LECTURE 7 多维数组、编码

工学院18-19学年秋季学期计算概论（邓习峰班）课后辅导

讲师：陈婉雯

日期：2018/12/1

多维数组

- type name[size1][size2]...[sizeN];
- 二维数组：
- type arrayName [x][y];

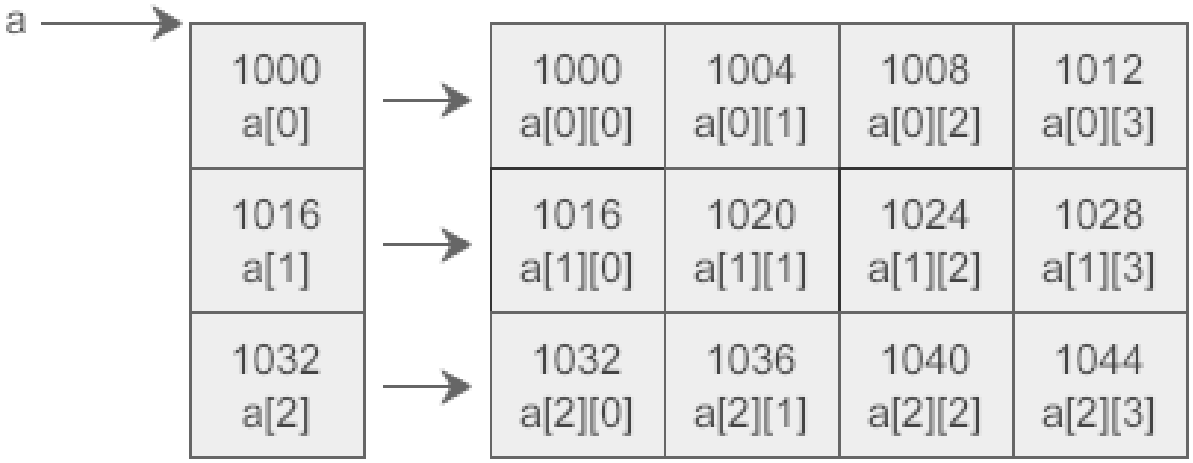
	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- 问题：arrayname是什么类型？
- type **？ ×

- 在C语言中，二维数组实际上是一种特殊的一维数组，它的每个元素也是一个一维数组
- type (*)[y] √

- 实际上并不存在多维数组，所谓的多维数组本质上是用一维数组模拟的。
- 二维数组在物理上是线性的，按行来依次进行存放，内存是连续的。
- 二维数组名的步长是一行的长度。
- a[i][j]=a+(i*n+j)*b

```
int a[3][4] = { {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };
```



初始化

- int matrix[3][6]={{1,3,5,7,9,11},{2,4,6,8,10,12},{3,5,7,11,13,17}};
- int matrix[3][6]={1,3,5,7,9,11,2,4,6,8,10,12,3,5,7,11,13,17};
- 还可以对部分元素赋初值，没有明确初值的元素清0：
  int matrix[3][6]={{1,3},{2,4},{3,5,7}};
  其结果等效于：
  int matrix[3][6]={{1,3,0,0,0,0},{2,4,0,0,0,0},{3,5,7,0,0,0}};

- 还可由初始化数据来确定数组最高维，如
  int matrix[][6]={1,3,5,7,9,11,2,4,6,8,10,12,3,5,7,11,13,17};
  结果定义的matrix是三行六列的数组。也
- 可以 int matrix[][6]={{1,3},{2,4},{3,5,7}};
  同样也是三行六列。注意这里只能最高维省略

多维数组作为函数参数

- void Func(int array[3][10]);
  void Func(int array[ ][10]);
- void Func(int (*array)[10]);
- 注意*array需要用括号括起来。
- 错误示例：
- void Func(int *array[10]);
  - [ ]的优先级高于*
  - array有10个元素，其中每个元素都是一个指向整型对象的指针
- void Func(int array[ ][ ]);
- void Func(int array[3][ ]);
  - 因为从实参传递来的是数组的起始地址，在内存中按数组排列规则存放(按行存放)，而并不区分行和列，如果在形参中不说明列数，则系统无法决定应为多少行多少列，不能只指定一维而不指定第二维。

动态分配多维数组存储空间

```
char **a, i;
// 先分配m个指针单元，注意是指针单元
// 所以每个单元的大小是sizeof(char *)
a = (char **) malloc(m * sizeof(char * ));
// 再分配n个字符单元，
// 上面的m个指针单元指向这n个字符单元首地址
for(i = 0; i < m; i++)
a[i] = (char * )malloc(n * sizeof(char ));
```

```
int i;
for(i=0;i<m;i++)
    free((void *)a[i]);
free((void *)a);
```

用一维数组指针：

```
double (*a)[3];
a = (double ((*)[3]))malloc (n*3*sizeof(double));
```

申请一维数据并强制将其转成二维数组指针：

```
 int *a = new int[nRow * MAXCOL];
int (*p)[MAXCOL] = (int(*)[MAXCOL]) a;
```

计算机编码

- 在计算机内部，所有的信息都是以二进制形式进行存储。无论是字符，或是视频音频文件，最终都会对应到一串由 0 和 1 构成的数字串。所以从我们能看懂的人类信息转变为机器级别的二进制语言的过程就可以理解为一种编码的过程，自然，相反的过程就是所谓的解码的过程。
- **所有的乱码都是源于解码方式与编码方式的不一致。**

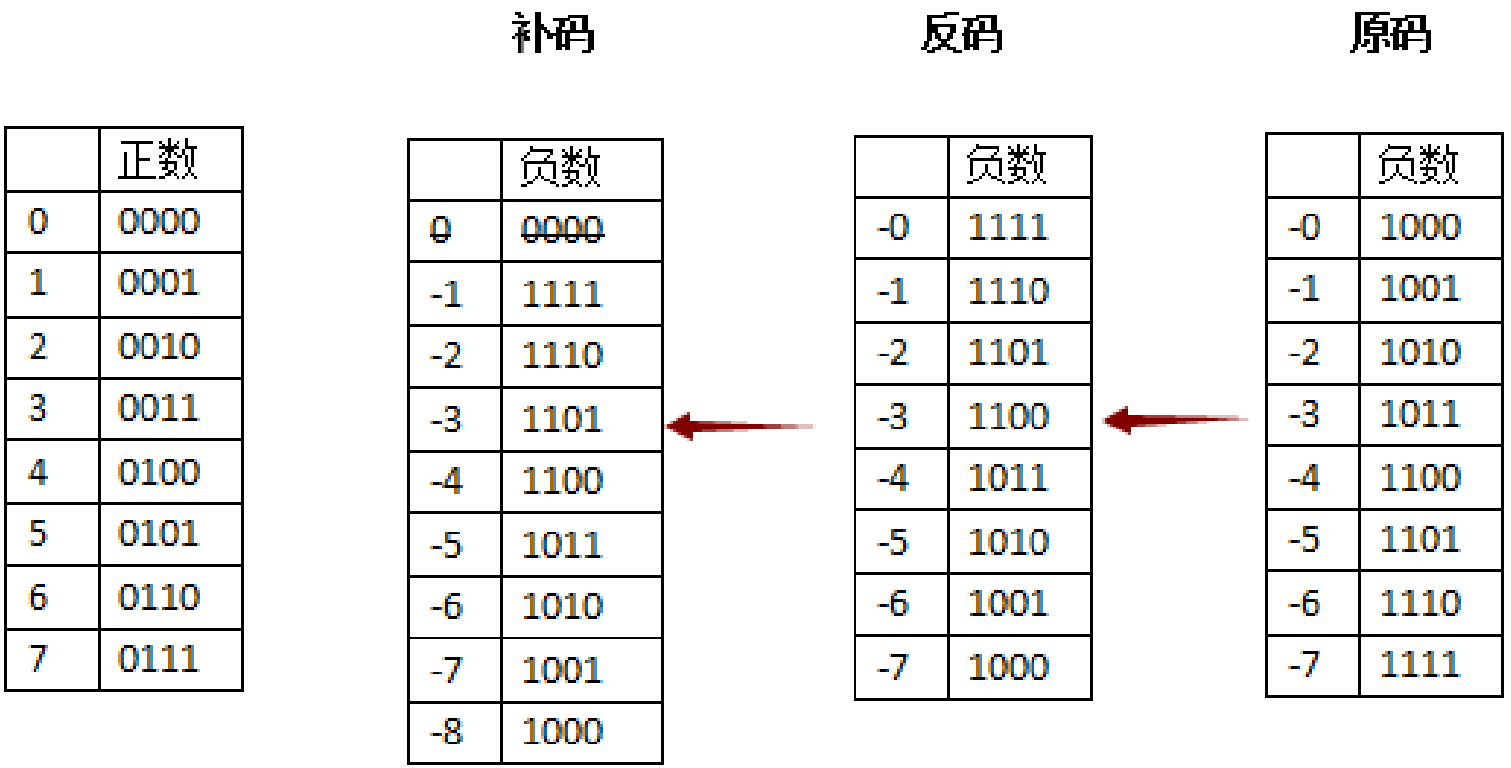
整数编码

- 无符号：十进制转换成二进制
- 有符号？
- 先用二进制数字表示它的绝对值，再将数字的第一位用来表示数字的正负（0开头表示正数，1开头表示负数）
- 5 - 3 = 5 + （-3）
- = 0101 + 1011 = 10000 = 0000 = 0
  - 4位二进制相加溢出结果截断保留后4位
- 0000(2)=0(10)1000(2)=−0(10)
  - 我们现实生活中的数字系统，并不存在-0

原码, 反码, 补码

- **原码**
- 原码就是符号位加上真值的绝对值, 即用第一位表示符号, 其余位表示值. 比如如果是8位二进制：
- $[+1]_{原}$ = 0000 0001
- $[-1]_{原}$ = 1000 0001
- **反码**
- 正数的反码是其本身，负数的反码是在其原码的基础上，符号位不变，其余各个位取反。
- $[+1] = [00000001]_{原} = [00000001]_{反}$
- $[-1] = [10000001]_{原} = [11111110]_{反}$

- **补码**
- 正数的补码就是其本身，负数的补码是在其原码的基础上，符号位不变，其余各位取反，最后+1，(即在反码的基础上+1)
- $[+1] = [00000001]_{原} = [00000001]_{反} = [00000001]_{补}$
- $[-1] = [10000001]_{原} = [11111110]_{反} = [11111111]_{补}$



	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

	负数
~~0~~	~~0000~~
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

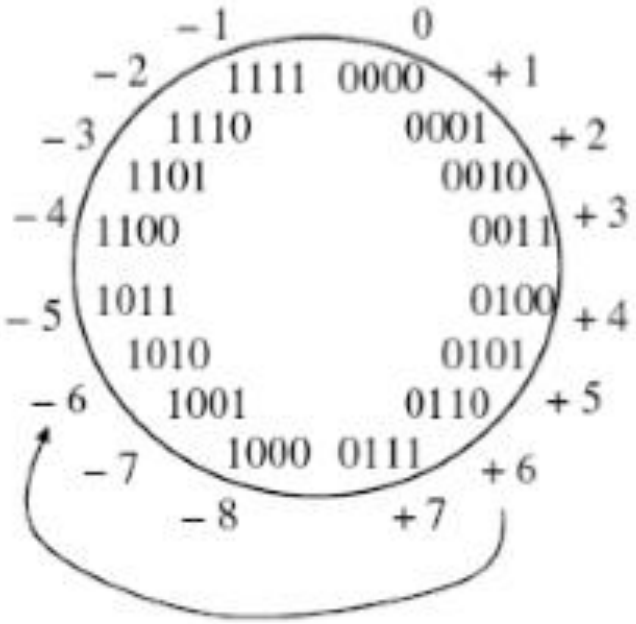
	负数
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

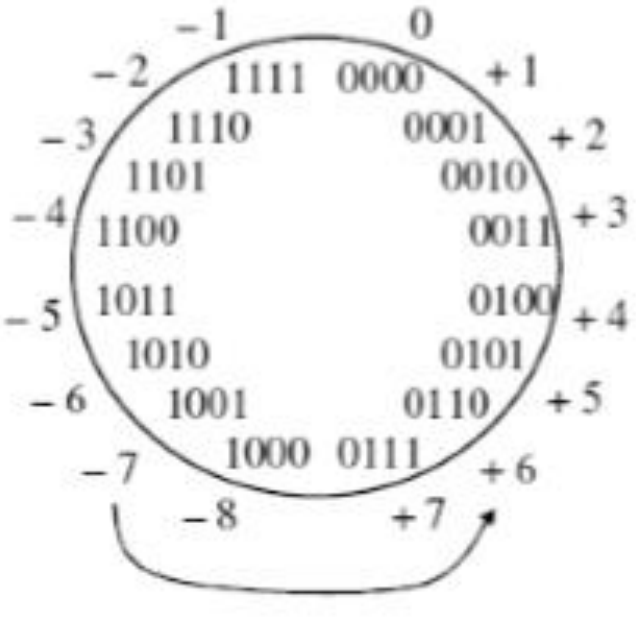
补码的好处

- 0的符号：
  - 1-1 = 1 + (-1) = $[0000\ 0001]_{原}$ + $[1000\ 0001]_{原}$ = $[0000\ 0001]_{补}$ + $[1111\ 1111]_{补}$ = $[0000\ 0000]_{补}$=$[0000\ 0000]_{原}$
  - 这样0用[0000 0000]表示，而以前出现问题的-0则不存在了
- 加减法都只需要用加法电路实现
  - 补码和正数的码形成了这样一种关系：所有位加起来等于(1)0000

- 计算机中的加法器是以2^n为模的有模器件（n为加法器的位数），因此可以引入补码，把减法运算转换为加法运算，以简化运算器的设计
- 负数补码的另一个定义：模数K减去其绝对值



$6 + 4 \Rightarrow -6$



$-7 - 3 \Rightarrow +6$

浮点数编码

- IEEE规定将浮点数拆分为3部分，符号、整数、尾数
- float类型在内存中占4个字节(32位)，最高位表示符号，从左向右取8位表示指数，其余23位用于表示尾数

$$N = M \times r^{E}$$

M —— 尾数；E —— 指数

- 例子：12.25
- 二进制表示：1100.01->用科学记数法表示为1.10001，小数点向左移3位
- 指数域初始值127，每+1表示左移了1位，-1反之->指数域为3+127
- 尾数域为10001000000000000000000（不足23位时，低位填0补充）
- 符号域0正1负->符号域为0

- 2进制的小数无法精确的表达10进制小数：
- 10进制小数转换为2进制的方法：**乘以2，取整，小数部分继续乘以2，取整，得到小数部分0为止，将整数顺序排列**
  0.8125 x 2 = 1.625 取 1
  0.625 x 2 = 1.25 取 1
  0.25 x 2 = 0.5 取 0
  0.5 x 2 = 1.0 取 1
  所以0.8125的二进制是0.1101

- 计算10进制0.2的2进制：
- 0.2 x 2 = 0.4
  0.4 x 2 = 0.8
  0.8 x 2 = 1.6
  0.6 x 2 = 1.2
  0.2 x 2 = 0.4

- 所以我们要尽量避免对浮点数直接比较大小：
- 0.1 + 0.2 == 0.3; ×
- 两个数之间的差值处于一个能接受的范围之内的话,那么,我们就认为这两个浮点数是相等的

字符编码

- ASCII 码：
- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange 美国信息交换标准代码
- 发音： /ˈæski/ ***ASS****-kee*
- ASCII第一次以规范标准的类型发表是在1967年，最后一次更新则是在1986年，至今为止共定义了128个字符
- 局限在于只能显示26个基本拉丁字母、阿拉伯数字和英式标点符号，因此只能用于显示现代美国英语（而且在处理英语当中，即使会违反拼写规则，外来词如naïve、café、élite等等时，所有重音符号都必须去掉）。

- Unicode：
- Unicode 是一个很大的集合，现在的规模可以容纳100多万个符号。每个符号的编码都不一样，比如，U+0041表示英语的大写字母A，U+4E25表示汉字严。
- Unicode 只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储。
  - 汉字严的 Unicode 是十六进制数4E25，转换成二进制数足足有15位（100111000100101）
  - 用几个字节来存储Unicode呢？

- 如何才能区别 Unicode 和 ASCII？计算机怎么知道三个字节表示一个符号，而不是分别表示三个符号呢？
- 我们已经知道，英文字母只用一个字节表示就够了，如果 Unicode 统一规定，每个符号用三个或四个字节表示，那么每个英文字母前都必然有二到三个字节是0，这对于存储来说是极大的浪费，文本文件的大小会因此大出二三倍，这是无法接受的。

- **UTF-8**
- **UTF-8 是 Unicode 的实现方式之一。**
- 一种变长的编码方式，可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。