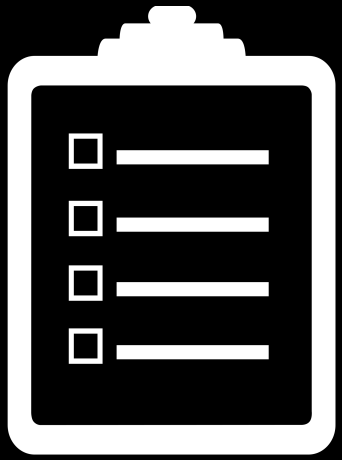
Associative Containers and Iterators

Ali Malik
malikali@stanford.edu

Game Plan



- Recap
- Associative Containers
- Iterators
- Map Iterators
- The `auto` keyword (maybe)
- Range-Based `for` Loops (maybe)

Recap

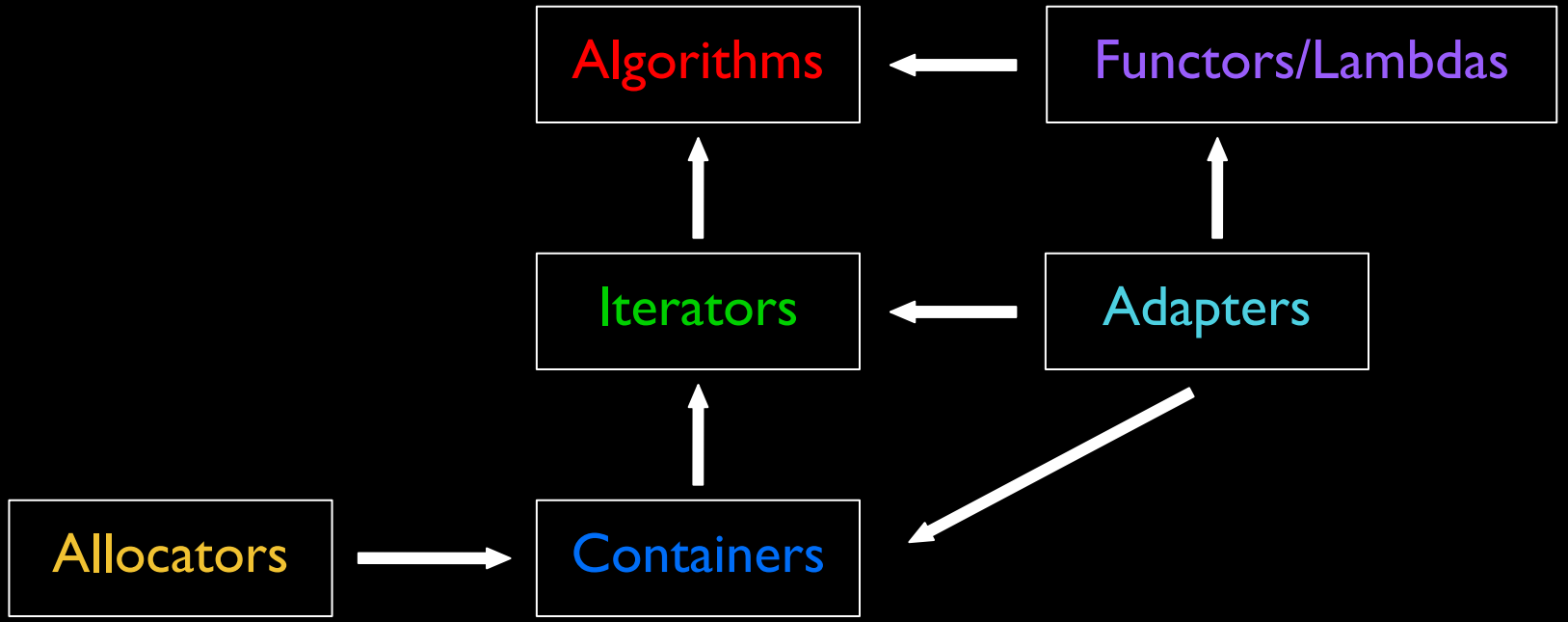
Structs

You can define your own mini-types that bundle multiple variables together:

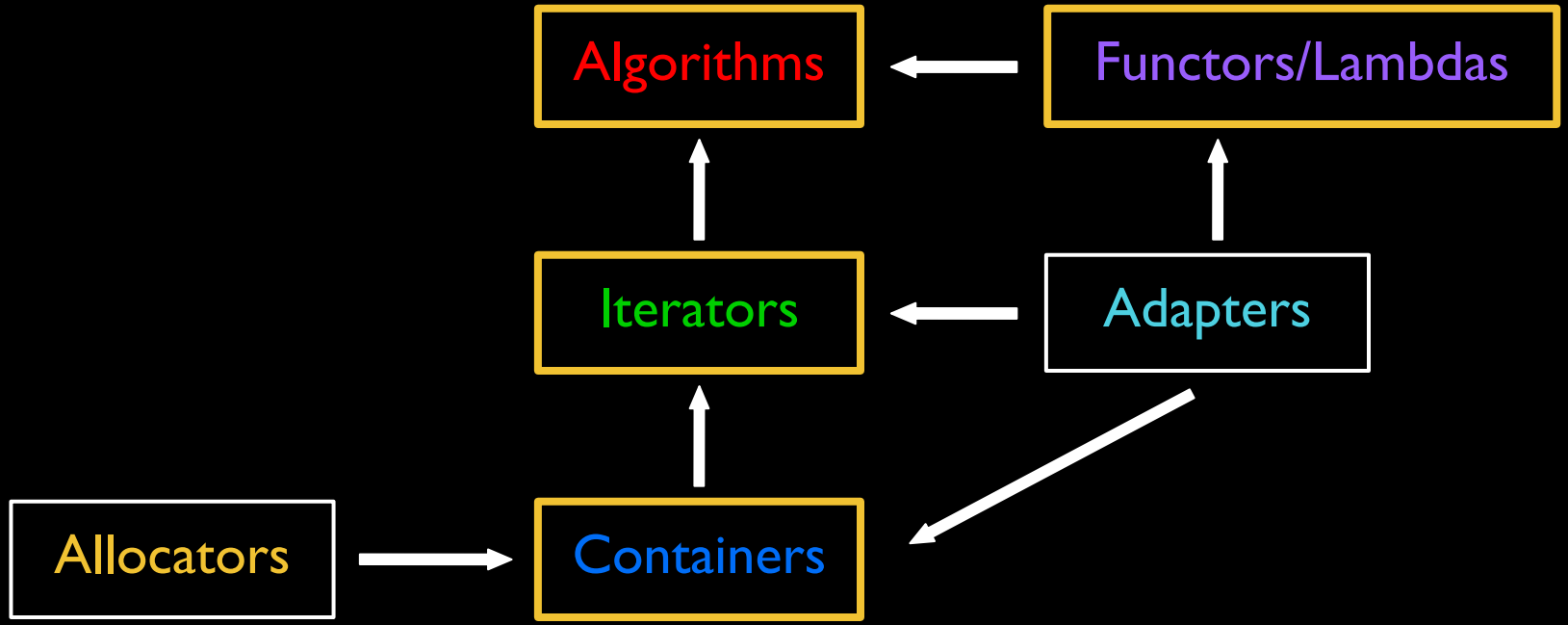
```
struct point {
    int x;
    int y;
};
```


Useful for Assignment 1

Overview of STL



Overview of STL



Sequence Containers

Provides access to sequences of elements.

Examples:

- `std::vector<T>`
- `std::list<T>`
- `std::deque<T>`

std::vector<T>

Some new stuff there:

```
const int kNumInts = 5;

using vecsz_t = std::vector<int>::size_type;

std::sort(vec.begin(), vec.end());
```

This let's us use `vecsz_t` as an alias/synonym for the type `std::vector<int>::size_type;`

`std::deque<T>`

A deque (pronounced “deck”) is a double ended queue.

Can do everything a vector can do

and also...

Unlike a vector, it is possible (and fast) to `push_front` and `pop_front`.

Container Adapters

How can we implement stack and queue using the containers we have?

Stack:

Just limit the functionality of a vector/deque to only allow `push_back` and `pop_back`.

Queue:

Just limit the functionality of a deque to only allow `push_back` and `pop_front`.

Plus only allow access to top element

Associative Containers

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- `vector<string>`, `vector<int>`

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- ~~`vector<string>, vector<int>`~~ ← We would need to keep two vectors with indexes denoting pairs

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- ~~`vector<string>`, `vector<int>`~~

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- ~~`vector<string>, vector<int>`~~
- `vector<std::pair<string,int>>`

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- ~~`vector<string>, vector<int>`~~
- ~~`vector<std::pair<string,int>>`~~ ← No quick way to lookup based on word

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- ~~`vector<string>, vector<int>`~~
- ~~`vector<std::pair<string,int>>`~~

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- ~~vector<string>, vector<int>~~
- ~~vector<std::pair<string, int>>~~
- `std::map<string, int>`

Associative Container

Scenario:

You want to count the frequency of words in a file.

What do you use?

- ~~`vector<string>, vector<int>`~~
- ~~`vector<std::pair<string,int>>`~~
- `std::map<string, int>`

Associative Container

It is useful to have a data structure that can associate values.

Associative containers do exactly that!

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

Based on ordering property of keys.

Keys need to be comparable using `<` (less than) operator.

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

Based on hash function. You need to define how the key can be hashed.

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

Associative Container

Have no idea of a sequence.

Data is accessed using the key instead of indexes.

- `std::map<T1, T2>`
- `std::set<T>`
- `std::unordered_map<T1, T2>`
- `std::unordered_set<T>`

You can define < and hash function operators for your own classes!

`std::map<T1, T2>`

Methods mostly same as Stanford map.

Check documentation for full list of methods.

We can do a small example, counting frequency of words in a file:

Map Example
(`MapExample.pro`)

std::set<T>

Methods mostly same as Stanford map.

Check documentation for full list of methods.

Key point:

A set is just a specific case of a map that doesn't have a value.

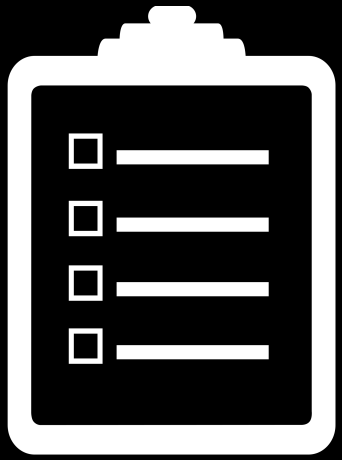
We can do a small example of adding and removing stuff:

Set Example

(SetExample.pro)

Announcements

Game Plan



- Recap
- Associative Containers
- Iterators
- Map Iterator
- The `auto` keyword
- Range-Based `for` Loops

Iterators

How do we iterate over associative containers?

Remember:

Assoc. containers have no notion of a sequence/indexing

```
for(int i = umm?; i < uhh?; i++ maybe?) {
```

Iterators

How do we iterate over associative containers?

Remember:

Assoc. containers have no notion of a sequence/indexing

~~`for(int i = umm?; i < uhh?; i++ maybe?) {`~~

Iterators

How do we iterate over associative containers?

Remember:

Assoc. containers have no notion of a sequence/indexing

~~`for(int i = umm?; i < uhh?; i++ maybe?) {`~~

C++ has a solution!

Iterators

First: A note

A note

We are going on a journey.

A note

FORODWAITH
ERED MITHRIN
IRON HILLS
RHÛN
The Lost Realm of Arnor
Here was of old the Witch-realm of Angmar
Carn Dûm
ERED LUIN
ERIADOR
THE SHIRE
Hobbiton
Bree
Weathertop (Amon Sûl)
Rivendell
MIRKWOOD
RHOVANION
Erebor
The Lonely Mountain
Esgaroth
The Long Lake
Sea of Rhûn
Moria Gate
LÓRIEN
Isengard
FANGORN
ROHAN
ENEDWAITH
MINHIRIATH
CARDOLAN
Tharbad
Dunland
The BROWN LANDS
Dagorlad (Battle Plain)
ERED LITHUI (Ash Mountains)
ERED NIMRAIS
GONDOR
Minas Tirith
Barad-dûr
MORDOR
NURN
Sea of Nûrnen
EPHEL DUATH
Anfalas (Langstrand)
Andrast (Ras Morthil)
Pinnath Gelin
Dol Amroth
Belfalas
Tolfalas
South Gondor (now a debatable and desert land)
NEAR HARAD
Harad Road
Forlindon
Harlindon
Gulf of Lhûn
Grey Havens
Himling
Icebay of Forochel
N
The Realm of Middle Earth
Miles
50
100
150
200

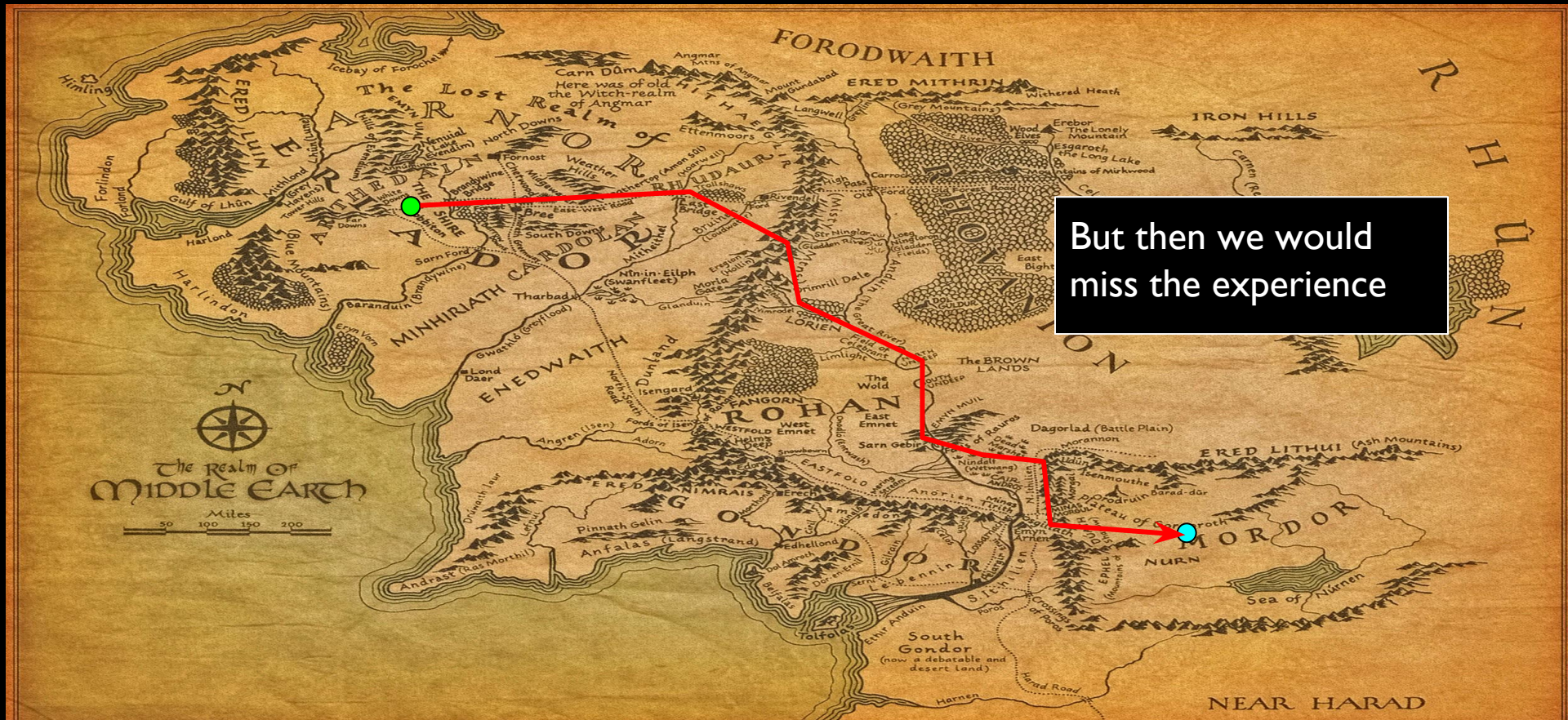
A note

We could in theory fly to the end destination.

A note


FORODWAITH
ERED MITHRIN
IRON HILLS
RHÛN
Sea of Rhûn
The Lost Realm of Angmar
Here was of old the Witch-realm of Angmar
Carn Dûm
Himling
Icebay of Forochel
ERED LUIN
Forlindon
Forlond
Gulf of Lhûn
Harlond
Harlindon
ARNOR
ERIADOR
ARTHEDAIN
THE SHIRE
Hobbiton
Fornost
North Downs
Weather Hills
Weathertop (Amon Sûl)
Bree
South Downs
East-West Road
Ettenmoors
RHUDAUR
Rivendell
CARDOLAN
Sarn Ford
Baranduin (Brandywine)
MINHIRIATH
Tharbad
Nîn-in-Eilph (Swanfleet)
Moria Gate
Drimrill Dale
LÓRIEN
Gwathló (Greyflood)
Lond Daer
ENEDWAITH
Dunland
Isengard
FANGORN
ROHAN
The Wold
East Emnet
West Emnet
Westfold
Helms Deep
Fords of Isen
Angren (Isen)
Adorn
Sarn Gebir
EASTFOLD
Anórien
Minas Tirith
ERED NIMRAIS
GONDOR
Pinnath Gelin
Anfalas (Langstrand)
Andrast (Ras Morthil)
Edhellond
Dol Amroth
Belfalas
Tolfalas
Ethir Anduin
Lebennin
South Gondor (now a debatable and desert land)
Harad Road
Harnen
NEAR HARAD
Erebor
The Lonely Mountain
Esgaroth
The Long Lake
Celduin (River Running)
Carnen (Redwater)
RHOVANION
MIRKWOOD
East Bight
The BROWN LANDS
Anduin (The Great River)
Field of Celebrant
Limlight
EMYN MUIL
Dagorlad (Battle Plain)
Dead Marshes
Morannon
ERED LITHUI (Ash Mountains)
Barad-dûr
Plateau of Gorgoroth
MORDOR
NURN
Sea of Núrnen
Crossings of Poros
N
The Realm of Middle Earth
Miles
50 100 150 200

A note


But then we would miss the experience
FORODWAITH
ERED MITHRIN
IRON HILLS
The Lost Realm of Angmar
Carn Dûm
Here was of old the Witch-realm of Angmar
ARNOR
ERIADOR
ARTHEDAIN
RHUDAUR
CARDOLAN
MINHIRIATH
ENEDWAITH
ROHAN
GONDOR
MORDOR
The BROWN LANDS
ERED LITHUI (Ash Mountains)
ERED NIMRAIS
Dagorlad (Battle Plain)
Barad-dûr
NURN
Sea of Nurnen
South Gondor
(now a debatable and desert land)
NEAR HARAD
The Realm of MIDDLE EARTH
Miles
50 100 150 200

A note

We are going on a journey.

A note

We are going on a journey.

At the end lies simplicity, but if we jump to it we miss out on understanding.

A note

We are going on a journey.

At the end lies simplicity, but if we jump to it we miss out on understanding.

So we will walk to it.

Iterators

Iterators

Iterators allow iteration over any container,

whether it is ordered or not.

Iterators

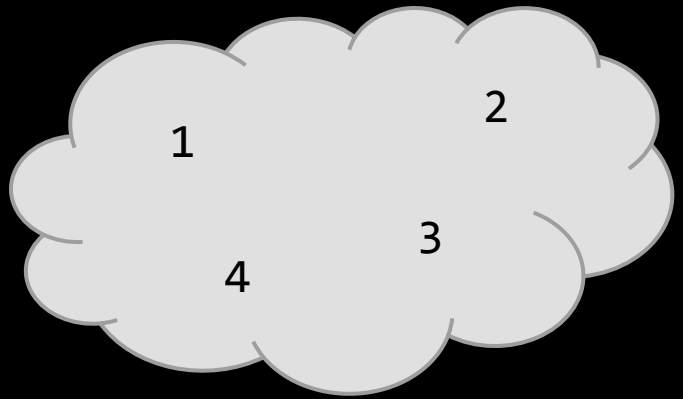
Let’s try and get a mental model of iterators:

Say we have a `std::set<int> mySet`

Iterators

Let’s try and get a mental model of iterators:

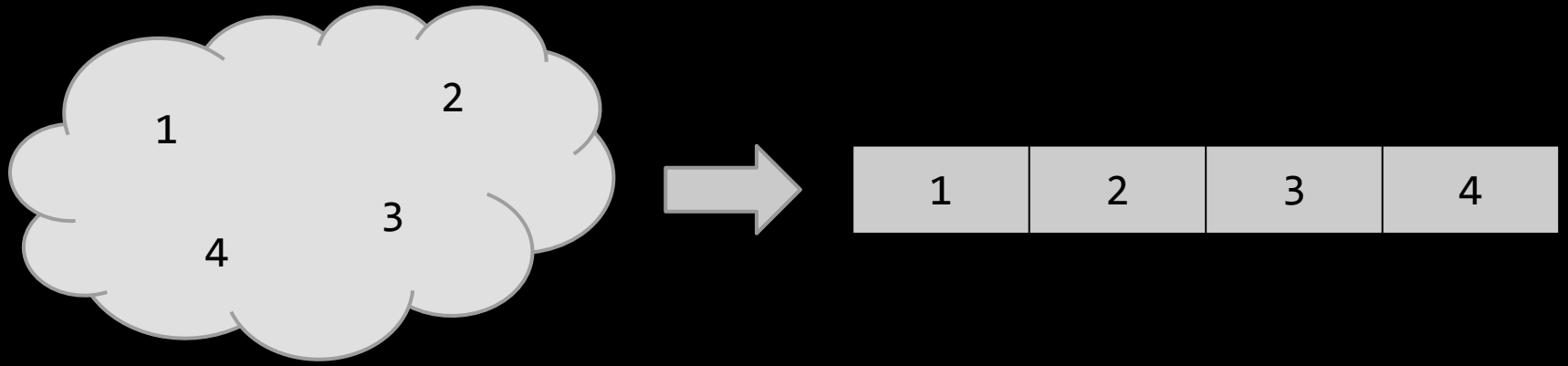
Say we have a `std::set<int> mySet`



Iterators

Let's try and get a mental model of iterators:

Say we have a `std::set<int> mySet`

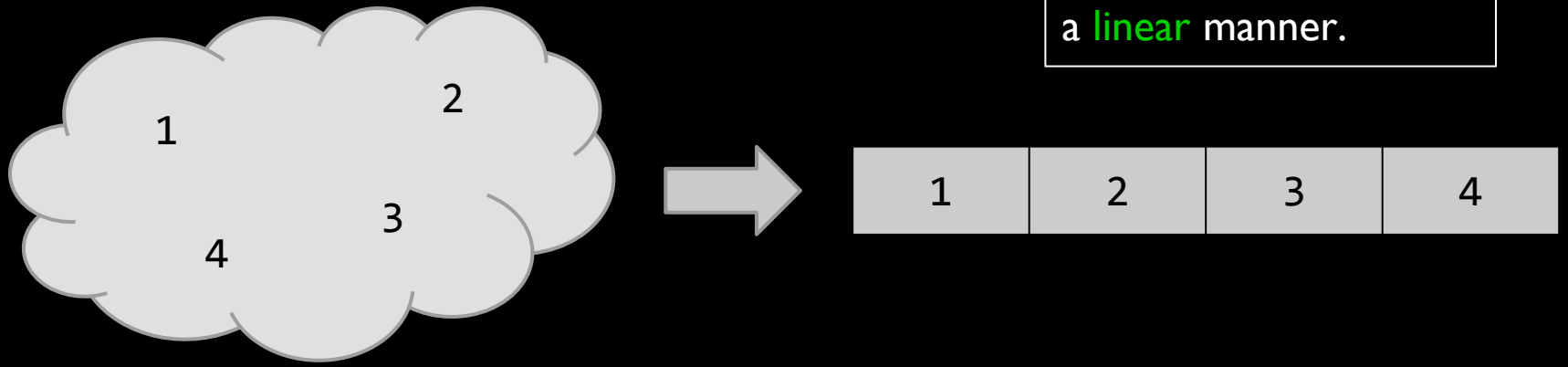


Iterators

Let's try and get a mental model of iterators:

Say we have a `std::set<int> mySet`

Iterators let us view a non-linear collection in a linear manner.



Iterators

How do they work?

We don’t care right now.

We will just use them like any other thing - assume they just work somehow.

Iterators - Usage

Let’s try and get a mental model of iterators:

Iterators - Usage

Let’s try and get a mental model of iterators:

We can get an iterator pointing to the “start” of the sequence by calling `mySet.begin()`


```
mySet.begin();
```

Iterators - Usage

Let’s try and get a mental model of iterators:


```
mySet.begin();
```

Iterators - Usage

Let’s try and get a mental model of iterators:

How do we store it in a variable?

```
mySet.begin();
```

Iterators - Usage

Let’s try and get a mental model of iterators:


```
??? iter = mySet.begin();
```

Iterators - Usage

Let’s try and get a mental model of iterators:

What is the type of the iterator?

```
??? iter = mySet.begin();
```

Iterators - Usage

Let's try and get a mental model of iterators:

What is the type of the iterator?

```
??? iter = mySet.begin();
```

```
set<int> mySet;
mySet.beg
    begin   iterator begin()
```

Iterators - Usage

Let’s try and get a mental model of iterators:


```
??? iter = mySet.begin();
```

Iterators - Usage

Let’s try and get a mental model of iterators:


```
set<int>::iterator iter = mySet.begin();
```

Iterators - Usage

Let's try and get a mental model of iterators:

It is the `iterator` type defined in the `set<int>` class!


```
set<int>::iterator iter = mySet.begin();
```

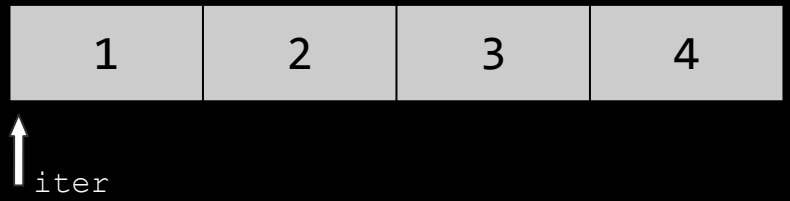
Iterators - Usage

Let’s try and get a mental model of iterators:


```
set<int>::iterator iter = mySet.begin();
```

Iterators - Usage

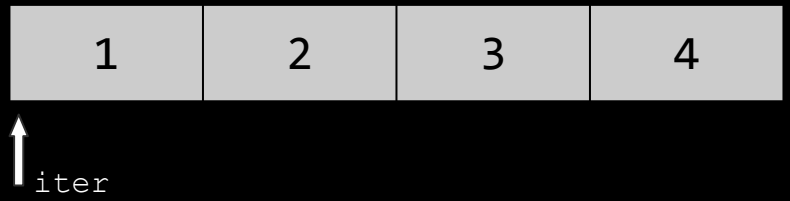
Let’s try and get a mental model of iterators:



Iterators - Usage

Let’s try and get a mental model of iterators:

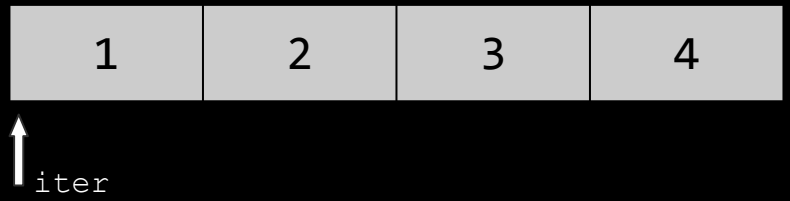
We can get the value of an iterator by using the dereference * operator.



Iterators - Usage

Let's try and get a mental model of iterators:

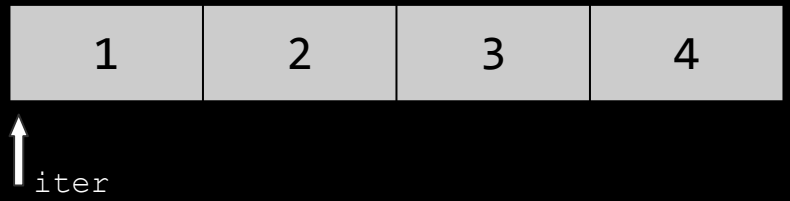
We can get the value of an iterator by using the dereference * operator.




```
cout << *iter << endl;     // prints 1
```

Iterators - Usage

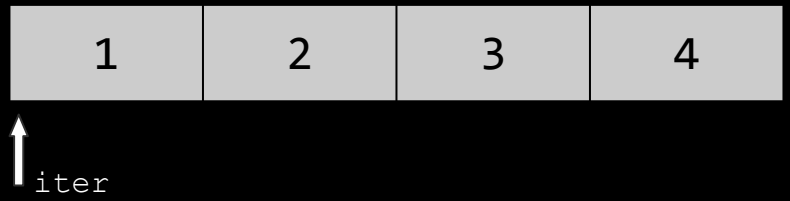
Let’s try and get a mental model of iterators:



Iterators - Usage

Let’s try and get a mental model of iterators:

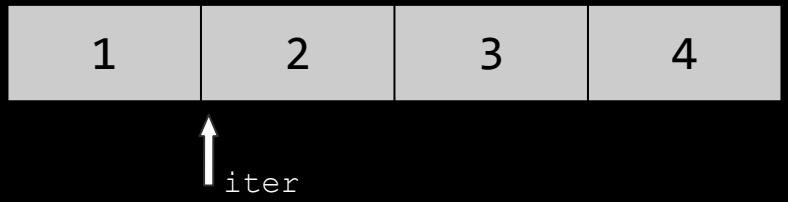
We can advance the iterator one by using the ++ operator (prefix)



Iterators - Usage

Let’s try and get a mental model of iterators:

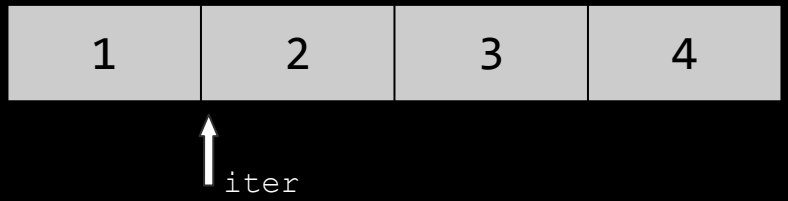
We can advance the iterator one by using the ++ operator (prefix)




```
++iter;        // advances iterator
```

Iterators - Usage

Let’s try and get a mental model of iterators:

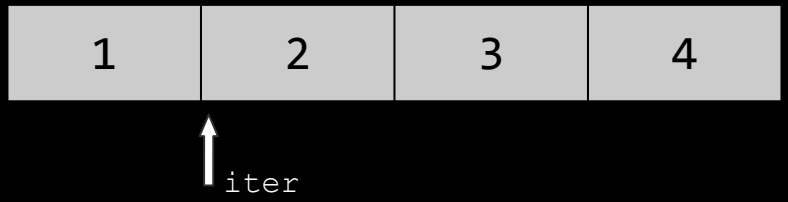


And so on...

Iterators - Usage

Let’s try and get a mental model of iterators:

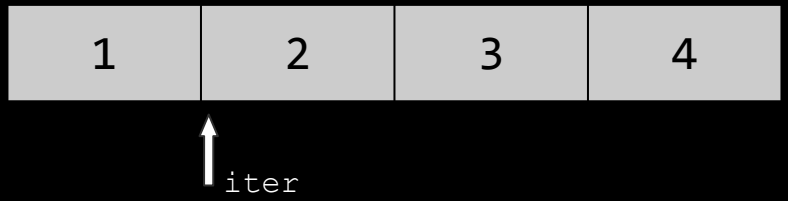
And so on...




```
cout << *iter << endl;     // prints 2
```

Iterators - Usage

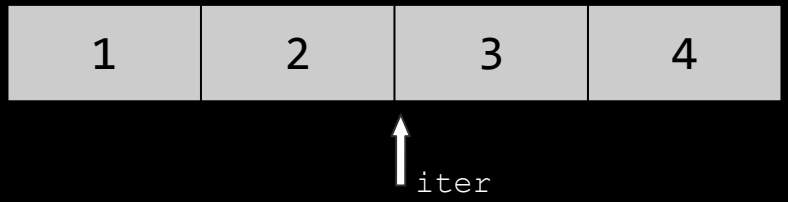
Let’s try and get a mental model of iterators:



And so on...

Iterators - Usage

Let’s try and get a mental model of iterators:

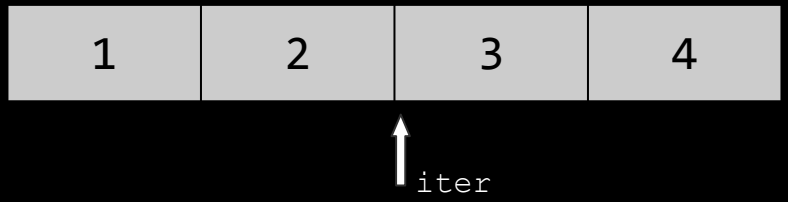


And so on...

```
++iter;        // advances iterator
```

Iterators - Usage

Let’s try and get a mental model of iterators:

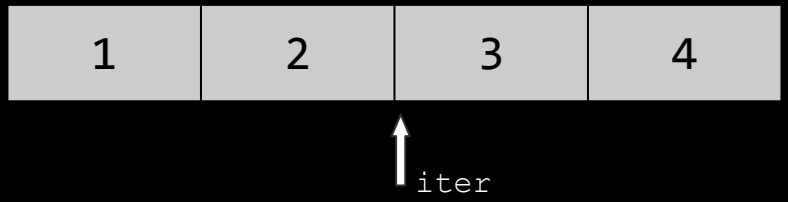


And so on...

Iterators - Usage

Let’s try and get a mental model of iterators:

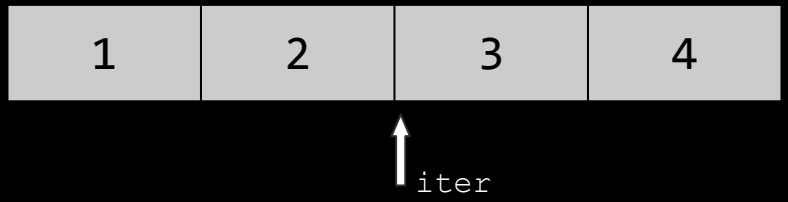
And so on...




```
cout << *iter << endl;      // prints 3
```

Iterators - Usage

Let’s try and get a mental model of iterators:

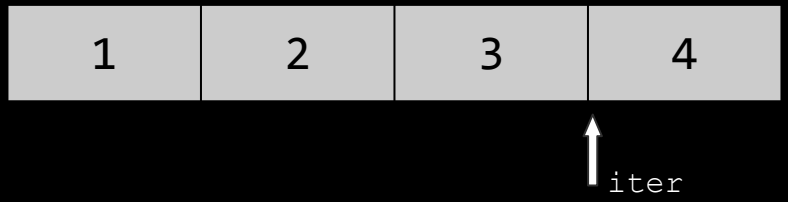


And so on...

Iterators - Usage

Let’s try and get a mental model of iterators:

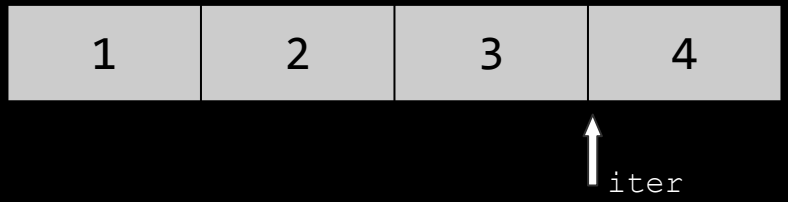
And so on...




```
++iter;        // advances iterator
```

Iterators - Usage

Let’s try and get a mental model of iterators:

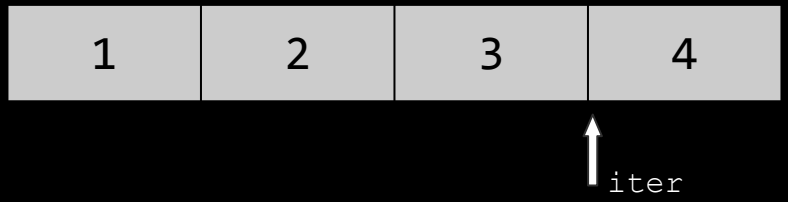


And so on...

Iterators - Usage

Let’s try and get a mental model of iterators:

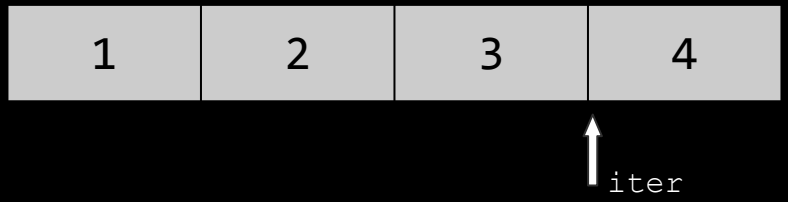
And so on...




```
cout << *iter << endl;     // prints 4
```

Iterators - Usage

Let’s try and get a mental model of iterators:

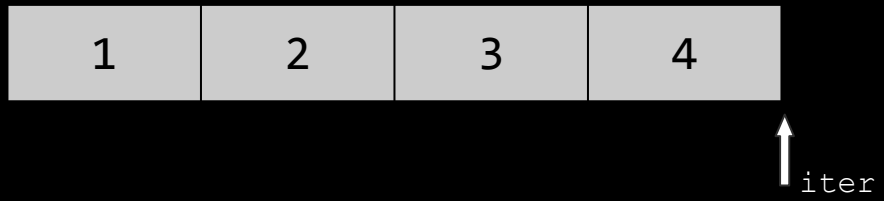


And so on...

Iterators - Usage

Let’s try and get a mental model of iterators:

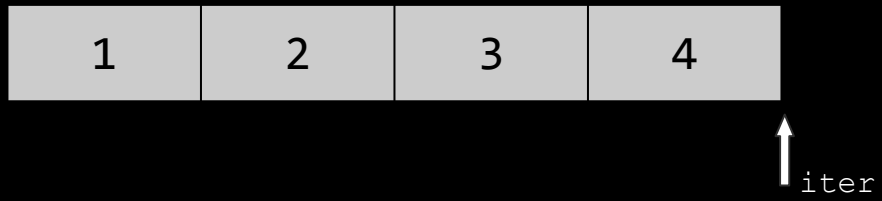
And so on...




```
++iter;       // advances iterator
```

Iterators - Usage

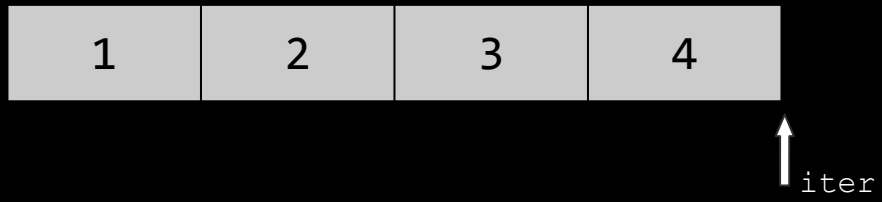
Let’s try and get a mental model of iterators:



Iterators - Usage

Let’s try and get a mental model of iterators:

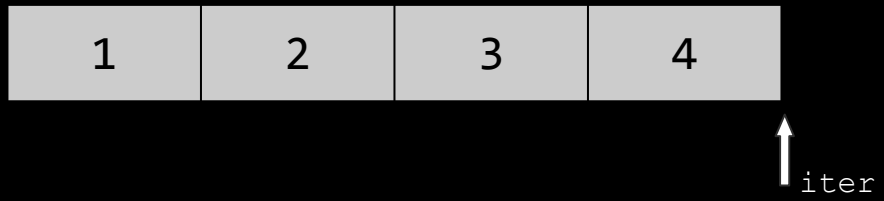
We can check if we have hit the end by comparing to `mySet.end()`



Iterators - Usage

Let’s try and get a mental model of iterators:

We can check if we have hit the end by comparing to `mySet.end()`




```
if(iter == mySet.end()) return;
```

Iterators - Usage

A summary of the essential iterator operations:

- Create iterator
- Dereference iterator to read value currently pointed to
- Advance iterator
- Compare against another iterator (especially `.end()` iterator)

Iterators - Usage

Let’s do some examples:

Basic Iterator

(`BasicIter.pro`)

Iterators

Our examples have used sets, but (almost) all C++ containers have iterators.

Why is this powerful?

- Many scenarios require looking at elements, regardless of what type of container is storing those elements.
- Iterators let us go through sequences of elements in a standardised way.

Iterators

Example (find number occurrences):

```
int numOccurences(vector<int>& cont, int elemToCount) {
    int counter = 0;
    vector<int>::iterator iter;
    for(iter = cont.begin(); iter != cont.end(); ++iter) {
        if(*iter == elemToCount)
            ++counter;
    }
    return counter;
}
```

Iterators

Example (find number occurrences):

Can I make this work for `std::list<int>`?

```
int numOccurences(vector<int>& cont, int elemToCount) {
    int counter = 0;
    vector<int>::iterator iter;
    for(iter = cont.begin(); iter != cont.end(); ++iter) {
        if(*iter == elemToCount)
            ++counter;
    }
    return counter;
}
```

Iterators

Example (find number occurrences):

Can I make this work for `std::list<int>`?

```
int numOccurences(vector<int>& cont, int elemToCount) {
    int counter = 0;
    vector<int>::iterator iter;
    for(iter = cont.begin(); iter != cont.end(); ++iter) {
        if(*iter == elemToCount)
            ++counter;
    }
    return counter;
}
```

Iterators

Example (find number occurrences):

```
int numOccurences(list<int>& cont, int elemToCount) {
    int counter = 0;
    list<int>::iterator iter;
    for(iter = cont.begin(); iter != cont.end(); ++iter) {
        if(*iter == elemToCount)
            ++counter;
    }
    return counter;
}
```

Iterators

This standard interface for looping through things is going to be really powerful.

We will cover it sometime this week or next week!

Map Iterators

Map Iterators

Map iterators are slightly different because we have both keys and values.

The iterator of a `map<string, int>` points to a `std::pair<string, int>`.

The `std::pair` Class

A pair is simply two objects bundled together.

Syntax:

```
std::pair<string, int> p;
p.first = "Phone number";
p.second = 6504550404;
```

Map Iterators

Example:

```
map<int, int> m;
map<int, int>::iterator i = m.begin();
map<int, int>::iterator end = m.end();
while(i != end) {
    cout << (*i).first << (*i).second << endl;
    ++i;
}
```

Next Time

Templates and Iterators