COP3502 - 2/20/24

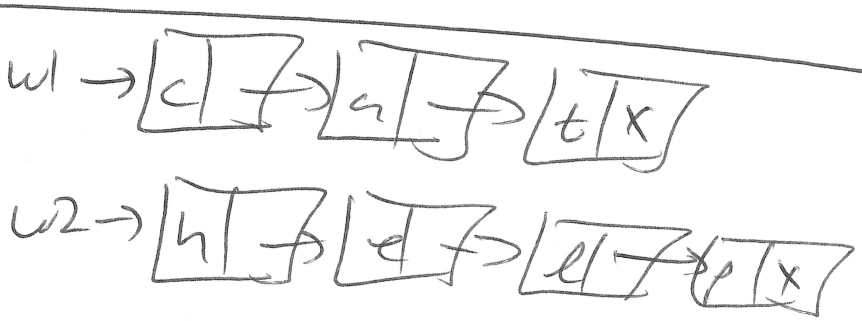
1. Linked Lists
   - a. bigint ✓
   - b. strings ✓
   - c. circular, double
   - d. LL of LL (cd example)
2. Stacks
   - a. array impl.
   - b. ~~stack~~ LL impl.
   - c. Eval Postfix
   - d. Infix → Postfix

781

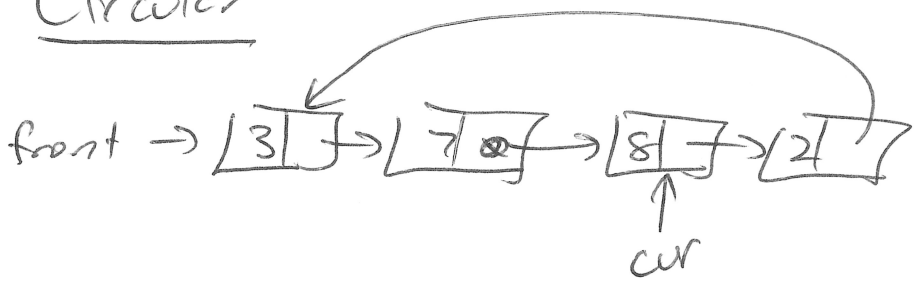
n1->data, n2->data

```
d = (n1->data + n2->data + carry) % 10;
carry = (n1->data + n2->data + carry) / 10;
```

String



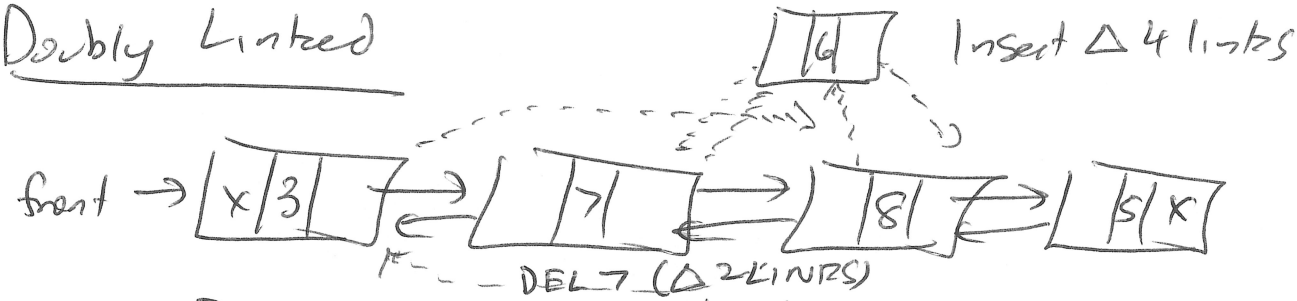
Circular



Care has to be taken to not infinite loop + make sure you go through everything once.

Special case 0→1 node 1→0 nodes

Doubly Linked

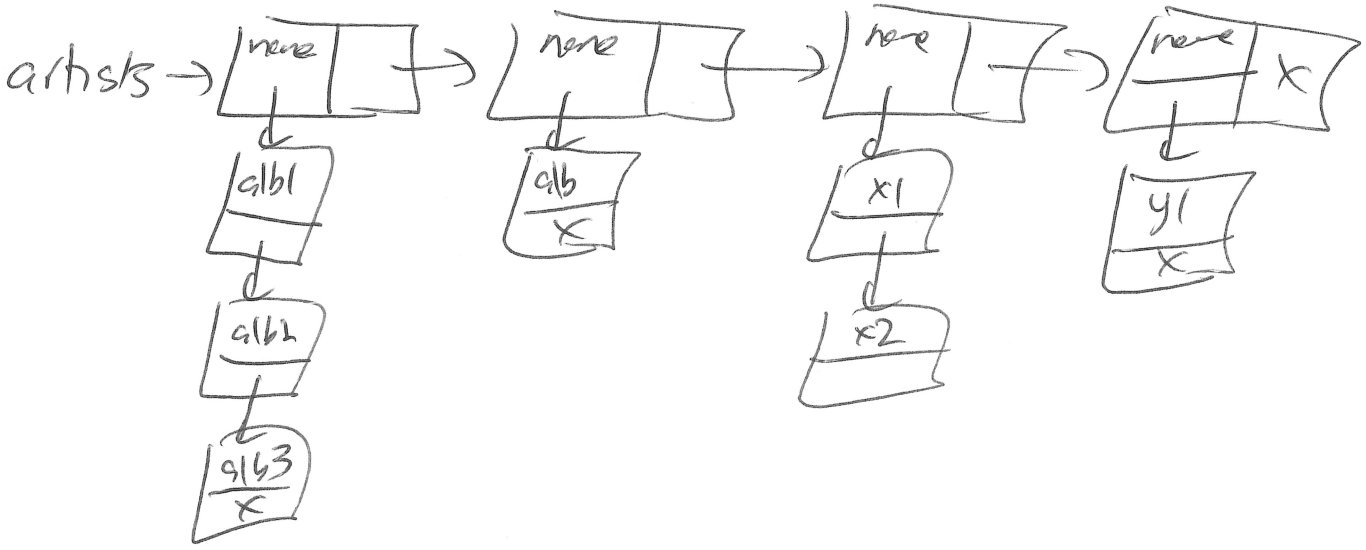


Benefit: Can go backwards

Drawback: More stuff to maintain

prev is the other link

LL of LL



Stacks (Abstract Data Type)

last in, first out

3
8

Stack

push(8)
push(3)
push(7)
x = pop() // x => 7

Operations

push - put on top
pop - remove, ret top item
top - return top item
size - how many

O(1) expected time

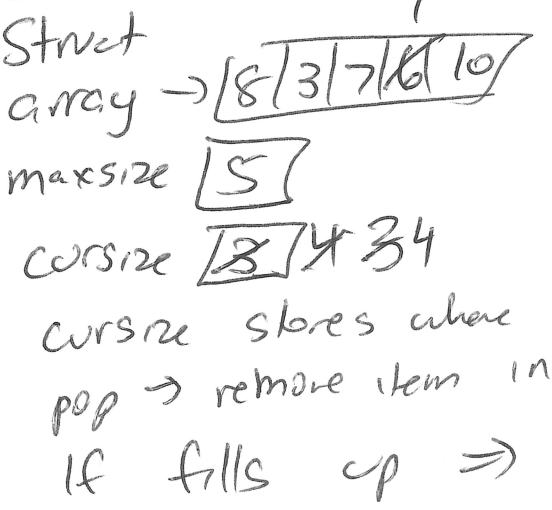
How to store it isn't specified, just behavior is.

---

(1) Implement via array

(2) Implement via LL

Array



cursize stores where next push will be

pop → remove item index cursize-1

If fills up ⇒ double size realloc.

push(7)
push(3)
push(8)

push insert front
pop del front

8
3
7

Stack

x = pop()

→ x

top → [7|x]

top → [3|→] → [7|x]

top → [8|→] → [3|→] → [7|x]

top → [3|→] → [7|x] , return 8

Stack Applications

(a) Evaluating Postfix Expression

(b) Converting Infix → Postfix

op1 op2 op

3 7 + (3+7) = 10

3 + 7 * 5 } We need order of op. to make infix unambiguous

(3+7) * 5

3 + (7 * 5) → opt

Postfix

3 7 + 5 *

3 7 5 * +

Alg to Eval a Postfix Expr

8 4 / 2 6 3 + * -

use operand stack
go through the symbols

if operand
  push onto stack

else
  operand2 = pop()
  operand1 = pop()
  push (operand1 op operand2)

8	4	2	6	3
	2	2	9	2
			18	18
				-16

8/4=2 6+3 2*9 2-18

if you ever try to pop empty stack
  expr is invalid
if at end ≥ 1 item stack
  expr is invalid