C4-2021级航类-第四次上机题解

A <math.h>

难度	考点
1	数学库函数

题目分析

学会调用库函数，直接计算即可

参考代码

```
#include<stdio.h>
#include<math.h>
int main()
{
    double x,y,z,s;
    scanf("%lf %lf %lf",&x,&y,&z);
    s=(pow(acos(sin(x)),log(1+fabs(sinh(y)))))/(2+cos(z));
    printf("%.2lf",s);
    return 0;
}
```

B 宋老师的名次预测2.0

难度	考点
1	数组、循环

题目分析

本题与C1 `甲贺忍蛙的名次预测` 的不同之处在于人数的增加，所以不能再采取C1中使用8个变量的做法，那样会使得程序代码非常繁琐；在这里，我们利用数组，将30人的名次依次存进一个数组当中，然后遍历数组的前30个元素，判断是否猜对并进行输出.

当然，你在本周学了函数之后，可以把输入、判断、输出的过程封装成一个函数guess()，使得程序更加清晰明了。

参考代码

```
#include <stdio.h>

void guess(void) //第一个void的意思是此函数没有返回值
{                //第二个void的意思是此函数没有参数
    int num = 0, tmp;
    for (int i = 1; i <= 30; i++)
    {
        scanf("%d", &tmp);
        num += (tmp == i);
    }

    if (num > 20)
        puts("****");
```

```
    else if (num > 10)
        puts("***");
    else if (num > 5)
        puts("**");
    else
        puts("#");
}
int main()
{

    int n;
    scanf("%d", &n);
    while (n--)
    {
        guess(); //调用函数
    }
    return 0;
}
```

C 滴滴滴递归

难度	考点
2	函数、递归

题目分析

本题给出了递归函数的公式，类似于阿克曼函数，直接代入公式调用就好了；

示例代码

```
#include <stdio.h>
#include <math.h>
int digui(int a, int b);

int main()
{
    int a, b;
    while (~scanf("%d%d", &a, &b))
        printf("%d\n", digui(a, b));

    return 0;
}

int digui(int a, int b)
{
```

```
    if (b == 0)
        return 1;
    else if (b % 2 == 0)
        return digui(a * a, b / 2);
    else
        return digui(a * a, b / 2) * a;
}
```

题外分析

通过简单的观察，题目给出的函数 $fun(a,b)$ 其实计算的就是 a^b （快速幂算法）。推导过程如下：

$$a^b=\begin{cases}1, & \text{if } \ b=0;\\ \\ a^{2\times\frac{b}{2}}=(a^2)^{\frac{b}{2}}, & \text{else if } \ b\%2=0;\\ \\ (a^2)^{\frac{b}{2}}\times a, & \text{else.}\end{cases}$$

这和递归函数给出的公式是一样的，注意 $\frac{b}{2}$ 是整型相除即可。

所以我们的代码也可以写成下面这样

```
#include <stdio.h>
int a, b;
int my_pow(int a, int b) //计算a^b
{
    int ans = 1;
    for (int i = 0; i < b; i++)
        ans *= a;
    return ans;
}
int main()
{
    while (~scanf("%d%d", &a, &b))
        printf("%d\n", my_pow(a, b));

    return 0;
}
```

D 求差向量的无穷范数

难度	考点
3	函数

问题分析

本题的难点在于如何实现函数的调用，传递适当的参数来实现题目要求的计算。具体解释见代码注释。

参考代码

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double x[10], y[10], z[10];         //空间向量的三个维度
double solve(int a, int b)          // double代表了返回值的类型，我们要计算返回的是double类型的浮点数
{                                   // a与b分别对应向量a和向量b
    double xx = fabs(x[a] - x[b]); //求差向量
    double yy = fabs(y[a] - y[b]);
    double zz = fabs(z[a] - z[b]);
    if (xx < yy)
    {
        xx = yy;
    }
    if (xx < zz)
    {
        xx = zz;
    }
    return xx; //返回无穷范数
}
int main()
{
    int i;
    for (i = 1; i <= 5; i++)
    {
        scanf("%lf%lf%lf", &x[i], &y[i], &z[i]);
    }
    printf("%.3lf\n", solve(5, 4)); // solve(5,4)代表对向量5和向量4求差向量的无穷范数后返回的答案，是double类型浮点数
    printf("%.3lf\n", solve(3, 1));
    printf("%.3lf\n", solve(1, 5));
    printf("%.3lf\n", solve(4, 2));
    printf("%.3lf\n", solve(3, 5));
    printf("%.3lf\n", solve(2, 1));
    printf("%.3lf\n", solve(3, 4));
    printf("%.3lf\n", solve(1, 4));
    return 0;
}
```

E 多项式相加2022

难度	考点
3	数组

解题思路

对于本道题，如果多项式的每一项指数部分都在 $[0, 10^5]$ 这样的范围内，则我们只需要设置一个数组 `int coe[100005]`，每次读入的时候以指数部分作为数组下标，将系数记录进 `coe` 数组中即可完成任务。但本题中指数范围较大（`int`范围，甚至可以是负数），直接采用上述方法进行计算将会导致数组越界，因此需要考虑其他方法。

注意到题目中的 $f(x), g(x)$ 都是以严格降序给出的，因此可以考虑这样一种做法：

1. 用两个变量 p, q 记录当前正在处理 $f(x), g(x)$ 从大到小的第 i 项。一开始时 $p = q = 1$ 。
2. 比较第 p 项和第 q 项的指数部分：

- 若指数部分相同，说明这两项的系数部分在结果中应当相加，并将 p, q 都向后移动一位；
- 若 p 对应的指数部分较大，说明只有 $f(x)$ 在结果的这一项中出现，并将 p 向后移动一位；
- 若 q 对应的指数部分较大，说明只有 $g(x)$ 在结果的这一项中出现，并将 q 向后移动一位；

当 p, q 将 $f(x), g(x)$ 的每一项都计算之后，最后结果的多项式也被计算出来了。可以发现，通过该方法得到的多项式，其指数部分也是严格递减的。由于 p, q 只会移动最多 $n + m$ 次，因此总循环次数不超过 $n + m$ 次，可以在题目要求的时间范围内完成计算。

另外需要注意的是，本题的系数可能超出 `int` 范围，因此需要采用 `long long` 进行存储。

示例代码1

```
#include <stdio.h>

#define N (200000 + 5)

long long a[N], b[N];
long long s[N], t[N];

int main()
{
    int n, m;
    int p = 1, q = 1;
    int i;

    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++)
        scanf("%lld%lld", &a[i], &s[i]);
    for (i = 1; i <= m; i++)
        scanf("%lld%lld", &b[i], &t[i]);

    while (p <= n || q <= m)
    {
        if ((p <= n && q <= m && s[p] > t[q]) || q > m)
```

```
        {
            // 只计算 p 的情况：p 对应的指数较小，或另一个数组已经扫描完了
            printf("%lld %lld ", a[p], s[p]);
            p++;
        }
        else if ((p <= n && q <= m && s[p] < t[q]) || p > n)
        {
            // 只计算 q 的情况：q 对应的指数较小，或另一个数组已经扫描完了
            printf("%lld %lld ", b[q], t[q]);
            q++;
        }
        else
        {
            // 合并系数的情况：能来到这个分支，一定是 pq 都没有扫描完，且 f 的第 p 项和 g 的第 y 项
系数相等
            if (a[p] + b[q] != 0)
                printf("%lld %lld ", a[p] + b[q], t[q]);
            p++, q++;
        }
    }

    return 0;
}
```

示例代码2

```
#include <stdio.h>
long long poly_f_coe[100010];
long long poly_f_pow[100010];
long long poly_g_coe[100010];
long long poly_g_pow[100010];
long long poly_ans_coe[200020];
long long poly_ans_pow[200020];
int main()
{
    int n, m, i;
    int current_i = 0;
    int current_j = 0;
    int current_k = 0;
    scanf("%d%d", &n, &m);
    for (i = 0; i < n; i++)
    {
        scanf("%lld%lld", &poly_f_coe[i], &poly_f_pow[i]);
    }
    for (i = 0; i < m; i++)
    {
        scanf("%lld%lld", &poly_g_coe[i], &poly_g_pow[i]);
    }
```

```
    while (current_i < n || current_j < m)
    {
        if (current_i < n)
        {
            if (current_j < m)
            {
                if (poly_f_pow[current_i] > poly_g_pow[current_j])
                {
                    poly_ans_coe[current_k] = poly_f_coe[current_i];
                    poly_ans_pow[current_k] = poly_f_pow[current_i];
                    current_k++;
                    current_i++;
                }
                else if (poly_f_pow[current_i] < poly_g_pow[current_j])
                {
                    poly_ans_coe[current_k] = poly_g_coe[current_j];
                    poly_ans_pow[current_k] = poly_g_pow[current_j];
                    current_k++;
                    current_j++;
                }
                else
                {
                    poly_ans_coe[current_k] = poly_f_coe[current_i] +
                                              poly_g_coe[current_j];
                    poly_ans_pow[current_k] = poly_f_pow[current_i];
                    current_k++;
                    current_i++;
                    current_j++;
                }
            }
            else
            {
                poly_ans_coe[current_k] = poly_f_coe[current_i];
                poly_ans_pow[current_k] = poly_f_pow[current_i];
                current_k++;
                current_i++;
            }
        }
        else
        {
            poly_ans_coe[current_k] = poly_g_coe[current_j];
            poly_ans_pow[current_k] = poly_g_pow[current_j];
            current_k++;
            current_j++;
        }
    }
    for (i = 0; i < current_k; i++)
    {
        if (poly_ans_coe[i] != 0)
```

```
        {
            if (i != current_k - 1)
            {
                printf("%d %d ", poly_ans_coe[i], poly_ans_pow[i]);
            }
            else
            {
                printf("%d %d\n", poly_ans_coe[i], poly_ans_pow[i]);
            }
        }
    }
    return 0;
}
```

F 简单的基物数据处理

难度	考点
2	数组、循环

问题分析

本题主要考察数组的运用以及如何实现特定的数学公式。

观察可知u_a的计算公式中涉及avg，所以先计算avg再计算标准差，最后得到u_a。

参考代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define ll long long
double x[1010];
int main()
{
    int n=0;
    double avg=0,ua=0;
    while(scanf("%lf",&x[n])!=EOF) //输入xi，同时计算累加和
    {
        avg+=x[n];
        n++;
    }
    avg/=n; //得到平均值avg
    for(int i=0;i<n;i++) ua+=(x[i]-avg)*(x[i]-avg); //计算标准差
    ua=sqrt(ua/n/(n-1)); //计算ua值
```

```
    printf("%.4lf %.4lf\n",avg,ua);
    // system("pause");
    return 0;
}
```

G 来个神奇日历

题目分析

本题是日期问题的一个典型问题。

采用的方法为模拟，从当年的第1天模拟到当年的第d天，只需要将每月对应的天数求出来即可。为此使用$getdays$函数获取某年某月一共有多少天

注意对2月进行特殊判断（因为闰年的2月是29年，其余年份的2月是28天）

示例代码1

```
#include <stdio.h>
int months[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int isleap(int y)
{
    if ((y % 400 == 0) || (y % 4 == 0 && y % 100 != 0))
    {
        return 1;
    }
    return 0;
}
int getdays(int y, int m)
{
    if (m != 2)
    {
        return months[m];
    }
    return months[m] + isleap(y);
}
int main()
{
    int n;
    scanf("%d", &n);
    while (n--)
    {
        int y, d;
        scanf("%d%d", &y, &d);
        int nowday = 0, find = 0;
        int i, j;
        for (i = 1; i <= 12; i++)
        {
            int day = getdays(y, i);
```

```
            for (j = 1; j <= day; j++)
            {
                nowday++;
                if (nowday == d)
                {
                    printf("%d %d\n", i, j);
                    break;
                }
            }
            if (find == 1)
            {
                break;
            }
        }
    }
    return 0;
}
```

示例代码2

```
#include <stdio.h>
int months[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int isleap(int y)
{
    if ((y % 400 == 0) || (y % 4 == 0 && y % 100 != 0))
    {
        return 1;
    }
    return 0;
}
int main()
{
    int n,y,d;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%d%d",&y,&d);

        if(isleap(y))months[2]=29;
        else months[2]=28;

        int mon=1;
        while(d>months[mon])
        {
            d-=months[mon];
            mon++;
        }
        printf("%d %d\n",mon,d);
```

```
    }
}
```

H 全排列组合

难度	考点
4	递归

问题分析

本题主要考察递归函数的使用。以数列 $\{4,5,6,7\}$ 为例，我们可以发现，当要对该数列进行排列组合时，其实可以看成确定好第一个元素后，对剩下的元素再进行一次全排列组合，即第一次交换后有四种情况：

$$4,\{5,6,7\}$$

$$5,\{4,6,7\}$$

$$6,\{4,5,7\}$$

$$7,\{4,5,6\}$$

在每一次情况下，分别对大括号内的数列再次进行全排列组合，依次类推。不妨以第一种情况 $4,\{5,6,7\}$ 继续推演，此时，应当确定好第二个元素，则有三种情况：

$$4,5,\{6,7\}$$

$$4,6,\{5,7\}$$

$$4,7,\{5,6\}$$

再往下依次确定元素，直至确定好最后一个元素后，我们就可以打印这个确定好的数列并返回，此时，这个返回的行为便被称作**递归出口**，任何一个递归行为都应当有至少一个递归出口，这一点是十分重要的。

值得注意的是，本题采用的是字典序排序，如果仅仅靠简单交换两不同位置的元素来更改第一个元素，那么最后生成的数列将不满足字典序，同样以 $\{4,5,6,7\}$ 为例，采用交换方式得到的第一次交换情况为：

$$4,\{5,6,7\}$$

$$5,\{4,6,7\}$$

$$6,\{5,4,7\}$$

$$7,\{5,6,4\}$$

和前文所述的交换数列对比即可发现有明显不同，这点还请同学们注意。

参考代码

```c
#include <stdio.h>

void swap(int a[], int p, int q) //采用字典序交换
{
    int i, temp;
    if (p < q)
    {
        temp = a[q];
        for (i = q; i > p; i--)
        {
            a[i] = a[i - 1];
        }
        a[p] = temp;
    }
    else if (p > q)
    {
        temp = a[q];
        for (i = q; i < p; i++)
        {
            a[i] = a[i + 1];
        }
        a[p] = temp;
    }
}

void perm(int a[], int p, int q) //全排列函数
{
    int i;
    if (p == q)
    {
        for (i = 0; i <= q; i++)
        {
            printf("%d ", a[i]);
        }
        printf("\n");
        return;
    }
    for (i = p; i <= q; i++)
    {
        swap(a, p, i);
        perm(a, p + 1, q);
        swap(a, i, p);
    }
}
int main()
{
    int num[55];
```

```
    int i, n, m;
    scanf("%d%d", &n, &m);
    for (i = 0; i < m - n + 1; i++)
    {
        num[i] = i + n;
    }
    perm(num, 0, m - n);
    return 0;
}
```

I 大师问题

经典递归问题。直接上代码：

参考代码

```
#include<stdio.h>

int arr[25];//arr记录每一行大师的坐标
int all;
int total;//合法数

int isPlaceOK(int n, int c)//检查位置是否可以放，c是将要放置的位置
{
    int i;
    for(i=1; i<=n-1; ++i)
    {
        if (arr[i] == c || arr[i]-i == c-n || arr[i]+i == c+n)//a[i] == c放在同一列；a[i]
-+ i = c -+ n 在对角线上
        {
            return 0;
        }
    }
    return 1;
}

void addMaster(int n)//n为当前行
{
    if(n>all)
    {
        total++;
        return;
    }
    int i;
    for (i=1; i<=all; ++i)//i从第1列到第all列遍历
    {
        if (isPlaceOK(n, i))
```

```
            {
                arr[n]=i;//如果可以放置，就把大师放在第n行第i列
                addMaster(n+1);
            }
        }
}

int main()
{
    total=0;
    scanf("%d",&all);
    addMaster(1);//从第一行开始放置
    printf("%d\n",total);
}
```

J dwy逛优狗超市

难度	考点
3	GCD及函数调用

问题分析

本题由多元线性丢范图方程的求解发展而来，即求多元方程是否有整数解。本题设置了找零，允许负整数解存在，而由定义可知方程$a_1x_1+a_2x_2+\cdots+a_nx_n=b$当且仅当$\gcd(a_1,a_2,\cdots a_n)$整除$b$时存在整数解（此处具体证明可以参考《初等数论及其应用》线性丢番图方程部分，不赘述）。因此本题实质上是最大公约数算法的应用，即：货币面值为系数，总金额为方程中的b。

在具体代码实现中，可以将求最大公约数的函数封装多次调用，求得所有货币面值的最大公约数，再判断其能否整除总金额，即可求解。

参考代码

```
#include <stdio.h>

int gcd(int a, int b);
int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        int n, k;
        scanf("%d%d", &n, &k);
        int ans = 0;
        while (n--)
```

```c
        {
            int a;
            scanf("%d", &a);
            ans = gcd(ans, a);
        }
        if (k % ans)
        {
            printf("Nyet.\n");
        }
        else
        {
            printf("Da!\n");
        }
    }
    return 0;
}

int gcd(int a, int b)
{
    if (b > a)
    {
        int tmp = b;
        b = a;
        a = tmp;
    }
    while (b != 0)
    {
        int tmp = b;
        b = a % b;
        a = tmp;
    }
    return a;
}
```