Sequential Containers

> Provides access to **sequence** elements

Before we start, let's see the following example

```
#include <iostream>
#include <vector>
#include <algorithm>//for generate() and copy
#include <iterator>//for ostream_iterator

int randGenerator() {//a function that generate random number [0:100)
    return std::rand() % 100;
}

int main() {
    const int CAPACITY = 200;
    std::vector<int> v(CAPACITY);
    std::generate(v.begin(), v.end(), randGenerator);
    std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, ","));
}
```

This program will generate a vector with random numbers, sort it and print it to console

- `generate(ForwardIterator first, ForwardIterator last, Generator gen)`
  - `gen` is either a function pointer or function object
- `copy(InputIterator first, InputIterator last, OutputIterator result)`
  - Copies the elements in the range [first,last) into the range beginning at result.

std::vector<T>

A vector represents a sequence of elements of any type

```
v[i]//no bound checking
v.at(i)//has bound checking
```

Why doesn't std::vector bounds check by default?

- Because of the philosophy of C++: Only add features if they solve an actual problem
- If you write your program **correctly**, bounds checking will just **slow** your code down.

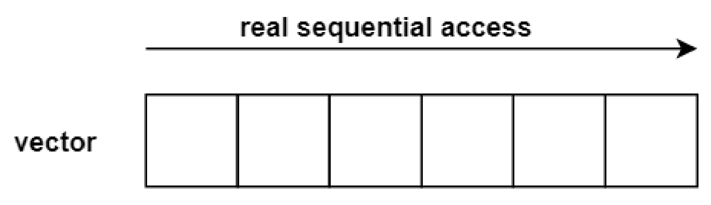
Vector only grows efficiently in one direction
Insertion at the **front** of std::vector: **O(N)**
Insertion at the **back** of std::vector: **O(1)**

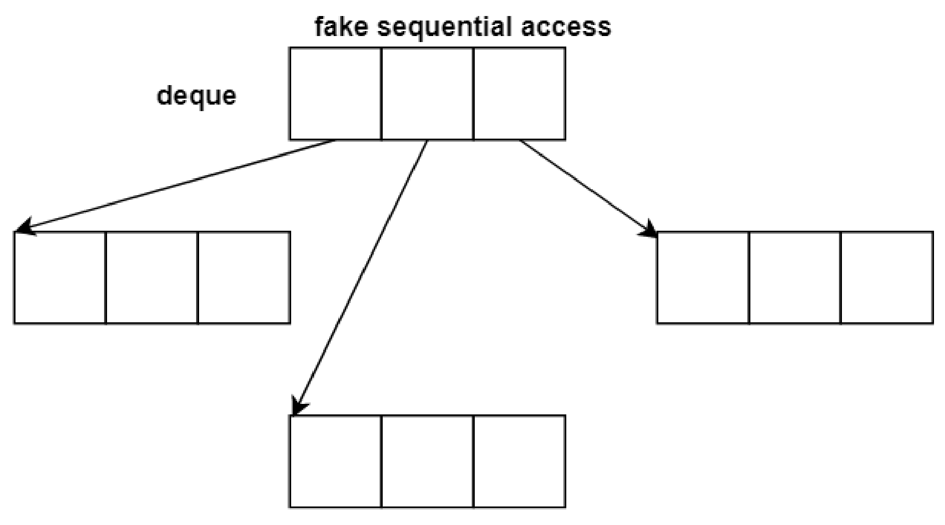
std::deque<T>

Can do everything vector can do, plus fast insertion and removal at the front

If deque can do everything a vector can do, and also provide fast insertion at the front, why use vector at all?

It's all about **Spatial Locality**
- vector preserves completely **Spatial Locality**
- deque does not





> "vector is the type of sequence that should be used by default...
> deque is the data structure of choice when most insertions and deletions take place at the beginning or at the end of the sequence."
> — C++ ISO Standard (section 23.1.1.2):