

**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： 2020.12.13**

**软件工程**

**目 录**

[**6 指针实验 1**](#_Toc404837920)

[6.1 实验目的 1](#_Toc404837921)

[6.2 实验内容 1](#_Toc404837922)

[6.3 实验小结 20](#_Toc404837923)

# 6 指针实验

## 6.1 实验目的

（1）熟练掌握指针的说明、赋值、使用。

（2）掌握用指针引用数组的元素，熟悉指向数组的指针的使用。

（3）熟练掌握字符数组与字符串的使用，掌握指针数组及字符指针数组的用法。

（4）掌握指针函数与函数指针的用法。

（5）掌握带有参数的main函数的用法。

## 6.2 实验内容

**6.2.1 源程序改错**

**1．程序改错**

在下面所给的源程序中，函数strcopy(t, s)的功能是将字符串s复制给字符串t，并且返回串t的首地址。请单步跟踪程序，根据程序运行时出现的现象或观察到的字符串的值，分析并排除源程序的逻辑错误，使之能按照要求输出如下结果：

Input a string:

programming↙ （键盘输入）

programming

Input a string again:

language↙ （键盘输入）

language

1 #include<stdio.h>

2 char \*strcopy(char \*, const char \*);

3 int main(void)

4 {

5 char \*s1, \*s2, \*s3;

6 printf("Input a string:\n", s2);

7 scanf("%s", s2);

8 strcopy(s1, s2);

9 printf("%s\n", s1);

10 printf("Input a string again:\n", s2);

11 scanf("%s", s2);

12 s3 = strcopy(s1, s2);

13 printf("%s\n", s3);

14 return 0;

15}

16

17/\*将字符串s复制给字符串t，并且返回串t的首地址\*/

18char \* strcopy(char \*t, const char \*s)

19{

20 while(\*t++ = \*s++);

21 return (t);

22}

**解答：**

1. 错误修改：

1) 第2行为保证编译正确，应引入新的库，正确形式为：

#include <stdlib.h>//导入新的库

2) 第7行为保证后序输入和处理时不会发生段超越的错误，应对指针初始化，正确形式为：

s1 = (char \*)malloc(sizeof(char) \* 100);//分配内存.

s2 = (char \*)malloc(sizeof(char) \* 100);

s3 = (char \*)malloc(sizeof(char) \* 100);

3) 第24，25行为使函数返回正确的字符指针首地址，应设立新的指针，正确形式为：

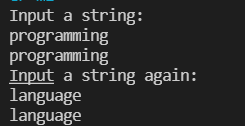
char \*p = t;//不修改t，修改p

while(\*p++ = \*s++);

4) 第26行为使字符串正确输出应增加终止符，正确形式为：

\*p++ = '\0';//终止符

（2）错误修改后运行结果：



**6.2.2** **源程序修改替换**

（1）下面程序中函数strsort用于对字符串进行升序排序，在主函数中输入N个字符串（字符串长度不超过49）存入通过malloc动态分配的存储空间，然后调用strsort对这N个串按字典序升序排序。

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>

#include<\_\_\_\_\_\_\_\_\_>

#include<string.h>

#define N 4

/\*对指针数组s指向的size个字符串进行升序排序\*/

void strsort(char \*s[], int size)

{

\_\_\_\_\_\_\_temp;

int i, j;

for(i=0; i<size-1; i++)

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

if (\_\_\_\_\_\_\_\_\_\_\_)

{

temp = s[j];

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

s[j+1] = temp;

}

}

int main()

{

int i;

char \*s[N], t[50];

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

{

gets(t);

s[i] = (char \*)malloc(strlen(t)+1);

strcpy(\_\_\_\_\_\_\_);

}

strsort(\_\_\_\_\_\_\_\_);

for (i=0; i<N; i++) {puts(s[i]); free(s[i]);}

return 0;

}

**解答：**

1. 第2行，为正确使用malloc函数，正确形式如下：

#include<stdlib.h>

2) 第3行，由二维数组存储的本质实际为指向一维字符数组的指针已知，为完成数组的交换，应定义一维字符指针变量，故正确形式如下：

char \*temp;

3) 第12行，为实现按字典序冒泡排序，应使用strcmp函数，strcmp(char \*a, char \*b) 当a的字典序大于b时返回正数，故正确形式如下：

if (strcmp(s[j], s[j+1]) > 0)

4) 第15行，为实现s[j]和s[j+1]的交换，故正确形式如下：

s[j] = s[j+1];

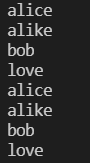
5) 第28行，正确保存每一个字符串，正确形式如下：

strcpy(s[i], t);

6) 第30行，正确调用定义的void strsort(char \*s[], int size)函数，正确形式如下：

strsort(s, N);

程序运行结果如下：



②数组作为函数参数其本质类型是指针。例如，对于形参char \*s[]，编译器将其解释为char \*\*s，两种写法完全等价。请用二级指针形参重写strsort函数，并且在该函数体的任何位置都不允许使用下标引用。

**解答：**

为保证函数体的任何为止都不允许下表的使用，我们使用两个二维字符指针q,s指向冒泡排序中我们要交换的两个一维字符串数组。随着j增加，两个二维指针不断自加1，可以直接指到下一个字符串。使用\*q, \*s取出字符串。

void strsort(char \*\*s, int size)

{

char \*temp;

int i, j;char \*\*q, \*\*r;

for(i=0; i<size-1; i++)

for (q = s, r = s+1, j=0; j<size-i-1; j++,q++,r++)

if (strcmp(\*q, \*r) > 0)

{

temp = \*q;

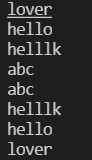
\*q = \*r;

\*r = temp;

}

}

运行结果如下：



（2）下面源程序通过函数指针和菜单选择来调用库函数实现字符串操作；串复制strcpy、串连接strcat或串分解strtok。

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，使之能按照要求输出下面结果：

1 copy string.

2 connect string.

3 parse string.

4 exit.

input a number (1-4) please!

2↙ （键盘输入）

input the first string please!

the more you learn,↙ （键盘输入）

input the second string please!

the more you get. ↙ （键盘输入）

the result is the more you learn, the more you get.

代码如下：

# include<stdio.h>

# include<string.h>

int main (void)

{

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

char a[80], b[80], \*result;

int choice;

while(1)

{

do

{

printf("\t\t1 copy string.\n");

printf("\t\t2 connect string.\n");

printf("\t\t3 parse string.\n");

printf("\t\t4 exit.\n");

printf("\t\tinput a number (1-4) please.\n");

scanf("%d", &choice);

}while(choice<1 || choice>4);

switch(choice)

{

case 1: p = strcpy; break;

case 2: p = strcat; break;

case 3: p = strtok; break;

case 4: goto down;

}

getchar();

printf("input the first string please!\n");

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

printf("input the second string please!\n");

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

result = \_\_\_\_\_\_\_\_\_\_\_(a, b);

printf("the result is %s\n", result);

}

down:

return 0;

}

**解答：**

1. 第2行，为正确声明函数指针，正确形式如下：

char\* (\*p) (char \*a,char \*b);

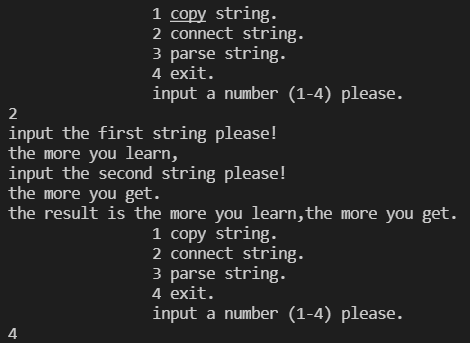
2) 第29，31行，使用C语言读入一行输入，故正确形式如下：

gets(a); gets(b);

3) 第32行，为正确使用函数指针调用所指代的函数，故正确形式如下：

result = p(a, b);

程序运行结果如下：



②函数指针的一个用途是用户散转程序，即通过一个转移表（函数指针数组）来实现多分枝函数处理，从而省去了大量的if语句或者switch语句。转移表中存放了各个函数的入口地址（函数名），根据条件的设定来查表选择执行相应的函数。请使用转移表而不是switch语句重写以上程序。

**解答：**

定义函数指针数组，然后通过函数指针数组实现拥护函数分支的处理。

数组是一个存放相同类型数据的存储空间，我们已经学习了指针数组，比如：

int \*arr[10];//数组的每个元素是int\*

那要把函数的地址存到一个数组中，那这个数组就叫函数指针数组，比如：

int (\*parr1[10])();

分析：parrl先和[]结合，说明parr1是数组，数组的内容是什么呢?是int (\*)() 类型的函数指针。

代码实现如下：

# include<stdio.h>

# include<string.h>

int main (void)

{

char\* (\*p[]) (char\* , char\*) = {0, strcpy, strcat, strtok};

char a[80], b[80], \*result;

int choice;

while(1)

{

do

{

printf("\t\t1 copy string.\n");

printf("\t\t2 connect string.\n");

printf("\t\t3 parse string.\n");

printf("\t\t4 exit.\n");

printf("\t\tinput a number (1-4) please.\n");

scanf("%d", &choice);

}

while(choice<1 || choice>4);

if(choice == 4)

break;

getchar();

printf("input the first string please!\n");

gets(a);//

printf("input the second string please!\n");

gets(b);//

result = p[choice](a, b);//

printf("the result is %s\n", result);

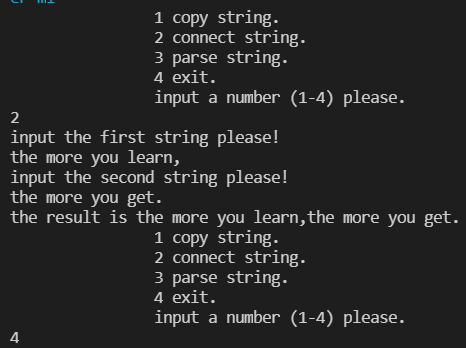
}

down:

return 0;

}

代码运行结果如下：



**6.2.3 跟踪调试题**

请按下面的要求对源程序进行操作，并回答问题和排除错误。

#include "stdio.h"

char \*strcpy(char \*,char \*);

void main(void)

{

char a[20],b[60]="there is a boat on the lake.";

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

}

char \*strcpy(char \*s,char \*t)

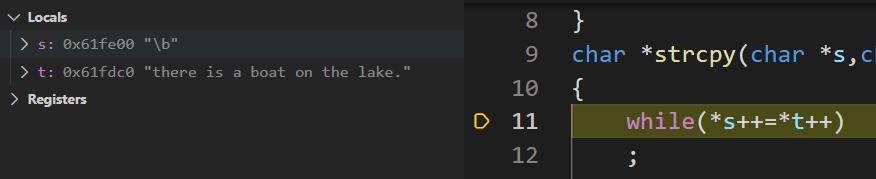
{

while(\*s++=\*t++);

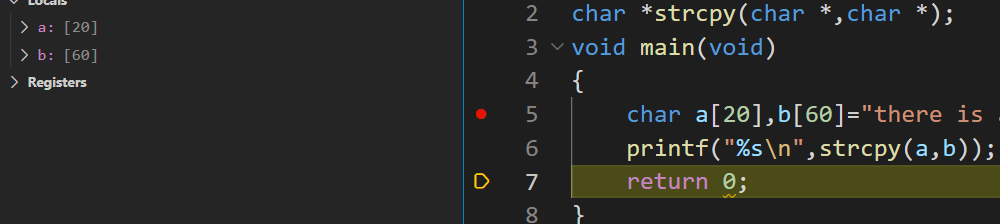
return (s);

}

（1）单步执行。进入strcpy时watch窗口中s为何值？返回main时, watch窗口中s为何值？



由上图容易知道，进入strcpy时watch窗口中s为一些随机序列组成的数组。



返回main时, watch窗口中s不能被发现，因为他是strcpy函数的形参，在外界无法访问。而由于s在strcpy函数内发生了移动，故直接返回s指针导致了程序异常。

（2）排除错误，使程序输出结果为：there is a boat on the lake.

**解答：**

错误修改如下：

1. 第5行，为a数组能容纳下b字符串的复制结果，正确形式如下：

char a[60],b[60]="there is a boat on the lake."

2) 第11行，为避免将主函数的指针指向的位置发生变化，应定义新的临时指针变量指向s字符串首地址，最后返回这个未改变的指针，故正确形式如下：

char \*p = s;

3) 第13行，为保证正确的输出字符串，应增加终止符，故正确形式如下：

\*s++ = '\0';

4) 第14行，为保证返回值正确，应返回指针p，故正确形式如下：

return (p);

程序运行结果如下：



**5.2.4 程序设计**

（1） 指定main函数的参数

在IDE（比如DevC++）中，选择“运行”｜“参数”菜单，在 “传递给主程序的参数”文本框中输入main函数的参数arg1 arg2 arg3，只输入命令行中文件名后的参数，文件名不作为参数输入，参数间以空格隔开。编写程序在命令行输出这三个参数。（注意不同IDE输入参数的方式不相同，可参考各个IDE的使用手册。）

**解答：**

1. 本实验学习指定main函数的参数。

由于main函数不能被其它函数调用，因此不可能在程序内部取得实际值。那么，在何处把实参值赋予main函数的形参呢？实际上，main函数的参数值是从操作系统命令行上获得的。当我们要运行一个可执行文件时，在DOS提示符下键入文件名，再输入实际参数即可把这些实参传送到main的形参中去。

DOS提示符下命令行的一般形式为：

C:\>可执行文件名 参数 参数 ……;

但是应该特别注意的是，main 的两个形参和命令行中的参数在位置上不是一一对应的。因为，main的形参只有二个，而命令行中的参数个数原则上未加限制。argc参数表示了命令行中参数的个数（注意：文件名本身也算一个参数），argc的值是在输入命令行时由系统按实际参数的个数自动赋予的。

gcc 编译器的常见用法

./a.out a.c b.c c.c

此时，我们知道：

argc ---> 4

argv[0] ---> a.out

argv[1] ---> a.c

argv[2] ---> b.c

argv[3] ---> c.c

我们在DevC++中设置参数较为容易。

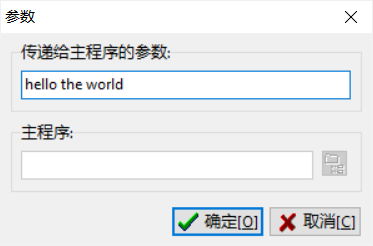


图6-1 编程题1的参数传递添加结果

1. 源程序清单

#include <stdio.h>

int main (int argc, char \*args[]) {

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

printf("%s\n", args[i]);

return 0;

}

3）测试

（a） 测试数据：

参数为Hello the world

（b） 对应测试数据的运行结果截图



图6-2 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）一个长整型变量占4个字节，其中每个字节又分成高4位和低4位。输入一个长整型变量，要求从高字节开始，依次取出每个字节的高4位和低4位并以十六进制数字字符的形式进行显示，通过指针取出每字节。

样例输入：15

样例输出：0000000F

**解答：**

1. 解题思路：

本题的关键在于对于C语言指针以及内存管理的理解。当我读到本题，心中一愣好像描述的不是非常清晰。

但是仔细分析。Longlong类型为四个字节，char类型为一个字节。用一个char\*的指针p来存放操作数的地址。p = p + 3，得到最高位，然后进行四次循环，便可得到计算地结果。

其算法流程图如下：

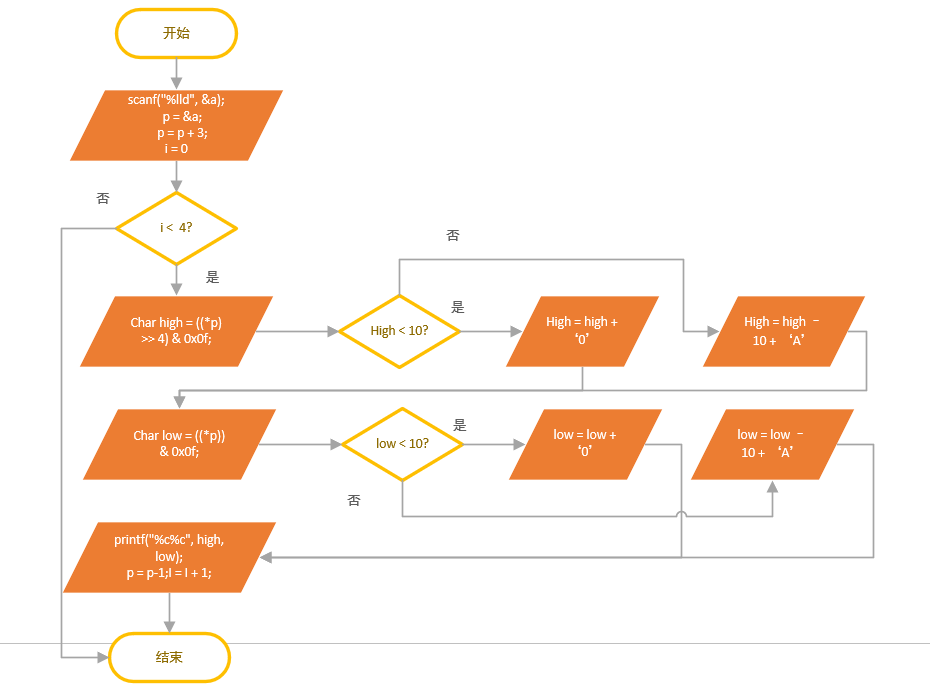


图6-3编程题2的算法流程示意图

2）程序清单：

#include <stdio.h>

int main () {

long long a;

char \*p;

scanf("%lld", &a);

p = &a;

p = p + 3;

for(int i = 0; i < 4; i++) {

char high = ((\*p) >> 4) & 0x0f;

if(high < 10)

high = high + '0';

else

high = high - 10 + 'A';

char low = (\*p) & 0x0f;

if(low < 10)

low = low + '0';

else

low = low - 10 + 'A';

printf("%c%c", high, low);

p = p-1;

}

return 0;

}

3）测试

（a） 测试数据：

15

2147483647

456456456

（b） 对应测试数据的运行结果截图







图6-4编程题2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。

（3）旋转是图像处理的基本操作，编程实现一个将一个图像逆时针旋转90°。提示：计算机中的图像可以用一个矩阵来表示，旋转一个图像就是旋转对应的矩阵。将旋转矩阵的功能定义成函数，通过使用指向数组元素的指针作为参数使该函数能处理任意大小的矩阵。要求在main函数中输入图像矩阵的行数n和列数m，接下来的n行每行输入m个整数，表示输入的图像。输出原始矩阵逆时针旋转90°后的矩阵。

**解答：**

1. 解题思路：

本题主要考察指针作为函数参数程序的编写。像数组、字符串、动态分配的内存等都是一系列数据的集合，没有办法通过一个参数全部传入函数内部，只能传递它们的指针，在函数内部通过指针来影响这些数据集合。

二维数组作为函数的参数，实参可以直接使用二维数组名，在被调用函数中可以定义形参所有维数的大小，也可以省略一维大小的说明。例如：

void find（char a[3][10]）;

void find (char a[ ][10]);

也可以使用数组指针来作为函数参数，例如：

void find （char (\*p)[10]);

但是不能像下面这样使用，例如：

void find（char a[ ][ ]）;

void find (char a[3][ ]);

而本题程序实现并不困难，可在函数中定义临时数组b用以存放逆时针旋转90°后的矩阵a，然后交换n，m。将临时数组b复制回a中。

而逆时针旋转九十度，即将a[i][j]放置于b[m - j + 1][i] 中（第一维和第二维均从1开始编号）。

2）程序清单

#include <stdio.h>

void RotateLeft90(int (\*a)[200], int\* n, int\* m) {

int b[200][200];

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

for(int j = 1; j <= \*m; j++)

b[\*m - j + 1][i] = a[i][j];

int temp = \*n;

\*n = \*m;

\*m = temp;

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

for(int j = 1; j <= \*m; j++)

a[i][j] = b[i][j];

}

int main () {

int n, m;int a[200][200];

scanf("%d%d", &n, &m);

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

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

scanf("%d",&a[i][j]);

RotateLeft90(a, &n, &m);

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

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

printf("%d%s",a[i][j],j == m? "\n":" ");

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6-1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| N和带旋转的矩阵 |
| 例1 | 1 1  5 | 5 | 5 |
| 例2 | 2 3  1 5 3  3 2 4 | 3 4  5 2  1 3 | 3 4  5 2  1 3 |
| 例3 | 10 10  38 7719 21238 2437 8855 11797 8365 32285 10450 30612  5853 28100 1142 281 20537 15921 8945 26285 2997 14680  20976 31891 21655 25906 18457 1323 28881 2240 9725 32278  2446 590 840 18587 16907 21237 23611 12617 12456 867  29533 6878 28223 17887 31597 20584 12212 31111 7578 17066  7629 29404 12279 13505 24388 11649 12329 7176 2331 19264  22114 14136 26928 1102 21652 8404 24337 27856 5598 24772  14097 13213 4683 16703 15260 15942 2747 27375 28871 18004  16673 3152 11819 23504 239 4186 2804 28937 3023 10335  20533 21393 16020 11574 25983 13961 624 7065 27569 12830 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 |

（b） 对应测试测试用例1的运行结果如图6-5所示。



图6-5 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6-6所示。

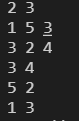


图6-6 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-7所示。

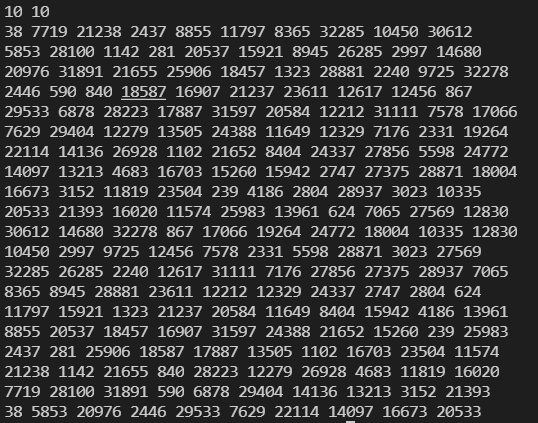


图6-7编程题3的测试用例三的运行结果

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）旋转是图像处理的基本操作，编程实现一个将一个图像逆时针旋转90°。提示：计算机中的图像可以用一个矩阵来表示，旋转一个图像就是旋转对应的矩阵。将旋转矩阵的功能定义成函数，通过使用指向数组元素的指针作为参数使该函数能处理任意大小的矩阵。要求在main函数中输入图像矩阵的行数n和列数m，接下来的n行每行输入m个整数，表示输入的图像。输出原始矩阵逆时针旋转90°后的矩阵。

**解答：**

1. 解题思路：

本题主要考察指针作为函数参数程序的编写。像数组、字符串、动态分配的内存等都是一系列数据的集合，没有办法通过一个参数全部传入函数内部，只能传递它们的指针，在函数内部通过指针来影响这些数据集合。

二维数组作为函数的参数，实参可以直接使用二维数组名，在被调用函数中可以定义形参所有维数的大小，也可以省略一维大小的说明。例如：

void find（char a[3][10]）;

void find (char a[ ][10]);

也可以使用数组指针来作为函数参数，例如：

void find （char (\*p)[10]);

但是不能像下面这样使用，例如：

void find（char a[ ][ ]）;

void find (char a[3][ ]);

而本题程序实现并不困难，可在函数中定义临时数组b用以存放逆时针旋转90°后的矩阵a，然后交换n，m。将临时数组b复制回a中。

而逆时针旋转九十度，即将a[i][j]放置于b[m - j + 1][i] 中（第一维和第二维均从1开始编号）。

2）程序清单

#include <stdio.h>

void RotateLeft90(int (\*a)[200], int\* n, int\* m) {

int b[200][200];

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

for(int j = 1; j <= \*m; j++)

b[\*m - j + 1][i] = a[i][j];

int temp = \*n;

\*n = \*m;

\*m = temp;

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

for(int j = 1; j <= \*m; j++)

a[i][j] = b[i][j];

}

int main () {

int n, m;int a[200][200];

scanf("%d%d", &n, &m);

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

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

scanf("%d",&a[i][j]);

RotateLeft90(a, &n, &m);

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

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

printf("%d%s",a[i][j],j == m? "\n":" ");

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6-1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| N和待旋转的矩阵 |
| 例1 | 1 1  5 | 5 | 5 |
| 例2 | 2 3  1 5 3  3 2 4 | 3 4  5 2  1 3 | 3 4  5 2  1 3 |
| 例3 | 10 10  38 7719 21238 2437 8855 11797 8365 32285 10450 30612  5853 28100 1142 281 20537 15921 8945 26285 2997 14680  20976 31891 21655 25906 18457 1323 28881 2240 9725 32278  2446 590 840 18587 16907 21237 23611 12617 12456 867  29533 6878 28223 17887 31597 20584 12212 31111 7578 17066  7629 29404 12279 13505 24388 11649 12329 7176 2331 19264  22114 14136 26928 1102 21652 8404 24337 27856 5598 24772  14097 13213 4683 16703 15260 15942 2747 27375 28871 18004  16673 3152 11819 23504 239 4186 2804 28937 3023 10335  20533 21393 16020 11574 25983 13961 624 7065 27569 12830 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 | 30612 14680 32278 867 17066 19264 24772 18004 10335 12830  10450 2997 9725 12456 7578 2331 5598 28871 3023 27569  32285 26285 2240 12617 31111 7176 27856 27375 28937 7065  8365 8945 28881 23611 12212 12329 24337 2747 2804 624  11797 15921 1323 21237 20584 11649 8404 15942 4186 13961  8855 20537 18457 16907 31597 24388 21652 15260 239 25983  2437 281 25906 18587 17887 13505 1102 16703 23504 11574  21238 1142 21655 840 28223 12279 26928 4683 11819 16020  7719 28100 31891 590 6878 29404 14136 13213 3152 21393  38 5853 20976 2446 29533 7629 22114 14097 16673 20533 |

（b） 对应测试测试用例1的运行结果如图6-5所示。



图6-8 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6-6所示。

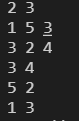


图6-9 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-7所示。

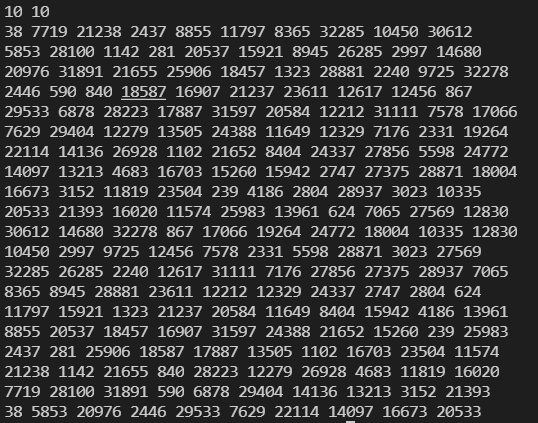


图6-10编程题3的测试用例三的运行结果

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。

（4）输入n行文本，每行不超过80个字符，用字符指针数组指向键盘输入的n行文本（n不作为输入，可理解为循环输入多行，以CTRL+Z结束），删除每一行中的前置空格（' '）和水平制表符（'\t'）。

要求：将删除一行文本中前置空格和水平制表符的功能定义成函数，在main函数中输出删除前置空格符的各行。（并在最后输出换行符。）

**解答：**

1. 解题思路：分析题目，本题的算法流程图可如下表示。

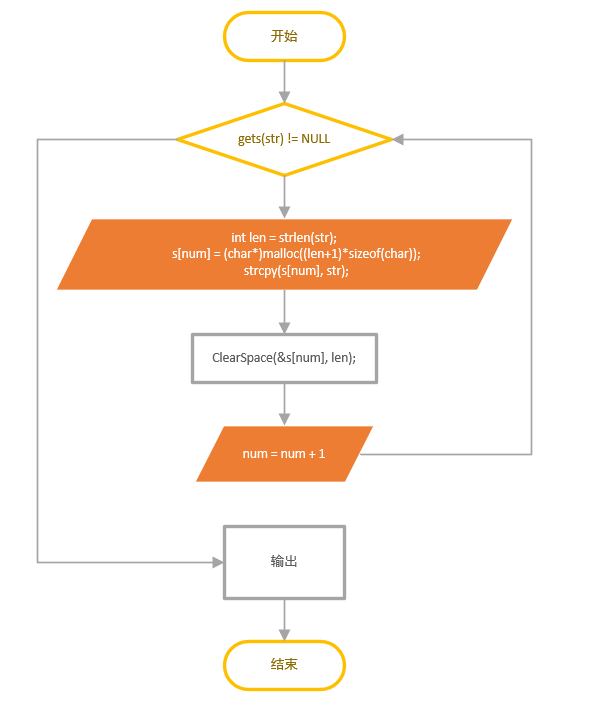


图6-11 编程题4的算法流程示意图

而核心函数ClearSpace(char \*\*a, int len)的算法流程也如下

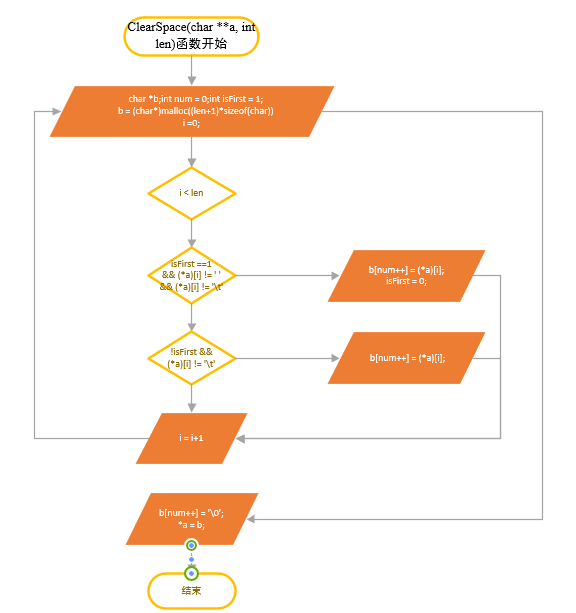


图6-12 编程题4的ClearSpace(char \*\*a, int len)函数的算法流程示意图

我们在此处意识到为了改变字符指针数组中某个元素s[i]指向的内容，必须用指针指向它，所以函数声明时采用了ClearSpace(char \*\*a, int len)的声明方式，调用时s[i]要取地址。函数内部使用数组时也要按照二维字符指针进行去单个字符。

我们也应该注意到为了判断是否开头，我们应该设置标记判断其是否为开头，若为开头，则要把空格和制表符全部删去，并把标记清空；否则，只需要删除制表符即可。

2）程序清单

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void ClearSpace(char \*\*a, int len) {

char \*b;int num = 0;int isFirst = 1;

b = (char\*)malloc((len+1)\*sizeof(char));

for(int i = 0; i < len; i++) {

if(isFirst && (\*a)[i] != ' ' && (\*a)[i] != '\t') {

b[num++] = (\*a)[i];

isFirst = 0;

}

else if(!isFirst && (\*a)[i] != '\t')

b[num++] = (\*a)[i];

}

b[num++] = '\0';

\*a = b;

}

int main () {

int num = 0;

char \*s[1000];

while(1) {

char str[100];

if(gets(str) == NULL)

break;

int len = strlen(str);

s[num] = (char\*)malloc((len+1)\*sizeof(char));

strcpy(s[num], str);

ClearSpace(&s[num], len);

num++;

}

for(int i = 0; i < num; i++)

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

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6-1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 输入文本信息 |
| 例1 | 123 45  67 89  Ctrl+Z | 12345  6789 | 12345  6789 |
| 例2 | aab c  bcd e  Ctrl+Z | aab c  bcde | aab c  bcde |
| 例3 | Ab c d  Lover | Abc d  Lover | Abc d  Lover |

（b） 对应测试测试用例1的运行结果如图6-10所示。

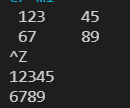


图6-13 编程题4的测试用例一的运行结果

对应测试测试用例2的运行结果如图6-11所示。

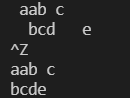


图6-14 编程题4的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-12所示。

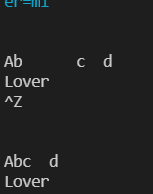


图6-15编程题4的测试用例三的运行结果

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）编写8个任务函数，一个scheduler调度函数和一个execute执行函数。仅在main函数中调用scheduler函数，scheduler函数要求用最快的方式调度执行用户指定的任务函数。

①先设计task0, task1, task2, task3, task4, task5, task6, task7共8个任务函数，每个任务函数的任务就是输出该任务被调用的字符串。例如，第0个任务函数输出“task0 is called!”，第1个任务函数输出“task1 is called!”，以此类推。

②scheduler函数根据键盘输入的数字字符的先后顺序，一次调度选择对应的任务函数。例如，输入：1350并回车，则scheduler函数一次调度选择task1, task3, taks5, task0，然后以函数指针数组和任务个数为参数将调度选择结果传递给execute函数并调用execute函数。

③execute函数根据scheduler函数传递的指针数组和任务个数为参数，按照指定的先后顺序依此调用执行选定的任务函数。

当输入13607122并回车，程序运行结果应当如下：

task1 is called!

task3 is called!

task6 is called!

task0 is called!

task7 is called!

task1 is called!

task2 is called!

task2 is called!

**解答：**

1. 解题思路：

本题的难点在于函数指针数组和函数指针的使用。且在函数中来回传递这些函数指针与数组。

可在scheduler函数中初始化一个用于分配人物的函数指针数组，其中放好了写的9个函数。void (\*p[]) () = {task0, task1, task2, task3, task4, task5, task6, task7, task8, task9};这样便可以避免使用switch的复杂，直接遍历字符数组得到需要的函数，压入work函数指针数组。

将work函数指针数组和总任务数n传递至execute函数，便可轻松地实现函数的调用。

2）程序清单

#include <stdio.h>

void task0 () { printf("task0 is called!\n");}

void task1 () { printf("task1 is called!\n");}

void task2 () { printf("task2 is called!\n");}

void task3 () { printf("task3 is called!\n");}

void task4 () { printf("task4 is called!\n");}

void task5 () { printf("task5 is called!\n");}

void task6 () { printf("task6 is called!\n");}

void task7 () { printf("task7 is called!\n");}

void task8 () { printf("task8 is called!\n");}

void task9 () { printf("task9 is called!\n");}

void execute(void (\*p[])(), int n) {

for(int i = 0; i < n; i++)

p[i]();

}

void scheduler() {

void (\*p[]) () = {task0, task1, task2, task3, task4, task5, task6, task7, task8, task9};

char a[1000];

void (\*work[1000]) ();

scanf("%s",a);

int n = 0;

for(n = 0; a[n] != '\0'; n++)

work[n] = p[a[n]-'0'];

execute(work, n);

}

int main () {

scheduler();

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6-1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| 输入序列 |
| 例1 | 13607122 | task1 is called!  task3 is called!  task6 is called!  task0 is called!  task7 is called!  task1 is called!  task2 is called!  task2 is called! | task1 is called!  task3 is called!  task6 is called!  task0 is called!  task7 is called!  task1 is called!  task2 is called!  task2 is called! |
| 例2 | 123450019941560198 | task1 is called!  task2 is called!  task3 is called!  task4 is called!  task5 is called!  task0 is called!  task0 is called!  task1 is called!  task9 is called!  task9 is called!  task4 is called!  task1 is called!  task5 is called!  task6 is called!  task0 is called!  task1 is called!  task9 is called!  task8 is called! | task1 is called!  task2 is called!  task3 is called!  task4 is called!  task5 is called!  task0 is called!  task0 is called!  task1 is called!  task9 is called!  task9 is called!  task4 is called!  task1 is called!  task5 is called!  task6 is called!  task0 is called!  task1 is called!  task9 is called!  task8 is called! |
| 例3 | 5210 | task5 is called!  task2 is called!  task1 is called!  task0 is called! | task5 is called!  task2 is called!  task1 is called!  task0 is called! |

（b） 对应测试测试用例1的运行结果如图6-13所示。

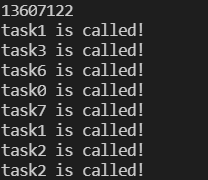


图6-16 编程题5的测试用例一的运行结果

对应测试测试用例2的运行结果如图6-14所示。

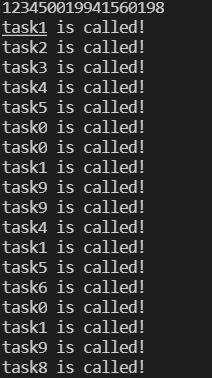


图6-17 编程题5的测试用例二的运行结果

对应测试测试用例3的运行结果如图6-15所示。

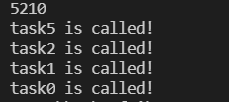


图6-18编程题5的测试用例三的运行结果

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。

## 6.3 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，我觉得我对C语言地重难点——指针有了更深刻的理解。熟练地掌握指针的说明、赋值、使用。通过源程序修改替换题（1）掌握了用指针引用数组的元素，熟悉指向数组的指针的使用。也在众多习题中对如何用函数传递数组，修改指针数组指向地址，这些都是非常有用的小技巧。而我也初次解除了函数指针，函数指针数组这一比较奇怪的东西，并在程序设计题和替换修改题（2）中亲手实现了他们并且理会到了这种方式可以起到分配任务，省去分支处理的妙用。我也在程序设计题（1）中第一次打开了DEVC++的参数工具，使用它这么多年来，就像老朋友突然发现了新的特长，十分亲切。

最让我印象深刻的是程序设计题（2），我对题意始终不了解，在同学的点拨下，我终于有所领悟，也更加深入的理解了指针的机制与丰富作用。