A - 点积

难度	考点
1	一维数组

示例代码 1

```c
#include <stdio.h>
#define N 1005

int dot_vec(int[], int[], int);

int main()
{
    int a[N], b[N], ans, n, i;

    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = 0; i < n; i++)
        scanf("%d", &b[i]);
    printf("%d", dot_vec(a, b, n));

    return 0;
}

int dot_vec(int va[], int vb[], int n)
{
    int s = 0, i;

    for (i = 0; i < n; i++)
        s += va[i] * vb[i];

    return s;
}
```

示例代码 2

```c
#include <stdio.h>
#define N 1005

int dot_vec(int[], int[], int);

int main()
{
    int a[N], b, ans = 0, n, i;

    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
```

```
    for (i = 0; i < n; i++)
    {
        scanf("%d", &b);
        ans += a[i] * b;
    }
    printf("%d", ans);

    return 0;
}
```

B - 统计成绩

难度	考点
2	一维数组，输出格式

题目分析

本题题意较为直观，同学们只需要按照题目要求编写代码即可。

易错点分析

一个比较容易出错的地方是在计算平均值时，很容易出现整数除法而导致结果出错（即 `avg = sum / n`，但 `sum` 和 `n` 如果都为整数，则会进行的是整数除法而非浮点数除法）。

另外一种错法是，有的同学已经意识到 `avg` 应该是个浮点数，但还是不会正确使用强制类型转换。如果 `avg` 是 `double`，`sum` 和 `n` 均是 `int`，则 `avg = (double)(sum / n)` 的计算结果错误，正确的写法应该是 `avg = (double)sum / n`，或者直接定义 `sum` 为 `double`。

示例代码

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

#define N 1010

int x[N];

int main()
{
    int i, n;
    int mn = 1001, mx = -1;
    double sum = 0, var = 0, avg;

    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &x[i]);
        if (x[i] > mx)
            mx = x[i];
        if (x[i] < mn)
            mn = x[i];
        sum += x[i];
    }
    avg = sum / n;
    for (i = 0; i < n; i++)
    {
        var += pow(x[i] - avg, 2);
    }
    var /= n;
    printf("%-5d%-5d%-8.2f%-10.2f", mx, mn, avg, var);

    return 0;
```

```
33 }
```

C - Quick Brown Fox

难度	考点
2	字符串处理

题目分析

本题的题意较为简单，为了检验 26 个字符是否出现，可以开一个 `vst[x]` 数组，标记字母 `x` 是否已经出现。对字符串的每个英文大小写字母都处理过后，统计 `vst` 数组中恰有 26 个出现的元素即可。

需要注意的是，由于本题有多组数据，因此每次都需要清空 `vst` 数组，避免上一组数据的结果对本次计算造成影响。

示例程序

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

#define SUM_OF_ALPHABET (26 + 5)
#define N (100 + 5)

// 记录字符是否出现
int vst[SUM_OF_ALPHABET];

int main()
{
    char s[N];
    int i, len, cnt;

    while (gets(s) != NULL)
    {
        for (i = 0; i < SUM_OF_ALPHABET; i++)
            vst[i] = 0;

        len = strlen(s); // 获取字符串长度
        for (i = 0; i < len; i++)
        {
            // 小写字母
            if (s[i] >= 'a' && s[i] <= 'z')
                vst[s[i] - 'a'] = 1;
            // 大写字母
            if (s[i] >= 'A' && s[i] <= 'Z')
                vst[s[i] - 'A'] = 1;
        }

        cnt = 0;
        for (int i = 0; i < 26; i++)
            cnt += vst[i];
        // 判断 26 个字母是否都出现了
        printf(cnt == 26 ? "Yes\n" : "No\n");
    }

```

```
    return 0;
}
```

D - 最长字符串

难度	考点
2	二维字符数组

题目分析

本题由 PPT 中的例题改编而来，考察了二维字符数组的应用。

易错点提示

- 需要使用 `gets` 函数读入一整行字符串。
- 输出的时候需要倒序输出。
- 大数组需要开全局数组，开在 `main` 函数的里面可能会 REG。

示例程序

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

char arr[1010][20010];
int max_len = 0;

int main()
{
    // top用于标记输入字符串的位置
    int i, top = 0;

    // while循环输入多组数据
    while (gets(arr[++top]) != NULL)
    {
        // 维护 max_len 的值
        if (strlen(arr[top]) > max_len)
        {
            max_len = strlen(arr[top]);
        }
    }
    // 输出 max_len
    printf("%d\n", max_len);
    // 倒序输出
    for (i = top; i >= 1; --i)
    {
        if (strlen(arr[i]) == max_len)
        {
            printf("%s\n", arr[i]);
        }
    }

    return 0;
}
```

E - 文法分析

难度	考点
3	字符串应用

题目分析

- 根据题目描述进行分析后可将题目简化为如下内容：
  - 给定两个字符串 a 和 b，选择二者中较长的字符串以第 n 个字符为界，将其分割成两部分：`<主语>` 和 `<宾语>`。再将较短的字符串作为 `<谓语>` 插进两部分的中间获得最终的 `<语句>`。
  - 输出最终获得的 `<语句>` 字符串。
- **注意**：如果考虑将 `<语句>` 字符串存进新的字符串，需要根据具体数据分析下标具体情况。

示例程序

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

char a[1100], b[1100], s[2200];
int l1, l2, n, count[30];

int main()
{
    int i;

    gets(a);
    gets(b);
    scanf("%d", &n);
    l1 = strlen(a);
    l2 = strlen(b);
    if (l1 > l2)
    {
        for (i = 0; i <= n; i++)
        {
            s[i] = a[i];
        }
        for (i = 0; i < l2; i++)
        {
            s[i + n + 1] = b[i];
        }
        for (i = n + 1; i < l1; i++)
        {
            s[i - n - 1 + n + l2 + 1] = a[i];
        }
    }
    else
    {
        for (i = 0; i <= n; i++)
        {
            s[i] = b[i];
        }
```

```
        for (i = 0; i < l1; i++)
        {
            S[i + n + 1] = a[i];
        }
        for (i = n + 1; i < l2; i++)
        {
            S[i - n - 1 + n + l1 + 1] = b[i];
        }
    }
    printf("%s\n", S);

    return 0;
}
```

F - 卷积运算

知识点	难度
二维数组、循环	3

题目分析

本题主要考察二维数组的输入、计算、输出，在计算过程中，需要用到四重循环，本题未设坑，总体来说只要细心，按照题干描述按部就班写代码，难度不大。

样例程序

```c
#include <stdio.h>

int f[100][100], h[100][100], g[100][100];

int main()
{
    int i, j, k, l, m1, n1, m2, n2;

    scanf("%d%d%d%d", &m1, &n1, &m2, &n2);
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < n1; j++)
        {
            scanf("%d", &f[i][j]);
        }
    }
    for (i = 0; i < m2; i++)
    {
        for (j = 0; j < n2; j++)
        {
            scanf("%d", &h[i][j]);
        }
    }
    for (i = 0; i < m1 - m2 + 1; i++)
    {
        for (j = 0; j < n1 - n2 + 1; j++)
        {
            for (k = 0; k < m2; k++)
            {
                for (l = 0; l < n2; l++)
                {
                    g[i][j] += f[i + k][j + l] * h[k][l];
                }
            }
        }
    }
    for (i = 0; i < m1 - m2 + 1; i++)
    {
        for (j = 0; j < n1 - n2 + 1; j++)
        {
```

```
            printf("%d ", g[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

G - Woof! Woof!

难度	考点
4	字符数组的读入、查找与替换

本题考查字符数组的一系列使用，同学们既可以自己写一个循环来查找（代码 1、2），可以使用字符串函数（代码放在示例 3 中，涉及到一点指针的知识，学有余力的同学可以进行学习）。

注意，`strlen()` 函数的返回值是 `size_t`（一个无符号类型，简单看可以认为等价于 `unsigned int`）无符号的，所以使用 `strlen(str)-3` 可能会有溢风险！比如输入字符串 `str="a"`，那么 `strlen(str)-3=(unsigned)1-3=4294967294`（还记得 `Overflow` 那道题吗？）。这个地方需要谨慎处理！

希望同学能学习一下 HINT 中提供的字符串读入方式，并在课外探索 `getchar()` / `gets()` / `scanf("%c", c)` / `scanf("%s", s)` 这几个读入字符/字符串函数的区别。

示例代码 1

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

char str[1000 + 5];

int main()
{
    int i, len, ans;

    while (gets(str) != NULL) // 使用 gets 读入一整行字符串
    {
        len = strlen(str), ans = 0;
        // 使用 strlen 函数求字符串长度，注意要先 include string.h 这个文件
        for (i = 0; i < len - 3; ++i)
        {
            if ((str[i] == 'w' || str[i] == 'W') && // 合理换行能让代码更清晰
                (str[i + 1] == 'o' || str[i + 1] == 'O') &&
                (str[i + 2] == 'o' || str[i + 2] == 'O') &&
                (str[i + 3] == 'f' || str[i + 3] == 'F'))
            {
                ++ans;
                str[i] = 'W';
                str[i + 1] = 'O';
                str[i + 2] = 'O';
                str[i + 3] = 'F';
            }
        }
        printf("%d\n%s\n", ans, str);
    }

    return 0;
}
```

示例代码 2

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

char str[1000 + 5];

int main()
{
    int i, len, ans;
    while (gets(str) != NULL)
    {
        len = strlen(str), ans = 0;

        for (i = 0; i < len - 3; ++i)
        {
            if (toupper(str[i]) == 'W' &&
                toupper(str[i + 1]) == 'O' &&
                toupper(str[i + 2]) == 'O' &&
                toupper(str[i + 3]) == 'F')
            { // 使用 toupper 函数将字符转换为大写，从而减小代码长度，注意要先 include ctype.h 这个文件
                ++ans;
                str[i] = 'W';
                str[i + 1] = 'O';
                str[i + 2] = 'O';
                str[i + 3] = 'F';
            }
        }
        printf("%d\n%s\n", ans, str);
    }

    return 0;
}
```

示例代码 3

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

char str1[1000], str2[1000];

int main()
{
    int i, cnt;

    while (gets(str1) != NULL)
    {
        cnt = 0;
        strcpy(str2, str1);
        for (i = 0; str1[i]; i++)
            str2[i] = tolower(str2[i]); // 预处理，全部转换为小写; str2[i] 放在条件中等价于 str2[i] != '\0'
```

```
17
18          char *ps1, *ps2; // 定义两个字符指针
19          for (ps1 = str2; (ps2 = strstr(ps1, "woof")); ps1 = ps2 + 4)
20          {                                                   // ps1 表示查找的起始位置，ps2 表示找到的位置
21              strncpy(str1 + (ps2 - str2), "WOOF", 4); // (p - str1) 是找到的位置
22              ++cnt;
23          }
24
25          printf("%d\n%s\n", cnt, str1);
26      }
27
28      return 0;
29  }
```

H - Minimize The Function

难度	考点
4	二分查找

题目分析

本题只要按照高中导数题的思路处理就可以了，一个连续函数在区间上的极值点要么是导函数的零点，要么是区间的端点。在本题中 $x \to \frac{1}{2}$ 时 $L(x) \to +\infty$，故最小值只可能在 $x = 1$ 处或者导函数零点处。对函数求导可得：

$$L^{'}(X) = -\frac{2a}{(2x-1)\ln(2)} + 2\lambda x$$

再求一次导：

$$L^{''}(X) = \frac{4a}{(2x-1)^2\ln(2)} + 2\lambda$$

由于 $a, \lambda > 0$ 故 $L^{'}(x)$ 单调递增，易于发现当 $x \to \frac{1}{2}$ 时 $L^{'}(x) \to -\infty$。

所以如果 $L^{'}(1) \leq 0$ 则 $L(x)$ 在区间上单调递减，最小值在 $L(1)$处取到。

若不然，那么有两种方法可以进行处理。

第一种方法是本题意图的考点，可以采用二分查找的方法找到单调函数的根。一开始令 l 是一个接近于 0.5 的数，在本代码中取 0.501，大家可以自行计算验证此时导数必定小于 0，故在二分过程中恒有 $L'(l) < 0$，再取 r 为 1，有 $L'(r) > 0$。随后令 $mid = \frac{l+r}{2}$，并不断计算$L'(mid)$：

- 若$L'(mid) < 0$，则把 l 更新为 mid
- 若 $L'(mid) > 0$，则把 r 更新为 mid
- 若$L'(mid) = 0$，则直接把 mid 作为最小值点

不断重复以上过程，直到 $r - l$ 的大小已经很小，达到题目中的精度要求即可结束循环。并输出最小值点和对应的函数取值（其实第一种情况可以合并到第二种内，想一想为什么？）。

第二种方法是通分可以得到一个分子是二次函数的形式，可以通过求根公式的方式求出它的具体的解析解。但要判断清楚解的范围是否在所要求的定义域内。

本题的常见错误有：考虑最小值时忽视区间范围（常见于二次方程解法），使用二分时不能设置合适的终止条件（注意导函数在给定区间上未必有零点），导致 TLE，以及设置中止条件精度不足，使用小步长步进导致 TLE 等。

示例代码 1

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

const double eps = 1e-8;

double f(double x, double a, double lambda)
{
```

```
    return -a * log2(2 * x - 1) + lambda * x * x;
}

double d(double x, double a, double lambda)
{
    return -a * 2 / (2 * x - 1) / log(2) + 2 * lambda * x;
}

int main()
{
    double a, lambda;
    double l, r, mid;

    while (~scanf("%lf%lf", &a, &lambda))
    {
        if (d(1, a, lambda) < eps)
        {
            printf("%.10f ", f(1, a, lambda));
            printf("%.10f\n", 1.0);
            continue;
        }
        else
        {
            l = 0.5 + 1e-2;
            r = 1;
            while ((r - l) > 1e-7)
            {
                mid = l + r;
                mid /= 2;
                if (d(mid, a, lambda) < -eps)
                    l = mid;
                else if (d(mid, a, lambda) > eps)
                    r = mid;
                else
                {
                    break;
                }
            }
            printf("%.10f ", f(mid, a, lambda));
            printf("%.10f\n", mid);
        }
    }

    return 0;
}
```

示例代码 2

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

int main()
{
    double a, b, n, i, j, l;

    while (~scanf("%lf%lf", &a, &b))
```

```
    {
        n = a / b / log(2);
        i = (sqrt(8 * n + 1) + 1) / 4;
        if (i > 1)
        {
            i = 1;
        }
        j = -a * log(2 * i - 1) / log(2) + b * i * i;
        printf("%.8lf %.8lf\n", j, i);
    }

    return 0;
}
```

I - 基因重组

难度	考点
5	排序，二维数组

题目分析

对题目要求最小化的式子进行变形：

$$\sum_{i=1}^{n}(a_i - b_{p_i})^2$$
$$= \sum_{i=1}^{n} a_i^2 + \sum_{i=1}^{n} b_i^2 - 2\sum_{i=1}^{n} a_i b_{p_i}$$

不妨假设 a 和 b 数组已经按照升序排好了，则由 Hint 中给出的不等式可知，当 $p_i = i$ 时 $\sum_{i=1}^{n} a_i b_{p_i}$ 取得最大值，上式取得最小值，因此只需要让 a_i 与 b_i 排序后，分别按大小一一配对即可。

为了输出编号，排序过程需要保留其初始序号，这里可以采用一个二维数组记录碱基，其第二维为 0 时表示碱基能量，为 1 时表示初始序号，并在排序交换时一起交换即可（具体参见代码）。由于数组已经排好序，因此最后计算 p_i 的值可以采用 `p[a[i][1]] = b[i][1]` 的方式进行赋值（可以想一想为什么是这样，注意 `a[]` 和 `b[]` 是已经有序的）。

示例代码

```c
#include <stdio.h>

typedef long long ll;

#define N (2000 + 5)

void bubble_sort(int arr[][2], int n)
{
    int i, j;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n - i; j++)
            if (arr[j][0] > arr[j + 1][0])
            {
                int val = arr[j][0], id = arr[j][1];
                arr[j][0] = arr[j + 1][0];
                arr[j][1] = arr[j + 1][1];
                arr[j + 1][0] = val;
                arr[j + 1][1] = id;
            }
}

// 二维数组第二维分别是能量与初始编号
int a[N][2], b[N][2];

int main()
{
    int n, i;
    ll ans = 0;
```

```
29
30      scanf("%d", &n);
31      for (i = 1; i <= n; i++)
32      {
33          scanf("%d", &a[i][0]);
34          a[i][1] = i;
35      }
36      for (i = 1; i <= n; i++)
37      {
38          scanf("%d", &b[i][0]);
39          b[i][1] = i;
40      }
41      bubble_sort(a, n);
42      bubble_sort(b, n);
43
44      // 记录编号的数组
45      static int p[N];
46      for (i = 1; i <= n; i++)
47      {
48          ans += 1LL * (a[i][0] - b[i][0]) * (a[i][0] - b[i][0]);
49          // a 中编号为 a[i][1] 的碱基，其对应的 b 中碱基编号为 b[i][1]
50          // 用 p 数组来保存这个对应关系
51          p[a[i][1]] = b[i][1];
52      }
53
54      printf("%lld\n", ans);
55      for (i = 1; i <= n; i++)
56          printf("%d ", p[i]);
57
58      return 0;
59  }
```

J - Render the Fraction!!!

难度	考点
6	字符串综合处理

题目解析

用三个数组保存三行答案，检测 `\frac` 的出现并将两部分保存下来即可，详见代码。

记得特判不出现 `\frac` 的情况。

示例代码

```
#include <stdio.h>
#define maxn 2333

char up[maxn], dn[maxn];
char res[3][maxn];
int cur;
int is_in_frac;
int is_no_effect;
int is_lower_part;
int n_up, n_dn;

void render_frac()
{
    // determine the length of the line
    int len = n_up > n_dn ? n_up : n_dn;
    int i, l_up, l_dn;

    l_up = (len - n_up) / 2;
    l_dn = (len - n_dn) / 2;
    // fill up needed space characters
    for (i = cur + l_up; i-- && res[0][i] == 0;)
        res[0][i] = ' ';
    for (i = cur + l_dn; i-- && res[2][i] == 0;)
        res[2][i] = ' ';
    // draw the parts
    for (i = 0; i < n_up; ++i)
        res[0][cur + l_up + i] = up[i];
    for (i = 0; i < len; ++i)
        res[1][cur + i] = '-';
    for (i = 0; i < n_dn; ++i)
        res[2][cur + l_dn + i] = dn[i];
}

int main()
{
    char ch;

    while (1)
    {
        ch = getchar();
```

```
        if (ch == EOF || ch == '\n')
            break;
        else if (ch == '\\')
        {
            is_in_frac = 1;
            is_no_effect = 1;
            n_up = n_dn = 0;
        }
        else if (ch == '{')
        {
            is_no_effect = 0;
        }
        else if (ch == '}')
        {
            if (is_lower_part)
            {
                render_frac();
                is_lower_part = 0;
                is_in_frac = 0;
                cur += (n_up > n_dn ? n_up : n_dn);
            }
            else
                is_lower_part = 1;
        }
        else
        {
            if (is_in_frac)
            {
                if (!is_no_effect)
                {
                    if (!is_lower_part)
                        up[n_up++] = ch;
                    else
                        dn[n_dn++] = ch;
                }
            }
            else
            {
                res[1][cur++] = ch;
            }
        }
    }
    if (strlen(res[0]) > 0 && strlen(res[2]) > 0)
        printf("%s\n%s\n%s", res[0], res[1], res[2]);
    else
        printf("%s", res[1]);

    return 0;
}
```