C2 - Solution

A ASCII 2024

难度	考点
1	ASCII 码，条件语句

题目分析

本题要求根据输入的ASCII码输出其对应的字符，并做一个简单的条件语句判断。

实际上，计算机中是无法直接存储字符的，只能存储一个又一个的 01 位，所以人们想了一个聪明的办法，就是把字符与整数相对应，然后把这个整数存到计算机中。这样，下次我从计算机中再取出一个整数，并告诉计算机：“这个整数是一个字符”，就能够成功地实现存取字符了。C 语言的数据类型 `char` 常以 ASCII 对字符进行编码；比如根据 Hint，`a` 对应的 ASCII 值就是整数 97，我们告诉计算机以字符的形式解释整数 97 并输出时，就会给我们输出字符 `a`。

示例代码

```c
#include<stdio.h>

int main() {
    int n;
    scanf("%d", &n);

    // 判断是不是可见字符
    if (n <= 126 && n >= 32) {
        printf("%c", n);
    } else {
        printf("Invisible Character!\n");
    }

    return 0;
}
```

B 不会算成绩的 shtog

难度	考点
2	循环结构, 浮点数, 格式化输出, 类型转换

题目分析

这道题首先要理解算数平均和加权平均的公式。前者很容易理解，就是不考虑学分，直接所有课的成绩加起来然后取个平均。后者的话则是：每门课的成绩乘以学分然后求和，求和的结果除以总学分（所有课的学分加起来）。理解公式之后就不断循环读入每门课的成绩和学分，得到四个值：总成绩，课的数量（这个不用算，是直接读入的）；加权总成绩，总学分。前两者相除得到算数平均分，后两者相除得到加权平均分。

示例代码

```
#include <stdio.h>

int main() {
    int n, i;
    // 注意对这些需要求和的变量的值进行初始化
    int sum = 0;
    double score = 0.0, xuefen = 0.0; // xuefen表示总学分，score表示加权总分
    scanf("%d", &n);
    for (i = 0; i < n; ++i) {
        int s;
        double x;
        scanf("%d%lf", &s, &x);
        // 在循环的过程中计算出三个值：总成绩sum，总学分xuefen，加权总分score
        sum += s;
        xuefen += x;
        score += x * s;
    }
    // 最后两两相除得到结果
    // 注意由于sum定义为了int类型，在除以n之前要先乘一个1.0将结果转为浮点类型
    printf("%.2lf %.2lf\n", 1.0 * sum / n, score / xuefen);
    return 0;
}
```

C 贪吃蛇（easy version）

难度	考点
2	循环，判断

题目分析

这题的思路很简单，每次读取一个整数 a，根据当前长度 len 来判断其是否可以加上 a。如果 $len+a\geq 1$，那么说明吃掉这个食物并不会使贪吃蛇的长度变为 0 甚至变为负数，因此，贪吃蛇一定会吃掉这个食物，此时应该更新 len；否则贪吃蛇就会忽略这个食物，len 保持不变。

当然，这题也可以使用大家刚刚学过的数组来完成，思路基本一样，只是将输入和遍历分开了，详见示例代码 2

注意本题的数据范围。虽然 $-10^9\leq a\leq 10^9$，但是随着贪吃蛇吃的食物越来越多，它的长度可能会超过 `int` 可以表示的最大范围。因此你需要使用 `long long` 来记录贪吃蛇当前的长度。

示例代码 1

```c
#include <stdio.h>
int main(void)
{
    int n, x, a;
    scanf("%d%d", &n, &x);
    int cnt = 0;
    long long len = x; // 使用 long long 类型存储贪吃蛇的长度，防止溢出
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a);
        if (len + a >= 1)
        {
            len += a;
            cnt++;
        }
    }
    printf("%d %lld", cnt, len); // 输出 long long 类型需要使用 %lld
    return 0;
}
```

D CVB

难度	考点
3	字符读入，循环，判断

题目分析

本题目只需要在读入数字之后结合循环和`getchar`读入字符串，然后对于每一个读入字符进行判断即可。

需要注意以下细节：读入数字后再读入字符序列的话需要考虑到换行符的问题，本题目中保证了换行符仅有`\n`,故需要一个额外的`getchar`来处理。也要注意不要在有多个`b`的时候输出多次`CVBB`

示例代码

```
#include<stdio.h>
int n;
char c;
int main(){
    scanf("%d",&n);
    getchar();
    for(int i=1;i<=n;++i){
        c=getchar();
        if(c=='b'){
            printf("CVBB");
            return 0;
        }
    }
    printf("CVB");
    return 0;
}
```

E 小懒獭与距离

难度	考点
3	浮点运算，输入输出，标准库函数

题目分析

参考题目中的示例，注意读入时候的处理。读入数据后，利用 `fabs` 以及 `sqrt` ，计算题目中给出的公式，题目要求输出小数点后四位，也可以参考示例，输出 `printf("%.6lf", distance);`

示例代码

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

int main() {
```

```
    double a, b, c, d;
    double x, y, z;

    // 注意读入时的处理
    scanf("%lf%lf%lf%lf", &a, &b, &c, &d);
    scanf(" (%lf,%lf,%lf)", &x, &y, &z);

    // 直接计算并输出
    printf("%.4lf", fabs(a * x + b * y + c * z + d) / sqrt(a * a + b
* b + c * c));

    return 0;
}
```

F 检查 AI 代码

难度	考点
4	数组

题目分析

共 n 段代码，我们可以先用数组存每段代码是否为可疑代码。

等全部代码是否为可疑代码标记完成后，我们可以按顺序读入每段可疑代码的人工评审结果，并计算认为这段代码由人工智能生成的助教数。如果大于等于 2 人，则由题意，这段代码被认为由人工智能生成，我们就输出它的编号。

注：由于题目并没有保证至少存在一个被认为由人工智能生成的代码，因此你可能不需要输出任何编号。不过由题知，这种情况下仍然需要输出 `Answer:`。

示例代码

```
#include<stdio.h>
int n,A[1024],a,b,c;
int main(){
    printf("Answer:");//题目要求输出以Answer:开头，无论是否有后续输出。
    scanf("%d",&n);
    for ( int i=1 ; i<=n ; i++ )
        scanf("%d",&A[i]);//数组中元素的读入和相同类型的普通变量相同。也可
以用A+i替换&A[i]。
    for ( int i=1 ; i<=n ; i++ )
        if ( A[i]==1 ){//可疑代码。
            scanf("%d%d%d",&a,&b,&c);//由于助教只会给出0或1，因此直接加起
来就是认为程序由AI生成的助教数。
```

```
            if ( a+b+c>=2 )//大于等于两名助教认为代码由AI生成。
                printf("%d ",i);//输出编号。会在行末多输出一个空格，但是OJ
的评判标准会忽略行末空格和输出末尾空行，因此不影响评判结果。
        }
    return 0;
}
```

G ddz 饮马

难度	考点
4	数学计算

题目分析

本题核心是经典将军饮马问题，对于只有一条河的解法是：将其中一个点（起点或终点）沿河对称过去再计算距离。（可自行百度将军饮马问题）

而本题有两条河，但因为只需要去一条河喝水就行，所以这两条河是可以独立计算的，将算出来的两个结果取最小值即可。

示例代码

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

int main() {
    int x1, y1, x2, y2;
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
    double distance1 = sqrt((x1 - x2) * (x1 - x2) + (y1 + y2) * (y1 +
y2));
    double distance2 = sqrt((x1 + x2) * (x1 + x2) + (y1 - y2) * (y1 -
y2));
    if (distance1 < distance2) {
        printf("%.2lf", distance1);
    } else {
        printf("%.2lf", distance2);
    }
    return 0;
}
```

H 板凳龙

难度	考点
5	较为复杂的if判断

题目分析

本题目可以用一些向量相关的数学知识解决，也可以用较为复杂的`if`语句暴力解决。解决方法多样。在此提供一种思维较为简单的做法。

依次判断点是否位于长方体内部，外部，如果都不位于，则此点位于边缘。判断方式则利用坐标大小之间的关系即可。

需要注意的是本题目并没有保证给出的矩形的点的位置，所以不能粗暴的假定给出的对角线的方向。

示例代码

```
#include<stdio.h>
int x1,x2;
int y1,y2;
int z1,z2;
int x,y,z;
int n;
int main(){
    scanf("%d%d%d%d%d%d",&x1,&y1,&z1,&x2,&y2,&z2);
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int flag=0;
        scanf("%d%d%d",&x,&y,&z);
        if(((x<x1)&&(x>x2))||((x<x2)&&(x>x1))){
            if(((y<y1)&&(y>y2))||(y<y2)&&(y>y1)){
                if(((z<z1)&&(z>z2))||(z<z2)&&(z>z1)){
                    flag=1;
                }
            }
        }
        if((x<x1&&x<x2)||(x>x1&&x>x2)||(y<y1&&y<y2)||(y>y1&&y>y2)||
(z<z1&&z<z2)||(z>z1&&z>z2)){
            flag=2;
        }
        if(flag==1)
            printf("Inner\n");
        if(flag==2)
            printf("Outer\n");
        if(flag==0)
            printf("Edge\n");
    }
```

```
    return 0;
}
```

I 最大子段和

难度	考点
5	贪心、动态规划

题目分析

由题意可知，我们要找出数列中**连续且非空**的一段，使其和最大，如果不看数据范围的话，我们可以很容易想到一种暴力的解法：直接枚举数列中所有存在的**连续且非空**的子列，将其和分别求出来，再用`ans`记录它们的最大值并输出，代码大致如下。

```
#include<stdio.h>
int main()
{
    int n,a[200005],ans=-10000000;
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++)//枚举数列的左边界
    {
        for(int j=i;j<=n;j++)//枚举数列的右边界
        {
            int sum=0;
            for(int k=i;k<=j;k++)
            {
                sum+=a[k];//求和
            }
            if(sum>ans) ans=sum;//记录最大子段和
        }
    }
    printf("%d",ans);
    return 0;
}
```

但我们会发现这种思路只能通过30%的数据点，因为对于 n 在10^5的数量级时会时间超限（~~题目背景有说 ddl 快到子~~），所以要想一个更优的做法。

首先，我们用一个数组`dp[i]`表示数列中以 a_i 结尾的子列的最大子段和，之后顺序遍历整个数列。可以发现，对于`dp[i]`的取值分为两种可能：一种可能是将 a_i 与前面的子列合并，另一种是只取 a_i（这是因为如果前面的子段和加上 a_i 的值比 a_i 小，还不如不合并，只要 a_i ），因此可以得到如下状态转移方程：

$$dp[1] = a[1] \quad (i = 1)$$

$$dp[i] = max(dp[i-1] + a[i], a[i]) \quad (2 \le i \le n)$$

由于数列可能分出很多段子列，最后要遍历`dp[i]`找出其中的最大值即为整个数列的最大子段和`ans`。

示例代码

```
#include<stdio.h>
int main()
{
    int n;
    int a[200005],dp[200005];
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
    }
    int ans=dp[1]=a[1];
    for(int i=2;i<=n;i++)
    {
        //按状态转移方程求dp[i]的值
        if(dp[i-1]+a[i]>a[i])
        {
            dp[i]=dp[i-1]+a[i];
        }
        else
        {
            dp[i]=a[i];
        }

        //ans始终为dp[i]中最大值
        if(dp[i]>ans)
        {
            ans=dp[i];
        }
    }
    printf("%d",ans);
    return 0;
}
```