Vector 删除偶数：迭代器失效与正确 erase 写法

【题目】

删掉 vector 里所有偶数，并保持剩余元素顺序。

【代码】

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

int main(){
    vector<int> a = {1,2,3,4,5,6,8,10};
    for(auto it = a.begin(); it != a.end(); ++it){
        if((*it)%2==0) a.erase(it);
    }
    for(int x: a) cout<<x<<" ";
}
```

【补充说明】

我觉得逻辑没问题，但输出总是怪怪的，有时候还崩。我不理解"迭代器失效"到底是什么...我是不是哪里基础没打好？

---

1. 题目概览

- 目标：把 `vector` 里所有偶数删除掉，并且**保留剩下元素的相对顺序**。
- 输入/输出：这里你直接写死了一个 `vector`，最后输出删完后的序列。
- 题型感觉：**数组/容器的遍历 + 删除元素**的小题，很常见～

---

2. 解题思路整理（C++ 视角）

我们有两种很常用的写法：

- **写法 A：遍历时删（手动迭代器）**
- 用迭代器从头走到尾
- 如果是偶数：`it = a.erase(it)`（拿到新的合法迭代器）

- 否则：`++it`

- **写法 B：更推荐：erase-remove 惯用法**

- `remove_if` 把“应该删的元素”挪到后面，返回“新结尾”

- 再 `erase` 一次性删掉尾巴那段

---

3. 头文件与库函数整理

你用了：

- `<bits/stdc++.h>`：竞赛常用“一把梭”，包含几乎所有标准库（OJ 常见）。
- `std::vector`：动态数组容器。
- `std::cout`：输出。 -（如果用 erase-remove 法会用到）
- `<algorithm>`：`std::remove_if`

小提醒：你现在是竞赛写法没问题～以后写工程/作业，慢慢改成明确的
`<vector> <iostream> <algorithm>` 会更专业更统一一点～

---

4. 代码可以改进的地方（配 C++ 示范）

你这段代码为什么会“怪/崩”？

关键点：`** 会让 **` **以及它后面的迭代器全部失效**。

原因很直观：`vector` 底层是一段连续内存。删掉中间一个元素后，后面的元素会整体往前挪一格填坑，所以：

- 原来指向某个位置的迭代器（本质像“指针”）就不再指向原来的那个元素了
- 你 `erase(it)` 后，循环还会 `++it`，但这个 `it` 已经不可靠了，于是就会跳元素、重复、甚至崩溃

✅修法 1：遍历时删（最贴近你原代码）

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

int main() {
    vector<int> a = {1,2,3,4,5,6,8,10};

    for (auto it = a.begin(); it != a.end(); ) {
        if ((*it) % 2 == 0) {
            it = a.erase(it);   // erase 返回“下一个有效位置”
        } else {
```

```
            ++it;
        }
    }

    for (int x : a) cout << x << " ";
}
```

这里的精髓就是：**删了就别 ++it，而是接住 erase 返回的新 it。**

✅修法 2：更推荐（更 C++、更干净）：erase-remove_if

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

int main() {
    vector<int> a = {1,2,3,4,5,6,8,10};

    a.erase(remove_if(a.begin(), a.end(),
                      [](int x){ return x % 2 == 0; }),
            a.end());

    for (int x : a) cout << x << " ";
}
```

这个写法好处是：**不需要你自己管迭代器**，也更不容易写错。

---

5. “迭代器失效”到底是什么（用很直觉的话）

你可以把迭代器理解成：**“指向容器里某个位置的定位器（像指针）”**。

- 对 `vector` 来说，元素是连续排队站好的。
- `erase` 会让后面的人**往前挪**。
- 你手里那张“座位号（迭代器）”就可能指向了一个已经变掉的位置——这就叫**失效**。

一句话记忆：

> **对 vector：删中间元素会搬家，所以很多旧的定位器都不能用了。**

---

6. 本题可以沉淀的小套路 / 小模板（C++ 框架）

模板 1：边遍历边删（安全写法）

```
for (auto it = v.begin(); it != v.end(); ) {
    if (need_delete(*it)) it = v.erase(it);
    else ++it;
}
```

模板 2：erase-remove（更推荐）

```
v.erase(remove_if(v.begin(), v.end(), pred), v.end());
```

加练时间到啦（两题）

练习题 A（同难度）

题意： 给你一个 `vector<int>`，删除所有能被 3 整除的数，保持顺序，输出结果。
思路： 直接用 `erase-remove_if`。
参考代码：

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

int main() {
    vector<int> v = {3,1,6,7,9,10,12,5};
    v.erase(remove_if(v.begin(), v.end(),
                      [](int x){ return x % 3 == 0; }),
            v.end());
    for (int x : v) cout << x << " ";
}
```

相似点：都是“按条件删除”。不同点：条件换成 `%3==0`。

练习题 B（更难一点、更像“小工程”）

题意： 输入 `n` 和 `n` 个整数，删除所有重复元素，只保留第一次出现的那个，并保持顺序。
例：`1 2 2 3 1 4` → `1 2 3 4`

思路拆解：

- 用一个 `unordered_set<int> seen` 记录出现过的数
- 遍历原数组，把第一次出现的数放进 `ans`
- 最后输出 `ans`

参考 C++ 框架：

```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];

    unordered_set<int> seen;
    vector<int> ans;
    ans.reserve(n);

    for (int x : a) {
        if (!seen.count(x)) {
            seen.insert(x);
            ans.push_back(x);
        }
    }

    for (int x : ans) cout << x << " ";
}
```

升级点：要多用一个容器配合，还要考虑“保持顺序”。