**CS2023 - Data Structures and Algorithms**

**Take Home Assignment**

Week 5 - Basic Data Structures

Submission by Sajeev Kugarajah (210554M)

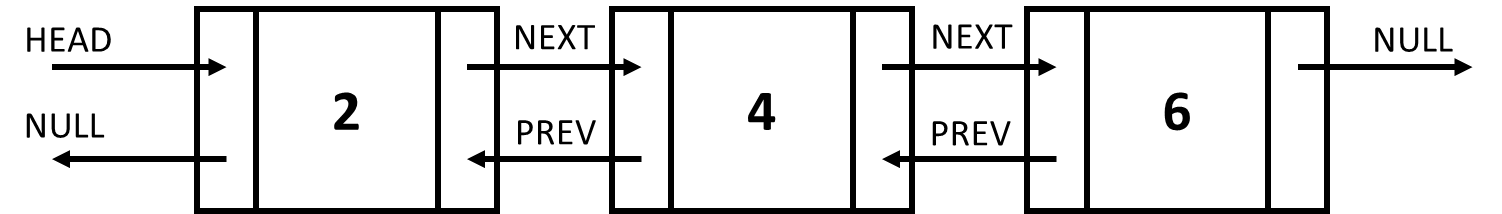
March, 2023

# Question 1

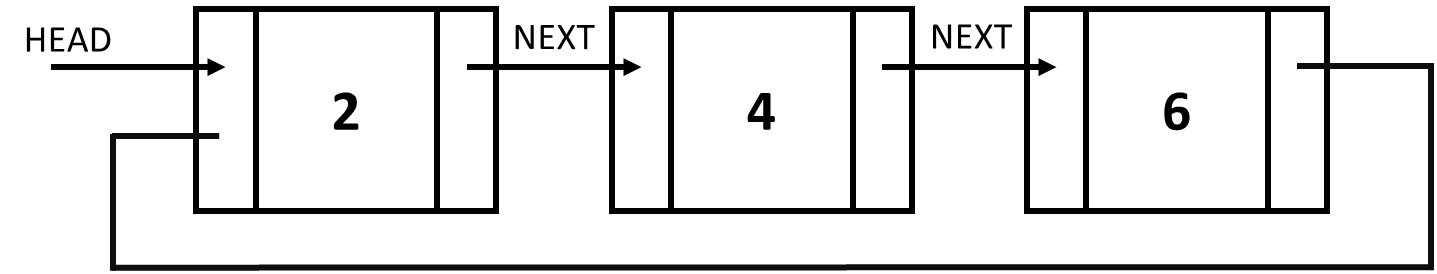
Explain briefly what a double linked list and a circular linked list is?

*Note: Please use diagrams in your explanations*

* Double linked list
  + it is a linked list where each node contains two pointers.
    - next 🡪 points to the next element’s address
    - previous 🡪 points to the previous element’s address
  + this allows efficient traversal in both directions.



* Circular linked list
  + unlike linked list, the nodes in circular linked list only have one pointer which points the next element’s address.
  + but the last element points to the first element in the list which makes it circular.
  + there’s no start or end in circular linked lists.



**Question 2**

Write pseudo codes for the operations of a single linked list.

Insert element at the front 🡪

Insert (list, value):

newNode <= new Node(value)

newNode.next <= list.head

list.head <= newNode

Insert element at a position 🡪

Insert\_at\_position (list, position, value):

newNode <= new Node(value)

if position = 0 then:

newNode.next <= list.head

list.head <= newNode

else:

current\_position <= 0

current\_node <= list.head

while current\_node is not NULL:

if current\_position + 1 <= position:

newNode.next <= current\_node.next

current\_node.next <= newNode

return

current\_node <= current\_node.next

current\_position <= current\_position + 1

print(“invalid position”)

Search element by value 🡪

search (list, value):

index <= 0

current\_node <= list.head

while current\_node is not NULL:

if current\_node.data = value:

return index

current\_node <= current\_node.next

index <= index + 1

return -1

Delete element by value 🡪

delete (list, value):

current\_node <= list.head

if list.head.data = value:

list.head <= list.head.next

return true

while current\_node.next is not NULL:

if current\_node.next.data = value:

current\_node.next <= current\_node.next.next

return true

return false

# Question 3

How can you implement a stack and a queue using a linked list?

*Note: Explain how you would do it and also write pseudo codes for all the operations.*

Implementing stack using linked list

stack is a LAST IN FIRST OUT data structure. since we are adding and removing from one end of the list we can maintain a single pointer. we can create a node called top and use new node as top each time we push an element to the stack.

push(list, value):

new\_node <= new Node(value)

new\_node.next <= list.head

list.head <= new\_node

pop(list):

if list.head is NULL:

return -1

current\_val <= list.head.data

list.head <= list.head.next

return current\_val

peak(list):

if list.head is NULL:

return -1

return list.head.data

is\_empty(list):

if list.head is NULL:

return true

else:

return false

implementing a queue using linked list

queue is a FIRST IN FIRST OUT data structure. we add elements to the queues from one side and remove elements from the other side. so we have to maintain two pointers to implement a queue using linked lists. to enque data we have to use the rear pointer and to deque the data we have to use the front pointer.

enqueue(list, value):

new\_node <= new Node(value)

if list.rear is NULL:

list.rear <= new\_node

list.front <= list.rear

else:

list.rear.next <= new\_node

list.resr <= new\_node

dequeue(list):

if list.front is NULL:

return -1

removed\_value <= list.front.data

list.front <= list.front.next

return removed\_value

front(list):

if list.front is NULL:

return -1

else:

return list.front.data

is\_empty(list):

if list.front is NULL:

return true

else:

return false