ICPC 省赛 · 字符串入门详解

每个操作先用一道小问题引入：先写最朴素的实现，再看字符串函数怎么写。
面向刚学会字符串的初学者 ｜ C++17 ｜ A4 打印版

0. 先记住 4 件事

1. 先保证能写对，再考虑写得短；朴素写法通常更不容易错。
2. 下标从 0 开始；s.size() 是无符号类型，和 int 混用要小心。
3. find 找不到返回 string::npos；读字符串统一用 cin >> s。
4. 每段代码自己敲一遍，再造 3 组输入测试。

1–10. 问题 → 朴素实现 → 库函数

1. 读入字符串：统计数字字符个数

> **问题：**输入一个只含字母和数字的字符串（没有空格），统计其中数字字符的个数。例：`a1b2c3` → `3`。

朴素实现（先自己想一遍）

```
string s;
cin >> s;                    // 读入一个字符串，遇到空白停止

int cnt = 0;                 // 数字字符个数
for (int i = 0; i < (int)s.size(); i++) {
    char c = s[i];           // 当前字符
    if (c >= '0' && c <= '9') {
        cnt++;               // 是数字字符
    }
}
cout << cnt << '\n';
```

- `cin >> s` 会跳过开头空白，一直读到下一个空白；本讲义统一用它读字符串。
- 判断数字也可以写 `isdigit((unsigned char)c)`，效果一样。
- 字符 `'0'..'9'` 在 ASCII 中连续，所以也能用范围判断。

比赛中的常见写法（更省事）

```
// 用标准库算法写更短（效果一样）
int cnt = count_if(s.begin(), s.end(),
                   [](char c) { return c >= '0' && c <= '9'; });
cout << cnt << '\n';
```

- `count_if` 对区间内满足条件的元素计数。
- lambda 这里只是写个判断，初学阶段用上面的 for 循环更直观。

> 本讲义统一只用 `cin >> s` 读字符串；以后遇到含空格的整行题目再单独处理。

2. 大小写转换：把句子全部变小写

> **问题：**输入一个只含字母的单词，把所有大写字母改成小写。例：`HeLLo` → `hello`。

朴素实现（先自己想一遍）

```
string s;
cin >> s;                    // 读入一个单词
for (int i = 0; i < (int)s.size(); i++) {
    if (s[i] >= 'A' && s[i] <= 'Z') {
        s[i] = s[i] - 'A' + 'a';   // 大写 -> 小写
    }
}
cout << s << '\n';
```

- 原理：ASCII 中 A–Z 与 a–z 都连续，差值 32；`'a' - 'A' == 32`。
- 只改 `'A'..'Z'`，数字、空格、中文原样保留。
- `(int)s.size()` 是避免无符号数与 int 比较时的怪问题。

比赛中的常见写法（更省事）

```
for (int i = 0; i < (int)s.size(); i++)
    s[i] = tolower((unsigned char)s[i]);   // 直接改原串
// 或 transform(s.begin(), s.end(), s.begin(), ::tolower);
```

- `tolower` 对非字母原样返回，但参数转 `(unsigned char)` 才安全。
- 用下标 `s[i]` 直接改原串，不需要引用。

> 大写化同理：`c - 'a' + 'A'` 或 `toupper`。

---

3. 字符当数字：求各位数字之和

> **问题：**输入一个数字字符串（可能很长），输出每一位数字之和。例：`2026` → `10`（2+0+2+6）。

朴素实现（先自己想一遍）

```
string s;
cin >> s;
int sum = 0;
for (int i = 0; i < (int)s.size(); i++) {
    sum += s[i] - '0';       // 字符 '2' -> 数字 2
}
cout << sum << '\n';
```

- `'0'` 的 ASCII 是 48，`'2'` 是 50，所以 `'2' - '0' == 2`。
- 反过来 `char('0' + d)` 把数字变回字符。
- 如果输入本身就是整数，也可以 `x % 10` 取位、`x /= 10` 去掉；字符串题直接处理字符更方便。

比赛中的常见写法（更省事）

```
for (int i = 0; i < (int)s.size(); i++) sum += s[i] - '0';
```

- 范围 for 只是写法短，本质仍逐个处理。
- 若字符可能不是数字，先 `if (isdigit((unsigned char)c))`。

---

4. 按分隔符分割：逗号分隔求和

> **问题**：输入一行逗号分隔的整数（如 `3,1,4,1,5`），求它们的和。输出：`14`。

朴素实现（先自己想一遍）

```
string s;
cin >> s;
long long sum = 0, cur = 0;
bool hasDigit = false;
for (int i = 0; i < (int)s.size(); i++) {
    char c = s[i];                  // 当前字符
    if (c == ',') {                 // 遇到分隔符：结算当前数
        if (hasDigit) sum += cur;
        cur = 0;
        hasDigit = false;
    } else {
        cur = cur * 10 + (c - '0'); // 边读边算
        hasDigit = true;
    }
}
if (hasDigit) sum += cur;           // 最后一段
cout << sum << '\n';
```

- 核心：一个变量 `cur` 累加当前数字，遇到 `,` 就加到总和并清零。
- 最后一段后面没有逗号，循环结束后要再结算一次。
- `hasDigit` 用来处理空段（如 `1,,2`）。

比赛中的常见写法（更省事）

```
// 另一种常见写法：用 find 找逗号，再用 substr 截出每一段
long long sum = 0;
size_t start = 0;                   // 当前段的起点
while (true) {
    size_t p = s.find(',', start);       // 从 start 开始找下一个逗号
    if (p == string::npos) p = s.size(); // 没有逗号了：最后一段到末尾
    sum += stoll(s.substr(start, p - start));  // 把这一段转成数字
    if (p == s.size()) break;        // 已经处理到最后一段
    start = p + 1;                   // 跳过逗号，继续下一段
}
cout << sum << '
';
```

- `find(ch, start)` 的第二个参数是“从哪里开始找”。

- `substr(start, len)` 截出当前段；`stoll` 把字符串转成 long long。
- 这种写法假设每个字段都非空；出现空段时 stoll 会抛异常，需要额外判断。

---

5. 子串查找：统计 "ana" 出现次数

> **问题：**输入一个字符串，统计子串 "ana" 出现次数（允许重叠）。例：`banana` → 2 次（位置 1 和 3）。

朴素实现（先自己想一遍）

```
string s;
cin >> s;
string p = "ana";            // 要找的子串
int cnt = 0;                 // 出现次数
for (int i = 0; i + (int)p.size() <= (int)s.size(); i++) {
    bool ok = true;
    for (int j = 0; j < (int)p.size(); j++) {
        if (s[i + j] != p[j]) { ok = false; break; }  // 逐字符比较
    }
    if (ok) cnt++;                   // 每个起点都比一遍
}
cout << cnt << '\n';
```

- 朴素做法：枚举每个起点 i，再逐字符和模式串比较，复杂度 O(n·m)。
- 循环条件 `i + p.size() <= s.size()` 防止越界。
- 允许重叠时 i 每次加 1；不允许重叠就匹配后 `i += p.size() - 1`（for 里再 +1）。

比赛中的常见写法（更省事）

```
for (size_t p = s.find("ana"); p != string::npos; p = s.find("ana", p + 1)) cnt++;
// 不允许重叠就把 p + 1 改成 p + 3
```

- `find` 内部做匹配，短串够用；超长串/多次查询再考虑 KMP。
- 找不到返回 `string::npos`，不是 -1。

> 替换全部：`while ((p = s.find("ana")) != string::npos) s.replace(p, 3, "X");` 改长度时要重新 find。

---

6. 字符计数：判断是否 Pangram

> **问题：**输入一个由字母组成的字符串（没有空格），判断是否 26 个英文字母都至少出现一次（不区分大小

写）。例：`Thequickbrownfoxjumpsoverthelazydog` → YES。

朴素实现（先自己想一遍）

```
string s;
cin >> s;                              // 读入字符串
int c[26] = {};                        // 0~25 对应 a~z
for (int i = 0; i < (int)s.size(); i++) {
    char ch = s[i];                     // 当前字符
    if (isalpha((unsigned char)ch)) {
        ch = tolower((unsigned char)ch);
        c[ch - 'a']++;                  // 对应字母 +1
    }
}
bool ok = true;
for (int i = 0; i < 26; i++)
    if (c[i] == 0) ok = false;         // 有字母没出现
cout << (ok ? "YES" : "NO") << '\n';
```

- `int c[26] = {};` 把 26 个计数全部初始化为 0；不写 `= {}` 是随机值。
- `ch - 'a'` 把字母映射到 0~25 的下标。
- 更“笨”的写法是 for a~z 每个字母重新扫一遍字符串（O(26n)），能过但没必要。

比赛中的常见写法（更省事）

```
map<char, int> mp;
for (int i = 0; i < (int)s.size(); i++) mp[s[i]]++;   // 不存在的 key 先变成 0 再加
// 单个字符计数：count(s.begin(), s.end(), 'a')
```

- 数组计数快，但只适合下标连续的情况（26 字母、10 数字、ASCII）。
- `map` 更通用，O(log n) 且常数大。

---

7. 排序去重 / 异位词：silent 和 listen

问题：输入两个单词，判断是否互为字母异位词（字母种类和次数都相同）。例：`silent`、`listen` → YES；

`apple`、`apply` → NO。

朴素实现（先自己想一遍）

```
string a, b;
cin >> a >> b;
if (a.size() != b.size()) { cout << "NO\n"; return 0; }

bool used[1000] = {};                // 标记 b 中哪些位置用过
for (int i = 0; i < (int)a.size(); i++) {
    char c = a[i];                   // 当前字符
    bool found = false;
    for (int j = 0; j < (int)b.size(); j++) {
        if (!used[j] && b[j] == c) { // 在 b 里找一个没用过的相同字母
            used[j] = true;
            found = true;
            break;
        }
    }
    if (!found) { cout << "NO\n"; return 0; }
}
cout << "YES\n";
```

- 朴素思路：拿 a 的每个字母去 b 里找一个还没被用掉的相同字母，像“配对”。
- 复杂度 $O(n^2)$，n 大时会慢，但思路最直观。
- `used` 防止同一个字母被重复匹配。

比赛中的常见写法（更省事）

```
sort(a.begin(), a.end());
sort(b.begin(), b.end());
cout << (a == b ? "YES" : "NO") << '\n';
```

- 排序后异位词变成完全相同的字符串，直接比较，O(n log n)。
- 去重：`sort(s.begin(), s.end()); s.erase(unique(s.begin(), s.end()), s.end());`
- `unique` 只把重复元素移到后面，不删元素，必须配合 erase。

---

8. 字符串转数字：读出年龄

问题：输入一行，格式为 `name age`（名字、空格、年龄），输出 name 和 age+1。例：`Alice 20` → `Alice`

> `21`。

朴素实现（先自己想一遍）

```
string name, s;
cin >> name >> s;

int age = 0;                          // 手写“字符串 -> 数字”
for (int i = 0; i < (int)s.size(); i++)
    age = age * 10 + (s[i] - '0');   // 每读一位：原来的数左移一位再加新位
cout << name << ' ' << age + 1 << '\n';
```

- 原理：从左往右，每读一位就 `age * 10 + 这一位`，和竖式一样。
- 这招不能处理负号、小数点、前导空格；但理解“字符串数字”一定要会。
- 反向数字转字符串可手写 `x % 10 + '0'`，或用 `to_string`。

比赛中的常见写法（更省事）

```
int age = stoi(s);                  // 或 stoll(s) 转 long long
string t = to_string(age + 1);
```

- `stoi` 遇到非法字符/超出范围会抛异常（`"a12"` 抛；`"12a"` 得到 12）。
- 要判断“整个串都是数字”：`size_t used; stoll(s, &used); used == s.size()`。

---

9. 取子串 / 判断后缀：文件名的扩展名

> **问题：**输入一个文件名，输出扩展名；没有扩展名输出 none。例：`main.cpp` → `cpp`；`README` → `none`。

朴素实现（先自己想一遍）

```
string s;
cin >> s;
int dot = -1;
for (int i = 0; i < (int)s.size(); i++)
    if (s[i] == '.') dot = i;         // 找最后一个 '.'

if (dot == -1) cout << "none\n";
else cout << s.substr(dot + 1) << '\n';   // 点后面的部分
```

- `substr(pos)` 从 pos 取到结尾；`substr(pos, len)` 取 len 个。
- pos 必须在 `0..size()` 之间，越界会抛异常。
- 找“最后一个点”必须遍历完整个串再决定。

比赛中的常见写法（更省事）

```
size_t dot = s.rfind('.');       // 从后往前找
if (dot == string::npos) cout << "none\n";
else cout << s.substr(dot + 1) << '\n';

// 判断后缀（C++17）：
bool cpp = s.size() >= 4 && s.compare(s.size() - 4, 4, ".cpp") == 0;
```

- `rfind` 从后往前找，返回最后一次出现的位置。
- `compare(pos, len, str)` 比较 [pos,pos+len) 与 str，相等返回 0。
- 先判 `s.size() >= 4`，否则 `size()-4` 下溢。C++20 可直接 `s.ends_with(".cpp")`。

---

10. 回文判断：level 是不是回文

> **问题：**输入一个只含小写字母的字符串，判断是否回文（正着读和反着读一样）。例：`level` → YES；`abc` → NO。

朴素实现（先自己想一遍）

```
string s;
cin >> s;

int i = 0, j = (int)s.size() - 1;   // 双指针：一个从头、一个从尾
bool ok = true;
while (i < j) {
    if (s[i] != s[j]) { ok = false; break; }  // 对称位置不同
    i++;
    j--;
}
cout << (ok ? "YES" : "NO") << '\n';
```

- 双指针每次向中间走一步，比较对称位置。
- 空串和单字符天然是回文；`s.size()` 先转 `(int)` 再减 1。

比赛中的常见写法（更省事）

```
string r = s;
reverse(r.begin(), r.end());
cout << (r == s ? "YES" : "NO") << '\n';
```

- 反转后直接比较，代码最短；但会额外复制一份字符串，内存 O(n)。
- 双指针不需要额外空间，大数据时更稳。

---

11. 常见坑（按出现频率排序）

1. **size() 无符号**：`s.size()-1` 在空串时会变成巨大数；先 `int n=(int)s.size();`。

2. **npos**：`if (s.find(x) == string::npos)` 才是“没找到”。
3. **不重叠计数**：`"aaa"` 中找 `"aa"` 不重叠 1 次，重叠 2 次，区别在 `p+len` 还是 `p+1`。
4. **unique 不删除**：必须 sort 后 unique 再 erase。
5. **tolower/toupper 参数**：先转 `(unsigned char)`。
6. **stoi 会抛异常**：非法或越界都会抛；不确定时先判 `isdigit`。
7. **substr 越界**：pos 必须 `<= s.size()`，否则抛异常。

12. 入门例题（按上面的操作对应）

难度	题目	练哪个操作	链接
800	CF 71A Way Too Long Words	长度判断 + 拼接	71/A
800	CF 281A Word Capitalization	首字母大写（第 2 节）	281/A
800	CF 112A Petya and Strings	大小写 + 字典序	112/A
800	CF 236A Boy or Girl	去重（第 7 节）	236/A
800	CF 59A Word	大小写计数（第 6 节）	59/A
800	CF 339A Helpful Maths	分割 + 排序	339/A
800	CF 41A Translation	反转/回文（第 10 节）	41/A
800	CF 520A Pangram	字母计数（第 6 节）	520/A
800	CF 141A Amusing Joke	异位词（第 8 节）	141/A
800	CF 202A LLPS	回文判断（第 10 节）	202/A
800	CF 978B File Name	连续字符计数	978/B
800	CF 118A String Task	过滤 + 大小写 + 拼接	118/A
900	CF 1005B Delete from the Left	双指针 / 尾部比较	1005/B
1000	CF 43A Football	map 计数	43/A
800	CF 1367A Short Substrings	字符串拼接（第 1 节）	1367/A
普及−	洛谷 P1308 统计单词数	分割 + 不区分大小写查找	P1308
普及/提高−	洛谷 P1553 数字反转（升级版）	按符号分类讨论	P1553

训练入口：洛谷【入门5】字符串 | CF strings ≤1000
建议顺序：71A / 281A / 112A（读入、长度、大小写）→ 236A / 59A / 520A（计数、去重）→ 339A / 118A / 978B（分割、过滤、替换）→ 41A / 202A / 洛谷 P1308 / P1553（回文、查找、分类）。
例题来源：Codeforces Problemset API 与洛谷【入门5】训练；题面以原站为准。