香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

INTRODUCTION TO COMPUTER SCIENCE: PROGRAMMING METHODOLOGY

TUTORIAL 13

LINKED LIST

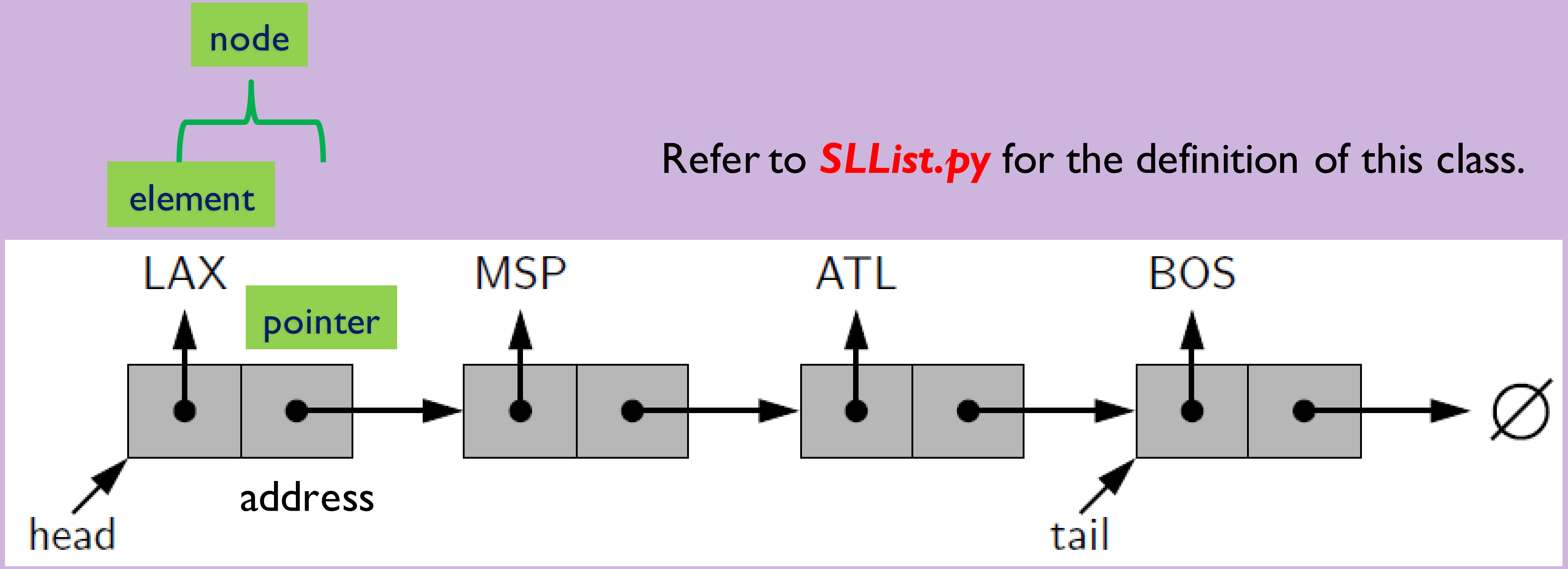
Florensia Widjaja

(122040013)

Singly Linked List

- Element: the value of a node.
- Pointer: pointing to the next node(by its address).
- Linked list stores elements in a determined sequence.

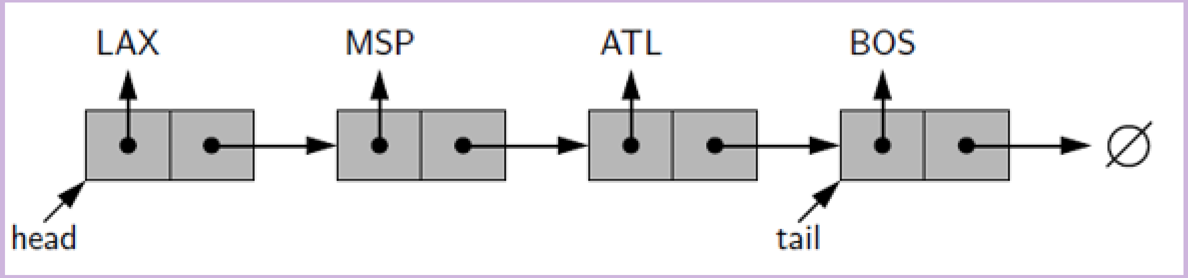
Refer to ***SLList.py*** for the definition of this class.



Advantage: save storing space.

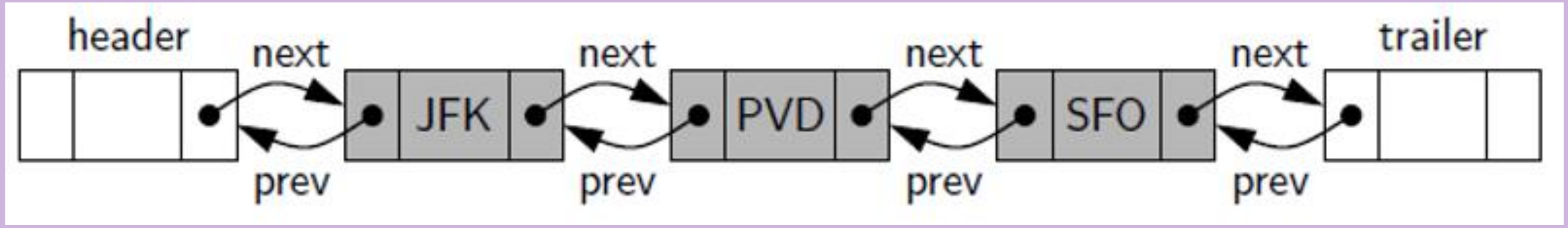
Three Types of Linked List

- ***Singly linked list:***



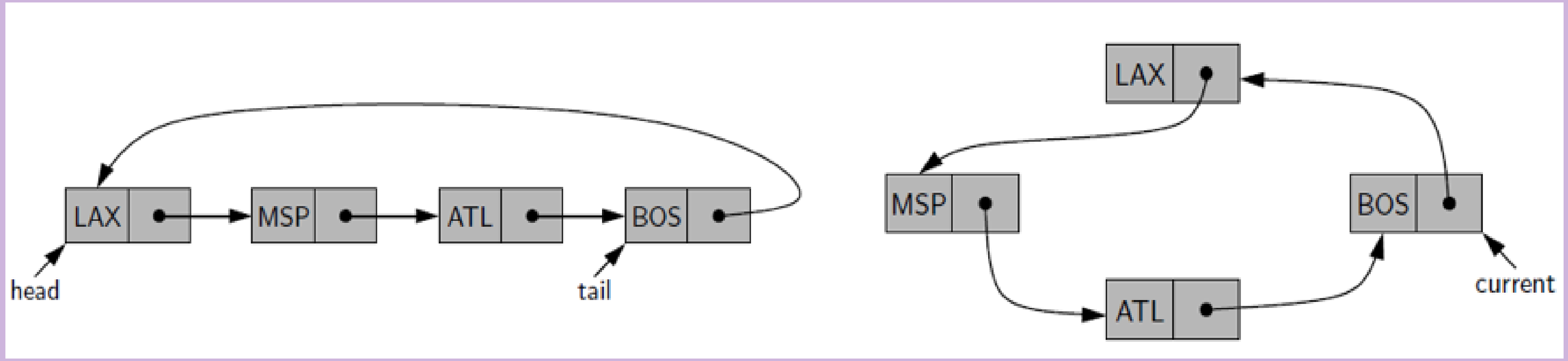
- ***Doubly linked list:***

  Refer to ***DLList.py*** for the definition of this class.



- ***Circularly linked list:***

  Refer to ***CLList.py*** for the definition of this class.



Q1: Concatenate Two Linked Lists

- Write a function to concatenate two singly linked lists **L** and **M**, given only references to the first node of each list, into a single linked list **L’** that contains all the nodes of L followed by all the nodes of M. (Comment: This function should be in the singly linked list class definition as a method.)

Q2: LinkedQueue

- Implement a queue class using linked list. You may fill the blanks ①②③④⑤ on the right. And on the left is the test program.

```
q=LinkedQueue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
q.enqueue(4)
print(q.dequeue())
print(q)
print(q.first())
print(q.end())
```

```
1
['2', '3', '4']
2
4
```

```
class Node:
    def __init__(self,e,node):
        self.element=e
        self.pointer=node

class LinkedQueue:

    def __init__(self):
        self.head=None
        self.tail=None
        self.size=0

    def __len__(self):
        return self.size

    def is_empty(self):
        return self.size==0
```

```
def first(self):
    if self.is_empty():
        print('Queue is empty.')
    else:
        #①

def end(self):
    if self.is_empty():
        print('Queue is empty.')
    else:
        #②

def dequeue(self):
    if self.is_empty():
        print('Queue is empty.')
    else:
        #③

def enqueue(self,e):
    #④

def __str__(self):
    #⑤
```

Q3: LinkedStack

➢ Implement a stack class using linked list. You may fill the blanks ①②③④ on the right. And on the left is the test program.

```python
s=LinkedStack()
s.push(1)
s.push(2)
s.push(3)
s.push(4)
print(s.pop())
print(s)
print(s.top())
```

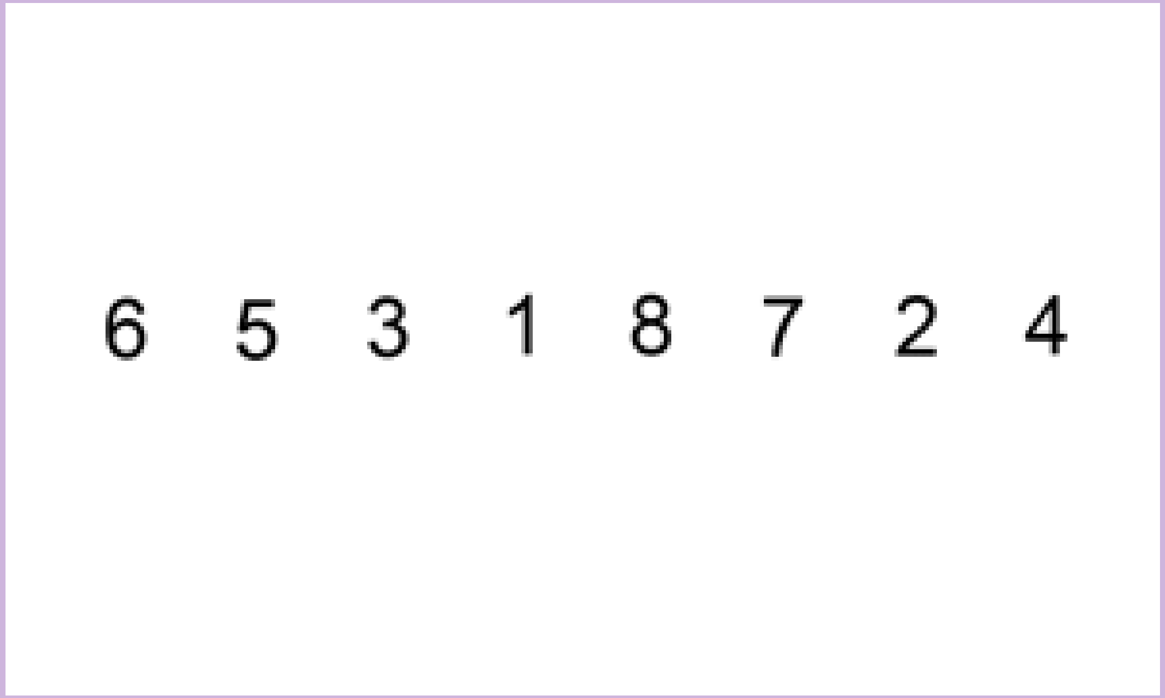
```
4
[1, 2, 3]
3
```

```python
class Node:

    def __init__(self,e,node):
        self.element=e
        self.pointer=node

class LinkedStack:

    def __init__(self):
        self.head=None
        self.size=0

    def __len__(self):
        return self.size

    def is_empty(self):
        return self.size==0
```

```python
def top(self):
    if self.is_empty():
        print('Stack is empty.')
    else:
        #①

def pop(self):
    if self.is_empty():
        print('Stack is empty.')
    else:
        #②

def push(self,e):
    #③

def __str__(self):
    #④
```

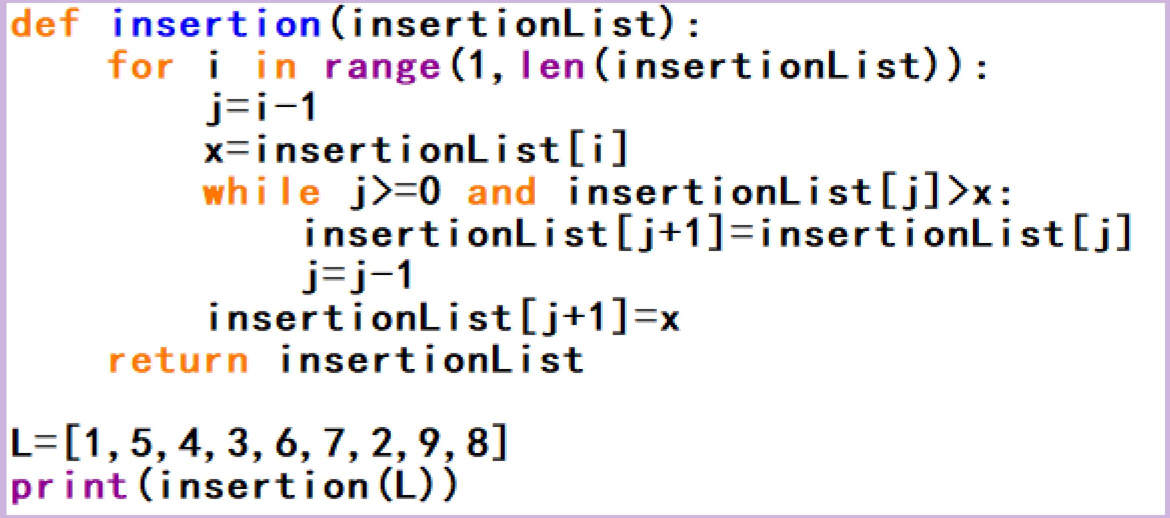
Q4: Insertion sort-I

- Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.
- Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

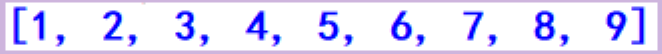


Q4: Insertion sort-I

- Below is the insertion sort algorithm implemented by standard list. Please use doubly linked list to implement that in a similar way.



```python
def insertion(insertionList):
    for i in range(1,len(insertionList)):
        j=i-1
        x=insertionList[i]
        while j>=0 and insertionList[j]>x:
            insertionList[j+1]=insertionList[j]
            j=j-1
        insertionList[j+1]=x
    return insertionList

L=[1,5,4,3,6,7,2,9,8]
print(insertion(L))
```



```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```