**泛型算法(generic algorithm)**

大多数情况下，每个算法都需要使用(至少)两个迭代器来指出该算法操纵的元素范围，第一个迭代器指向第一个元素，第二个迭代器指向最后一个元素的下一位置，第二个迭代器(超出末端迭代器(off-the-end iterator))所指向的元素本身不是要操作的元素，而被用作终止遍历的哨兵(sentinel)

算法永不执行容器提供的操作

泛型算法本身从不执行容器操作，只是单独依赖迭代器和迭代器操作实现；算法也许会改变存储在容器中的元素的值，也许会在容器内移动元素，但算法从不直接添加或删除元素

迭代器类：插入器(inserter)，在给这类迭代器赋值时，在基础容器上将执行插入运算，若算法操纵这类迭代器，迭代器可能导致在容器中添加元素，但算法本身不这么做

使用泛型算法必须包含algorithm头文件

#include<algorithm>

标准库还定义了一组泛化的算术算法，其命名习惯与泛型算法相同，使用时必须包含numeric头文件

#include<numeric>

除少数情况外，所有算法都在一段范围内的元素上操作，称为“输入范围”，带有输入范围参数的算法总是使用头两个形参标记该范围，其分别指向第一个元素和最后一个元素的下一位置的迭代器

只读算法

只会读取其输入范围内的元素，而不会写这些元素

例：find、accumulate

accumulate函数头两个形参指定要累加的元素范围，第三个形参是累加的初值；将它的一个内部变量设置为指定的初值，在此初值上累加输入范围内所有元素的值；accumulate算法返回累加的结果，其返回值就是第三个实参的类型

用于指定累加起始值的第三个实参是必要的

accumulate对要累加的元素类型一无所知，1. 调用该函数时必须传递一个起始值，2. 容器内的元素类型必须与第三个实参的类型匹配，或者可以转换为第三个实参的类型

string sum = accumulate(v.begin(), v.end(), string(“”));

从空字符串开始，把v里的每个元素连接成一个字符串；程序显式地创建了一个string对象，用作该函数调用的第三个实参；传递一个字符串字面值，将导致编译时错误，因为此时累加和的类型是const char\*，而string的加法操作符所使用的操作数分别是string和const char\*类型，加法的结果将产生一个string对象，而不是const char\*指针