COP 3502 2/22/2024

1. Infix to Postfix
2. Queue -> Quickly go through Corn Maze
3. LL implementation Queue
4. Array = Queue
5. Live Code: Grid uses Queue

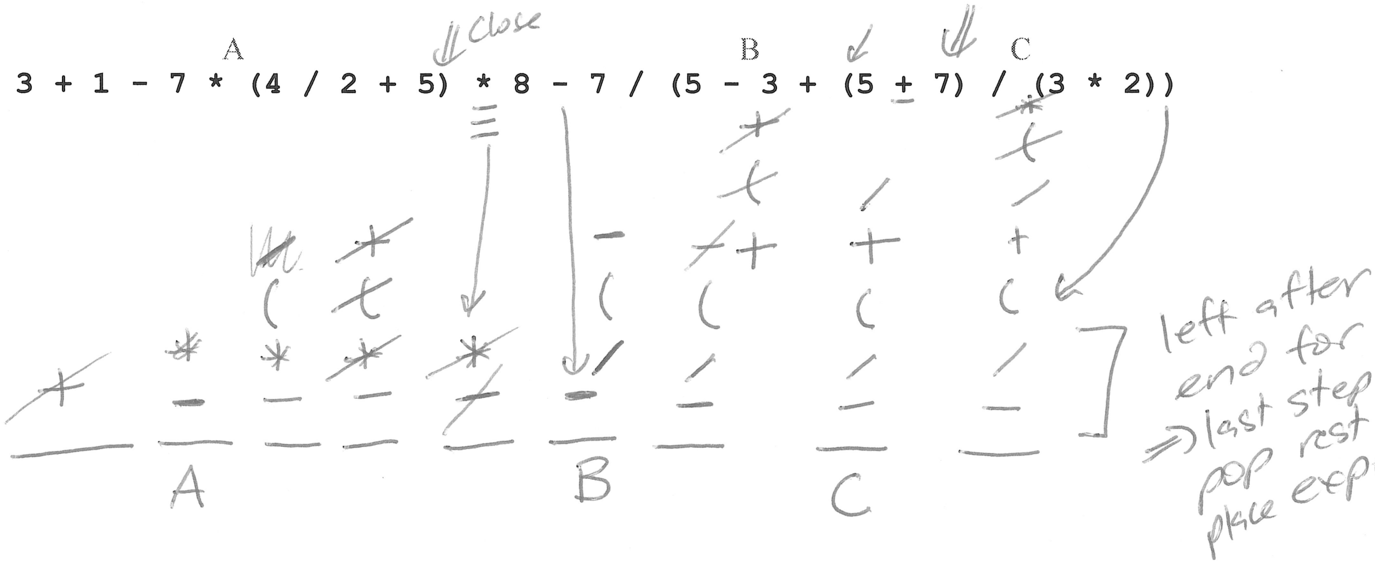
Rules

Stack = Operator Stack + Parens (Open)
Goal: Output a Postfix Expression
Read symbols L -> R, process them (Same other als)
Go through expr:

```
if it's an operand (#)
    place it at the end of the expression

else if it's open parenthesis
    push onto stack

else if it's a close parenthesis:
    pop each item of stack and place it in
    the expression, stopping right after you pop
    the corresponding open paren (not placed)

else // it's operator
    pop off each operator from stack that is
    of equal or greater precedence, placing it into
    the expression. Stop when you hit a paren,
    empty stack, OR item of lower precedence. Then push
```

Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (A, B, and C) in the infix expression.

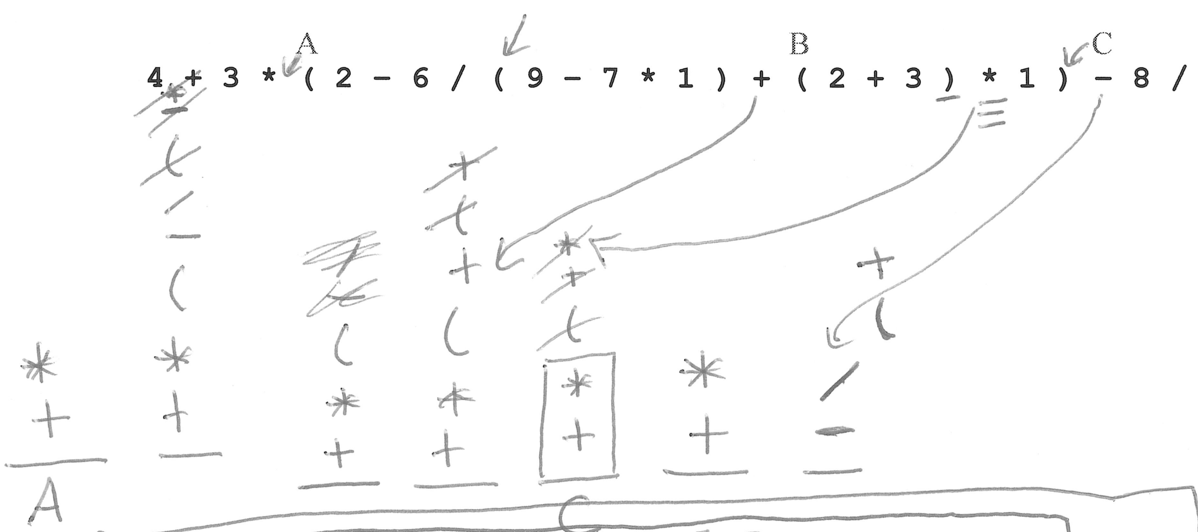
```
                A                              B             C
3 + 1 - 7 * (4 / 2 + 5) * 8 - 7 / (5 - 3 + (5 + 7) / (3 * 2))
```



expr: 3 1 + 7 4 2 / 5 + * 8 * - 7 5 3 - 5 7 + 3 2 * / + / -

Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (A, B, and C) in the infix expression.

```
          A                                 B                     C
4 + 3 * ( 2 - 6 / ( 9 - 7 * 1 ) + ( 2 + 3 ) * 1 ) - 8 / ( 1 + 1 )
```



expr: 4 3 2 6 9 7 1 * - / - 2 3 + 1 * + * + 8 1 1 + / -

Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (1, 2, and 3) in the infix expression.

```
                          1            2            3
A + (B * (C – D          )) + ((E /    F ) + G )    + H
```

$$A+(B*(C-D))+1((E/F)^2+G)+H$$

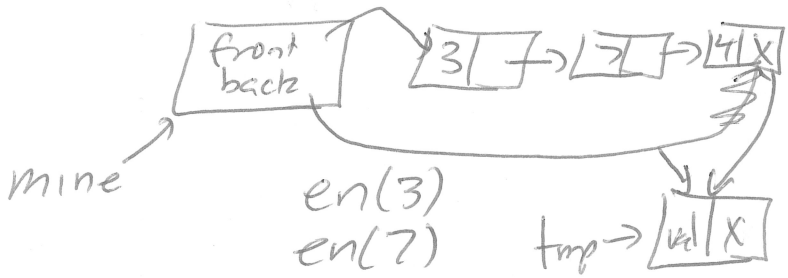
Queue ~~Implementation~~ Operations

enqueue (add back)
dequeue (remove from front)
size
front - return front w/o dequeue

Implementation

LL easier

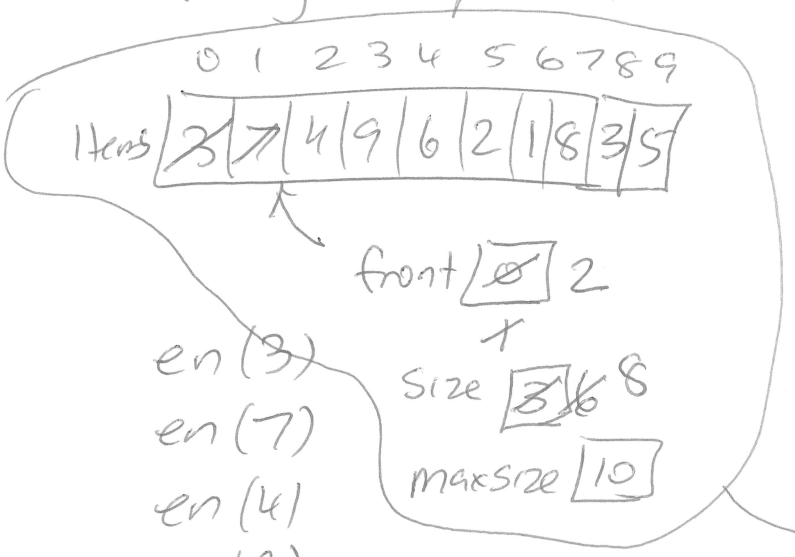
```
struct ~~node~~ lnklist {
    node* front;
    node* back;
};
```




```
struct lnklist* mine = malloc ...

void enqueue(struct lnklist* mine, int val) {
    node* tmp = makeNode(val);
    if (mine->front == NULL) {
        mine->front = tmp;
        mine->back = tmp;
        return;
    }
    mine->back->next = tmp;
    mine->back = tmp;
}
```

free(a)
what b is pointing to disappears!

Array Implementation of a Queue



enq(3)
enq(7)
enq(4)
enq(9)
x = deq() x 3

If we do 10^5 enqueues, one dequeue
if we ask all items to move up one index
arr[i] = arr[i+1]
this is O(n)
n = # items in queue

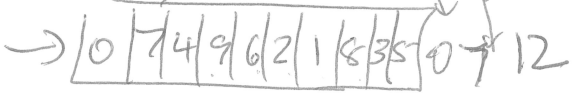
all part of queue struct with array impl.

INDEX to enqueue

$(q \to front + q \to size) \% q \to maxsize$

Dequeue $q \to front = (q \to front + 1) \% q \to maxsize$

Dynamically Grow



front 2
size 10 11
maxsize 10 20

enqueue(12)

Idea #1:
new malloc
copy index front → ms
to index 0, wraparound
copy, reassign array.

Idea #2:

```
q->items = realloc(q->items,
                   sizeof(int) * 2 *
                   q->maxsize);
for (int i = q->maxsize; i < q->maxsize + q->front; i++)
    q->items[i] = q->items[i - q->maxsize];
q->maxsize *= 2;
```