Lecture #15 | C++ standard library: container

SE271 Object-oriented Programming (2017)

Prof. Min-gyu Cho

Components of C++ standard library

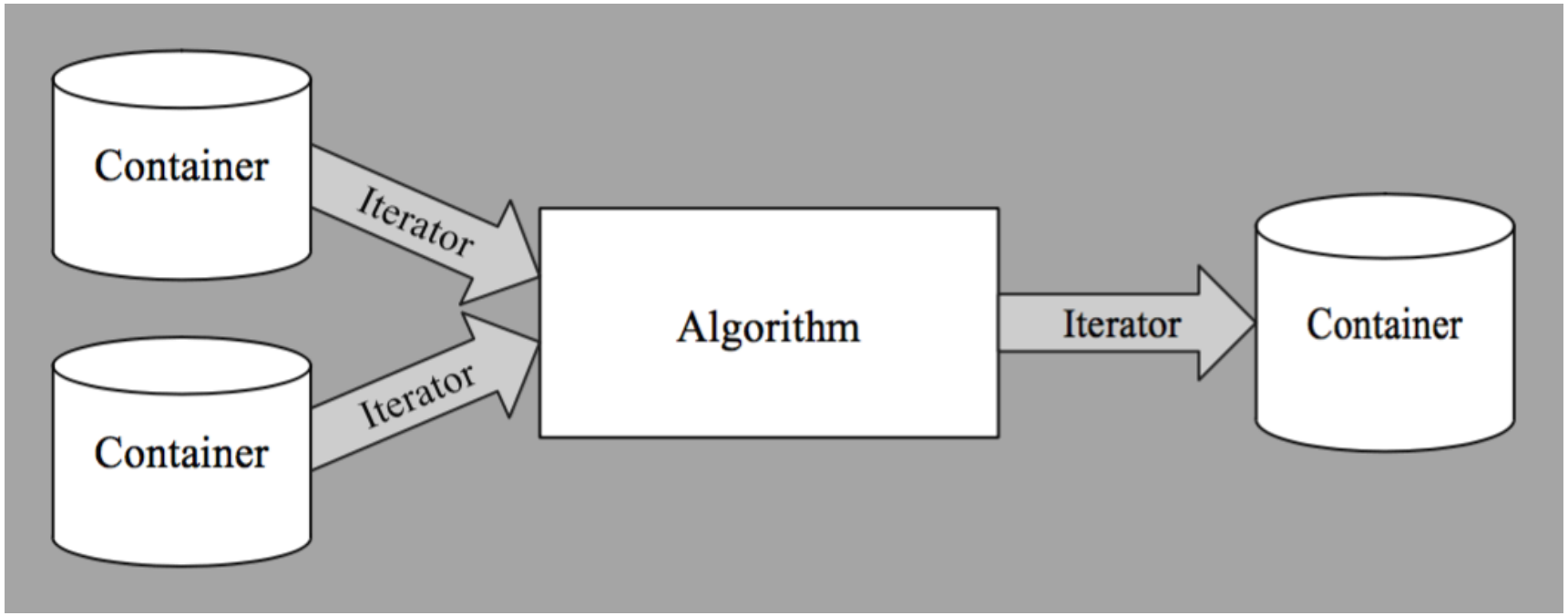
- Run-time language support (e.g., for allocation and run-time type information).
- The C standard library (w/minor modifications)
- Strings and regular expressions
- Input/output (I/O) streams
- Standard Template Libraries (STL)
  - *Containers* manage collection of objects (e.g., `vector`, `list`, `map`)
  - *Algorithms* process the elements in containers (e.g., `sort`, `search`)
  - *Iterators* step through the elements in containers
- Numerical computation (e.g., standard mathematical functions, complex numbers, vectors with arithmetic operations, and random number generators)
- Support for concurrent programming, including `threads` and locks
- Smart pointers and an interface to garbage collectors
- Special-purpose containers, such as `array`, `bitset` and `tuple`

Design consideration for C++ standard library

- Useful for almost C++ programmers (both novices & experts), including the builders of other libraries
- Efficient; incur (almost) no performance overhead (memory & time)
- Primitive and complete; do one thing very well
- Type safe by default
- Easy to work with built-in types and operators
- Supportive of commonly accepted programming style
- Extensible to handle user-defined types as built-in types and standard-library types are handled

STL components and interactions

- *Containers* manage collection of objects (e.g., `vector`, `list`, `map`)
- *Algorithms* process the elements in containers (e.g., `sort`, `search`)
- *Iterators* step through the elements in containers

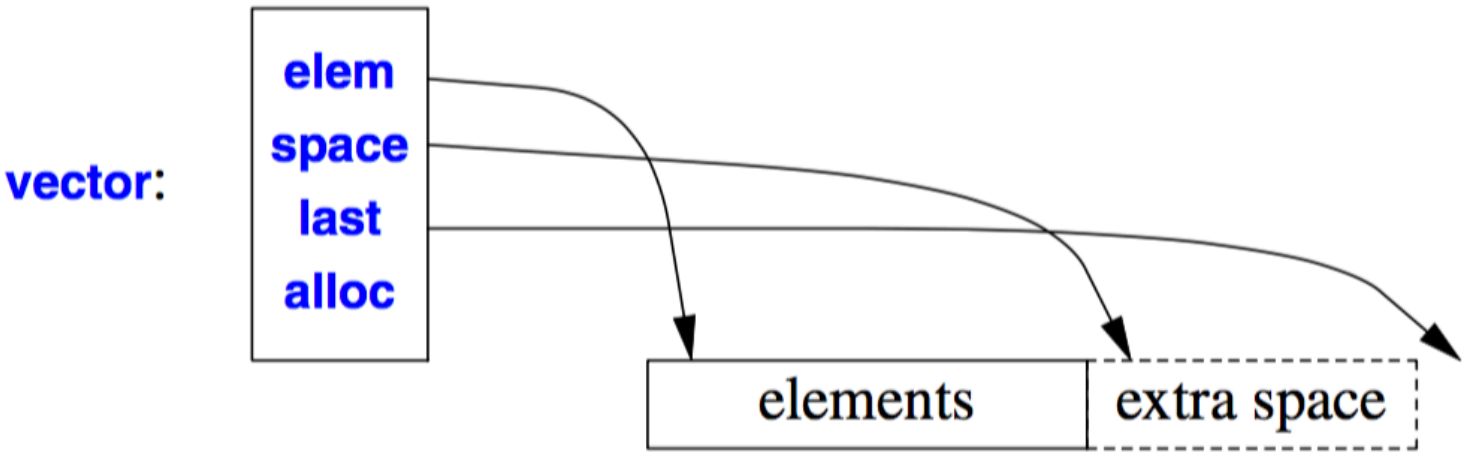


Containers

- Containers manage collection of the elements (of the same data type)
  - Containers share similar (but not exactly the same) interfaces
  - Each container has different data structure to store elements, different memory and time overhead
    →You need to select a proper container for your purpose
- Containers provide similar interfaces (i.e., public member functions) for generic programming as well as easy maintenance
- Types
  - Sequential: `vector, list, forward_list, deque, array`
  - Associative: `set, multiset, map, multimap`
  - Unordered: `unordered_set, multiset, unordered_map, multimap`

vector

- One of the most frequently used containers
- Similar to, but more convenient than C++ array
- Store elements of the same data type in a continuous memory space
- Dynamically increase or decrease reversed storage size



Example (initialization): vector

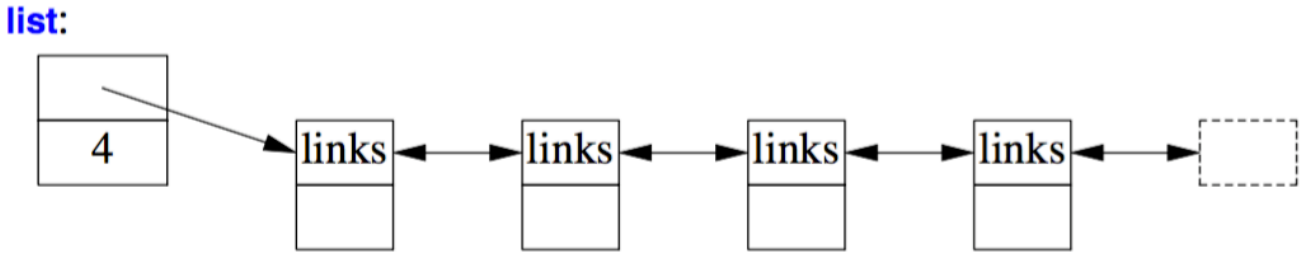
```
#include <iostream>
#include <vector>
using namespace std;
class Shape;

int main()
{
    vector<int> v1 {1, 2, 3, 4}; // size is 4
    vector<string> v2;           // size is 0
    vector<Shape*> v3(23);       // size is 23;
                                 // initial element value: nullptr
    vector<double> v4(32,9.9);   // size is 32;
                                 // initial element value: 9.9
}
```

Example: vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v1 {1, 2, 3, 4};
    v1.push_back(5);
    cout << v1.size() << " " << v1.capacity() << endl;
    v1.reserve(32);
    cout << v1.size() << " " << v1.capacity() << endl;
    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << " ";
    cout << endl;
}
```

list

- A doubly-linked list to store sequence of elements
- Cheap to add/remove element in any location
- Hard to randomly access elements




```
template<
    class T,
    class Allocator = std::allocator<T>
> class list;
```

Example: list

```
list<int> lst;
lst.push_back(42);
lst.push_back(23);

cout << "lst.front()=" << lst.front() << endl;
cout << "lst.back()=" << lst.back() << endl;

lst.pop_back();
cout << "lst.back()=" << lst.back() << endl;

cout << "lst.size()=" << lst.size() << endl;
cout << "lst.empty()=" << lst.empty() << endl;
lst.clear();
cout << "lst.size()=" << lst.size() << endl;
cout << "lst.empty()=" << lst.empty() << endl;
```

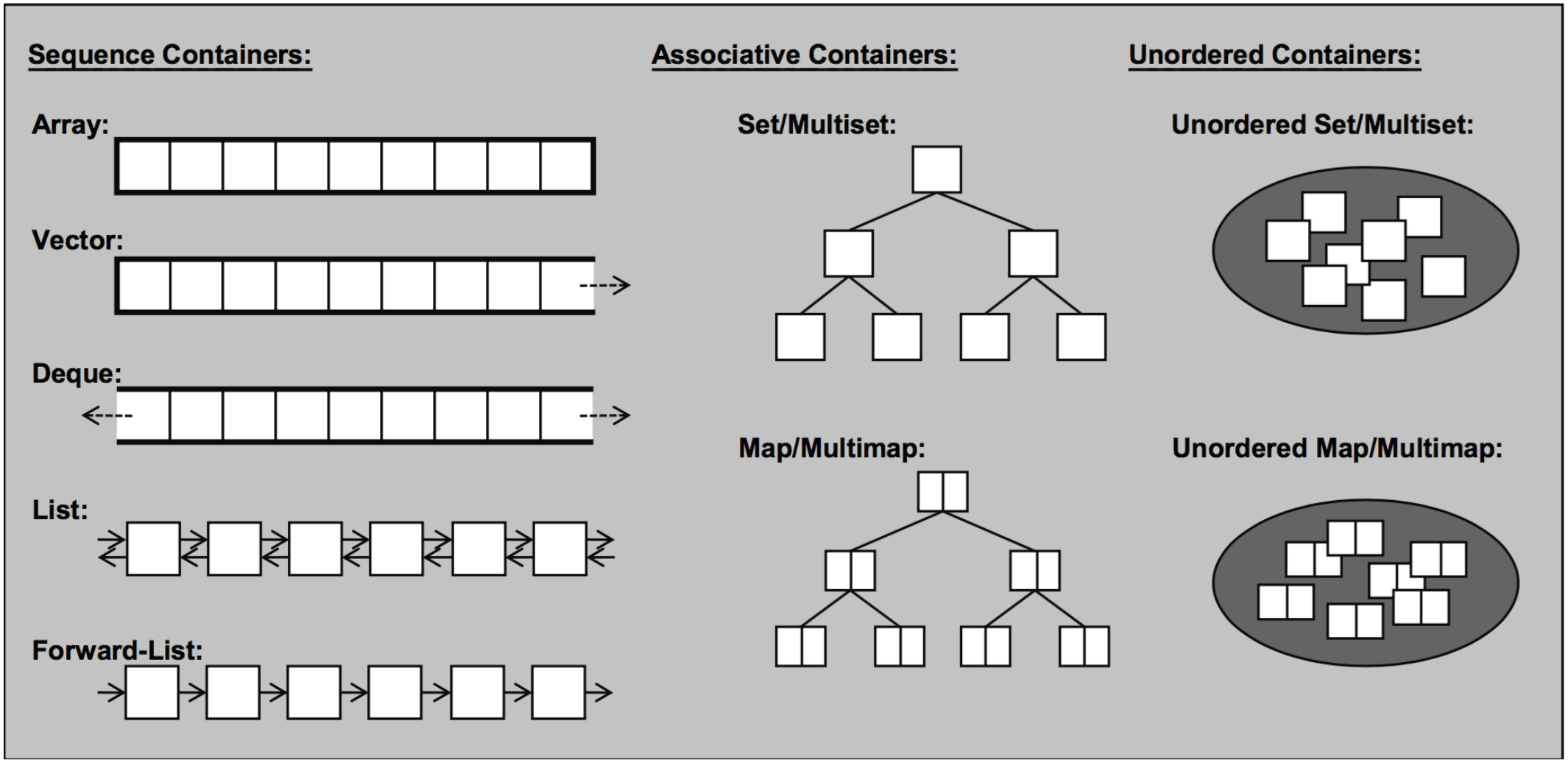
Example: list (cont.)

```
lst.push_back(42);
lst.push_front(23);
lst.push_back(6);
cout << "lst.front()=" << lst.front() << endl;
cout << "lst.back()=" << lst.back() << endl;

lst.pop_back();
cout << "lst.front()=" << lst.front() << endl;
cout << "lst.back()=" << lst.back() << endl;

lst.pop_front();
cout << "lst.front()=" << lst.front() << endl;
cout << "lst.back()=" << lst.back() << endl;
```

Standard Container Summary

Container	Internal data structure
`vector<T>`	A variable-size vector
`list<T>`	A doubly-linked list
`forward_list<T>`	A singly-linked list
`deque<T>`	A double-ended queue
`set<T>`	A set (a map with just a key and no value)
`multiset<T>`	A set in which a value can occur many times
`map<K,V>`	An associative array (§9.4)
`multimap<K,V>`	A map in which a key can occur many times
`unordered_map<K,V>`	A map using a hashed lookup (§9.5)
`unordered_multimap<K,V>`	A multimap using a hashed lookup
`unordered_set<T>`	A set using a hashed lookup
`unordered_multiset<T>`	A multiset using a hashed lookup

Standard Container Summary (Visualized)



* From “The C++ Standard Library: A tutorial and Reference (2nd edition)”

Standard Container Summary (complexity)

Container	Insertion	Access	Erase	Find	Persistent Iterators
vector / string	Back: O(1) or O(n) Other: O(n)	O(1)	Back: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	No
deque	Back/Front: O(1) Other: O(n)	O(1)	Back/Front: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	Pointers only
list / forward_list	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	O(n)	Yes
set / map	O(log n)	-	O(log n)	O(log n)	Yes
unordered_set / unordered_map	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	Pointers only
priority_queue	O(log n)	O(1)	O(log n)	-	-

Reading list

- Learn C++
  - C++ standard library: Ch. 9.8
- STL containers
  - List of member functions: http://en.cppreference.com/w/cpp/container
- (Advanced)
  - LLVM C++ standard library implementation: http://libcxx.llvm.org
    - Follow ‘building libc++’ link for download

ANY QUESTIONS?