

# ECE 3574: Applied Software Design: C++ Standard Library

Chris Wyatt

The goal of today's meeting is to review the standard library

- ▶ Containers and Iterators
- ▶ Algorithms
- ▶ Exercise 5

“The best code is that already written and tested”

## The C++ standard library is well-constructed and tested

- ▶ prefer to use containers and algorithms from the standard library rather than hand-coded data structures and algorithms.

In 2574 you saw how to implement data structures and common algorithms for sorting and searching. However, the C++ standard library provides implementations of these that are efficient and well tested, so you should prefer to use them over hand-coded approaches whenever feasible.

## std::array is a wrapper around raw arrays

- ▶ supports standard access members (at, [], front, back)
- ▶ has a size() member
- ▶ supports fill and swap
- ▶ can be empty
- ▶ very low overhead

Example:

```
std::array<int,10> a;  
a.fill(1);  
assert(a[3] == 1);  
assert(a.size() == 10);
```

## std::vector is a dynamically sized array-based container

- ▶ the most useful linear data structure
- ▶ see members size, capacity, and reserve
- ▶ grows exponentially
- ▶ supports insert - much more efficient than you might think
- ▶ watch out for iterator invalidation

### Example

```
std::vector<int> v;  
std::cout << v.capacity() << std::endl;  
for(int i = 0; i < 100; ++i){  
    v.push_back(i);  
    std::cout << v.capacity() << std::endl;  
}
```

## std::deque is a dynamically sized double ended queue

- ▶ not contiguous in memory
- ▶ access either end: push\_front or push\_back
- ▶ generally better performance than std::list

Example:

```
std::deque<int> d;
for(int i = 0; i < 100; ++i){
    d.push_back(i);
    d.push_front(i);
}
```

## std::list and std::forward\_list

- ▶ doubly and singly linked-lists respectively
- ▶ constant time insertion anywhere
- ▶ no random access
- ▶ std::list supports bidirectional iteration
- ▶ space efficient, no extra space as in std::vector
- ▶ less time efficient than std::vector because of cache misses

## adaptors provide wrappers around other containers

- ▶ `stack` (wraps a deque)
- ▶ `queue` (wraps a deque)
- ▶ `priority_queue` (a heap using vector for storage)



## std::map and std::multimap are dictionaries (key,value)

- ▶ std::map requires unique keys and value
- ▶ implemented as red-black tree (balanced binary tree)
- ▶ index operator[] is very handy

Example:

```
std::map<std::string, int> occurrences;
occurrences["hello"] += 1;
occurrences["hello"] += 1;
occurrences["goodbye"] += 1;

for(auto it = occurrences.begin();
    it != occurrences.end();
    ++it)
{
    std::cout << "You said " << it->first << " "
              << it->second << " times." << std::endl;
}
```

# Hash Tables

- ▶ unordered\_set / unordered\_map
- ▶ unordered\_multiset / unordered\_multimap
- ▶ constant (amortized) time find, insert, remove

## Same Example

```
std::unordered_map<std::string, int> occurrences;
occurrences["hello"] += 1;
occurrences["hello"] += 1;
occurrences["goodbye"] += 1;

for(auto it : occurrences)
{
    std::cout << "You said " << it->first << " "
              << it->second << " times." << std::endl;
}
```

## algorithms library

- ▶ Non-modifying sequence operations
- ▶ Modifying sequence operations
- ▶ Partitioning operations
- ▶ Binary search
- ▶ Set operations
- ▶ Heap operations
- ▶ min/max
- ▶ numeric (see random number generators too)

## Exercise 5

See Website

## Next Actions and Reminders

- ▶ Read The Pragmatic Programmer Sections 7, 8, 26
- ▶ Milestone 0 is due Friday.