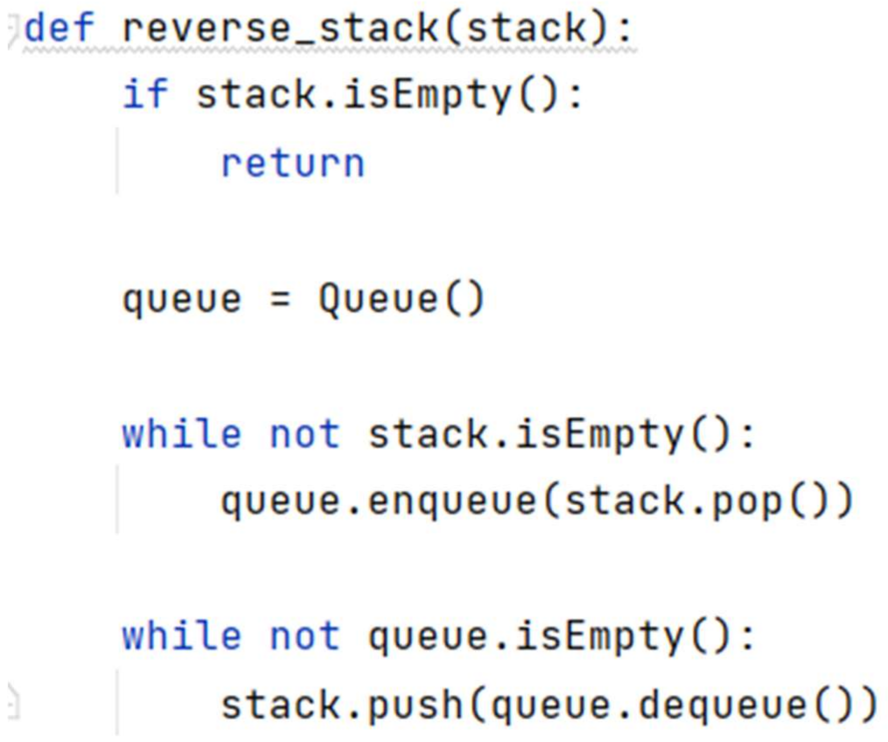
Tutorial 2 (Week 6)
Stack and Queues

Instructor: Dr. Quah T.S., Jon
Email: itsquah@ntu.edu.sg

Q1 (reverseStack)

Full program is uploaded to our tutorial folder

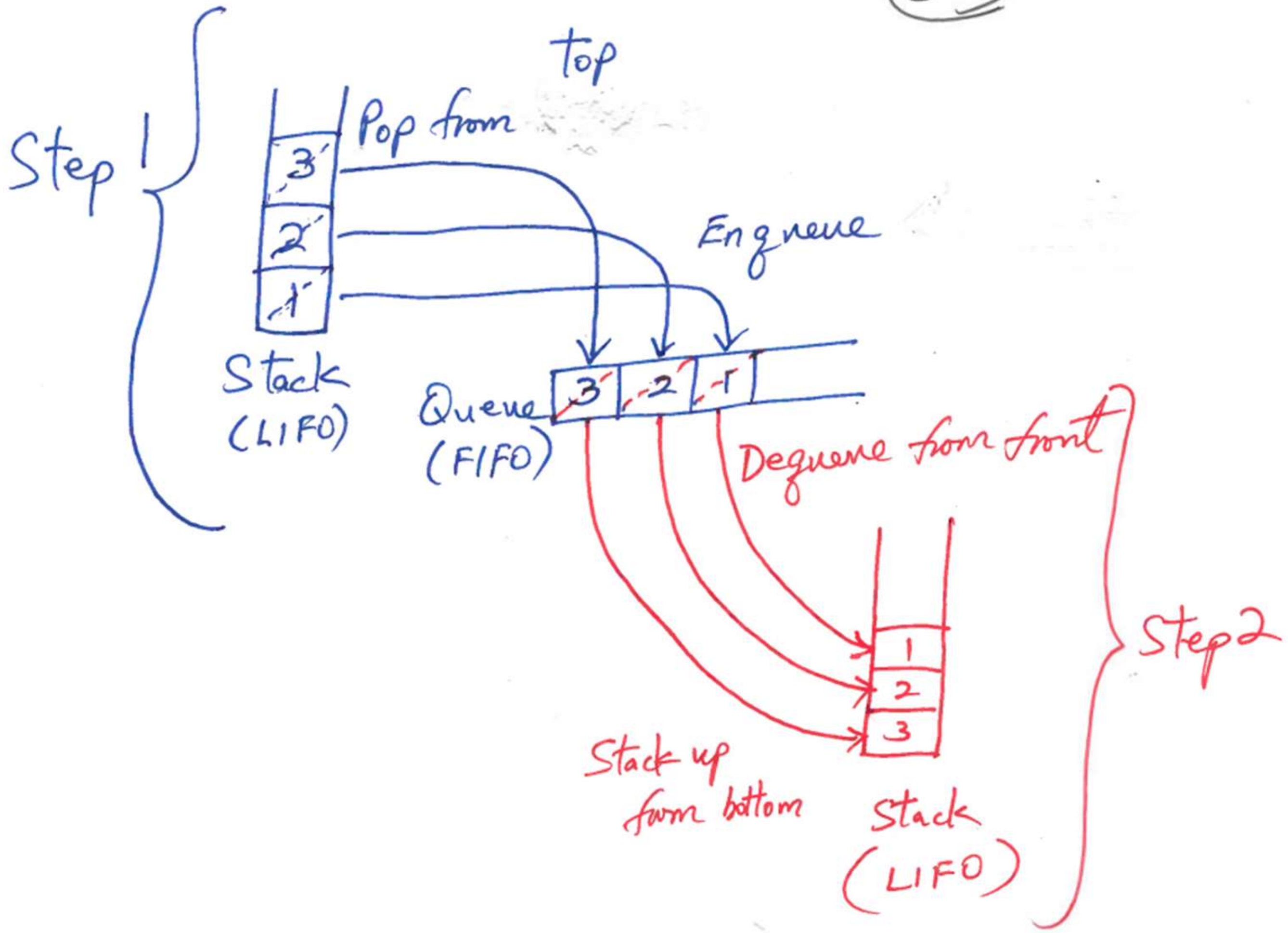


```python
def reverse_stack(stack):
    if stack.isEmpty():
        return

    queue = Queue()

    while not stack.isEmpty():
        queue.enqueue(stack.pop())

    while not queue.isEmpty():
        stack.push(queue.dequeue())
```

Step 1

Step 2

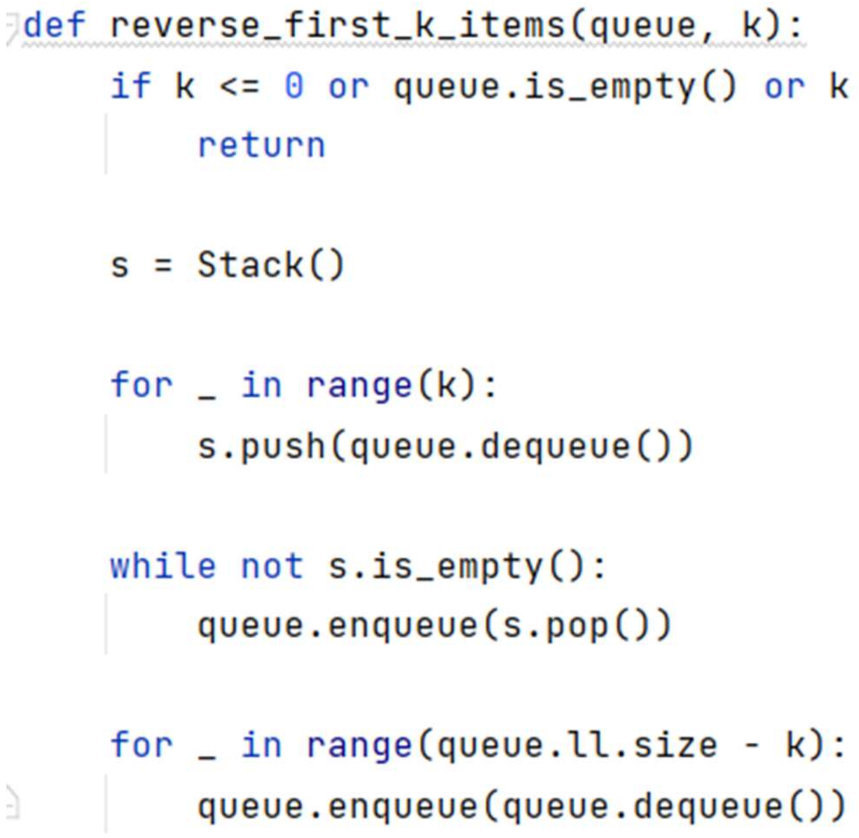
Q1



Step 1
top
Pop from
3
2
1
Enqueue
Stack
(LIFO)
Queue
(FIFO)
3
2
1
Dequeue from front
1
2
3
Step2
Stack up
from bottom
Stack
(LIFO)

Q2 (reverseFirstKItems)

Full program is uploaded to our tutorial folder

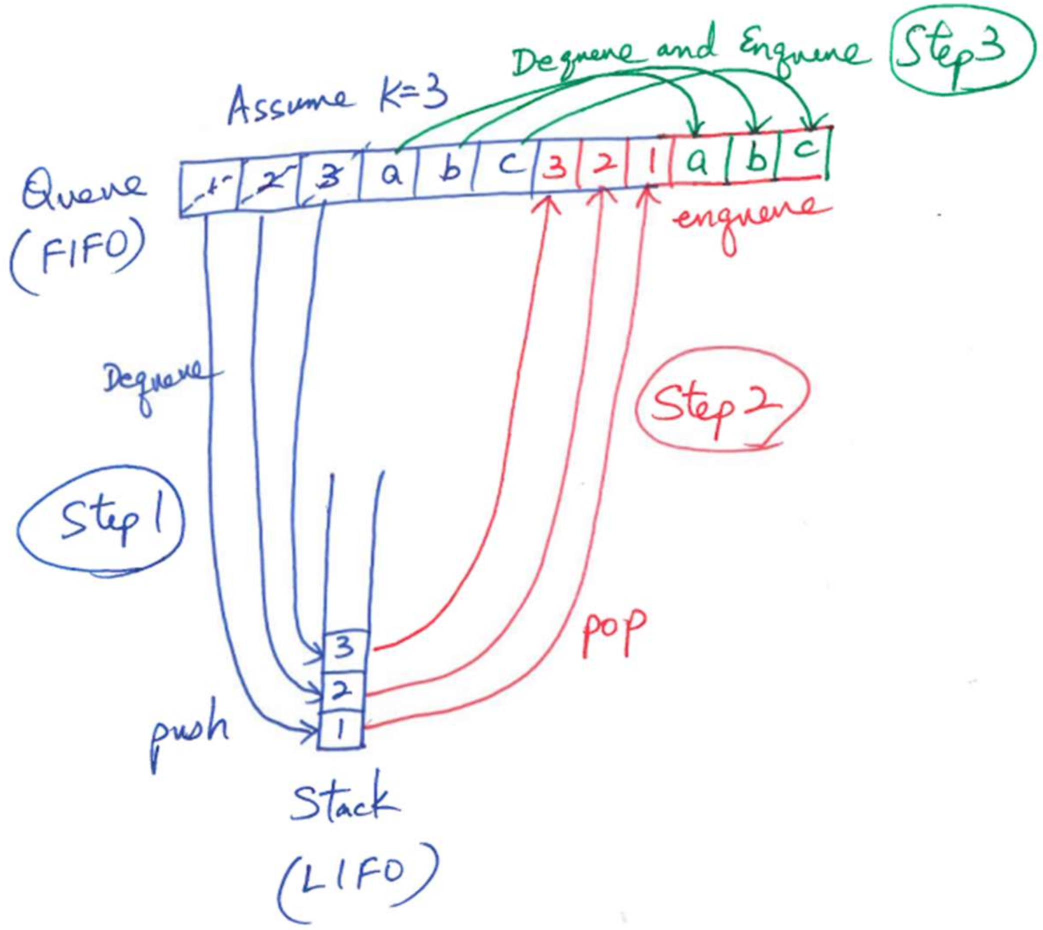


```python
def reverse_first_k_items(queue, k):
    if k <= 0 or queue.is_empty() or k > queue.ll.size:
        return

    s = Stack()

    for _ in range(k):
        s.push(queue.dequeue())

    while not s.is_empty():
        queue.enqueue(s.pop())

    for _ in range(queue.ll.size - k):
        queue.enqueue(queue.dequeue())
```

Step 1

Step 2

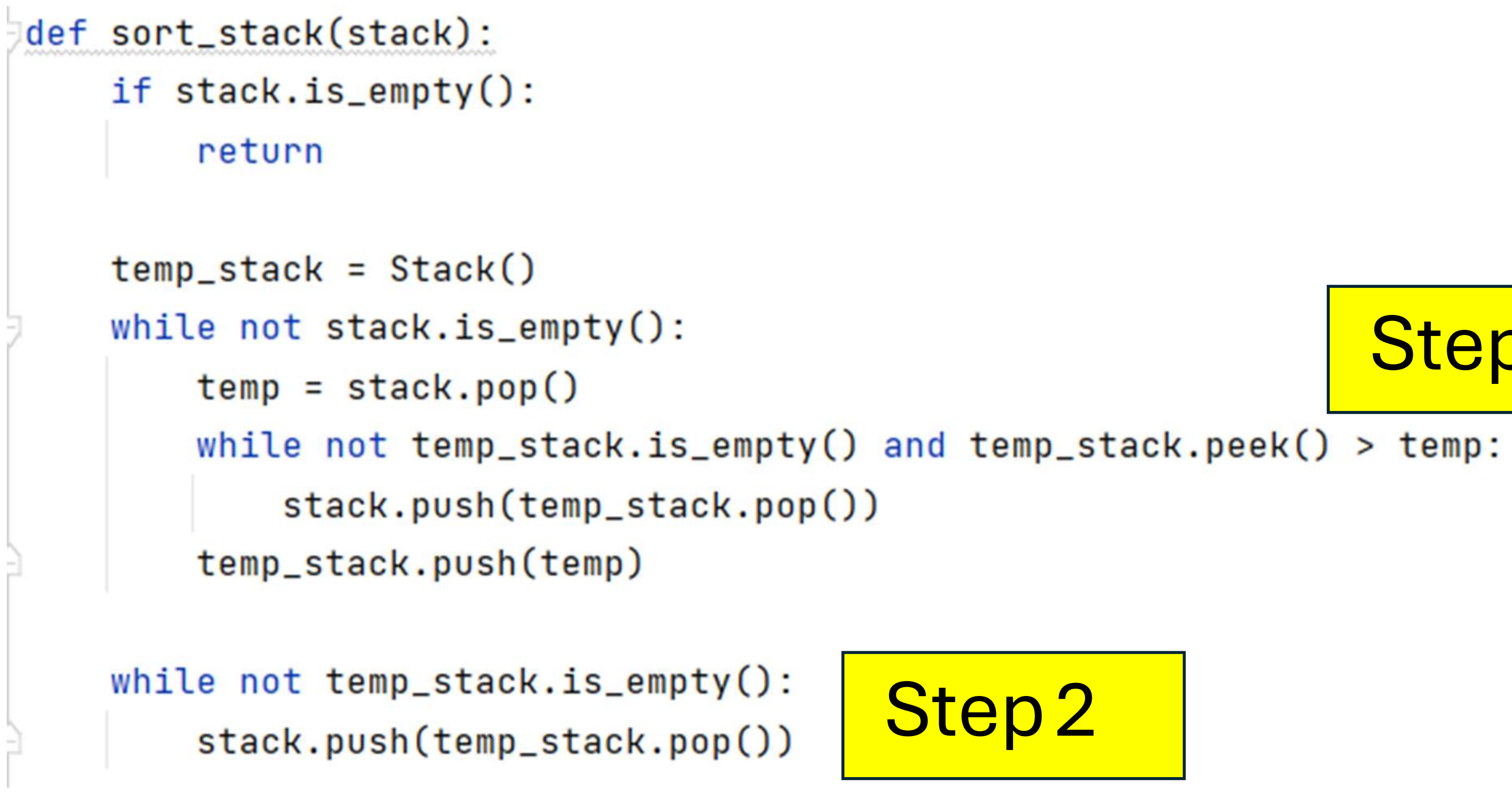
Step 3


Assume K=3
Dequeue and Enqueue
Step3
Queue
(FIFO)
1
2
3
a
b
c
3
2
1
a
b
c
enqueue
Dequeue
Step 2
Step 1
pop
3
2
1
push
Stack
(LIFO)

Q2

Q3 (sortStack)

Full program is uploaded to our tutorial folder

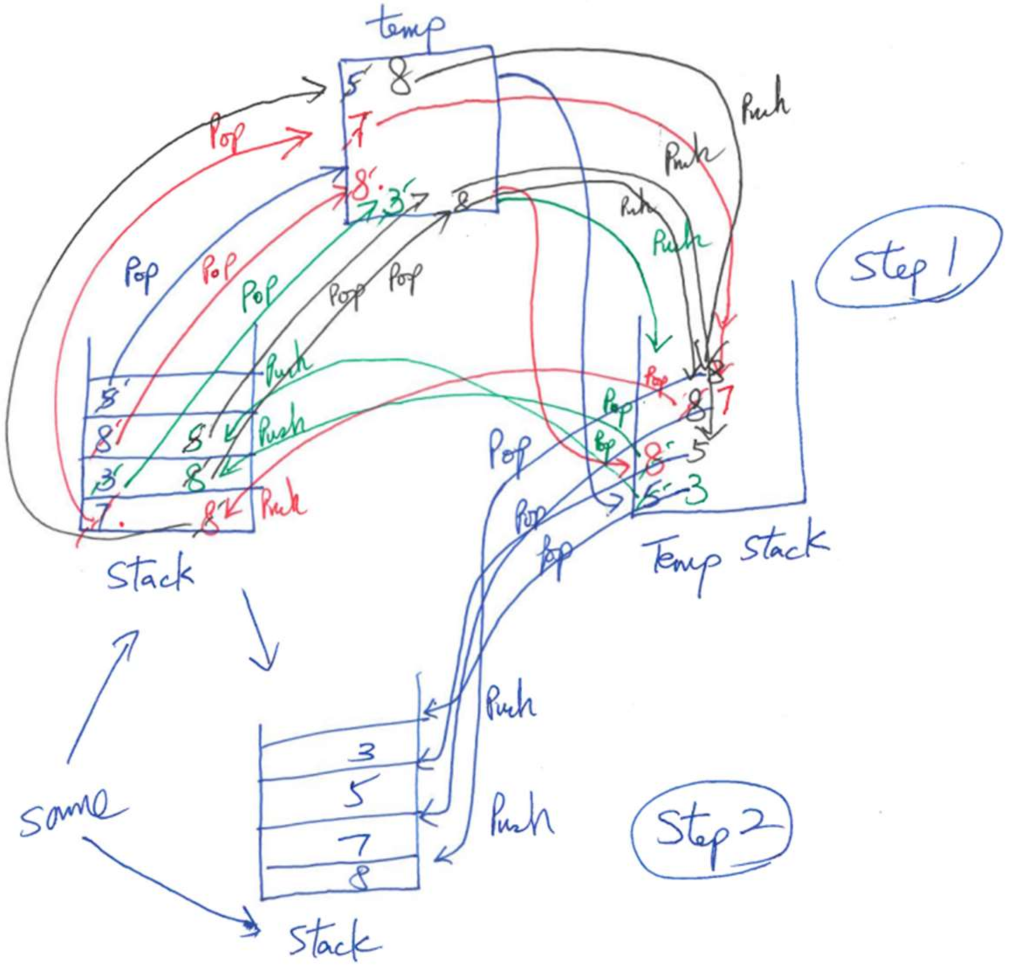


```python
def sort_stack(stack):
    if stack.is_empty():
        return

    temp_stack = Stack()
    while not stack.is_empty():
        temp = stack.pop()
        while not temp_stack.is_empty() and temp_stack.peek() > temp:
            stack.push(temp_stack.pop())
        temp_stack.push(temp)

    while not temp_stack.is_empty():
        stack.push(temp_stack.pop())
```

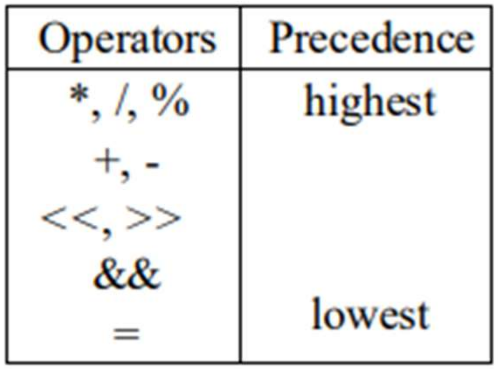
Step 1

Step 2

Q.3

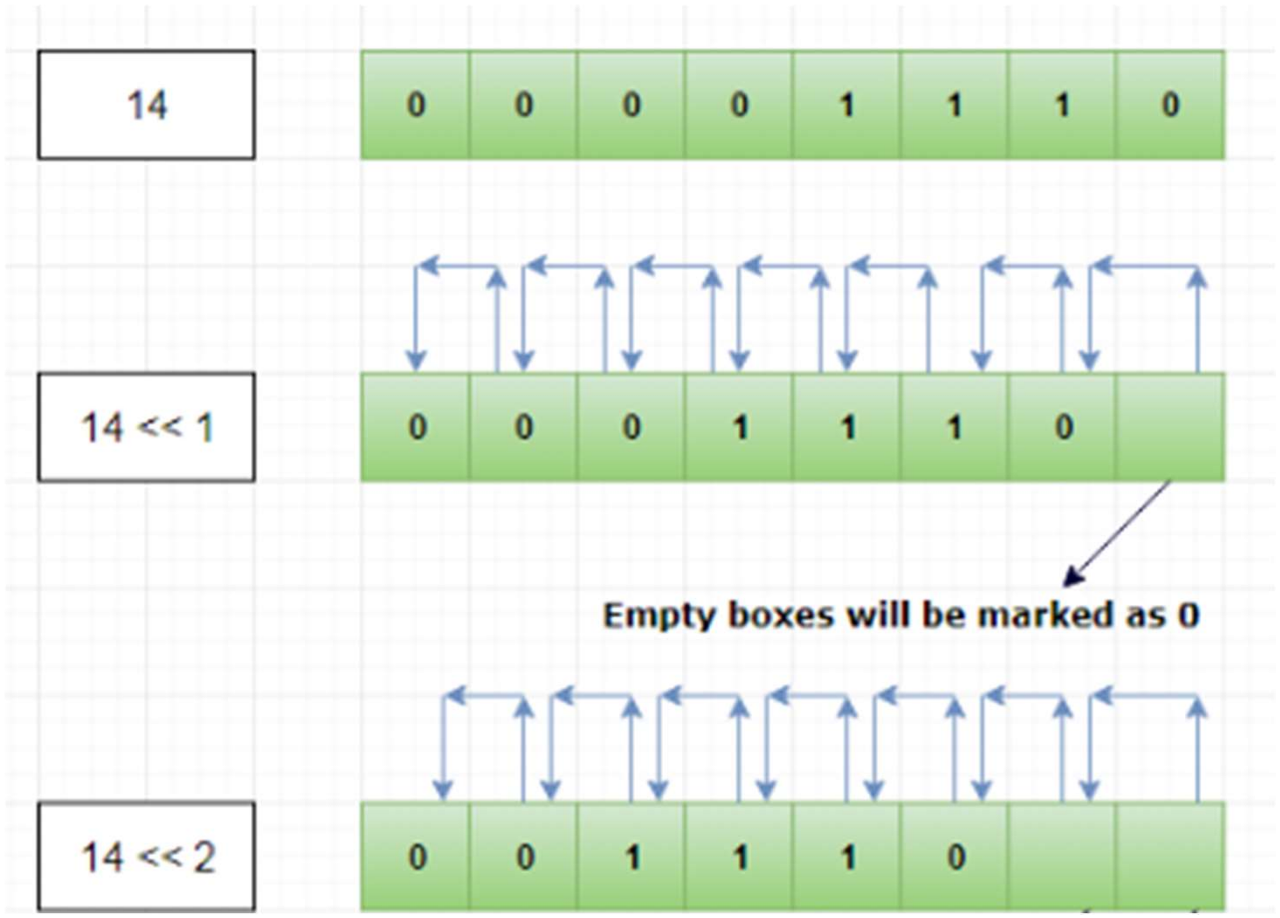


Q4 (expressions)



Operators	Precedence
*, /, %	highest
+, -	
<<, >>	
&&	
=	lowest

(a) convert an infix expression, x = a + b c%d >> e, to a postfix expression
(b) convert a prefix expression, = y&& << ab >> c + de, to an infix expression
(c) convert a postfix expression, xabc d% + e >>=, to a prefix expressio



14
0 0 0 0 1 1 1 0
14 << 1
0 0 0 1 1 1 0
Empty boxes will be marked as 0
14 << 2
0 0 1 1 1 0

Infix to Postfix

- Converting infix to postfix *also* uses a stack
  - Postfix needs to re-arrange operators into the right place
  - So we need to 'hold on' to operators until we reach the right point in the equation to insert them back in
    - Remember that operands don't change their order
  - The method behind this is to hold back an operator until we see an equal-or-lower-precedence operator
    - If the new operator is higher precedence, we have to put it 'on top' of the other operator (in a stack), since it takes precedence

(a) convert an infix expression, x = a + b c%d >> e, to a postfix expression

(i) x =

(ii)
a +
x =

(iii)
b *
a +
x =

c (%)
(iv)
b *
a +
x =

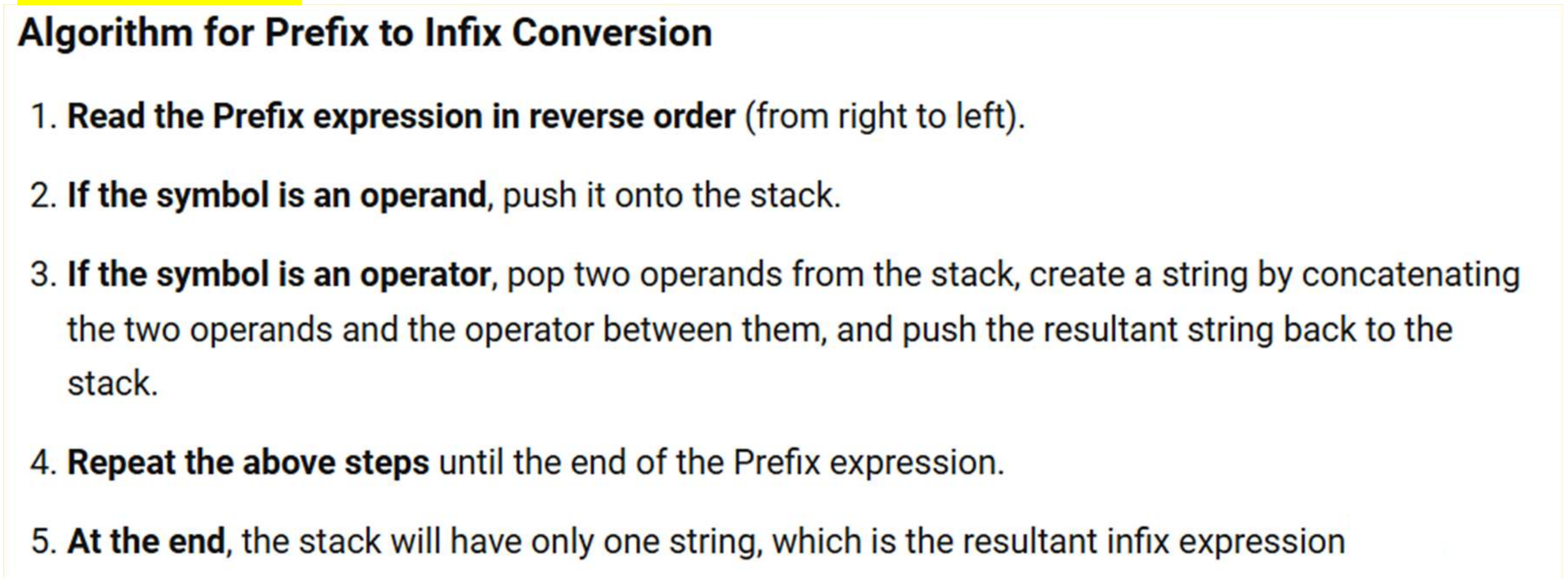
(v) x a b c * = + %

(vi) x a b c * d = + % (>>)

(vii) x a b c * d % + e
= >>

(viii) x a b c * d % + e >> =

Prefix to Infix



Algorithm for Prefix to Infix Conversion

1. **Read the Prefix expression in reverse order** (from right to left).
2. **If the symbol is an operand**, push it onto the stack.
3. **If the symbol is an operator**, pop two operands from the stack, create a string by concatenating the two operands and the operator between them, and push the resultant string back to the stack.
4. **Repeat the above steps** until the end of the Prefix expression.
5. **At the end**, the stack will have only one string, which is the resultant infix expression

(b) convert a prefix expression, $\boxed{= y \&\& << ab >> c + de}$ to an infix expression

$$\begin{tikzcd} \begin{matrix}\textcircled{+}\\ d\\ e\end{matrix} \arrow[r] & \begin{matrix}\textcircled{>>}\\ c\\ d+e\end{matrix} \arrow[r] & \begin{matrix}\textcircled{<<}\\ a\\ b\\ c>>d+e\end{matrix} \arrow[r] & \begin{matrix}\textcircled{\&\&}\\ a<<b\\ c>>d+e\end{matrix} \arrow[r] & \begin{matrix}\textcircled{=}\\ y\\ a<<b \;\&\&\; c>>d+e\end{matrix} \end{tikzcd}$$

$$\rightarrow y = a<<b \;\&\&\; c>>d+e$$

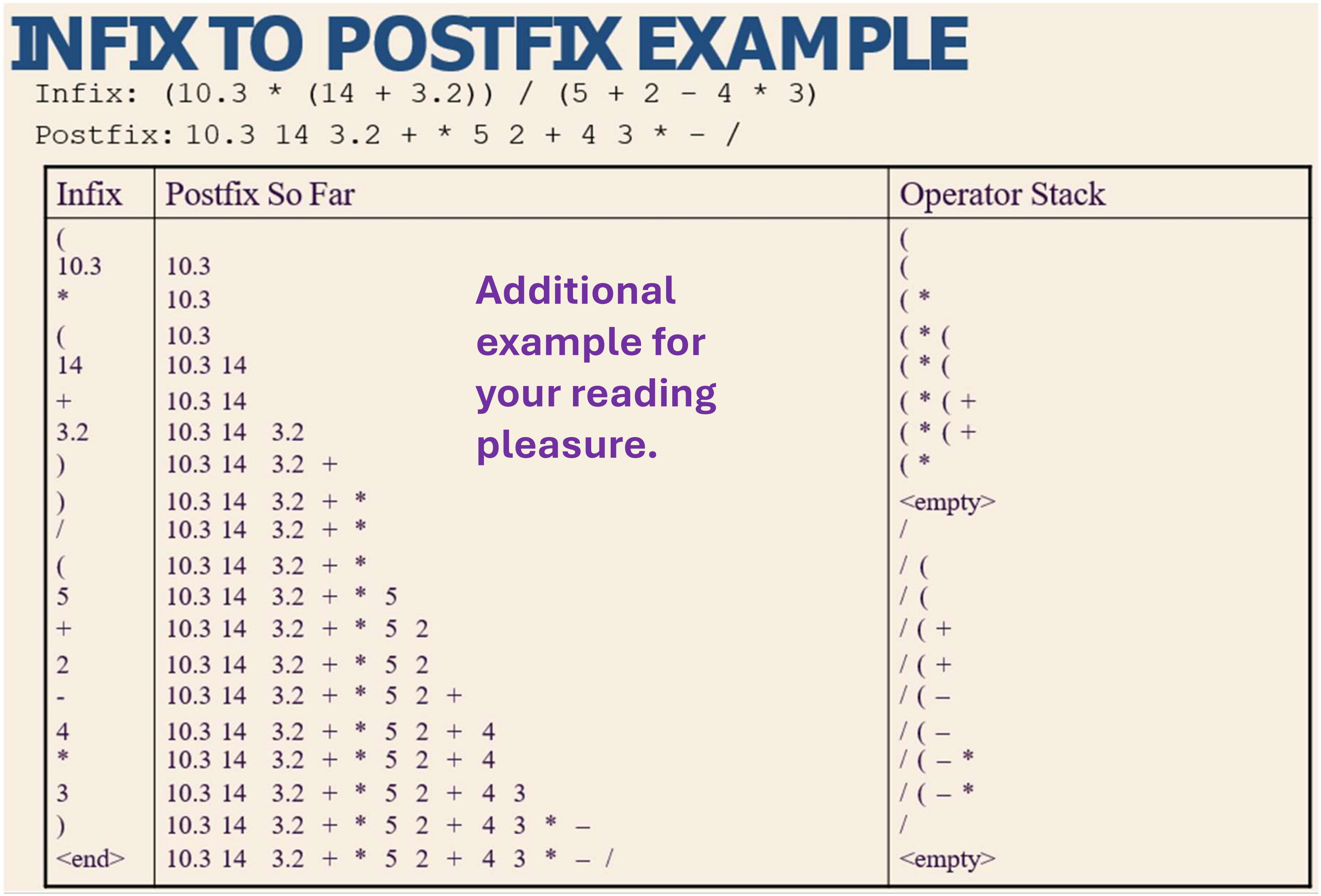
Postfix to Prefix

1. **Read the postfix expression from left to right.**
2. **If the symbol is an operand, push it onto the stack.**
3. **If the symbol is an operator, pop two operands from the stack.**
4. **Create a string by concatenating the operator before the two operands.**
5. **Push the resultant string back to the stack.**
6. **Repeat the above steps until the end of the postfix expression.**

(c) convert a postfix expression, $\boxed{\text{xabc*d\% + e >>=}}$, to a prefix expressio

$$\begin{tikzcd}
 & \textcircled{*} & & \textcircled{\%} & & & & \textcircled{>>} & & \\
 & c & & d & & \textcircled{+} & & e & & \textcircled{=} \\
\textcircled{c} & b \arrow[r] & {} \arrow[r, phantom] & *bc \arrow[rr] & & \%*bcd \arrow[rr] & & +a\%*bcd \arrow[rr] & & >>+a\%*bcde \\
 & a & & a & & a & & x & & x \\
 & x & & x & & x & & & &
\end{tikzcd}$$

$$\rightarrow\ =x>>+a\%*bcde$$



INFIX TO POSTFIX EXAMPLE

```
Infix: (10.3 * (14 + 3.2)) / (5 + 2 - 4 * 3)
Postfix: 10.3 14 3.2 + * 5 2 + 4 3 * - /
```

Infix	Postfix So Far	Operator Stack
(		(
10.3	10.3	(
*	10.3	(*
(	10.3	(* (
14	10.3 14	(* (
+	10.3 14	(* (+
3.2	10.3 14 3.2	(* (+
)	10.3 14 3.2 +	(*
)	10.3 14 3.2 + *	<empty>
/	10.3 14 3.2 + *	/
(	10.3 14 3.2 + *	/ (
5	10.3 14 3.2 + * 5	/ (
+	10.3 14 3.2 + * 5 2	/ (+
2	10.3 14 3.2 + * 5 2	/ (+
-	10.3 14 3.2 + * 5 2 +	/ (–
4	10.3 14 3.2 + * 5 2 + 4	/ (–
*	10.3 14 3.2 + * 5 2 + 4	/ (– *
3	10.3 14 3.2 + * 5 2 + 4 3	/ (– *
)	10.3 14 3.2 + * 5 2 + 4 3 * –	/
<end>	10.3 14 3.2 + * 5 2 + 4 3 * – /	<empty>

Additional example for your reading pleasure.