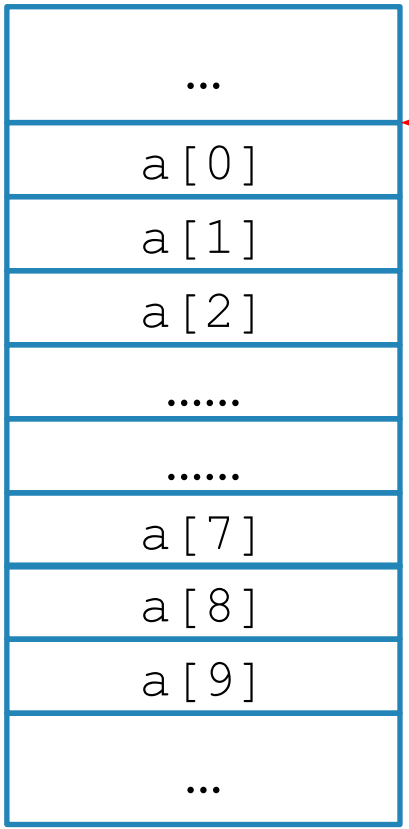
# C语言程序设计基础

林川

# 第七章 数 组

- 什么是数组？为什么需要数组？
- 如何使用数组？数组的存放方式？
- 什么是字符串？
- 有哪些字符串的操作？
- 字符串和数组的关系？

# 7.1.1 一维数组元素存储




```
int a[10];
a
```

可以用一个下标，例如$i$，
指示第$i$个元素。

# 7.1.2 一维数组的定义和引用

类型名 数组名[数组长度]

类型名：数组元素的类型

数组名：数组的名称，合法的标识符

数组长度：一个整数，给定数组的大小。

例如：

int a[10]; 定义一个含有10个整型元素的数组 a

char c[200]; 定义一个含有200个字符元素的数组 c

float f[5]; 定义一个含有5个浮点型元素的数组 f

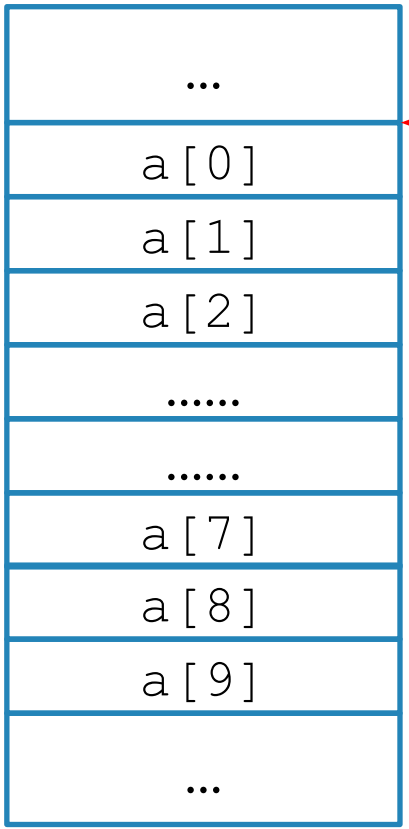
# 一维数组的引用

- 数组名[下标]
  - 下标:整型表达式
  - 下标>=0 &&下标<数组长度

- 例如:a[0]表示数组a的首元素, a[k]表示第k个元素。

- 数组元素的使用方法与同类型的变量相同
  - scanf("%d", &a[i]);
  - temp = a[index]; a[index]= a[k]; a[k] = temp;
  - printf("%d ", a[i]);

# 数组的下标不能越界




```
int a[10]
a
```

数组a有10个元素
  a[0]是首元素
  a[9]是末尾元素(最后一个)

如果引用a[10], 可能导致程序崩溃
  因为数组的后面存储了其他的数据
  非法修改将导致意外发生

# 7.1.3 一维数组的初始化

类型名 数组名[长度]= {初值表};

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

初值表的值依次赋予数组元素

char s[3] = { 'a', 'b'};

如果初值的个数少于长度，那么数组后面的原数将不被初始化

float b[ ] = {1, 2, 3};

如果长度省略，那么：长度=初值的数量

# 数组可以定义为static类型

static 类型名 数组名[长度];

例如

```
static int a[10] = {1, 2, 3};
```

# 7.1.4 使用数组编程[例7-2]

使用数组计算斐波那契数列前10个元素

```
int i;
int fib[10] = {1,1};

for( i=2; i<10; i++ )
   fib[i] = fib[i-2] + fib[i-1];
```

```
1, 1, 2, 3, 5, 8                --- 数组的内容
0, 1, 2, 3, 4, 5, 6, 7, 8, 9    --- 下标
```

# 使用数组编程[数组作为函数参数]

在一个数组 a 中查找指定的元素 x。
如果找到了，那么返回下标；否则返回-1。

```
int search(int a[ ], int n, int x)
{
   int i;
   for( i=0; i<n; i++ )
      if( a[i]==x )
          return i;
    return (-1);
}
```

# 使用数组编程[最小元素]

在一个数组 a 中查找最小的元素，并将其与数组的首个元素交换位置。

```
void find_and_mov_min(int a[ ], int n)
{
   int imin = 0, i;
   int temp;

   for( i=1; i<n; i++ )
      if( a[i]<a[imin] ) imin = i;

    temp = a[0];  a[0] = a[imin]; a[imin] = temp;
}
```

# 使用数组编程[选择法排序]例7-5

```
/* 排序a[0], a[1],…, a[n-1] */
for( k=0; k<n-1; k++ )
{
    /* 排序a[k], a[k+1],…, a[n-1] */
    index = k; /* 找其中的最小元素*/
    for( i = k+1; i<n; i++ )
        if( a[i] < a[index] )
            index = i;
    /* 将最小元素和a[k]交换 */
    temp = a[index];
    a[index] = a[k];
    a[k] = temp;
}
```

# 7.2 二维数组

## 一维

- int a[4];

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

## 二维

- int a[4][3];

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

# 7.2.2 二维数组定义与引用

## 类型名 数组名[行数][列数];

- 例如 int a[20][30];

| | | | |
|---|---|---|---|
| a[0][0] | a[0][1] | ... | a[0][29] |
| ... | ... | ... | ... |
| a[19][0] | a[19][1] | ... | a[19][29] |

- 二维数组在内存中逐行、连续存放
  - 行0，行1，行2，...
  - 每一行的元素连续存放：列0，列1，列2，...

# 二维数组的逐行存放

int a[3][2];

- 3 行 2 列，6 个元素

```
a[0][0]   a[0][1]
a[1][0]   a[1][1]
a[2][0]   a[2][1]
```

# 7.2.2 二维数组定义与引用

- 通过2个下标，行下标和列下标，引用数组元素

  a[i][j] – 表示第i行，第j列的元素

# 7.2.1 找出矩阵的最大值及其位置[例7-7]

- 将1个3*2的矩阵存入1个3*2的二维数组中
- 找出最大值以及它的行下标和列下标

并输出该矩阵。

```
int a[3][2];
int i, j, col, row;
```

# 7.2.1 找出矩阵的最大值及其位置[例7-7]

```
/* 输入矩阵 */
/* 二维数据，常采用2重循环*/
printf("Enter 6 integers:\n");
for( i=0; i<3; i++ )
  for( j=0; j<2; j++ )
    scanf("%d", &a[i][j]);
```

# 7.2.1 找出矩阵的最大值及其位置[例7-7]

```
/* 输出矩阵 */
for( i=0; i<3; i++ )
{
   for( j=0; j<2; j++ )
      printf("%4d", a[i][j]);
   printf("\n");
}
```

# 7.2.1 找出矩阵的最大值及其位置[例7-7]

```
/* 遍历矩阵，求最大元素 */
row = col = 0;
for( i=0; i<3; i++ )
   for( j=0; j<2; j++ )
      if(a[i][j]>a[row][col])
      {   /* 更新最大值的下标 */
          row = i;
          col = j;
       }
printf("max = a[%d][%d] = %d\n",
    row, col, a[row][col]);
```

# 7.2.3 二维数组的初始化

- 分行初始化

```
类型名 数组名[行数][列数] =
  {
   { 行0初值表 }, { 行1初值表 }, ...,
  };
```

  - 初始化所有元素

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

  - 初始化前2行

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

  - 行1无初始化

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

# 7.2.3 二维数组的初始化

- 顺序初始化

类型名 数组名[行数][列数] = { 初值表 };

二维数组的元素按照在内存中存放 顺序（逐行连续存放）与初始化的 值对应。

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

等价于

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

# 7.2.3 二维数组的初始化[省略行长度]

- 对全部元素都赋了初值，

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

数组a

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

- 或在分行初始化时，在初值表中列出了全部行

```
static int b[ ][3] = { {1,2,3},{},{4,5},{} };
```

数组b

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 0 | 0 | 0 |
| 4 | 5 | 0 |
| 0 | 0 | 0 |

# 7.2.4 二维数组编程[定义矩阵] [例7-8]

定义一个3x2的二维数组，其元素值满足：

```
    a[i][j] = i + j
```

```
int a[3][2];
int i, j;
/* 计算元素值 */
for ( i=0; i<3; i++ )
   for ( j=0; j<2; j++ )
      a[i][j] = i + j;
```

# 7.2.4 二维数组编程[定义矩阵][例7-8]

```
/* 按矩阵格式输出二维数组 */
for ( i=0; i<3; i++ )
{
    for ( j=0; j<2; j++ )
        printf("%4d", a[i][j]);
    printf("\n");
}
```

可以 printf('\n'); 吗？

# 7.2.4 二维数组编程[矩阵转置][例7-9]

int a[N][N]; N是正整数

- 在a[i][j]中，i、j的合法取值范围[0，N-1]

- 矩阵与二维数组
  - 用二维数组a表示N*N方阵时，对应关系：

| | | | | |
|---|---|---|---|---|
| a[0][0] | a[0][1] | a[0][2] | 主对角线 | i=j |
| a[1][0] | a[1][1] | a[1][2] | 上三角 | i<j |
| a[2][0] | a[2][1] | a[2][2] | 下三角 | i>j |

# 7.2.4 二维数组编程[矩阵转置][例7-9]

```
int a[6][6], temp;
int i, j;

for ( i=0; i<6; i++ )
   for ( j=0; j<i; j++ )
   {
      temp = a[i][j];
      a[i][j] = a[j][i];
      a[j][i] = temp;
   }
```

可以 j<6; 吗？

# 7.2.4 二维数组编程[求第几天][例7-10]

定义函数 day_of_year(year, month, day),计算给定的年月日在这一年中是第几天

关键是每个月有几天？

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 非闰 | 0 | 31 | 28 | 30 | 31 | 30 | 31 | 30 | 31 | 30 | 31 | 30 | 31 |
| 闰年 | 0 | 31 | 29 | 30 | 31 | 30 | 31 | 30 | 31 | 30 | 31 | 30 | 31 |

```
int tab[ ][13] = {
  {0,31,28,30,31,30,31,30,31,30,31,30,31},
  {0,31,29,30,31,30,31,30,31,30,31,30,31}};
```

# 7.2.4 二维数组编程[求第几天][例7-10]

```
int day_of_year(int year, int month, int day)
{
   int k, leap;
   int tab[][13] = …<略>
   leap = ( (year%4==0 && year%100 ! =0)
          || year%400==0 );
   for( k=1; k<month; k++ )
      day += tab[leap][k];
   return day;
}
```

# 7.3 字符数组

char 数组名[长度]

```
char t[10];
char t[5] = {'h', 'a', 'p', 'p', 'y'};
char t[10] = {'h', 'a', 'p', 'p', 'y'};
```

# 7.3 字符数组[字符串]

字符串是一串字符序列

C语言中，字符串存储于一段连续的内存中

以字符'\0' -> 0结束 (字符'\0'的值为0) ‘0’ -> 48

用字符数组表示字符串

```
char s[6] = {'h', 'a', 'p', 'p', 'y', '\0' };
char s[6] = {'h', 'a', 'p', 'p', 'y', 0 };
```

可以更方便直观的写法：

```
char s[6] = "happy";
```

# 7.3.1 判断回文

输入一个以回车为结束符的字符串(少于9个字符)，判断其是否为回文(逆序不变)。

```
char s[10];
int n, k, j;
```

```
12321
```

```
123321
```

# 7.3.1 判断回文

```
/* 读入字符串 */
printf("Enter a string:");
n = 0;
while ( (s[n]=getchar()) != '\n' )
   n ++;
s[n] = '\0'；/* 字符串结束标识符 */
****************************************
while ( (s[n++]=getchar()) != '\n' )
   ;
s[--n] = '\0';
```

# 7.3.1 判断回文

```
/* 判断是否为回文 */
for ( j=0, k=n-1; j<k; j++, k-- )
    if( s[j]!=s[k] )
         break;
if( j<k )
   printf("不是回文\n");
else
   printf("是回文\n");

123231
```

# 7.3.3 字符串

- 字符串常量
  - 用一对双引号括起来的字符序列

    "Happy"，"Monday"
  - 字符串结束符是'\0'
    - 隐含在"Happy"，"Monday"中
    - 注意：“Happy"的长度是5，但是其中的字符个数是 6
  - 区分"a" 和 'a'
    - "a": 字符串
    - 'a': 字符

# 7.3.3 字符串

- 字符串的定义和初始化

  char s[6]={'H','a','p','p','y',0};

  或者

  char s[6] = "Happy";

- 如果已经定义了char s[6];

  不能: s = "Happy";

  也不能: s[ ] = "Happy";

  但是可以按照数组方式，逐个字符赋值。

  例如: s[0]='H'; s[1]='a'; ...

# 字符串与一维字符数组

- 字符串:一个特殊的一维字符数组
- 把字符串放入一维字符数组(存储)
- 对字符串的操作 ===> 对字符数组的操作

# 7.3.3 字符串

- 字符串的操作

```
char str[80];
```

  - 格式化输入字符串

```
scanf("%s", str);
```

  - 格式化输出字符串

```
printf("%s", str);
```

str必须是一个以'\0'结束的字符串

否则崩溃

```
char a[] = {‘a’, ‘b’, ‘c’};
```

# 2. 对字符串的操作

- 把字符串放入一维字符数组(存储)
- 对字符串的操作 ===> 对字符数组的操作
  - 普通字符数组:数组元素的个数是确定的,一般用下 标控制循环
  - 字符串:没有 显式地给出有效字符的个数,只 规定在字符串 结束符 '\0' 之前的字符都是字符串的有效字符,一般用 结束符 '\0' 来控制循环
  - 循环条件:s[i] != '\0'

# 计算字符串的有效长度

```
int strlength( char s[] )
{
   int n = 0;
   while( s[n] != '\0' )
       n ++;
   return n;
}
```

# 计算字符串的有效长度

```
int strlength( char s[] )
{
   int n = 0;
   while( s[n++] != '\0' )
       ;
   return  (n-1) ;
}
```

# 输出字符串

```
void output( char s[] )
{
   int i = 0;
   while( s[i]!= '\0' )
       putchar( s[i++] );
}
```

# 7.3.4 字符串编程：进制转换[例7-14]

输入一个以回车结束的字符串(少于80个字符)，过滤其中的非16进制字符，生成新的字符串，然后将该16进制字符串转化为10进制数。

```
char str[80];
int i, n;
```

# 7.3.4 字符串编程：进制转换[例7-14]

```
/* 输入字符串 */
printf("Enter a string:");
i = 0;
while( (str[i]=getchar())!='\n' )
   i++;
str[i] = 0;
```

# 7.3.4 字符串编程：进制转换[例7-14]

```
/* 过滤非16进制字符 */
n = 0 ; /* 过滤后的字符数 */
for ( i=0; str[i] != 0 ; i++ )
   if( str[i]>='0' && str[i]<='9' ||
       str[i]>='a' && str[i]>='f' ||
       str[i]>='A' && str[i]>='F' )
       str[n++] = str[i];
str[n] = 0;
```

# 7.3.4 字符串编程：进制转换[例7-14]

```
/* 转换进制 */
n = 0 ; /* 10进制数值 */
for ( i=0; str[i] != 0 ; i++ )
   if( str[i]>='0' && str[i]<='9' )
       n = n * 16 + str[i]- '0';
   else if( str[i]>='a' && str[i]>='f' )
       n = n * 16 + str[i]- 'a';
   else
       n = n * 16 + str[i]- 'A';

printf("digit = %d\n", n);
```

# 作业