E3-2021级航类-第三次练习赛题解

A 数数逆序对

难度	考点
1	循环

题目分析

两层循环找出所有 $i < j$ 的序对 (a_i, a_j) 然后判断 a_i 是否大于 a_j 即可

参考代码

```
#include <stdio.h>
int a[1005];
int n, ans; //全局变量默认初始值为0
int main()
{
    scanf("%d", &n);

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

    for (int i = 0; i < n; i++) //外层循环i从0～n-1
    {
        for (int j = i + 1; j < n; j++) //内层循环j从i+1～n-1
            if (a[i] > a[j])            //判断
                ans++;                  //累加
    }
    printf("%d\n", ans);

    return 0;
}
```

B 完全数!

难度	考点
1	循环嵌套

题目分析

依照题意，对于给定的x，我们需要进行两步操作，其一找到其所有的真因子并求和，其二判断和与x是否相等，我们可以通过一个双重循环来实现这一功能。

参考代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define ll long long
int main()
{
    int n,x;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&x);
        int tmp=0;
        for(int j=1;j<x;j++)
        if(x%j==0) tmp+=j; //求所有真因子之和
        if(tmp==x) printf("YES\n"); //判断是否为完全数
        else printf("NO\n");
    }
    // system("pause");
    return 0;
}
```

其实，小于10000的完全数只有4个，分别是$6, 28, 496, 8128$，可以通过一个`if`语句直接判断。

C 求e的近似值

难度	考点
1	循环

题目分析

依照题目要求，我们很自然的能想到通过一个简单的循环将这个表达式模拟出来：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define ll long long
int main()
{
    int n;
    scanf("%d",&n);
    double tmp=1.0,ans=1.0; //设置初始值
    for(int i=1;i<=n;i++)
    {
        tmp*=i; //计算i的阶乘
        ans+=1/tmp; //ans累加第i项的值
    }
    printf("%.14lf\n",ans); //输出小数点后十四位
    // system("pause");
```

```
    return 0;
}
```

在计算阶乘的过程中，我们可以通过一个在循环外定义一个变量，随着循环的进行依次累乘，即代码中的`tmp`。

但提交这份代码后，我们会发现正如HINT中提到的那样，它`TLE`（超时）了。

这时我们可以思考一下怎样减少循环次数，可以尝试改变n的值，看看答案的变化情况，多次尝试后我们会发现当$n \geq 17$时，结果便不再变化了，所以对于$n \geq 17$的情况，我们至多只用循环17次即可。

至于为什么是17，我们可以计算一下$\frac{1}{17!}$，它大约等于2.81×10^{-15}，其之后的数对我们要求的14位的精度不会再有影响。

很多工程上的问题也是需要我们进行类似的取舍的，当迭代次数到达一定值的时候便可以停止迭代从而保证时间效率。

参考代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define ll long long
int main()
{
    int n;
    scanf("%d",&n);
    if(n>17) n=17;
    double tmp=1.0,ans=1.0;
    for(int i=1;i<=n;i++)
    {
        tmp*=i;
        ans+=1/tmp;
    }
    printf("%.14lf\n",ans);
    // system("pause");
    return 0;
}
```

D 统计胜场数

难度	考点
2	循环

题目分析

“统计单词”的复习题，巧用数组下标，`win[i]`表示球队`i+'A'`的获胜场数。

数据输入用以下两个示例代码都可以，注意用`while (getchar() != '\n')`可能会导致死循环，因为输入数据的最后一行可能没有换行符；

由于限定了每支参赛队伍只能与其他队伍相遇一次，因此每支球队胜场数最多为 $n-1$，按字母表顺序枚举每一支球队，看是否有赢 $n-1, n-2, \ldots, 0$ 场比赛的球队，有就直接输出，即能达到排序的效果。

示例代码1

```c
#include <stdio.h>

int main() {
    char c1, c2, result;
    int n, win[10] = {0}, i, j;
    scanf("%d", &n);
    // scanf语句一开始先写一个换行符\n，目的是匹配上一行未读取的换行符
    while (scanf("\n%c %c %c", &c1, &result, &c2) != EOF) {
        if (result == '>')
            win[c1 - 'A']++;
        else
            win[c2 - 'A']++;
    }
    for (i = n - 1; i >= 0; i--) {
        for (j = 0; j < n; j++) {
            if (win[j] == i)
                printf("%c:%d\n", 'A' + j, i);
        }
    }
    return 0;
}
```

示例代码2

```c
#include <stdio.h>

int main() {
    char c1, c2, result, c;
    int n, win[10] = {0}, i, j;
    scanf("%d", &n);
    while ((c = getchar()) != '\n');    // 读入n之后肯定有换行符，因此这个语句安全
    while (scanf("%c %c %c", &c1, &result, &c2) != EOF) {
        if (result == '>')
            win[c1 - 'A']++;
        else
            win[c2 - 'A']++;
        // 循环里就要采取这样的写法
        c = getchar();                      // 尝试读取一个字符
        while (c != '\n' && c != EOF)       // 若该字符不是换行符且不是EOF
            c = getchar();                  // 再继续读取
    }
    for (i = 4; i >= 0; i--) {
        for (j = 0; j < n; j++) {
            if (win[j] == i)
                printf("%c:%d\n", 'A' + j, i);
        }
    }
    return 0;
```

```
}
```

E 直线数局

难度	考点
3	模拟

题目分析

本题是一个简单的小型模拟问题，即在一个数组上移动，判断能否到达特定格。

本题难点主要在于走不出去的部分。许多同学只判断了从两边“飞出”的情形。但还有另一种情况是在数组的其中几个点“来回横跳”的情形。这种情况可以等效为“不能经过同一个点两次”，在程序设计中有一种常用的手段，即用一个`via`数组表示已经到达的点。

示例代码

```c
#include <stdio.h>

int numLine[100];
int path[100];

int main()
{
    int n = 0, i = 0, arrive = 0, num = 0, maxIndex = 0;
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        scanf("%d", &numLine[i]);
    }

    i = 0;

    while (i >= 0 && i < n) {

        if (i == n - 1) {
            arrive = 1;
            break;
        }

        if (path[i] == 1) {
            break;
        }
        path[i] = 1;

        i += numLine[i];

        if (i > maxIndex && i < n) {
            maxIndex = i;
        }

        num++;
```

```
    }

    if (arrive) {
        printf("True\n");
        printf("%d", num);
    } else {
        printf("False\n");
        printf("%d", maxIndex);
    }

    return 0;
}
```

F 溢0数

难度	考点
3	二进制

题目分析

本题是一个数学问题，在分析时，可以考虑十进制下的平行情况，由此发现，只需要统计每个乘数分解后，其因子2的个数即可

由于计算机储存数据本身就是二进制，所以可以利用位运算来简化代码

示例代码

```
#include<stdio.h>
int main()
{
    int n;
    int sum;
    int i,j,k;
    int p;

    while( scanf("%d",&n) != EOF )
    {
        sum = 0;
        for(i = 1 ; i <= n ; i++)
        {
            scanf("%d",&p);
            for(j = 0 ; ; j++)//乘数中因子2的个数，即乘数二进制末尾的0的个数
            {
                if( (p>>j) & 1)
                    break;
                else
                    sum++;
            }
        }
        printf("%d\n",sum);
    }
 }
```

G 高精度加法

难度	考点
4	数组

问题分析

- 什么情况下要**使用高精度**?

当两个数超过long long的大小并且要对这两个大数进行运算的时候。

- 既然数这么大，我们用什么存放呢?

用**字符串**存放。

- 怎么运算呢?

从最低位开始计算，两两相加，逢十进一。我们也可以用计算机模拟这一过程。

- 既然要进行运算，我们总得知道字符串的长度吧？怎么获取呢?

用头文件string.h下的strlen函数。

- 要运算，怎么知道某一位的具体数值是几呢?

这个跟ascii码有关了。

一个字符数字的ascii码 - 48（也就是0的ascii码）就是那个数字的值。

- 最后，一定要考虑**进位**

具体的操作请大家参考代码。

参考代码

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
int main( )
{
    int n,i,j,x,y,jw,maxn;
    char ab[510],bc[510];
    int a[510],b[510];
    scanf("%d",&n);
    for(j=1;j<=n;j++)
    {
        jw=0;//记录进位信息
        scanf("%s%s",ab,bc);//使用字符串输入
        x=strlen(ab);//得到字符长度
        y=strlen(bc);
        for(i=0;i<=500;i++)//记得算一组数之前要把数组初始化为0
        {
            a[i]=0;
```

```
            b[i]=0;
        }
        for(i=0;i<x;i++)
            a[i]=ab[x-i-1]-'0';//因为输入的时候是从高位向低位输出，所以要转换一下高低位，
同时把字符转成对应的数字
        for(i=0;i<y;i++)
            b[i]=bc[y-i-1]-'0';
        if(x>y)maxn=x;else maxn=y;//maxn记录最长位数是多少
        for(i=0;i<maxn;i++)
        {
            a[i]=a[i]+b[i]+jw;
            jw=a[i]/10;
            a[i]=a[i]%10;
        }
        if(jw==1) printf("%d",jw);//如果算到最后还有进位的话，那么进位一定是1
        for(i=maxn-1;i>=0;i--)
        {
            printf("%d",a[i]);//从高位到低位输出
        }
        printf("\n");
    }
    return 0;
}
```

H zhn の money

难度	考点
1	循环

先给出答案：$\lfloor log_2 m \rfloor + 1$
第一步，证明 $\lfloor log_2 m \rfloor$ 不可能。假设有 $\lfloor log_2 m \rfloor$ 个钱袋，总共有 $2^{\lfloor log_2 m \rfloor}$ 种组合方式，除去空集剩余 $2^{\lfloor log_2 m \rfloor} - 1$ 种，但是需要表示出 $1 \sim m$ 元钱这 m 种情况。注意到 $2^{\lfloor log_2 m \rfloor} - 1 \leq 2^{log_2 m} - 1 = m - 1 < m$，所以一定不可能表示出所有情况。

第二步，证明 $\lfloor log_2 m \rfloor + 1$ 一定可以。记 $u = \lfloor log_2 m \rfloor$，那么 $2^u \leq m < 2^{u+1}$。令 $r = m - 2^u + 1$，那么 $1 \leq r \leq 2^u$。先用 u 个钱袋分别装入 $2^0, 2^1, 2^2, \cdots, 2^{u-1}$ 元，则剩余 r 元。如果 r 不在 $2^1, 2^2, \cdots, 2^{u-1}$ 中，那么作为第 $u+1$ 个钱袋即可。如果 r 在 $2^1, 2^2, \cdots, 2^{u-1}$ 中，设 $r = 2^t$，那么将原来的 2^t 替换为 2^{t+1}，并将 $r - 1$ 元放入第 $u+1$ 个钱袋。易证这种构造可以组合出所有 $1 \sim m$ 元（二进制位表示即可）。

```
#include<stdio.h>
#include<math.h>
#include<ctype.h>
int main(){
    long long m;
    scanf("%lld",&m);
    printf("%.0f",floor(log2(m))+1);
   return 0;
}
```

I 林士谔算法

难度	考点
2	函数

题目分析

板子：前人已经写好的，可以直接调用使用的代码段。

本题考察函数的正确调用方式，就是看你会不会调，很简单。未来如果有机会学到算法等后续课程的时候，经常会见到前人已经写好的板子，比如多项式、计算几何这种板子。这些板子你可能完全看不懂，而看不懂是非常正常的。这个时候要做的，就是先信任写板子人，假设这个板子实现了它的功能，简单调用测试确认无误后，再写进自己的代码。如果板子测试有误，那就考虑寻找问题并修正，或者重构，或者放弃这个板子。

在算法课的上机禁止联网或携带存储设备，但允许携带打印好的板子。

未来的代码分工合作也是这样。当然，在真正的工程实践中，调用板子要考虑版权问题，原作者是否授权了调用，板子是否开源等等，在商业中一定要考虑这些问题。

示例代码

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

double b[20]; //b是多项式a除以当前迭代二次三项式的商
double c[20]; //c是多项式b乘以x平方再除以当前迭代二次三项式的商
double p;     //p是待求的一次项
double q;     //q是待求的常数项

void Shie(double a[], int n)//a是原始的多项式，n是多项式次数
{
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
    p = 0;
    q = 0;
    double dp = 1;
    double dq = 1;
    while (dp > 1e-12 || dp < -1e-12 || dq > 1e-12 || dq < -1e-12)
    {
        double p0 = p;
        double q0 = q;
        b[n - 2] = a[n];
        c[n - 2] = b[n - 2];
        b[n - 3] = a[n - 1] - p0 * b[n - 2];
        c[n - 3] = b[n - 3] - p0 * b[n - 2];
        int j;
        for (j = n - 4; j >= 0; j--)
        {
            b[j] = a[j + 2] - p0 * b[j + 1] - q0 * b[j + 2];
            c[j] = b[j] - p0 * c[j + 1] - q0 * c[j + 2];
        }
        double r = a[1] - p0 * b[0] - q0 * b[1];
        double s = a[0] - q0 * b[0];
        double rp = c[1];
        double sp = b[0] - q0 * c[2];
```

```
        double rq = c[0];
        double sq = -q0 * c[1];
        dp = (rp * s - r * sp) / (rp * sq - rq * sp);
        dq = (r * sq - rq * s) / (rp * sq - rq * sp);
        p += dp;
        q += dq;
    }
}

double a[20];

int main()
{
    int n;
    while(scanf("%d",&n)!=EOF)
    {
        int i;
        for(i=n;i>=0;i--)
        {
            scanf("%lf",&a[i]);
        }
        Shie(a,n);
        printf("1.000000 %.6lf %.6lf\n",p,q);
    }
}
```

J 测评机没有sqrt（）！

难度	考点
3	二分法

题目分析

本题有若干种做法，有的做法奇奇怪怪，了解即可。

示例代码

1. 二分法。二分法解方程在C5会讲。

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>

#define eps 1e-8

int main()
{
    double a,min=0,max=50,b=25;
    scanf("%lf",&a);
    if(fabs(a)<eps)
```

```
    {
        printf("0.00000000");
        return 0;
    }
    while(fabs(b*b-a)>eps)
    {
        if(b*b>a)
        {
            max=b;
            b=(min+max)/2;
        }
        else if(b*b<a)
        {
            min=b;
            b=(min+max)/2;
        }
    }
    printf("%.8lf",b);
    return 0;
}
```

2. 标准的牛顿法。

```
/*
 Author: 华佳瑞(50215)
 Result: AC Submission_id: 4232581
 Created at: Sat Mar 26 2022 11:01:36 GMT+0800 (China Standard Time)
 Problem: 5425  Time: 24    Memory: 1676
*/

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#define eps 1e-9
int main ()
{
    double a,t;
    scanf("%lf",&a);
    t=a/2;
    while(a-t*t>eps||a-t*t<-eps)
    {
        t=(a/t+t)/2;
    }
    printf("%.8lf",t);
    return 0;
}
```

3. 在Hint 2中给出的做法。

这种做法等价于给出了牛顿法的一个很近的迭代起点，可以减少迭代次数。

```
#include<stdio.h>

int main()
{
    double x;
    scanf("%lf",&x);
    if(x==0)
    {
        printf("%.8lf\n",0);
        return 0;
    }
    double xhalf = 0.5f * x; //32位浮点数
    long long i = *(long long*)&x; //按位强制按32位整数理解
    i = 0x5FE6EB50C0000000LL - (i >> 1); //整数位运算
    x = *(double*)&i; //按位强制按32位浮点数理解
    x = x*(1.5f-(xhalf*x*x)); //再来一次牛顿法
    x = x*(1.5f-(xhalf*x*x)); //再来一次牛顿法
    x = x*(1.5f-(xhalf*x*x)); //再来一次牛顿法
    printf("%.8f\n",1/x);
}
```

4. 其他的一些计算办法。

这些办法是呱呱泡蛙没想到的。呱呱泡蛙只想到了数学分析中常见的公式$\sqrt{x}=e^{\frac{1}{2}\ln x}$，便把它禁掉了。可是，呱呱泡蛙没有想到其他简单办法，例如借助二倍角公式等也能做。参见某同学的代码：

```
/*
 Author: 王政阳(50452)
 Result: AC Submission_id: 4232630
 Created at: Sat Mar 26 2022 11:12:26 GMT+0800 (China Standard Time)
 Problem: 5425  Time: 70    Memory: 2416
*/

#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
#include<math.h>
//第三次练习赛
//测评机没有...
//不会真要用牛顿法写吧，一种猜想，有的计算器计算...本身就是用的牛顿法
//用cos和acos就行了
int main()
{
    double n,m;
    scanf("%lf",&n);
    n=n/1000;
    m=cos(acos(2*n-1)/2);
    printf("%.8lf",31.622776601684*m);
}
```

上述这样的做法真的是让呱呱泡蛙大吃一惊呢。

5. 借助宏的高级做法。

这个做法用到了本课不会涉及到的冷门知识点，然而可以绕过通常的字符串匹配代码封禁。C语言当中宏定义的高级玩法可以借助双井号连接，如下面的代码：

```
/*
 Author: 呱呱泡蛙(21563)
 Result: AC Submission_id: 4232756
 Created at: Sat Mar 26 2022 11:55:47 GMT+0800 (China Standard Time)
 Problem: 5425  Time: 13    Memory: 2160
*/
#include<stdio.h>
#include<math.h>
#define F(sth) sq##sth
int main()
{
    double x;
    scanf("%lf",&x);
    printf("%.8lf",F(rt(x)));
}
```

我们的评测机制禁掉函数使用的是代码文本字符串匹配，故只要在文本中不出现该字符串即可。去年的题目《4257 小迷弟积不出分》（该题目不公开，可以参见Gitee）也可以借助这个办法，以及被禁掉的erf函数水过去。

清华大学的评测机是直接在编译器函数库里将头文件移除，这种办法就行不通了。

6. 其他方法

```
/*
 Author: 史海容(50182)
 Result: AC Submission_id: 4232299
 Created at: Sat Mar 26 2022 10:06:46 GMT+0800 (China Standard Time)
 Problem: 5425  Time: 49    Memory: 1696
*/

#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>

#define MAX( x, y ) ( ((x) > (y)) ? (x) : (y) )
#define MIN( x, y ) ( ((x) < (y)) ? (x) : (y) )
#define UPCASE( c ) ( ((c) >= 'a' && (c) <= 'z') ? ((c) - 0x20) : (c) )
#define DOWNCASE( c ) ( ((c) >= 'A' && (c) <= 'Z') ? ((c) + 0x20) : (c) )
double sum(double a, int b[], double c[], int i, int j){
    int k;
    double sum1 = 0;
    double sum2 = 0;

    for(k = 0; k < j ; k++){
        sum1 += c[k] * b[k];
        sum2 += c[k] * b[k];
    }
```

```
    sum1 += c[j] * i;
    sum2 += c[j] * (i + 1);
    if((sum1 * sum1) <= a && (sum2 * sum2) > a){
        return 1;
    }else{
        return 0;
    }
}


int main()
{
    double a;
    int i, j;
    scanf("%lf", &a);
    double c[10];
    c[0] = 1;


    for(i = 1; i < 10; i++){
        c[i] = 0.1 * c[i - 1];
    }


    int b[10] = {0};
    for(i = 0; i < 40; i++){
        if((i * i) <= a&&((i + 1) * (i + 1)) > a){
            b[0] = i;
        }
    }
    for(j = 1; j < 10; j++){//跑每一位
        for(i = 0; i < 10; i++){//判断每一位数字
            if(sum(a, b, c, i, j) == 1){
                b[j] = i;
                break;
            }
        }
    }
    if(b[9] > 5){
        b[8]++;
    }

    for(i = 8; i >= 1; i--){
        if (b[i] == 10) {
            b[i - 1]++;
            b[i] -= 10;
        }
    }
    printf("%d.", b[0]);
    for(i = 1; i < 9; i++){
        printf("%d", b[i]);
    }
    return 0;
}
```

K 荷家军进攻汉诺塔

难度	考点
3	递归

题目分析

本题是PPT中的原题，解题核心是一个递归函数，将整个搬动过程分为三个部分，以将柱从 `from` 经过 `via` 移动到 `to` 为例：

1. 将上面n-1个盘子从 `from` 经由 `to` 搬动到 `via` 。
2. 将最下面的n号盘从 `from` 搬动到 `to` 。
3. 将第一步搬动到 `via` 的塔经由 `from` 搬到 `to` 。
   本题只要注意输入语句的写法即可,由于柱的标号是一个大写字母，所以使用 `%c` 输入，使用 `char` 型变量存储即可，无需使用字符串。

示例代码

```
#include <stdio.h>

void hanoi(int n, char a, char b, char c);
// 把n个盘.从柱. a 通过 b 挪到 c 上
void move(int n, char a, char c); // 把第n号盘.从柱.a挪到c上

int main()
{
    int num;
    char a, b, c;
    scanf("%d %c %c %c", &num, &a, &b, &c);
    hanoi(num, a, b, c);
    return 0;
}

void move(int n, char from, char to)
{
    printf("move %d from %c to %c\n", n, from, to);
}

void hanoi(int n, char from, char via, char to)
{
    if (n == 1)
    {
        move(n, from, to);
        return;
    }
    hanoi(n - 1, from, to, via); // 把n-1个盘.从from通过
    move(n, from, to);
    hanoi(n - 1, via, from, to);
}
```