实验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 实验内容及要求

5.2.1 源程序改错

下面是用来将数组a中元素按升序排序后输出的源程序。分析源程序中存在的问题，并对源程序进行修改，使之能够正确完成任务。

**源程序**

1 #include<stdio.h>

2 int main(void)

3 {

4 int a[10] = {27, 13, 5, 32, 23, 3, 17, 43, 55, 39};

5 void sort(int [],int);

6 int i;

7 sort(a[0],10); **// sort(a, 10);**

8 for(i = 0; i < 10; i++)

9 printf("%6d",a[i]);

10 printf("\n");

11 return 0;

12 }

13 void sort(int b[], int n)

14 {

15 int i, j, t;

16 for (i = 0; i < n - 1; i++)

17 for ( j = 0; j < n - i - 1; j++)

18 if(b[j] < b[j+1])

19 t = b[j], b[j] = b[j+1], b[j+1] = t;

20 }

**运行结果截图**



5.2.2 源程序完善、修改、替换

(1) 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。请在源程序中的下划线处填写合适的代码来完善该程序。

**源程序**

#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? **a[j-1]** : **a[i-1]**; /\* 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 \*/

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

}

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;

}

**运行结果截图**



(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。因此，请采用做标记的办法修改（1）中的程序，并使修改后的程序与（1）中的程序具有相同的功能。

**源程序**

#include<stdio.h>

#define M 10

#define N 3

int main(void)

{

int a[M];

int i, j, k; /\* 这里j为报数，k用来给标记计数，i用来做遍历变量 \*/

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

a[i] = i + 1;

}

i = 0, j = 1;

while (k != M) {

if (a[i]) { /\* a[i]非零 -> 仍在圈里 \*/

if (j == N) { /\* 数到3 \*/

printf("%6d", a[i]);

j = 1; /\* j重置 \*/

a[i] = 0; /\* 做标记 \*/

++k; /\* 计数器++ \*/

}

else {

++j;

}

}

/\* 循环变量处理 - 不管当前元素在不在圈内，指针都要往前走 \*/

if (++i == M) { i = 0; }

}

putchar('\n');

return 0;

}

**运行结果截图（同上）**

5.2.3 跟踪调试源程序

在下面所给的源程序中，函数strncat(s,t,n)本来应该将字符数组t的前n个字符连接到字符数组s中字符串的尾部。但函数strncat在定义时代码有误，不能实现上述功能。请按下面的要求进行操作，并回答问题和排除错误。

（1）单步执行源程序。进入函数strncat后观察表达式s、t和i。当光条落在for语句所在行时，i为何值？当光条落在strncat函数块结束标记（右花括号 }）所在行时, s、t分别为何值？

当光条落在for语句所在行时，i为23；

当光条落在strncat函数块结束标记所在行时，s为字符串a首地址（字符串未改变），t为字符串b首地址。

截图：





（2）分析函数出错的原因，排除错误，使函数正确实现功能，最后写出程序的输出结果。

在遍历s的时候，i走过了’\0’，所以’\0’仍保留在串中间。

**源程序**

#include<stdio.h>

void strncat(char [],char [],int);

int main(void)

{

char a[50]="The adopted symbol is ",b[27]="abcdefghijklmnopqrstuvwxyz";

strncat(a, b, 4);

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

return 0;

}

void strncat(char s[],char t[], int n)

{

int i = 0, j;

while(s[i++]) ;

**// --i;**

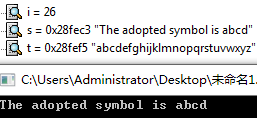
for(j = 0; j < n && t[j];)

s[i++] = t[j++];

s[i] = '\0';

}

**运行结果截图**



5.2.4 程序设计

编写并上机调试运行能实现以下功能的程序。

（1）编写一个程序,从键盘读取数据，对一个3×4矩阵进行赋值，求其转置矩阵，然后输出原矩阵和转置矩阵。

**算法流程**

1. entrMatrix()输入矩阵
2. reprOriginal()打印原矩阵
3. reprTransposed()打印转置后矩阵

（其中，reprOriginal()在打印时先循环第二个下标，reprTransposed()则先循环第一个下标）

**源程序**

#include <stdio.h>

#define ROW 3

#define COL 4

int matrix[ROW][COL] = {0};

void entrMatrix(int [ROW][COL]);

void reprOriginal(int [ROW][COL]);

void reprTransposed(int [ROW][COL]);

void main(void) {

entrMatrix(matrix);

reprOriginal(matrix);

putchar('\n');

reprTransposed(matrix);

return;

}

void entrMatrix(int m[ROW][COL]) {

int i, j;

for (j = 0; j < ROW; ++j) {

for (i = 0; i < COL; ++i) {

scanf("%d", \*(m + j\*COL + i));

} // end of ROW

} // end of all

return;

}

void reprOriginal(int m[ROW][COL]) {

int i, j;

for (j = 0; j < ROW; ++j) {

for (i = 0; i < COL; ++i) {

printf("%5d", \*\*(m + j\*COL + i));

} // end of ROW

putchar('\n');

} // end of all

return;

}

void reprTransposed(int m[ROW][COL]) {

int i, j;

for (i = 0; i < COL; ++i) {

for (j = 0; j < ROW; ++j) {

printf("%5d", \*\*(m + j\*COL + i));

} // end of COL

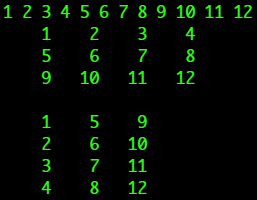
putchar('\n');

} // end of all

return;

}

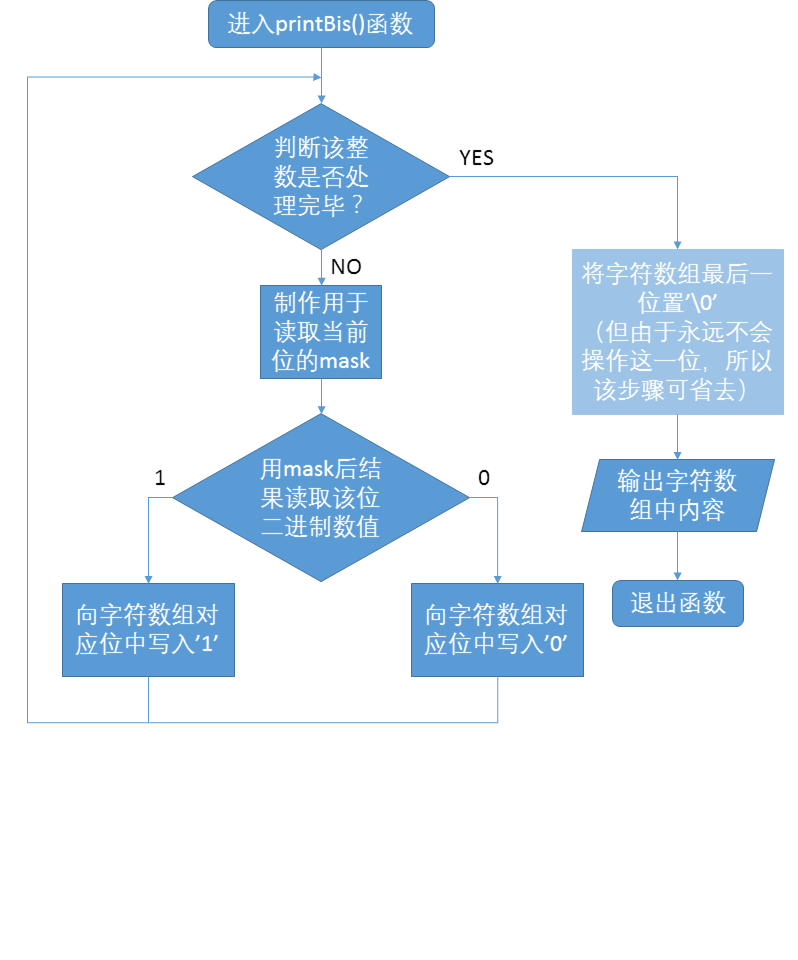
**运行结果截图**



（2）编写一个程序, 其功能要求是：输入一个整数，将它在内存中二进制表示的每一位转换成为对应的数字字符，存放到一个字符数组中，然后输出该整数的二进制表示。

**算法流程**

这里只给出关键函数算法：

****

**源程序**

#include <stdio.h>

void printBits(int, char \*);

void main(void) {

unsigned N;

scanf("%u", &N); getchar();

int nums[120] = {0};

char strNums[sizeof(int)\*8+1] = {'\0'};

unsigned i;

for (i = 0; i < N; ++i) {

scanf("%d", (nums + i));

}

for (i = 0; i < N; ++i) {

printBits(nums[i], strNums);

}

}

void printBits(int dec, char \*str) {

int cur;

for (cur = sizeof(int)\*8-1; cur >= 0; --cur) {

str[sizeof(int)\*8-1-cur] = ( ( dec & (1<<cur) ) ? '1' : '0' );

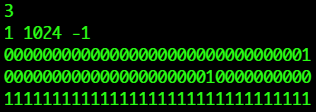
}

str[sizeof(int)\*8] = 0;

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

}

**运行结果截图**



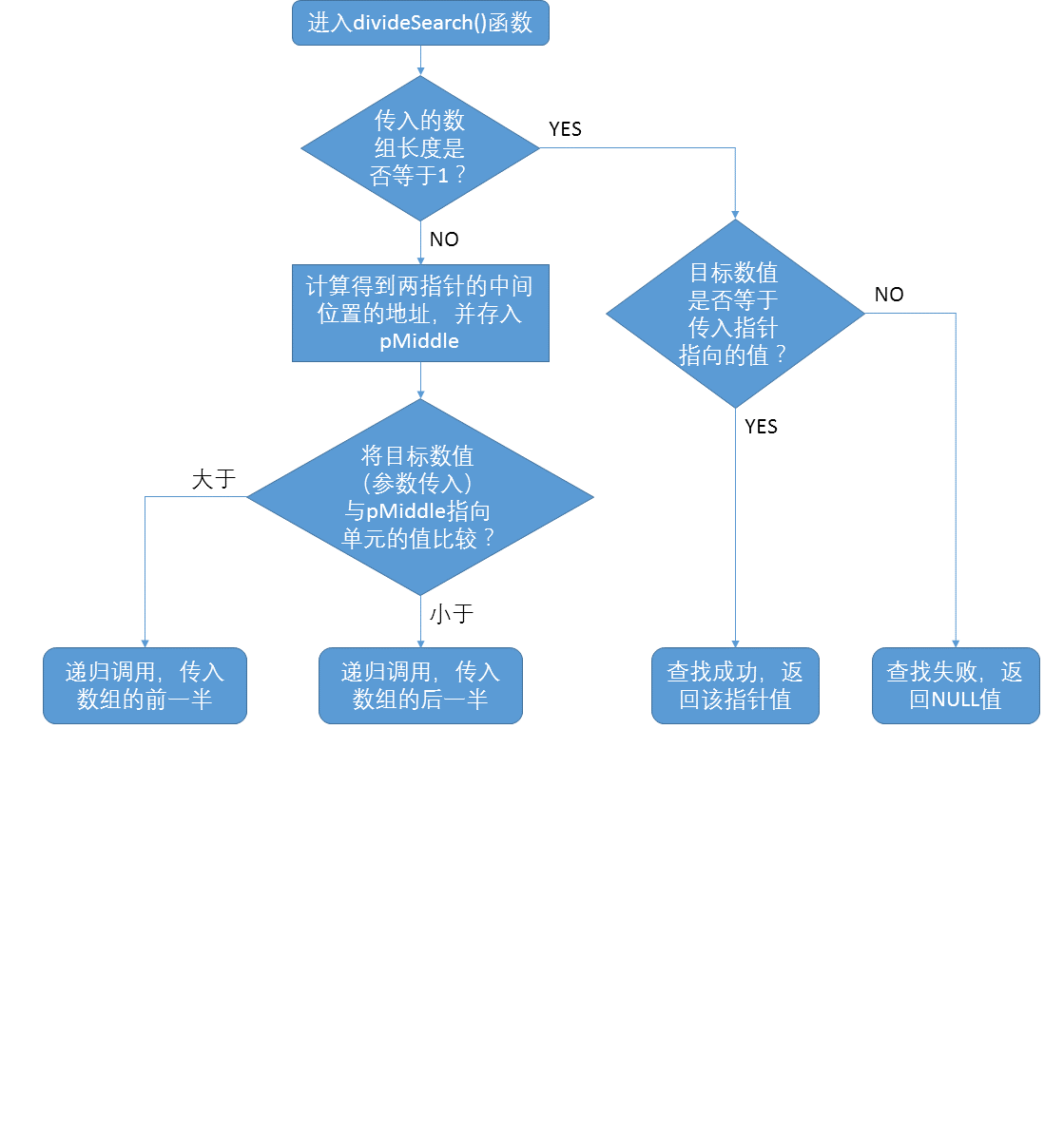
（3）编写一个程序, 其功能要求是：输入n个学生的姓名和C语言课程的成绩，将成绩按从高到低的次序排序，姓名同时作相应调整，输出排序后学生的姓名和C语言课程的成绩。然后，输入一个C语言课程成绩值，用二分查找进行搜索。如果查找到有该成绩，输出该成绩同学的姓名和C语言课程的成绩；否则输出提示“not found!”。

**算法流程**

（注：这里将学生姓名与其C语言成绩分别存在两个相互独立的数组中，但相同学生的信息在两个数组中下标相同。输入和在排序过程中交换数据时，两个数组对应的数据都需要交换）

（排序使用冒泡法进行排序）

二分查找算法（注：此时成绩已经过从大到小的排序）：



**源程序**

#include <stdio.h>

#include <string.h>

unsigned char \* divideSearch(unsigned char target, unsigned char \*start, unsigned count);

char Student\_name[100][21] = {'\0'};

unsigned char Student\_grade[100] = {0};

void main(void) {

unsigned N;

scanf("%u", &N); getchar();

unsigned idx, n; // loop var

char ctmp[21]; unsigned char uctmp;

for (idx = 0; idx < N; ++idx) {

scanf("%s %hhu", \*(Student\_name+idx), (Student\_grade+idx));

getchar(); // swallow '\n'

}

for (idx = 0; idx < N; ++idx) {

for (n = 0; n < N-1-idx; ++n) {

if (Student\_grade[n] < Student\_grade[n+1]) {

uctmp = Student\_grade[n+1];

Student\_grade[n+1] = Student\_grade[n];

Student\_grade[n] = uctmp;

strcpy(ctmp, \*(Student\_name+n+1));

strcpy(\*(Student\_name+n+1), \*(Student\_name+n));

strcpy(\*(Student\_name+n), ctmp);

}

}

}

for (idx = 0; idx < N; ++idx) {

printf("%-20s %hhu\n", Student\_name[idx], Student\_grade[idx]);

}

putchar('\n');

////////////////////////////////

unsigned M;

unsigned char search\_cont[100] = {0};

scanf("%u", &M); getchar();

if (M > N) {

puts("There isn't that much students here!");

return;

}

for (idx = 0; idx < M; ++idx) {

scanf("%hhu", (search\_cont+idx));

getchar();

}

for (n = 0; n < M; ++n) {

unsigned char \* cur = divideSearch(search\_cont[n], Student\_grade, N);

if (cur != NULL) {

printf("%-20s %hhu\n", Student\_name[cur-Student\_grade], \*cur);

}

else {

puts("Not found!");

}

} // end of search loop

return;

}

unsigned char \* divideSearch(unsigned char target, unsigned char \*start, unsigned count) {

unsigned char \* pMiddle = start + (int)count/2;

if (target == \*pMiddle) { return pMiddle; }

else if (count == 1) { return NULL; }

else if (target > \*pMiddle) { return divideSearch(target, start, count/2); }

else if (target < \*pMiddle) { return divideSearch(target, pMiddle, count - count/2); }

else {

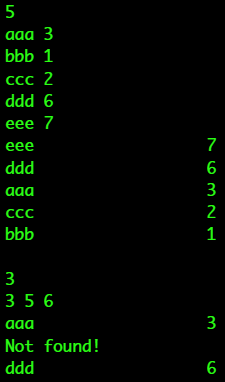
puts("Unexpected operation!");

return NULL;

}

}

**运行结果截图**



5.2.5 选做题

（1）编写函数strnins(s,t,n)，其功能是：可将字符数组t中的字符串插入到字符数组s中第n个字符的后面。

**算法流程**

strnins()功能实现：

1. 通过将数组片段整体后移，将需要占用的空间腾出来

（注：移动过程中采用从后向前的顺序，否则片段前面的数据可能会冲掉片段后面的数据，导致后面的数据丢失、错误）

1. 向腾出的空间写入要插入的字符

**源程序**

#include <stdio.h>

unsigned myStrlen(char s[]) {

unsigned i;

for (i = 0; s[i]; ++i) ;

return i;

}

char \* strnins(char s[], char t[], unsigned n) {

if (n > myStrlen(s)) { return NULL; }

unsigned i, j = myStrlen(t);

// 挪动s中n之后的元素

for (i = myStrlen(s)-1; i >= n; --i) {

s[i+j] = s[i];

}

// 在挪出来的空位中填充t

for (i = n; t[i-n]; ++i) {

s[i] = t[i-n];

}

return s;

}

void main(void) {

char s[20] = "abcdefg", t[20] = "123";

puts(strnins(s, t, 2));

}

**运行结果截图**



（2）编写一个实现八皇后问题的程序，即：在8×8方格国际象棋盘上放置8个皇后，任意两个皇后不能位于同一行、同一列或同一斜线（正斜线或反斜线）上，并输出所有可能的放法。

**算法流程**

这里采用模拟八进制数进位的方式来列举出每一种情况。

（也可以用实际的数值代替每一种情况，但是这里没有使用这种实现方法）

checkerboard数组中每一位数代表该列（行）上，皇后所在的行（列）。这样省去了判断是否有棋子在同一列（行）上的步骤，因为数组的一个元素单元中只能存放一个数值。

printCheckerBoard()函数用于打印棋盘

judge()函数用于判断是否符合“八皇后”的情况。

plusOne()函数用于在checkerboard数组上模拟八进制加一的情况，传入的参数为需要加一的位置（checkerboard的下标）：

1. 如果传入的值为-1，则表明已经遍历了所有情况，返回False（0）；
2. 如果指定位的值不为8，则加一操作合法，加一后返回True（1）；
3. 如果指定位的值为8，则当前位置初始值（1），递归调用，对当前位的前一位进行加一操作。

**源程序**

#include <stdio.h>

int checkerboard[8] = {1, 1, 1, 1, 1, 1, 1, 1};

// no need for judging row - only one 'i' in a row

void printCheckerboard(void) {

int i, j;

for (i = 0; i < 8; ++i) {

for (j = 1; j <= 8; ++j) {

putchar((checkerboard[i] == j) ? 'i' : '\_');

putchar(' ');

}

putchar('\n');

}

putchar('\n');

}

int judge(void) {

int i, j;

// column

for (i = 0; i < 8; ++i) {

for (j = 0; j < 8; ++j) {

if (checkerboard[i] == checkerboard[j] && i != j) {

return 0;

}

}

}

// pos\_diagonal

for (i = 0; i < 8; ++i) {

for (j = 0; j < 8; ++j) {

if (checkerboard[i]+j == checkerboard[j]+i && i != j) {

return 0;

}

}

}

// neg\_diagonal

for (i = 0; i < 8; ++i) {

for (j = 0; j < 8; ++j) {

if (checkerboard[j]+j == checkerboard[i]+i && i != j) {

return 0;

}

}

}

return 1;

}

// simulating octal +1 and carry

int plusOne(int cur) {

// exit status

if (cur == -1) { return 0; }

// no carry

if (checkerboard[cur] != 8) {

checkerboard[cur]++;

return 1;

}

// carry

else if (checkerboard[cur] == 8) {

checkerboard[cur] = 1;

// pass value directly

switch (plusOne(cur-1)) {

case 0: { return 0; }

case 1: { return 1; }

default: { puts("This should not show!"); return -1; }

}

}

}

void main(void) {

int count = 0;

// no need to judge the first case

while (plusOne(7)) {

if (judge()) {

printCheckerboard();

++count;

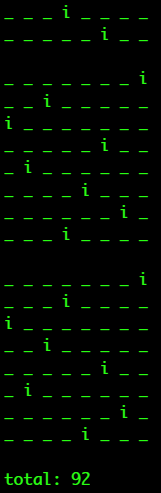
}

}

printf("total: %d case(s)\n", count);

}

**运行结果截图**



5.2.6 自设题

实现简单的列表切片功能：静态数组，正向切片可循环

语法：直接输入数字可获取对应元素；r"[(0-9):(0-9)]"获得两数之间的所有元素。

**算法流程**

用取余运算实现绕回。

getFromArray()函数中有简单的语义分析：

输入的第一个字符是否是'['：

1. 是：切片操作。继续读取，':'之前的数为开始位置，之后直到']'的数为结束位置
2. 否：直接用下标取元素

**源程序**

#include <stdio.h>

unsigned char str2int(char c) {

return c - '0';

}

void getFromArray(char oper[], int \* array) {

unsigned char start = 0, end = 0;

unsigned i;

if (oper[0] != '[') {

for (i = 0; oper[i]; ++i) {

start += str2int(oper[i]);

}

printf("%4d\n", array[start]);

return;

}

for (i = 1; oper[i] != ':'; ++i) {

start += str2int(oper[i]);

}

for (++i; oper[i] != ']'; ++i) {

end += str2int(oper[i]);

}

for (i = start; i < end-1; ++i) {

printf("%4d", \*(array + i%5) );

}

printf("%4d\n", \*(array + i%5));

return;

}

void main(void) {

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

char oper[10] = {'\0'};

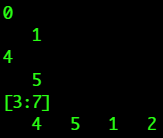
while (scanf("%s", oper), getchar(), oper[0] != 'q') {

getFromArray(oper, array);

}

}

**运行结果截图**



5.3 实验小结

在本次实验中，发现程序设计题第3题在提交至作业系统的时候，我并没有使用二分法，于是在实验课上修改。在重写查找算法的过程中，我充分认识到了认真思考递归结束条件的重要性，必须要全面、不遗漏任何一种情况，否则就会core dump！在以后的编程中，在碰到递归的时候要多加小心！