计算机导论与程序设计 [CS006001-60]

段江涛

机电工程学院

西安电子科技大学 XIDIAN UNIVERSITY

2019 年 10 月

lecture-10 主要内容

利用数组处理批量数据

1. 定义数组: int a[10];
2. 引用数组: int i=0; a[i]
3. 初始化数组: int a[5]={1,2,3,4,5};
4. 冒泡排序

为什么需要数组

- 要向计算机输入全班 50 个学生的成绩一门课的成绩;
- 用 50 个 float 型简单变量表示学生的成绩
  - **烦琐**,如果有 1000 名学生怎么办呢?
  - 没有反映出这些**数据间的内在联系**,实际上这些数据是同一个班级、同一门课程的成绩,它们具有相同的属性。

```
float s0,s1,s2,...,s49; // 50名学生一门课的成绩
```

```
float s[50]; // 50名学生一门课的成绩
int i; // 表示数组下标
for(i=0;i<50;i++) scanf("%f",&s[i]);
```

数组

1. 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的,下标代表数据在数组中的序号。
2. 用数组名和下标即可唯一地确定数组中的元素。
3. 数组中的每一个元素都属于同一个数据类型。

为什么需要数组

- 要向计算机输入全班 50 个学生的成绩一门课的成绩;
- 用 50 个 float 型简单变量表示学生的成绩
  - **烦琐**,如果有 1000 名学生怎么办呢?
  - 没有反映出这些**数据间的内在联系**,实际上这些数据是同一个班级、同一门课程的成绩,它们具有相同的属性。

```
float s0,s1,s2,...,s49; // 50名学生一门课的成绩

float s[50]; // 50名学生一门课的成绩
int i; // 表示数组下标
for(i=0;i<50;i++) scanf("%f",&s[i]);
```

数组

1. 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的,下标代表数据在数组中的序号。
2. 用数组名和下标即可唯一地确定数组中的元素。
3. 数组中的每一个元素都属于同一个数据类型。

定义数组: int a[10];

定义一维数组:

元素类型　数组名 [常量表达式 (表示元素个数—数组的长度)]

```
#define NUM 100
float s[50]; // 50名学生一门课的成绩
int a[10]; // 10个元素的整型数组
double b[NUM]; // 常量NUM个元素的double数组
char c[50]; // 50个元素的char型数组
```

Notes

数组元素的下标从 0 开始,int a[10]; 10 个整型元素,则最大下标值为 9,不存在数组元素 a[10]

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

引用数组: int i=0; a[i]

```
int a[10]; // 10个元素的整型数组
int i;
a[0]=10; // 给a数组的第一个元素赋值
a[9]=10; // 给a数组的最后一个元素赋值
printf("%d,%d",a[0],a[9]);
for(i=0;i<10;i++) a[i] = i+1; // 给数组的第i个元素赋值
for(i=0;i<10;i++) printf("%d\t",a[i]); // 输出数组a的10个元素
for(i=0;i<10;i++) scanf("%d",&a[i]); // 输入10个整数, 存入数组a中。注意'&'
```

Notes

数组元素的下标从 0 开始, int a[10]; 10 个整型元素, 则最大下标值为 9, 不存在数组元素 a[10]

[例 6.1] 对 10 个字符型数组元素依次赋值为'a','b',· · · 。要求按逆序输出。

```
char c[10];
int i;
for(i=0;i<9;i++) c[i]='a'+i;
for(i=9;i>0;i--) printf("%c␣", c[i]);
printf("\n");
```

[例 6.1] 对 10 个字符型数组元素依次赋值为'a','b',· · · 。要求按逆序输出。

```
char c[10];
int i;
for(i=0;i<9;i++) c[i]='a'+i;
for(i=9;i>0;i--) printf("%c␣", c[i]);
printf("\n");
```

初始化数组: int a[5]={1,2,3,4,5};

为了使程序简洁,常在定义数组的同时给各数组元素赋值,这称为数组的初始化。

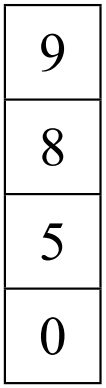
```
int a[10]={1,2,3,4,5,6,7,8,9,10}; // 在定义数组时对全部数组元素赋予初值。
char c[10]={'a','b'}; // 可以只给数组中的一部分元素赋值。其他元素的值不确定
double d[]={10.0,10.2,10.3}; // 等效于 double d[3]={10.0,10.2,10.3}
d[2] = 20.2; // 修改第3个元素
```

[例 6.2] 用数组来处理求 Fibonacci 数列问题。

```
#include <stdio.h>
int main()
{
  int i;
  int f[20]={1,1}; //对最前面两个元素f[0]和f[1]赋初值1
  for(i=2;i<20;i++)
     f[i]=f[i-2]+f[i-1]; //先后求出f[2]~f[19]的值
  for(i=0;i<20;i++)
  {
     if(i%5==0) printf("\n"); //控制每输出5个数后换行
     printf("%12d",f[i]); //输出一个数
  }
  printf("\n");
  return 0;
}
```

冒泡排序 (第 1 趟)

int a[4]={9,8,5,0}; // n 个元素, 要求从小到大顺序排列



第 1 趟原始数据

第 1 趟第 1 次相邻两数比较

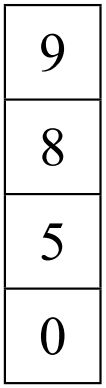
第 1 趟第 2 次相邻两数比较

第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

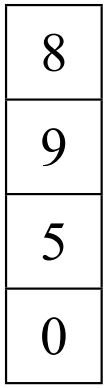
第 1 趟, 相邻两数比较 3-0 次

冒泡排序 (第 1 趟)

int a[4]={9,8,5,0}; // n 个元素, 要求从小到大顺序排列



第 1 趟原始数据



第 1 趟第 1 次相邻两数比较

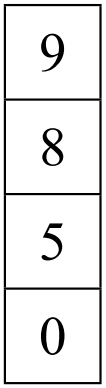
第 1 趟第 2 次相邻两数比较

第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

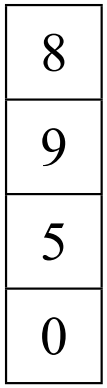
第 1 趟, 相邻两数比较 3-0 次

冒泡排序 (第 1 趟)

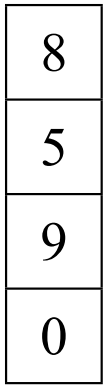
int a[4]={9,8,5,0}; // n 个元素, 要求从小到大顺序排列



第 1 趟原始数据



第 1 趟第 1 次相邻两数比较



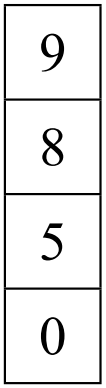
第 1 趟第 2 次相邻两数比较

第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

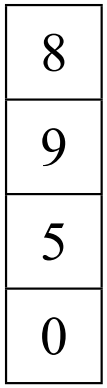
第 1 趟, 相邻两数比较 3-0 次

冒泡排序 (第 1 趟)

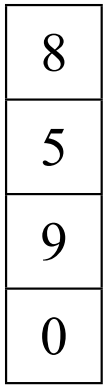
int a[4]={9,8,5,0}; // n 个元素, 要求从小到大顺序排列



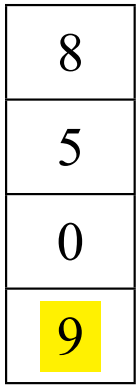
第 1 趟原始数据



第 1 趟第 1 次相邻两数比较



第 1 趟第 2 次相邻两数比较

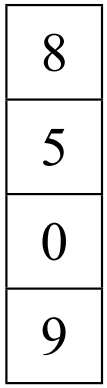


第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

第 1 趟, 相邻两数比较 3-0 次

冒泡排序 (第 2 趟)

int a[4]={9,8,5,0}; // 要求从小到大顺序排列



第 2 趟原始数据

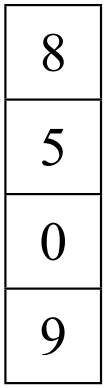
第 2 趟第 1 次相邻两数比较

第 2 趟第 2 次相邻两数比较 (第 2 大沉底)

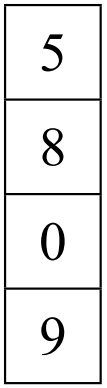
第 2 趟, 相邻两数比较 3-1 次

冒泡排序 (第 2 趟)

int a[4]={9,8,5,0}; // 要求从小到大顺序排列



第 2 趟原始数据



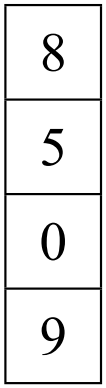
第 2 趟第 1 次相邻两数比较

第 2 趟第 2 次相邻两数比较 (第 2 大沉底)

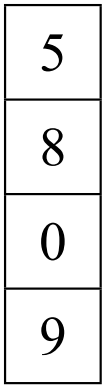
第 2 趟, 相邻两数比较 3-1 次

冒泡排序 (第 2 趟)

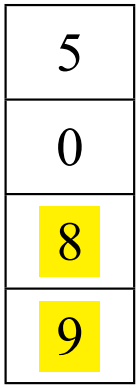
int a[4]={9,8,5,0}; // 要求从小到大顺序排列



第 2 趟原始数据



第 2 趟第 1 次相邻两数比较

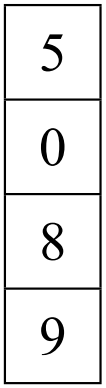


第 2 趟第 2 次相邻两数比较 (第 2 大沉底)

第 2 趟, 相邻两数比较 3-1 次

冒泡排序 (第 3 趟)

int a[4]={9,8,5,0}; // 要求从小到大顺序排列



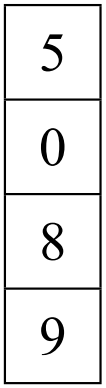
第 3 趟原始数据

第 3 趟第 1 次相邻两数比较 (第 3 大沉底)

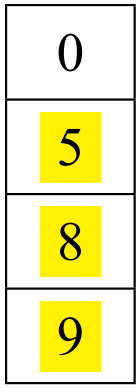
第 3 趟, 相邻两数比较 3-2 次

冒泡排序 (第 3 趟)

int a[4]={9,8,5,0}; // 要求从小到大顺序排列



第 3 趟原始数据



第 3 趟第 1 次相邻两数比较 (第 3 大沉底)

第 3 趟, 相邻两数比较 3-2 次

冒泡排序 (总结)

- n: 表示 n 个元素
- j=1,2,..n-1
- 表示第 j 趟排序,相邻元素两两比较,必要时交换。
- 第 j 趟排序进行 n-j 次相邻元素两两比较;
- 最多进行 n-1 趟排序。
- 注意排序趟数,两两比较的边界条件。用从 2 个数,3 个数,4 个数排序演练

冒泡排序 (核心程序)

```
int a[10]={9,8,7,6,5,4,3,2,1,0},i,j,t;
for(j=0;j<9;j++) //进行9次循环，实现9趟比较
{
  for(i=0;i<9-j;i++) //在每一趟中进行9-j次比较
  {
    if(a[i]>a[i+1]) //相邻两个数比较
       { t=a[i]; a[i]=a[i+1]; a[i+1]=t; } // 交换
  }
}
printf("the␣sorted␣numbers␣:\n");
for(i=0;i<10;i++)
   printf("%d␣",a[i]);
```

冒泡排序 (优化)

优化：第 j 趟排序中，没有进行相邻元素的交换，表示数据已经排序好，没有必要进行此后的 $(n-1-j)$ 趟排序。

```
int a[10]={7,8,7,6,5,6,7,8,9,10},i,j,t,flag;
for(j=0;j<9;j++) //进行9次循环，实现9趟比较
{
   flag = 0; // 每趟排序，初始化flag，表示未进行交换
   for(i=0;i<9-j;i++) //在每一趟中进行9-j次比较
   {
     if(a[i]>a[i+1]) //相邻两个数比较
       { t=a[i]; a[i]=a[i+1]; a[i+1]=t; flag=1; } // 交换，设置标志变量
   }
   if(!flag) break; // 表示第j趟未交换，排序好了！
}
printf("the␣sorted␣numbers␣:\n");
for(i=0;i<10;i++) printf("%d␣",a[i]);
```

冒泡排序 (输出每趟排序的结果)

优化：第 j 趟排序中，没有进行相邻元素的交换，表示数据已经排序好，没有必要进行此后的 $(n-1-j)$ 趟排序。

```
int a[10]={7,8,7,6,5,6,7,8,9,10},i,j,t,flag;
for(j=0;j<9;j++) //进行9次循环，实现9趟比较
{
   flag = 0; // 每趟排序，初始化flag，表示未进行交换
   for(i=0;i<9-j;i++) //在每一趟中进行9-j次比较
   {
      if(a[i]>a[i+1]) //相邻两个数比较
       { t=a[i]; a[i]=a[i+1]; a[i+1]=t; flag=1; } // 交换，设置标志变量
   }
   printf("第%d趟排序:␣\n", j)
   for(t=0;t<10;t++) printf("%d␣",a[t]); // 临时变量t的复用
   if(!flag) break; // 表示第j趟未交换，排序好了！
}
printf("the␣sorted␣numbers␣:\n");
for(i=0;i<10;i++) printf("%d␣",a[i]);
```

欢迎批评指正！