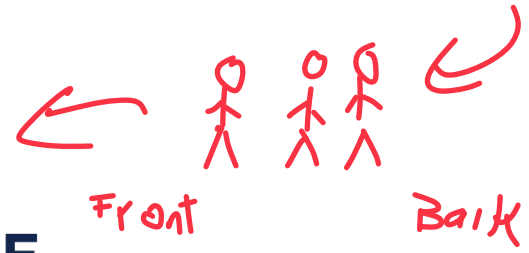
Data Structures

Queues, Iterators, and maybe Trees?

CS 225

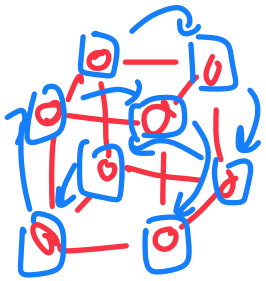
Brad Solomon

September 12, 2025



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science



Exam 1 (9/17 — 9/19)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam is out on PL now

Topics covered can be found on website

Register now

https://courses.engr.illinois.edu/cs225/fa2025/exams/

Learning Objectives

Queue!!

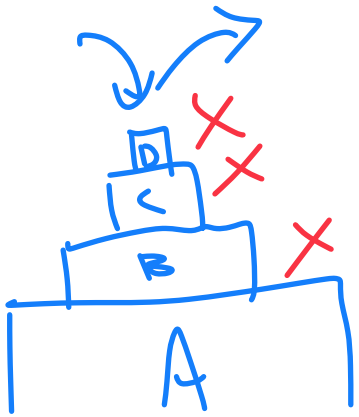
Discuss the importance of iterators

Review trees and binary trees

Discuss the tree ADT

Stack ADT

- [Order]: LIFO



- [Implementation]: Array (such as std::vector)

- [Runtime]: O(1) Push and Pop (No random access)

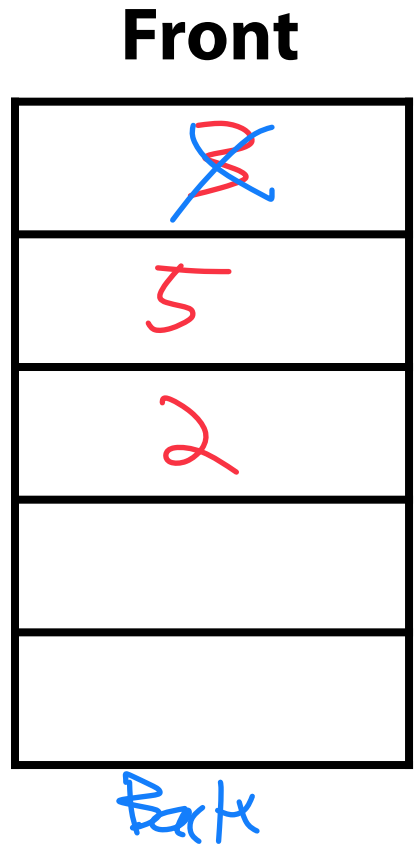
Queue Data Structure

A **queue** stores an ordered collection of objects (like a list)

However you can only do two* operations:

Enqueue: Put an item at the back of the queue

Dequeue: Remove and return the front item of the queue



Front & back both 0

back index 1

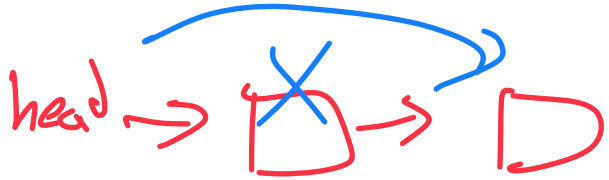
```
enqueue(3); enqueue(5); dequeue(); enqueue(2)
```

Queue Data Structure

The queue is a **first in — first out** data structure (FIFO)

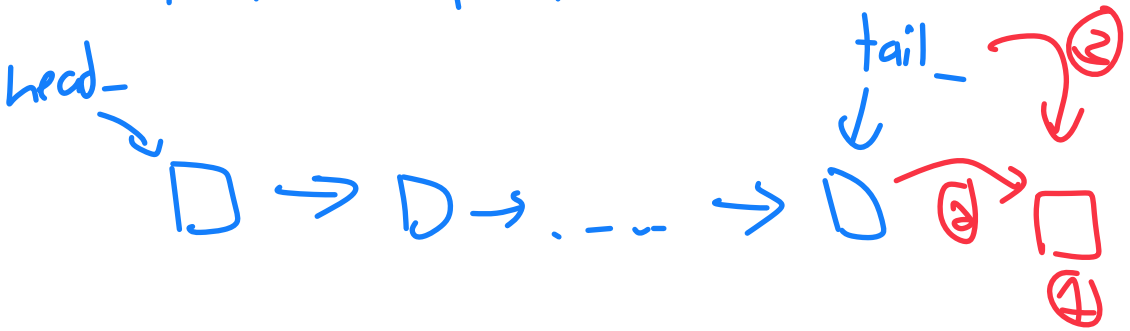
What data structure excels at removing from the front?

Linked List



Can we make that same data structure good at inserting at the end?

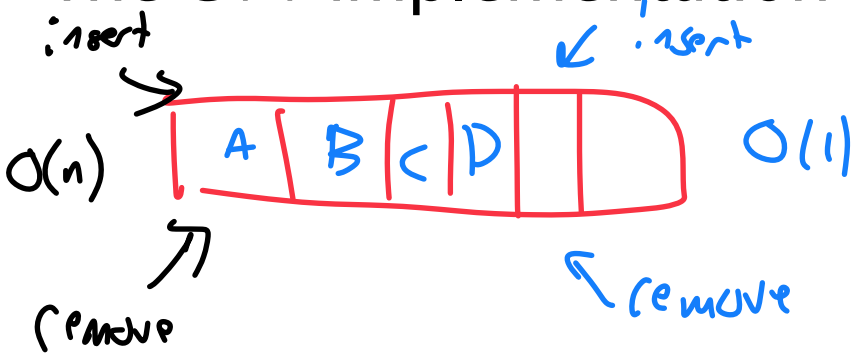
↳ Add a tail pointer



Queue Data Structure

an array!

The C++ implementation of a queue is also a vector or deque — why?



Two reasons

1) Engineering! ← CS 225 will ignore this!

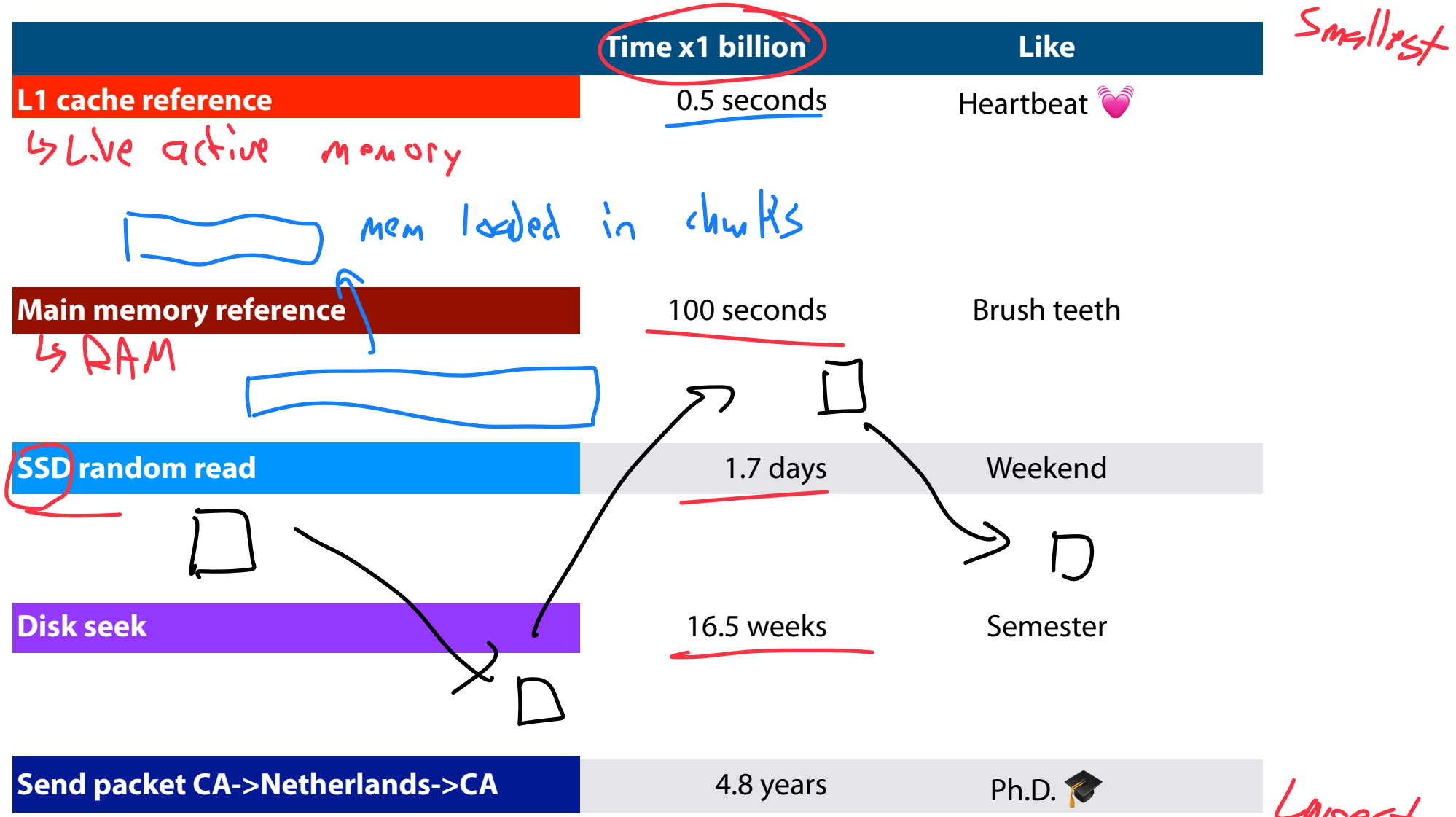
2) We can make theory better!

Engineering vs Theory Efficiency

	Time x1 billion	Like
L1 cache reference	0.5 seconds	Heartbeat 💓
Branch mispredict	5 seconds	Yawn 😯
L2 cache reference	7 seconds	Long yawn 😯😯😯
Mutex lock/unlock	25 seconds	Make coffee ☕
Main memory reference	100 seconds	Brush teeth
Compress 1K bytes	50 minutes	TV show 📺
Send 2K bytes over 1 Gbps network	5.5 hours	(Brief) Night's sleep 🛌
SSD random read	1.7 days	Weekend
Read 1 MB sequentially from memory	2.9 days	Long weekend
Read 1 MB sequentially from SSD	11.6 days	2 weeks for delivery 📦
Disk seek	16.5 weeks	Semester
Read 1 MB sequentially from disk	7.8 months	Human gestation 🐣
Above two together	1 year	🌍☀️
Send packet CA->Netherlands->CA	4.8 years	Ph.D. 🎓

(Care of https://gist.github.com/hellerbarde/2843375)

Engineering vs Theory Efficiency



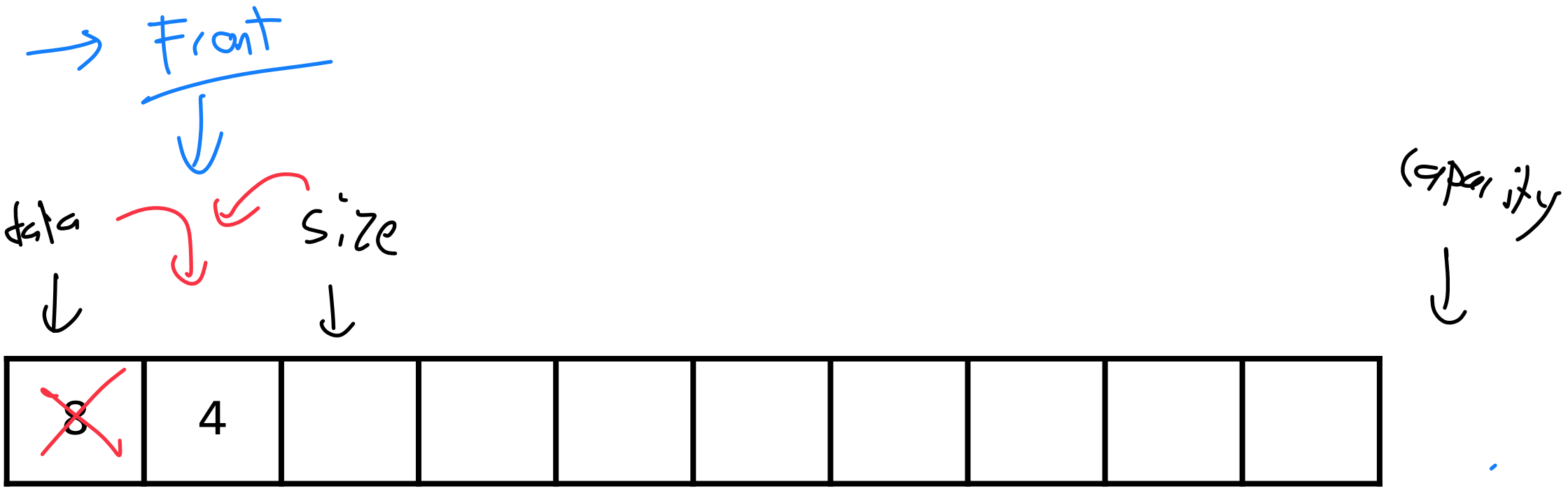
(Care of https://gist.github.com/hellerbarde/2843375)

Queue Data Structure

```
q.enqueue(8);
q.enqueue(4);
q.dequeue();
```

What do we need to track to maintain a queue with an array list?

Goal: Remove 8 in O(1) time



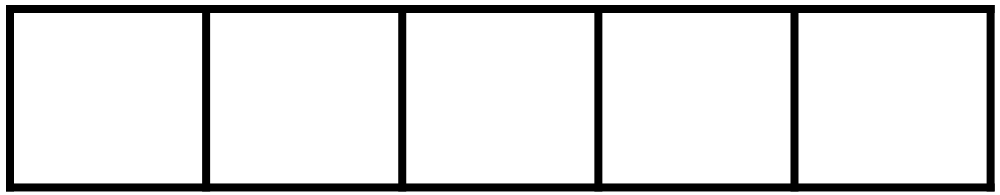
Queue Data Structure

Unlike the array list, it is easier to implement a Queue using unsigned ints

Queue.h

```
 1  #pragma once
 2
 3  template <typename T>
 4  class Queue {
 5    public:
 6      void enqueue(T e);
 7      T dequeue();
 8      bool isEmpty();
 9
10    private:
11      T *data_;
12      unsigned size_;
13      unsigned capacity_;
14      unsigned front_;
15  };
```

New entry "front"

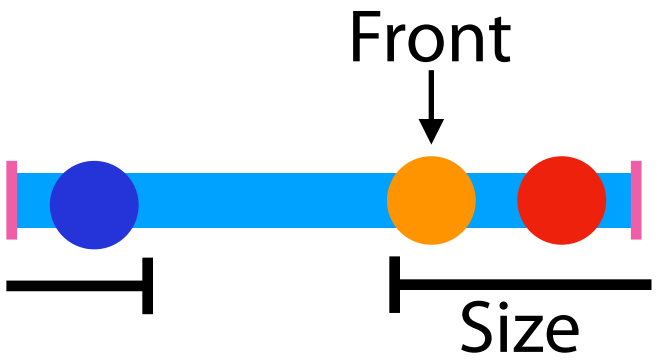


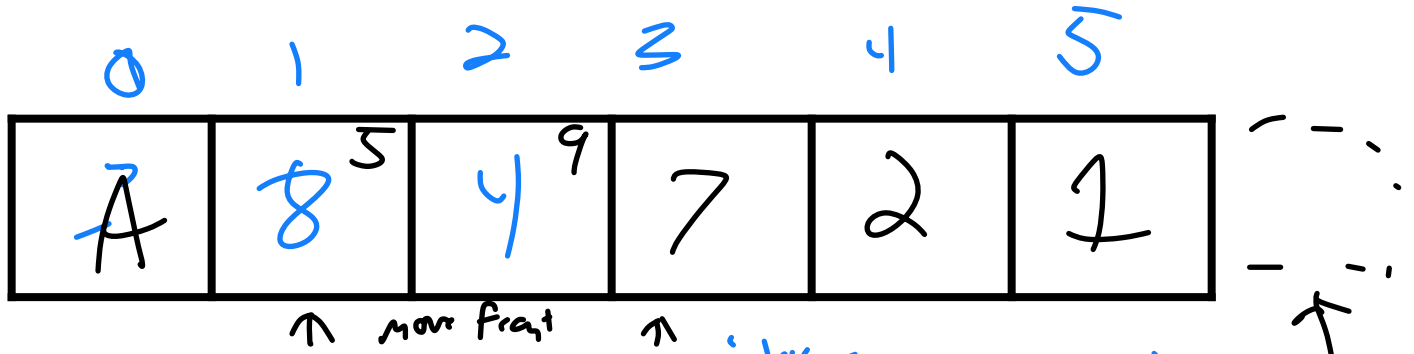
(Circular) Queue Data Structure

Queue.h

```
 1  #pragma once
 2
 3  template <typename T>
 4  class Queue {
 5    public:
 6      void enqueue(T e);
 7      T dequeue();
 8      bool isEmpty();
 9
10    private:
11      T *data_;
12      unsigned capacity_;
13      unsigned size_;
14      unsigned front_;
15  };
```

total # of spaces





Enqueue(D): Add D to index (front + size) % capacity

size++;

Dequeue(): Remove & return front

size--;

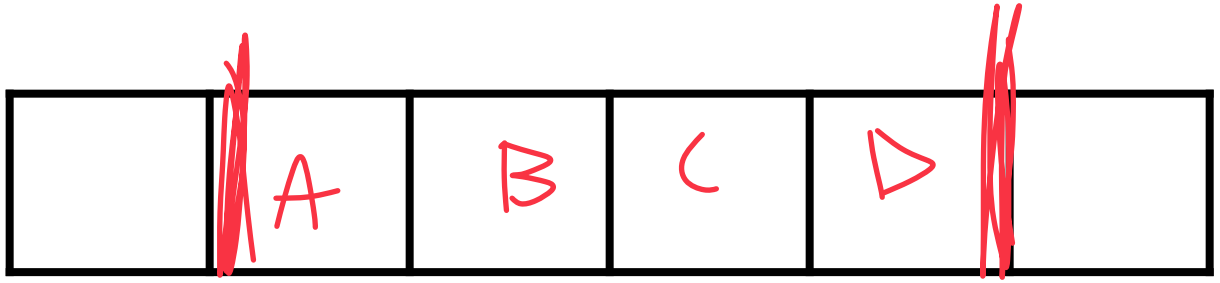
front++;

Size: 0 1 2 3 4 3

Front: 0 1

Capacity: 6

```
Queue<int> q;
q.enqueue(3);      1
q.enqueue(8);      2
q.enqueue(4);      3
0→1 q.dequeue();   2
q.enqueue(7);      3
1→2 q.dequeue();   2
2→3 q.dequeue();   1
q.enqueue(2);      2
q.enqueue(1);      3
3+3 = 6 → Insert @ 0   q.enqueue(A);
                       q.enqueue(5);
q.dequeue();
q.enqueue(9);
```



Back is size + front

Enqueue(D): Insert @ (size+front) % capacity

size++ until size == capacity

Dequeue(): Remove @front

front = (front+1) % capacity

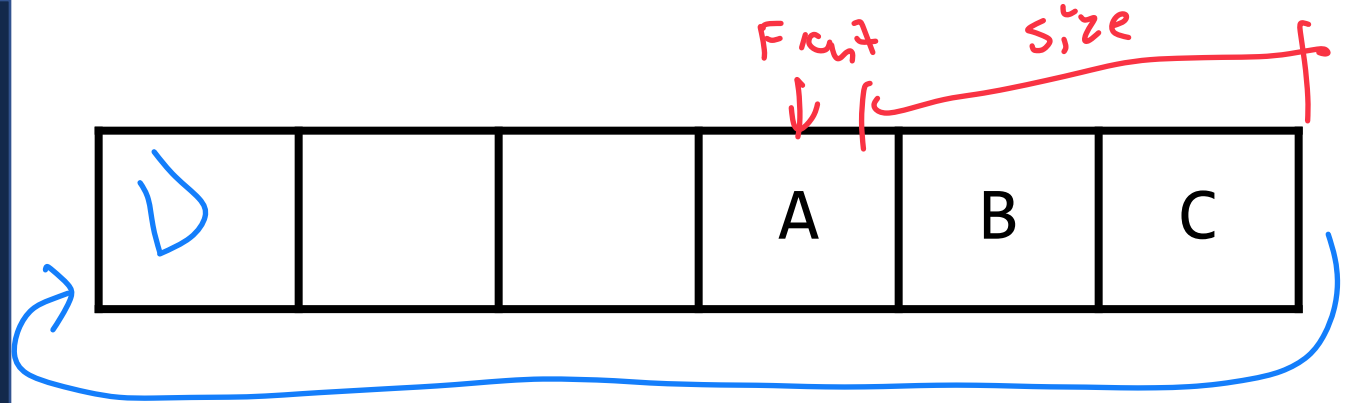
size--

Size:

Front:

Capacity:

```
Queue<int> q;
q.enqueue(3);
q.enqueue(8);
q.enqueue(4);
q.dequeue();
q.enqueue(7);
q.dequeue();
q.dequeue();
q.enqueue(2);
q.enqueue(1);
q.enqueue(3);
q.enqueue(5);
q.dequeue();
q.enqueue(9);
```




```
Queue<int> q;
...
q.enqueue(D);
q.dequeue();
q.dequeue();
q.dequeue();
q.dequeue();
q.enqueue(E);
```

Enqueue(D): Add data to ‘back’ of queue

Insert D at index (size+front) % capacity

size++ (as long as size != capacity)

Dequeue(): Remove data at index front

front = (front+1) % capacity

size-- (as long as size != 0)

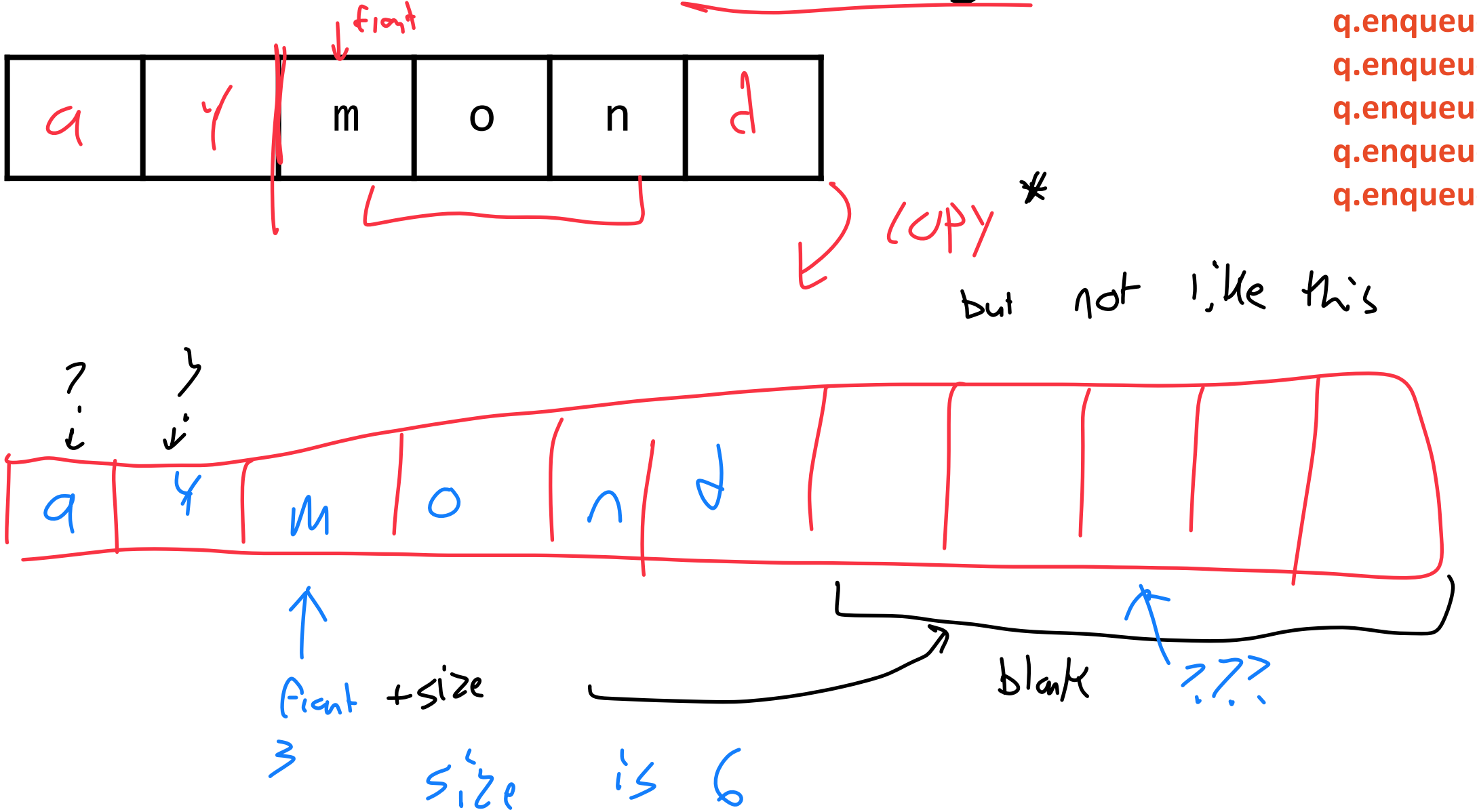
Size: 3

Front: 3

Capacity: 6

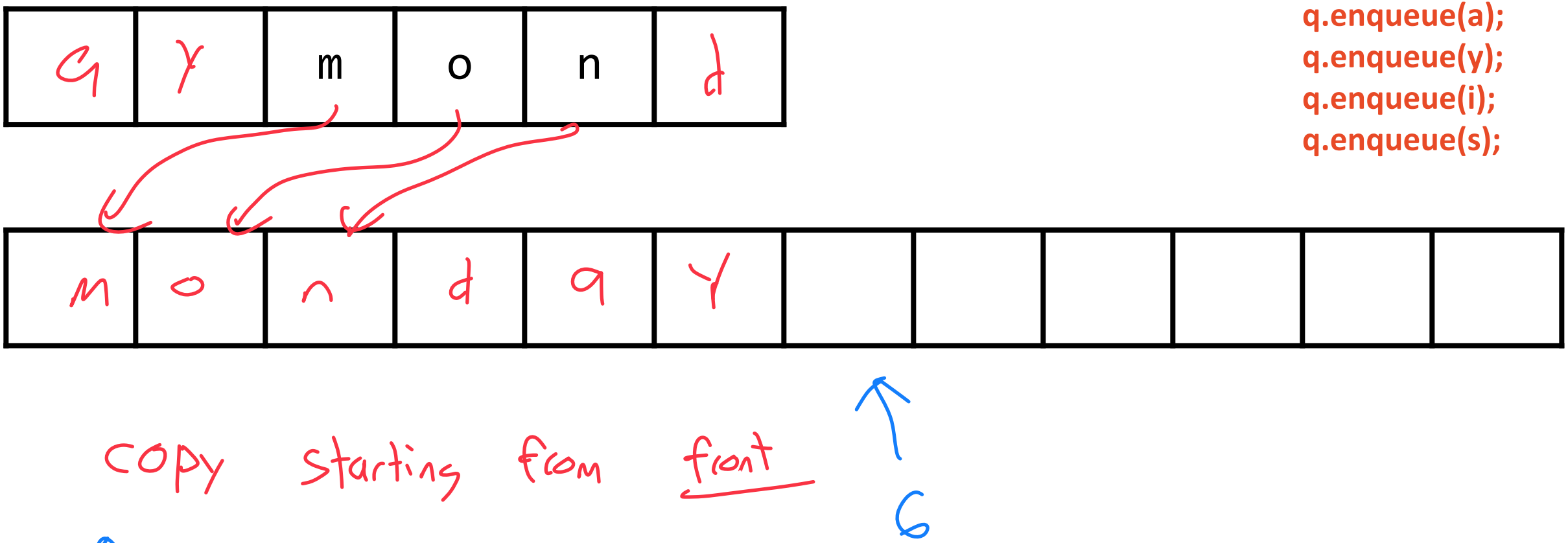
Queue Data Structure: Resizing

```
{ Queue<char> q;
  ...
  q.enqueue(d);
  q.enqueue(a);
  q.enqueue(y);
  q.enqueue(i);
  q.enqueue(s);
```



Queue Data Structure: Resizing

```
Queue<char> q;
...
q.enqueue(d);
q.enqueue(a);
q.enqueue(y);
q.enqueue(i);
q.enqueue(s);
```



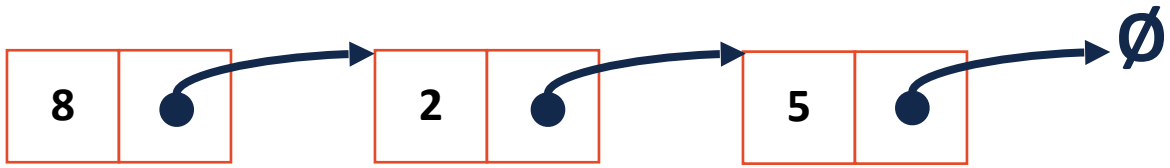
Queue ADT

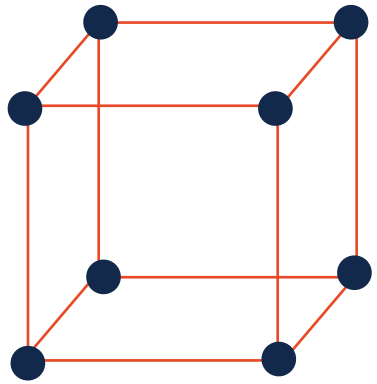
- [Order]: First in first out
- [Implementation]: Trivially as LL
  w/ circular queue as array
- [Runtime]: $O(1)$ *
  when array amortized $O(1)$

Iterators

np_lists

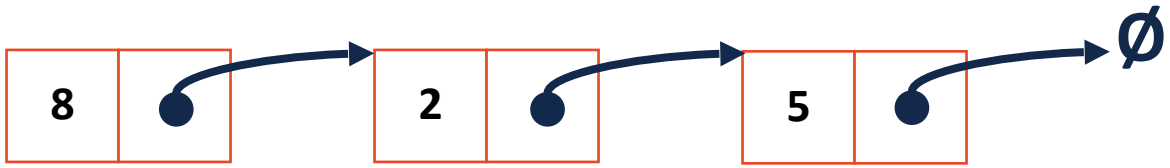
We want to be able to loop through all elements for any underlying implementation in a systematic way

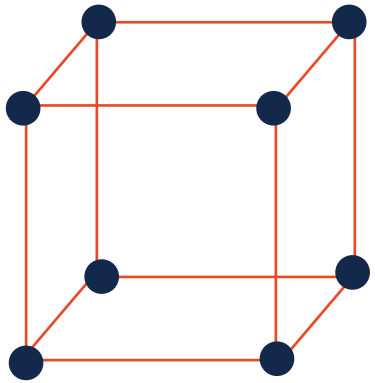




Iterators

We want to be able to loop through all elements for any underlying implementation in a systematic way



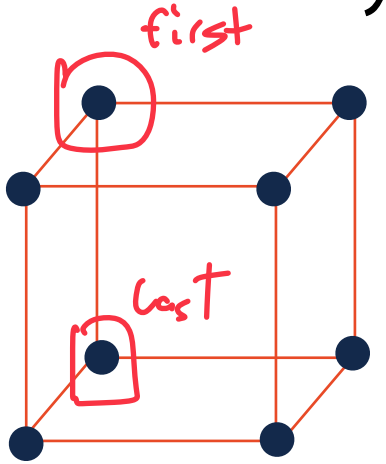


Cur. Location	Cur. Data	Next
`ListNode * curr`	curr→data	curr→next
`unsigned index`	array[index]	index++;
`Some form of (x, y, z)`	???	???

Iterators

Cube defines its own Iterator

Iterators provide a way to access items in a container without exposing the underlying structure of the container



gives first item

gives last item

```
1  Cube::Iterator it = myCube.begin();
2
3  while (it != myCube.end()) {
4      std::cout << *it << " ";
5      it++;
6  }
7
```

increment to get next item

dereference iterator

Iterators

For a class to implement an iterator, it needs two functions:

```
Iterator begin()
```


```
Iterator end()
```

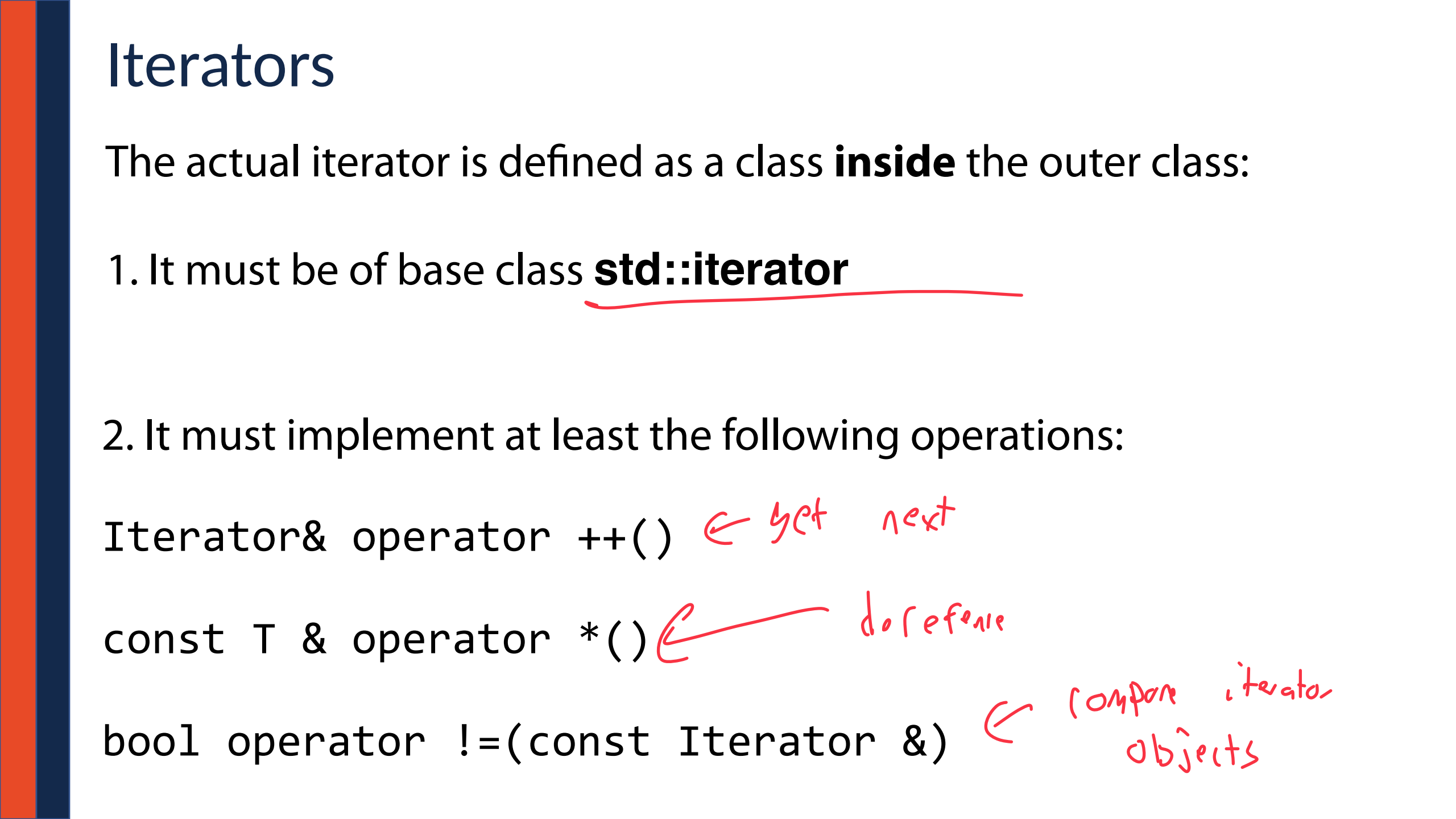
Iterators

The actual iterator is defined as a class **inside** the outer class:

1. It must be of base class **std::iterator**

2. It must implement at least the following operations:

```
Iterator& operator ++()
const T & operator *()
bool operator !=(const Iterator &)
```



Iterators

Here is a (truncated) example of an iterator:

```
1  template <class T>
2  class List {
3
4      class ListIterator : public
   std::iterator<std::bidirectional_iterator_tag, T> {
5        public:
6
7          ListIterator& operator++();
8
9          ListIterator& operator--()
10
11         bool operator!=(const ListIterator& rhs);
12
13         const T& operator*();
14     };
15
16     ListIterator begin() const;
17
18     ListIterator end() const;
19 };
```

class inside class

get next

get prev

stlList.cpp

```
1  #include <list>
2  #include <string>
3  #include <iostream>
4
5  struct Animal {
6    std::string name, food;
7    bool big;
8    Animal(std::string name = "blob", std::string food = "you", bool big = true) :
9      name(name), food(food), big(big) { /* nothing */ }
10 };
11
12 int main() {
13   Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
14   std::vector<Animal> zoo;
15
16   zoo.push_back(g);
17   zoo.push_back(p);   // std::vector's insertAtEnd
18   zoo.push_back(b);
19
20   for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
21     std::cout << (*it).name << " " << (*it).food << std::endl;
22   }
23
24   return 0;
25 }
```

```
1
2  std::vector<Animal> zoo;
3
4
5  /* Full text snippet */
6
7    for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
8      std::cout << (*it).name << " " << (*it).food << std::endl;
9    }
10
11
12 /* Auto Snippet */
13
14   for ( auto it = zoo.begin(); it != zoo.end; ++it ) {
15     std::cout << (*it).name << " " << (*it).food << std::endl;
16   }
17
18 /* For Each Snippet */
19
20   for ( const Animal & animal : zoo ) {
21     std::cout << animal.name << " " << animal.food << std::endl;
22   }
23
24
25
```

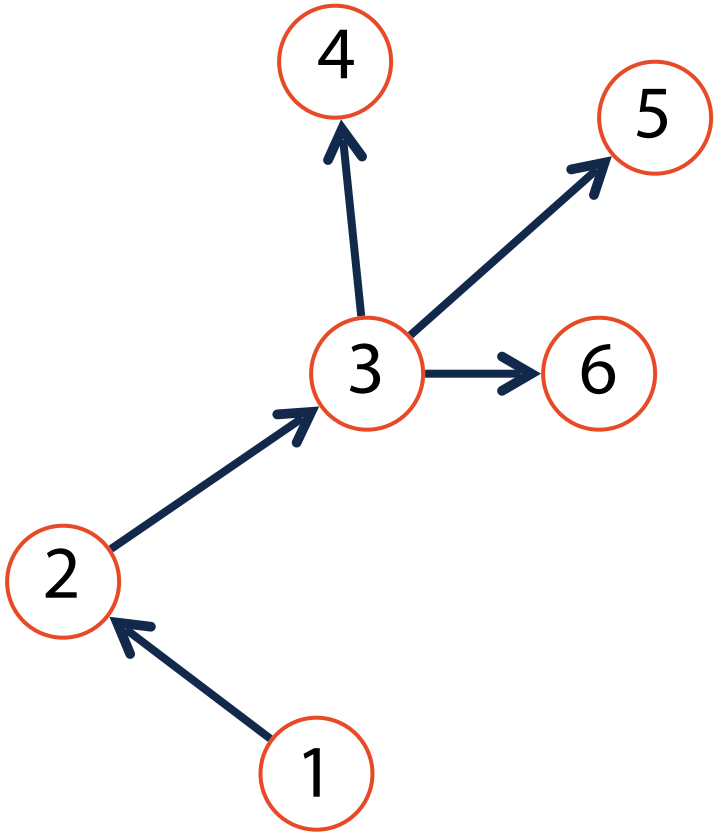
Trees

A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

[In CS 225] a tree is also:

1) Acyclic — No path from node to itself

2) Rooted — A specific node is labeled root

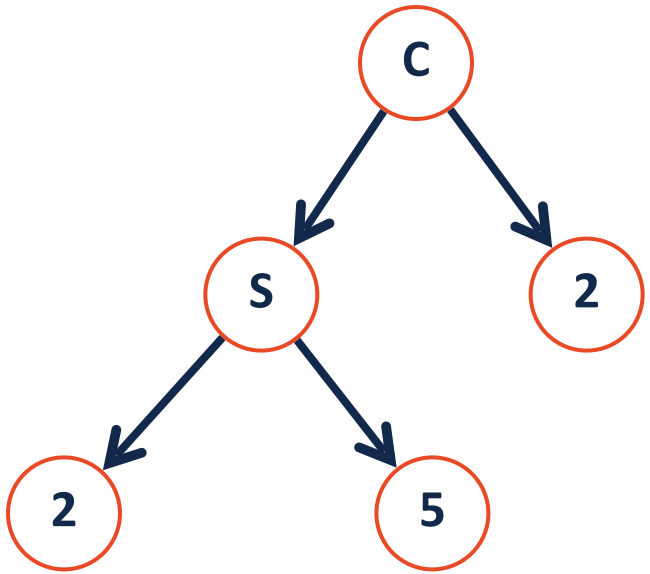


Binary Tree

A **binary tree** is a tree T such that:

1. $T = \emptyset$

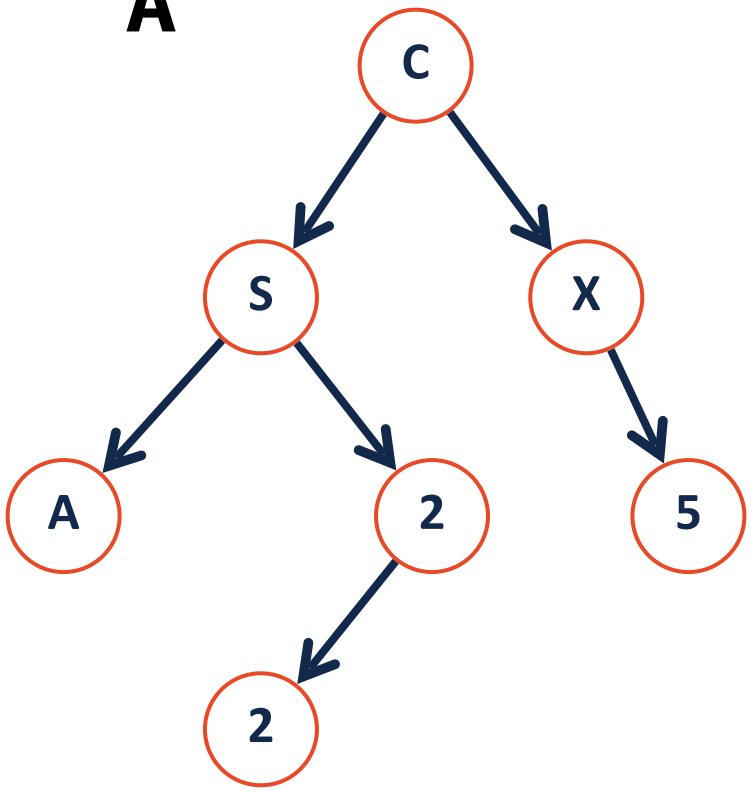
2. $T = (data, T_L, T_R)$



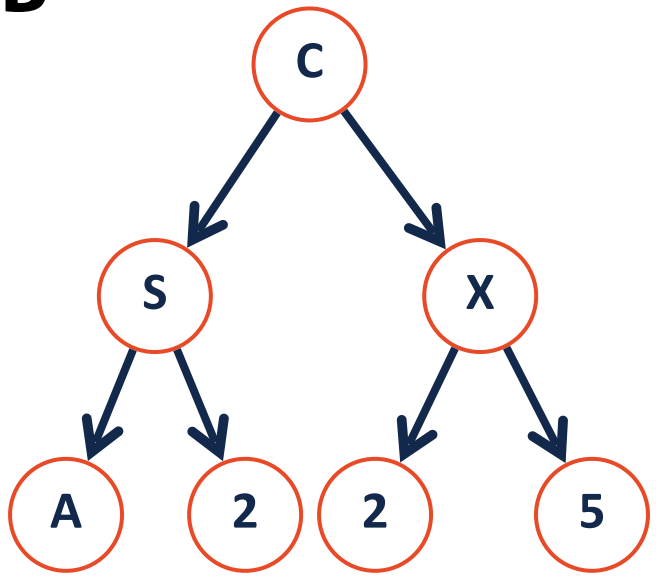
Which of the following are binary trees?

Join Code: 225

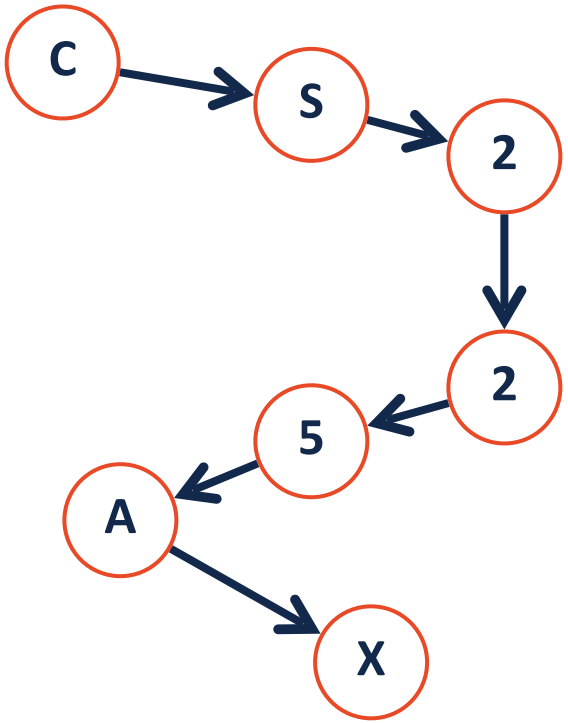
A



B



C



Tree ADT