

# 程序分组设计训练 实验 1

## 实验报告

学期：2022-2023 第二学期

学院：计算机与信息技术学院

姓名：张颢沅

学号：22281052

班级：计算机 2202 班

编制日期：2023 年 3 月 1 日

# 目录

## 实验一 矩阵转置

|       |                   |    |
|-------|-------------------|----|
| 1     | 矩阵转置 .....        | 1  |
| 1.1   | 初始数据观察 (1) .....  | 1  |
| 1.1.1 | 表格填空 .....        | 1  |
| 1.1.2 | 问题回答 .....        | 1  |
| 1.2   | 初始数据观察 (2) .....  | 3  |
| 1.2.1 | 表格填空 .....        | 3  |
| 1.2.2 | 问题回答 .....        | 3  |
| 1.3   | 初始数据观察 (3) .....  | 5  |
| 1.3.1 | 表格填空 .....        | 5  |
| 1.3.2 | 问题回答 .....        | 5  |
| 1.4   | 初始数据观察 (4) .....  | 8  |
| 1.4.1 | 表格填空 .....        | 8  |
| 1.4.2 | 问题回答 .....        | 8  |
| 1.5   | 转置过程记录与程序修改 ..... | 10 |
| 1.5.1 | 过程记录 .....        | 10 |
| 1.5.2 | 修改程序代码 .....      | 11 |

## 实验二 矩阵转置

|     |                   |    |
|-----|-------------------|----|
| 2   | 矩阵鞍点 .....        | 12 |
| 2.1 | 原数据输出结果: .....    | 12 |
| 2.2 | 数据观察与相关问题回答 ..... | 12 |

## 实验总结

|     |                                   |    |
|-----|-----------------------------------|----|
| 3   | 后续相关问题 .....                      | 16 |
| 3.1 | 设置断点并监控变量 .....                   | 16 |
| 3.2 | 条件断点 .....                        | 17 |
| 3.3 | step in 、step over、step out ..... | 17 |
| 4   | 实验的收获与心得 .....                    | 19 |
|     | 参考文献 .....                        | 19 |

# 实验一 矩阵转置

## 1 矩阵转置

### 1.1 初始数据观察（1）

#### 1.1.1 表格填空

表 1 数据初始值表 （在第一个 for 循环执行前，各个变量值监视结果）

| 变量/常量/表达式名 | IDE 软件名称               | 断点监视结果                                                                                                                  | printf 输出语句              | printf 输出结果 |
|------------|------------------------|-------------------------------------------------------------------------------------------------------------------------|--------------------------|-------------|
| a          | VS 2022 <sup>[1]</sup> | 0x0000004e97cff790<br>{0x0000004e97cff790 {1, 2, 3, 4}, 0x0000004e97cff7a0 {5, 6, 7, 8}, 0x0000004e97cff7b0 {...}, ...} | printf("%x ", a);        | 97cff790    |
| *a         | VS 2022                | 0x0000004e97cff790 {1, 2, 3, 4}                                                                                         | printf("%x ", *a);       | 97cff790    |
| a[0]       | VS 2022                | 0x0000004e97cff790 {1, 2, 3, 4}                                                                                         | printf("%x ", a[0]);     | 97cff790    |
| &a[0][0]   | VS 2022                | 0x0000004e97cff790 {1}                                                                                                  | printf("%x ", &a[0][0]); | 97cff790    |
| p          | VS 2022                | 0x0000004e97cff790 {1, 2, 3, 4}                                                                                         | printf("%x ", p);        | 97cff790    |
| *p         | VS 2022                | 0x0000004e97cff790 {1, 2, 3, 4}                                                                                         | printf("%x ", *p);       | 97cff790    |
| p[0]       | VS 2022                | 0x0000004e97cff790 {1, 2, 3, 4}                                                                                         | printf("%x ", p[0]);     | 97cff790    |

[1]: VS 2022 指的是 Visual Studio 2022。后续表格类同。

#### 1.1.2 问题回答

a) printf 输出的 a、p、\*a、\*p、a[0]、p[0]、&a[0][0]的值是否一样，为什么这些变量/常量或表达式的 printf 输出值均一样？

答：是一样的。以上各变量/常量/表达式都指向的是程序中二维数组第一个元素的地址。

b) 上述变量/常量或表达式虽然输出值一样，但实质上它们之间还是有区别的，请尝试论述一下上述表格中变量或表达式的区别。

答：“a”是二维数组名，包括了已经录入的数据，是一个地址常量，指向的是数组第一个元素的地址。      二维数组的首地址

“\*a”代表这个二维数组所在的地址，即就是第一行第一列元素的地址

“a[0]”代表数组第一行元素开始位置的指针，即就是第一行第一列元素的地址，类同“\*a”

“&a[0][0]”是数组中第一行第一列元素的地址

“p”是指向 a 的指针，指向了数组中第一行的地址

“\*p”是指向长度为 4 的 int 数组的指针，代表的是数组 a 所在地址的数值。

“p[0]”是指向二维数组第一行第一个元素的指针。

**数组的首元素地址：表示数组的首个元素的地址。**

**数组的首地址：表示整个数组的地址**

c) 通过 IDE 软件监控到的变量/常量或表达式的值与 printf 输出的值是否一致，如果不一致，尝试论述一下不一致在哪里，是什么原因导致的。

答：不一致。因为 printf 与“%x”结合只会输出变量/常量/表达式所在地址表达的数值。而监控到的变量是能够监控到变量的具体内容的。



## 1.2 初始数据观察（2）

### 1.2.1 表格填空



表 2 数据初始值表 （在第一个 for 循环执行前，各个变量值监视结果）

| 变量/常量/表达式<br>名 | IDE 软件名<br>称 | 断点监视结果                             | printf 输出语句                 | printf 输出结果 |
|----------------|--------------|------------------------------------|-----------------------------|-------------|
| &a[1][0]       | VS 2022      | 0x000000b9ac7efbe0 {5}             | printf("%x ",<br>&a[1][0]); | ac7efbe0    |
| a[1]           | VS 2022      | 0x000000b9ac7efbe0 {5, 6, 7,<br>8} | printf("%x ", a[1]);        | ac7efbe0    |
| a+1            | VS 2022      | 0x000000b9ac7efbe0 {5, 6, 7,<br>8} | printf("%x ", a+1);         | ac7efbe0    |
| *(a+1)         | VS 2022      | 0x000000b9ac7efbe0 {5, 6, 7,<br>8} | printf("%x ",<br>*(a+1));   | ac7efbe0    |
| &a[0][1]       | VS 2022      | 0x000000b9ac7efbd4 {2}             | printf("%x ",<br>&a[0][1]); | ac7efbd4    |
| a[0]+1         | VS 2022      | 0x000000b9ac7efbd4 {2}             | printf("%x ",<br>a[0]+1);   | ac7efbd4    |
| *a+1           | VS 2022      | 0x000000b9ac7efbd4 {2}             | printf("%x ", *a+1);        | ac7efbd4    |

### 1.2.2 问题回答

a) 比较列表中表达式的 printf 输出值与断点监视结果（使用 VS 进行开发的同学可以点击监视变量前面的   a+1 加号展开监视变量进行比较），尝试阐述 \*a+1 与 \*(a+1) 的区别？

答：\*(a+1) 表示的是二维数组的第二行，而 \*a+1 表示的是数第一行的第二个数。

b) a+1 与 \*(a+1) 的区别是什么？这两个表达式的断点监视结果有什么不同（使用 VS 进行开发的同学可以点击监视变量前面的   a+1 加号展开监视变量进行比较）？这两个表达式哪个与 a[1] 等价。

答：①  $a+1$  是二维数组第二行的地址，是首地址。.....

② 监视的类型不同， $a+1$  是 “ $\text{int}[4]*$ ” 类型，而  $*(a+1)$  是 “ $\text{int}$ ” 类型。

③  $*(a+1)$  和  $a[1]$  等价

c) 本程序中，上述表达式中字母  $a$  用  $p$  替换，表达式是否仍然合法？替换后表达式值与原表达式值是否一样？

答：仍然合法。替换后表达式值与原表达式值一样。

```
//进行数组转置操作
printf("%x\n", &p[1][0]);
printf("%x\n", p[1]);
printf("%x\n", p + 1);
printf("%x\n", *(p + 1));
printf("%x\n", &p[0][1]);
printf("%x\n", p[0] + 1);
printf("%x\n", *p + 1);

for (i = 0; i < 4; i++) //遍历每一行
```

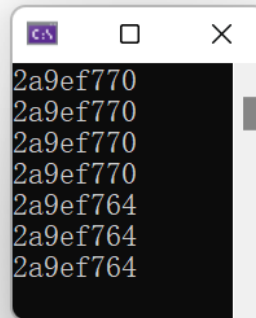


图 1-1  $a$  用  $p$  替换后运行结果

## 1.3 初始数据观察（3）

### 1.3.1 表格填空

表 3 数据初始值表 （在第一个 for 循环执行前，各个变量值监视结果）

| 序号 | 变量/常量/表达式名    | 表达式值       |
|----|---------------|------------|
| 1  | a[2][0]       | 9          |
| 2  | *a[2]         | 9          |
| 3  | ** (a+2)      | 9          |
| 4  | (* (a+2)) [0] | 9          |
| 5  | *(a+2) [0]    | 9          |
| 6  | a[2][3]       | 12         |
| 7  | *(*(a+2)+3)   | 12         |
| 8  | (* (a+2)) [3] | 12         |
| 9  | *(a[2]+3)     | 12         |
| 10 | *a[2]+3       | 12         |
| 11 | *(a+2) [3]    | -858993460 |
| 12 | *(a+11)       | 12         |
| 13 | (*a) [11]     | 12         |
| 14 | *a[11]        | -858993460 |

### 1.3.2 问题回答

a) 序号为 1~5 的表达式值是否相同？序号为 6~10 的表达式值是否相同？

答：序号为 1~5 的表达式值相同。序号为 6~10 的表达式值相同。

b) 序号为 9 和 10 的表达式值相同，这两个表达式是等价的么？是在任何情况下这两个表达式的值都相同么？

答：不是等价的。并不是在任何情况下这两个表达式的值都相同。

序号 9 是先取到了 a 第三行的地址后加 3，再用\*取他的地址得到结果。序号



10 先取到了 a 第三行的地址后用\*取值后，再加 3 得最终结果。

c) 比较序号为 4、5 的表达式与序号为 8、11 的表达式，两组表达式在形式上是类似的，但是为什么 4、5 表达式的值相同，而 8、11 表达式的值不同？通过比较回答表达式  $a[2][0]$  应该与序号为 4 的表达式等价还是与序号为 5 的表达式等价？

答：① “ $(*(a+x))[y]$ ” 相当于 a 为二级指针，之后向后移动 x 行后转换为一级指针，[y] 相当于在一级指针表示 “ $a[0+x][0]$ ” 的基础上向后移动 y 位，取值得到最终数字。

“ $*(a+x)[y]$ ” 相当于 a 为二级指针，之后向后移动 x 行后，[y] 相当于在二级指针表示在 x 的基础上向后移动 y 行，之后转换为一级指针。

当  $x=2, y=0$  时，当移动 2 行后，没有一级指针变化，因此最终指向  $a[2][0]$ 。

当  $x=2, y=3$  时，当移动 2 行后，序号 11  $*(a+2)[3]$  在移动两行的基础上移动了 3 行，导致指针越界，因此最终显示了随机数，与序号 8 的结果不同。

② 由①分析可得， $a[2][0]$  应该与序号为 4 的表达式等价。

d) 比较序号为 12、13 的表达式值，与序号为 6 的表达式值是否相同，尝试回答序号为 12、13 的表达式最终取到的 int 值是数组元素  $a[2][3]$  对应内存空间中的值么？为什么可以用这样的表达式取值？

答：序号为 12、13 的表达式值，与序号为 6 的表达式值相同

序号 12 “ $*(a+11)$ ” 是将 a 从二级指针降为一级指针，之后在此基础上向后移动 11 位，到了  $a[2][3]$  的位置。

序号 13 “ $(a)[11]$ ” 相当于 a 从二级指针降为一级指针后，[11] 相当于在

一级指针表示“a[0][0]”的基础上向后移动 11 位，到了 a[2][3] 的位置。

e) 在本程序中考虑将序号为 12、13 的表达式由 a 变成 p，既把两个表达式变为  $*(p+11)$  和  $(*p)[11]$ ，重新定义一个一维数组：`int c[4]={8,8,8,8}`；请考虑当  $p=a$  时和  $p=\&c$  两种情况下，这两个表达式是否都能够正常编译，是否都会引起逻辑错误？

答：当  $p=a$  时和  $p=\&c$  时，这两个表达式都能够正常编译，不会引起逻辑错误。因为 p 是指向[4]的数组。

f) 比较序号为 14 的表达式和序号为 11 的表达式，尝试找出“()”、“[]”和“\*”运算符的优先级关系。

答：优先级由“()”“[]”“\*”依次递减。

**int (\*p)[4] 和 int \*p[4]的区别!!!**

**\*降级 &升级**  
**(review：一级指针与二级指针)**

## 1.4 初始数据观察（4）

### 1.4.1 表格填空

表 4 数据初始值表 （在第一个 for 循环执行前，各个变量值监视结果）

| 序号 | 变量/常量/表达式名 | 表达式值                                |
|----|------------|-------------------------------------|
| 1  | p          | 0x0000007c060ff590 {1, 2, 3, 4}     |
| 2  | *p         | 0x0000007c060ff590 {1, 2, 3, 4}     |
| 3  | q          | 0x0000007c060ff590 {1}              |
| 4  | *q         | 1                                   |
| 5  | p[0]       | 0x0000007c060ff590 {1, 2, 3, 4}     |
| 6  | q[0]       | 1                                   |
| 7  | p[3]       | 0x0000007c060ff5c0 {13, 14, 15, 16} |
| 8  | *p[3]      | 13                                  |
| 9  | q[3]       | 4                                   |
| 10 | q[11]      | 12                                  |
| 11 | *p[11]     | -858993460                          |

### 1.4.2 问题回答

a) p 和 q 的区别是什么？

答：p 是指向一个规模为[4]数组的指针，而 q 是一个指向数组元素的指针。

b) 如果想访问数组中 a 中第 4 行第 2 列的元素 a[3][1]，使用指针 p 和指针 q 分别可以用哪些表达式访问？如果是想访问 a 中第 i 行第 j 列的元素，可以用哪些表达式访问？

答：

① p:  $*(*(p+3)+1)$  /  $(*p)[13]$  /  $p[3][1]$  /  $*(p+13)$  /  $*p[3]+1$  /  $*(p[3]+1)$

q:  $q[13]$  /  $*(q+13)$  /  $*q+13$

② p:  $*(*(p+i-1)+j-1) / (*p)[n] / p[i-1][j-1] / *(p+n) / *p[i-1]+$   
 $j-1 / *(p[i-1]+j-1)$

q:  $q[n] / *(q+n) / *q+n$

其中:  $n=(i-1)*4+j$

## 1.5 转置过程记录与程序修改

### 1.5.1 过程记录

表 5 转置过程记录表

| 第几次<br>交换 | 交换前<br>矩阵值                                                                                              | i | j | p              | q | p+j             | *(p+j)          | *(p+j)+i | *(*(p+j)+i) | q+j  | *(q+j) | 交换后<br>矩阵值                                                                                              |
|-----------|---------------------------------------------------------------------------------------------------------|---|---|----------------|---|-----------------|-----------------|----------|-------------|------|--------|---------------------------------------------------------------------------------------------------------|
| 1         | $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$ | 0 | 1 | {1, 5, 3, 4}   | 1 | {2, 6, 7, 8}    | {2, 6, 7, 8}    | {2}      | 2           | {5}  | 5      | $\begin{bmatrix} 1 & 5 & 3 & 4 \\ 2 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$ |
| 2         | $\begin{bmatrix} 1 & 5 & 3 & 4 \\ 2 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$ | 0 | 2 | {1, 5, 3, 4}   | 1 | {3, 10, 11, 12} | {3, 10, 11, 12} | {3}      | 3           | {9}  | 9      | $\begin{bmatrix} 1 & 5 & 9 & 4 \\ 2 & 6 & 7 & 8 \\ 3 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$ |
| 3         | $\begin{bmatrix} 1 & 5 & 9 & 4 \\ 2 & 6 & 7 & 8 \\ 3 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$ | 0 | 3 | {1, 5, 9, 13}  | 1 | {4, 14, 15, 16} | {4, 14, 15, 16} | {4}      | 4           | {13} | 13     | $\begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 7 & 8 \\ 3 & 10 & 11 & 12 \\ 4 & 14 & 15 & 16 \end{bmatrix}$ |
| 4         | $\begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 7 & 8 \\ 3 & 10 & 11 & 12 \\ 4 & 14 & 15 & 16 \end{bmatrix}$ | 1 | 2 | {1, 5, 10, 13} | 1 | {3, 9, 11, 12}  | {3, 9, 11, 12}  | {9}      | 9           | {10} | 10     | $\begin{bmatrix} 1 & 5 & 10 & 13 \\ 2 & 6 & 7 & 8 \\ 3 & 9 & 11 & 12 \\ 4 & 14 & 15 & 16 \end{bmatrix}$ |
| 5         | $\begin{bmatrix} 1 & 5 & 10 & 13 \\ 2 & 6 & 7 & 8 \\ 3 & 9 & 11 & 12 \\ 4 & 14 & 15 & 16 \end{bmatrix}$ | 1 | 3 | {1, 5, 10, 14} | 1 | {4, 13, 15, 16} | {4, 13, 15, 16} | {13}     | 13          | {14} | 14     | $\begin{bmatrix} 1 & 5 & 10 & 14 \\ 2 & 6 & 7 & 8 \\ 3 & 9 & 11 & 12 \\ 4 & 13 & 15 & 16 \end{bmatrix}$ |
| 6         | $\begin{bmatrix} 1 & 5 & 10 & 14 \\ 2 & 6 & 7 & 8 \\ 3 & 9 & 11 & 12 \\ 4 & 13 & 15 & 16 \end{bmatrix}$ | 2 | 3 | {1, 5, 10, 14} | 2 | {4, 13, 8, 16}  | {4, 13, 8, 16}  | {8}      | 8           | {15} | 15     | $\begin{bmatrix} 1 & 5 & 10 & 14 \\ 2 & 6 & 7 & 15 \\ 3 & 9 & 11 & 12 \\ 4 & 13 & 8 & 16 \end{bmatrix}$ |

### 1.5.2 修改程序代码

```
for (i = 0; i < 4; i++)    //遍历每一行
{
    for (j = i + 1; j < 4; j++)    //<遍历每一列>
    {
        //进行元素交换
        temp = (*(p + j) + i);
        (*(p + j) + i) = *(q + j);
        *(q + j) = temp;
        for (k = 0; k < 4; k++)
        {
            for (l = 0; l < 4; l++)
            {
                printf("%d ", a[k][l]);
            }

            printf("\n");
        }
        printf("\n");
    }

    //<指针q每次执行完一行的转置后，转下一行继续进行>
    q = q + 4;
}
```

图 1-2 修改程序代码<sup>①</sup>

①：红色划线部分是程序修改处。

```
1 5 9 13
2 6 10 14
3 7 11 12
4 8 15 16

1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16

1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16

D:\Learning_ComputerLanguages\learning_language_c\_2_ProgramGroupTraining\_1_Experiments\_1_20230301_Experiment_1_1\_1_20230301_Experiment_1_1\x64\Debug\_1_20230301_Experiment_1_1.exe (进程 31368)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .
```

图 1-3 修改代码后的运行结果

之前错误原因是因为语句“ $q = q + i * 4;$ ”运行时。第一次的  $i$  为 0，导致

第一次运行此语句等价于“ $q = q;$ ”， $q$  未转到第二行导致逻辑错误。



| 监视 1               |                                                                      |          |
|--------------------|----------------------------------------------------------------------|----------|
| 搜索(Ctrl+E) 搜索深度: 3 |                                                                      |          |
| 名称                 | 值                                                                    | 类型       |
| ColIndex           | 0x00000015fabdf5b0 {2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... | int[100] |
| 添加要监视的项            |                                                                      |          |

图 2-2-1-2 测试用例 1 监视图 2

测试用例 2:

| 监视 1               |                                                                      |          |
|--------------------|----------------------------------------------------------------------|----------|
| 搜索(Ctrl+E) 搜索深度: 3 |                                                                      |          |
| 名称                 | 值                                                                    | 类型       |
| ColIndex           | 0x00000041fc4ff340 {1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... | int[100] |
| 添加要监视的项            |                                                                      |          |

图 2-2-2-1 测试用例 2 监视图 1

| 监视 1               |                                                                      |          |
|--------------------|----------------------------------------------------------------------|----------|
| 搜索(Ctrl+E) 搜索深度: 3 |                                                                      |          |
| 名称                 | 值                                                                    | 类型       |
| ColIndex           | 0x00000041fc4ff340 {2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... | int[100] |
| 添加要监视的项            |                                                                      |          |

图 2-2-2-2 测试用例 2 监视图 2

测试用例 3:

| 监视 1               |                                                                      |          |
|--------------------|----------------------------------------------------------------------|----------|
| 搜索(Ctrl+E) 搜索深度: 3 |                                                                      |          |
| 名称                 | 值                                                                    | 类型       |
| ColIndex           | 0x000000b8af0ff680 {2, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... | int[100] |
| 添加要监视的项            |                                                                      |          |

图 2-2-3-1 测试用例 3 监视图

| 监视 1               |                                                                      |          |
|--------------------|----------------------------------------------------------------------|----------|
| 搜索(Ctrl+E) 搜索深度: 3 |                                                                      |          |
| 名称                 | 值                                                                    | 类型       |
| ColIndex           | 0x000000b8af0ff680 {3, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... | int[100] |
| 添加要监视的项            |                                                                      |          |

图 2-2-3-2 测试用例 3 监视图 2



测试用例 4:



图 2-2-4-1 测试用例 4 监视图 1



图 2-2-4-2 测试用例 4 监视图 2

ColIndex[0]是以行为单位，矩阵中最大数的个数。

ColIndex[1]是第一行中最大数的列数。

ColIndex[2]是第二行中最大数的列数。

ColIndex[n]是第 n 行中最大数的列数。

(2) 通过程序跟踪，明确结构体数组 ans 记录了哪些信息，并在实验报告中加以说明。

答：ans[0].x 用来记录鞍点的个数。

ans[ans[0].x].x 记录对应第 n 个鞍点的行坐标。

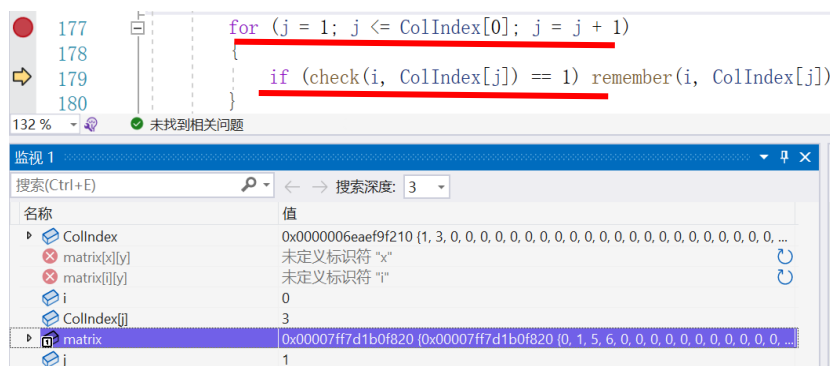
ans[ans[0].x].y 记录对应第 n 个鞍点的列坐标。

(3) 程序中 check 函数的功能是什么，请在实验报告中加以说明。

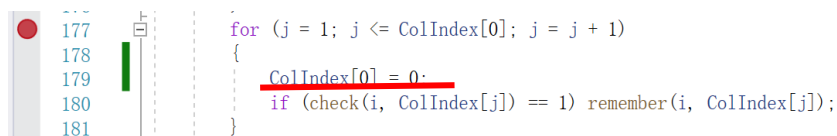
答：check 函数是检查筛选的“准鞍点”是否是每列的最小值。如果是，则记录。

(4) 通过设置断点、单步执行、监控变量找到 CodeForLabB 例程的逻辑错误，并进行修改，将错误原因及如何做的修改在实验报告中加以说明。

这里对于 ColIndex[0]是不需要验证的，它是以行为单位，矩阵中最大数的个数，所以需要将第一位清零，避免影响后续程序。

图 2-2-5-1 程序问题段<sup>①</sup>

①：红色划线部分是程序问题处。

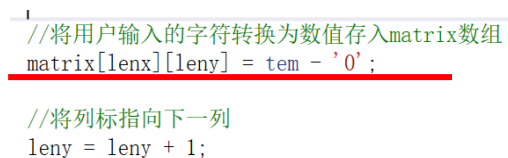
图 2-2-5-2 修改程序后代码<sup>②</sup>

②: 红色划线部分是程序修改处。

(5) 认真阅读 CodeForLabB 例程中的 prepare 函数，根据该函数的实现逻辑，指出 CodeForLabB 例程的局限性，什么情况下程序能正常运行，什么情况下不能？请在实验报告中加以说明。

答：当程序输入的是能被 ASCII 能代替字符程序能正常运行。

其他程序不能正常运行。（小数也无法运行）

图 2-2-6 程序问题段<sup>③</sup>

③: 红色划线部分是程序问题处。

# 实验总结

## 3 后续相关问题

### 3.1 设置断点并监控变量

题目：通过截图和文字论述如何在您所使用的集成开发环境中为程序设置断点并监控变量的值。

答：

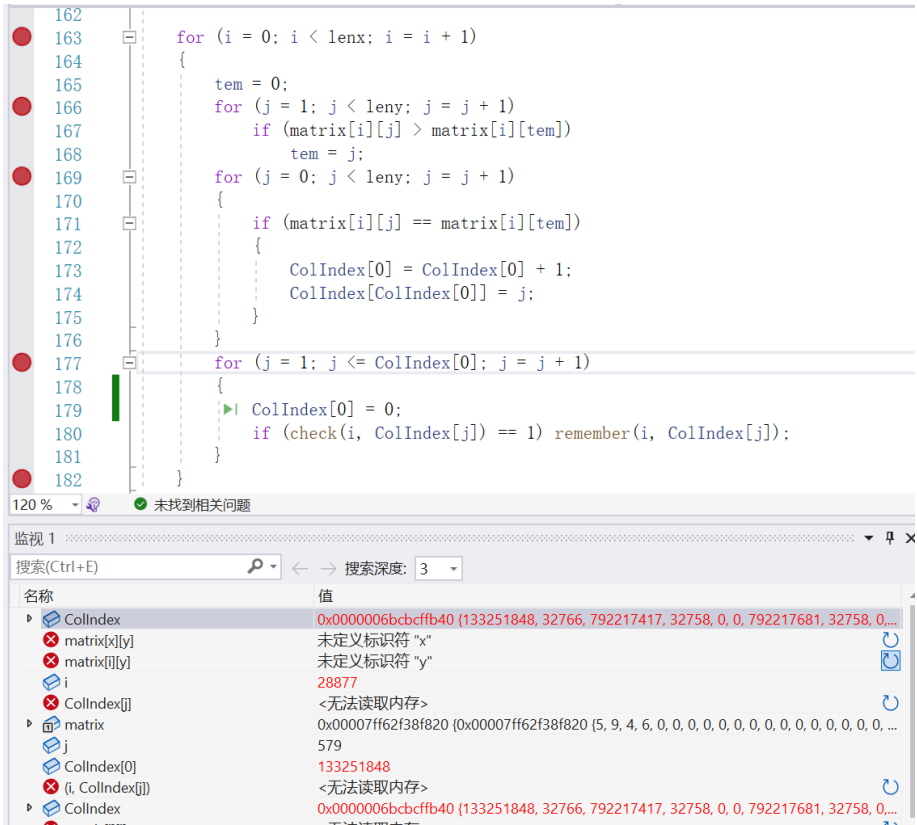


图 3-1 断点设置与监控变量

在程序的下方监视框中，输入要监控的变量。在对应的循环处点击屏幕右边，设置断点。当此程序不必监控时，运用“step out”将不必监视程序段进行跳过。

## 3.2 条件断点

题目：什么是条件断点，通过截图和文字论述在您所使用的集成开发环境中如何设置条件断点。

答：

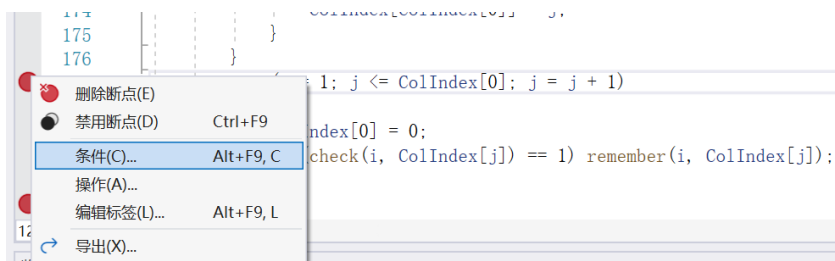


图 3-2-1 条件断点设置 1

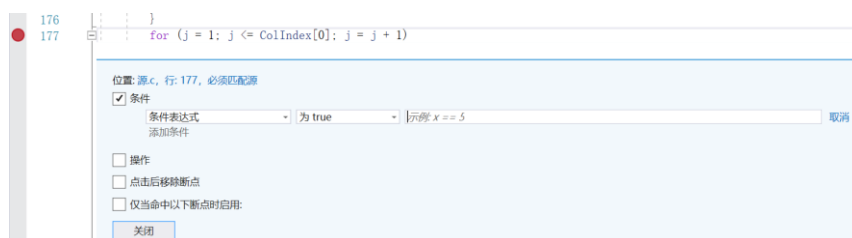


图 3-2-2 条件断点设置 2

(1) 条件断点是当设置某一条件后，当符合此条件时运行断点，否则略过此断点。

(2) 当设置好普通断点后，右键点击断点，选择“条件(C)”，即会弹出图 3-2-2 的对话框，之后根据需求进行设置即可。

## 3.3 step in、step over、step out

题目：通过截图和文字阐述在您所使用的集成开发环境中进行程序单步调试时“step-in”、“step-over”和“step-out”操作的作用；

答：step-in：进入函数里，点击后执行下一个语句。

step-over: 如果当前行是一个函数调用, 则调试器将在函数调用之后的下一条语句停止。调试器不会进入函数体。(不常用)

step-out: 跳出函数。



图 3-3 continue、step in、step over、step out (从左到右)

## 4 实验的收获与心得

(1) 运用 step-out 简化调试步骤。

本次实验通过探索“step-out”的运用，感受到了 debug 过程中的步骤简化。不必像之前一直点“step-in”浪费时间与精力。这次实验让我对于断点的使用更加灵活和熟练。

(2) 本次的实验，让我对于指针和二维数组的关系理解得更加深入。通过复习上课老师讲解的顺时针右旋法则，并且通过程序调试，理解了“()”“[]”“\*”的优先级。提升了我的复杂指针的理解程度。

### 类型解读

- 曾经有一个顺时针右旋法则 [Clockwise/Spiral Rule](#)
- 但有时会错: [The spiral rule about declarations — when is it in error?](#)
- 这个可以参考: [Complicated declarations in C](#)
- 不用难为情，很多人都会搞错。相信我，认真学完这个幻灯片，你就是高手了。如有不懂，一定要问！

图 4-1 老师上课讲解课件<sup>[1]</sup>

(3) 本次的实验让我感受到当发现问题后，通过 debug 理解程序逻辑错误，并去解决问题的整个思维，是十分严密的。网络上并没有相关的资料问题，或者类似的解释，只能通过自己的探索去解决这个问题。这样子提升了我的看代码找问题的能力，对以后的程序编程有很大的作用。

### 参考文献

[1] 杨武杰. C 语言的指针数组. <https://yangwujie.github.io/slides/c/01.html>. 2023-02