CSC 120 **ICA-42**

Work with your neighbor. (This will be graded for participation only.)

1. Write a function `next_words(w1, words_list)` that takes a word, `w1`, and a list of valid words, `words_list`, and returns a list of words that differ from `w1` by one letter. Add only valid words that appears in `words_list`.

   ANS:
   In pseudocode:

```
For each position i in w1,
for each letter let in the alphabet,
     create a new word by changing the letter at position i to let
     if the new word is in the list of valid words
              add it to the set of words*
*unless it's been seen already
```

```
def next_words(w1, words_list):
    new_words = []
    for i in range(len(w1)):
        for char in "abcdefghijklmnopqrstuvwxyz":
            new_word = w1[:i] + char + w1[i+1:]
            if new_word in words_list:
                new_words.append(new_word)
    return new_words
```

2. Write a function `dist(w1, w2)` that returns the number of positions where words `w1` and `w2` differ.
   Requirements:
   - Use an assert to verify the lengths of `w1` and `w2` are the same.
   - Use a list comprehension in your function.

   ANS:

```
def dist(w1, w2):
      assert len(w1) == len(w2)
      diffs = [i for i in range(len(w1)) if w1[i] != w2[i]]
      return len(diffs)
```

Final exam review (ADT)

3. **Recursion**. Write a recursive function `count_occurrences(alist, value)` that returns the number of the occurrences of `value` in `alist`.

ANS:

```
def count_occurrences(alist, value):
    if len(alist) == 0:
        return 0
    if alist[0] == value:
        return 1 + count_occurrences(alist[1:], value)
    else:
        return count_occurrences(alist[1:], value)
```

4. **LinkedLists**. Write a method `insert_after_pos(self, node, pos)` for the `LinkedList` class that adds the node `new` after position `pos`. Node positions begin at 0, i.e., the first node in the list has position 0. You can assume that the list will **NOT** be empty and that `pos >= 0`. If `pos` is greater than the length of the list, add `new` to the end of the list.

ANS:

```
    #look-ahead method
    def insert_after_pos(self, new, pos):
        index = 0
        curr = self._head
        # pos might be greater than length of the list
        # so end loop if curr is the last element
        while curr._next != None and index != pos:
            index += 1
            curr = curr._next
        # make sure to keep the rest of the list
        # (if there are more nodes after curr)
        new._next = curr._next
        # add curr after next
        curr._next = new

   #alternate little brother method
    def insert_after_pos(self, new, pos):
        #Note: the list will not be empty
        i = 0
        current = self._head
        prev = current
        while current != None and pos > i:
            prev = current
            current = current._next
            i += 1
        if pos == i:
            new._next = current._next
            current._next = new
        # we fell off the end of the list
        elif pos > i:
            prev._next = new
```