CSCA48 Term Test 1 Seminar

Brian Chen and Joshua Concon

January 30, 2017

Implementation

```
class LLNode(object):
    """A Node in a singly-linked list"""

    def __init__(self, data, link = None):
        """(LLNode, object) -> NoneType
        Create a new node to hold data
        """
        self.data = data
        self.link = link

class DLLNode(object):
    """A Node in a doubly-linked list"""

    def __init__(self, data, nxt = None, prev = None):
        """(DLLNode, object) -> NoneType
        Create a new node to hold data
        """
        self.data = data
        self.next = nxt
        self.prev = prev

class EmptyStackError(Exception):
    pass

class Stack():
    """
    A class to represent a Stack.

    Raises EmptyStackError when popping from an empty stack.
    """
    def __init__(self: 'Stack') -> None:
        self._contents = []

    def __str__(self: 'Stack') -> str:
        s = ""
```

```
    for item in self._contents:
        s = s + str(item) + ", "
    return s[:-2]

def push(self: 'Stack', item: 'object') -> None:
    self._contents.append(item)

def pop(self: 'Stack') -> object:
    if self.isempty():
        raise EmptyStackError
    return self._contents.pop()

def isempty(self: 'Stack') -> bool:
    return (len(self._contents) == 0)
```

Question 1

Implement a Queue using only the Stack ADT. Assume that module stack.py defines a class Stack that provides the usual methods: `push(item)`, `pop()`.

```
from stack import *

class QueueUsingStack():
    """A Queue implemented with Stack Objects"""

    def __init__(self):









    def push(self, obj):









    def pop(self):
```

Question 2

Write the body of the function below so that it satisfies its docstring.

```
def reverse(head):
    """(LLNode) -> LLNode

    Given the head pointer to a linked list,
    reverse the linked list and return a pointer
    to the head of the reversed list.)
    """
```

Question 3

We are given a task by a customer. He wants to have a data structure that can be added to and removed from on both sides. If the data structure is empty, he would like to be notified by an error. Implement this ADT and write an Representative Invariant for it.

```
class Deque():
    '''A double ended queue'''

    def __init__(self: 'Deque') -> None:
        """Representation invariant:




        """




    def enqueue_left(self, item):









    def enqueue_left(self, item):









    def dequeue_left(self):
```

```
def dequeue_right(self):
```

```
def is_empty(self: 'Deque'):
```

```
def get_deque(self: 'Deque'):
```

Question 4

Finish the following function

```
def remove(head, item):
    """(DLLNode, Object) -> DLLNode

    Removes the first DLLNode with the data 'item' from
    the doubly-linked list and returns the list.

    If item is not in the list at head, just return
    the list.
    """
```

Answers

Question 1

```
from stack import *

class QueueUsingStack():
    """A Queue implemented with Stack Objects"""

    def __init__(self):
        self._inbox = Stack()
        self._outbox = Stack()

    def push(self, obj):
        self._inbox.push(obj)

    def pop(self):
        if(self._outbox.isempty()):
            while(not(self._inbox.isempty())):
                self._outbox.push(self._inbox.pop())
        return self._outbox.pop()
```

Question 2

```
def reverse(head):
     """(LLNode) -> LLNode

     Given the head pointer to a linked list,
     reverse the linked list and return a pointer
     to the head of the reversed list.)
     """
    current = head
    previous = None
    while current:
        next = current.next
        current.next = previous
        previous = current
        current = next
    head = previous
    return head
```

Question 3

```
class Deque():
    '''A double ended queue'''

    def __init__(self: 'Deque') -> None:
        # Representation invariant:
        # _contents is a list of object.
        # _contents[:] are the objects in the dequeue.
        # if i < j, i >=0, j < len(_contents), then
        #    _contents[i] is to the left of _contents[j] in
            the deque
        #    _contents[j] is to the right of _contents[i] in
             the deque

        self._contents = []

    def enqueue_left(self, item):
        self._contents = [item] + self._contents

    def enqueue_left(self, item):
        self._contents = self._contents + [item]

    def dequeue_left(self):
        if(self.is_empty()):
            raise EmptyQueueError("Can't dequeue from an
                empty queue")
        else:
            item = self._contents[0]
            self._contents = self._contents[1:]
            return item

    def dequeue_right(self):
        if(self.is_empty()):
            raise EmptyQueueError("Can't dequeue from an
                empty queue")
        else:
            item = self._contents[-1]
            self._contents = self._contents[:-1]
            return item

    def is_empty(self: 'Deque'):
        return len(self._contents) == 0

    def get_deque(self: 'Deque'):
        return self._contents
```

Question 4

```
def remove(head, item):
    """(DLLNode, Object) -> DLLNode

    Removes the first DLLNode with the data 'item' from
    the doubly-linked list and returns the list.

    If item is not in the list at head, just return
    the list.
    """
    current = head
    removed = False
    while((current is not None) and (not removed)):
        if(current.cargo = item):
            removed = True
            # if current is not the head
            if(current.prev is not None):
                before = current.prev
                after = current.next
                before.next = after
                after.prev = before
            # if current is the head
            else:
                head = current.next
                head.prev = None
        current = current.next
```