

**课 程 实 验 报 告**

**课程名称： C语言程序设计实验**

**专业班级： 信安2班**

**学 号： U201614836**

**姓 名： 林晓斌**

**指导教师： 祝 建 华**

**报告日期：**

**计算机科学与技术学院**

目 录

[1 表达式和标准输入与输出实验 2](#_Toc484993434)

[1.1 实验目的 2](#_Toc484993435)

[1.2 实验内容 3](#_Toc484993436)

[1.2.1 源程序改错 3](#_Toc484993437)

[1.2.2 源程序修改替换 5](#_Toc484993438)

[1.2.3程序设计 6](#_Toc484993439)

[1.3实验心得 10](#_Toc484993440)

[2 流程控制实验 11](#_Toc484993441)

[2.1 实验目的 11](#_Toc484993442)

[2.2 实验内容 11](#_Toc484993443)

[2.2.1源程序改错题 11](#_Toc484993444)

[2.2.2源程序修改替换题 12](#_Toc484993445)

[2.2.3编程设计题 16](#_Toc484993446)

[2.2.4 选做题牛顿近似法 22](#_Toc484993447)

[2.3实验心得 23](#_Toc484993448)

[3 函数与程序结构实验 24](#_Toc484993449)

[3.1实验目的 24](#_Toc484993450)

[3.2、实验内容 24](#_Toc484993451)

[3.2.1源程序改错题 24](#_Toc484993452)

[3.2.2源程序修改替换题 26](#_Toc484993453)

[3.3.3跟踪调试题 27](#_Toc484993454)

[3.3.4编程设计题 29](#_Toc484993455)

[3.3.5选做题 32](#_Toc484993456)

[3.3实验心得 33](#_Toc484993457)

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

在这个例子程序中存在若干语法和逻辑错误。要求参照1.3和1.4的步骤对下面程序进行调试修改，使之能够正确完成指定任务。

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；

1. #include<stdio.h>

2 #define PI 3.14159;

3 voidmain( void )

4 {

5 int f ;

6 short p, k ;

7 double c , r , s ;

8 /\* for task 1 \*/

9 printf(“Input Fahrenheit:” ) ;

10 scanf(“%d”, f ) ;

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

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

13 /\* for task 2 \*/

14 printf("input the radius r:");

15 scanf("%f", &r);

16 s = PI \* r \* r;

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

18 /\* for task 3 \*/

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

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

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

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

}

**解答**：本实验共有9个错误，修改数为10。

第2行#define定义符号常量指令行末尾不加分号。改为：#define PI 3.14159

第3行void main是非标准语句，需改为int main，同时最后加上return 0.

第10行scanf函数格式错误，应为：scanf("%d",&f);

第11行的5与9都是整型数，相除之后结果为0，故应将数字类型转换，改为：c=5.0/9 \*(f-32)

第15行r为双精度变量，故应改为：scanf(“%lf”,&r);

第17行printf语句不需要加地址符&，故改为：printf("\nThe acreage is %.2f\n\n",s);

第20行应取作16进制运算，故改为：scanf("%hx %hx", &k, &p );

newint变量并没有定义，故应在第6行声明它的定义：short p, k ,newint;

第21行转换式子出错，应为：newint = (p&0xff00)|(k&0xff00)>>8;

以下为经过修改后的源程序。

#include <stdio.h>

#include <stdlib.h>

#define PI 3.14159

int main(void)

{

int f;

short p,k,newint;

double c,r,s;

/\*for task1\*/

printf("Input Fahrenheit:");

scanf("%d",&f);

c=5.0/9\*(f-32);

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

/\*for task2\*/

printf("inout the radius r:");

scanf("%lf",&r);

s=PI\*r\*r;

printf("\nThe acreage is %.2f\n\n",s);

/\*for task3\*/

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

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

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

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

return 0;

}

对修改后的程序进行一次简单的数据检验，如图1.1所示。

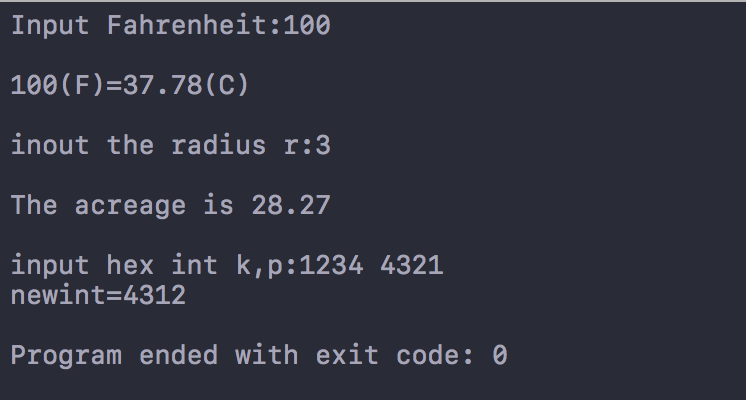


图 1.1 修改后的程序运行结果

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，请改用不使用第3个变量的方法实现。该程序中t是中间变量，要求将定义语句中的t删除，修改下划线处的语句，使之实现两数对调的操作。

#include<stdio.h>

void 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);

}

**解答**：利用按位加运算可以完成，程序如下

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a,b;

printf("Input two integers:");

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

a^=b;

b^=a;

a^=b;

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

return 0;

}

对完成的程序进行一次简单的程序验证，如图1.2所示。

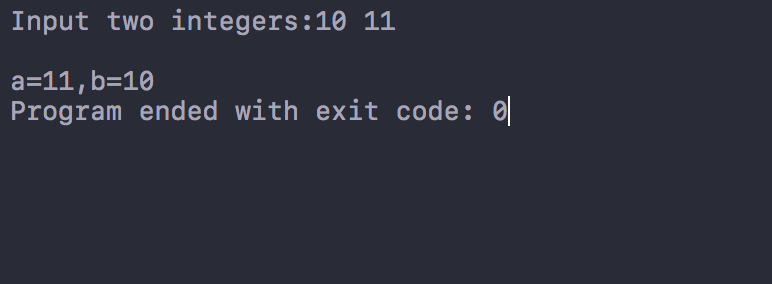


图1.2简单的程序验证

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**:首先判断是否为大写字母，若为大写字母，则将该字母的ASCII码减去32，若不是，则直接输出。转化为流程图（图1.3）

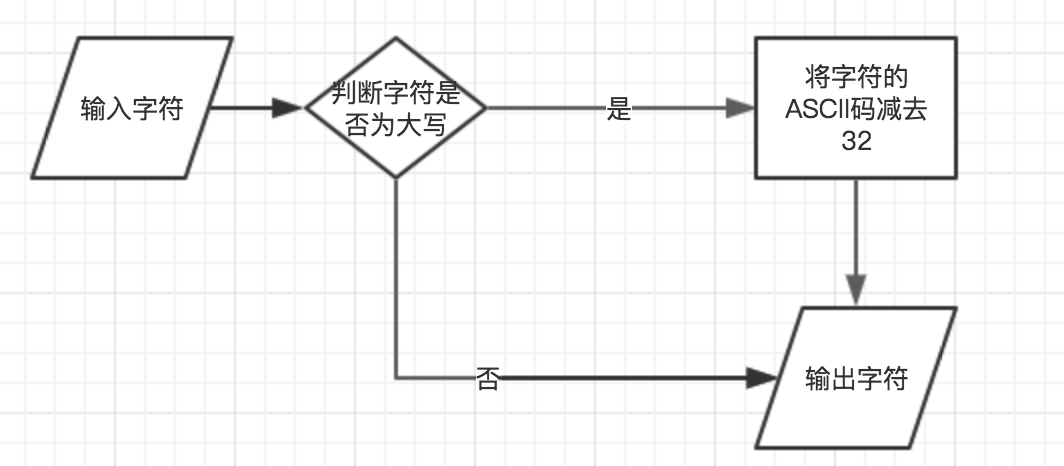


图1.3程序设计第一题的流程图

利用流程图思路写成源程序如下

#include <stdio.h>

#include <stdlib.h>

int main()

{

char x;

printf("Input your char:");

scanf("%c",&x);

if(x>='a'&&x<='z')

{

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

}

else if(x>='A'&&x<='Z')

{printf("%c\n",x+32);}

else printf("please input right char");

return 0;

}

向程序输入小写的“a”，应该输出也是小写的“a”，实验结果（图1.4）与预设一致

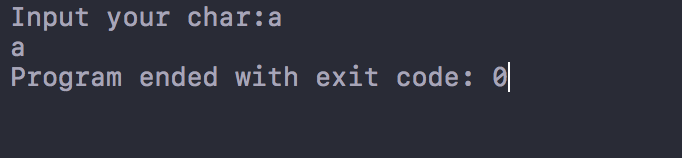


图1.4程序设计第一题的输出结果

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**：解题思路主要通过判断语句完成。

1.输入x，m，n，为了方便分析测试结果，x的输入采用16进制

2.如果0 ≤ｍ≤ 15, 1 ≤ ｎ≤ 16-ｍ，转2.1，否则转3.

2.1 首先x>>m，将要处理的n位移动到最右；

2.2 再将上一步的结果左移16-n位，即：x<<16-n;

2.3 用16进制输出结果并转4.

3. 显示输入错误信息；

4. 结束

依照解题思路写出源程序

#include <stdio.h>

#include <stdlib.h>

int main()

{

unsigned short int x,m,n;

printf("Please enter x(16进制,m(0~15),n(1~16-n):\n" );

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

if ((m>=0&&m<=15)&&(n>=0&&n<=15-m))

{

x>>=m;

x<<=(16-n);

printf("%hx",x);

}

else printf("Error!\n");

return 0;

}

分三种不同的情况对源程序进行测试，对于输入的n、m值，分为m超边界条件，n超边界条件，m、n均不超出边界条件进行讨论。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 | D000 或 图1.5 |
| 用例2 | 1101 0101 1000 0011（D583） | 16 | 1 | 输入错误（m值超范围） | 图1.6 |
| 用例3 | 1101 0101 1000 0011（D583） | 13 | 5 | 输入错误（n值超范围） | 图1.7 |

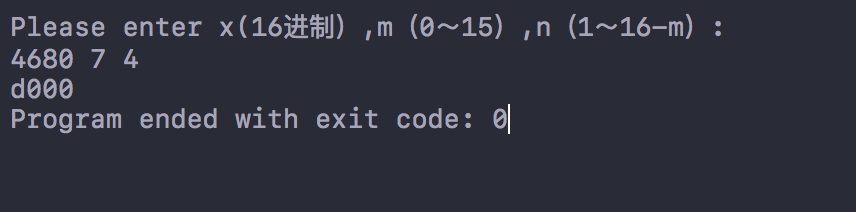


图 1.5 编程题2，m,n均不超出边界条件情况

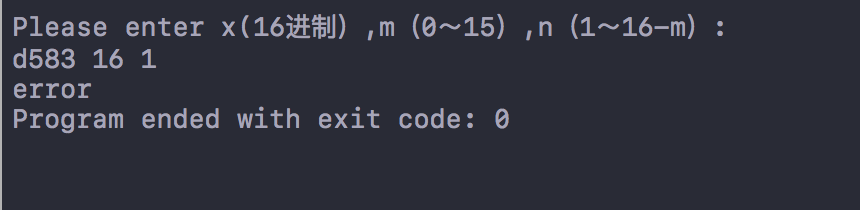


图 1.6编程题2，m超出边界条件情况

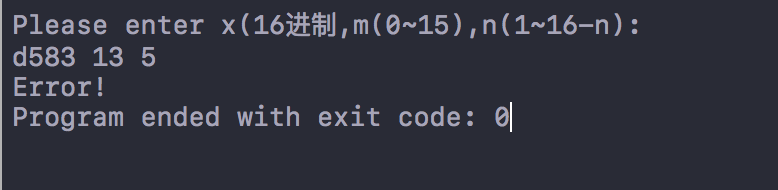


图 1.7编程题2，n超出边界条件情况

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0机器中用无符号长整数表示。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的二进制表示是00101000 01011000 01011100 11010011.按照8位一组可表示40 88 92 211.由于CPU处理数据的差异，他的顺序是颠倒的，所以最终格式为211.92.88.40。

**解答**：本题依据mask的遮盖作用可以将IP地址逐位提出然后再将其反向重新组装，既得到答案。

#include <stdio.h>

#include <stdlib.h>

int main()

{

unsigned long IP;

unsigned short one,two,three,four;

printf("Please enter your IP:");

scanf("%lu",&IP);

one=(IP&0xFF000000)>>24;

two=(IP&0xFF0000)>>16;

three=(IP&0xFF00)>>8;

four=(IP&0xFF);

printf("%hu.%hu.%hu.%hu",four,three,two,one);

return 0;

}

对源程序进行测试，整形676879571的二进制表示是00101000 01011000 01011100 11010011.按照8位一组可表示40 88 92 211.由于CPU处理数据的差异，他的顺序是颠倒的，所以最终格式为211.92.88.40。故输出应为211.92.88.40，结果如图1.8.

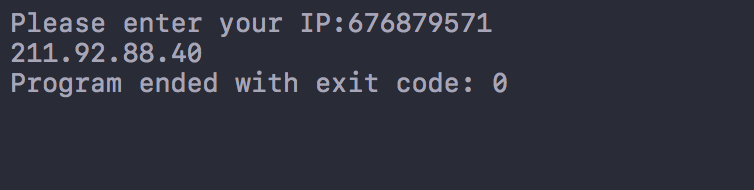


图 1.8 编程题3的输出结果

1.3实验心得

这次c上机实验给了我很多心得体会。其中，最为深刻的是程序的界面友好很关键。众所周知如果你不加以阐释，c语言的运行界面就是一片漆黑，别人根本不知道要输入什么数字，也不知道这些数字会被用来做什么。所以，一个良好的提示界面就显得尤为重要。以及，适当地在自己的代码后加上一点注释，有助于别人理解你的程序，自己以后打开也不会迷茫。

还有就是细心极为重要，这次改错的许多错误也是我经常犯的，比如语句打不打分号的问题等等。对于位运算，以前一直认为其作用不大，今天做了实验题之后我才真正了解到了位运算的简便以及高效，包括mask以及按位加的作用，位运算虽然比较费脑筋但是比起简单的循环确实是高效了许多。

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）练习循环结构for、while、do-while语句的使用。

（3）练习转移语句和标号语句的使用。

（4）使用Turbo C 2.0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源程序改错题

下面是计算s=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。例如，8！=40320。

#include <stdio.h>

void 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);

}

**解答：**以上程序有4处错误，其中第1个错误属于数值范围问题，n的阶乘往往很大，故用int作为其变量的类型不是很合适，将s改为long型更为合适。第二个为格式问题，scanf语句中需要使用地址符为变量赋值。第三个为for语句格式错误，for语句括号中需要用“；”隔开，不能省略也不能替换。第四个错误为return语句位置错误，return语句应在函数结束之前出现。下面列出修改后的源程序。

#include <stdio.h>

int main()

{

int i,n;//s为n！的值，int类型过于小，应改为long

long s=1;

printf("Please enter n:");

scanf("%d",&n);//scanf语句错误，缺少地址符&

for(i=1;i<=n;i++)//for 语句错误，中间应用；隔开

s=s\*i;

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

putchar('\n');

return 0;

}

//return 0 (return 函数应该在大括号内)

对程序输入n=10，则此时输出的结果应该是10！=3268800，程序测试如图2.1，与预测结果一致。

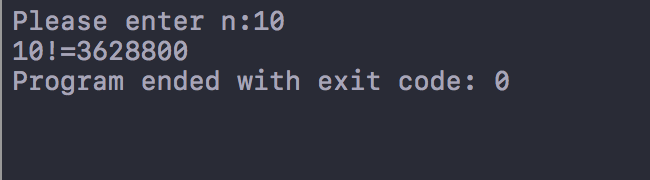


图2.1改错题修改后的程序测试

2.2.2源程序修改替换题

（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，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40310，输出结果为n=8。

**解答（1）**：相同的循环内容用不同的循环结构实现需要注意控制循环的条件及不同循环结构的格式。While语句循环结构流程图如图2.2

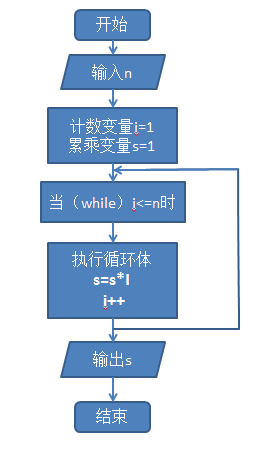


图2.2 while语句流程图

对第一题用while语句替换for语句

#include <stdio.h>

int main()

{

int i=1,n;

long s=1;

printf("Please enter n:");

scanf("%d",&n);

while (i<=n) {

s\*=i;

i++;

}

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

putchar('\n');

return 0;

}

do while语句流程图如图2.3

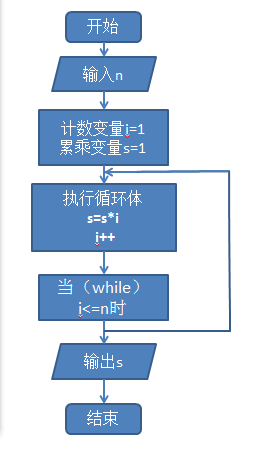


图2.3 do while语句流程图

对第一题的程序使用do while替换for语句

#include <stdio.h>

int main()

{

int i=1,n;

long s=1;

printf("Please enter n:");

scanf("%d",&n);

do {

s\*=i;

i++;

}while (i<=n);

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

putchar('\n');

return 0;

}

与第一题一样，使用n=10作为测试样例，此时输出的结果应该是10！=3268800，替换为while的程序测试如图2.4，替换为do while的程序测试如图2.5，均与预测结果一致。

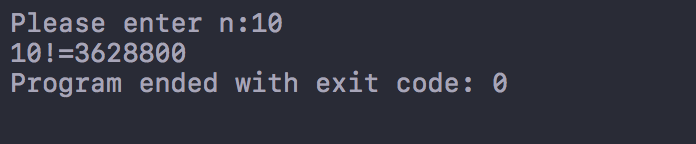


图 2.4 while的程序测试结果

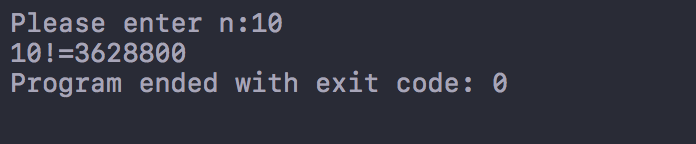


图 2.5 do while的程序测试结果

**解答（2）**：向求n！的循环中加入判断条件判断是否大于s，如果大于s则输出n，否则继续循环。流程图如图2.6。

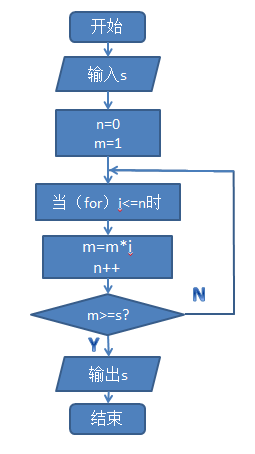


图2.6求最小n的流程图

#include <stdio.h>

int main()

{

int i,n=0;

long s1=1,s2;

printf("Please enter s:");

scanf("%d",&s2);

for (i=1; s1<s2; i++) {

s1\*=i;

n++;

}

printf("满足n!>=s的最小整数n=%d",n);

putchar('\n');

return 0;

}

对程序输入s=40310，因为8！=40320，所以程序应输出n=8，结果如图2.7.

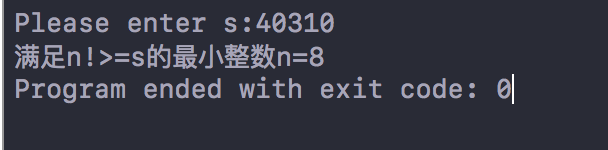


图2.7求最小n的程序测试

2.2.3编程设计题

（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：**该问题为数学分段函数问题，可以利用if语句和switch语句实现。用switch语句是需要注意case的设置。If语句流程图如图2.8所示，switch语句流程图如图2.9所示。

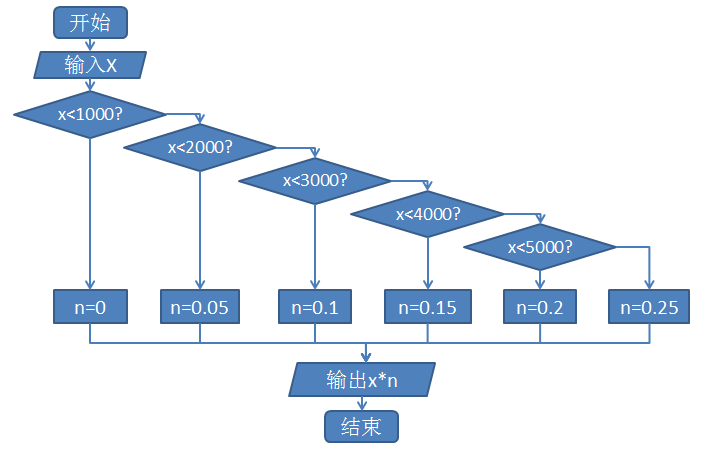


图2.8 if语句流程图

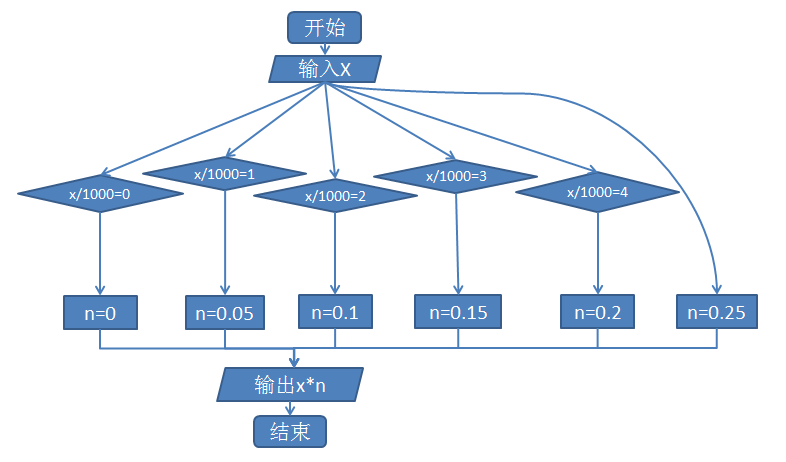


图2.9 switch语句流程图

if型源程序

#include <stdio.h>

int main(void)

{

float x,m;

printf("请输入您的工资金额:");

scanf("%f",&x);

if (x<1000)

m=0;

else if(x<2000)

m=x\*0.05;

else if(x<3000)

m=x\*0.1;

else if(x<4000)

m=x\*0.15;

else if(x<5000)

m=x\*0.2;

else m=x\*0.25;

printf("应收取税金额度：%f",m);

putchar('\n');

return 0;

}

switch型源程序

#include <stdio.h>

int main(void)

{

float x,m;

printf("输入工资金额:");

scanf("%f",&x);

switch ((int)(x/1000)) {

case 0:m=0;break;

case 1:m=x\*0.05;break;

case 2:m=x\*0.1;break;

case 3:m=x\*0.15;break;

case 4:m=x\*0.2;break;

default:m=x\*0.25;

break;

}

printf("应收取税金额:%f",m);

putchar('\n');

return 0;

}

对于该分段函数，选择代表性的100，3500，6000作为输入，输出结果应对应为0，525，1500。输入为100的输出结果见图2.10，输入为3500的输出结果见图2.11，输入为6000的输出结果见2.12。

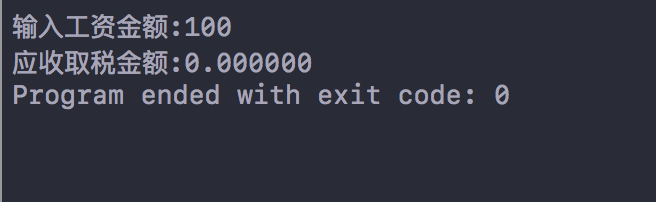


图2.10 工资为100时所收税金

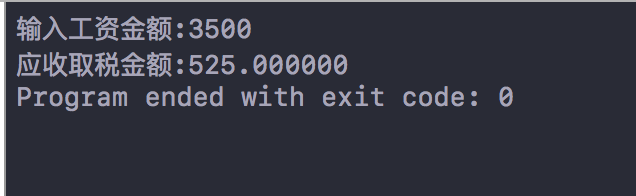


图 2.11工资为3500时所收税金

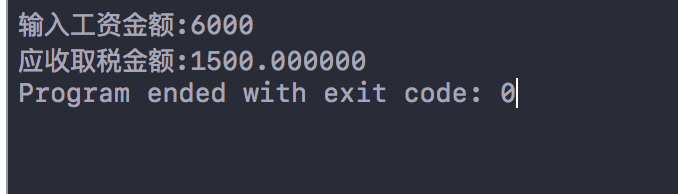


图2.12 工资为6000时所收的税金

（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**：通过状态机判断不同空格的作用，从而消去一些空格的作用。流程图如图2.13。

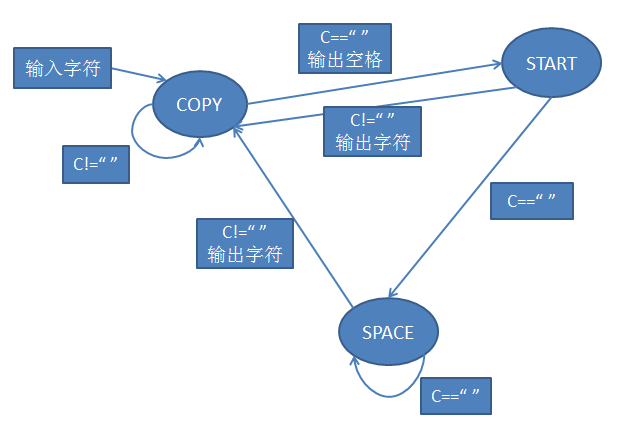


图2.13程序设计题2的流程图

根据流程图编写的程序设计题2的程序如下所示

#include <stdio.h>

enum{copy,start,space};

int main(void)

{

char c;

int state=copy;

printf("please input chars:\n");

while ((c=getchar())!=EOF)

switch(state) {

case copy:

if(c==' ')state=start;

else putchar(c);

break;

case start:

putchar(' ');

if(c==' ')state=space;

else {state=copy;

putchar(c);}

break;

case space:

if (c!=' ') {

state=copy;

putchar(c);

}else state=space;

break;

}

return 0;

}

向程序中输入“ 324 as”和“ghsu sada s aaa sd”程序应该输出“ 324 as”和“ghsu sada s aaa sd”，程序测试结果如图2.14所示。

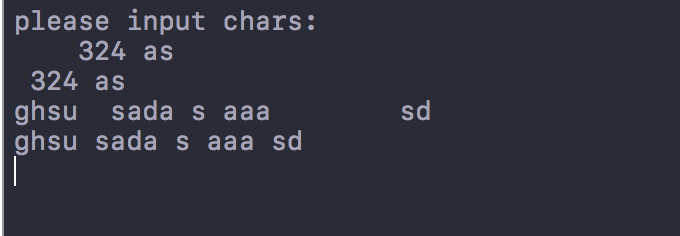


图2.14 将多个空格改为一个的程序结果

（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)

**解答**：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。利用不同的变量分别对行、列进行定义利用循环及判断条件可以完成程序。

源代码如下所示。

#include <stdio.h>

int main(void){

int i,j,c,m,n=10;

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

{

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

{ printf(" ");}

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

if(j==0)c=1;

else c=c\*(i-j+1)/j;

printf("%-4d",c);

}

printf("\n");

}

return 0;

}

将程序运行，查看“金字塔”效果。结果如图2.15所示。

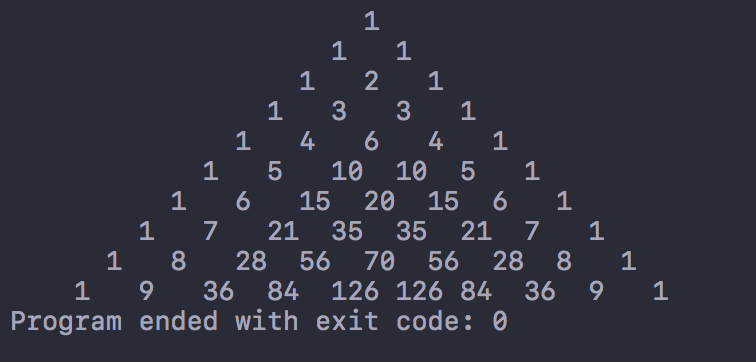


图2.15杨辉三角程序题输出结果

（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**：多次除10取余，每次都分别输出余数，然后除以10并赋给原数重复计算，如流程图2.16所示。

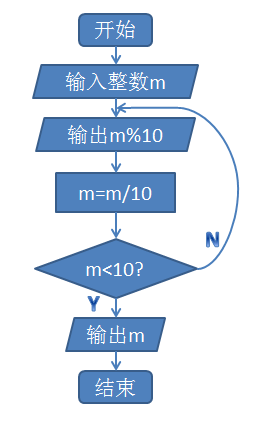


图2.16逆转正整数流程图

根据流程图写出程序如下。

#include <stdio.h>

int main(void)

{

long m,n;

printf("请输入一个正整数:");

scanf ("%ld",&n);

printf("逆转后的整数为：");

while (n>=10) {

m=n%10;

printf("%ld",m);

n/=10;

}

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

return 0;

}

输入1234，则根据题意应输出4321，程序测试结果如图2.17所示。

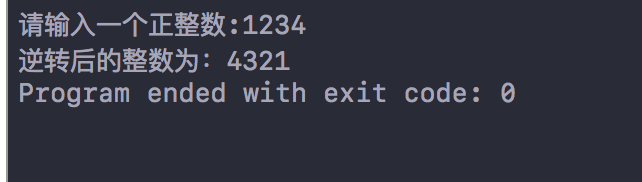


图2.17 逆转正整数

2.2.4 选做题牛顿近似法

编写一个程序，用牛顿迭代法求方程f(x)=3x^2-4x^2-5x+13=0满足精度1e-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：

x0=a; x(k+1)=xk-f(xk)/f’(xk);

其中，f’(x)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a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|xn-xn-1|<=精度e，此时值xn即为所求的近似根。

**解答**：利用do while循环可以不断地近似，再通过if语句判断直到最后一项|xn-1-xn|<e时程序终止，输出结果。其中需注意fx的表达形式，应选择效率最高的表达形式即运算量最少的形式。代码如下所示。

#include <stdio.h>

#define a 1e-6

int main(void)

{

double x,d;

printf("input your initial root:\n");

scanf("%lf",&x);

do{

d=-((((3\*x-4)\*x)-5)\*x+13)/((9\*x-8)\*x-5);

x+=d;

}while (fabs(d)>a);

printf("x=%f\n",x);

return 0;

}

取1为牛顿迭代法的近似根，程序运行结果如图2.18所示。

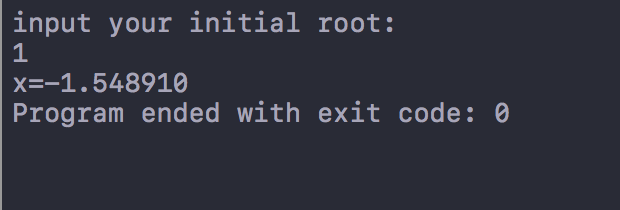


图2.18牛顿迭代法的运行结果

2.3实验心得

通过这次关于流程控制的试验，首先，我意识到了控制流程的方式不止有一种，在多种流程中我们应选择最为高效简便的方案，在写程序之前先思考完方案再开始写代码。感觉与写作文相同，先拟定写作手法再下笔。其次，有些题目看起来思路不是那么的清晰，切勿死磕，应该动动笔，试着画看看流程图，或者带入一些特殊值，尝试找找规律。最后，以前我写代码的时候经常会将循环语句后的“{}”混淆，导致程序出错时检查十分艰辛，流程变得混乱不堪，既不利于自己检查，别人也看得费劲。后来我的一个舍友教了我一个窍门，他让我将“{}”打在关键字下面，并且一一对齐，这样检查的时候就能轻松地分辨括号内对应的是什么函数，也不容易出错。这点我受益匪浅。

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源程序改错题

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。

1 #include "stdio.h"

2 void main(void)

3 {

4 int k;

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

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

7 }

8 long sum\_fac(int n)

9 {

10 long s=0;

11 int i;

12 long fac;

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

14 fac\*=i;

15 s+=fac;

16 return s;

17 }

**解答**：本程序共有3处错误。除此之外，还有一处问题，即C99之后函数尽量不要用void型，所以定义函数有一个返回值0。

第2行错误，在引用副函数之前应该先声明副函数。

第12行错误，fac变量没有定义初始值，有可能会导致计算出错，故添加初始值为0。

第15行错误，s+=fac；这条语句应该在for循环语句里面否则不能起到累加的作用。

将以上错误进行修改得到程序如下。

#include "stdio.h"

long sum\_fac(int n);//先声明函数，否则函数无法使用

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=1;//为fac赋初值，防止计算出错

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

{

fac\*=i;

s+=fac;//该语句应该在循环体里面

}

return s;

}

3.2.2源程序修改替换题

（1）修改第1题中sum\_fac函数，使其计算量最小。

**解答**：sum\_fac函数为求1!+2!+3!+…+n!的函数，可以利用静态存储变量来简化运算。将sum\_fac函数修改如下所示。流程图如图3.1所示。

long sum\_fac(int n)

{

static long f=1;

f\*=n;

return f;

}

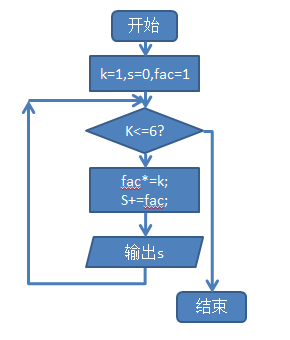


图3.1源程序修改替换第1题流程图

运行结果如图3.2所示。

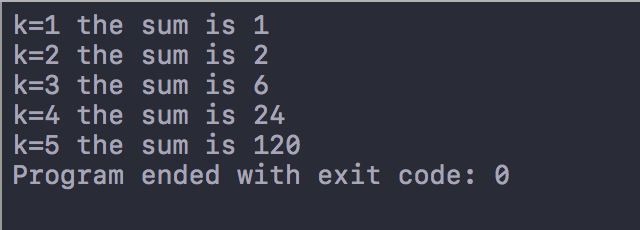


图3.2修改后编程题1的运行结果

（2）修改第1题中sum\_fac函数，计算

**解答**：修改sum\_fac使得n！可以单独求出，然后在主函数中求出它们倒数最后相加即得到。修改后的sum\_fac函数如下所示。流程图如图3.3所示。

long sum\_fac(int n)

{

int i;

long f=1;

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

return f;

}

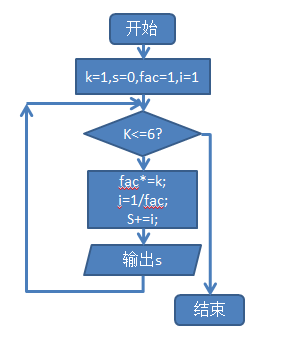


图3.3源程序修改替换题2流程图

运行结果如图3.4所示。

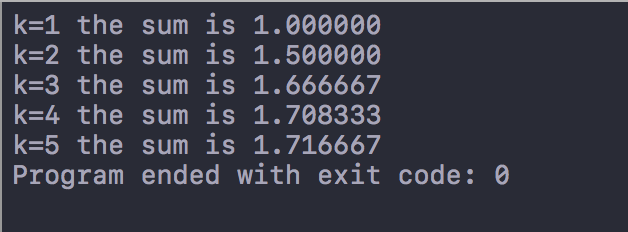


图3.4编程题2 的运行结果

3.3.3跟踪调试题

计算fabonacci数列前n项和的程序如下：

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）。

void main(void)

{

int i,k;

long sum=0,\*p=&sum;

scanf("%d",&k);

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

sum+=fabonacci(i);

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

}

}

long fabonacci(int n)

{

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

return 1;

else

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

}

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？

**解答**：（1）刚执行完scanf("%d",&k);语句，p值是0\*7fff5fbff798，i值是24630。如图3.5所示。

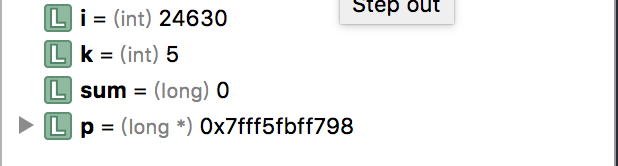


图3.5刚执行完scanf("%d",&k);语句，p,i值

（2）从fabonacci函数返回后光条停留在sum+=fabonacci(i);语句上，如图3.6所示。

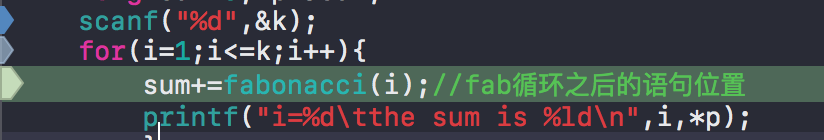


图3.6从fabonacci函数返回后的语句

（3）进入fabonacci函数，watch窗口显示的是n的值。

（4）当i=3，从调用fabonacci函数到返回，n值变化为3，2，3，1，3。

3.3.4编程设计题

（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。

（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）利用辗转相除法求两个数的最大公约数。流程图如图3.7所示。

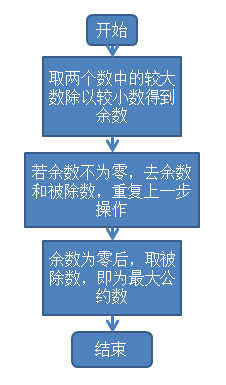


图3.7编程设计题1中辗转相除法流程图

根据流程图设计的程序代码如下所示。

#include <stdio.h>

int last (int m,int n){

int z,r;

if (m>=n&&m%n==0) z=n;

else if (m<=n&&m%n==0) z=m;

else

{

r=m%n;

m=n;

n=r;

z=last(m,n);

}

return z;

}

int main (void){

int m,n;

printf("please input two integers:\n");

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

printf("两整数的最大公约数为：%d\n",last(m,n));

}

程序测试样例为148，35以及46，23，其输出结果应为1，23。程序测试结果如图3.8和3.9所示。

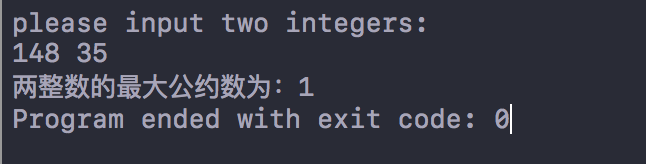


图3.8程序设计测试1

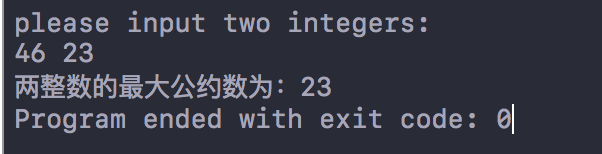


图3.9程序设计测试2

（2）首先先写出对于给定的偶数判断是否满足哥德巴赫猜想的式子。然后控制循环条件使其能在begin与end给定的数的范围中执行。判断是否满足哥德巴赫猜想流程图如图3.10。

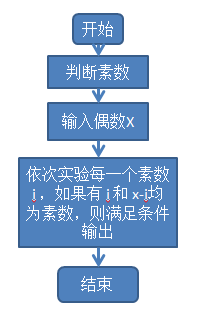


图3.10判断是否满足哥德巴赫猜想

由流程图写出程序如下。

#include <stdio.h>

int prime(int n)

{

int k;

for(k=2;k<n;k++)

if (!(n%k))return 0;

return 1;

}

int main ()

{

int y,n,m,i;

printf("input your numbers range:");

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

for (i=n; i>=n&&i<=m; i++)

{

for (y=2; y<i; y++)

{

if (prime(y)&&prime(i-y))

{

printf("%d=%d+%d\n",i,y,i-y);

break;

}

}

}

return 0;

}

程序测试样例为begin10，end20。输出如图3.11所示，均满足哥德巴赫猜想。

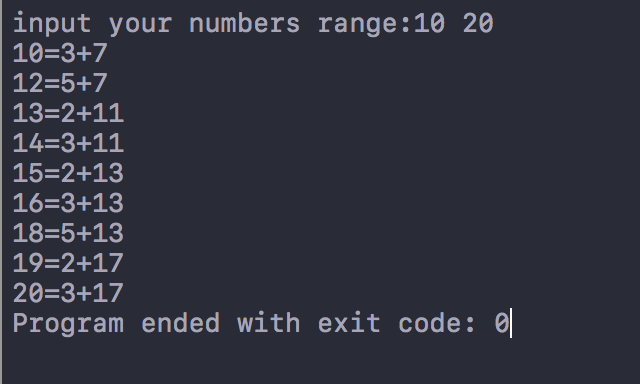


图3.11编程题2样例测试结果

3.3.5选做题

设file1.c如下：

#include <stdio.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 \*/

void main(void)

{

x=10;

y=20;

ch=getchar();

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

func1();

}

file2.c如下：

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)

{

x++;

y++;

ch++;

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

}

试用TCC进行多文件编译和链接。然后在DOS环境下运行生成的可执行文件。

**解答**：由于我使用的IDE不是windows系统支持的，所以无法在dos环境下运行生成可运行文件。不过我在project中仍可运行上述程序，运行结果如图3.12所示，其中样例为“a”.

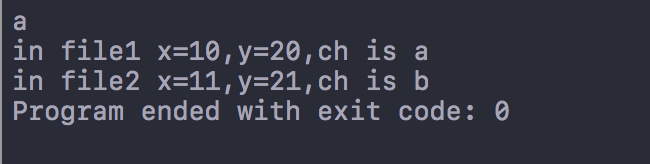


图3.12多文件编译的运行结果

3.3实验心得

首先，我觉得我在本次实验中对于副函数的认识更加全面，以前总是有“丢三落四”的习惯，常常在主函数前忘了定义副函数。而且对于副函数的计算量控制的也不是很好，源程序修改替换题1为我敲响了警钟，要追求更加简洁，计算量更小的程序。

其次，之前因为对Xcode这个IDE不熟悉，每次进行debug的时候就会莫名其妙进入汇编语言模式，弄得我丈二和尚摸不着头脑，专门请教了助教老师，他帮我把汇编模式关闭了，也算是个不大的收获吧。在写冰雹数的时候深刻体会到Debug对于调试程序错误时十分有效，可以定位到程序错误的点，这是最大的收获。