# CS 1110 Review: Lists, Sequences, Loop Invariants, Sequence Algorithms

This worksheet contains various recursion problems for you to practice. We will go over only some problems during the final review session and will let you take this worksheet home to practice more.

## List and sequences

```
def find_in_list(lst, v):
    """ Returns: the position of the first occurrence of v in lst or -1
    if not found
    Parameters:
    lst - a possibly empty list
    v   - a value that may or may not be in the list"""
```

```
def sum_nums(s):
    """ Returns a string representing the sum of the numbers separated by
    spaces in the string s
    Parameter: s is a string with spaces and/or digits"""
```

```
def transpose(x):
    """ Returns: a nested list representing the transposed matrix x.

    Example: transpose([[1,2,3],[4,5,6]]) returns [[1,4],[2,5],[3,6]]

    Parameter: x - a nested list representing a rectangular matrix (the
    length of each row is the same)"""
```

```
def creditsToClasses(classes):
    """Returns: a dict with number of credits (ints) as keys and the
    corresponding classes (a list of str) as values. The order of the
    classes within the lists does NOT matter.

    Example: given classes {'CS 1110': 4,'CS 7090': 1, 'CS 1112': 4},
    you would return {4: ['CS 1110', 'CS 1112'], 1: ['CS 7090']}

    Parameter: classes is a dict with the keys being course names as
    strs (e.g. 'CS 1110') and the values being the number of credits as
    ints (e.g., classes['CS 1110'] = 4) """
```

The following function is supposed to take in a list `lst` and a value `v` that occurs more than once in `lst`, and return the index of the second occurrence of v. However, this is not what happens. Instead, the function always returns the index of the first occurrence of `v`. For example, the output of `secondInd([1,2,3,1], 1)` is equal to `0`. What is the issue?

```
def secondInd(lst,v):
    """ Returns: the index of the second occurence of v

    Given a list lst and value v that occurs more than once in lst,
    return the index of the second position where v occurs in lst.

    Parameter: v appears more than once in lst """

    seen_once = False
    for i in lst:
        if i == v:
            if not seen_once:
                seen_once = True
            else:
                return lst.index(v)
```

The following function is supposed to remove all values of `lst` that are even. However, the function does not behave as expected. Why did this happen?

```
def removeEvens(lst):
    """ Given a list lst, remove all even elements from lst in-place
    (does not return)

    Parameter: lst is a list of ints """

    for i in lst:
        if i%2 == 0:
            lst.remove(i)

>>> a = [1,1,2,2,2,3,3,4,4,4]
>>> removeEvens(a)
>>> a
[1, 1, 2, 3, 3, 4]
```

## Loop invariants and sequence algorithms

Draw boxes for the preconditions, postconditions, and invariant for the function smallest_index. Then, write the actual code for the loops. This loop should take a list of ints `s` and returns the index `x` of the smallest int (`s[x]`). `k` has been filled out for you.

Pre: s[0..k] is ???

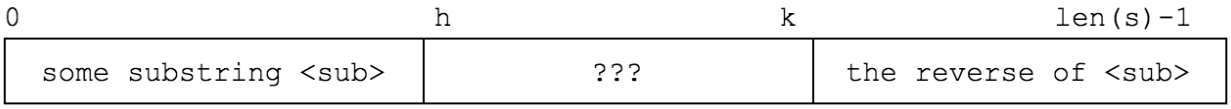
Invariant: s[x] is smallest number in s[0..m], s[m+1..k] is ???

Post: s[x] is smallest number of s[0..k]

```
def smallest_index(s):
    """ Returns: an int representing the index of the smallest
    number in s.
    Parameter: s is a non-empty string.
    """
    x =
    m =
    k = len(s) - 1
```

A palindrome is a sequence of characters which reads the same backward or forward. For example, madamimadam is a palindrome. Given the below function and invariant, write out the invariant in range notation and fill in the missing lines of the function.

invariant:




Invariant in range notation:

```
def is_palindrome(s):
    """returns True if the string is a palindrome, false otherwise
    Parameter: s is a string"""
    #initialize loop variable here




    #write your while loop below
```