UCONN
UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

1

CSE-2050 – Data Structures and Object-Oriented Design

Recap

2

Announcements

- Updates in syllabus
  - Office hours
  - Schedule
    - Assignment due date extended
    - Exam 1 date announced

Weeks	Modules	Assignments Schedule
8/29 – 9/2	Mod 1 – Basic Python	
9/5 – 9/9	Mod 2 – Object-oriented Programming & testing	
9/12 – 9/16	Mod 3 – Running Time Analysis	Assignment 1 release
9/19 – 9/23	Mod 4 – Linear Data Structures	
9/27	**Assignment 1 Due**	
9/29	**Exam 1**	

Week 4 – 09/19 – 09/23 – **Lecture 1**

Hasan Baig

2

Quick Recap

- Learnt the identity to evaluate the sum of all K integers
- Asymptotic Analysis
  - Allow us to measure performance in terms of input size
- Big-O notation
  - Allow us to ignore lower order terms and constants

```python
def sum_clever(k):
    total = k*(k+1)//2
    return total

print(sum_clever(3))
print(sum_clever(6))
print(sum_clever(7))
print(sum_clever(100))
```


Activity # 6 (1/2)

Calculate the time complexity (Big-O) for the following codes

```python
#Practice question 1
def func(L):
    x = 0
    for i in L:
        for j in L:
            x += i * j
    return x
```

1, n, n, 3, 1

$1 + n(n \times 3) + 1 = 3n^2 + 2$

$\rightarrow = O(n^2)$

```python
# Practice question 2
a = 0
b = 0
for i in range(N):
    a = a + random()

for i in range(M):
    b= b + random()
```

1 ✓, 1 ✓, 3N ✓, 1, 3M ✓

$\rightarrow N+M \Rightarrow O(N+M)$

Activity # 6 (2/2)

Calculate the time complexity (Big-O) for the following codes

```
#Practice question 3
a = 0;
for i in range(N):
    for j in reversed(range(i,N)):
        a = a + i + j;
```

1

→ N

0 → 10

0 – 9

9 8 7 6 5 4 3 2 1 0

9 8 7 6 5 4 3 2 1

4 3 2

→ 1+2+3+… → N(N+1) → O(N²)

1+2+3

9+8+7+6…

```
# Practice question 4
i = 1
while i <= n:
    print(i)
    i *= 2
```


Types of functions

- Constant Functions → O(1)
- Logarithmic Functions → O(log n)
- Linear Functions → O(n)
- n Log n → O(nlogn)
- Quadratic Functions → O(n^2)
- Polynomial Functions → O(n^k) for some constant k.
- Exponential Functions → O(2^n)
- Factorial Functions → O(n!)

Module 4

Linear Data Structures

7

Abstract Data Types (ADT)

What is Data Structure?

It is a technique to *structure* data so that it can be utilized efficiently using:

- Some protocols or rules
- Implementation via programming

- ADT → *abstraction* of a data structure that provides only the *interface* to which the data must adhere

- Interface does not give any details about the implementation or programming language
  - What data we are dealing with
  - What operations can be performed on that data

8

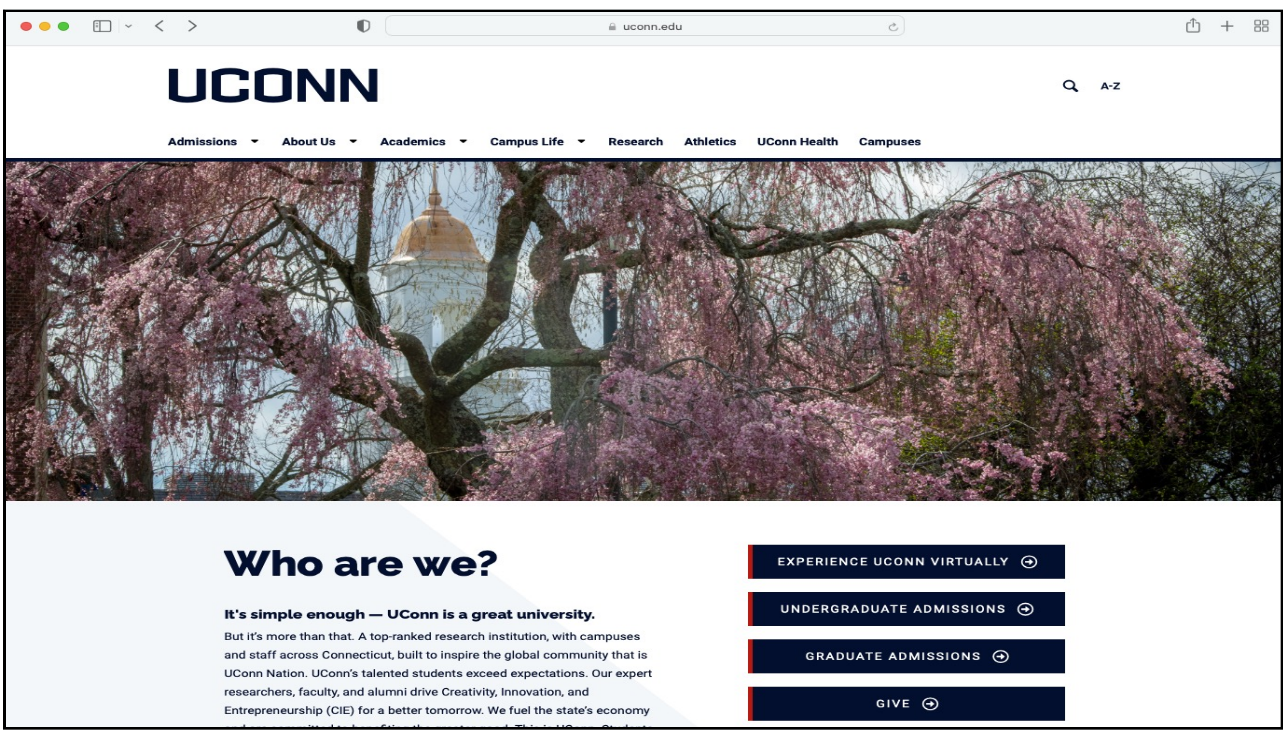
Abstract Data Types (ADT)

In ADT,

1. What input it takes
2. What methods (or operations) it supports on the data
3. What is the expected output

- Sometimes, ADT also describe error situations and what happens if an error occurs.
  - The implementation of ADT is called "Data Structure" or "Concrete Data Structure".
- Concept of Encapsulation will be used to implement data structure

9



10

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

11

Last-In-First-Out (LIFO)

Week 4 – 09/19 – 09/23 – **Lecture 1**

Hasan Baig

11

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

12

Stack ADT

A ***stack*** is a collection of objects that are inserted and removed according to the ***last-in, first-out*** (***LIFO***) principle.

- A user may insert objects into a stack at any time
- Only access or remove the most recently inserted object that remains (at the so-called “top” of the stack)

Week 4 – 09/19 – 09/23 – **Lecture 1**

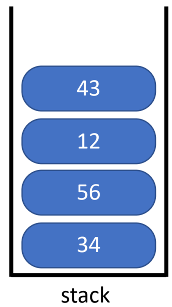
Hasan Baig

12

Stack ADT | Operations
PUSH

Add the element at the top of the stack

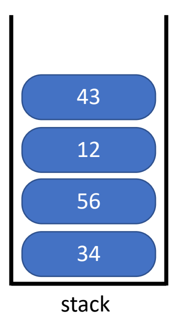
stack.push(34)
stack.push(56)
stack.push(12)
stack.push(43)



Stack ADT | Operations
POP

Returns and removes the element from the top of the stack

stack.pop( )
stack.pop( )

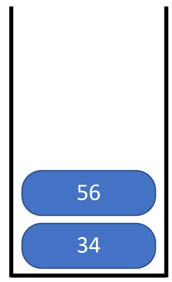


Stack ADT Operations

`PEEK/TOP`

Returns the element from the top of the stack without removing it

stack.peek( )



stack

Hasan Baig

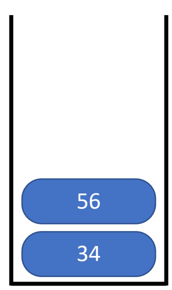
15

Stack ADT Operations

`is_empty`

Returns True if stack does not contain any element

stack.is_empty( )



stack

FALSE

Hasan Baig

16

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

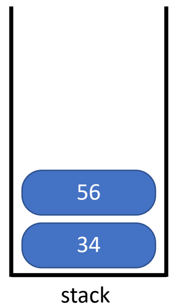
17

Stack ADT Operations

len

Returns the number of elements in the stack

stack.len ( )



2

Week 4 – 09/19 – 09/23 – **Lecture 1**

Hasan Baig

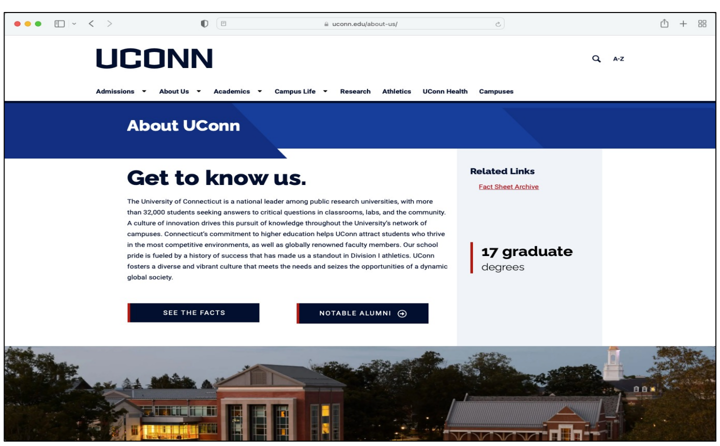
17

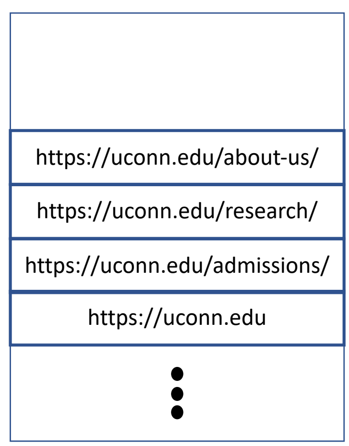
CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

18

Stack ADT Operations





Week 4 – 09/19 – 09/23 – **Lecture 1**

Hasan Baig

18

Stack ADT Implementation

Stack ADT can be implemented easily with a `list`

- What OOP strategies are used?
  - Class, encapsulation, composition
- Bottom of stack → first element in the list
- Top of stack → last element in the list

All operations can be performed in constant time

Stack.py

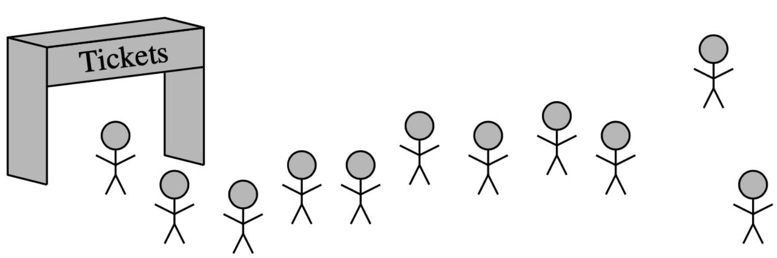
```
class ListStack:
    def __init__(self):
        self._L = []

    def push(self, item):
        self._L.append(item)          O(1)
    def pop(self):
        return self._L.pop()          O(1)
    def peek(self):
        return self._L[-1]            O(1)
    def __len__(self):
        return len(self._L)           O(1)
    def isempty(self):
        return len(self) == 0         O(1)
```


Queue ADT

A ***queue*** is a collection of objects that are inserted and removed according to the ***first-in, first-out*** (***FIFO***) principle.

- Elements can be inserted at the ***back*** in the queue
- Element, in ***front***, that has been in the queue for longest can be removed



Queue ADT Operations

Q.enqueue(e)	Add element e to the back of queue Q.
Q.dequeue()	Remove and return the first element from queue Q; an error occurs if the queue is empty.
Q.first() or Q.peek()	Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.
Q.is empty( ):	Return True if queue Q does not contain any elements.
len(Q)	Return the number of elements in queue Q. In Python, we implement this with the special method `__len__`.

Queue ADT Operations

Operation	Return Value	first ← Q ← last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue( )	5	[3]
Q.is empty( )	False	[3]
Q.dequeue( )	3	[ ]
Q.is empty( )	True	[ ]
Q.dequeue( )	“error”	[ ]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first( )	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]
Q.dequeue( )	7	[9, 4]

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

24

Queue ADT Implementation

Can easily use list to implement

→ inefficient

- Pop(0) → shift elements towards left to fill the gap
- Causes worst case behavior O(n)

```python
class ListQueueSimple:
    def __init__(self):
        self._L = []

    def enqueue(self, item):          # O(1)
        self._L.append(item)

    def dequeue(self):                # O(n)
        return self._L.pop(0)

    def peek(self):                   # O(1)
        return self._L[0]

    def __len__(self):                # O(1)
        return len(self._L)

    def isempty(self):                # O(1)
    return len(self._L) == 0
```

O(1)

O(n)

O(1)

O(1)

O(1)

Week 4 – 09/19 – 09/23 – Lecture 1

Hasan Baig

24

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

25

Queue ADT Implementation

Alternate approach – avoid the call to pop(0) entirely

- Use variable `head` to store the index of element in front of queue
- Dequeue element using `head` without removing it.
- → dequeue operation will run in O(1) time
- There is a drawback of this approach!
  - Size of list → O(m)
  - m is the total number of enqueue operations

```python
class ListQueueHead:
    def __init__(self):
        self._L = []
        self._head = 0

    def enqueue(self, item):
        self._L.append(item)

    def dequeue(self):
        front_item = self.peek()
        self._head += 1
        return front_item

    def peek(self):
        return self._L[self._head]

    def __len__(self):
        return len(self._L)

    def isempty(self):
        return len(self._L) == 0
```

Week 4 – 09/19 – 09/23 – Lecture 1

Hasan Baig

25

UCONN
UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

26

Quick Recap

- Abstract Data Types (ADT)
- Stacks – LIFO
  - Operations: PUSH, POP, PEEK, IS_EMPTY, LEN
  - Cost: O(1)
- Queues – FIFO
  - Operations: ENQUEUE, DEQUEUE, FIRST, IS_EMPTY, LEN
  - Cost: Dequeue → O(n)
- To overcome this, we made use of `head` variable to dequeue the element in front of queue
  - Space issue

Error Handling

- Displaying an error message is the correct behavior.
- Example: Executing a pop operation on an empty stack.

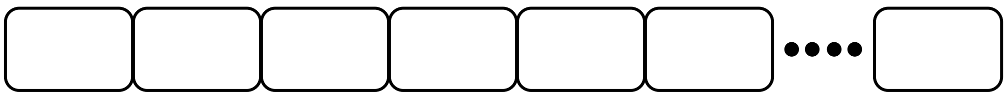
```
def pop(self):
    try:
        return self._L.pop()
    except:
        raise Exception("Trying to pop from empty stack.")
```


Deque ADT

- Deque (pronounced as "deck") is a **D**oubly-**E**nded-**QUE**ue .
- Acts like a Stack and Queue
  - Add or remove elements from both the beginning and the end
- **addfirst(item)** - add item to the front of the deque.
- **addlast(item)** - add item to the end of the deque.
- **removefirst(item)** - remove and return the first item in the deque.
- **removelast(item)** - remove and return the last item in the deque.
- **len** - return the number of items in the deque.

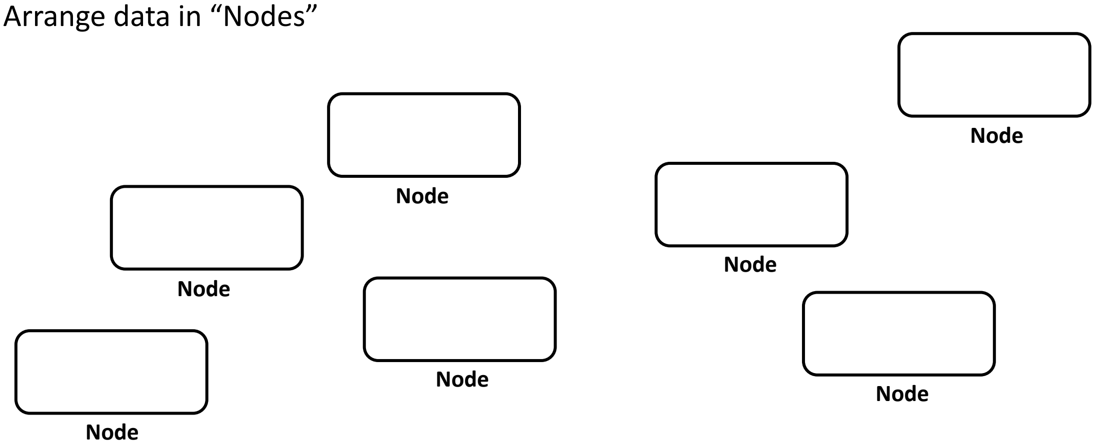
```
class ListDeque:
    def __init__(self):
        self._L = []
    def addfirst(self, item):
        self._L.insert(0, item)
    def addlast(self, item):
        self._L.append(item)
    def removefirst(self):
        return self._L.pop(0)
    def removelast(self):
        return self._L.pop()
    def __len__(self):
        return len(self._L)
```

Same problem of shifting elements towards right or left!
Only because of
The arrangement of items in a memory sequentially.



Drop the idea of storing elements sequentially in a memory.

Arrange data in “Nodes”

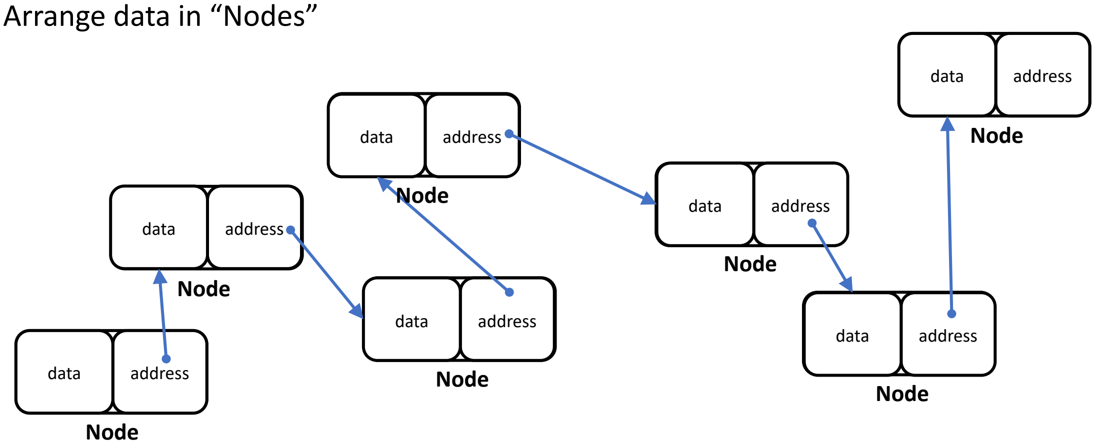


CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

32

Arrange data in "Nodes"



Week 4 – 09/19 – 09/23 – Lecture 2

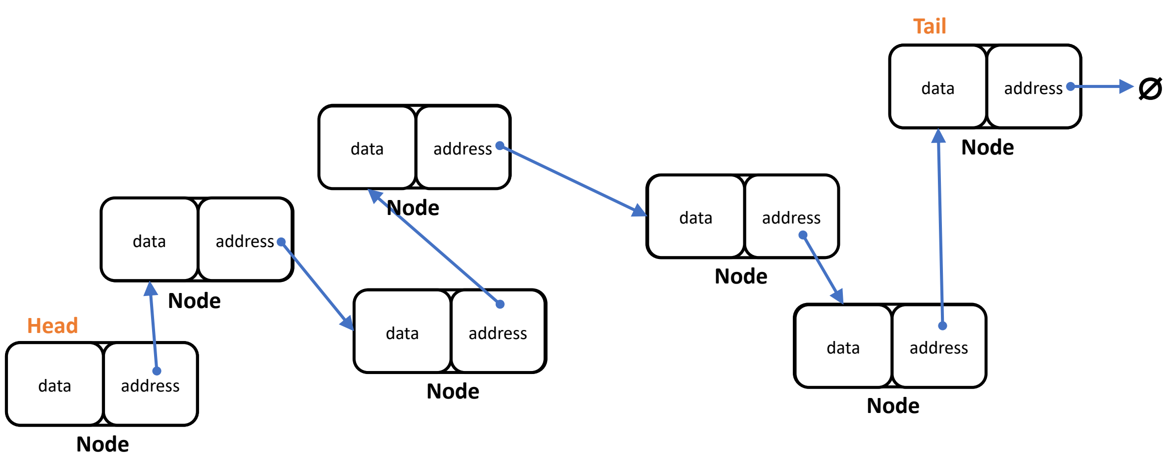
Hasan Baig

32

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

33



Week 4 – 09/19 – 09/23 – Lecture 2

Hasan Baig

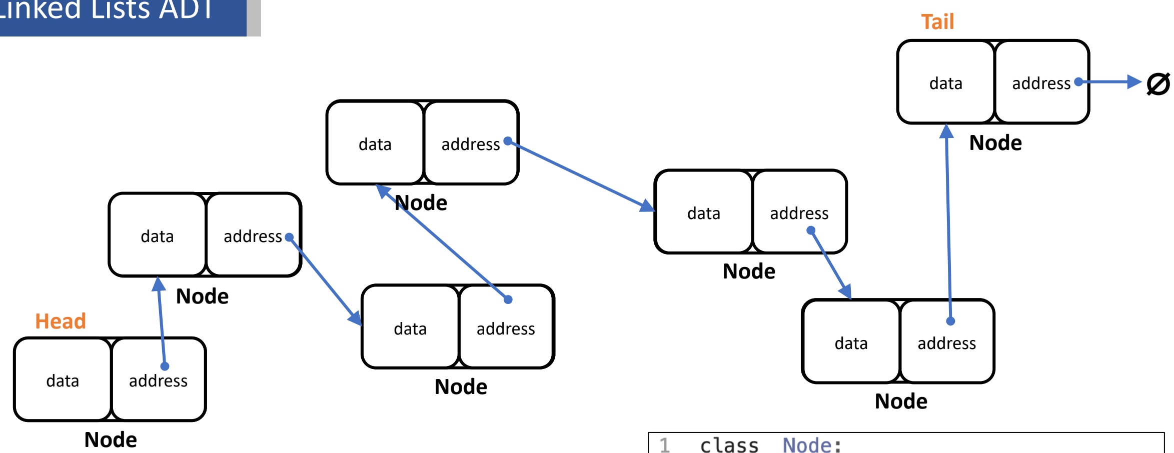
33

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

34

Linked Lists ADT



- Data is arranged in distributed nodes
- Each node contains the element (data) + the address of next node

```python
class _Node:
    """ Node class to create
    individual linked list  nodes """
    def __init__(self, element, next):
        self._element = element
        self._next = next
```

Hasan Baig

Week 4 – 09/19 – 09/23 – Lecture 2

34

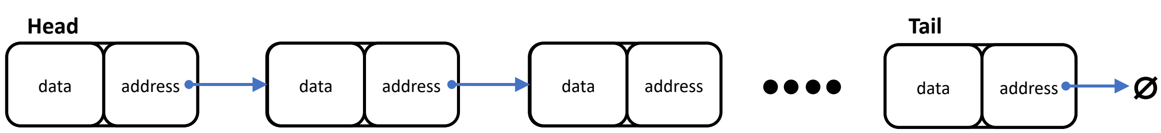
CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

35

Linked Lists ADT | Singly Linked List

- Most simplest form of linked list
- First node is the head node. Last node is the tail node.



- Tail node can be determined by :
  - checking its pointer to next node → None
  - Traversing the list
- Since the next reference of a node can be viewed as a *link* or a *pointer* → traversal is also referred to as "link hopping" or "pointer hopping"

Hasan Baig

Week 4 – 09/19 – 09/23 – Lecture 2

35

CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

36

Linked Lists ADT | Singly Linked List

- Minimally, instance of a linked list must keep a reference of **head** node
- Also keep the reference of a **tail** node.
  → Traversal can help us get to the tail node ➔ O(n)
- Size of the linked list can also be continuously monitored
  → Traversal can help us count the nodes ➔ O(n)

Week 4 – 09/19 – 09/23 – Lecture 2

Hasan Baig

36

CSE-2050 – Data Structures and Object-Oriented Design

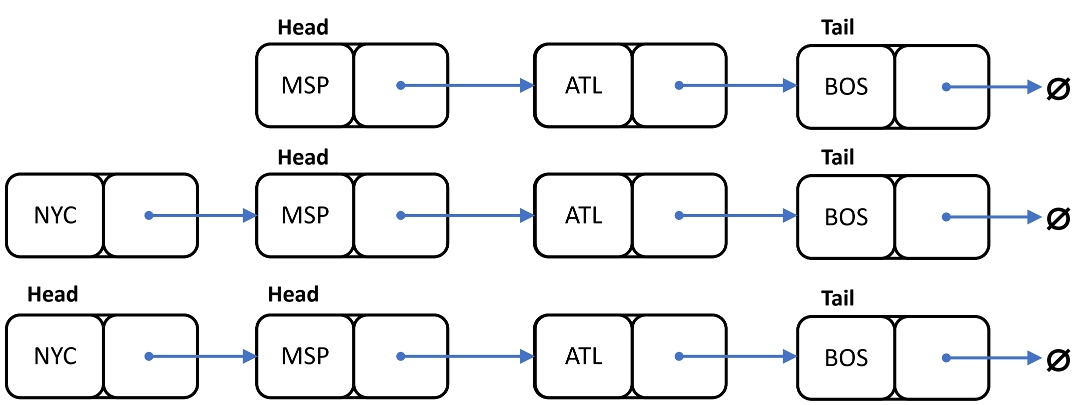
Linear Data Structures

37

Linked Lists ADT | Singly Linked List

Inserting element at:

The HEAD of linked list



Week 4 – 09/19 – 09/23 – Lecture 2

Hasan Baig

37

Linked Lists ADT | Singly Linked List

Inserting element at:

The HEAD of linked list

Pseudocode:

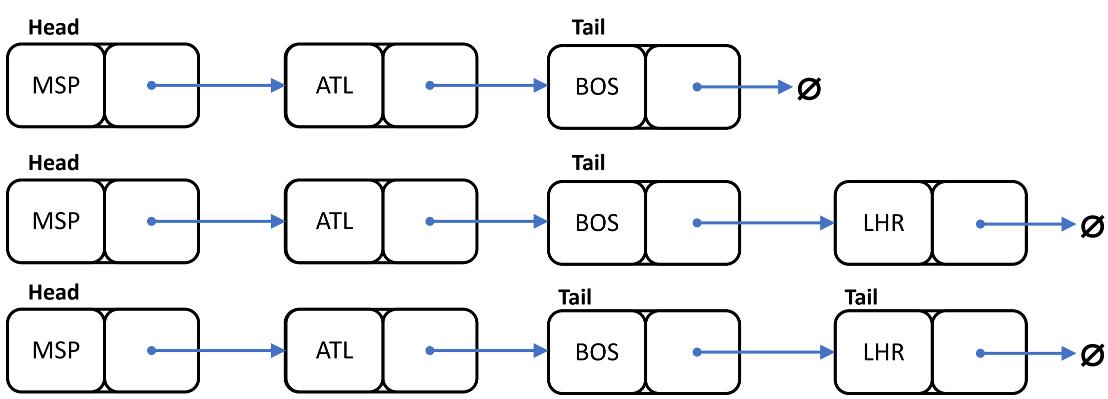
1. Create a node with an element
   `newest = _Node(element)`
2. Set the `newest` node `next` reference to the current head node:
   `newest._next = L.head`
3. Set the variable “`head`” to refer the newest node as the head node
   `L.head = newest`

4. Update the size of the list
   `L.size += 1`

Linked Lists ADT | Singly Linked List

Inserting element at:

The TAIL of linked list



CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

40

Linked Lists ADT | Singly Linked List

Inserting element at:

The TAIL of linked list

Pseudocode:

1. Create a node with an element
   `newest = Node(element)`
2. Set the `newest` node `next` reference to `None`
   `newest.next = None`
3. Set the next node reference of current `tail` to point to the `newest` node
   `L.tail.next = newest`

4. Set the variable "`tail`" to reference to newest node
   `L.Tail = newest`

4. Update the size of the list
   `L.size += 1`

Week 4 – 09/19 – 09/23 – Lecture 2

Hasan Baig

40

CSE-2050 – Data Structures and Object-Oriented Design

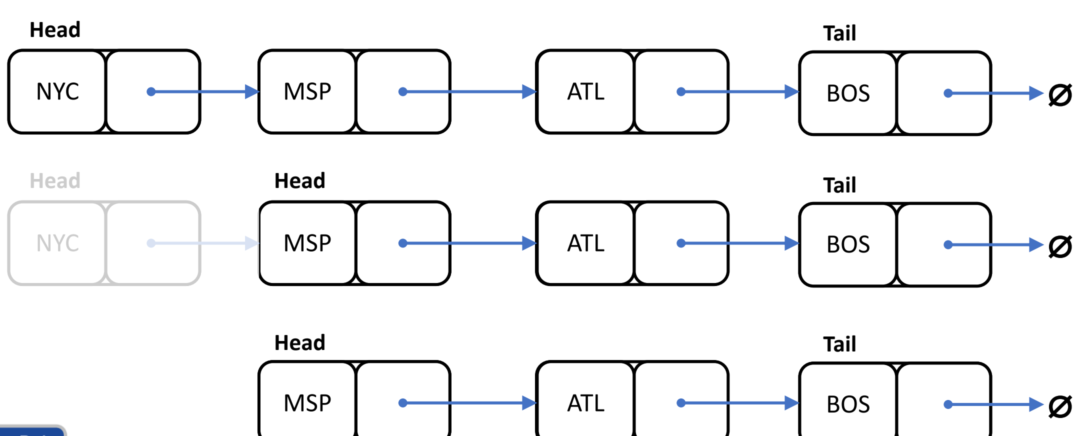
Linear Data Structures

41

Linked Lists ADT | Singly Linked List

Deleting element from:

The HEAD of linked list



Week 4 – 09/19 – 09/23 – Lecture 2

Hasan Baig

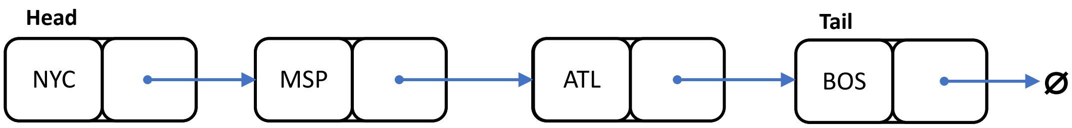
41

Linked Lists ADT

Singly Linked List

Deleting element from:

The TAIL of linked list



- We cannot easily delete the last node of a singly linked list
  - Though, we can hold a reference to the Tail node
  - There is no way to determine the node preceding to the Tail node
- List traversal may help, but takes O(n) time

Activity # 7

Write a pseudocode (steps) for implementing the Stack ADT using Linked List.

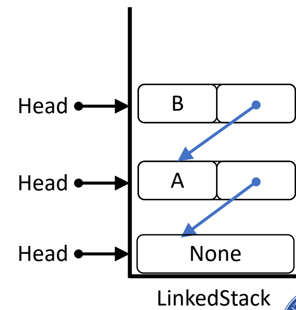
Linked Lists ADT | Stack ADT as Linked List

Write a pseudocode (steps) for implementing the Stack ADT using Linked List.

```
class _Node:
    """ Node class to create
    individual linked list  nodes """
    def __init__(self, element, next):
        self._element = element
        self._next = next
```

1. Start off with an instance of `ListStack` with:
   head = None
2. PUSH operation → Create an instance of the private Node class with:
   head = _Node("A", head)
2. PUSH operation → Create an instance of the private Node class with:
   head = _Node("B", head)

For each PUSH operation, increase the size to keep track of the stack size



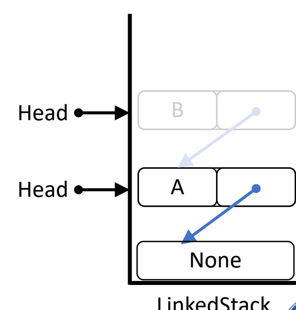
Linked Lists ADT | Stack ADT as Linked List

Write a pseudocode (steps) for implementing the Stack ADT using Linked List.

```
class _Node:
    """ Node class to create
    individual linked list  nodes """
    def __init__(self, element, next):
        self._element = element
        self._next = next
```

To POP the element from stack

1. Get the element of "head" node
   read_element = head
2. Set the next head node to the "next" pointer of the current head node
   head = head._next
2. Reduce the size of the stack and return the read_element
   size -= 1
   return read_element



CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

46

Linked Lists ADT | Queue ADT as Linked List

LinkedQueue

1. Start off by creating an instance of `LinkedQueue` with:
   head = None
   tail = None

Week 4 – 09/19 – 09/23 – Lecture 2

Hasan Baig

46

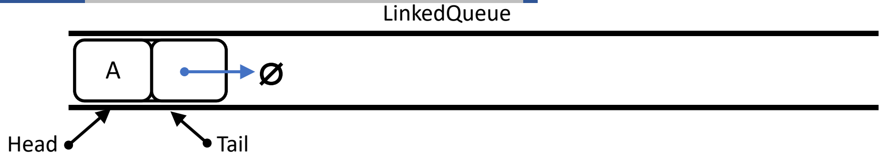
CSE-2050 – Data Structures and Object-Oriented Design

Linear Data Structures

47

Linked Lists ADT | Queue ADT as Linked List

LinkedQueue



Enqueue → Create an instance of a private class Node with:
   newest = _Node("A", None)
   if it is the first element then
      head = newest

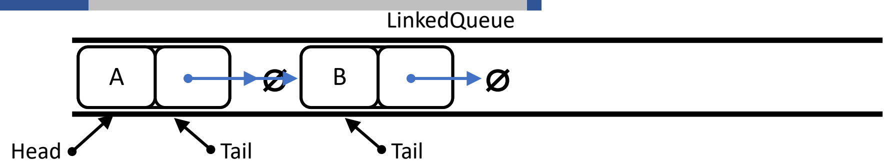
- Set the element as a Tail
   tail = newest
- Increase the size of queue

Week 4 – 09/19 – 09/23 – Lecture 2

Hasan Baig

47

Linked Lists ADT — Queue ADT as Linked List

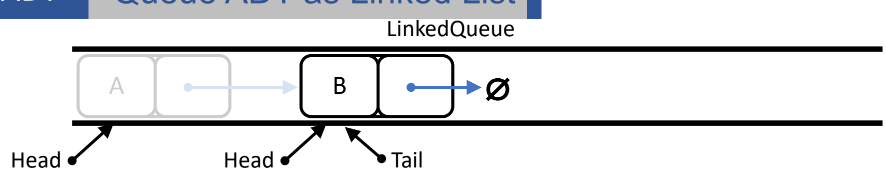


Enqueue → Create an instance of a private class Node with:

newest = _Node("B", None)
if it is the first element then
head = newest
else
tail._next = newest

- Set the element as a Tail
  tail = newest
- Increase the size of queue

Linked Lists ADT — Queue ADT as Linked List



Dequeue → Read the element

read_element = head._element

- Update the head variable to be set to the next node
  head = head._next
  if it was the last element in queue
  tail = None
- Reduce the size of queue
- Return read_element