

**课程 实 验 报 告**

**课程名称： C语言程序设计课程实验**

修改部分用蓝字表示

**专业班级： 自实1901**

**学 号： U201916457**

**姓 名： 张皓然**

**指导教师：**

**报告日期： 2020年4月13日**

**人工智能与自动化学院**

# 实验1用实例说明函数间数据传递的方法

## 实验目的

（1）通过实际的程序，函数间数据传递的方法，包括使用函数参数在函数间传递数据（调用值的传递，地址传递），使用返回值传递数据，使用全局变量传递数据。

## 实验内容

（1）编写相关代码。

（2）运行程序，总结分析结果。

## 代码

1. #include<stdio.h>
2. **int** quanju;
3. **int** fun(**int**, **int**\*);
4. **void** main()
5. {
6. **int** hancan1 = 1, hancan2 = 1;
7. **int**\* dizhi;
8. **int** fanhui;
9. dizhi = &hancan2;
10. fanhui = fun(hancan1, dizhi);
11. printf("hancan1=%d,hancan2=%d,\*dizhi=%d,fanhui=%d,quanju=%d",hancan1,hancan2,\*dizhi,fanhui,quanju);
12. }
13. **int** fun(**int** hancan1, **int**\* dizhi)
14. {
15. hancan1++;
16. \*dizhi += 2;
17. quanju++;
18. **return** hancan1;
19. }

## 实验过程

输入代码，按CTRL\_F9运行，发现编译连接运行正常，按alt\_F5打开用户界面观察结果。

用户界面显示的结果为：

hancan1=1,hancan2=3,\*dizhi=3,fanhui=2,quanju=1

## 实验结果与分析

想要在函数间传递数据，我们有以下三种方法：使用函数参数在函数间传递数据（调用值的传递，地址传递），使用返回值传递数据，使用全局变量传递数据。

上述代码中，hancan1和dizhi是使用函数参数在函数间传递数据，hancan1是调用值的传递，仅仅把hancan1变量所具有的值传递到了函数fun内，不过由于auto型变量作用域以及寿命的问题，在函数中虽然对hancan1进行了一次自增，但却不会影响到main函数中hancan1的值，所以在最后输出hancan1的值时，它依旧是1。dizhi传递前我们将指针变量dizhi指向了hancan2，fun函数获得了dizhi的值，其实就是hancan2的地址，所以fun函数可以间接改变hancan2的值以及\*dizhi的值。在fun函数的最后，我们让函数返回了hancan1，注意，这个hancan1是fun函数中的hancan1，与main函数中名称相同的变量无关。当fun函数运行结束后，hancan1消失，fanhui获得fun函数的返回值，也就是fun函数中hancan1的值。此外，在fun函数中对quanju也进行了自增操作，这属于使用全局变量传递数据。因为全局变量直至整个程序运行结束后才会消失，所以fun函数内对quanju自增，main函数中也能够又所反应。不过全局变量破坏了程序的结构性，所以我们尽量避免使用全局变量。

## 心得与体会

函数间数据的传递是构建一个好函数的基础，三种主要的方法各有优劣，使用全局变量传递虽然方便，但却破坏了程序的结构性。使用返回值也很简单，但只能传递一个值。使用函数参数传递，调用值传递只能向“下层”传递数据，在我看来，地址传递没有以上方法的缺点，是最佳的函数间数据传递的方法。在实际编程的过程中，我们应该参考实际需求，选择最合适，最简单的数据传递方法。

# 实验2 上机验证完成第四章习题3、4 题

## 习题3

求方程ax2+bx+c=0的根，用三个函数分别求不相等实根，相等实根，共轭复根，并在函数中输出结果，a,b,c从主函数输入。

## 代码

1. **void** fun2(**double**, **double**);
2. **void** fun3(**double**, **double**, **double**);
3. **int** main()
4. {
5. **double** a, b, c;
6. **double** sign;
7. scanf\_s("%lf %lf %lf", &a, &b, &c);
8. **if** (a == 0)
9. {
10. printf("\nx=%lf", -c / b);
11. **return** 0;
12. }
13. sign = b \* b - 4 \* a \* c;
14. **if** (sign > 0)
15. {
16. fun1(a, b, sign);
17. }
18. **else** **if** (sign == 0)
19. {
20. fun2(a, b);
21. }
22. **else**
23. {
24. fun3(a, b, sign);
25. }
26. **return** 0；
27. }
28. **void** fun1(**double** a, **double** b, **double** sign)
29. {
30. **double** x1, x2;
31. x1 = (-b + sqrt(sign)) / (2 \* a);
32. x2 = (-b - sqrt(sign)) / (2 \* a);
33. printf("\nx1=%f,x2=%f", x1, x2);
34. }
35. **void** fun2(**double** a, **double** b)
36. {
37. **double** x;
38. x = -b / (2 \* a);
39. printf("\nx1=x2=%f", x);
40. }
41. **void** fun3(**double** a, **double** b, **double** sign)
42. {
43. **double** shibu, xubu;
44. shibu = -b / (2 \* a);
45. xubu = sqrt(-sign) / (2 \* a);
46. printf("\nx1=%f+i%f,x2=%f-i%f", shibu, xubu, shibu, xubu);
47. }

## 测试集：

|  |  |  |
| --- | --- | --- |
| a | b | c |
| 1 | 2 | 3 |
| 2.4 | 2.1 | -5.1 |
| 0 | 2 | 4.5 |
| 2.3 | 0 | 4 |

## 输出结果

第一组

x1=-1.000000+i1.414214,x2=-1.000000-i1.414214

第二组

x1=1.084474，x2=-1.959474

第三组

x=-2.250000

第四组

x1=-0.000000+i1.318761,x2=-0.000000-i1.318761

## 代码及结果的分析

本段程序实际就是利用求根公式计算一元二次方程的解。需要注意，如果用户输入的a=0,那么这就不再是一个二次方程了，求根公式的分母就变成了0，如果还按原先解二次方程的方法处理，就会出错。所以我在程序录入a,b,c的值后，先用if语句判断a是否为零，若为零则按一元一次方程的解法解答，并用return命令结束程序。

## 习题4

写一个函数，判断一个自然数是否为素数，编写判断函数和测试主函数。

## 代码

1. #include<stdio.h>
2. **int** fun(**int**);
3. **int** main()
4. {
5. **int** x;
6. scanf\_s("%d", &x);
7. **if** (x == 1 || x == 0)
8. {
9. printf("\nnot sushu");
10. **return** 0;
11. }
12. **if** (fun(x))
13. {
14. printf("\nsushu");
15. }
16. **else** {
17. printf("\nnot sushu");
18. }
19. **return** 0;
20. }
21. **int** fun(**int** x)
22. {
23. **int** i;
24. **int** sign = 1;
25. **for** (i = 2; i < x / 2; i++)
26. {
27. **if** (x % i == 0)
28. {
29. sign--;
30. **break**;
31. }
32. }
33. **return** sign;
34. }

## 测试集

0 1 2 3 4 9 20 88 97

## 输出结果

各测试数据的输出结果依次为

not sushu

not sushu

sushu

sushu

not sushu

not sushu

not sushu

not sushu

sushu

## 代码及结果的分析

首先应当明确素数的定义，素数指指在一个大于1的自然数中，除了1和此整数自身外，没法被其他自然数整除的数。也就是说，0和1虽然是自然数，但却不在素数考虑的范畴里。所以在输入x后，先判断一下x是不是0或者1，如果是，直接输出不是素数并结束程序。如果不是，再进行后续的操作。我选择了用取余运算来判断是否为素数，同时注意到，取余运算进行到x/2即可，这样可以稍微简化程序的运行过程。

## 心得与体会

在检验自己的程序是否完备时，最好的办法就是提供一个完备的测试集，输入不同特点的数据，判断程序是否能够正常运行。

# 实验3 掌握一维数组和多维数组的定义和应用，熟练运用一维数组和多维数组的初始化方法，用一个实例进行说明；体会二维数组的降维特点

## 实验目的

（1）通过实际的程序掌握一维数组和多维数组的定义与运用，体会二维数组的降维特点。

## 实验内容

编写程序，编译运行，进行分析总结。

## 代码

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** a1[] = { 1,2,3,4 };
5. **int** a2[4] = { 1,2,3 };
6. **int** b1[][2] = { {1,2},{1,2}, {1,2} };
7. **int** b2[2][2] = { {1},{1,2} };
8. printf( "a2[3]=%d,b2[0][1]=%d",a2[3],b2[0][1] );
9. }

## 实验结果

用户界面输出的结果为

a2[3]=0,b2[0][1]=0

## 实验结果分析

一维数组的定义方式为 存储类型+数据类型+数组名+常量表达式，其中存储类型允许缺省，缺省时存储类型就是auto，特点与变量的存储类型类似。数据类型指的是数组中存储的各个元素的数据类型。数组名实际是该数组中数据存储区域的首地址，要想使用数组中的某个元素，必须加上标号。同时要注意，数组名本身实际是一个地址常量，不能对单独的数组名赋值或是进行其他的运算。数组作为整体不能参加运算，只有数组中的各个元素才可以参加运算。再定义数组时，数组的大小只能由常量或是符号常量表示！当到底需要多少数据并不确定的时候，我们可以适当把数组的元素个数扩大。

多维数组的定义与之类似，以二维数组为例，如a[2][2],说是二维数组，但实际上应该明白该数组的数据在内存中依旧按照一维的方式排列。可以把a[2][2]看成一个一维的数组，其数组中的元素是3个数组。第一个元素是数组a[0][2],数组名为a[0]，即该数组中的元素为a[0][0],a[0][1],第二个元素是数组a[1][2],数组名为a[1],即该数组中的元素为a[1][0],a[1][1].

关于数组的初始化，即对数组中各个元素的初始化，用大括号括起来，注意语句结束后的分号。如果对数组的每个元素都进行了初始化，我们甚至可以省去数组定义时的常量表达式。但是如果我们没有定义全部的数组元素，那常量表达式就不能省去。同时，在初始化部分元素后，未被初始化的数组元素自动变成0.

至于二维数组的初始化，如果所有元素都将被赋初值，那么一维可以省略，二维不可以省略，即例如a[][2]={{1,2},{1,2},{1,2}}.注意，左式中花括号内的花括号可以省略，但为了层次性与结构性，我们不建议这样做。如果只对部分元素初始化，则常量表达式不可省略，同时在初始化后，未被初始化的元素自动赋0.

此外注意，数组元素自身的标号从0开始，即a[3]的元素为a[0],a[1],a[2].

# 实验4 用一维数组处理10个数的排序，体会数组和离散变量的使用的区别，为什么数组带来了程序实现的简捷

## 实验目的

（1）体会数组和离散变量的使用的区别，分析数组的便捷之处。

## 代码

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** a[10],t;
5. **int** i, j;
6. **for** (i = 0; i < 10; i++)
7. {
8. scanf("%d", &a[i]);
9. }
10. **for** (i = 0; i < 9; i++)
11. {
12. **for** (j = i + 1; j < 10; j++)
13. {
14. **if** (a[i] > a[j])
15. {
16. t = a[i];
17. a[i] = a[j];
18. a[j] = t;
19. }
20. }
21. }
22. **for** (i = 0; i < 10; i++)
23. {
24. printf("%d\t", a[i]);
25. }
26. }

## 测试集

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 第一组 | 1 | 3 | 4 | 0 | 9 | 5 | 7 | 3 | 2 | 1 |
| 第二组 | 5 | 5 | 5 | 3 | -4 | 7 | 21 | 60 | 32 | 0 |
| 第三组 | 1 | 0 | 1 | 15 | 14 | 2 | 14 | 15 | 17 | 19 |
| 第四组 | 20 | 21 | 0 | -20 | 21 | 22 | 60 | 70 | 100 | 11 |

## 实验结果

各组的结果依次为

0 1 1 2 3 3 4 5 7 9

-4 0 3 5 5 5 7 21 32 60

0 1 1 2 14 14 15 15 17 19

-20 0 11 20 21 21 22 60 70 100

## 实验结果分析

该程序即为选择排序法，即先确定第一位（最小值），然后第一位已排好，再确定第二位（次小值）……通过多次位置的互换，最终实现排序的操作。

相较于离散变量，我认为数组在处理这类排序问题时的优点在于，数组能够轻松将多个变量“打包”，并用数来代表这些数据。如果不使用数组的话，对十个数排序就需要定义从x1到x10一共十个变量，而且每一次的比较也不能用for函数表示成一组循环，实在是非常冗繁。

## 心得与体会

在使用for循环的时候，应当十分注意循环开始的初值条件，以及循环结束的条件，当对一个数组进行遍历的操作时，我们常常令i=0作为初值条件，数组有10个元素，则令循环条件为i<10，这样就非常简洁的实现数组元素的遍历操作了。

# 实验5 掌握选择排序和冒泡排序这两种数据排序方法，体会选择法和冒泡法排序的区别，两种不同选择法实现方法的异同，用实例说明。说明各种不同排序中两层循环循环的边界（起点和终点）异同

## 实验目的

（1）掌握选择排序法和冒泡排序法。

（2）分析两种排序法的区别。

（3）说明两种排序法两层循环的边界的异同。

## 选择排序法的思路

选择排序法又细分为两种方法，一种是边比较边交换，一种是边比较边设立标记并在最后交换。两种方法的区别不大，实际都是逐个确定每一位应该是哪一个数，我们以边比较边交换的方法为例解释选择排序法的思路。

以1 3 4 5 2这五个数的排序为例。

先将1与3比较，1更小，不交换；1与4比较，1更小，不交换；1与5比较，1更小，不交换；1与2比较，1更小，不交换。于是第一个数就应该是1.

第二组比较，3与4比较，3更小，不交换；3与4比较，3更小，不交换；3与2比较，2更小，交换，现在2移动到了第二位，3移动到了第五位。于是第二个数应该是2.

第三组比较，与以上两组类似，最终4与3交换，于是第三位应该是3.

第四组比较，类似的过程，最终5与4交换，于是第四位应该是4.

因为前四位都已经确定好了，所以第五位就不需要再进行比较了。

## 冒泡排序法的思路

以1 3 4 5 2这五个数的排序为例。

第一组比较，1和3比较，3更大，3放在后面（不交换）；3和4比较，4更大，4放在后面（不交换）；4和5比较，5更大，5放在后面（不交换）；5和2比较，5更大，5放在后面（交换），于是第五位应该是5.

第二组比较，与第一组类似，不过比较三次即可，从而确定第四位应该是4.

再进行第三组和第四组的比较，依次确定第三位应该是3，第二位应该是2。因为从第二位到第五位都已经确定好了，所以第一位无需进行比较，第一位就是1.

## 选择排序法代码（边找边交换）

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** a[5],t;
5. **int** i, j;
6. **for** (i = 0; i < 5; i++)
7. {
8. scanf("%d", &a[i]);
9. }
10. **for** (i = 0; i < 4; i++)
11. {
12. **for** (j = i + 1; j < 5; j++)
13. {
14. **if** (a[i] > a[j])
15. {
16. t = a[i];
17. a[i] = a[j];
18. a[j] = t;
19. }
20. }
21. }
22. **for** (i = 0; i < 5; i++)
23. {
24. printf("%d\t", a[i]);
25. }
26. }

## 冒泡排序法代码

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** i, j;
5. **int** a[5], t;
6. **for** (i = 0; i < 5; i++)
7. {
8. scanf\_s("%d", &a[i]);
9. }
10. **for** (i = 0; i < 4; i++)
11. {
12. **for** (j = 0; j < 4 - i; j++)
13. {
14. **if** (a[j] > a[j + 1])
15. {
16. t = a[j];
17. a[j] = a[j + 1];
18. a[j + 1] = t;
19. }
20. }
21. }
22. **for** (i = 0; i < 5; i++)
23. {
24. printf("%d\t", a[i]);
25. }
26. }

## 实验过程

输入两种排序法，对1 3 4 5 2这组数据排序。

排序的结果分别为：

选择排序法：

1 2 3 4 5

冒泡排序法

1 2 3 4 5

## 两种排序法的分析

选择排序法是从低往高排序，即先确定最小值，再确定次小值……冒泡排序法是从高往低排序，即先确定最大值，再确定次大值……至于相同点，分析两段代码我们可以发现，这两种排序方法都需要两重循环，都经历N-1组排序，且随着排序的进行，接下来每组排序内部的排序次数会逐渐减少。同时，这两种排序方法也都需要借助一个变量t作为过渡，进行交换的操作。

至于两种排序方法的循环边界。以本实验中用到的具体程序为例，选择排序法，第一层循环进行5次，而第二层循环每次的起点受到第一层循环的影响。因为第一层循环每次可以固定一个较小的数，第二层循环就不必再考虑这个数了，于是第二层循环的起点随着第一层循环的进行而不断向右移动，终点无变化。冒泡排序法，第一层循环进行5次，而第二层循环的终点受到第一层循环的影响，因为第一层循环每次可以固定一个较大的数，第二层循环不需要再考虑这个数，所以第二层循环的终点随着第一层循环的进行而不断向左移动，起点无变化。

# 实验6 上机验证完成第五章习题11、13题

## 习题11

求二维整型数组a[N][N]中每一行的数的和，并放在另一个一维数组b[N]中。

## 代码（以N为5为例）

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** a[5][5], b[5] = { 0,0,0,0,0 };
5. **int** i, j;
6. **for** (i = 0; i < 5; i++)
7. {
8. **for** (j = 0; j < 5; j++)
9. {
10. scanf\_s("%d", &a[i][j]);
11. }
12. }
13. **for** (i = 0; i < 5; i++)
14. {
15. **for** (j = 0; j < 5; j++)
16. {
17. b[i] += a[i][j];
18. }
19. }
20. **for** (i = 0; i < 5; i++)
21. {
22. printf("%d\t", b[i]);
23. }
24. }

## 测试集

|  |  |
| --- | --- |
| 第一组 | 1 1 1 1 1  2 2 2 2 2  3 3 3 3 3  4 4 4 4 4  5 5 5 5 5 |
| 第二组 | 0 0 0 0 0  1 2 3 4 5  9 9 2 1 0  3 2 1 9 8  6 10 20 3 1 |
| 第三组 | 1 7 9 8 2  3 1 4 1 5  2 7 1 8 2  6 0 9 4 3  2 3 4 9 9 |

## 实验结果

各种实验结果依次为

5 10 15 20 25

0 15 21 23 40

27 14 20 24 27

## 代码及结果分析

深刻理解二维数组降维的特点，实际就是计算数组a[0]五个元素的的和并赋给b[0],计算数组a[1]五个元素的和并赋给b[1]，注意要事先对b[5]初始化，令b的每一个元素都是0.

## 习题13

输出二维数组中行为最大，列为最小的元素（成为鞍点）及其位置。如果不存在任何鞍点，则也输出相应的信息。

## 代码（以N为5为例）

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** a[5][5];
5. **int** i, k, m;
6. **int** max;
7. **int** sign = 0;
8. **for** (i = 0; i < 5; i++)
9. {
10. **for** (j = 0; j < 5; j++)
11. {
12. scanf\_s("%d", &a[i][j]);
13. }
14. }
15. **for** (i = 0; i < 5; i++)
16. {
17. sign = 0;
18. max = 0;
19. **for** (k = 1; k < 5; k++)
20. {
21. **if** (a[i][max] < a[i][k])
22. {
23. max = k;
24. }
25. }
26. **for** (m = 0; m < 5; m++)
27. {
28. **if** (a[i][max] > a[m][max])
29. {
30. sign++;
31. **break**;
32. }
33. }
34. **if** (sign)
35. {
36. printf("hang %d no andian\n", i + 1);
37. }
38. **else** {
39. printf("hang %d,lie %d,andian %d\n", i + 1, max + 1, a[i][max]);
40. }
42. }
43. **return** 0;
44. }

## 测试集

|  |  |
| --- | --- |
| 第一组 | 6 5 3 2 1  9 1 2 6 8  8 1 6 7 9  7 4 1 4 6  7 5 1 5 3 |
| 第二组 | 11 3 5 6 9  12 4 7 8 10  10 5 6 9 11  8 6 4 7 2  15 10 11 20 25 |
| 第三组 | 1 2 3 4 5  2 3 4 5 6  3 4 5 6 7  4 5 6 7 8  5 6 7 8 9 |

## 实验结果

三组测试数据的结果依次为

第一组：

hang 1,lie 1,andian 6

hang 2 no andian

hang 3 no andian

hang 4 no andian

hang 5 no andian

第二组：

hang 1 no andian

hang 2 no andian

hang 3 no andian

hang 4,lie 1,andian 8

hang 5 no andian

第三组：

hang 1,lie 5,andian 5

hang 2 no andian

hang 3 no andian

hang 4 no andian

hang 5 no andian

## 代码及结果分析

鞍点有两个条件，首先该点在该行最大，同时在该列最小。所以代码中对每一行，先找出该行的最大值，然后再判断该最大值是不是对应列的最小值。该题有很多循环，一定要看清循环从哪开始，从哪结束，循环的初值条件是什么。在出错后，使用单步调试的技巧分析到底是哪里错了。

# 实验7 指针变量的存储类型和数据类型的具体含义？指针变量使用需要注意的事项有哪些？

## 实验目的

（1）掌握指针变量的定义，了解指针变量的注意事项

## 实验内容

编写相关程序，编译运行，分析结果，结合书本总结指针相关知识要点

## 代码

1. #include<stdio.h>
2. **int** main()
3. {
4. **int**\* p;
5. **char** cc;
6. **char**\* pc = &cc;
7. **static** **int**\* p0;
8. **if** (p0 == NULL)
9. {
10. printf("yes");
11. }
12. **if** (p0 == 0)
13. {
14. printf("\nyes");
15. }
16. }

## 结果分析

指针变量的存储类型指的是指针本身，作为一个地址，存储的方式。与普通的变量类似，指针变量的存储类型有auto，register，static，extern。不同的存储类型的特性与普通变量的完全相同。指针变量的数据类型指的是指针所指向的变量的数据类型，比如int\* p=&a，则说明指针变量p指向了int型变量a。

使用指针时需要注意，\*是读指针（即取内容），\*p就是p指针指向的变量的值。&是取地址符，&a就是变量a的地址。但是，在说明性的语句中，我们要明白，\*也变成了一个说明，而非运算。比如int\* p=&a,应该理解为，首先定义一个int\* p，（指针变量），然后初始化令p=&a。而不是定义一个int\* p，然后令\*p=&a。这两者是截然不同的，在编写程序时建议在定义指针变量时，\*和p间加一个空格，以提醒自己\*只是单纯的说明接下来要定义一个指针变量，而不是运算。

另外，指针只能指向内存中的变量。register型变量不存储在内存中，所以不能把register型变量的地址赋给指针。不过指针变量自身却可以存储在寄存器中。

该段代码表明，静态指针变量初始化为零，其实就是空指针。且空指针的值就是0.

# 实验8 分别用实例说明一维数组和一般指针变量形式使用的互换性，说明指针变量使用数组形式的前提以及数组用指针形式有哪些限制（哪些运算不能进行）？

## 实验目的

（1）掌握一维数组和一般指针变量的互换性

（2）掌握指针变量使用数组形式的前提，数组用指针形式的限制

## 代码（参考教材第六章例6.6，并适当改写）

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** i, \* pa;
5. **int** a[] = { 2,4,6,8,10 };
6. pa = a;
7. **for** (i = 0; i < 5; i++)
8. {
9. printf("a[%d]:%d\t", i, pa[i]);
10. }
11. printf("\n");
12. **for** (i = 0; i < 5; i++)
13. {
14. printf("\*(pa+%d):%d", i, \*(a + i));
15. }

## 实验结果

a[0]:2 a[1]:4 a[2]:6 a[3]:8 a[4]:10

\*(pa+0):2\*(pa+1):4\*(pa+2):6\*(pa+3):8\*(pa+4):10

## 实验分析

数组名实际上就是数组第一位元素的地址，即数组名本身就是一个指针。代码中将指针变量pa指向数组a，即pa=a,此时pa与a基本完全相同，既可以把指针变量pa看成数组，输出pa[i],也可以把a看成一个地址，输出\*(a+i).这就说明的一维数组和一般指针变量的互换性。

不过，一维数组与指针变量并非完全相同，他们也有一些区别。最重要的区别就是指针变量是变量，一维数组的数组名作为一个指针，是常量。pa可以被赋予不同的地址，但a不能，a自定义之初指向的位置就已经固定下来了。此外，要想事先数组和指针的互换性，就要首先确保指针pa指向了数组a，即pa指向了数组a的第一个元素的地址。在经过一轮循环后，有可能指针pa就不再指向数组a了，此时应当及时“归零”。

# 实验9 一维数组在函数间的传递，需要几个参数，并进行实例说明

## 实验目的

（1）掌握一维数组在函数间传递的要点

## 代码

1. #include<stdio.h>
2. **void** fun(**int** a[], **int**);
3. **int** main()
4. {
5. **int** a[10] = { 1,2 };
6. **int** n = 10;
7. fun(a, n);
8. }
9. **void** fun(**int** a[], **int** n)
10. {
11. **int** i;
12. **for** (i = 0; i < n; i++)
13. {
14. printf("%d\t", a[i]);
15. }
16. }

## 实验结果

1 2 0 0 0 0 0 0 0 0

## 实验分析

该代码将数组传递到fun函数中，然后在fun函数中将数组元素依次输出。

在讨论数组传递到函数中需要提供几个参数前，我们需要先弄清楚，fun(int a[])是将什么传递给了函数。这实际上是将数组a的首个元素的地址传递给了函数fun。a就是指针，实际就是传递了一个指针。于是，传递a给函数，就相当于告诉了函数，这个数组的地址在哪里。那还需要什么数据呢？要描述一个数组，我们还需要这个数组元素的个数，这个在数组地址里是看不出来的，所以我们需要额外再增加一个变量n，将n也传递给函数fun，这样就将数组完整地传递给一个函数了。

# 实验10 上机验证完成第六章例6.9

## 实验目的

（1）分析验证第六章6.9，熟悉数组的使用，函数的使用。

## 例6.9题目

编写三个函数分别完成指定一维数组元素的数据输入，求一维数组的平均值，求一维数组的最大值和最小值。由主函数完成这些函数的调用。

## 代码

1. #include<stdio.h>
2. **void** input(**float**\*, **int**);
3. **float** average(**float**\*, **int**);
4. **void** maxmin(**float**\*, **int**, **float**\*, **float**\*);
5. **int** main()
6. {
7. **float** data[10];
8. **float** aver, max, min;
9. **float**\* num = data;
10. input(data, 10);
11. aver = average(data, 10);
12. maxmin(num, 10, &max, &min);
13. printf("aver=%f\n", aver);
14. printf("max=%f,min=%f\n", max, min);
15. }
16. **float** average(**float**\* pdata, **int** n)
17. {
18. **int** i;
19. **float** avg;
20. **for** (avg = 0, i = 0; i < n; i++)
21. {
22. avg += pdata[i];
23. }
24. avg /= n;
25. **return** avg;
26. }
27. **void** input(**float**\* pdata, **int** n)
28. {
29. **int** i;
30. printf("please input array data:");
31. **for** (i = 0; i < n; i++)
32. {
33. scanf("%f", pdata + i);
34. }
35. }
36. **void** maxmin(**float**\* pdata, **int** n, **float**\* pmax, **float**\* pmin)
37. {
38. **int** i;
39. \*pmax = \*pmin = pdata[0];
40. **for** (i = 1; i < n; i++)
41. {
42. **if** (\*pmax < pdata[i])
43. {
44. \*pmax = pdata[i];
45. }
46. **if** (\*pmin > pdata[i])
47. {
48. \*pmin = pdata[i];
49. }
50. }
51. }

## 测试集

|  |
| --- |
| 1 2 3 4 5 6 7 8 9 10 |
| 2.1 0 8 1.4 1.2 9 7 1 1.5 2.9 |
| 1.4 -5 4 2 1 0 -7 8 9 10 |
| 1.1 1 2 9 60 20 4.3 8.3 10 42.1 |

## 实验结果

第一组

aver=5.500000

max=10.000000,min=1.000000

第二组

aver=3.410000

max=9.000000,min=0.000000

第三组

aver=2.340000

max=10.000000,min=-7.000000

第四组

aver=15.780001

max=60.000000,min=1.000000

## 代码及结果分析

在本例子中，除main函数外，我们需要再定义3个函数。input函数用于完成一维数组的数据输入，既然要给一个一维数组赋值，那么就需要将数组的信息传递到input函数中。在之前的实验中我们已经得知，数组在函数间的传递需要两个参数，一个是数组的首地址，即数组的数组名，用于给数组定位，另一个是数组的元素多少，用于确定数组整个的大小。于是我们确定input的函数原型应该是void input(float\*,int);在实际的代码实现中，我们又定义了一个指针pdata，这样使数据之间的关系更加清楚。

第二个函数需要求一维数组的平均值。求一维数组的平均值，那肯定需要把一维数组传递给函数了，所以要传递数组的首地址以及数组元素的个数。此外，求出平均值后，为了在主函数中输出，就需要将这个平均值传递给主函数。因为只需要传递一个值给主函数，所以用返回值传递显然更加方便，于是函数原型可以写为float average(float\*,n);该函数最后把计算出的平均值return。

第三个函数需要求一维数组的最大值和最小值。首先肯定也是要把一维数组传递给函数，传递数组的首地址和数组元素的个数。此外，得到最大值和最小值后需要将他们传递回主函数。因为需要传递两个数，所以不能用返回值的方式传递。因为全局变量会破坏程序的结构性，所以我们也不采用这种方法。于是我们选择用函数参数传递（地址传递）。于是在主函数中先定义min，max两个变量，然后再定义pmin，pmax两个指针，并将这两个指针指向min和max。于是可以将这个函数的原型写为void maxmin(float\*,int,float\*,float\*).

# 实验11 输入n个整数，排序后输出。排序的原则由函数的一个参数决定，参数值为1，按递减顺序排序，否则按递增顺序排序。用函数实现排序，main测试函数完成数的输入、排序函数的调用以及排序后的输出

## 实验目的

（1）完成该题目

（2）体会数组以及函数的运用

## 代码

1. #include<stdio.h>
2. **void** rank(**int**, **int**, **int**\*,**int**\*);
3. **int** main()
4. {
5. **int** a[100];
6. **int** b[100];
7. **int** i;
8. **int** n, sign;
9. printf("please input n and rule:\n");
10. scanf\_s("%d %d", &n, &sign);
11. printf("please input the array:\n");
12. **for** (i = 0; i < n; i++)
13. {
14. scanf\_s("%d", &a[i]);
15. }
16. rank(sign, n, a, b);
17. **for** (i = 0; i < n; i++)
18. {
19. printf("%d ", b[i]);
20. }
21. }
22. **void** rank(**int** sign, **int** n, **int**\* a, **int**\* b)
23. {
24. **int** i,j;
25. **int** t;
26. **for** (i = 0; i < n - 1; i++)
27. {
28. **for** (j = i + 1; j < n; j++)
29. {
30. **if** (a[i] > a[j])
31. {
32. t = a[i];
33. a[i] = a[j];
34. a[j] = t;
35. }
36. }
37. }
38. **if** (sign == 1)
39. {
40. **for** (i = n - 1, j=0; i >= 0; i--, j++)
41. {
42. b[j] = a[i];
43. }
44. }
45. **else**
46. {
47. **for** (i = 0; i < n; i++)
48. {
49. b[i] = a[i];
50. }
51. }
52. }

## 测试集

|  |  |  |
| --- | --- | --- |
| n | rule | 数组 |
| 5 | 1 | 1 2 3 4 5 |
| 10 | 0 | 9 10 1 4 3 2 5 8 7 6 |
| 5 | 12 | 8 7 10 8 2 |
| 5 | 1 | -3 8 21 3 0 |

## 实验结果

第一组

5 4 3 2 1

第二组

1 2 3 4 5 6 7 8 9 10

第三组

2 7 8 8 10

第四组

21 8 3 0 -3

## 代码及结果分析

排序的方法我们已经很清楚了，有选择排序法和冒泡排序法。不过不论是选择排序法，还是冒泡排序法，他们排出来的结果都是从小到大。该实验要求我们额外输入一个参数，如果这个参数是1，就从大到小排序，如果不是1，就从小到大排序。我们把这个参数记作rule。在本段代码中，我选择的方式是，定义两个数组，a[100],b[100].先把数据安放在数组a中，并用选择排序法的便比较边交换方法进行从小到大的排序。排序结束后，再判断rule是否为1，如果为1，则将a[100]中的元素逆序依次赋给b[100].如果不为1，则按顺序将a[100]中的元素赋给b[100],最终在main函数中输出数组b的值。

此外还要注意，该题目中数组的大小是由用户输入的n自定义的。但是在C语言中，定义数组的元素个数只能是一个常量表达式，要么是个常数，要么是个常量。不能用用户输入的n来规定数组的大小，那怎么办呢？对于这个问题，我们将数组定义的大一点即可，比如a[100],b[100],只要用户输入的n在100以内，就可以完成操作了。

补充：数组指针和指针数组的区别

## 数组指针

### 定义格式

int (\*ap)[10]; 加括号以和指针数组区分，表明ap和\*紧密相连，可以理解为，括号里的\*不同于其他定义语句里的\*（表示说明），int (\*ap)[10]的\*就是取内容符。先假定ap是个指针，然后\*取出ap的内容，告诉电脑这个位置要装一个10元素的数组。所以ap指向的不是单个的元素，而是一个10元素的数组。（这只用于帮助理解，实际计算机可能不是这么读的）

### 代码

int i=0,j=0;

int a[10][10]; //定义二维数组

int b[10]; //定义一维数组

int(\*ap)[10]; //定义数组指针

ap=(int(\*)[10])b; //将数组指针指向一维数组，需要将一维数组的首地址进行强制转换。转换的格式类似(int(\*)[10]),即定义时的格式，但不必加变量名。

ap=a; //将数组指针指向二维数组，二维数组的数组名本身就是一个数组指针常量，不需要额外强制的格式转换。

//下面同行的表达式是等价的

a, pa; //都是数组指针，只不过a是常量，pa是变量

\*a, \*pa, a[0], pa[0], \*(a+0), \*(pa+0), \*++pa;//表示二维数组a第一行一维数组的首地址。\*和++是同级运算符，从右往左运算。

\*(a+i), \*(pa+i), a[i], pa[i];

\*(a+i)+j, \*(pa+i)+j, &a[i][j], &pa[i][j], a[i]+j, pa[i]+j;

\*(\*(a+i)+j), \*(\*(pa+i)+j), a[i][j], pa[i][j], \*(a[i]+j), \*(pa[i]+j), (\*(a+i))[j], (\*(pa+i))[j];

## 指针数组

### 定义格式：

int\* ap[10];

### 用数组指针和指针数组处理二维数组

int a[3][4];

int (\*pa)[4];

int\* ppa[3];

pa=a;

ppa[0]=a[0], ppa[1]=a[1], ppa[2]=a[2];

//指针数组比二维数组低一级，数组指针和二维数组同级