(0) An abstract vector \( V \) is a sequence of elements \([v_0, \ldots, v_{n-1}]\). An abstract vector iterator refers to a position in this sequence; in specifications, we can refer to “the iterator for position \( i \)” (for \( 0 \leq i \leq n \)).

Most STL containers offer insert and erase methods that take iterators as arguments and return iterators as results. For example, iterator \( \text{vector<T>::erase(iterator it)} \) erases the element at \( it \), shifts later elements down, and returns an iterator pointing at the formerly next element (the element that followed \( it \), which was shifted down one position by erase).

Write a specification for this function. Make sure to define its return value. Since erase mutates its container, use syntax like old end() and new end() to refer to values before and after the call.

(1) Use such erase methods to write a method with the following specification. Never use an invalid iterator— all iterators to a vector become invalid after any insert or erase—and your method should work for other STL containers, such as map and list.

```cpp
/** Erase all elements of @a x for which @a pred returns true.
 * @param [in , out] x Container of elements.
 * @param [in] pred Predicate that takes an element and returns a bool.
 *
 * @post new @a x. size() <= old @a x. size()
 * @post new @a x contains exactly those elements e of old @a x for
 *  which pred(e) returned false, and in the same relative order.
 */
template <typename Container, typename Predicate>
void erase_if(Container & x, Predicate pred);
```

(2) Consider a data type \( T \) with abstract type \( T = [t_0, \ldots, t_{n-1}] \), where each \( t_i \) is an integer. \( T \)'s iterator iterates over the values in order. Assume \( T \)'s representation is a std::vector<int> value. What is the likely complexity of \( T::iterator T::erase(T::iterator it) \)?

(3) Consider a data type \( U \) with abstract type \( U = \{u_0, \ldots, u_{n-1}\} \), where each \( u_i \) is an integer. \( U \)'s iterator iterates over the values in any order. Assume \( U \)'s representation is a std::vector<int> value. Write an implementation of \( U::iterator U::erase(U::iterator it) \) that has \( O(1) \) time complexity.