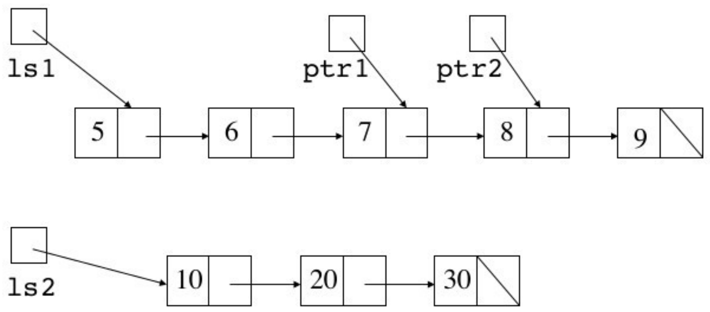
ECS529U Algorithms and Data Structures **Week 8 Lab sheet**

This lab gets you to work with linked lists.

Marks (max 5): Questions 1-2: 1.5 each | Questions 3-4: 0.5 each | Question 5: 1

Question 1
You are asked to solve the following pointer game. Start with this initial configuration:



Perform the following groups of operations and draw the new final lists in each case:

a)
```
ls1.next = ls1.next.next
```

b)
```
ls2.next = ls1.next
ls2.next.data = 42
```

c)
```
while ptr1.next != None:
    ptr1.data = 42
    ptr1 = ptr1.next
ptr1.data = 24
```

d)
```
ptr1.next = ls2
ptr1.next.data = 40
ptr2 = ptr2.next
ls2 = Node(4,ls1)
ls1=ptr2
```

For the next 3 questions, you are asked to work on the Python implementation of the `LinkedList` class that we saw in the lecture (week 8). The code is in `lecture8.ipynb`.

Question 2

Add in `LinkedList` the following functions:

a) `def appendAll(self, A)`

that appends all elements of the array `A` in the linked list (the one represented by `self`). For example, if `ls` is [2,3,4,5] then `ls.appendAll([42,24])` changes `ls` to [2,3,4,5,42,24]

b) `def nullify(self)`

that removes all elements from the list. I.e. `ls.nullify()` changes `ls` to [ ].

c) `def merge(self, other)`

that merges the linked list with the linked list `other`. For example, if `ls` is [2,3,4,5] and `ls2` is [42,17], then:

```
ls.merge(ls2)         # changes ls to [2,3,4,5,42,17]
ls2.head.data = 24    # changes ls2 to [24,17] and ls to [2,3,4,5,24,17]
```

Question 3

Add a function

```
def sort(self)
```

that sorts the linked list using quicksort.

Hint: you can adapt the algorithm we saw in Lecture 3. Starting from the current list (`self`), let `pivot` be the data at the head of the list, and create two new linked lists: one called `smaller` (which contains all elements whose data is smaller than `pivot`), and one `other` (containing all remaining elements, apart from `pivot`). Then, call `smaller.sort()` and `other.sort()`, set the current list to `smaller`, then append `pivot` and merge with `other`.

Question 4

Add a function

```
def isCyclic(self)
```

that checks if the linked list contains a cycle, and returns `True` if it does (otherwise, `False`). For example, if `ls` is the linked list [2,3,4,5]:

```
print(ls.isCyclic())          # prints False
ls.head.next.next = ls.head   # makes ls cyclic: 2 -> 3 -> (head)
print(ls.isCyclic())          # prints True
```

Question 5

You are asked to write a class `DLinkedList` which implements doubly linked lists. A doubly linked list is one in which each node points both to the next and to the previous node in the list (see Lecture 8, slide 23).
Your implementation should keep track of both the head and the tail (i.e. the last node) of the doubly linked list. Here is the constructor you can use:

```
class DLinkedList:
    def __init__(self):
        self.head = None   # this is of type DNode
        self.tail = None   # this is of type DNode
        self.length = 0
```

The nodes of the linked list are taken from the class `DNode2` which we define as:

```
class DNode:
    def __init__(self, d, n, p):
        self.data = d
        self.next = n
        self.prev = p
```

where each node has a piece of data and two pointers: one to the next element in the list, and one to the previous one. You are asked to implement the following functions:

- `search(self,d)` for searching the value `d` in the doubly linked list
- `append(self,d)` for appending the value `d` at the tail of the doubly linked list
- `insert(self,i,d)` for inserting the value `d` in position `i` of the doubly linked list
- `remove(self,i)` for removing the `i`-th element of the doubly linked list

To help you debug your code, you can use the following function to print a linked list starting from its head, and starting from its tail:

```
def __str__(self):
    st = ""
    ptr = self.head
    while ptr != None:
        st = st + str(ptr.data)
        st = st+" -> "
        ptr = ptr.next
    st += "None, and reversed: "
    ptr = self.tail
    while ptr != None:
        st = st + str(ptr.data)
        st = st+" -> "
        ptr = ptr.prev
    return st+"None"
```