写在最前的tips

1. 保证一定的做题量以获得编程思维是有必要的。在常见的OJ上可以在课程难度范围内适量做一些题（如洛谷的试炼场），保证至少编程这块到期末手是熟的。
2. 仔细研读历次上机/练习赛的题解，里面有很多隐藏的知识点复习/添加。

A. 翘课的HugeGun_

解题思路

本题的题意为：给定直角三角形的面积 S 和斜边长 l，求解其两条直角边的长度。

设两条直角边长度分别为 a 和 b，很容易得出等式 $(a-b)^2=l^2-4S$，$(a+b)^2=l^2+4S$，方程是否有解的条件在于 l^2-4S 是否非负，判断有解后直接求解即可。

AC代码

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

int main()
{
    long long l,s;
    int n;
    scanf("%d",&n);
    while(n--)
    {
        scanf("%lld%lld",&l,&s);
        if(l*l<s*4) puts("hhhg");
        else
        {
            double a=sqrt(l*l-s*4),b=sqrt(l*l+s*4);
            double x=(a+b)/2,y=(b-a)/2;
            printf("%.2f %.2f\n",x<y?x:y,x<y?y:x);
        }
    }
    return 0;
}
```

B. bganteneq

解题思路

我们只需要判断$x(int类型)$的二进制表示下的第y位是不是1即可。

我们可以通过使用位运算轻松的达到这个目的。

AC代码

```
#include <stdio.h>
int main()
{
    int t, x, y;
    scanf("%d", &t);
    while (t--) {
        scanf("%d%d", &x, &y);
        if (x & (1 << y))
            printf("branch\n");
        else
            printf("in-order\n");
    }
    return 0;
}
```

举个例子，假设x的二进制表示为`11010`，$y=1$，那么$1<<y=00010$

如果x的第y位为1，那么$x\&(1<<y)=(1<<y)>0$，否则$x\&(1<<y)=0$

我们也可以使用`>>`运算符，将$x>>y$后得到的数的最后一位即是x的第y位：

```
#include <stdio.h>
int main()
{
    int t, x, y;
    scanf("%d", &t);
    while (t--) {
        scanf("%d%d", &x, &y);
        if ((x >> y) & 1)
            printf("branch\n");
        else
            printf("in-order\n");
    }
    return 0;
}
```

位运算是一个很实用的东西，大家请尽量掌握。

C. 表达式计算

解题思路

注意到题目只含有+, -, *, /四种运算，且不包含括号，故从左往右扫描表达式，先将*和/的结果先算出来，再做+或-，最终合并得出结果。

AC代码

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define N 1005
#define max(a,b) ((a)>(b)?(a):(b))
#define min(a,b) ((a)<(b)?(a):(b))
char a[N];
int x[N];

int main()
{
    scanf("%s", a + 1);
    a[0] = '+';
    int len = strlen(a);
    for (int i = 0; i < len; i++) {
        if (a[i] == '+' || a[i] == '-') // 出现+ -就标记一下
            x[i] = 1;
    }
    int temp = 0, preop, pre = 1;
    int t = 0, i = 0, ans = 0;
    while (i < len) {
        if (x[i] == 1) { // 当前是+或者-
            if (pre == 1) // 上一个运算是+
                ans += temp;
            else // 上一个运算是-
                ans -= temp;
            if (a[i] == '+') // 更新pre, temp和preop
                pre = 1;
            else
                pre = 2;
            temp = 0;
            preop = 0;
            i++;
        } else {
            if (a[i] == '\'') { // 是字符常量
                i++;
                t = a[i];
                i++; i++;
            } else { // 是无符号整数
                t = 0;
                while (isdigit(a[i])) {
                    t *= 10;
                    t += a[i] - '0';
                    i++;
                }
```

```
            }
            if (preop == 0) { // 前面没有运算
                temp = t;
                if (a[i] == '*') {
                    preop = 1;
                    i++;
                } else if(a[i] == '/') {
                    preop = 2;
                    i++;
                }
            } else { // 前面有运算
                if (preop == 1)
                    temp *= t;
                else if (preop == 2)
                    temp /= t;
                if (a[i] == '*') {
                    preop = 1;
                    i++;
                } else if (a[i] == '/') {
                    preop = 2;
                    i++;
                }
            }
        }
    }
    if (pre == 1) // 合并得到最终结果
        ans += temp;
    else
        ans -= temp;
    printf("%d\n", ans);
    return 0;
}
```

D. 第K大全排列

解题思路

乍一看到这道题，可能很多人都想通过递归来完成这道题，但是发现在做搜索的时候，时间很容易就会炸掉导致TLE（虽然这个题n的上限是8不会，但万一遇到大一点的就有可能GG）。

这个题的大小排列是很有规律的，因此我们不妨按照这个规律，用一个O（1）的算法直接来获得答案。全排列，就是算一下阶乘，我们先算出来各个从2到n - 1的阶乘，然后再用对k和各次除法运算出来的余数，一一对应做除法和求余运算就可以获得答案。

AC代码

```
#include <stdio.h>
int main()
{
    int p[10], getin[8];
    int l, i, j, n, k, temp;

    p[1] = 1;
    for(i = 2; i < 8; i++)
    {
        p[i] = i * p[i - 1];
    }

    scanf("%d%d", &n, &k);
    for(i = 0; i < n; i++) scanf("%d", &getin[i]);
    for(i = 0; i < n; i++)
    for(j = 0; j < n - i - 1; j++)
    {
        if(getin[j] < getin[j + 1])
        {
            temp = getin[j];
            getin[j] = getin[j + 1];
            getin[j + 1] = temp;
        }
    }

    for(i = 1; i <= n; i++)
    {
        j = k / p[n - i] + (k % p[n - i] != 0);
        //这里进行分析发现第1大到第（n-1)!大此类的全排列有相同的首数字，故向上取整
        if(k != 0)
        {
            printf("%d ", getin[j - 1]);
            getin[j - 1] = 0;
            k %= p[n - i];
        }
        else
        {
            //需要特殊判断余数为0的情况
            printf("%d ", getin[n - i]);
            getin[n - i] = 0;
        }
```

```
        //每次是在剩余的数里找新的第k大全排列，需要重新排序
        for(l = 0; l < n; l++)
        for(j = 0; j < n - l - 1; j++)
        {
            if(getin[j] < getin[j + 1])
            {
                temp = getin[j];
                getin[j] = getin[j + 1];
                getin[j + 1] = temp;
            }
        }
    }

}
```

E. 爱吃猪脚的猪脚的Excel

解题思路

本题考察了排序。因为数据范围不大，所以可以使用简单的冒泡/选择/插入排序。当然也可以使用库函数`qsort`。面对学生这样独立的个体，可以考虑将其封装成结构体，这样也可以减少高维指针的使用。尤其是在考试中，如果你不是很擅长指针，结构体是一个更好的选择（两者的代码量也几乎没有差别）。

AC代码（结构体排序）

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

#define MK 4
#define ML 55
#define MN 333
#define cmpfp_cast(fp) (int(*)(const void *, const void *))(fp)


typedef struct {char str[MK][ML];} Stu;
Stu stu[MN];
int pri[MK];

int cmp(const Stu *p, const Stu *q)
{
    int i, lexcmp = 0;
    for (i=0; i<MK && lexcmp==0; ++i)
        lexcmp = strcmp(p->str[pri[i]], q->str[pri[i]]);
    return lexcmp;
}

int main()
{
    int n, i, k;

    scanf("%d", &n);
    for (i=1; i<=n; ++i)
        for (k=0; k<MK; ++k)
            scanf("%s", stu[i].str[k]);
    for (k=0; k<MK; ++k)
        scanf("%d", pri+k), --pri[k];

    qsort(stu+1, n, sizeof(*stu), cmpfp_cast(cmp));
    for (i=1; i<=n; ++i)
        if (cmp(stu+i-1, stu+i) != 0)
            for (k=0; k<MK; ++k)
                printf("%s%c", stu[i].str[k], " \n"[k==MK-1]);

    return 0;
}
```

AC代码（高维指针排序）

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

#define MK 4
#define ML 55
#define MN 333
#define cmpfp_cast(fp) (int(*)(const void *, const void *))(fp)


char stu[MN][MK][ML];
int pri[MK];

int cmp(const char (*p)[MK][ML], const char (*q)[MK][ML])
{
    int i, lexcmp = 0;
    for (i=0; i<MK && lexcmp==0; ++i)
        lexcmp = strcmp(p[0][pri[i]], q[0][pri[i]]);
    return lexcmp;
}

int main()
{
    int n, i, k;

    scanf("%d", &n);
    for (i=1; i<=n; ++i)
        for (k=0; k<MK; ++k)
            scanf("%s", stu[i][k]);
    for (k=0; k<MK; ++k)
        scanf("%d", pri+k), --pri[k];

    qsort(stu+1, n, sizeof(*stu), cmpfp_cast(cmp));
    for (i=1; i<=n; ++i)
        if (cmp(stu+i-1, stu+i) != 0)
            for (k=0; k<MK; ++k)
                printf("%s%c", stu[i][k], " \n"[k==MK-1]);

    return 0;
}
```

F. 颠倒ABC

解题思路

如题目描述中所强调，在OJ通过本题不代表真实通过本题。

本题考察点为排序的稳定性。

解法1：因为不仅要通过哈希值比较，还要通过出现的前后比较，也即双关键词排序，所以可以使用快速排序通过。

AC代码1

```
#include<stdio.h>
#include<stdlib.h>
int hash[256];
typedef struct{
    char a[1030];
    int id;
}st;
st p[2010];
int cmps(char *a,char *b){
    int i;
    for(i=0;a[i]&&b[i];i++){
        if(hash[a[i]]>hash[b[i]])return -1;
        if(hash[a[i]]<hash[b[i]])return 1;
    }
    return a[i]?-1:b[i]?1:0;
}
int cmp(const void *a,const void*b){
    st x=*(st*)a,y=*(st*)b;
    if(cmps(x.a,y.a)>0)return -1;
    if(cmps(x.a,y.a)<0)return 1;
    return x.id-y.id;
}
int main(){
    int i,n;
    for(i=0;i<26;i++)scanf("%d",&hash['a'+i]);
    for(i=0;i<26;i++)scanf("%d",&hash['A'+i]);
    scanf("%d",&n);
    for(i=0;i<n;i++)scanf("%s",p[i].a),p[i].id=i;
    qsort(p,n,sizeof(st),cmp);
    for(i=0;i<n;i++)printf("%s\n",p[i].a);
    return 0;
}
```

解法2：由于第二维是出现的前后位置，所以利用稳定排序的性质可以直接对第一维进行排序（如冒泡排序），此时能够自动保证第二维是按顺序排好的。

请务必注意冒泡排序和选择排序的区别，写法很像但是冒泡是稳定的，选择是不稳定的。

快速排序是不稳定的，但在OJ环境下没有出出卡掉快速排序的数据，所以在这一维普通的快速排序也会显示Accepted。有兴趣的同学可以搜索快速排序的执行过程，并分析各种排序为何是稳定/不稳定的。

AC代码2

```
#include<stdio.h>
#include<stdlib.h>
char word[505][1050],tmp[1050];
int hash[256];
int cmps(char *a,char *b){
    int i;
    for(i=0;a[i]&&b[i];i++){
        if(hash[a[i]]>hash[b[i]])return -1;
        if(hash[a[i]]<hash[b[i]])return 1;
    }
    return a[i]?-1:b[i]?1:0;
}
int main(){
    int n,i,j;
    for(i=0;i<26;i++)scanf("%d",&hash['a'+i]);
    for(i=0;i<26;i++)scanf("%d",&hash['A'+i]);
    scanf("%d",&n);
    for (i=0;i<n;i++)scanf("%s",word[i]);
    for (i=0;i<n;i++){
        for(j=0;j<n-i-1;j++){
            if (cmps(word[j+1],word[j])==1){
                strcpy(tmp,word[j]);
                strcpy(word[j],word[j+1]);
                strcpy(word[j+1],tmp);
            }
        }
    }
    for(i=0;i<n;i++)printf("%s\n",word[i]);
    return 0;
}
```

G. 斐波那契那契斐波那契

解题思路

由于输出的字符串长度很短，因此可以利用递归将需要输出的字符串在前$max(a,b)$中找到。我们只需要实现函数$f(n,k,l)$表示在第n个字符串中输出起点为k，长度为l的子串，然后利用拼接的性质递归寻找。

AC代码

```
#include<stdio.h>
#include<string.h>
int a,b;
long long len[555];
char s[11][22];
long long getlen(int x)
{
    return len[x];
}
int max(int a,int b){return a<b?b:a;}
long long min(long long a,long long b){return a<b?a:b;}
void js(int n,long long k,int l)
{
    if(n<=max(a,b))
    {
        int i;
        for(i=k;i<=k+l-1&&i<=len[n];i++)putchar(s[n][i]);
        return ;
    }
    long long L=getlen(n);
    if(k+l>L)l=L-k+1;
    long long L1=getlen(n-a);
    if(k>L1)js(n-b,k-L1,l);
    else if(k+l<=L1)js(n-a,k,l);
    else
    {
        long long l1=L1-k+1;
        js(n-a,k,l1);
        js(n-b,1,l-l1);
    }
}
int main()
{
    scanf("%d%d",&a,&b);
    int i;
    for(i=1;i<=max(a,b);i++)scanf("%s",s[i]+1);
    for(i=1;i<=max(a,b);i++)len[i]=strlen(s[i]+1);
    for(i=max(a,b)+1;i<=500;i++)len[i]=min(len[i-a]+len[i-b],100000000000000LL);
    int t;
    scanf("%d",&t);
    while(t--)
    {
        int n,l;
        long long k;
```

```
        scanf("%d%lld%d",&n,&k,&l);
        js(n,k,l);
        puts("");
    }
    return 0;
}
```