第30条：确保目标范围足够大。

transform算法是C++标准模板库（STL）中的一个算法，用于将一个操作应用到序列（例如容器或者数组）中的每个元素上，并将结果存储在另一个序列中。这个算法的行为类似于“映射”，因为它将输入序列中的每个元素通过一个给定的函数映射到输出序列中。

第31项：了解您的排序选项。

我对选择排序算法的建议是，根据需要完成的任务进行选择，而不是考虑性能。如果你选择一个只做你需要做的事情的算法（例如，分区而不是全排序），你可能最终得到的代码不仅是你想要做的事情的最清晰的表达，而且也是使用STL完成它的最有效的方法。

第32条：如果你真的想删除一些东西，就按照类似删除的算法删除。

remove并不“真正”删除任何东西，因为它不能。

remove并不“真正”删除任何东西，因为它不能。

* **remove的行为**：remove算法并不实际从容器中移除元素，而是将所有不需要移除的元素移动到序列的开始部分，并返回一个指向最后一个不需要移除元素之后的位置的迭代器（即newEnd）。被“移除”的元素实际上并没有被删除，它们只是被后面的元素覆盖了。
* **元素的顺序**：remove不会将所有要移除的元素移动到序列的末尾，而是保持未被移除元素的相对顺序不变。
* **保留的元素**：在remove操作之后，newEnd到v.end()之间的元素仍然是原序列中的元素，但它们的值可能不再有意义，因为它们已经被覆盖了。
* **误解**：人们通常期望remove会将所有要移除的元素放到容器的末尾，但实际上，remove并不保证这一点。如果需要保留所有元素，应该使用partition或stable\_partition算法，这两个算法会保持所有元素的顺序。
* **算法操作**：remove通过赋值操作来覆盖要移除的元素，而不是删除它们。因此，容器的大小在调用remove之后不会改变，改变的只是元素的分布。如果需要实际删除元素，通常会在remove之后使用erase来删除newEnd到v.end()之间的元素

条款33：小心指针容器上的类似删除的算法。

永远担心指针。

第34项：注意哪些算法需要排序范围。

第36项：理解copy\_if的正确实现。

第37条：使用accumulate或for\_each来汇总范围。

第37条建议使用 accumulate 或 for\_each 来汇总范围，意味着你应该利用标准库提供的算法来执行这些常见操作，而不是手动编写循环。这样做有几个好处：

1. **代码清晰度**：使用标准算法可以使代码更简洁、更易于理解。
2. **减少错误**：标准算法经过了广泛的测试，使用它们可以减少引入错误的可能性。
3. **灵活性**：标准算法通常提供了更多的灵活性，比如可以很容易地更改操作或累加的方式。

总之，第37条是鼓励程序员使用标准库提供的工具来提高代码的质量和效率