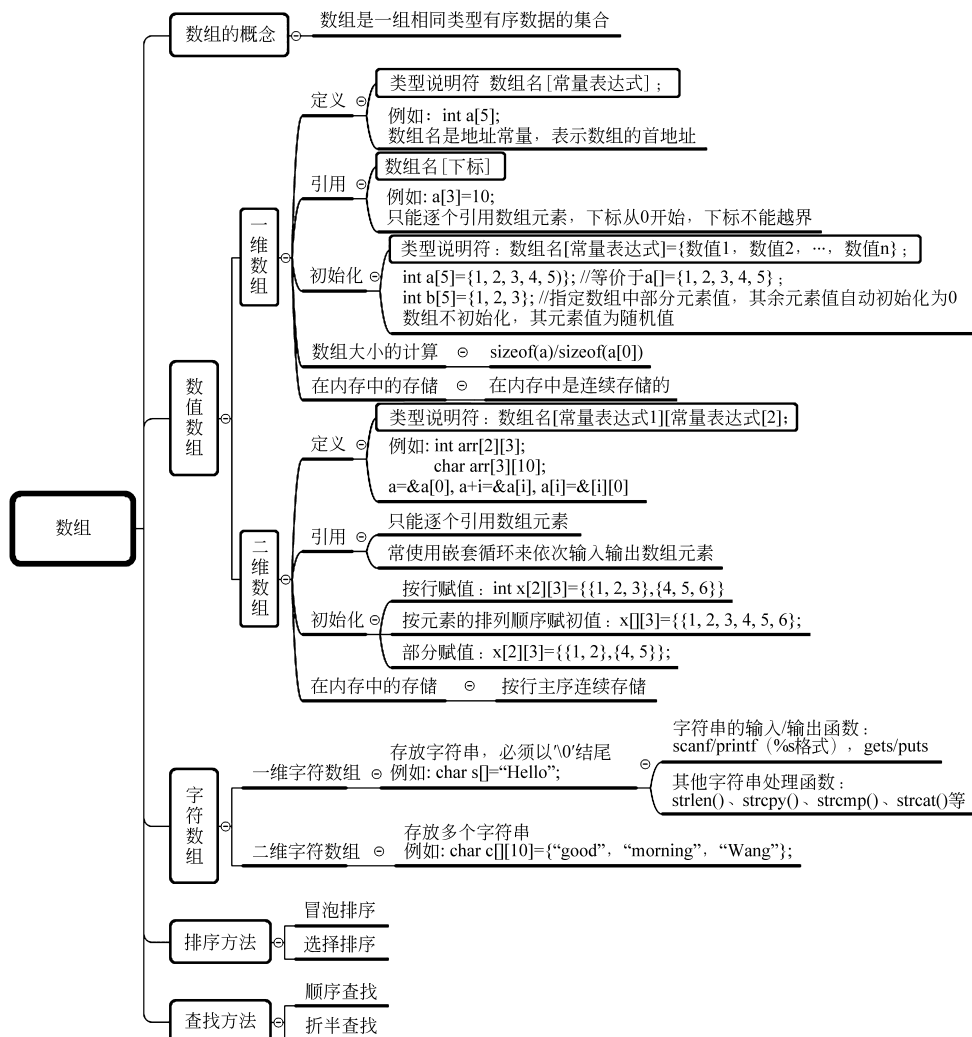


第5章 数组

5.1 基本知识提要

5.1.1 知识结构图



5.1.2 重点知识整理

1. 数组的概念

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。数组中的每一个元素都属于同一种数据类型。

2. 一维数组

(1) 一维数组的定义。在 C 语言中使用数组必须先进行定义。一维数组的定义方式为:

类型说明符 数组名 [常量表达式];

其中:

- ① 类型说明符可以是任意一种基本数据类型或构造数据类型。
- ② 数组名是用户定义的数组标识符,命名规则遵循标识符命名规则。
- ③ 方括号中的常量表达式表示数组元素的个数,也称为数组的长度。

例如:

```
float b[10], c[20];    //定义浮点型数组 b,有 10 个元素,浮点型数组 c,有 20 个元素
char ch[20];          //定义字符数组 ch,有 20 个元素。
```

(2) 一维数组的引用。一维数组元素的引用格式:

数组名 [下标]

数组的引用下标取值范围为 0 到数组长度值-1,不能越限。如用 `int a[15]` 定义的数组,引用时的下标不能超过或等于 15,语句 `a[15]=10` 是错误的。

定义数组时使用的“数组名[常量表达式]”和引用数组元素时使用的“数组名[下标]”形式相同,但含义不同。数组定义的方括号中给出的是数组的长度;而数组元素中的下标是该元素在数组中的位置标识。前者只能是常量,后者可以是常量、变量或表达式。

(3) 一维数组的初始化。数组的初始化就是在定义数组的同时,给其数组元素赋初值。数组初始化是在编译阶段进行的。这样将减少运行时间,提高效率。

一维数组初始化赋值的一般形式为:

类型说明符 数组名 [常量表达式]={数值 1,数值 2, ..., 数值 n};

C 语言对数组的初始化有以下几点规定:

① 可以只给部分数组元素赋初值。当大括号中值的个数少于数组元素个数时,只给前面部分数组元素赋值。例如:

```
int a[10]={0,1,2,3,4};
```

相当于只给 $a[0]$ 、 $a[1]$ 、 $a[2]$ 、 $a[3]$ 、 $a[4]$ 赋初值,而后 5 个元素自动赋为 0 值。

② 只能给数组元素逐个赋值,不能给数组整体赋值。例如,给 10 个元素全部赋值为 1,只能写为:

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

而不能写为:

```
int a[10]=1;
```

③ 如给全部元素赋值,则在数组声明中,可以不给出数组元素的个数。例如:

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

可写为:

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

④ 大括号中数值的个数多于数组元素的个数是语法错误。

(4) 一维数组排序方法:冒泡排序。冒泡排序是交换排序中一种简单的排序方法。它的基本思想是对所有相邻数组元素进行大小比较,如果是逆顺($a[j]>a[j+1]$),则将其交换,最终达到有序化。其处理过程为:

① 将整个待排序的数组元素划分成有序区和无序区,初始状态有序区为空,无序区包括所有待排序的记录。

② 对无序区从前向后依次将相邻数组元素进行大小比较,若逆序将其交换,从而使值小的数组元素向上“飘浮”(或左移),值大的数组元素好像石块,向下“堕落”(或右移)。每经过一趟冒泡排序,都使无序区中值最大的数组元素进入有序区,对于由 n 个元素组成的数据序列,最多经过 $n-1$ 趟冒泡排序,就可以将这 n 个元素重新按值从小到大的顺序排列。

有序序列,第一趟定位第 n 个记录,此时有序区只有一个记录;第二趟定位第 $n-1$ 个记录,此时有序区有两个记录;以此类推,算法框架为:

```
for (i=n;i>1;i-) { 定位第 i 个记录; }
```

对数组元素进行排序的方法有很多种,本章上机实验部分再介绍一种比较简单的选择排序。更多排序方法将在后续的数据结构课程中学习,目前只需掌握简单的冒泡排序和选择排序。

(5) 一维数组查找方法。

- 顺序查找:就是从第一个元素开始,按索引顺序遍历待查找序列,直到找出给定目标或者查找失败。
- 折半查找:也称二分查找,它是一种效率较高的查找方法。但是,折半查找要求数组中的数组元素已有序排列。关于折半查找的详细内容参见本章上机实验部分。

3. 二维数组

(1) 二维数组的定义。二维数组定义的一般形式是:

类型说明 二维数组名 [常量表达式 1] [常量表达式 2];

其中常量表达式 1 表示第一维的长度,常量表达式 2 表示第二维的长度。例如:

```
int x[2][3];
```

x 是二维数组名,这个二维数组共有 6 个元素($2 \times 3 = 6$),它们是 $x[0][0]$ 、 $x[0][1]$ 、 $x[0][2]$ 、 $x[1][0]$ 、 $x[1][1]$ 、 $x[1][2]$ 。且其全部元素数值均为整型。

(2) 二维数组的引用。二维数组元素的引用格式为:

数组名 [下标 1] [下标 2]

同样要注意引用数组元素时下标不能越限。

(3) 二维数组的初始化。二维数组的初始化有两种方法:

① 把初始化值括在一对大括号内,例如,对于二维数组

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

初始化结果是: $x[0][0]=1$, $x[0][1]=2$, $x[0][2]=3$, $x[1][0]=4$, $x[1][1]=5$, $x[1][2]=6$ 。

② 把二维数组看作一种特殊的一维数组,该数组的每一个元素又是一个一维数组。例如:

```
int x[2][3];
```

可把数组 x 看成是具有两个元素的一维数组,其元素是 $x[0]$ 和 $x[1]$ 。而每个元素 $x[0]$ 、 $x[1]$ 又都是具有三个元素的一维数组,其元素是 $x[0][0]$ 、 $x[0][1]$ 、 $x[0][2]$ 、 $x[1][0]$ 、 $x[1][1]$ 、 $x[1][2]$ 。

$$x: \begin{cases} x[0]: x[0][0], x[0][1], x[0][2] \\ x[1]: x[1][0], x[1][1], x[1][2] \end{cases}$$

因此,上例二维数组的初始化可分解成两个一维数组的初始化:

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

对于二维数组初始化赋值还有以下几点说明:

① 可以只对部分元素赋初值,未赋初值的元素自动取 0 值。例如:

```
int x[2][2]={{1},{2}};
```

是对每一行的第一列元素赋值,未赋值的元素取 0 值。赋值后各元素的值为: $x[0][0]=1$, $x[0][1]=0$, $x[1][0]=2$, $x[1][1]=0$ 。

② 如对全部元素赋初值,则第一维的长度可以不给出。例如,二维数组 x 的初始化过程:

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

也可写成:

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

即第一维的长度省略,但第二维的长度不能省略。

4. 字符数组和字符串

(1) 字符数组。字符型数组的定义格式:

```
char 数组名[字符个数];
```

例如:

```
char c[5];
```

字符数组也可以是二维或多维数组。例如:

```
char c[3][4];
```

即为二维字符数组。

同样,字符数组也允许在定义时进行初始化赋值。例如:

```
char c[4]={'G','o','o','d'};
```

赋值后各元素的值为 $c[0]='G'$, $c[1]='o'$, $c[2]='o'$, $c[3]='d'$ 。如果提供赋值的字符个数多于数组元素的个数,则为语法错误。

字符型数组与数值型数组在初始化中的区别:初始化时,提供的数据个数如果少于数组元素个数,则多余的数组元素初始化为空字符 $\backslash 0$,而数值型数组初始化为 0。

(2) 字符串。在 C 语言中没有专门的字符串类型,C 语言使用字符数组来存放字符串。

存放字符串的字符数组的初始化有两种方法:

① 用字符常量初始化数组,用字符常量给字符数组赋初值要用大括号将赋值的字符常量括起来。例如:

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

数组 $str[6]$ 被初始化为 "China"。其中最后一个元素的赋值 $\backslash 0$ 可以省略。

② 用字符串常量初始化数组。例如:

```
char str[6]="China";
```

或

```
char str[6]={ "China" };
```

不管用字符常量初始化字符数组,还是用字符串常量初始化数组,若字符个数少于数组长度,程序都自动在末尾字符后加 $\backslash 0$ 。

注意:

① 用字符串常量初始化时,字符数组的长度可以省略,其数组存放字符的个数由赋

值的字符串的长度决定。

② 初始化时,若字符个数与数组长度相同,则字符末尾不加'\0',此时字符数组不能作为字符串处理,只能作为字符逐个处理。初始化时是否加'\0',要看是否作为字符串处理。

(3) 字符串处理函数。C 语言标准库函数中,提供了一些专门用于处理字符串的函数,常用的有 scanf()/printf() 函数(%s 格式)、gets()/puts() 函数、strlen() 函数、strcmp() 函数、strlwr()/strupr() 函数、strcat() 函数、strcpy()/strncpy() 函数。

5.2 主教材习题解答

一、选择题

1. D 2. A 3. C 4. D 5. D 6. A 7. A 8. B 9. D 10. C
11. B 12. D 13. B 14. C 15. B

二、填空题

1. 0、6
2. 2、0、0
3. windows2000
4. a[i]、2、i+1
5. t=i、i、'\0'

三、编程题

1. 解题思路:以数组中间的元素为中心,将其两侧对称的元素的值互换即可。
参考代码如下:

```
#include <stdio.h>
#define N 5
int main()
{
    int a[N];
    int i, j, temp;
    printf("Please enter array a:\n");
    for(i=0; i<N; i++)
        scanf("%d", &a[i]);           //从键盘输入数组元素的值
    printf("array a:\n");
    for(i=0; i<N; i++)                 //输出数组
        printf("%4d", a[i]);
    printf("\n");
    for(i=0; i<N/2; i++)               //将数组前后对称位置的元素值进行交换
```

```

    {
        j=N-i-1;
        temp=a[i];a[i]=a[j];a[j]=temp;
    }
    printf("reverse array a:\n");        //输出逆序后的数组
    for(i=0;i<N;i++)
        printf("%4d",a[i]);
    printf("\n");
    return 0;
}

```

2. 解题思路:

(1) 将一年 12 个月每月的天数存放到一个整型数组 days 中,2 月份天数的初始值为 28。

(2) 判断所输入的年份是否是闰年? 若是闰年,修改 2 月份天数为 29。因为平年 2 月是 28 天,闰年 2 月是 29 天。

(3) 累加该年该月之前的各月份天数和,存入变量 sum,sum 的初始值为 0。

(4) sum+输入的天数,即为题目所求。

参考代码如下:

```

#include <stdio.h>
int main()
{
    //存放每个月的天数的数组,2 月份的天数初始化为 28 天
    int days[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    int year,month,day;
    int i,sum=0;
    printf("Please enter year,month,day:\n");
    scanf("%d%d%d",&year,&month,&day);
    if(year%400==0 || (year%4==0 && year%100!=0)) //闰年
        days[1]=29;                                //修改 2 月份的天数为 29 天
    for(i=1;i<month;i++)
        sum+=days[i-1];
    sum+=day;
    printf("%d 年%d 月%d 日是这一年的第%d 天\n",year,month,day,sum);
    return 0;
}

```

3. 解题思路: 从数组的第一个元素开始顺序查找,将要找的数与数组中的每个元素逐一比较,直到找到为止(如果数组中无此数,则应找到最后一个数,然后判断“找不到”)。

参考代码如下:

```

#include <stdio.h>
#define N 10

```

```

int main()
{
    int a[N]={7,20,10,3,9,6,12,15,21,32};
    int i,x;
    printf("Please enter x:");
    scanf("%d",&x);
    for(i=0;i<N;i++)                //顺序查找,从数组的第一个元素开始比较
        if(a[i]==x)                  //找到
        {
            printf("Find,the order of x is: %d",i+1);
            break;
        }
    if(i>=N)                          //找不到
        printf("Not find! \n");
    return 0;
}

```

4. 解题思路

- (1) 使用表达式“rand() % 100 + 1”产生一个大小为 1~100 的随机数。
- (2) 产生 10 个随机数并存入长度为 10 的整型数组 a。
- (3) 利用改进的冒泡排序对数组 a 中的元素进行升序排序。

冒泡排序(Bubble sort)是基于交换排序的一种算法。它是依次两两比较待排序元素,若为逆序(递增或递减)则进行交换。将待排序元素从左至右比较一遍称为一趟“冒泡”。每趟冒泡都将待排序列中的最大关键字交换到最后(或最前)位置,直到全部元素有序为止。若本次冒泡处理过程中,没有进行任何交换,说明序列已有序,则停止交换。这就是改进的冒泡算法的处理思想。可以通过设置标志变量来判断是否提前完成排序。

参考代码如下:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 10
int main()
{
    int a[N];                        //用于保存 10 个随机数
    int i,j,temp,flag;               //flag 为标志变量
    srand((unsigned int)time(NULL)); //设置当前时间为种子
    for(i=0; i<10;i++)
    {
        a[i]=rand() %100 +1;        //产生 1~100 的随机数
        printf("%d ",a[i]);         //打印生成的随机数
    }
    printf("\n");
}

```

```

for(i=N-1;i>0;i--)          //改进的冒泡排序
{
    flag=0;                  //每一趟排序,首先置标志变量 flag 的值为 0
    for(j=0;j<i;j++)
        if(a[j]>a[j+1])
        {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
            flag=1;          //如果发生数据的交换,置标志变量 flag 的值为 0
        }
    //如果一趟排序过程中,未发生数据交换,说明数据已有序,则退出循环,提前结束排序
    if(flag==0)    break;
}
//输出排序后的数组 a
for (i =0; i<10; i++)
{
    printf("%d ",a[i]);
}
printf("\n");
return 0;
}

```

5. 解题思路: 假设要插入的数为 x , 从数组的最后一个元素开始搜索 x 的插入位置, 如果数组中的元素比 x 大, 则将此数组元素后移一个位置, 并继续往前搜索, 直到找到一个小于或等于 x 的元素为止, x 插入此元素的后面这个位置, 该位置已腾空, 直接插入 x 即可。如果数组中的所有元素都比 x 大, 则 x 插入到数组的最前面(下标为 0 的位置)。

参考代码如下:

```

#include <stdio.h>
#include <time.h>
#define MAXSIZE 10
int main()
{
    int a[MAXSIZE]={10,20,30,45,45,50};          //非递减有序的数组 a
    int size=6;                                  //数组 a 的长度为 6
    int i,x;
    printf("Please enter x:");
    scanf("%d",&x);
    //查找 x 的插入位置,从后往前查找,边查找边后移比 x 大的元素
    i=size-1;
    while(i>=0 && a[i]>x)
    {
        a[i+1]=a[i];

```

```

        i--;
    }
    a[i+1]=x;                //插入 x
    size++;                  //数组长度增加 1
    for(i=0;i<size;i++)printf("%d ",a[i]);    //输出数组
    printf("\n");
    return 0;
}

```

6. 解题思路: 该题采用逐一判断的方式, 周边元素的下标一定有 0 或 4, 并且只要下标有一个 0 或 4, 则它一定是周边元素。

参考代码如下:

```

#include <stdio.h>
int main()
{
    int a[5][5]={0,1,2,7,9,1,11,21,5,5,2,21,6,11,1,9,7,9,10,2,5,4,1,4,1};
    int i,j;
    int s=0;
    printf("*****The array*****\n");
    for(i=0;i<5;i++)
    {
        for(j=0;j<5;j++)
        {
            printf("%4d",a[i][j]);
        }
        printf("\n");
    }
    for(i=0;i<5;i++)
        for(j=0;j<5;j++)
            if(i==0||i==4||j==0||j==4)
                s=s+a[i][j]*a[i][j];
    printf("*****The result*****\n");
    printf("The sum is:%d\n",s);
    return 0;
}

```

7. 解题思路: 以字符串中间的字符为中心, 比较其两侧对称的字符是否相等。

参考代码如下:

```

#include <stdio.h>
#include <string.h>
#define N 10
int main()
{

```