COP 3502 9/13/23

1. P1 SOLIDATA POSTED
2. P2 POSTED - START EARLY!!!
3. STACKS - INFIX → POSTFIX ALG
4. QUEUES

Infix

$$(5 + 7) * (2 - (3 - 5)) / 6$$

Read Left to Right

1. Paren Open - Push onto stack (parens / operators)
2. Operand (number) - placed into the expression (output)
3. Paren Close - Pop off each operator on the stack + place it into the expression UNTIL you hit the 1st open parenthesis. Pop this off + stop.
4. Operator - continue popping items off stack + placing into expression as long as they are operators of equal or greater precedence.
   STOP: paren, operator with lower precedence, empty stack

After done reading:

5. pop off any operators still on stack + place in expression

$$(5+7)*(2-(3-5))/6$$

Stack	Stack after)	Upto 3-5	Stack after)	Stack after))	after /
		−			
		(			
+		(	(		
(		*	*	*	/

Expr: 5 7 + 2 3 5 − − * 6 /

Queue - Line

enqueue - add to back of line
dequeue - remove front queue + return

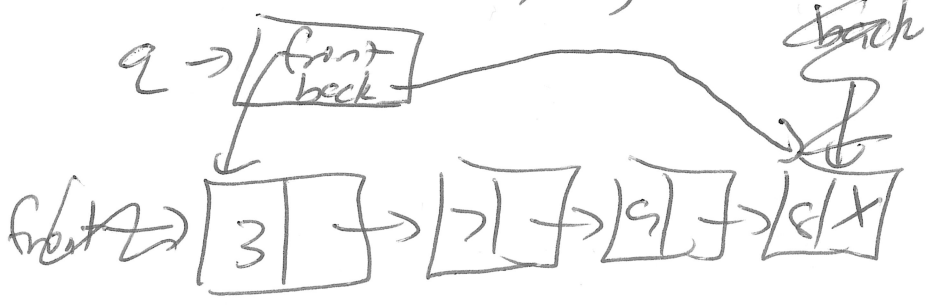
FIFO - "first in first out"

} O(1)

empty
full
size
front - return what's front w/o removing

enqueue(3)
enqueue(7)
enqueue(9)
x = dequeue()
x [3]
enqueue(8)
enqueue(2)
y = dequeue
y [7]

~~3~~, 7, 9, 8, 2



implement enqueue(2)

ISSUE: loop through whole list just to find where to add the ~~next~~ new item. O(n)
n = # items

```
typedef struct node {
    int data;
    struct node* next;
} node;
```

LL node

```
typedef struct queue {
    node* front;
    node* back;
    int size;
} queue;
```

for P2 add this

```
Void enqueue (queue* qPtr, int data) {
    node* tmp = makeNode (data);
    if (qPtr->front == NULL) {
        qPtr->front = tmp;
        qPtr->back = tmp;
        qPtr->size++;
    }
    else {
        qPtr->back->next = tmp;
        qPtr->back = tmp;
    }
    qPtr->size++;
}
```