

**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](#_Toc500340330)

[1.1 实验目的 1](#_Toc500340331)

[1.2 实验内容 1](#_Toc500340332)

[1.3 自设题 11](#_Toc500340333)

[1.4 实验小结 12](#_Toc500340334)

[实验2 流程控制实验 14](#_Toc500340335)

[2.1 实验目的 14](#_Toc500340336)

[2.2 实验内容 14](#_Toc500340337)

[2.3 自设题 32](#_Toc500340338)

[2.4 实验小结 32](#_Toc500340339)

[实验3 函数与程序结构实验 33](#_Toc500340340)

[3.1 实验目的 33](#_Toc500340341)

[3.2 实验内容 33](#_Toc500340342)

[3.3 自设题 45](#_Toc500340343)

[3.4 实验小结 45](#_Toc500340344)

[附录1 第一章自设题部分内容 46](#_Toc500340345)

[1.3.1 reading.c 46](#_Toc500340346)

[1.3.2 output.c 48](#_Toc500340347)

[附录2 第二章自设题部分内容 55](#_Toc500340348)

[2.3 output.c 55](#_Toc500340349)

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。

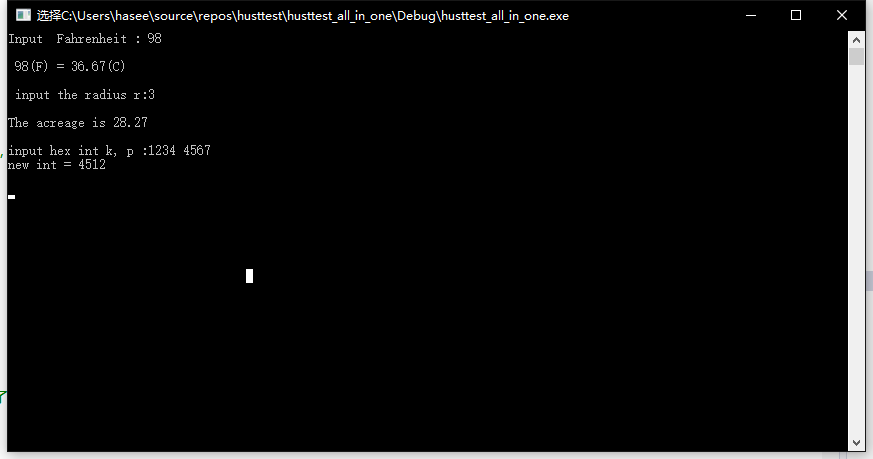


图1-1 改错题的测试结果

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。

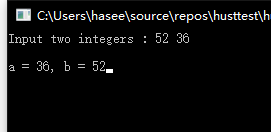


图1-3 替换题的测试结果

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：

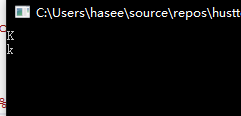


图1-5 编程题1的测试结果1

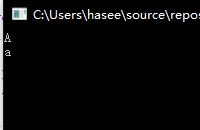
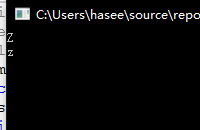
 

图1-6 编程题1的测试结果2

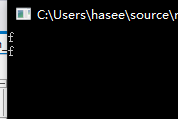


图1-7 编程题1的测试结果3

**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

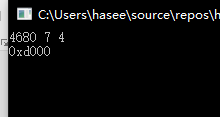


图1-9 编程题2的测试结果1

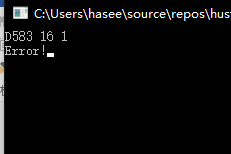


图1-10 编程题2的测试结果2

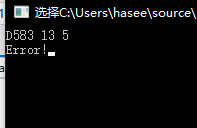


图1-11 编程题2的测试结果3

**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：

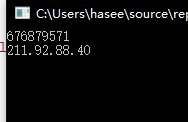


图1-11 编程题3的测试结果1

**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。

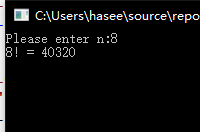


图2-1 改错题的测试结果1

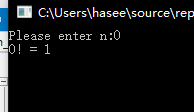


图2-2 改错题的测试结果2

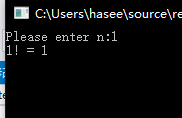


图2-3 改错题的测试结果3

**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：

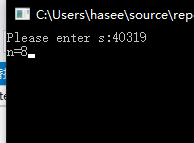


图2-7 替换题3的测试结果1

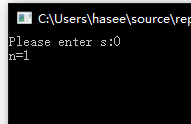


图2-8 替换题3的测试结果2

**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：

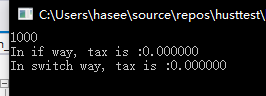


图2-11 编程题1的测试结果1

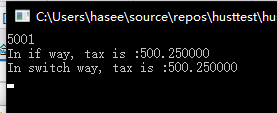


图2-12 编程题1的测试结果2

**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：

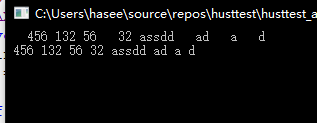


图2-14 编程题2的测试结果

**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：

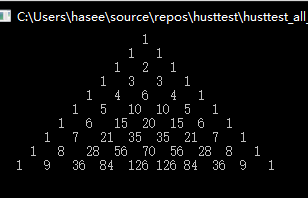


图2-16 编程题3的测试结果

**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：

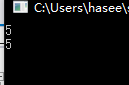


图2-18 编程题4的测试结果1

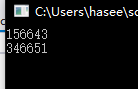


图2-19 编程题4的测试结果2

**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：

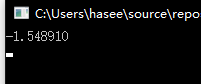


图2-20 选做题的测试结果

**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。

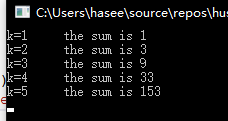


图3-1 修改题的测试结果

**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。

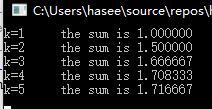


图3-2 替换题2的测试结果

**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

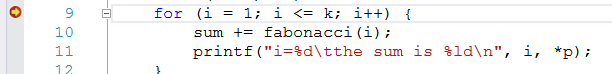
return fabonacci(n-1)+fabonacci(n-2);

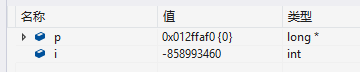
}

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。**





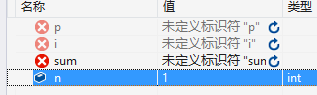
**图3-3 调试题的运行截图1**

**（2）停留在的语句如图3-4所示.**



**图3-4 调试题的运行截图2**

**（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3**

**（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 调试题的运行截图4**

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：

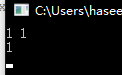


图3-8 编程题1的测试结果1

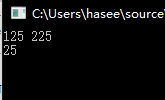


图3-9 编程题1的测试结果2

**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：

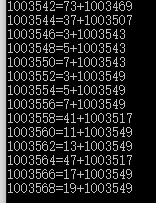
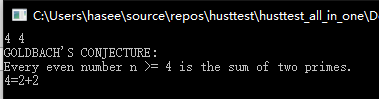


图3-11 编程题2的测试结果1



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。



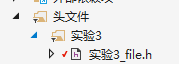


图3-13 选做题的运行截图1

（1）运行生成的.exe文件，结果如图3-14所示。

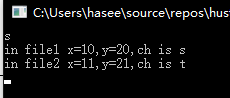


图3-14 选做题的运行截图2

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。

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 {

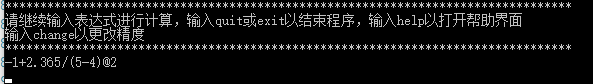
return 1;

}

}

测试

此部分的测试在调试模式下进行：



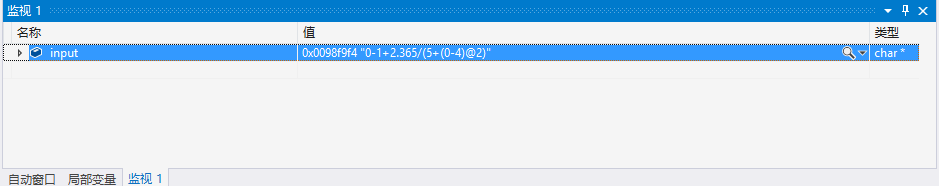


图1-1 input函数转换功能的测试结果

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。

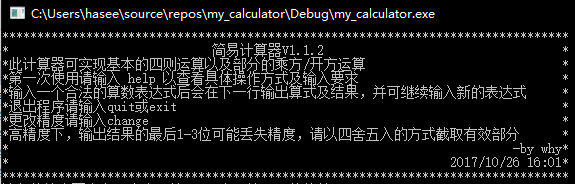


图1-3 baseoutput函数的输出

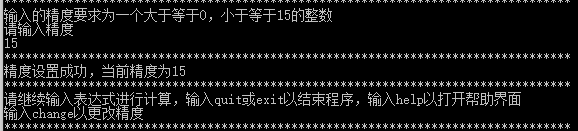
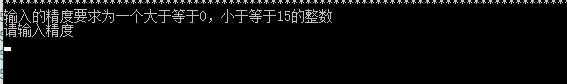


图1-4 preciquest及endingoutput函数的输出

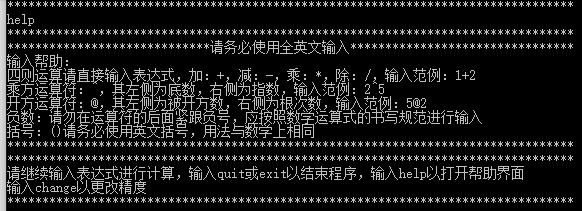


图1-5 outputhelp函数的输出

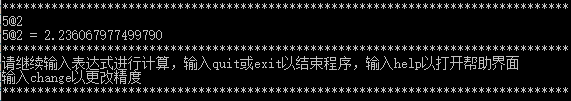


图1-6 outputresult函数的输出

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

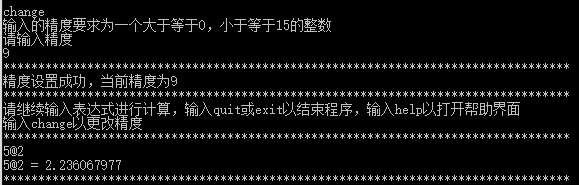


图2-1 分精度输出测试

说明上述函数的输出均达到要求，可以正确表达所需意思。