 matrices?

```
#include <stdio.h>
main()
{
    int a[2][2] = {{1, 2},{4, 5}};
    int b[2][3] = {{2, 2, 1},{1, 3, 2}};
    int c[2][3]={0};
    for (int i = 0; i < 2;i++){
        for (int j = 0; j < 3; j++){
            for (int k = 0; k < 2; k++){

                c[i][j]+=a[i][k]*b[k][j];


        }}}
    printf("Cross product of A and B:\n");
    for(int i = 0; i < 2; i++){
        for (int j = 0; j < 3;j++){
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
}
```

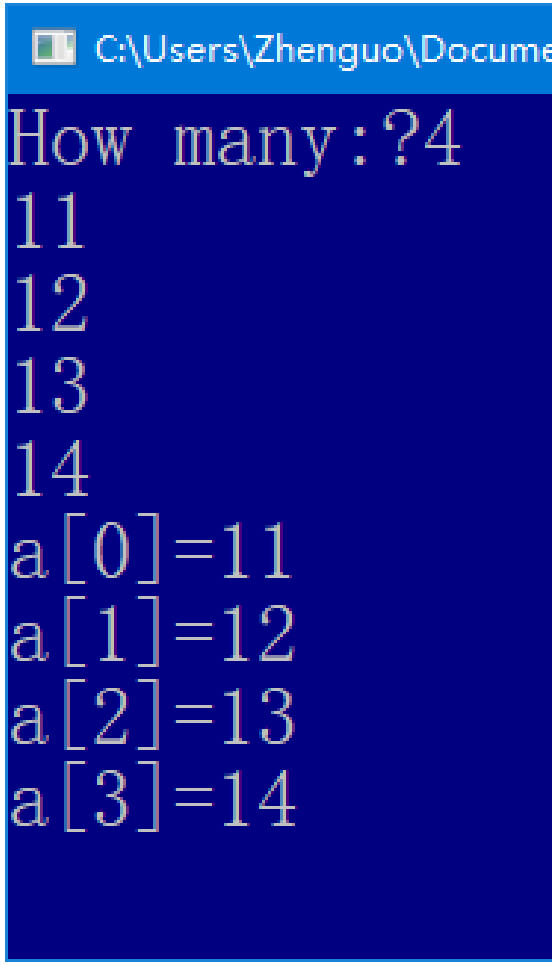
怎么改进该片段？

```
Matrix A:
1 2
4 5

Matrix B:
2 2 1
1 3 2

Cross product of A and B:
4 8 5
13 23 14
```

Variable-Length Arrays(VLAS)

```
#include <stdio.h>
main() {
     int i, n;
     printf("How many:?");
     scanf("%d", &n);

     int a[n];
     for (i = 0; i < n; i++)
          scanf("%d", &a[i]);

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

C99可实现变长数组，在函数调用中有诸多好处…

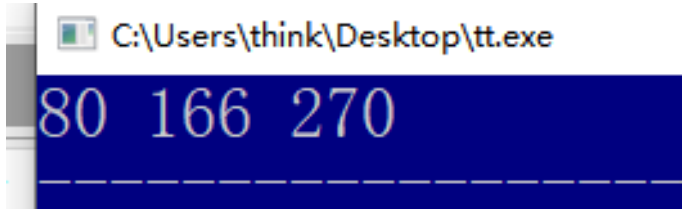


Variable-Length Arrays(VLAS)

“变”并不表示再创建数组，其意思是其维可以用变量来指定

```
#include <stdio.h>
int sum2d(int rows, int cols,  int ar[rows][cols]) {
        int r,c;
        int tot = 0;
        for (r = 0; r < rows; r++)
                for (c = 0; c < cols; c++)
                        tot += ar[r][c];
        return tot;
}
```

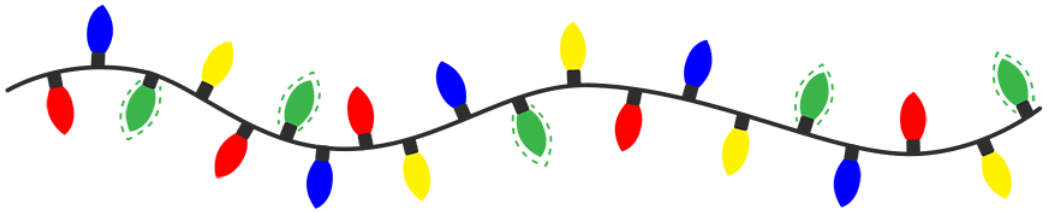
```
int main() {
        int i, j;
        int tot1, tot2, tot3;
        int rs = 3;
        int cs = 10;
        int varr[rs][cs];
        int junk[3][4] = {{2, 4, 6, 8}, {3, 5, 7, 9}, {12, 10, 8, 6}};
        int morejunk[2][4] = {{20, 30, 40, 50}, {5, 6, 7, 8}};
        for (i = 0; i < rs; i++)
                for (j = 0; j < cs; j++)
                        varr[i][j] = i * j + j;
        tot1 = sum2d(3, 4, junk );
        tot2 = sum2d( 2, 4, morejunk );
        tot3 = sum2d(rs, cs, varr);
        printf("%d %d %d", tot1, tot2, tot3);
        return 0;
}
```



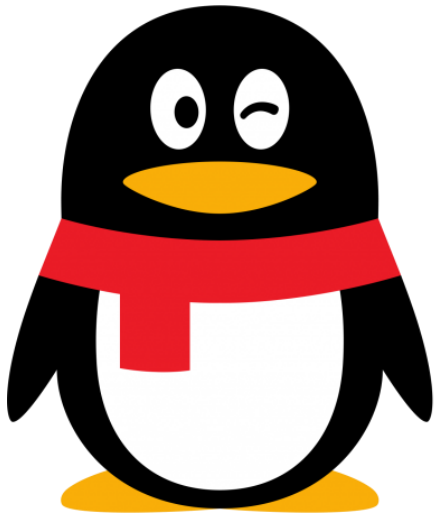
Content

1. 1-D array
2. 2-D and N-D array
3. **String**
4. Row-major or column-major order

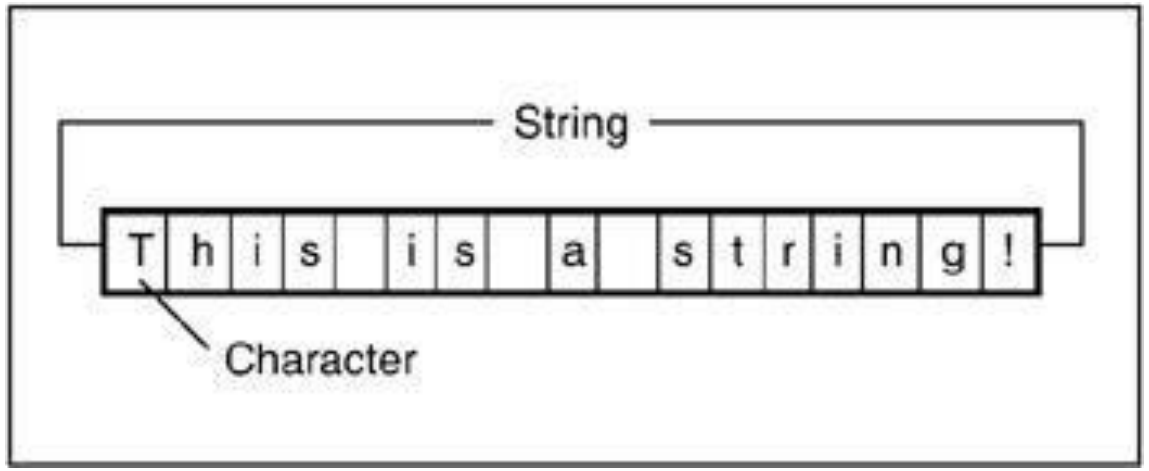
String in life



String in life



QQ



String in life

What do you search?
& What do you match?

String

String is an array of characters.

```
char name[size] = {‘’,’’,…,’’};

char name[size] = {“…”};

char name[] = {“…”};

char name[] = “…”;
```

String

```
char c[10] = {'I', ' ', 'a', 'm', ' ', 'h', 'a', 'p' , 'p', 'y'}; // length is 10
char c[10] = {"I am happy"};
char c[] = {"I am happy"};
char c[] = "I am happy"; // preferred
```

I		a	m		h	a	p	p	y
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]

1D and 2D String

1D char array holds the characters!

char c[10] = “I am happy”;

Machine thinks it as a single “word”!

I		a	m		h	a	p	p	y

2D char array holds the words!

char c[3][10] = {“I”, “am”, “happy”};

Machine thinks it as a group of word!

I									
a	m								
h	a	p	p	y					

在c语言中，字符串是作为字符数组来处理

1D and 2D String

This may be beyond of the scope!

Can someone understand what this code block means?

```
int main(int argc, char *argv[]){
    if(argc != 2) {
        printf("Usage: %s n\n", argv[0]);
        exit(1);
    }
```

String

```
char c[10] = {‘S’, ‘ U’, ‘S’, ‘T’, ‘e ’, ‘c’, ‘h’}; // length is 10
char c[10] = {“SUSTech”};
char c[] = {“SUSTech”};
char c[] = “SUSTech”; // preferred
```

S	U	S	T	e	c	h	\0	\0	\0
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]

String

为了测定字符串的实际长度，C语言规定了一个“字符串结束标志”——‘\0’。

字符数组并不要求它的最后一个字符为'\0'，甚至可以不包含'\0'。

char c[5] = {'C'，'h'，'i'，'n'，'a'};

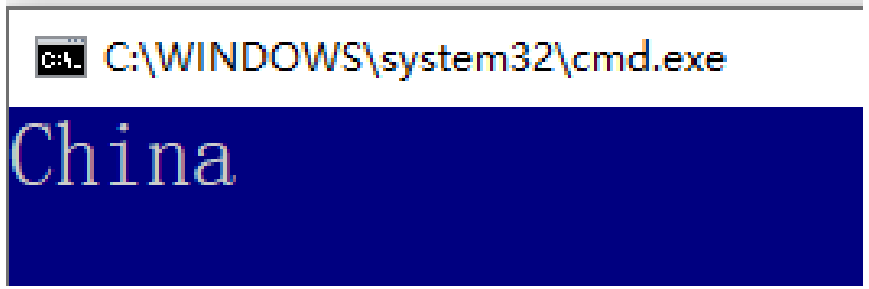
但是由于系统对字符串常量自动加一个'\0'。因此，为了使处理方法一致，在字符数组中也常人为地加上一个'\0'。

char c[6] = {'C'，'h'，'i'，'n'，'a'，'\0'};

IO - printf

```
char c [] ={" China" };
printf(" %s" , c);
```

在内存中的状态



c 数组

2000	C
2001	h
2002	i
2003	n
2004	a
2005	\0

printf输出时，输出的字符不包括结束符'\0'

IO - printf

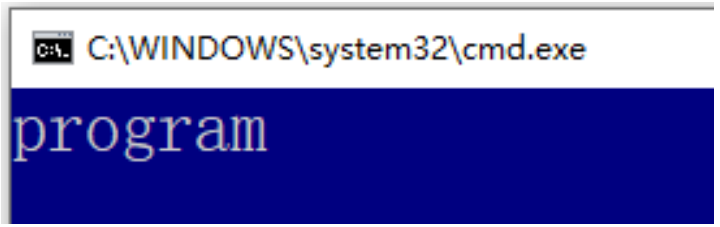
(2)用“%s”格式符输出字符串时，printf函数中的输出项是字符数组名，而不是数组元素名

```
printf("%s", c[0]);
```

×

(3)如果数组长度大于字符串的实际长度，也只能输出遇'\0'结束：

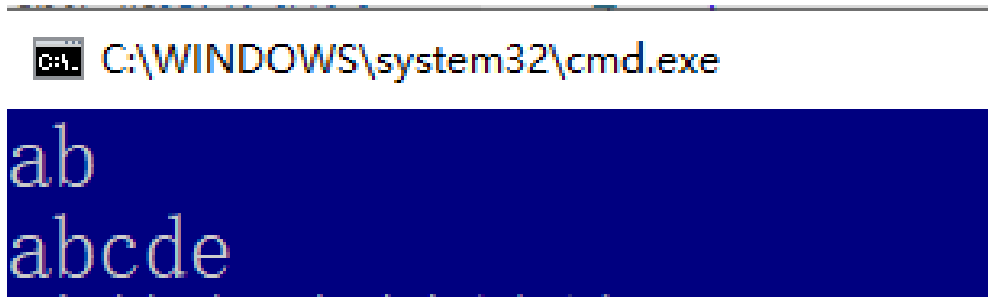
```
char c[10] = "program";
printf("%s\n", c);
```



IO - printf

(4)如果一个字符数组中包含一个以上′\0′，则遇第一个′\0′时输出就结束。

```
#include <stdio.h>

int main()
{
    char a[10] = {'a', 'b', '\0', 'c', 'd', 'e'};
    char b[10] = {'a', 'b', 'c', 'd', 'e'};
    printf("%s\n", a);
    printf("%s", b);
    return 0;
}
```



在执行printf函数时，每输出一个字符检查一次，看下一个字符是否为'\0'，遇'\0'就停止输出

IO - scanf

(1)如果利用一个scanf函数输入多个字符串，则在输入时以空格分隔。

```
char str1 [5] , str2 [5] , str3 [5] ;
scanf(" %s %s %s" , str1, str2, str3);
```

输入数据：

How are you?

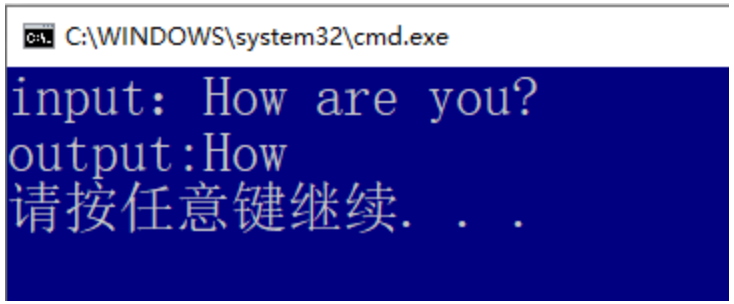
数组中未被赋值的元素的值自动置'\0'。

H	o	w	\0	\0
a	r	e	\0	\0
y	o	u	?	\0

IO - scanf

(2)如果利用一个scanf函数输入的字符串中若输入空格，其后都为'\0'

```
int main()
{
        char str[13];
        printf("input: ");
        scanf("%s", str);
       printf("output:%s", str);
}
```



H	o	w	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

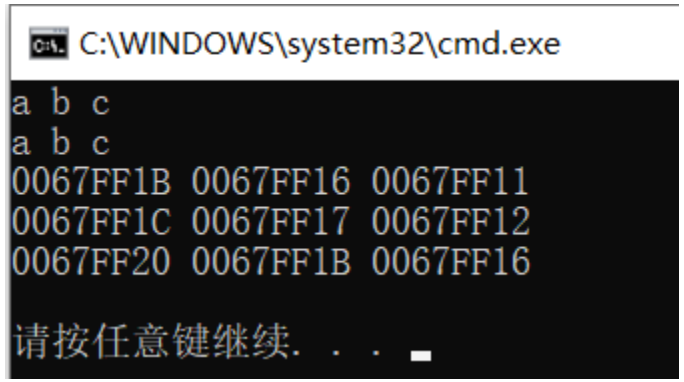
IO - scanf

（3）scanf函数中的输入项如果是字符数组名。不要再加地址符&，因为在C语言中数组名代表该数组的起始地址。下面写法不正确：

scanf(" %s" , &str);

```
#include <stdio.h>
int main()
{
    char str1[5], str2[5], str3[5];
    scanf("%s %s %s", str1, str2, str3);
    printf("%s %s %s\n", str1, str2, str3);
    printf("%p %p %p\n", str1, str2, str3);
    printf("%p %p %p\n", str1 + 1, str2 + 1, str3 + 1);
    printf("%p %p %p\n", &str1 + 1, &str2 + 1, &str3 + 1);
    return 0;
}
```

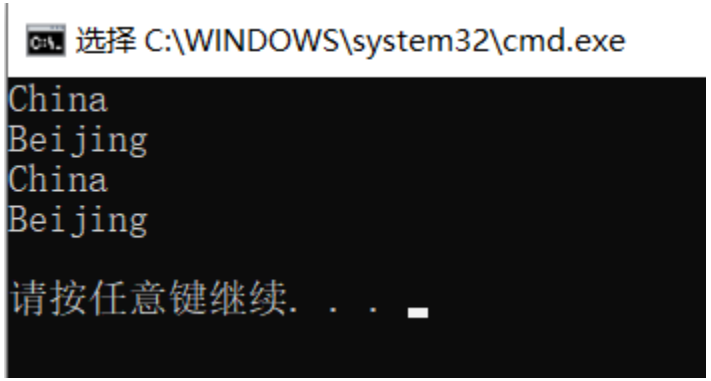
格式控制符“%p”中的p是pointer（指针）的缩写。printf函数族中对于%p一般以十六进制整数方式输出指针的值。



IO - puts

其作用是将一个字符串(以'\0'结束的字符序列)输出到终端。

```
#include <stdio.h>
int main()
{
    char str[] = {"China\nBeijing"};
    puts(str);
    puts(str);
    return 0;
}
```

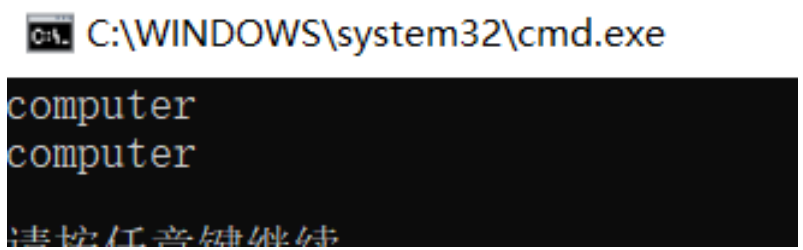


在输出时，将字符串结束标志'\0'转换成'\n'，即输出完字符串后换行。

IO - gets

其作用是从终端输入一个字符串到字符数组，并且得到一个函数值。该函数值是字符数组的起始地址。

```
#include <stdio.h>
int main()
{
    char str[10];
    gets(str);
    puts(str);
    return 0;
}
```



函数值为字符数组str的起始地址。一般利用gets函数的目的是向字符数组输入一个字符串，而不大关心其函数值。

注意： 用puts和gets函数只能输入或输出一个字符串，不能写成puts(str1，str2) **或** gets(str1，str2)!!

String operations

```
#include <string.h>
```

C supports a wide range of functions that manipulate strings.

Operators	Description	Example s1=A, S2 = B;
strcpy(s1, s2)	Copy s2 into s1	s1 = B
strcat(s1, s2)	Concatenate s1 and s2	S1 = AB
strlen(s1)	Return length of s1	Length = 1
strcmp(s1, s2)	Compare s1 and s2	A<B, return -1
strlwr(s1)	Convert s1 to lower case	A to a
strupr(s1)	Convert s1 to upper case	a to A

strcpy(s1, s2)

```
char str1[12] = "Hello";
char str2[12] = "World";
char str3[12];


strcpy(str3, str1);
printf("%s\n", str3); //Hello

strcpy(str3, str2);
printf("str3 = %s\n", str3); //World
```

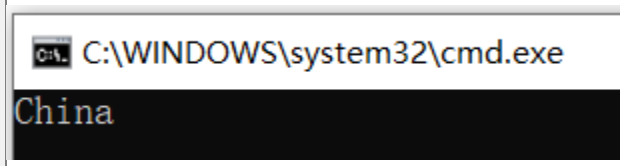
strcpy(s1, s2)

（1）字符数组1必须定义得足够大，以便容纳被复制的字符串。字符数组1的长度不应小于字符串2的长度。

（2）“字符数组1”必须写成数组名形式(如str1)，“字符串2”可以是字符数组名，也可以是一个字符串常量。如strcpy(str1，″China″);

（3）复制时连同字符串后面的′\0′一起复制到字符数组1中。

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[10] = {"abc"}, str2[] = {"China"};
    strcpy(str1, str2);
    puts(str1);
    return 0;
}
```



strcpy(s1, s2)

（4）可以用strcpy函数将字符串2中前面若干个字符复制到字符数组1中去。例如:strcpy(str1，str2，2);

作用是将str2中前面2个字符复制到str1中去，然后再加一个‘\0’。

（5）不能用赋值语句将一个字符串常量或字符数组直接给一个字符数组。字符数组名是一个地址常量，不能改变值，如：

str1=" China" ;　不合法

str1=str2;　　不合法

- 用strcpy函数只能将一个字符串复制到另一个字符数组中去。
- 用赋值语句只能将一个字符赋给一个字符型变量或字符数组元素。
- 复制的字符串的字符个数n不应多于str1中原有的字符(不包括'\0')

strcat(s1, s2)

```
char str1[12] = "Hello";
char str2[12] = "World";
char str1[12] = "123";

strcat(str1, str2);
printf("str1 = %s\n", str1); //HelloWorld

strcat(str3, str2);
printf("str3 = %s\n", str3); //123World
```

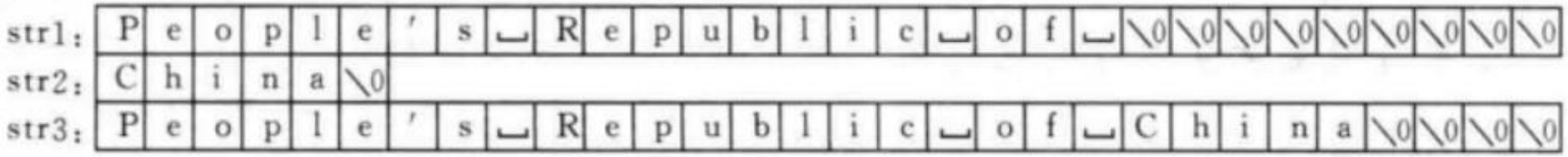
strcat(s1, s2)

```
#include <string.h>
#include <stdio.h>
int main(void)
{
    char str1[30] = {"People's Republic of "};
    char str2[] = {"China"};
    strcat(str1, str2);
    printf("%s", str1);
    return 0;
}
```

其作用是把两个字符数组中的字符串连接起来，把字符串2接到字符串1的后面，结果放到字符数组1中，函数调用后**得到字符数组1的地址**

C:\WINDOWS\system32\cmd.exe

People's Republic of China



（1）字符数组1必须足够大，一边容纳连接后的新字符串；

（2）连接前两个字符串后面都有'\0'，连接时，字符串1后面的'\0'取消，只在新字符串最后保留'\0'

strlen(s1)

```
char str1[12] = "Hello";
char str2[] = "World";
char str3[12];

printf("str1 = %d\n", strlen(str1)); //5

printf(“str2 = %d\n”, strlen(str2)); //5

printf("str3 = %d\n", strlen(str3)); //0
```

测试字符串长度的函数。函数的值为字符串中的实际长度(不包括′\0′在内)。

sizeof(s1)

```
char str1[12] = "Hello";
char str2[] = "World";
char str3[12];

printf("str1 = %d\n", sizeof(str1)); //12

printf(“str2 = %d\n”, sizeof(str2)); //6, end with ‘\0’

printf("str3 = %d\n", sizeof(str3)); //12
```

strcmp(s1, s2)

```
char str1[] = "ABCD";
char str2[] = "BCD";
char str3[] = "ABCE";
char str4[] = "1234";

printf("cmp = %d\n", strcmp(str1, str2));// -1

printf("cmp = %d\n", strcmp(str1, str3));// -1

printf("cmp = %d\n", strcmp(str1, str1));// 0
```

str1 > str2 → 1
str1 < str2 → -1
str1 = str2 → 0

若出现不相同的字符，以第1对不相同的字符的比较结果为准

strcmp(s1, s2)

Application scene:

```
if (strcmp(s, "+") == 0) {
        c = a + b;
} else if (strcmp(s, "-") == 0) {
        c = a - b;
} else if (strcmp(s, "*") == 0) {
        c = a * b;
} else if (strcmp(s, "/") == 0) {
        c = a / b;
} else {
        printf("ERROR");
        return 0;
}
```

It's useful when we have different parameters.

It is a very simple calculator.

run L08_strcmp.c

you can write code with much complexity

strlwr(s1)

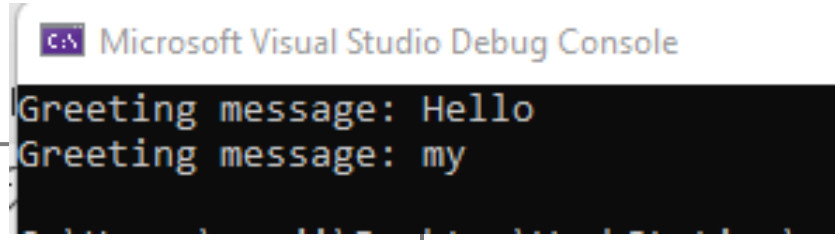
```
char str1[] = "ABCD";
char str2[] = "abcd";
char str3[] = "012abcDE";


printf("strupr  =%s\n", strlwr(str1)); // abcd

printf("strupr  =%s\n", strlwr(str2)); // abcd

printf("strupr  =%s\n", strlwr(str3)); // 012abcde
```

strupr(s1)

```
char str1[] = "ABCD";
char str2[] = "abcd";
char str3[] = "012abcDE";


 printf("strupr =%s\n", strupr(str1)); // ABCD

printf("strupr =%s\n", strupr(str2)); // ABCD

printf("strupr =%s\n", strupr(str3)); // 012ABCDE
```

Case study: dictionary

Case: can we create a sentence?



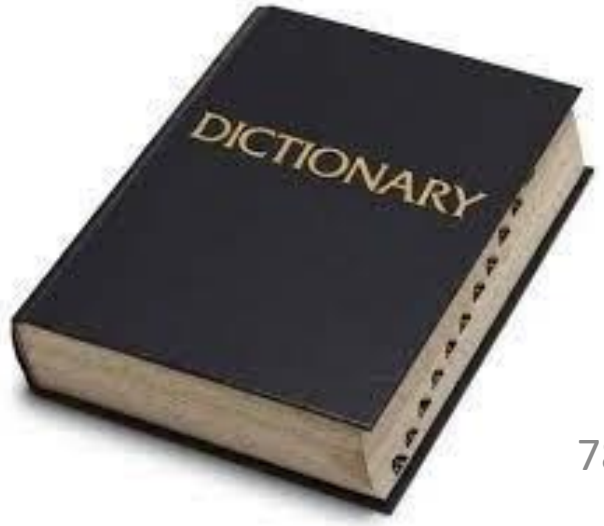

```c
#include <stdio.h>

int main(void)
{
   char greeting[10] = "Hello";
   char greetings[3][10] = { "Hello", "my", "friend" };

   printf("Greeting message: %s\n", greeting);
   printf("Greeting message: %s\n", greetings[1]);
}
```

Case study: dictionary

Case: can we create a simple dictionary?

```
#include <stdio.h>
#include <string.h>
int main(void) {
    char EngWords[][8] = { "apple", "orange", "banana" };
    char ChineseWords[][8] = { "苹果", "橘子", "香蕉"};
    char text[128];
    while (gets(text)) {
        for (int i = 0; i < 3; i++) {
            if (strcmp(text, EngWords[i]) == 0) {
                printf("%s 中文为: %s\n", text, ChineseWords[i]);
                break; }
            else if (strcmp(text, ChineseWords[i]) == 0) {
                printf("%s 英文为 %s\n", text, EngWords[i]);
                break;}
        if (strcmp(text, "exit") == 0)  break;
    }
}
```

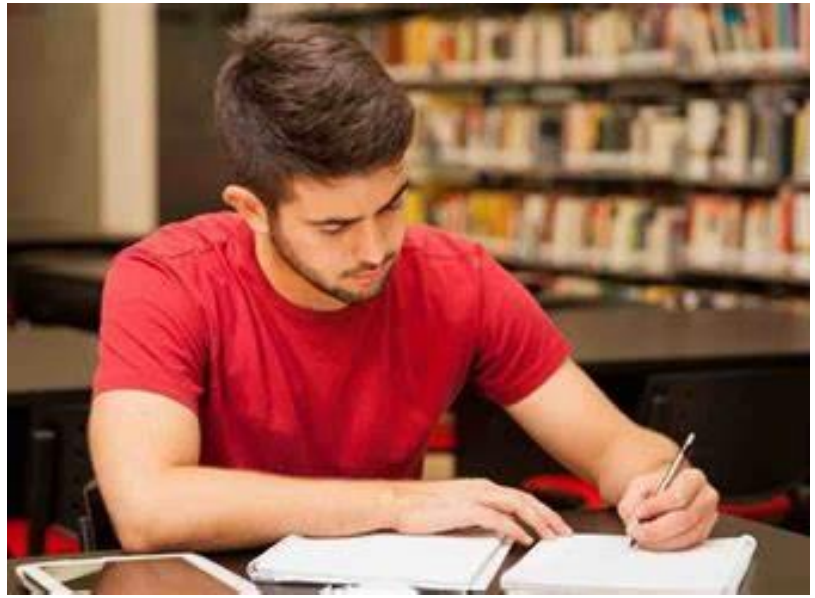
```
Z:\Courses\CS111\Code\L08_dict.exe
apple
apple 中文为: 苹果
香蕉
香蕉 英文为 banana
orangle
orange
orange 中文为: 橘子
exit

--------------------------------
-
Process exited after 34.49 seco
nds with return value 0
```



run L08_dict.c, L08_dict2.c

Case study: student's information

```
#include <stdio.h>
#include <string.h>

int main(void) {
    char stu_name[][8] = { "张三", "李四", "王五" };
    char stu_id[][8] = { "111", "112", "113"};
    char text[128];
    while (gets(text)) {
        for (int i = 0; i < 3; i++) {
          if (strcmp(text, stu_name[i]) == 0) {
              printf("%s 学号为: %s\n", text, stu_id[i]);
               break; }
          else {printf(" % s 不在名单当中! \n", text);
             break; }
        }
     }
     return 0;
}
```

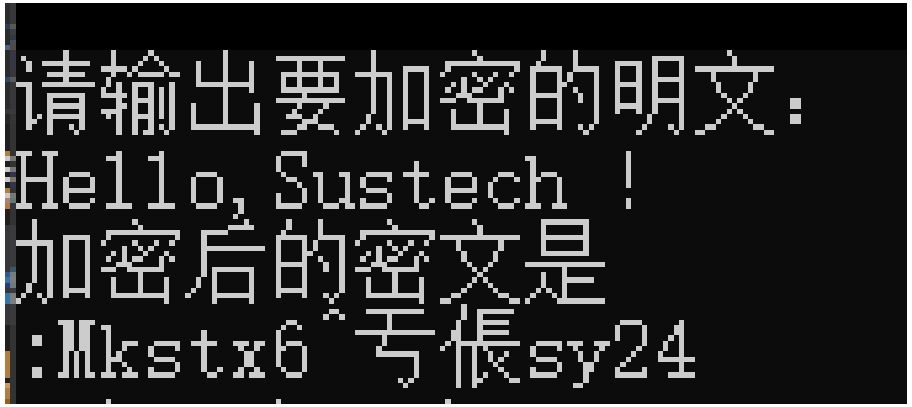


run L08_studentIF.c

Case study: encryption

Case: can we encrypt a message?

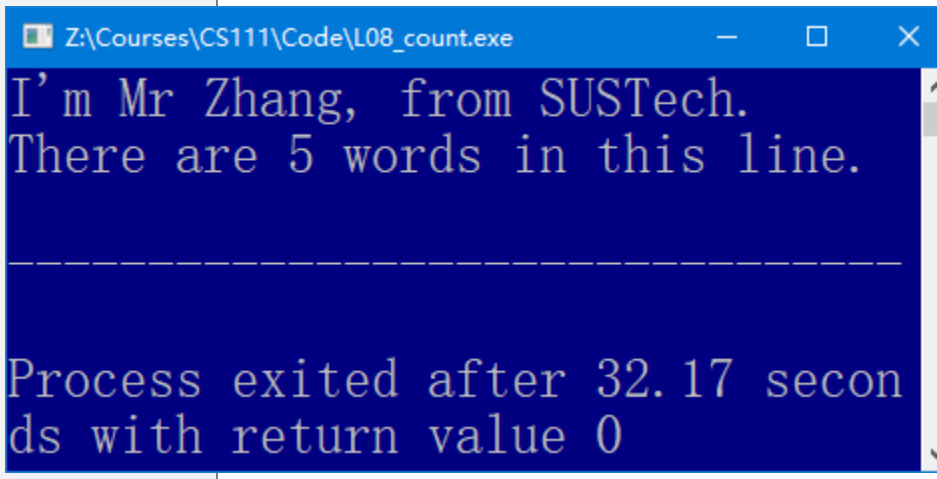
```
#include <stdio.h>
#include <string.h>

int main(void) {
        char text[128] = { '\0' };
        char cryptograph[128] = { '\0'};
        printf("请输出要加密的明文: \n");
        gets(text);
        int count = strlen(text);
        for (int i = 0; i < count; i++) {
                cryptograph[i] = text[i] + i + 5;
        }
        cryptograph[count] = '\0';
        printf("加密后的密文是\n:%s", cryptograph);
}
```



Case study: count

Case: can we count how many words there are?

```
#include <stdio.h>
int main(void){
    char string[81];
    int i, num = 0, word = 0;
    char c;
    gets(string);
    for (i = 0; (c = string[i]) != '\0'; i++)
        if (c == ' ')
            word = 0;
        else if (word == 0)
            {
                word = 1;
                num++;
            }
        printf("There are %d words in this line.\n", num);
        return 0;
}
```



check the variable **word** with L08_count.c

const

有时程序**只需要从数组中读取数值**，但是程序不向数组中写数据。在这种情况下，声明并初始化数组时，建议使用关键字**const**

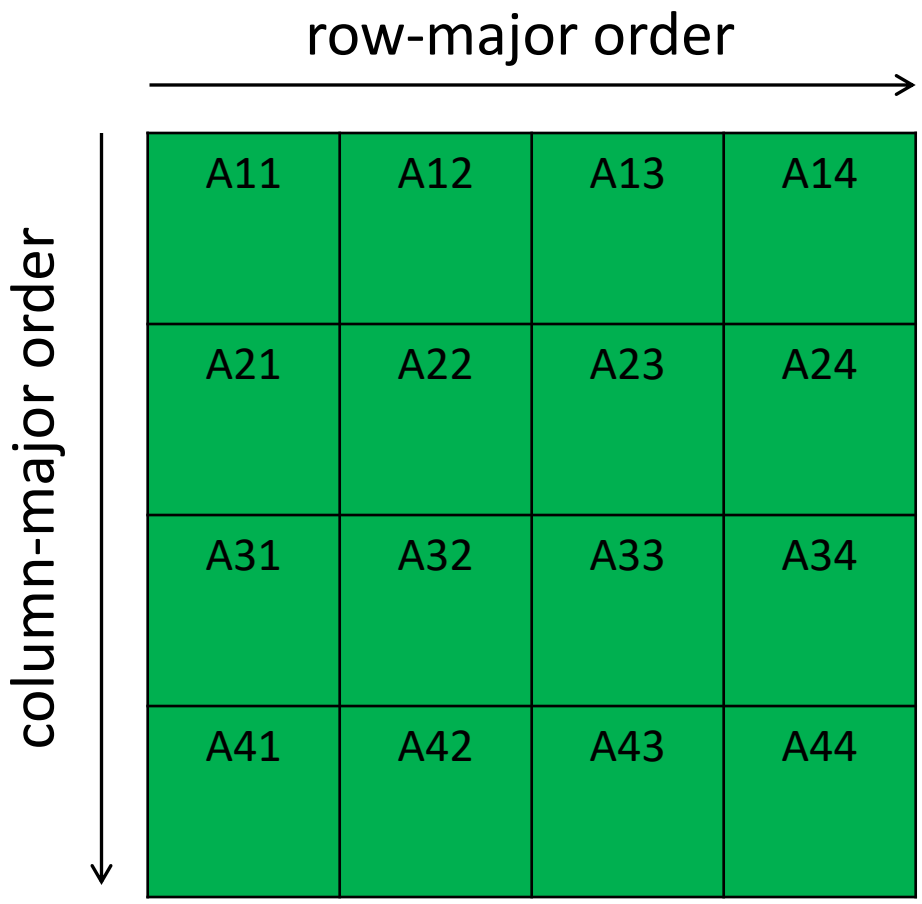
```
const int days[MONTHS] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

程序会把每个元素当成常量来处理，**在声明的时候需要对其初始化**，因为初始化后，不能再对它赋值。

Content

1. 1-D array
2. 2-D and N-D array
3. String
4. **Row-major or column-major order**

Row-major order or column-major order?



对于C语言来说，访问二维数组的顺序不同，时间消耗也是不同的。行优先遍历和列优先遍历进行对比，行优先更佳。

接下来通过C语言访问一个二维数组赋值操作来说对比优先与列优先的时间消耗差异，并给出相应的代码例子。

Row-major order or column-major order?

C语言按照行列优先顺序耗时差异在数组较小时体现不出来，当我们把二维数组大小设置为800x800时才体现出两者的操作耗时差异。

在接下来的例子中将二维数组大小设置为1300x1300。

Row-major order or column-major order?

```
#include <stdio.h>
#include <time.h>
#define N  1300
int a[N][N] = {0};
int main(){
int i, j;
    double t1, t2, t3;
    t1 = clock();
    for(i = 0; i < N; i++)
        for(j = 0; j < N; j++){
            a[i][j] = 1;
        }
    t2 = clock();
    for(i = 0; i < N; i++)
        for(j = 0; j < N; j++) {
            a[j][i] = 1;
        }
    t3 = clock();

    printf("Time used for row:    %10.6lf second(s)!\nTime used for column: %10.6lf second(s)!\n", \
            (t2-t1)/CLOCKS_PER_SEC, (t3-t2)/CLOCKS_PER_SEC);
    return 0;
}
```

通过宏定义数组大小为NxN

按行优先顺序给数组赋值

按列优先顺序给数组赋值

Row-major order or column-major order?

输出结果：

```
Time used for row:      0.010000 second(s)!
Time used for column:   0.020000 second(s)!
```

通过给1300X1300大小的整型数组赋值操作，
可以看出列优先顺序耗时为行优先顺序的两倍！

Row-major order or column-major order?

为了使得按行优先顺序和列优先顺序访问二维数组耗时数据比较稳定，可以多次执行这两个操作，并计算出两者耗时执行多次的平均值。

Row-major order or column-major order?

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N  1300
int a[N][N] = {0};
double one_time_operation(int flag){
    int  i, j;
    double t1, t2, t0;
    t1 = clock();
    if(flag == 0)
        for(i = 0; i < N; i++)
            for(j = 0; j < N; j++)
                a[i][j] = 1;
    else
        for(i = 0; i < N; i++)
            for(j = 0; j < N; j++)
                a[j][i] = 1;
    t2 = clock();
    t0 = (t2 - t1) / CLOCKS_PER_SEC;
    return t0;
}
```

定义并实现用于统计行单次对二维数组进行列优先操作所用时间的函数

注：大数组放在main之外。

内部：栈（小）

全局：数据区（大）

Row-major order or column-major order?

代码续

```
double average_time_used(int n, int flag){
    double t = 0.;
    int i;
    for (i = 0; i < n; i ++)
        t += one_time_operation(flag);
    t /= n;
    return t;
}
```

定义并实现用于统计行n次对二维数组进行列优先操作所用时间的平均值的函数

Row-major order or column-major order?

代码续

```
int main(int argc, char *argv[]){
    if(argc != 2) {
        printf("Usage: %s n\n", argv[0]);
        exit(1);
    }
    int n = atoi(argv[1]);
    double t1, t2;
    t1 = average_time_used(n, 0);
    t2 = average_time_used(n, 1);
    printf("%d %lf %lf\n", n, t1, t2);
    return 0;
}
```

主函数，命令行参数为接收统计行列优先操作的统计次数

Row-major order or column-major order?

统计1000次的平均耗时结果：

```
Time used for row:      0.004050 second(s)!
Time used for column:   0.011280 second(s)!
```

从两者耗时的平均值统计结果可以看出存在数量级的差异，行优先耗时比列优先第一个数量级

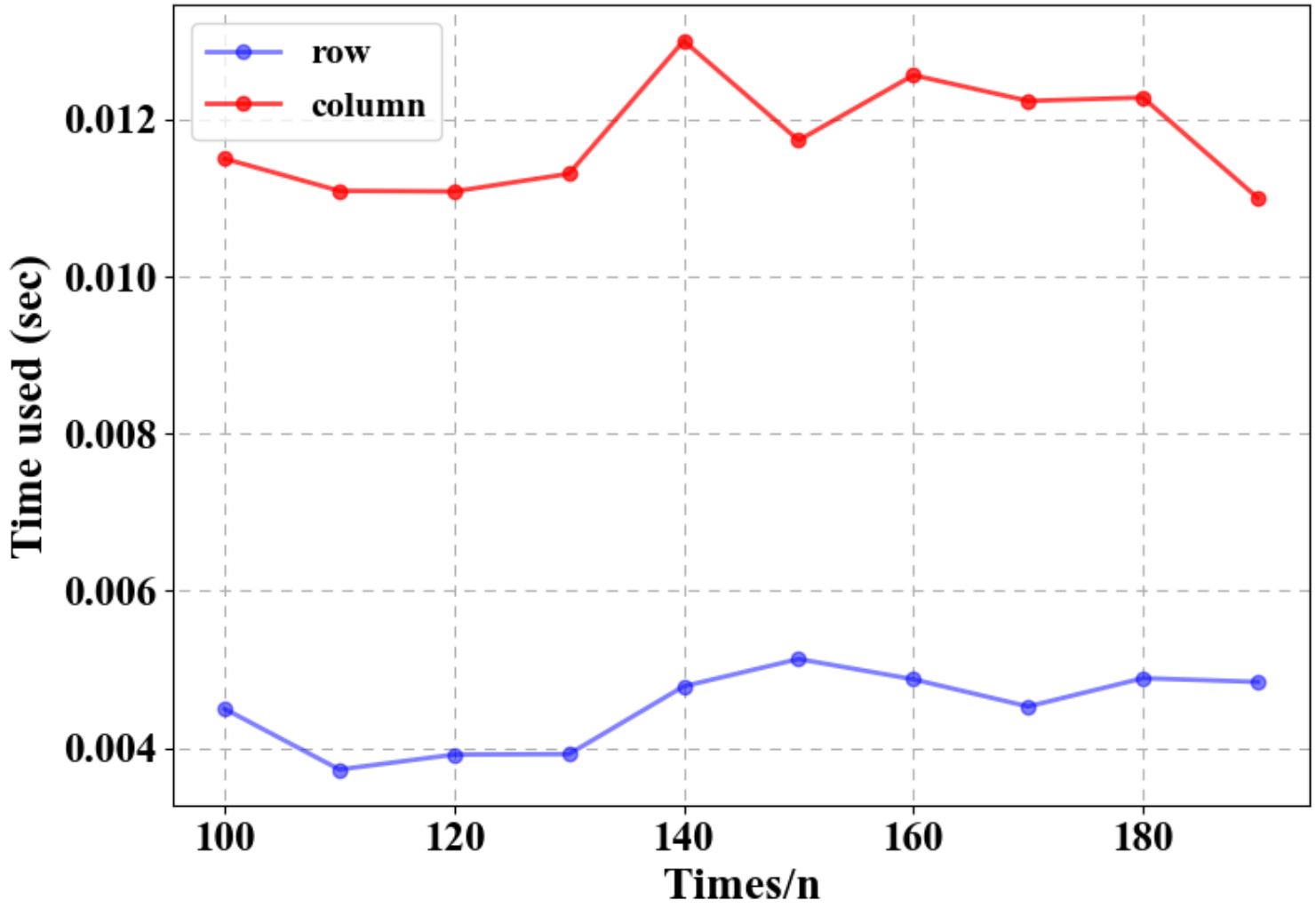
Row-major order or column-major order?

现统计不同次数情况下行列优先耗时的平均结果：

次数n	行优先(s)	列优先(s)
100	0.004500	0.011500
110	0.003727	0.011091
120	0.003917	0.011083
130	0.003923	0.011308
140	0.004786	0.013000
150	0.005133	0.011733
160	0.004875	0.012563
170	0.004529	0.012235
180	0.004889	0.012278
190	0.004842	0.011000

Row-major order or column-major order?

现统计不同次数情况下行列优先耗时的平均结果：



从图中可看出按行优先顺序执行二维数组幅值操作所耗时间稳定地低于按照列优先顺序的结果

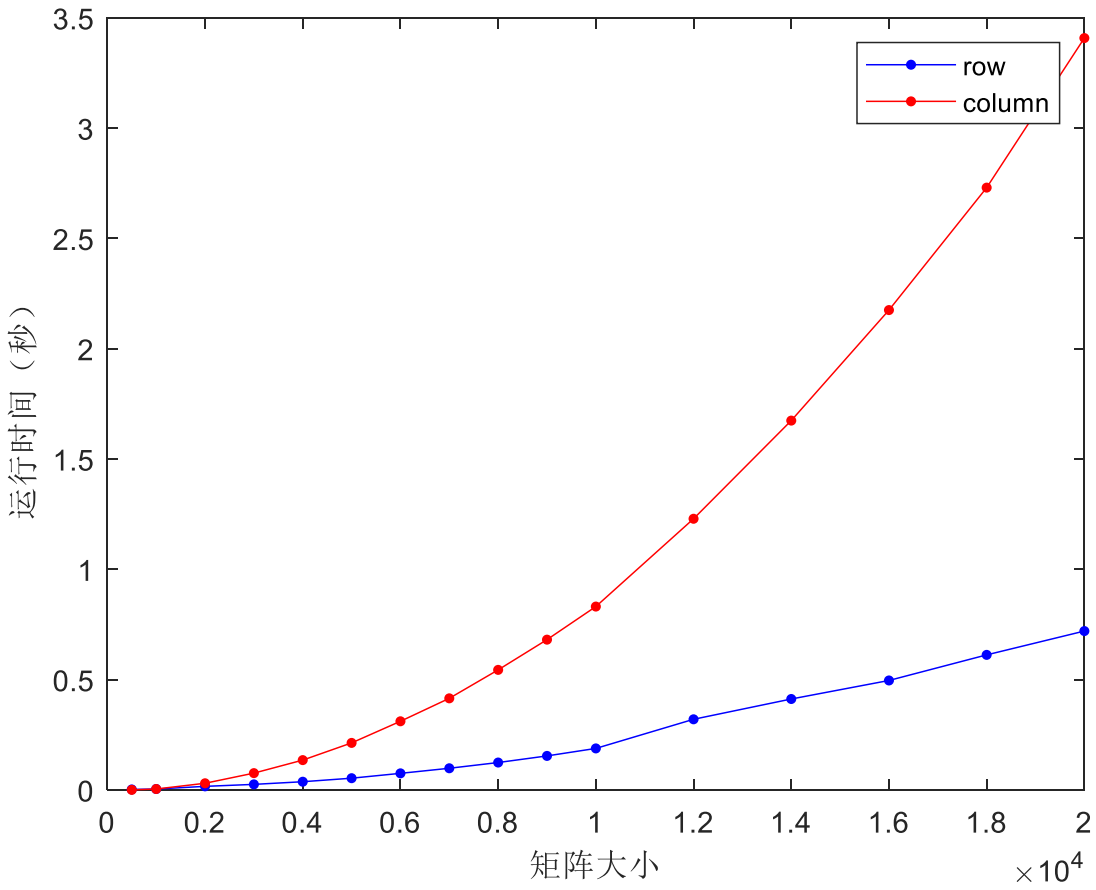
Row-major order or column-major order?

现统计不同矩阵大小【n】情况下行列优先耗时的结果：

大小	行优先(s)	列优先(s)
500	0.002	0
1000	0.004	0.004
2000	0.012	0.03
3000	0.025	0.076
4000	0.037	0.135
5000	0.053	0.213
6000	0.075	0.311
7000	0.098	0.415
8000	0.124	0.544
9000	0.154	0.681
10000	0.188	0.831
12000	0.320	1.229
14000	0.412	1.674
16000	0.496	2.175
18000	0.612	2.730
20000	0.72	3.408

Row-major order or column-major order?

现统计不同矩阵大小下行列优先耗时的平均结果：



从图中可看出按行优先顺序执行二维数组赋值操作所耗时间稳定地低于按照列优先顺序的结果

Row-major order or column-major order?

与其它语言的比较：

虽然C/C++采用行优先顺序原则，但像Fortran语言则采用列优先顺序，Matlab早期作为Fortran库的封装，因此其采用了列优先原则。这给我们的启示是在选择不同编程语言操作多维数组时，应遵循他们各自的行列优先原则，从而提高计算效率。

Summary

- We can use **array** to hold may data for group processing
- Array has the **fixed size** and can only be used to hold data with **same type**
- **Different types of array** can be created, e.g. int array, float array, char array (string)
- **Different dimensional array** can be created, from 1D array to ND array
- Array enables the processing of **vectors, matrices, strings**, etc.
- Row-major order or column-major order