

A - 点积

难度	考点
1	一维数组

示例代码 1

```
1  #include <stdio.h>
2  #define N 1005
3
4  int dot_vec(int[], int[], int);
5
6  int main()
7  {
8      int a[N], b[N], ans, n, i;
9
10     scanf("%d", &n);
11     for (i = 0; i < n; i++)
12         scanf("%d", &a[i]);
13     for (i = 0; i < n; i++)
14         scanf("%d", &b[i]);
15     printf("%d", dot_vec(a, b, n));
16
17     return 0;
18 }
19
20 int dot_vec(int va[], int vb[], int n)
21 {
22     int s = 0, i;
23
24     for (i = 0; i < n; i++)
25         s += va[i] * vb[i];
26
27     return s;
28 }
```

示例代码 2

```
1  #include <stdio.h>
2  #define N 1005
3
4  int dot_vec(int[], int[], int);
5
6  int main()
7  {
8      int a[N], b, ans = 0, n, i;
9
10     scanf("%d", &n);
11     for (i = 0; i < n; i++)
12         scanf("%d", &a[i]);
```

```
13     for (i = 0; i < n; i++)
14     {
15         scanf("%d", &b);
16         ans += a[i] * b;
17     }
18     printf("%d", ans);
19
20     return 0;
21 }
```

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

```
1  #include <math.h>
2  #include <stdio.h>
3
4  #define N 1010
5
6  int x[N];
7
8  int main()
9  {
10     int i, n;
11     int mn = 1001, mx = -1;
12     double sum = 0, var = 0, avg;
13
14     scanf("%d", &n);
15     for (i = 0; i < n; i++)
16     {
17         scanf("%d", &x[i]);
18         if (x[i] > mx)
19             mx = x[i];
20         if (x[i] < mn)
21             mn = x[i];
22         sum += x[i];
23     }
24     avg = sum / n;
25     for (i = 0; i < n; i++)
26     {
27         var += pow(x[i] - avg, 2);
28     }
29     var /= n;
30     printf("%-5d%-5d%-8.2f%-10.2f", mx, mn, avg, var);
31
32     return 0;
```


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

```
1  #include <stdio.h>
2  #include <string.h>
3
4  #define SUM_OF_ALPHABET (26 + 5)
5  #define N (100 + 5)
6
7  // 记录字符是否出现
8  int vst[SUM_OF_ALPHABET];
9
10 int main()
11 {
12     char s[N];
13     int i, len, cnt;
14
15     while (gets(s) != NULL)
16     {
17         for (i = 0; i < SUM_OF_ALPHABET; i++)
18             vst[i] = 0;
19
20         len = strlen(s); // 获取字符串长度
21         for (i = 0; i < len; i++)
22         {
23             // 小写字母
24             if (s[i] >= 'a' && s[i] <= 'z')
25                 vst[s[i] - 'a'] = 1;
26             // 大写字母
27             if (s[i] >= 'A' && s[i] <= 'Z')
28                 vst[s[i] - 'A'] = 1;
29         }
30
31         cnt = 0;
32         for (int i = 0; i < 26; i++)
33             cnt += vst[i];
34         // 判断 26 个字母是否都出现了
35         printf(cnt == 26 ? "Yes\n" : "No\n");
36     }
37 }
```

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

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

```
1  #include <stdio.h>
2  #include <string.h>
3
4  char arr[1010][20010];
5  int max_len = 0;
6
7  int main()
8  {
9      // top用于标记输入字符串的位置
10     int i, top = 0;
11
12     // while循环输入多组数据
13     while (gets(arr[++top]) != NULL)
14     {
15         // 维护 max_len 的值
16         if (strlen(arr[top]) > max_len)
17         {
18             max_len = strlen(arr[top]);
19         }
20     }
21     // 输出 max_len
22     printf("%d\n", max_len);
23     // 倒序输出
24     for (i = top; i >= 1; --i)
25     {
26         if (strlen(arr[i]) == max_len)
27         {
28             printf("%s\n", arr[i]);
29         }
30     }
31
32     return 0;
33 }
```

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

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char a[1100], b[1100], s[2200];
7      int l1, l2, n, count[30];
8      int i;
9
10     gets(a);
11     gets(b);
12     scanf("%d", &n);
13     l1 = strlen(a);
14     l2 = strlen(b);
15     if (l1 > l2)
16     {
17         for (i = 0; i <= n; i++)
18         {
19             s[i] = a[i];
20         }
21         for (i = 0; i < l2; i++)
22         {
23             s[i + n + 1] = b[i];
24         }
25         for (i = n + 1; i < l1; i++)
26         {
27             s[i - n - 1 + n + l2 + 1] = a[i];
28         }
29     }
30     else
31     {
32         for (i = 0; i <= n; i++)
33         {
34             s[i] = b[i];
35         }
36         for (i = 0; i < l1; i++)
```



```
37     {
38         s[i + n + 1] = a[i];
39     }
40     for (i = n + 1; i < 12; i++)
41     {
42         s[i - n - 1 + n + 11 + 1] = b[i];
43     }
44 }
45 printf("%s\n", s);
46
47 return 0;
48 }
```

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

```
1  #include <stdio.h>
2
3  int f[100][100], h[100][100], g[100][100];
4
5  int main()
6  {
7      int i, j, k, l, m1, n1, m2, n2;
8
9      scanf("%d%d%d%d", &m1, &n1, &m2, &n2);
10     for (i = 0; i < m1; i++)
11     {
12         for (j = 0; j < n1; j++)
13         {
14             scanf("%d", &f[i][j]);
15         }
16     }
17     for (i = 0; i < m2; i++)
18     {
19         for (j = 0; j < n2; j++)
20         {
21             scanf("%d", &h[i][j]);
22         }
23     }
24     for (i = 0; i < m1 - m2 + 1; i++)
25     {
26         for (j = 0; j < n1 - n2 + 1; j++)
27         {
28             for (k = 0; k < m2; k++)
29             {
30                 for (l = 0; l < n2; l++)
31                 {
32                     g[i][j] += f[i + k][j + l] * h[k][l];
33                 }
34             }
35         }
36     }
37     for (i = 0; i < m1 - m2 + 1; i++)
38     {
39         for (j = 0; j < n1 - n2 + 1; j++)
40         {
```

```
41         printf("%d ", g[i][j]);
42     }
43     printf("\n");
44 }
45
46 return 0;
47 }
```

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

```
1  #include <stdio.h>
2  #include <string.h>
3
4  char str[1000 + 5];
5
6  int main()
7  {
8      int i, len, ans;
9
10     while (gets(str) != NULL) // 使用 gets 读入一整行字符串
11     {
12         len = strlen(str), ans = 0;
13         // 使用 strlen 函数求字符串长度，注意要先 include string.h 这个文件
14         for (i = 0; i < len - 3; ++i)
15         {
16             if ((str[i] == 'w' || str[i] == 'W') && // 合理换行能让代码更清晰
17                 (str[i + 1] == 'o' || str[i + 1] == 'O') &&
18                 (str[i + 2] == 'o' || str[i + 2] == 'O') &&
19                 (str[i + 3] == 'f' || str[i + 3] == 'F'))
20             {
21                 ++ans;
22                 str[i] = 'w';
23                 str[i + 1] = 'o';
24                 str[i + 2] = 'o';
25                 str[i + 3] = 'f';
26             }
27         }
28         printf("%d\n%s\n", ans, str);
29     }
30
31     return 0;
32 }
```

示例代码 2

```
1  #include <ctype.h>
2  #include <stdio.h>
3  #include <string.h>
4
5  char str[1000 + 5];
6
7  int main()
8  {
9      int i, len, ans;
10     while (gets(str) != NULL)
11     {
12         len = strlen(str), ans = 0;
13
14         for (i = 0; i < len - 3; ++i)
15         {
16             if (toupper(str[i]) == 'W' &&
17                 toupper(str[i + 1]) == 'O' &&
18                 toupper(str[i + 2]) == 'O' &&
19                 toupper(str[i + 3]) == 'F')
20             { // 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
21                 ++ans;
22                 str[i] = 'W';
23                 str[i + 1] = 'O';
24                 str[i + 2] = 'O';
25                 str[i + 3] = 'F';
26             }
27         }
28         printf("%d\n%s\n", ans, str);
29     }
30
31     return 0;
32 }
```

示例代码 3

```
1  #include <ctype.h>
2  #include <stdio.h>
3  #include <string.h>
4
5  char str1[1000], str2[1000];
6
7  int main()
8  {
9      int i, cnt;
10
11     while (gets(str1) != NULL)
12     {
13         cnt = 0;
14         strcpy(str2, str1);
15         for (i = 0; str1[i]; i++)
16             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     {
21         // ps1 表示查找的起始位
22         // ps2 表示找到的位置
23         strncpy(str1 + (ps2 - str2), "WOOF", 4); // (p - str1) 是找到的位
24         // 置
25         ++cnt;
26     }
27     printf("%d\n%s\n", cnt, str1);
28 }
29
30     return 0;
31 }
```

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 $x \rightarrow \frac{1}{2}$ 时 $L(x) \rightarrow +\infty$ ，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 $x \rightarrow \frac{1}{2}$ 时 $L'(x) \rightarrow -\infty$ 。

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

```
1 #include <math.h>
2 #include <stdio.h>
3
4 const double eps = 1e-8;
5
6 double f(double x, double a, double lambda)
7 {
```

```

8     return -a * log2(2 * x - 1) + lambda * x * x;
9 }
10
11 double d(double x, double a, double lambda)
12 {
13     return -a * 2 / (2 * x - 1) / log(2) + 2 * lambda * x;
14 }
15
16 int main()
17 {
18     double a, lambda;
19     double l, r, mid;
20
21     while (~scanf("%lf%lf", &a, &lambda))
22     {
23         if (d(l, a, lambda) < eps)
24         {
25             printf("%.10f ", f(l, a, lambda));
26             printf("%.10f\n", 1.0);
27             continue;
28         }
29         else
30         {
31             l = 0.5 + 1e-2;
32             r = 1;
33             while ((r - l) > 1e-7)
34             {
35                 mid = l + r;
36                 mid /= 2;
37                 if (d(mid, a, lambda) < -eps)
38                     l = mid;
39                 else if (d(mid, a, lambda) > eps)
40                     r = mid;
41                 else
42                 {
43                     break;
44                 }
45             }
46             printf("%.10f ", f(mid, a, lambda));
47             printf("%.10f\n", mid);
48         }
49     }
50
51     return 0;
52 }

```

示例代码 2

```

1  #include <math.h>
2  #include <stdio.h>
3
4  int main()
5  {
6      double a, b, n, i, j, l;
7
8      while (~scanf("%lf%lf", &a, &b))

```



```
9      {
10          n = a / b / log(2);
11          i = (sqrt(8 * n + 1) + 1) / 4;
12          if (i > 1)
13          {
14              i = 1;
15          }
16          j = -a * log(2 * i - 1) / log(2) + b * i * i;
17          printf("%.8lf %.8lf\n", j, i);
18      }
19
20      return 0;
21  }
```

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：

$$\begin{aligned} & \sum_{i=1}^n (a_i - b_{p_i})^2 \\ &= \sum_{i=1}^n a_i^2 + \sum_{i=1}^n b_{p_i}^2 - 2 \sum_{i=1}^n a_i b_{p_i} \end{aligned}$$

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

```
1  #include <stdio.h>
2
3  typedef long long ll;
4
5  #define N (2000 + 5)
6
7  void bubble_sort(int arr[][2], int n)
8  {
9      int i, j;
10     for (i = 1; i <= n; i++)
11         for (j = 1; j <= n - i; j++)
12             if (arr[j][0] > arr[j + 1][0])
13             {
14                 int val = arr[j][0], id = arr[j][1];
15                 arr[j][0] = arr[j + 1][0];
16                 arr[j][1] = arr[j + 1][1];
17                 arr[j + 1][0] = val;
18                 arr[j + 1][1] = id;
19             }
20 }
21
22 // 二维数组第二维分别是能量与初始编号
23 int a[N][2], b[N][2];
24
25 int main()
26 {
27     int n, i;
28     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

```
1  #include <stdio.h>
2  #define maxn 2333
3
4  char up[maxn], dn[maxn];
5  char res[3][maxn];
6  int cur;
7  int is_in_frac;
8  int is_no_effect;
9  int is_lower_part;
10 int n_up, n_dn;
11
12 void render_frac()
13 {
14     // determine the length of the line
15     int len = n_up > n_dn ? n_up : n_dn;
16     int i, l_up, l_dn;
17
18     l_up = (len - n_up) / 2;
19     l_dn = (len - n_dn) / 2;
20     // fill up needed space characters
21     for (i = cur + l_up; i-- && res[0][i] == 0;)
22         res[0][i] = ' ';
23     for (i = cur + l_dn; i-- && res[2][i] == 0;)
24         res[2][i] = ' ';
25     // draw the parts
26     for (i = 0; i < n_up; ++i)
27         res[0][cur + l_up + i] = up[i];
28     for (i = 0; i < len; ++i)
29         res[1][cur + i] = '-';
30     for (i = 0; i < n_dn; ++i)
31         res[2][cur + l_dn + i] = dn[i];
32 }
33
34 int main()
35 {
36     char ch;
37
38     while (1)
39     {
40         ch = getchar();
```

```

41     if (ch == EOF || ch == '\n')
42         break;
43     else if (ch == '\\')
44     {
45         is_in_frac = 1;
46         is_no_effect = 1;
47         n_up = n_dn = 0;
48     }
49     else if (ch == '{')
50     {
51         is_no_effect = 0;
52     }
53     else if (ch == '}')
54     {
55         if (is_lower_part)
56         {
57             render_frac();
58             is_lower_part = 0;
59             is_in_frac = 0;
60             cur += (n_up > n_dn ? n_up : n_dn);
61         }
62         else
63             is_lower_part = 1;
64     }
65     else
66     {
67         if (is_in_frac)
68         {
69             if (!is_no_effect)
70             {
71                 if (!is_lower_part)
72                     up[n_up++] = ch;
73                 else
74                     dn[n_dn++] = ch;
75             }
76         }
77         else
78         {
79             res[1][cur++] = ch;
80         }
81     }
82 }
83 if (strlen(res[0]) > 0 && strlen(res[2]) > 0)
84     printf("%s\n%s\n%s", res[0], res[1], res[2]);
85 else
86     printf("%s", res[1]);
87
88 return 0;
89 }

```