HW2

Problem 2.1

a) Queue

b) Stack

c) Queue

d) Stack

e) List

f) Stack

g) List

h) Stack

i) List

j) Queue

k) Queue

l) List

m) Queue

n) Stack

o) Queue

Problem 2.2

The two stacks can be implement by dividing the array into two halves and assign the halfes to two stacks. First stack use arr[0] to arr[$n/2$], Second stack use arr[$n/2+1$] to arr[n-1], where n is the size of arr[]. There will be overflow if push more than $n/2$ to one of the stack even if there is no element in another stack.

```
Function push1 (obj x) :
    if (L1 > n/2) then error
    else
        L1 ← L1 + 1
        arr[L1 - 1] ← x

Function push2 (obj x) :
    if (L2 > n/2) then error
    else
        L2 ← L2 + 1
        arr[n/2 + L2 - 1] ← x

Function pop1 ( ) : obj
    If (L1 == 0) then error
    else
        x ← arr[L1-1]
        L1 ← L1 - 1
        return x

Function pop2() : obj
    if (L2 == 0) then error
    else
        x ← arr[n/2+L2-1]
        L2 ← L2 - 1
    return x.

Function top1() : obj
    if (L1 == 0) then error
    else
    return arr[L1-1]

Function top2() : obj
    If (L2 == 0) then error
    else
    return [n/2+L2-1]

Function isEmpty1() : boolean
    if (L1 == 0) return ~~take~~ true
    else false

Function isEmpty2() : boolean
    if (L2 == 0) return true
    else false
```

Problem 2.3

```
while ! pezContainer.isEmpty()
        x ← pezContainer.pop()
        if x == yellowCandy then
                eat(x)
        else
                storage.push(x)

while ! storage.isEmpty()
        x ← storage.pop()
        pezContainer.push(x)
```

Problem 2.4

The implementation will have issue when the position of the first element and the position of the last element is equal. This will cause the queue to be uncertain whether it is empty or full.

To prevent, the queue should never store more than N-1 if the array size is N.

Problem 2.5

a) function insert (Node head, Key k)

next (head) ← newNode (next (head), k)

* insert required constant time since there is only single statement.

function find (key k, Node head)

current ← next (head)

while current ≠ null

~~[illegible]~~

if current(key) = k then return current (key)

current ← next (current)

return null

* find could required up to the time insert function was called.

function remove (key k, Node head)

current ← head

while next (current) ≠ null

if next(current (key)) = k then

tail ← next (current)

next (current) ← next (tail)

else current ← next (current)

* remove required the number of insert function made that was not deleted by remove function

Problem 2.6

$$\sum_{i=1}^{n} k_i = \sum_{i=1}^{n} \frac{c}{i} = 1$$

$$= c \sum_{i=1}^{n} \frac{1}{i}$$

$$= c H_n$$

H_n is the n-th harmonic number.

$$H_n \approx \ln n + \gamma$$

$$c H_n = 1$$

$$c \ln n + \gamma = 1$$

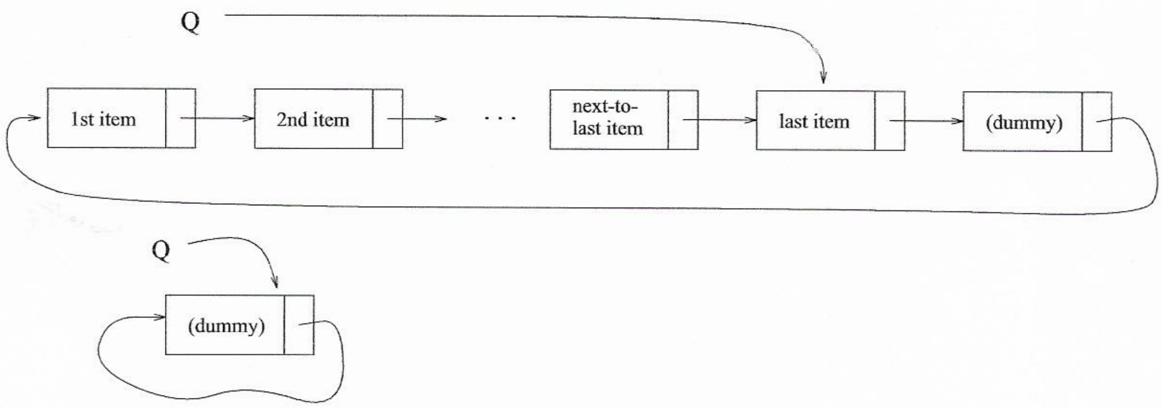
$$c = \frac{1}{\ln n + \gamma}$$

$$\sum_{i=1}^{n} i k_i = \sum_{i=1}^{n} i \frac{c}{i} = \sum_{i=1}^{n} c = c \cdot n = \frac{n}{\ln n + \gamma}$$

Problem 2.7 (15 Points)
Consider a circular list with a dummy node, depicted below, as one possible linked representation for the "queue" abstract data type. The queue on top contains several (at least 4) items, while the one below is empty.

(a) Give pseudo-code to implement `isEmpty()`, `enqueue()` and `dequeue()` in this representation.

(b) Compare this representation to a circular list *without* the dummy node (in terms of handling an empty or empty-to-be queue).



Problem 2.8 (15 Points)
Assume that a singly linked list is being used to implement a dictionary. The four keys in the dictionary are `I, M, P`, and `S`. `find` is performed 12 times, on the sequence `I,P,P,I,S,S,I,S,S,I,M,I`. How many comparisons are required for each search when the three methods listed below are used? For Move-to-Front and Transpose assume that the list initially contains the keys in the order `I, P, S, M` (from first to last), while for Best Static Ordering assume that all keys are already listed in decreasing frequency.

	I	P	P	I	S	S	I	S	S	I	M	I
Move-to-Front Strategy	1	2	1	2	3	1	2	2	1	2	4	2
Transpose Strategy	1	2	1	2	3	2	2	2	1	2	4	1
Best Static Ordering	1	3	3	1	2	2	1	2	2	1	4	1