

**课程 实 验 报 告**

**课程名称： C语言程序设计课程实验**

**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 二维数组的存储和初始化的特点，用一个实例进行说明；体会二维数组的降维特点

## 实验目的

掌握了解二维数组存储和初始化的特点，体会二维数组的降维特点。

## 设计思路与代码实现

### 设计思路

定义不同储存方式的二维数组，并以不同方式初始化。

设计一个函数将数组元素依次输出。

### 代码实现

1. #include<stdio.h>
2. **void** print(**int**[][2], **int**);
3. **int** main()
4. {
5. **int** a1[3][2] = { {1,2},{1,2},{1,2} };
6. **int** a2[][2] = { {1,2},{1,2},{1,2} };
7. **int** a3[][2] = { {1,2},{1},{1} };
8. print(a1, 3);
9. print(a2, 3);
10. print(a3, 3);
11. }
12. **void** print(**int** a[][2], **int** n)
13. {
14. **int** i, j;
15. **for** (i = 0; i < 3; i++)
16. {
17. **for** (j = 0; j < 2; j++)
18. {
19. printf("%d\t", a[i][j]);
20. }
21. printf("\n");
22. }
23. printf("\n");
24. }

## 测试集

无

## 实验过程

（1）在Borland C++ 3.1中输入程序

（2）保存程序，编译运行（CTRL\_F9）

（3）调出用户界面，查看实验结果（ALT\_F5）

## 实验结果

1 2

1 2

1 2

1 2

1 2

1 2

1 2

1 0

1 0

（经验证，结果符合预期。）

## 实验分析

### 二维数组存储和初始化的特点

二维数组的存储类型与变量的存储类型相似，分为auto型，register型，一般全局型，static型。注意，这里的存储，指的是数组中各个元素的存储类型。

初始化可以有两种方式，第一种形如{1,2,3,4}，第二种形如{{1,2},{3,4}}.两种初始化的方法区别并不大，但是后者更能体现二维数组的层次感，所以我们推荐使用第一种方法。

二维数组定义时a[3][2],中的3可以省去，这样的话，省去的数字由初始化提供的数据决定。

初始化并非一定需要定义全部的数组元素。可以只定义一部分元素，剩余的元素自动赋为0.

### 二维数组降维的特点

从二维数组的初始化看二维数组的降维度。如{{1,2},{1,2},{1,2}}可以将这个二维数组理解为：一个以数组为元素的一维数组（就像是以集合为元素的集合）。在计算机的内存中，二维数组的存储方式实际上也并不存在“二维”的概念，而是像一维数组那样依次排列的。先排列第1个一维数组的元素（1，2），再顺次排列第2个一维数组的元素……这些一维数组在内存中组合在一起，就成为了二维数组。

# 实验2 用二维数组实现两个矩阵的乘积

## 实验目的

编写一个程序，用二维数组完成两个矩阵的乘积。

## 设计思路与代码实现

### 设计思路

借助线性代数知识，两个矩阵乘积A\*B也是一个矩阵，该矩阵第i行第j列的元素为A矩阵第i行的元素与对应B矩阵第j列的元素的乘积的和。

A矩阵，B矩阵，以及结果矩阵，所以需要定义3个二维数组。

然后利用for循环，将A矩阵第i行的元素与B矩阵第j列的元素相乘后再相加，将这个值赋给结果矩阵。

为程序的简洁性，我们在main函数外设计input函数，fun函数，print函数。

### 代码实现（以3\*3矩阵为例）

1. #include<stdio.h>
2. **void** print(**int**[][3], **int**);
3. **void** input(**int**[][3], **int**);
4. **void** fun(**int**[][3], **int**[][3], **int**[][3], **int**);
5. **int** main()
6. {
7. **int** A[3][3], B[3][3], result[3][3];
8. input(A, 3);
9. input(B, 3);
10. fun(A, B, result, 3);
11. print(result, 3);
12. }
13. **void** print(**int** a[][3], **int** n)
14. {
15. **int** i, j;
16. **for** (i = 0; i < 3; i++)
17. {
18. **for** (j = 0; j < 3; j++)
19. {
20. printf("%d ", a[i][j]);
21. }
22. printf("\n");
23. }
24. printf("\n");
25. }
26. **void** input(**int** a[][3], **int** n)
27. {
28. **int** i, j;
29. **for** (i = 0; i < n; i++)
30. {
31. **for** (j = 0; j < 3; j++)
32. {
33. scanf\_s("%d", &a[i][j]);
34. }
35. }
36. }
37. **void** fun(**int** A[][3], **int** B[][3], **int** result[][3], **int** n)
38. {
39. **int** i, j;
40. **for** (i = 0; i < n; i++)
41. {
42. **for** (j = 0; j < 3; j++)
43. {
44. result[i][j] = A[i][0] \* B[0][j] + A[i][1] \* B[1][j] + A[i][2] \* B[2][j];
45. }
46. }
47. }

## 测试集

|  |  |
| --- | --- |
| A | B |
| 1 1 1  1 1 1  1 1 1 | 1 1 1  1 1 1  1 1 1 |
| 1 1 1  2 2 2  3 3 3 | 1 2 3  1 2 3  1 2 3 |
| 0 0 0  0 0 0  0 0 0 | 0 0 0  0 0 0  0 0 0 |

## 实验过程

（1）在Borland C++ 3.1中输入程序

（2）保存程序，编译运行（CTRL\_F9），调试程序（F4,ALT\_F4,F8,F7等）。

（3）调出用户界面，查看实验结果（ALT\_F5）

## 实验结果

|  |  |
| --- | --- |
| 第一组 | 3 3 3  3 3 3  3 3 3 |
| 第二组 | 3 6 9  6 12 18  9 18 27 |
| 第三组 | 0 0 0  0 0 0  0 0 0 |

（经验证，结果符合预期）

## 实验分析

一维数组的赋值需要一个for循环来实现，二维数组的赋值就更加复杂了。为了让main函数显得更加简洁，我选择额外设计了input函数来对二维数组赋值。同时数组的输出也较为繁琐，放在main函数中同样十分复杂，所以也设计一个函数来解决。

设计出合适的fun函数用来计算矩阵的乘积是本实验的关键，这就需要我们明确线性代数中矩阵乘积的定义，设计合理的循环。

# 实验3 二维数组和数组指针使用形式的互换性，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）明确二维数组和数组指针的形式互换性。

（2）说明指针变量使用数组形式的前提，以及数组使用指针形式的限制。

## 设计思路与代码实现

### 设计思路

写出二维数组和数组指针所有的互换形式，逐一验证他们是否相等。

### 代码实现

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** i = 2, j = 3;
5. **int** a[10][10];
6. **int**(\*pa)[10];
7. pa = a;
8. a[2][3] = 1;
9. **if** (a == pa)
10. {
11. printf("a==pa");
12. }
13. **else**
14. {
15. printf("a!=pa");
16. }
17. **if** (\*a == \*pa && \*pa == a[0] && a[0] == pa[0] && pa[0] == \*(a + 0) && \*(a + 0) == \*(pa + 0) && \*(pa + 0) == \*pa++)
18. {
19. printf("yes, they all equal\n");
20. }
21. pa = a;
22. **if** (\*(a + i) == \*(pa + i) && \*(pa + i) == a[i] && a[i] == pa[i])
23. {
24. printf("yes, they all equal\n");
25. }
26. **if** (\*(a + i) + j == \*(pa + i) + j && \*(pa + i) + j == &a[i][j] && (&a[i][j]) == &pa[i][j] && (&pa[i][j]) == a[i] + j && a[i] + j == pa[i] + j)
27. {
28. printf("yes, they all equal\n");
29. }
30. printf("\*(\*(a+i)+j), \*(\*(pa+i)+j), a[i][j], pa[i][j], \*(a[i]+j), \*(pa[i]+j), \*(a+i)[j], \*(pa+i)[j]:\n%d %d %d %d %d %d %d %d", \*(\*(a + i) + j), \*(\*(pa + i) + j), a[i][j], pa[i][j], \*(a[i] + j), \*(pa[i] + j), (\*(a + i))[j], (\*(pa + i))[j]);
31. }

## 测试集

无

## 实验过程

（1）在Borland C++ 3.1中输入程序

（2）保存程序，编译运行（CTRL\_F9），调试程序（F4,ALT\_F4,F8,F7等）。

（3）调出用户界面，查看实验结果（ALT\_F5）

## 实验结果

a==payes, they all equal

yes, they all equal

yes, they all equal

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

1 1 1 1 1 1 1 1

## 实验分析

从代码的运行结果来看，我们可以总结出二维数组和数组指针的所有互换形式：

|  |  |
| --- | --- |
| 二维数组的地址 | a, pa |
| 二维数组的第一行的一维数组的地址 | \*a, \*pa, a[0], pa[0], \*(a+0), \*(pa+0), \*pa++（发生自增操作，要重新指向a） |
| 二维数组第i行的一维数组的地址 | \*(a+i), \*(pa+i), a[i], pa[i] |
| 二维数组第i行一维数组的第j个元素的地址 | \*(a+i)+j, \*(pa+i)+j, &a[i][j], &pa[i][j], a[i]+j, pa[i]+j |
| 二维数组第i行第j个元素的数值 | \*(\*(a+i)+j), \*(\*(pa+i)+j), a[i][j], pa[i][j], \*(a[i]+j), \*(pa[i]+j), (\*(a+i))[j], (\*(pa+i))[j] |

依靠数组和指针的互换性，我们在编写程序时就更加灵活多变了。

指针变量使用数组形式的前提。指针变量使用数组形式的前提就是指针需要指向数组的首地址。如\*pa++，在运行后pa指向发生了变化，这个时候，我们就需要手动把pa重新指向a，以确保指针变量能够继续使用数组形式。

数组使用指针形式的限制。不论是一维数组还是二维数组，他们的数组名都是一个地址常量，所以不能进行自增操作，也不能进行其他赋值的操作。

# 实验4 二维数组在函数间函数传递，需要几个参数？用实例进行说明

## 实验目的

（1）联系二维数组在函数间的传递方式，明确传递需要的参数个数。

## 设计思路与代码实现

### 设计思路

我参考了课本例5.8的程序，并在原有代码的基础上进行了一定程度的合理修改 (周纯杰, 2016)。

### 代码实现

1. #include<stdio.h>
2. **int** findmax(**int** a[][3], **int** n);
3. **int** main()
4. {
5. **int** a[3][3];
6. **int** i, j, temp;
7. printf("input the numbers:\n");
8. **for** (i = 0; i < 3; i++)
9. {
10. **for** (j = 0; j < 3; j++)
11. {
12. scanf\_s("%d", &a[i][j]);
13. }
14. }
15. temp = findmax(a, 3);
16. printf("the max = %d", temp);
17. }
18. **int** findmax(**int** a[][3], **int** n)
19. {
20. **int** max, i, j;
21. max = a[0][0];
22. **for** (i = 1; i < n; i++)
23. {
24. **for** (j = 0; j < 3; j++)
25. {
26. **if** (a[i][j] > max)
27. {
28. max = a[i][j];
29. }
30. }
31. }
32. **return** max;
33. }

## 测试集

|  |  |
| --- | --- |
| 第一组 | 0 0 0  0 0 0  0 0 0 |
| 第二组 | 1 2 3  4 5 6  9 8 7 |
| 第三组 | -2 -8 10  11 -9 21  10 21 25 |

## 实验过程

（1）在Borland C++ 3.1中输入程序

（2）保存程序，编译运行（CTRL\_F9）

（3）调出用户界面，查看实验结果（ALT\_F5）

（4）总结，确定传递二维数组到函数中所需的参数个数

## 实验结果

the max = 0

the max = 9

the max = 25

## 实验分析

依据代码所示，我们设计的函数原型int findmax(int a[][3], int n); 即说明二维数组在函数间传递需要2个参数。第一个参数是二维数组的首地址，同时也包含了这个二维数组中一维数组的大小（列数），即[3]。第二个参数是二维数组的行数，即int n。

# 实验5 上机验证6.12, 6.13, 6.14

## 实验目的

（1）验证6.12，6.13，6.14 (周纯杰, 2016)

## 题目与对应代码

### 例6.12

有一个班，3个学生，各学4门功课，计算总平均成绩和第二个学生的平均成绩。

### 例6.12代码

1. #include<stdio.h>
2. **float** aver(**float**\*, **int** n);
3. **int** main()
4. {
5. **float** score[3][4] = { {63,65,75,61},{83,87,90,85}shuru,{90,95,100,93} };
6. printf("total average score=%f\n", aver(\*score, 12));
7. printf("second student average score=%f", aver(score[1], 4));
8. }
9. **float** aver(**float**\* pdata, **int** n)
10. {
11. **int** i;
12. **float** average = 0;
13. **for** (i = 0; i < n; i++)
14. {
15. average += pdata[i];
16. }
17. average = average / n;
18. **return** average;
19. }

### 例6.13

有一个4×3的矩阵，求所有元素的最大值

### 例6.13代码

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

### 例6.14

一个班有3个学生，他们各学4门功课，编程实现分别显示每个学生有几门课程是优秀的（90分以上为优秀）以及每个学生的成绩

### 例6.14代码

1. #include<stdio.h>
2. **void** search(**int**(\*p)[4], **int** n);
3. **int** main()
4. {
5. **int** score[3][4] = { {93,96,44,61},{83,87,90,45},{58,95,26,59} };
6. search(score, 3);
7. }
8. **void** search(**int**(\*p)[4], **int** n)
9. {
10. **int** i, j, k, flag\_f;
11. **for** (i = 0; i < n; i++)
12. {
13. k = 0;
14. flag\_f = 0;
15. **do**
16. {
17. **for** (j = k; j < 4; j++)
18. {
19. **if** (p[i][j] > 90)
20. {
21. flag\_f++;
22. **break**;
23. }
24. }
25. k = j + 1;
26. } **while** (j < 4);
27. printf("No.%d has %d grade excellently, his score are:\n", i + 1, flag\_f);
28. **for** (j = 0; j < 4; j++)
29. {
30. printf("%-4d", p[i][j]);
31. }
32. printf("\n");
33. }
34. }

## 实验过程

（1）依次输入程序，编译运行（CTRL\_F9）

（2）打开用户界面观察结果（ALT\_F5）

（3）进行逐步调试，理解程序每一步的过程（F4，ALT\_F4，F8，F9）

## 实验结果

例子6.12

total average score=82.250000

second student average score=86.250000

例6.13

Max value is 8

例6.14

No.1 has 2 grade excellently, his score are:

93 96 44 61

No.2 has 0 grade excellently, his score are:

83 87 90 45

No.3 has 1 grade excellently, his score are:

58 95 26 59

## 实验分析

# 实验6 第6章习题，“三、编程题”的第4题（具体功能用函数实现，再用main()函数测试）

## 题目

将一个5×5的矩阵中最大的元素放在中心，4个角分别放4个最小的元素（顺序为从左到右，从上到下顺序依次从小到大存放），编写一个函数，并用main函数调用来实现

## 设计思路与代码实现

### 设计思路

我们构造的change函数需要干两件事情：将最大值移动到矩阵中央；将最小的4个值移动到矩阵四角。

第一步，设计函数原型。由于需要对二维数组进行操作，所以函数传递两个参数——二维数组首地址，数组的列数。

第二步，设计函数。移动最大值的操作较为简单。难度较大的是如何选取最小的四个值。我采取的方法是：首先提前将四角的原有元素赋入orgin[4]数组中。然后，每找到一个最小值，用指针数组p[4]指向最小值的地址，将最小值录入min[4]数组中，再将最小值地址所在处的数改为max。如此循环，找到4个最小值。

第三步，设计实验函数（main函数）

（下图为change函数流程图）

循环，比较pa[2][2]pa[i][j]

两者不交换

两者交换

pa[2][2]大

pa[i][j]大

继续循环，直至比较所有元素

找最大值

四角元素逐一赋给orgin数组

利用循环找最小值

用指针数组记录最小值的地址

用min数组记录最小值的数字

找到最小值

将该位置处的值改为最大值

四次

找四个最小值

### 代码实现

1. #include<stdio.h>
2. **void** change(**int**(\*)[5], **int**);
3. **int** main()
4. {
5. **int** a[5][5];
6. **int** i, j;
7. **for** (i = 0; i < 5; i++)
8. {
9. **for** (j = 0; j < 5; j++)
10. {
11. scanf\_s("%d", &a[i][j]);
12. }
13. }
14. change(a, 5);
15. **for** (i = 0; i < 5; i++)
16. {
17. **for** (j = 0; j < 5; j++)
18. {
19. printf("%d\t", a[i][j]);
20. }
21. printf("\n");
22. }
23. }
24. **void** change(**int**(\*pa)[5], **int** n)
25. {
26. **int** t;
27. **int** min[4];
28. **int** orgin[4];
29. **int** i, j, k;
30. **int**\* p[4];
31. **for** (i = 0; i < n; i++)
32. {
33. **for** (j = 0; j < 5; j++)
34. {
35. **if** (pa[2][2] < pa[i][j])
36. {
37. t = pa[2][2];
38. pa[2][2] = pa[i][j];
39. pa[i][j] = t;
40. }
41. }
42. }
43. **for** (i = 0, k = 0; i < n; i += 4)
44. {
45. **for** (j = 0; j < 5; j += 4)
46. {
47. orgin[k++] = pa[i][j];
48. }
49. }
50. **for** (k = 0; k < 4; k++)
51. {
52. min[k] = pa[0][0];
53. p[k] = &pa[0][0];
54. **for** (i = 0; i < n; i++)
55. {
56. **for** (j = 0; j < 5; j++)
57. {
58. **if** (min[k] > pa[i][j])
59. {
60. min[k] = pa[i][j];
61. p[k] = &pa[i][j];
62. }
63. }
64. }
65. \*p[k] = pa[2][2];
66. }
67. **for** (i = 0; i < 3; i++)
68. {
69. **for** (j = i + 1; j < 4; j++)
70. {
71. **if** (min[i] > min[j])
72. {
73. t = min[i];
74. min[i] = min[j];
75. min[j] = t;
76. }
77. }
78. }
79. **for** (i = 0; i < 4; i++)
80. {
81. \*p[i] = min[i];
82. }
83. **for** (i = 0, k = 0; i < n; i += 4)
84. {
85. **for** (j = 0; j < 5; j += 4)
86. {
87. t = pa[i][j];
88. pa[i][j] = min[k];
89. \*p[k] = t;
90. k++;
91. }
92. }
93. **for** (i = 0, k = 0; i < n; i += 4)
94. {
95. **for** (j = 0; j < 5; j += 4, k++)
96. {
97. **if** (pa[i][j] != min[k])
98. {
99. pa[i][j] = min[k];
100. }
101. }
102. }
103. }

## 测试集

|  |  |
| --- | --- |
| 第一组 | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 |
| 第二组 | 9 10 11 0 12 3 17 30 21 34 20 30 36 44 45 65 21 2 1 28 32 23 43 34 33 |
| 第三组 | 90 80 70 60 -4 -10 0 0 0 20 32 43 0 21 47 29 222 34 57 87 63 39 56 43 23 |
| 第四组 | 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 |

## 实验过程

（1）设计函数，输入代码，保存

（2）编译运行（CTRL\_F9），查看结果（ALT\_F5）

（3）判断结果是否符合预期

## 实验结果

第一组：

1 5 20 24 2

6 7 8 9 10

11 12 25 13 14

15 16 17 18 19

3 21 22 23 4

第二组：

0 10 11 9 1

33 17 30 21 34

20 30 65 36 44

45 21 32 12 28

2 23 43 34 3

第三组：

-10 80 70 60 -4

0 23 0 0 20

32 43 222 21 47

29 90 34 57 87

0 39 56 43 0

第四组：

26 49 48 47 27

45 44 43 42 41

40 39 50 37 36

35 34 33 32 31

28 38 30 46 29

实验结果符合预期。

# 实验7 第6章习题，“三、编程题”的8题

## 题目

有一个班的4个学生，有5门课程，分别编写3个函数实现下面的三个要求：

a）求第一门课的平均分；

b）找出有两门以上成绩不及格的学生，输出他们的学号和全部课程成绩及平均分；

c）找出平均分在90分以上或全部课程成绩在85分以上的学生。

## 设计思路与代码实现

### 设计思路

四个学生，五门成绩，所以推知需要一个a[4][5]数组来储存数据。

三个函数的设计都比较简单，函数原型可以分别设计为

void average(float (\*)[5], int n);

void fail(float (\*)[5], int n);

void great(float(\*)[5], int n);

相较而言，第二个函数稍显复杂，在这里提供他的流程图。

定义int型变量sign

对第i位学生每门成绩判断

（初始i为0）

sign加一

sign不变

循环直至经过该学生每科

判断sign

是否大于2

将学生学号，平均分分别导入对应数组

不及格

及格

无操作

循环为下一位学生

### 代码实现

1. #include<stdio.h>
2. **void** average(**int**(\*)[5], **int**);
3. **void** fail(**int**(\*)[5], **int**);
4. **void** great(**int**(\*)[5], **int**);
5. **int** main()
6. {
7. **int** a[4][5];
8. **int** i, j;
9. **for** (i = 0; i < 4; i++)
10. {
11. **for** (j = 0; j < 5; j++)
12. {
13. scanf\_s("%d", &a[i][j]);
14. }
15. }
16. average(a, 4);
17. fail(a, 4);
18. great(a, 4);
19. }
20. **void** average(**int**(\*pa)[5], **int** n)
21. {
22. **int** i;
23. **int** sum = 0;
24. **for** (i = 0; i < n; i++)
25. {
26. sum += pa[i][0];
27. }
28. sum /= 4;
29. printf("average = %d\n", sum);
30. }
31. **void** fail(**int**(\*pa)[5], **int** n)
32. {
33. **int** i, j, k;
34. **int** sign = 0;
35. **int** sum = 0;
36. **for** (i = 0; i < n; i++)
37. {
38. sum = 0;
39. sign = 0;
40. **for** (j = 0; j < 5; j++)
41. {
42. **if** (pa[i][j] < 60)
43. {
44. sign++;
45. }
46. }
47. **if** (sign > 2)
48. {
49. **for** (k = 0; k < 5; k++)
50. {
51. sum += pa[i][k];
52. }
53. sum /= 5;
54. printf("No.%d is failed\n%d %d %d %d %d, average: %d\n", i + 1, pa[i][0], pa[i][1], pa[i][2], pa[i][3], pa[i][4], sum);
55. }
56. }
57. }
58. **void** great(**int**(\*pa)[5], **int** n)
59. {
60. **int** i, j;
61. **int** sum[4] = {0};
62. **int** sign1 = 0, sign2 = 0;
63. **for** (i = 0; i < n; i++)
64. {
65. **for** (j = 0; j < 5; j++)
66. {
67. sum[i] += pa[i][j];
68. }
69. sum[i] /= 5;
70. }
71. **for** (i = 0; i < n; i++)
72. {
73. sign1 = 0;
74. sign2 = 0;
75. **if** (sum[i] > 90)
76. {
77. sign1++;
78. }
79. **for** (j = 0; j < 5; j++)
80. {
81. **if** (pa[i][j] > 85)
82. {
83. sign2++;
84. }
85. }
86. **if** (sign1 == 1 || sign2 == 5)
87. {
88. printf("No.%d is very great!\n", i + 1);
89. }
90. }
91. }

## 测试集

|  |  |
| --- | --- |
| 第一组 | 77 86 90 98 94  94 81 86 94 100  60 59 80 98 77  90 91 92 93 94 |
| 第二组 | 96 84 98 91 90  88 81 87 85 85  0 85 85 85 100  86 86 86 86 86 |
| 第三组 | 90 90 90 90 90  60 59 32 11 0  88 79 90 91 97  95 92 91 90 98 |

## 实验过程

（1）设计函数，输入代码，保存

（2）编译运行（CTRL\_F9），查看结果（ALT\_F5）

（3）判断结果是否符合预期

## 实验结果

第一组：

average = 80

No.2 is very great!

No.4 is very great!

第二组：

average = 67

No.1 is very great!

No.4 is very great!

第三组：

average = 83

No.2 is failed

60 59 32 11 0, average: 32

No.1 is very great!

No.4 is very great!

实验结果与预期相符。

# 实验8 第5章习题，第11题（具体功能用函数实现，再用main()函数测试）

## 题目

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。

## 设计思路与代码实现

### 设计思路

二维数组分行求和并不困难。输出sum[N]时，我们既可以在主函数中输出，也可以在自定义的函数中输出。在这里我们依据题意，选择将sum[N]在自定义函数中直接输出。

下面是自定义函数的流程图

对第i行求和

令sum[i]等于其和

循环输出sum数组

### 代码实现（以N=3为例）

1. #include<stdio.h>
2. **void** function(**int**(\*pa)[3], **int** n);
3. **int** main()
4. {
5. **int** a[3][3];
6. **int** i, j;
7. **for** (i = 0; i < 3; i++)
8. {
9. **for** (j = 0; j < 3; j++)
10. {
11. scanf\_s("%d", &a[i][j]);
12. }
13. }
14. function(a, 3);
15. }
16. **void** function(**int**(\*pa)[3], **int** n)
17. {
18. **int** i, j;
19. **int** sum[3] = { 0 };
20. **for** (i = 0; i < n; i++)
21. {
22. **for** (j = 0; j < 3; j++)
23. {
24. sum[i] += pa[i][j];
25. }
26. }
27. **for** (i = 0; i < n; i++)
28. {
29. printf("%d\t", sum[i]);
30. }
31. }

## 测试集

|  |  |
| --- | --- |
| 第一组 | 10 19 20  30 21 12  87 99 0 |
| 第二组 | 9 10 18  -9 9 0  11 19 29 |
| 第三组 | 78 87 1  4 3 1  80 20 1 |

## 实验过程

（1）设计函数，输入代码，保存

（2）编译运行（CTRL\_F9），查看结果（ALT\_F5）

（3）判断结果是否符合预期

## 实验结果

第一组：

49 63 186

第二组：

37 0 59

第三组：

166 8 101

实验结果与预期相符。

附录

## 开发环境

本次实验中使用的环境配置如下：

（1）操作系统版本：Arch Linux x86\_64

（2）编译器及其版本：GCC version 8.1.1

（3）自动编译工具：CMake version 3.11.4

（4）编程环境：Borland C++ 3.1

# 引用

**周纯杰何顶新,周凯波,彭刚,张惕远. 2016.** 程序设计教程. 北京 : 机械工业出版社, 2016.