**What are iterators? Why do we need them?**

You can think of an iterator as pointing to an item. For instance, all containers support a function called .begin(), which will return an iterator pointing to the beginning of the container (the first element) and function .end(), that returns an iterator corresponding to having reached the end of the container. You can access the element by "dereferencing" the iterator with a \*, just as you would dereference a pointer.

The special thing about iterators is that they provide the glue between STL algorithms and containers. For generic code, the recommendation would be to use a combination of STL algorithms (e.g. find, sort, remove, copy) etc. that carries out the computation on your data structure (vector, list, map etc.). For these algorithms to work, you need to supply them with iterators into your container.

For example: std::sort(my\_vec.begin(), my\_vec.end())

**Iterators make your code more generic.**  
Every standard library container provides an iterator hence if you change your container class in future the loop won’t be affected. Iterators work with all containers - indexes work only with some of the containers.

For example: the std::list interface intentionally does not offer operator[](size\_t n) because it would be O(n). So looping the list by indexes is impossible.

**1.**

**This is how iterators work for containers in C++:**

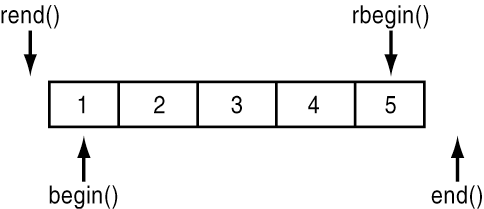
std::vector<int> example\_vec { 1, 2, 3, 4, 5 };

for(std::vector<int>::iterator it = example\_vec.begin(); it != example\_vec.end(); ++it)

std::cout << \*it << " "; //Outputs: 1 2 3 4 5

for(std::vector<int>::reverse\_iterator it = example\_vec.rbegin(); it != example\_vec.rend(); ++it)

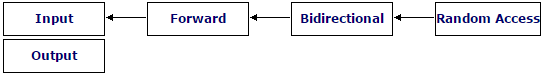
std::cout << \*it << " "; //Outputs: 5 4 3 2 1

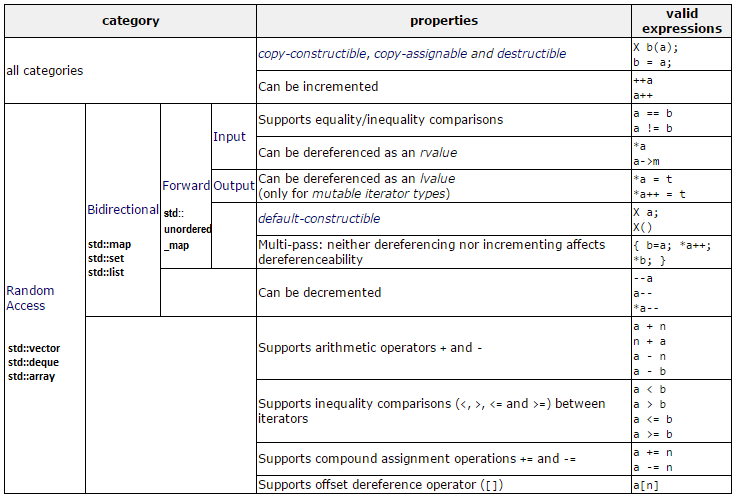


**2.**

**Advanced**

Iterators are classified into five categories depending on the functionality they implement.





**2.1**