JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Introduction to Algorithms

This topic introduces Algorithms and at the end of of this topic, you will understand when and which iterators become invalidated as a result of insertion or erasure

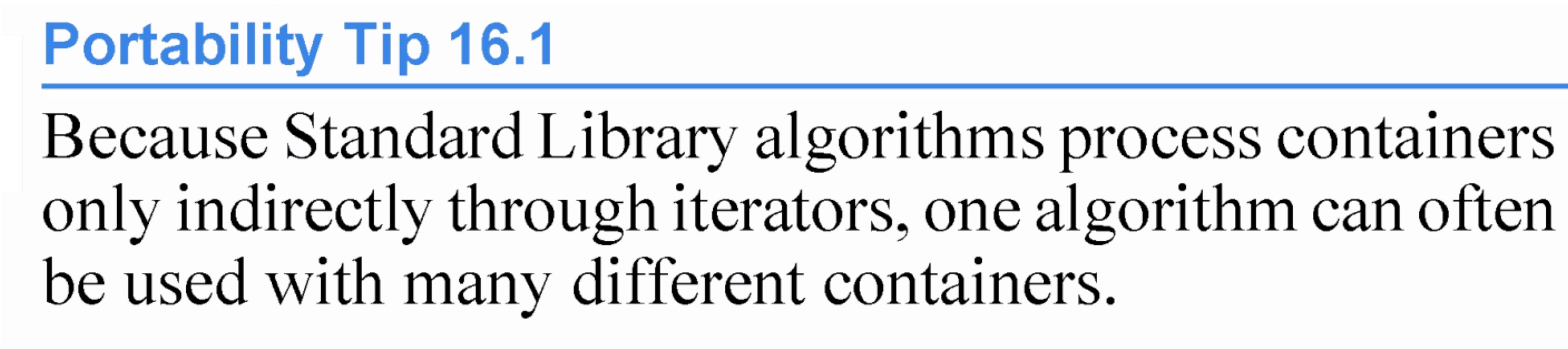
- The Standard Library provides over 90 algorithms, many of which are new in C++11
- Most of them use iterators to access container elements
- Various algorithms can receive a function pointer as an argument
  - Use the pointer to call the function—typically with one or two container elements as arguments.

- *With few exceptions, the Standard Library separates algorithms from containers.*
  - An important part of every container is the type of iterator it supports
  - This determines which algorithms can be applied to the container
  - For example, both vectors and arrays support random-access iterators
    - All Standard Library algorithms can operate on vectors and the ones that do not modify a container's size can also operate on arrays.

- Each Standard Library algorithm that takes iterator arguments requires those iterators to provide a minimum level of functionality.
  - If an algorithm requires a forward iterator
    - That algorithm can operate on any container that supports
      - *forward iterators*
      - *bidirectional iterators*
      - or *random-access iterators*

Software Engineering Observation 16.1

Standard Library algorithms do not depend on the implementation details of the containers on which they operate. As long as a container's (or built-in array's) iterators satisfy the requirements of an algorithm, the algorithm can work on the container.



Portability Tip 16.1

Because Standard Library algorithms process containers only indirectly through iterators, one algorithm can often be used with many different containers.

Software Engineering Observation 16.2

The Standard Library containers are implemented concisely. The algorithms are separated from the containers and operate on elements of the containers only indirectly through iterators. This separation makes it easier to write generic algorithms applicable to a variety of container classes.

JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Software Engineering Observation 16.3

Using the “weakest iterator” that yields acceptable performance helps produce maximally reusable components. For example, if an algorithm requires only forward iterators, it can be used with any container that supports forward iterators, bidirectional iterators or random-access iterators. However, an algorithm that requires random-access iterators can be used only with containers that have random-access iterators.

- Iterators simply *point* to container elements
  - So it's possible for iterators to become *invalid* when certain container modifications occur
  - If you invoke clear on a `vector`, *all* of its elements are *removed*
    - If a program had any iterators that pointed to that `vector`'s elements before `clear` was called
      - Those iterators would now be *invalid*

- When *inserting* into a:
  - **`Vector`**
    - If the vector is reallocated
      - All iterators pointing to that `vector` are invalidated
      - Iterators from the insertion point to the end of the `vector` are invalidated
  - **`Deque`**
    - All iterators are invalidated
  - **`list`** or **`forward_list`**
    - All iterators *remain valid*
  - **Ordered associative container**
    - All iterators *remain valid*
  - **Unordered associative container**
    - All iterators are invalidated if the containers need to be reallocated

- When *erasing* from a container
  - Iterators to the *erased* elements are invalidated
  - In addition:
    - `Vector`
      - Iterators from the erased element to the end of the `vector` are invalidated
    - `Deque`
      - If an element in the middle of the `deque` is erased
        - » All iterators are invalidated

This topic introduced Algorithms and the details of when and which iterators become invalidated as a result of insertion or erasure