Polytechnic Tutoring Center

Exam 2 Review CS 1134, Spring 2026

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department.

Question 1 A (5 points)

Translate the following prefix expressions into postfix, and find its value:

"- * 3 + 2 4 / 5 + 6 1"

Value = _____

Postfix expression = _____

Question 1 B (1 points each, total 8 points)

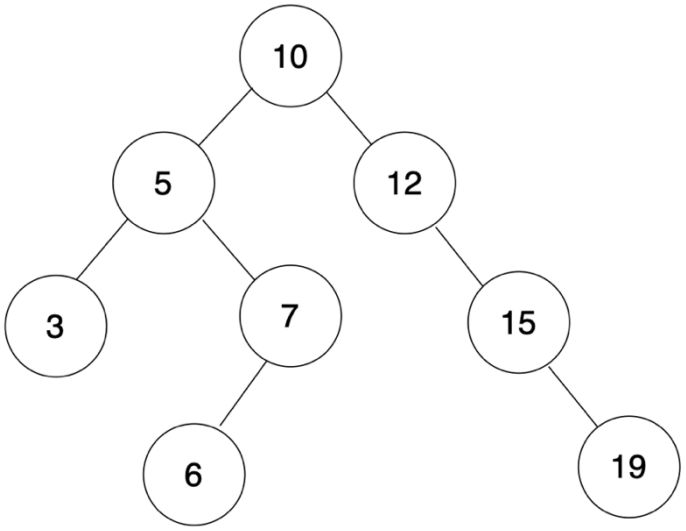
```
class ArrayQueue:
    INITIAL_CAPACITY = 2
    def __init__(self):
        self.data_arr = make_array(ArrayQueue.INITIAL_CAPACITY)
        self.capacity = ArrayQueue.INITIAL_CAPACITY
        self.n = 0
        self.front_ind = None
    def __len__(self): ...
    def is_empty(self): ...
    def enqueue(self, elem): ...
    def dequeue(self): ...
    def first(self): ...
    def resize(self, new_cap): ...
```

Consider the ArrayQueue class on page 1. Fill in the table after each operation:

Operator	Return value	front_ind	n	capactiy	data_arr
q = ArrayQueue()					
q.enqueue('A')					
q.enqueue('B')					
q.first()					
q.dequeue()					
q.enqueue('C')					
q.enqueue('D')					
q.first()					

Question 2 (3 points each, total 12 points)

From the following binary tree, write down the different traversals inside of the table on the next page



Traversal	Your Answer
Preorder	
Inorder	
Postorder	
Breadth First Search	

Question 3 (20 points)

We define a new data structure called **ReversibleQueue**, which behaves like the ArrayQueue studied in class, but additionally supports a reverse() operation. Implement the operations on the next page using the requirements below.

Implementation Requirements:

A ReversibleQueue has the following operations:

1. rq = ReversibleQueue(): creates a new empty ReversibleQueue object.
2. rq.is_empty(): returns True if there are no elements in rq, or False otherwise.
3. len(rq): returns the number of elements currently stored in rq.
4. rq.enqueue(elem): inserts elem into rq.
5. rq.dequeue(): removes and returns the element that is currently first in line in rq, or raises an Exception if rq is empty.
6. rq.first(): returns (without removing) the element that is currently first in line in rq, or raises an Exception if rq is empty.
7. rq.reverse(): mutates rq so that the order of its elements is reversed.

For example, these operations should work:

```
>>> rq = ReversibleQueue()
>>> rq.enqueue(10)
>>> rq.enqueue(20)
>>> rq.enqueue(30)
>>> rq.first()
10
>>> rq.reverse()
>>> rq.first()
30
>>> rq.dequeue()
30
>>> rq.dequeue()
20
>>> rq.dequeue()
10
```

```
class ReversibleQueue:

    def __init__(self):









    def __len__(self):



    def is_empty(self):



    def enqueue(self, elem):









    def dequeue(self):
```

```
def first(self):
```

```
def reverse(self):
```

Question 4 (15 points)

Add a method to the DoublyLinkedList class:

```
def move_to_front(self, node_to_move)
```

when called on a non-empty DoublyLinkedList object, the method should **mutate** the list in-place to move a node and make it the topmost node of the list. The node_to_move is passed in as a reference to any node in any part of the list. For example, given a list that looks like this:

[1–2–3–4–5]

and given a reference to the node containing data 4, we should get:

[4–1–2–3–5]

you can assume node_to_move will never be a reference to the header or trailer node.

The method must have a linear runtime, or better

Answer is re-implementation of delet_node and add_first.

Question 5 (15 points)

Implement the following function which takes in a queue and a non negative integer k.
`def reverse_first_k(q, k):`
When called, it should reverse the first k elements of the queue. The remaining elements must stay in the same relative order.

If the q initially contains: `<1, 2, 3, 4, 5>`

After calling: `reverse_first_k(q, 3)`

Q will be: `<3, 2, 1, 4, 5>`

Requirements

You may use one ArrayStack and the function should run in O(n) time.

Question 6 (10 points)

Consider the following definition of the **left circular shift** operation:
If q is a queue containing the sequence
$\langle a_1, a_2, \ldots, a_{n-1}, a_n \rangle$
the **left circular shift** of q is:
$\langle a_2, a_3, \ldots, a_n, a_1 \rangle$
That is, the sequence obtained by moving the **front element to the back**, while shifting all other elements forward.
For example:

If variable q contains = `<2, 4, 6, 8, 10>`

After calling after calling `left_circular_shift(q),`
q contains = `<4, 6, 8, 10, 2>`

```
def left_circular_shift(q):
```