EXAM PREPARATION SECTION 7

OBJECT-ORIENTED PROGRAMMING TREES AND LINKED LISTS

April 3 to April 5, 2018

1 Object-Oriented Programming Trees

1. **Linky Paths** Implement `linky_paths` which takes in a Tree `t` and modifies each label to be the path from that node to the root.

```
def linky_paths(t):
    """
    >>> t = Tree(1, [Tree(2)])
    >>> linky_paths(t)
    >>> t
    Tree(Link(1), [Tree(Link(2, Link(1))]
    """
        def helper(t, path_so_far):

                t.label = ___________________________________

                for ________________________________________:

                    ________________________________________


        helper(___________________, ___________________)
```

2. **Find File Path** Implement `find_file_path` which takes in a Tree `t` and a string `file_str` and returns the full path of a file that we search for if the file exists. If the file does not exist, then return `None`.

```
def find_file_path(t, file_str):
    """
    >>> t = Tree('data', [Tree('comm', [Tree('dummy.py')]),
           Tree('ecc',
[Tree('hello.py'), Tree('file.py')]), Tree('file2.py')])
    >>> find_file_path(t, 'file2.py')
    '/data/file2.py'
    >>> find_file_path(t, 'dummy.py')
    '/data/comm/dummy.py'
    >>> find_file_path(t, 'hello.py')
    '/data/ecc/hello.py'
    >>> find_file_path(t, 'file.py')
    '/data/ecc/file.py'
    """"
    def helper(t, file_str, path_so_far):

        if __________________________________________________:

            return ___________________________________________

        elif t.is_leaf():

            return

        for _________________________________________________:


            result = _________________________________________

            if _______________________________________________

                return result

    return ______________________________________________
```

2 Linked Lists

1. **Convert to String** Implement `convert_to_string` which takes in a Linked List `link` and coverts the Linked List to a file path.

```
def convert_to_string(link):
        """
        >>> link = Link( d a t a  , Link( f i l e 2 . p y  ))
        >>> convert_to_string(link)
        '/data/file2.py'
        """
        if ____________________________________________:

                return ________________________________

        return ________________________________________
```

2. **All Paths Linked** Implement `all_paths_linked` which takes in a Tree `t` and returns a list of all paths from root to leaf in a tree with one catch – each path is represented as a linked list.

```
def all_paths_linked(t):
        """
        >>> t1 = Tree(1, [Tree(2), Tree(3)])
        >>> t2 = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)])
           ])
        >>> all_paths(t1)
        [Link(1, Link(2)), Link(1, Link(3))]
        >>> all_paths(t2)
        [Link(1, Link(2)), Link(1, Link(3, Link(4))), Link(1,
           Link(3, Link(5)))]
        """

        if ________________________________:

                return ______________________________________

        result = []
        for branch in t.branches:

                result = _____________________________________

        return result
```

3. **Find File Path 2** Implement `find_file_path` which takes in a Tree `t` and a string `file_str` and returns the full path of a file that we search for if the file exists. If the file does not exist, then return `None`.
   For this question, use the definition of `all_paths_linked` and `convert_to_string`

```
def find_file_path2(t, file_str):
    """
    >>> t = Tree('data', [Tree('comm', [Tree('dummy.py')]),
       Tree('ecc',
[Tree('hello.py'), Tree('file.py')]), Tree('file2.py')])
    >>> find_file_path2(t, 'file2.py')
    '/data/file2.py'
    >>> find_file_path2(t, 'dummy.py')
    '/data/comm/dummy.py'
    >>> find_file_path2(t, 'hello.py')
    '/data/ecc/hello.py'
    >>> find_file_path2(t, 'file.py')
    '/data/ecc/file.py'
    """
    for link in ___________________________________________:

            original = ____________________________________

            while _________________________________________:

                  if ________________________________:

                        return _________________________

        ___________________________________________
```

4. **Skip** Implement `skip` which takes in a Linked List `lnk` and an integer `n` which is great than 1 and mutates `lnk` such that every `nth` element is skipped.

```
def skip(lnk, n):
    """
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5, Link(6))
       ))))
    >>> skip(lnk, 2)
    >>> lnk
    Link(1, Link(3, Link(5)))
    >>> lnk2 = Link(1, Link(2, Link(3, Link(4, Link(5, Link(6)
       )))))
    >>> skip(lnk2, 4)
    >>> lnk2
    Link(1, Link(2, Link(3, Link(5, Link(6)))))
    """
    ____________________________________________________________

    def skipper(lst):

        ________________________________________________________

        count += 1

        if ___________________________________________________:

            return

        elif _________________________________________________:

            lst.rest = ________________________________________

            count = ___________________________________________

        ________________________________________________________

    ____________________________________________________________
```