# COP 3502 - 9/12/23

① Recitaton Reminders

- Quiz back
- Groups

② Stacks

→ Abstract Data Structure

- linked list implementaton
- array implementation
- 2 applications of a stack
  - Evaluating a postfix expr
  - Converting Infix to Postfix

→ where the behavior is specified but how the data is stored to support it is NOT.

Behaviors

push - top an obj on top

pop - remove top item stack (usually returned)

top - return top stack

empty - true iff stack is empty

full - if there's a limit

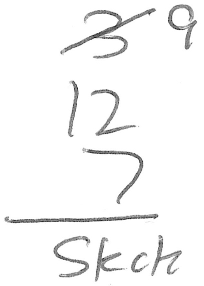
Supported in O(1) time

LIFO

Last In

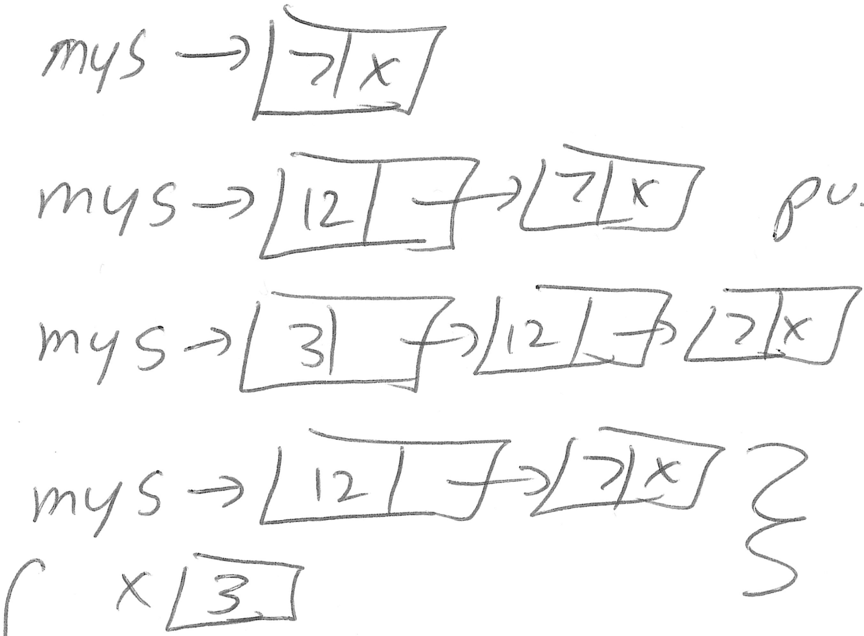
First Out

push(7)
push(12)
push(3)
x = pop()
x [3]
push(9)



etc.

## Implementation LL



push ⟺ insertFront

pop ⟺ deleteFront

x [3.]

3 solns

### SOL 1

node* pop(node* front, int* ptrRetVal)
ptrRetVal

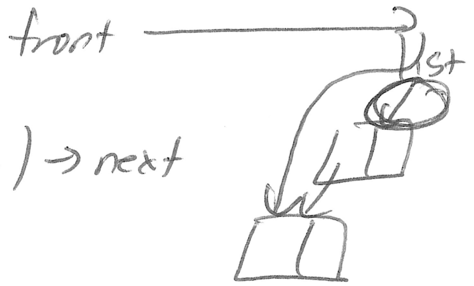
main
val [3]
pop(list, &val)

```
*ptrRetVal = front -> data
  // free stuff
  // ret new front
```

```
Int pop(node** front) {

    *front = (*front)->next

}
```



main

```
node* list;
  }

Int top=pop(&list);
```

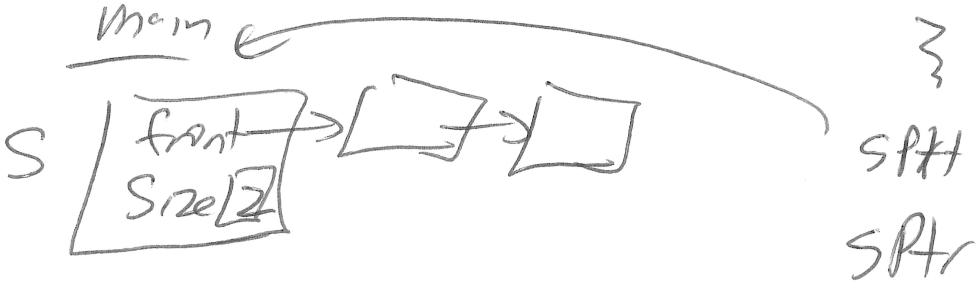
---

Sol 3

```
typedef struct stack {
    node* front;
    int size
} stack;
```

```
Int pop( ~~stack~~ stack* sPtr){

}
```



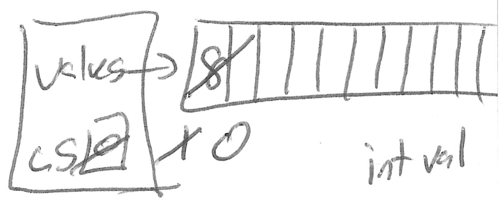
sPtr->front->next, etc

---

## Array Implementation

```
typedef struct stackarray {
    int values[10];
    int cursize;
} ~~cursize~~;
  stackarray
```

push(8)



```
~~void~~ int push(stackarray* sPtr, int val){
    if (full(sPtr)) return 0;
    sPtr->values[sPtr->cs]=val;
    sPtr->cs++;
}   return 1;
```

```
int  pop(stackarray* sPtr) {
   if (sPtr->cs == 0) return -1; //failed
   int retval = sPtr->values[sPtr->cs - 1];
   sPtr->cs--;
   return retval;
}
```

## Stack Applications

① Evaluating a Postfix Expression

3 7 + ⇐⇒ 3 + 7 (operator is in between operands)

postfix (operator is at the end)

infix

Infix ambiguous

$3 + \underline{7 * 5}$

$(3+7) * 5$

3 7 + 5 *

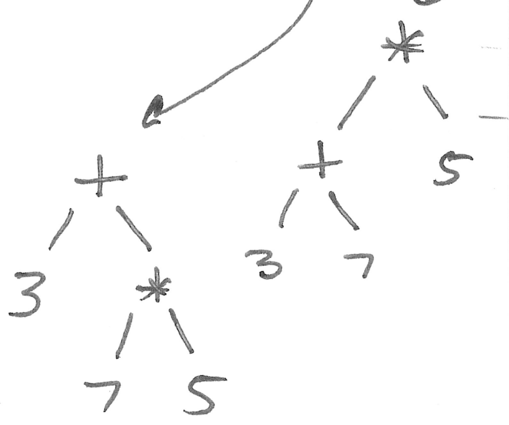
3 7 5 * +

6 3 - ⇒ 3

3 6 - ⇒ -3

op1 op2

3 5 + 2 4 + * 7 4 - /



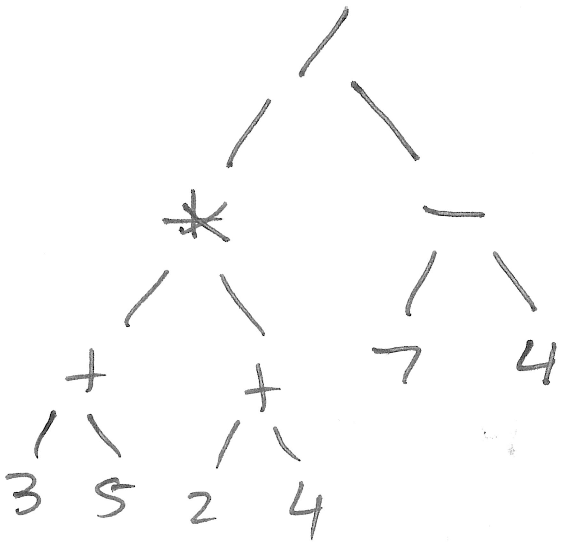
$$\underbrace{\underbrace{\underbrace{3\ 5\ +}\ \underbrace{2\ 4\ +}\ *}\ \underbrace{7\ 4\ -}\ /}$$

Read L→R
if operand,
push stack
if operator,
pop off last 2
(op2, op1)
calculate
op1 op op2
push onto stack

| | | | | | |
|---|---|---|---|---|---|
| | 4 | | ~~4~~ | | |
| 5 | 2 | 6 | ~~7~~ | 3 | |
| 3 | 8 | 8 | 48 | 48 | $\boxed{16}$ |
| Stack | Stack | Stack | Stack | 7-4 | |
| | 3+5 | 2+4 | 8*6 | | 48/3 |

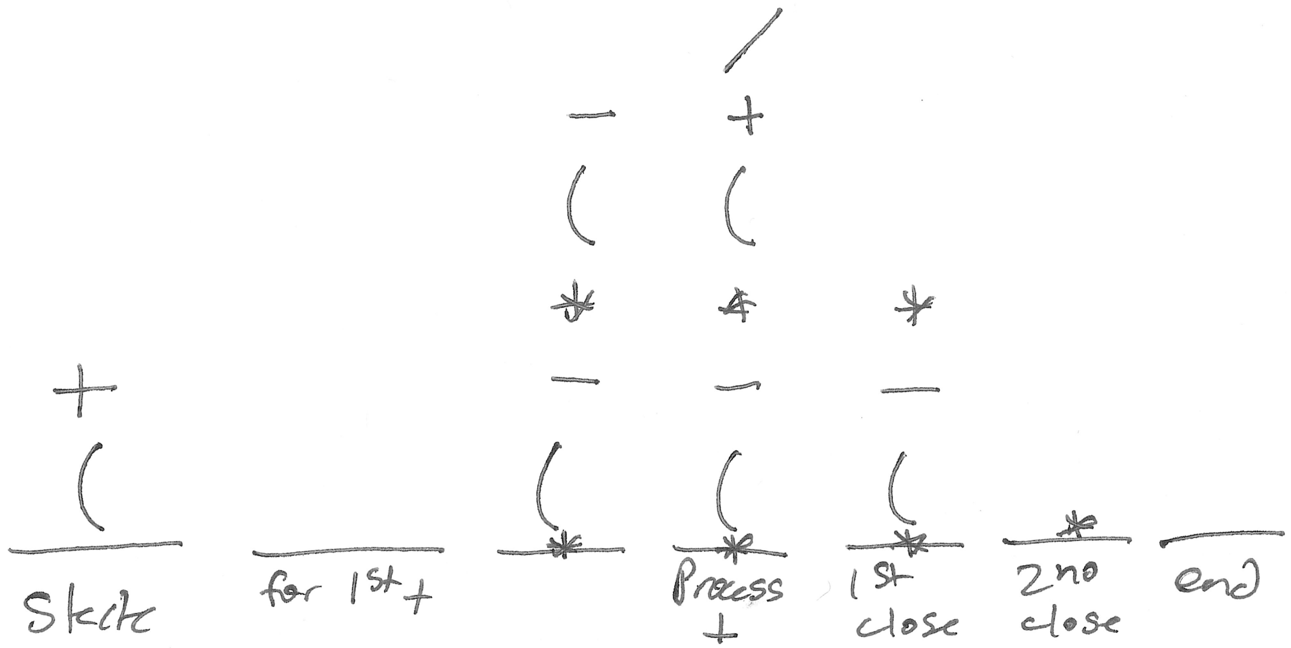
Number on Stack at end is value of the expression

if stack try pop ⇒ expr invalid
if end > 1 item ⇒ expr invalid



# Infix → Postfix

(3 + 4) * (6 - 2 * (8 - 7 + 6 / 6))



Output: 3 4 + 6 2 8 7 - 6 6 / + * - *

3 4 + 6 2 8 7 - 6 6 / + * - *

# Rules for Alg

✓ 1. Open Paren : Push onto Stack (parens and operators)

✓ 2. Number : place it in the expression.

✓ 3. Close Paren : Pop off each operator from stack and place into the expression (in this order) until you reach the corresponding open parenthesis in the stack

✓ 4. Operator : While the precedence of the operators on the stack are equal of greater then the new operator pop off each operator + place into the expression, stop stack empty or hit a ~~pra~~ parenthesis. Push new operator onto stack.