CSC 120 **ICA-41**

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

1. Here is the example use of filter that filters out the words that start with a vowel:

```
list(filter(lambda w: w[0] in "aeiou", alist)
```

For comparison, write this using a list comprehension below:

ANS:
```
[ elem for elem in alist if elem[0] in "aeiou" ]
```

2. Iterators) Below is the example user-defined iterator `Reverse` discussed in lecture:

```
class Reverse:
    def __init__(self, data):
        self._data = data
        # start the index at the end of the list
        # next() will decrement it before use
        self._index = len(data)

    def __iter__(self):
        return self

    def __next__(self):
        if self._index == 0:
            raise StopIteration
        self._index = self._index - 1
        return self._data[self._index]
```

Using this as a template, write an iterator called `EveryOther` that produces every other element of a list. The first element produced is the element at index 0, the second element produced is the element at index 2, and so on.

ANS:

```
class EveryOther:

    def __init__(self, data):
        self._data = data
        self._index = 0

    def __iter__(self):
        return self
    def __next__(self):
        if self._index == len(self._data):
            raise StopIteration
        index = self._index
        self._index += 2
        return self._data[index]
```

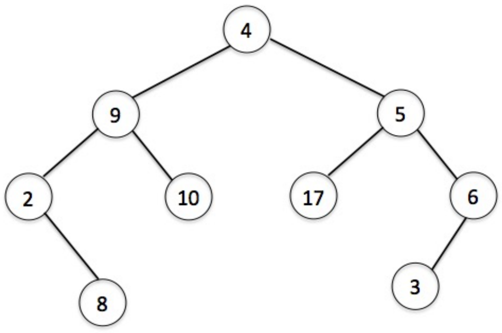
3. (Generators) Now write a generator called `every_other()` that is a generator version of `EveryOther`.

   ANS:

```
def every_other(data):
    for i in range(0,len(data), 2):
        yield data[i]
```

Final Exam Review

4. (Trees) Use the tree below to answer the following questions:



a) What is the height of the tree?
   ANS: 3

b) Write the inorder traversal:
   ANS:
   2, 8, 9, 10, 4, 17, 5, 3, 6

c) Write the postorder traversal:
   ANS:
   8, 2, 10, 9, 17, 3, 6, 5, 4

5. **(ADT)** For this problem, you are to implement an abstract data type named `CharStack` that represents a stack of characters (strings of length 1).

   Implement the following **methods** for the `CharStack` class:
   - `__init__`: initializes an empty stack
   - `push`: puts its argument, a character, on the top of the stack; returns `None`
   - `pop`: removes the top character from the stack and returns it; returns `None` if the stack is empty
   - `swap`: swaps the top two characters of the stack; always returns `None`; has no effect if the stack has less than two elements
   - `is_empty`: returns `True` if the stack is empty and `False` otherwise

Restriction: The `CharStack` class has only one **attribute** which is of type string.

Here is an example of usage:

```
  >>> cs = CharStack()
  >>> cs.push('p')
  >>> cs.push('a')
  >>> cs.push('n')
  >>> cs.push('s')
  >>> print(cs)
  snap
  >>> cs.swap()
  >>> cs.pop()
  'n'
  >>> print(cs)
  sap
  >>> cs.is_empty()
  False
>>>
```

ANS:

```
class Stack:
    def __init__(self):
        self._items = ""

    def push(self, item):
        self._items = item + self._items

    def pop(self):
        top = self._items[0]
        self._items = self._items[1:]
        return top
```

```
def swap(self):
    self._items = self._items[1] + self._items[0] + \
                                  self._items[2:]

def is_empty(self):
    return self._items == ""

def __str__(self):
    return str(self._items)
```