# 6 指针程序设计实验

## 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实验环境

操作系统：Windows

编程语言：C

集成开发环境IDE：Visual Studio 2022

## 6.3 实验内容

**6.3.1 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. }

/\*将字符串 s 复制给字符串 t，并且返回串 t 的首地址\*/

1. char \* strcopy(char \*t, const char \*s)
2. {
3. while(\*t++ = \*s++);
4. return (t);
5. }

**解答：**

（1）错误修改：

1. 第5行的字符数组定义错误，正确形式为：

char s1[100] , s2[100] , \*s ;

1. strcopy函数编写错误，正确形式为：

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

{

char\* start = t;

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

return start;

}

修改完成后，发现程序可以正常运行

（2）修改后的源程序：

#include<stdio.h>

#include<string.h>

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

int main(void)

{

char s1[100], s2[100], \* s3;

printf("Input a string:\n");

scanf("%s", s2);

strcopy(s1, s2);

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

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

scanf("%s", s2);

s3 = strcopy(s1, s2);

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

return 0;

}

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

{

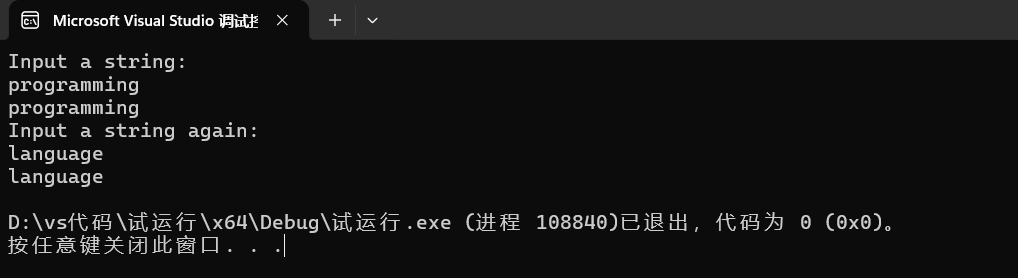
char\* start = t;

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

return start;

}

（3）错误修改后运行结果如图6-1所示：

图6-1 程序改错与跟踪调试题程序修改后运行结果截图

**6.3.2 源程序完善和修改替换**

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。

1. #include<stdio.h>
2. #include<\_\_\_\_\_\_\_\_\_>
3. #include<string.h>
4. #define N 4

/\*对指针数组s指向的size个字符串进行升序排序\*/

1. void strsort(char \*s[], int size)
2. {
3. \_\_\_\_\_\_\_temp;
4. int i, j;
5. for(i=0; i<size-1; i++)
6. for (j=0; j<size-i-1; j++)
7. if (\_\_\_\_\_\_\_\_\_\_\_)
8. {
9. temp = s[j];
10. \_\_\_\_\_\_\_\_\_\_\_;
11. s[j+1] = temp;
12. }
13. }
14. int main()
15. {
16. int i;
17. char \*s[N], t[50];
18. for (i=0; i<N; i++)
19. {
20. gets(t);
21. s[i] = (char \*)malloc(strlen(t)+1);
22. strcpy(\_\_\_\_\_\_\_);
23. }
24. strsort(\_\_\_\_\_\_\_\_);
25. for (i=0; i<N; i++) {puts(s[i]); free(s[i]);}
26. return 0;
27. }

2行填入：stdlib.h

7行填入：char\*

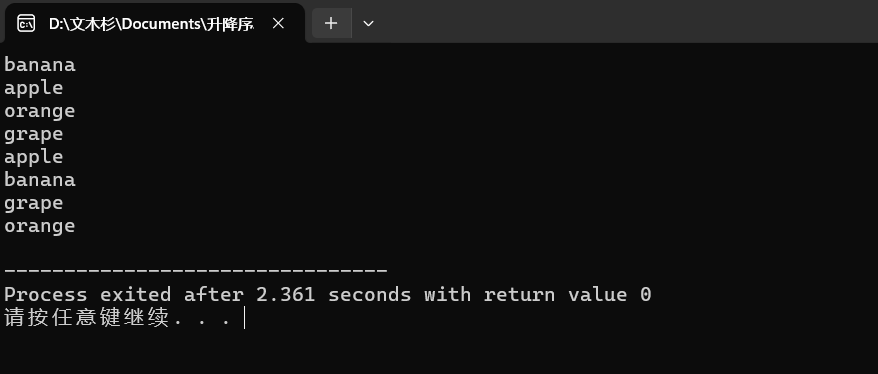
11行填入：strcmp(s[j], s[j + 1]) > 0

14行填入：s[j]=s[j+1]

27行填入：s[i],t

29行填入：s,N

补充完善后的源程序运行结果如图所示：

图6-2 源程序完善和修改替换题程序完善后运行结果截图

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。

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

{

char\* temp;

char\*\* p1, \*\* p2;

for (p1 = s; p1 < s + size - 1; p1++)

{

for (p2 = p1 + 1; p2 < s + size; p2++)

{

if (strcmp(\*p1, \*p2) > 0)

{

temp = \*p1;

\*p1 = \*p2;

\*p2 = temp;

}

}

}

}

优化完善后的源程序运行结果如图所示：

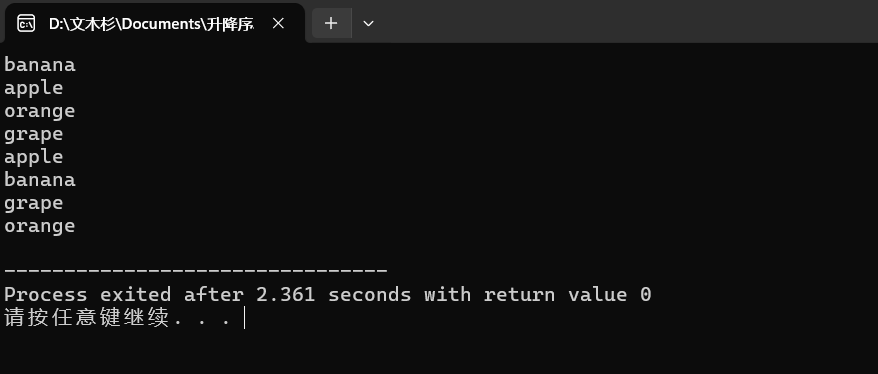


图6-3 源程序完善和修改替换题程序修改后运行结果截图

（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.

1. # include<stdio.h>
2. # include<string.h>
3. int main (void)
4. {
5. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
6. char a[80], b[80], \*result;
7. int choice;
8. while(1)
9. {
10. do
11. {
12. printf("\t\t1 copy string.\n");
13. printf("\t\t2 connect string.\n");
14. printf("\t\t3 parse string.\n");
15. printf("\t\t4 exit.\n");
16. printf("\t\tinput a number (1-4) please.\n");
17. scanf("%d", &choice);
18. }while(choice<1 || choice>4);
19. switch(choice)
20. {
21. case 1: p = strcpy; break;
22. case 2: p = strcat; break;
23. case 3: p = strtok; break;
24. case 4: goto down;
25. }
26. getchar();
27. printf("input the first string please!\n");
28. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
29. printf("input the second string please!\n");
30. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
31. result = \_\_\_\_\_\_\_\_\_\_\_(a, b);
32. printf("the result is %s\n", result);
33. }
34. down:
35. return 0;
36. }

5行填入：char\* (\*p)(char\*, const char\*) = NULL

28行填入：fgets(a, 100, stdin); if (a[strlen(a) - 1] == '\n')a[strlen(a) - 1] = '\0'

30行填入：fgets(b, 100, stdin); if (b[strlen(b) - 1] == '\n')b[strlen(b) - 1] = '\0'

31行填入：p(a, b)

补充完善后的源程序运行结果如图所示：

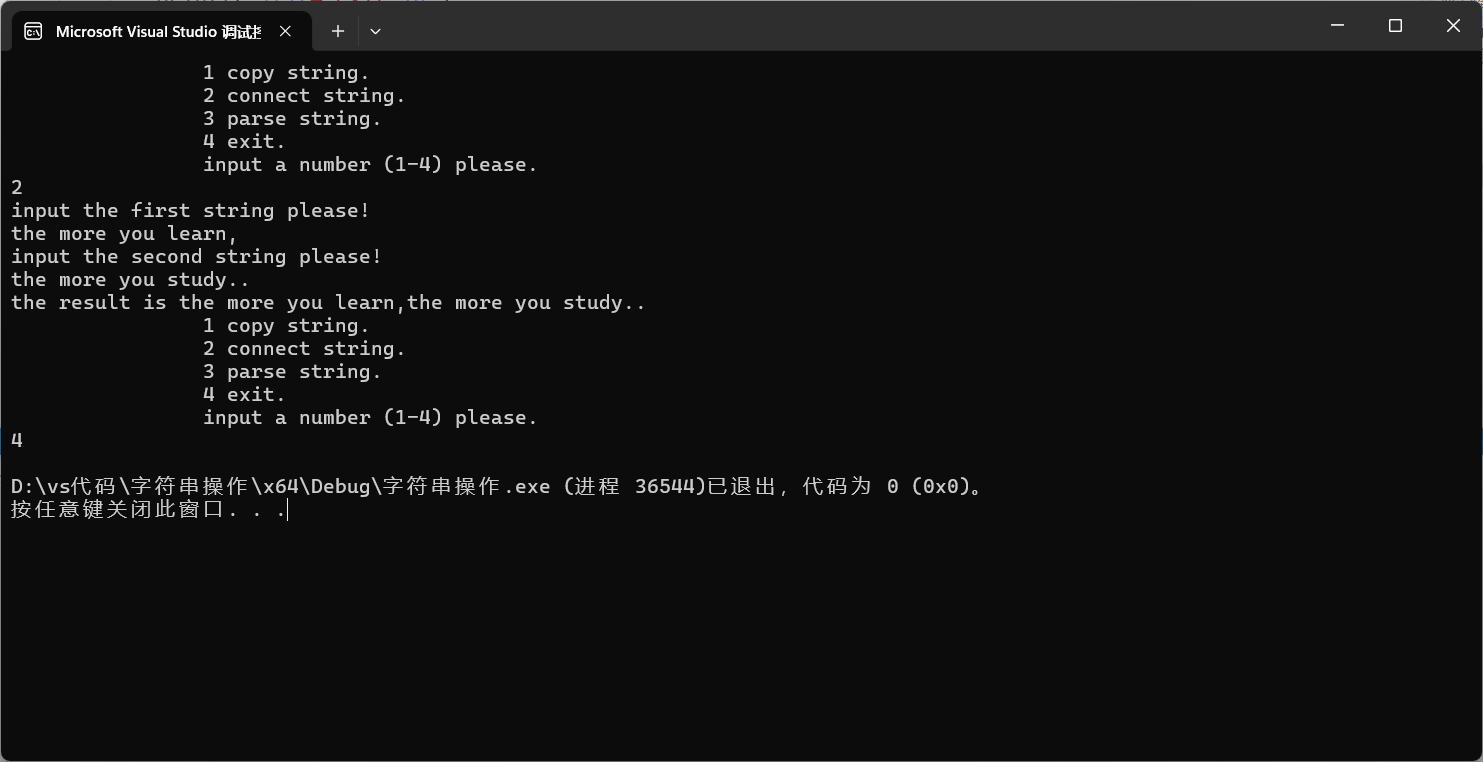


图6-4 源程序完善和修改替换题程序修改后运行结果截图

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。

# include<stdio.h>

# include<string.h>

int main(void)

{

char\*(\*p)(char\*,const char\*)=NULL;

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);

char\* (\*add[3])(char\*, const char\*) = { strcpy,strcat,strtok };

if (choice == 4)goto down;

p = add[choice - 1];

getchar();

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

fgets(a, 100, stdin); if (a[strlen(a) - 1] == '\n')a[strlen(a) - 1] = '\0';

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

fgets(b,100,stdin); if (a[strlen(a) - 1] == '\n')a[strlen(a) - 1] = '\0';

result = p(a, b);

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

}

down:

return 0;

}

优化完善后的源程序运行结果如图所示：

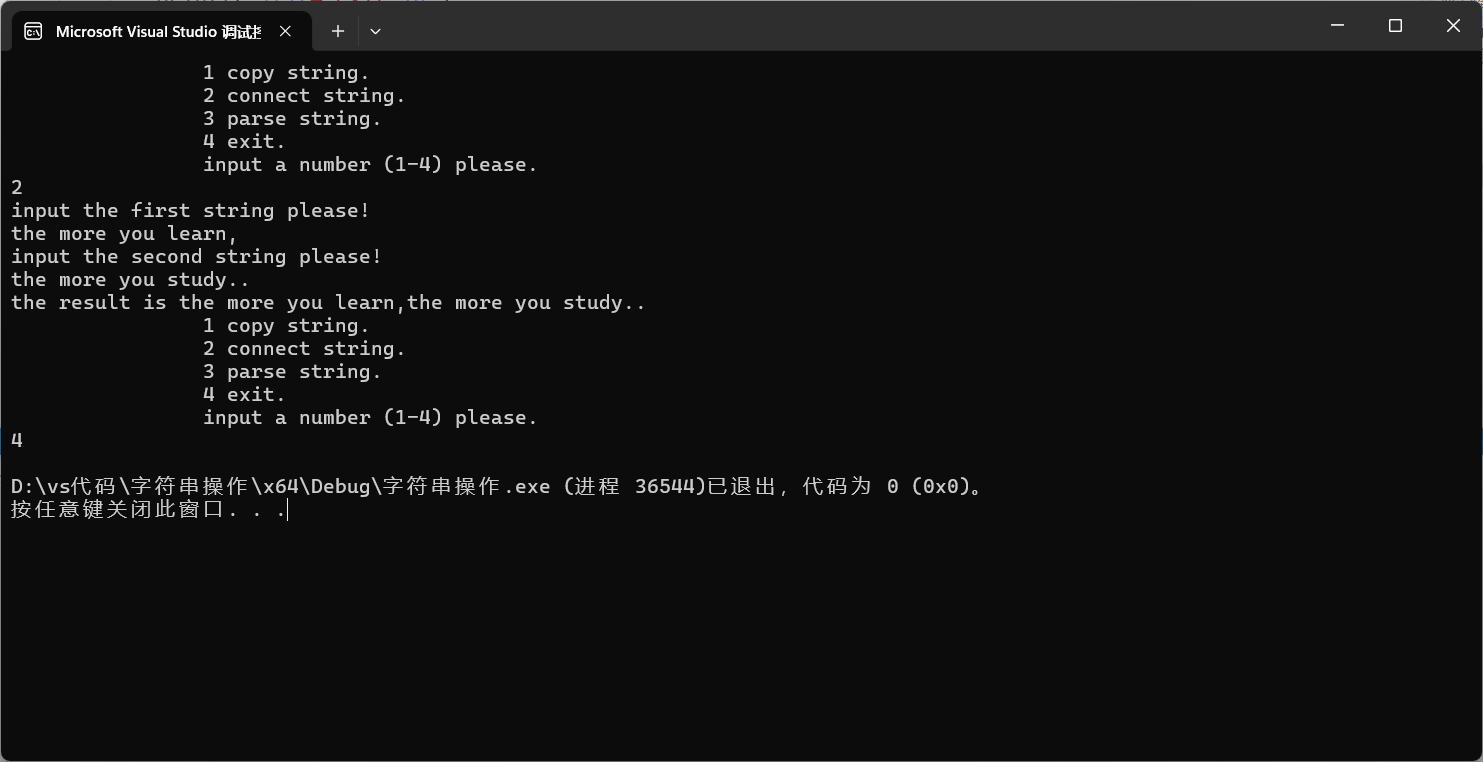


图6-5 源程序完善和修改替换题程序修改后运行结果截图

**6.3.3 源程序跟踪调试**

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。

（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？

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

#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);

}

**解答：**

1. 跟踪观察函数strcpy进入时watch窗口中s的值：

|  |  |  |  |
| --- | --- | --- | --- |
|  | **名称** | **值** | **类型** |
| ▶ | s | 0x000000444e2ff558 "烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫there is a boat on the lake." | char \* |

返回main时,watch窗口中s的值：

|  |  |  |  |
| --- | --- | --- | --- |
|  | **名称** | **值** | **类型** |
| ▶ | s | 0x000000314ceffc65 "烫烫烫烫烫烫烫烫烫蘴here is a boat on the lake." | char \* |

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

给出源程序清单如下：

#include "stdio.h"

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

int 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)

{

char\* temp = s;

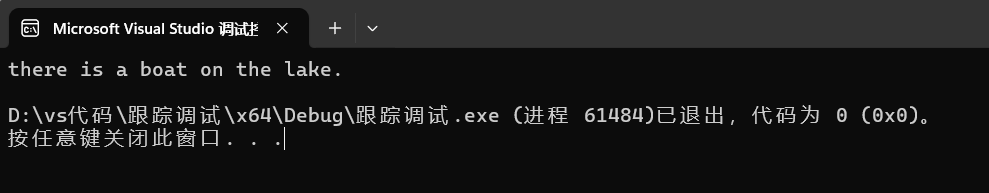
while (\*s++ = \*t++)

;

return temp;

}

1. 修改后的源程序运行结果如图所示：

图6-6 源程序跟踪调试题程序修改后运行结果截图

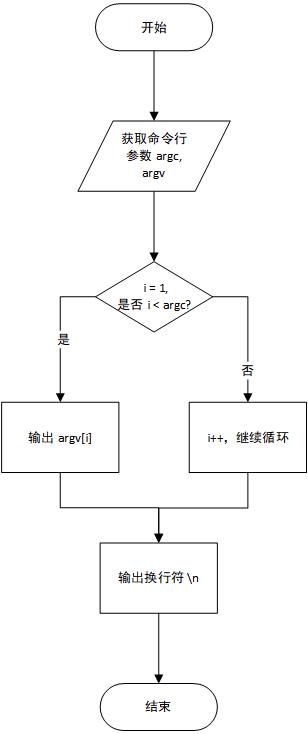
**6.3.4 程序设计**

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. **设计**。算法流程如图6-7所示。



6-7 程序设计题1的程序流程图

1. **实现。**源程序清单。

#include <stdio.h>

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

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

printf("%s ", argv[i]);

}

printf("\n");

return 0;

}

1. **测试**
2. 测试数据设计：如表6-1所示。

表6-1 程序设计题1的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序 输入 | 理 论 结 果 | 运 行 结 果 |
| x |  |  |
| 例1 | arg1 arg2 arg3 | arg1 arg2 arg3 | arg1 arg2 arg3 |

## 对应测试数据的运行结果如图5-5所示：

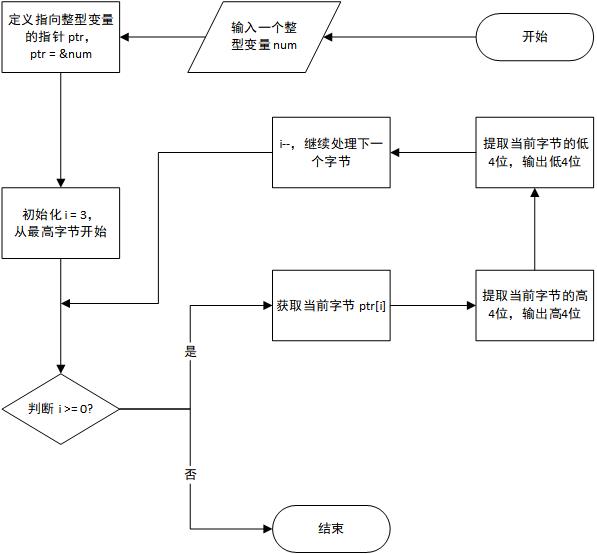


图6-8 设计程序题1的运行截图

1. 一个整型变量占4个字节，其中每个字节又分成高4位和低4位。输入一个整型变量，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，要求通过指针取出每字节。

**解答：**

1. **设计**。算法流程如图6-9所示。



6-9 程序设计题2的程序流程图

1. **实现。**源程序清单。

#include <stdio.h>

int main() {

// 输入整型变量

int num;

scanf("%d", &num); // 读取输入的整数

// 定义指向整数的指针

unsigned char \*ptr = (unsigned char \*)&num;

// 遍历每个字节，输出高4位和低4位

for (int i = 3; i >= 0; i--) {

unsigned char byte = ptr[i]; // 获取每个字节

// 提取高4位并输出

unsigned char high = (byte >> 4) & 0x0F;

// 提取低4位并输出

unsigned char low = byte & 0x0F;

// 输出高4位和低4位

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

}

printf("\n"); // 换行

return 0;

}

1. **测试**
2. 测试数据设计：

表6-2 编程题2的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| X |  |  |
| 用例1 | 2147483647 | 7FFFFFFF | 7FFFFFFF |
| 用例2 | 12 | 0000000C | 0000000C |
| 用例3 | -249 | FFFFFF07 | FFFFFF07 |

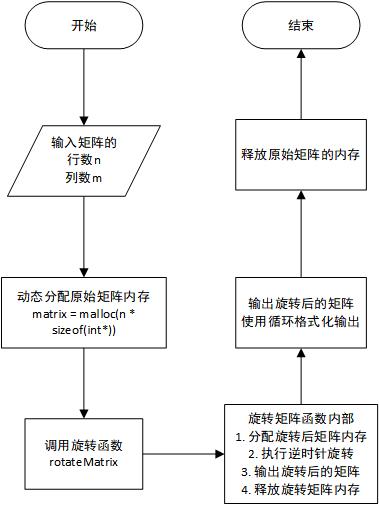
## 对应测试数据的运行结果如图6-10所示：284032bb521c237147acd7c4ba22dd68e4138f9be213347bbe7cc840c1f76d4e7f0d62e1aede751dd645e347768f6

图6-10 设计程序题2的运行截图

1. 旋转是图像处理的基本操作，编程实现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. **设计**。算法流程如图6-11所示。



6-11 程序设计题3的程序流程图

1. **实现。**源程序清单。

#include <stdio.h>

#include <stdlib.h>

// 旋转矩阵的函数

void rotateMatrix(int\*\* matrix, int n, int m) {

// 创建一个新的矩阵，用于存储旋转后的结果

int\*\* rotated = (int\*\*)malloc(m \* sizeof(int\*));

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

rotated[i] = (int\*)malloc(n \* sizeof(int));

}

// 执行逆时针旋转90度

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

for (int j = 0; j < m; j++) {

rotated[m - j - 1][i] = matrix[i][j];

}

}

// 输出旋转后的矩阵

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

for (int j = 0; j < n; j++) {

if (j != n - 1) {

printf("%d ", rotated[i][j]);

}

else

printf("%d", rotated[i][j]);

}

if (i != m - 1) {

printf("\n");

}

}

// 释放旋转后的矩阵内存

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

free(rotated[i]);

}

free(rotated);

}

int main() {

int n, m;

// 输入矩阵的行数n和列数m

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

// 动态分配内存存储矩阵

int\*\* matrix = (int\*\*)malloc(n \* sizeof(int\*));

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

matrix[i] = (int\*)malloc(m \* sizeof(int));

}

// 输入原始矩阵

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

for (int j = 0; j < m; j++) {

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

}

}

// 调用旋转函数

rotateMatrix(matrix, n, m);

// 释放原始矩阵的内存

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

free(matrix[i]);

}

free(matrix);

return 0;

}

1. **测试**
2. 测试数据设计：如表6-3所示。

表6-3 程序设计题3的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程序 输入 | 理 论 结 果 | 运 行 结 果 |
| x |  |  |
| 例1 | 2 3  4 7 8  5 3 2 | 8 2  7 3  4 5 | 8 2  7 3  4 5 |
| 例2 | 1 1  5 | 5 | 5 |
| 例3 | 4 3  1 1 2  1 3 4  2 4 3  2 6 5 | 2 4 3 5  1 3 4 6  1 1 2 2 | 2 4 3 5  1 3 4 6  1 1 2 2 |

1. 对应测试数据的运行结果如图6-12所示：

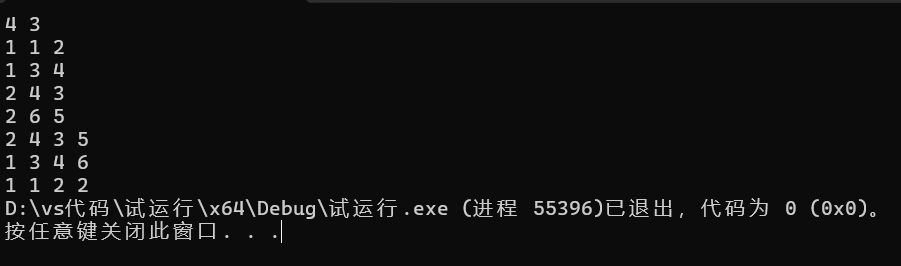
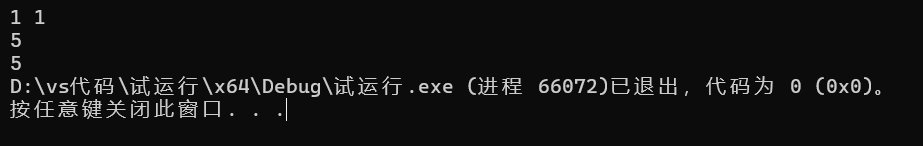
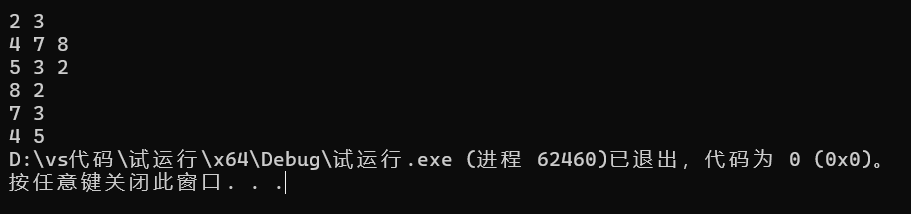
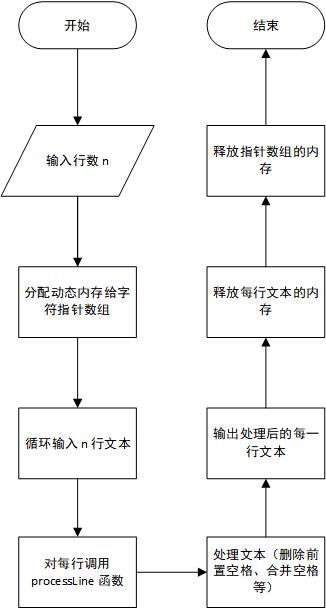


图6-12 设计程序题3的运行截图

1. 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，且n行文本的存储无冗余。然后执行如下操作：删除每一行中的前置空格（’ ‘）和水平制表符（’\t’），并将文本中多个空格合并为一个空格，如果有。**该操作要求单独定义成函数**。在main函数中输出执行了以上操作的各行。

**解答：**

1. **设计**。算法流程如图6-13所示。



6-13 程序设计题4的程序流程图

1. **实现。**源程序清单。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

// 函数声明：删除每行文本中的前置空格和水平制表符，合并多个空格

void processLine(char\* line);

int main() {

int n; // 输入的行数

scanf("%d", &n);

getchar(); // 吸收换行符

// 动态分配字符指针数组

char\*\* lines = (char\*\*)malloc(n \* sizeof(char\*));

if (lines == NULL) {

printf("内存分配失败!\n");

return 1;

}

// 输入n行文本

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

lines[i] = (char\*)malloc(81 \* sizeof(char)); // 为每行分配81个字符的空间（包括终止符'\0'）

if (lines[i] == NULL) {

printf("内存分配失败!\n");

return 1;

}

fgets(lines[i], 81, stdin);

// 去掉每行末尾的换行符（如果有）

lines[i][strcspn(lines[i], "\n")] = '\0';

}

// 对每行文本进行处理

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

processLine(lines[i]);

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

}

// 释放每行文本的内存

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

free(lines[i]);

}

// 释放指针数组的内存

free(lines);

return 0;

}

// 删除每行文本中的前置空格和制表符，合并多个空格

void processLine(char\* line) {

int i = 0, j = 0;

int length = strlen(line);

int inSpace = 0;

// 删除前置空格和水平制表符

while (line[i] == ' ' || line[i] == '\t') {

i++;

}

// 合并空格并删除多余的空格

for (; i < length; i++) {

// 如果当前字符不是空格和制表符

if (line[i] != ' ') {

line[j++] = line[i];

inSpace = 0;

}

// 如果是空格或制表符

else {

// 如果前一个字符不是空格（即不重复的空格）

if (!inSpace) {

line[j++] = ' ';

inSpace = 1;

}

}

}

// 添加字符串的终止符

line[j] = '\0';

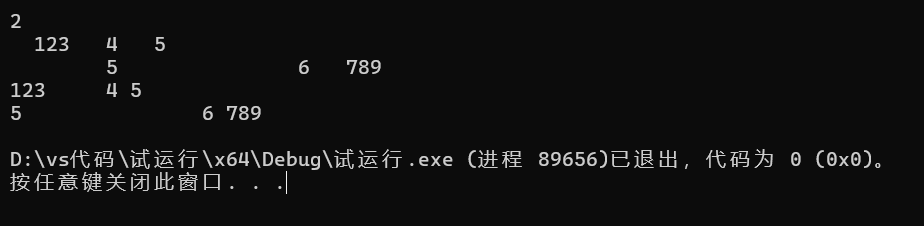
}

1. **测试**
2. 测试数据设计：如表6-4所示。

表6-4 程序设计题4的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 例1 | 2  123 4 5  5 6 789 | 123 4 5  5 6 789 | 123 4 5  5 6 789 |
| 例2 | 3  abc de f  abc def  abc d f | abc de f  abc def  abc d f | abc de f  abc def  abc d f |

1. 对应测试数据的运行结果如图6-14所示：



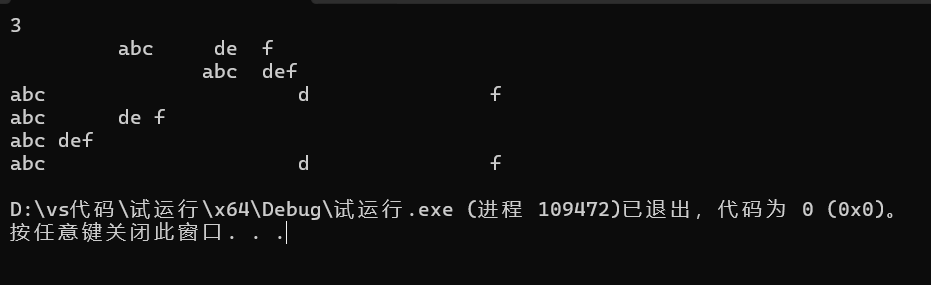


图6-14 设计程序题4的运行截图

1. 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。

**解答：**

1. **设计。**算法流程如图6-15所示。

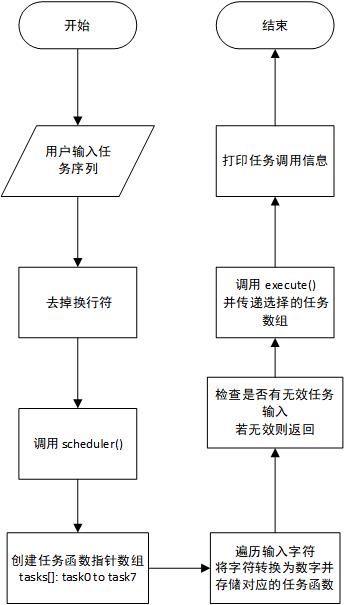
****

图6-15 程序设计题5的程序流程图

1. **实现。**源程序清单。

#include <stdio.h>

#include <stdlib.h>

// 任务函数声明

void task0(void);

void task1(void);

void task2(void);

void task3(void);

void task4(void);

void task5(void);

void task6(void);

void task7(void);

// scheduler函数声明

void scheduler(const char\* task\_order);

// execute函数声明

void execute(void (\*tasks[])(void), int task\_count);

// 任务函数实现

void task0(void) {

printf("task0 is called!\n");

}

void task1(void) {

printf("task1 is called!\n");

}

void task2(void) {

printf("task2 is called!\n");

}

void task3(void) {

printf("task3 is called!\n");

}

void task4(void) {

printf("task4 is called!\n");

}

void task5(void) {

printf("task5 is called!\n");

}

void task6(void) {

printf("task6 is called!\n");

}

void task7(void) {

printf("task7 is called!\n");

}

// scheduler函数实现

void scheduler(const char\* task\_order) {

// 函数指针数组

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

// 计算输入任务的数量

int task\_count = 0;

while (task\_order[task\_count] != '\0') {

task\_count++;

}

// 存储选择的任务函数指针

void (\*selected\_tasks[task\_count])(void);

// 根据输入的数字字符选择任务函数

for (int i = 0; i < task\_count; i++) {

int task\_index = task\_order[i] - '0'; // 将字符转换为数字

if (task\_index >= 0 && task\_index < 8) {

selected\_tasks[i] = tasks[task\_index];

}

else {

printf("Invalid task number: %c\n", task\_order[i]);

return;

}

}

// 调用execute函数执行选中的任务

execute(selected\_tasks, task\_count);

}

// execute函数实现

void execute(void (\*tasks[])(void), int task\_count) {

for (int i = 0; i < task\_count; i++) {

tasks[i](); // 执行任务

}

}

// main函数

int main() {

char task\_order[256];

// 提示用户输入任务序列

fgets(task\_order, sizeof(task\_order), stdin);

// 去除输入的换行符

for (int i = 0; task\_order[i] != '\0'; i++) {

if (task\_order[i] == '\n') {

task\_order[i] = '\0';

break;

}

}

// 调用scheduler函数进行调度

scheduler(task\_order);

return 0;

}

1. **测试**
2. 测试数据设计：如表6-5所示。

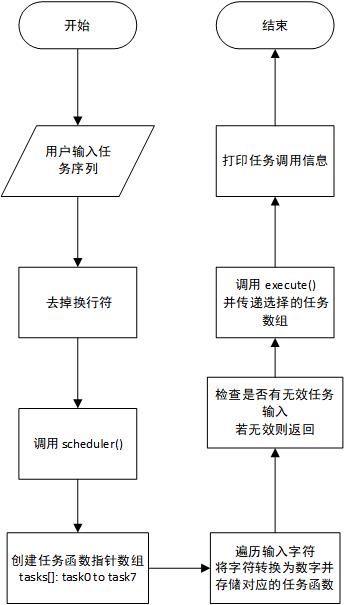
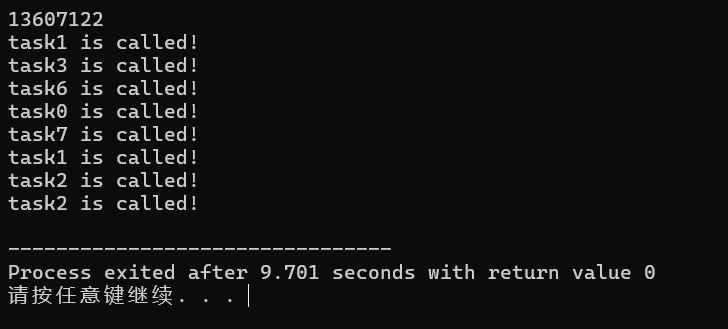
****

表6-5 程序设计题5的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| x |  |  |
| 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 | 6735643 | task6 is called!  task7 is called!  task3 is called!  task5 is called!  task6 is called!  task4 is called!  task3 is called! | task6 is called!  task7 is called!  task3 is called!  task5 is called!  task6 is called!  task4 is called!  task3 is called! |
| 例3 | 0 | task0 is called! | task0 is called! |

1. 对应测试数据的运行结果如图6-16所示：



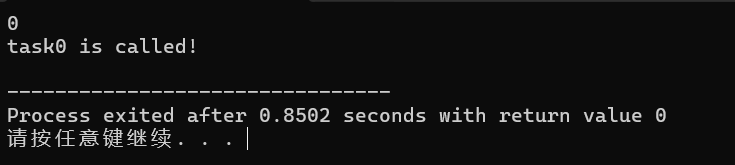
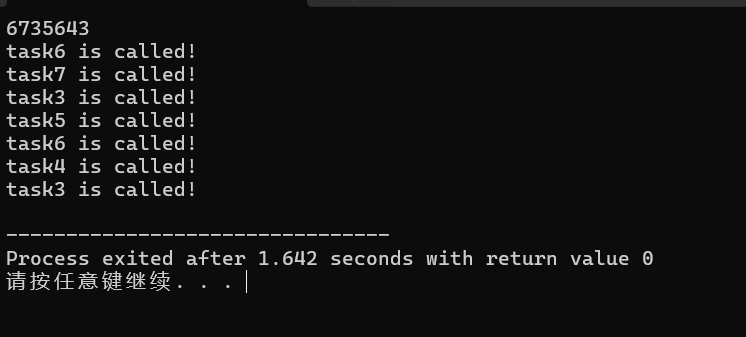
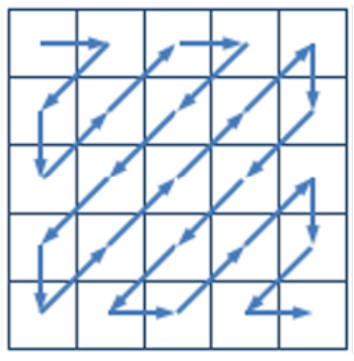


图6-16 程序设计题5的运行截图

1. 在图像编码算法中，需要将一个给定的方形矩阵进行Z字形扫描，如图6-1所示。给定一个*m*×*n*的矩阵，输出对这个矩阵进行Z字形扫描的序列。**要求使用指针操作矩阵中的元素**。



**解答：**

1. **设计。**算法流程如图6-17所示。

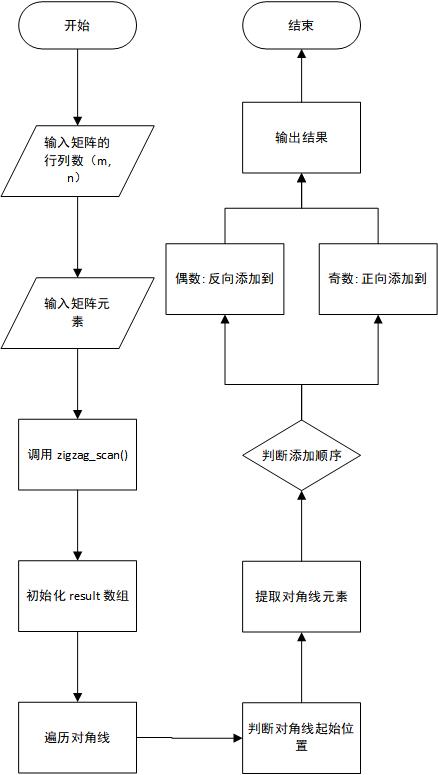


图6-17 程序设计题6的程序流程图

1. **实现。**源程序清单。

#include <stdio.h>

void zigzag\_scan(int m, int n, int matrix[m][n]) {

// 结果数组大小最大为 m \* n

int result[m \* n];

int idx = 0;

// 遍历所有对角线

for (int sum\_index = 0; sum\_index < m + n - 1; sum\_index++) {

int row\_start, col\_start;

// 计算当前对角线的起始位置

if (sum\_index < n) {

row\_start = 0;

col\_start = sum\_index;

} else {

row\_start = sum\_index - n + 1;

col\_start = n - 1;

}

// 存储当前对角线的元素

int diagonal\_elements[m + n]; // 对角线最多包含 m + n 个元素

int diagonal\_idx = 0;

int row = row\_start, col = col\_start;

// 获取当前对角线上的所有元素

while (row < m && col >= 0) {

diagonal\_elements[diagonal\_idx++] = matrix[row][col];

row++;

col--;

}

// 根据对角线编号决定是否反转

if (sum\_index % 2 == 0) {

// 偶数编号对角线，反向添加

for (int i = diagonal\_idx - 1; i >= 0; i--) {

result[idx++] = diagonal\_elements[i];

}

} else {

// 奇数编号对角线，正向添加

for (int i = 0; i < diagonal\_idx; i++) {

result[idx++] = diagonal\_elements[i];

}

}

}

// 输出结果

for (int i = 0; i < m \* n; i++) {

if (i > 0) printf(" ");

printf("%d", result[i]);

}

printf("\n");

}

int main() {

int m, n;

// 读取矩阵的行和列数

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

// 读取矩阵的元素

int matrix[m][n];

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

for (int j = 0; j < n; j++) {

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

}

}

// 获取 Z 字形扫描的结果

zigzag\_scan(m, n, matrix);

return 0;

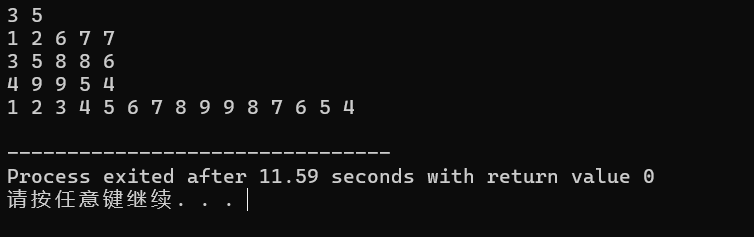
}

1. **测试**
2. 测试数据设计：如表6-6所示。

表6-6 程序设计题6的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| x |  |  |
| 例1 | 3 5  1 2 6 7 7  3 5 8 8 6  4 9 9 5 4 | 1 2 3 4 5 6 7 8 9 9 8 7 6 5 4 | 1 2 3 4 5 6 7 8 9 9 8 7 6 5 4 |
| 例2 | 1 1  9 | 9 | 9 |
| 例3 | 2 5  1 2 6 7 7  3 5 8 8 6 | 1 2 3 5 6 7 8 8 7 6 | 1 2 3 5 6 7 8 8 7 6 |

1. 对应测试数据的运行结果如图6-18所示：



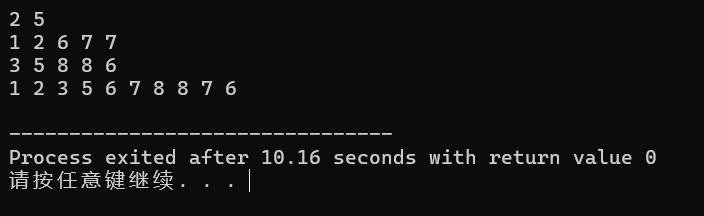
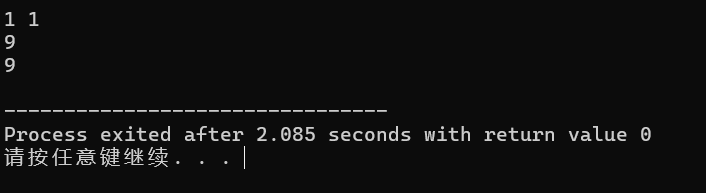


图6-18 程序设计题6的运行截图

1. 编写一个函数，从n个字符串中找出最长的串，如果有多个则取最先找到的那一个，n和表示n个字符串的指针数组由参数提供，指向最长串的指针通过函数返回，最长串的长度由参数带回。编写main函数测试该函数的正确性。

**解答：**

1. **设计。**算法流程如图6-19所示。

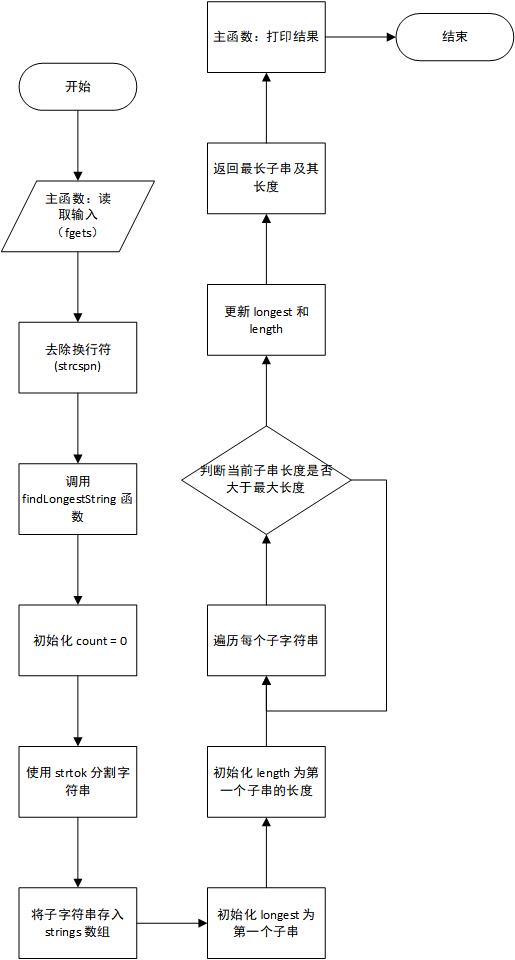
****

图6-19 程序设计题7的程序流程图

1. **实现。**源程序清单。

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char \*findLongestString(char \*line, int \*length) {

char \*token;

char \*strings[100]; // 假设最多 100 个字符串

int count = 0;

token = strtok(line, " ");

while (token!= NULL) {

strings[count] = token;

count++;

token = strtok(NULL, " ");

}

char \*longest = strings[0];

\*length = strlen(strings[0]);

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

int currentLength = strlen(strings[i]);

if (currentLength > \*length) {

\*length = currentLength;

longest = strings[i];

}

}

return longest;

}

int main() {

char line[1000]; // 假设输入行不超过 1000 个字符

fgets(line, sizeof(line), stdin);

line[strcspn(line, "\n")] = '\0'; // 去除 fgets 读入的换行符

int length;

char \*longest = findLongestString(line, &length);

printf("%s %d\n", longest, length);

return 0;

}

1. **测试**
2. 测试数据设计：如下所示。

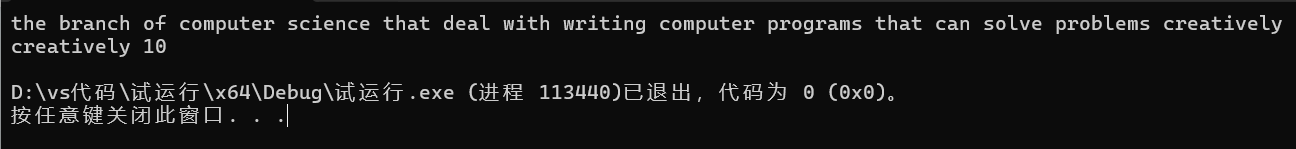
例1输入：the branch of computer science that deal with writing computer programs that can solve problems creatively

输出creatively 10

例2输入：Artificial intelligence is a type of computer technology which is concerned with making machines work in an intelligent way similar to the way that the human mind works The abbreviation AI is also used

输出：intelligence 12

1. 对应测试数据的运行结果如图5-13所示：



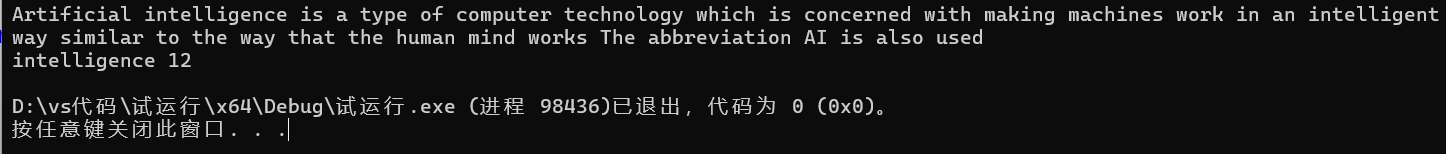


图6-20 程序设计题7的运行截图

参考文献

[1] 卢萍,李开,王多强等. C语言程序设计, 北京：清华大学出版社, 2021

[2] 卢萍,李开,王多强等. C语言程序设计典型题解与实验指导, 北京：清华大学出版社, 2019