

COMP6771

Advanced C++ Programming

Week 2.2

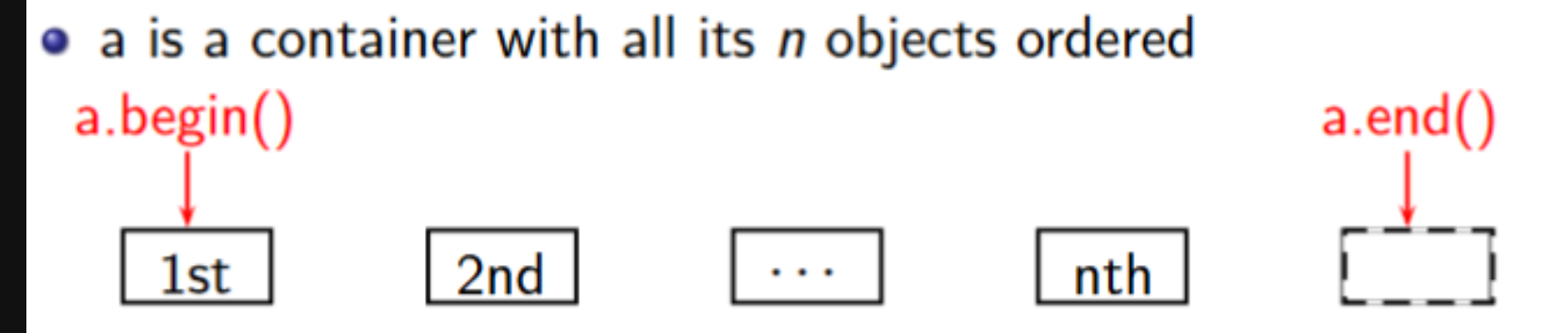
STL Iterators

STL: Iterators

- Iterator is an abstract notion of a **pointer**
- Iterators are types that abstract container data as a **sequence** of objects (i.e. linear)
- Iterators will allow us to connect a wide range of containers with a wide range of algorithms via a common interface

STL: Iterators

- **a.begin()**: abstractly "points" to the first element
- **a.end()**: abstractly "points" to one past the last element
 - a.end() is not an invalid iterator value
- If iter abstractly points to the **k-th** element, then:
 - ***p** is the object it abstractly points to
 - **++p** abstractly points to the (k + 1)-st element

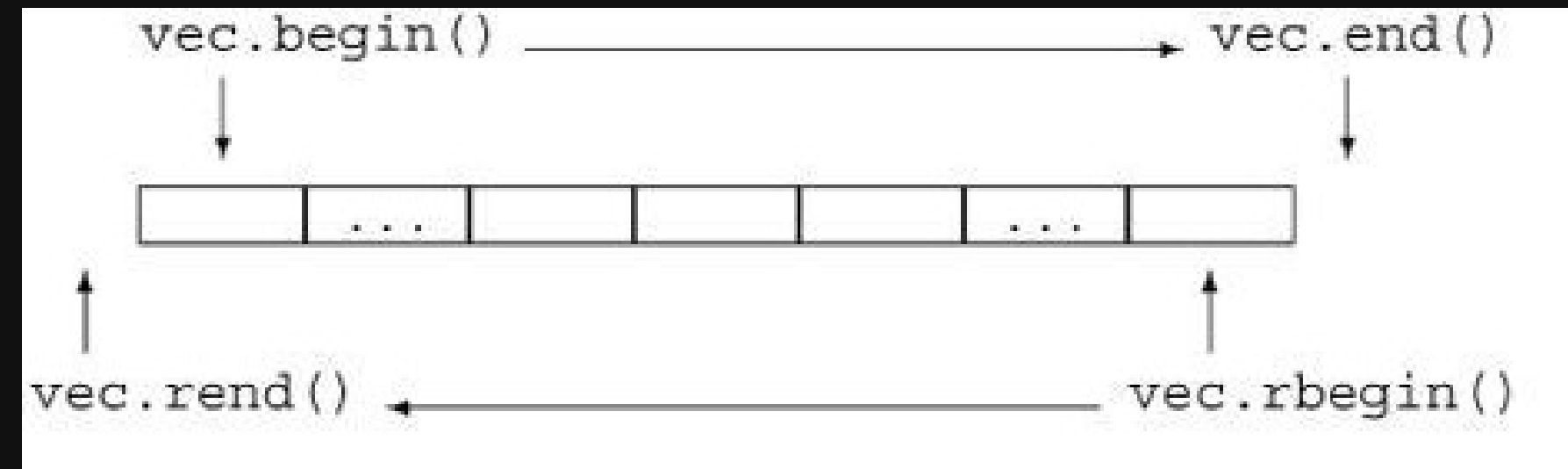


```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4
5 int main() {
6     std::vector<std::string> names;
7     for (auto iter = names.begin(); iter != names.end(); ++iter) {
8         std::cout << *iter << "\n";
9     }
10    for (std::vector<std::string>::iterator iter = names.begin(); iter != names.end(); ++iter) {
11        std::cout << *iter << "\n";
12    }
13 }
```

Iterators, Constness, Reverse

```
1 #include <iostream>
2 #include <vector>
3
4 int main() {
5     std::vector<int> ages;
6     ages.push_back(18);
7     ages.push_back(19);
8     ages.push_back(20);
9
10    // type of iter would be std::vector<int>::iterator
11    for (auto iter = ages.begin(); iter != ages.end(); ++iter) {
12        (*iter)++; // OK
13    }
14
15    // type of iter would be std::vector<int>::const_iterator
16    for (auto iter = ages.cbegin(); iter != ages.cend(); ++iter) {
17        //(*iter)++; // NOT OK
18    }
19
20    // type of iter would be std::vector<int>::reverse_iterator
21    for (auto iter = ages.rbegin(); iter != ages.rend(); ++iter) {
22        std::cout << *iter << "\n"; // prints 20, 19, 18
23    }
24
25    // Can also use crbegin and crend
26 }
```

demo205-iter2.cpp



Stream Iterators

```
1 #include <fstream>
2 #include <iostream>
3 #include <iterator>
4
5 int main() {
6     std::ifstream in("data.in");
7
8     std::istream_iterator<int> begin(in);
9     std::istream_iterator<int> end;
10    std::cout << *begin++ << "\n"; // read the first int
11
12    ++begin; // skip the 2nd int
13    std::cout << *begin++ << "\n"; // read the third int
14    while (begin != end) {
15        std::cout << *begin++ << "\n"; // read and print the rest
16    }
17 }
```

demo206-iter3.cpp