
6.1. STL Components

The STL is based on the cooperation of various well-structured components, key of which are containers, iterators, and algorithms:

- **Containers** are used to manage collections of objects of a certain kind. Every kind of container has its own advantages and disadvantages, so having different container types reflects different requirements for collections in programs. The containers may be implemented as arrays or as linked lists, or they may have a special key for every element.
- **Iterators** are used to step through the elements of collections of objects. These collections may be containers or subsets of containers. The major advantage of iterators is that they offer a small but common interface for any arbitrary container type. For example, one operation of this interface lets the iterator step to the next element in the collection. This is done independently of the internal structure of the collection. Regardless of whether the collection is an array, a tree, or a hash table, it works. This is because every container class provides its own iterator type that simply "does the right thing" because it knows the internal structure of its container.

  The interface for iterators is almost the same as for ordinary pointers. To increment an iterator, you call operator `++`. To access the value of an iterator, you use operator `*`. So, you might consider an iterator a kind of a smart pointer that translates the call "go to the next element" into whatever is appropriate.
- **Algorithms** are used to process the elements of collections. For example, algorithms can search, sort, modify, or simply use the elements for various purposes. Algorithms use iterators. Thus, because the iterator interface for iterators is common for all container types, an algorithm has to be written only once to work with arbitrary containers.

  To give algorithms more flexibility, you can supply certain auxiliary functions called by the algorithms. Thus, you can use a general algorithm to suit your needs even if that need is very special or complex. For example, you can provide your own search criterion or a special operation to combine elements. Especially since C++11, with the introduction of lambdas, you can easily specify almost any kind of functionality while iterating over the elements of a container.

The concept of the STL is based on a separation of data and operations. The data is managed by container classes, and the operations are defined by configurable algorithms. Iterators are the glue between these two components. They let any algorithm interact with any container (Figure 6.1).

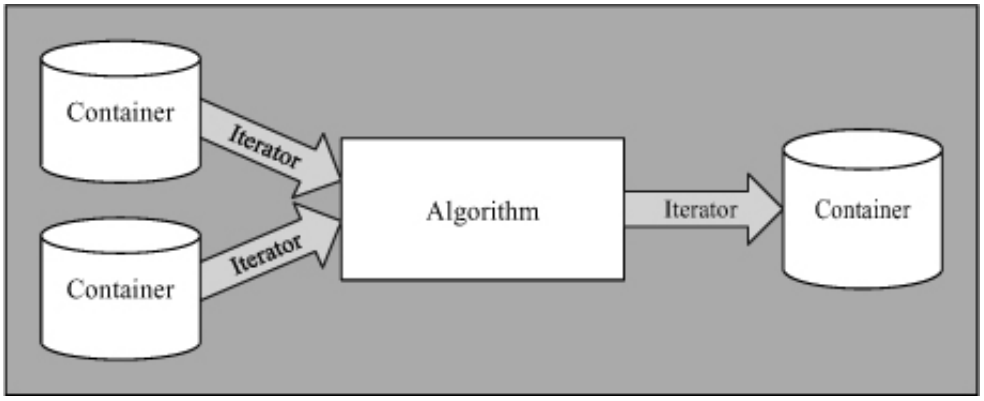

Figure 6.1. STL Components

In a way, the STL concept contradicts the original idea of object-oriented programming: The STL *separates* data and algorithms rather than combining them. However, the reason for doing so is very important. In principle, you can combine every kind of container with every kind of algorithm, so the result is a very flexible but still rather small framework.

One fundamental aspect of the STL is that all components work with arbitrary types. As the name "standard template library" indicates, all components are templates for any type, provided that type is able to perform the required operations. Thus, the STL is a good example of the concept of *generic programming*. Containers and algorithms are generic for arbitrary types and classes.

The STL provides even more generic components. By using certain *adapters* and *function objects* (or *functors*), you can supplement, constrain, or configure the algorithms and the interfaces for special needs. However, I'm jumping the gun. First, I want to explain the concept step-by-step by using examples. This is probably the best way to understand and become familiar with the STL.