## Topics

1. Implement Node Class

class Node:

def \_\_init\_\_(self, data):

self.data = data

# Create nodes

node1 = Node(10)

node2 = Node(20)

node3 = Node(30)

# Link the nodes

node1.next = node2

node2.next = node3 self.next = None

1. Generics

public class Box<T> {

private T item;

public T getItem() {

return item;

}

public void setItem(T item) {

this.item = item;

}

}

Box<Integer> intBox = new Box<>();

intBox.setItem(10);

int item = intBox.getItem(); // Retrieves the item of type Integer

Box<String> stringBox = new Box<>();

stringBox.setItem("Hello");

String item = stringBox.getItem(); // Retrieves the item of type String

1. Implement SinglyLinkedList Class

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class SinglyLinkedList:

def \_\_init\_\_(self):

self.head = None

def is\_empty(self):

return self.head is None

def size(self):

count = 0

current = self.head

while current:

count += 1

current = current.next

return count

def first(self):

if self.is\_empty():

return None

return self.head.data

def last(self):

if self.is\_empty():

return None

current = self.head

while current.next:

current = current.next

return current.data

def add\_first(self, data):

new\_node = Node(data)

if self.is\_empty():

self.head = new\_node

else:

new\_node.next = self.head

self.head = new\_node

def add\_last(self, data):

new\_node = Node(data)

if self.is\_empty():

self.head = new\_node

else:

current = self.head

while current.next:

current = current.next

current.next = new\_node

def remove\_first(self):

if self.is\_empty():

return

self.head = self.head.next

def display(self):

current = self.head

while current:

print(current.data, end=" ")

current = current.next

print()

1. Implement Basic Methods of SinglyLinkedList

* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class SinglyLinkedList:

def \_\_init\_\_(self):

self.head = None

def is\_empty(self):

return self.head is None

def append(self, data):

new\_node = Node(data)

if self.is\_empty():

self.head = new\_node

else:

current = self.head

while current.next:

current = current.next

current.next = new\_node

def prepend(self, data):

new\_node = Node(data)

if self.is\_empty():

self.head = new\_node

else:

new\_node.next = self.head

self.head = new\_node

def delete(self, data):

if self.is\_empty():

return

if self.head.data == data:

self.head = self.head.next

return

current = self.head

while current.next:

if current.next.data == data:

current.next = current.next.next

return

current = current.next

def search(self, data):

current = self.head

while current:

if current.data == data:

return True

current = current.next

return False

def display(self):

current = self.head

while current:

print(current.data, end=" ")

current = current.next

print()

## Homework

1. develop an implementation of the equals method in the context of the SinglyLinkedList class.

class SinglyLinkedList:

def \_\_init\_\_(self):

self.head = None

# Other methods...

def rotate(self):

if self.head is None or self.head.next is None:

return

old\_head = self.head

self.head = old\_head.next

current = self.head

while current.next:

current = current.next

current.next = old\_head

old\_head.next = None

1. Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.

def find\_second\_to\_last\_node(head):

if head is None or head.next is None:

return None

prev = head

current = head

while current.next is not None:

prev = current

current = current.next

return prev

1. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.

class SinglyLinkedList:

def \_\_init\_\_(self):

self.head = None

# Other methods...

def size(self):

count = 0

current = self.head

while current:

count += 1

current = current.next

return count

1. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.

class SinglyLinkedList:

def \_\_init\_\_(self):

self.head = None

# Other methods...

def size(self):

count = 0

current = self.head

while current:

count += 1

current = current.next

return count

1. Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.

def concatenate\_lists(L, M):

if L is None:

return M

if M is None:

return L

L\_prime = L

current = L\_prime

while current.next:

current = current.next

current.next = M

return L\_prime

1. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

def reverse\_linked\_list(head):

if head is None or head.next is None:

return head

prev = None

current = head

while current is not None:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

return prev