

**课 程 实 验 报 告**

**课程名称： C语言程序设计**

**专业班级： CS1705**

**学 号： U201714635**

**姓 名： 伍瀚缘**

**指导教师： 卢萍**

**报告日期： 2017/11/22**

**计算机科学与技术学院**

**目 录**

[实验1 表达式和标准输入与输出实验 1](#_Toc500444764)

[1.1 实验目的 1](#_Toc500444765)

[1.2 实验内容 1](#_Toc500444766)

[1.3 自设题 11](#_Toc500444767)

[1.4 实验小结 12](#_Toc500444768)

[实验2 流程控制实验 14](#_Toc500444769)

[2.1 实验目的 14](#_Toc500444770)

[2.2 实验内容 14](#_Toc500444771)

[2.3 自设题 32](#_Toc500444772)

[2.4 实验小结 32](#_Toc500444773)

[实验3 函数与程序结构实验 33](#_Toc500444774)

[3.1 实验目的 33](#_Toc500444775)

[3.2 实验内容 33](#_Toc500444776)

[3.3 自设题 45](#_Toc500444777)

[3.4 实验小结 46](#_Toc500444778)

[实验4 函数与程序结构实验 47](#_Toc500444779)

[4.1 实验目的 47](#_Toc500444780)

[4.2 实验内容 47](#_Toc500444781)

[4.4 实验小结 59](#_Toc500444782)

[附录1 第一章自设题部分内容 60](#_Toc500444783)

[1.3.1 reading.c 60](#_Toc500444784)

[1.3.2 output.c 63](#_Toc500444785)

[附录2 第二章自设题部分内容 69](#_Toc500444786)

[2.3 output.c 69](#_Toc500444787)

实验1 表达式和标准输入与输出实验

1.1 实验目的

（1）熟练掌握各种运算符的运算功能，操作数的类型，运算结果的类型及运算过程中的类型转换，重点是C语言特有的运算符，例如位运算符，问号运算符，逗号运算符等；熟记运算符的优先级和结合性。

（2）掌握getchar, putchar, scanf 和printf 函数的用法。

（3）掌握简单C程序（顺序结构程序）的编写方法。

1.2 实验内容

1.2.1 源程序改错

下面给出了一个简单C语言程序例程，用来完成以下工作：

（1）输入华氏温度f，将它转换成摄氏温度C后输出；

（2）输入圆的半径值ｒ，计算并输出圆的面积ｓ；

（3）输入短整数ｋ、ｐ，将ｋ的高字节作为结果的低字节，ｐ的高字节作为结果的高字节，拼成一个新的整数，然后输出；

在这个例子程序中存在若干语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。

#include<stdio.h>

#define PI 3.14159;

intmain( void )

{

int f ;

short p, k ;

double c , r , s ;

/\* for task 1 \*/

printf(“Input Fahrenheit:” ) ;

scanf(“%d”, f ) ;

c = 5/9\*(f-32) ;

printf( “ \n %d (F) = %.2f (C)\n\n ”, f, c ) ;

/\* for task 2 \*/

printf("input the radius r:");

scanf("%f", &r);

s = PI \* r \* r;

printf(“\nThe acreage is %.2f\n\n”,&s);

/\* for task 3 \*/

printf("input hex int k, p :");

scanf("%x %x", &k, &p );

newint = (p&0xff00)|(k&0xff00)<<8;

printf("new int = %x\n\n",newint);

return 0;

}

解答：

1. 错误修改：

1）第2行：#define PI 3.14159；

错误原因：宏定义在预处理时会将后续程序中的标示符展开为对应全部字符串，而PI不需要分号。

正确形式为：去掉其后分号

2）第3行：intmain(void) {

错误原因：函数返回值与函数名之间应加空格

正确形式为：int main(void) {

3）第6行：short p,k;

错误原因：第24、25行的newint未定义，在这里补上

正确形式为：short p,k,newint;

4）第9、12行：printf(“Input Fahrenheit:” ) ;、

printf( “ \n %d (F) = %.2f (C)\n\n ”, f, c ) ;

错误原因：使用了中文引号

正确形式为：printf("Input Fahrenheit:" ) ;、

printf( " \n %d (F) = %.2f (C)\n\n ", f, c ) ;

4）第11行：scanf("%d", f );

错误原因：scanf函数需传递要写入的变量的地址，应给f加上地址运算符

正确形式为：scanf("%d",&f);

5）第12行：c = 5/9\*(f-32) ;

错误原因：5和9是整型常数，相除会截断小数部分，应该将其中任意一个输表示为或转换为浮点数类型

正确形式为：c = 5.0/9\*(f-32);

6）第17行：scanf("%f", &r);

错误原因：double类型对应的占位符是%lf

改正：scanf("%lf",&r);

7）第19行：printf("\nThe acreage is %.2f\n\n",&s);

错误原因：要输出s的值应该将s作为参数传递给printf，不应传递其地址

正确形式为：printf("\nThe acreage is %.2f\n\n",s);

8）第23行：scanf("%x %x", &k, &p );

错误原因：k,p为short类型，对应十六进制占位符应为%hx

正确形式为：scanf("%hx %hx",&k,&p);

9）第25行：printf("new int = %x\n\n",newint);

错误原因：newint为short类型，对应十六进制占位符应为%hx

正确形式为：printf("new int = %hx\n\n",newint);

10）第24行：newint = (p&0xff00)|(k&0xff00)<<8;

错误原因：要求将k的高字节作为结果的低字节，应该将k的高字节提取出后右移至低字节

正确形式为：newint=(p&0xff00)|(((k&0xff00)>>8)&0x00ff);

1. 错误修改后运行结果：

输入98（华氏度），应该得到36.67（摄氏度）；

输入半径为3，应该得到面积为28.27；

输入k=1234，p=4567，应该得到的new int = 4512;

程序运行结果图1-1所示，程序正确。

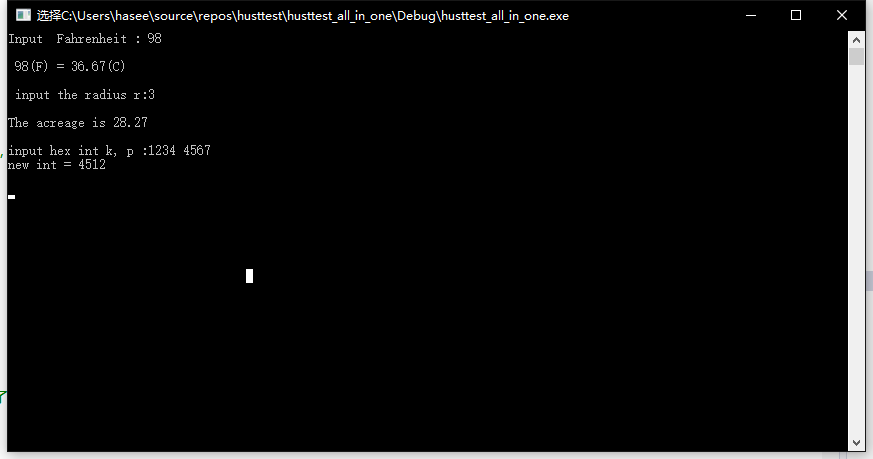


图1-1 改错题的测试结果

1.2.2 源程序修改替换

下面的程序利用常用的中间变量法实现两数交换，请改用不用第三个变量的交换法实现。

#include<stdio.h>

int main( )

{

int a, b, t;

printf(“Input two integers:”);

scanf(“%d %d”,&a,&b);

t=a, a=b, b=t;

prinf(“\na=%d,b=%d”,a,b);

return 0;

}

解答：

考虑到应该把a和b的值的信息合成后存储于同一变量（a或b）中，然后用b的值进行拆解得到a的值，再利用a的值进行拆解得到b的值，利用了某数与自己异或结果为0及异或运算符的结合律/交换律，可用异或运算式直接实现。

1）算法流程如图1-2所示：



图1-2 替换题的程序流程图

2）源程序清单：

#include<stdio.h>

int main(void){

int a, b;

printf("Input two integers : ");

scanf("%d %d", &a, &b);

a ^= b; //a的值为a^b

b ^= a; //b的值为a^b^b=b^b^a=0^a=a

a ^= b; //a的值为a^b^a=a^a^b=0^b=b

printf("\na = %d, b = %d", a, b);

return 0;

}

3）运行结果：

如果输入52 36，即a=52，b=36，应该输出：a=36，b=52;

程序运行结果图1-3所示，程序正确。

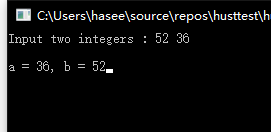


图1-3 替换题的测试结果

1.2.3 程序设计

**（1）**编写一个程序，输入字符ｃ，如果ｃ是大写字母，则将ｃ转换成对应的小写，否则ｃ的值不变，最后输出ｃ。

**解答：**

1）解题思路：

算法流程如图1-4所示：



图1-4 编程题1的程序流程图

2）源程序清单

#include<stdio.h>

int main(void) {

char c;

scanf("%c%\*c", &c);

if (c >= 'A'&&c <= 'Z') {

c += 32;

}

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

return 0;

}

3）测试

1. 测试数据：

（1）测试大写字母变小写字母一般情况。若输入K，应输出k，程序运行结果如图1-5所示。

（2）测试边界情况。若输入A，应输出a，若输入Z，应输出Z，程序运行结果如图1-6所示。

（3）测试非大写字母输入，若输入f，应输出f，程序运行结果如图1-7所示。

1. 对应测试数据的运行结果截图：

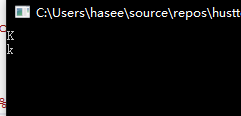


图1-5 编程题1的测试结果1

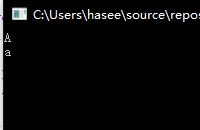
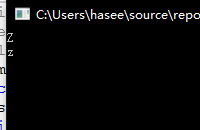
 

图1-6 编程题1的测试结果2

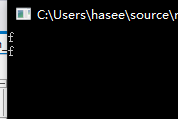


图1-7 编程题1的测试结果3

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**编写一个程序，输入无符号短整数x，ｍ，ｎ（0 ≤ｍ≤ 15, 1 ≤ ｎ≤ 16-ｍ），取出x从第ｍ位开始向左的ｎ位（ｍ从右至左编号为0～15），并使其向左端（第15位）靠齐。

**解答：**

1）解题思路：将x右移m位可将第m位前的置零，再左移16-n位可将m+n-1位后的置零同时将剩余部分向左靠齐

算法流程如图1-8所示：



图1-8 编程题2的程序流程图

2）源程序清单

#include<stdio.h>

int main(void) {

unsigned short x, m, n;

scanf("%hx %hu %hu", &x, &m, &n);

x >>= m;

x <<= (16 - n);

if (m <= 15 && m >= 0 && n >= 1 && ((m + n) <= 16)) {

printf("%#x\n", x);

}

else {

printf("Error!");

}

return 0;

}

3）测试

1. 测试数据：

选择一组正常的测试数据，一组m超出范围的测试数据，一组m位后不足n位的测试数据。

如表1-1所示：

表1-1 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 | m | N |
| 用例1 | 0100 0110 1000 0000（4680） | 7 | 4 | 计算结果1101 0000 0000 0000 即D000 | 图1-9 |
| 用例2 | 1101 0101 1000 0011（D583） | 16 | 1 | Error! | 图1-10 |
| 用例3 | 1101 0101 1000 0011（D583） | 13 | 5 | Error! | 图1-11 |

1. 对应测试数据的运行结果截图

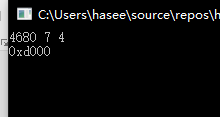


图1-9 编程题2的测试结果1

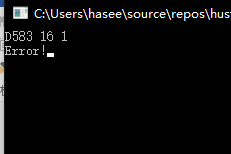


图1-10 编程题2的测试结果2

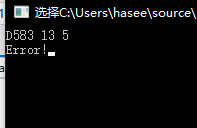


图1-11 编程题2的测试结果3

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

**（3）**IP地址通常是4个用句点分隔的小整数（即点分十进制），如32.55.1.102。这些地址在机器中用无符号长整形表示。编写一个程序，以机器存储的形式读入一个互联网IP地址，对其译码，然后用常见的句点分隔的4部分的形式输出。

例如，整形676879571二进制表示就是：00101000 01011000 01011100 11010011，按照8位一组可表示为：40 88 92 211，由于CPU处理数据的差异，它的顺序是颠倒的，所有最终格式为211.92.88.40。

**解答：**

1）解题思路：用逻辑尺与输入的十进制表示的IP地址进行按位与，分别取出其4个字节并相应右移对应位后，倒序输出所得的4个整数与句点即可。

算法流程如图1-12所示：



图1-12 编程题3的程序流程图

2）源程序清单

#include<stdio.h>

int main(void) {

unsigned long num;

unsigned short place1, place2, place3, place4;

scanf("%lu", &num);

place1 = (num & 0xFF000000) >> (6 \* 4);

place2 = (num & 0x00FF0000) >> (4 \* 4);

place3 = (num & 0x0000FF00) >> (2 \* 4);

place4 = (num & 0x000000FF) >> (0 \* 4);

printf("%hu.%hu.%hu.%hu\n", place4, place3, place2, place1);

return 0;

}

3）测试

1. 测试数据：输入676879571，二进制表示是：00101000 01011000 01011100 11010011，即40 88 92 211，于是由题意，输出的IP地址应为211.92.88.40，程序运行结果如图1-13所示。
2. 对应测试数据的运行结果截图：

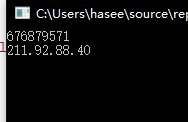


图1-11 编程题3的测试结果1

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

1.3 自设题

**（1）** 自设实验题目：制作自己的计算器（一）——计算器的基本框架及输入输出处理

**（2）** 实验目的：利用各个章节所学习的内容来实现一个能执行四则及乘方、开方、括号运算的可调精度计算器。本章进行计算器的流程设计以及基本输入输出部分的设计。

**（3）**实验程序：

将计算器功能的实现分为读取输入，计算，输出三部分，分别置于源文件reading.c ，calculating.c ，output.c 中，建立库文件why\_calculator.h，主文件why\_calculator.c。

·reading.c:

1）基本思路：

在reading.c 中需实现两个功能：

1.将输入的中序表达式进行格式化处理，储存在input数组中。

2.将input数组转换为逆波兰形式的栈repol。

在本章中只实现功能1，功能2将在后续相关章节中实现。

2）源程序清单：

见附录1.3.1.源程序清单。

3）测试：见附录1.3.1.测试。

·output.c:

1）基本思路：

在output.c 中需实现两个功能：

1.设置基准输出格式。

2.设置输出结果的精度。

3.设置异常安全系统。

在本章中只实现功能1、2，功能3将在后续相关章节中实现。

2）源程序清单：

见附录1.3.2.源程序清单。

3）测试：见附录1.3.2.测试。

·why\_calculator.h:

1）基本思路：

向库中添加本次实现的函数以及必要常量。

2）源程序清单：

#ifndef \_MAIN\_H

#define \_MAIN\_H

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<malloc.h>

#include<stdlib.h>

#define MAXSIZE 500//输入表达式的最大长度

int readinput(char \*input, char \*buf);//"reading.c"

int preciquest(void);//"output.c"

void baseoutput(void);//"output.c"

void endingoutput(void);//"output.c"

void outputhelp(void);//"output.c"

void outputresult(double result);//"output.c"

#endif //\_MAIN\_H

3）测试：此文件无测试内容。

1.4 实验小结

在做改错题时，先尽量不依赖编译器直接找出语法错误，从而减少自己在编程时出现语法错误的可能。再结合IDE调试功能捋清程序逻辑，找出与导致程序与要求不符合的逻辑错误并修改。这个过程需要细心和耐心，是对我们编程能力和心理素质的双重考验。这次的题目中要特别注意的是中文和英文符号的不同，在编程的语法中所有符号都应采用英文符号。

源程序修改与替换题是对我们对知识的理解能力的考察，理解的越深入，越可能找出多种替代方案。本次实验中的替换题就有多种解答方案，可以用异或，也可以用加减来实现，其本质都是用第二个数对第一个数进行加密，然后用另一个数来解密两次，结果分别储存在第二个和第一个数中，从而实现交换。

程序设计题在实践中提高了我们对所学知识的熟练和理解程度，这次的三个题目都较为简单，但在第三个题目中，右移对应位数时应该用逻辑清晰的表达式而不是其结果来表示所需右移的位数，这样若出现问题可以很容易改正。

自设题是国庆期间产生的想法，在实现的过程中也有了很多收获，对栈的操作，输入缓冲区的处理等有了更深的理解。

实验2 流程控制实验

2.1 实验目的

（1）掌握复合语句、if语句、switch语句的使用，熟练掌握for、while、do-while三种基本的循环控制语句的使用，掌握重复循环技术，了解转移语句与标号语句。

（2）熟练运用for、while、do-while语句来编写程序。

（3）练习转移语句和标号语句的使用。

（4）使用集成开发环境中的调试功能：单步执行、设置断点、观察变量值。

2.2 实验内容

2.2.1 源程序改错

下面是用宏来计算平方差和交换两数的源程序。在这个源程序中,存在若干语法和逻辑错误,要求在计算机上对这个源程序进行调试修改，使之能够正确完成计算平方差和交换两数的任务。

#include <stdio.h>

int main(void)

{

int i,n,s=1;

printf("Please enter n:");

scanf("%d", n);

for(i=1,i<=n,i++)

s=s\*i;

printf("%d! = %d",n,s);

return 0;

}

解答：

1. 错误修改：

1）第6行：scanf("%d", n);

错误原因：scanf函数需要接收需写入的变量的地址而不是值。

正确形式为：scanf("%d", &n);

2）第7行：for(i=1,i<=n,i++)

错误原因：for循环的各部分之间用分号而不是逗号连接。

正确形式为：for(i=1;i<=n;i++)

1. 错误修改后运行结果：

输入8，应该得到40310，运行结果如图2-1所示。

输入半径为0，应该得到1，运行结果如图2-2所示。

输入1，应该得到1，运行结果如图2-3所示。

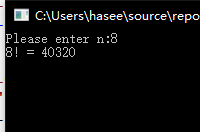


图2-1 改错题的测试结果1

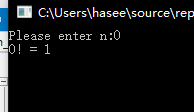


图2-2 改错题的测试结果2

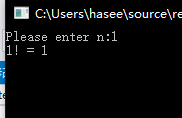


图2-3 改错题的测试结果3

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

2.2.2 源程序修改替换

**（1）**修改第1题，分别用while和do-while语句替换for语句。

**（2）**修改第1题，要求输入改为“整数s”，输出改为“满足n! ≥ s 的最小整数n”。例如，输入整数为40310，输出结果为n=8。

**解答：**

**（1）**

1）解题思路：

while语句的算法流程如图2-4所示：



图2-4 替换题1的程序流程图

2）源程序清单

#include <stdio.h>

int main(void)

{

int i = 1, n, s = 1;

printf("Please enter n:");

scanf("%d", &n);

while (i <= n) {

s = s\*i;

i++;

}

printf("%d!=%d", n, s);

return 0;

}

3）测试

测试数据及结果与2.2.1完全相同，**说明运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**

1）解题思路：

do-while语句的算法流程如图2-5所示：



图2-5 替换题2的程序流程图

2）源程序清单

#include <stdio.h>

int main(void)

{

int i = 1, n, s = 1;

printf("Please enter n:");

scanf("%d", &n);

do {

s = s\*i;

i++;

} while (i <= n);

printf("%d!=%d", n, s);

return 0;

}

3）测试

测试数据及结果与2.2.1完全相同，**说明运行结果与理论分析吻合，验证了程序的正确性。**

**（3）**

1）解题思路：

算法流程如图2-6所示：



图2-6 替换题3的程序流程图

2）源程序清单

#include <stdio.h>

int main(void)

{

int i, n = 0, s, m = 1;

printf("Please enter s:");

scanf("%d", &s);

if (s <= 0) {

printf("Error");

}

else if (s == 1) {

printf("0");

}

else {

for (i = 1;; i++) {

m = m\*i;

n++;

if (m >= s) {

printf("n=%d", n);

break;

}

}

}

return 0;

}

3）测试

1. 测试数据：

（1）输入40319，应输出8，运行结果如图2-7所示。

（2）输入1，应输入0，运行结果如图2-8所示。

1. 对应测试数据的运行结果截图：

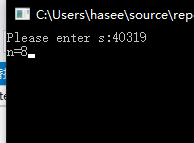


图2-7 替换题3的测试结果1

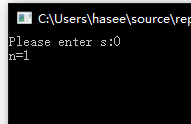


图2-8 替换题3的测试结果2

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

2.2.3 程序设计

**（1）** 假设工资税金按以下方法计算：x ＜ 1000元，不收取税金；1000 ≤ x ＜ 2000，收取5%的税金；2000 ≤ x ＜ 3000，收取10%的税金；3000 ≤ x ＜ 4000，收取15%的税金；4000 ≤ x ＜ 5000，收取20%的税金；x＞5000，收取25%的税金。编写一个程序，输入工资金额，输出应收取税金额度，要求分别用if语句和switch语句来实现。

解答：

1）解题思路：

（1）if语句：用连续的if-else语句判断出x属于哪个区间内，并用对应区间的税金计算方式来计算。程序流程如图2-9所示。



图2-9 编程题1的程序流程图1

（2）switch语句：计算x/1000的结果的整数部分作为switch的选择依据，并选择对应的税金计算方式。程序流程如图2-10所示。



图2-10 编程题1的程序流程图2

2）源程序清单

#include<stdio.h>

void wayif(double x);

void waysw(double x);

int main(void) {

double x;

scanf("%lf", &x);

wayif(x);

waysw(x);

return 0;

}

void wayif(double x) {

double res = 0;

if (x < 1000) {

res = 1000 \* 0;

}

else if (x < 2000) {

res = 1000 \* 0 + (x - 1000)\*0.05;

}

else if (x < 3000) {

res = 1000 \* (0 + 0.05) + (x - 2000)\*0.10;

}

else if (x < 4000) {

res = 1000 \* (0 + 0.05 + 0.10) + (x - 3000)\*0.15;

}

else if (x < 5000) {

res = 1000 \* (0 + 0.05 + 0.10 + 0.15) + (x - 4000)\*0.20;

}

else {

res = 1000 \* (0 + 0.05 + 0.10 + 0.15 + 0.20) + (x - 5000)\*0.25;

}

printf("In if way, tax is :%lf\n", res);

}

void waysw(double x) {

double res = 0;

switch ((int)x / 1000) {

case 0:

res = 1000 \* 0;

break;

case 1:

res = 1000 \* 0 + (x - 1000)\*0.05;

break;

case 2:

res = 1000 \* (0 + 0.05) + (x - 2000)\*0.10;

break;

case 3:

res = 1000 \* (0 + 0.05 + 0.10) + (x - 3000)\*0.15;

break;

case 4:

res = 1000 \* (0 + 0.05 + 0.10 + 0.15) + (x - 4000)\*0.20;

break;

default:

res = 1000 \* (0 + 0.05 + 0.10 + 0.15 + 0.20) + (x - 5000)\*0.25;

break;

}

printf("In switch way, tax is :%lf\n", res);

}

3）测试

1. 测试数据：

选取一组边界条件，输入1000，一组一般数据，输入5001，如表2-1所示

表2-1 编程题1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 | 1000 | 0 | 图2-11 |
| 用例2 | 5000 | 500.25 | 图2-12 |

1. 对应测试数据的运行结果截图：

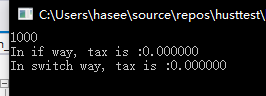


图2-11 编程题1的测试结果1

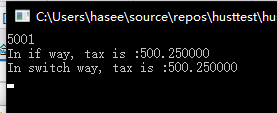


图2-12 编程题1的测试结果2

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**编写一个程序,将输入的一行字符复制到输出，复制过程中将一个以上的空格字符用一个空格代替。

解答：

1）解题思路：

程序流程如图2-13所示。



图2-13 编程题2的程序流程图

2）源程序清单

#include<stdio.h>

#define MAXSIZE 500

int main(void) {

char input[MAXSIZE];

int i = 0, j = 0;

while ((input[i] = fgetc(stdin)) != '\n') {

if (input[i] == ' '&&(i!=0)) {

if (input[i-1] == ' ') {

--i;

}

}

++i;

}

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

fputc(input[j], stdout);

}

return 0;

}

3）测试

1. 测试数据：

测试数据应既包含开头处的多重空格也包含段中的多重空格，输入数据及运行结果如图2-14所示。

1. 对应测试数据的运行结果截图：

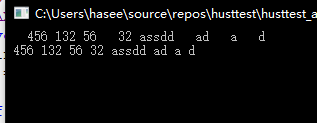


图2-14 编程题2的测试结果

**说明上述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，打印如下的杨辉三角形。

1 /\*第0行 \*/

1 1 /\*第1行 \*/

1 2 1 /\*第2行 \*/

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

每个数据值可以由组合计算（表示第i行第j列位置的值），而的计算如下：

 (i=0,1,2,…)

 (j=0,1,2,3,…,i)

说明：本程序中为了打印出金字塔效果，要注意空格的数目。一位数之间是3个空格，两位数之间有2个空格，3位数之间只有一个空格，程序编制过程中要注意区分。

解答：

1）解题思路：

程序流程如图2-15所示。这里只需修改参数i即可实现输出不同层数的杨辉三角。



图2-15 编程题3的程序流程图

2）源程序清单

#include <stdio.h>

void choose(i) {

if (i == 0)

;

else if (i<10)

printf(" %d", i);

else if (i<100)

printf(" %d", i);

else if (i<1000)

printf(" %d", i);

}

int yanghui[12][50] = { { 0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0 } };

void sanjiao(int i) {

int line, raw, k, a;

int flag = 1;

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

printf(" ");

printf("1");

for (line = 1; line <= (i - 1); line++) {

flag = 1;

for (raw = 1; raw <= 50; raw++) {

yanghui[line][raw] = (yanghui[line - 1][raw - 2] + yanghui[line - 1][raw]);

if (yanghui[line][raw] != 0 && flag) {

flag = 0;

printf("%d", yanghui[line][raw]);

}

else if (raw != line)

choose(yanghui[line][raw]);

else {

printf("\n");

for (k = 0; k < (i - 1) - line; k++)

printf(" ");

}

}

}

printf("\n");

}

int main(void) {

int i = 10;

sanjiao(i);

return 0;

}

3）测试

1. 测试数据：

本题没有输入，直接运行即可得到结果，运行结果如图2-16所示。

1. 对应测试数据的运行结果截图：

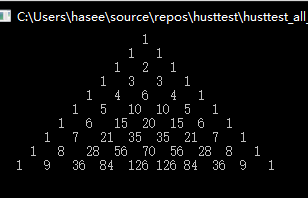


图2-16 编程题3的测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

**（4）**编写一个程序，将用户输入的任意正整数逆转，例如，输入1234，输出4321。

解答：

1）解题思路：

算法流程如图2-17所示：



图2-17 编程题4的程序流程图

2）源程序清单

#include<stdio.h>

int main(void) {

int num = 0, sum = 0;

int i = 0;

int place;

scanf("%d", &num);

for (i = 0; num != 0; i++) {

place = num % 10;

num /= 10;

sum += place;

sum \*= 10;

}

sum /= 10;

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

return 0;

}

3）测试

1. 测试数据：

选取一组边界条件，输入5，一组一般数据，输入156643，如表2-2所示

表2-2 编程题4的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| num |
| 用例1 | 5 | 5 | 图2-18 |
| 用例2 | 156643 | 346651 | 图2-19 |

1. 对应测试数据的运行结果截图：

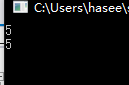


图2-18 编程题4的测试结果1

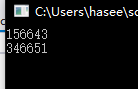


图2-19 编程题4的测试结果2

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

2.2.4 选做题

编写一个程序，用牛顿迭代法求方程满足精度e=10-6的一个近似根，并在屏幕上输出所求近似根。

牛顿迭代法求方程近似根的迭代公式为：



其中,是函数f(x) 的导函数。牛顿迭代法首先任意设定的一个实数来作为近似根的迭代初值x0，然后用迭代公式计算下一个近似根x1。如此继续迭代计算x2, x3, …, xn, 直到，此时值xn即为所求的近似根。

解答：

1）解题思路：

利用循环结构，循环执行题目所给的迭代公式，直到结果满足精度要求后输出所得结果。

2）源程序清单

#include<stdio.h>

#define fun(x) (((3\*x-4)\*x-5)\*x+13)

#define dfun(x) ((9\*x-8)\*x-5)

double newtown(double x);

int main(void) {

int x = 1;

double res = newtown(x);

printf("%f\n", res);

return 0;

}

double newtown(double x) {

double newx = x - fun(x) / dfun(x);

double dec = newx - x;

while (!(dec < 1e-6&&dec>(-1e-6))) {

x = newx;

newx = x - fun(x) / dfun(x);

dec = newx - x;

}

return newx;

}

3）测试

1. 测试数据：

本题没有输入，直接运行即可得到结果，运行结果如图2-20所示。

1. 对应测试数据的运行结果截图：

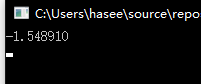


图2-20 选做题的测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

2.3 自设题

**（1）** 自设实验题目：制作自己的计算器（二）——计算器的精度选择

**（2）** 实验目的：利用各个章节所学习的内容来实现一个能执行四则及乘方、开方、括号运算的可调精度计算器。本章进行计算器的精度处理。

**（3）**实验程序：

·output.c:

1）基本思路：

请求用户输入精度，写入到全局变量preci中，并通过switch语句判断精度，决定输出方式。

2）源程序清单：

见附录2.3.1.源程序清单。

3）测试：

见附录2.3.1.测试。

2.4 实验小结

本次实验的改错题较为简单，没有遇到什么问题。而在替换题的第三题中最初忘记了对边界条件s=1的处理，导致s=1时输出结果为1而不是0，以后应更加重视边界条件，同时要注意测试数据中一定要包含边界条件测试。

编程题的重点在于第二题中若输入的第一个字符就是空格，应直接读取下一个，而不能继续寻找前一个字符（会导致数组越界）。

选做题只要能考虑到精度误差值是有正有负的就不会出现太大问题了。

实验3 函数与程序结构实验

3.1 实验目的

（1）熟悉和掌握函数的定义、声明；函数调用与参数传递方法；以及函数返回值类型的定义和返回值使用。

（2）熟悉和掌握不同存储类型变量的使用。

（3）熟悉多文件编译技术。

3.2 实验内容

3.2.1 源程序改错

下面是计算s=1!+2!+3!+…+n!的源程序，在这个源程序中存在若干语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。。

#include <stdio.h>

int main(void)

{

int k;

for(k=1;k<6;k++)

printf("k=%d\tthe sum is %ld\n",k,sum\_fac(k));

return 0;

}

long sum\_fac(int n)

{

long s=0;

int i;

long fac;

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

fac\*=i;

s+=fac;

return s;

}

解答：

1. 错误修改：

1）第2行：int main(void)

错误原因：缺少对sum\_fac函数的声明

正确形式为：第1、2行间插入一行long sum\_fac(int n);

2）第13行：long fac;

错误原因：fac应有初始值1。

正确形式为：long fac = 1;

3）第14-16行：for(i=1;i<=n;i++)

fac\*=i;

s+=fac;

错误原因：s+=fac;语句应在for循环内才能达到每次加一个新的1!的效果。

正确形式为：for(i=1;i<=n;i++){

fac\*=i;

s+=fac;

}

1. 错误修改后运行结果：

此程序无输入，运行结果如图3-1所示。

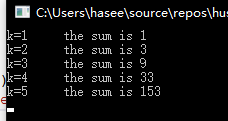


图3-1 修改题的测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

3.2.2 源程序修改替换

**（1）**修改第1题中sum\_fac函数，使其计算量最小。

**（2）**修改第1题中sum\_fac函数，计算。

**解答：**

**（1）**

1）解题思路：

将s、i、fac均定义为static类型的静态变量，它们的初始化语句只会运行一次，因此在每次调用sum\_fac函数时，其值都保留了上次调用完毕时的值，从而不必在每次调用时都再将已经计算过的部分用循环语句再计算一次。

2）源程序清单

#include <stdio.h>

long sum\_fac(int n);

int main(void)

{

int k;

for (k = 1; k<6; k++)

printf("k=%d\tthe sum is %ld\n", k, sum\_fac(k));

return 0;

}

long sum\_fac(int n)

{

static int s = 0, fac = 1, i = 1;//static

fac \*= i;

s += fac;

i++;

return s;

}

3）测试

测试数据及结果与3.2.1完全相同，**说明运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**

1）解题思路：

将s的类型改为double以实现浮点运算，并将fac\*=i改为fac/=i，即将计算n!变为了计算1/(n!)

2）源程序清单

#include <stdio.h>

double sum\_fac(int n);

int main(void)

{

int k;

for (k = 1; k<6; k++)

printf("k=%d\tthe sum is %lf\n", k, sum\_fac(k));

return 0;

}

double sum\_fac(int n){

double s = 0;

int i;

double fac = 1;

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

fac /= i;

s += fac;

}

return s;

}

3）测试

此程序无输入，运行结果如图3-2所示。

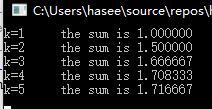


图3-2 替换题2的测试结果

**说明运行结果与理论分析吻合，验证了程序的正确性。**

3.2.3 跟踪调试

下面是计算fabonacci数列前n项和的程序，要求单步执行程序，观察p,i,sum,n值。

（1）刚执行完scanf("%d",&k);语句，p,i值是多少？

（2）从fabonacci函数返回后光条停留在哪个语句上？

（3）进入fabonacci函数，watch窗口显示的是什么？

（4）当i=3，从调用fabonacci函数到返回，n值如何变化？

#include <stdio.h>

int main(void)

{

int i,k;

long sum=0,\*p=&sum; /\* 声明p为长整型指针，指向sum \*/

scanf("%d",&k);

for(i=1;i<=k;i++){

sum+=fabonacci(i);

printf("i=%d\tthe sum is %ld\n",i,\*p); /\* \*p等价于sum \*/

}

return 0;

}

long fabonacci(int n)

{

if(n==1 || n==2)

return 1;

else

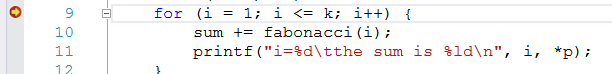
return fabonacci(n-1)+fabonacci(n-2);

}

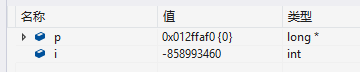
其中，“ long sum=0,\*p=&sum; ”声明p为长整型指针并用&sum取出sum的地址对p初始化。\*p表示引用p所指的变量（\*p即sum）。

解答：

**（1）p为sum的地址，i未初始化，其值为内存中随机值。如图3-3所示。**



**图3-3-1 调试题的运行截图1**



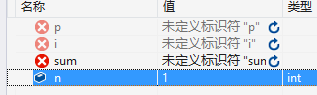
**图3-3 调试题的运行截图2**

**（2）停留在的语句如图3-4所示.**



**图3-4 调试题的运行截图3**

**（3）p, i, sum均为未定义，n的值为1。如图3-5所示。**



**图3-5 调试题的运行截图4**

**（4）n的值变化过程为in->3->2->3->1->3->out。如图3-6所示。**



**图3-6-1 调试题的运行截图5**



**图3-6-2 调试题的运行截图6**



**图3-6-3 调试题的运行截图7**



**图3-6-4 调试题的运行截图8**



**图3-6-5 调试题的运行截图9**



**图3-6-6 调试题的运行截图10**

3.2.4 程序设计

**（1）**编程让用户输入两个整数，计算两个数的最大公约数并且输出之（要求用递归函数实现求最大公约数）。同时以单步方式执行该程序，观察递归过程。

解答：

1）解题思路：

程序流程如图3-7所示。



图3-7 编程题1的程序流程图

2）源程序清单

#include<stdio.h>

int gcd(int x, int y)

{

return y ? gcd(y, x%y) : x;

}

int main(void) {

int num1, num2;

scanf("%d %d", &num1, &num2);

int max = gcd(num1, num2);

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

return 0;

}

3）测试

1. 测试数据：

选取一组边界条件，输入1 1，应输出1，一组一般数据，输入125 225，应输出25，如表3-1所示

表3-1 编程题1的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| num1 num2 |
| 用例1 | 1 1 | 1 | 图3-8 |
| 用例2 | 125 225 | 25 | 图3-9 |

1. 对应测试数据的运行结果截图：

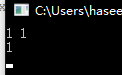


图3-8 编程题1的测试结果1

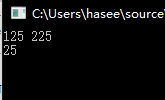


图3-9 编程题1的测试结果2

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**编程验证歌德巴赫猜想：一个大于等于4的偶数都是两个素数之和。

编写一个程序证明对于在符号常量BEGIN和END之间的偶数这一猜测成立。例如，如果BEGIN为10，END为20，程序的输出应为：

GOLDBACH'S CONJECTURE:

Every even number n>=4 is the sum of two primes.

10=3+7

12=5+7

20=3+17

解答：

1）解题思路：

程序流程如图3-10所示。



图3-10 编程题2的程序流程图

2）源程序清单

#include<stdio.h>

#include<math.h>

int checkprime(int num);

void search(int begin, int end, int\* prime);

void calprime(int\* prime, int max);

int main(void) {

int begin, end;

while ((scanf("%d %d", &begin, &end), begin) != 0) {

int len = end - 1;

int \*prime = (int\*)calloc(len, sizeof(int));

printf("GOLDBACH'S CONJECTURE:\n"

"Every even number n >= 4 is the sum of two primes.\n");

calprime(prime, len);

if (begin % 2) {

search(begin + 1, end, prime);

printf("\n");

}

else {

search(begin, end, prime);

printf("\n");

}

free(prime);

prime = NULL;

}

return 0;

}

int checkprime(int num) {

int i = 2;

for (i = 2; i <= (int)sqrt(num); i++) {

if (!(num%i)) {

return 0;

}

}

return 1;

}

void calprime(int\* prime, int max) {

prime[0] = 2;

int i = 0;

int j = 1;

for (i = 3; i < max + 1; i++) {

if (checkprime(i)) {

prime[j] = i;

j++;

}

}

}

void search(int begin, int end, int\* prime) {

int now = begin;

int flag = 0;

for (now = begin; now <= end; now += 2) {

int i = 0, j = 0;

flag = 0;

int len = end / 2 + 1;

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

if (prime[i] == 0) {

printf("Error!!!!!!!\n");

break;

}

int dec = now - prime[i];

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

if ((dec == prime[j]) && (prime[j] != 0)) {

flag = 1;

break;

}

}

if (flag) {

printf("%d=%d+%d\n", now, prime[i], dec);

break;

}

}

}

}

3）测试

1. 测试数据：

选取一组一般数据，输入1000000 1003568，一组边界条件，输入4 4，如表3-2所示

表3-2 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 |
| num1 num2 |
| 用例1 | 1000000 1003568 | 均能分解 | 图3-11 |
| 用例2 | 4 4 | 4=2+2 | 图3-12 |

1. 对应测试数据的运行结果截图：

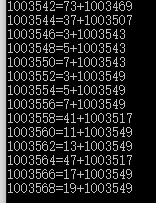
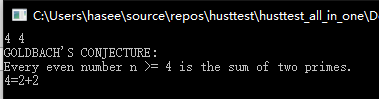


图3-11 编程题2的测试结果1



3-12 编程题2的测试结果2

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

3.2.5 选做题

假设一个C程序由file1.c和file2.c两个源文件及一个file.h头文件组成，file1.c、file2.c和file.h的内容分别如下所述。试编辑该多文件C程序，并编译和链接。然后运行生成的可执行文件。

源文件file1.c的内容为：

#include "file.h"

int x,y; /\* 外部变量的定义性说明 \*/

char ch; /\* 外部变量的定义性说明 \*/

int main(void)

{

x=10;

y=20;

ch=getchar( );

printf("in file1 x=%d,y=%d,ch is %c\n",x,y,ch);

func1( );

return 0;

}

源文件file2.c的内容为：

#include "file.h"

void func1(void)

{

x++;

y++;

ch++;

printf("in file2 x=%d,y=%d,ch is %c\n",x,y,ch);

}

头文件file.h的内容为：

#include <stdio.h>

extern int x,y; /\* 外部变量的引用性说明 \*/

extern char ch; /\* 外部变量的引用性说明 \*/

void func1(void); /\* func1函数原型 \*/

解答：

（1）文件存放方式如图3-13所示。编译时，编译器将每一个源码文件生成一个".o"后缀的临时文件，链接器再对这些文件进行链接，生成.exe文件。



图3-13-1 选做题的运行截图1

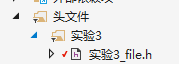


图3-13-2 选做题的运行截图2

（1）运行生成的.exe文件，结果如图3-14所示。

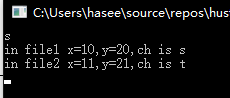


图3-14 选做题的运行截图3

3.3 自设题

计算器的各函数设计见1.3自设题中的[my\_calculator.h](#书签1)文件内容

3.4 实验小结

（1）本次改错题较为简单，没有遇到问题。

（2）替换题中要注意，static声明的变量在函数内外声明均可，因为static变量的初始化语句只会被执行一次，不会导致初始化语句使变量的值重置的问题。

（3）调试题要求对单步调试，监视有所掌握，便于理解程序运行的过程并修改程序中的问题。

（4）编程题的第二题较复杂，要提前构思好各个函数并画出简易流程图才能更好地完成程序设计。

（5）选做题旨在帮助我们理解编译与链接的差别以及生成可执行文件的过程。

实验4 函数与程序结构实验

4.1 实验目的

（1）掌握文件包含、宏定义、条件编译、assert宏的使用；

（2）练习带参数的宏定义、条件编译的使用；

（3）练习assert宏的使用；

（4）使用集成开发环境中的调试功能：单步执行、设置断点、观察变量值。

4.2 实验内容

4.2.1 源程序改错

下面是用宏来计算平方差和交换两数的源程序。在这个源程序中,存在若干语法和逻辑错误,要求在计算机上对这个源程序进行调试修改，使之能够正确完成计算平方差和交换两数的任务。

#include<stdio.h>

#define SUM a+b

#define DIF a-b

#define SWAP(a,b) a=b,b=a

int main(void)

{

int b, t;

printf("Input two integers a, b:");

scanf("%d,%d", &a,&b);

printf("\nSUM=%d\n the difference between square of a and square of b is:%d", SUM, SUM\*DIF);

SWAP(a,b);

printf("\nNow a=%d,b=%d\n",a,b);

return 0;

}

解答：

1. 错误修改：

1）第2、3行：

#define SUM a+b

#define DIF a-b

错误原因：

定义宏时表达式要打上括号，不然会导致计算时优先级错误。

正确形式为：#define SUM (a+b)

#define DIF (a-b)

2）第4行：#define SWAP(a,b) a=b,b=a

错误原因：交换a与b的值所使用的表达式错误，这样会使a与b的值相等，无法实现交换。

正确形式为：#define SWAP(a,b) t=a, a=b, b=t

3）第7行：int b, t;

错误原因：变量a未定义，应在这里补上定义。

正确形式为：int a, b, t;

1. 错误修改后运行结果：

测试边界条件，输入0,0，得到的SUM应为0，平方差为9，交换后a,b仍均为0，运行结果如图4-1所示

测试一般情况，输入12,32，得到的SUM应为44，平方差为-880，交换后a为32，b为12，运行结果如图4-2所示。

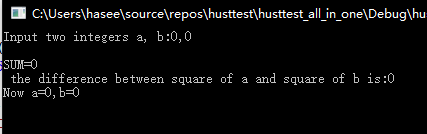


图4-1 修改题的测试结果1

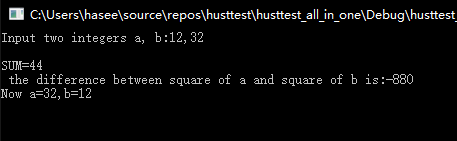


图4-2 修改题的测试结果2

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

4.2.2 源程序修改替换

下面是用函数实现求三个数中最大数、计算两数之和的源程序。在这个源程序中存在若干语法和逻辑错误,要求：

(1) 对这个例子程序进行调试修改，使之能够正确完成指定任务。

(2) 用带参数的宏替换函数max，来实现求最大数的功能。

#include<stdio.h>

int main(void)

{

int a, b, c;

float d, e;

printf("Enter three integers:");

scanf("%d,%d,%d",&a,&b,&c);

printf("\nthe maximum of them is %d\n",max(a,b,c));

printf("Enter two floating point numbers:");

scanf("%f,%f",&d,&e);

printf("\nthe sum of them is %f\n",sum(d,e));

return 0;

}

int max(int x, int y, int z)

{

int t;

if (x>y) t=x;

else t=y;

if (t<z) t=z;

return t;

}

float sum(float x, float y)

{

return x+y;

}

**解答：**

**（1）**

1）解题思路：

为float sum(float x, float y)函数添加声明语句即可

2）源程序清单

#include<stdio.h>

float sum(float x, float y);

int main(void)

{

int a, b, c;

float d, e;

printf("Enter three integers:");

scanf("%d,%d,%d", &a, &b, &c);

printf("\nthe maximum of them is %d\n", max(a, b, c));

printf("Enter two floating point numbers:");

scanf("%f,%f", &d, &e);

printf("\nthe sum of them is %f\n", sum(d, e));

return 0;

}

int max(int x, int y, int z)

{

int t;

if (x>y) t = x;

else t = y;

if (t<z) t = z;

return t;

}

float sum(float x, float y)

{

return x + y;

}

3）测试

1. 测试数据：

选取一组边界条件，输入5,5,5/0,1，应输出5/1.000000，如图4-3所示，一组一般数据，输入5,35,8/465.3595,13.237，应输出35/478.596500，如表4-4所示

由于浮点数的储存形式较为特殊，计算结果与实际值可能有所差别，误差与两数中较大一个的数量级有较大差别，通常情况下可以直接对误差进行舍入

表4-1 替换题1的测试数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | | 理 论 结 果 | 运 行 结 果 |
| a b c | d e |
| 用例1 | 5 5 5 | 0 1 | 5  1.000000 | 图4-3 |
| 用例2 | 5 35 8 | 465.3595 13.237 | 35  478.596500 | 图4-4 |

1. 对应测试数据的运行结果截图：

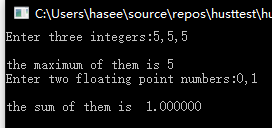


图4-3 替换题1的测试结果1

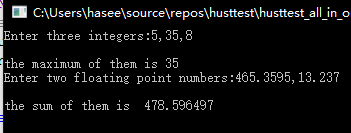


图4-4 替换题1的测试结果2

**说明运行结果与理论分析的误差在合理范围内，验证了程序的正确性。**

**（2）**

1）解题思路：

将max定义为含参宏，用问号运算符实现在宏定义中对max计算结果的判断。

2）源程序清单

#include<stdio.h>

#define max(x, y, z) ((x>y)?((x>z)?x:((z>y)?z:y)):(y>z)?y:((z>x)?z:x))

float sum(float x, float y);//声明sum函数

int main(void)

{

int a, b, c;

float d, e;

printf("Enter three integers:");

scanf("%d,%d,%d", &a, &b, &c);

printf("\nthe maximum of them is %d\n", max(a, b, c));

printf("Enter two floating point numbers:");

scanf("%f,%f", &d, &e);

printf("\nthe sum of them is %f\n", sum(d, e));

return 0;

}

float sum(float x, float y)

{

return x + y;

}

3）测试

测试数据及结果与替换题1完全相同，**说明运行结果与理论分析吻合，验证了程序的正确性。**

4.2.3 跟踪调试

下面程序的功能是利用R计算圆的面积s，以及面积s的整数部分。现要求：

(1) 修改程序，使程序编译通过且能运行。

(2) 单步执行。进入函数integer\_fraction时,watch窗口中x为何值？在返回main时, watch窗口中i为何值？

(3) 排除错误，使程序能正确输出面积s值的整数部分，不会输出错误信息Assertion Failed。

#include<stdio.h>

#define R

int main(void)

{

float r, s;

int s\_integer=0;

printf ("input a number: ");

scanf("%f",&r);

#ifdef R

s=3.14159\*r\*r;

printf("area of round is: %f\n",s);

s\_integer= integer\_fraction(s);

assert((s-s\_integer)>=1.0);

printf("the integer fraction of area is %d\n", s\_integer);

#endif

return 0;

}

int integer\_fraction(float x)

{

int i=x;

return i;

}

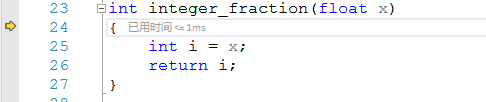
解答：

**（1）修改：**

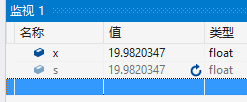
**assert函数在assert.h中声明，需在包含文件时包含assert.h头文件。**

int integer\_fraction(float x)函数未声明，需补充其声明。

**（2）此时x的值等于在进入此函数前s的值，如图4-5所示。**



**图4-5-1 调试题的运行截图1**

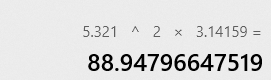


**图4-5-2 调试题的运行截图2**

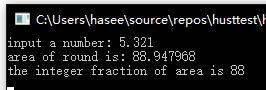
**（3）修改：**

**将第13行中的大于等于改为小于，即改为assert((s - s\_integer) < 1.0);**

**修改完成后输入5.321，用计算器计算得面积精确到小数点后六位应为88.947966，见图4-6，由于浮点数的特性，可能有一定误差，实际运行结果见图4-7**



**图4-6 调试题的理论结果**



**图4-7 调试题的运行截图3**

**说明运行结果与理论分析的误差在合理范围内，验证了程序的正确性。**

4.2.4 程序设计

**（1）**已知三角形的面积是，其中，a,b,c为三角形的三边。定义两个带参数的宏，一个用来求s，另一个用来求area,试编写一程序，用带参数的宏来计算三角形的面积。

解答：

1）解题思路：

程序流程如图4-8所示。



图4-8 编程题1的程序流程图

2）源程序清单

#include<stdio.h>

#include<math.h>

#define cals(a,b,c) ((a+b+c)/2)

#define calarea(a,b,c) (sqrt(s\*(s-a)\*(s-b)\*(s-c)))

int main(void) {

double a, b, c, s;

double area;

scanf("%lf %lf %lf", &a, &b, &c)

s = cals(a, b, c);

area = calarea(a, b, c);

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

return 0;

}

3）测试

1. 测试数据：

选取一组边界条件，输入0 0 0，应得到s与area均为0，一组一般数据，输入2.51 3.23 4.23，应得到s为44，area为，一组特殊数据，输入1.2 5.5 3.12，由于不能构成三角形，area计算结果应不存在，应得到s为4.91,area为-nan(ind)。如表4-2所示

表4-2 编程题1的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| a b c |
| 用例1 | 0 0 0 | 0.000000 0.000000 | 图4-9 |
| 用例2 | 2.51 3.23 4.23 | 4.985000 4.043265（图4-10） | 图4-11 |
| 用例3 | 1.2 5.5 3.12 | 4.910000 –nan(ind) | 图4-12 |

1. 对应测试数据的运行结果截图：



图4-9 编程题1的测试结果1

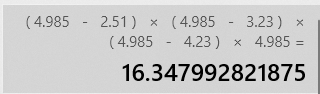


图4-10-1 测试结果2的理论结果1

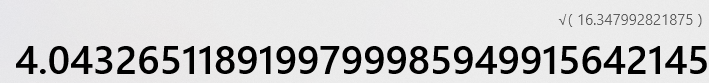


图4-10-2 测试结果2的理论结果2

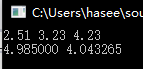


图4-11 编程题1的测试结果2



图4-12 编程题1的测试结果3

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**使用条件编译方法编写一程序,其功能要求是,输入一行电报文字，可以任选两种输出：一为原文输出；二为变换字母的大小写（如小写‘a’变成大写‘A’，大写‘D’变成小写‘d’），其他字符不变。用#define命令控制是否变换字母的大小写。例如，#define CHANGE 1 则输出变换后的文字，若#define CHANGE 0则原文输出。

解答：

1）解题思路：

程序流程如图4-13所示。



图4-13 编程题2的程序流程图

2）源程序清单

#include<stdio.h>

#define MAXLINE 1024

#define CHANGE 0

int main(void) {

int n;

char c, ch[MAXLINE];

int i = 0;

scanf("%d", &n);

getchar();

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

int p = 0;

for (p = 0; (scanf("%c", &ch[p]), ch[p]) != '\n'; p++);

int j = 0;

for (j = 0; ch[j] != '\n'; j++) {

#if CHANGE 1

if ((ch[j] >= 'a') && (ch[j] <= 'z')) {

ch[j] -= 32;

}

else if ((ch[j] >= 'A') && (ch[j] <= 'Z'))

{

ch[j] += 32;

}

#endif

printf("%c", ch[j]);

}

printf("%c", ch[j]);

}

return 0;

}

3）测试

1. 测试数据：

选取两组数据，第一组输入asdsafa6321()@Q$!@$，第二组输入sdaa asdf.23+.#` （最左端有一个空格）如表4-3所示

表4-3 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 |
| 用例1 | asdsafa6321()@Q$!@$ | ASDSAFA6321()@q$!@$ | 图4-14 |
| 用例2 | sdaa asdf.23+.#` （最左端有一个空格） | sdaa asdf.23+.#` （最左端有一个空格） | 图4-15 |

1. 对应测试数据的运行结果截图：



图4-14 编程题2的测试结果1



图4-15 编程题2的测试结果2

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

4.4 实验小结

（1）改错题旨在帮助我们理解宏定义的“替换”，在使用宏定义是要重复了解“替换”的意思，注意括号的使用。

（2）替换题则是提醒我们注意函数声明的重要性，调用未声明的函数可能产生无法预料的结果。

（3）调试题让我们初步接触了assert函数，并需要了解声明它的头文件。

（4）这次的编程题难度不大，只要充分理解了宏定义的“替换”性质，不会有太大问题。

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

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。

#include<stdio.h>

int main(void)

{

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

void sort(int [],int);

int i;

sort(a[0],10);

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

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

printf("\n");

return 0;

}

void sort(int b[], int n)

{

int i, j, t;

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

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

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

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

}

解答：

1. 错误修改：

1）第2行：

int main(void)

错误原因：

sort函数未声明，应在main函数之前进行声明。

正确形式为：void sort(int b[], int n);

int main(void)

2）第7行：sort(a[0], 10);

错误原因：应将数组首地址而不是其第一个元素的值作为实参传递给sort函数

正确形式为：sort(a, 10);

1. 错误修改后运行结果：

此题无输入数据，直接运行并查看运行结果，若输出结果为此十个数从大到小有序排列，则程序修改无误。测试结果如图5-1所示。

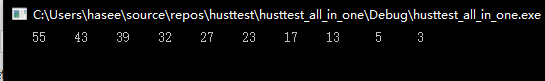


图5-1 修改题的测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

5.2.2 源程序修改替换

(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。

#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;

}

(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。

**解答：（1）**

1）解题思路：

前两个空是对边界条件的判断，因为j为0时表示的实际是最后一个人，不能直接使用数组的第j-1个元素，要进行判断。

第二个空是把数组a从移除元素后的元素依次向前靠拢，实现对数组a的压缩。

2）源程序清单

#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;

}

3）测试

1. 测试数据：

此题无输入数据，直接运行并查看运行结果，纸面分析得，结果应该为3，6，9，2，7，1，8，5，10，4。测试结果如图5-2所示。

1. 运行结果截图：

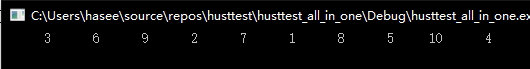


图5-2 替换题1的测试结果

**说明运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**

1）解题思路：

将数组元素的值设置为0与1，用下标表示编号，对于过程中遇到编号为0的元素则将其跳过，不改变计数器的值。

2）源程序清单

#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;

a[0] = 10;

for (i = M, j = 0; i > 0; 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 ((a[++j]) == 0) {

--k;

continue;

}

if (j > 9) {

j = 0;

}

}

/\* 最后一个人报数后第一个接着报，形成一个圈 \*/

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

if (i != 1)

a[j] = 0;

}

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

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

return 0;

}

3）测试

测试数据及结果与替换题1完全相同，**说明运行结果与理论分析吻合，验证了程序的正确性。**

5.2.3 跟踪调试

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？

(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。

**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++]) ;

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

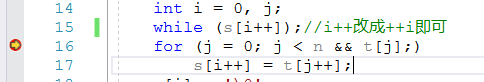
s[i++] = t[j++];

s[i] = '\0';

}

解答：

**（1）光条落在for语句所在行时i的值为23，如图5-3-2所示。**



**图5-3-1 调试题的运行截图1**



**图5-3-2 调试题的运行截图2**

**（2）光条落在strncat函数块结束标记所在行时，s的值为"The adopted symbol is "，t的值为"abcdefghijklmnopqrstuvwxyz"。如图5-3-3所示。**



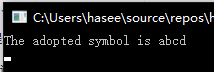
**图5-3-3 调试题的运行截图3**

**（2）修改：**

**由于i++本身的值为i未自增时的值，因此用第13行的i++会导致在遇到s字符串的结尾后i又向后移动了一步，使得'\0'没有被覆盖掉**

**故需将第13行的while(s[i++]);改为while(s[++i]);**

**修改完成后重新运行，理论结果应输出The adopted symbol is abcd，测试结果如图5-4所示。**



**图5-4 调试题的运行截图4**

**说明运行结果与理论分析吻合，验证了程序的正确性。**

5.2.4 程序设计

**（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。

解答：

1）解题思路：

程序流程如图5-5所示。



图5-5 编程题1的程序流程图

2）源程序清单

#include<stdio.h>

#define R 3

#define C 4

int main(void) {

int row, col;

int\* outp;

int ori[R][C], tar[C][R];

printf("Please input the original matrix:\n");

for (row = 0; row < R; row++) {

for (col = 0; col < C; col++) {

scanf("%d", &ori[row][col]);

tar[col][row] = ori[row][col];

}

}

printf("The original matrix is:\n")

for (row = 0; row < R; row++) {

for (col = 0; col < C; col++) {

printf("%-8d", ori[row][col]);

}

printf("\n");

}

printf("The transpose matrix is:\n")

for (col = 0; col < C; col ++) {

for (row = 0; row < R; row++) {

printf("%-8d ", tar[col][row]);

}

printf("\n");

}

return 0;

}

3）测试

1. 测试数据：

输入从-6至5的整数矩阵，理论输出应为格式化后的原矩阵及其转置矩阵，测试结果如图5-6所示。

1. 对应测试数据的运行结果截图：

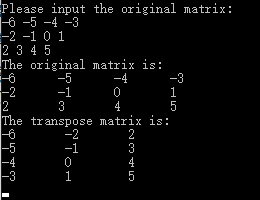


图5-6 编程题1的测试结果

**说明上述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。

解答：

1）解题思路：

程序流程如图5-7所示。



图5-7 编程题2的程序流程图

2）源程序清单

#include<stdio.h>

int main(void) {

int i;

int num;

unsigned int mask = 0x01 << (sizeof(int) \* 8 - 1);

int bitarray[sizeof(int) \* 8] = { 0 };

printf("Please input the number:\n");

scanf("%d", &num);

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

bitarray[i] = ((unsigned int)num & mask) >> (sizeof(int) \* 8 - 1);

num <<= 1;

}

printf("The bit type of this number is:\n");

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

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

}

printf("\n");

return 0;

}

3）测试

1. 测试数据：

选取一组边界条件，输入-1，应输出与数据类型的总bit数相同的1（此处为32个），一组普通数据，输入345234123 如表5-1所示

表5-1 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 |
| 用例1 | -1 | 11111111111111111111111111111111 | 图5-8 |
| 用例2 | 345234123 | 00010100100100111101101011001011 | 图5-9 |

1. 对应测试数据的运行结果截图：

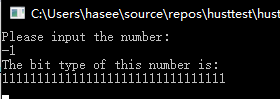


图5-8 编程题2的测试结果1

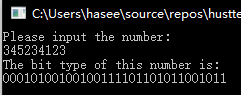


图5-9 编程题2的测试结果2

**说明上述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, 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!”。

解答：

1）解题思路：

程序流程如图5-10所示。



图5-10 编程题3的程序流程图

2）源程序清单

#include<stdio.h>

#include<malloc.h>

typedef struct {

char name[50];

int score;

} student;

void bisearch(int tar, student\* stu, int n);

int main(void) {

int n;

int tarscore;

int i = 0, j = 0;

printf("Please input the number of students:\n");

scanf("%d", &n);

student \*stu = (student\*)malloc(sizeof(student)\*n), temp;

printf("Please input the info of students:\n");

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

printf("Name %d:\n", i + 1);

scanf("%s", &((stu + i)->name));

printf("Score %d:\n", i + 1);

scanf("%d", &((stu + i)->score));

}

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

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

if ((stu + j)->score < (stu + j + 1)->score) {

temp = \*(stu + j);

\*(stu + j) = \*(stu + j + 1);

\*(stu + j + 1) = temp;

}

}

}

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

printf("Name %d: %s\n", i + 1, ((stu + i)->name));

printf("Score %d: %d\n", i + 1, ((stu + i)->score));

}

printf("Please input the score to search:\n");

scanf("%d", &tarscore);

bisearch(tarscore, stu, n);

return 0;

}

void bisearch(int tar, student\* stu, int n) {

int left = 0, right = n - 1, mid = (0 + n) / 2;

int i;

while (left <= right) {

mid = (left + right) / 2;

if ((stu + mid)->score < tar)

{

right = mid - 1;

}

else if ((stu + mid)->score > tar)

{

left = mid + 1;

}

else

{

int now;

int flag1 = 0, flag2 = 0;

printf("These are the students with the score of %d :\n", tar);

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

now = mid + i;

if ((stu + now)->score == tar && (!flag1)) {

printf("Name: %s Score: %d\n", (stu + now)->name, (stu + now)->score);

}

else {

flag1 = 1;

}

now = mid - i;

if ((stu + now)->score == tar && (!flag2)) {

if (i != 0) {

printf("Name: %s Score: %d\n", (stu + now)->name, (stu + now)->score);

}

}

else {

flag2 = 1;

}

if (flag1&&flag2)break;

}

return;

}

}

printf("not found\n");

}

3）测试

1. 测试数据：

选取两组数据，第一组输入asdsafa6321()@Q$!@$，第二组输入sdaa asdf.23+.#` （最左端有一个空格）如表5-2所示

表5-2 编程题3的测试数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | | 理 论 结 果 | | 运 行 结 果 |
| 排序后 | |
| 学生  （num:8） | 姓名 | 分数 | 姓名 | 分数 |
| 学生1 | why | 59 | wjy | 66 | 图5-11 |
| 学生2 | lyt | 60 | lyt | 66 |
| 学生3 | xy | 61 | zxl | 64 |
| 学生4 | cyx | 62 | yzx | 63 |
| 学生5 | yzx | 63 | cyx | 62 | 图5-12 |
| 学生6 | zxl | 64 | xy | 61 |
| 学生7 | wjy | 66 | lyt | 60 |
| 学生8 | cjy | 66 | why | 59 |
| 查找分数 | 66 | | wjy | 66 | 图5-13 |
| cjy | 66 |

1. 对应测试数据的运行结果截图：

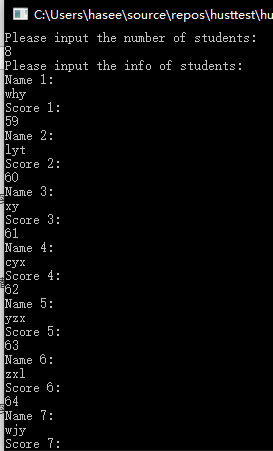


图5-11 编程题3的测试结果1

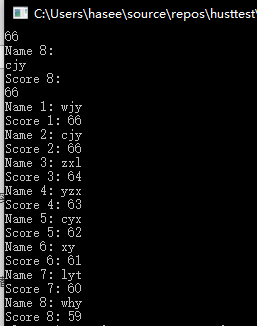


图5-12 编程题3的测试结果2

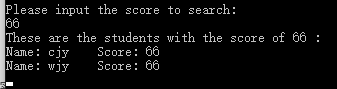


图5-13 编程题3的测试结果3

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

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中字符串的第n个字符的后面，编写main测试strnins的正确性。

解答：

1）解题思路：

程序流程如图5-14所示。



图5-14 选做题1的程序流程图

2）源程序清单

#include<stdio.h>

#include<malloc.h>

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

int main(void)

{

char a[60] = "The adopted symbol is ", b[27] = "abcdefghijklmnopqrstuvwxyz";

strnins(a, b, 4);

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

return 0;

}

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

{

int i = 0, j = 0;

int lent, lens;

while (t[++i]);

lent = i;

i = 0;

while (s[++i]);

lens = i;

i = 0;

char\* buf = (char\*)malloc(sizeof(char)\*(lens - n));

for (i = n, j = 0; i < lens; i++, j++) {

buf[j] = s[i];

}

for (i = n, j = 0; i < lent + n; i++, j++) {

s[i] = t[j];

}

for (i = lent + n, j = 0; i < lens + lent - 1; i++, j++) {

s[i] = buf[j];

}

s[i] = '\0';

}

3）测试

1. 测试数据：

采用调试题中的字符数组a与b作为传递给strnins的实参，数组b插入数组a的第四个元素之后，理论结果应为The abcdefghijklmnopqrstuvwxyzadopted symbol is ，测试结果如图5-15所示

1. 对应测试数据的运行结果截图：

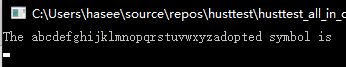


图5-15 选做题1的测试结果

**说明上述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。

解答：

1）解题思路：

用一维数组存放每一行中棋子所在的列，用循环穷举所有棋盘，每次循环更新当前棋子位置后进行判断是否出现不符合条件的棋子对，若出现则此棋盘无效，若未出现则输出此棋盘并令计数器加一，遍历完成后输出符合的情况总数即可。

2）源程序清单

#include <stdio.h>

void execute(int\* sum);

int main(void)

{

int count = 0;

execute(&count);

printf("There are %d possible ways\n", count);

return 0;

}

void execute(int\* sum)

{

int i, j, flag;

int pos[9] = { 0 };

while (pos[8] != 1)

{

++pos[0];

//穷举,j表示第i行的棋子放在哪一个位置

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

{

if (pos[i] == 8)

{

pos[i] = 0;

++pos[i + 1];

}

}

flag = 1;

//判断

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

{

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

{

if (i != j)

{

if (pos[i] == pos[j]) {

flag = 0;

break;

}

else if ((pos[i] - pos[j]) == (i - j) || (pos[i] - pos[j]) == (j - i)) {

flag = 0;

break;

}

}

}

if (!flag)break;

}

//输出

if (flag)

{

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

{

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

{

if (pos[i] == j)

printf("1 ");

else

printf("0 ");

}

printf("\n");

}

printf("\n");

}

\*sum += flag;

}

}

3）测试

1. 测试数据：

此题无输入，直接运行，理论应输出所有可能的棋盘以及可能情况的总数，测试结果如图5-16所示（总棋盘数过多，此处只列出部分棋盘）。

1. 对应测试数据的运行结果截图：

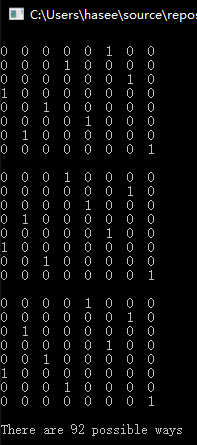


图5-16 编程题2的测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

5.4 实验小结

（1）改错题再次提醒了我们函数声明的重要性，同时考察了同学们对数组名的理解。

（2）替换题是对算法优化以及边界条件处理的综合考察，旨在培养同学们严谨认真的品质以及对完美的追求。

（3）调试题依然是对细节的考察，平时写程序时稍不注意就可能导致此类错误，导致花费大量时间去寻找错误地点。要尽量在写程序的时候就考虑周到。

（4）编程题是对数组知识点的综合能力考察，只要对数组理解透彻便很容易完成。

（5）选做题的第一题类似于调试题，要注意插入位置必须十分精确，稍有偏差就会导致严重错误。第二题则是一道经典算法题目，在计算过程中记得对于不符合情况的棋盘及时break，否则会大大延长运行时间。

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

下面的源程序中是否存在错误？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源程序进行调试修改，使之能够正确执行。

#include<stdio.h>

int main(void)

{

float \*p;

scanf("%f",p);

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

return 0;

}

}

解答：

1. 错误修改：

第4行：

float \*p;

错误原因：

指针p未初始化，必须先使其指向一个float类型变量才能利用p表示一个float类型变量的地址

正确形式为：float f;

float \*p = &f;

1. 错误修改后运行结果：

输入一个小数，理论运行效果为输出其值。测试结果如图6-1所示。

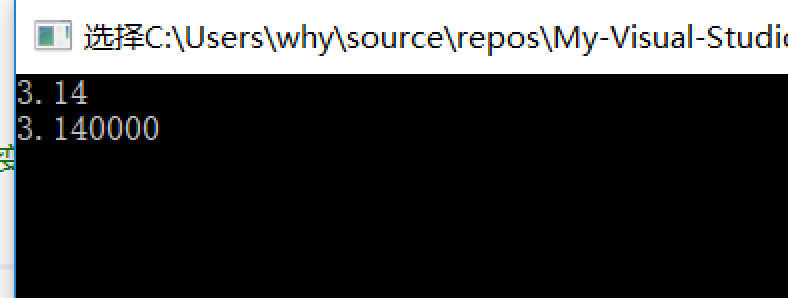


图6-1 修改题的测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

6.2.2 源程序修改替换

**2．源程序完善、修改、替换**

(1) 下面源程序的功能是,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。请在程序中的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>

int 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:

return 0;

}

(2) 请上机运行第（1）题程序，使之能按要求输出下面结果（注：（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）**

1）解题思路：

定义一个函数指针p，通过输入来判断将其赋为strcpy还是strcat，而在读取字符串时，通过getchar而非scanf(“%s”,…)来读取，从而正确地读取含有空格的字符串

2）源程序清单

#include<stdio.h>

#include <string.h>

int main(void)

{

char \*(\*p)(char \*, const char \*);//填空1

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;

for (; (a[i] = getchar()) != '\n'; i++); a[i] = '\0';//填空2

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

i = 0;

for (; (b[i] = getchar()) != '\n'; i++); b[i] = '\0';//填空3

result = p(a, b);

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

down:

return 0;

}

3）测试

1. 测试数据：

直接采用替换题2的测试数据及理论结果，实际运行结果如图6-2所示。

1. 运行结果截图：

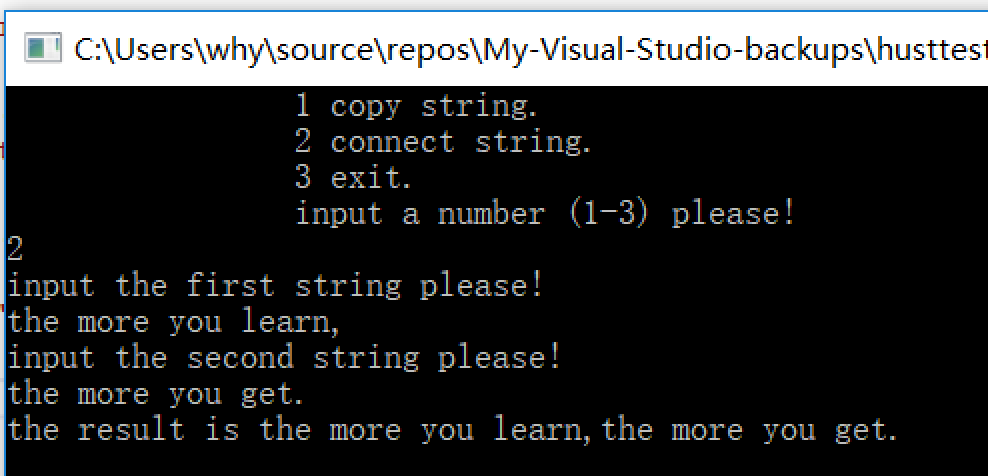


图6-2 替换题1的测试结果

**说明运行结果与理论分析吻合，验证了程序的正确性。**

**（2）**

此题已在替换题1中一并完成。

6.2.3 跟踪调试

请按下面的要求对所给源程序进行操作，并回答问题和排除错误。

(1) 单步执行源程序。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) 排除源程序中的错误，使程序输出结果为： there is a boat on the lake.

**源程序：**

#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));

return 0;

}

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

{

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

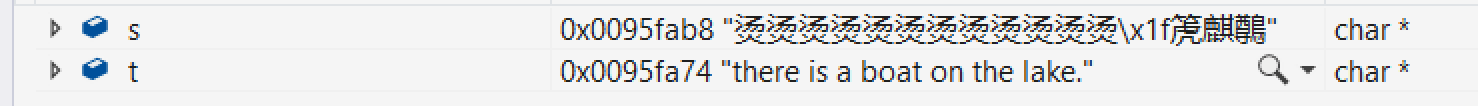
;

return (s);

}

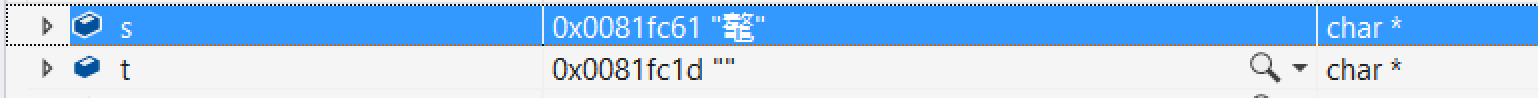
解答：

**（1）刚进入strcpy时，s的值是未被赋值的随机值如图6-3-1所示。**

****

**图6-3-1 调试题的运行截图1**

**返回main时，swatch窗口中s是未被赋值的随机值，如图6-3-2所示。**

****

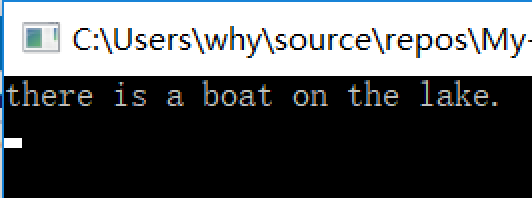
**图6-3-2 调试题的运行截图2**

**（2）修改：**

**由于在strcpy函数中，s所指的地址是不断改变的，因此在函数返回时，s实际上已经指向了字符串的结束符所在地址之后的一个地址，而非字符串首地址，故导致无法正确输出字符串。**

**故需在第12行前定义一个缓冲指针char \*buf，将其指向s所指的地址，并在14行返回buf。**

**修改完成后重新运行，理论结果应输出there is a boat on the lake.，测试结果如图6-4所示。**

****

**图6-4 调试题的运行截图4**

**说明运行结果与理论分析吻合，验证了程序的正确性。**

6.2.4 程序设计

**（1）**已知一个长整型变量占4个字节，其中每个字节又分成高4位和低4位。试编写一个程序,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位并以十六进制数字字符的形式进行显示。

解答：

1）解题思路：

程序流程如图6-5所示。



图6-5 编程题1的程序流程图

2）源程序清单

#include<stdio.h>

int main(void) {

long num;

int i;

unsigned long maskh = 0xf0, maskl = 0x0f;

printf("Please input a number:\n");

scanf("%ld", &num);

for (i = sizeof(long) - 1; i >= 0; i--) {

printf("The higher bits of byte %d is %x\t", i + 1, ((num >> i \* 8) & maskh) >> 4);

printf("The lower bits of byte %d is %x\n", i + 1, (num >> i \* 8) & maskl);

}

return 0;

}

3）测试

1. 测试数据：

输入一个较大数字123980981，利用计算器算出其十六进制表示为0763ccb5，测试结果如图6-6所示。

1. 对应测试数据的运行结果截图：

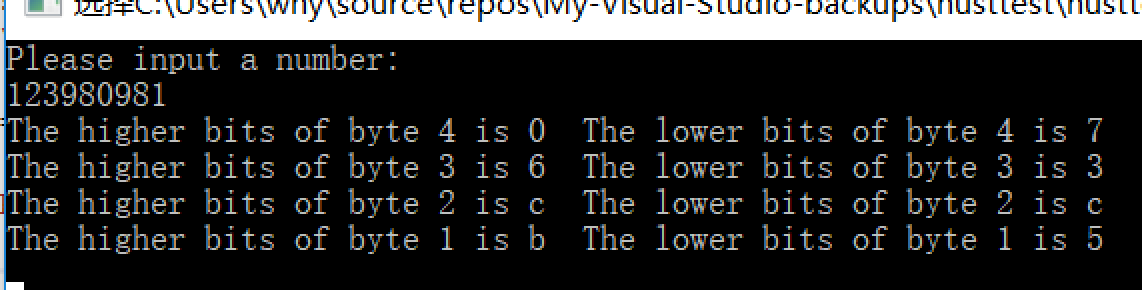


图6-6 编程题1的测试结果

**说明上述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。试编写一个函数，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：

程序流程如图6-7所示。



图6-7 编程题2的程序流程图

2）源程序清单

#include<stdio.h>

#include<malloc.h>

void execute(char \*ori);

int main(void) {

int n;

int i;

printf("Please input the number of lines:\n");

scanf("%d", &n);

getchar();

printf("Start inputing:\n");

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

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

buf[i]=(char\*)malloc(sizeof(char) \* 80);

}

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

gets(buf[i]);

}

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

execute(buf[i]);

}

return 0;

}

void execute(char \*ori) {

int i = 0, j = 0, t = 0;

int dec = 0;

int flag = 0;

char\* buf = ori;

while (ori[i++] != '\0');

for (j = 0; buf[j] != '\0'; j++) {

if (buf[j] == ' ') {

if (j == 0);

else if (buf[j - 1] == ' ') {

for (t = j; t < i - dec; t++) {

buf[t] = buf[t + 1];

}

++dec;

--j;

}

}

else {

flag = 1;

}

}

if (flag) {

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

printf("%c", buf[t]);

}

printf("\n");

}

}

3）测试

1. 测试数据：

输入6行，其中一行以多个空格开头，一行以单个空格开头，中间两行一行直接留空，一行输入多个空格，每行的中间都既含有多个空格也含有单个空格，具体数据如表6-1所示。

表6-1 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 |
| 行1 | • • •saer • •sdklj •ojojl | •saer •sdklj •ojojl | 图6-8 |
| 行2 | •test •test • • •test | •test •test •test |
| 行3 | • • • | 无 |
| 行4 | 直接换行 | 无 |
| 行5 | test • •test •test • • •test | test •test •test •test |
| 行6 | test • • •test •test • • t | test •test •test •t |

1. 对应测试数据的运行结果截图：

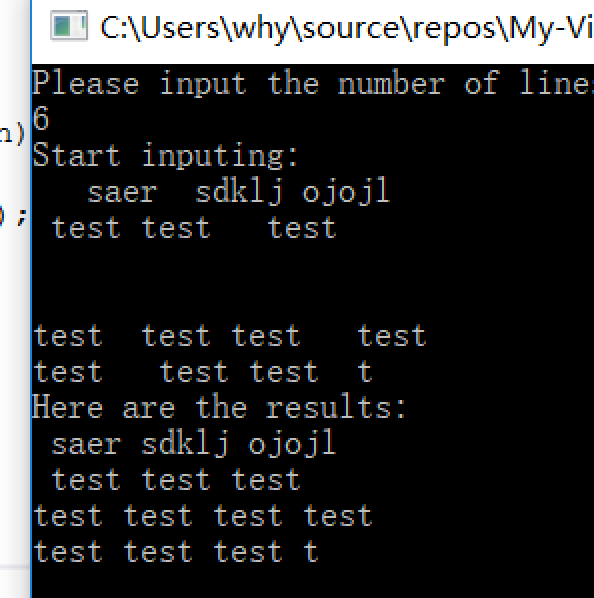


图6-8 编程题2的测试结果1

**说明上述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，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：

程序流程如图6-9所示。



图6-9编程题3的程序流程图

2）源程序清单

#include<stdio.h>

#include<malloc.h>

#include<string.h>

void upper(int\* raw, int n);

void lower(int\* raw, int n);

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

int n;

int i;

printf("Please the number of inputs:\n");

scanf("%d", &n);

printf("Start inputing:\n");

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

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

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

}

void (\*p)(int\*, int);

if (argc == 1) {

p = upper;

}

else {

if (strcmp(argv[1], "-d") == 0){ //这不是Py！！！不能直接用等号比较字符串

p = lower;

}

}

p(num, n);

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

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

}

printf("\n");

return 0;

}

void upper(int\* raw, int n) {

int i = 0, j = 0;

int buf = 0;

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

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

if (raw[j] > raw[j + 1]) {

buf = raw[j + 1];

raw[j + 1] = raw[j];

raw[j] = buf;

}

}

}

}

void lower(int\* raw, int n) {

int i = 0, j = 0;

int buf = 0;

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

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

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

buf = raw[j + 1];

raw[j + 1] = raw[j];

raw[j] = buf;

}

}

}

}

3）测试

1. 测试数据：

选取两组数据，第一组不添加命令行参数，第二组添加命令行参数-d，两组的待排序数据均为12，20，-2，123，87，28989471，-971，0，90，89。理论结果为：第一组输出-971，-2，0，12，20，87，89，90，123，28989471。第二组输出28989471，123，90，89，87，20，12，0，-2，-971。实际运行结果如图6-10所示。

1. 对应测试数据的运行结果截图：

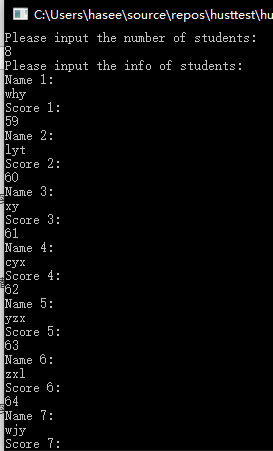


图6-10-1 编程题3的第一组测试结果

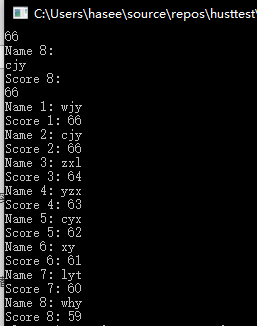


图6-10-2 编程题3的第二组测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

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中字符串的第n个字符的后面，编写main测试strnins的正确性。

解答：

1）解题思路：

程序流程如图5-14所示。



图5-14 选做题1的程序流程图

2）源程序清单

#include<stdio.h>

#include<malloc.h>

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

int main(void)

{

char a[60] = "The adopted symbol is ", b[27] = "abcdefghijklmnopqrstuvwxyz";

strnins(a, b, 4);

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

return 0;

}

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

{

int i = 0, j = 0;

int lent, lens;

while (t[++i]);

lent = i;

i = 0;

while (s[++i]);

lens = i;

i = 0;

char\* buf = (char\*)malloc(sizeof(char)\*(lens - n));

for (i = n, j = 0; i < lens; i++, j++) {

buf[j] = s[i];

}

for (i = n, j = 0; i < lent + n; i++, j++) {

s[i] = t[j];

}

for (i = lent + n, j = 0; i < lens + lent - 1; i++, j++) {

s[i] = buf[j];

}

s[i] = '\0';

}

3）测试

1. 测试数据：

采用调试题中的字符数组a与b作为传递给strnins的实参，数组b插入数组a的第四个元素之后，理论结果应为The abcdefghijklmnopqrstuvwxyzadopted symbol is ，测试结果如图5-15所示

1. 对应测试数据的运行结果截图：

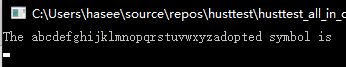


图5-15 选做题1的测试结果

**说明上述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。

解答：

1）解题思路：

用一维数组存放每一行中棋子所在的列，用循环穷举所有棋盘，每次循环更新当前棋子位置后进行判断是否出现不符合条件的棋子对，若出现则此棋盘无效，若未出现则输出此棋盘并令计数器加一，遍历完成后输出符合的情况总数即可。

2）源程序清单

#include <stdio.h>

void execute(int\* sum);

int main(void)

{

int count = 0;

execute(&count);

printf("There are %d possible ways\n", count);

return 0;

}

void execute(int\* sum)

{

int i, j, flag;

int pos[9] = { 0 };

while (pos[8] != 1)

{

++pos[0];

//穷举,j表示第i行的棋子放在哪一个位置

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

{

if (pos[i] == 8)

{

pos[i] = 0;

++pos[i + 1];

}

}

flag = 1;

//判断

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

{

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

{

if (i != j)

{

if (pos[i] == pos[j]) {

flag = 0;

break;

}

else if ((pos[i] - pos[j]) == (i - j) || (pos[i] - pos[j]) == (j - i)) {

flag = 0;

break;

}

}

}

if (!flag)break;

}

//输出

if (flag)

{

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

{

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

{

if (pos[i] == j)

printf("1 ");

else

printf("0 ");

}

printf("\n");

}

printf("\n");

}

\*sum += flag;

}

}

3）测试

1. 测试数据：

此题无输入，直接运行，理论应输出所有可能的棋盘以及可能情况的总数，测试结果如图5-16所示（总棋盘数过多，此处只列出部分棋盘）。

1. 对应测试数据的运行结果截图：

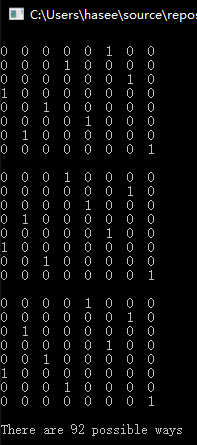


图5-16 编程题2的测试结果

**说明上述运行结果与理论分析吻合，验证了程序的正确性。**

5.4 实验小结

（1）改错题再次提醒了我们函数声明的重要性，同时考察了同学们对数组名的理解。

（2）替换题是对算法优化以及边界条件处理的综合考察，旨在培养同学们严谨认真的品质以及对完美的追求。

（3）调试题依然是对细节的考察，平时写程序时稍不注意就可能导致此类错误，导致花费大量时间去寻找错误地点。要尽量在写程序的时候就考虑周到。

（4）编程题是对数组知识点的综合能力考察，只要对数组理解透彻便很容易完成。

（5）选做题的第一题类似于调试题，要注意插入位置必须十分精确，稍有偏差就会导致严重错误。第二题则是一道经典算法题目，在计算过程中记得对于不符合情况的棋盘及时break，否则会大大延长运行时间。

附录1 第一章自设题部分内容

注：附录中所有测试都是基于完整项目进行的，如需检测请对后续附录中的项目整合包进行编译。

1.3.1 reading.c

源程序清单

·int readinput(char \*input, char \*buf):

//函数功能：储存输入的表达式(buf)用于格式化输出，并建立一个处理了负号的版本(input)以进行后续计算，同时处理特殊输入指令

int readinput(char \*input, char \*buf) {//input进行后续计算,buf用于格式化输出

int i = 0,j=0;

int minus = 0;

while (((input[i] = fgetc(stdin))) != '\n')

{

buf[j] = input[i];

if (input[i] == '-') {//非常神秘的没有验证正确性的自己脑子一抽想出来的负数的暴力处理办法

if (i == 0) {//负号在首位时直接添0

input[i++] = '0';

input[i] = '-';

}

else if (input[i - 1] == '(') {//负号在左括号后时直接添0

/\*\*\*\*\*\*\*这个不能和上面的合并，否则读取input[-1]会导致内存错误\*\*\*\*\*\*/

input[i++] = '0';

input[i] = '-';

}

else{//负号在数字的后面时添置括号

input[i++] = '+';

input[i++] = '(';

input[i++] = '0';

input[i] = '-';

++minus;

}

}

if (input[i] == '+') {//顺便把正号一起处理了

if (i == 0) {//首位添0

input[i++] = '0';

input[i] = '+';

}

else if (input[i - 1] == '(') {//左括号后添0

input[i++] = '0';

input[i] = '+';

}

}if (input[i] == '.'&&!(input[i-1]>='0'&&input[i-1]<='9')) {//当然还有小数点

input[i++] = '0';

input[i] = '.';//任何情况下没接在数字后面的小数点只要直接变成0.就行了

}

if (input[i] == ' ')

{

i--;

j--;

}

i++;

j++;

}

buf[j] = '\0';

for (int k = 1; k <= minus; k++) {//补全由于处理负数产生的括号

input[i++] = ')';

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*错误8：无输入\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (input[0]=='\n') {

return errorfound(8);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ERROR8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

input[i] = '\0';

//特殊命令：

//1.exit（或quit）退出程序

//2.help 打开帮助界面

//3.change 修改精度

if (strcmp(input, "quit")==0 || strcmp(input, "exit")==0) {

strcpy(input, "");

return 0;

}

else if (strcmp(input, "help") == 0) {

outputhelp();

return 2;

}

else if (strcmp(input, "change") == 0) {

preciquest();

return 2;

}

else {

return 1;

}

}

测试

此部分的测试在调试模式下进行：

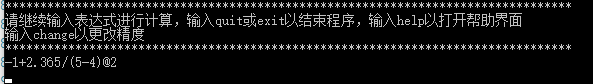


图1-1-1 input函数转换功能的测试结果1

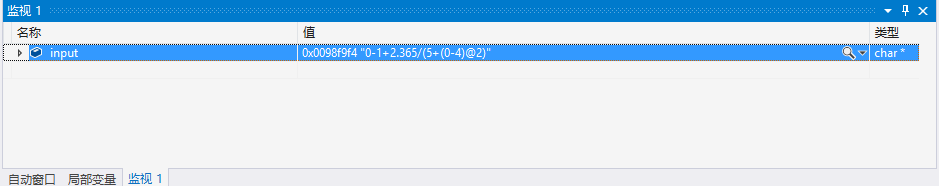


图1-1 input函数转换功能的测试结果2

说明input函数能按要求将输入的字符串格式化并储存，验证了其功能。

1.3.2 output.c

源程序清单

·int preciquest(void):

//请求输入精度

int preciquest(void) {

char foo;

printf("输入的精度要求为一个大于等于0，小于等于15的整数\n");

printf("请输入精度\n");

/\*那么为什么下面那种方法可以，而这样清空并关闭的话在连续输入两次2+.23之类的表达式的时候仍有问题呢？\*/

//fflush(stdin);

int right = scanf("%d", &preci);//这里会出现输入残留问题么？

scanf("%c", &foo);//当然会的啊

if (right&&preci >= 0 && preci <= 15&&foo=='\n') {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("精度设置成功，当前精度为%d\n", preci);

return 1;

}

else {

printf("精度不符合要求，请重新输入\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

//setbuf(stdin, NULL);//这一句是与上面的fflush函数对应使用的

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*可以这样清除缓冲区（而非关闭缓冲区）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

char\*buf = (char \*)malloc(BUFSIZ);

free(buf);

buf = (char \*)malloc(BUFSIZ);

setbuf(stdin,buf);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

return preciquest();

}

}

·void baseoutput(void):

void baseoutput(void) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* 简易计算器V1.1.2 \*\n");

printf("\*此计算器可实现基本的四则运算以及部分的乘方/开方运算 \*\n");

printf("\*第一次使用请输入 help 以查看具体操作方式及输入要求 \*\n");

printf("\*输入一个合法的算数表达式后会在下一行输出算式及结果，并可继续输入新的表达式 \*\n");

printf("\*退出程序请输入quit或exit \*\n");

printf("\*更改精度请输入change \*\n");

printf("\*高精度下，输出结果的最后1-3位可能丢失精度，请以四舍五入的方式截取有效部分 \*\n");

printf("\* -by why\*\n");

printf("\* 2017/10/26 16:01\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

preciquest();

endingoutput();

}

·void endingoutput(void):

void endingoutput(void) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请继续输入表达式进行计算，输入quit或exit以结束程序，输入help以打开帮助界面\n");

printf("输入change以更改精度\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

·void outputresult(double result):

//蠢蠢的分精度输出

void outputresult(double result) {//result为计算的结果

int intlen = 0;

switch (preci) {

case 0:

printf("%.0f\n", result);

break;

case 1:

printf("%.1f\n", result);

break;

case 2:

printf("%.2f\n", result);

break;

case 3:

printf("%.3f\n", result);

break;

case 4:

printf("%.4f\n", result);

break;

case 5:

printf("%.5f\n", result);

break;

case 6:

printf("%.6f\n", result);

break;

case 7:

printf("%.7f\n", result);

break;

case 8:

printf("%.8f\n", result);

break;

case 9:

printf("%.9f\n", result);

break;

case 10:

printf("%.10f\n", result);

break;

case 11:

printf("%.11f\n", result);

break;

case 12:

printf("%.12f\n", result);

break;

case 13:

printf("%.13f\n", result);

break;

case 14:

printf("%.14f\n", result);

break;

case 15:

printf("%.15f\n", result);

break;

}

}

·void outputhelp(void):

//打印帮助界面

void outputhelp(void) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*请务必使用全英文输入\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("输入帮助：\n");

printf("四则运算请直接输入表达式，加：+，减：-，乘：\*，除：/，输入范例：1+2\n");

printf("乘方运算符：^，其左侧为底数，右侧为指数，输入范例：2^5\n");

printf("开方运算符：@，其左侧为被开方数，右侧为根次数，输入范例：5@2\n");

printf("负数：请勿在运算符的后面紧跟负号，应按照数学运算式的书写规范进行输入\n");

printf("括号：()请务必使用英文括号，用法与数学上相同\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

·void endingoutput(void):

void endingoutput(void) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请继续输入表达式进行计算，输入quit或exit以结束程序，输入help以打开帮助界面\n");

printf("输入change以更改精度\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

测试

此部分的测试需在输入某些满足要求的语句来运行对应输出函数，具体输入见图。

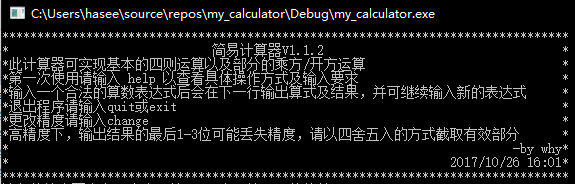


图1-3 baseoutput函数的输出

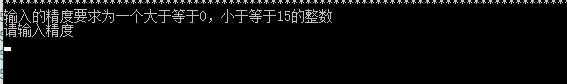


图1-4-1 preciquest及endingoutput函数的输出1

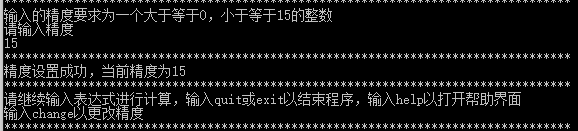


图1-4-2 preciquest及endingoutput函数的输出2

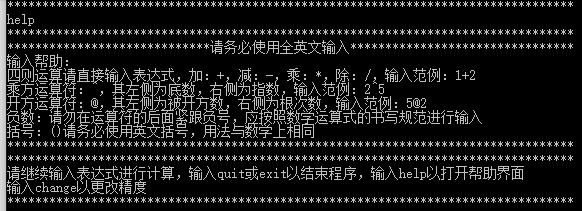


图1-5 outputhelp函数的输出

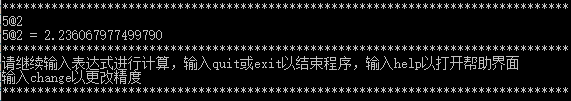


图1-6 outputresult函数的输出

说明上述函数的输出均达到要求，可以正确表达所需意思。

附录2 第二章自设题部分内容

2.3 output.c

源程序清单

·int preciquest(void):

extern int execstatus;//"why\_calculator.c"

extern int preci;//"why\_calculator.c"

//请求输入精度

int preciquest(void) {

char foo;

printf("输入的精度要求为一个大于等于0，小于等于15的整数\n");

printf("请输入精度\n");

/\*那么为什么下面那种方法可以，而这样清空并关闭的话在连续输入两次2+.23之类的表达式的时候仍有问题呢？\*/

//fflush(stdin);

int right = scanf("%d", &preci);//这里会出现输入残留问题么？

scanf("%c", &foo);//当然会的啊

if (right&&preci >= 0 && preci <= 15&&foo=='\n') {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("精度设置成功，当前精度为%d\n", preci);

return 1;

}

else {

printf("精度不符合要求，请重新输入\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

//setbuf(stdin, NULL);//这一句是与上面的fflush函数对应使用的

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*可以这样清除缓冲区（而非关闭缓冲区）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

char\*buf = (char \*)malloc(BUFSIZ);

free(buf);

buf = (char \*)malloc(BUFSIZ);

setbuf(stdin,buf);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

return preciquest();

}

}

·void outputresult(double result):

//蠢蠢的分精度输出

void outputresult(double result) {//result为计算的结果

int intlen = 0;

switch (preci) {

case 0:

printf("%.0f\n", result);

break;

case 1:

printf("%.1f\n", result);

break;

case 2:

printf("%.2f\n", result);

break;

case 3:

printf("%.3f\n", result);

break;

case 4:

printf("%.4f\n", result);

break;

case 5:

printf("%.5f\n", result);

break;

case 6:

printf("%.6f\n", result);

break;

case 7:

printf("%.7f\n", result);

break;

case 8:

printf("%.8f\n", result);

break;

case 9:

printf("%.9f\n", result);

break;

case 10:

printf("%.10f\n", result);

break;

case 11:

printf("%.11f\n", result);

break;

case 12:

printf("%.12f\n", result);

break;

case 13:

printf("%.13f\n", result);

break;

case 14:

printf("%.14f\n", result);

break;

case 15:

printf("%.15f\n", result);

break;

}

}

测试

此部分的测试通过输入不同的精度，检测计算结果是否能正确保留对应位数来检测程序是否正确。具体输入见图

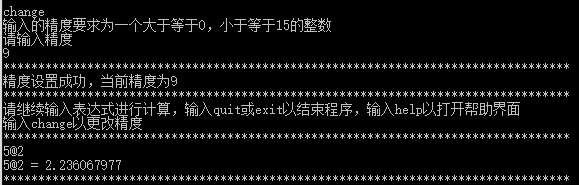


图2-1 分精度输出测试

说明上述函数的输出均达到要求，可以正确表达所需意思。