

**课 程 实 验 报 告**

**课程名称： C语言程序设计**

**专业班级：计算机科学与技术1607**

**学 号： U201614700**

**姓 名： 王亚宁**

**指导教师： 周时阳**

**报告日期： 2017年3月19日**

**计算机科学与技术学院**

目录

[**1表达式和标准输入输出实验 4**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837920)

[1.1必做题 4](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837921)

[1.2小结 14](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837923)

[**2流程控制实验 2**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837924)

[2.1必做题 2](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837925)

[2.2选做题 2](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837926)

[2.3自设题 2](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837927)

[2.4小结 2](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837928)

[**3函数与程序结构实验 3**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837929)

[3.1必做题 3](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837930)

[3.2选做题 3](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837931)

[3.3自设题 3](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837932)

[3.4小结 3](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837933)

[**4编译预处理实验 4**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837934)

[4.1必做题 4](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837935)

[4.2自设题 4](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837936)

[4.3小结 4](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837937)

[**5数组实验 5**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837938)

[5.1必做题 5](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837939)

[5.2选做题 5](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837940)

[5.3自设题 5](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837941)

[5.4小结 5](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837942)

[**6指针实验 6**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837943)

[6.1必做题 6](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837944)

[6.2选做题 6](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837945)

[6.3自设题 6](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837946)

[6.4小结 6](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837947)

[**7结构与联合实验 7**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837948)

[7.1必做题 7](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837949)

[7.2选做题 7](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837950)

[7.3自设题 7](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837951)

[7.4小结 7](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837952)

[**8文件实验 8**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837953)

[8.1必做题 8](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837954)

[8.2自设题 8](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837955)

[8.3小结 8](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837956)

[**参考文献 9**](file:///D:\Yaning%20Wang\Documents\作业\《C语言程序设计实验》%20-%20副本.doc#_Toc404837957)

# 1表达式和标准输入输出实验

## 1.1必做题

题目详见课本。

### 1.1.1 源程序改错

程序代码：

#include<stdio.h>

#define PI 3.14159;

voidmain(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);

}

本程序共存在9处错误，分析如下：

(1)#define PI 3.14159;错误原因：宏定义在编译时会原封不动展开，所以后面不应该有分号，改正：去掉后面的分号。

(2)voidmain(void)错误原因：主函数名应该和其返回类型间有空格，而且推荐返回值为int。改正：int main(void)

(3)short p, k;错误原因：在最后输出的新整数newint未定义，应在此处声明。改正：short p, k, newint;

(4)scanf(“%d”, f );错误原因：scanf函数后面的参数应该是要输入的数据的地址。改正：scanf(“%d”, &f);

(5)c = 5/9\*(f-32) ;错误原因：5和9是整型常数，两者进行/运算的结果为0无法得到想要的结果。改正：c=5.0/9\*(f-32);

（6）scanf("%f", &r);错误原因：r是double类型变量，输入应使用%lf。改正：scanf(“%lf”m, &r);

（7）printf("\nThe acreage is %.2f\n\n", &s);错误原因：pintf函数参数应该是变量本身而不是变量的地址。改正：printf("\nThe acreage is %.2f\n\n", s);

(8)scanf("%x %x", &k, &p );printf("new int = %x\n\n",newint);错误原因：k,p为short类型，对应十六进制占位符应为%hx改正：scanf("%hx %hx",&k,&p);printf("new int = %hx\n\n",newint);

（9）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。改正：newint=(p&0xff00)|(((k&0xff00)>>8)&0x00ff);

### 1.1.2 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，请改用不使用第三个变量的方法实现。该程序中t是中间变量，要求将定义语句中的t删除，修改下列划线处的语句，使之实现两数对调的操作。

程序代码：

#include<stdio.h>

void main()

{

int a, b, t;

printf(“Input two integers : ”);

scanf(“%d %d”, &a, &b);

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

printf(“\na = %d, b = %d”, a, b);

}

【分析】将a, b求和存储在其中一个变量中，再加减运算进行交换，此方法有可能会发生溢出，所以改用异或位运算将a, b的值互换。

流程图见图1-1：

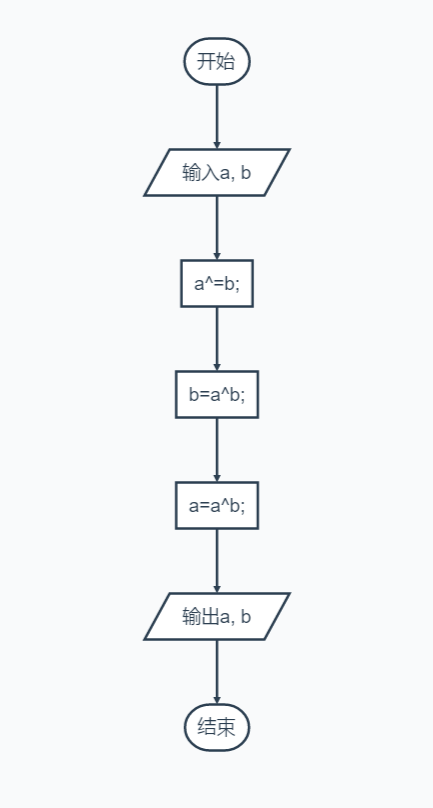


图1-1

#include<stdio.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;

}

测试数据见表1-1：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 0 7 | 7 0 |
| 200 -100 | -100 200 |

结果见图1-2，1-3

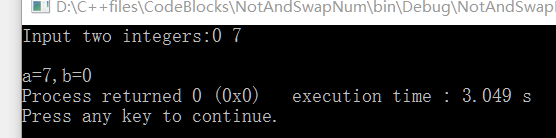


图1-2

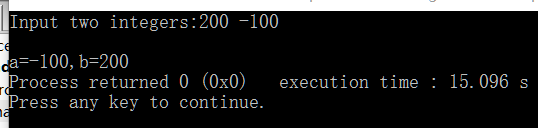


图1-3

### 1.1.3 程序设计

编写并上机调试运行能实现以下功能的程序。

（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。

【分析】可以用getchar得到字符c，再比较c与’A’,’Z’的大小，如果c的值位于两者之间，则将c转换为小写字母，否则不变，最后putchar输出。

流程图见图1-4：

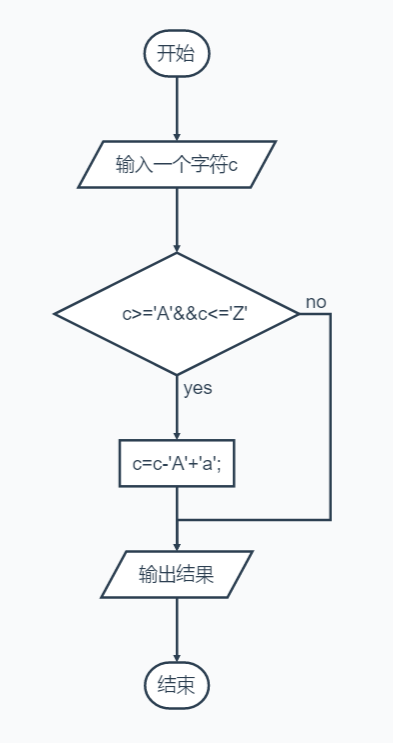


图1-4

#include <stdio.h>

int main()

{

char c;

printf("请输入一个字符：");

c = getchar();

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

{

c = c - 'A' + 'a';

}

putchar(c);

return 0;

}

测试数据见表1-2：

|  |  |
| --- | --- |
| 输入 | 输出 |
| a | a |
| F | f |
| 2 | 2 |
| & | & |

表1-2

结果见图1-5，1-6，1-7，1-8：

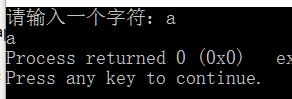


图1-5

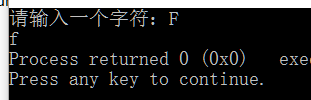


图1-6



图1-7

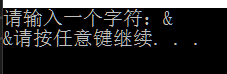


图1-8

【分析】可以先把x向右靠齐到所需位置，然后对不需要的位数与对应的构造的逻辑尺进行与运算，得出答案并输出。流程图见图1-9：

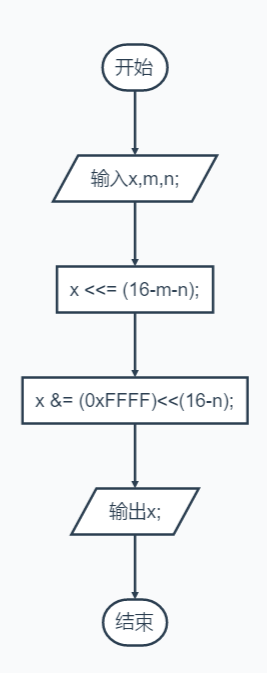


图1-9

#include <stdio.h>

int main()

{

unsigned short x, m, n;

printf("输入无符号短整数x, m, n(0 <= m <= 15， 1 <= n <= 16 - m) :");

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

x <<= (16 - m - n);

x &= (0xFFFF) << (16 - n);

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

return 0;

}

测试数据见表1-3：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 32768 15 1 | 32768 |
| 1 0 1 | 32768 |
| 1 0 2 | 16384 |

表1-3

测试结果见图1-10，1-11，1-12：

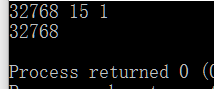


图1-10

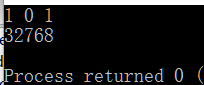


图1-11



图1-12

IP地址通常是4个用句点分隔的小整数（即点分十分制），如32.55.1.102。这些地址在机器中用无符号长整型表示。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：整型676879571的二进制表示是00101000 01010000 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，所以最终格式为211.92.88.40。

【分析】使用逻辑尺可以方便的取出所需的数字，再按照要求逆序输出即可。流程图见图1-13：

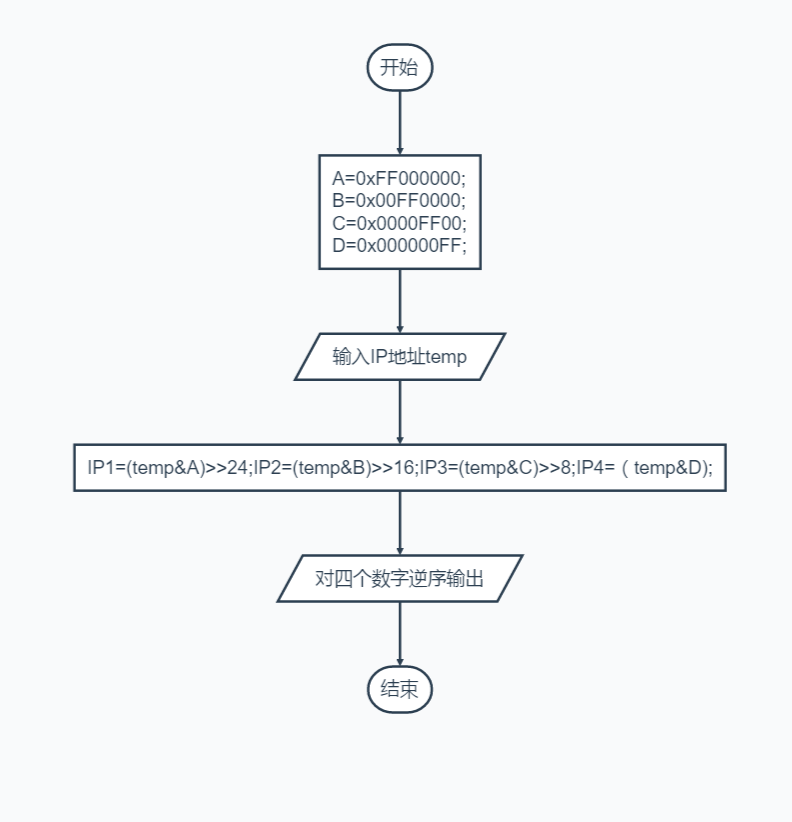


图1-13

#include <stdio.h>

#define A 0xFF000000

#define B 0x00FF0000

#define C 0x0000FF00

#define D 0x000000FF

int main()

{

unsigned long temp = 0;

unsigned char IP1, IP2, IP3, IP4;

printf("Please input the valid IP address : ");

scanf("%ld", &temp);

IP1 = (temp&A) >> 24;

IP2 = (temp&B) >> 16;

IP3 = (temp&C) >> 8;

IP4 = (temp&D);

printf("%u.%u.%u.%u\n", IP4, IP3, IP2, IP1);

return 0;

}

测试数据见表1-4：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 134744072 | 8.8.8.8 |
| 16843009 | 1.1.1.1 |
| 26975347 | 115.156.155.1 |

测试结果见图1-14，1-15，1-16：

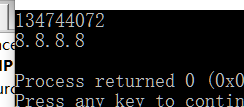


图1-14

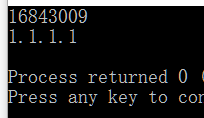


图1-15

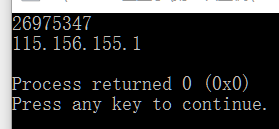


图1-16

## 1.2小结

# 2流程控制实验

## 2.1必做题

### 2.1.1源程序改错

原题目见课本51页。此处指出程序里的3处错误：

#include <stdio.h>

int main()

{

int i, n, s = 1;

printf("Please enter n:");

scanf("%d", &n); //1 输入变量没有取地址符

for (i = 1; i <= n; i++) //2 for里面是分号隔开3个语句，不是逗号

s = s \* i;

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

return 0;

}

// return 0;//3 return 0;应该在main最后不应该在外面

改正后的测试为表2-1：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 0 | 1 |
| 6 | 720 |
| 9 | 362880 |

表2-1

测试输入及其结果见图2-1，2-2，2-3：

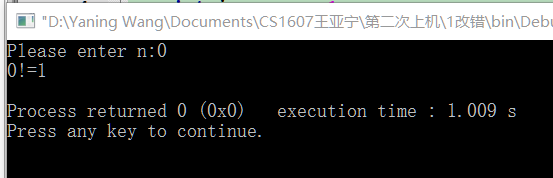


图2-1

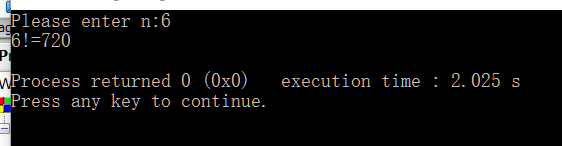


图2-2

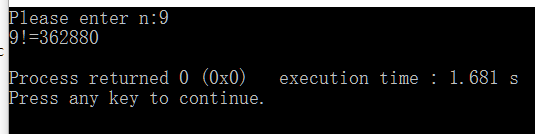


图2-3

### 2.1.2源程序修改替换

题目详见课本52页。第一小题源代码如下：

#include <stdio.h>

int main()

{

int i, n, s = 1;

printf("Please enter n:");

scanf("%d", &n);

// while实现

s = 1;

i = 1;

while (i <= n)

{

s \*= i;

++i;

}

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

// do-while实现

i = 1;

s = 1;

do {

s \*= i;

++i;

} while (i <= n);

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

return 0;

}

测试数据见表2-2：

第1小题测试数据

|  |  |
| --- | --- |
| 输入 | 输出 |
| 0 | 1 |
| 3 | 6 |
| 9 | 362880 |

第一小题结果见图2-4，2-5，2-6：

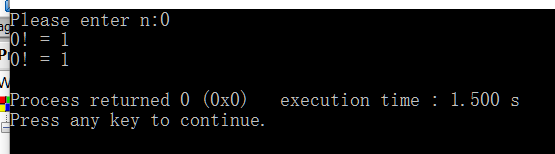


图2-4

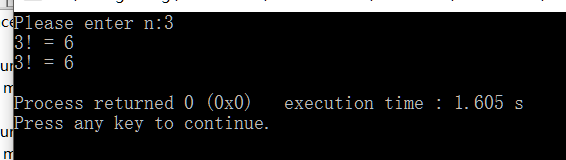


图2-4

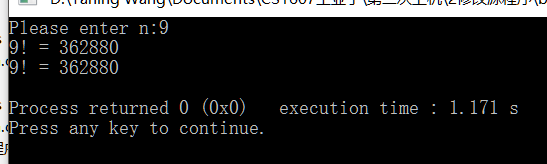


图2-5

第二小题思路是使用while循环直到得出的阶乘大于等于输入的数，第二小题流程图见图2-7。

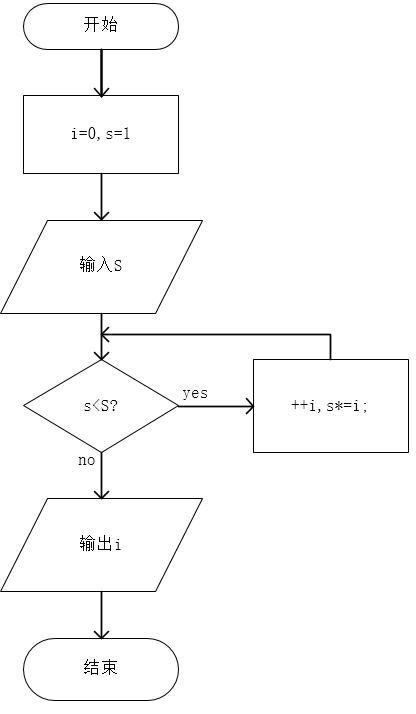


图2-7

第二小题源代码：

#include <stdio.h>

#include <stdlib.h>

int main()

{

unsigned int s = 1;

unsigned int S = 0;

int i = 0;

printf("Please a number:");

scanf("%d", &S);

while (s < S)

{

s \*= ++i;

}

printf("%d! >= %d\n", i, S);

return 0;

}

第二小题测试数据见表2-3：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 0 | 0 |
| 1 | 0 |
| 6 | 3 |
| 40000 | 8 |

测试结果见图2-8，2-9，2-10，2-11

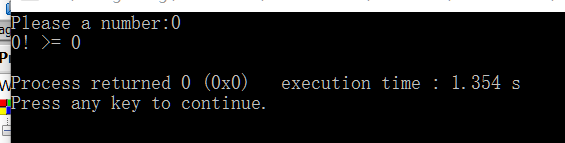


图2-8

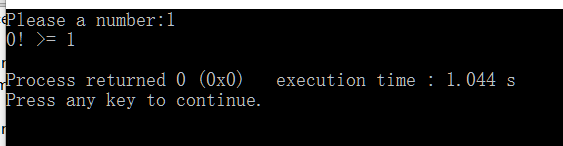


图2-9

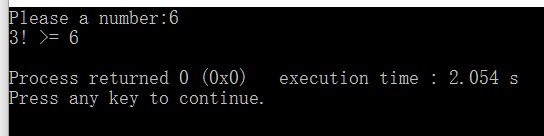


图2-10

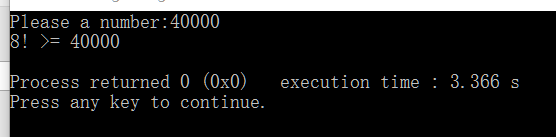


图2-11

### 2.1.3程序设计

题目内容见课本52页。第一小题为输入收入输出所要交的税，税率都已经给出，只需要选择switch或者if-else结构完成计算即可。使用switch的思路是将工资除以1000取其整数部分，进行选择，计算过此工资段的税以后将工资设置为此税率的最低工资，直接进入下一个税率对应的税的计算。if语句思路基本一致，不过不能使用else否则会在一个税计算完了之后就结束运算。流程图见图2-12。

#include <stdio.h>

#include <stdlib.h>

#define EPS 1e-6

#define SWITCH 1 // value 1:using switch; value 0: using if;

#define RATE0 0 // define tax rate

#define RATE1 0.05

#define RATE2 0.1

#define RATE3 0.15

#define RATE4 0.2

#define RATE5 0.25

double Tax(double salary)

{

double tax = 0;

#if (SWITCH)

{

switch ((int)salary / 1000)

// default set salary is greater than 0

{

default:

tax += (salary - 5000) \* RATE5;

salary = 5000;

case 4:

tax += (salary - 4000) \* RATE4;

salary = 4000;

case 3:

tax += (salary - 3000) \* RATE3;

salary = 3000;

case 2:

tax += (salary - 2000) \* RATE2;

salary = 2000;

case 1:

tax += (salary - 1000) \* RATE1;

salary = 1000;

case 0:

tax += (salary - 0) \* RATE0;

salary = 0;

}

}

#else

if (salary > 5000)

{

tax += (salary - 5000) \* RATE5;

salary = 5000;

}

if (salary > 4000)

{

tax += (salary - 4000) \* RATE4;

salary = 4000;

}

if (salary > 3000)

{

tax += (salary - 3000) \* RATE3;

salary = 3000;

}

if (salary > 2000)

{

tax += (salary - 2000) \* RATE2;

salary = 2000;

}

if (salary > 1000)

{

tax += (salary - 1000) \* RATE1;

salary = 1000;

}

if (salary > 0)

{

tax += (salary - 0) \* RATE0;

salary = 0;

}

#endif // SWITCH

return tax;

}

int main()

{

double salary = 0;

printf("Please input salary:");

scanf("%lf", &salary);

printf("%f\n", Tax(salary));

return 0;

}

测试数据见表2-4：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 500 | 0.000000 |
| 1000 | 0.000000 |
| 1500 | 25.000000 |
| 2000 | 50.000000 |
| 2500 | 100.000000 |
| 3000 | 150.000000 |
| 3500 | 225.000000 |
| 4000 | 300.000000 |
| 4500 | 400.000000 |
| 5000 | 500.000000 |
| 5500 | 625.000000 |

表2-4

测试结果如图2-13，2-14，2-15，2-16，2-17，2-18，2-19，2-20，2-21，2-22，2-23：

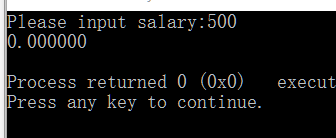


图2-13

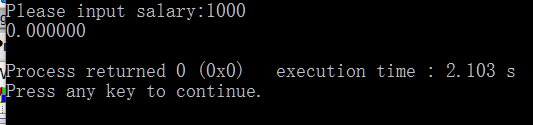


图2-14

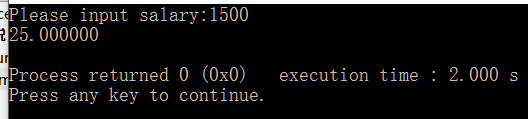


图2-15

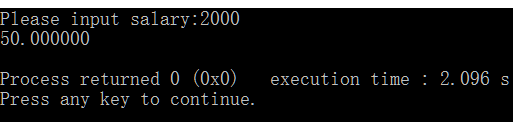


图2-16

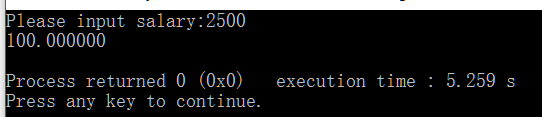


图2-17

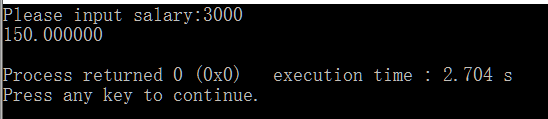


图2-18

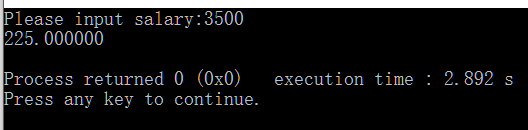


图2-19

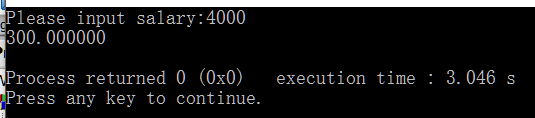


图2-20

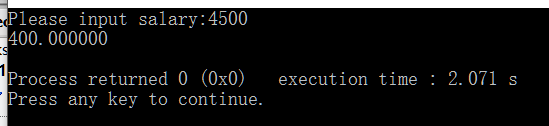


图2-21

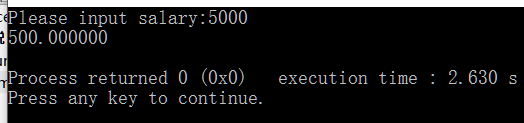


图2-22

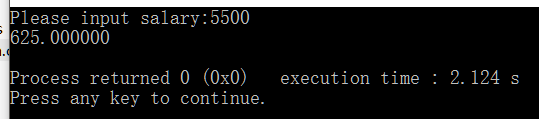


图2-23

第二小题要求将字符串中多于1个的连起来的空格换成一个空格，并输出，思路可以这么想，使用string.h的库函数求出字符串的长度，for语句按字符遍历整个字符串，如果这个字符不是空格或者这个是但下一个不是，就可以输出它。

流程图见图2-24：

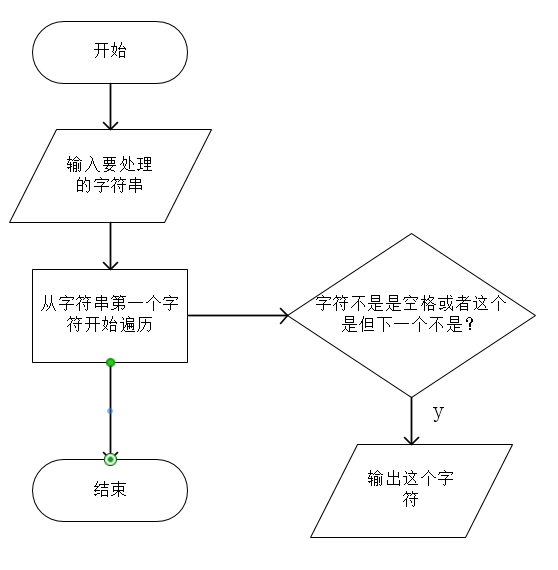


图2-24

#include <stdio.h>

#include <string.h>

int main()

{

char str[1000];

char result[1000];

int count = 0;

printf("Please enter a string:");

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

for (int i = 0; i < strlen(str); ++i)

{

if (str[i] != ' ' || (str[i] == ' ' && str[i + 1] != ' '))

{

result[count] = str[i];

++count;

}

}

result[count] = '\0';

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

return 0;

}

测试数据见表2-5：

|  |  |
| --- | --- |
| 输入 | 输出 |
| Hello world ! | Hello world ! |
| Hello world! | Hello world! |
| Romance is the glamour which turns the dust of everyday life into a golden haze. | Romance is the glamour which turns the dust of everyday life into a golden haze. |

表2-5

测试结果见图2-25，2-26，2-27：

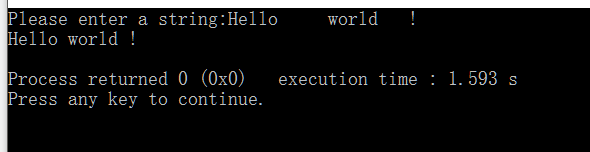


图2-25

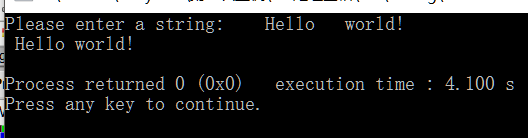


图2-26

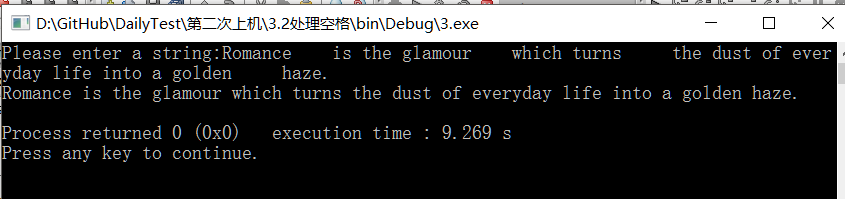


图2-27

第三题是根据输入的数字输出对应层数的杨辉三角，具体实现过程中需要注意空格的输出，数字之间的对齐，输出格式的控制，还有就是三角上的数字不能计算错误。流程图见图2-28：

#include <stdio.h>

void PrintSpace(unsigned int N)

{

char space[3] = { ' ', ' ', '\0' };

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

{

printf("%s", space);

}

}

void Print(unsigned int N)

{

unsigned int temp = 0;

unsigned int j = 0;

for (unsigned int i = 1; i <= N; ++i)

{

temp = 1;

PrintSpace(N - i);

printf("%d", temp);

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

{

temp = temp \* (i - j) / j;

printf("%4d", temp);

}

printf("\n");

}

}

int main()

{

unsigned int N = 0;

printf("Please enter the number of plies you want:");

scanf("%d", &N);

Print(N);

return 0;

}

测试数据设为1，4，9，运行结果见图2-29，2-30，2-31：

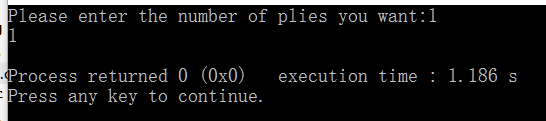


图2-29

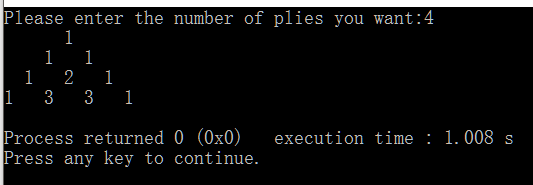


图2-30

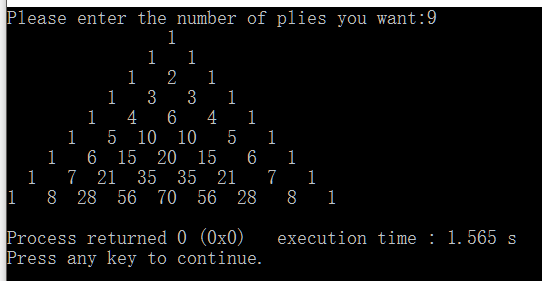


图2-31

## 2.2选做题

题目详见课本53页。题目要求使用牛顿迭代法求方程的近似根，要求精度为1e-6，所以并不需要设计算法来解决问题，直接使用公式即可。流程图见图2-32：

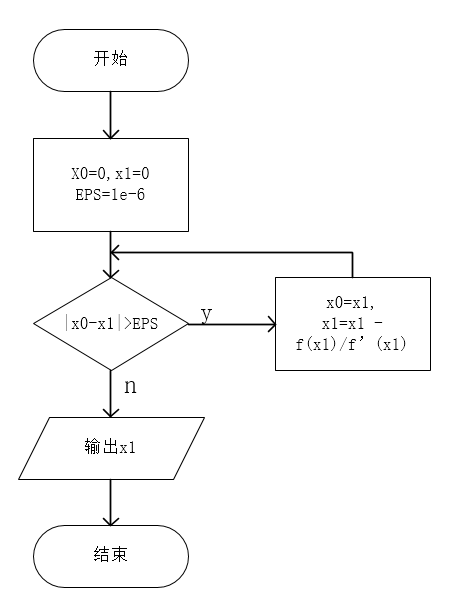


图2-32

源代码：

#include <stdio.h>

#include <stdlib.h>

#define EPS 1e-6

double Func(double x)

{

return (3 \* x \* x \* x - 4 \* x \* x - 5 \* x + 13);

}

double FuncDeri(double x)

{

return (9 \* x \* x - 8 \* x - 5);

}

int main()

{

double x0 = 0;

double x1 = 0;

double diff = 1;

double temp = 0;

while (diff >= EPS)

{

x1 = x0 - Func(x0) / FuncDeri(x0);

diff = x1 - x0;

diff = diff > 0 ? diff : -diff;

x0 = x1;

}

printf("%f\n", x0);

return 0;

}

无需测试数据输出结果如图2-33所示:

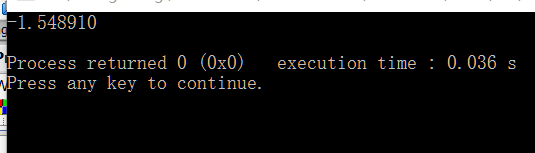


图2-33

## 2.3小结

# 3函数与程序结构实验

## 3.1必做题

### 3.1.1源程序改错

原题目见课本53页，程序中共有4处错误。

#include <stdio.h>

long sum\_fac(int n); //1，添加声明

int main(void)

{

int k;

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

{

printf("k=%d\tthe sum is %ld\n", k, sum\_fac(k)); // 与第1处相关调用的sum\_fac函数未声明

}

return 0;

}

// return 0; 2，return 应该写在main函数最后而不是外面

long sum\_fac(int n)

{

static long s = 0; // 修改s为静态变量

int i;

long fac = 1; // 3，fac未初始化，值未知，初始化为1

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

fac \*= i;

s += fac; //4， s += fac;这里的s是0，加上fac并不是1到n的阶乘和

// 可以把s设置为静态变量，即可满足题意

return s;

}

运行结果见图3-1：

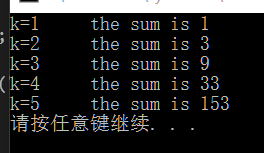


图3-1

### 3.1.2源程序修改替换

具体题目见课本54页，第一小题要求改进算法，使得运算量最小，由于函数的大部分运行时间都浪费在for循环里面，根据题目的结构，可以将fac变量设置为静态变量来避免for循环，从而大大节约运行的时间，减少计算量。优化后的程序：

#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 long s = 0;

static long fac = 1;

fac \*= n;

s += fac;

return s;

}

运行结果见图3-2：

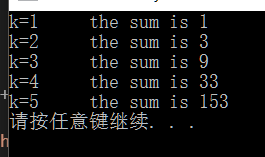


图3-2

第二小题要求修改程序逻辑，计算，在这里同样可以使用静态变量减少运算量，每次fac不需要从1开始算起，而是利用上一次的结果，s也是利用上一次的结果，但是要注意由于这里的结果是小数，所以需要将变量类型设置为double，代码见下：

#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 %f\n", k, sum\_fac(k));

}

return 0;

}

double sum\_fac(int n)

{

static double s = 0;

static double fac = 1;

fac /= n;

s += fac;

return s;

}

运行结果见图3-3：

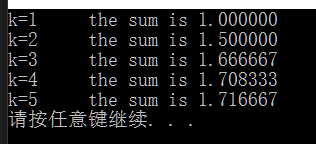


图3-3

### 3.1.3跟踪调试

原题目见课本54页，本题目为单步调试测试。

第一问，刚执行完scanf函数p，i的值是多少，p为一个指向long变量sum的指针，每次运行的时候都会不同，i是前面定义的未初始化的变量，运行期间未对其复制所以i的值是无意义的，VS中调试结果见图3-4：

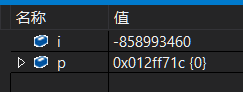


图3-4

可见p的值是一个指针，i的值无意义。

第二小题问当程序从fabonacci函数返回时停留在哪条语句上面，实际上它应该返回到调用它的那一句上面，见图3-5：

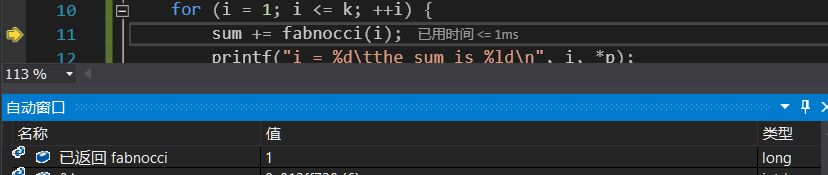


图3-5

黄色箭头所指语句就是返回后所停留语句，就是调用函数的那一句上面。

第三题询问进入fabnocci函数时watch窗口显示，见图3-6：

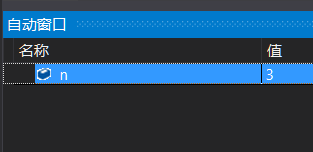


图3-6

可见窗口只有n的值，main函数已被压栈，它里面的变量p和i见图3-7

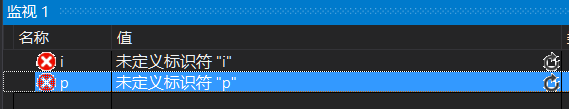


图3-7

在执行函数fabonacci时是不可访问的。

第四题问当i为3时，从调用fabonacci函数返回到main函数时n的值如何变化，见图3-8

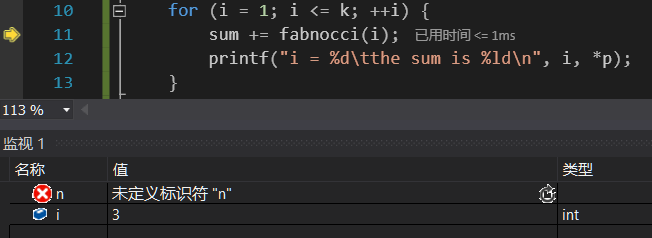


图3-8

可见此时因为fabonacci函数已经结束，而n是局部变量，所以此时n的值是无法访问的，实质上就是这个时候根本没有n。

### 3.1.4程序设计

第一题，使用递归计算两个数的最大公约数，可以使用经典的欧几里得算法，具体实现可以参考流程图图3-9：

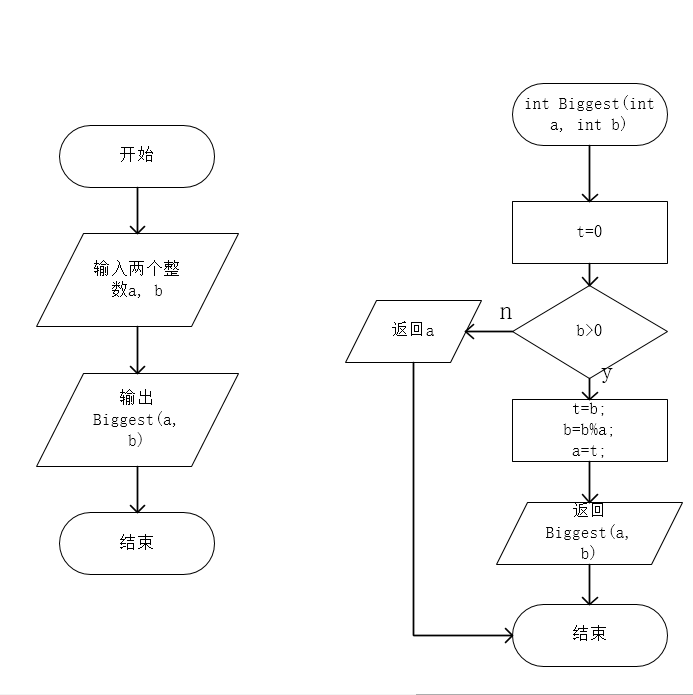


图3-9

#include <stdio.h>

int Biggest(int numA, int numB)

{

int t = 0;

if (numB > 0)

{

t = numB;

numB = numA % numB;

numA = t;

return Biggest(numA, numB);

}

else

{

return numA;

}

}

int main(void)

{

int numA = 0, numB = 0;

printf("Please enter two number:");

scanf("%d %d", &numA, &numB);

printf("Biggest common is %d\n", Biggest(numA, numB));

return 0;

}

测试数据见表3-1：

|  |  |  |
| --- | --- | --- |
| a | b | 输出 |
| 3 | 9 | 3 |
| 2 | 7 | 1 |
| 35 | 28 | 7 |

运行结果见图3-10，3-11，3-12：

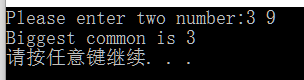


图3-10

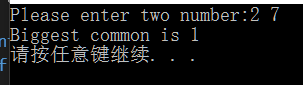


图3-11

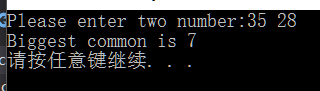


图3-12

第二题，设计程序验证哥德巴赫猜想，即验证任意一个大于等于4的偶数都是两个素数的和，本题思路，如果输入的数字是4由于2的特殊性，我选择直接返回2作为要求的2个素数，对于大于4的偶数，显然它的对应两个素数不会是偶数，所以计算它的对应素数从3开始，每次递增2，可以减少运算量。

程序流程图见图3-13：

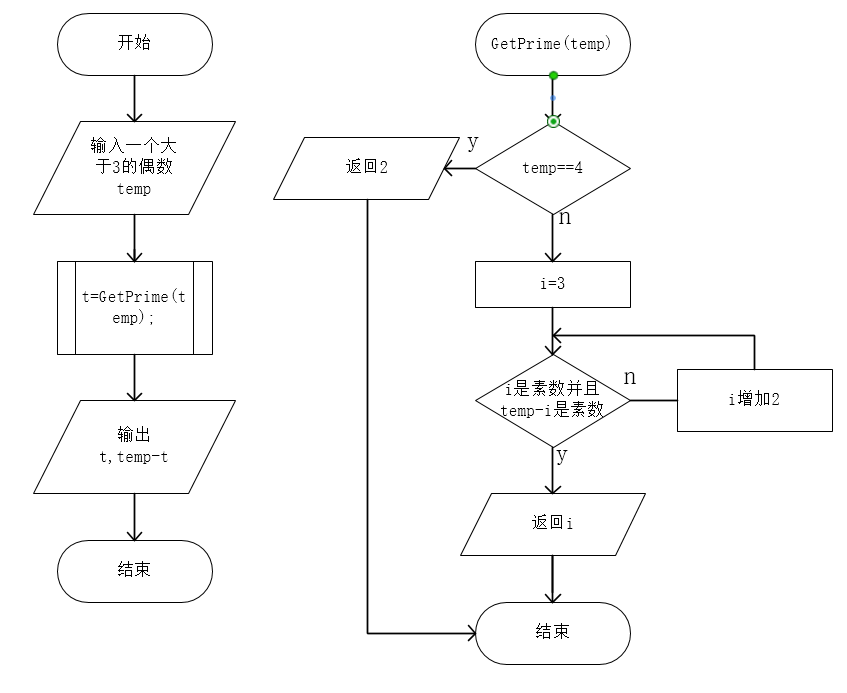


图3-13

#include <stdio.h>

#include <math.h>

int IsPrime(int N) {

for (int i = 3; i <= N \* 0.5; ++i) {

if (N % i == 0) {

return 0;

}

++i;

}

return 1;

}

// 用来获取满足条件的N的第一个素数也是较小的

int GetPrime(int N) {

// 考虑4的特殊性

if (N == 4){

return 2;

}

for (int i = 3; i <= N \* 0.5; ++i) {

if (IsPrime(i) && IsPrime(N - i)) {

return i;

}

++i;

}

return -1;

}

int main() {

int temp = 0;

int t = 0;

printf("Please enter a even number greater than 3:");

scanf("%d", &temp);

t = GetPrime(temp);

printf("%d = %d + %d\n", temp, t, temp - t);

return 0;

}

测试数据见表3-2：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 4 | 2 2 |
| 48 |  |
| 1200 |  |

结果见图3-14，3-15，3-16：

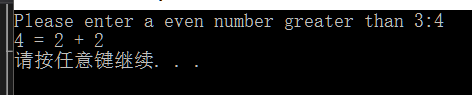


图3-14

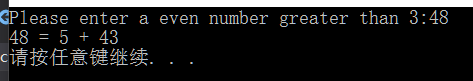


图3-15

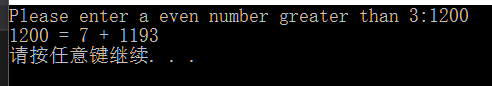


图3-16

第三题，编写函数，证明在两个常量间的偶数均满足哥德巴赫猜想成立，在这里可以借用上一题的一些代码，即判断一个数是否满足哥德巴赫猜想，只不过这一次变成了多个数字而已，大体思路没有改变，只不过要注意一下起始值如果是奇数的话需要 对其加一变成偶数，再进入判断，每次增量为2，流程图省略，代码如下：

#include <stdio.h>

#define BEGIN 10

#define END 20

int IsPrime(int N) {

for (int i = 3; i <= N \* 0.5; ++i) {

if (N % i == 0) {

return 0;

}

++i;

}

return 1;

}

// 用来获取满足条件的N的第一个素数也是较小的

int GetPrime(int N) {

// 考虑4的特殊性

if (N == 4) {

return 2;

}

for (int i = 3; i <= N \* 0.5; ++i) {

if (IsPrime(i) && IsPrime(N - i)) {

return i;

}

++i;

}

return -1;

}

int main() {

int numA = 0, numB = 0;

int t = 0;

numA = BEGIN;

numB = END;

printf("GOLDBACH'S CONJECTURE:\n");

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

// 判断初始值奇偶

if (numA % 2 == 1) {

++numA;

}

for (int i = numA; i <= numB; ++i) {

t = GetPrime(i);

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

++i;

}

return 0;

}

执行结果见图3-17

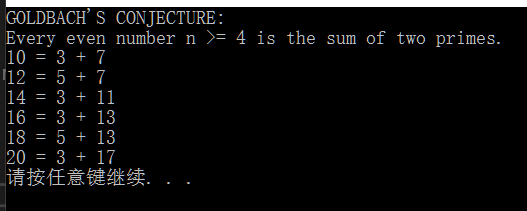


图3-17

## 3.2选做题

选做题为测试多源文件编程，具体代码见附带的文件。

## 3.3小结

# 4编译预处理实验

## 4.1必做题

### 4.1.1源程序改错

原题目见课本59页，本程序中共有4处错误，错误和改正如下：

#include <stdio.h>

#define SUM ((a)+(b)) //1, 宏定义最好都加上括号

#define DIF ((a)-(b))

#define SWAP(a, b) a^=b,b^=a,a^=b //2，a=b,b=a此交换无效

int main(void) {

int a, b, t;

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

scanf("%d,%d", &a, &b); //3，a未定义

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);//4，printf的p小写

return 0;

}

测试数据见表4-1：

|  |  |
| --- | --- |
| 输入a, b | 输出a,b的和，平方差交换后的数值 |
| 100 32 | 132 8976 32 100 |
| 0 255 | 255 -65025 255 0 |
| -10 23 |  |

运行结果见图4-1，4-2，4-3：

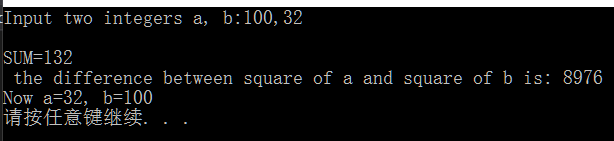


图4-1

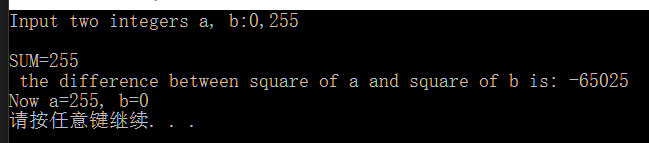


图4-2

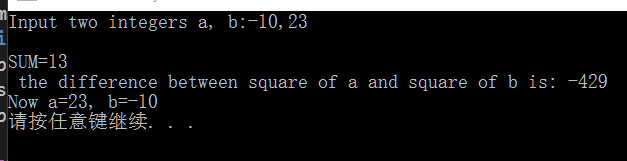


图4-3

### 4.1.2源程序修改替换

原题目见课本59页，题目为使用函数求三个数字中的最大数和最小数，错误和改正如下：

// 1，添加使用了的头文件

#include <stdio.h>

// 2，函数调用时没有声明，添加声明

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

float sum(float x, float y);

// 3，main函数最后返回了0，那么应该为int

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;

}

测试数据见表4-2：

|  |  |
| --- | --- |
| 输入3个数求最大，输入2个数求和 | 输出最大数，输出和 |
| 1 5 4 38 29 | 5 67 |
| 23 -678 7 73 12.3 | 23 85.3 |
| 234 -32 3264 76723 172 | 3264 76895 |

测试结果见图4-4，4-5，4-6

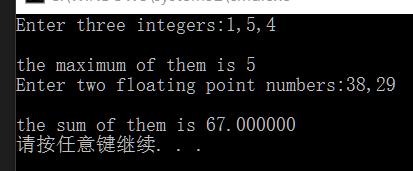


图4-4

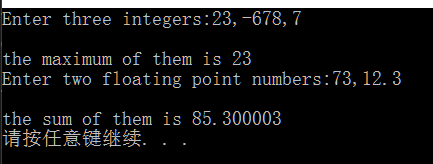


图4-5

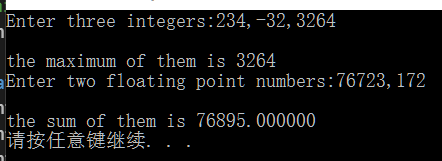


图4-6

第二问要求使用宏替代函数max实现求最大数的功能，可以考虑连续使用使用条件运算符，求出三个数字中的最大数，思路就是比较a和b，c中较大的比较，返回a或者b，c中较大的数，而b，c中较大的数也可以使用条件运算符求出，简短代码就是#define MAX(a, b, c) ((a)>((b)>(c)?(b):(c))?(a):((b)>(c)?(b):(c)))，可以完成任务。此处省略测试，结果与函数相同。

### 4.1.3跟踪调试

原题目见课本60页，程序功能是利用R计算圆的面积，并且取出面积的整数部分，源代码中的错误及改正如下：

// 1，添加程序需要的头文件

#include <stdio.h>

#include <assert.h>

#define R

// 2，给integer\_fraction添加声明

int integer\_fraction(float x);

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);

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

assert((s - s\_integer)<1.0);

#endif // R

return 0;

}

int integer\_fraction(float x) {

int i = x;

return i;

}

第二问，单步执行进入函数integer\_fraction时x的值为多少，返回main函数时i的值为多少，假定输入数据为4，结果如下：

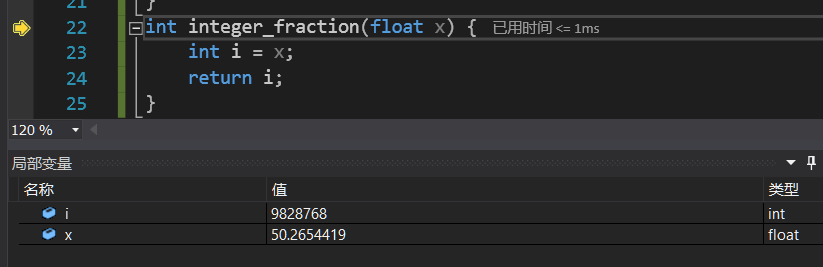


图4-7进入时的x的值

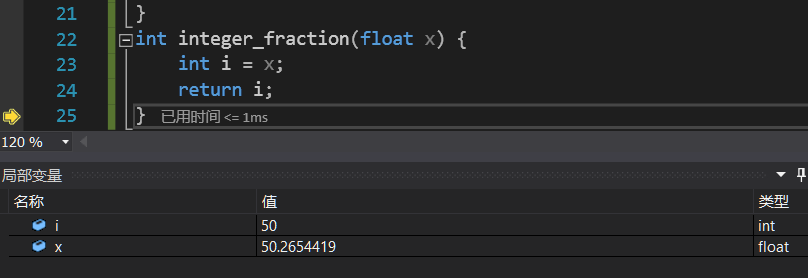


图4-8返回时i的值

### 4.1.4程序设计

第一小题，使用宏定义来求出三角形的面积，细节就是要使用math.h标准库里的sqrt函数实现开平方功能，而此函数参数为大于等于0的一个浮点数，因此在此之前需要检查输入的数字是否合法，是否为三角形的三边即是否满足两边之和大于第三边。流程图见图4-9：

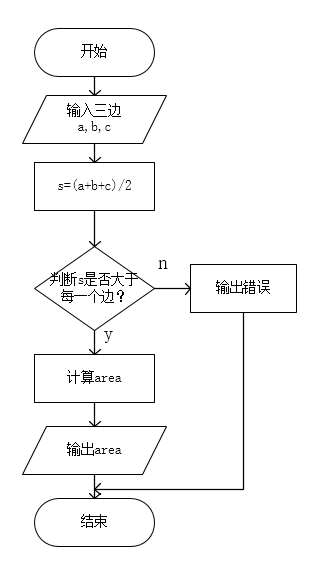


图4-9

#include <stdio.h>

#include <math.h>

// 定义S宏和area宏

#define S(a, b, c) (((a) + (b) + (c)) \* 0.5)

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

int main() {

double a = 0, b = 0, c = 0;

// 输入

printf("Input the triangle's three sides:");

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

float s = S(a, b, c);

// 判断数据是否合法

if (s - a > 0 && s - b > 0 && s - c > 0) {

printf("the area is %f\n", AREA(a, b, c, S(a, b, c)));

}

else {

printf("Numbers is illegal.\n");

}

return 0;

}

测试数据见表4-3

|  |  |
| --- | --- |
| 输入三边 | 输出面积 |
| 1 4 5 | 数据不合法 |
| 3 4 5 | 6 |
| 6 8 10 | 24 |

测试结果见图4-9，4-10，4-11：

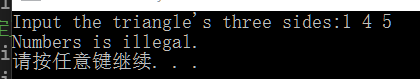


图4-9

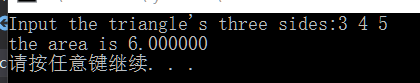


图4-10

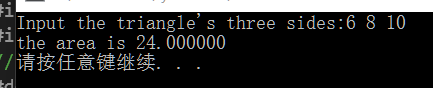


图4-11

第二题，输入一段电报文字，根据定义的宏控制输出的格式，如果CHANGE的值为1，则输出变换后的文字，0则按照原文输出。本题目的在于测试使用#if宏，条件为真就编译其包括的代码，否则不编译。

#include <stdio.h>

#define CNANGE 0

void Print(char\* str) {

#if CNANGE

char\*\_str = str;

// 当字符不为结束字符时

while (\*\_str) {

// 判断是否是英文字母

if (\*\_str >= 'a' && \*\_str <= 'z') {

printf("%c", \*\_str - 'a' + 'A');

}

else if (\*\_str >= 'A' && \*\_str <= 'Z') {

printf("%c", \*\_str - 'A' + 'a');

}

else {

printf("%c", \*\_str);

}

// 指向下一个字符

++\_str;

}

#else

printf("%s", str);

#endif // CNANGE

// 不要忘了换行

printf("\n");

}

int main(void) {

char str[100];

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

// 输入字符串

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

// 输出字符串

Print(str);

return 0;

}

测试数据见图4-12，4-13：

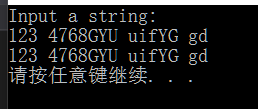


图4-12：change为0时

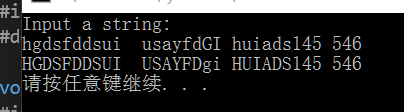


图4-13：change为1时

## 4.2小结

# 5数组实验

## 5.1必做题

### 5.1.1源程序改错

具体题目见课本第63页，本程序的目的是将数组中的元素按照升序排列，问题和改正如下：

#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, 10); // 1，参数要求是数组首地址，不是首元素

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 - 1 - i; ++j)

if (b[j] > b[j + 1]) // 2，升序输出应为大于号

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

}

改正过后运行结果如图5-1所示：

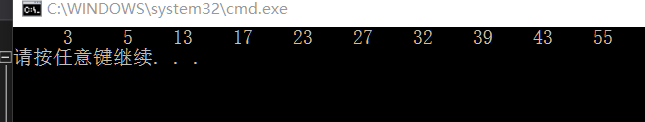


图5-1

### 5.1.2源程序完善、修改、替换

题目具体细节见课本64页，程序的功能是求解瑟夫问题，程序补全后如下：

#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)/\* 对圈中人顺序编号1~M \*/

a[i] = i + 1;

for (i = M, j = 0; i > 1; i--) {

/\* i表示圈内人数，初始化为M个，剩下一个人时结束循环；j表示当前报数人的位置 \*/

for (k = 1; k <= N; k++) {/\* 1到N报数 \*/

if (++j > i - 1)j = 0; /\* 最后一个人报数后第一个人接着报形成一个圈 \*/

}

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 \*/

//1. j为0表示不需要压缩，最后那个扔掉就是了

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]; //2. 类似插入排序，将报数为N的人从后面覆盖掉，所以填这个

}

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;

}

运行结果见图5-2：

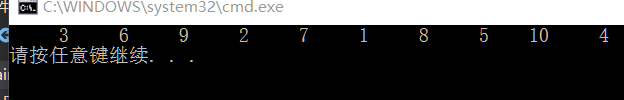


图5-2

可知最后剩下编号为4的人。

第二题要求改进改进，将压缩数组的操作压缩掉，具体实现我选择的方法是将被选中的人编号置为0（因为编号的范围不包括0，所以把0作为标记），在第一个for循环里就不能直接++j结束，而是需要找到下一个编号不为0的才行，在循环里引入一个while循环，用来找到满足条件的下标。具体修改如下：

#include <stdio.h>

#define M 10

#define N 3

int main(void) {

int a[M], b[M];

int i, j, k;

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

a[i] = i + 1;

for (i = M, j = 0; i > 1; i--) {

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

++j; // 下一个

if (j == M)j = 0; // 编号为M也就是0，走了一圈

while (a[j ? j - 1 : M - 1] == 0) { // 当没有找到符合的下标时

if (j == M - 1)j = 0; // 如果到了结尾下一个是0而不是M

else ++j; // 没有到尾部加一

}

}

b[M - i] = a[j ? j - 1 : M - 1]; // 和while里的保持一致

a[j ? j - 1 : M - 1] = 0; // 找到的地方置0

}

// 输出

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

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

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

if (a[i]) // 这个时候只有一个编号不为0

{

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

break;

}

}

return 0;

}

运行结果见图5-3：

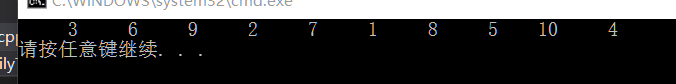


图5-3

可见修改没有错，运行结果和未修改时一样。

### 5.1.3跟踪调试源程序

题目具体细节见课本65页，函数strncat(s, t, n)功能为将字符串t的前n个字符链接到字符串s的尾部，现在需要找出程序的错误，并改正使其完成任务。第一问是查看进入函数后，光条落在for所在行时i的值，结果见图5-4：

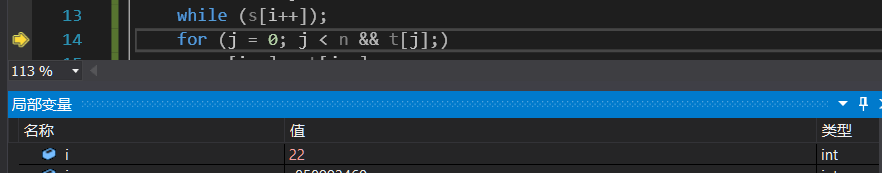


图5-4

此时观察可以知道i的值是22，在字符串s。里对应的是'\0'后面的那一个，由此可知第一个从t复制的字符串没有把'\0'覆盖掉，所以程序并不会输出正确的答案，解决方法就是在while循环结束后i的值减一，让它成为'\0'所在的下标，再进行覆盖就会把'\0'覆盖掉，输出结果就会符合要求了。运行结果见图5-5：

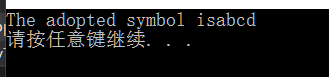


图5-5

结果正确。代码如下：

#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++]);

--i; // 将i减一

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

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

s[i] = '\0';

}

### 5.1.4程序设计

具体题目见课本65页，第一题是输入一个3x4的矩阵，按照要求输出原矩阵和它的转置矩阵，题目没有什么难度，只需要申请两个数组即可，输入的时候可以顺便将转置的矩阵也设置好，然后按照要求输出就好了。

#include <stdio.h>

#define ROW 3

#define COL 4

int main(void) {

int matrix[ROW][COL];

int resMar[COL][ROW];

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

// 每次输入4个数据

scanf("%d %d %d %d", &matrix[i][0], &matrix[i][1], &matrix[i][2], &matrix[i][3]);

// 顺便倒腾到转置矩阵里面去

resMar[0][i] = matrix[i][0], resMar[1][i] = matrix[i][1];

resMar[2][i] = matrix[i][2], resMar[3][i] = matrix[i][3];

}

for (int i = 0; i < ROW; i++) {// 输出原矩阵

printf("%d %d %d %d\n", matrix[i][0], matrix[i][1], matrix[i][2], matrix[i][3]);

}

printf("\n");

for (int i = 0; i < COL; i++) {// 输出转置矩阵

printf("%d %d %d\n", resMar[i][0], resMar[i][1], resMar[i][2]);

}

return 0;

}

输入数据和运行结果见图5-6：

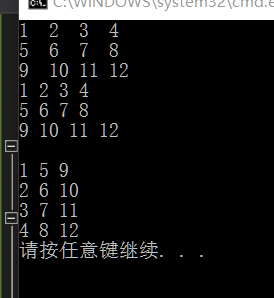


图5-6

可见运行正确。

第二题，要求输入一个整数，输出它在内存中二进制表示的每一位换成对应的数字字符，存放在一个字符数组中，最后输出。根据题意可以编写一个Print函数，接收一个整型参数，先将其转换为一个无符号整数，两者在内存中的表示是一样的，但是无符号的不会是负数，再从数组对应位置开始赋值一直到0处即可，代码：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

void Print(const int N) {

char chs[sizeof(int) \* 8 + 1] = { 0, }; // 根据系统选择chs大小

int count = sizeof(int) \* 8 - 1; // 调控字符数组的赋值

unsigned int temp = N;

while (count >= 0) { // 根据count来控制循环保证是32个0~31

chs[count--] = temp % 2 + '0';

temp /= 2;

}

printf("%s\n", chs); // 输出

}

int main(void) {

int N;

scanf("%d", &N); // 输入数据

Print(N);

return 0;

}

测试数据见表5-1：

|  |  |
| --- | --- |
| 输入 | 输出 |
| 1 | 00000000000000000000000000000001 |
| 1024 | 00000000000000000000010000000000 |
| 26 | 00000000000000000000000000011010 |
| 0 | 00000000000000000000000000000000 |
| -1 | 11111111111111111111111111111111 |
| 65 | 00000000000000000000000001000001 |
| 4 | 00000000000000000000000000000100 |
| -1024 | 11111111111111111111110000000000 |

测试结果如下：

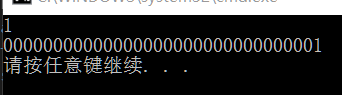


图5-7

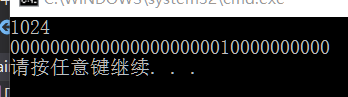


图5-8

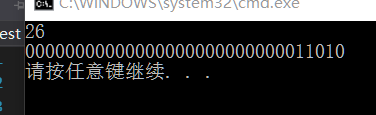


图5-9

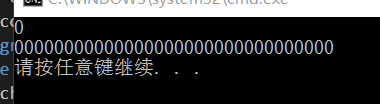


图5-10

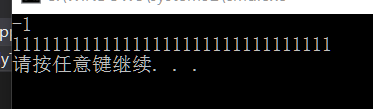


图5-11

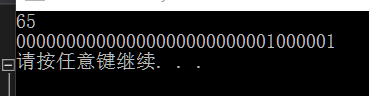


图5-12

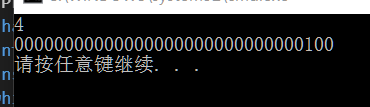


图5-13

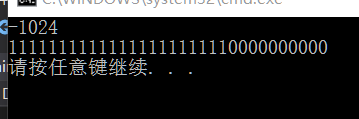


图5-14

答案正确。

第三题要求编写一个程序对输入的学生成绩进行排序并输出，再按照成绩查找学生是否存在，具体实现可以使用结构体减小输入输出和排序时的负担，使用快速排序和二分查找提高效率，快速排序和二分查找的原理网上都有，具体细节可以阅读代码：

#include <stdio.h>

#include <malloc.h> // 开辟空间

// 使用结构体，简化运算

typedef struct Stu {

char name[20]; // 名字

int sco; // 分数

}Stu;

typedef Stu \*pStu;

// 输入数据

void InputStu(pStu stus, const int N) {

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

scanf("%s %d", stus[i].name, &stus[i].sco);

}

}

// 快排提高效率

void Sort(pStu stus, const int start, const int end) {

if (start >= end) {

return;

}

Stu key = stus[start];

int s = start;

int e = end;

while (s < e) {

while (stus[e].sco <= key.sco && s < e) --e;

stus[s] = stus[e];

while (stus[s].sco >= key.sco && s < e)++s;

stus[e] = stus[s];

}// 结束时\_end\_start

stus[e] = key;

Sort(stus, start, s - 1);

Sort(stus, s + 1, end);

}

// 按要求输出

void Print(pStu stus, int N) {

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

// %-20s

printf("%-20s %d\n", stus[i].name, stus[i].sco);

}

printf("\n");// 要求的空行

}

// 查找

void Search(pStu stus, const int N, const int sear) {

// result存储结果顺便充当是否查找到的标志

int result = 0;

int mid = N / 2;

int first = 0;

int last = 0;

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

first = 0;

last = N - 1;

result = -1;

// 由于数据已经有序，所以使用二分查找

while (first <= last) {

mid = (first + last) / 2;

if (stus[mid].sco == sear) {// 找到结束

result = mid;

break;

}

else if (stus[mid].sco > sear) {

first = mid + 1;

}

else {

last = mid - 1;

}

}

if (result != -1) {

printf("%-20s %d\n", stus[result].name, stus[result].sco);// 打印结果

}

else {

printf("Not found!\n");

}

}

}

int main(void) {

int N = 0;

printf("请输入学生个数：");

scanf("%d", &N);

pStu stus = (pStu)malloc(N \* sizeof(Stu));// 开辟数组

InputStu(stus, N); // 输入

Sort(stus, 0, N - 1); // 排序

Print(stus, N); // 打印排好的

int sear = 0;// 要查找的人

printf("请输入要查找的学生的分数：");

scanf("%d", &sear);

Search(stus, N, sear);

free(stus);

return 0;

}

测试结果见图5-15，5-16：

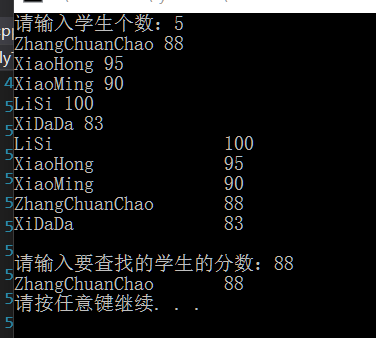


图5-15

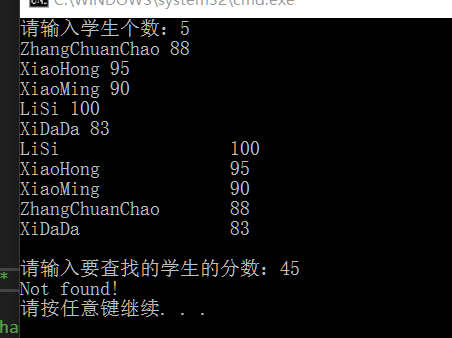


图5-16

运行正确。

## 5.2选做题

第一题要求实现函数strnins(s, t, n)将字符数组t中的字符串插入到字符数组s中的第n个字符后面，具体实现时具体问题会有插入位置越界等，而且目标字符串大小能否存储要插入的字符串也未知，这就要求调用者注意了。细节主要是开辟新数组存储目标字符串将要被覆盖的字符串，将要插入的字符串接到指定位置，将原来的字符串接回去，为了程序安全性我返回一个int值来表示是否插入成功，0表示失败，1表示成功。具体代码如下：

int strnins(char\* s, char\* t, int n) {

char temp[1000]; // 存储要被覆盖的部分

if (n > (int)strlen(s)) {

printf("目标字符串长度不足！\n");

return 0;

}

if (n < 0) {

printf("插入位置应该不小于0！\n");

return 0;

}

unsigned len = strlen(s);

// 从N开始到len的都需要往相应的地方移动

for (unsigned i = n; i <= len; ++i) {

temp[i - n] = s[i];

}

// 插入字符串

for (unsigned i = 0; i <= strlen(t); ++i) {

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

}

// 将原来的补回去

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

s[n + strlen(t) + i] = temp[i];

}

// 插入成功返回1

return 1;

}

测试及其结果见图5-17，5-18，5-19：

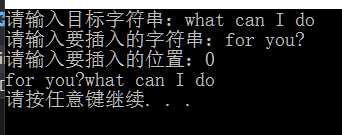


图5-17

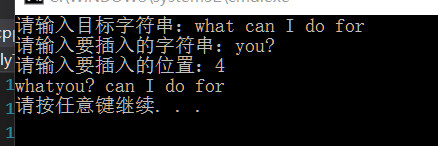


图5-18

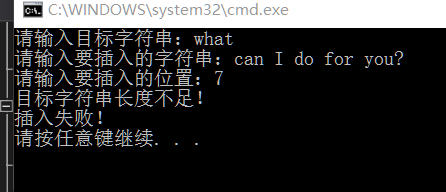


图5-19

运行正确。

第二题要求编写实现八皇后的程序，为了程序的可读性和代码的便于书写，我选择使用DFS来进行求解，当一个皇后的位置满足其条件时就向下一层递归调用 DFS，如果这一层已经是第八层就可以记录下数据，不需要继续递归，当递归进行完了也就得到所有答案（鉴于本人水平暂时还不会去除重解，即轴对称的和旋转对称的解），由于解的个数有点多，所以我新建了一个txt文件存储结果，具体代码如下：

#include <stdio.h>

#include <stdlib.h>

#define MAX 8 // 设置皇后总数

#define true 1

#define false 0

FILE\* file = NULL;

typedef int bool;

int count = 0; // 全局记录解的总数

// 检测该深度时是否满足

bool Check(int\* arr, int depth) {

int val\_dif = 0; // 左右差

int ind\_dif = 0; // 上下差

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

for (int j = i + 1; j <= depth; j++) {

val\_dif = arr[i] - arr[j];

ind\_dif = i - j;

if (val\_dif == 0 || val\_dif == ind\_dif || val\_dif == -ind\_dif) {

return false;

}

}

}

return true;

}

void DFS(int \*arr, const int now) {

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

arr[now] = i;

if (Check(arr, now)) {// 如果已经是第八个并且满足条件，打印

if (now == MAX) {

printf("%d", arr[1]);

fprintf(file, "%d", arr[1]);

for (int i = 2; i <= MAX; i++) {

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

fprintf(file, " %d", arr[i]);

}

printf("\n");

fprintf(file, "\n");

++count;

continue; // 不需要到下一层继续找了

}// 未到第八层且前面的几层可以满足条件

DFS(arr, now + 1);

}

}

}

int main(void) {

file = fopen("result.txt", "w");

if (file == NULL) {

printf("打开文件失败！\n");

exit(0);

}

fprintf(file, "本文件输入意思为一行就是一个解，由从第一个皇后到第8个在其所在行的位置组成。\n");

int a[MAX + 1] = { 0, }; // 定为9维，第一维不要啦，写着方便

DFS(a, 1); // 从第一层开始找

printf("一共%d种情况。\n", count);

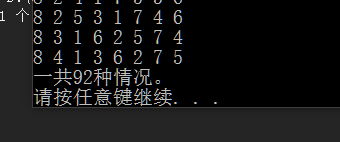
fprintf(file, "一共%d种情况。\n", count);

fclose(file);

return 0;

}

运行结果见图5-20：



生成的包含解的txt文件请见附件result.txt。

## 5.4小结

# 6指针实验

## 6.1必做题

## 6.2选做题

## 6.3自设题

## 6.4小结

# 7结构与联合实验

## 7.1必做题

## 7.2选做题

## 7.4小结

# 8文件实验

## 8.1必做题

## 8.2自设题

## 8.3小结

# 参考文献

[1]曹计昌,卢萍,李开. C语言程序设计,北京：科学出版社,2013

[2]李开,卢萍,曹计昌. C语言实验与课程设计,北京：科学出版社,2011