Last Name: Annapareddy First Name: Manoj Sai

NetId: mannapareddy@arizona.edu

TA DR

CSc 120: Introduction to Computer Programming II

Fall 2024

Midterm 2 : Friday, Nov. 15, 2024

Time: 50 minutes

In order to give all students the same amount of time to do the exam, please do not open this exam until you are asked to do so. When told to begin, double-check that your name is at the top of this page and that your exam has all **11** pages.

IMPORTANT: You may not refer to any books, notes, or reference materials for this midterm.

Problem	Description	Earned	Max
1	Short Answer		8
2	Stacks & Queues		16
3	Recursion		20
4	Linked Lists		20
5	Tree Basics		15
6	Binary Search Trees		16
7	Traversals to Tree		5
TOTAL	**Total Points**		**100**

1. **Short Answer.** [1 point each]

(a) A binary tree can have multiple root nodes. (True or False?)

False

(b) In Python, lists can be used as dictionary keys. (True or False?)

true

(c) The `is` operator checks for object identity in Python. (True or False?)

true

(d) In object-oriented programming, an object is an instance of a class. (True or False?)

True

Note: Answer (e) through (h) with one word.

(e) Which tree traversal method visits nodes in ascending order in a Binary Search Tree?

inorder

(f) Name a built-in Python data type that cannot be modified after it is created.

tuple

(g) In a binary tree, how many child nodes can a node have at most?

2

(h) In object-oriented programming, what is the name of the first argument in instance methods?

Self

2. **Stacks and Queues.**

a) [4 points] What do the following queue operations do?

ENQUEUE: This function inserts a value to 0^{th} index element in the queue

DEQUEUE: This function removes the last or -1^{th} index in the queue

b) [2 points] Specify whether a stack or a queue would be the appropriate data structure for the problem below:

i. Simulating a printer's job sequence: Queue

ii. Reversing the order of items in a list: Stack

c) [4 points] For this problem, assume the following implementation of a stack.

```
class Stack:
    def __init__(self):
        self._items = []          # uses a Python list

    def push(self, item):
        self._items.append(item)

    def pop(self):
        return self._items.pop()  # pops from the end of the list

    def __str__(self):
        return str(self._items)   # prints the underlying list
```

Assume a `Stack` is created by the statement below:
`st = Stack()`

Write what `print(st)` would output *after* each of the statements below:

`st.push(3)` [3]

`st.push(10)` [3, 10]

`st.push(5)` [3, 10, 5]

`st.pop()` [3, 10]

d) [6 points] Implement a `Queue` class that conforms to the usual queue definition.

Requirements:

- use a ***LinkedList*** in your implementation
- The front of the queue is the first element of the linked list, and the rear is the last element.
- The LinkedList and Node classes have the following methods defined that you may use in your implementation:

  `Node(value)` – creates a node and sets the `_value` attribute to `value`

  `LinkedList()` – creates and returns an empty linked list

  `add(self, node)` - adds a node to the front of the linked list

  `append(self, node)` - adds node to the end of the linked list

  `remove_first(self)` - removes and returns the first node in the linked list (for queues, this will be used for dequeue

  `is_empty(self)` - returns True if the linked list is empty, and False otherwise .

```
class Queue:
  def __init__(self):
     Self._items = LinkedList()

  def enqueue(self, item):
     temp = Node(item)
     Self._items.add(temp)

  def dequeue(self):
     Cur = Self._items._head
     While Cur._next._next is not None:
           Cur = Cur._next
     Cur._next = None
  def is_empty(self):
     return Self._items.is_empty()
```

3. **Recursion.** For the problems below, you **must use recursion**. Your solution may not have loop constructs or list comprehensions.

a) [10 points] Write a ***recursive*** function `ends_with_vowel(alist)` that takes a list of strings `alist` and returns a list of the strings in `alist` that end with a vowel. A vowel is defined as one of the characters in the string `"aeiou"`. For example:

```
ends_with_vowel(["to", "lake", "area", "cat", "ever"])
```

Should return the list:

```
["to", "lake", "area"]
```

Assume that the strings in `alist` are all lowercase letters.

Note: You may **not** use a helper function. You may use the **`in`** operator.

```
def ends_with_vowel(alist):
    if alist == []:
        return []
    else:
        if alist[0][-1] in "aeiou":
            return [alist[0]] + ends_with_vowel(alist[1:])
        return ends_with_vowel(alist[1:])
```

b) [10 points] Write a ***recursive*** function `sum_byindex(alist)` that takes a list `alist` of integers and returns the sum of each element of `alist` multiplied by the index of the integer in `alist`. For example,

```
sum_byindex([2, 8, 5, 7])
```

returns the list

```
(0 + 8 + 10 + 21) = 39
```

Note: You **are allowed** to use a helper function. (Using a helper function will make this easier.)

```
def sum_byindex(alist):
    return helper_sum_byindex(alist, 0):

def helper_sum_by_index(alist, num):
    if alist == None:
        return 0
    else:
        temp = num
        num = num + 1
        return (atist[0] * temp) + helpe_sum_by_index
                                        (alist[1:], num)
```

4. Linked Lists.

For problems a) and b) below, use the following implementations of `Node` and `LinkedList`.

`class LinkedList:`
`    def __init__(self):`
`        self._head = None`
 	
`    def add(self,new):`
`        new._next =`
`self._head`
`        self._head = new`
	`class Node:`
	`    def __init__(self,value):`
	`        self._value = value`
	`        self._next = None`

Note: You may access the attributes directly without getter and setter methods.

a) [10 points] Write a method `sum_elements_at_even_pos(self)` for the `LinkedList` class that sums the elements at even positions in the list. An empty list should return `0`. Linked list index positions start at 0, just as they do for built-in Python lists.

Use an iterative solution.

```
def Sum_elements_at_even_pos(self):
    return helper_elm_even(0)

def helper_elm_even(self, n):
    if self._head is None:
        return 0
    cur = self._head
    else:
        if n % 2 == 0:
            temp = cur._value
            return temp + helper_elm_even(n+1)
        n += 1
        return helper_elm_even(n)
```

b) [10 points] Write a method `remove_last_two(self)` for the `LinkedList` class that takes a linked list as an argument and removes the last two elements of the linked list. The `LinkedList` object is modified and the method returns `None`. Return `None` if the list is empty.

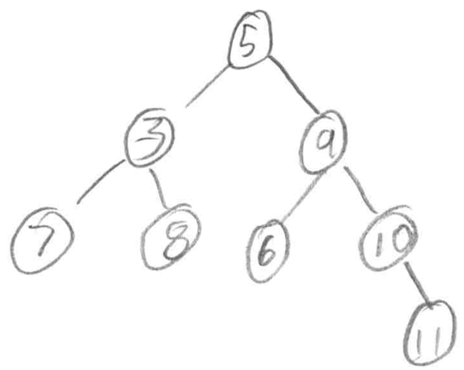
Use an iterative solution.

```
def remove_last_two(self):
    if self._head is None:
        return None
    else:
        cur = self._head
        prev = cur
        cur = cur._next
        while cur._next._next is not None:
            prev = cur
            cur = cur._next
        prev._next = None
```

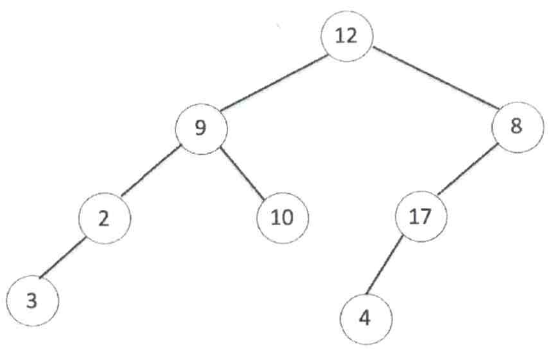
5. Tree Basics

a) [2 point] Can a queue be used to implement a breadth-first search of a tree?
Answer yes or no: no

b) [5 point] Draw a **binary tree** with eight nodes, four of which are leaves. All node values must be integers (you can choose any integers).



Use the tree below to answer the following questions:



c) [2 point] How many interior nodes are there in the tree?

5

d) [2 points] Write the inorder traversal:

3, 2, 9, 10, 12, 4, 17, 8

e) [2 points] Write the preorder traversal:

12, 9, 2, 3, 10, 8, 17, 4

f) [2 points] Write the breadth first traversal:

12, 9, 8, 2, 10, 17, 3, 4

6. Binary Search Trees. For a) and b), use the following the definition of a binary search tree class:

```
class BinarySearchTree:
    def __init__(self, value):
        self._value = value
        self._left = None
        self._right = None
```

Note: You may access the attributes directly or use the getter methods `value()`, `left()`, and `right()`.

(a) [8 points] Write a **recursive function** `get_smallest(tree)` that takes a **binary search tree** `tree` as an argument and returns the smallest value in the tree. If the tree is empty, return `None`. You may assume that the `_value` attribute is always an integer.

```
def get_smallest(tree):
    if tree is None:
        return None
    else
        if tree._left == None:
            return tree._value
        get_smallest(tree._left)
```

(b) [8 points] Write **a recursive function** `pre_order(tree)` that takes a binary tree and returns a Python *list* of the node values in the order of the pre-order traversal.

```
def pre_order(tree):
    if tree is None:
        return
    else:
        return [tree._value] + pre_order(tree._left)
            + pre_order(tree._right)
```

7. **Traversals to Tree.**
[5 points] Given the preorder and inorder traversals below, draw the resulting tree.

Preorder: 10, 15, 7, 3, 8, 5, 12

Inorder: 7, 15, 10, 3, 8, 5, 12