A - 字母统计

难度	考点
1	字符串

题目分析

主要考察了对字符串最基本的操作：遍历、求长度。两种遍历方式，第一个示例程序利用字符串中的结束符号 `'\0'` 判断字符串结尾；第二个先用 `strlen()` 函数求出字符串长度 `len`，在利用循环语句访问 `s[0]...s[len-1]`。 需要注意的是，一定不要写成 `for(i = 0; i <= strlen(str) - 1; i++)`。两个原因如下：

1. `strlen` 函数的时间复杂度是 $O(n)$ 的。也就是说，同样是遍历一遍字符串 `s`，`int len = strlen(s); for(int i = 0; i < len; i++);` 的复杂度是 $O(n)$ 的；而 `for(int i = 0;i < strlen(s); i++);` 的复杂度是 $O(n^2)$ 的。后者的写法可能会导致 `TLE`。
2. `strlen` 函数的返回值是 `unsigned` 无符号整型，对于 `i <= strlen(str) - 1` 这种表达式，只要运算符的任一边出现了无符号类型的变量，则整个式子中的变量都会被隐式地转换为无符号类型变量进行计算。例如字符串 s，长度为 $|s| = 1$，则在无符号类型运算下有 strlen(s) - 2 = 4294967295，也就是说循环的范围将非常大，很可能导致 `TLE`。 所以，比较保险的方式是用 '\0' 判断是否访问到了字符串末尾，或者先用一个变量记录下 `strlen(s)` 的值。

示例程序 1

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

char s[100], t[100];

int main()
{
    int i, sum = 0;

    scanf("%s%s", s, t);
    for (i = 0; s[i] != '\0'; i++)
        if (s[i] == t[i]) sum++;
    printf("%d", sum);

    return 0;
}
```

示例程序 2

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

char s[100], t[100];

int main()
```

```c
{
    int i, sum = 0;

    scanf("%s%s", s, t);
    int len = strlen(s);
    for (i = 0; i < len; i++)
        if (s[i] == t[i]) sum++;
    printf("%d", sum);

    return 0;
}
```

B - 指针?

难度	考点
1	指针

题目分析

本题源自 PPT 最后的思考题，程小设因为向指针中传入了一个指针变量，导致变量的原值也能被函数访问到并改变，因而得到了错误的结果。将其修改成普通 `int` 型参量就可以正确通过本题。

示例程序

```c
#include <stdio.h>

int sum(int x, int py)
{
    x += py;
    py = x;
    return x;
}

int main()
{
    int a = 2, b = 3, c = 4;
    int sumnum;
    c = sum(a, b);
    int i;
    for (i = 1; i <= 100; i++)
    {
        sumnum += i;
    }
    sumnum += b;
    printf("%d %d %d %d %d",sumnum, a, b, c, i);
    return 0;
}
```

C - * 与 &

难度	考点
2	指针 基本概念

题目分析

这道题主要考察大家对取址运算符 `&` 和间接运算符 `*` 概念的理解。

因为 `data` 是一个 `int` 变量，一个显然的结论是：从右往左看第一个运算符号只能是 `&`，第二个运算符只能是 `*`，第三个运算符只能是 `&`……以此类推，最左侧的运算符决定了最终的结果是地址还是数值。

示例程序

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

char s[100];

int main() {
    int t;
    int i;
    int data;
    int len, flag;

    scanf("%d", &t);
    while (t--) {
        scanf("%d", &data);
        scanf("%s", s);
        len = strlen(s);
        flag = 1;
        for (i = len - 1; i >= 0; i--) {
            if ((len - i) % 2 == 1) {
                if (s[i] != '&') {
                    flag = 0;
                    break;
                }
            } else {
                if (s[i] != '*') {
                    flag = 0;
                    break;
                }
            }
        }
        if (!flag)
            printf("CE\n");
        else if (s[0] == '&')
            printf("Address\n");
        else
            printf("%d\n", data);
    }
```

```
38
39      return 0;
40  }
```

D - Digital reversal

难度	考点
4	字符串

题目分析

思路：使用一个变量 `tmp` 维护当前读入的数值，当读到符号 `.`，`/`，或 `%` 时，输出 `tmp` 对应反转后的值。特殊情况是对 `0` 的判定，符号前的 `tmp` 为零不会造成影响，因为读到符号时会反转输出，此时会正常输出零。换行符前的 `tmp` 为零时会有影响。由于只有 `%` 后面不需要输出数字，其它的都需要，因此可以使用 `fl` 标记判断一下属于哪种情况，是否要输出零。

示例程序

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

char c;
int fl;
unsigned long long tmp;

// 反转函数
void rev(unsigned long long n) {
    unsigned long long res = 0;
    while (n) {
        res = res * 10 + n % 10;
        n /= 10;
    }
    printf("%llu", res);
    fl = 1;
}

int main() {
    while ((c = getchar()) != EOF) {
        if (c == '\n') {
            if (tmp) rev(tmp);    // 如果 tmp > 0，反转
            if (!fl) printf("0"); // 标记判断，用于处理 tmp = 0，且后面需要数字输出的情况
            puts("");             // 换行
            tmp = 0;
            fl = 0;
        } else if ('0' <= c && c <= '9') {
            tmp = tmp * 10 + c - 48; // 维护tmp的值
        }
        if (c == '.') {
            rev(tmp); // 反转
            tmp = 0;
            fl = 0;
            printf(".");
        }
        if (c == '/') {
            rev(tmp); // 反转
```

```
            tmp = 0;
            fl = 0;
            printf("/");
        }
        if (c == '%') {
            rev(tmp); // 反转
            tmp = 0;  // 只有符号为%的时候不需给fl置0
            printf("%%");
        }
    }

    return 0;
}
```

E - 蒙德城吃鱼大赛

考点	难度
递推	4

题目分析

要吃 n 条鱼的方法总数，就是吃 $n-1$ 条鱼的方法总数、吃 $n-2$ 条鱼的方法总数和吃 $n-3$ 条鱼且上一次没有吃 3 条鱼的方法总数的和。

因此可以用一个二维数组分别保存吃n条鱼的时候上一次吃 3 条鱼和没有吃 3 条鱼的方法数。

注意：在 n 大于 45 的时候，方法总数会超过 `int` 的最大值，因此，需要设为 `long long` 型变量。

样例代码

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

int n;
long long a[51][2];

int main() {
    int i;

    a[1][0] = 1;
    a[2][0] = 2;
    a[3][0] = 3;
    a[1][1] = 0;
    a[2][1] = 0;
    a[3][1] = 1;
    for (i = 4; i <= 50; i++) {
        a[i][0] = a[i - 1][0] + a[i - 2][0] + a[i - 1][1] + a[i - 2][1];
        a[i][1] = a[i - 3][0];
    }
    while (~scanf("%d", &n)) {
        printf("%lld\n", a[n][0] + a[n][1]);
    }

    return 0;
}
```

F - *ptr == BUG

难度	考点
4	指针的常见错误用法

题目分析

题面中给出的代码一共有 10 处错误，每一处错误均在下述正确代码中标出。

在本部分对 10 处错误进行简短的解析：

- 1, 2： `swap` 函数不应用值传递，而应用地址传递，即用指针作为函数的参数。如果用值传递，则实际上传入函数的是实参值的拷贝，并不能修改到传入的原始变量，只有用地址传递才能利用这个函数修改变量。（下方的 `big_add` 函数的 `int *len` 参数也是如此）
- 3：指针的间接运算符 `*` 高于算术运算符的优先级，间接运算符只和其右侧的第一个变量或表达式结合。
- 4：当数组用作函数参数传递时等价于指针（地址）传递，在函数 `big_add` 中，参数 `s` 的地位等同于一个指针，而不再是调用时传入的数组名 `sum`。`sum` 是数组名，其具备一些和指针的相似之处，表示一个地址，但不可修改(这个"指针"自身不可以偏移，但其指向的值可以修改)，且可以用 `sizeof` 关键字获取到数组在内存中占用的字节数；而 `s` 相当于一个指针，其可以偏移，指向的值可以修改，用 `sizeof` 只能获取到这个指针作为一个变量本身占用的字节数，其原本代表数组的一些属性丧失了。
- 5：间接运算符的优先级低于自增运算符，指针 `len` 优先和自增运算符 `++` 结合，其效果为指针先移动，而后用间接运算符取出指针移动前指向的值（实际上是对指针指向的值进行自增），所以应加小括号。
- 6：指针类型与变量类型不匹配，不同类型的指针的"步长"是不同的，这里所谓的 "步长" 即为对这个指针进行 `++` / `--` / `+1` 等算术运算后得到的新地址相对原地址相差的字节数（等于这个指针的类型所代表的变量的字节数，例如 `char*` 的步长为 `char` 的字节数，为 1，而 `int*` 的步长为 4，是 `int` 变量的字节数）
- 7： `printf` 要输出变量的值，而不是变量的地址，所以应该用间接运算符取出指针指向的值。
- 8：野指针，一个指针必须指向一个有效的变量或者一段由 `malloc` 动态分配的内存，不可以未经初始化就直接使用。
- 9, 10：函数 `num_rev_to_str` 所需的第三个参数为数值，而不是指针，所以这里可以改成 `*pl1` 或者 `len1`。（这个错误和 第 7 处类似）

示例代码

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

#define MAXN 305

char s1[MAXN], s2[MAXN];
int a1[MAXN], a2[MAXN];
int sum[MAXN];

void swap(int *a, int *b) { // 1
    int t = *a; *a = *b; *b = t; // 2
}

```

```c
void num_rev_to_str(int *to, char *src, int l) {
    int i;
    for (i = 0; i < l; i++) {
        *(to + i) = *(src + l - i - 1) - '0'; // 3
    }
}

void big_add(int s[], int *len, int a[], int lena, int b[], int lenb) {
    int i;
    *len = lena > lenb ? lena : lenb;
    memset(s, 0, MAXN * sizeof(int)); // 4
    for (i = 0; i < *len; i++) {
        *s += *a + *b;
        if (*s >= 10) {
            *s -= 10;
            *(s + 1) += 1;
            if (i == *len - 1) (*len)++; // 5
        }
        s++, a++, b++;
    }
}

void rev_arr(int *l, int *r) { // 6
    while (l < r) {
        swap(l, r);
        l++, r--;
    }
}

void print_arr(int s[], int l) {
    int *p;
    for (p = s; p < s + l; p++) {
        printf("%d", *p); // 7
    }
    printf("\n");
}

int main()
{
    int len1, len2, len;
    int *pl1 = &len1, *pl2 = &len2, *pl = &len; // 8

    while (scanf("%s %s", s1, s2) != EOF) {
        *pl1 = strlen(s1);
        *pl2 = strlen(s2);
        memset(a1, 0, sizeof(a1));
        memset(a2, 0, sizeof(a2));
        num_rev_to_str(a1, s1, *pl1); // 9
        num_rev_to_str(a2, s2, *pl2); // 10
        big_add(sum, pl, a1, len1, a2, len2);
        rev_arr(sum, sum + len - 1);
        print_arr(sum, len);
    }

    return 0;
}
```

G - 山彦的振兴运动

难度	考点
5	字符串综合处理

题目分析

本题主要考察对字符数组的理解和处理。

在本题中，你不仅需要会使用 `<string.h>` 中的库函数，还需要理解那些函数的实现方式，自己实现略有变化的字符串比较等功能。

接下来，笔者将介绍本题的思路、注意点，并简述每个回答模式的处理方式。

- 本题的基本思路就是同时维护本条喊话与上一条喊话，再用 `if-else` 或其他结构依次判断是否满足条件。
  - 可以将所有喊话保存在二维数组里，再重新遍历一遍进行回应；
  - 可以读入一句处理一句，这种方法要注意保存上一条喊话。
- 每种回答模式的处理方式：
  - `"Yahoo"`。直接用 `strcmp` 即可。注意要完全相等，故不宜使用 `strstr` 等函数；
  - `+=`。这个有多种解决方案，可以顺序检查字符串里的字符，首先寻找 `'+'`，找到之后再寻找 `=`。注意，若使用 `strstr` 函数则要注意，不能用第一个 `'='` 出现在第一个 `'+'` 之后这种判断条件，否则形如 `"=+="` 的语句将被判断错误；
  - `\`。注意在拼接时要去掉 `'\'`。笔者的处理方式是：将上一句 `strcpy` → 将复制后的字符串的最后一个非 `'\0'` 字符变为 `'\0'`（删除 `'\'`） → 将本句 `strcat`；
  - `Repeat`。比较过程需要手动模拟 `strcmp` 函数，并将条件适当更改；
  - `\XXX/`。判断大小写后按要求生成新字符串即可。注意需要判断以下两个条件：出现小写字母、不出现大写字母。
- 注意点：
  - 如果不满足一个条件，则应继续判断是否满足下一个条件，而不是直接开始处理下一句话；
  - 对于需要改变字符串内容的模式（如大小写转换、`Repeat` 型、去掉 `'\'` 等），**不要直接在原字符串上动刀**，否则可能会在处理下一句时出现错误，比如连续两句以 `'\'` 结尾的句子，第二句在拼接时，应该保留第一句的 `'\'`；
  - 字符串应该以 `'\0'` 结尾。如果是自己手动模拟字符串处理过程，记得在结尾加上 `'\0'`。如果使用库函数，一定要弄清楚该库函数在操作之后，是否会在末尾补 `'\0'`（比如 `strncpy` 就不会补 `'\0'`）。

示例程序

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

int plusEqual(char s[]) {   // += 情况的判断
    int r = 0;
    for (int i = 0; i < strlen(s); i++) {
        if (s[i] == '+') {
            r = 1;
        }
```

```
        if (s[i] == '=' && r == 1) {
            return 1;
        }
    }
    return 0;
}

int cmp(char s1[], char s2[]) { // 两个字符串的比较，适用于 Repeat 情况判断
    if (strlen(s1) != strlen(s2)) {
        return 0;
    } else {
        int l = strlen(s1);
        int d = 0;
        for (int i = 0; i < l; i++) {
            d += s1[i] != s2[i];
        }
        return d <= 1;
    }
}

int allLower(char s[]) {    // \XXX/ 情况的判断
    int has = 0, m = 0;
    for (int i = 0; i < strlen(s); i++) {
        if (islower(s[i])) {
            has = 1;
        }
        if (isupper(s[i])) {
            m = 1;
        }
    }
    return has && !m;
}

int main() {
    char s0[25] = {0}, s1[25] = {0};    // s0 存储上一条喊话，s1 存储本条喊话
    char res[105][50] = {0};
    int i;
    for (i = 0; ~scanf("%s", s1); i++, strcpy(s0, s1)) {    // 用 for 写了循环，大家可以仔细看看这个循环条件里写了些什么
        if (strcmp(s1, "Yahoo!") == 0) {
            strcpy(res[i], "Yahuu!");
        } else if (plusEqual(s1)) {
            strcpy(res[i], "......");
        } else if (s1[strlen(s1) - 1] == '\\') {
            strcpy(res[i], s0);
            strcat(res[i], s1);
            res[i][strlen(res[i]) - 1] = 0;
        } else if (cmp(s1, s0)) {
            strcpy(res[i], "Repeat:");
            strcat(res[i], s0);
        } else if (allLower(s1)) {
            strcpy(res[i], "\\");
            strcat(res[i], s1);
            strcat(res[i], "/");
            for (int ti = 0; ti < strlen(res[i]); ti++) {
                if (islower(res[i][ti])) {
                    res[i][ti] = res[i][ti] - 'a' + 'A';
                }
```

```
            }
        } else {
            strcpy(res[i], s1);
        }
    }
    for (i--; i >= 0; i--) {    // 倒序输出
        puts(res[i]);
    }
    return 0;
}
```

H - 电影

难度	考点
5	字符串比较

题目分析

考虑到题目中影评字符数与垃圾电影数量并不大，本题完全不需要哈希表就可以解决问题。

具体做法如下：

- 对影评预处理，大写转为小写，遇到其他字符时就可以分割出一个新的电影名称。
- 构建结构体数组，用于存储每个合法电影名称以及该名称在影评中出现的次数。
- 将新分割出的电影名称保存到结构体数组中，并统计每个电影名称的出现次数。
- 如果新分割出的电影名称位于垃圾电影名单中，则不考虑将其保存到结构体数组中。
- 遍历最终得到的结构体数组，选择第一个出现次数最多的电影名称作为答案输出。

示例程序

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

char review[1100];
int n, movie_list_count;
char trash_list[110][20];

struct movie {
    char name[20];
    int cnt;
} movie_list[110];

int main() {

    gets(review);
    review[strlen(review)] = ' ';

    scanf("%d", &n);
    for (int i = 0;i < n;i++) {
        scanf("%s",trash_list[i]);
    }

    char temp_movie[20];
    memset(temp_movie,0,sizeof(temp_movie));

    for (int i = 0;i < strlen(review);i++) {

        if ((review[i] < 'a' || review[i] > 'z') && (review[i] < 'A' ||
review[i] > 'Z')) {
            if (strlen(temp_movie) != 0) {
                int flag = 0;

                for (int j = 0;j < movie_list_count;j++) {
```

```
                    if (strcmp(movie_list[j].name, temp_movie) == 0) {
                        movie_list[j].cnt++;
                        flag = 1;
                        break;
                    }
                }

                if (!flag) {

                    int whether_trash = 0;

                    for (int j = 0;j < n;j++) {
                        if (strcmp(trash_list[j], temp_movie) == 0) {
                            whether_trash = 1;
                            break;
                        }
                    }

                    if (!whether_trash) {
                        movie_list[movie_list_count].cnt = 1;
                        strcpy(movie_list[movie_list_count].name,
temp_movie);
                        movie_list_count++;
                    }

                }
                memset(temp_movie,0,sizeof(temp_movie));
            }
        }
        else {
            temp_movie[strlen(temp_movie)] = tolower(review[i]);
        }
    }

    int max_count = 0;
    int max_index;

    for (int i = 0;i < movie_list_count;i++) {
        if (movie_list[i].cnt > max_count) {
            max_count = movie_list[i].cnt;
            max_index = i;
        }
    }

    printf("%s", movie_list[max_index].name);

    return 0;
}
```

I - 最后的字符串排序

难度	考点
6	字符串排序

题目分析

设 a_i 为 s_i 右侧第一个和它不同的字母，如果没有，令 $a_i = -\infty$。

我们依次考虑 $S_1, S_2, \cdots, S_n$，对于 S_i，如果 $s_i > a_i$，那么它的字典序不大于 $S_{i+1}, S_{i+2}, \cdots, S_n$；如果 $s_i < a_i$，那么它的字典序不小于 $S_{i+1}, S_{i+2}, \cdots, S_n$。

因此，我们可以考虑如下的算法：

- 维护两个变量 l 和 r，初始令 $l = 1, r = n$。
- 从 1 到 n 枚举 i，如果 $s_i > a_i$，令 S_i 的排名为 l，并令 $l = l + 1$；如果 $s_i < a_i$，令 S_i 的排名为 r，并令 $r = r - 1$。

最后，因为题目要求相同字符串按下标从下到大排序，而数个相同的字符串对应的下标一定是连续的，因此我们可以将再将这些区间内的排名从小到大排序。观察上述算法，可以发现只需将 $s_i < a_i$ 区间的排名翻转即可。时间复杂度为 $O(n)$。

示例程序

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

#define maxn 100086

char s[maxn], a[maxn];
int rk[maxn];

int main()
{
    int t, n, tmp, i, j, l, r, x;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%s", s + 1), n = strlen(s + 1);
        for (i = n; i; i--)
        {
            if (s[i] == s[i + 1])
                a[i] = a[i + 1];
            else
                a[i] = s[i + 1];
        }
        l = 1, r = n;
        for (i = 1; i <= n; i++)
        {
            if (s[i] > a[i])
                rk[i] = l++;
            else
                rk[i] = r--;
```

```
        }
        x = 0;
        for (i = 1; i <= n + 1; i++)
        {
            if (s[i] ^ s[i - 1])
            {
                if (x && rk[x] > rk[i - 1])
                {
                    for (j = 1; j < i - x + 1 - j; j++)
                    {
                        tmp = rk[x - 1 + j];
                        rk[x - 1 + j] = rk[x - 1 + i - x + 1 - j];
                        rk[x - 1 + i - x + 1 - j] = tmp;
                    }
                }
                x = i;
            }
        }
        for (i = 1; i <= n; i++)
            printf("%d ", rk[i]);
        puts("");
    }
    return 0;
}
```

J - 二元多项式求偏导

难度	考点
7	字符串，函数应用

题目分析

对于这种综合性题目，首先进行全局规划，进而进行细节调整。

首先考虑总体程序规划。用三元组 (a, b, c) 表示单独某一项，按题目要求进行先求导后排序输出。

考虑具体实现的细节调整：

1. 对于第一项读入输出正项均省略正号：读入时可以以 `[+-]cx^ay^b` 作为每一项的通式读入，故可以预处理第一项（示例程序中的 `normalize`）；
2. 处理每一项时，合适地处理是否包含 x、y 的各种情况，并给 a, b, c 赋合适的值（示例程序中的 `parseFactor`）；
3. 将求导输出写成一到两个函数（合并对 x 和对 y 求导，使得程序可维护性、可读性较强），通过函数参数（比较函数、输出顺序）进行特殊规则的处理。示例程序中给出了函数指针的写法，可以参考学习。
4. 具体排序过程。学过结构体的同学可能很快想到用结构体排序，这里给出类似的、使用下标进行排序的方法，使用 `index` 数组进行排序。
5. 最后注意单独为 0 的情况和数据范围需要在合适的地方开 `long long`。

示例程序

```c
#include <stdio.h>
#include <stdlib.h>
typedef long long ll;

char input[10000];
char *p;
// 处理字符串的第一项，让每一项格式相同
char *normalize(char *str) {
    if (str[1] == '-')
        return str + 1;
    else {
        str[0] = '+';
        return str;
    }
}

#define N (100 + 5)

int a[N], b[N], c[N];
int curIndex;

int isDigit(char x) {
    return x >= '0' && x <= '9';
}

int getInteger() {
```

```
    int ret = 0;
    while (isDigit(*p))
        ret = ret * 10 + (*p++) - '0';
    return ret;
}
void parseFactor() {
    int A = 0, B = 0, C = 0;
    // 系数符号
    int sign = (*p++) == '+' ? 1 : -1;
    // 系数的绝对值
    C = isDigit(*p) ? getInteger() : 1;
    // 指数
    if ((*p) == 'x') {
        p++;
        if ((*p) == '^') p++, A = getInteger();
        else A = 1;
    }
    if ((*p) == 'y') {
        p++;
        if ((*p) == '^') p++, B = getInteger();
        else B = 1;
    }
    a[curIndex] = A;
    b[curIndex] = B;
    c[curIndex] = C * sign;
}

int da[N], db[N];
ll dc[N];
int index[N];

int cmpX(const void *px, const void *py) {
    int x = *(int *)px, y = *(int *)py;
    if (db[x] > db[y] || (db[x] == db[y] && da[x] > da[y])) return -1;
    return 1;
}

int cmpY(const void *px, const void *py) {
    int x = *(int *)px, y = *(int *)py;
    if (da[x] > da[y] || (da[x] == da[y] && db[x] > db[y])) return -1;
    return 1;
}

void printSingle(char x, int c) {
    if (c == 1)
        printf("%c", x);
    else if (c)
        printf("%c^%d", x, c);
}

int print(int a, int b, ll c, char x, char y, int flag, int first) {
    if (!c) return 0;
    if (!first && c > 0) printf("+");
    if (c < 0) c = -c, printf("-");
    if (!(c == 1 && (a || b)))
        printf("%lld", c);
    if (flag) {
        printSingle(x, a);
```

```
85          printSingle(y, b);
86      } else {
87          printSingle(y, b);
88          printSingle(x, a);
89      }
90      return 1;
91  }
92
93  // cx^ay^b 对 x 求导，输出按照 cmp 顺序输出，flag 表示先输出 x 还是先输出 y
94  void printDerivative(int *a, int *b, int *c, char x, char y, int (*cmp)
    (const void *x, const void *y), int flag) {
95      int i, first = 1;
96      // 求导，排序
97      for (i = 0; i < curIndex; i++) {
98          da[i] = a[i] - 1; // x^0 会被处理成 x^-1，但系数为 0，不会有影响
99          db[i] = b[i];
100         dc[i] = 1LL * c[i] * a[i];
101         index[i] = i;
102     }
103     qsort(index, curIndex, sizeof(int), cmp);
104     i = 0;
105     int nonzero = 0;
106     while (i < curIndex) {
107         int A = da[index[i]], B = db[index[i]];
108         ll C = dc[index[i]];
109         // 合并同类项
110         i++;
111         while (i < curIndex && (da[index[i]] == A && db[index[i]] == B))
112             C += dc[index[i++]];
113         // 输出
114         if (print(A, B, C, x, y, flag, first)) {
115             nonzero++;
116             first = 0;
117         }
118     }
119     if (!nonzero) printf("0");
120     puts("");
121 }
122
123 int main() {
124     int i;
125     scanf("%s", input + 1);
126     p = normalize(input);
127     while (*p) {
128         parseFactor();
129         curIndex++;
130     }
131     printDerivative(a, b, c, 'x', 'y', cmpX, 1);
132     printDerivative(b, a, c, 'y', 'x', cmpY, 0);
133     return 0;
134 }
```