C++ 编程语言 · 第七章

结构体

把一条记录的所有字段装在一起

HGNU-ACM 实验室算法组

截图配套：6 道练习全部逐题精讲

本章新增：**lambda** 表达式从零讲起





本章目录

定义与内存

成员访问

结构体数组

比较与排序

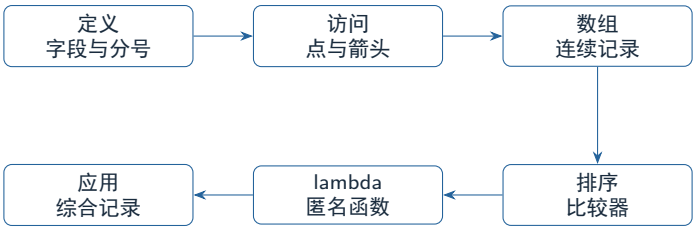
lambda 表达式

综合应用

常见错误

小结与索引

本章学习路线



- 先弄懂“一条记录”是什么，再学怎样访问、怎样批量存。
- 排序和 lambda 是本章难点：比较器决定顺序，lambda 让比较器就地写。
- 最后用 6 道练习把定义、访问、排序、稳定排序和综合字段串起来。

本页要点

结构体是“数据怎么放”，比较器是“顺序怎么定”，两者要一起学。

一个学生有姓名、学号、成绩，怎么放？

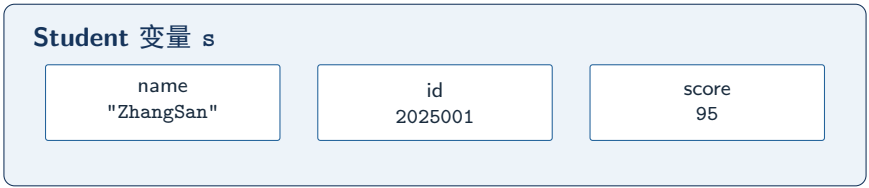
- 分开存三个数组：name[]、id[]、score[]。
- 排序时只能让一个数组动，姓名、学号与分数就会错位。
- 交换两名学生要同时改三个数组，漏改一个就“张冠李戴”。
- 函数想返回“一个学生”，也没法一次返回三个值。



本页要点

把属于同一个对象的字段打包成一种新类型，这种类型叫结构体（struct）。

一条记录长什么样



- 三个格子是三个字段（也叫成员），类型可以不同。
- 整个盒子是一个变量；赋值、传参、排序都以“整条记录”为单位。
- 字段名在盒子内部，必须通过变量名加点访问。

本页要点

结构体解决的是“把分散的数据绑定在一起”的问题。

定义结构体：先声明一种新类型

```
struct Student {
    string name; // 姓名
    int id; // 学号
    int score; // 成绩
}; // 这个分号不能漏
```

- `struct` 是关键字，`Student` 是新类型名（通常首字母大写）。
- 花括号里每行声明一个字段，字段之间用分号隔开。
- 这条语句只是“告诉编译器这种类型长什么样”，不创建变量。
- 之后可以写 `Student s;`、`Student a[105];`。

本页要点

结构体定义末尾的分号不是可选项；漏写是本章最常见的编译错误。

定义放在哪里：位置、分号与作用域

- 位置：一般放在 `#include` 之后、`main` 之前，所有函数都能看到。
- 分号：漏写时编译器常把错误报在下一行；看到 `expected ';'` 要先检查上一行结尾。
- 同时定义变量：`struct Point { int x, y; } p, q;` 合法，但初学阶段建议类型和变量分开写。
- 作用域：在函数内部定义的结构体，其他函数看不见；需要共用的类型要定义在全局。
- C++ 中类型名直接写 `Student`，不必像 C 那样每次写 `struct Student`。

本页要点

先定义类型，再定义变量；类型定义放在所有使用者之前。

字段可以是什么类型

```
struct Date {
    int year, month, day;
};
struct Student {
    string name;
    Date birthday; // 字段本身是结构体
    int score[3]; // 字段可以是数组
};
```

- 基本类型都可以：int、double、char、bool。
- 也可以是 string、数组，甚至另一个已经定义好的结构体。
- 字段类型决定它能做什么运算；name 是字符串，不能直接当数字加。

本页要点

结构体的字段就是普通变量，只是被放进了同一个“盒子”里。

创建、初始化与整体赋值

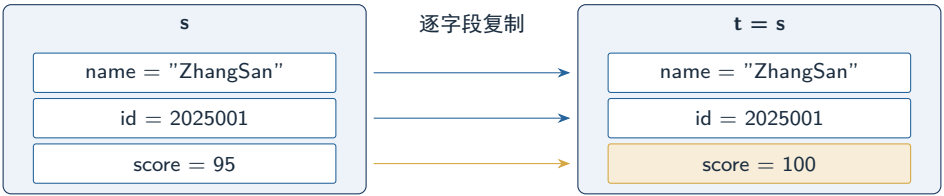
```
Student s{"ZhangSan", 2025001, 95}; // 按声明顺序
Student t = s; // 复制所有字段
t.score = 100; // 只改 t
cout << s.score << '\n'; // 95，未受影响
Student a[3] = {}; // 全部零初始化
```

- 花括号初始化按字段声明顺序：姓名、学号、成绩。
- `Student t = s;` 复制所有字段；之后修改 `t` 不影响 `s`。
- 局部变量不写初始化时，字段是未定义的随机值；读之前必须先赋值。
- `Student a[3] = {};` 把数组元素全部零初始化，`string` 为空串。

本页要点

结构体赋值是“整盒复制”，不是让两个变量指向同一份数据。

整体赋值：逐字段复制，不是共享

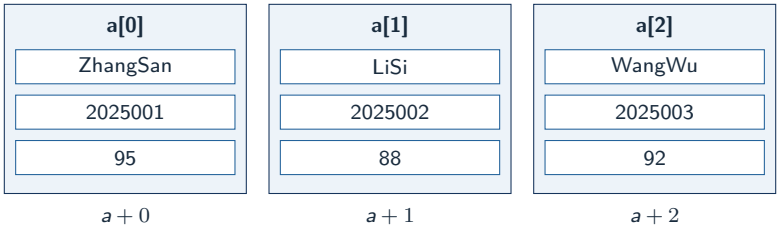


- 等号右边每个字段复制一份到左边，两个变量各自拥有独立的数据。
- 之后修改 `t.score`，`s.score` 仍是 95。
- 大结构体按值传参、按值返回时，发生的也是这种整体复制。

本页要点

结构体变量之间可以直接赋值，但复制的是内容，不是“绑定在一起”。

结构体数组的内存布局



- `a[i]` 是第 i 条完整记录，不是某个字段；取分数要写 `a[i].score`。
- 相邻元素的地址相差一条记录的大小；`a + i` 指向第 i 条记录。
- 排序交换的是整条记录，因此姓名和分数永远一起移动。
- 编译器可能为了对齐插入填充字节，实际字节偏移不要手算。

本页要点

结构体数组 = 连续存放的“记录表”，一行就是一条记录。

练习 1147 | 最高分数的学生姓名：题意与样例

题意

有 n 名学生（$n \le 100$），每行给出“分数姓名”。分数是不超过 100 的非负整数，姓名不含空格。数据保证最高分只有一位，输出这位学生的姓名。

轮次	读入	当前最高分	当前姓名
1	87 lilei	87	lilei
2	99 hanmeimei	99	hanmeimei
3	97 lily	99	hanmeimei
4	96 lucy	99	hanmeimei
5	77 jim	99	hanmeimei

注意：读入顺序是分数在前、姓名在后，与“姓名分数”的习惯相反。输出 `hanmeimei`。

练习 1147 | 完整代码（1/2）：定义与准备

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Student {
    int score;
    string name;
};

int main() {
    int n;
    cin >> n;
```

- 结构体里字段顺序可以自定，但读入语句必须按题面顺序。
- `best` 马上会用来保存“当前分数最高的整条记录”。

练习 1147 | 完整代码（2/2）：读入与比较

```
    Student best{-1, ""}, cur; // 当前最优记录
    for (int i = 0; i < n; ++i) {
        cin >> cur.score >> cur.name; // 分数在前，姓名在后
        if (cur.score > best.score) {
            best = cur; // 整体复制整条记录
        }
    }
    cout << best.name << '\n';
}
```

- 初值分数为 -1，保证第一个学生一定更新；$n \geq 1$。
- 保存整条记录，最后才能知道最高分对应哪个姓名。
- 时间复杂度 $O(n)$，额外空间 $O(1)$。

学校 OJ：P5147 | 最高分数的学生姓名

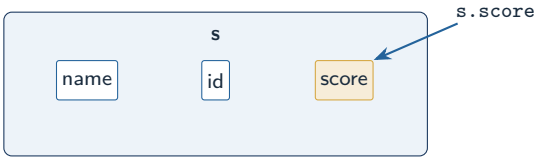
练习 1147 | 易错点

- 读入顺序：题目是“分数、姓名”；写成 `cin >> name >> score` 会把姓名当成分数读入失败。
- 只记最高分：如果只保存一个 `int bestScore`，最后不知道要输出谁；要保存整条记录。
- 初值：`best` 的分数设为 -1，保证第一个学生一定更新；$n \geq 1$。
- 并列：题面保证最高分唯一，用 `>` 即可；若要求并列取最早出现，也不能用 `>=`。
- 输出只写姓名，不要带分数或提示文字。

本页要点

“最高分”题的标准写法：保存当前最优的整条记录，而不是只保存一个数值。

点运算符：打开盒子取字段



- `s.name`、`s.id`、`s.score`。
- 点左边必须是结构体变量（或引用），右边必须是该类型有的字段。
- `s.score += 5;`、`cin >> s.score;`、`int x = s.score;` 都合法。
- 数组元素：`a[i].name`；点运算优先级高于算术，`a[i].score + 1` 不会歧义。

本页要点

点运算符是“打开盒子取东西”，取出来的字段就是普通变量。

点运算符的优先级与链式访问

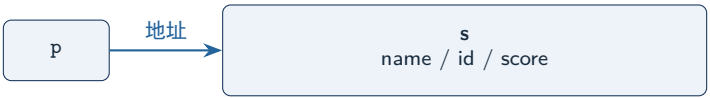
```
Student s;
s.birth.year = 2007; // 连续用点：先 birth，再 year
cout << a[i].score + 1; // 先取 a[i]，再取 score，最后加 1
cout << (*p).name << '\n'; // 括号不能省
cout << p->name << '\n'; // 与上一行等价
```

- 下标 [] 和点 . 从左到右结合，a[i].score 不会理解成 a[i.score]。
- 链式访问要求每一层都是结构体：s.birth 是 Date，再取 .year。
- *p.name 会被理解为 *(p.name)；想先解引用必须写 (*p).name。
- 记不住优先级时，直接加括号，代码更清楚。

本页要点

点用于对象，箭头用于指针；拿不准就加括号。

指针与箭头：p->score




```
Student s{"ZhangSan", 2025001, 95};
Student* p = &s; // p 保存 s 的地址
cout << (*p).name << '\n'; // 先解引用，再取成员
cout << p->name << '\n'; // 等价写法
```

- 指针变量存地址；`&s` 取地址，`*p` 取回 `p` 指向的对象。
- `p->name` 完全等价于 `(*p).name`，`->` 专用于“指针访问成员”。
- `vector` 中取地址：`Student* q = &a[i]; q->score`。

本页要点

有对象用点，有指针用箭头；两者不要混。

引用参数：让函数真正改到原记录

```
void read(Student& s) { // 引用：改到原记录
    cin >> s.name >> s.id >> s.score;
}
void print(const Student& s) { // 只读：不复制
    cout << s.name << ' ' << s.score << '\n';
}
void wrong(Student s) { // 按值：只是副本
    s.score = 0; // 外面不受影响
}
```

- `Student&` 是引用：函数内的 `s` 就是调用者的那条记录。
- `const Student&` 只读、不复制；比较器和大结构体传参都这样写。
- `Student s` 按值传参：复制整条记录，函数内修改不会影响外面。

本页要点

想修改原记录用 `&`；只读用 `const &`；按值传参只是副本。

按值传参与按引用传参：内存里发生了什么



- 按值传参多复制一份，适合小结构体；修改副本不影响调用者。
- 按引用传参不复制，函数直接操作原对象；只读时加 `const` 防止误改。

本页要点

大结构体优先用 `const Student&`；要改原对象才用 `Student&`。

练习 1399 | 甲流病人初筛：题意与样例

题意

n 位病人（$n < 200$），每人有姓名、体温、是否咳嗽。体温 ≥ 37.5 且咳嗽（值为 1）判为甲流初筛对象。按输入顺序输出这些人的姓名，最后单独输出人数。

病人	体温	咳嗽	判定
Zhang	38.3	0	否（不咳嗽）
Li	37.5	1	是（等于 37.5 也算）
Wang	37.1	1	否（体温不够）
Zhao	39.0	1	是
Liu	38.2	1	是

输出为 `Li`、`Zhao`、`Liu`，最后一行 `3`。

练习 1399 | 完整代码（1/2）：结构体与主程序开头

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Patient {
    string name;
    double temperature;
    int cough;
};

int main() {
    int n, cnt = 0;
    cin >> n;
```

- 体温是小数，必须用 `double`，不能用 `int`。
- `cnt` 统计符合条件的人数，初值为 0。

练习 1399 | 完整代码（2/2）：边读边判断

```
    for (int i = 0; i < n; ++i) {
        Patient p;
        cin >> p.name >> p.temperature >> p.cough;
        if (p.temperature >= 37.5 && p.cough == 1) {
            cout << p.name << '\n'; // 先输出姓名
            ++cnt; // 再累计人数
        }
    }
    cout << cnt << '\n'; // 最后输出人数
}
```

- 条件必须是“体温 ≥ 37.5 且咳嗽”，用 `&&`。
- 37.5 本身算发烧，写 `>= 37.5`，不能写 `> 37.5`。
- 姓名输出后不再需要保存，因此不必开数组。

练习 1399 | 易错点

- 体温用 `double`（或 `float`），不要用 `int`；37.5 是小数。
- 条件是“体温 ≥ 37.5 且咳嗽”，用 `&&`，不是“或”。
- 37.5 本身算发烧，必须写 `>= 37.5`，不能写 `> 37.5`。
- 先按输入顺序输出所有姓名，最后再输出人数；不要每输出一个名字就输出一次计数。
- 计数变量初始化为 0；不要把咳嗽状态 `cough` 和人数混用同一个变量。
- 本题没有学校 OJ 题号，使用题包练习。

本页要点

“筛选 + 计数”题：判断条件写全，输出顺序照题面。

结构体数组：一条记录就是一个元素

```
Student a[105]; // 固定长度
vector<Student> b(n); // 动态长度
a[0] = b[1]; // 整体赋值
```

- `Student a[105];` 固定长度；`vector<Student> a(n);` 动态长度。
- `a[i]` 的类型是 `Student`，可以整体赋值：`a[j] = a[i];`。
- `a[i].score` 才是 `int`；`a.score` 没有意义。
- 排序后“第 k 名”是 `a[k-1]`，不是 `a[k]`。
- 如果题目还要原始编号，把编号作为字段存进去，排序后仍能查到。

本页要点

数组的每个元素是一条记录；先把“记录”想清楚，再想下标。

批量读入与遍历

```cpp
int n;
cin >> n;
vector<Student> a(n);
for (int i = 0; i < n; ++i) {
    cin >> a[i].name >> a[i].id >> a[i].score;
}
for (const Student& s : a) {
    cout << s.name << ' ' << s.score << '\n';
}
```

- `const Student&` 避免每个元素复制一遍；只读时加 `const`。
- `for (auto& s : a)` 可以修改元素；只读时写 `const auto&`。
- 读入顺序必须逐项对照题面：1147 是“分数、姓名”，1839 有六个字段。

本页要点

读入之前先把题面的字段顺序抄在纸上，能避免大半的“样例不过”。

结构体数组作函数参数

```
void readAll(vector<Student>& a) {
    for (Student& s : a) {
        cin >> s.name >> s.id >> s.score;
    }
}
void printAll(const vector<Student>& a) {
    for (const Student& s : a) {
        cout << s.name << ' ' << s.score << '\n';
    }
}
```

- 传 `vector<Student>&` 不复制整个数组，函数内能改到原元素。
- 只读加 `const`；范围 `for` 里用 `Student&` 才能改元素。

本页要点

数组传参一律加 `&`；只读再加 `const`。

结构体数组的常见操作：找最值

```cpp
int best = 0;
for (int i = 1; i < n; ++i) {
    if (a[i].score > a[best].score) {
        best = i; // 记录下标
    }
}
cout << a[best].name << '\n';
```

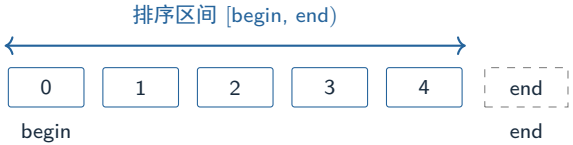
- 保存“最优元素的下标”，而不是只保存分数，最后才能取到整条记录。
- 从下标 1 开始比较，`best` 初值为 0，天然处理 $n = 1$。
- 若要求并列取最早出现，比较用 `>`，不要用 `>=`。

本页要点

找最值、找位置：保存下标；要输出整条记录时尤其方便。

sort 怎么用：区间与比较器

```
sort(a.begin(), a.end()); // 默认按 < 升序
sort(a.begin(), a.end(), cmp); // 使用自定义比较器
```



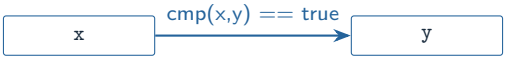
- 区间是 [begin, end)，左闭右开；end 指向最后一个元素的下一个位置。
- 不传比较器时，元素必须支持 <；结构体默认没有顺序，必须自己给规则。
- cmp(x, y) 返回 true，表示 x 应排在 y 前面。
- 时间复杂度 $O(n \log n)$；排序后原下标不再对应原来的学生。

本页要点

比较器回答的是“谁在前”，不是“谁更大”。

比较器：返回 true 表示 x 排在 y 前面

```
bool cmpScore(const Student& x, const Student& y) {
    return x.score > y.score; // true: x 排在 y 前面
}
sort(a.begin(), a.end(), cmpScore);
```



表示 x 应排在 y 前面

- 普通函数也能当比较器：函数名就是“可调用对象”，直接传给 sort。
- 降序用 x.score > y.score；升序用 <。
- 比较器参数用 const Student&，避免每次比较都复制整条记录。

本页要点

比较器只做一件事：回答“x 是否应该排在 y 前面”。

比较器必须满足严格弱序

三条实用检查

1. 自己和自己比必须为假：`cmp(x, x) == false`。
2. 不能同时认为 x 在 y 前、y 在 x 前。
3. 顺序关系要能传递；互相都不在对方前面的元素视为等价，等价关系也要能传递。

```
bool bad(const Student& x, const Student& y) {
    return x.score >= y.score; // 同分也返回 true
}
bool good(const Student& x, const Student& y) {
    return x.score > y.score; // 同分返回 false
}
```

- 升序用 `<`，降序用 `>`；绝不写 `<=`、`>=`。
- 违反后果：结果错乱甚至崩溃，属于未定义行为。

比较器写错会怎样：同分时的三种关系

x 分数	y 分数	正确 >	错误 >=	后果
90	90	false	true	互相认为对方在前
90	80	true	true	正常
80	90	false	false	正常

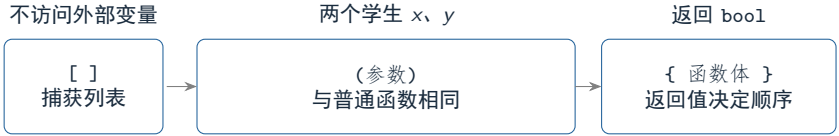
- 严格弱序要求“等价”只能来自“谁也不在谁前面”，不能来自“两边都为真”。
- 同分时返回 `true`，排序算法可能陷入错误的划分，甚至越界崩溃。
- 多关键字排序时，只有主键相等才看次键；全部相等必须返回 `false`。
- 记住一句话：比较器里只出现 `<` 或 `>`，不出现 `<=` 或 `>=`。

本页要点

严格弱序不是“更严格”，而是“相等时必须说相等”。

lambda（一）：把一小段函数写在调用处

```
auto cmp = [](const Student& x, const Student& y) {
    return x.score > y.score; // 分数高的排在前面
};
sort(a.begin(), a.end(), cmp); // 把 cmp 作为比较器
```



整体是一个匿名函数对象，可以赋给 auto，也可以直接传给 sort

- 语法 `[](参数){ 函数体 }`；`[]` 是捕获列表，这里不访问外部变量。

本页要点

lambda 是匿名的函数对象：没有名字、就地定义；返回类型由 return 自动推导。

lambda（二）：捕获、返回类型与比较器

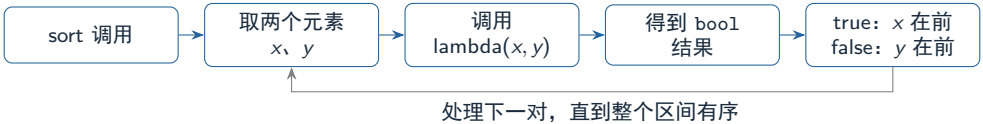
```
sort(a.begin(), a.end(),
    [](const Student& x, const Student& y) {
        if (x.score != y.score) {
            return x.score > y.score; // 主键降序
        }
        return x.name < y.name; // 次键升序
    });
```

写法	定义位置	能否捕获外部变量	复用性
普通函数	全局	不能	可多处调用
函数对象	`struct` 内	可保存状态	可多处调用
lambda	调用处	可以（`[=]`、`[&]`）	通常一次性

- `[]` 不捕获；`[=]` 按值捕获；`[&]` 按引用捕获；比较器通常只用 `[]`，保持无状态。
- 返回类型从 `return` 推导；多个 `return` 必须同类型，也可写 `-> bool` 明确指定。
- `sort` 的第三个参数是模板：只要能用 `cmp(x,y)` 调用并返回 `bool`，普通函数、函数对象、lambda 都可以。
- 严格弱序：相等返回 `false`；先主键、后次键，绝不用 `>=`。

lambda 作为参数传给 sort 的流程

```
sort(a.begin(), a.end(),
    [](const Student& x, const Student& y) {
        return x.score > y.score; // true: x 在前
    });
```



- sort 只负责取元素、移动元素；“谁在前”完全由 lambda 的返回值决定。
- lambda 不修改元素，也不依赖外部状态；同一个比较器可以反复调用。

本页要点

sort 是“执行者”，比较器是“规则”；lambda 把规则直接写在调用处。

练习 1176 | 谁考了第 k 名：题意与样例

题意

n 名学生（$1 \le n \le 100$），每个学生的成绩都不相同。每行给出“学号成绩”，求第 k 名（$1 \le k \le n$）的学号和成绩。成绩按题面要求用 %g 输出。

名次	学号	成绩
1	90788002	90.3
2	90788005	73.9
3	90788004	68.4
4	90788001	67.8
5	90788003	61

$k = 3$，答案是 90788004 68.4。
思路：读入所有记录；按成绩降序排序；取下标 $k-1$。

练习 1176 | 完整代码（1/2）：定义与读入

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Student {
    long long id;
    double score;
};

int main() {
    int n, k;
    cin >> n >> k;
    vector<Student> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i].id >> a[i].score;
    }
```

- 学号建议用 `long long`，避免位数增加后溢出。

练习 1176 | 完整代码（2/2）：排序与取第 k 名

```
    sort(a.begin(), a.end(),
        [](const Student& x, const Student& y) {
            return x.score > y.score; // 成绩降序
        });
    printf("%lld %g\n", a[k - 1].id, a[k - 1].score);
}
```

- 第 k 名对应下标 $k-1$；写 `a[k]` 会取到下一名或越界。
- 题面要求用 `%g` 输出成绩；格式串写 `"%lld %g"`，再补换行符。
- 排序 $O(n \log n)$，空间 $O(n)$。

学校 OJ：P5170 | 谁考了第 k 名

练习 1176 | 易错点

- 名次与下标：第 k 名是 `a[k-1]`；写 `a[k]` 会取到下一名或越界。
- 升降序：`x.score > y.score` 是降序；写成 `<` 会得到升序。
- 浮点类型：成绩是小数，用 `double` 读入；用 `int` 会截断。
- 输出格式：题面要求用 `%g` 输出成绩；不要输出多余提示文字。
- 学号：建议用 `long long`，避免学号位数增加后溢出。
- 题目保证成绩互不相同；若出现相等，比较器必须返回 `false`。

本页要点

排序题先确认三件事：按什么排、升序还是降序、取第几个。

多关键字排序：先主键，后次键

```
sort(a.begin(), a.end(),
    [](const Student& x, const Student& y) {
        if (x.score != y.score) {
            return x.score > y.score; // 主键降序
        }
        return x.name < y.name; // 次键升序
    });
```

- 1178 的规则：成绩从高到低；成绩相同，姓名字典序小的在前。
- 字典序：从第一个字符起逐个比较；前缀较短的字符串更小；`string` 的 `<` 已是字典序。
- 写法：主键不同先按主键决定；主键相同再看次键；全部相同返回 `false`。

本页要点

多关键字不是写两个并列的 `if`，而是“主键相同才轮到次键”。

练习 1178 | 成绩排序：题意与样例

题意

$0 < n < 20$。每行给出姓名和整数成绩；按成绩从高到低输出；成绩相同则姓名字典序小的在前。每行输出“姓名成绩”。

输入	排序后输出
Kitty 80	Joey 92
Hanmeimei 90	Hanmeimei 90
Joey 92	Kitty 80
Tim 28	Tim 28

样例没有同分。若再加一条 `Amy 90`，则 `Amy` 应排在 `Hanmeimei` 前（字典序 $A < H$）。
思路：姓名与成绩绑成一条记录；先按成绩降序，成绩相同再按姓名升序。

练习 1178 | 完整代码（1/2）：结构体与读入

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Student {
    string name;
    int score;
};

int main() {
    int n;
    cin >> n;
    vector<Student> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i].name >> a[i].score;
    }
```

- 读入顺序是“姓名、成绩”，与 1147 的“分数、姓名”不同。

练习 1178 | 完整代码（2/2）：双关键字排序

```
    sort(a.begin(), a.end(),
        [](const Student& x, const Student& y) {
            if (x.score != y.score) {
                return x.score > y.score; // 主键降序
            }
            return x.name < y.name; // 次键升序
        });
    for (const Student& s : a) {
        cout << s.name << ' ' << s.score << '\n';
    }
}
```

- 主键相同才执行第二个 `return`；全部相同返回 `false`。
- 输出“姓名空格成绩”，不要输出名次或提示文字。

学校 OJ：P5172 | 成绩排序

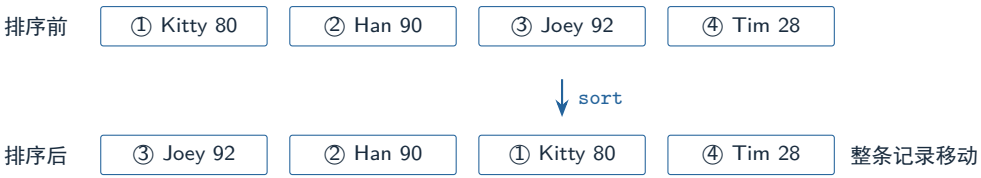
练习 1178 | 易错点

- 只按成绩排序，同分时顺序由 sort 任意决定，可能不符合“姓名字典序”。
- 次键升序用 x.name < y.name；写成 > 会变成降序。
- 比较器在全部字段相同时必须返回 false，不能写 >=。
- 输出格式是“姓名空格成绩”，不要输出名次或提示文字。
- 姓名只含字母，直接用 string 比较即可，不需要逐字符手写。
- 分数是不超过 100 的整数，用 int 足够。

本页要点

“同分再按姓名”是典型的多关键字排序：主键降序，次键升序。

排序前后：整条记录一起移动



- 比较器只回答“谁在前”，真正交换元素的是 `sort`。
- 交换的单位是整个 `Student`，姓名和成绩在同一个盒子里，不会错位。
- 圆圈数字是原始位置：排序后记录带着自己的全部字段一起走。

本页要点

结构体排序的关键不是“分别排两个数组”，而是“排一条条记录”。

stable_sort：相等记录保留原顺序

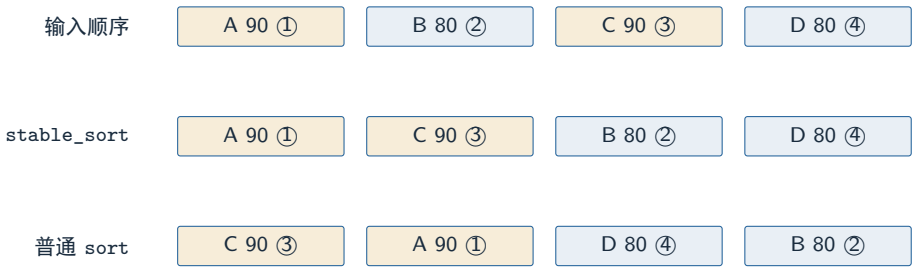
```
stable_sort(a.begin(), a.end(), cmp); // 相等元素保持原顺序
```

- stable_sort 的用法与 sort 相同，但相等元素保持原来的相对顺序。
- 适用场景：题目要求“年龄相同按登记顺序”“非老人按登记顺序”。
- 比较器仍须满足严格弱序；相等时返回 false，由稳定排序保留顺序。
- 复杂度通常为 $O(n \log^2 n)$ 或 $O(n \log n)$（需要额外内存）；$n < 100$ 完全够用。
- 另一种写法：给每条记录存原始编号 order，比较器最后按 order 升序做最终区分，再用普通 sort。

本页要点

题目出现“相同……按输入/登记顺序”时，优先想到 stable_sort。

稳定与不稳定：相等键的相对顺序



普通 `sort` 不保证相等键的原顺序；`stable_sort` 保证 ①→③、②→④

- 金色是 90 分档，蓝色是 80 分档；同色内部的圆圈数字表示原来的先后。
- 稳定排序只保证“相等键”的顺序；不同键之间仍按比较器排列。

本页要点

“并列时按登记顺序”是稳定排序的典型信号。

练习 1183 | 病人排队：题意与样例

排队规则

1. 老人（年龄 ≥ 60）比非老人优先。
2. 老人按年龄从大到小；年龄相同的按登记先后。
3. 非老人按登记先后。

登记顺序	ID	年龄	类别
1	021075	40	非老人
2	004003	15	非老人
3	010158	67	老人
4	021033	75	老人
5	102012	30	非老人

老人：021033、010158；非老人：021075、004003、102012。

思路：`stable_sort`；老人优先、老人按年龄降序，其余保持登记顺序。

练习 1183 | 完整代码（1/3）：结构体与读入

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Patient {
    string id;
    int age;
};

int main() {
    int n;
    cin >> n;
    vector<Patient> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i].id >> a[i].age;
    }
```

- ID 可能以 0 开头，必须用 `string` 读入。

练习 1183 | 完整代码（2/3）：稳定排序与优先级

```
stable_sort(a.begin(), a.end(),
    [](const Patient& x, const Patient& y) {
        bool ex = x.age >= 60;
        bool ey = y.age >= 60;
        if (ex != ey) {
            return ex; // 老人优先
        }
        if (ex && x.age != y.age) {
            return x.age > y.age; // 老人按年龄降序
        }
        return false; // 并列保持登记顺序
    });
```

- 先判断“是否老人”，再判断“老人年龄”，最后返回 `false`。

练习 1183 | 完整代码（3/3）：输出

```
    for (const Patient& p : a) {
        cout << p.id << '\n';
    }
}
```

- 每行只输出 ID，不要输出年龄或名次。
- 非老人之间全部“相等”，`stable_sort` 会保持他们的登记顺序。

学校 OJ：P5177 | 病人排队

练习 1183 | 易错点：为什么不能用普通 sort

- 普通 `sort` 不保证相等元素的原顺序；非老人之间全部“相等”，可能被打乱。
- 同龄老人之间也“相等”，同样必须保持登记顺序。
- `stable_sort` 恰好保留这些相等组的输入顺序。
- 若坚持用 `sort`，可给每人加 `order` 字段，比较器最后按 `order` 升序。
- 优先级顺序：先判断“是否老人”，再判断“老人年龄”，最后返回 `false`；顺序写反会把非老人排到前面。

本页要点

稳定排序解决“并列时按原顺序”，比较器只负责分出优先级的层级。

练习 1839 | 谁拿了最多奖学金：题意

题意

N 名学生（$1 \le N \le 100$），每人有姓名、期末平均成绩、班级评议成绩、是否学生干部、是否西部省份学生、发表论文数。五项奖学金可同时获得。求奖金最多者的姓名、其奖金，以及全体奖金总和；并列最多取输入最早者。

奖学金	奖金	条件
院士奖学金	8000	期末 > 80 且论文 ≥ 1
五四奖学金	4000	期末 > 85 且班级 > 80
成绩优秀奖	2000	期末 > 90
西部奖学金	1000	期末 > 85 且西部为 `Y`
班级贡献奖	850	班级 > 80 且干部为 `Y`

所有成绩条件都是严格大于。

练习 1839 | 样例手推

姓名	期末	班级	干部	西部	论文	奖金
YaoLin	87	82	Y	N	0	$4000+850=4850$
ChenRuiyi	88	78	N	Y	1	$8000+1000=9000$
LiXin	92	88	N	N	0	$4000+2000=6000$
ZhangQin	83	87	Y	N	1	$8000+850=8850$

- 最高奖金是 ChenRuiyi 的 9000，输出其姓名和奖金。
- 总奖金 $4850+9000+6000+8850=28700$。
- LiXin 同时满足五四奖学金和成绩优秀奖，最容易漏掉其中一个。

本页要点

每项奖学金独立判断，符合条件的都要累加。

练习 1839 | 完整代码（1/5）：结构体与奖金函数开头

```
#include <bits/stdc++.h>
using namespace std;

struct Student {
    string name;
    int finalScore, classScore;
    char leader, west;
    int papers;
};

int award(const Student& s) {
    int money = 0;
```

- 五个字段类型不同：姓名 `string`，成绩 `int`，两个标志 `char`。
- `award` 只负责算一个人的奖金，主程序负责比较和累加。

练习 1839 | 完整代码（2/5）：前两项奖学金

```
if (s.finalScore > 80 && s.papers >= 1) {
    money += 8000; // 院士奖学金
}
if (s.finalScore > 85 && s.classScore > 80) {
    money += 4000; // 五四奖学金
}
if (s.finalScore > 90) {
    money += 2000; // 成绩优秀奖
}
```

- 每个 `if` 独立，不能用 `else if` 串联。
- 条件里的成绩都是严格大于；例如期末 80 分拿不到院士奖学金。

练习 1839 | 完整代码（3/5）：后两项与返回

```
    if (s.finalScore > 85 && s.west == 'Y') {
        money += 1000; // 西部奖学金
    }
    if (s.classScore > 80 && s.leader == 'Y') {
        money += 850; // 班级贡献奖
    }
    return money;
}
```

- 字符比较用 `'Y'`，不要和字符串 `"Y"` 混用。
- 五项奖金可以同时累加，`money` 的初值为 0。

练习 1839 | 完整代码（4/5）：主程序读入

```
int main() {
    int n;
    cin >> n;
    string bestName;
    int bestMoney = -1, total = 0;
    for (int i = 0; i < n; ++i) {
        Student s;
        cin >> s.name >> s.finalScore >> s.classScore;
        cin >> s.leader >> s.west >> s.papers;
        int money = award(s);
        total += money;
```

- 六个字段分两行读入，顺序仍是题面的顺序。
- bestMoney 初值为 -1，保证第一个学生一定更新。

练习 1839 | 完整代码（5/5）：更新最优与输出

```
        if (money > bestMoney) {
            bestMoney = money;
            bestName = s.name;
        }
    }
    cout << bestName << '\n';
    cout << bestMoney << '\n';
    cout << total << '\n';
}
```

- 只有严格大于才更新，并列时保留最早输入者；用 >= 会改成“最晚”。
- 第三行输出全体奖金之和 `total`，不是最高奖金。

学校 OJ：P20072 | 【05NOIP 提高组】谁拿了最多奖学金

练习 1839 | 易错点

- 严格大于：条件写 `>`，不是 `>=`；例如期末 80 分拿不到院士奖学金。
- 独立判断：写成 `else if` 会漏奖；LiXin 就同时拿两项。
- 并列最早：只有 `money > bestMoney` 才更新；用 `>=` 会改成“最晚”出现者。
- 读入顺序：姓名、期末、班级、干部、西部、论文，六项一个都不能乱。
- 总和：第三行输出全体奖金之和，不是最高奖金。
- 字符：干部和西部字段是 `char`，比较时用 `'Y'`，不要和字符串 `"Y"` 混用。

本页要点

条件多、字段多的题，先把规则列成表，再逐条翻译成 `if`。

四个高频错误

1. 忘记分号：`struct` 定义末尾漏写 `;`，编译器常把错误报到下一行。修正：每个字段和整个定义都检查分号。
2. 比较器不满足严格弱序：写成 `>=`、`<=`，或同分时忘记处理次键。后果是排序结果错乱甚至崩溃。修正：升序 `<`、降序 `>`，相等返回 `false`。
3. 读入顺序错：1147 是“分数姓名”，1839 是六个字段；凭感觉读入会导致样例都过不了。修正：把题面字段顺序抄在代码旁。
4. 名次与下标混用：第 k 名是 `a[k-1]`；排序后原下标也已改变。

本页要点

编译错误先看结构体分号；答案错误先查读入顺序和比较器。

编译错误实战：漏分号与箭头误用

```
struct Student {
    string name;
    int score;
} // 错误：这里漏了分号
int main() {
    Student s;
    cout << s->name; // 错误：s 不是指针
}
```

- 漏分号时，编译器常报 `expected ';' before ...`，位置在下一行；要回头看结构体结尾。
- `s` 是对象，取成员用 `s.name`；只有指针 `p` 才能写 `p->name`。

本页要点

先分清“对象”还是“指针”，再决定用点还是箭头。

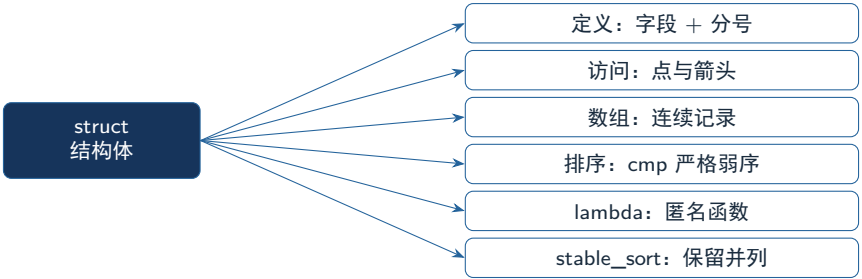
提交前的八项自查

1. 结构体定义末尾有分号吗？字段类型和题面一致吗？
2. 每个字段的读入顺序与题面完全相同吗？
3. 数组下标从 0 开始，第 k 名是否用了 `k-1`？
4. 比较器是否只用了 `<` 或 `>`，相等时返回 `false`？
5. 多关键字排序是否做到“主键不同先主键，主键相同再比次键”？
6. 需要保留输入顺序时，是否用了 `stable_sort` 或原始编号字段？
7. 输出格式、空格、换行、浮点精度是否符合题面？
8. 是否手算过样例，并测试了 $n = 1$、全部相同、最大 n 等边界？

本页要点

把这张清单贴在显示器旁，比反复提交试错更快。

本章知识地图



- 数据组织用 `struct`；顺序规则用比较器（严格弱序）；并列保序用 `stable_sort`，就地写规则用 lambda。

本页要点

结构体负责“装”，比较器负责“排”，lambda 负责“就地写规则”。

练习安排：从定义到综合

1. 先做 1147、1399：练结构体定义、点访问、边读边处理。
2. 再做 1176、1178：练结构体数组、`sort` 比较器与多关键字排序。
3. 接着做 1183：练 `stable_sort` 与多级优先级。
4. 最后做 1839：综合字段、函数、最大值与总和。

做题方法

每题先手推样例，把“记录里有哪些字段、按什么规则排序、输出什么”写成三行笔记，再开始写代码。

本章全部 6 道练习索引

一本通 1147：P5147 | 最高分数的学生姓名

一本通 1399：甲流病人初筛 P4957

一本通 1176：P5170 | 谁考了第 k 名

一本通 1178：P5172 | 成绩排序

一本通 1183：P5177 | 病人排队

一本通 1839：P20072 | 【05NOIP 提高组】谁拿了最多奖学金

点击蓝色题名进入学校 OJ；1399 暂无学校 OJ 题号，使用题包练习。本章 6 道练习全部列出，无遗漏。