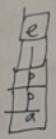
2) a)

input	stack
{	{
[	{[
]	{
[	{[
(	{[(
)	{[
]	{
null	stack not empty, so unbalanced

b)

input	stack
(	(
(	((
)	(
{	({
(	({(
[	({([
]	({(
)	({
}	(
)	
null	stack empty so balanced!

3) - put input string char by char onto stack. For example:

apple Celppa becomes → 

- if next char is == C compare next @ input which is "e" to top of stack. ~~If so, then~~
- If the two are equal, pop the top of stack and continue w next input.
- If not equal, then x and y are not palindromes
- When stack is empty, repeat.

example

apple Celppa

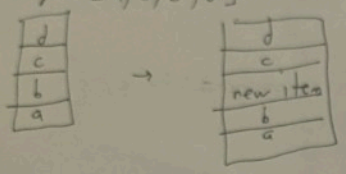
	input	stack	equal?
	a	a	
	p	ap	
	p	app	
	l	appl	
	e	apple	
start popping →	C		
	e	appl	
	l	appl	
	p	app	
	p	ap	
	a	a	
	null	null	@ empty so palindrone!

4)

```
isValid(inputString) {
  comparing = false
  for (i = 0; i < inputString.length; i++) {
    letter = inputString[i]
    if (letter != "C" and letter != "D" and comparing == False) {
        stack.push(letter)
    }
    if (letter == "C") {
        comparing = True
    }
    if (comparing == True)
        while (!stack.isempty()) {
            stackItem = stack.pop()
            if(!stackItem == letter) {
                throw Error
            }
        }
    if (letter == "D") {
        comparing = False
    }
}
return True    # if no error, then String is valid.
```

5) Say array[2] = "new item"
we could use a stack

Say array = [a, b, c, d]



how? →

algo:
if index == 2 and item = "new item"

left	stack	aux stack
	abcd	
	abc	d
	ab	dc
	a	dcb
		dcba

2 times pop and push to original stack

a	dcb
ab	dc

Insert new item

ab "new item"	dc
ab "new item" c	d
ab "new item" c d	

6) 10 stack size in array

index	
0	S
1	T
2	A
3	C
4	K
5	
6	H
7	E
8	A
9	P

in this scenario, stack1 size = 5
stack2 size = 4
Last Index = 9

```
def push1(item)
    if (space() != 0) {
        array[stack1size] = item
        stack1size += 1
    }
    else {
        throw Error("no space, will overflow")
    }
def push2(item) {
    if (space() != 0) {
        array[lastIndex - stack2size] = item
        stack2size += 1
    }
}
space() {
return stack1size - stack2size
}
```

only if stack1size > 0

```
def pop1() {
    temp = array[stack1size - 1]
    array[stack1size - 1] = null
    stack1size -= 1
    return temp
```

only if stack2size > 0

```
def pop2() {
    temp = array[lastIndex + 1 - stack2size]
    array[lastIndex + 1 - stack2size] = null
    stack2size -= 1
    return temp
```

7) (A+B) * (C $ (D-E) + F) - G

postfix ~~AB+*C(C$(DE-)+F)-G~~

input	stack	postfix
(	(	
A	(	A
+	(+	A
B	(+	AB
)		AB+
*	*	AB+
(	*(	AB+
C	*C	AB+C
$	*($	AB+C
(	*($(	AB+C
D		AB+CD
-	*($(-	AB+CD
E	*($(-	AB+CDE
)	*($	AB+CDE-
+	*(+	AB+CDE-$
F	*(+	AB+CDE-$F
)	*	AB+CDE-$F+
-	-	AB+CDE-$F+*
G	-	AB+CDE-$F+*G
null		AB+CDE-$F+*G-

8) A + (((B - C) * (D - E) + F) / G) $ (H - J)

input	stack	postfix
A		A
+	+	A
(	+(	A
(	+((	A
(	+(((	A
B	+(((	AB
-	+(((-	AB
C	+(((-	ABC
)	+((	ABC-
*	+((*	
(	+((*(	
D	+((*(-	ABC-D
-	+((*(-	ABC-D
E	+((*(-	ABC-DE
)	+((*	ABC-DE-
+	+((	ABC-DE-*
F	+((+	ABC-DE-*F
)	+(	ABC-DE-*F+
/	+(/	ABC-DE-*F+
G	+(/	ABC-DE-*F+G
)	+	ABC-DE-*F+G/
$	+$	
(	+$(	
H	+$(	ABC-DE-*F+G/H
-	+$(-	
J	+$(-	ABC-DE-*F+G/HJ
)	+	ABC-DE-*F+G/HJ-
null		ABC-DE-*F+G/HJ-$+

9) a) ~~[illegible]~~ A B + C - B A + C $ -
1 2 + 3 - 2 1 + 3 $ -

stack	op
1 [illegible]	
1, 2	+
3	
3, 3	-
0	
[illegible] 2	
2, 1	+
3, 3	$
27	-

Incomplete?

b) 1 2 3 + * 3 21 - + *

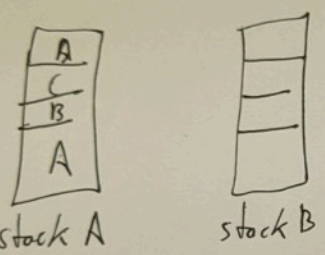
1		
1, 2		
1, 2, 3	+	
1 5	*	
5, 3		
5, 3, 2		
5, 3, 2, 1	-	b - a → 2 - 1
5, 3, 1	+	
5, 4	*	
20		

10)
```
create Prefix (infix)
  stack = new Stack   <- operators = {"+":1, "-":1, "*":2, "/":2, "$":4}
  infix = infix.split()
    for (i = infix.length-1 ; i ≥ 0 ; i--) {
      if (infix[i] not in operators.keys())
          ~~stack.push~~
          output.prepend(infix[i])
    if (infix[i] [illegible] = ")")
          stack.push(")")
    if (infix[i] == "(")
      while (stack.peek() != [illegible] ")")
          item = stack.pop()
          output.prepend(item)
      if stack.peek() == ")"
          stack.pop()
      else
          throw Error.
```

10) (continued)

```
if (infix[i] in operators.keys())
    if operators[stack.peek()] < operators[infix[i]]
        output.prepend(infix[i])
else:
    while (operators[stack.peek()] > operators[infix[i]]
        item = stack.pop()
        output.prepend(infix[i])
```

1)




```
getBottom():
i = null
while stackA.isEmpty(stackA) = False
    if stackA.peek() != null
        item = stackA.pop()
        stackB.push(item)
    else:
        i = stackA.pop()
        stackB.push(i)
while (stackB.isEmpty(stackB) == False) {
    item = stackB.pop()
    stackA.push(item)
```

2) Set X From Bottom (num): num == 2

```
i = null
while isEmpty(stackA) = False
    item = stackA.pop()
    stackB.push(item)
for (i=0; i < num; i++)
    item = stackB.pop()
    stackA.push(item)
i = stackB.pop()
while stackB.isEmpty(stackB) == False
    item = stackB.pop()
    stackA.push(item)
```