

CSE 332 Winter 2024

Lecture 1: Intro to ADTs, Stacks, Queues

Nathan Brunelle

<http://www.cs.uw.edu/332>

Nathan Brunelle

- Born: Virginia Beach, VA
- Ugrad: Math and CS at University of Virginia
- Grad: CS at University of Virginia
- Taught at UVA for 6 years
 - Intro to programming (e.g. 121)
 - Discrete Math (e.g. 311)
 - Algorithms (e.g. 412) ←
 - Theory of Computation (e.g. 431)



Warm Up!

Put up one hand (you can switch if it gets tired)!

While (you and at least one other person have a hand up){

make a partnership with someone whose hand is still raised

share your name with your partner

determine which of you has run the longest distance (as a single run)


release partnership

if you ran the shorter distance, then put your hand down and return
to your seat

}

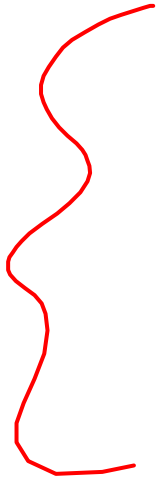
About this course

Topics covered:

- Data Structures 
 - Specific “classic” data structures
- Introduction to Algorithms and Analysis
- Parallelism and Concurrency
 - Parallelism: Use multiple processors to finish sooner
 - Concurrency: Correct access to shared resources

Course Staff

- Instructor:
 - ~~Nathan Brunelle~~
- TAs:



Course Info

- Text (optional):
 - Data Structures & Algorithm Analysis in Java, (Mark Allen Weiss), 3rd edition, 2012
(2nd edition also o.k.)
- Course Page:
 - <http://www.cs.uw.edu/332>

Communication

- Course email list:
 - cse332_wi24@uw
 - You are already subscribed
 - You must get and read announcements sent there
- Ed STEM Discussion board
 - Your first stop for questions about course content & assignments

Course Meetings

- Lecture
 - Materials posted (slides before class, inked slides after)
 - Recorded using Panopto
 - Ask questions, focus on key ideas (rarely coding details)
- Section
 - Practice problems!
 - Answer Java/project/homework questions, etc.
 - Occasionally may introduce new material
 - An important part of the course (not optional)
- Office hours
 - Use them: *please visit us!*

Grading

- 12ish Weekly individual homework exercises (25%)
 - Lowest 2ish dropped (best 10 count)
- 3 programming projects (with phases) (35%)
 - Use Java and IntelliJ, Gitlab
 - Done individually
- Midterm and final exam (40%)
 - In-person
 - Midterm in this room
 - Final location TBD
- Dates:
 - Midterm: Monday Feb 5, during lecture
 - Final Exam: Thursday March 14, 12:30pm-2:20pm

Collaboration

- Try it yourself first
- Collaborate with classmates (no external interactive help on assignments permitted)
 - Collaboration is “whiteboard only”
 - Looking for a collaborator?
 - Post on the Ed Discussion board
 - Go to the CSE study room (Allen Center 006, there’s a table specifically for 332!)
- Cite your sources!

Terminology

"Interface"

- Abstract Data Type (ADT)

- Mathematical description of a "thing" with set of operations on that "thing"

- Algorithm

- A high level, language-independent description of a step-by-step process

- Data structure

- A specific organization of data and family of algorithms for implementing an ADT

- Implementation of a data structure

- A specific implementation in a specific language

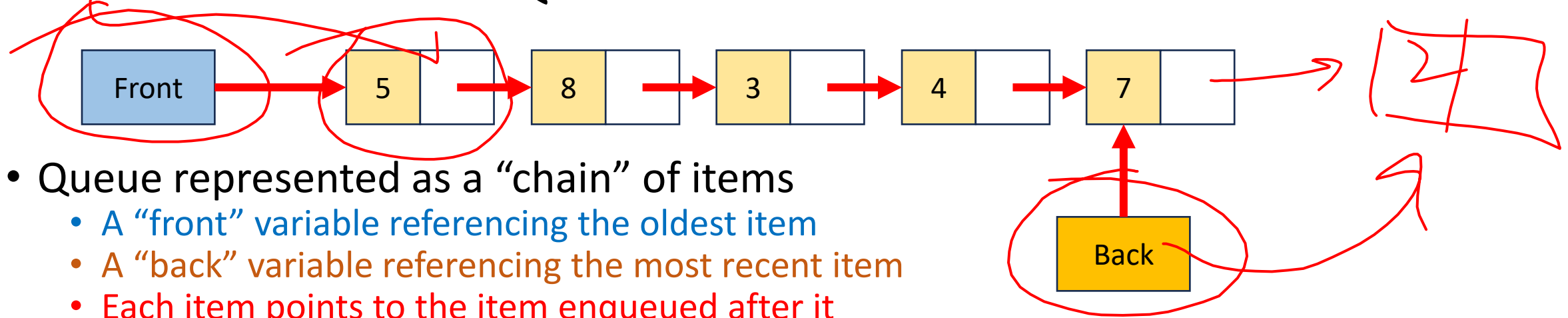
ADT: Queue

- What is it?
 - FIFO collection of things
- What Operations do we need?
 - Enqueue:
 - Add something to the data structure
 - Dequeue:
 - Remove the oldest thing from the data structure
 - isEmpty:
 - Are there any things still in the DS?

ADT: Queue

- What is it?
 - A “First In First Out” (FIFO) collection of items
- What Operations do we need?
 - Enqueue
 - Add a new item to the queue
 - Dequeue
 - Remove the “oldest” item from the queue
 - IsEmpty
 - Indicate whether or not there are items still on the queue

Linked List – Queue Data Structure



- Queue represented as a “chain” of items
 - A “front” variable referencing the oldest item
 - A “back” variable referencing the most recent item
 - Each item points to the item enqueued after it

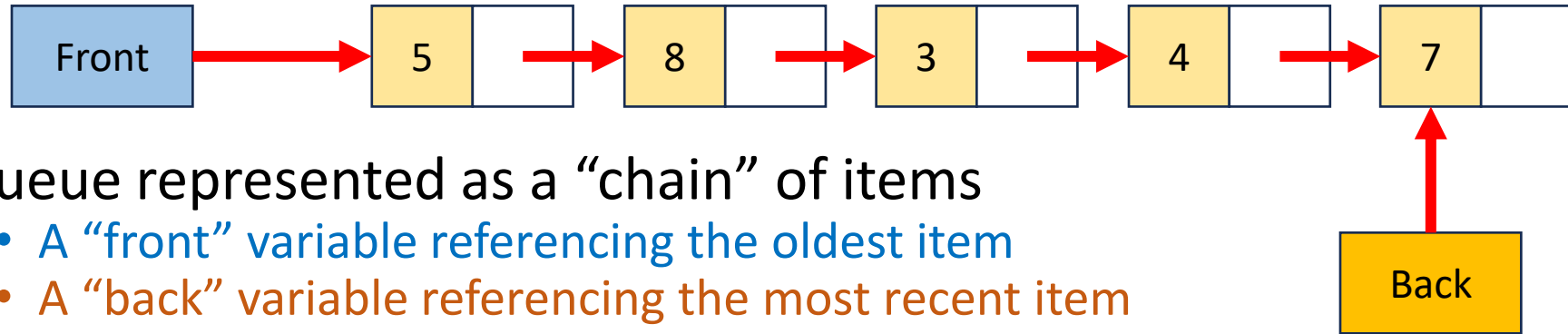
• Enqueue Procedure:

*create a new val, point
move seq, point to new*

• Dequeue Procedure:

• Is_empty Procedure:

Linked List – Queue Data Structure



- Queue represented as a “chain” of items
 - A “front” variable referencing the oldest item
 - A “back” variable referencing the most recent item
 - Each item points to the item enqueued after it

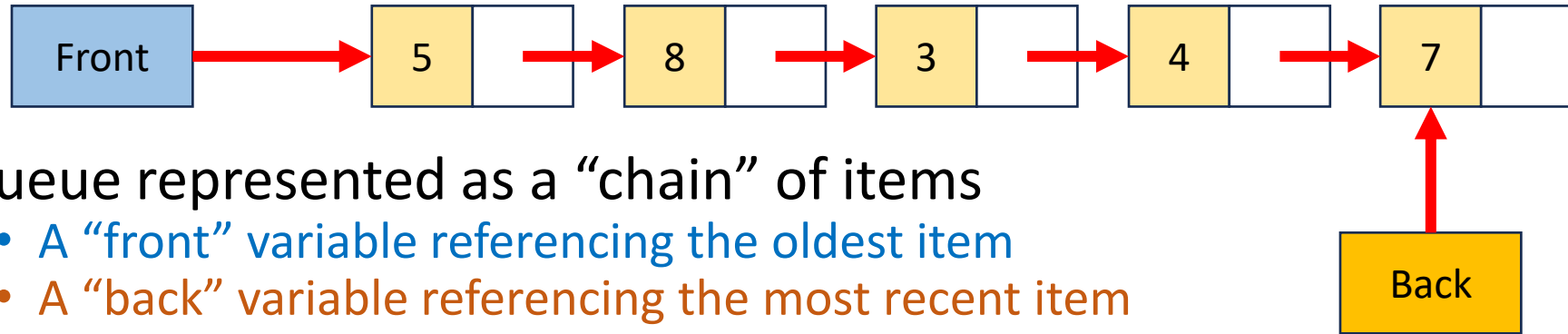
- Enqueue Procedure:

```
enqueue(x){
    last = new Node(x)
    back.next = last
    back = last
}
```
- Dequeue Procedure:

```
dequeue(){
    first = front.item
    front = front.next
    return first
}
```
- Is_empty Procedure:

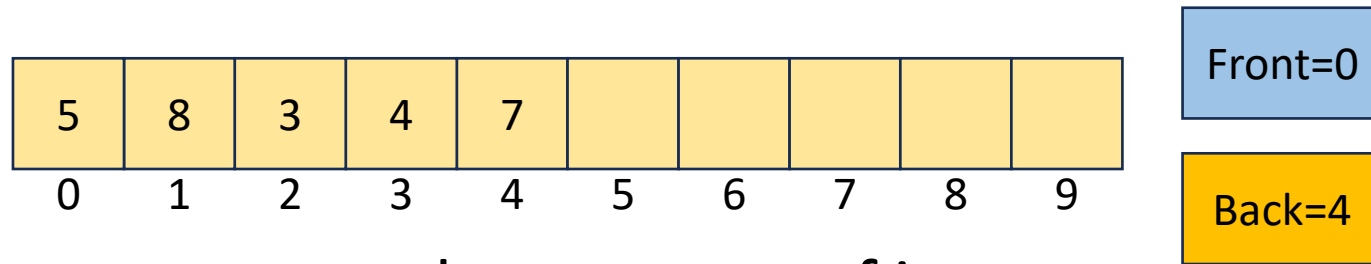
```
is_empty(){
    return front.equals(Null)
}
```

Circular Array – Queue Data Structure



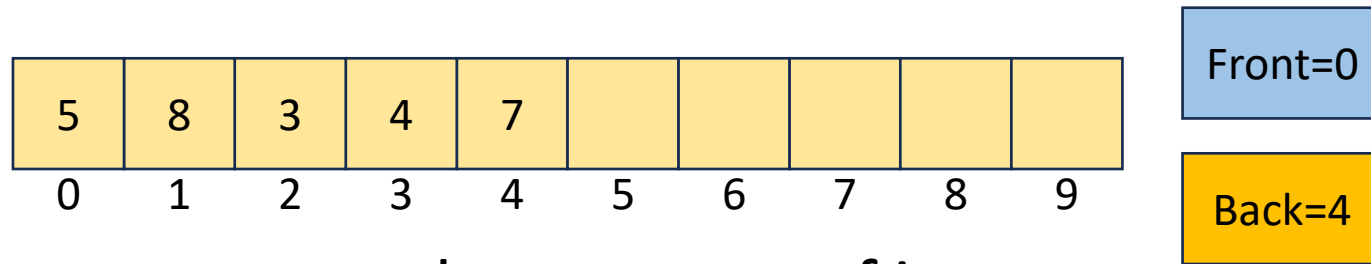
- Queue represented as a “chain” of items
 - A “front” variable referencing the oldest item
 - A “back” variable referencing the most recent item
 - Each item points to the item enqueued after it
- Enqueue Procedure:
- Dequeue Procedure:
- Is_empty Procedure:

Circular Array – Queue Data Structure



- Queue represented as an array of items
 - A “front” index to indicate the oldest item in the queue
 - A “back” index to indicate the most recent item in the queue
- Enqueue Procedure:
- Dequeue Procedure:
- Is_empty Procedure:

Circular Array – Queue Data Structure



- Queue represented as an array of items
 - A “front” index to indicate the oldest item in the queue
 - A “back” index to indicate the most recent item in the queue
- Enqueue Procedure:

```
enqueue(x){
    queue[back] = x
    back = (back + 1) % queue.length
}
```
- Dequeue Procedure:

```
dequeue(){
    first = queue[front]
    front = (front + 1) % queue.length
}
```
- Is_empty Procedure:

```
is_empty(){
    return front == back
}
```

Linked List vs. Circular Array

ADT: Stack

- What is it?
- What Operations do we need?

ADT: Stack

- What is it?
 - A “Last In First Out” (LIFO) collection of items (sometimes called FILO)
- What Operations do we need?
 - Push
 - Add a new item onto the stack
 - Peek
 - Return the value of the most recently pushed item
 - Pop
 - Return the value of the most recently pushed item and remove it from the stack
 - Is_empty
 - Indicate whether or not there are items still on the stack