

**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" \l "_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" \l "_Toc404837921)

[1.2小结 13](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837923)

**[2流程控制实验 1](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837924)4**

[2.1必做题 1](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837925)4

[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" \l "_Toc404837926)9

[2.3小结 3](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837928)1

**[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" \l "_Toc404837929)2**

[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" \l "_Toc404837930)2

[3.2选做题 43](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837931)

[3.3自设题 43](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837932)

[3.4小结 43](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_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" \l "_Toc404837934)4**

[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" \l "_Toc404837935)4

[4.2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" \l "_Toc404837936)3

[4.3小结 5](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837937)3

**[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" \l "_Toc404837938)4**

[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" \l "_Toc404837939)4

[5.2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" \l "_Toc404837940)7

[5.3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" \l "_Toc404837941)7

[5.4小结 7](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837942)1

**[6指针实验 7](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837943)2**

[6.1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" \l "_Toc404837944)2

[6.2选做题 86](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837945)

[6.3自设题 86](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837946)

[6.4小结 86](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837947)

**[7结构与联合实验 87](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837948)**

[7.1必做题 87](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837949)

[7.2选做题 9](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837950)6

[7.3自设题 9](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837951)9

[7.4小结 9](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837952)9

**[8文件实验 1](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837953)00**

[8.1必做题 1](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837954)00

[8.2自设题 1](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837955)05

[8.3小结 1](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837956)05

**[参考文献 1](file:///D:\\Yaning%20Wang\\Documents\\作业\\《C语言程序设计实验》%20-%20副本.doc" \l "_Toc404837957)06**

# 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：

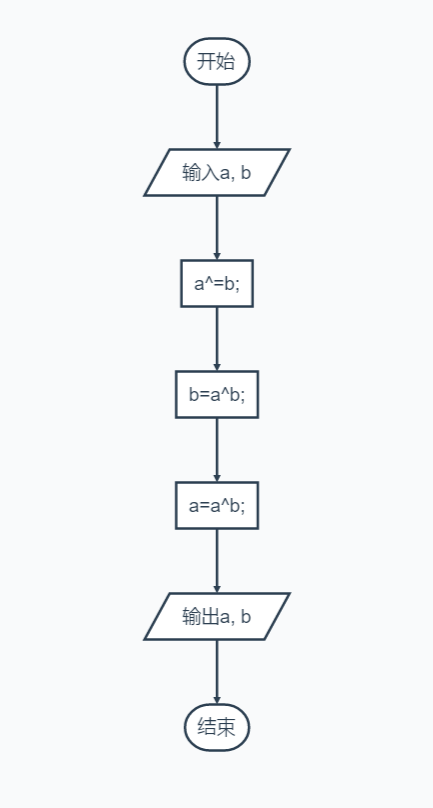


图1-1

#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

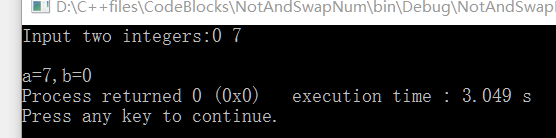


图1-2

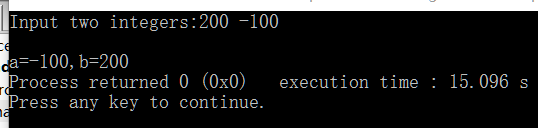


图1-3

### 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：

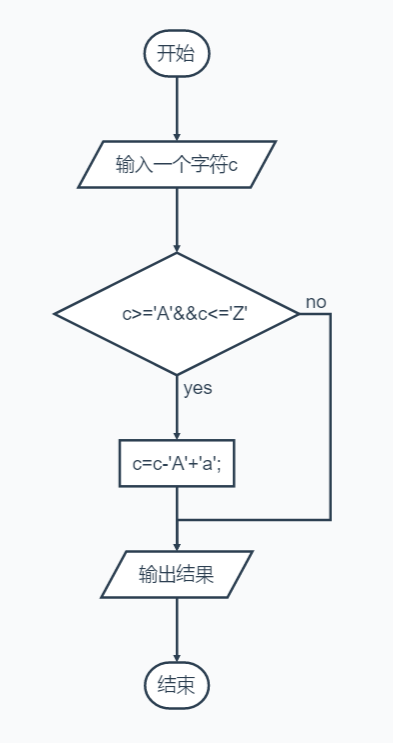


图1-4

#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：

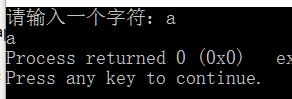


图1-5

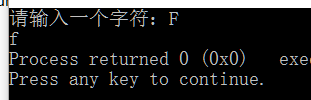


图1-6



图1-7

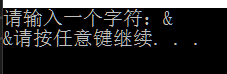


图1-8

【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：

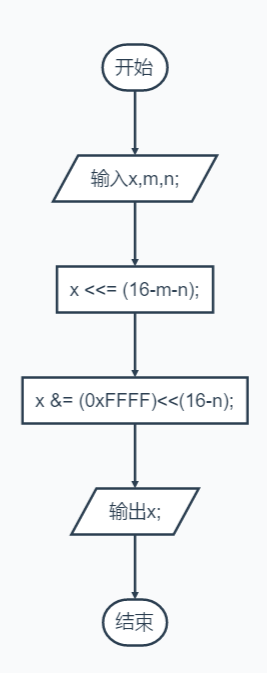


图1-9

#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：

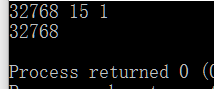


图1-10

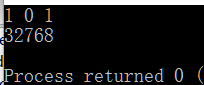


图1-11



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：

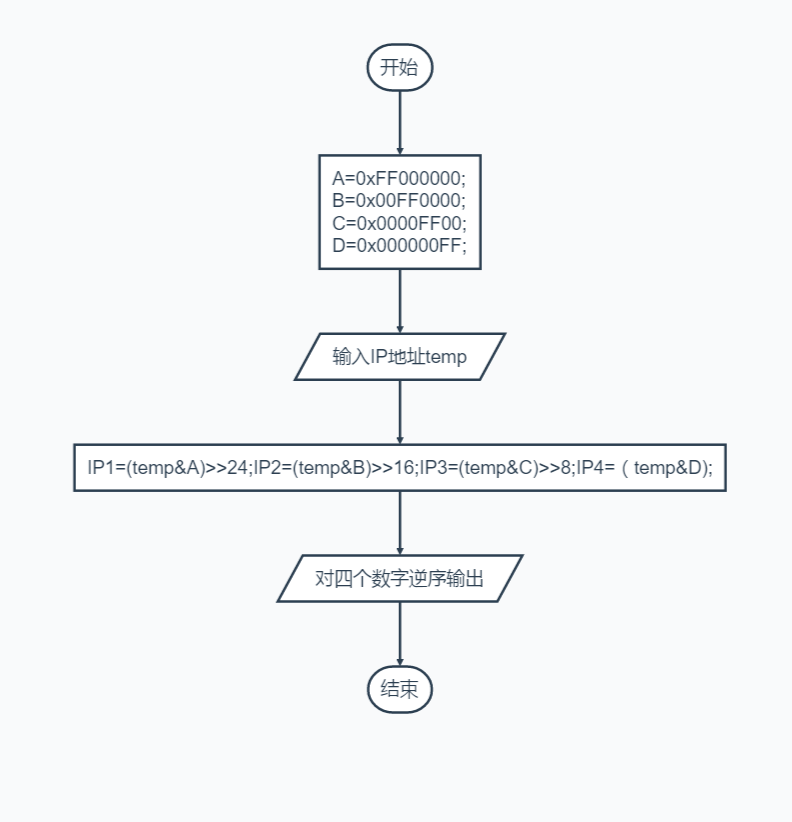


图1-13

#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：

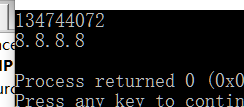


图1-14

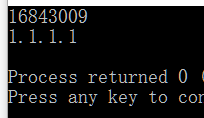


图1-15

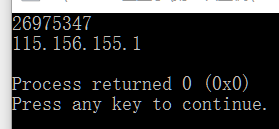


图1-16

## 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：

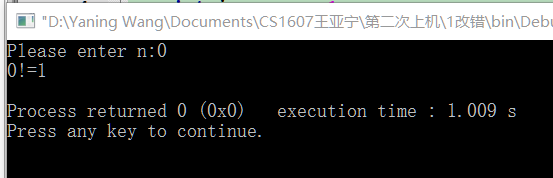


图2-1

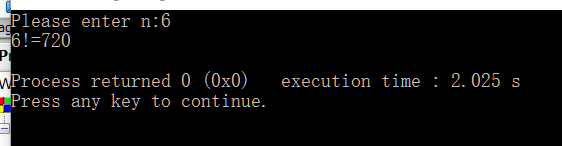


图2-2

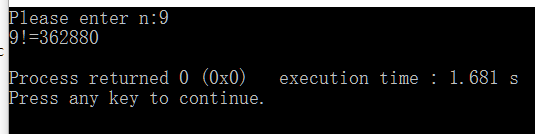


图2-3

### 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：

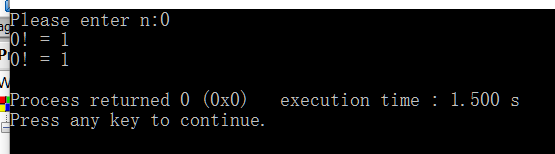


图2-4

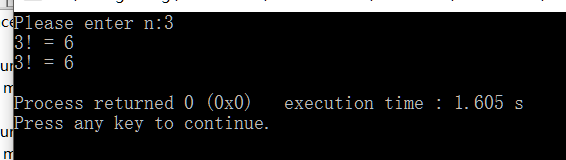


图2-4

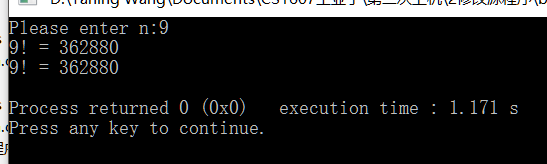


图2-5

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。

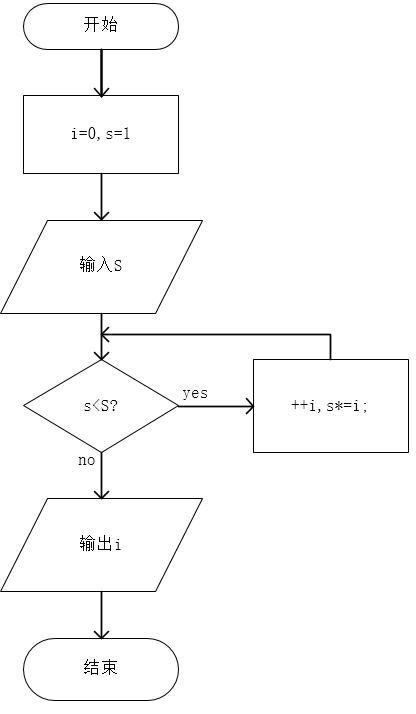


图2-7

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

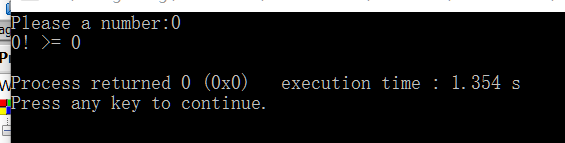


图2-8

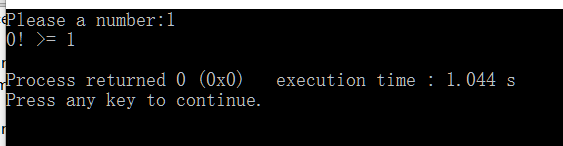


图2-9

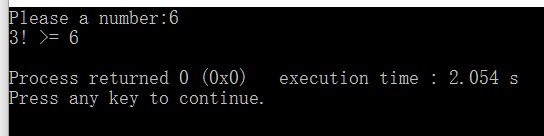


图2-10

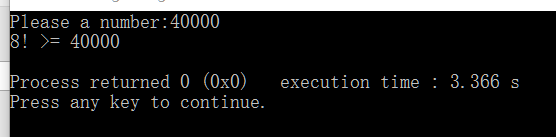


图2-11

### 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：

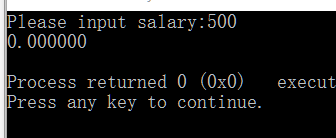


图2-13

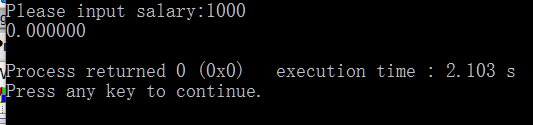


图2-14

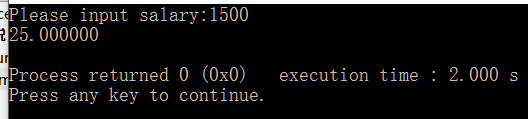


图2-15

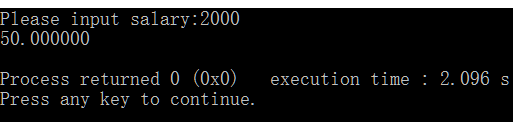


图2-16

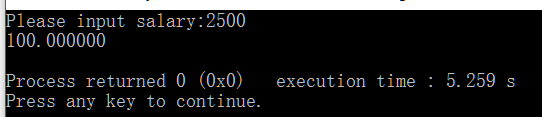


图2-17

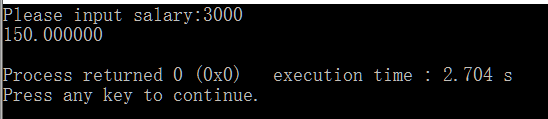


图2-18

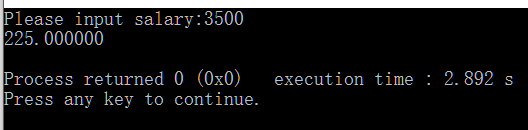


图2-19

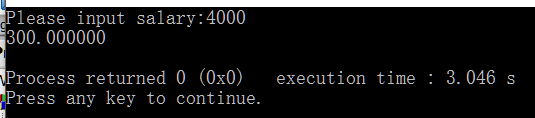


图2-20

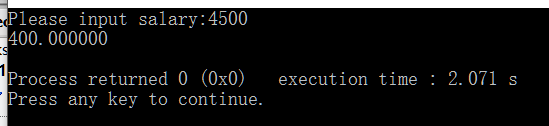


图2-21

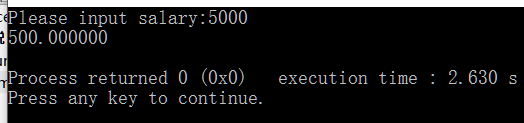


图2-22

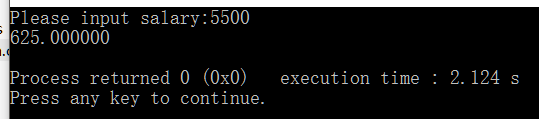


图2-23

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：

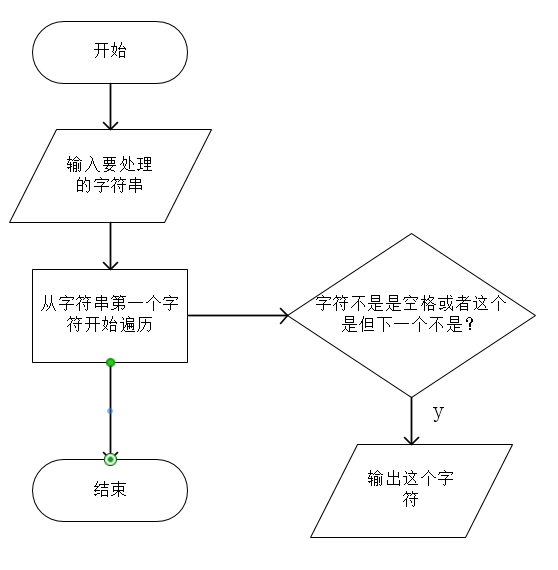


图2-24

#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：

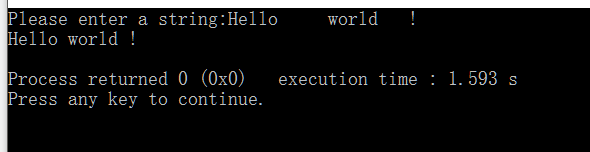


图2-25

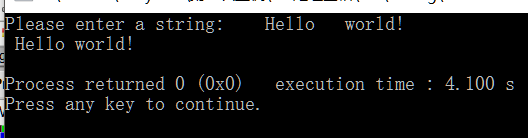


图2-26

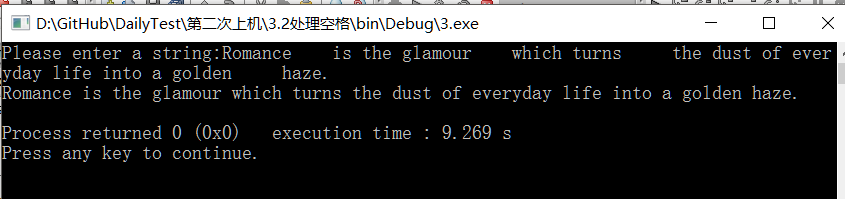


图2-27

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：

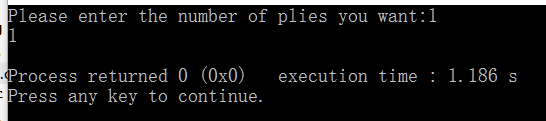


图2-29

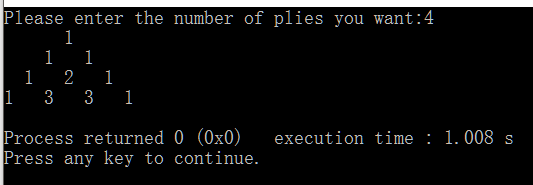


图2-30

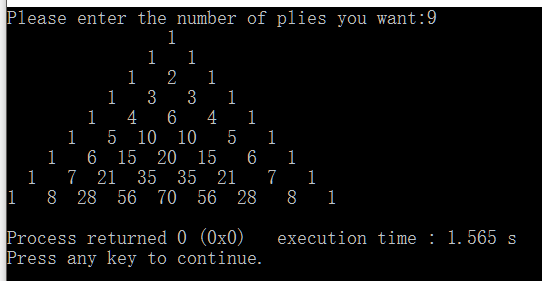


图2-31

## 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：

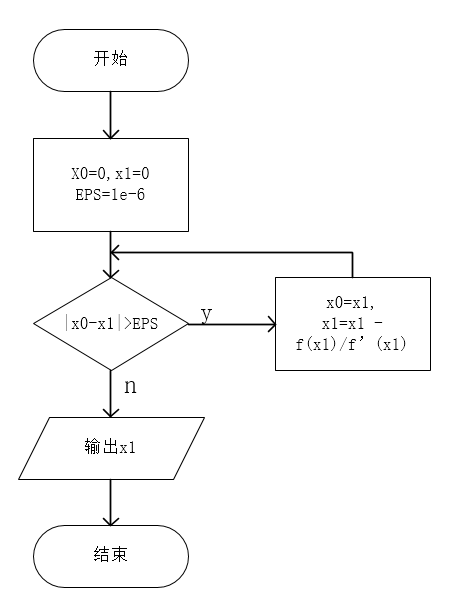


图2-32

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:

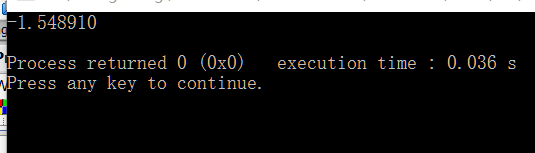


图2-33

## 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：

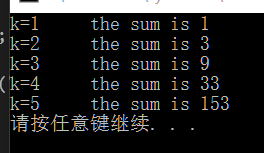


图3-1

### 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：

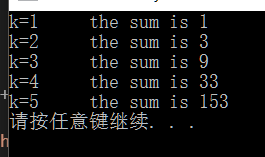


图3-2

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：

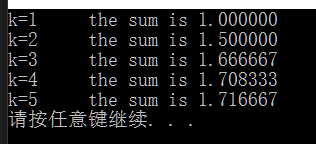


图3-3

### 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：

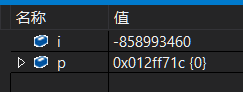


图3-4

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：

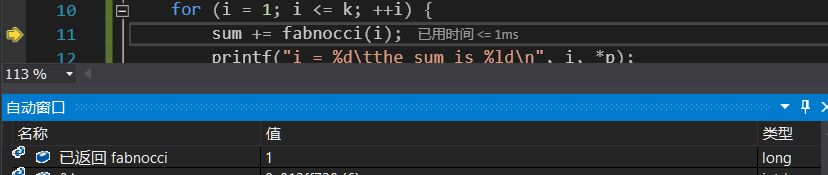


图3-5

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：

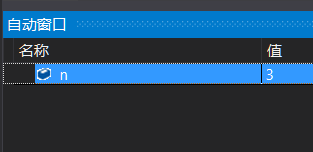


图3-6

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

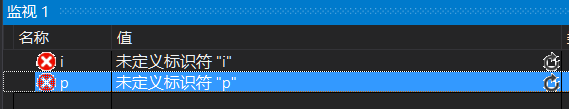


图3-7

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

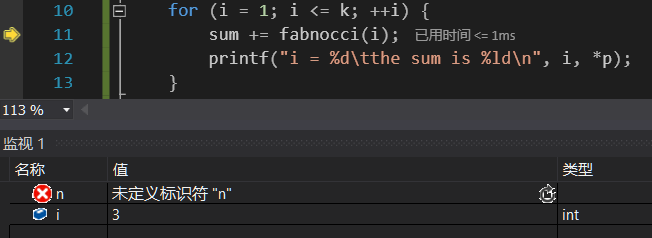


图3-8

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：

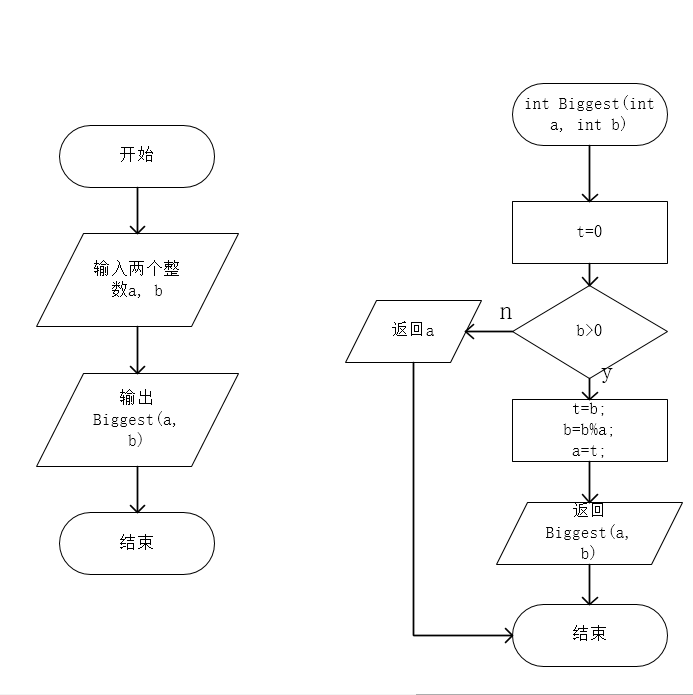


图3-9

#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：

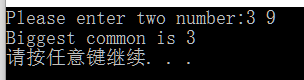


图3-10

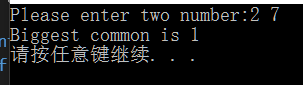


图3-11

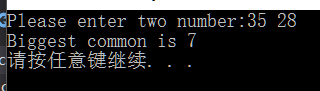


图3-12

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：

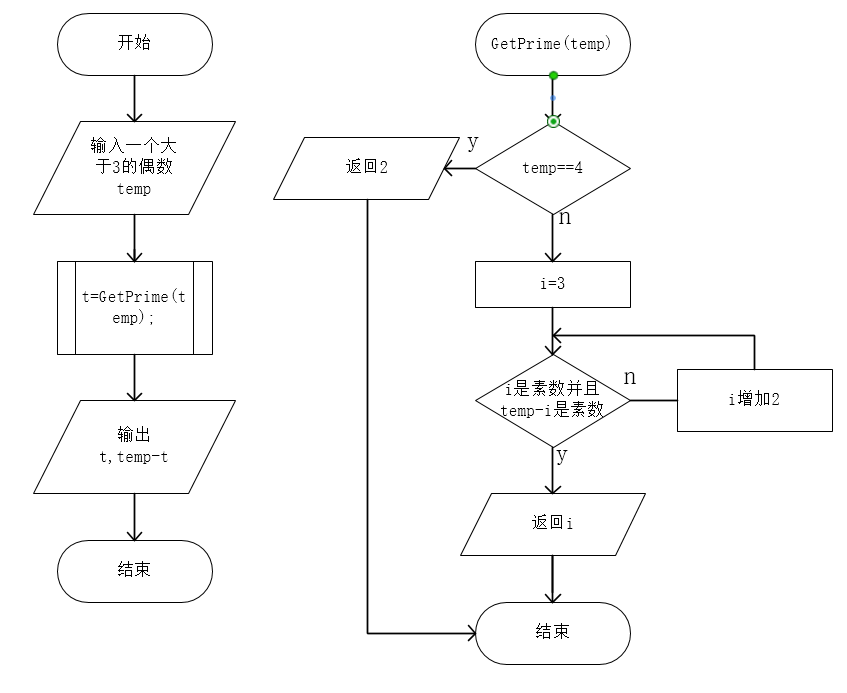


图3-13

#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：

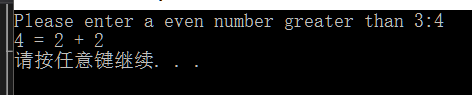


图3-14

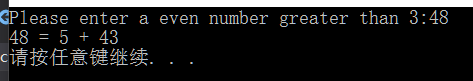


图3-15

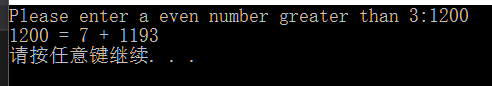


图3-16

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

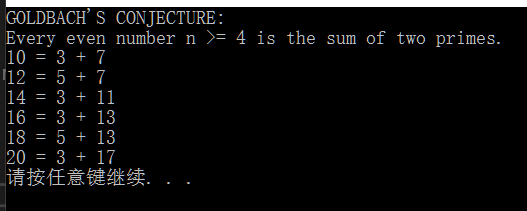


图3-17

## 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：

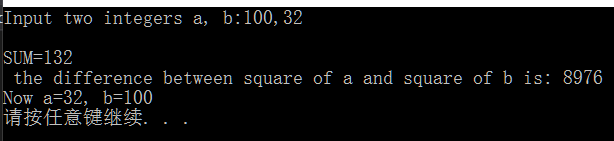


图4-1

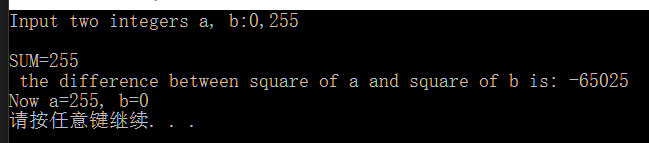


图4-2

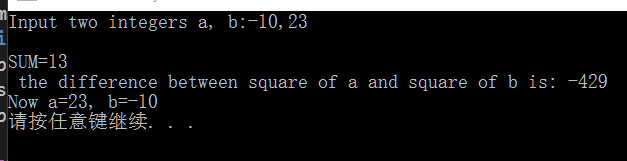


图4-3

### 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

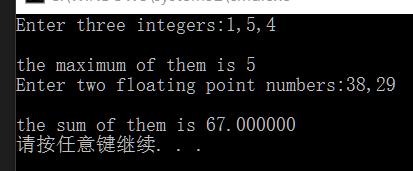


图4-4

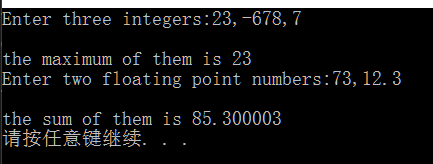


图4-5

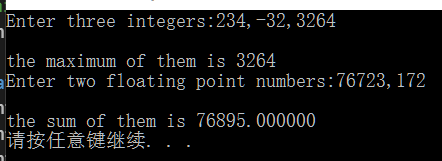


图4-6

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：

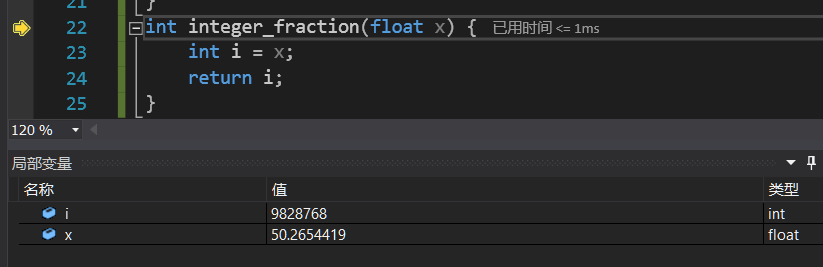


图4-7进入时的x的值

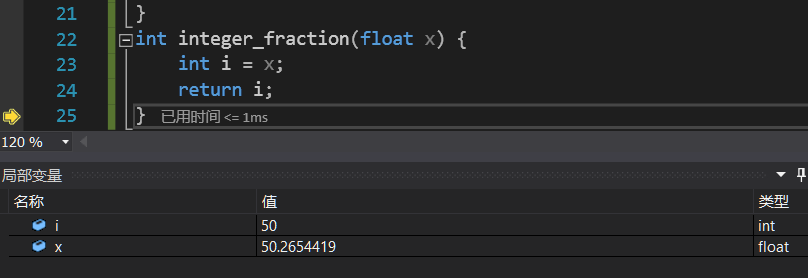


图4-8返回时i的值

### 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：

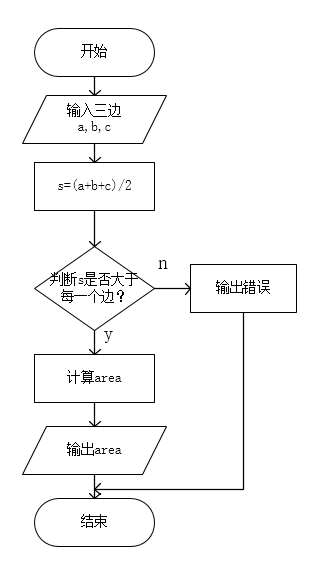


图4-9

#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：

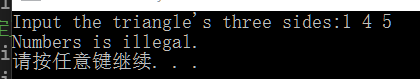


图4-9

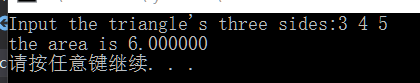


图4-10

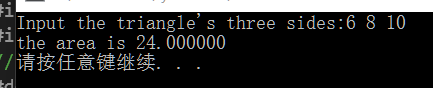


图4-11

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：

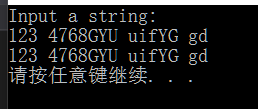


图4-12：change为0时

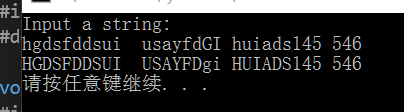


图4-13：change为1时

## 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：

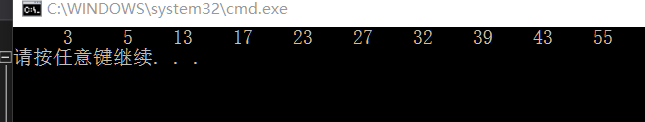


图5-1

### 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：

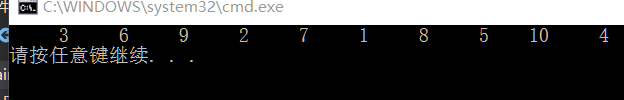


图5-2

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：

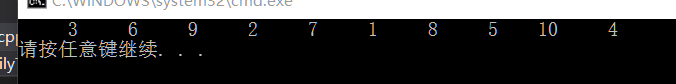


图5-3

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：

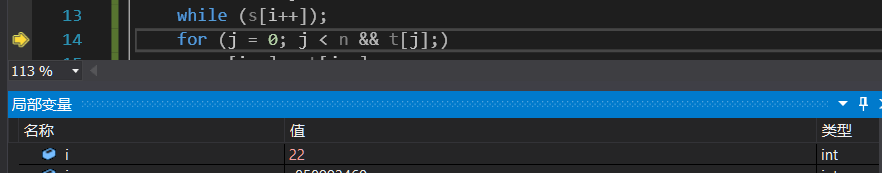


图5-4

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：

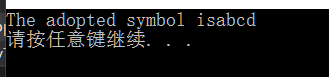


图5-5

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：

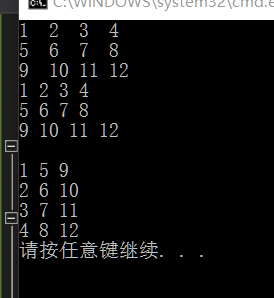


图5-6

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：

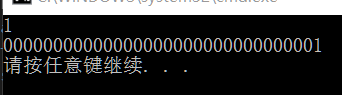


图5-7

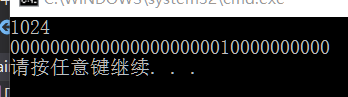


图5-8

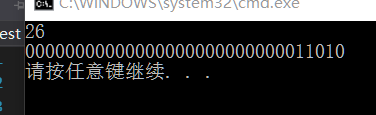


图5-9

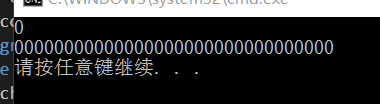


图5-10

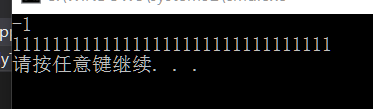


图5-11

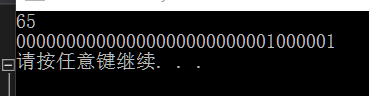


图5-12

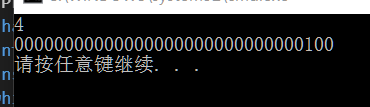


图5-13

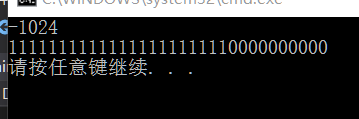


图5-14

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：

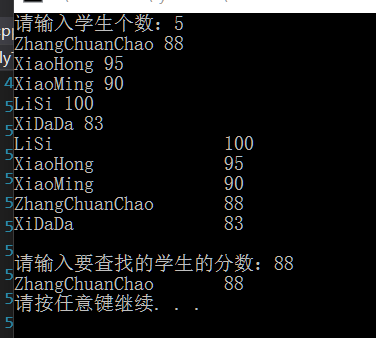


图5-15

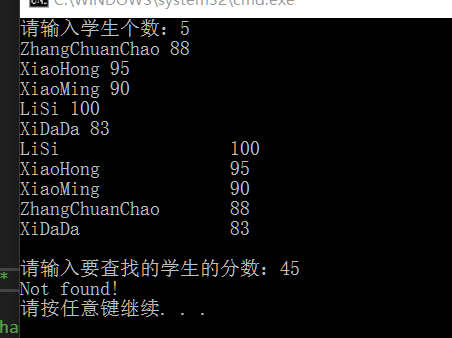


图5-16

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：

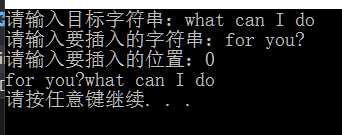


图5-17

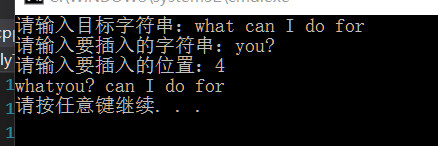


图5-18

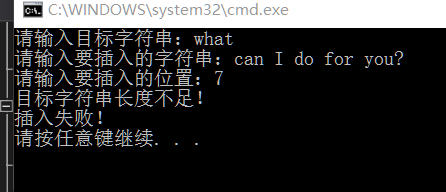


图5-19

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);

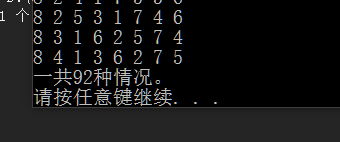
fprintf(file, "一共%d种情况。\n", count);

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

### 6.1.1源程序改错

题目细节见课本66页。本题要求修改源程序中的错误，完成程序的功能，经过检查发现只有一处错误，修改后如下：

#include<stdio.h>

int main(void){

float t = 0;

float \*p = &t; // 初始化

scanf("%f", p); // 1，p是野指针，没有指向正确的地方，应该为其初始化

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

return 0;

}

题目很简单，因此不进行测试。

### 6.1.2源程序完善、修改、替换

具体题目细节见课本66页。第一题要求补全源程序，使之能按要求完成任务，补全后代码如下：

#include<stdio.h>

#include<string.h>

int main(void){

char\*(\*p)(char[], const char[]); // 1，下面用到了函数指针p，在此处定义，可推断参数为2个字符指针，返回值为一个字符指针

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

int choise, i;

do{

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

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

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

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

scanf("%d", &choise);

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

switch(choise){

case 1:

p = strcpy;

break;

case 2:

p = strcat;

break;

case 3:

goto down;

}

getchar();

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

i = 0;

scanf("%[^\n]", a), getchar(); //2， 在此处输入第一个字符串

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

i = 0;

scanf("%[^\n]", b), getchar(); // 3，在此处输入第二个字符串

result = p(a, b); //4，此处调用函数

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

down:

;

return 0;

}

第二题就要测试源程序，测试如下，输入2和两个字符串，把它们拼接起来。运行结果如下图6-1：

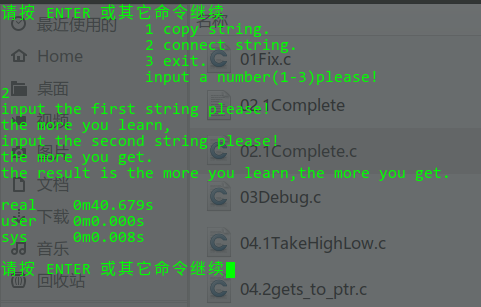


图6-1

运行结果正确。

### 6.1.3跟踪调试源程序

题目具体细节见课本，第一题要求查看进入strcpy函数时watches窗口中s的值，返回main函数时s的值，值见图6-2,6-3。

03.1

图6-2

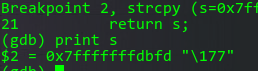


图6-3

可见两处s的值都为地址，但是地址的值已经发生了改变，这是因为在while循环里对s进行了自增操作，所以它的值会变大。最终输出为一个空行。

具体修改如下：

#include<stdio.h>

char \* strcpy(char\*, char\*);

int main(void){

char a[60], b[60] = "there is a boat on the lake."; // 1，a字符数组长度不够，调大一点

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

return 0;

}

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

// 2，返回的不是字符数组首地址，可以设置一个指针专门存放首地址

char \*str = s;

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

;

return(str);

return s;

}

运行结果见图6-4：

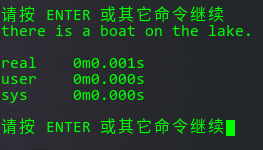


图6-4

结果正确。

### 6.1.4程序设计

具体题目见课本68页。第一题要求将一个输入的长整数，每个长整数都由4个字节组成，每个字节都由高四位和低四位，程序要求从变量的高字节开始，依次取出每个字节的高四位和低四位并以数字字符的形式进行输出。具体实现可以使用位运算和无符号整数，无符号可以避免处理符号正负的问题，使用8个逻辑尺分别取出对应的位置的数字并进行移位输出。

#include<stdio.h>

#define A 0xf0000000

#define B 0x0f000000

#define C 0x00f00000

#define D 0x000f0000

#define E 0x0000f000

#define F 0x00000f00

#define G 0x000000f0

#define H 0x0000000f

void ToChar(unsigned int N){

int i = 0;

const int count = 5;

char str[count] = { 0, };

i = count - 2; // 初始化i在正确地方

while (i >= 0){

str[i] = N % 2 + '0';

N /= 2;

--i;

}

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

}

int main(void){

// printf("%lu\n", sizeof(int));

unsigned int t = 0;

printf("Please enter an integer:");

scanf("%ud", &t);

ToChar((t&A)>>28);

ToChar((t&B)>>24);

ToChar((t&C)>>20);

ToChar((t&D)>>16);

ToChar((t&E)>>12);

ToChar((t&F)>>8);

ToChar((t&G)>>4);

ToChar((t&H)>>0);

return 0;

}

测试数据见表6-1：

|  |  |
| --- | --- |
| 次数 | 输入 |
| 1 | 1 |
| 2 | -1 |
| 3 | 123456789 |

表6-1

输出结果见图6-5,6-6,6-7

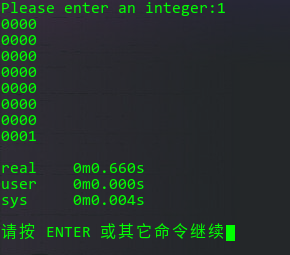


图6-5



图6-6



图6-7

第二题为使用gets输入几行字符串，使用函数把每行中的多个连续空格字符合并为1个，并将处理过的输出，空行不予输出。处理时首先要判断字符串的长度，可以使用标准库的strlen函数，字符串长度大于0就不会是空行，再对连续空格判断时，可以判断本字符是否是空格，如果不是就可以输出，是的话判断下个字符是否是空格，是的话则表示2个空格连在了一起，不予输出，不是则说明不是两个空格字符连接在一起，输出这个空格就可以了。

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void concen\_space(char (\*str)[81], int num){

int temp = 0;

int len = 0;

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

temp = 0;

len = strlen(str[i]);

// 首先判断长度，0的话表示是空行，不用输出

if (len == 0){

continue;

}

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

// if 判断是否满足输出的条件

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

printf("%c", str[i][temp]);

}

}

printf("\n");

}

}

int main(void){

int N; // n行

printf("请输入行数：");

scanf("%d", &N);

getchar();

char strs[100][81];

char (\*pstr)[81] = strs;

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

gets(strs[i]);

}

concen\_space(pstr, N);

return 0;

}

测试见图6-8,6-9,6-10：

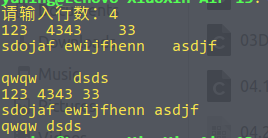


图6-8

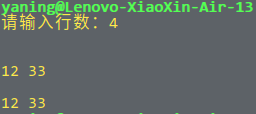


图6-9

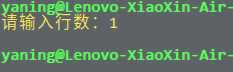


图6-10

第三题要求输入n个数，凭借main函数的参数的个数，来判断使用升序排列还是降序排列，当main函数不带有参数-d时，它的argc参数值是1,当带有-d时main函数里的argc值为2,可以使用这个来判断应该使用的排序方式，具体的排序就可以使用一个函数指针来指定排序的比较方式，所以思路就就是先输入数组，再依据argc的值来选择比较大小的方式，再使用排序算法排序，

#include<stdio.h>

#include<stdlib.h>

// 有参数减序

int Desce(int a, int b){

return a <= b; //至于为什么要等于，这是经过教训得出的

}

// 无参数升序

int Ascen(int a, int b){

return a >= b;

}

void Input(int\*arr, int N){

for (int i = 0; i < N; ++i)scanf("%d", &arr[i]);

}

void Print(int\* arr, int N){

printf("%d", arr[0]);

for (int i = 1; i < N; ++i)printf(" %d", arr[i]);

printf("\n");

}

void Sort(int \*arr, const int start, const int end, int (\*comp)(int a, int b)){

if (start >= end)return;

int key = arr[start]; // 选取中间数

int s = start;

int e = end;

while(s < e){

while(comp(arr[e], key) && s < e)--e; // 当key和arr[]满足大小比较时继续往下走

arr[s] = arr[e]; // 不满足了交换

while(comp(key,arr[s]) && s < e)++s;

arr[e] = arr[s];

}

arr[s] = key; // 将key放在制定位置

Sort(arr, start, s - 1, comp); // 递归向下

Sort(arr, s + 1, end, comp);

}

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

// printf("argc = %d\n", argc);

int \*arr = NULL;

int N = 0;

scanf("%d", &N);

arr = (int\*)malloc(N \* sizeof(int));

Input(arr, N);

// argc 为 1 没有-d，2有

Sort(arr, 0, N - 1, (argc - 1) ? Desce : Ascen);

Print(arr, N);

free(arr);

return 0;

}

测试数据见表6-2（第一个数据是数组大小）：

|  |
| --- |
| 输入 |
| 9 1 2 3 4 5 4 3 2 1 |
| 12 1 2 7 3 4 2 6 7 7 3 4 3 |
| 6 6 5 7 8 4 3 |

运行结果见图6-11：



图6-11

第四题要求编写程序统计学生成绩，一共N名学生，M门课程，为了方便起见，可以直接定义全局的数组存放学生和课程平均成绩等信息，方便传递参数，再具体使用嵌套的for循环计算所求的平均值、及格人数等数据，测试数据选择定义2门课，4名学生。代码如下：

#include<stdio.h>

#include<stdlib.h>

#define M 2 //课程

#define N 4 //人数

#define GOOD 90

#define BAD 60

double ave\_stu[N];//所有学生平均

double ave\_cla[M]; // 课程平均

int low\_cla[M]; // 低于班级平均

int low\_ave[M]; // 低于课程平均

int good[M]; // 好学生

int bad[M]; // 坏学生

// 计算每个学生的平均分

void AveStu(double (\*scores)[M]){

double sum;

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

sum = 0;

for (int cla = 0; cla < M; ++cla){

sum += \*(\*(scores + stu) + cla);

}

ave\_stu[stu] = sum / M;

}

}

// 计算每门课的平均分

void AveCla(double (\*scores)[M]){

double sum = 0;

for (int cla = 0; cla < M; ++cla){

sum = 0;

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

sum += \*(\*(scores + stu) + cla);

}

\*(ave\_cla + cla) = sum / N;

}

}

// 计算低于班级平均分的人数

void LowClass(double (\*scores)[M]){

for (int cla = 0; cla < M; ++cla){

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

if (\*(\*(scores + stu) + cla) < \*(ave\_cla + cla)){

++low\_cla[cla];

}

}

}

}

// 计算每门课超过了90分的人数和低于60分的人数

void BadAndGood(double (\*scores)[M]){

for (int cla = 0; cla < M; ++cla){

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

if (\*(\*(scores + stu) + cla) < BAD){

++\*(bad + cla);

}

if (\*(\*(scores + stu) + cla) >= GOOD){

++\*(good + cla);

}

}

}

}

// 没有重载只好这么写了，其实应该写模板的

void Print(double \*arr, int n){

printf("%f", arr[0]);

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

printf(" %f", \*(arr + i));

}

printf("\n");

}

void \_Print(int \*arr, int n){

printf("%d ", arr[0]);

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

printf(" %d", \*(arr + i));

}

printf("\n");

}

int main(void){

double scores[N][M]; // N个学生，M门课程

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

printf("请输入第%d名学生的%d门课的成绩：", i + 1, M);

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

scanf("%lf", \*(scores + i) + j);

}

}

AveStu(scores);

printf("每个学生平均：");

Print(ave\_stu, N);

AveCla(scores);

printf("每门课程平均：");

Print(ave\_cla, M);

LowClass(scores);

printf("每门课低于班级平均：");

\_Print(low\_cla, N);

BadAndGood(scores);

printf("每门课里分数不低于90的学生：");

\_Print(good, M);

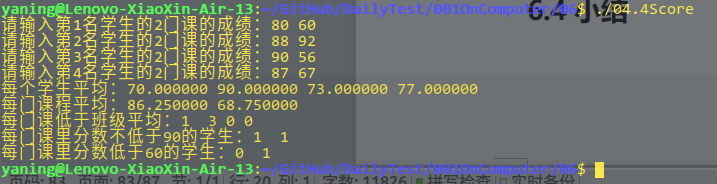
printf("每门课里分数低于60的学生：");

\_Print(bad, M);

return 0;

}

具体测试数据及结果如下：



## 6.2选做题

## 6.3自设题

## 6.4小结

指针，C语言中迷人又让人害怕的部分，有人说指针就是C的灵魂，C语言指针的用法复杂多样，那么多的定义都能让人懵圈，深入理解之后也能发现其实指针也不过是变量的一种罢了，C指针中我们比较容易犯的错估计就是使用野指针或者越界访问了，学好C的指针有助于我们在硬件的层面理解计算机的一些运行原理。

# 7结构与联合实验

## 7.1必做题

### 7.1.1表达式求值的程序验证

具体题目细节见课本70页，题目给定一个结构体的定义，并且为其定义了几个结构体还有一个结构体指针，要求计算给出的表达式的值，细节如下：

char u[] = "UVWXYZ";

char v[] = "xyz";

struct T{

int x;

char c;

char\*t;

}a[] = {{ 11, 'A', u }, { 100, 'B', v }}, \*p = a;

要求值的表达式：1：(++p)->x，分析如下，a是一个结构体数组，数目为2,p为结构体指针，指向a中第一个结构体的首地址，++p的值为a中第二个结构体的地址，所以再使用->取出其中x的值就是取出了100,所以这个表达式值是100;

2：p++,p->c：p++表达式将p指向了第二个结构体，p->c取出了其中c的值即字符’B’;

3：\*p++->t,\*p->t：根据优先级，表达式等价于以下表示\*((p++)->t),\*(p->t);逗号运算符结果总是最后一个表达式的值，前一个表达式将p指向了第二个结构体，所以表达式的值就是后面的表达式的值就是第二个结构体取出它的字符串的首字符，值为’x’;

4：\*(++p)->t：根据优先级，表达式等价于\*((++p)->t)表达式的值是’x’;

5：\*++p->t：根据优先级表达式等价于\*(++(p->t))取出了第一个结构体的字符串的第二个字符，表达式的值是’V’;

6：++\*p->t;该表达式等价于++(\*(p->t))，指向第一个结构体的字符串的第一个字符在字母表中的下一个字母，由已知可知是’U’的下一个字母，即’V’;

程序运行结果见图7-1：

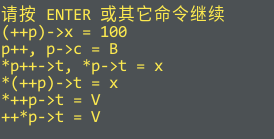


图7-1

与我的推断结果一致，程序运行正确。

### 7.1.2源程序修改、替换

题目细节见课本70页，程序的功能是给定一批整数，以0作为结束标志且不作为结点，建成一个先进先出的链表，头指针始终指向最先创建的结点，后建结点始终是尾节点。据我分析原来代码的错误主要是没有考虑到传指针做参数时只有进行解引用才能改变原来指向的元素的值，而直接修改传入的指针是不会达到预期效果的，我的做法是改用二级指针，这样就可以改变原来的值了。也可以选择更改返回值的类型，返回创建的链表的头部，让主函数中的指针指向它，也是可以的。

#include<stdio.h>

#include<stdlib.h>

struct s\_list{

int data;

struct s\_list\* next;

};

// 修改可以使用二级指针或者改变返回值类型，这里选择使用二级指针。

void create\_list(struct s\_list\*\*head\_p, int\*p);

int main(void){

struct s\_list\*head = NULL, \*p;

int s[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };

create\_list(&head, s);

p = head;

while(p){

printf("%d\t", p->data);

p = p->next;

}

printf("\n");

return 0;

}

void create\_list(struct s\_list\*\*headp, int\*p){

struct s\_list\*loc\_head = NULL, \*tail;

if (p[0] == 0){

;

}

else{

loc\_head = (struct s\_list\*)malloc(sizeof(struct s\_list));

loc\_head->data = \*p++;

tail = loc\_head;

while(\*p){

tail->next = (struct s\_list\*)malloc(sizeof(struct s\_list));

tail = tail->next;

tail->data = \*p++;

}

tail->next = NULL;

}

\*headp = loc\_head;

}

// 出现的错误，没有输出数据，原因：函数传参是传值，这里指指针的值，

// 指针的值改变是局部变量的改变，而不能影响main函数中的值

运行结果见图7-2：

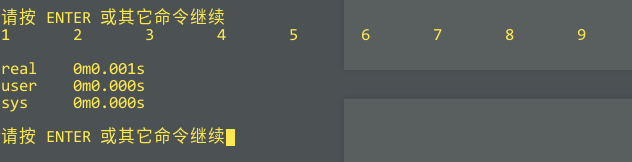


图7-2

题目还要求修改create\_list函数，使之成为建立后进先出的链表，后建结点指向先建结点，先建结点始终是尾结点。主要做法就是合理的处理先建结点和后建结点的关系，需要引入几个变量来控制结点的指针指向。细节见修改后的函数代码：

void create\_list(struct s\_list\*\*headp, int\*p){

struct s\_list\*head = NULL, \*tail; // 创建暂时的头

head = (struct s\_list\*)malloc(sizeof(struct s\_list));

head->next = NULL; //后继无人， 赋值为空指针

head->data = \*p++;

tail = head; // 尾指针存储head

while(\*p){

// 现在就是在头上开新空间

head = (struct s\_list\*)malloc(sizeof(struct s\_list));

head->data = \*p++;

head->next = tail; // 指向原来的头

tail = head; // 保存头

}

\*headp = head;

}

运行结果见图7-3：

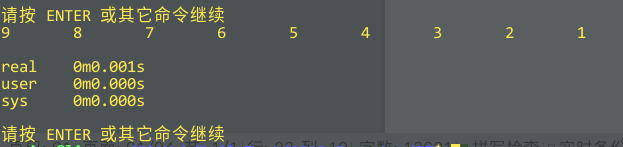


图7-3

运行结果正确。

### 7.1.3程序设计

第一题要求实现结构体的字段结构，每个成员占用1个bit，并且练习使用函数指针。

#include<stdio.h>

#include<stdlib.h>

typedef unsigned int u\_int;

typedef struct bits{

u\_int bit0 : 1;u\_int bit1 : 1;

u\_int bit2 : 1;u\_int bit3 : 1;

u\_int bit4 : 1;u\_int bit5 : 1;

u\_int bit6 : 1;u\_int bit7 : 1;

}bits;

// 为结构体的成员赋值

// 从最低位开始取出目标数字的每一位

void SetBit(bits\* res, unsigned short tar){

res->bit0 = tar % 2;

tar /= 2;

res->bit1 = tar % 2;

tar /= 2;

res->bit2 = tar % 2;

tar /= 2;

res->bit3 = tar % 2;

tar /= 2;

res->bit4 = tar % 2;

tar /= 2;

res->bit5 = tar % 2;

tar /= 2;

res->bit6 = tar % 2;

tar /= 2;

res->bit7 = tar % 2;

}

// 调用函数，如果不是0就调用

void Print(const bits\* tar, void (\*p\_fun[8])(void)){

if (tar->bit0)p\_fun[0]();

if (tar->bit1)p\_fun[1]();

if (tar->bit2)p\_fun[2]();

if (tar->bit3)p\_fun[3]();

if (tar->bit4)p\_fun[4]();

if (tar->bit5)p\_fun[5]();

if (tar->bit6)p\_fun[6]();

if (tar->bit7)p\_fun[7]();

}

// 要求的函数

void f0(void){

printf("The function 0 is called!\n");

}

void f1(void){

printf("The function 1 is called!\n");

}

void f2(void){

printf("The function 2 is called!\n");

}

void f3(void){

printf("The function 3 is called!\n");

}

void f4(void){

printf("The function 4 is called!\n");

}

void f5(void){

printf("The function 5 is called!\n");

}

void f6(void){

printf("The function 6 is called!\n");

}

void f7(void){

printf("The function 7 is called!\n");

}

int main(void){

bits\* temp = (bits\*)malloc(sizeof(bits));

// 函数指针数组，直接初始化即可

void (\*p\_fun[8])(void) = { f0, f1, f2, f3, f4, f5, f6, f7 };

unsigned short t;

scanf("%hu", &t);

SetBit(temp, t);

Print(temp, p\_fun);

printf("\n");

// 释放内存

free(temp);

return 0;

}

测试数据见表7-1：

|  |  |
| --- | --- |
| 输入 | 应该调用 |
| 0 | 无 |
| 1 | 1 |
| -1 | 全部 |
| 204 | 2,3,6,7 |

结果见图7-4,7-5,7-6,7-7：



图7-4

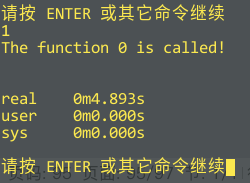


图7-5

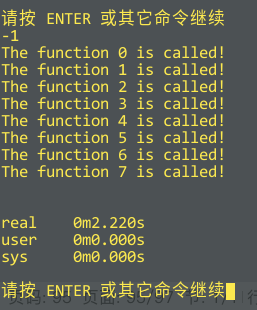


图7-6

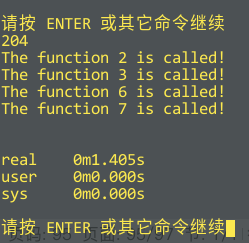


图7-7

第二题要求建立一个班级成绩单，包括每个学生的学号、姓名、英语、高等数学、普通物理等科目成绩，要求能够修改指定的数据，统计学生平均成绩。难点在于修改数据，只要选取学号作为主键进行搜索就可以了，再修改为指定数据即可。代码过长，见附件03.2ScoreHandle.c，测试数据见表7-2。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 | 姓名 | 英语 | 数学 | 物理 | C语言 |
| U2000 | Mary | 95 | 90 | 89 | 93 |
| U2001 | Tom | 87 | 98 | 88 | 70 |
| U2002 | Jack | 79 | 98 | 86 | 100 |

修改U2001的C语言成绩为90。结果见图7-8,7-9

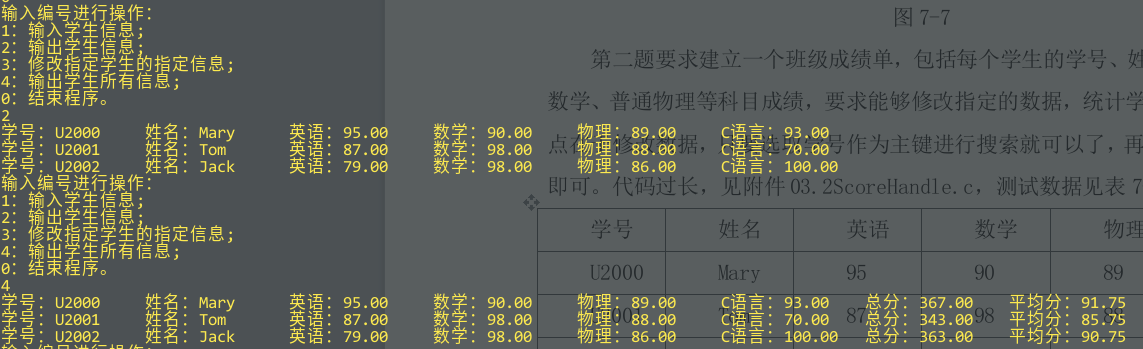


图7-8

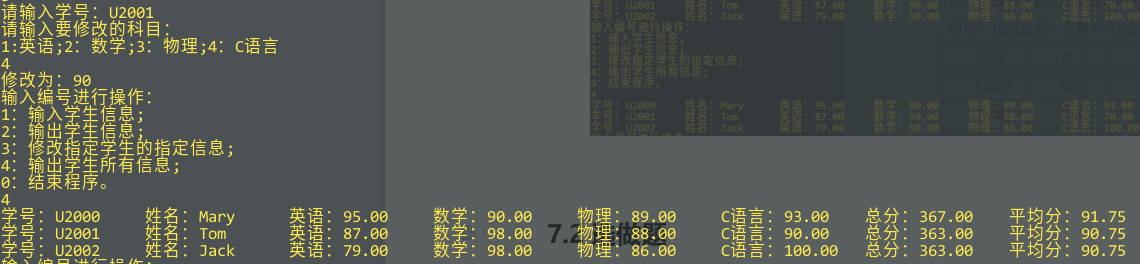


图7-9

结果符合预期，成绩被正确修改，平均成绩也改变了。

## 7.2选做题

选做题使用交换数据域和指针域两种方法对成绩进行排序，交换数据域需要注意不要忘记交换指针域，否则链表会变短的，交换结点域需要注意需要交换4个结点的内容，需要交换的两个结点的指针域，需要交换的结点前面的结点的指针域。

// 交换数据域

void SwapData(pStu s1, pStu s2) {

Stu t = \*s1;

\*s1 = \*s2;

\*s2 = t;

pStu tp = s1->next;

s1->next = s2->next;

s2->next = tp;

}

// 使用选择排序

void Sort\_data(pStu\*head) {

pStu s1 = \*head;

pStu s2 = NULL;

if (s1 == NULL){

return;

}

pStu min = NULL;

while (s1 != NULL){

min = s1;

s2 = s1->next;

while (s2!= NULL) {

if (s2->ave < min->ave){

min = s2;

}

s2 = s2->next;

}

if (s1 != min){

SwapData(s1, min);

}

s1 = s1->next;

};

}

// 交换指针域

void SwapPoint(pStu s1, pStu s2, pStu l1, pStu l2){

pStu tp = s1->next;

s1->next = s2->next;

s2->next = tp;

tp = l1->next;

l1->next = l2->next;

l2->next = tp;

}

// 使用选择排序

void Sort\_point(pStu\*head){

// 空结点返回

if (\*head == NULL) {

return;

}

// 标记修改的是第一个结点的位置

int first = 1;

// 存储要交换的结点

pStu s1 = \*head;

pStu s2 = NULL;

// pre1存放s1前结点

// pre2存放s2前结点

pStu pre1 = (pStu)malloc(sizeof(Stu));

pre1->next = \*head;

pStu pre2 = NULL;

// 存放min前结点

pStu min\_pre;

// 最小元素的结点

pStu min = NULL;

while (s1) {

min = pre2 = s1;

s2 = pre2->next;

while (s2) {

if (s2->ave < min->ave) {

min = s2;

min\_pre = pre2;

}

s2 = s2->next;

pre2 = pre2->next;

}

if (s1 != min){

// 交换对应结点的指针

SwapPoint(pre1, min\_pre, s1, min);

}

if (first){

// 修改的是第一个结点需要修改head指针的内容

\*head = pre1->next;

first = 0;

}

// 向后移动

s1 = s1->next;

pre1 = pre1->next;

}

free(pre1);

}

结果正确。

## 7.3小结

结构体属于程序设计之中比较常用的数据结构之一，从结构体可以衍生出链表、堆、栈、树、图等复杂的数据结构，因此对结构掌握熟练有助于以后的数据结构的学习，对理解程序的设计有较大帮助。在linux系统中就有大量的结构体的使用，时间的表示、文件读写等，都有对应的结构体，可以降低设计系统的难度，便于构思。在函数参数过多并且使用此函数很频繁时可以考虑设计一个结构体来简化传参过程，在函数多返回值时也可以考虑结构体的使用，有一点需要记住就是结构体是值类型，不同于数组字符串是引用类型，所以它可以整体赋值，对于有些需要交换内容的地方会比较方便。

# 8文件实验

## 8.1必做题

### 8.1.1文件类型的程序验证

题目具体细节见课本72页，代码主要功能是测试文件类型对存储数据的影响，程序新建2个文件，一个是文本文件，一个是二进制文件，对它们写入同样的内容即十六进制0x253f和0x7b7d，再打印出来看有何异同。运行结果如图8-1所示：



图8-1

从结果上看第一行对应的是二进制文件的输入方式，输出的似乎是乱码一团，可是在查阅ASCII表后发现第一个符号’?’对应ASCII表上的0x3f，’%’对应0x25,’}’对应0x7d，’{’对应0x7b，这里需要说明我的计算机CPU是intel的，所以在存储数字的时候默认是使用小端模式即数字高字节存储在低位，由此可见文件中存储的就是两个short类型的数字0x253f和0x7b7d，每个数字是2个字节，使用char型将字节按顺序一个一个输出就是上面的那4个符号了。对于文本类型存储的是2个十六进制数字253f和7b7d，说明在文本模式写入的时候是把数字拆分成一个一个字符写入文件的，和二进制文件不同。

第二问将sizeof(short)改为sizeof(char)，查看运行结果的不同，修改后的结果见图8-2,结果是二进制文件的输出变为了”?}”，说明写入的是高字节，低字节被舍弃，我的计算机在写入的时候是从高字节向低字节顺序写入文件的。



图8-2

第三问要求将fprintf(fp2, “%hx &hx”, a, b)修改为fprintf(fp2, “%d %d”, a, b)，查看运行的结果有何不同，结果如图8-3，文本的数字变成了十进制，和原来的数字数值相同，可见只是表示的进制改变了。



图8-3

### 8.1.2源程序修改替换

题目细节见课本73页，使用命令行输出指定的文本的内容，第一步是改错，修改如下：

#include<stdio.h>

#include<stdlib.h>

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

char ch;

FILE \*fp;

if (argc != 2){

printf("ARguments error!\n");

exit(-1);

}

if ((fp = fopen(argv[1], "r")) == NULL){ /\* fp指向filename \*/

printf("Can't open %s file!\n", argv[1]);

exit(-1);

}

// 根据优先级，ch得不到字符，修改加括号

while ((ch = fgetc(fp)) != EOF){ /\* 从fileame中读字符 \*/

putchar(ch); /\* 向显示器写字符 \*/

}

fclose(fp); /\* 关闭filename \*/

return 0;

}

具体错误就是在输出的时候由于优先级的问题ch得到的是比较的结果而不是字符，输出的只能是ASCII表中0和1对应的字符，只要在其中加上括号提高优先级就可以了。

测试如下：



图8-4

### 8.1.3程序设计

编写程序实现把输入的英语句子每个单词首字母大写，将其输入到一个磁盘文件中，代码如下：

#include<stdio.h>

#include<stdlib.h>

// 检测字符是否是英文字母

int isCharacter(char ch){

return (ch >= 'a'&& ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

// 判断是否是小写字母

int isLower(char ch){

return ch >= 'a' && ch <= 'z';

}

// 函数将句子单词首字母大些

char\* Upper(char\* str){

char \*temp = str;

// 单独判断第一个字符

if (isLower(\*temp)){

\*temp = \*temp + 'A' - 'a';

}

temp++;

// 开始处理整个句子

while (\*temp != '\0'){

// 如果这个字符是字母并且上一个字符不是字母就进行转换

if (isCharacter(\*temp) && !isCharacter(\*(temp - 1))){

if (isLower(\*temp)){

\*temp = \*temp + 'A' - 'a';

}

}

++temp;

}

return str;

}

int main(void){

char str[1000];

printf("Please enter a sentence:");

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

// 去除回车字符

getchar();

FILE\* fp;

// 输入到文件

fp = fopen("./test", "w+");

fprintf(fp, "%s\n", Upper(str));

// 关闭文件

fclose(fp);

return 0;

}

运行结果见图8-5：

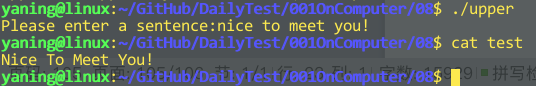


图8-6

运行结果正确。

## 8.2自设题

## 8.3小结

C语言是很接近底层的语言，C语言处理普通文件的时候有2种存储方式，一种是二进制的方式，一种是文本方式，二进制读写速度快但是不便于人处理，文本方式便于人读懂但是速度较慢，各有利弊，应该根据使用环境选取适合的处理方式。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