# 实验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 源程序改错题

下面程序是否存在错误？如果存在，原因是什么？如果存在错误，要求在计算机上对这个例子程序进行调试修改，使之能够正确执行。

1 #include "stdio.h"

2 void main(void)

3 {

4 float \*p;

5 scanf("%f",p);

6 printf("%f\n",\*p);

7 }

解答:

1. 错误修改:
2. 第4行的float类型指针为空指针，无指向内容，应使其指向一个float型变量，正确形式为：

float a;

float \*p=&a;

错误修改后源程序:

#include "stdio.h"

void main(void)

{

float a;

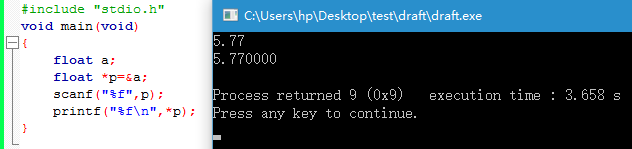
float \*p=&a;

scanf("%f",p);

printf("%f\n",\*p);

}

1. 错误修改后运行结果:



6.2.2 源程序修改替换题

（1）下面的程序通过函数指针和菜单选择来调用字符串拷贝函数或字符串连接函数，请在下划线处填写合适的表达式、语句、或代码片段来完善该程序。

#include "stdio.h"

#include "string.h"

void main(void)

{

char a[80],b[80],c[160],\*result=c;

int choice,i;

do{

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

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

printf("\t\t3 exit.\n");

printf("\t\tinput a number (1-3) please!\n");

scanf("%d",&choice);

}while(choice<1 || choice>5);

switch(choice){

case 1:

p=strcpy;

break;

case 2:

p=strcat;

break;

case 3:

goto down;

}

getchar();

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

i=0;

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

i=0;

result= (a,b);

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

down:

;

}

（2）为了使程序不受scanf、getchar、gets等函数输入后回车符的影响，请修改第（1）题程序，按要求输出下面结果：（（输入）表示该数据是键盘输入数据）

1 copy string.

2 connect string.

3 exit.

input a number (1-3) please!

2 （输入）

input the first string please!

the more you learn, （输入）

input the second string please!

the more you get. （输入）

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

解答：

（1） 第一个下划线处填写的语句为：

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

第二个下划线处填写的语句为：

scanf("%s",a);

第三个下划线处填写的语句为：

scanf("%s",b);

第四个下划线处填写的语句为：

result=p(a,b);

替换后的程序如下所示：

#include "stdio.h"

#include "string.h"

void main(void)

{

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

char a[80],b[80],c[160],\*result=c;

int choice,i;

do{

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

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

printf("\t\t3 exit.\n");

printf("\t\tinput a number (1-3) please!\n");

scanf("%d",&choice);

}while(choice<1 || choice>5);

switch(choice){

case 1:

p=strcpy;

break;

case 2:

p=strcat;

break;

case 3:

goto down;

}

getchar();

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

i=0;

scanf("%s",a);

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

i=0;

scanf("%s",b);

result=p(a,b);

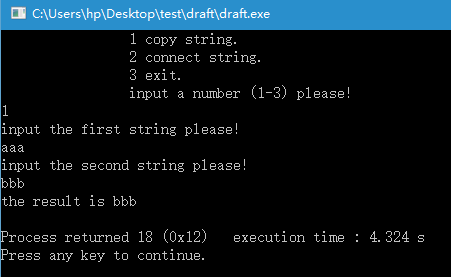
printf("the result is %s\n",result);

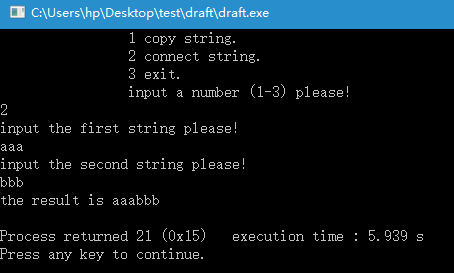
down:

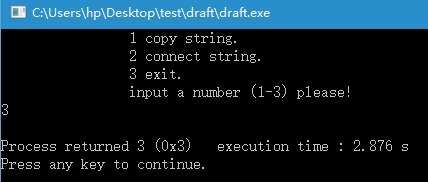
;

}

运行截图：







（2） 第一个下划线处填写的语句为：

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

第二个下划线处填写的语句为：

scanf("%[^\n]s",a);getchar();

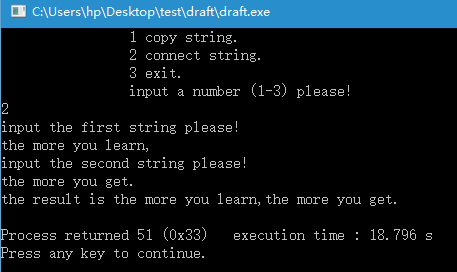
第三个下划线处填写的语句为：

scanf("%[^\n]s",b);getchar();

第四个下划线处填写的语句为：

result=p(a,b);

运行截图：



6.2.3 跟踪调试题

#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？

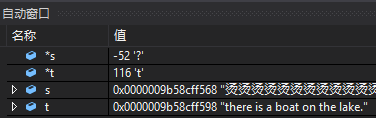
（2）排除错误，使程序输出结果为：

there is a boat on the lake.

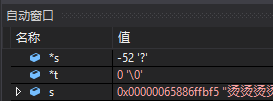
（3）选做：由于watch窗口中只显示s所指串的值，不显示s中存储的地址值，怎样才能观察到s值的变化呢？

解答：

（1）单步执行。进入strcpy时watch窗口中s为所指串值，如图：



返回main时, watch窗口中s为所指串值，如图：



（2）排除错误：通过新指针char \*p进行strcpy操作，避免s值的改变：

修改后的程序为：

#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)

{

char \*p=s;

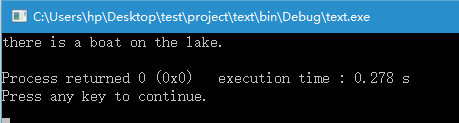
while ((\*p++ = \*t++))

;

return (s);

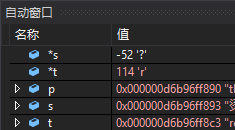
}

运行截图：



（3）选做：定义新指针char \*p后，用p代替s作为函数的返回值，从而能够观察到s值的变化，如图：

→



6.2.4 程序设计

（1）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。试从该长整型变量的高字节开始，依次取出每个字节的高4位和低4位并以数字字符的形式进行显示。

解答：

1. 解题思路：
2. 提示并输入长整形变量个数n，随后输入n个长整形变量并存储于数组中；
3. 依次对数组中每个长整形变量用自定义函数itoh进行处理，具体流程转2.1；
   1. 构建逻辑尺mask=0xF0000000；
   2. 通过for循环与逻辑尺mask，依次取出长整形变量中从高到低每个字节的高四位和低四位，即for(i=0;i<8;i++) temp=( ( (mask>> (i\*4) ) & \*(a+j) ) >> (28-i\*4) )& 0xF ；
   3. 对取出的每四位依次进行处理，使用指针将对应的十六进制字符存入二维字符数组s[N][N]，并在每个字符串后面添加’\0’以结束字符串；
   4. 转2.1处理下一个长整形变量，处理完所有长整形变量后转3。
4. 依次输出对每个长整形变量处理后的结果；
5. 结束。
6. 算法流程如图6-1所示：

提示并输入长整形变量个数n

输入第i个长整形变量

i<=n?

定义逻辑尺mask=0XF0000000

j<=8?

取出整形变量的第j位，

并将其转换为十六进制字符

存入字符数组

j++,

mask=mask>>4\*j

Y

i++

N

Y

依次输出字符数组中的字符

N

图6-1 编程题6.2.4.(1)的程序流程图

1. 程序清单：

#include<stdio.h>

#define N 100

#define length 32

char s[N][N];

void itoh(long a[],int n);

int main (void)

{

long a[N];

int n,i,j;

scanf("%d",&n);

//input

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

scanf("%ld",a+i);

//main

itoh(a,n);

//output

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

for(j=0;j<15;j++)

printf("%c%c",\*(\*(s+i)+j),j==14?'\n':' ');

}

void itoh(long a[],int n)

{

int i,j,temp=0;

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

{

long mask=0XF0000000;

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

{

temp=( ( (mask>> (i\*4) ) & \*(a+j) ) >> (28-i\*4) )& 0xF ;

if(temp>=0&&temp<10)

\*(\*(s+j)+i)='0'+temp;

else if(temp>9)

\*(\*(s+j)+i)='A'+temp-10;

}

\*(\*(s+j)+i)='\0';

}

}

1. 测试
2. 测试数据：

为使测试结果具有代表性，选择长整形变量分别为1、0、-23、88888等特殊数据，另外再选取两个一般的数据10和90。如表6-1所示。

表6-1 编程题6.2.4.(1)的测试数据

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| 长 整 形 变 量 (个数n=6) |
| 用例1 | 1 | 处理结果 0 0 0 0 0 0 0 1 |
| 用例2 | 0 | 处理结果 0 0 0 0 0 0 0 0 |
| 用例3 | 88888 | 处理结果 0 0 0 1 5 B 3 8 |
| 用例4 | 90 | 处理结果 0 0 0 0 0 0 5 A |
| 用例5 | -23 | 处理结果 F F F F F F E 9 |
| 用例6 | 10 | 处理结果 0 0 0 0 0 0 0 A |

1. 对应测试测试用例的运行结果如图6-2所示。

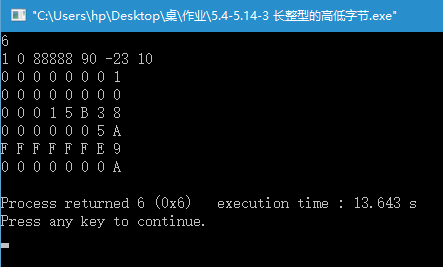


图6-2 编程题6.2.4.(1)的测试用例的运行结果

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。

（2）利用大小为n的指针数组指向用gets函数输入的n行，每行不超过80个字符。编写一个函数，它将每一行中连续的多个空格字符压缩为一个空格字符。在调用函数中输出压缩空格后的各行，空行不予输出。

解答：

1. 解题思路：
2. 输入一组数据，输入的第一个数字i代表将要处理的数据的行数，随后对字符数组进行初始化并通过scanf("%[^\n]", s[i])输入i行字符；
3. 通过自定义函数zip对每行字符进行处理，具体流程转2.1；
   1. 使指针数组中的一个指针p[t]指向存储了一行字符的字符串s[t]；
   2. 若指针指向的字符不为’\0’，则对字符串进行压缩空格的处理；
   3. 若指针指向的字符为空格’ ‘，则判断其上一个字符是否为空格，是则跳过，不是则将其存储入新的字符串；
   4. 若指针指向的字符不为空格’ ‘，则将其存储入新的字符串；
   5. 若指针指向的字符为’\0’，处理结束，并在新的字符串后面添加’\0’；
   6. 转2.1处理下一行字符，处理完所有行后转3；
4. 通过for循环将处理后生成的字符串输出，若遇到空行即只含’\0’的字符串，则不输出，即：

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

if (s[i][0] != '\0') printf("%s%c", s[i], '\n')；

1. 转1输入下一组数据，直至输入的数字为0，结束。
2. 程序清单：

#include<stdio.h>

#define N 100

void zip(char s[N][N]);

int n;

char \*p[N];

int main(void)

{

char s[N][N];

int i;

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

{

if (n == 0) return 0;

else

{

//input

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

{

getchar();

s[i][0]='\0';

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

}

//main

zip(s);

//output

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

if (s[i][0] != '\0') printf("%s%c", s[i], '\n');

printf("\n");

}

}

}

void zip(char s[N][N])

{

int i, j, t;

for (t = 0;t<n;t++)

{

i=0;j=0;

p[t] = s[t];

while (\*(p[t] + j) != '\0')

{

if (\*(p[t] + j) == ' ')

{

if (\*(p[t] + j - 1) == ' ')

j++;

else

\*(p[t] + i++) = \*(p[t] + j++);

}

else

\*(p[t] + i++) = \*(p[t] + j++);

}

\*(p[t]+i)='\0';

}

}

1. 测试
2. 测试数据

为便于分析结果，选择N分别为134744072、16843009等特殊数字，另外输入一组较一般的数据16885952。如表6-2和表6-3所示。

表6-2 编程题6.2.4.(2)的测试数据一

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| 一行字符（N=3） |
| 用例1 | zhang chuan chao | Zhang chuan chao |
| 用例2 | (空行) | (不输出) |
| 用例3 | xi da da | xi da da |

表6-3 编程题6.2.4.(2)的测试数据二

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| 一行字符（N=4） |
| 第一行 | If only we’d stop trying to be happy | If only we’d stop trying to be happy |
| 第二行 | we’d have a pretty good time. | we’d have a pretty good time. |
| 第三行 | （空行） | （不输出） |
| 第四行 | （空行） | （不输出） |

1. 对应测试测试数据1、2的运行结果如图6-3所示。

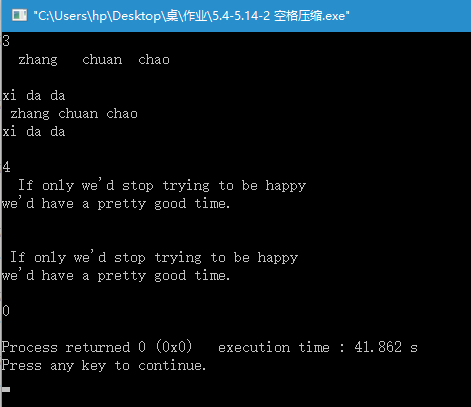


图6-3 编程题6.2.4.(2)的测试数据一、二的运行结果

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）编写一个程序，输入n个整数，排序后输出。排序的原则由命令行可选参数-d决定，并且有参数-d时按递减顺序排序，否则按递增顺序排序。要求将排序算法定义成函数，利用指向函数的指针使该函数实现递增或递减排序。

解答：

1. 解题思路：
2. 利用cmd打开.exe文件，并在打开文件时输入其他参数；
3. main函数返回的参数个数为1时，输入五个数据并增序排序。参数个数为2且另一个参数为-d时，输入五个数据并降序排序。其余情况输出Error；
4. 通过指针指向的函数使数据排序并输出；
5. 结束。
6. 程序清单：

#include<stdio.h>

#define n 5

void sort(int stuff[],int x,int flag);

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

{

int flag,n,i;

int \*p(int,int,int)=sort;

if(argc==1) flag=1;

else if(argc==2)

{

if(\*argv[1]!='-'||\*(argv[1]+1)!='d')

{

printf("Error!\n");

}

flag=0;

}

else if(argc>2)

{

printf("Error!\n");

return -1;

}

int stuff[n]={0};

//input

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

{

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

}

//main

p(stuff,n,flag);

//output

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

{

printf("%d\t",stuff[i]);

}

printf("\n");

return 0;

}

void sort(int stuff[],int x,int flag)

{

int i,j;

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

{

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

{

if(flag?(stuff[j]<stuff[j+1]):(stuff[j]>stuff[j+1]))

{

int x=0;

x=stuff[j];

stuff[j]=stuff[j+1];

stuff[j+1]=x;

}

}

}

}

1. 测试
2. 测试数据

为便于分析结果，用cmd打开文件时分别:不使用参数-d、使用参数-d、使用两个以上的参数、使用两个参数但不是-d。用以验证程序正确性

1. 对应测试测试用例1的运行结果如图6-4所示。



图6-4 编程题6.2.4.(3)的测试用例的运行结果

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）设某个班有N个学生，每个学生修了M门课程（用#define定义N、M）。输入M门课程的名称，然后依次输入N个学生中每个学生所修的M门课程的成绩并且都存放到相应的数组中。编写下列函数：

a.计算每个学生各门课程平均成绩；

b.计算全班每门课程的平均成绩；

c.分别统计低于全班各门课程平均成绩的人数；

d.分别统计全班各门课程不及格的人数和90分以上（含90分）的人数。

解答：

1. 解题思路：
2. 提示并输入5个科目的名称、5个学生的姓名以及他们五个科目的成绩；
3. 通过自定义函数ave\_stu、ave\_sub、low\_than\_ave、fail、perfect分别实现计算每个学生各门课程的平均成绩、计算全班每门课程的平均成绩、统计低于全班各门课程平均成绩的人数、各门课程不及格的人数和90分以上的人数。具体操作流程转2.1；
   1. 利用自定义函数ave\_stu计算每个学生各门课程的平均成绩：定义变量第i个学生、第j门课程，通过for循环，用指针遍历学生的成绩，并将学生所有课程成绩相加后处理得到平均成绩，将结果存入数组p1。即：

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

{

sum=0;

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

sum += \*(\*(a+i)+j);

\*(p1+i)=sum/M;

}

* 1. 利用自定义函数ave\_sub计算每门课程的平均成绩：操作与2.1类似，相加时将同一课程不同学生的成绩相加，计算结果存入数组p2。即：

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

{

sum=0;

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

sum += \*(\*(a+j)+i);

\*(p2+i)=sum/N;

}

* 1. 利用自定义函数low\_than\_ave统计每门课程中低于平均成绩的同学的个数：对每门成绩初始化count=0，便利该课程所有同学的成绩，若同学成绩低于平均成绩，count++，最后将count存入数组p3。即：

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

{

count=0;

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

if(\*(\*(a+j)+i)<\*(p2+i)) count++;

\*(p3+i)=count;

}

* 1. 利用自定义函数count\_fail\_and\_perfect统计每门课程中不及格人数与高于90分人数：操作与2.3类似，同学成绩不及格时count1++，高于90时count2++，最后将count1存入数组p4，count2存入数组p5。即：

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

{

count1=0;

count2=0;

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

{

if(\*(\*(a+j)+i)<60.00) count1++;

if(\*(\*(a+j)+i)>=90.00) count2++;

}

\*(p4+i)=count1;

\*(p5+i)=count2;

}

1. 依次输出处理数据后得到的结果。
2. 结束。
3. 程序清单：

#include<stdio.h>

#define M 5

#define N 5

void ave\_stu (double a[N][M]);

void ave\_sub (double a[N][M]);

void low\_than\_ave (double a[N][M]);//each sub

void count\_fail\_and\_perfect (double a[N][M]);

double stu\_ave[N] ,sub\_ave[M] ; int stu\_lower[M] ,stu\_fail[M] ,stu\_perfect[M];

double \*p1=stu\_ave ,\*p2=sub\_ave ; int \*p3=stu\_lower ,\*p4=stu\_fail ,\*p5=stu\_perfect;

int main(void)

{

char sub\_name[M][100],stu\_name[N][100];

double stu\_grade[N][M];

int i,j;

//input subject

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

scanf("%s",sub\_name[i]);

//input stu\_name and grades

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

{

scanf("%s",stu\_name[i]);

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

scanf("%lf",&stu\_grade[i][j]);

}

//main

ave\_stu (stu\_grade);

ave\_sub (stu\_grade);

low\_than\_ave (stu\_grade);

count\_fail\_and\_perfect (stu\_grade);

//output

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

printf("Average score of %s is %.2f\n",stu\_name[i],\*(p1+i));

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

printf("Average score of %s is %.2f\n",sub\_name[i],\*(p2+i));

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

printf("Number of students lower than avg of %s is %d\n",sub\_name[i],\*(p3+i));

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

printf("Number of students %s fail is %d\n",sub\_name[i],\*(p4+i));

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

printf("Number of students %s perfect is %d\n",sub\_name[i],\*(p5+i));

}

void ave\_stu (double a[N][M])//\*a=stu\_grade

{

int i,j;

double sum;

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

{

sum=0;

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

sum += \*(\*(a+i)+j);

\*(p1+i)=sum/M;

}

}

void ave\_sub (double a[N][M])//\*a=stu\_grade

{

int i,j;

double sum;

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

{

sum=0;

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

sum += \*(\*(a+j)+i);

\*(p2+i)=sum/N;

}

}

void low\_than\_ave (double a[N][M])//each sub

{

int count,i,j;

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

{

count=0;

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

if(\*(\*(a+j)+i)<\*(p2+i)) count++;

\*(p3+i)=count;

}

}

void count\_fail\_and\_perfect (double a[N][M])

{

int count1,count2,i,j;

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

{

count1=0;

count2=0;

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

{

if(\*(\*(a+j)+i)<60.00) count1++;

if(\*(\*(a+j)+i)>=90.00) count2++;

}

\*(p4+i)=count1;

\*(p5+i)=count2;

}

}

1. 测试
2. 测试数据

输入课程名分别为A、B、C、D、E。输入同学的姓名与分数如表6-4所示。

表6-4 编程题6.2.4.(4)的测试数据

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 测 试 结 果 |
| N |
| 用例1 | Zhang 87 99 77 87 95 | 如图6-4 |
| 用例2 | Li 45 98 100 48 75 |
| 用例3 | Wang 85 68 95 47 59 |
| 用例4 | Han 50 89 100 85 20 |
| 用例5 | Gan 87 68 87 89 100 |

1. 对应测试测试用例的运行结果如图6-5所示。

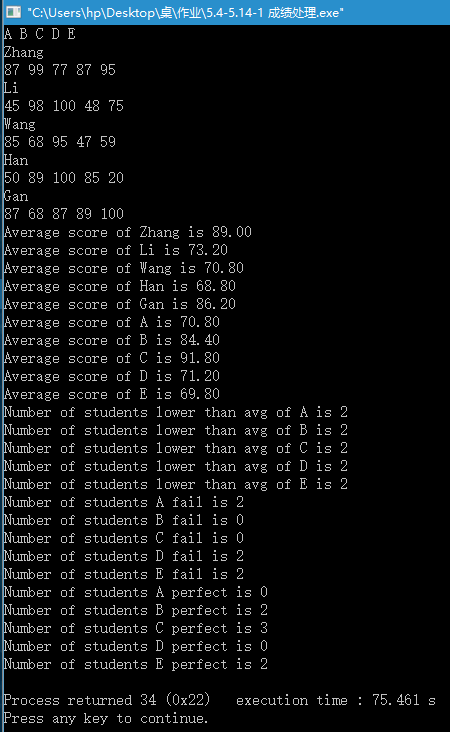


图6-5 编程题6.2.4.(4)的测试用例的运行结果

经检验上述的运行结果与理论分析吻合，验证了程序的正确性。

6.2.5 选做题

（1）设有N位整数和M位小数（N=20，M=10）的数据a,b。编程计算a+b并输出结果。

如：12345678912345678912.1234567891 + 98765432109876543210.0123456789

解答：

1. 解题思路：
2. 输入两个大数字；
3. 将数字作为字符串进行存储，随后对字符串进行处理，具体流程转2.1；
   1. 从最后一位数字开始进行加法，通过for循环实现进位；
   2. 遇到小数点时进行如下操作

if(k==N+1)

{

c[k]='.';

k--;

c[k] = (a[i-1] - '0' + b[j-1] - '0' + flag) % 10 + '0';

i--;

j--;

}

从而跳过小数点对下一位继续操作

* 1. 生成新字符串后，若进位flag为1，将1赋值给c[0]，否则将新字符串左移一位以实现正常输出；

1. 输出结果；
2. 结束。
3. 程序清单：

#include <stdio.h>

#include <string.h>

#define N 20//xxx.

#define M 10//.xxx

void main( )

{

char a[1000], b[1000], c[1001] = {0}, flag = 0;

int i, j, k;

scanf("%s %s", a, b);

for(i = M+N, j = M+N, k = M+N+1 ; i >= 0 && j >= 0; i--, j--, k--)

{

// '.'

if(k==N+1)

{

c[k]='.';

k--;

c[k] = (a[i-1] - '0' + b[j-1] - '0' + flag) % 10 + '0';

i--;

j--;

}

//main

else

{

c[k] = (a[i] - '0' + b[j] - '0' + flag) % 10 + '0';

flag = (a[i] - '0' + b[j] - '0' + flag) / 10;

}

}

if(flag != 0)

{

c[0] = flag + '0';

}

else if(flag == 0)

{

for(i=0;c[i+1]!='\0';i++)

c[i]=c[i+1];

c[i]='\0';

}

printf("%s + %s = %s\n", a, b, c);

}

1. 测试
2. 测试数据：

使用题目中给出的例子。

1. 对应测试测试用例的运行结果如图6-6所示。

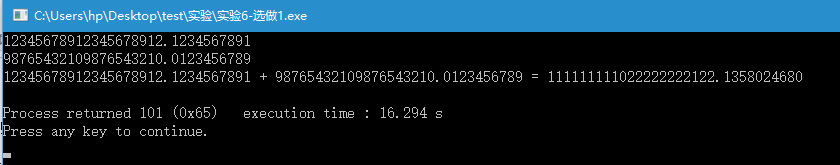


图6-6 选做题6.2.5.(1)的测试用例的运行结果

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。

（2）编写使用复杂声明char \*(\*p[2])(const char \*,const char \*);的程序。

提示：p中元素可为strcmp、strstr等函数名。

解答:

1. 解题思路：
2. 用指针函数数组指向strcmp函数与strstr函数；
3. 初始化两个数组并用指针函数对其进行处理，字符串a>b返回正数，字符串b是a的子串返回地址；
4. 输出结果；
5. 结束。
6. 程序清单：

#include<stdio.h>

#include<string.h>

int main (void)

{

char\*(\*p[2])(const char\*,const char\*);

p[0]=strcmp;

p[1]=strstr;

char a[]="abcd";

char b[]="abc";

printf("%d\n",p[0](a, b));

printf("%d\n",p[1](a, b));

return 0;

}

1. 测试
2. 测试数据：

如图6-6

1. 对应测试测试用例的运行结果如图6-7所示。

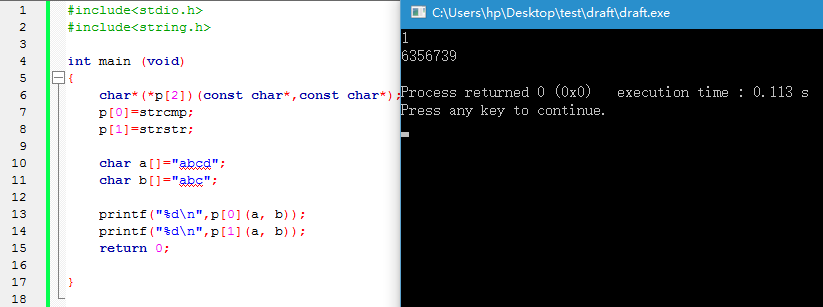


图6-7 选做题6.2.5.(2)的运行结果

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 自设题

**（1）**自设实验题目：定义一个函数strrindex(s,t)，返回t在s中最右边出现的位置（s中的第1个字符的位置为0），找不到返回-1（用指针实现）。

**（2）**实验目的：通过设计实验程序，熟练使用指针，并熟悉掌握指针与数组下标之间的关系。

**（3）**实验程序：

#include<stdio.h>

#include<string.h>

int strrindex(char s[],char t[])

{

int a,b,r=-1;

a=strlen(s);

b=strlen(t);

for(;a>=0;a--)

{

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

{

for(;\*(s+a)==\*(t+b)&&(b>=0)&&(a>=0);a--,b--)

if(b==0)

{

r=a;

return r;

}

}

}

return -1;

}

#define N 80

int main()

{

char s[N+1],t[N+1];

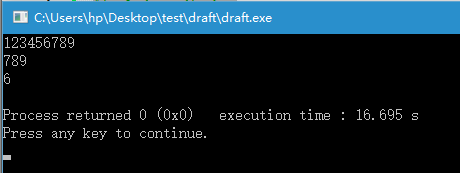
scanf("%s%s",s,t);

printf("%d\n",strrindex(s,t));

return 0;

}

**（4）**实验用例：如图



**（5）**实验结论：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。

## 6.4 实验小结

在实验中主要遇到的问题有：

1. 在6.2.2源程序修改替换中，使用指针函数填空较为困难，暴露了自己当时对指针的掌握还不够充分，后在查阅课本时找到解决方案。
2. 在6.2.3跟踪调试题中开始没有注意到指针的偏移，尝试时输出了正确结果却不知道为什么，后询问同学得到解答。
3. 在解决编程题6.2.4（4）成绩处理时，因为数据较多逻辑稍复杂，加上对指针的理解还不够透彻，多次在语法和逻辑上遇到问题，多次修改后通过作业系统。

体会：

1. 指针是C语言中最重要且富有难度的一个部分，在完成作业及实验要求的时候发现单单掌握其简单的运用是不够的，多重指针、函数指针、指针数组等都要善于使用。在对数组的引用中尝试使用指针进行替代，会有助于自己对指针和数组的理解；
2. 使用指针时应时刻注意指针指向地址的变化，必要时通过跟踪来进行判断，以免出现问题或错误；
3. C语言不仅可以在编译器中运行，也可以通过cmd来进行访问，同时main函数中的参数有时也可以用来实现开关的作用。善于利用main函数的参数能使C语言更加灵活。