第五章　数组

把同类数据排成一列

HGNU-ACM 实验室算法组

基础讲解 43 页；17 道例题逐题推导；47 道练习索引。

从下标、遍历走向矩阵与字符串，代码全部逐行重排。





本章目录

数组基础：声明、下标与容器

一维数组例题

二维数组例题

字符数组与字符串例题

全部练习与 OJ 索引

本章怎么学

1. 先看基础：声明、下标、越界、`vector`、二维数组、字符串，每个知识点都有小例子。
2. 再动手：把每个小例子抄进编译器，改一两个数，观察输出变化。
3. 最后做题：例题先自己推样例，再看完整程序；练习按一维、二维、字符串的顺序做。

代码约定

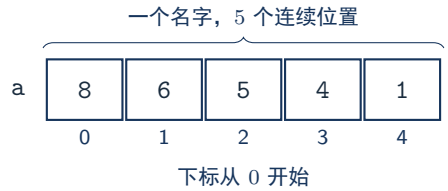
本章代码统一使用 C++17 与 `<bits/stdc++.h>`；每条语句单独一行，缩进 4 个空格，关键行写注释。

本页要点

看懂不等于会写：合上课件，能把“读入 n 个数再遍历”完整写出来，才算掌握。

为什么需要数组

- 读入 100 个数并倒序输出。若每个数用一个变量，就要写 $a_1, a_2, \ldots, a_{100}$，无法用循环统一处理。
- 数组把类型相同的数据连续排在一起，用一个名字和下标访问。
- 数组的每个位置叫一个元素；元素类型相同，所以每个元素占的字节数相同。



本页要点

一个数组名加一个下标，就能定位任意一个元素。

先看一个最小例子

```
int a[5] = {8, 6, 5, 4, 1};
for (int i = 0; i < 5; ++i) {
    cout << a[i] << ' ';
}
```

- 五个数据只用一个名字 a，第 i 个写作 a[i]。
- 循环从 0 走到 4，正好访问每个元素一次。
- 数据增加到 100 个时，只需把循环条件改成 100，语句数量不变。

本页要点

数组让“批量处理”变成一段固定长度的循环。

数组的声明

语法：类型数组名 [长度];

```
int a[10]; // 10 个 int
double score[50]; // 50 个 double
char name[20]; // 20 个 char
const int N = 100;
int b[N]; // 长度是 const 常量
```

- 长度必须是编译期常量：字面量、const int、#define。
- 数组名代表整块空间，不能整体输入、输出或赋值。
- 长度至少为 1；下标范围是 0 ～ 长度 −1。

本页要点

int a[10] 只声明，不初始化；里面是 10 个不确定的值。

长度为什么必须是常量

```
int n;
cin >> n;
int a[n]; // 不要这样写：长度不是常量

const int MAXN = 100005;
int a[MAXN]; // 推荐：全局静态数组
vector<int> b(n); // 推荐：长度可以运行时决定
```

- 标准 C++ 的静态数组长度必须在编译时确定；变长数组只是某些编译器的扩展。
- 局部数组太大（如 10^6 个 `int`）会撑爆栈，放全局或改用 `vector`。
- 题目给上限时，静态数组开到“上限 $+5$”，留出安全余量。

本页要点

长度固定用静态数组；长度由输入决定用 `vector`。

数组的初始化：一次给出所有值

```
int a[5] = {8, 6, 5, 4, 1};
int b[] = {3, 1, 4, 1, 5}; // 长度自动为 5
char s[] = "C++"; // 4 个字符：C + + \0
```



- 初始化列表按顺序对应下标 $0, 1, 2, \ldots$。
- 省略长度时，编译器按列表元素个数确定长度。
- `"C++"` 是字符串字面量，末尾自动带结束符 `'\0'`。

本页要点

定义时才能用 `{...}` 整体初始化；定义之后只能逐个元素赋值。

部分初始化与全零

```
int a[5] = {1, 2}; // 1 2 0 0 0
int b[5] = {}; // 0 0 0 0 0
int c[5] = {0}; // 0 0 0 0 0
```

- 列表元素少于长度时，剩下的元素自动补 0。
- `{}` 和 `{0}` 都得到全零；但 `{1}` 只有第一个是 1。
- 全局数组和 `static` 数组即使不写初始化列表，也会自动全零。

易错点

`int a[5] = {1};` 不是“全部设为 1”，而是 $1, 0, 0, 0, 0$。

本页要点

想让整型数组全零，写 `int a[5] = {};` 最省事。

定义之后的赋值

```
int a[5];
fill(a, a + 5, 7); // 全部改成 7
for (int i = 0; i < 5; ++i) {
    a[i] = i * i; // 逐个赋值
}
```

- `a = {1, 2, 3};` 在定义之后不合法。
- `fill(首指针, 尾后指针, 值)` 可以批量赋值；头文件 `<algorithm>`。
- `memset(a, 1, sizeof(a))` 按字节填充，`int` 不会变成 1，别用它初始化整数数组。

本页要点

初始化列表属于“定义”；之后的修改都要通过下标或 `fill`。

下标从 0 开始

- n 个元素的合法下标是 $0, 1, \ldots, n-1$，共 n 个。
- `a[0]` 是第一个，`a[n-1]` 是最后一个。
- 题面说“第 i 个”时，程序里通常写 `a[i-1]`。
- 下标本质是“离起始位置偏移几个元素”。



本页要点

看到 `a[n]`、`a[-1]` 就要立刻想到越界。

下标与地址

- 数组元素在内存中连续存放，每个 `int` 占 4 字节。
- 若首地址是 `0x1000`，则 `a[i]` 的地址是 `0x1000 + 4*i`。
- 所以下标从 0 开始：`a[0]` 的偏移是 0。
- `sizeof(a)` 是整个数组的字节数；`sizeof(a[0])` 是单个元素的字节数。



相邻元素相差 4 字节

本页要点

`sizeof(a) / sizeof(a[0])` 可以在数组未退化时求元素个数。

访问与修改

```cpp
int n;
cin >> n;
vector<int> a(n); // n 个 0
for (int i = 0; i < n; ++i) {
    cin >> a[i];
}
for (int i = 0; i < n; ++i) {
    cout << a[i] << ' ';
}
cout << '\n';
```

- `a[i]` 既可以读，也可以写：`a[i] = 7`、`++a[i]`、`a[i] += 2`。
- `vector<int> a(n)` 的 n 个元素都是 0；空 vector 不能用下标直接写。

本页要点

先保证下标在 $0 \sim n-1$，再读写。

顺序遍历：累加与最值

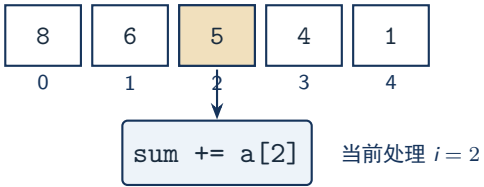
```
long long sum = 0;
for (int i = 0; i < n; ++i) {
    sum += a[i];
}
int best = a[0]; // 不能写成 0：可能有负数
for (int i = 1; i < n; ++i) {
    if (a[i] > best) {
        best = a[i];
    }
}
```

- 遍历就是从第一个到最后一个，每个元素恰好访问一次。
- 累加器初值为 0；最值初值取第一个元素，不要取 0。

本页要点

固定骨架：`for (int i = 0; i < n; ++i)`；求和用 `long long` 防溢出。

遍历过程：一个元素接一个元素



- 第 i 轮只看 `a[i]`，把结果累加到 `sum`。
- 访问顺序 $0 \to 1 \to \cdots \to n-1$，每个元素一次，所以是 $O(n)$。
- 循环变量只负责“当前看到哪”，不要在下标里写死常数。

本页要点

遍历是数组题最重要的基本功：读入、输出、统计、找最值都靠它。

逆序遍历

```cpp
for (int i = n - 1; i >= 0; --i) {
    cout << a[i] << ' ';
}
reverse(a.begin(), a.end()); // 也可以直接反转
```

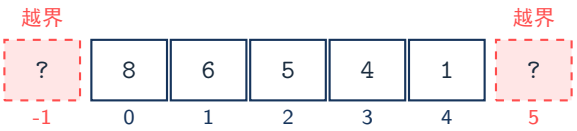
- 逆序从 n-1 走到 0，循环条件是 i >= 0。
- 若把 i 写成 size_t（无符号），i >= 0 永远为真，会死循环。
- 因此与 a.size() 比较时，先转成 (int)a.size()。

本页要点

逆序三要素：起点 n-1、条件 >=0、步长 --i。

越界：编译器不会替你检查

```
vector<int> a = {1, 2, 3, 4, 5};
cout << a[5] << '\n'; // 越界：结果不可预测
cout << a.at(5) << '\n'; // at 会检查，抛出异常
```



- 合法下标只有 $0 \sim n-1$；a[n]、a[-1] 都是未定义行为。
- 越界可能立刻崩溃，也可能悄悄改坏别的变量，最难查。

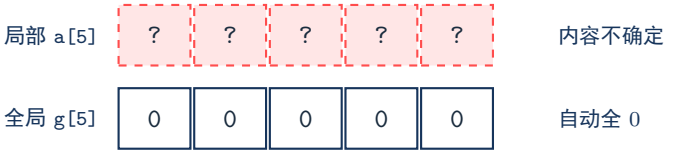
本页要点

写循环时先问：下标最大能取到几？

未初始化：局部数组里是“垃圾值”

```cpp
int g[5]; // 全局：自动全 0

int main() {
    int a[5]; // 局部：5 个不确定的值
    static int s[5]; // 静态局部：自动全 0
    vector<int> v(5); // 5 个 0
}
```



本页要点

使用前必须初始化；不要假设局部数组一开始是 0。

静态数组与 vector：两种存法

```cpp
const int N = 100;
int a[N]; // 长度固定
for (int i = 0; i < N; ++i) {
    a[i] = i;
}
```

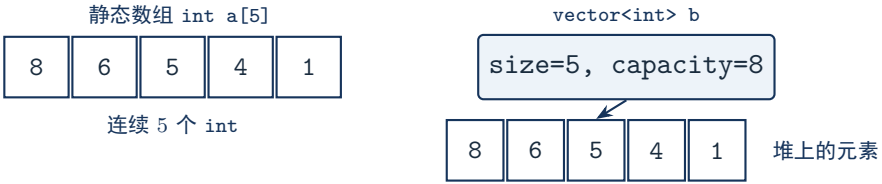
```cpp
int n;
cin >> n;
vector<int> b(n); // 长度随 n
for (int i = 0; i < n; ++i) {
    b[i] = i;
}
```

- 两者都用 `b[i]` 访问，下标都是 $0 \sim n-1$。
- 静态数组长度写死在代码里；`vector` 的长度可以来自输入。

本页要点

先把两者都当成“一排格子”；区别在于长度何时确定、能否追加。

两种存法的内存布局



- 静态数组的元素就在数组名对应的那块连续空间里，长度固定，不能追加。
- vector 对象只保存数据指针、size、capacity；元素在堆上。
- 扩容时 vector 换一块更大的堆空间，复制旧元素，再释放旧空间。

本页要点

静态数组“就地”存；vector 是“对象 + 堆数据”两层结构。

静态数组与 vector 的对比

	静态数组 `int a[N]`	`vector<int> a`
长度	编译期常量，之后不能改	运行时确定，可增删
内存	局部在栈，全局在数据区	元素在堆，对象在栈
初始化	`{...}` 或全零	`a(n, 0)` 或 `{...}`
越界	不检查	`[]` 不检查，`at()` 检查
追加	不支持	`push_back`
传参	退化为指针，丢失长度	按引用传递，保留 `size()`
复制	不能整体赋值	`=` 深拷贝
适用	长度固定的小表	长度动态、需要操作

本页要点

读入 n 个数时优先 `vector`；长度固定的计数、标记数组可用静态数组。

什么时候用哪一种

- 长度由输入 n 决定：`vector<int> a(n);`
- 长度有明确上限、反复使用：`const int MAXN = 100005; int a[MAXN];`
- 需要 `push_back`、`size()`、整体复制：用 `vector`。
- 作为函数参数时，静态数组会退化成指针；`vector` 用引用传递。

```
const int MAXN = 100005; // 题目上限 + 5
int cnt[MAXN]; // 全局数组，自动全 0
vector<int> a; // 长度未知时使用
a.reserve(MAXN); // 可选：预留空间
```

本页要点

竞赛常用组合：全局静态数组开到最大，`vector` 处理动态数据。

vector 的声明

```
vector<int> a; // 空，size = 0
vector<int> b(5); // 5 个 0
vector<int> c(5, 7); // 5 个 7
vector<int> d = {3, 1, 4}; // 3 个元素
vector<vector<int>> g(3, vector<int>(4, 0));
```

- `vector<int> a;` 之后不能写 `a[0]`，要先 `push_back` 或 `resize`。
- `vector<int> b(n);` 的 n 个元素值初始化成 0。
- 元素类型可以是 `long long`、`string`、`pair<int,int>`、结构体。

本页要点

`vector` 知道自己的长度：`a.size()`。

push_back、size、empty

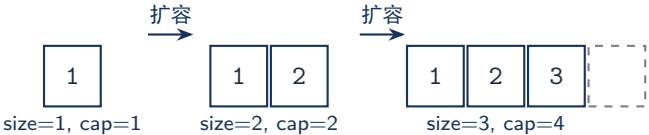
```cpp
vector<int> a;
a.push_back(8); // {8}
a.push_back(6); // {8, 6}
cout << a.size() << '\n'; // 2
cout << a.empty() << '\n'; // 0（false）
a.pop_back(); // {8}
```

- `push_back(x)` 在末尾追加，均摊 $O(1)$。
- `size()` 返回 `size_t`（无符号），输出、比较时注意类型。
- `empty()` 判断是否为空，比 `size() == 0` 更直观。

本页要点

空 vector 的 `a[0]` 是越界；先追加或 `resize`。

vector 为什么会“动态扩容”



- vector 的元素仍然连续存放；size 是元素个数，capacity 是当前容量。
- 空间满了再 push_back，会申请更大的空间，复制旧元素，再释放旧空间。
- 扩容通常按倍数增长，所以连续追加的均摊复杂度是 $O(1)$。
- 扩容后旧的指针、引用、迭代器都会失效。

本页要点

数据量大时可用 a.reserve(n) 一次预留，避免反复复制。

vector 的三种遍历

```cpp
for (int i = 0; i < (int)a.size(); ++i) {
    cout << a[i] << ' ';
}
for (int &x : a) {
    x *= 2; // 修改原元素
}
for (const int &x : a) {
    cout << x << ' ';
}
```

- 下标循环：需要下标或要逆序时用。
- `for (int x : a)` 是复制，改 `x` 不影响数组；要修改必须写 `&x`。
- `const int &x` 只读且不复制，适合大元素。

本页要点

`auto &x` 可以代替 `int &x`，让编译器推导类型。

vector 常用操作速查

操作	写法	说明
尾部追加	`a.push_back(x)`	均摊 $O(1)$
尾部删除	`a.pop_back()`	不返回被删元素
元素个数	`a.size()`	返回 `size_t`
是否为空	`a.empty()`	空为 `true`
清空	`a.clear()`	`size` 变 0
改长度	`a.resize(n)`	变长补 0
首尾元素	`a.front()`, `a.back()`	空时不要调用
下标访问	`a[i]`, `a.at(i)`	`at` 检查越界
预留容量	`a.reserve(n)`	不改变 `size`
排序反转	`sort`, `reverse`	配合 `begin/end`

本页要点

最常用的五个：`push_back`、`size`、`empty`、`clear`、`resize`。

小例子：读入 n 个数，求和与最大值（1/2）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
```

小例子：读入 n 个数，求和与最大值（2/2）

```
    long long sum = 0;
    int best = a[0];
    for (int i = 0; i < n; ++i) {
        sum += a[i];
        best = max(best, a[i]);
    }
    cout << sum << ' ' << best << '\n';
    return 0;
}
```

本页要点

求和用 `long long`；最值初值取 `a[0]`，并在 $n \geq 1$ 时使用。

二维数组：行与列

```
int a[3][4]; // 3 行 4 列，共 12 个元素
a[0][0] = 1; // 第 0 行第 0 列
a[2][3] = 9; // 第 2 行第 3 列
```

- 第一个下标是行，第二个下标是列；两个都从 0 开始。
- $n \times m$ 的矩阵：行 $0 \sim n-1$，列 $0 \sim m-1$。
- 内存按行优先存放：先放完第 0 行，再放第 1 行。
- 二维数组不能写成 `a[i,j]`；逗号表达式只会取到一个下标。

本页要点

`a[i][j]` 读作“第 i 行第 j 列”。

矩阵的行列坐标

列下标 j

行下标 i	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=0$	0	1	2	3	4
$i=1$	10	11	12	13	14
$i=2$	20	21	22	23	24
$i=3$	30	31	32	33	34

$a[2][3]$

- 主对角线满足 $i=j$；副对角线满足 $i+j=n-1$。
- 边缘元素满足 $i=0$ 或 $i=n-1$ 或 $j=0$ 或 $j=m-1$。

本页要点

先画坐标，再写条件；不要凭感觉写 `i+j`。

二维数组的初始化

```
int a[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
int b[2][3] = {1, 2, 3, 4, 5, 6}; // 按行填
int c[3][4] = {}; // 全 0
```

- 内层花括号对应一行；省略内层时按行依次填。
- 元素少于总数时，剩余元素补 0。
- 第一维可以省略：`int d[][3] = {{1,2,3}};`，第二维不能省。

本页要点

二维数组的“长度”有两维，写循环时不要只用一个 n。

二维数组的遍历（以 3×4 为例）

```
int a[3][4] = {};
for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 4; ++j) {
        cin >> a[i][j];
    }
}
for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 4; ++j) {
        cout << a[i][j] << ' ';
    }
    cout << '\n';
}
```

本页要点

外层枚举行、内层枚举列；每行输出完换行。

二维 vector：行数列数都可以变

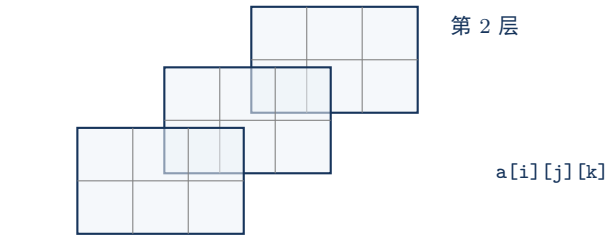
```
int n, m;
cin >> n >> m;
vector<vector<int>> a(n, vector<int>(m)); // n 行 m 列
vector<vector<int>> b; // 不规则表
for (int i = 0; i < n; ++i) {
    b.push_back(vector<int>(i + 1, 0)); // 第 i 行有 i+1 个 0
}
cout << a.size() << ' ' << a[0].size(); // n m
```

- a.size() 是行数，a[i].size() 是第 i 行的列数。
- 每一行可以长度不同，适合邻接表、三角形的存储。
- 传参写 vector<vector<int>> &a，避免整表复制。

本页要点

二维 vector 是“vector 的 vector”，先取行，再取列。

多维数组：一层一层往下选



- `int a[2][3][4];` 有 $2 \times 3 \times 4 = 24$ 个元素。
- 访问 `a[i][j][k]`：先选第 i 层，再选第 j 行，最后选第 k 列。
- 维度越多，内存增长越快：100^3 个 `int` 已是 4 MB。

本页要点

大多数题目二维就够；看到三维先确认内存上限。

char：一个字符

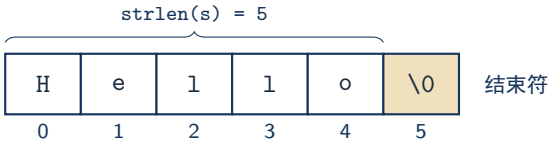
```
char c = 'A'; // 单引号，一个字符
cout << c << '\n'; // A
cout << (int)c << '\n'; // 65
cout << (char)(c + 1) << '\n'; // B
```

- `'A'` 是字符常量，`"A"` 是字符串，两者不同。
- ASCII：`'A'`=65，`'a'`=97，`'0'`=48。
- char 参与运算时会转成整数：`'a' - 'A'` 等于 32。

本页要点

字符可以比较、相减、加偏移；输出时注意要不要转回 char。

字符数组的结束符



- `char s[10] = "Hello";` 实际占 6 个位置：5 个字母加一个 `'\0'`。
- `'\0'` 是结束标记，`strlen(s)` 数到它为止，结果为 5。
- 数组长度至少要比字符数多 1；`char s[5] = "Hello";` 放不下结束符。

本页要点

`'\0'` 不算可见字符，但必须留位置。

字符数组的读入与风险

```
char s[20];
cin >> s; // 读到空格为止，自动补 '\0'
cout << s << '\n';
cout << strlen(s) << '\n'; // 数到 '\0'
strcpy(s, "Hi"); // 复制，必须保证空间够
```

- `cin >> s` 不检查长度，输入太长会越界写坏内存。
- 字符数组不能整体赋值：`s = "Hi"` 不合法，要用 `strcpy`。
- `strlen`、`strcpy`、`strcmp` 在 `<cstring>` 中。

本页要点

能不用 `char[]` 就不用；需要字符串时优先 `string`。

string：更安全的字符串

```
string s = "Hello";
string t(5, 'a'); // "aaaaa"
string u = s; // 复制一份
s += " C++"; // 追加
cout << s.size() << '\n'; // 9
cout << s[0] << s.back() << '\n'; // H+
```

- string 自动管理长度和结束符，不需要手写 '\0'。
- 支持 =、+、==、<，按字典序比较。
- s.size() 是 $O(1)$；s[i] 不检查越界。

本页要点

声明、拼接、比较、求长，string 都比 char[] 直接。

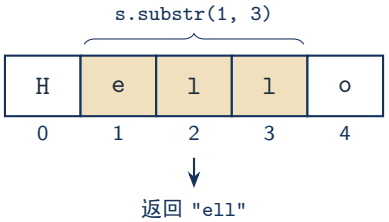
string 常用操作

操作	写法	说明
长度	`s.size()`	$O(1)$
判空	`s.empty()`	空为 `true`
下标	`s[i]` / `s.at(i)`	`at` 检查越界
末尾	`s.back()` / `s.pop_back()`	删除末尾字符
追加	`s += t` / `s.push_back(c)`	自动扩容
子串	`s.substr(pos, len)`	从 `pos` 起取 `len` 个
查找	`s.find(t)`	找不到返回 `string::npos`
反转	`reverse(s.begin(), s.end())`	原地反转
排序	`sort(s.begin(), s.end())`	按字符升序

本页要点

`find` 的返回值是 `size_t`，不要和 -1 比较；用 `string::npos`。

子串与查找



- `s.substr(1, 3)` 从下标 1 开始取 3 个字符，得到 `"ell"`。
- `s.find("ll")` 返回第一次出现的起始下标 2。
- 找不到时返回 `string::npos`，不能写 `== -1`。

本页要点

子串用“起点 + 长度”，查找返回“起点下标”。

char[] 与 string 的差异

	char[]	string
长度	定义时固定	可自动增长
结束符	必须自己留 '\0'	内部管理
赋值	不能 s = "Hi"	可以 s = "Hi"
拼接	strcat，要防越界	s += t
比较	strcmp	==, <
求长	strlen，$O(n)$	size()，$O(1)$
越界	不检查	[] 不检查，at() 检查
适用	C 接口、底层处理	绝大多数字符串题

本页要点

新代码优先 string；只有题目明确要求字符数组时才用 char[]。

整行读入：cin 与 getline

```
string s;
getline(cin, s); // 读一整行，保留空格

int n;
cin >> n;
cin.ignore(); // 吃掉 n 后面的换行
string t;
getline(cin, t);
```

- `cin >> s` 跳过空白，遇到空格就停；`getline` 读到换行为止。
- `cin >> n` 后缓冲区还剩换行，直接 `getline` 会读到空串。
- 更稳妥的写法：`cin.ignore(10000, '\n')`。

本页要点

读单词用 `cin >>`，读整行用 `getline`；混用时先清换行。

字符串的遍历与修改

```cpp
for (int i = 0; i < (int)s.size(); ++i) {
    cout << s[i] << '\n';
}
for (char &c : s) {
    c = toupper((unsigned char)c); // 转大写
}
```

- `s[i]` 返回字符的引用，可以修改；`s.at(i)` 会检查越界。
- `toupper`、`tolower`、`isalpha` 的参数要先转 `unsigned char`。
- 反转也可以双指针交换：`swap(s[i], s[j])`。

本页要点

逐字符扫描是字符串题最常用的套路。

数组与字符串的常见错误

错误写法	后果	正确做法
`int a[n];`（n 运行时）	非标准，可能栈溢出	`vector<int> a(n);`
`for (i=1;i<=n)` 访问 `a[i]`	越界	写 $0 \sim n-1$
`a[i,j]`	只取后一个下标 j，类型也不对	`a[i][j]`
`vector<int> v; v[0]=1;`	越界写	`push_back` 或 `resize`
`int sum; sum+=a[i];`	初值不确定	`int sum=0;`
`s.size()-1` 且 `s` 为空	无符号下溢	先判 `!s.empty()`
`cin>>n; getline(cin,s);`	读到空行	`cin.ignore()`
`char s[5]="Hello";`	没有 `'\0'` 位置	至少 `char s[6]`
`memset(a,1,sizeof a)`	每字节 1，不是整数 1	`fill(a,a+n,1)`

本页要点

数组题的错误大多来自下标：先检查 $0 \le i < n$，再检查行列范围。

基础自测

1. `int a[5] = {1, 2};` 中 `a[4]` 是多少？
2. $n = 5$ 时，合法下标有哪些？
3. `vector<int> v; v.push_back(3);` 后 `v.size()` 是多少？
4. `char s[] = "AB";` 占几个字节？
5. `a[2][3]` 是第几行第几列？

答案

1. 0。 2. $0, 1, 2, 3, 4$。 3. 1。 4. 3（含 `'\0'`）。 5. 第 2 行第 3 列。

本页要点

能不看笔记答对这五题，再进入例题。

【例 5.1】反序输出｜题意和推导

题意

输入一行 n 个整数（$n \le 100$，n 不单独给出），按输入时的逆序输出。

1. 没有单独的 n，用 `while (cin >> x)` 读到文件结束。
2. 把读到的整数依次存入 `vector`。
3. 从最后一个下标往前输出，第一个数前不加空格。

样例手算

`1 7 3 4 5` $\rightarrow$ `5 4 3 7 1`

本页要点

读到 EOF 的循环是“不知道个数”时的标准写法。

【例 5.1】反序输出 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    vector<long long> a;
    long long x;
    while (cin >> x) { // 读到文件结束
        a.push_back(x);
    }
    for (int i = (int)a.size() - 1; i >= 0; --i) {
        if (i + 1 < (int)a.size()) {
            cout << ' ';
        }
        cout << a[i]; // 逆序输出
    }
    cout << '\n';
    return 0;
}
```

【例 5.2】平移数据｜题意和推导

题意

数组长度 $n < 50$。把第一个元素移到末尾，其余元素依次左移一个位置。

1. 先保存第一个元素，避免左移时被覆盖。
2. 对 $i = 0, 1, \ldots, n-2$，执行 `a[i] = a[i+1]`。
3. 把保存的首项放到 `a[n-1]`；$n = 1$ 时结果不变。

样例手算

1 2 3 4 $\rightarrow$ 2 3 4 1

本页要点

“先保存再覆盖”是数组移动类题目的通用技巧。

【例 5.2】平移数据｜完整程序（1/2）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    int first = a[0]; // 先保存首项
    for (int i = 0; i + 1 < n; ++i) {
        a[i] = a[i + 1]; // 其余左移
    }
    a[n - 1] = first; // 首项放到末尾
```

【例 5.2】平移数据｜完整程序（2/2）

```
    for (int i = 0; i < n; ++i) {
        if (i > 0) {
            cout << ' ';
        }
        cout << a[i];
    }
    cout << '\n';
    return 0;
}
```

本页要点

输出时用 `i > 0` 控制空格，保证第一个数前没有多余空格。

【例 5.3】开关门 | 题意和推导

题意

n 个房间（$2 \le n \le 1000$）。第 k 个服务员翻转所有编号为 k 的倍数的门。求最后开着的门。

1. 门 k 被它的每个约数对应的服务员翻转一次。
2. 约数通常成对出现，只有完全平方数的约数个数是奇数。
3. 因此最后开着的是 $1^2, 2^2, \ldots$，输出不超过 n 的平方数。

样例手算

$n = 10$ 时打开的是 $1, 4, 9$。

本页要点

把“翻几次”转化为“约数个数”，问题就从模拟变成数学。

【例 5.3】开关门 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    bool first = true;
    for (int i = 1; i * i <= n; ++i) {
        if (!first) {
            cout << ' ';
        }
        cout << i * i; // 完全平方数
        first = false;
    }
    cout << '\n';
    return 0;
}
```

【例 5.4】约瑟夫问题｜题意和推导

题意

N 个人围成一圈，从第一个人开始报数，数到 M 的人出圈，再由下一个人继续。输出出圈顺序（$2 \le N, M \le 1000$）。

1. 用数组保存仍在圈内的人的编号。
2. 当前位置为 `pos`，数 M 个人后出圈位置是 `(pos + M - 1) % size`。
3. 输出后删除该位置；下一轮从同一下标开始，正好是“下一个人”。

样例手算

$N = 8, M = 5$ 时先出圈 5，随后是 $2, 8, 7, 1, 4, 6, 3$。

本页要点

删除后不用 `pos++`：原位置的新元素就是下一个人。

【例 5.4】约瑟夫问题 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n);
    iota(a.begin(), a.end(), 1); // 编号 1..n
    int pos = 0;
    while (!a.empty()) {
        pos = (pos + m - 1) % a.size();
        cout << a[pos] << (a.size() > 1 ? ' ' : '\n');
        a.erase(a.begin() + pos);
    }
    return 0;
}
```

【例 5.5】最大数位置｜题意和推导

题意

读入 n 个正整数（$n \le 1000$，每个数小于 2^{32}），输出最大数所在的位置，位置从 1 开始。

1. 从左到右读入，维护当前最大值 `best` 和位置 `at`。
2. 只有新数严格大于 `best` 时才更新，平局保留最早的位置。
3. 第一个数直接作为初值；输出 `at`。

样例手算

67 43 90 78 32 的最大值是 90，位置为 3。

本页要点

“严格大于”保证重复最大值时输出第一次出现的位置。

【例 5.5】最大数位置｜完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    unsigned long long best = 0, x;
    int at = 0;
    for (int i = 1; i <= n; ++i) {
        cin >> x;
        if (i == 1 || x > best) { // 平局保留最早
            best = x;
            at = i;
        }
    }
    cout << at << '\n';
    return 0;
}
```

【例 5.6】冒泡排序｜题意和推导

题意

输入 n 个小于 1000 的非负整数（$1 \le n \le 20$），用冒泡排序按从大到小输出，每个数一行。

1. 比较相邻两个数，若前小后大就交换。
2. 每完成一轮，当前最小值沉到末尾；下一轮少看一个元素。
3. 共做 n 轮，最后按顺序输出。

样例手算

`2 5 8 6 12` $\rightarrow$ `12,8,6,5,2`（每个数一行）。

本页要点

冒泡排序的核心是“相邻逆序就交换”，外层控制轮数，内层控制范围。

【例 5.6】冒泡排序 | 完整程序（1/2）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    // 每轮把当前最小值换到末尾
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j + 1 < n - i; ++j) {
            if (a[j] < a[j + 1]) {
                swap(a[j], a[j + 1]);
            }
        }
    }
```

【例 5.6】冒泡排序｜完整程序（2/2）

```
    for (int i = 0; i < n; ++i) {
        cout << a[i] << '\n';
    }
    return 0;
}
```

本页要点

内层写 `j + 1 < n - i`，既防越界又减少无用的比较。

【例 5.7】筛选法找质数 | 题意和推导

题意

用筛法求出 n 以内（$2 \le n \le 1000$）的全部质数，按从小到大输出，每个一行。

1. 先假设 $2 \sim n$ 都是质数，`prime[i] = true`。
2. 若 p 仍是质数，从 p^2 开始把 p 的倍数全部划掉。
3. 最后按顺序输出仍为 `true` 的数。

样例手算

$n = 10$ 时输出 $2, 3, 5, 7$。

本页要点

从 p^2 开始划，是因为小于 p^2 的 p 的倍数已被更小的质数划掉。

【例 5.7】筛选法找质数 | 完整程序（1/2）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<bool> prime(n + 1, true);
    prime[0] = prime[1] = false; // 0、1 不是质数
    for (int p = 2; p * p <= n; ++p) {
        if (!prime[p]) {
            continue;
        }
```

【例 5.7】筛选法找质数 | 完整程序（2/2）

```
        for (int q = p * p; q <= n; q += p) {
            prime[q] = false; // 划掉 p 的倍数
        }
    }
    for (int p = 2; p <= n; ++p) {
        if (prime[p]) {
            cout << p << '\n';
        }
    }
    return 0;
}
```

本页要点

筛法复杂度 $O(n \log \log n)$；$n = 1000$ 时非常快。

【例 5.9】新矩阵 | 题意和推导

题意

给定 $n \times n$ 方阵（$2 \le n \le 20$），把两条对角线上的元素加 10，输出新矩阵。

1. 主对角线满足 $i = j$；副对角线满足 $i + j = n - 1$。
2. 两个条件用“或”连接，中心元素只加一次。
3. 读入时可以边读边处理，不必保存整个矩阵。

样例手算

$n = 5$ 时中心元素 $38 \to 48$，四个角分别加 10。

本页要点

判断对角线用坐标条件，不要一条条写死位置。

【例 5.9】新矩阵 | 完整程序（1/2）

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            int x;
            cin >> x;
            // 主对角线或副对角线
            if (i == j || i + j == n - 1) {
                x += 10;
            }
```

【例 5.9】新矩阵 | 完整程序（2/2）

```
            if (j > 0) {
                cout << ' ';
            }
            cout << x;
        }
        cout << '\n';
    }
    return 0;
}
```

本页要点

中心只满足一次 `if`，不会重复加 10。

【例 5.10】稀疏矩阵 | 题意和推导

题意

$n \times m$ 矩阵中大部分元素是 0。把每个非零元素记成“行、列、值”，先输出 $n, m,$ 非零个数，再输出所有三元组。

1. 双重循环按行、列读取矩阵。
2. 遇到非零元素，把 `(i, j, x)` 追加到表中。
3. 行列号按题面从 1 开始。

样例手算

样例中第 1 行第 5 列是 5，输出 `1 5 5`。

本页要点

稀疏矩阵用“只存非零项”的方式压缩，适合非零元素远少于总元素的情况。

【例 5.10】稀疏矩阵 | 样例勘误与输出格式

样例勘误

OJ 样例第一行写作 `3 5 1`，但输入中第 3 行第 5 列还有一个 1，共有 4 个非零元素。按题意应输出 `3 5 4` 并列出 4 个三元组，本程序按题意输出。

- 输出顺序：先输出 `n m k`，再按行优先顺序输出每个非零项的三元组。
- 本例的非零项是 $(1,5,5)$、$(2,3,4)$、$(3,1,1)$、$(3,5,1)$，所以 $k=4$。
- 读入时用 1 开始的行列号，正好与题面输出一致，不需要再转换。

本页要点

遇到样例与题意矛盾时，先按题意写对，再在注释或说明里记录差异。

【例 5.10】稀疏矩阵 | 完整程序（1/2）

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, m;
    cin >> n >> m;
    vector<tuple<int, int, int>> a;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            int x;
            cin >> x;
```

【例 5.10】稀疏矩阵 | 完整程序（2/2）

```
            if (x != 0) {
                a.emplace_back(i, j, x); // 行、列、值
            }
        }
    }
    cout << n << ' ' << m << ' ' << a.size() << '\n';
    for (auto [i, j, x] : a) {
        cout << i << ' ' << j << ' ' << x << '\n';
    }
    return 0;
}
```

本页要点

tuple 与结构化绑定让“一行三元组”读起来很自然。

【例 5.11】杨辉三角形｜题意和推导

题意

打印杨辉三角形的前 n 行（$2 \le n \le 20$），每行相邻数之间一个空格。

1. 每行首尾都是 1。
2. 内部数字等于上一行正上方两个数之和：`b[j] = a[j-1] + a[j]`。
3. 先输出当前行，再据此生成下一行。

样例手算

第 5 行是 $1, 4, 6, 4, 1$。

本页要点

用两个一维数组滚动，比开二维数组更省空间。

【例 5.11】杨辉三角形 | 完整程序（1/2）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<long long> a(1, 1); // 第 0 行只有 1
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j <= i; ++j) {
            if (j > 0) {
                cout << ' ';
            }
            cout << a[j];
        }
```

【例 5.11】杨辉三角形｜完整程序（2/2）

```
        cout << '\n';
        vector<long long> b(i + 2, 1);
        for (int j = 1; j <= i; ++j) {
            b[j] = a[j - 1] + a[j]; // 肩上两数之和
        }
        a = b;
    }
    return 0;
}
```

本页要点

下一行长度是 `i + 2`，首尾初始化为 1，中间再填。

【例 5.12】回文字串 | 题意和推导

题意

输入一串字符（不超过 100 个），以句点结束。判断去掉句点后是否为回文。

1. 整行读入，若末尾是句点就删掉。
2. 复制一份并反转，得到逆序串。
3. 原串与逆序串相等才是回文。

样例手算

`abccb` 的逆序是 `bccba`，两者不同，输出 `No`。

本页要点

回文判断的核心：正着读和反着读一样。

【例 5.12】回文字串 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    string s;
    getline(cin, s);
    if (!s.empty() && s.back() == '.') {
        s.pop_back(); // 去掉句点
    }
    string t = s;
    reverse(t.begin(), t.end());
    if (s == t) {
        cout << "Yes\n";
    } else {
        cout << "No\n";
    }
    return 0;
}
```

【例 5.13】蛇形填数 | 题意和推导

题意

在 $n \times n$ 方阵中填入 $1 \sim n^2$，从右上角开始，按“下、左、上、右”循环，遇到边界或已填就转向（$n \leq 20$）。

1. 起点是右上角 $(0, n-1)$，初始方向向下。
2. 每填一个数，尝试沿当前方向走一步。
3. 下一步出界或已填，就顺时针转向，再走一步。

样例手算

$n = 4$ 时第一列从上到下是 $10, 9, 8, 7$。

本页要点

方向数组 `di/dj` 把“转向”变成下标加一取模。

【例 5.13】蛇形填数 | 完整程序（1/3）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<vector<int>> a(n, vector<int>(n, 0));
    int i = 0, j = n - 1, d = 0; // 从右上角出发
    int di[4] = {1, 0, -1, 0}; // 下、左、上、右
    int dj[4] = {0, -1, 0, 1};
    for (int x = 1; x <= n * n; ++x) {
        a[i][j] = x;
        if (x == n * n) {
            break;
        }
```

【例 5.13】蛇形填数 | 完整程序（2/3）

```
        int ni = i + di[d];
        int nj = j + dj[d];
        if (ni < 0 || ni >= n || nj < 0 || nj >= n || a[ni][nj]) {
            d = (d + 1) % 4; // 出界或已填：转向
            ni = i + di[d];
            nj = j + dj[d];
        }
        i = ni;
        j = nj;
    }
```

【例 5.13】蛇形填数 | 完整程序（3/3）

```
    for (int r = 0; r < n; ++r) {
        for (int c = 0; c < n; ++c) {
            if (c > 0) {
                cout << ' ';
            }
            cout << a[r][c];
        }
        cout << '\n';
    }
    return 0;
}
```

本页要点

先判断下一步是否合法，再决定走不走；不要先走再退回来。

【例 5.15】替换字母 | 题意和推导

题意

第一行是原文（不超过 200 个字符），第二行是两个字符 A, B。把原文中所有 A 换成 B，区分大小写。

1. 用 `getline` 读整行，保留空格和标点。
2. 再读入两个字符 A, B。
3. 逐字符扫描，遇到 A 就改成 B。

样例手算

原文中的大写 `I` 换成 `U`，其他字符不变。

本页要点

整行文本用 `getline`，不要用 `cin >>`。

【例 5.15】替换字母 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    string s;
    getline(cin, s);
    char a, b;
    cin >> a >> b;
    for (char &c : s) {
        if (c == a) {
            c = b;
        }
    }
    cout << s << '\n';
    return 0;
}
```

【例 5.16】过滤空格 | 题意和推导

题意

一行字符串（不超过 200 个字符），把连续多个空格压缩成一个，句子首尾没有空格。

1. 从左到右扫描，记录上一个字符是否为空格。
2. 当前是空格且上一个是空格，就跳过。
3. 其他字符原样输出，并更新标记。

样例手算

`Hello  world` → `Hello world`。

本页要点

用一个布尔标记记录“上一个是不是空格”，比回头删字符简单。

【例 5.16】过滤空格 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    string s;
    getline(cin, s);
    bool last_space = false;
    for (char c : s) {
        if (c == ' ' && last_space) {
            continue; // 连续空格跳过
        }
        cout << c;
        last_space = (c == ' ');
    }
    cout << '\n';
    return 0;
}
```

【例 5.18】串排序｜题意和推导

题意

输入 n 个国家名（$1 \le n \le 20$，长度不超过 20），按字母顺序输出，每个一行。

1. 把 n 个国家名存入 `vector<string>`。
2. 用 `sort` 按字典序排序。
3. 依次输出，每个名字一行。

样例手算

`Korea`，`China`，`Japan` → `China`，`Japan`，`Korea`。

本页要点

`string` 的 `<` 就是字典序，`sort` 可以直接用。

【例 5.18】串排序 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<string> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    sort(a.begin(), a.end()); // 字典序
    for (int i = 0; i < n; ++i) {
        cout << a[i] << '\n';
    }
    return 0;
}
```

【例 5.19】字符串判等｜题意和推导

题意

两行字符串，只含字母和空格。忽略大小写、忽略空格后，判断是否相等。

1. 分别整行读入两个字符串。
2. 删除空格，并把字母统一成小写。
3. 比较处理后的字符串，相等输出 `YES`，否则 `NO`。

样例手算

`a A bb BB ccc CCC` 与 `Aa BBbb CCCccc` 都变成 `aabbbbcccccc`。

本页要点

“先规范化，再比较”是字符串判等题的通用流程。

【例 5.19】字符串判等 | 完整程序（1/2）

```cpp
#include <bits/stdc++.h>
using namespace std;
string clean(string s) {
    string t;
    for (char c : s) {
        if (c == ' ') {
            continue; // 跳过空格
        }
        t += tolower((unsigned char)c); // 统一小写
    }
    return t;
}
```

【例 5.19】字符串判等｜完整程序（2/2）

```
int main() {
    string a, b;
    getline(cin, a);
    getline(cin, b);
    if (clean(a) == clean(b)) {
        cout << "YES\n";
    } else {
        cout << "NO\n";
    }
    return 0;
}
```

本页要点

把清洗逻辑抽成函数，主程序只剩“读入、比较、输出”。

【例 5.20】字串包含 | 题意和推导

题意

给定两个字符串（只含字母和数字，长度不超过 30），判断其中一个是否是另一个经过若干次循环移位后的子串。

1. 先让 a 是较长串，b 是较短串。
2. 循环移位得到的所有长度为 $|a|$ 的串，都出现在 a + a 中。
3. 因此只需判断 b 是否为 a + a 的子串。

样例手算

AABCD + AABCD 含 CDAA，输出 true。

本页要点

“循环移位”转成“拼接后查找”，是这类题最省力的做法。

【例 5.20】字串包含 | 完整 C++ 程序

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    string a, b;
    cin >> a >> b;
    if (a.size() < b.size()) {
        swap(a, b); // a 为较长串
    }
    string doubled = a + a;
    if (doubled.find(b) != string::npos) {
        cout << "true\n";
    } else {
        cout << "false\n";
    }
    return 0;
}
```

练习路线

1. 先做一维数组的统计、查找、标记和排序。
2. 再做二维矩阵：交换、边界、转置、旋转、乘法。
3. 最后练字符串：逐字符处理、单词处理、子串判断。

板块	题号范围	数量	重点
一维数组	P5102–P5118	17	统计、查找、排序、标记
二维数组	P5119–P5128	10	边界、转置、旋转、乘法
字符与字符串	P20072, P5131–P5149	20	逐字符、单词、子串

本页要点

提交前检查：空数组、首尾位置、重复元素、单行或单列矩阵、空格与大小写。

第一节一维数组 | 练习 1–7

1. P5102 | 与指定数字相同的数的个数
2. P5103 | 陶陶摘苹果
3. P5104 | 计算书费
4. P5105 | 数组逆序重存放
5. P5106 | 年龄与疾病
6. P5107 | 校门外的树
7. P5108 | 向量点积计算

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。

第一节一维数组 | 练习 8–14

1. P5109 | 开关灯
2. P5110 | 查找特定的值
3. P5111 | 不高兴的津津
4. P5112 | 最大值和最小值的差
5. P5113 | 不与最大数相同的数字之和
6. P5114 | 白细胞计数
7. P5115 | 直方图

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。

第一节一维数组 | 练习 15–17

1. P5116 | 最长平台
2. P5117 | 整数去重
3. P5118 | 铺地毯

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。

第二节二维数组 | 练习 1–7

1. P5119 | 矩阵交换行
2. P5120 | 同行列对角线的格
3. P5121 | 计算矩阵边缘元素之和
4. P5122 | 计算鞍点
5. P5123 | 图像相似度
6. P5124 | 矩阵加法
7. P5125 | 矩阵乘法

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。

第二节二维数组 | 练习 8–10

1. P5126 | 矩阵转置
2. P5127 | 图像旋转
3. P5128 | 图像模糊处理

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。

第三节字符类型和字符数组 | 练习 1–7

1. P20072 | 【05NOIP 提高组】谁拿了最多奖学金
2. P5131 | 基因相关性
3. P5132 | 石头剪子布
4. P5133 | 输出亲朋字符串
5. P5134 | 合法 C 标识符查
6. P5135 | 配对碱基链
7. P5136 | 密码翻译

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。

第三节字符类型和字符数组 | 练习 8–14

1. P5137 | 加密的病历单
2. P5138 | 将字符串中的小写字母转换成大写字母
3. P5139 | 整理药名
4. P5140 | 验证子串
5. P5141 | 删除单词后缀
6. P5142 | 单词的长度
7. P5143 | 最长最短单词

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。

第三节字符类型和字符数组 | 练习 15–20

1. P5144 | 单词翻转
2. P5145 | 字符串 p 型编码
3. P5146 | 判断字符串是否为回文
4. P5147 | 最高分数的学生姓名
5. P5148 | 连续出现的字符
6. P5149 | 最长单词 2

蓝色题名可打开学校 OJ；链接已逐页核对题名与样例。