

**课 程 实 验 报 告**

**课程名称： C语言程序设计实验**

**专业班级： 软件工程2003班**

**学 号： U202010851**

**姓 名： 侯皓斐**

**指导教师： 唐赫**

**报告日期： 2020.12.5**

**软件工程**

**目 录**

[**5 数组实验 1**](#_Toc404837920)

[5.1 实验目的 1](#_Toc404837921)

[5.2 实验内容 1](#_Toc404837922)

[5.3 实验小结 20](#_Toc404837923)

# 5 数组实验

## 5.1 实验目的

（1）掌握数组的说明、初始化和使用。

（2）掌握一维数组作为函数参数时实参和形参的用法。

（3）掌握字符串处理函数的设计，包括串操作函数及数字串与数之间转换函数实现算法。

（4）掌握基于分治策略的二分查找算法和选择法排序算法的思想，以及相关算法的实现。

## 5.2 实验内容

**5.2.1 源程序改错**

**1．程序改错**

在下面所给的源程序中，函数strcate(t,s)的功能是将字符串s连接到字符串t的尾部；函数strdelc(s,c)的功能是从字符串s中删除所有与给定字符c相同的字符，程序应该能够输出如下结果：

Programming Language

ProgrammingLanguage Language

ProgrmingLnguge

跟踪和分析源程序中存在的问题，排除程序中的各种逻辑错误，使之能够输出正确的结果。

1 /\*实验5-1程序改错与跟踪调试题程序\*/

2 #include<stdio.h>

3 void strcate(char [],char []);

4 void strdelc(char [],char );

5 int main(void)

6 {

7 char a[]="Language", b[]="Programming";

8 printf("%s %s\n", b,a);

9 strcate(b,a); printf("%s %s\n",b,a);

10 strdelc(b, 'a'); printf("%s\n",b);

11 return 0;

12}

13void strcate(char t[],char s[])

14{

14 int i = 0, j = 0;

15 while(t[i++]) ;

16 while((t[i++] = s[j++] )!= '\0');

17}

18void strdelc(char s[], char c)

19{

20 int j,k;

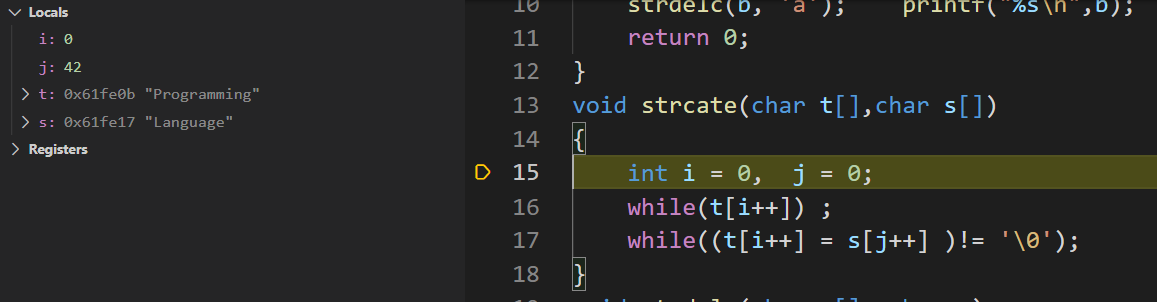
21 for(j=k=0; s[j] != '\0'; j++)

22 if(s[j] != c) s[k++] = s[j];

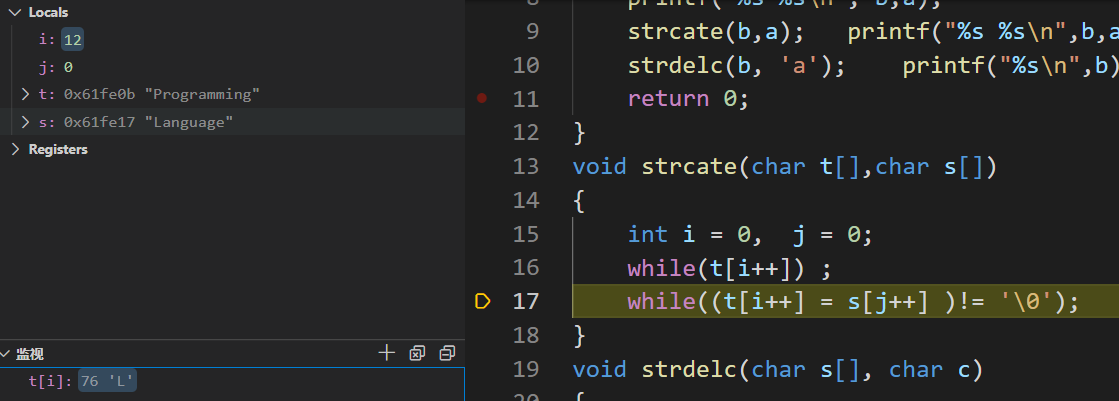
23}

**解答：**

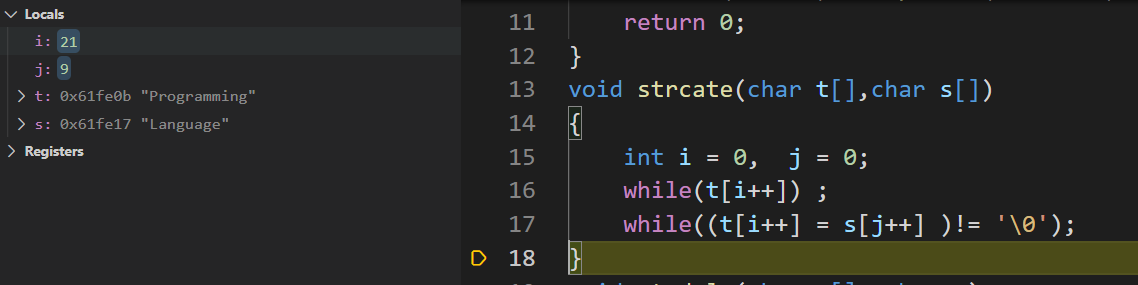
（1）单步执行源程序。进跟踪进入strcate时，观察字符数组t和s中的内容，分析结果是否正确。当单步执行光条刚落在第二个while语句所在行时，i为何值？t[i]为何值？分析该结果是否存在问题。当单步执行光条落在strcate函数块结束标记即右花括号“}”所在行时，字符数组t和s分别为何值？分析是否实现了字符串连接。



单步执行源程序。进跟踪进入strcate时，观察字符数组t和s中的内容，结果并无问题。

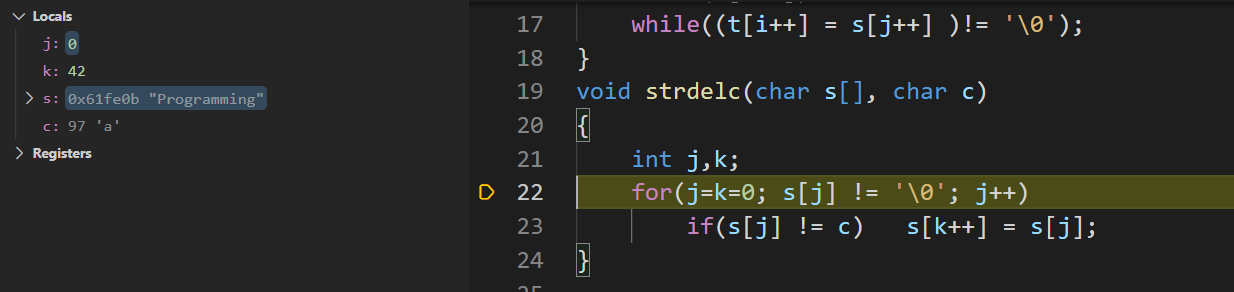


当单步执行光条刚落在第二个while语句所在行时，i为12，t[i]为’L’。结果出现了问题，i本应该为t数组的长度10，t[i]指向’\0’。应该是循环中的t[i++]语句使得，程序在运行到t[i]=’\0’时先i++再进行判断，使得i偏大。而两个字符数组相连，出现越界，产生重大问题。



当单步执行光条落在strcate函数块结束标记即右花括号“}”所在行时，字符数组t和s分别仍为原值不变，分析得是因为数组越界后，t[i]与s[j]的增长完全保持一致，没有起到字符串连接的作用。

（2）跟踪进入函数strdelc时，观察字符数组s中的内容和字符c的值，分析结果是否正确。单步执行for语句过程中，观察字符数组s, j和k值的变化，分析该结果是否存在问题。当单步执行光条落在strdelc函数块结束标记“}”所在行时，字符串s为何值？分析是否实现了所要求的删除操作。



跟踪进入函数strdelc时，字符数组s中的内容并未发生改变，与上一题分析的原因相关，而字符c的值正确。

单步执行单步执行for语句过程中，观察字符数组s, j和k值的变化，k始终小于等于j，保证了删除的正确性，但是由于最后’g’字符未作处理，产生了删除错误，应当删除后重新添加结尾标识符’\0’。

（3）错误修改：

1) 第7行为保证后序修改时不出现数组越界的情况，应大开数组，正确形式为：

char a[1000]="Language", b[1000]="Programming";

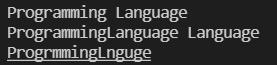
2) 第16行，保证指针指向’\0’的点，正确形式为：

while(t[++i]) ;

3) 第24行为使字符串正确的输出，不会出现运行时错误，正确形式为：

s[k++] = '\0';//增加终止

（4）错误修改后运行结果：



**5.2.2** **源程序修改替换**

下面的源程序用于求解瑟夫问题：M个人围成一圈，从第一个人开始依次从1至N循环报数，每当报数为N时报数人出圈，直到圈中只剩下一个人为止。

（1）请在源程序中的下划线处填写合适的代码来完善该程序。

#include<stdio.h>

#define M 10

#define N 3

int main(void)

{

int a[M], b[M]; /\* 数组a存放圈中人的编号，数组b存放出圈人的编号 \*/

int i, j, k;

for(i = 0; i < M; i++) /\* 对圈中人按顺序编号1—M \*/

a[i] = i + 1;

for(i = M, j = 0; i > 1; i--){

/\* i表示圈中人个数，初始为M个，剩1个人时结束循环；j表示当前报数人的位置 \*/

for(k = 1; k <= N; k++) /\* 1至N报数 \*/

if(++j > i - 1) j = 0;/\* 最后一个人报数后第一个人接着报，形成一个圈 \*/

b[M-i] = j ? \_\_\_\_\_\_\_:\_\_\_\_\_\_; /\* 将报数为N的人的编号存入数组b \*/

if(j)

for(k = --j; k < i; k++) /\* 压缩数组a，使报数为N的人出圈 \*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

for(i = 0;i < M-1; i++) /\* 按次序输出出圈人的编号 \*/

printf(“%6d”, b[i]);

printf(“%6d\n”, a[0]); /\* 输出圈中最后一个人的编号 \*/

return 0;

}

**解答：**

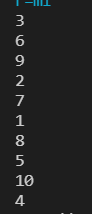
1) 第19行将报数为N的人（现在仅剩的人数中的第j个人）的编号存入数组b，当j>0时，由a数组的定义（a数组从0到i-1存放仅剩余的i个人，而0到i-1依次对应剩下的第1到第i个人的编号）知，报数为N的人的编号为a[j-1]。而j=0时，即j=i而被取模回来，故报数为N的人的编号为a[i-1]。正确形式如下：

b[M-i] = j ? a[j-1]:a[i-1]; /\* 将报数为N的人的编号存入数组b \*/

2) 第22行，压缩数组a，使报数为N的人出圈。即现在仅剩的人数中的第j个人，a[j-1]元素删除，j=j-1。第22行的正确形势故如下：

a[k] = a[k+1];

运行结果如下：



（2）上面的程序中使用数组元素的值表示圈中人的编号，故每当有人出圈时都要压缩数组，这种算法不够精炼。如果采用做标记的办法，即每当有人出圈时对相应数组元素做标记，从而可省掉压缩数组的时间，这样处理效率会更高一些。请采用做标记的办法修改程序，并使修改后的程序与原程序具有相同的功能。

**解答：**

我们仍然用j从0到M-1为M个人标号，以保证数组使用的正确性。而1到M-1依次对应为第1到第M-1人的编号，0为第M个人的编号。并为他们建立是否出圈的标记c[M]数组。若其仍在圈中c[i] = 1，否则为0。

可构建算法流程图如下：

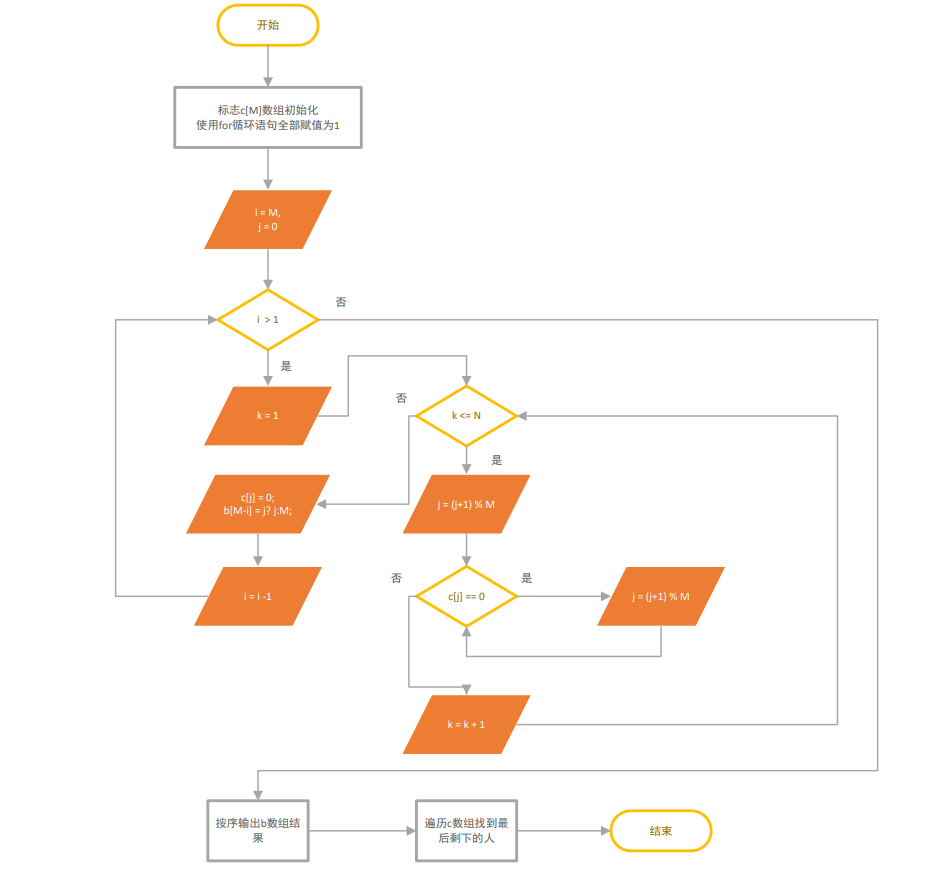


图5-1 源程序修改替换题的程序流程图

#include<stdio.h>

#define M 10

#define N 3

int main(void)

{

int c[M], b[M];

int i, j, k;

for(i = 0; i < M; i++)

c[i] = 1;

for(i = M, j = 0; i > 1; i--){

for(k = 1; k <= N; k++) {/\* 1至N报数 \*/

j = (j + 1) % M;

while(c[j] == 0) {

j = (j + 1) % M;

}

}

c[j] = 0;

b[M-i] = j? j:M;

}

for(i = 0;i < M-1; i++) /\* 按次序输出出圈人的编号 \*/

printf("%d\n", b[i]);

for(i = 0;i < M; i++)

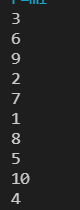
if(c[i])

printf("%d\n",i? i: M);

return 0;

}

运行结果如下：



**5.2.3 程序设计**

（1） 输入一个整数，将它在内存中二进制表示的每一位转化成对应的数字字符并且存放到一个字符数组中，然后输出该整数的二进制表示。

**解答：**

1）本题为简单的循环流程题目，但是需要借助数组进行编写。

可构建算法流程图如下：

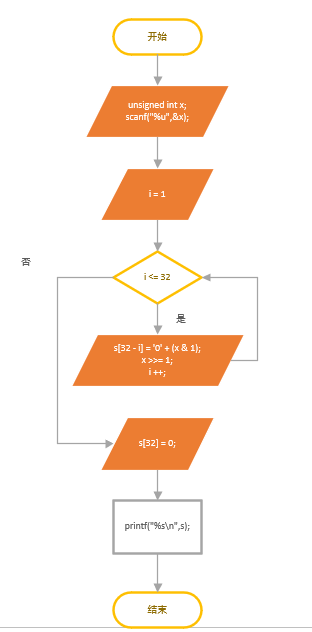


图5-2 编程题1的程序流程图

2）源程序清单

#include <stdio.h>

char s[100];

int main () {

unsigned int x;

scanf("%u",&x);

for(int i = 1; i <= 32; i++) {

s[32 - i] = '0' + (x & 1);

x >>= 1;

}

s[32] = 0;

printf("%s\n",s);

return 0;

}

3）测试

（a） 测试数据：

7

-7

456456456

-987987987

（b） 对应测试数据的运行结果截图









图5-3 编程题1的测试运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（2）编写一个C程序，要求采用模块化程序设计思想，将相关功能用函数实现，并提供菜单选项。该程序具有以下功能：

①“成绩输入”，输入n个学生的姓名和C语言课程的成绩。

②“成绩排序”，将成绩按从高到低的次序排序，姓名同时进行相应调整。成绩相同的，按照输入先后次序排列。

③“成绩输出”，输出排序后所有学生的姓名和C语言课程的成绩。

④“成绩查找”，输入一个C语言课程成绩值，用二分查找进行搜索。如果查找到有该成绩，则输出该成绩学生的姓名和C语言课程的成绩；否则，输出提示“not found!”。

**解答：**

1）解题思路：

本题分析与建模后应完成如下函数：

1. input函数：输入学生数量n，完成n个学生的姓名和C语言成绩的输入至两个储存信息的数组Name[1000][1000], Score[1000]。Sorted标志变量设为0，保证成绩输出和查找时的信息有序。
2. sort函数：按照成绩将Name和Score数组排序，本程序使用了冒泡排序法。比较相邻的元素的Score。如果第一个比第二个小，就交换他们（包含Name和Score），对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最小的数。持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。越大的元素会经由交换慢慢"浮"到数列的顶端。Sorted标志变量置为1，表明学生已经按照成绩排序，信息有序。
3. output函数：若Sorted标志变量为0，则需调用sort函数排序后按序输出。若Sorted标志变量为1，则直接按序输出。
4. Find函数：若Sorted标志变量为0，则需调用sort函数排序后再进行二分查找。二分查找就是将查找的键和子数组的中间键作比较，如果被查找的键小于中间键，就在左子数组继续查找；如果大于中间键，就在右子数组中查找，否则中间键就可能是要找的元素。若ans == n + 1 或者 Score[ans] != x，可能不存在该成绩的同学。而也有可能多位同学有同一成绩，应通过向前遍历找到同一成绩的同学的数目。

2）程序清单：

#include <stdio.h>

#include <string.h>

void sort(char Name[][1000], int Score[],int \*n, int \*sorted) {

for(int i = 1; i < (\*n); i++)

for(int j = 1; j <= (\*n) - i; j++) {

if(Score[j] < Score[j+1]) {

int temp = Score[j+1];

Score[j+1] = Score[j];

Score[j] = temp;

char TEMP[1000];

memcpy(TEMP, Name[j+1], sizeof(TEMP));

memcpy(Name[j+1], Name[j], sizeof(TEMP));

memcpy(Name[j], TEMP, sizeof(TEMP));

}

}

(\*sorted) = 1;

}

void input(char Name[][1000], int Score[],int \*n, int \*sorted) {

\*sorted = 0;

scanf("%d", n);

for(int i = 1; i <= \*n; i++)

scanf("%s%d",Name[i], &Score[i]);

printf("%d records were input!\n",\*n);

}

void Find(char Name[][1000], int Score[],int \*n, int \*sorted) {

if(sorted == 0)

sort(Name, Score, n, sorted);

int x = 0;

scanf("%d", &x);

int L = 1, R = \*n, mid, ans = \*n + 1;

while(L <= R) {

mid = (L + R) >> 1;

if(Score[mid] >= x) {

ans = mid;

L = mid + 1;

}

else

R = mid - 1;

}

if(ans == \*n + 1 || Score[ans] != x)

printf("not found!\n");

else {

int tot = 0;

for(int i = 1; i <= ans; i++)

if(Score[ans - i + 1] == x)

tot ++;

for(int i = ans - tot + 1; i <= ans; i++)

printf("%s %d\n",Name[i], Score[i]);

}

}

void output(char Name[][1000], int Score[],int \*n, int \*sorted) {

for(int i = 1; i <= \*n; i++)

printf("%s %d\n",Name[i],Score[i]);

}

int main () {

int opt;

char Name[1000][1000];int Score[1000];int n;int sorted = 0;

while(scanf("%d", &opt) != EOF) {

if(opt == 0)

break;

else if(opt == 1)

input(Name, Score, &n, &sorted);

else if(opt == 2) {

sort(Name, Score, &n, &sorted);

printf("Reorder finished!\n");

}

else if(opt == 3)

output(Name, Score, &n, &sorted);

else if(opt == 4)

Find(Name, Score, &n, &sorted);

}

return 0;

}

3）测试

（a） 测试数据：

根据题目要求以及各种可能出现的错误情况构造数据如表5-1所示。

表5-1 编程题2的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 操作与信息 |
| 例1 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  0 | 6 records were input! | 6 records were input! |
| 例2 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  0 | 6 records were input!  Reorder finished! | 6 records were input!  Reorder finished! |
| 例3 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  3  0 | 6 records were input!  Reorder finished!  Jack 95  Andy 95  Mike 90  Rose 89  Sophia 77  Joe 75 | 6 records were input!  Reorder finished!  Jack 95  Andy 95  Mike 90  Rose 89  Sophia 77  Joe 75 |
| 例4 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  4 89  0 | 6 records were input!  Reorder finished!  Rose 89 | 6 records were input!  Reorder finished!  Rose 89 |
| 例5 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  4  88  0 | 6 records were input!  Reorder finished!  not found! | 6 records were input!  Reorder finished!  not found! |
| 例6 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  2  4 77  0 | 6 records were input!  Reorder finished!  Sophia 77 | 6 records were input!  Reorder finished!  Sophia 77 |
| 例7 | 1  6  Jack 95  Mike 90  Joe 75  Andy 95  Rose 89  Sophia 77  4 95  0 | 6 records were input!  Jack 95  Andy 95 | 6 records were input!  Jack 95  Andy 95 |

（b） 对应测试测试用例1的运行结果如图5-4所示。

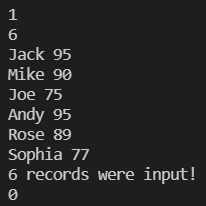


图5-3 编程题2的测试用例一的运行结果

对应测试测试用例2的运行结果如图5-5所示。

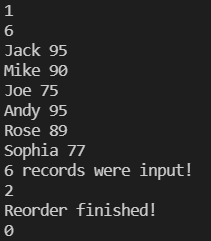


图5-5 编程题2的测试用例二的运行结果

对应测试测试用例3的运行结果如图5-6所示。

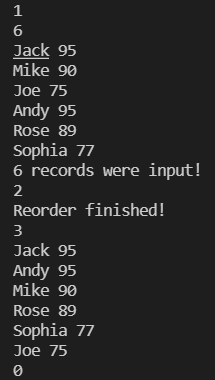


图5-6 编程题2的测试用例三的运行结果

对应测试测试用例4的运行结果如图5-7所示。

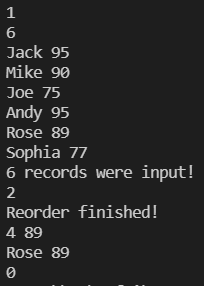


图5-7 编程题2的测试用例四的运行结果

对应测试测试用例5的运行结果如图5-8所示。

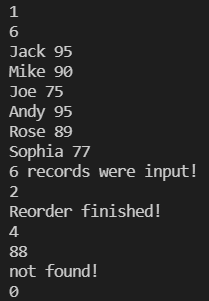


图5-8 编程题2的测试用例五的运行结果

对应测试测试用例6的运行结果如图5-9所示。

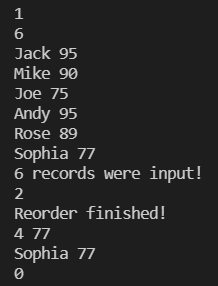


图5-9 编程题2的测试用例六的运行结果

对应测试测试用例7的运行结果如图5-10所示。

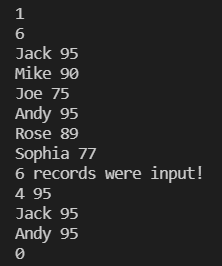


图5-10 编程题2的测试用例七的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（3）求解N皇后问题，即在N×N的棋盘上摆放N个皇后，要求任意两个皇后不能在同一行、同一列、同一对角线上。输入棋盘的大小N（N取值1-10），如果能满足摆放要求，则输出所有可能的摆放法的数量，否则输出“无解。”

**解答：**

1）解题思路：

求解N皇后问题是算法中回溯法应用的一个经典案例。

回溯算法也叫试探法，它是一种系统地搜索问题的解的方法。回溯算法的基本思想是：从一条路往前走，能进则进，不能进则退回来，换一条路再试。

算法基本流程如下：

1.算法开始, 清空棋盘，当前行设为第一行，当前列设为第一列

2.在当前行，当前列的位置上判断是否满足条件(即保证经过这一点的行,列与斜线上都没有两个皇后)，若不满足，跳到第4步

3.在当前位置上满足条件的情形：

3.1在当前位置放一个皇后，若当前行是最后一行，记录一个解；

3.2若当前行不是最后一行，当前行设为下一行, 当前列设为当前行的第一个待测位置；

3.3若当前行是最后一行，当前列不是最后一列，当前列设为下一列；

3.4若当前行是最后一行，当前列是最后一列，回溯，即清空当前行及以下各行的棋盘，然后，当前行设为上一行，当前列设为当前行的下一个待测位置；

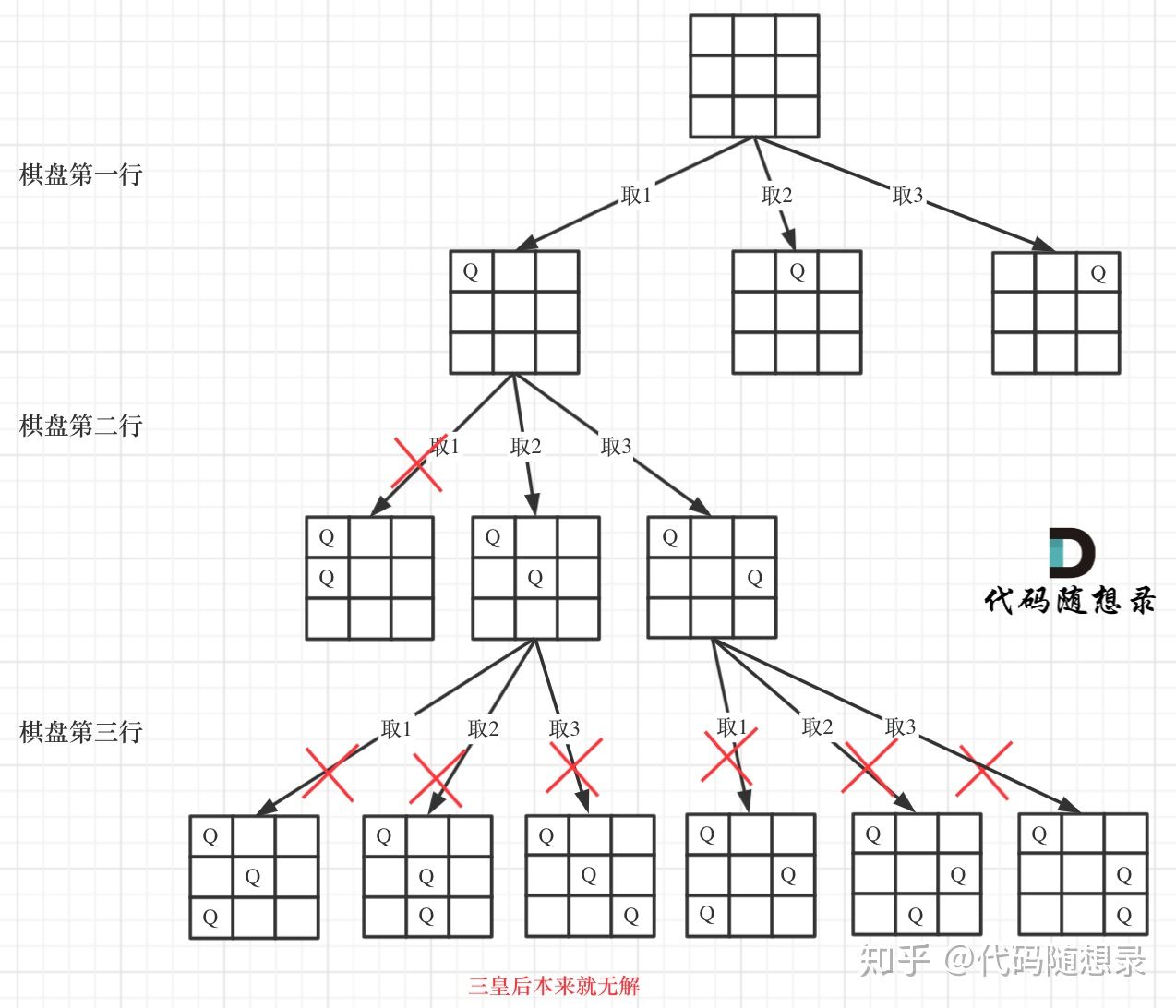
3.5以上返回到第2步

4.在当前位置上不满足条件的情形：

4.1若当前列不是最后一列，当前列设为下一列，返回到第2步;

4.2若当前列是最后一列了，回溯，即，若当前行已经是第一行了，算法退出，否则，清空当前行及以下各行的棋盘，然后，当前行设为上一行，当前列设为当前行的下一个待测位置，返回到第2步;

可有如下模拟（来源自互联网）：



而判断当前位置是否能够放入皇后时，需要判断之前的列，左对角线，右对角线是否为空，我们需要建立三个标记数组，用以标记每个列，左对角线，右对角线是否有放置皇后，因此需对每一个点进行编号，初始化maplex，maprex数组（左对角线编号，右对角线编号），建立一个N×N数对到1到2\*N – 1的映射。

2）程序清单

#include <stdio.h>

int N, tot, row[100], lex[100], rex[100];

int maplex[100][100], maprex[100][100];

void pre () {

int count = 0;

for(int i = 1; i <= N; i++) {

count ++;

for(int j = 1; j <= i; j++)

maplex[j][N-i+j] = count;

}

for(int i = 2; i <= N; i++) {

count ++;

for(int j = 1; j <= N - i + 1; j++)

maplex[i+j-1][j] = count;

}

for(int i = 1; i <= N; i++)

for(int j = 1; j <= N; j++)

maprex[i][j] = maplex[i][N-j+1];

}

void dfs (int n) {

if(n == N + 1) {

tot ++;

return ;

}

for(int j = 1; j <= N; j ++) {

if((!row[j]) && (!lex[maplex[n][j]]) && (!rex[maprex[n][j]])) {

row[j] = lex[maplex[n][j]] = rex[maprex[n][j]] = 1;

dfs(n+1);

row[j] = lex[maplex[n][j]] = rex[maprex[n][j]] = 0;

}

}

}

int main () {

scanf("%d", &N);

pre();

dfs(1);

if(tot == 0)

printf("无解");

else

printf("%d\n", tot);

return 0;

}

3）测试

（a） 测试数据：

根据题目要求以及各种可能出现的错误情况构造数据如表5-2所示。

表5-2 编程题3的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| N |
| 例1 | 1 | 1 | 1 |
| 例2 | 2 | 无解 | 无解 |
| 例3 | 6 | 4 | 4 |
| 例4 | 9 | 352 | 352 |

（b） 对应测试测试用例1的运行结果如图5-11所示。



图5-11编程题3的测试用例一的运行结果

对应测试测试用例2的运行结果如图5-12所示。



图5-12 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图5-13所示。



图5-13 编程题3的测试用例三的运行结果

对应测试测试用例4的运行结果如图5-14所示。



图5-14 编程题3的测试用例四的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

## 5.3 实验小结

主要叙述实验过程中遇到的问题，如何解决的，通过分析、结果问题后的体会。

通过本次实验熟练地掌握了数组的说明、初始化和使用，和一维数组作为函数参数时实参和形参的用法。在改错题一中掌握了部分字符串处理函数的写法，了解了字符串信息的特殊性，也更深刻的理解C语言应时时刻刻关注数组越界问题的内涵。也在分析约瑟夫问题的过程中，加深了自己对数学模型的分析与建构能力。编程题2中掌握了基于分治策略的二分查找算法和选择法排序算法的思想，以及相关算法的实现，初步使用函数写了比较大型的C语言代码。在编程题3的N皇后问题中又一次加深了对递归策略的理解，使用深度优先搜索的方法实际枚举解决问题。