

# 体育课的队列 ☆☆

## 题目描述

班级里共有  $N$  名同学，每位同学都有一个学号（从 1 到  $N$ ）、性别（用整数表示，1 代表男生，0 代表女生）和身高（整数，单位 cm）。

体育老师要求的队列规则如下：

1. **性别优先：**所有的男生必须排在队伍的前半部分，所有的女生必须排在队伍的后半部分。
2. **男生排序：**在前半部分的男生中，按照身高从高到低（降序）排列。
3. **女生排序：**在后半部分的女生中，按照身高从低到高（升序）排列。
4. **同等条件：**如果性别相同且身高也相同，则按照学号从小到大（升序）的顺序排列。

请你输出排好队后，队伍中从前到后依次对应的学号。

## 输入格式

第一行包含一个整数  $N$ ，表示班级人数。

接下来  $N$  行，每行包含两个整数  $g_i$  和  $h_i$ 。

第  $i$  行（输入数据的第 2 到  $N + 1$  行）的数据对应学号为  $i$  的同学的信息：

- $g_i$  表示性别（1 为男生，0 为女生）。
- $h_i$  表示身高。

## 输出格式

共一行，包含  $N$  个整数，表示排序后队列中同学的学号。整数之间用一个空格分隔。

## 输入输出样例

输入 #1	输出 #1
5 1 175 0 160 1 180 0 165 1 175	3 1 5 2 4

## 算法分析

本题属于典型的 **结构体多关键字排序** 问题。解题核心在于自定义比较函数（Comparator）。

## 1. 数据结构设计

为了方便排序时同时移动学号、性别和身高，我们需要定义一个结构体（struct）来存储每个学生的信息。

```
struct Student {  
    int id;      // 学号  
    int gender; // 性别: 1男 0女  
    int height; // 身高  
};  
Student a[100005];
```

## 2. 比较逻辑 ( $O(N \log N)$ )

我们需要实现一个比较函数 `cmp`，配合 `sort` 使用。比较逻辑遵循题目要求的四个优先级：

- **第一优先级（性别）：**男生排在女生前面。若  $A$  是男生 (1)， $B$  是女生 (0)，则  $A$  优于  $B$ 。
- **第二优先级（身高）：**当性别相同时，需要分情况讨论：
  - 若都是男生 ( $gender == 1$ )：身高从高到低，即  $A.height > B.height$ 。
  - 若都是女生 ( $gender == 0$ )：身高从低到高，即  $A.height < B.height$ 。
- **第三优先级（学号）：**当性别和身高都相同时，学号小的在前，即  $A.id < B.id$ 。

```
bool cmp(const Student &x, const Student &y) {  
    // 1. 性别不同，男生(1)排在女生(0)前  
    if (x.gender != y.gender) {  
        return x.gender > y.gender;  
    }  
  
    // 2. 性别相同，判断身高  
    if (x.height != y.height) {  
        if (x.gender == 1) {  
            // 男生：身高降序  
            return x.height > y.height;  
        } else {  
            // 女生：身高升序  
            return x.height < y.height;  
        }  
    }  
  
    // 3. 兜底：学号升序  
    return x.id < y.id;  
}
```

## 3. 主函数实现

读入数据时注意手动赋值 ‘`id`’，然后调用 `sort` 排序，最后输出 ‘`id`’ 即可。

```

for (int i = 1; i <= n; i++) {
    a[i].id = i; // 记录原始学号
    cin >> a[i].gender >> a[i].height;
}

sort(a + 1, a + 1 + n, cmp);

for (int i = 1; i <= n; i++) {
    cout << a[i].id << (i == n ? "" : " ");
}

```

## 拓展思考

### 1. 进阶写法：重载小于号运算符 (*operator <*)

在竞赛中，我们常常不写独立的 `cmp` 函数，而是直接在结构体内部重载小于号运算符。这样做的好处是结构体“自带”了排序规则，不仅可以直接调用 `sort`，还能直接放入 `priority_queue`（优先队列）或 `set` 中使用。

```

struct Student {
    int id, gender, height;
    bool operator < (const Student &other) const {
        if (gender != other.gender) return gender > other.gender;
        if (height != other.height) {
            return gender == 1 ? height > other.height : height <
                other.height;
        }
        return id < other.id;
    }
};

```

### 2. 变式思考：关于“稳定性” (Stability)

**问题：**如果题目删去规则4（“如果性别相同且身高也相同，则按照学号从小到大的顺序排列”），改为“保留输入时的相对顺序”，代码需要修改吗？

**分析：**

- 既然 `id` 就是按照输入顺序生成的 ( $1 \dots N$ )，那么原本的规则 4 实际上就是“保留输入相对顺序”的显式表达，所以原代码依然有效。
- 如果数据中没有 `id` 这一项呢？此时不能使用普通的 `sort`，因为它是**不稳定排序** (Unstable Sort，基于快排)，相等元素的相对位置可能会被打乱。
- 解决方案：**使用 C++ STL 提供的 `stable_sort`。它的用法与 `sort` 完全一致，但底层通常使用归并排序，能够保证“相等元素的相对位置不变”。
- 课后思考：**常见的诸如**冒泡排序** (Bubble Sort)、**插入排序** (Insertion Sort) 是稳定的吗？**快速排序** (Quick Sort) 呢？为什么？