Polytechnic Tutoring Center

Exam 2 Review - CS 1134,Fall 2025

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.
This first page is for your reference to help solve this set of problems.

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

   `“- * 3 + 2 4 / 5 + 6 1”`

2. Write a function that takes a singly linked list and recursively prints it out in reverse. Assume that the function takes in the front node of the list. A sample node class is defined below. Assume that the first node has valid data and is not an empty header node.

```
class Node(object):
     def __init__(self, data=None, next_node=None):
     self.data = data
     self.next_node = next_node
```

3. Consider an ArrayQueue object, `my_queue`, with **capacity set to 4**, and the following code is executed:

   ```
   my_queue.enqueue(0)

   my_queue.enqueue(1)

   my_queue.enqueue(2)

   my_queue.dequeue()

   my_queue.enqueue(3)

   my_queue.dequeue()

   my_queue.enqueue(4)
   ```

   Draw the memory image after the above code is executed.

4. Given a string with an undefined number of open or closed parentheses and braces: ([and]), determine if the parentheses are balanced:

   “()[([])]” is balanced because every open (or [has a matching closing) or]. “[()(])]” is NOT balanced.

   You may assume you have predefined implementations of an array, stack, queue, and dequeue. You may assume the string passed as a parameter will only consist of ‘(‘, ’)’, ’[‘, ’]’ characters.

5. Implement a data structure, ReversibleQueue, using **only one data structures** provided at the beginning of this paper, you can choose any of them. There is more than one way of doing it. ReverseibleQueue must support the method, reverse(), that **mutates** the queue in-place such that the order of the elements inside is reversed. The first element in line becomes the last, and the last element becomes the first.

   <u>Implementation Requirements:</u>
   The ReversibleQueue must support all methods ArrayQueue supports in this paper.
   The runtime of every method must be at least as efficient as the ArrayQueue equiv.
   The runtime of reverse() must be **constant**
   The additional memory use to support reverse() must be **constant**

6. 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

7. Given a doubly linked list that contains repeatable digits, define a function that mutates this doubly linked list in place to group together all repeating digits. Linear runtime.

   Example:
   ```
   lnk_lst=[1 <--> 7 <--> 3 <--> 3 <--> 1 <--> 5 <--> 7]
   ```
   becomes:
   ```
   lnk_lst=[1 <--> 1 <--> 7 <--> 7 <--> 3 <--> 3 <--> 5]
   ```

8. Given a node in a singly linked list, write a function to remove all subsequent instances of a single number passed as a parameter. Assume the list has at least one element

   ```
   class Node(object):
         def __init__(self, data=None, next_node=None):
         self.data = data
         self.next_node = next_node
   ```

9. Assume that a predefined implementation of a **dequeue** has been provided with the following methods:

```
deq.push_front(element):
"""inserts an element to the back of the dequeue"""

deq.push_back(element):
"""removes the front of the dequeue"""

deq.pop_front()
"""removes the front of the dequeue"""

deq.pop_back()
"""removes the back of the dequeue"""

deq.front()
"""returns the front element from the dequeue"""

deq.back()
"""returns the back element from the dequeue"""

deq.__len__()
"""returns the number of elements in the dequeue"""
```

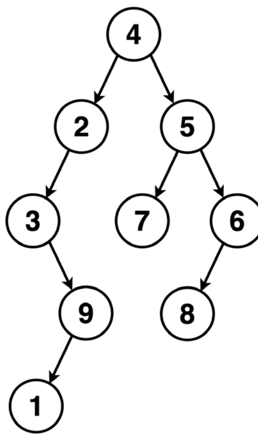
A *palindrome* is a string that is identical if read from front to back or from back to front. For example: "racecar" is a palindrome. Given a dequeue of characters, write a function that determines if the dequeue currently holds a palindrome. (you may modify the contents of the deque).

10. Write a function that, given a LinkedBinaryTree object, the function **recursively** checks if every single node in the tree contains the same data value. If so, return True. Otherwise, return False (If the tree is empty, return True) The function **must only take in one parameter/argument**, such as:

```
def mono(root):
```

11. Write a function that, given a LinkedBinaryTree object, the function **recursively** finds the longest branch in that tree and returns that length. **The function takes only one parameter/argument**

For example, if the tree looks like this with starting root containing the value ‘4’, the function returns 5, since the longest branch (down the path of 4 – 2 – 3 – 9 – 1) is 5-node-long.



```
class ArrayStack:
      def __init__(self):
      """initializes an empty ArrayStack object. A stack object has: data – an array, storing the elements currently in the stack in the order they
      entered the stack"""

      def __len__(self):
      """returns the number of elements stored in the stack"""

      def is_empty(self):
      """returns True if and only if the stack is empty"""

      def push(self, elem):
      """inserts elem to the stack"""

      def pop(self):
      """removes and returns the item that entered the stack last (out of all the items currently in the stack), or raises an Exception, if the stack
      is empty"""

      def top(self):
      """returns (without removing) the item that entered the stack last (out of all the items currently in the stack), or raises an Exception, if the
      stack is empty"""
```

```
class ArrayQueue:
      def __init__(self):
      """initializes an empty ArrayQueue object.
      A queue object has the following data members:
            1. data – an array, holding the elements currently in the queue in the order they entered the queue. The elements are stored
            in the array in a “ circular”  way (not necessarily starting at index 0)
            2. front_ind – holds the index, where the (cyclic) sequence starts, or None if the queue is empty
            3. num_of_elems – holds the number of elements that are currently stored in the queue"""

      def __len__(self):
      """returns the number of elements stored in the queue"""

      def is_empty(self):
      """returns True if and only if the queue is empty"""

      def enqueue(self, elem):
      """inserts elem to the queue"""

      def dequeue(self):
      """removes and returns the item that entered the queue first (out of all the items currently in the queue), or raises an Exception,
      if the queue is empty"""

      def first(self):
      """returns (without removing) the item that entered the queue first (out of all the items currently in the queue), or raises an
```

```
class DoublyLinkedList:
        class Node:
                def __init__(self, data=None, prev=None,
                next=None):
                """initializes a new Node object containing the
                following attributes:
                1. data - to store the current element
                2. next - a reference to the next node in the list
                3. prev - a reference to the previous node in the
                list """

                def disconnect(self):
                """detaches the node by setting all its attributes
                to None"""

        def __init__(self):
        """initializes an empty DoublyLinkedList object.
        A list object holds references to two "dummy" nodes:
                1. header - a node before the primary sequence
                2. trailer - a node after the primary sequence
                also a size count attribute is maintained"""

        def __len__(self):
        """returns the number of elements stored in the list"""

        def is_empty(self):
        """returns True if and only if the list is empty"""

        def first_node(self):
        """returns a reference to the node storing the first element
        in the list"""

        def last_node(self):
        """returns a reference to the node storing the last element
        in the list"""

        def add_after(self, node, data):
        """adds data to the list, after the element stored in node.
        returns a reference to the new node (containing data)"""

        def add_first(self, data):
        """adds data as the first element of the list"""

        def add_last(self, data):
        """adds data as the last element of the list"""

        def add_before(self, node, data):
        """adds data to the list, before the element stored in node.
        returns a reference to the new node (containing data)"""

        def delete_node(self, node):
        """removes node from the list, and returns the data stored
        in it"""

        def delete_first(self):
        """removes the first element from the list, and returns its
        value"""

        def delete_last(self):
        """removes the last element from the list, and returns its
        value"""

        def __iter__(self):
        """an iterator that allows iteration over the elements of the
        list from start to end"""

        def __repr__(self):
        """returns a string representation of the list, showing data
        values separated by <--> """
```

```
class LinkedBinaryTree:
    class Node:
        def __init__(self, data, left=None, right=None,
parent=None):
            """Initializes a Node with:
            - data: element value
            - left: left child reference
            - right: right child reference
            - parent: parent reference"""

    def __init__(self, root=None):
         """Initializes a LinkedBinaryTree with:
        - root: root node reference
        - size: number of nodes"""

    def __len__(self):
        """Returns the number of nodes in the tree."""

    def is_empty(self):
       """Returns True if and only if the tree is
       empty."""

    def subtree_count(self, curr_root):
       """Returns number of nodes in the subtree rooted
       at curr_root."""

    def preorder(self):
       """Generator to iterate over all nodes in
       preorder."""

    def subtree_preorder(self, curr_root):
       """Generator to iterate subtree rooted at
       curr_root in preorder."""

    def postorder(self):
       """Generator to iterate over all nodes in
       postorder."""

    def subtree_postorder(self, curr_root):
       """Generator to iterate subtree rooted at
       curr_root in postorder."""

    def inorder(self):
       """Generator to iterate over all nodes in
       inorder."""

    def subtree_inorder(self, curr_root):
       """Generator to iterate subtree rooted at
       curr_root in inorder."""



    def __iter__(self):
       """Generator to iterate data level by level, left
       to right."""
```