计算机程序设计

Computer Programming

数组

主讲：吴锋

目录
CONTENTS

一维数组
- 声明、引用、初始化

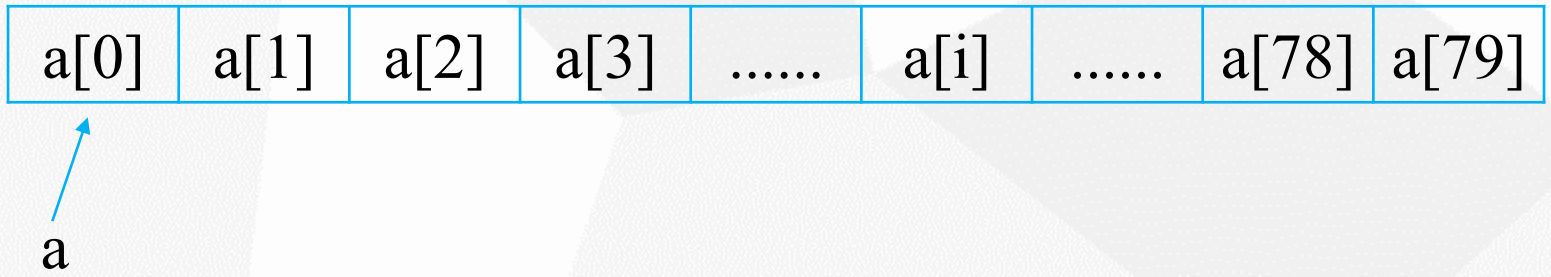
字符数组
- 声明、初始化、常用函数

多维数组
- 声明、存储、初始化

数组的应用

数组的概念

- 数组（Array）是一种构造型数据类型，是具有相同类型的数据的集合，在内存中连续有序地排列
- 数组类型的目的是为了用一个名字方便地处理同类数据
  - 例如，一个班级80人的成绩，不能设80个变量，a1,a2, a3,…,a80，为每个变量写一段处理代码，程序会很繁琐。用数组，可以声明为a[80]，使用80个连续的存储单元存储成绩，使用a[i]表示其中第i个人的成绩，使用相同的代码处理所有数据，非常简捷



◎ 数组的声明

- 与一般变量一样，数组必须先声明后使用
- 声明一维数组的一般形式：

  数组元素类型 数组名[整型常量表达式];

- 例
  - int a[10];
    //表示数组名是a，由10个元素构成a[0]~a[9]
    //a的元素的类型是整型，每个元素可以被视为一个变量来使用
    //名字a的类型是10个整型数构成的数组
  - long int array[3*4];
    //表示array是由3*4=12个长整型数构成的数组

◎ 数组的声明

数组声明时应注意：

- 数组名是标识符，应遵循标识符的取名规则
- 数组名后的方括号[ ]是数组的标志
- 方括号内是整型常量表达式，表示该数组包含元素的个数，即数组的大小。下标从0开始
- 定义数组时，一般应同时指定数组的大小，且不能使用变量
  - C99允许使用变量来定义数组大小。执行到声明语句时，根据表达式的值来分配空间。但为了兼容性，不建议这么做

◎ 数组元素的引用

- C语言中，数组的使用是通过对单个元素的引用实现的
- C语言不支持把数组当作整体进行运算
  - 可以借助结构体进行整体赋值
- 引用一维数组的一般形式

  数组名[下标]

  下标可以是整型常量或整型表达式，可以使用变量

例：float a[10]; a[0]=a[1]+a[2*3]-a[8%6];

例：int x=6, y[10];
y[x]=18;

◎ 数组的初始化

- 方式一：与普通变量一样，逐个元素赋值
  - 例，a[0]=1; a[1]=5;
  - 例，for (i=0; i<80; i++) a[i]=0;
  - 例，scanf("%d",&a[3]); scanf("%d",a+3);

- 方式二：在声明时进行整体的初始化
  - 例，int a[6]={ 10, 20, 30, 40, 50, 60 };

- 不能在一般的语句中通过数组名进行整体赋值

◎ 数组的初始化

整体初始化时应注意：

- 初值数据项的个数不能多于数组元素的个数，但允许少于数组元素的个数，此时，依次从初值列表中取值对前面的数组元素赋值，而其余的数组元素隐含用0赋初值

  例：int a[5]={1,2}; 等价于 int a[5]={1,2,0,0,0};

- 在定义数组时若同时赋初值，且不指定数组的具体长度，则编译器会根据初值列表中的数据个数确认数组中元素的个数

  例：int a[]={1,2,3,4,5}; 等价于 int a[5]={1,2,3,4,5};

- 初值类型与数组元素类型不一致时，会把初值类型转换为数组元素的类型

数组的初始化

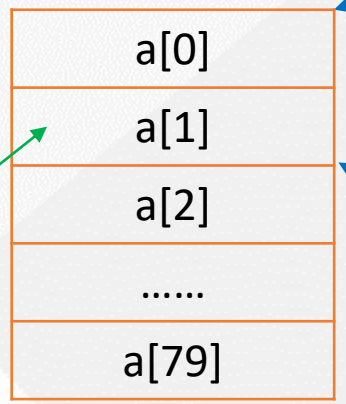
- C99允许指定初始化，将指定的元素赋值，其它为0
  - 例如 int a[6]={[5]=50}; 等价于 int a[6]={0,0,0,0,0,50};
  - 此时赋值的顺序可以任意
  - 例如 int a[15]={ [2]=29, [14]=7, [9]=48};

- 混合初始化
  - 例如 int a[]={1, 2, 3, [4]=5, 6, 7, [9]=10};
    等价于 a[10]={1, 2, 3, 0, 5, 6, 7, 0, 0, 10};

◎ 数组在内存中的存储

- 在函数被调用的时候，数组被分配存储空间，如有初始化也同时完成

数组名a表示数组第一个元素在内存中的起始地址（也叫基地址）



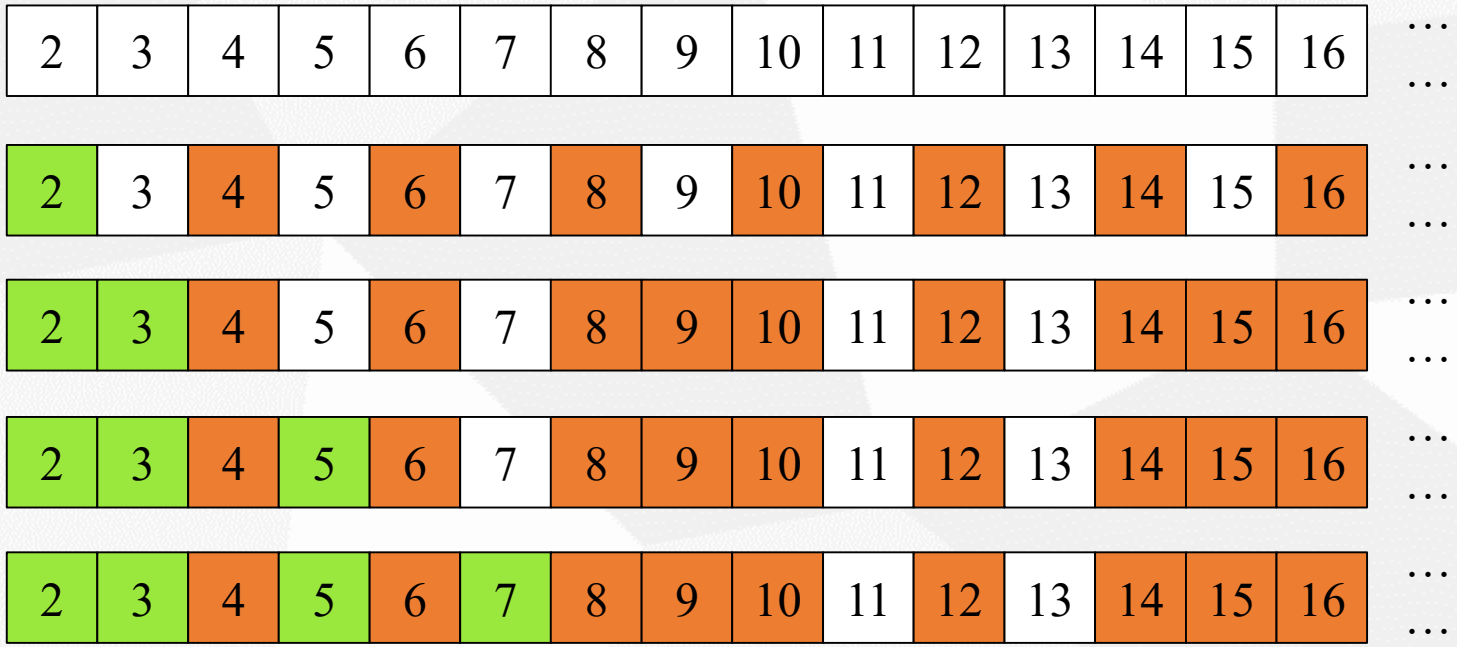
a+i指向第i个元素的起始地址，具体大小与数组元素类型有关

a[i]是具体的一个元素，与变量一样，是某个存储单元的名字

- 若需要计算数组元素个数，sizeof(array) / sizeof(int)与 sizeof(array) / sizeof(array[0])哪个更好？
- 都是编译时计算，速度一样。后者可以在不修改这条语句的情况下，修改数组array的类型

◎ 数组在内存中的存储（举例）

- 用筛选法求2-N(N<1000)之间的所有素数
  - 筛选法，从2开始，从小到大反复进行：找到一个数，它就是素数。然后删除所有它的倍数



◎ 数组在内存中的存储（举例）

- 用筛选法求2-N(N<1000)之间的所有素数

```
#include <stdio.h>
void main() {
    int n=1000,i,j;
    char mark[1001]={0}; //用输出化0做为初始状态
    for (i=2; i<=n; i++)
        if (!mark[i]) {
            printf("%6d", i);
            for (j=i+i; j<=n; j+=i) mark[j]=1;
        }
}
```

◎ 字符数组

- 字符串是一个有限长度的字符序列
  "abcdef" "asdf1234"
- C语言中，字符串并非专门的数据类型，通常用字符数组存放
- 字符数组即元素类型为字符的数组
- 字符串的长度是其中包含的有效字符数加1（结尾隐含字符 '\0'）
- 字符数组如以 '\0' 结尾，则称之为字符串

"abcdef"

a	b	c	d	e	f	\0

在内存中的表示

◎ 字符数组

字符串常量的表示

- 一对双引号括起来的字符序列
  - 引号内可以包括0个、1个或多个字符
  - 字符串中的特殊字符用转义字符表示
  - 举例，`"abc"`、`"a"`、`""`、`"␣"`、`"我"`、`"12.4"`、`"$#*!"`、`"C:\\windows"`
- 字符常量和字符串常量是不同类型的数据
  - 例如，`'a'`与`"a"`是不同的
  - 不能把一个字符串赋值给一个字符型变量
- C编译器会把连续多个字符串看作是一个字符串
  - 例如,"abc""def"等价于"abcdef"

◎ 字符数组初始化

- 方式一：与普通变量一样，逐个元素赋值
  - char name[8];
    name[0]='a'; name[1]='b'; .....; name[5]='f';
    name[6]=name[7]='\0';
  - char name[8]={'a','b','c','d','e','f','\0'}; //少一个'\0'？
- 方式二：在声明时进行整体的初始化
  - char name[8]={"abcdef"};
  - char name[8]="abcdef";
  - char name[ ]="abcdef"; //此时字符数组大小与前不同！
- 方式三：利用格式化输入进行初始化
  - char name[10];
    scanf("%s", name); //注意name已是地址，无需再加"&"

◎ 字符数组初始化

- 注意如下初始化方式的差别

```
char c1[ ]={'C','h','i', 'n', 'a'};
char c2[ ]={"China"};
```

数组c1大小为5，数组c2大小为6（尾部隐含字符'\0'）

```
char t[ ]="abc", s[3]="abc";
```

数组t的大小为4，而数组s的大小为3 //少了字符'\0'!但语法没有错误

- 字符串操作很常见，因此字符数组往往需要加'\0'以便作为字符串使用
- 对字符数组初始化时，初值表中的初值个数不能大于数组的长度，否则使用时会有未知问题（编译器不一定报错）

◎ 常用的字符串函数

- 输入输出： gets(str)和puts(str)

```
char name[10];
gets(name); //输入一个字符串（按实际输入长度接收）
puts(name); //输出一个字符串
```

- 大小写转换函数： strupr(str)和strlwr(str)

使用函数前需要#include<string.h>

```
char lname[ ]="wang";
strupr(lname); //将字符串lname中的小写字母转换成大写字母，不改变其它字符
strlwr(uname); //只将大写字母转换成小写
```

◎ 常用的字符串函数

- 字符串转换成数值：atof(str)、atoi(str)、atoll(str)
  - 使用函数前需要#include<stdlib.h>

atof(str); //将字符串str转换成一个双精度的数值

atoi(str); //将字符串str转换成一个整型的数值

atol(str); //将字符串str转换成一个长整型的数值

```
double score;
int snum;
long sid;
char tmpstr[20];
gets(tmpstr); score=atof(tmpstr);
gets(tmpstr); snum=atoi(tmpstr);
gets(tmpstr); sid=atol(tmpstr);
```

◎ 常用的字符串函数

- 字符串连接函数：strcat(str1,str2)
  - 将str2添加在str1的后面
  - 合并后覆盖str1末尾的'\0'，只在结尾有一个'\0'
  - str1的空间要足够大，以容纳str2（以及那个'\0'）

```
#include<stdio.h>
#include<string.h>
void main() {
    char str1[20]="Hello ", str2[ ]="World!";
    printf("%s\n", strcat(str1, str2));
}
```

  - 执行结果是：Hello World!
  - 提示：所有str开头的函数，都需要#include<string.h>

常用的字符串函数

- 计算字符串长度函数strlen(str)
  - 求字符串str长度，不包括结束标志'\0'

```
#include<stdio.h>
#include<string.h>
void main() {
    char str1[ ]="Hello", str2[20];
    int charnum;
    charnum=strlen(str1);
    gets(str2);
    printf("%d, %d", charnum, strlen(str2));
}
```

  - 输入 abcdefglmn 时，执行结果是：5，10

◎ 常用的字符串函数

- 字符串比较函数strcmp(str1,str2)
  - 逐个比较两个字符串中字符的ASCII值，全部相等则得0，当出现第一个不同字符时，根据ASCII码值的差的符号决定比较结果

```
#include<stdio.h>
#include<string.h>
void main() {
    printf("%d, ", strcmp("Good", "Good");
    printf("%d, ", strcmp("Well", "Good");
    printf("%d\n", strcmp("Good", "Well");
}
```

输出结果：0,1,-1

  - 注意：两个字符串不能用关系运算符

◎ 常用的字符串函数

- 字符串复制函数strcpy(str1,str2)
  - 把一个字符串str2（常量或数组）复制到一个字符数组str1中（不能使用=赋值）

```
#include<stdio.h>
#include<string.h>
void main() {
    char str1[ ]="Hello,", str2[20];
    strcpy(str2, str1);
    strcpy(str2+6, "world!");
    printf("%s\n", str2);
}
```

  - 执行结果是：Hello,world!
  - 注意：str1的长度不应小于str2的长度！

常用的字符串函数

- 字符串复制函数strncpy(str1,str2,n)
  - 将源字符串str2中前面最多n个字符复制到目标字符数组str1中

```
#include<stdio.h>
#include<string.h>
void main() {
    char src[ ]="Hello,World!";
    char dest1[10]="", dest2[10]={0};
    strncpy(dest1, src, 5);                    //复制最左侧5个字符
    strncpy(dest2, src+strlen(src)-6, 6);  //复制最右侧6个字符
    printf("%s,%s\n", dest1, dest2);
}
```

  - 执行结果是：Hello,World!
  - strncpy不会自动添加 '\0' ,因此dest1和dest2都要初始化为 '\0'

◎ 字符数组（举例）

- 编写程序，实现对一段文本中特定的某个单词进行统计。

```
#include <stdio.h>
#include <string.h>
void main() {
    int i=0, sum=0, length;
    char word[20], temp[20]={0};
    char text[]="… …";
    printf("Input the word.\n"); gets(word);
    length=strlen(word);
    while (text[i]!='\0') {
        strncpy(temp, text+i, length);
        if (strcmp(word, temp)==0) {
            sum++;
            i+=length;
        }
        i++;
    }
    printf("The word %s appears %d times.\n",word,sum);
}
```

注意本程序没有处理空格。
一旦匹配成功，i已经向后移动了length，再加1是不稳妥的。如果单词间没有空格，就可能漏掉。严格应该是else i++;

字符数组

- 关于 '\0'
  - 空字符串""其实不空，里面隐含一个'\0'，所以其长度为1（与strlen函数不同），但打印时没有任何显示
  - '\0'是空字符，不显示，但确实是一个字符
  - 空格即'␣'是可显示的，与'\0'不是一回事
  - '\0'的ASCII码值是0， 空格'␣'的ASCII码值是32

- 字符串函数的安全问题
  - 部分strXXX函数是不安全的，可用strXXX_s代替
  - 这些函数不考虑字符数组的边界，如果字符数组中不含'\0'，会造成不确定的结果。

◎ 多维数组

- C将多维数组定义为元素也是数组的数组
- 多(n)维数组的声明

  数据类型 数组名[整型常量表达式1]...[整型常量表达式n]

  - 例：float score[3][5][30]; //可记录3个年级，每年级最多5个班，每班最多30人的成绩

- 二维数组的声明

  数据类型 数组名[整型常量表达式1][整型常量表达式2]

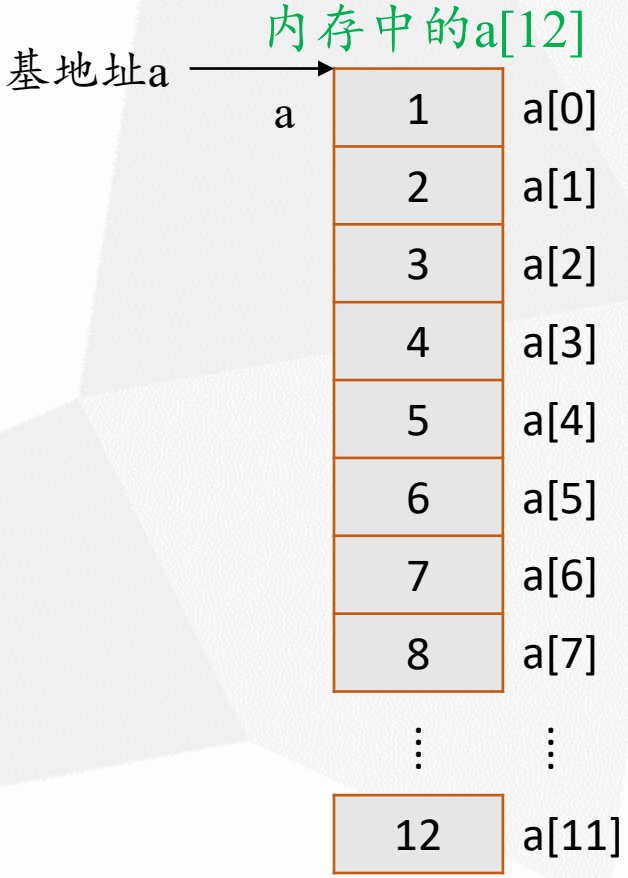
  - 例：int a[4][5]; //表示a为4*5(可视为4行5列)的整型数组
  - 例：char str[2*5][5]; //表示str为10行5列的字符数组

◎ 多维数组的内部表示

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

概念上的a[12]

1	2	3	……	11	12

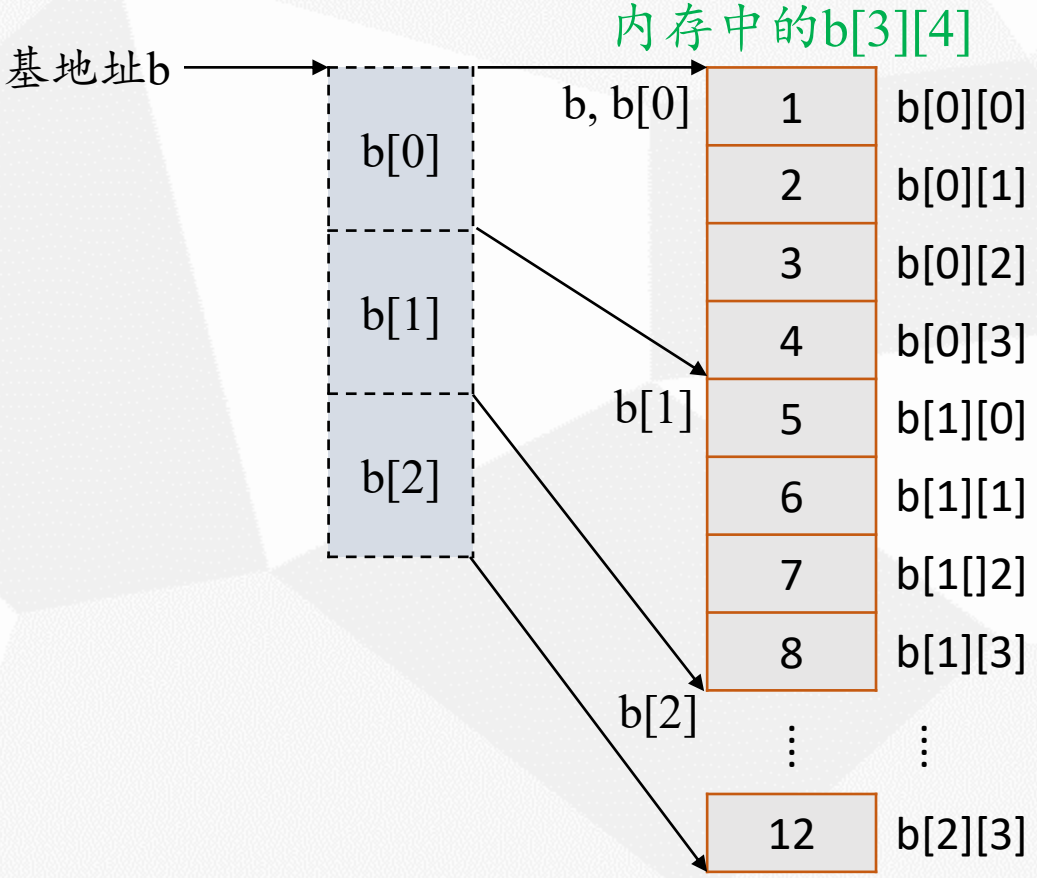


int b[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

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

概念上的b[3][4]

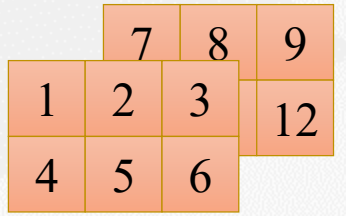
1	2	3	4
5	6	7	8
9	10	11	12



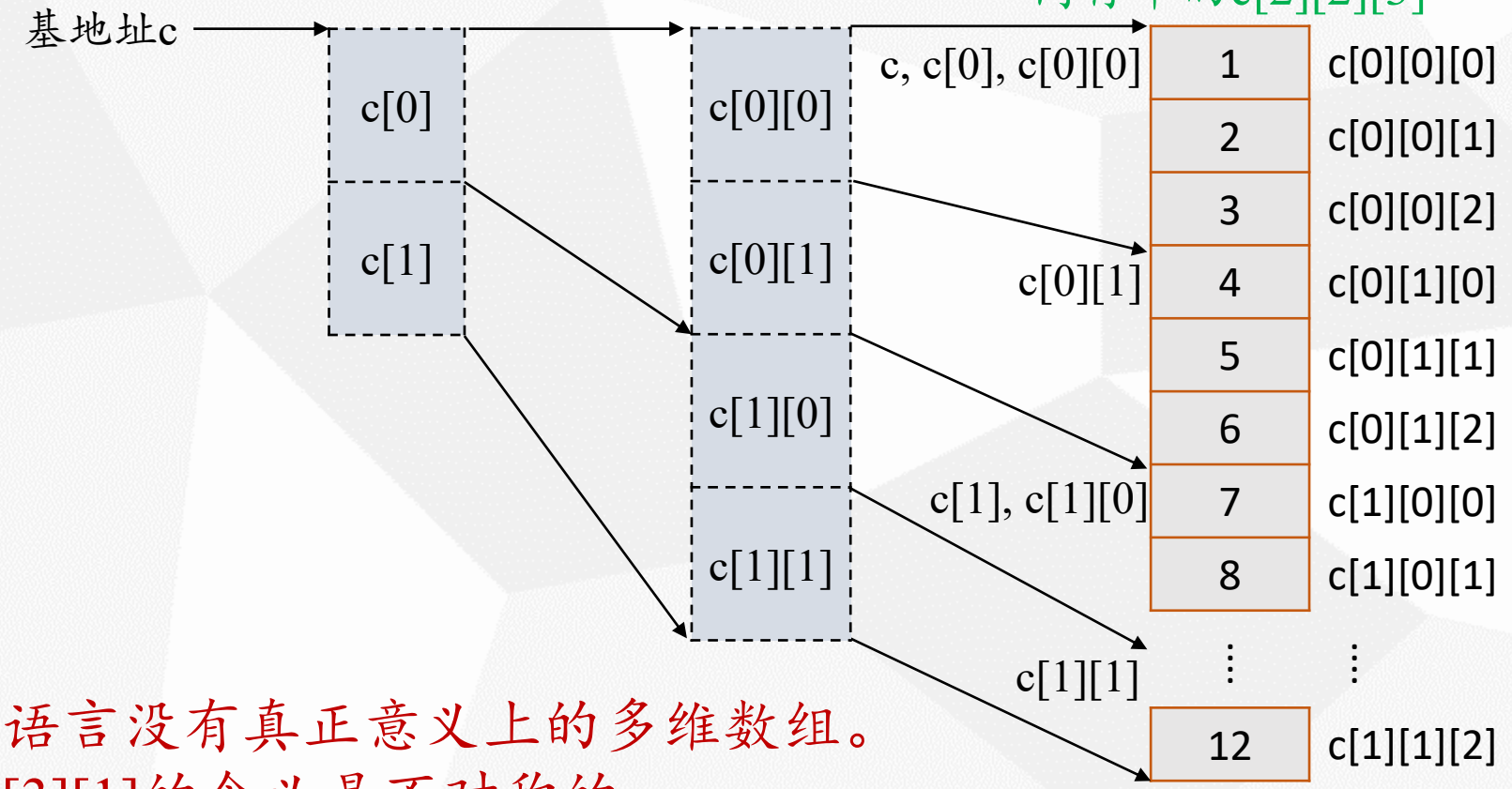
◎ 多维数组的内部表示

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

概念上的c[2][2][3]



内存中的c[2][2][3]



本质上，C语言没有真正意义上的多维数组。
C[1][3]和C[3][1]的含义是不对称的。

多维数组的内部表示

- 多维数组的内部表示
  - 有序地占用一片连续的内存区域
  - 数组的名字表示这片存储区域的首地址
  - 存储时，按照第n维先变化，第1维最后变化的“行优先”顺序存储
    - 例如二维数组int a[2][3]={{1,2,3},{4,5,6}};
    - 习惯上，将{1,2,3}视为第0行，{4,5,6}是第1行
    - 元素按行优先顺序存储，先存储1,2,3，再接着存储4,5,6
  - 二维数组适合用于矩阵的表示与存储
  - 下面主要以二维数组为例进行讨论

二维数组

- 二维数组的声明注意事项
  - 两对[]之间不能有空格（但很多编译器可识别）

    int a[5] [10]; //不合法
  - 每对[]内都必须有一个整型常量表达式(除第一对外...)

    int a[5][]; //不合法

    float b[][3]={{1.1, 1.2, 1.3}, {2.1, 2.2, 2.3}}; //合法
  - 关于整型常量表达式：

    char names['z'-'a'+1][10]; //合法

- 引用二维数组元素
  - 与一维数组类似，每个下标都是从0开始引用

    如：names[0][0], names[25][9]

二维数组的初始化

- 可以与一维数组初始化相同
  - 如：int a[2][3]={1,2,3,4,5,6}; //注意顺序

- 也可分行（维）赋初值
  - 如：int a[2][3]={ {1,2,3}, {4,5,6} };
    int a[3][2]={ {1,2}, {3,4}, {5,6} };

- 若定义二维数组的同时赋初值，则第一维的长度可以省去，但第二维的长度不能缺少
  - 如：float ff[][3]={{1.1,1.2,1.3},{2.1,2.2,2.3}};
    等价于 float ff[2][3]={{1.1,1.2,1.3},{2.1,2.2,2.3}};

二维数组初始化

- 可以只对部分元素赋初值，其余元素自动赋0
  - 如：int a[3][4]={ {1,2,3,4},{5,6,7}}; //注意只能缺失某一维的后面数据
- 可以只对部分元素赋初值，且省略第一维（行）的长度。应分行赋值，即使某行没有初值，也必须保留该行的一对{}
  - 如：int u[][4]={{1,2},{},{6,7,8,9},{10}};
    等价于 int u[4][4]={{1,2,0,0},{0,0,0,0},{6,7,8,9},{10,0,0,0}};
- 对二维字符数组可用字符串直接进行初始化，每个字符串对应一行数组元素
  - 如：char name[ ][10]={"invalid", "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"};

二维数组（举例）

- 编写矩阵相乘的程序
  - 矩阵A乘以B的条件：A的列数与B的行数相同
  - 计算规则：A的第i行的所有元素乘B的第j列对应的元素，并把结果相加，可得乘积矩阵C的第i行第j列的值
  - 设A为m行n列，B为r行v列，则n=r，且C为m行v列

$$c_{ij} = \sum_{k=0}^{n} a_{ik} \times b_{kj}$$

$$\begin{bmatrix} 5 & 8 & 3 \\ 11 & 0 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 18 \\ 2 & 11 \\ 10 & 3 \end{bmatrix} = \begin{bmatrix} 51 & 187 \\ 61 & 213 \end{bmatrix}$$

◎ 二维数组（举例）

矩阵相乘的程序

```
#include<stdio.h>
void main() {
    int a[2][3]={{5,8,3},{11,0,5}};
    int b[3][2]={{1,18},{2,11},{10,3}};
    int c[2][2],i,j,k,s;
    for(i=0; i<2; i++)   //A矩阵的行
        for(j=0; j<2; j++) {       //B矩阵的列
            for(k=s=0; k<3; k++) s+=a[i][k]*b[k][j];
            c[i][j]=s;
        }
    for(i=0; i<2; i++)
        for(j=0; j<2; j++)
            printf("%6d%c", c[i][j], ((j+1)%2==0)? '\n':' ');
}
```

上机作业

1. 输入两个字符串str1和str2，计算str2在str1中出现的位置（从0开始计算）。输出位置结果。例如：str1为“how are you!”，str2为“are”，那么输出4。

2. 约瑟夫问题：古代某法官要判决n名凡人死刑，他将犯人排成一个圆圈，然后从第s个人开始从1报数，每数到第m个犯人，就把他拉出来处决，然后再从1报数。到剩下最后一个人时，就把他赦免。编写程序，输入n、s、m，给出处决顺序，以及被赦免者编号。

3. 凯撒密码：凯撒生活在充满危险和阴谋的年代。为了生存，他首次发明了密码，用于军队的消息传递。假设你是凯撒军团中的一名军官，需要把凯撒发送的消息破译出来、并提供给你的将军。消息加密的办法是：对消息原文中的每个字母，分别用该字母之后的第5个字母替换（例如：消息原文中的每个字母A 都分别替换成字母F），其他字符不 变，并且消息原文的所有字母都是大写的。要求编程将输入的密文译回原文。

上机作业

4. 魔方阵：每一行、每一列和对角线之和均相等的方阵为魔方阵。例如三阶魔方阵如右图所示。要求编程输出元素为1~n^2（n为奇数的魔方阵）。

8	1	6
3	5	7
4	9	2

   - 提示：参考贾书第130页第21题的描述生成魔方阵。

5. 编写程序，生成一种贯穿10*10字符数组的随机步法。字符数组初始化为全“.”。程序随机地从一个元素“走到”另一个元素，每次可以向左、向右、向上、向下移动一个元素位置，不可越出边界。已访问过的元素按字母顺序标记为A到Z，不可再次走入。到达Z为一次成功行走。若4个方向都被堵住，则行走失败。输入初始位置，输出一次行走结果。

   - 提示：使用stdlib.h中的rand()函数来生成随机数，例如：rand()%4 输出0~3的随机数。为方便助教检查，要求以输入初始位置为随机种子。例如：输入初始位置为x行y列，则设置随机种子为 x+10*y，通过开始前调用 srand(x+10*y) 来完成。

```
A.........
BCD.......
.FE.......
HG........
I.........
J.......Z.
K..RSTUVY.
LMPQ...WX.
.NO.......
..........
```

上机作业

6. 编程分析一手5张牌的牌型，输出最好的牌型。每张牌有四种花色（方块Diamonds、梅花Clubs、红桃Hearts、黑桃Spades）和等级（2、3、4、5、6、7、8、9、10、J、Q、K、A），A即可作为最大等级，也可作为最小等级。牌型从大到小有：
   - 同花顺 Straight Flush（同花色且等级顺序相连）
   - 四张 Four of A Kind（4张牌等级相同）
   - 葫芦 Full House（3张牌等级相同，另2张牌也等级相同）
   - 同花 Flush（5张牌花色相同）
   - 顺子 Straight（5张牌等级顺序相连）
   - 三张 Three of A Kind（3张牌等级相同）
   - 两对 Two Pairs（存在两个对子）
   - 对子 Pair（2张牌等级相同）
   - 其他 Other

   输入时，花色用首字母表示，等级用数字或字母表示（10用T表示）。

   若输入非法牌或重复牌则丢弃。若输入为0则结束程序。