

段江涛

计算机导论与程序设计 [CS006001038,X07]

机试练习参考程序代码

2021 年 11 月 28 日

目录

1	第 1 次机试练习: 熟悉 DEV-C++ 开发平台, 基本输入输出语句练习	5
1.1	计算球体重量	5
1.2	温度转化	6
1.3	整数简单运算	6
1.4	A+B+C	7
1.5	字符输入输出	8
1.6	数字字符	8
1.7	实数运算	9
2	第 2 次机试练习: 选择与循环语句练习	11
2.1	四则运算	12
2.2	数位输出	13
2.3	阶梯电价计费	17
2.4	计算某月天数	19
2.5	计算整数各位数字之和	20
2.6	完数	22
3	第 3 次机试练习: 继续分支与循环练习	27
3.1	最大公约数	27
3.2	角谷定理	32
3.3	整数分析	33
3.4	冰箱温度预测	34
3.5	除法计算器	35
3.6	自然数分解	36
3.7	选号程序	37
4	第 4 次机试练习: 继续分支与循环练习	41
4.1	完全平方数	41
4.2	成绩分级	42
4.3	abc 组合	43
4.4	工资计算	44
4.5	跳一跳	45

4.6	车牌限行	47
4.7	气温波动	50
5	第 5 次机试练习: 继续练习基本输入输出语句, 分支与循环, 简单数组应用	53
5.1	最小差值	53
5.2	PM2.5	54
5.3	折点计数	56
5.4	寻找最大整数	57
5.5	ISBN 号码	59
6	第 6 次机试练习: 函数, 流程控制, 字符串, 数组	63
6.1	歌德巴赫猜想	63
6.2	矩阵	67
6.3	回文数	70
6.4	马鞍点	74
6.5	密码强度	76
6.6	数字分解排序	77

Chapter 1

第 1 次机试练习: 熟悉 DEV-C++ 开发平台, 基本输入输出语句练习

1.1 计算球体重量

已知铁的比重是 7.86(克/立方厘米), 金的比重是 19.3(克/立方厘米)。写一个程序, 分别计算出给定直径的铁球与金球的质量, 假定 $PI=3.1415926$

输入说明:

输入两个整数, 分别表示铁球与金球的直径 (单位为毫米)

输出说明:

输出两个浮点数, 分别表示铁球与金球的质量 (单位为克), 小数点后保留 3 位小数, 两个浮点数之间用空格分隔

输入样例:

100 100

输出样例:

4115.486 10105.456

提示:

用scanf输入, 用printf输出, 保留 3 位小数的格式控制字符为%.3f

```
#include<stdio.h>
#include<math.h>    // 数学库函数
#define PI 3.1415926
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    float v1= 4.0/3.0*pow(a/2.0/10,3)*PI;
    float v2= 4.0/3.0*pow(b/2.0/10,3)*PI;
    printf("%.3f□%.3f\n",7.86*v1,19.3*v2);
    return 0;
}
```

Note 1.1 (要点).

1. 整数除以整数, 结果为整数。

4.0/3.0 结果是浮点数, 4/3 结果是整数

2. 化简公式会引起精度问题, 不要随意化简公式。

3. pow 函数原型: `double pow(double x, double y)`

当形参数是整数时, 由于精度问题, 不要使用此函数计算 x^y . 推荐使用循环语句, 易计算 x^y 。如果必要, 可自定义函数: `int mypow(int x, int y)`。见课件。

1.2 温度转化

已知华氏温度到摄氏温度的转换公式为: 摄氏温度 = (华氏温度 - 32) × 5/9, 写程序将给定的华氏温度转换为摄氏温度输出。

输入说明:

只有一个整数, 表示输入的华氏温度

输出说明:

输出一个表示摄氏温度的实数, 小数点后保留 2 位有效数字, 多余部分四舍五入

输入样例:

50

输出样例:

10.00

提示:

用 scanf 输入, 用 printf 输出, 保留 2 位小数的格式控制字符为

```
#include <stdio.h>

int main()
{
    int f;
    float c;
    scanf("%d",&f);
    c = (f-32)*5.0/9;    // (1)
    //c = (f-32)*5/9;    // (2)
    printf("%.2f\n",c);
    return 0;
}
```

Note 1.2 (思考). 为何语句 (1),(2) 计算结果不一致, 哪一条语句正确?

1.3 整数简单运算

编写程序, 计算用户输入的两个整数的和、差、乘积 (*) 和商 (/)。

输入格式：输入两个整数，整数之间用空格分隔。

输出格式：输出四个整数结果，分别表示和、差、积和商，每输出一个结果换行。

输入样例：

3 4

输出样例：

7

-1

12

0

```
#include<stdio.h>
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n%d\n%d\n%d\n",a+b,a-b,a*b,a/b);
    return 0;
}
```

Note 1.3 (思考). b=0 时如何处理?

1.4 A+B+C

通过键盘输入三个整数 a, b, c, 求 3 个整数之和。

输入说明：

三整形数据通过键盘输入，输入的数据介于-100000 和 100000 之间，整数之间以空格、跳格或换行分隔。

输出说明：

输出 3 个数的和。

输入样例：

-6 0 39

输出样例：

33

```
#include<stdio.h>
int main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    printf("%d\n",a+b+c);
    return 0;
}
```

1.5 字符输入输出

通过键盘输入 5 个大写字母, 输出其对应的小写字母, 并在末尾加上 “!”。

输入说明:

5 个大写字母通过键盘输入, 字母之间以竖线 “|” 分隔。

输出说明:

输出 5 个大写字母对应的小写字母, 之间无分隔, 并在末尾加上 “!”。

输入样例:

H|E|L|L|O

输出样例:

hello!

```
#include <stdio.h>
int main()
{
    char c1, c2, c3, c4, c5;
    scanf("%c| %c| %c| %c| %c", &c1, &c2, &c3, &c4, &c5);
    c1 += 32; c2 += 32; c3 += 32; c4 += 32; c5 += 32;
    printf("%c%c%c%c%c!", c1, c2, c3, c4, c5);
    return 0;
}
```

Note 1.4 (要点). scanf(“原样输入”,...);

Note 1.5. (大小写字符转化关系) 小写字符 ASCII 码 = 大写字符 ASCII 码 + 32

1.6 数字字符

通过键盘输入 1 个整数 $a(0 \leq a \leq 4)$, 1 个数字字符 $b('0' \leq b \leq '5')$ 求 $a+b$ 。

输入说明:

整形数据、数字字符通过键盘输入, 输入的整形数据介于 0 和 4 之间, 输入的数字字符介于 ‘0’ 和 ‘5’ 之间, 二个输入数之间用 “,” 分隔。

输出说明:

分别以整数形式及字符形式输出 $a+b$, 输出的二个数之间用 “,” 分隔。

输入样例:

3,5

输出样例:

56,8


```
#include<stdio.h>
int main()
{
    int a;
    char b;
    scanf("%d,%c",&a,&b);
    printf("%d,%c",a+b,a+b);
    return 0;
}
```

Note 1.6. (scanf 函数) scanf("原样输入",...);

Note 1.7. (整型数值与字符混合运算) 字符对应的 ASCII 编码参与整数运算, 其结果也是整数。注意 '0' 与 0 不同, 本例中输入 0,0, 则 a=0, b='0', 变量 a 的值是整数 0, 变量 b 的值是字符 '0' 对应的 ASCII 编码, 即整数 48。

如果本题改为计算整数 a+(字符 b 对应的数字), 则, printf("%d",a+b-'0');

1.7 实数运算

通过键盘输入长方体的长、宽、高, 求长方体的体积 V(单精度)。

输入说明:

十进制形式输入长、宽、高, 输入数据间用空格分隔。

输出说明:

单精度形式输出长方体体积 V, 保留小数点后 3 位, 左对齐。

输入样例:

15 8.12 6.66

输出样例:

811.188

```
#include<stdio.h>
int main()
{
    float a,b,c;
    scanf("%f%f%f",&a,&b,&c);
    printf("%.3f",a*b*c);
    return 0;
}
```

Note 1.8. (精度问题) 32 位编译器: a*b*c 与 a*c*b 结果一致。但是在 64 位编译器中, 二者不一致。

因此, 浮点数运算会存在精度问题, 不要随意改变运算顺序。

Chapter 2

第 2 次机试练习：选择与循环语句练习

— 特别提示 —

Note 2.1 (不该再次发生的常见错误, 输入输出格式转换符不对应, 导致的严重错误).

```
int a; float b; double c; char d;  
scanf("%d",a); // 遗忘变量前的取地址符&  
scanf("%d\n",&a); // 多余'\n', 导致不能正常输入  
scanf("%d%f%lf%c",&a,&b,&c,%d); // 正确对应关系  
scanf("%d%c%f",&a,&d,&b); // 正确对应关系  
printf("%d,%f,%lf,%c",a,b,c,d); // 正确对应关系
```

Note 2.2 (不该再次发生的常见错误, 由 ‘;’ 引发的悲剧).

```
if ();  
{  
    ...  
}  
  
while ();  
{  
    ...  
}  
  
for (;;);  
{  
    ...  
}
```

Note 2.3 (用 C 语言关系表达式准确表达数学含义).

```
int a;
if(110<=a<=210) // 错误
{ }
if(110<=a && a<=210) // 正确
{ }
if(a>=110 && a<=210) // 正确
{ }
```

Note 2.4 (学习体会编程技巧).

- 使用 printf() 语句, 追踪程序执行细节, 查找出错原因。
- 对于条件结构, 循环结构, 首先书写整体结构, 再添加细节, 避免低级错误。
- 提倡一题多解, 举一反三, 体会编程技巧。

2.1 四则运算

输入两个整数和一个四则运算符, 根据运算符计算并输出其运算结果 (和、差、积、商、余之一)。注意做整除及求余运算时, 除数不能为零。

输入说明:

使用 scanf() 函数输入两个整数和一个运算符, 格式见输入样例。

输出说明:

输出使用 printf() 函数, 格式见输出样例。

输入样例:

5%2

输出样例:

5%2=1

```
#include <stdio.h>
int main()
{
    int a,b;
    char op;
    scanf("%d%c%d",&a,&op,&b);
    switch(op)
    {
        case '+': printf("%d%c%d=%d\n",a,op,b,a+b); break;
        case '-': printf("%d%c%d=%d\n",a,op,b,a-b); break;
        case '*': printf("%d%c%d=%d\n",a,op,b,a*b); break;
        // 注意分母为0时, 不会正确运算/,%
        case '/': if (b!=0) printf("%d%c%d=%d\n",a,op,b,a/b); break;
```

```
        case '%': if (b!=0) printf("%d%c%d=%d\n",a,op,b,a%b); break;
    }
    return 0;
}
```

Note 2.5 (printf 双引号中的 % 输出, %% 表示输出 %).

```
int a,b;
char op;
printf("%d%%d=%d\n",a,b,a%b);
// 或当 op='%'时
printf("%d%c%d=%d\n",a,op,b,a%b);
```

2.2 数位输出

输入一个 5 位整数, 求出其各数位数值, 并按照从高位到低位的顺序输出, 如: 输入 12345, 输出为 1 2 3 4 5。

输入说明:

输入一个五位正整数。

输出说明:

按数位从高到低依次输出, 各数位之间以一个空格相分隔。

输入样例:

96237

输出样例:

9 6 2 3 7

```

#include<stdio.h>

/*****
5位整数已知. 首先用10000除以整数a(分子), 得到分子最高位。
改变分子分母, 循环迭代, 依次获得分子的最高位。
*****/

int main1()
{
    int a,b=10000,i=5; // i记录整数a的初始位数
    scanf("%d",&a);
    while(i>=1)
    {
        if (i==1) printf("%d\n",a/b); // 输出当前a的最高位
        else printf("%d□",a/b);
        a = a-a/b*b; // 去除当前a的最高位, 准备下轮迭代的分子a
        b/=10; // b=b/10, 准备下轮迭代的分母b
        i--;
    }
    return 0;
}

/*****
假设不知整数a的位数。
除10取余, 迭代循环, 可方便获取整数a的个位, 十位, 百位, 千位, ...
利用数组存储个位, 十位, 百位, 千位, ... 最后反序输出即是所求。
*****/

int main2()
{
    int a, tmp[100]; // tmp数组存储100(估计的最大值)个整数, 用tmp[0],tmp
[1],tmp[2],...读写各个整数。
    int i=0, j; // i: 记录整数a的位数
    scanf("%d",&a);
    if(a==0) // 考虑整数0的特殊情况, 直接输出即可。
    {
        printf("%d\n",a);
    }
    else // 因为循环语句判断a是否为0, 因此要有上述判断才能考虑到所有可能情
况的发生
    {
        while(a!=0) // 迭代逆序求出整数a的各位数字
        {
            tmp[i]=a%10; // 存储本轮循环a的末位数

```

```

        //printf("调式查看tmp[i]=%d\n",tmp[i]); // 提交时，别忘了注释或删除调试语句
        a=a/10;        // 改变分子，准备下轮循环
        i++;           // 位数递增
    }
    //printf("调式查看i=%d\n",i);
    // 逆序输出tmp，此时的i是整数a的位数，注意tmp的下标从i-1开始到下标0结束。
    for(j=i-1;j>=0;j--)
    {
        printf("%d□",tmp[j]);
    }
}
return 0;
}

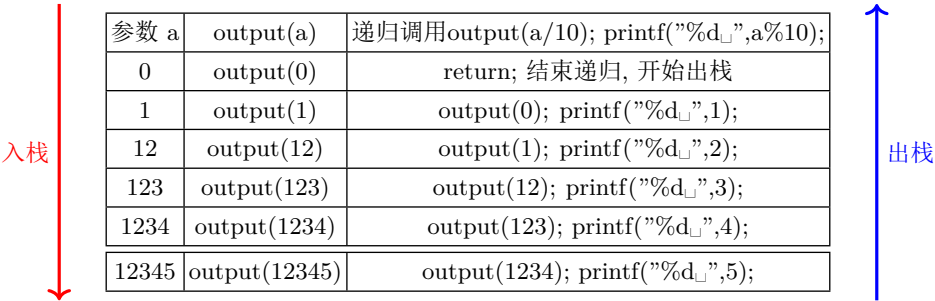
/*****
假设不知整数a的位数。
利用递归函数求解，a==0的情况在函数外处理输出较方便。
因此该函数仅考虑a!=0的情况。
*****/
void output(int a)
{
    if(a!=0) // 如果考虑a==0的情况，不好判断是初始a=0还是迭代后a=0的情况。
    这里考虑后者。前者的处理留给调用它的程序。
    {
        // '栈'是一种'先进后出'的数据结构
        output(a/10); // 递归调用，函数参数会自动存储在系统维护的'栈'中。
        printf("%d□",a%10); // 从内部存储'栈'中，依次弹出各位数，输出之。
    }
    else // a==0 ,可省略else语句，隐含结束递归调用
    {
        return; // 函数结束，注意本函数无返回值，因此return后无表达式。
    }
}

int main()
{
    int a;
    scanf("%d",&a);
    if(a==0) // 考虑整数0的特殊情况，直接输出即可。

```

```
{
    printf("%d\n",a);
}
else
{
    output(a); // 函数调用，完成逆序输出。
}
return 0;
}
```

图 2.1: 递归函数void output(int a)中系统内部维护的‘栈’结构示意图



Note 2.6 (知识点).

- 1. 体会除 10 取余, 迭代循环的整数分解技巧;
- 2. 第一种解法的 b=1000 初值是可计算的, 这样就可扩充此解法为任意位的整数 a。

```
// 因为a要在main1()函数的while循环中使用。
// 因此，定义临时变量，存储a的值，用于计算b的初值。
int tmp;
b=1; tmp=a;
while (tmp!=0)
{
    b=b*10;
    tmp=tmp/10;
}
```

- 3. 预习数组使用技巧;
- 4. 预习函数定义及调用;
- 5. 预习递归函数的定义, 体会系统维护的内部存储‘栈’的数据存储特点。

2.3 阶梯电价计费

电价分三个档次，[0,110] 度电，每度电 0.5 元；(110,210] 度电，超出 110 部分每度电 0.55 元，超过 210 度电，超出 210 部分每度电 0.70 元，给出一个家庭一月用电量，请计算出应缴的电费 (四舍五入，保留小数点后两位小数)。

输入说明：

输入数据为一个正实数，表示一月用电量

输出说明：

输出应缴电费，四舍五入保留 2 位小数。

输入样例：

输入样例 1

100

输入样例 2

200

输入样例 3

329.75

输出样例：

输出样例 1

50.00

输出样例 2

104.50

输出样例 3

193.83

```
#include <stdio.h>
int main()
{
    float sum,u1=0.5,u2=0.55,u3=0.70; // 用电量,每度电单价
    float fee = 0; // 应缴电费

    scanf("%f",&sum);

    if (sum > 210)
    {
        fee = (sum-210)*u3;
        sum = 210;
    }
    if (sum > 110)
    {
        fee += (sum-110)*u2; // fee=fee+(sum-110)*u2;
        sum = 110;
    }
}
```

```
    fee += sum*u1;

    printf("%.2f\n", fee);
    return 0;
}

int main2() // 另解
{
    float sum, u1=0.5, u2=0.55, u3=0.70; // 用电量, 每度电单价
    float fee = 0; // 应缴电费
    scanf("%f", &sum);

    if (sum >= 210)
        fee = 110*u1 + (210-110)*u2 + (sum-210)*u3;
    else if (sum >= 110)
        fee = 110*u1 + (sum-110)*u2;
    else
        fee = sum*u1;

    printf("%.2f\n", fee);
    return 0;
}

int main3() // 另解
{
    float sum, u1=0.5, u2=0.55, u3=0.70; // 用电量, 每度电单价
    float fee = 0; // 应缴电费

    scanf("%f", &sum);

    if (sum <= 110) fee = sum*u1;
    else if (sum <= 210)
    {
        fee = 110*u1;
        sum -= 110; // sum=sum-110;
        fee += sum*u2;
    }
    else // sum > 210
    {
        fee = 110*u1;
        fee += (210-110)*u2; // fee = fee+(210-110)*u2
    }
}
```

```
        sum -= 210;    // sum=sum-210;
        fee += sum*u3; // fee = fee+ sum*u3;
    }

    printf("%.2f\n", fee);
    return 0;
}
```

Note 2.7 (四舍五入问题). 不同的编译系统, 处理结果可能不一致, `printf("%.2f\n", fee);` 默认输出即可。

Note 2.8. 练习 if 语句的不同组合形式, 杜绝出现 `if(110<=sum<=210)` 的错误形式。

2.4 计算某月天数

每年的 1, 3, 5, 7, 8, 10, 12 月有 31 天, 4, 6, 9, 11 月有 30 天, 闰年 2 月 29 天, 其他年份 2 月 28 天, 给定年份和月份求该月的天数

输入说明:

输入由两个正整数 a 和 b 构成, a 表示年份, b 表示月份, a 和 b 之间用空格分隔

输出说明:

根据年份和月份计算该月天数并输出

输入样例

输入样例 1

2000 3

输入样例 2

2001 2

输出样例

输出样例 1

31

输出样例 2

28

```
#include <stdio.h>
int main()
{
    int a, b, t = 0;
    scanf("%d%d", &a, &b);
    if((a%4 == 0 && a%100 != 0) || (a%100 == 0 && a%400 == 0))
    {
        if(b == 2) t = 29;
    }
    else if (b == 2) t = 28;
```

```
    if(b == 1 || b == 3 || b == 5 || b == 7 || b == 8 || b == 10 || b ==  
12) t = 31;  
    else if(b == 4 || b == 6 || b == 9 || b == 11) t = 30;  
  
    printf("%d\n",t);  
    return 0;  
}
```

Note 2.9. (逻辑运算符) &&, ||, !, 练习符合逻辑的各种组合形式。

Note 2.10. (闰年判断) ppt 中已详细说明, 还有同学写错。

2.5 计算整数各位数字之和

假设 n 是一个由最多 9 位数字 (d_9, \dots, d_1) 组成的正整数。编写一个程序计算 n 的每一位数字之和。

输入说明:

输入数据为一个正整数 n

输出说明:

对整数 n 输出它的各位数字之和后换行

输入样例:

3704

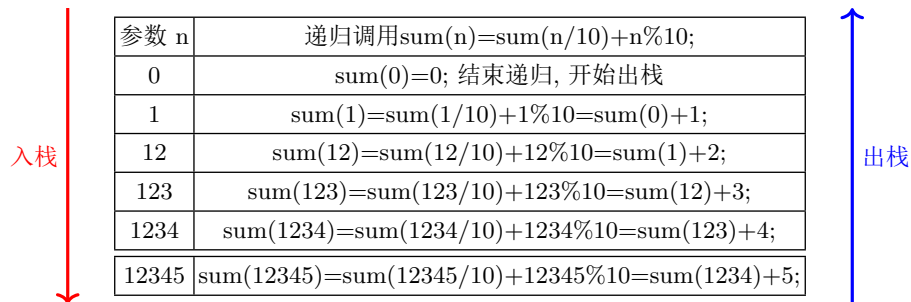
输出样例:

14

```
#include <stdio.h>
// 体会除10取余，迭代循环的整数分解技巧；
int main1()
{
    int n,sum = 0; // 注意初始化sum
    scanf("%d",&n);
    while(n) // 等效于n!=0
    {
        sum += n%10; // 累加本轮循环的末位数
        n /= 10;     // 准备下轮循环的分子
    }
    printf("%d",sum);
    return 0;
}

// 另解：定义递归函数，返回整数n的各位数之和
int sum(int n)
{
    if(n!=0)
    {
        // 递归调用，累加本轮循环的末位数
        return (sum(n/10)+n%10);
    }
    else // n==0时，结束递归调用
    {
        return 0; // 函数结束，返回整数0
    }
}

int main()
{
    int n;
    scanf("%d",&n);
    printf("%d\n",sum(n)); // 函数调用。
    return 0;
}
```

图 2.2: 递归函数 `int sum(int n)` 中系统内部维护的‘栈’结构示意图

Note 2.11 (知识点).

1. 体会除 10 取余, 迭代循环的整数分解技巧;
2. 预习递归函数定义及调用。

2.6 完数

请写一个程序, 给出指定整数范围 $[a, b]$ 内的所有完数, $0 < a < b < 10000$ 。一个数如果恰好等于除它本身外的所有因子之和, 这个数就称为“完数”。例如 6 是完数, 因为 $6 = 1 + 2 + 3$

输入说明

输入为两个整数 a 和 b , a 和 b 之间用空格分隔

输出说明

输出 $[a, b]$ 内的所有完数, 每个数字占一行

输入样例

1 10

输出样例

6

```
#include <stdio.h>

/*****
采用两层循环方案
(1) 外层循环使整数i递增, 完成区间[n1,n2]区间的完数计算
(2) 内层循环, 累加整数i的各因子
(3) 判断整数i是否是完数, 如果是, 输出之
*****/

int main1()
{
    int i, j, n1, n2, sum = 0;
    scanf("%d%d", &n1, &n2);
    for (i = n1; i <= n2; i++) // 外层循环使整数i递增, 完成区间[n1,n2]区间的完数计算
    {
```

```

        if(i == 1) continue; // 避免输出1, 1不是完数
        // i不等于1, 计算各因子
        sum = 1; // 不要忘记, 内层循环前sum的初始化。1总是一个整数的合法因子
        for(j = 2; j < i; j++) // 累加整数i的所有因子
        {
            if(i%j == 0) sum += j; // 如果j是i的因子, 累加之。
        }
        if(sum == i) printf("%d\n", i); // 如果i是完数, 输出之。
    }

    return 0;
}

/*****
采用一重循环 + 调用函数方案
(1) 一重循环使整数i递增, 函数compute调用, 完成区间[n1,n2]区间的完数计算
(2) 定义函数compute, 判断整数参数是否是完数, 如果是, 返回它, 否则返回-1
*****/

// 定义函数compute, 判断整数参数a是否是完数, 如果是, 返回a, 否则返回-1
int compute(int a)
{
    int i, s=1; // s用于存储a的各因子累加值, 1总是一个整数的合法因子
    if(a == 1)
    {
        return -1; // 1不是完数
    }
    // a不为1, 计算各因子
    for(i = 2; i < a; i++) // 累加整数a的所有因子
    {
        if(a%i == 0) s += i; // 如果i是a的因子, 累加之。
    }
    if(s == a)
    {
        return a; // 如果a是完数, 返回之。
    }
    // 如果程序执行到此处必然不是完数
    return -1;
}

```

```
// 另一种方式定义函数compute, 判断整数参数a是否是完数, 如果是, 返回a, 否则返回-1
// 一条return函数返回语句
int compute1(int a)
{
    int i, s=1; // s用于存储a的各因子累加值, 1总是一个整数的合法因子
    int ret=-1; // 用于返回值, 默认为-1

    for(i = 2; i < a; i++) // 累加整数a的所有因子
    {
        if(a%i == 0) s += i; // 如果i是a的因子, 累加之。
    }
    if(s == a && a!=1) // 如果a是完数, 返回值是本身。1不是完数
    {
        ret = a;
    }
    else // a不是完数
    {
        ret = -1;
    }
    return ret;
}

int main()
{
    int i, n1, n2;
    scanf("%d%d", &n1, &n2);
    for(i = n1; i <= n2; i++) // 调用函数compute, 完成区间[n1, n2]区间的完数计算
    {

        if(compute(i) != -1) printf("%d\n", i); // 如果i是完数, 输出之。
        // 测试函数compute1的调用
        // if(compute1(i) != -1) printf("%d\n", i); // 如果i是完数, 输出之。
    }

    return 0;
}
```

Note 2.12 (特别注意). 且记: 进入内层循环前, 相关变量的初始化问题。

Note 2.13 (函数定义和调用).

- 函数定义: 返回类型 函数名(参数列表) { 函数体 }
- `int fun1(float a, float b) { return a/b; // 返回整数部分 }`
- `void fun2(float a, float b) { printf(a/b); // 输出整数部分 }`
- 函数调用

```
float m,n;  
int ret;  
ret = fun1(m,n); // 调用函数fun1, 其返回值赋值给变量ret;  
fun2(m,n); // 调用函数fun2, 无返回值可用;
```


Chapter 3

第 3 次机试练习：继续分支与循环练习

— 特别提示 —

- 定义变量名, 要有含义, 如: sum,select。否则, 程序不易读。
- 用尽可能少的变量, 完成题设, 好排错。
- 用尽可能简单的循环迭代结构, 好排错。不要将简单问题复杂化, 不好排错。
- 注意进入内层循环前的相关变量初始化问题。
- 变量的默认值必须是实际存在的值。例如, 选号程序中, 假设的 qq 号就是读取的第一个 qq 号。
- 提倡一题多解, 认真消化, 体会参考代码, 积累编程技巧。

3.1 最大公约数

最大公约数 (GCD) 指某几个整数共有因子中最大的一个, 最大公约数具有如下性质,

$\gcd(a,0)=a$

$\gcd(a,1)=1$

因此当两个数中有一个为 0 时, gcd 是不为 0 的那个整数, 当两个整数互质时最大公约数为 1。

输入两个整数 a 和 b, 求最大公约数

输入说明:

输入为两个正整数 a 和 b ($0 \leq a, b < 10000$), a 和 b 之间用空格分隔,

输出说明:

输出其最大公约数

输入样例:

样例 1 输入

2 4

样例 2 输入:

12 6

样例 3 输入:

3 5

输出样例:

样例 1 输出

2

样例 2 输出

6

样例 3 输出

1

```
#include <stdio.h>
// 递归函数
int gcd(int a, int b)
{
    if (b == 0) return a;    // 公约数就是a
    return gcd(b, a % b);    // 递归调用
}

int main() // 调用递归函数
{
    int a, b, t;
    scanf("%d%d", &a, &b);
    if (a < b) { t = a; a = b; b = t; } // 交换a, b
    printf("%d\n", gcd(a, b));    // 函数调用
    return 0;
}

int main1() // 暴力循环求解，效率低。
{
    int a, b, t = -1, i; // t给初值是好习惯，否则下面程序逻辑有可能使t得到随机值。
    scanf("%d%d", &a, &b); // 机试系统不要想当然给提示语句，除非题目要求
    if (a < b) { t = a; a = b; b = t; } // 交换a, b, 使a是较大者
    if (b == 0)
    {
        t = a; // 考虑分母为0的情况，比如：5, 0的最大公约数为5
    }
    else
    {
        for (i = b; i > 0; i--)
        {
            if (a % i == 0 && b % i == 0)
            {
                t = i; break; // 求得最大公约数，a, b互质，必然t=1
            }
        }
    }
    printf("%d\n", t);
    return 0;
}

int main2() // 利用欧几里得定理循环求解，效率高。
```

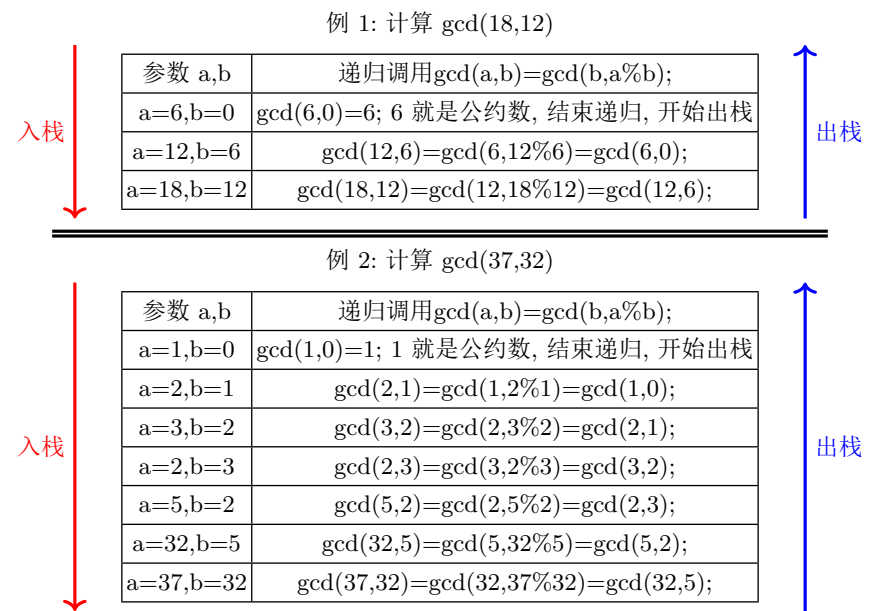
```
{
    int a,b,r,t;
    scanf("%d%d",&a,&b); // 机试系统不要想当然给提示语句, 除非题目要求
    if(a<b) { t=a; a=b; b=t; } // 交换a,b,使a是较大者
    while(1)
    {
        if(b==0) { t=a; break; } // 分母为0时, a就是最大公约数
        r = a%b;
        if(r==0) {t=b; break;} // b就是最大公约数
        a=b; b=r; // 准备下一轮迭代
    }
    printf("%d\n",t); // 输出最大公约数
    return 0;
}

int main3() // 利用欧几里得定理循环求解, 效率高。
{
    int a,b,r,t;
    scanf("%d%d",&a,&b); // 机试系统不要想当然给提示语句, 除非题目要求
    if(a<b) { t=a; a=b; b=t; } // 交换a,b,使a是较大者
    if (b==0) // 考虑分母为0的情况, 比如: 5,0的最大公约数为5
    {
        printf("%d\n",a);
    }
    else
    {
        // 排除了分母为0时不能求余数的情况
        while((r=a%b)!=0) // a/b的余数赋值给r,r不等于0时执行循环体
        {
            a=b;
            b=r;
        }
        printf("%d\n",b);
    }
    return 0; // 主函数结束
}

int main4() // 体会函数结束语句return的使用
{
    int a,b,r,t;
    scanf("%d%d",&a,&b); // 机试系统不要想当然给提示语句, 除非题目要求
```

```
if(a<b) { t=a; a=b; b=t; } // 交换a,b,使a是较大者
if (b==0) // 考虑分母为0的情况，比如：5,0的最大公约数为5
{
    printf("%d\n",a);
    return 0; // 主函数结束
}
// 排除了分母为0时不能求余数的情况
while((r=a%b)!=0) // a/b的余数赋值给r,r不等于0时执行循环体
{
    a=b; b=r; // 准备下一轮迭代
}
printf("%d\n",b);
return 0; // 主函数结束
}
```

图 3.1: 递归函数int gcd(int a,int b)中系统内部维护的‘栈’结构示意图



Note 3.1 (欧几里得定理).

a(大),b(小)的最大公约数：因为： $a=mb+r$ ， $m=a/b$ ； $r=a\%b$ ， $\Rightarrow a,b$ 的公约数能整除 b 和 r 。
 $r=a\%b$, r 为 0，则 b 就是最大公约数。否则迭代循环， $a=b$ ， $b=r$ ，直到余数为零，则分母就是最大公约数。

Note 3.2. 预习题及递归函数的使用。

3.2 角谷定理

角谷定理定义如下: 对于一个大于 1 的整数 n , 如果 n 是偶数, 则 $n = n / 2$ 。如果 n 是奇数, 则 $n = 3 * n + 1$, 反复操作后, n 一定为 1。

例如输入 22 的变化过程: 22 -> 11 -> 34 -> 17 -> 52 -> 26 -> 13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1, 数据变化次数为 15。

输入一个大于 1 的整数, 求经过多少次变化可得到自然数 1。

输入说明

输入为一个整数 n , $1 < n < 100000$ 。

输出说明

输出变为 1 需要的次数

输入样例

样例 1 输入

22

样例 2 输入

33

输出样例

样例 1 输出

15

样例 2 输出

26

```
#include <stdio.h>
int main()
{
    int n, i=0; // 变量i用于计数的辅助变量
    scanf("%d",&n);
    // 因为题目输入假设n>1, 因此不必考虑n=1时的情况
    while(n!=1) // n不等于1时执行循环体中的语句
    {
        if(n%2==0) n=n/2;
        else n=3*n+1;
        i++;
    }
    printf("%d\n", i);
    return 0;
}

// 含程序调试语句, 不吝惜写一些printf语句, 观察程序的执行过程。
int main()
{
    int n=22, i=0; // 变量i用于计数的辅助变量
```



```

//scanf("%d",&n); // 调试时可以注释掉输入语句，改变变量n的值，观察执行过程

printf("%d->",n);
while(n!=1) // n不等于1时执行循环体中的语句
{
    if(n%2==0)
    {
        n=n/2;
    }
    else
    {
        n=3*n+1;
    }
    printf("%d->",n);
    i++;
}
printf("\n总共变化次数%d\n",i);
return 0;
}

```

Note 3.3. 试着用do{ }while(); for(;;) 改写此程序，执行相同功能。

3.3 整数分析

给出一个整数 n ($0 \leq n \leq 1000000000$)。求出该整数的位数，以及组成该整数的所有数字中的最大数字和最小数字。

输入说明

输入一个整数 n ($0 \leq n \leq 1000000000$)

输出说明

在一行上依次输出整数 n 的位数，以及组成该整数的所有数字中的最大数字和最小数字，各个数字之间用空格分隔。

输入样例

217

输出样例

3 7 1

```

#include <stdio.h>
// 循环除10取余是整数分解的基本技巧
int main()
{

```

```

int i = 0, n, bit, max, min;
scanf("%d",&n);
while(n) // 等效于 while(n!=0)
{
    bit = n%10; // 获取n的最低为
    // 切记: 初始化时, 假设的max和min必须是实际存在的数。
    if(i == 0) // 初始化: 原始n的最低位设为最大和最小数字
    {
        max = min = bit;
    }
    else
    {
        if(bit > max) max = bit;
        if(bit < min) min = bit;
    }
    n /= 10; // 去除最低位
    i++;
}
// (i == 0 ? 1 : i)是条件表达式, 表达式的值是:
// 如果i==0,则表达式的值为1否则表达式的值是i
printf("%d_%d_%d\n", (i == 0 ? 1 : i), max, min); // 考虑原始n==0的情况
return 0;
}

```

Note 3.4 (知识点).

1. 整数数位分解是基本编程练习之一。
2. 切记: 初始化时, 假设的 max 和 min 必须是实际存在的数。比如不能想当然假设 max=1000, min=0.
3. 注意审题: “输入一个整数 n , ($0 \leq n \leq 100000000$)”, 因此, 0 也是一个合法输入。

3.4 冰箱温度预测

编写一个程序, 用于预测冰箱断电后经过时间 t (以小时为单位) 后的温度 T 。已知计算公式如下所示

$$T = \frac{4t^2}{t+2} - 20$$

输入说明

输入两个整数 h 和 m 表示冰箱断电后经过的时间, h 表示小时, m 表示分钟

输出说明

输出冰箱断电后经过时间 t (以小时为单位) 后的温度 T , 保留两位小数

输入样例

2 0

输出样例

-16.00

```
#include <stdio.h>
int main()
{
    int h,m;
    float t,T;
    scanf("%d%d",&h,&m);
    t = h + m/60.0;    // 必须是60.0, why?
    T = 4*t*t/(t+2)-20; // 优先级保证了计算的正确性, why?
    printf("%.2f\n",T);

    return 0;
}
```

Note 3.5 (知识点). 整数/整数, 表达式的值是整数部分, 自动舍去小数部分。

3.5 除法计算器

小明的弟弟刚开始学习除法, 为了检查弟弟的计算结果是否正确, 小明决定设计一个简单计算器程序来验算。

输入说明

输入数据由四个整数 m , n , q , r 构成, m 为被除数, n 为除数, q 和 r 为小明的弟弟计算出的商和余数。整数之间用空格分隔, 所有整数取值范围在 $(-100000 \sim 100000)$, n 不为 0。

输出说明

如果验算结果正确, 输出 yes, 否则输出正确的商和余数

输入样例:

样例 1:

10 3 3 1

样例 2:

10 3 3 2

输出样例

样例 1 输出:

yes

样例 2 输出:

3 1

```
#include <stdio.h>
int main()
{
```

```

int m,n,q,r;
scanf("%d%d%d%d",&m,&n,&q,&r);
if(m==q*n+r && q==m/n && r==m%n) printf("yes\n");
else printf("%d□%d\n",m/n,m%n);
return 0;
}

```

Note 3.6. 改变题设条件, 修改此程序, 进行各种表达式计算练习, 分析优先级。如果 $n=0$ 时, 如何处理。

3.6 自然数分解

任何一个自然数 m 的立方均可写成 m 个连续奇数之和。例如:

$$1^3 = 1$$

$$2^3 = 3 + 5$$

$$3^3 = 7 + 9 + 11$$

$$4^3 = 13 + 15 + 17 + 19$$

编程实现: 输入一自然数 n , 求组成 n^3 的 n 个连续奇数。

输入说明

一个正整数 n , $0 < n < 30$ 。

输出说明

输出 n 个连续奇数, 数据之间用空格隔开, 并换行

输入样例

4

输出样例

13 15 17 19

```

#include <stdio.h>
// 从估计的第一个奇数开始, 循环迭代求解。
int main()
{
    int n,i,j,sum,first;
    scanf("%d",&n);

    // 第一个可能的奇数:
    if(n%2) first = n;    // n是奇数
    else first = n + 1;   // n是偶数

    while(1)
    {

```

```

sum = 0; // 每趟内层循环前，必须置0
// 从first开始，n个连续奇数，i：表示连续奇数，j：计数。
for(i = first ,j = 1; j <= n; i += 2,j++)
{
    sum += i; // 连续奇数累加
    if(sum == n*n*n)
    {
        // 输出
        for(i = first ,j = 1; j <= n; i += 2,j++)
        {
            if (j == n)printf("%d\n",i);
            else printf("%d_",i);
        }
        return 0; // 函数结束
    }
}
first += 2;
}
return 0;
}

```

Note 3.7 (要点). 再次强调进入内层循环前，相关变量的初始化；以及标志变量（如本例 first）的使用技巧。

3.7 选号程序

小明决定申请一个新的 QQ 号码，系统随机生成了若干个号码供他选择。小明的选号原则是：

1. 选择所有号码中各位数字之和最大的号码。
2. 如果有多个号码各位数字之和相同则选择数值最大的号码。

请你写一个程序帮助小明选择一个 QQ 号码。

输入说明

输入数据由两行构成，第一行为一个整数 n 表示有 n 个待选号码 ($0 < n < 100$)，第二行有 n 个正整数，表示各个待选的号码，每个号码长度不超过 9 位数。每个号码之间用空格分隔，且每个号码都不相同。

输出说明

输出根据小明的选号原则选出的号码。

输入样例

5

10000 11111 22222 333 1234

输出样例

22222

```

#include <stdio.h>
// 在循环语句中, 读取备选qq号, 计算各位之和, 依据筛选条件选取qq号
int main()
{
    // 关键变量含义说明:
    // select_qq, select_sum表示备选qq及其各位之和
    // qq, sum表示当前读取的qq及其各位和
    int i, n, select_qq, select_sum, qq, sum, tmp;
    scanf("%d", &n);
    for (i = 0; i < n; i++) // 注意条件表达式, 表明i的最大值是n-1, 因为i是0开始的,
        因此共执行n次循环
    {
        scanf("%d", &qq); // 读取当前备选qq号
        tmp = qq; // 保存到临时变量中, 因为下面的循环语句要更改。
        sum = 0; // 当前读取qq号的各位之和。 注意: 一定要初始化, 否则上一个
        备选号的sum值会带入本轮循环中。
        while (tmp) // 计算各位之和
        {
            sum += tmp % 10;
            tmp /= 10;
        }
        // 第1轮迭代(i==0), 当前读取的qq就是所选, 其它根据题设条件选号
        // 因为三个表达式为||运算, 从左到右依次计算各表达式的值, 如果为真,
        则不会计算后边表达式。
        // 因此, 当i==0时不会计算其它两个表达式的值, if条件为真。
        if (i == 0 || sum > select_sum || (sum == select_sum && qq > select_qq))
        {
            select_qq = qq;
            select_sum = sum; // i==0时, select_sum初值为第一个号码各位之和。
        }
    }
    printf("%d", select_qq);
    return 0;
}

// 解法2: 用二维数组存储所有qq号及其各位和
#define N 100 // 估计最大数组长度
int main1()
{
    // 二维数组No, 第一列表示qq号, 第二列表示该qq号的各位数字之和。
    int i, n, No[N][2], tmp, sum, max = 0, largest = 0, select;

```

```
scanf("%d",&n);  
// 筛选条件2  
for(i=0;i<n;i++)  
{  
    scanf("%d",&No[i][0]);  
    tmp=No[i][0];  
    sum=0; // 一定初始化  
    while(tmp)  
    {  
        sum+=tmp%10;  
        tmp/=10;  
    }  
    No[i][1]=sum;  
    if(sum>=max) max=sum;  
}  
// 筛选条件1  
for(i=0;i<n;i++)  
{  
    if(No[i][1]==max) // 备选号码  
    {  
        if(No[i][0]>=largest)  
        {  
            select=No[i][0];  
            largest=No[i][0];  
        }  
    }  
}  
printf("%d",select);  
return 0;  
}
```

Note 3.8 (要点).

1. || 和 && 运算从左到右执行, 取得结果, 则不执行后面的表达式。

取得结果的含义是:

if (条件 1|| 条件 2|| 条件 3) 运算中, 只要有一个条件表达式为真 (非 0), 即整个条件 () 结果即为真。

if (条件 1 && 条件 2 && 条件 3) 运算中, 只要有一个条件表达式为假 (0), 即整个条件 () 结果即为假。

2. 比较两种解法的优缺点。
3. 本例是循环迭代的范例, 应反复演练, 领会迭代程序的编程技巧。
4. 试着定义函数, 改写此程序。
5. 本题不必使用排序算法, 使程序复杂化。

Chapter 4

第 4 次机试练习：继续分支与循环练习

— 特别提示 —

- 本次练习题不必使用数组, 如果使用数组, 必须正确定义数组, 举例如下:

```
// 估计数组的最大长度
#define N 100
int a[N], n, i; // 按照最大长度定义数组, 用n表示数组的实际长度.
scanf("%d",&n); // 一般从键盘接收数组的实际长度
// 举例输入数组各个元素
for(i=0;i<n;i++) scanf("%d",&a[i]);
// 举例输出数组各个元素
for(i=0;i<n;i++) printf("%d\t",a[i]);
printf("\n");
```

- 定义数组的常见错误: `int n,a[n];`

4.1 完全平方数

若一个整数 n 能表示成某个整数 m 的平方的形式, 则称这个数为完全平方数。写一个程序判断输入的整数是不是完全平方数。

输入说明

输入数据为一个整数 n , $0 \leq n < 1000000$ 。

输出说明

如果 n 是完全平方数, 则输出构成这个完全平方数的整数 m , 否则输出 `no`。

输入样例

样例 1:

144

样例 2:

15

输出样例

样例 1 输出:

12

样例 2 输出:

no

```
#include <stdio.h>
#include <math.h> // 数学函数头文件
int main()
{
    int n,m;
    scanf("%d",&n);
    m=(int)sqrt(n); // sqrt(n)计算的结果为double类型，此语句表示把它转化为int类型，自动舍去小数部分(不会四舍五入)，并赋值给m。
    if(n==m*m) printf("%d\n",m);
    else printf("no");
    return 0;
}
```

Note 4.1 (要点).

1. 数据类型的强制转换, sqrt 函数原型: `double sqrt(double x);`

`m=(int)sqrt(n)`是函数调用语句, 等效于:

```
double y=sqrt(n); // n自动由int转化为double类型的数据，不会损失精度
m=(int)y;
```

2. 注意审题: “输入一个整数 n , ($0 \leq n \leq 1000000$)”, 因此, 0 也是一个合法输入。

4.2 成绩分级

给出一个百分制的成绩, 要求输出成绩等级'A','B','C','D','E'。90 分以上为'A',80~89分为'B',70~79分为'C',60~69分为'D', 60分以下为'E'。

输入说明

输入一个正整数 m ($0 \leq m \leq 100$)

输出说明

输出一个字符

输入样例

59

输出样例

E

```
#include <stdio.h>
```

```
int main()
{
    int grade;
    scanf("%d",&grade);
    grade /= 10;
    switch(grade)
    {
        case 0: case 1: case 2: case 3: case 4:
        case 5: printf("E"); break;
        case 6: printf("D"); break;
        case 7: printf("C"); break;
        case 8: printf("B"); break;
        case 9:
        case 10: printf("A"); break;

    }
    return 0;
}
```

Note 4.2 (要点). 熟练掌握 switch 语句。

4.3 abc 组合

已知 $abc+cba=n$, 其中 a,b,c 均为一位数, $1000<n<2000$, 编程求出满足条件的 a,b,c 所有组合。

输入说明

一个整数 n

输出说明

按照整数 abc 从小到大的顺序, 输出 a, b, c , 用空格分隔, 每输出一组 a, b, c 后换行。

输入样例

1352

输出样例

3 7 9

4 7 8

5 7 7

6 7 6

7 7 5

8 7 4

9 7 3

```
#include <stdio.h>
```

```
int main()
{
    int n,a,b,c;
    scanf("%d",&n);
    for(a = 0; a <= 9; a++)
        for(b = 0; b <= 9; b++)
            for(c = 0; c <= 9; c++)
                if(a*100+b*10+c + c*100+b*10+a == n)
                    printf("%d_ %d_ %d\n",a,b,c);
    return 0;
}
```

Note 4.3 (思考). 如何用数字分解技巧改写此题。

4.4 工资计算

小明的公司每个月给小明发工资，而小明拿到的工资为交完个人所得税之后的工资。假设他一个月的税前工资为 S 元，则他应交的个人所得税按如下公式计算：

1. 个人所得税起征点为 3500 元，若 S 不超过 3500，则不交税，3500 元以上的部分才计算个人所得税，令 $A=S-3500$ 元；
2. A 中不超过 1500 元的部分，税率 3
3. A 中超过 1500 元未超过 4500 元的部分，税率 10
4. A 中超过 4500 元未超过 9000 元的部分，税率 20
5. A 中超过 9000 元未超过 35000 元的部分，税率 25
6. A 中超过 35000 元的部分，税率 30

例如，如果小明的税前工资为 10000 元，则 $A=10000-3500=6500$ 元，其中不超过 1500 元部分应缴税 $1500 \times 3\%=45$ 元，超过 1500 元不超过 4500 元部分应缴税 $(4500-1500) \times 10\%=300$ 元，超过 4500 元部分应缴税 $(6500-4500) \times 20\%=400$ 元。总共缴税 745 元，税后所得为 9255 元。

已知小明这个月税前所得为 S 元，请问他的税后工资 T 是多少元。

输入格式

输入为一个整数 S ，表示小明的税前工资。所有评测数据保证小明的税前工资为一个整百的数。

输出格式

输出一个整数 T ，表示小明的税后工资。

样例输入

10000

样例输出

9255

评测用例规模与约定对于所有评测用例， $1 \leq T \leq 100000$ 。

```
#include <stdio.h>
```

```
int main()
{
    int S,T,A;
    float tax=0.0;
    scanf("%d",&S);
    A=S-3500;
    if(A<=0) tax=0;
    else
    {
        if(A<=1500) tax=A*0.03;
        else if(A>1500 && A<=4500)
            tax=1500*0.03+(A-1500)*0.1;
        else if(A>4500 && A<=9000)
            tax=1500*0.03+(4500-1500)*0.1+(A-4500)*0.2;
        else if(A>9000 && A<=35000)
            tax=1500*0.03+(4500-1500)*0.1+(9000-4500)*0.2+(A-9000)*0.25;
        else
            tax=1500*0.03+(4500-1500)*0.1+(9000-4500)*0.2+(35000-9000)
            *0.25+(A-35000)*0.3;
    }
    T=S-tax;
    printf("%d\n",T);
    return 0;
}
```

Note 4.4 (要点). 掌握基本条件语句。练习 if else 语句的各种组合形式。

4.5 跳一跳

跳一跳是一款微信小游戏，游戏规则非常简单，只需玩家要从一个方块跳到下一个方块，如果未能成功跳到下一个方块则游戏结束。

计分规则如下：

1. 如果成功跳到下一个方块上，但未跳到方块中心，加 1 分
2. 如果成功跳到下一个方块上，且刚好跳到方块中心，则第一次加 2 分，此后连续跳到中心时每次递增 2 分。也就是说，第一次跳到方块中心加 2 分，连续第二次跳到方块中心加 4 分，连续第三次跳到方块中心加 6 分,...，以此类推。
3. 如果未能成功跳到方块上，加 0 分，且游戏结束

现在给出玩家一局游戏的每次跳跃情况，请计算玩家最终得分。

输入说明

输入为若干个非零整数 (整数个数小于 1000), 表示玩家每次的跳跃情况。整数之间用空格分隔, 整数取值为 0, 1, 2。

0 表示未能成功跳到下一个方块上,

1 表示成功跳到下一个方块上但未跳到方块中心,

2 表示成功跳到下一个方块上, 且刚好跳到方块中心。

输入的数据只有最后一个整数是 0, 其余均非零。

输出说明

输出一个整数表示该玩家的最终得分。

输入样例

1 1 2 1 2 2 2 0

输出样例

17

```
#include <stdio.h>
// 无限循环, 符合结束条件, break
int main()
{
    // last 记录上一次的跳跃情况, num 表示连续跳至方框中心次数。
    int score=0, a, i, last=0, num=0;
    while(1) // 无限循环, a==0 时, break;
    {
        scanf("%d",&a);
        if(a==1) score++;
        if((last==1 || last==0) && a==2) // 第一次跳至中心
        {
            score=score+2;
            num=0; // 连续跳至中心清0
        }
        if(last==2 && a==2) // 连续跳至中心
        {
            score=score+2;
            num++;
        }
        if(last==2 && (a==1 || a==0)) // 连续跳至中心结束, 开始清算
        {
            for(i=1; i<=num; i++) // 结算递增情况
            {
                score=score+i*2;
            }
            num=0; // 已经结算, 清0连续跳至中心次数
        }
        if(a==0) break; // 结束
    }
}
```

```
        last=a; // 记录上一次的跳跃情况
    }
    printf("%d\n",score);
    return 0;
}
```

Note 4.5 (要点). 通过本题编程, 有助于训练自己的逻辑思维能力。

本题的 last 变量的使用是要点, 它记录上一次的跳跃情况。根据 last 与本次的跳跃情况的变量 a 的值, 进行条件分类即可得解。

连续跳至中心的次数用 num 变量记录, if(last=2 && (a==1 || a==0))条件成立时, 结算递增奖励。

4.6 车牌限行

受雾霾天气影响, 某市决定当雾霾指数超过设定值时对车辆进行限行, 假设车牌号全为数字, 且长度不超过 6 位, 限行规则如下:

1. 限行时间段只包括周一至周五, 周六周日不限行;
2. 如果雾霾指数低于 200, 不限行;
3. 如果雾霾指数大于等于 200 且低于 400, 每天限行两个尾号的汽车, 周一限行 1 和 6, 周二限行 2 和 7, 周三限行 3 和 8, 周四限行 4 和 9, 周五限行 5 和 0;
4. 如果雾霾指数大于等于 400, 每天限行五个尾号的汽车, 周一、周三和周五限行 1,3,5,7,9, 周二和周四限行 0,2,4,6,8。

现在给出星期几、雾霾指数和车牌号, 判断该车牌号是否限行。

输入说明

输入分为三个整数, 第一个整数表示星期几 (1~7, 1 表示周一, 2 表示周二, 依次类推, 7 表示周日), 第二个整数表示雾霾指数 (0~600), 第三个整数表示车牌号, 整数之间用空格分隔。

输出说明

输出为两个部分, 第一部分为车牌最后一位数字, 第二部分为限行情况, 限行输出 yes, 不限行输出 no。

输入样例

输入样例 1

4 230 80801

输入样例 2

3 300 67008

输出样例

输出样例 1

1 no

输出样例 2

8 yes

```
int main1()
{
```

```
int week, hazeIndex, No; // 星期几, 雾霾指数, 车牌号码
int LastNo; // 车牌号最后一位数字
int control=0; // 标志变量, 0: 不限行; 1: 限行

scanf("%d%d%d",&week,&hazeIndex,&No);
LastNo=No%10;
switch(week)
{
    case 1:
        if(hazeIndex>=200 && hazeIndex<400 && (LastNo==1 || LastNo==6))
            control=1;
        if(hazeIndex>=400 && (LastNo%2 != 0))
            control=1;
        break;
    case 2:
        if(hazeIndex>=200 && hazeIndex<400 && (LastNo==2 || LastNo==7))
            control=1;
        if(hazeIndex>=400 && (LastNo%2 == 0))
            control=1;
        break;
    case 3:
        if(hazeIndex>=200 && hazeIndex<400 && (LastNo==3 || LastNo==8))
            control=1;
        if(hazeIndex>=400 && (LastNo%2 != 0))
            control=1;
        break;
    case 4:
        if(hazeIndex>=200 && hazeIndex<400 && (LastNo==4 || LastNo==9))
            control=1;
        if(hazeIndex>=400 && (LastNo%2 == 0))
            control=1;
        break;
    case 5:
        if(hazeIndex>=200 && hazeIndex<400 && (LastNo==5 || LastNo==0))
            control=1;
        if(hazeIndex>=400 && (LastNo%2 != 0))
            control=1;
        break;
    case 6:
    case 7: break;
}
```



```
    if(control==1) printf("%d_yes",LastNo);
    else printf("%d_no",LastNo);
    return 0;
}

int main1() // 另解(三维数组标志变量)
{
    int week, hazeIndex, No; // 星期几, 雾霾指数, 车牌号码
    int LastNo; // 车牌号最后一位数字
    // 三维数组用作标志变量, 描述限行规则, 请分析数组的含义.
    int control[2][5][10]={
        // hazeIndex>=200 && hazeIndex<400
        {
            {0,1,0,0,0,1,0,0,0,0}, // 周一
            {0,0,1,0,0,0,0,1,0,0}, // 周二
            {0,0,0,1,0,0,0,0,1,0}, // 周三
            {0,0,0,0,1,0,0,0,0,1}, // 周四
            {1,0,0,0,1,0,0,0,0,0}, // 周五
        },
        // hazeIndex>=400
        {
            {0,1,0,1,0,1,0,1,0,1}, // 周一
            {1,0,1,0,1,0,1,0,1,0}, // 周二
            {0,1,0,1,0,1,0,1,0,1}, // 周三
            {1,0,1,0,1,0,1,0,1,0}, // 周四
            {0,1,0,1,0,1,0,1,0,1}, // 周五
        }
    };

    scanf("%d%d%d",&week,&hazeIndex,&No);
    LastNo=No%10;
    if(hazeIndex>=200 && hazeIndex<400)
    {
        if(control[0][week-1][LastNo]) printf("%d_yes",LastNo);
        else printf("%d_no",LastNo);
    }
    else if(hazeIndex>=400)
    {
        if(control[1][week-1][LastNo]) printf("%d_yes",LastNo);
        else printf("%d_no",LastNo);
    }
    else
```

```

    {
        printf("%d\n", LastNo);
    }
    return 0;
}

```

Note 4.6 (要点).

1. 首先假定标志变量的值 (如, `int control=0;`), 再根据题目要求, 计算它的真实值, 是基本技巧。
2. 用数组作为标志变量 (如, `control [2][5][10];` 是另一技巧。

4.7 气温波动

最近一段时间气温波动较大。已知连续若干天的气温, 请给出这几天气温的最大波动值是多少, 即在这几天中某天气温与前一天气温之差的绝对值最大是多少。

输入说明

输入数据分为两行。

第一行包含了一个整数 n , 表示给出了连续 n 天的气温值, $2 \leq n \leq 30$ 。

第二行包含 n 个整数, 依次表示每天的气温, 气温为 -20 到 40 之间的整数。

输出说明

输出一个整数, 表示气温在这 n 天中的最大波动值。

输入样例

```

6
2 5 5 7 -3 5

```

输出样例

```

10

```

```

#include <stdio.h>
#include <math.h>
// 不使用数组存储气温值
int main1()
{
    // last: 前一天的气温, temperature: 当天气温, undulation: 波动值
    int i, n, last, temperature, undulation = 0;
    scanf("%d", &n);

    // 当天气温 temperature 与前一天气温比较
    for (i = 0; i < n; i++)
    {
        scanf("%d", &temperature);
        // 注意 i==0 时, last 无值
    }
}

```

```
        if (i != 0 && fabs(temperature - last) > undulation)
            undulation = fabs(temperature - last);
        last = temperature;
    }
    printf("%d\n", undulation);
    return 0;
}

// 使用数组存储最多30个气温值
int main()
{
    int i, n, temperature[30], undulation = 0; // temperature数组: 气温值,
    undulation: 波动值
    scanf("%d",&n);
    for(i = 0; i < n; i++)
    {
        scanf("%d",&temperature[i]);
    }

    undulation= fabs(temperature[0]-temperature[1]); // 初始波动值

    for(i = 2; i < n-1; i++) // 注意数组边界, 保证数组不越界
    {
        if (fabs(temperature[i] - temperature[i+1]) > undulation)
            undulation = fabs(temperature[i] - temperature[i+1]);
    }

    printf("%d\n", undulation);
    return 0;
}
```

Note 4.7. 借助变量 last 表示前一天的气温, 即可不用数组存储所有数据, 是基本技巧。

Chapter 5

第 5 次机试练习：继续练习基本输入输出语句，分支与循环，简单数组应用

5.1 最小差值

给定 n 个数，请找出其中相差（差的绝对值）最小的两个数，输出它们的差值的绝对值。

输入格式

输入第一行包含一个整数 n 。

第二行包含 n 个正整数，相邻整数之间使用一个空格分隔。

输出格式

输出一个整数，表示答案。

样例输入

```
5
1 5 4 8 20
```

样例输出

```
1
```

样例说明

相差最小的两个数是 5 和 4，它们之间的差值是 1。

样例输入

```
5
9 3 6 1 3
```

样例输出

```
0
```

样例说明

有两个相同的数 3，它们之间的差值是 0。

数据规模和约定

对于所有评测用例， $2 \leq n \leq 1000$ ，每个给定的整数都是不超过 10000 的正整数。

```
#include <stdio.h>
#include <math.h>
#define N 10000 // 估计数组num的最大长度
int main()
{
    int i, j, n, num[N], smallest, temp;
```

```

scanf("%d",&n);
// 输入数组各元素
for(i=0;i<n;i++) // 实际数组的最大长度n, 下标由0到(n-1)
{
    scanf("%d",&num[i]);
}
// 初始的最小值就是前两个数的差值, 注意初始化值必须是实际存在的值, 而不能想当然给值。
smallest=(int) fabs(num[0]-num[1]); // 整数绝对值函数int abs(int x)在低版本编译器中有问题, 此处用双精度绝对值函数代替, 其结果转换为整数。

// 前后两项比较
for(i=0;i<=n-2;i++) // 循环变量i用于访问数组元素, 注意数组边界问题
{
    for(j=i+1;j<n;j++)
    {
        temp=(int) fabs(num[i]-num[j]);
        if(smallest>temp) smallest=temp;
    }
}
printf("%d\n",smallest);
return 0;
}

```

Note 5.1 (整数求绝对值函数). `int abs(int x)`; 在有些低版本编译器中, `math.h`头文件无此函数原型说明, 可用`double fabs(double x)`;代替。见本例。

5.2 PM2.5

给出一组 PM2.5 数据, 按以下分级标准统计各级天气的天数, 并计算出 PM2.5 平均值。PM2.5 分级标准为:

- 一级优 ($0 \leq \text{PM2.5} \leq 50$)
- 二级良 ($51 \leq \text{PM2.5} \leq 100$)
- 三级轻度污染 ($101 \leq \text{PM2.5} \leq 150$)
- 四级中度污染 ($151 \leq \text{PM2.5} \leq 200$)
- 五级重度污染 ($201 \leq \text{PM2.5} \leq 300$)
- 六级严重污染 ($\text{PM2.5} > 300$)

输入说明

输入分为两行,

第一行是一个整数 n 表示天数 ($1 < n \leq 100$);

第二行为 n 个非负整数 P_i ($0 \leq P_i \leq 1000$), 表示每天的 PM2.5 值, 整数之间用空格分隔。

输出说明

输出两行数据,

第一行为 PM2.5 平均值, 结果保留 2 位小数;

第二行依次输出一级优, 二级良, 三级轻度污染, 四级中度污染, 五级重度污染, 六级严重污染的天数。

输入样例

10

50 100 120 80 200 350 400 220 180 165

输出样例

186.50

1 2 1 3 1 2

```
#include <stdio.h>

int main()
{
    // 用数组变量day存储数据, 避免设置6个变量存储。
    int i = 0, n, pm25, day[6] = {0, 0, 0, 0, 0, 0}, sum = 0;
    scanf("%d", &n);
    while(i < n)
    {
        scanf("%d", &pm25);
        sum += pm25;
        if(pm25 >= 0 && pm25 <= 50 ) day[0]++;
        else if(pm25 >= 51 && pm25 <= 100 ) day[1]++;
        else if(pm25 >= 101 && pm25 <= 150 ) day[2]++;
        else if(pm25 >= 151 && pm25 <= 200 ) day[3]++;
        else if(pm25 >= 201 && pm25 <= 300 ) day[4]++;
        else day[5]++;
        i++;
    }
    printf("%.2f\n", (float)sum/n);
    for(i = 0; i < 6; i++) // 视作一条语句, 省略{ }
        if(i == 5) printf("%d\n", day[i]);
        else printf("%d□", day[i]);

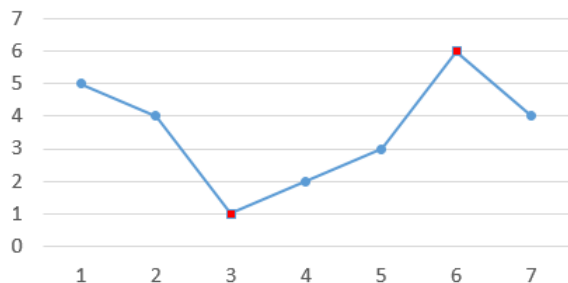
    return 0;
}
```

Note 5.2 (要点). `if() { } else if() { } else { }` 的用法, 循环语句的 `{ }`.

体会数组变量 `day` 的使用, 避免变量过多, 难于管理的麻烦。

5.3 折点计数

给定 n 个整数表示一个商店连续 n 天的销售量。如果某天之前销售量在增长, 而后一天销售量减少, 则称这一天为折点, 反过来如果之前销售量减少而后一天销售量增长, 也称这一天为折点, 其他的天都不是折点。如图所示, 第 3 天和第 6 天是折点。



给定 n 个整数 a_1, a_2, \dots, a_n 表示连续 n 天中每天的销售量。请计算出这些天总共有多少个折点。

输入说明

输入的第一行包含一个整数 n 。

第二行包含 n 个整数, 用空格分隔, 分别表示 a_1, a_2, \dots, a_n 。

$3 \leq n \leq 100$, 每天的销售量是不超过 1000 的非负整数。为了减少歧义, 输入数据保证: 在这 n 天中相邻两天的销售量总是不同的, 即 $a_{i-1} \neq a_i$ 。

输出说明

输出一个整数, 表示折点数量。

输入样例

7

5 4 1 2 3 6 4

输出样例

2

```
#include <stdio.h>
// 解法一, 使用标志变量up, 判断是否后一数据大于前一数据.
int main()
{
    // 估计数组a的实际长度为100, 实际长度是n(待输入的值)
    int i = 0, points = 0, n = 7, a[100] = {5, 4, 1, 2, 3, 6, 4}; // 把样例数据作为初始化, 方便调试
    int up; // 标志变量

    // 有了初始化数据, 这些输入语句在调试时就可注释掉, 达到快速调试程序逻辑的目的。
    scanf("%d", &n); // 输入数组a的实际长度
    // 输入各元数值
    for(i = 0; i < n; i++) scanf("%d", &a[i]);

    // 标志变量up的初始值必须是真实存在的值, 不要想当然。
```



```

    up = a[1] > a[0] ? 1 : -1; // 如果a[1] > a[0]成立, up=1,否则up=-1
    for(i = 2; i < n; i++)
    {
        if((a[i] > a[i-1] && up < 0) || (a[i] < a[i-1] && up > 0))
            points++;
        up = a[i] > a[i-1] ? 1 : -1;
    }

    printf("%d\n", points);
    return 0;
}

// 解法二, 通过判断a[i]前后相邻两数据, 确定是否折点.
int main()
{
    // 估计数组a的实际长度为100, 实际长度是n(待输入的值)
    int i = 0, points = 0, n = 7, a[100] = {5,4,1,2,3,6,4}; // 把样例数据作为初始化, 方便调试
    scanf("%d",&n);
    for(i=0;i<n;i++) scanf("%d",&a[i]);

    for(i=1;i<n-1;i++) // 注意数组边界, 不要越界。
    {
        if((a[i]<a[i-1] && a[i]<a[i+1]) || (a[i]>a[i-1] && a[i]>a[i+1]))
            points++;
    }
    printf("%d\n", points);
    return 0;
}

```

Note 5.3 (要点). 善用标志变量, 标志变量的初始值必须是真实存在的值, 不要想当然。

样例数据作为初始化数据, 调试时注释掉输入语句, 便于进行快速调试程序。

5.4 寻找最大整数

从键盘输入四个整数, 找出其中的最大值并将其输出。

输入说明

输入 4 个整数, 用空格分隔

输出说明

输出值最大的一个整数

输入样例

25 99 -46 0

输出样例

99

```
#include <stdio.h>
// 不用存储整数序列, 采用一条循环语句, 合并输入和计算, 减少出错概率。
int main()
{
    int i, num, max;
    // 输入, 并计算
    for(i = 0; i < 4; i++)
    {
        if (i==0) scanf("%d",&max); // 假定第一个数就是最大的数
        else
        {
            scanf("%d",&num);
            if(num > max) max=num;
        }
    }
    printf("%d\n",max);
    return 0;
}

int main1() // 另解, 存储整数序列
{
    int i, num[4], max;
    // 输入
    for(i = 0; i < 4; i++)
        scanf("%d",&num[i]);
    // 假定的最大值必须是实际存在的, 不要想当然是0,9999,等等。
    max = num[0];
    for(i = 0; i < 4; i++)
        if(max < num[i]) max = num[i];

    printf("%d\n",max);
    return 0;
}
```

Note 5.4 (要点). 题目虽然简单, 你能体会哪种解法更好? 特别注意假定变量的值必须是实际存在的数。

5.5 ISBN 号码

每一本正式出版的图书都有一个 ISBN 号码与之对应，ISBN 码包括 9 位数字、1 位识别码和 3 位分隔符，其规定格式如“x-xxx-xxxxx-x”，其中符号“-”是分隔符（键盘上的减号），最后一位是识别码，例如 0-670-82162-4 就是一个标准的 ISBN 码。

ISBN 码的首位数字表示书籍的出版语言，例如 0 代表英语；

第一个分隔符“-”之后的三位数字代表出版社，例如 670 代表维京出版社；

第二个分隔之后的五位数字代表该书在出版社的编号；

最后一位为识别码。识别码的计算方法如下：

首位数字乘以 1 加上次位数字乘以 2……以此类推，用所得的结果 mod 11，所得的余数即为识别码，如果余数为 10，则识别码为大写字母 X。例如 ISBN 号码 0-670-82162-4 中的识别码 4 是这样得到的：对 067082162 这 9 个数字，从左至右，分别乘以 1, 2, ..., 9，再求和，即 $0 \times 1 + 6 \times 2 + \dots + 2 \times 9 = 158$ ，然后取 $158 \bmod 11$ 的结果 4 作为识别码。

编写程序判断输入的 ISBN 号码中识别码是否正确，如果正确，则仅输出“Right”；如果错误，则输出正确的 ISBN 号码。

输入说明

输入只有一行，是一个字符序列，表示一本书的 ISBN 号码（保证输入符合 ISBN 号码的格式要求）。

输出说明

输出一行，假如输入的 ISBN 号码的识别码正确，那么输出“Right”，否则，按照规定的格式，输出正确的 ISBN 号码（包括分隔符“-”）。

输入样例

样例输入 1

0-670-82162-4

样例输入 2

0-670-82162-0

输出样例

样例输出 1

Right

样例输出 2

0-670-82162-4

```
#include <stdio.h>
int main()
{
    // 按题意，存储ISBN需要13个字符，再加上字符串结束字符'\0'.
    // 样例数据作为初始化，方便调试
    char ISBN[14] = "0-670-82162-4"; // ISBN[13]='\0'
    //char ISBN[14] = "0-670-82162-0";
    int i, j, sum=0, r; // r=sum % 11

    // 调试时，注释输入语句
    // 末尾自动添加'\0'.
```

```
scanf("%s",ISBN); // 或 gets(ISBN);

// 对识别码之前的数字求和, 注意边界.
for(i = 0,j = 1; i < 11; i++)
{
    if (ISBN[i]=='-') continue;
    sum += (ISBN[i]-'0')*j; // 整数与单个数字字符的关系: 9 = '9' - '0'
    j++;
}

r=sum%11;

// 特别处理识别码 ISBN[12]
if (r==10)
{
    if (ISBN[12]=='X') printf("Right\n");
    else
    {
        ISBN[12]='X';
        printf("%s\n",ISBN); // 或 puts(ISBN);
    }
}
else
{
    // 数字转字符 r+'0'
    if (ISBN[12]== r+'0') printf("Right\n");
    else
    {
        ISBN[12]= r+'0';
        printf("%s\n",ISBN); // 或 puts(ISBN);
    }
}

return 0;
}
```

Note 5.5 (要点).

1. 仔细审题, 考虑输入的识别码, 可能是'X' 的情况.
2. 定义字符数组表示字符串时, 且记给'\0'留一个字符的位置, 表示字符串的结尾.
3. ASCII编码(整数)=字符-'0';
4. 字符=ASCII编码(整数)+'0';

5. 整数可以表示字符的 ASCII 编码 (整数), 整数和字符类型可以 “混用”, 详见课件。

```
int a; char c='A';  
a = c+1; // c当作整数运算  
printf("%d_%c_%d_%c\n",a,a,c,c); //66 B 65 A
```


Chapter 6

第 6 次机试练习：函数，流程控制，字符串，数组

6.1 歌德巴赫猜想

德巴赫猜想：任意一个大偶数都能分解为两个素数的和，对于输入的一个正偶数，写一个程序来验证歌德巴赫猜想。

由于每个正偶数可能分解成多组素数和，仅输出分解值分别是最小和最大素数的一组，按从小到大顺序输出。

输入说明

输入一个正偶数 n ， $1 < n < 1000$ 。

输出说明

输出分解出的两个最小和最大素数。

输入样例

10

输出样例

3 7

```
#include <stdio.h>
#include <math.h>

// 判断参数n是否素数，如果是返回1，否则返回0
int isPrime(int n)
{
    int i;
    if(n < 2) return 0; // 最小素数是2，1不是素数也不是合数，题意(1<n<1000)
    不包含1,因此此语句不是必须的
    //for(i = 2; i <= sqrt((double)n); i++) // vs2013编译器要求数学函数严格
    按原型解释
    for(i = 2; i <= sqrt(n); i++) // 或条件表达式：i*i<=n
    {
        if (n%i == 0) return 0; // n不是素数
    }
    return 1; // n是素数
}
```

```
}

// 解法一: 递归函数
// 第1个参数: i是递归可能的素数(第一个可能的素数是big_even-1), 第2个参数:
// big_even是大偶数
void recursiveComputing(int i, int big_even)
{
    if(isPrime(i) && isPrime(big_even-i)) // 两个素数和是big_even
    {
        // 注意, 较小的在前
        printf("%d□%d\n", big_even - i, i); // 输出后, return; 结束递归
    }
    else // 递归
    {
        recursiveComputing(i-1, big_even); // 递归调用
    }
    return; // 递归结束
}

int mian()
{
    int num; // num大偶数
    scanf("%d", &num);
    recursiveComputing(num-1, num);
    return 0;
}

// 解法二: 二重循环, 从最小素数开始迭代计算, 获取符合题意的两个素数
int main1()
{
    int j, k, num; // num大偶数
    int flag; // 标志变量: 用于标识是否找到符合要求的素数对
    scanf("%d", &num);
    // 对于大偶数num, 分解为两个素数
    for(j = 2; j < num; j++) // 找出第一个素数(最小的) for #1
    {
        if(!isPrime(j)) continue; // 如果j不是素数, 继续下一轮迭代

        flag = 0; // 初始化, 未找出素数对
        for(k = j + 1; k < num; k++) // 找出第二个素数 for #2
        {
```



```
        if(isPrime(k) && j+k == num) // j是最小素数, k必然是最大素数
        {
            printf("%d_ %d\n",j,k);
            flag = 1; // 找出素数对, 如果没有此设置, 将会输出多组, 例如
num=2020时会多组素数
            break; // break for #2
        }
    }
    if(flag) break; // break for #1
}
return 0;
}
```

// 解法三: 不用标志变量版本

```
int main2()
{
    int j,k,num; // num大偶数
    scanf("%d",&num);
    // 对于大偶数num, 分解为两个素数
    for(j = 2; j < num; j++) // 找出第一个素数(最小的) for #1
    {
        if(!isPrime(j)) continue; // 如果j不是素数, 继续下一轮迭代

        for(k = j + 1; k < num; k++) // 找出第二个素数 for #2
        {
            if(isPrime(k) && j+k == num) // j是最小素数, k必然是最大素数
            {
                printf("%d_ %d\n",j,k);
                return 0; // 找出素数对, 结束主函数。 如果不结束, 将会输出
多组
            }
        }
    }
    return 0;
}
```

// 解法四: 优化, 根据题意找出一组: 最小素数+最大素数=偶数

```
int main3()
{
    int j,k,num; // num大偶数
    scanf("%d",&num);
```

```

// 对于大偶数num, 分解为两个素数
for(j = 2; j < num; j++) // 找出第一个素数(最小的) for #1
{
    if(!isPrime(j)) continue; // 如果j不是素数, 继续下一轮迭代

    for(k = num-1; k>=2; k--) // 找出第二个素数(最大的) for #2
    {
        // j是最小素数, 判断k是否是最大素数并且二者之和=num
        if(isPrime(k) && j+k == num)
        {
            printf("%d_%d\n", j, k);
            return 0; // 找出素数对, 结束主函数。如果不结束, 将会输出多
组
        }
    }
}
return 0;
}

// 解法五: 一重循环, 从最小素数开始迭代计算, 获取符合题意的两个素数
int main4()
{
    int num, i;
    scanf("%d", &num);

    for(i = 2; i < num; i++)
    {
        if(isPrime(i)) // i是素数
        {
            if(isPrime(num-i)) // 如果num-i也是素数, 即满足题意num=i+j
            {
                printf("%d_%d\n", i, num-i);
                break;
            }
        }
    }
    return 0;
}

```

Note 6.1 (要点). 再次体会标志变量的用法及内层循环前的初始化。仔细审题, 本题要求一组输出: 最小素数 + 最大素数 = 偶数。

如果要求找出最接近的一组素数, 因为 $n = \frac{n}{2} + \frac{n}{2}$, 如果 $\frac{n}{2}$ 是素数, 即为所求。否则, 所求的两个素数在 $\frac{n}{2}$ 附近, 分别小于和大于 $\frac{n}{2}$ 。

例如: 修改 main4() 中的 `if(isPrime(num-i))` 为 `if(isPrime(num-i) && num-i <= i)`, 则输出 num-i,i; 即可。
或者: 由 $i = \frac{n}{2} \rightarrow i = 2$ 迭代, 第一个找到的素数 i 和 $num - i$ 即为所求。

6.2 矩阵

请写一个程序, 对于一个 m 行 m 列 ($2 < m < 20$) 的方阵, 求其每一行、每一列及主、辅对角线元素之和, 然后按照从大到小的顺序依次输出这些值。

注: 主对角线是方阵从左上角到右下角的一条斜线, 辅对角线是方阵从右上角到左下角的一条斜线。

输入说明

输入数据的第一行为一个正整数 m ;

接下来为 m 行、每行 m 个整数表示方阵的元素。

输出说明

从大到小排列的一行整数, 每个整数后跟一个空格, 最后换行。

输入样例

```
4
15 8 -2 6
31 24 18 71
-3 -9 27 13
17 21 38 69
```

输出样例

```
159 145 144 135 81 60 44 32 28 27
```

```
#include <stdio.h>
// 估计方阵行列数
#define M 20

// input, m是实际方阵行列数
void input(int matrix[][M], int m)
{
    int i, j;
    for(i = 0; i < m; i++)
        for(j = 0; j < m; j++)
            scanf("%d", &matrix[i][j]);
}

// 计算主对角线之和, m是实际方阵行列数
int main_diagonal(int matrix[][M], int m)
{
    int i, j, sum = 0;
```

```
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < m; j++)
        {
            if(i == j) sum += matrix[i][j];    // 主对角线
        }
    }
    return sum;
}

// 计算副对角线之和, m是实际方阵行列数
int counter_diagonal(int matrix[][M], int m)
{
    int i, j, sum = 0;
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < m; j++)
        {
            if(j == m-i-1) sum += matrix[i][j];    // 副对角线之和
        }
    }
    return sum;
}

// 计算第i行之和, m是实际方阵行列数
int sumI(int matrix[][M], int m, int i)
{
    int j, sum = 0;
    for(j = 0; j < m; j++)    // 遍历列
    {
        sum += matrix[i][j];    // 第i行之和
    }
    return sum;
}

// 计算第j行之和, m是实际方阵行列数
int sumJ(int matrix[][M], int m, int j)
{
    int i, sum = 0;
    for(i = 0; i < m; i++)    // 遍历行
    {
```

```
        sum += matrix[i][j]; // 第j行之和
    }
    return sum;
}

// 交换两个元素值
void swap(int *p1, int *p2)
{
    int temp;
    temp = *p1; *p1 = *p2; *p2 = temp;
}

// 选择法排序（降序）
void sorts(int a[], int n)
{
    int i, j, k;
    for(i = 0; i < n-1; i++)
    {
        k = i;
        for (j = i+1; j < n; j++)
            if(a[j] > a[k]) k = j;
        if (k != i) swap(&a[i], &a[k]);
    }
}

int main()
{
    int matrix[M][M], a[2*M+2]; // 以估计行列数，定义数组
    int i, m;
    scanf("%d", &m); // 实际方阵行列数

    input(matrix, m); // input

    // 调用各函数，装配数组a
    int n = 0; // 记录数组a的实际长度
    for(i = 0; i < m; i++)
    {
        a[n++] = sumI(matrix, m, i); // 第i行之和
        a[n++] = sumJ(matrix, m, i); // 第i列之和
    }
}
```

```
a[n++] = main_diagonal(matrix,m);    // 主对角线之和
a[n++] = counter_diagonal(matrix,m);  // 副对角线之和

// 排序数组a
sorts(a,n);

// 输出
for(i = 0; i < n; i++)
    printf("%d□",a[i]);

printf("\n");

return 0;
}
```

Note 6.2 (要点).

1. 思路: 定义功能单一的函数, 实现简单功能, 主程序调用各个函数。
2. 一维数组 $a[2*M+2]$ 存储相关函数计算结果, 排序数组 a 即是所求。
3. 避免过多循环嵌套, 不易出错, 简化程序设计。
4. 但是缺点是在各函数中分别循环遍历方阵, 效率低。
5. 优化方案是不采用独立函数计算, 在主函数中一次遍历方阵, 计算各值。
6. 二维数组表示矩阵, 是常见题型, 必须熟练掌握元素的下标规律及其遍历技巧。

6.3 回文数

若一个非负整数其各位数字按照正反顺序读完全相同, 则称之为回文数, 例如 12321。判断输入的整数是否是回文数。若是, 则输出该整数各位数字之和, 否则输出 no。

输入说明

输入为一个整数 n , $0 \leq n < 100000000$ 。

输出说明

若该整数为回文数, 则输出整数各位数字之和, 否则输出 no。

输入样例

样例 1 输入

131

样例 2 输入

24

输出样例

样例 1 输出

5

样例 2 输出

no

```
#include <stdio.h>
#include <string.h>

// 思路1: 求该整数的反序组成的整数, 如整数1234, 其反序整数即为4321, 如果二者相等即为回文数
// 判断num是否是回文数, 是: 返回1; 不是, 返回0
// 指针参数*sum, 返回这个数的各位之和
int isPalindromic1(int num, int *sum)
{
    int reverse = 0, tmp = num;
    *sum = 0; // 初始化指针内容
    while(tmp)
    {
        reverse = reverse*10 + tmp%10;
        *sum += tmp%10;
        tmp /= 10;
    }
    if(reverse == num ) return 1;
    else return 0;
}

// 思路2: 构造数组reverse, 反序存储该整数各位数, 按照数组下标, 前后数组元素相等则为回文数
// 判断num是否是回文数, 是: 返回1; 不是, 返回0
// 指针参数*sum, 返回这个数的各位之和
int isPalindromic2(int num, int *sum)
{
    int reverse[9], len=0, i=0; // 依题意数组最大长度为9, 最多存储9位数。实际长度用len变量表示

    *sum = 0; // 初始换指针内容
    //构造数组reverse, 反序存储该整数各位数
    while(num)
    {
        reverse[i]=num%10;
        *sum += num%10; // 累加各位数字
        num /= 10;
        len++; // 计算数组实际长度
        i++;
    }
}
```

```
//按照数组下标, 前后数组元素相等则为回文数
for(i=0; i<len/2; i++)
{
    if(reverse[i] != reverse[len-i-1]) return 0; // 不是回文数
}
return 1; // 至此, 必然是回文数
}

// 测试方法1和2的主程序
int main12()
{
    int i, num, sum;

    scanf("%d", &num);
    // if(isPalindromic1(num, &sum)) printf("%d\n", sum);
    if(isPalindromic2(num, &sum)) printf("%d\n", sum);
    else printf("no\n");
    return 0;
}

// 思路3: 按照字符串处理输入的整数, 前后数组元素相等则为回文数
int main3()
{
    char s[10]; // 留出'\0', 最多存储9位数。实际长度用len变量表示
    int sum=0, len=0, i=0;

    // 以字符串形式接收输入的整数, 末尾自动追加'\0'
    gets(s);

    // 计算len, 或者len=strlen(s), 同时计算各位数字之和
    for(len=0; s[i]!='\0';)
    {
        len++;
        sum=sum+s[i]-'0'; // 计算各位数字之和
        i++;
    }

    //按照数组下标, 前后数组元素相等则为回文数
    for(i=0; i<len/2; i++)
    {
        if(s[i] != s[len-i-1]) // 不是回文数
```



```
        {
            printf("no\n");
            return 0; // 主函数结束
        }
    }
    // 至此，必然是回文数，包含一位数不进入上面的for循环的情况
    printf("%d\n",sum);
    return 0;
}

// 思路4： 按照字符串处理输入的整数，前后数组元素相等则为回文数。
// 使用指针操作
int main()
{
    char s[10]; // 留出'\0'，最多存储9位数。实际长度用len变量表示
    int sum=0;
    char *p1=s,*p2=s; // 用于正序和反序遍历s数组，初始指向第一个元素

    // 以字符串形式接收输入的整数，末尾自动追加'\0'
    gets(s);

    // 用p2遍历字符串，同时计算各位数字之和
    for (;*p2!='\0';p2++)
    {
        sum=sum+(*p2)-'0'; // 计算各位数字之和
    }
    // 至此，p2指向最后一个元素'\0'，我们使它指向最后一个有效元素：
    p2--;

    //按照数组下标，前后数组元素相等则为回文数
    for (;p1<p2;p1++,p2--)
    {
        if(*p1 != *p2) // 不是回文数
        {
            printf("no\n");
            return 0; // 主函数结束
        }
    }
    // 至此，必然是回文数
    printf("%d\n",sum);
    return 0;
}
```

```
}
```

Note 6.3 (要点).

1. 掌握函数的地址传递方法。
2. 使用两个指针变量 p1, p2, 其中 p1 指向待查找子串的首字母, 另一个指向末尾, p1++, p2--; 是判断字符串是否是回文的有效技巧。

6.4 马鞍点

若一个矩阵中的某元素在其所在行最小而在其所在列最大, 则该元素为矩阵的一个马鞍点。请写一个程序, 找出给定矩阵的马鞍点。

输入说明

输入数据第一行只有两个整数 m 和 n ($0 < m < 100, 0 < n < 100$), 分别表示矩阵的行数和列数;

接下来的 m 行、每行 n 个整数表示矩阵元素 (矩阵中的元素互不相同), 整数之间以空格间隔。

输出说明

在一行上输出马鞍点的行号、列号 (行号和列号从 0 开始计数) 及元素的值 (用一个空格分隔), 之后换行; 若不存在马鞍点, 则输出一个字符串 “no” 后换行。

输入样例

```
4      3
11      13      121
407      72      88
23      58      1
134      30      62
```

输出样例

```
1 1 72
```

```
#include <stdio.h>
// 估计的二维数组最大行列数
#define M 100
#define N 100

// 判断a[row,col]是否是马鞍点, 是: 返回1; 否则返回0
// m,n是二维数组实际行列数
int compute(int a[][N], int m, int n, int row, int col)
{
    int i, element = a[row][col];
    // element在其所在行最小而在其所在列最大
    for(i = 0; i < n; i++)
        if(a[row][i] < element) return 0; // 不是马鞍点, 直接返回0
    for(i = 0; i < m; i++)
        if(a[i][col] > element) return 0; // 不是马鞍点, 直接返回0
```

```
    return 1;    // 如果执行至此，肯定是马鞍点，直接返回0
}

int main()
{
    int matrix[M][N]; // 按照估计的最大行列数定义二维数组
    int i, j, m, n, flag = 0;

    scanf("%d%d", &m, &n); // 实际行列数

    // input
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
            scanf("%d", &matrix[i][j]);

    // 遍历二维数组，判断马鞍点
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
        {
            if(compute(matrix, m, n, i, j))
            {
                printf("%d□%d□%d\n", i, j, matrix[i][j]);
                flag = 1;
            }
        }
    }
    if (!flag) printf("no\n");

    return 0;
}
```

Note 6.4 (要点).

1. 思路：定义函数计算单个元素 $a[i,j]$ 是否是马鞍点，主程序遍历二维数组，调用此函数。
2. 避免过多循环嵌套，不易出错，简化程序设计。
3. 掌握二维数组作为函数参数的定义, 调用方式。

6.5 密码强度

每个人都有很多密码, 你知道你的密码强度吗? 假定密码由大写字母、小写字母、数字和非字母数字的符号这四类字符构成, 密码强度计算规则如下:

1. 基础分: 空密码 (密码长度为零)0 分, 非空密码 1 分
 2. 加分项 1: 密码长度超过 8 位, +1 分
 3. 加分项 2: 密码包含两类不同字符 +1 分, 包含三类不同字符 +2 分, 包含四类不同字符 +3 分
- 按照此规则计算的密码强度为 0~5。请你设计一个程序计算给出的密码的强度。

输入说明

输入为一个密码字符串, 字符串长度不超过 50 个字符。

输出说明

输出一个整数表示该密码的强度。

输入样例

输入样例 1

abcd

输入样例 2

ab123

输出样例

样例 1 输出:

1

样例 2 输出

2

```
#include <stdio.h>
#include <string.h>
int main()
{
    char p[51]; // 记得给 '\0' 留位置
    int i, strength = 0;
    // class4[0]=1 大写字母, class4[1]=1 小写字母, class4[2]=1 数字, class4
    [3]=1 非字母数字
    int class4[4] = {0,0,0,0};

    //scanf("%s",p); // 不能完整接收含空格的字符串和空密码
    gets(p); // last char: '\0', 直接回车, 就是空密码

    // 1. 基础分: 空密码(密码长度为零)0分, 非空密码1分
    if(strlen(p) == 0) strength += 0;
    else strength += 1;

    // 2. 加分项1: 密码长度超过8位, +1分
    if(strlen(p) > 8) strength += 1;
```

```

// 3. 加分项2: 密码包含两类不同字符+1分, 包含三类不同字符+2分, 包含四
类不同字符+3分
for(i = 0; p[i] != '\0'; i++)
{
    if(p[i] >= 'A' && p[i] <= 'Z') class4[0] = 1;
    else if(p[i] >= 'a' && p[i] <= 'z') class4[1] = 1;
    else if(p[i] >= '0' && p[i] <= '9') class4[2] = 1;
    else class4[3] = 1;
}
int c = 0;
for(i = 0; i < 4; i++) c += class4[i];

if(c >= 4) strength += 3;
else if(c >= 3) strength += 2;
else if(c >= 2) strength += 1;

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

return 0;
}

```

Note 6.5 (要点). 字符串处理的典型问题: '\0', 字符串相关函数 `char s1[81], s2[81]; strlen(s1), strcmp(s1, s2), strcpy(s1, s2); scanf("%s", s1), gets(s1)` 的区别等, 应该充分掌握。

6.6 数字分解排序

输入一个 9 位以内的正整数 n , 按数值从高到低的顺序输出 n 的各位数字。

输入说明

一个正整数 $n(0 < n < 1000000000)$

输出说明

按数值从高到低的顺序输出 n 的各位数字, 数字之间用空格隔开

输入样例

564391

输出样例

9 6 5 4 3 1

```

#include <stdio.h>

void order(int a[], int n); // 排序函数说明

```

```
int main()
{
    int i = 0, j, k, n, num[9];
    scanf("%d", &n);
    k=0; // 数字个数
    while(n) // 构造数组num存储n的各位数字(从最低位到最高位存储)
    {
        num[i++] = n%10;
        n /= 10;
        k++;
    }

    // 输出验证, 调试技巧之一。
    //for(j = 0; j < k; j++) printf("%d ", num[j]);
    //    printf("\n");

    order(num, k); // 排序函数

    // 输出
    for(j = 0; j < k; j++) printf("%d□", num[j]);

    printf("\n");
    return 0;
}

// 冒泡排序函数
void order(int a[], int n)
{
    int i, j, t, flag;
    for(j = 1; j <= n-1; j++)
    {
        flag=0; // 且记! 必须在进入内层循环前初始化。
        for(i = 0; i < n - j; i++)
            if (a[i] > a[i+1])
            {
                t = a[i]; a[i] = a[i+1]; a[i+1] = t;
                flag=1;
            }
        if(!flag) break;
    }
}
```

Note 6.6 (要点). 其它解法及其变种见课件。

排序函数必须掌握，注意检查数组是否越界问题。