A

破译Yt

题目分析

读入字符串后对其中字母依次判断、修改即可。

示例代码

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    char s[105];
    scanf("%s", s);
    for (int i = 0; s[i]; i++)
        if (isupper(s[i])) //若为大写字母则修改
            s[i] = 'a' + i % 26;
    printf("%s", s);
    return 0;
}
```

B

分久必合，合久必分

题目分析

此题的本意是想要考察`scanf`与`gets`两种输入字符串方式的不同之处。

- 用`scanf`读入字符串时，会在空格处停止。
- 用`gets`读入字符串时，在空格处不停止，并将空格读入字符串后继续往下读，直到换行或输入结束时停止。

当然，本题难度较低，因此解法较多，没有局限性。

示例代码

```
#include <stdio.h>

char buf[200];

int main()
{
    int n = 0;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        int flag = 0;
        char tmp = 0;
```

```
        scanf("%d", &flag);
        if (flag == 0) {
            tmp = getchar();
            gets(buf);
            puts(buf);
        } else {
            while (1) {
                scanf("%s", buf);
                puts(buf);
                tmp = getchar();
                if (tmp != ' ') {
                    break;
                }
            }
        }
    }

    return 0;
}
```

C

收集石樱

难度	考点
2	贪心

问题分析

为了求得收集石樱的最大值，可把石樱的来源分为两类：原有的和新生长的。

对于新生长的石樱，简单分析题意可知，由于节点在每个单位时间内都会长出一个石樱，为了收集到更多的新生石樱，应当不重复地尽量走完每一个没有收集过的节点，如果走回头路而导致最终没有将所有节点都遍历一遍，显然会造成浪费。因为往回走只会收集到 1 个新生石樱，而往前走收集到的新生石樱数量显然大于等于 1。因此有一个基本的贪心策略：**在没有一次走完 n 个节点的时候，不要走回头路。**对 m, n 的关系分类讨论如下：

1. 当 $m <= n$ 时，此时无法遍历所有的节点或刚好遍历完一次，则选择 m 个节点收集，那么所收集的新生石樱的数量为 $\frac{(m-1)m}{2}$，而所收集的原有石樱数量为连续 m 个节点的总和最大值。对于长为 m 的固定区间最大值求解：考虑遍历每一个节点为区间起始点，再遍历该区间内所有石樱数量，相加后更新最大值，时间复杂度为 $O(mn)$，注意到 $1 \leq n \leq 10^5$，无法通过本题；注意到每个区间都可以由上个区间平移一位得到，记 sum 为上一个区间和，考虑用如下方法计算每次区间的最大值：

```
    for (i = m; i < n; i++)
    {
        sum += a[i] - a[i - m];
        max = MAX(sum, max);
    }
```

2. 当 $m > n$ 时，此时可以收集完所有原有石樱。对于新生石樱的数量，由于遍历完一次后会有回头路，因此不能如 1 中情况计算。由于路径不同，不易求解收集到的新生石樱数量。考虑从节点 1 走到节点 n，**未收集**到的新生石樱数量分别为 $n, n-1, \ldots, 1$，即 $\frac{n(n+1)}{2}$。可以发现，如果每次走过 n 个节点，该值不会随起点和终点的位置而变化；进一步采用我们的贪心策略，该值在任何时候都不会发生变化。由于 m 个单位时间内新生石樱数量总和为 mn，可得收集的数量为 $mn - \frac{n(n+1)}{2}$。

注意开 `long long`。

由于两种情况代码有重复内容，重构后答案如下：

参考代码

```
#include <stdio.h>
#define MAX(a,b) (((a)<(b))?(b):(a))
#define LL long long

LL a[100005];

int main()
{
    LL n, m, T, i, sum, max;
    scanf("%lld", &T);
    while (T--)
    {
        sum = max = 0;
        scanf("%lld%lld", &n, &m);
        for (i = 0; i < n; i++)
        {
            scanf("%lld", &a[i]);
            if (i < m)
                sum += a[i];
            else
            {
                max = MAX(sum, max);
                sum += a[i] - a[i - m];
            }
        }
        max = MAX(sum, max);
        if (m > n)
            printf("%lld\n", sum + n * m - n * (n + 1) / 2);
        else
            printf("%lld\n", max + (m - 1)* m / 2);
    }
    return 0;
}
```

D

阿求的岁月史书PLUS

考点	难度

考点	难度
字符串函数	3

题意分析

由题可知需要在字符串 S 中找到子串 s 并用 s' 替换之。

在讲解题目之前，说明一下关于 `scanf` 函数和 `gets` 函数读取字符串的区别

> 首先是两者认为字符串从何处开始的区别:

`scanf` 会**跳过**所有的空白符，直到找到第一个非空白符，并认为要读取的字符串从这个非空白符开始(事实上对除了 `%c` 之外的占位符都是这么处理)

而 `gets` 不管输入流中是什么字符都直接读取，也不管这一行有多长，都将这一整行视为一整个字符串——即使这一行只有一个换行符

> 另一个区别则是两者认为字符串从何处结束的区别:

`scanf` 会将开始读取后的第一个空白符视为字符串的结束，并认为要读取的字符串到这个空白符之前结束——那个空白符并**不会**进入字符串，也不会进行处理，而是留在输入流中等待后续处理(事实上对除了 `%c` 之外的占位符都是这么处理)。具体的例子就是换行符——你通过 `scanf("%s")` 的方式是不会将换行符读取进字符串的，而是留下一个换行符在输入流中。

`gets` 则将换行符 `\n` 或者文件结尾视为字符串的结束。如果以换行符结尾的话，换行符**会**进入字符串，并被改变为终止符 `\0` ，读取完之后输入流中是没有这个换行符的。

作为对照，你可以看看以下两部分代码对同一输入的不同处理:

输入为

```
1
abcd
```

`scanf` 的场合

```
//using scanf
int a; char s[100];
scanf("%d", &a);
scanf("%s", s);
printf("%s,%d", s, a);
```

输出为

```
abcd,1
```

`gets` 的场合

```
//using gets
int a; char s[100];
scanf("%d", &a);
gets(s);
printf("%s,%d", s, a);
```

输出为

```
,1
```

这是因为使用 `scanf` 读取 `%d` 时 `scanf` 将换行符留在了输入流中， `scanf` 读取字符串时会将这个换行符跳过并读取 `abcd` 作为读取的字符串，而 `gets` 则只读取了一行——剩下的换行符所在的一行，并将换行符变成了终止符，从而读取了一个空字符串，输出时也不会输出任何内容。
因此在做字符串类题目时需要小心处理换行符，不同函数（尤其是 `gets` ）对其处理不同从而容易导致操作错误。

从一个字符串中找另一个字符串可以用 `string.h` 库中的函数 `strstr` 来实现，其返回值是字符串 S 中子串 s 首次出现的位置的指针，不存在 s 则返回 `NULL` 。

替换子串 `s` 为 `s'` 可以遍历一遍依次替换各个字符。

注意！不能使用 `strcpy` 函数，这会把 S 中 s 之后的部分截断，你可以尝试以下代码

```
char S[20] = "hello,world!", s[20] = "llo", _s[20] = "aaa";
char *p = strstr(S, s);
strcpy(p, _s);
puts(S);
```

其输出结果为

```
heaaa
```

依次替换这一部分还可以使用 `string.h` 库中的函数 `memcpy` ，其函数原型为：

```
void *memcpy(void *str1, const void *str2, size_t n)
```

其效果为，将 `str2` 所指的内存复制 n 个字节进入 `str1` 中，随后返回指针str1

这个函数可以方便地将一个指针指向的内容复制进另一个指针指向的内容，例如

```
int a[5]={0,1,2,3,4},b[5]={4,3,2,1,0};
memcpy(a,b,3*sizeof(int));
for(int i=0;i<5;i++) printf("%d ",a[i]);
```

这串代码将数组 `b` 的前三个元素复制了数组 `a` ，因此输出结果为

```
4 3 2 3 4
```

再或者，可以按ppt上的演示，不进行替换，而是直接输出三段字符串，也是可以的

示例代码

```
#include <stdio.h>
#include <string.h>
int main(void) {
    int T;
    char S[1005] = "", s[1005] = "", _s[1005] = "";
    scanf("%d", &T);
    while (T--) {
        scanf("%s%s%s", S, s, _s);
        char *p = strstr(S, s);
        if (p != NULL) {
            // S中找到了s，将s'的对应部分复制给S中的s
            for (int i = 0; i < strlen(_s); i++) {
                p[i] = _s[i];
            }
            //或者你也可以使用以下语句
            // memcpy(p, _s, strlen(s) * sizeof(char));
            printf("%s\n", S);
        } else {
            // S中没找到s
            printf("I'll make this night never have happened!\n");
        }
    }
    return 0;
}
```

E

坤坤打篮球

题目分析

本题考察逆向思路，考虑得到n分的前一种状态分数有哪些可能，由题意可得n分前有$n-1$,$n-2$,$n-3$三种得分情况。所以我们只需要考虑上一种状态，直到推到最初状态。本题可用循环和递归实现。

参考代码

解法1：递归

```
#include<stdio.h>
int fib(int a) {
    if(a <= 0) {
        return 0;
    } else if(a==1) {
        return 1;
    } else if(a==2) {
        return 2;
    } else if(a==3) {
        return 4;
    } else {
        return fib(a-1)+ fib(a-2) + fib(a-3);//得分可能为三种情况的得分之和
```

```
    }
}
int main() {
    int n,t;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        printf("%d\n",fib(n));
    }
    return 0;
}
```

解法2：循环

```
#include<stdio.h>
int f[50];
int main() {
    int n,t;
    f[1] = 1;
    f[2] = 2;
    f[3] = 4;
    for(int i = 4; i <= 20;i++){
        f[i] = f[i-1]+f[i-2]+f[i-3];
    }
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        printf("%d",f[n]);
    }
    return 0;
}
```

F

这就是爵士乐吗

题目分析

本题理解题意可能较有难度，不要气馁。

每组四个输入，建议作为`%s`读入，并按照第一个字母给出对应的半音高，并逐个按照升降号个数加减一个半音高。

用半音高表示音符，在后续运算中进行比对时，对 12 取模即可让所有音平移到一个八度中。

完成后，设计一个重复十二次的循环，每次将所有音高升高半音，并用函数验证，就可以输出答案了。

代码示例

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LL long long
```

```c
#define min(a, b) (a > b ? b : a)
#define max(a, b) (a < b ? b : a)

char s[100] = "C D EF G A B";
char read[20010];
int fit(int x) {
    //I choose to compare the note with B,
    //and caculate the semi-tones.
    //It has the same result.
    if (x == 1)
        return 1;
    if (x == 3)
        return 1;
    if (x == 5)
        return 1;
    if (x == 6)
        return 1;
    if (x == 8)
        return 1;
    if (x == 10)
        return 1;
    if (x == 0)
        return 1;
    return 0;
}

int main() {
    int n;
    scanf("%d", &n);
    for (int q = 1; q <= n; q++) {
        int tone[5];
        for (int i = 1; i <= 4; i++) {
            scanf("%s", read);
            for (int j = 0; j <= 13; j++) {
                if (read[0] == s[j]) {
                    tone[i] = j;
                    break;
                }
            }
            for (int j = 1; j <= 20000; j++) {
                if (read[j] == 0) {
                    break;
                }
                if (read[j] == '#')
                    tone[i]++;
                if (read[j] == 'b')
                    tone[i] += 11;
            }
        }

        int ans = 0;
        for (int j = 1; j <= 12; j++) {
            int flag = 1;
            for (int k = 1; k <= 4; k++) {
                if (fit((tone[k] + j) % 12) == 0)
```

```
                    flag = 0;
            }
            if (flag)
                ans++;
        }
        printf("%s\n", ans ? "harmonic" : "SOUNDS JAZZ");
    }
    return 0;
}
```

G

表达式的值

题目分析

用加号划分段，段内只有乘号，统计数字然后乘起来

而且本题只保证了输入的数据都是int，所以结果可能是longlong

代码

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#include<string.h>
#include<time.h>
#define REP(i,a,b) for(register int i=a;i<=b;i++)
#define PER(i,a,b) for(register int i=a;i>=b;i--)
typedef long long LL;
//做法：用加号划分区间，计算内部的连乘
char s[1010];
LL solve(int l,int r,int flag){//对l~r区间的表达式求值
    LL res=0;
    if(flag){//加法乘法混合
        int lastpos=0;
        for(int i=l;i<=r;i++){
            if(s[i]=='+'){
                res+=solve(lastpos,i-1,0);
                lastpos=i+1;
            }
        }
        res+=solve(lastpos,r,0);
        return res;
    }
    LL tmp=0;
    res=1;
    for(int i=l;i<=r;i++)
        if(s[i]>='0'&&s[i]<='9')
            tmp=tmp*10ll+s[i]-'0';
        else
            res*=tmp,tmp=0;
    res*=tmp;
```

```
    return res;
}
int main(){
    scanf("%s",&s);
    printf("%lld",solve(0,strlen(s)-1,1));
}
```

H

打音游

题目分析

本题是打音游 $plus$ 版本，虽然增加了许多要求，但本质上依然是一个模拟题。

在写本题的时候要注意的几个细节：满血的时候吃心可以加分，吃心和吃音符不加连击数，死亡之后分数不再增加，以及对 $All\ Perfect$ 和 $Full\ Combo$ 条件的判断。

标准程序

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

const double mul[10] = {1.0, 1.1, 1.2, 1.3, 1.4, 1.5};
char str[10010];
int main(){
    while(scanf("%s", str) != EOF){
        int len = strlen(str);
        int ans = 0;
        int health = 200;       //初始生命值
        int combo = 0;
        int flag_perfect = 1;   //判断是否All Perfect
        int flag_combo = 1;     //判断是否Full Combo
        int flag_death = 0;     //判断是否死亡

        for(int i = 0; i < len; i++){
            if(str[i] == 'P' || str[i] == 'p' || str[i] == 'G' || str[i] == 'g')
{
                if(str[i] == 'g' || str[i] == 'G')
                    flag_perfect = 0;   //没有All Perfect
                combo++;
                int score;
                double rate = 1;    //判定系数
                if(str[i] == 'P' || str[i] == 'G')  //判断基础分值
                    score = 300;
                else
                    score = 200;

                if(str[i] == 'P' || str[i] == 'p')  //判断判定系数
                    rate = 1;
                else
                    rate = 0.6;

                if(combo < 50)                      //计算得分
```

```
                    ans += score * rate * mul[combo / 10];
                else
                    ans += score * rate * mul[5];
            }
            if(str[i] == 'M' || str[i] == 'm'){
                flag_combo = flag_perfect = 0;  //既没有All Perfect也没有Full Combo
                combo = 0;
                if(str[i] == 'M')   //扣除生命值
                    health -= 40;
                else
                    health -= 20;

                if(health <= 0){    //如果死亡就直接结束即可
                    flag_death = 1;
                    break;
                }
            }
            if(str[i] == 'n'){
                ans += 150;         //音符只增加分数
            }
            if(str[i] == 'H'){      //如果满血就加分
                if(health == 200)
                    ans += 600;
                health += 50;
                if(health > 200)    //血量上限是200
                    health = 200;
            }
        }

        printf("%d\n", ans);
        if(flag_perfect)            //对各种输出情况进行判断
            printf("All Perfect!\n");
        else if(flag_combo)
            printf("Full Combo!\n");
        else if(flag_death)
            printf("Game Over!\n");

    }

    return 0;
}
```

字符串大练习

题目分析

? 实现对字符串的增删改查。每个操作实现方法如下：

1. 尾接：直接调用 $strcat$ 函数；
2. 尾删：仅需将 $str[i]$ 置为 `'\0'` ；
3. 插入：调用 $strcat$ 将 str 第 i 位及之后的字符组成的字符串尾接到 s 上，再调用 $strcpy$ 将 s 复制到 str 的第 i 位上；

4. 删除：调用 $strcpy$ 将 str 第 j 位之后（不含第 j 位）的字符组成的字符串复制到 str 的第 i 位上；
5. 修改：若 str 不够将 s 所有字符替换上去，需要向后延长直至 s 结束，则可以直接调用 $strcpy$ 将 s 复制到 str 的第 i 位上；若 s 长度不到 s 结尾，可以调用 $memcpy$ 将 s 长度为 $strlen(s)$ 的内容复制到 str 的第 i 位上。二者区别在于 $strcpy$ 会复制 `'\0'`，而 $memcpy$ 不会；
6. 结束：判断读入的数字，如果是 `-1` 跳出循环。

? 最后需要查询字符串出现次数：用指针 p 记录当前遍历到的位置，每次从 p 开始调用 $strstr$ 函数查找字符串 s，若找到 s（即返回值不为 $NULL$），则计数 $+1$，否则跳出循环，每次更新 p 为找到 s 的位置 $+1$。

? 虽然题意有一点无趣，但希望大家能从本题中有所收获，理解字符串，注意 `'\0'`，善用指针，善用库函数，如 $str+i$ 既代表了 str 第 i 位的地址，也代表了 str 第 i 位及之后的字符组成的字符串等等。

示例代码

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[10005], s[10005];
    gets(str);
    while(1) {
        int op, i, j;
        scanf("%d", &op);
        if(op == -1) break;
        switch(op) {
            case 1:
                scanf("%s", s);
                strcat(str, s);
                break;
            case 2:
                scanf("%d", &i);
                str[i] = '\0';
                break;
            case 3:
                scanf("%d%s", &i, s);
                strcat(s, str + i);
                strcpy(str + i, s);
                break;
            case 4:
                scanf("%d%d", &i, &j);
                strcpy(str + i, str + j + 1);
                break;
            case 5:
                scanf("%d%s", &i, s);
                if(i + strlen(s) < strlen(str)) memcpy(str + i, s, strlen(s));
                else strcpy(str + i, s);
                break;
        }
    }
    scanf("%s", s);
    int num = 0;
    for(char *p = strstr(str, s); p != NULL; p = strstr(p + 1, s))
```

```
        ++num;
    printf("%d\n%s", num, str);
}
```

Author: 哪吒

J

Yt那山峦般沉重的歉意

题目分析

本题使用到了贪心和动态规划的思路。
记录数列为 $\{a_n\}$ 首先，对于区间 $[1,r]$ ，一定存在一个 l 使得 $[l,r]$ 比任何 $[x,r]$ 都要大。记录这个值为 m_r 。 若 m_{r-1} 为负数，则 $m_r=a_r$ ，否则 $m_r=a_r+m_{r-1}$ 。由于最大区间和一定有一个结尾，所以求出所有 m_i 后保留最大值即可。

代码示例

```
#include <stdio.h>

#define max(a, b) (a > b) ? a : b
#define min(a, b) (a < b) ? a : b

int main() {
    long long n, p;
    long long sum = 0, max = -99999999, bestl = 1, bestr = 1;
    long long reset = 0;
    scanf("%lld%lld", &n, &p);
    for (int i = 1; i <= n; i++) {
        long long a = ((n + i) * (n + i)) % p - (p / 2);
        sum += a;
        if (sum > max) {
            bestl = reset + 1;
            bestr = i;
            max = sum;
        }
        if (sum < 0) {
            sum = 0;
            reset = i;
        }
        // printf("%d ",a); print the array
    }
    // putchar(10);
    printf("%lld %lld %lld", bestl, bestr, max);
}
```

K

高高地飞起来吧，北京五号！

题目分析

要想飞的远，即要尽可能的往前走，尽可能减少向后走产生的反面影响。所以我们总可以按下面四个步骤走：

- 走完向前的距离。
- 尽可能往180度方向调整。
- 走完向后的距离。
- 原地转完剩余的角度。

那么我们核心就是使用这些角度能形成最接近180度（取模后）的角度。这里我们使用的递推的思想，**遍历每种角度用或不用**（也被称作01背包），进而获得所有可以达到的角度，得到最接近的角度。最后使用余弦定理计算距离出发点的水平距离。

示例代码

```
#include <stdio.h>
#include <math.h>
#define PI acos(-1)
#define N 10010
#define  min(x,y) (((x)<(y)) ? (x) : (y))
int n, op, x, y, dis, minn = 180, a[N], f[N];

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d%d", &op, &dis);
        if (op == 1)
            x += dis;
        if (op == 2)
            y += dis;
        if (op == 3)
            dis %= 360, a[dis]++;
        if (op == 4)
            dis %= 360, a[360 - dis]++;
    }
    f[0] = 1;
    for (int i = 0; i < 360; ++i)// 遍历得到每一种可能到达的角度
        while (a[i]--)
            for (int j = 720; j >= 0; --j)
                if (f[j])
                    f[(j + i) % 720]++, minn = min(minn, abs((i + j) % 360 -
180));//找到最接近180的角度
    double res = sqrt(1ll * x * x + 1ll * y * y - 2ll * x * y * cos((180 - minn)
* PI / 180));//余弦定理求距离
    if ((int)res % 2)
        puts("SHIE");
    else
        puts("CHUANYUAN");
    printf("%lf", res);
    return 0;
}
```