C6-2021级航类-第六次上机题解

A 简单的统计

难度	考点
1	模拟

题目分析

由数据范围可以知道，即使所有同学都满足情况，和也在 int 范围内，所以不需要开 $long\ long$。

定义 sum 保存满足条件的同学的分数之和，定义 cnt 保存满足条件的同学的个数，最后将 sum 除以 cnt 即可得到答案。

注意满足的条件是：**大于等于** 60 分，且**小于** 80 分的同学。

示例代码

```c
#include<stdio.h>

int n, cnt;
int a[1000005];
double sum, ave;

int main(){
    int i;

    scanf("%d", &n);
    for(i=1; i<=n; ++i){
        scanf("%d", &a[i]);
        if(a[i]>=60 && a[i]<80){
            sum += a[i];
            ++ cnt;
        }
    }
    ave = sum/cnt;

    printf("%.2lf", ave);
    return 0;
}
```

B LJF排排坐

难度	考点
1	快排

题目分析

本题中 n 最大能达到 2^{20}，若直接使用冒泡排序会出现 TLE，应使用 $qsort$ 。

由于数据范围较大，自定义 cmp 函数不能返回两数之差。

示例代码

```c
#include <stdio.h>
#include <stdlib.h>
int cmp(const void *p1, const void *p2)
{
    int a = *(int *)p1, b = *(int *)p2;
    return a > b ? 1 : -1;
    //注意！！这里不可以返回 a-b
    //假设 a = 2147483647, b = -2147483648, a - b = 4294967295造成溢出
    //此时的返回值为-1
}
int main()
{
    int n;
    scanf("%d", &n);
    int *ret = (int *)malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++)
        scanf("%d", ret + i);
    qsort(ret, n, sizeof(int), cmp);
    for (int i = 0; i < n; i++)
        printf("%d ", ret[i]);
    free(ret);
}
```

C 爱祖国、爱人民、爱劳动、爱科学、爱社会主义

难度	考点
2	字符串

题目分析

本题是对字符串子串匹配的考察，了解 `strstr()` 函数即可，如果存在待查找子串，该函数返回子串首次出现的地址；如不存在待查找的子串，则返回 NULL 。本题给出两种实例代码，示例代码 1 使用 `gets()` 函数获取输入，示例代码 2 使用 `scanf()` 的正则表达式获取输入，其中 `%[^\n]` 含义为：连续读入若干个字符，直到遇到 `\n` 停止。

示例代码 1

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

int main()
{
    char R[50],S[2022];
    gets(R);
    gets(S);
    if(strstr(S,R)==NULL) printf("Oh!No!");
    else printf("%d",strstr(S,R)-S+1);//此处为指针运算
    return 0;
}
```

实例代码 2

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

int main()
{
    char R[50],S[2022];
    scanf("%[^\n]",R);
    scanf(" %[^\n]",S);//注意清除上一行遗留的换行符
    if(strstr(S,R)==NULL) printf("Oh!No!");
    else printf("%d",strstr(S,R)-S+1);
    return 0;
}
```

D Laika的密码

难度	考点
1	回文字符串

题目分析

简单的回文字符串判断，题解给出的是使用前后两个双指针进行判断。同时考察读取字符串特定位的操作，没有什么难度。

示例代码

```c
#include<stdio.h>
#include<string.h>
char s[10005];
int main()
{
    fgets(s,10003,stdin);
    int l=strlen(s);
    int i;
    int flag=1;
    for(i = 0; i < (l/ 2); i++)
    {
        if(s[i] != s[l - 1 - i])
        {
            flag=0;
            //printf("%d",i);
            break;
        }
    }
    if(flag)
    {
        i=1;
    }
    else
    {
        i=0;
    }
    while(i<l)
    {
        //printf("%d",i+1);
        printf("%c",s[i]);
        i+=2;
    }
    return 0;
}
```

E 小小的统计

难度	考点
3	字符串，数据类型

题目分析

观察输入要求，所有的数在同一行上，而且在输入这个数之前我们并不知道这个数是浮点数还是整数，又由于存在 2.0 这种小数，所以本题无法通过全部以浮点数类型输入，最后判断与对应整数的差值是否小于一定范围的方法来判断输入的数字是整数还是小数。

正确的方法为利用 `scanf("%s",s)` 实现输入，`scanf("%s",s)` 在遇到空白字符时就停止，符合题目中两个数中间用空格隔开的要求。在输入完成后，利用 `strchr` 函数判断字符串中有没有小数点，从而判断输入数字是整数还是小数。最后利用 `sscanf` 函数从字符串中读进与格式相符的数据，进行要求的统计运算即可。

示例代码

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
int main()
{
    char s[100];
    int a;
    double f;
    int sumd=0,numd=0,numf=0;
    double sumf=0.0;
    while(scanf("%s",s)!=EOF)
    {
        if(strchr(s,'.')==NULL)
        {
            sscanf(s,"%d",&a);
            sumd+=a;
            numd++;
        }
        else
        {
            sscanf(s,"%lf",&f);
            sumf+=f;
            numf++;
        }
    }
    printf("%d %.5lf\n",numd,1.0*sumd/numd);
    printf("%d %.5lf\n",numf,sumf/numf);
    return 0;
}
```

F 宋老师的名次预测4.0

难度	考点
3	指针、字符串

题目分析

本题主要是考察 `sscanf` +指针的用法，`sscanf` 的使用参考如下

```
//可以使用stdio.h头文件中的sscanf函数从一个字符串中读进与指定格式相符的数据，用法如下：
char s[]={"123"};
int a;
sscanf(s,"%d",&a);//其他数据类型同理，同时读进多个亦可。
printf("%d",a);//输出为123
```

对于本题，我们可以每次读取一行字符串 s，然后定义一个字符型指针 p 指向该字符串。初始时指针 p 指向 s 首位置，然后使用 `sscanf(p, "%d", &num)` 进行读取，读取完成后将指针 p 移到下一个数字开始的地址，同样进行读取，直到无法正确读取数字。

示例代码

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

int main() {

    int num, cnt = 0, now = 0;
    char s[1005], *p, sNum[20];
    while (gets(s) != NULL) {//gets每次读取一行
        now = cnt = 0;
        p = s;//p初始指向s首位置
        while (sscanf(p, "%d", &num) == 1) {//每次只要能够成功读取一个数字
            now++;
            cnt += num == now;//计数
            while (isdigit(*p))//指针向后走过当前数字
                p++;
            while (isspace(*p))//指针向后走过空格
                p++;
        }
        printf("%d\n", cnt);//输出
    }

    return 0;
}
```

G 图图的红包区间合并

难度	考点
3	散列表

题目分析

课件题目的改编，注意到课件中数据都为正整数，而本题输入为浮点数，又有浮点数仅存在两位小数，于是我们想到将输入的浮点数乘 100 从而转化为正整数，就能利用散列表来解决问题了。

注意浮点数乘 100 后要用 `round` 函数进行四舍五入，否则可能产生误差。举例：`printf("%.15lf", 0.29 * 100)`

总结与 `爱偷懒的球场管理员` 一题的不同：本题数据量大，但每个数据不大（不超过两万），适合使用散列表；“球场管理员”一题数据量相对较小，但每个数据较大（达到 10^8 量级），适合使用排序。我们须依据数据的特点选择合适的方法来解决问题。

示例代码

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

int main() {

    double m1, m2;
    int n, left, right, range[20005] = {0}, i, j;       // 注意range数组须初始化为0

    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        scanf("%lf%lf", &m1, &m2);
        left = round(m1 * 100);          // 取整须用round函数四舍五入
        right = round(m2 * 100);
        if (right > range[left])
            range[left] = right;
    }

    for (i = 1; i <= 20000; i++) {
        if (range[i]) {
            left = i;
            right = range[i];
            for (j = i + 1; j <= right; j++) {
                if (range[j] > right)                           // 新区间更远
                    right = range[j];                           // 拓展右区间
            }
            printf("%.2lf %.2lf\n", left / 100.0, right / 100.0);
            i = right;
        }
    }

    return 0;
}
```

H zhnの镜像串

难度	考点
6	字符串

题目分析

本题首先注意是多组输入，要注意输入方法。

之后在判断的过程中，我们只需要判断第 i 个与第 $len-i-1$ 个是否满足对称的条件即可。

（len 为长度，我的题解是从 0 开始存入的字符串）

check 函数中是判断这字符是否满足对称的条件。

函数中的 `*a` 和 `` `*b `` 可以理解成为两个数组，不过我是用指针定义的。

zhn的水题，希望大家食用愉快。

示例代码

```
#include <stdio.h>
#include <string.h>
char s[1005];
int check(char x,char y){
    char *a = "AbdHIMOopqTUVvWwXxY";
    char *b = "AdbHIMOoqpTUVvWwXxY";
    int len=strlen(a);
    for(int i=0;i<len;i++){
        if((x==a[i])&&(y==b[i])) return 1;
    }
    return 0;
}
int main(){
    while(scanf("%s",s)!=EOF){
            int len=strlen(s);
            int flag=1;
            for(int i=0;i<len;i++){
                if(!check(s[i],s[len-i-1])) {flag=0;break;}
            }
            if(!flag) puts("OH,NO!");
            else puts("Cool");
    }
    return 0;
}
```

I 小型文本编辑器

难度	考点
3	字符串

题目解析

在处理本题的时候，需要理解“在一次替换（删除）操作中新出现的字符串不应该参与到当前操作的匹配中”这句话的含义。实际上样例2为我们具体说明了特殊情况的处理方式：

- 对于第一个查询操作：在当前 `text` 的第三个和第四个字符处都出现了 `AA` 这个字符串，因此输出 `3 4` ；
- 对于第二个替换操作：尽管在当前 `text` 的第五个和第七个字符处都出现了 `ABA` 这个字符串，但是我们需要优先替换掉先出现的第五个字符处的 `ABA` ，此时 `text` 变为 `BBAAABA` ，这时虽然看上去第五个位置仍然是 `ABA` 这个字符串，但是这个 `ABA` 的第一个字符来自于替换后的 `A` ，实际上是新组合出来的字符串，不应该参与到当前的匹配中，因此不能被替换；
- 对于第三个删除操作：在当前 `text` 的第二个和第六个字符处都出现了 `BA` 这个字符串，需要注意的是第一次将第二个字符处的 `BA` 删除后，此时 `text` 变为 `BAABA` ，这时虽然看上去第一个位置仍然是 `BA` 这个字符串，但是这个 `BA` 是删除后新组合出来的字符串，不应该参与到当前的匹配中，因此不能被删除。

因此对于替换和删除两个操作，在 `text` 中匹配 `str` 时，若此时在第 `i` 个字符处匹配到了 `str` ，那么下一次匹配应该从第 `i + strlen(str)` 个字符处开始。而对于查询操作没有这个限制，下一次匹配应该从第 `i + 1` 个字符处开始。

此外，本题推荐采用函数式编程。一来使得逻辑清晰；二来实际上删除操作等同于替换操作，即替换为空字符串，可以简化编程。

示例代码

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

#define LEN 1007

/*
 * 作用: 从标准输入流里读取一个字符串
 * 参数:
 * - dst: 从标准输入流里读入一个字符串到dst
 * - len: 最多读len个字符
 * 返回值:
 * - 读取到文件尾时返回NULL
 * 备注:
 * - 此函数会去除字符串结尾处的\n
 */
char *fgets_without_n(char *dst, int len) {

    char *ret = fgets(dst, len, stdin);
    if (ret != NULL && dst[strlen(dst) - 1] == '\n') {
        dst[strlen(dst) - 1] = 0;
    }
```

```c
    return ret;
}

/*
 * 作用: 将一个字符串的特定子字符串替换为另一个字符串
 * 参数:
 * - text: 原字符串，直接在这个字符串上进行修改
 * - str: 需要被替换掉的字符串
 * - replace_str: 被替换为的字符串
 * 返回值:
 * - 无
 */
void replace(char *text, char *str, char *replace_str) {

    int len = (int)strlen(str);
    char res[LEN] = {0}, *start = text, *find; // 处理完的字符串暂存在res中
    while ((find = strstr(start, str)) != NULL) {
        *find = 0; // 方便我们使用strcat
        strcat(res, start);
        strcat(res, replace_str);
        start = find + len; // 跳过len个字符再进行匹配
    }
    strcat(res, start);
    strcpy(text, res);
}

/*
 * 作用: 将一个字符串的特定子字符串删除
 * 参数:
 * - text: 原字符串，直接在这个字符串上进行修改
 * - str: 需要被删除的字符串
 * 返回值:
 * - 无
 */
void delete(char *text, char *str) {

    replace(text, str, "");
}

/*
 * 作用: 查找一个字符串中的特定子字符，并输出其位置
 * 参数:
 * - text: 原字符串
 * - str: 需要被查找的字符串
 * 返回值:
 * - 无
 */
void find(char *text, char *str) {

    char *start = text, *find;
    int exist = 0;
    while ((find = strstr(start, str)) != NULL) {
        printf("%d ", find - text + 1); // 从1开始计数
        start = find + 1; // 从下一个字符开始进行匹配
        exist = 1;
```

```
    }
    if (!exist) {
        printf("NOT EXIST");
    }
    printf("\n");
}

int main(int argc, const char * argv[]) {

    char text[LEN], str[LEN], replace_str[LEN];

    fgets_without_n(text, LEN);

    int mode;  // 1-find 2-repalce 3-delete
    while (~scanf("%d", &mode)) {
        getchar();
        switch (mode) {
            case 1:
                fgets_without_n(str, LEN);
                find(text, str);
                break;
            case 2:
                fgets_without_n(str, LEN);
                fgets_without_n(replace_str, LEN);
                replace(text, str, replace_str);
                printf("%s\n", text);
                break;
            case 3:
                fgets_without_n(str, LEN);
                delete(text, str);
                printf("%s\n", text);
                break;
            default:
                break;
        }
    }

    return 0;
}
```

J 安全小达人

难度	考点
6	二维数组、字符串

题目分析

如果我们使用暴力枚举，检查所有从 $newpd$ 转换到 $oldpw$ 的方法，再取最短的一个，时间复杂度会达到指数级别，是不满足我们的时间限制的。由于我们只需要找到距离最短的序列而不是所有可能的序列，我们考虑使用动态规划，动态规划的步骤大致分为以下几个：找到子问题，分析状态转移方程，定义边界条件，最后写代码。

首先我们看看什么是子问题，子问题就是问题规模严格小于原问题，而求解方法等和原问题完全相同的问题。我们观察问题的条件，分别是新密码 $newpw$ 和旧密码 $oldpw$ ，问题的规模可能和字符的种类以及字符串的长度有关。再仔细想想，把 `b` 变成 `a` 和把 `c` 变成 `a` 是没有区别的，所以问题的规模就是字符串的长度。

有了子问题，我们就可以定义子状态 $dp[i][j]$ 表示从长度为 i 的新密码和长度为j的旧密码的差别。下面我们分析 $dp[i][j]$ 和其子问题之间的关系，我们只需要考虑 $oldpw$ 的第 j 个字符和 $newpw$ 的第 i 个字符的匹配是怎么转换过来的。由于操作只有插入删除修改三种，下面我们一一讨论。

1. $oldpw$ 的第 j 个字符和 $newpw$ 的第 i 个字符的匹配是通过插入操作转换的。那么子问题就是将 $newpw$ 的前 i 个字符转换成 $oldpw$ 的前 $j-1$ 个字符，需要的操作数为 $dp[i][j-1]+1$
2. $oldpw$ 的第 j 个字符和 $newpw$ 的第 i 个字符的匹配是通过删除操作转换的。那么子问题就是将 $newpw$ 的前 $i-1$ 个字符转换成 $oldpw$ 的前 j 个字符，需要的操作数为 $dp[i-1][j]+1$
3. $oldpw$ 的第 j 个字符和 $newpw$ 的第 i 个字符的匹配是通过替换操作转换的。那么子问题就是将 $newpw$ 的前 $i-1$ 个字符转换成 $oldpw$ 的前 j 个字符，需要的操作数是 $dp[i-1][j-1]+1$ （ $oldpw[j]!=newpw[i]$ ）或者 $dp[i-1][j-1]$ （ $oldpw[j]=newpw[i]$ ）

由此我们可以得到状态转移方程如下：

$$dp[i][j]=\min(dp[i][j-1]+1,dp[i-1][j]+1,dp[i-1][j-1]+(oldpw[j]!=newpw[o]))$$

最后我们讨论边界条件，对于 $dp[i][0]$ ，我们只需要删除 i 次就能将新密码转换成旧密码，所以 $dp[i][0]=i$ ；对于 $dp[0][j]$ ，我们只需要插入 j 次就能够将新密码转换为旧密码，所以 $dp[0][j]=j$ 。

示例代码

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

const int N=1010,INF=1E9;

int min(int x,int y) {
    if (x<y)
        return x;
    return y;
}
```

```
11
12  int main() {
13      int dp[N][N];
14      char src[N],dst[N];
15      int len1,len2;
16      int i,j;
17      scanf("%s",src+1); //新密码字符串
18      len1 = strlen(src+1);
19      scanf("%s",dst+1); //旧密码字符串
20      len2 = strlen(dst+1);
21
22      //边界条件+初始化（因为要算最小值，所以dp先初始化为一个比较大的值）
23      memset(dp,INF,sizeof(dp));
24      for(i=0;i<=len1;i++) dp[i][0]=i;
25      for(i=0;i<=len2;i++) dp[0][i]=i;
26
27      for(i=1;i<=len1;i++) {
28          for(j=1;j<=len2;j++) {
29              //状态转移方程
30              dp[i][j] = min(dp[i][j-1],dp[i-1][j])+1;
31              dp[i][j] = min(dp[i][j],dp[i-1][j-1]+(src[i]!=dst[j]));
32          }
33      }
34
35      printf("%d\n",dp[len1][len2]);
36
37      return 0;
38  }
```

K 终极进制

难度	考点
4	大数进制转换

问题分析

我们都学过用“模 2 取余法”来将一个 10 进制数转换为一个二进制数，进而可以推广到“模 n 取余法”，经其转换为 n 进制（ n 任意指定）。第一次的商被用作第二次的除数，循环直至商为 0，先余为低位，后余为高位，转换后的数为余数组的逆序。

可以想象，大数转换步骤同样如此。问题只在于大数的整除和取余。对于大数的除法计算，我们不妨先假设一个小数 12，用 12 除以 2 。12 是一个十位数，十位上是 1 ，个位上是 2，按照我们正常的思维来看，这个计算应该是下面这样的：

$$
\begin{array}{r}
06 \\
2\overline{)\,12} \\
0 \\
\hline
12 \\
12 \\
\hline
00
\end{array}
$$

我们发现在第一轮运算时，十位上的 1 作为被除数，2 作为除数，得到的商是 0 ，余数是 1 (可以断言只考虑当前这一个数位的计算，余数或是 0 ，或是 1 ，若是 1 的话，则进下一数位（这里即对个位进行运算）时，要用 1 乘上进制（这里是 10 ）再加上下一个数位上的值（这里是 2 ）），即得到运算进入个位时被除数是 12 ，除数是 2 ，得到的商是 6 ，余数是 0 。第一轮运算的结果是商是 06 ，余数是 0 。

进入第二轮运算，则上一轮的商 6（这里首先要去掉前面多余的 0 ）变成本轮的被除数，如此下去，即可得到每轮的余数。

推广开来，如果被除数是一个 1000 位的大数，例如 “12343435154324123.....342314324343”

那么我们照样可以从第一个数位开始逐位考虑，比如第一位是 1（作为被除数），2 是除数，得到的商是 0 ，余数是 1 ，然后是第二个数位 2 ，由于上一位留下了余数 1 ，则此时被除数应该是 $1\times10+2=12$,所以得到的商是 6 ，余数是 0 ，即运算到此时的商是 06 ，然后是第三个数位 3 ，由于上一个数位留下的余数是 0 ，所以此时被除数就是 3 。以此类推就完成第一轮的运算，这一轮完毕后，需要把得到的商变成下一轮的被除数，继续上述的运算，直到被除数为 0 才停止。

参考代码

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
char a[10005], ans[10005], tem[10005];//储存余数的数组稍微开大一点
void myatoi(char a[]){//将字符转化为数字
    int i;
    for(i = 0; a[i] != 10 && a[i] != -1; i++);//读取字符长度
```

```c
    int a_len = i;//记录字符长度
    for(i = 0; i < a_len; i++){
        a[i] -= '0';//通过减去0的ascii码来转换
    }
}
int div_cal(char a[], int scale, char ans[]){
    int i = 0, j = 0, k = 0, res = 0, temp, begin;
    while(1){//每一轮计算后，可以选择去除前导零，该份代码未去除
        k = 0, res = 0;
        for(i = 0; a[i] != -1; i++){
            tem[i] = (res * 10 + a[i]) / scale;//整型相除
            res = (res * 10 + a[i]) - tem[i] * scale;//计算余数留作下一位计算
        }
        ans[j++] = res;//将最后的余数记录在ans数组中
        memcpy(a, tem, sizeof(a[0]) * (i + 5));//将tem数组复制到a数组
        memset(tem, -1, sizeof(tem[0]) * (i + 5));//tem数组初始化
        while(a[k] == 0)    k++;
        if(a[k] == -1)
            break;
    }
    return j;
}
int main() {
    while(1){//多组不定组数据输入
        int scale = 4, i = 0, j = 0, k = 0;
        memset(ans, -1, sizeof(ans)), memset(a, -1, sizeof(a)), memset(tem, -1, sizeof(a));//初始化为-1
        scanf("%s", a);
        i = strlen(a);
        if(a[0] == EOF) break;//读到结束符时，意味着没有更多输入数据了，直接跳出循环
        a[i] = -1;
        myatoi(a);//将字符转化为数字
        j = div_cal(a, scale, ans);
        for(i = j - 1; i >= 0; i--){
            if(ans[i] == 0 && k == 0 && ans[i + 1] != -1)
                continue;//输出时消除前导零
            k = 1;
            if(ans[i] != 3)
                printf("%c", ans[i] + '0');//打印结果
            else
                printf("moo~");
        }
        puts("");
    }
    return 0;
}
```