Linked List
UNIT-3

Unit: 3

Data Structures

(B Tech 3rd Sem)

Dr. Ritesh Rastogi
Associate Professor
Dept. of CSE,
M.Tech(Integrated)

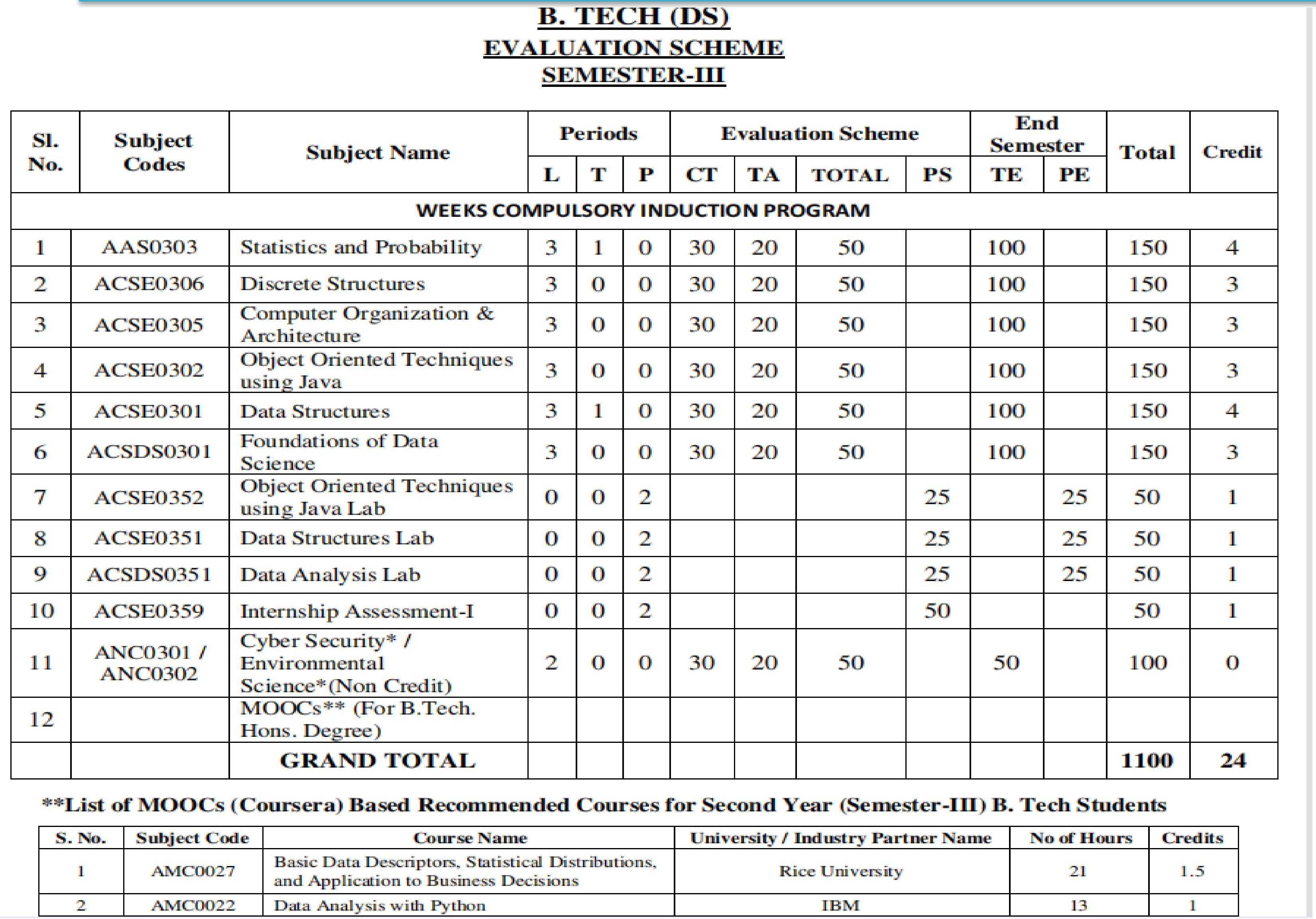
Faculty Profile

- Dr. Ritesh Rastogi (M.Tech,, Ph.D)
  (Associate Professor, Dept. of IT, M.Tech (CSE) Integrated
  NIET, Greater Noida

- **Experience :** 25 Years (Teaching and Research)
- **Area of Interest :** Software Engg./Testing ,DBMS, Cloud
- **Honors , Awards and Achievements**
- Published more than **50 papers in SCI/Scopus/peer reviewed national/international journals and conferences**
- Authored **four books** of computer science and a **book chapter**.
- Published and Granted **04 Patents**
- Guided around 120 PG thesis and projects of M.Tech and MCA Students
- **Awarded** as **"Corona Warrior"** by Engineering and Management College Teachers Development Association
- **Research Excellence Award 2020** from *Institute of Scholars for research work.*
- **Best Research Paper Award**, in ICFCCT 2021 organized by IFERP on 27-28th Oct 2021 Mumbai
- **Humanitarian Excellence Awards 2021**, Certificate of Appreciation by I Can Foundation
- **Member** of professional societies like ISTE, IFERP, C# Corner, Oracle Incorp. IAENG, InSc.
- Brand ambassador of IFERP. And Coordinator for IIRS-ISRO Dehra dun.
- Member of Editorial board of International Journal.

Dr. Ritesh Rastogi

Evaluation Scheme



B. TECH (DS)
EVALUATION SCHEME
SEMESTER-III

Sl. No.	Subject Codes	Subject Name	Periods			Evaluation Scheme				End Semester		Total	Credit
			L	T	P	CT	TA	TOTAL	PS	TE	PE		
		WEEKS COMPULSORY INDUCTION PROGRAM											
1	AAS0303	Statistics and Probability	3	1	0	30	20	50		100		150	4
2	ACSE0306	Discrete Structures	3	0	0	30	20	50		100		150	3
3	ACSE0305	Computer Organization & Architecture	3	0	0	30	20	50		100		150	3
4	ACSE0302	Object Oriented Techniques using Java	3	0	0	30	20	50		100		150	3
5	ACSE0301	Data Structures	3	1	0	30	20	50		100		150	4
6	ACSDS0301	Foundations of Data Science	3	0	0	30	20	50		100		150	3
7	ACSE0352	Object Oriented Techniques using Java Lab	0	0	2				25		25	50	1
8	ACSE0351	Data Structures Lab	0	0	2				25		25	50	1
9	ACSDS0351	Data Analysis Lab	0	0	2				25		25	50	1
10	ACSE0359	Internship Assessment-I	0	0	2				50			50	1
11	ANC0301 / ANC0302	Cyber Security* / Environmental Science*(Non Credit)	2	0	0	30	20	50		50		100	0
12		MOOCs** (For B.Tech. Hons. Degree)											
		GRAND TOTAL										1100	24

****List of MOOCs (Coursera) Based Recommended Courses for Second Year (Semester-III) B. Tech Students**

S. No.	Subject Code	Course Name	University / Industry Partner Name	No of Hours	Credits
1	AMC0027	Basic Data Descriptors, Statistical Distributions, and Application to Business Decisions	Rice University	21	1.5
2	AMC0022	Data Analysis with Python	IBM	13	1

Unit III Syllabus

- Advantages of linked list over array,
- Self-referential structure,
- Singly Linked List, Doubly Linked List, Circular Linked List.
- **Operations on a Linked List:** Insertion, Deletion, Traversal, Reversal, Searching, Polynomial Representation and Addition of Polynomials.
- Implementation of Stack and Queue using Linked lists.

Unit Content

- Advantages of Linked List over Array
- Singly Linked List
- Doubly Linked List
- Circular Linked List
- Operation on Linked List
  - Insertion
  - Deletion
  - Traversal
  - Reversal
  - Searching Polynomial Representation
  - Addition, Subtraction and Multiplication of Polynomials
- Implementation of Stack and Queue using Linked List

Unit Objective

- To learn about linked lists.
- To understand different types of Linked list.
- Basic operations of linked list.

Course Objective

- Introduction to basic data structures.
- To know about the basic properties of different data structures.
- Classification and operations on data structure
- Understand algorithms and their efficiency
- Study logical and mathematical description of array and link list.
- Implementation of array and link list on computer.
- Differentiate the usage of array and link list in different scenarios.

Course Outcome

CO	CO Description	Bloom's Knowledge Level (KL)
CO 1	Describe the need of data structure and algorithms in problem solving and analyze Time space trade-off.	K2, K4
CO 2	Describe how arrays are represented in memory and how to use them for implementation of matrix operations, searching and sorting along with their computational efficiency.	K2, K6
CO 3	Design, implement and evaluate the real-world applications using stacks, queues and non-linear data structures.	K5, K6
CO 4	Compare and contrast the advantages and disadvantages of linked lists over arrays and implement operations on different types of linked list.	K4, K6
CO 5	Identify and develop the alternative implementations of data structures with respect to its performance to solve a real-world problem.	K1, K3, K5, K6

Program Outcomes (POs)

1. Engineering knowledge

2. Problem analysis

3. Design/development of solutions

4. Conduct investigations of complex problems

5. Modern tool usage
6. The engineer and society

7. Environment and sustainability

8. Ethics

9. Individual and team work

10. Communication

11. Project management and finance

12. Life-long learning

CO-PO Mapping

CO-PO correlation matrix of Data Structure (KCS 301)

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
ACSE0301.1	3	3	3	2	-	1	-	1	2	2	2	2
ACSE0301.2	3	3	2	2	-	1	-	1	2	2	1	2
ACSE0301.3	3	3	2	2	-	1	-	1	2	2	2	2
ACSE0301.4	3	3	2	2	-	1	-	1	2	2	2	2
ACSE0301.5	3	3	3	3	2	2	2	2	3	3	3	3
Average	3	3	2.4	2.2	0.4	1.2	0.4	1.2	2.2	2.2	2	2.2

Program Specific Outcomes (PSOs)

On successful completion of graduation degree the Engineering graduates will be able to:

PSO1: The ability to design and develop the hardware sensor device and related interfacing software system for solving complex engineering problem.

PSO2: The ability to understanding of Inter disciplinary computing techniques and to apply them in the design of advanced computing .

PSO 3: The ability to conduct investigation of complex problem with the help of technical, managerial, leadership qualities, and modern engineering tools provided by industry sponsored laboratories.

PSO 4: The ability to identify, analyze real world problem and design their solution using artificial intelligence ,robotics, virtual. Augmented reality ,data analytics, block chain technology and cloud computing.

CO-PSO Mapping

Mapping of Program Specific Outcomes and Course Outcomes

	PSO1	PSO2	PSO3	PSO4
ACSE0301.1	3	3	2	2
ACSE0301.2	3	3	2	3
ACSE0301.3	3	3	2	2
ACSE0301.4	3	3	3	3
ACSE0301.5	3	3	3	3
Average	3	3	2.4	2.6

Prerequisite and Recap

- Interest
- Get Familiar with any programming language. C, C++ and Python.
- Start learn Data Structure and Algorithm daily.
- Practice ! Because practice makes you perfect.

Faculty Video Links, Youtube & NPTEL Video Links and Online Courses Details

- Youtube/other Video Links
- Implementation of link list
  - https://www.youtube.com/watch?v=6wXZ_m3SbEs
- Polynomial addition using link list
  - https://www.youtube.com/watch?v=V_ZNKu_pUPQ

Basic Terminology(CO1)

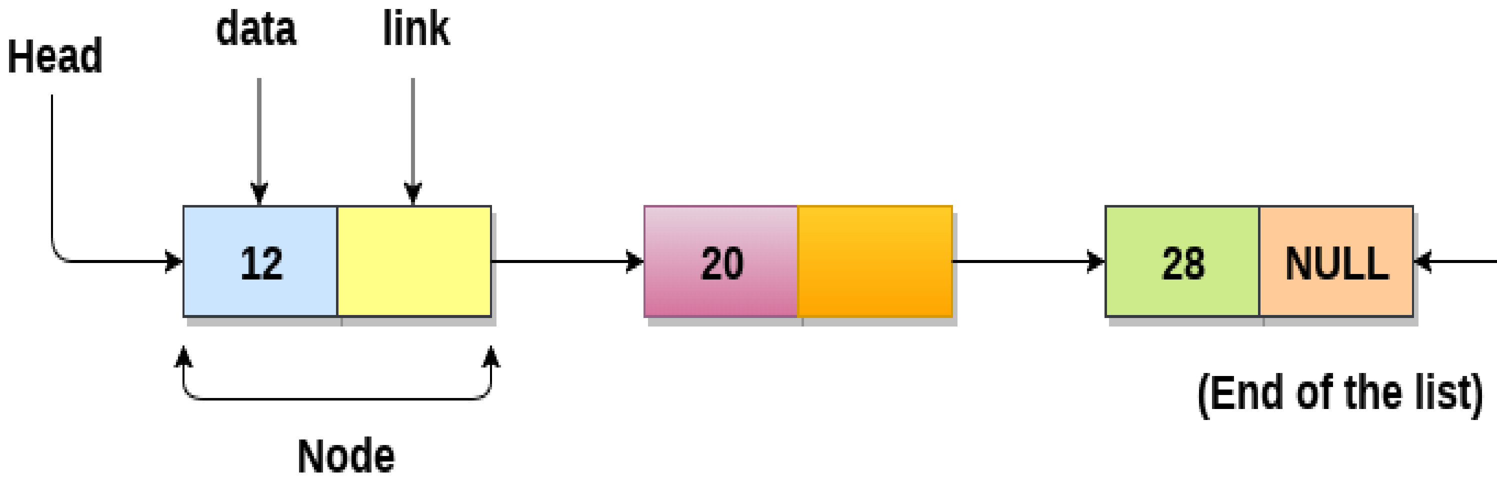
- **Linked List**
- **Doubly Linked List**
- **Circularly Linked List**
- **Circularly Doubly Linked List**

Topic Objective

- To understand linked list and the operations of linked list.
- To implement Linked list program using Python

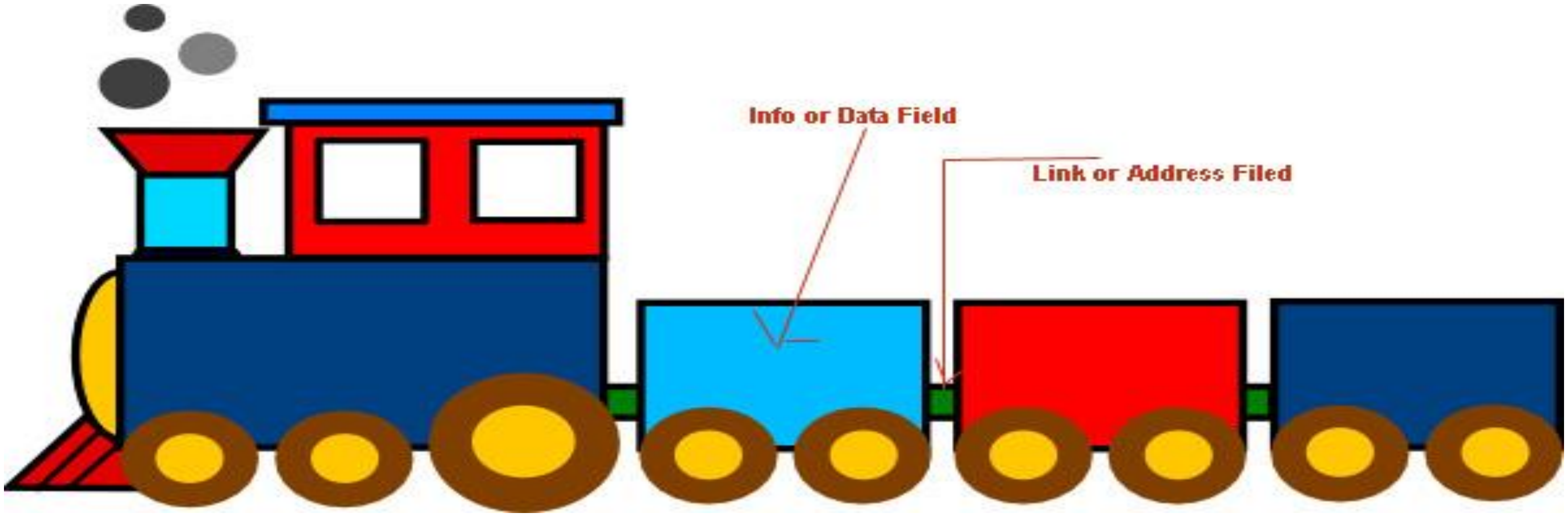
Linked List

- Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



Linked List

- A linked list is a linear data structure.
- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node.
- Usually the pointer is called next.



Introduction to Linked List

Linked List

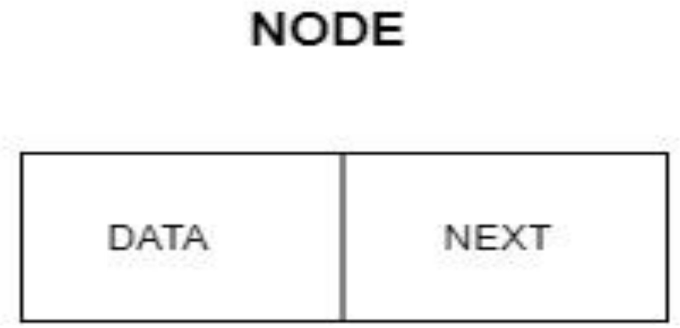
- The elements of a linked list are not stored in adjacent memory locations as in arrays.
- It is a linear collection of data elements, called nodes, where the linear order is implemented by means of pointers.

Continued….

Linked List

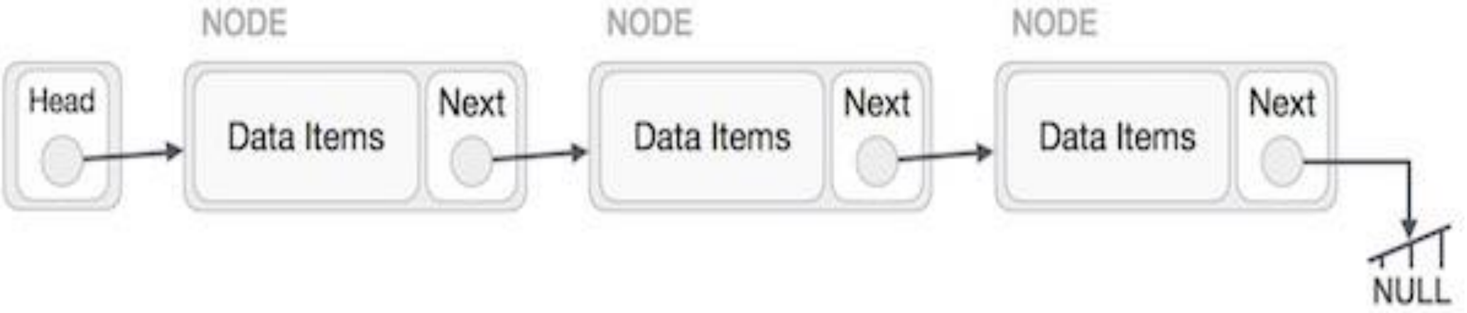
- In a linear or single-linked list, a node is connected to the next node by a single link.

- A node in this type of linked list contains two types of fields
  - data: which holds a list element
  - next: which stores a link (i.e. pointer) to the next node in the list.



Linked List Representation

- Linked list can be visualized as a chain of nodes, where every node points to the next node.



- As per the above illustration, following are the important points to be considered.
  - Linked List contains a link element called first.
  - Each link carries a data field(s) and a link field called next.
  - Each link is linked with its next link using its next link.
  - Last link carries a link as null to mark the end of the list.

Properties of linked list

- The nodes in a linked list are not stored contiguously in the memory
- You don’t have to shift any element in the list
- Memory for each node can be allocated dynamically whenever the need arises.
- The size of a linked list can grow or shrink dynamically

Basic Operations on Linked List

- Following are the basic operations supported by a list.
  - **Insertion** − Adds an element at the beginning of the list.
  - **Deletion** − Deletes an element at the beginning of the list.
  - **Display** − Displays the complete list.
  - **Search** − Searches an element using the given key.
  - **Delete** − Deletes an element using the given key.

Arrays & Linked list

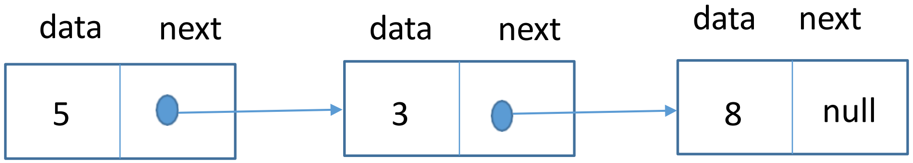
Arrays	Linked list
Fixed size: Resizing is expensive	Dynamic size
Insertions and Deletions are inefficient: Elements are usually shifted	Insertions and Deletions are efficient: No shifting
Random access i.e., efficient indexing	No random access → Not suitable for operations requiring accessing elements by index such as sorting
No memory waste if the array is full or almost full; otherwise may result in much memory waste.	Since memory is allocated dynamically(acc. to our need) there is no waste of memory.
Sequential access is faster [Reason: Elements in contiguous memory locations]	Sequential access is slow [Reason: Elements not in contiguous memory locations]

Types of Link List

- Following are the various types of linked list.
  - **Singly Linked List** − Item navigation is forward only.
  - **Doubly Linked List** − Items can be navigated forward and backward.
  - **Circular Linked List** − Last item contains link of the first element as next
  - **Circular Doubly Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous. Items can be navigated forward and backward.

Singly Linked list

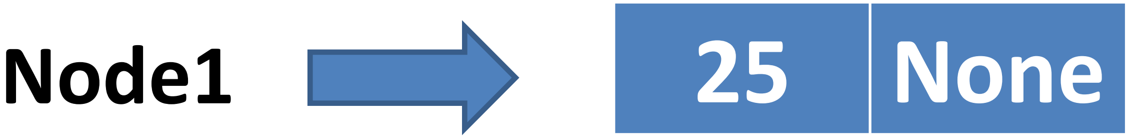
- A singly linked list is a dynamic data structure which may grow or shrink, and growing and shrinking depends on the operation made.
- In this type of linked list each node contains two fields one is data field which is used to store the data items and another is next field that is used to point the next node in the list.



Creating a node of linked list

```
# Node class (Creating a node of linked list)
class Node:
      # Function to initialize the node object
    def __init__(self, data):
        self.data = data  # Assign data
        self.next = None  # Initialize next as null
```

```
Node1=Node(25)
```



Node1

25	None

Creating an empty linked list

```
# Node class (Creating a node of linked list)
class Node:
      # Function to initialize the node object
    def __init__(self, data):
        self.data = data  # Assign data
        self.next = None  # Initialize next as null

# Linked List class (Linking the nodes of linked list)
class LinkedList:
        # Function to initialize the Linked List object
    def __init__(self):
        self.head = None
```

Creating a linked list with single node

```
class Node:
    def __init__(self, data):
    self.data = data
    self.next = None

class LinkedList:
    def __init__(self):
    self.head = None

LL = LinkedList()
LL.head = Node(3)
print(LL.head.data)
```

Creation and Traversal of single linked list

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
  self.data = data
  self.next = None


# A Linked List class with a single
   head node
class LinkedList:
 def __init__(self):
  self.head = None
```

```
# insertion method for the linked list
 def insert(self, data):
  newNode = Node(data)
  if(self.head):
   current = self.head
   while(current.next):
    current = current.next
   current.next = newNode
  else:
   self.head = newNode
```

Creation and Traversal of single linked list (contd..)

```
# print method for the linked list
 def printLL(self):
   current = self.head
   while(current):
     print(current.data)
     current = current.next

# Singly Linked List with insertion and print methods
LL = LinkedList()
LL.insert(3)
LL.insert(4)
LL.insert(5)
LL.printLL()
```

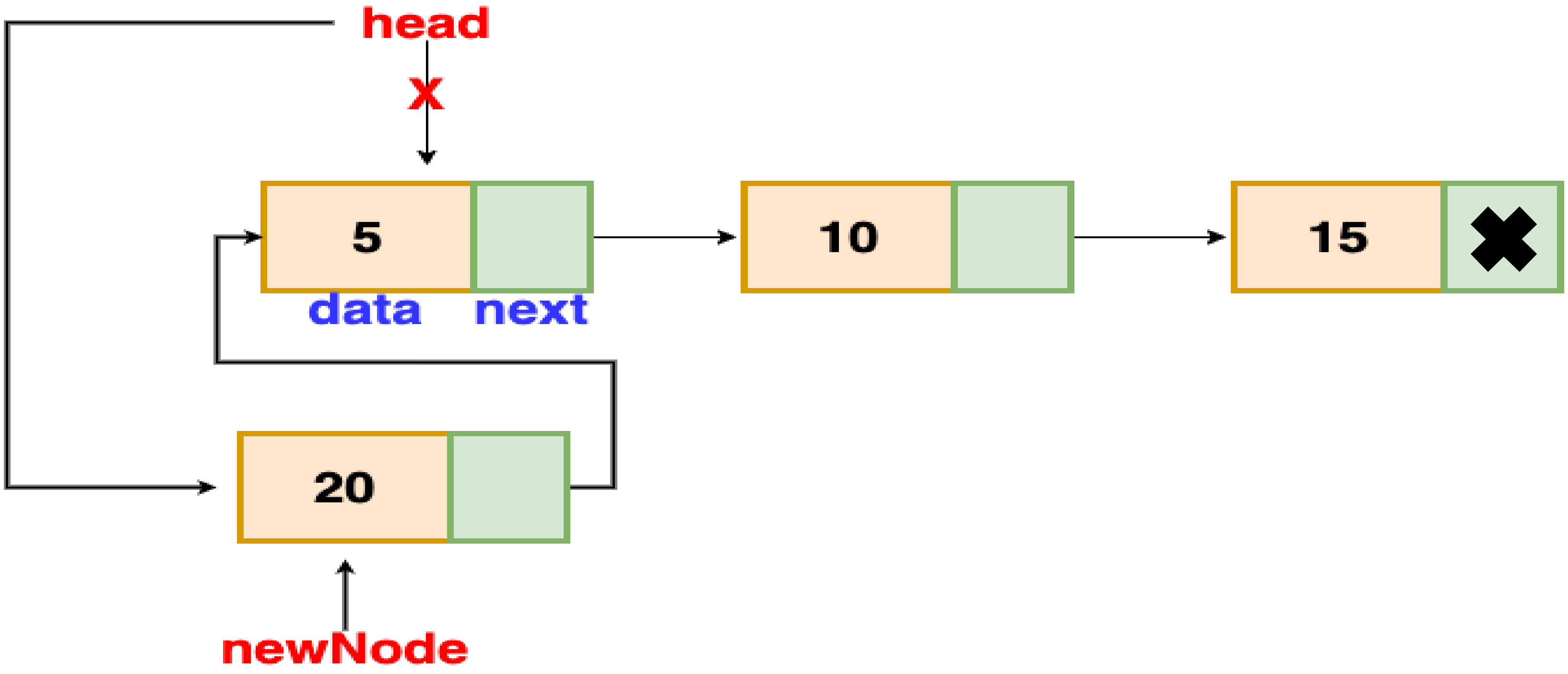
Insertion in a Single Linked List

- There are three possible positions where we can enter a new node in a linked list –
  - **Insertion at beginning**
  - **Insertion at end**
  - **Insertion at given position**

- Adding a new node in linked list is a more than one step activity.

Insertion in a Single Linked List (at beginning)

- Insertion at beginning



Insertion at the beginning

Insertion in single linked list (at beginning)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
    self.data = data
    self.next = None
```

```
# A Linked List class with a single head node
class LinkedList:
  def __init__(self):
    self.head = None
```

```
# insertion method for the linked list at beginning

def insert_beg(self, data):
    newNode = Node(data)
    if(self.head):
      newNode.next=self.head
      self.head=newNode
    else:
      self.head = newNode
```

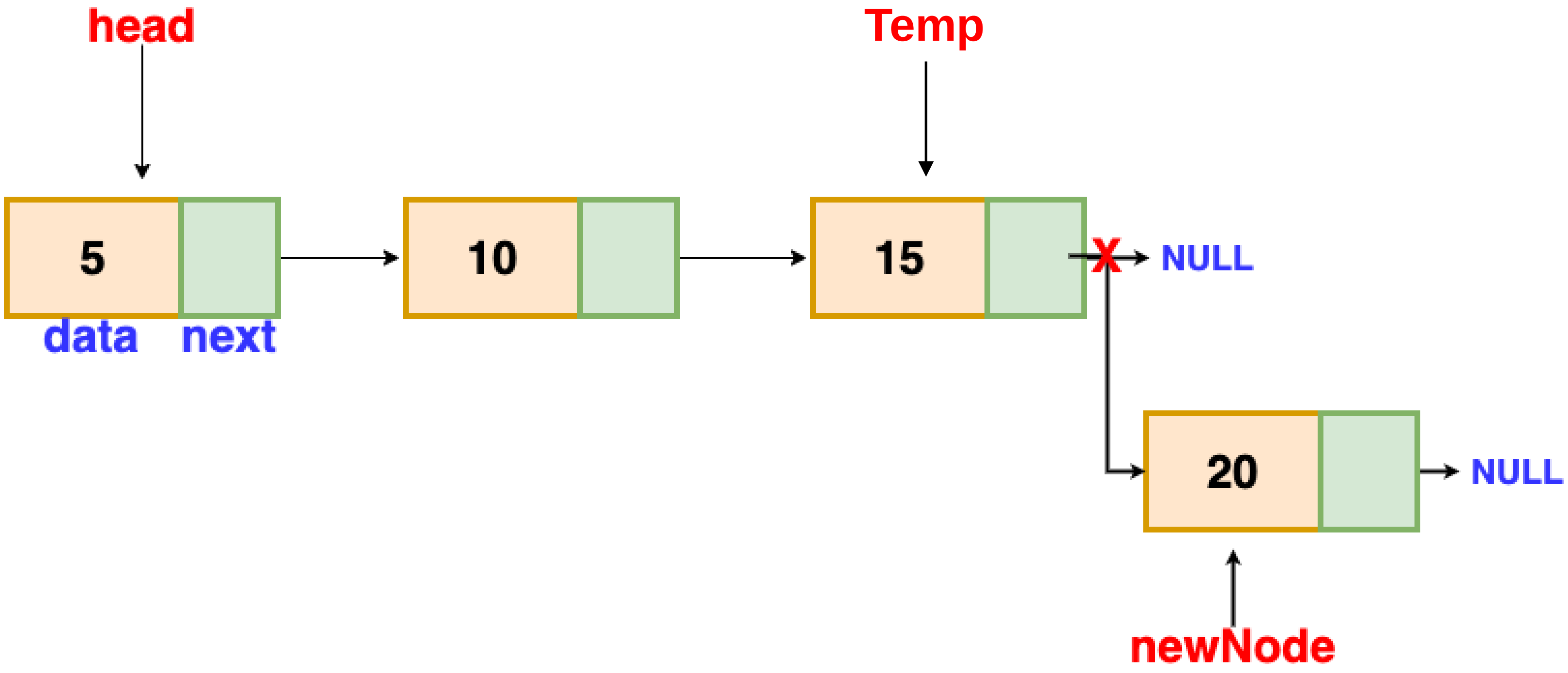
Insertion in single linked list (at beginning) (contd..)

```
# print method for the linked list
 def printLL(self):
  current = self.head
  if(current!=None):
    print("The List Contains:",end="\n")
    while(current):
      print(current.data)
      current = current.next
  else:
    print("List is Empty.")

# Singly Linked List with insertion and print methods
LL = LinkedList()
LL.insert_beg(3)
LL.insert_beg(4)
LL.insert_beg(5)
LL.printLL()
```

Insertion in a Single Linked List (at end)

- **Insertion at end**



Insertion at the end

Insertion in single linked list (at end)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
    self.data = data
    self.next = None


# A Linked List class with a single head node
class LinkedList:
  def __init__(self):
    self.head = None
```

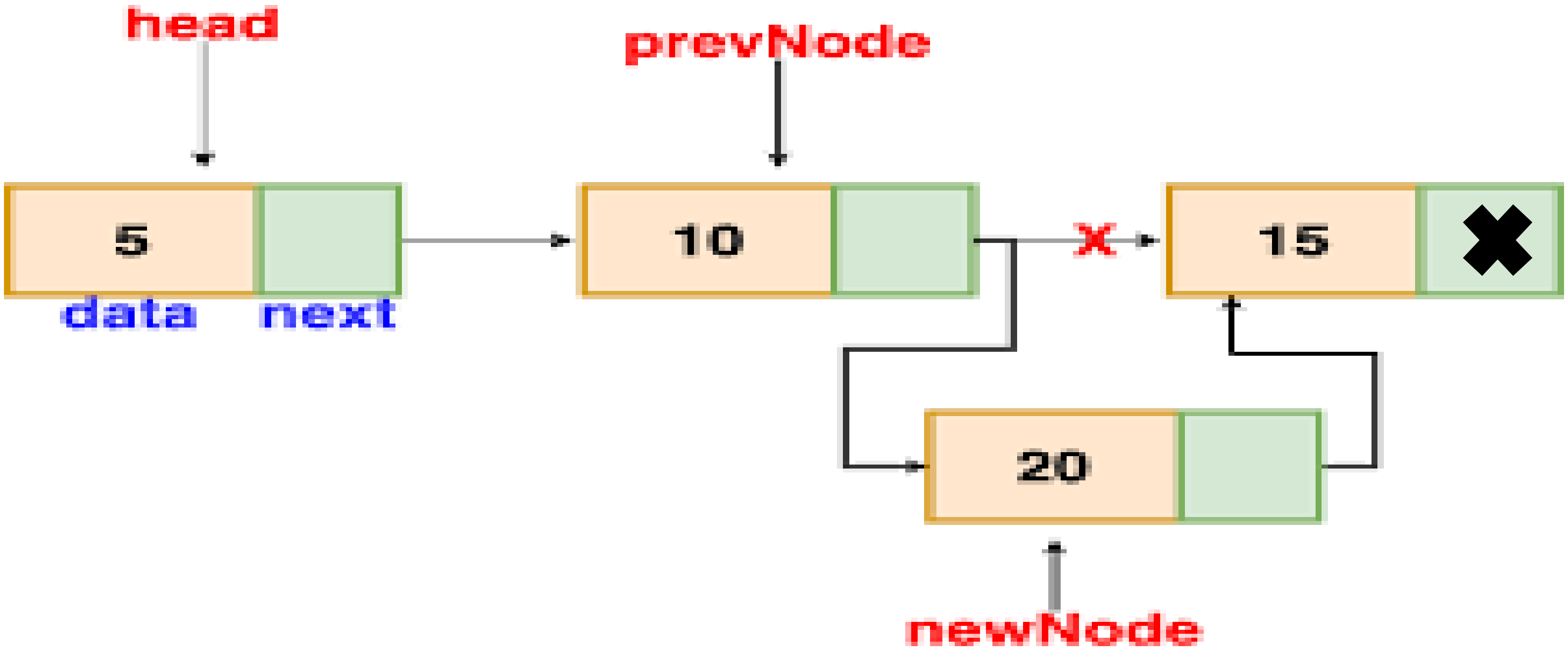
```
# insertion method for the linked list at end
def insert_end(self, data):
  newNode = Node(data)
  if(self.head):
    current = self.head
    while(current.next):
      current = current.next
    current.next = newNode
  else:
    self.head = newNode
```

Insertion in single linked list (at end) (contd..)

```
# print method for the linked list
 def printLL(self):
  current = self.head
  if(current!=None):
    print("The List Contains:",end="\n")
    while(current):
      print(current.data)
      current = current.next
  else:
    print("List is Empty.")
# Singly Linked List with insertion and print methods
LL = LinkedList()
LL.insert_end(3)
LL.insert_end(4)
LL.insert_end(5)
LL.printLL()
```

Insertion in a Single Linked List (at given position)

- **Insertion at given position**



Insertion after a given node

Insertion in single linked list (at position)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
    self.data = data
    self.next = None


# A Linked List class with a single
      head node
class LinkedList:
  def __init__(self):
    self.head = None
```

```
# creation method for the linked list
  def create(self, data):
    newNode = Node(data)
    if(self.head):
      current = self.head
      while(current.next):
        current = current.next
      current.next = newNode
    else:
      self.head = newNode
```

Insertion in single linked list (at position)

```
# insertion method for the linked list at given position

def insert_position(self, data, pos):
  newNode = Node(data)
  if(pos<1):
      print("\nPosition should be >=1.")

  elif(pos==1):
      newNode.next=self.head
      self.head=newNode
  else:
      current=self.head
      for i in range(1, pos-1):
          if(current!=None):
              current=current.next
      if(current!=None):
          newNode.next=current.next
          current.next=newNode
      else:
          print("\nThe previous node is null.")
```

Insertion in single linked list (at position)

```
# print method for the linked list
 def printLL(self):
  current = self.head
  if(current!=None):
    print("The List Contains:",end="\n")
    while(current):
      print(current.data)
      current = current.next
  else:
    print("List is Empty.")
```

```
# Singly Linked List with insertion and print methods
LL = LinkedList()
LL.create(2)
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.insert_position(9, 4)
LL.printLL()
```

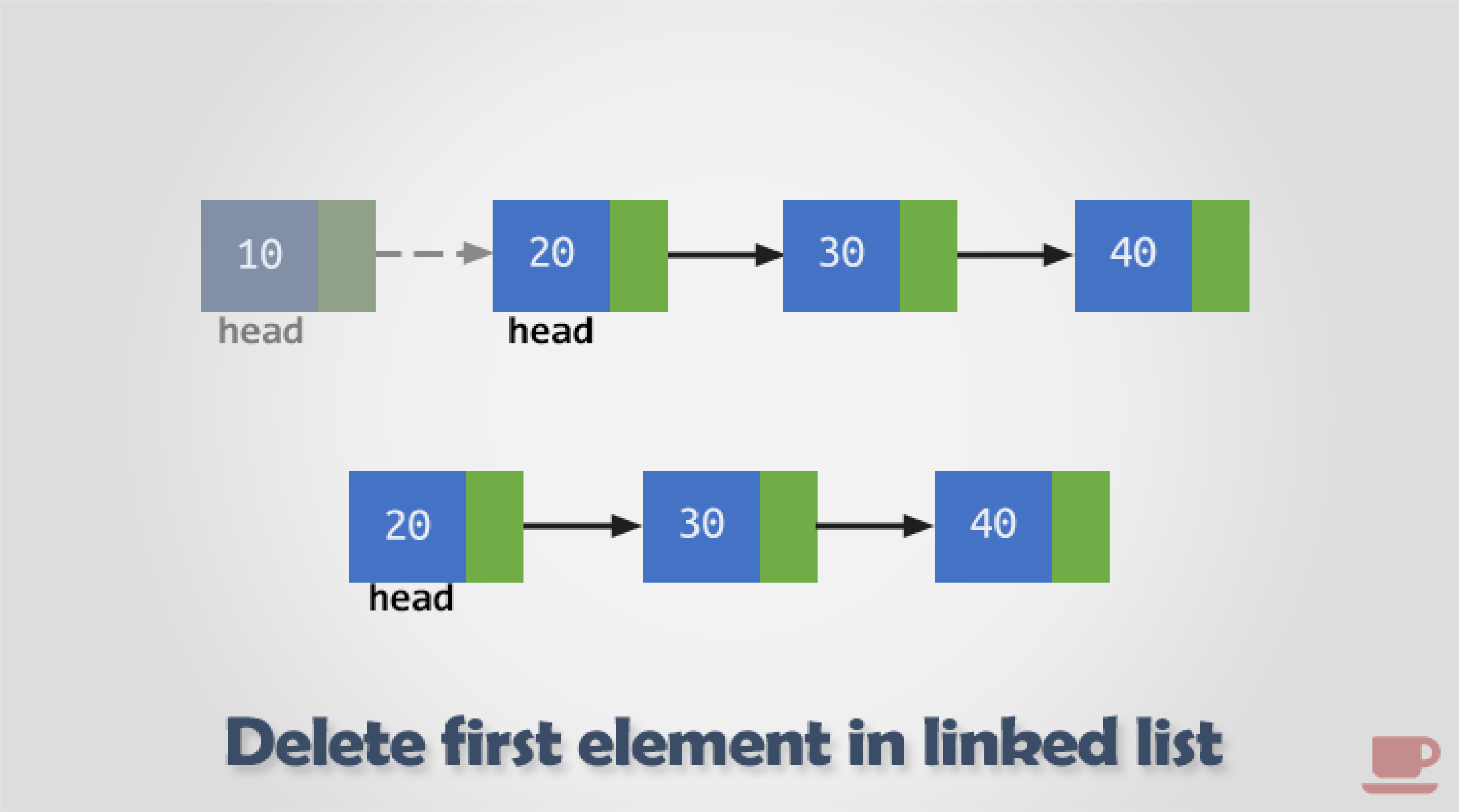
Deletion in a Single Linked List

- There are three possible positions where we can enter a new node in a linked list –
  - **Deletion at beginning**
  - **Deletion at end**
  - **Deletion from given position**

- Deleting new node in linked list is a more than one step activity.

Deletion in Single Linked List (from beginning)

- **Deletion from beginning**



Delete first element in linked list

Deletion in Single Linked List (from beginning)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
    self.data = data
    self.next = None


# A Linked List class with a single
      head node
class LinkedList:
  def __init__(self):
    self.head = None
```

```
# create method for the linked list
  def create(self, data):
    newNode = Node(data)
    if(self.head):
      current = self.head
      while(current.next):
        current = current.next
      current.next = newNode
    else:
      self.head = newNode
```

Deletion in Single Linked List (from beginning)

```
#Delete first node of the list
def del_beg(self):
    if(self.head == None):
        print("Underflow-Link List is empty")

    else:
        temp = self.head
        self.head = self.head.next
        print("the deleted element is", temp.data)
        temp = None
```

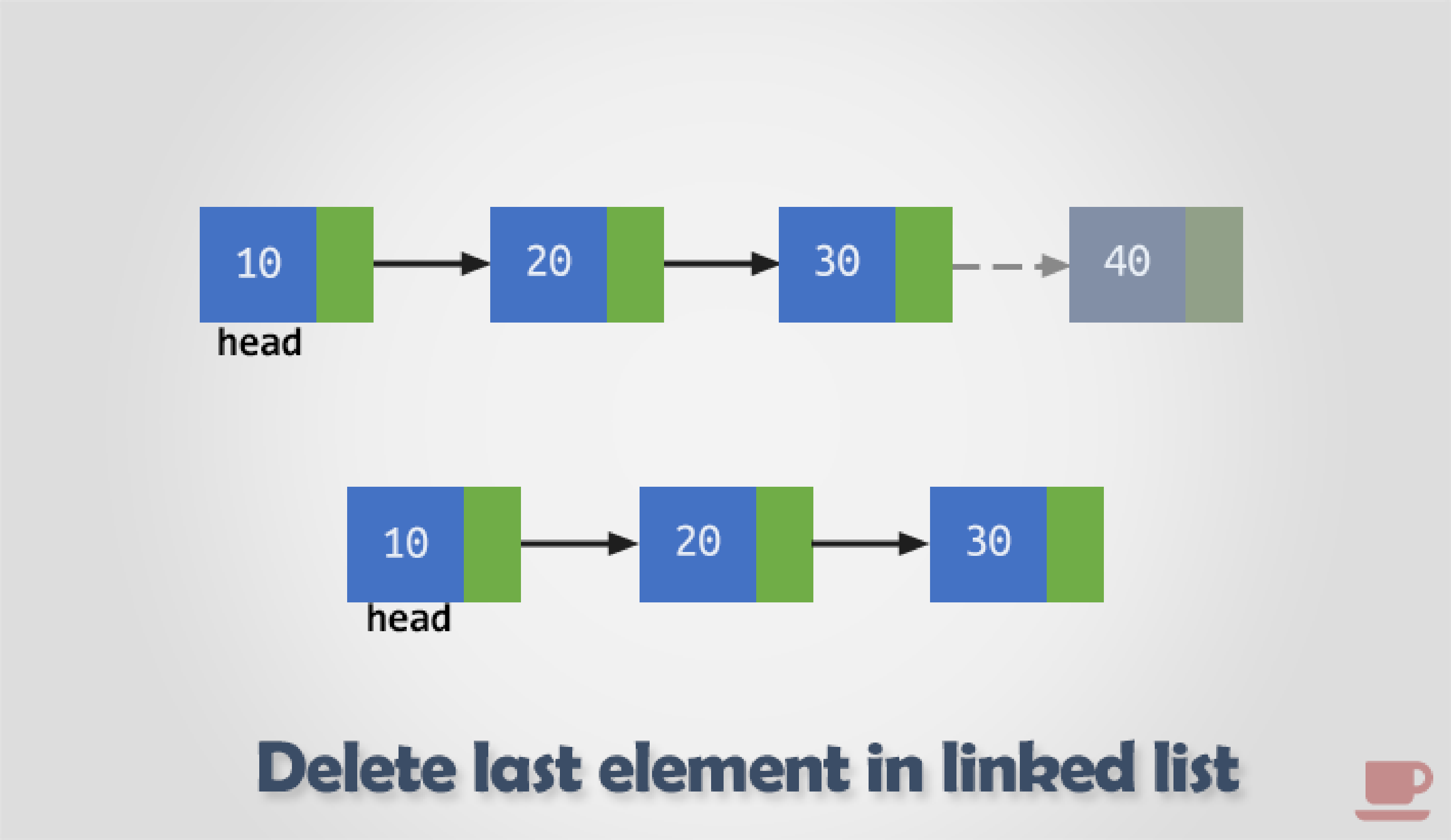
```
# print method for the linked list
def printLL(self):
    current = self.head
    if(current!=None):
        print("The List Contains:",end="\n")
        while(current):
            print(current.data)
            current = current.next
    else:
        print("List is Empty.")
```

Deletion in Single Linked List (from beginning)

```
# Singly Linked List with deletion and print methods
LL = LinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.printLL()
LL.del_beg()
LL.printLL()
```

Deletion in Single Linked List (from end)

- **Deletion from end**



Delete last element in linked list

Deletion in Single Linked List (from end)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
  self.data = data
  self.next = None


# A Linked List class with a single
   head node
class LinkedList:
 def __init__(self):
  self.head = None
```

```
# create method for the linked list
 def create(self, data):
  newNode = Node(data)
  if(self.head):
   current = self.head
   while(current.next):
    current = current.next
   current.next = newNode
  else:
   self.head = newNode
```

Deletion in Single Linked List (from end)

```
#Delete last node of the list
 def del_end(self):
  if(self.head == None):
      print("Underflow-Link List is
    empty")

  else:
    temp = self.head
    while(temp.next!=None):
        prev=temp
        temp=temp.next
    prev.next=None
    print("The deleted element is",
    temp.data)
    temp = None
```

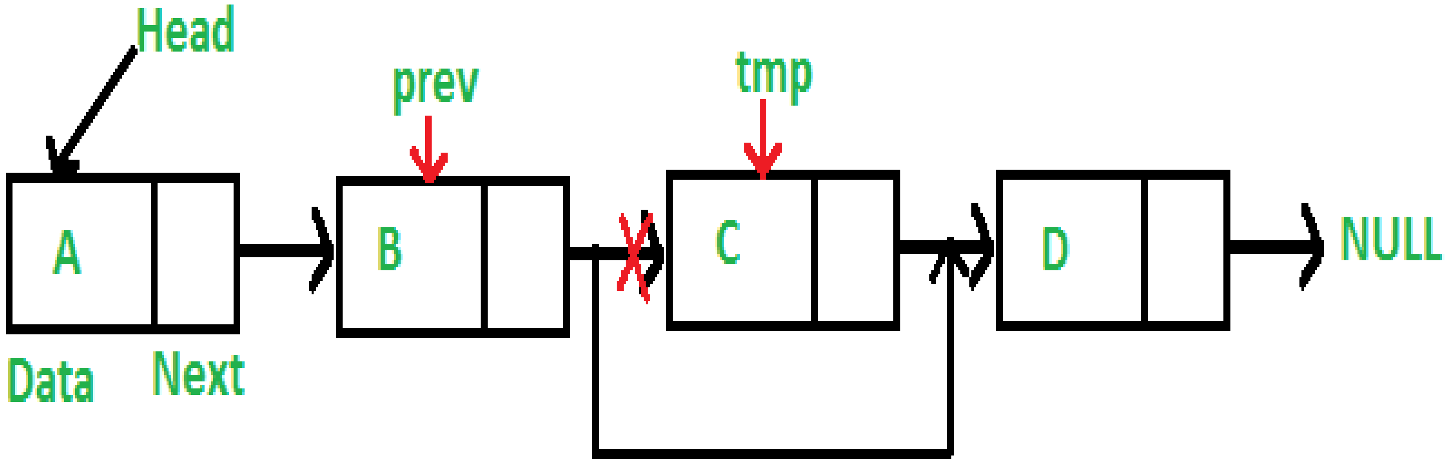
```
# print method for the linked list
  def printLL(self):
    current = self.head
    if(current!=None):
        print("The List Contains:",end="\n")
        while(current):
            print(current.data)
            current = current.next
    else:
        print("List is Empty.")
```

Deletion in Single Linked List (from end)

```
# Singly Linked List with deletion and print methods
LL = LinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.printLL()
LL.del_end()
LL.printLL()
```

Deletion in Single Linked List (from position)

- **Deletion from position**



Deletion in Single Linked List (from position)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
  self.data = data
  self.next = None
```

```
# A Linked List class with a single head node
class LinkedList:
 def __init__(self):
  self.head = None
```

```
# create method for the linked list
 def create(self, data):
  newNode = Node(data)
  if(self.head):
   current = self.head
   while(current.next):
    current = current.next
   current.next = newNode
  else:
   self.head = newNode
```

Deletion in Single Linked List (from position)

```
# Deletion method from the linked list at
   given position
 def del_position(self, pos):
  if(pos<1):
    print("\nPosition should be >=1.")

  elif(pos==1):
    temp = self.head
    self.head = self.head.next
    print("the deleted element is",
   temp.data)
    temp = None
```

```
else:
  temp=self.head
  for i in range(1, pos):
    if(temp!=None):
      prev=temp
      temp=temp.next

  if(temp!=None):
      prev.next=temp.next
      print("the deleted element
      is", temp.data)
      temp=None
  else:
    print("\nThe position does not
      exist in link list.")
```

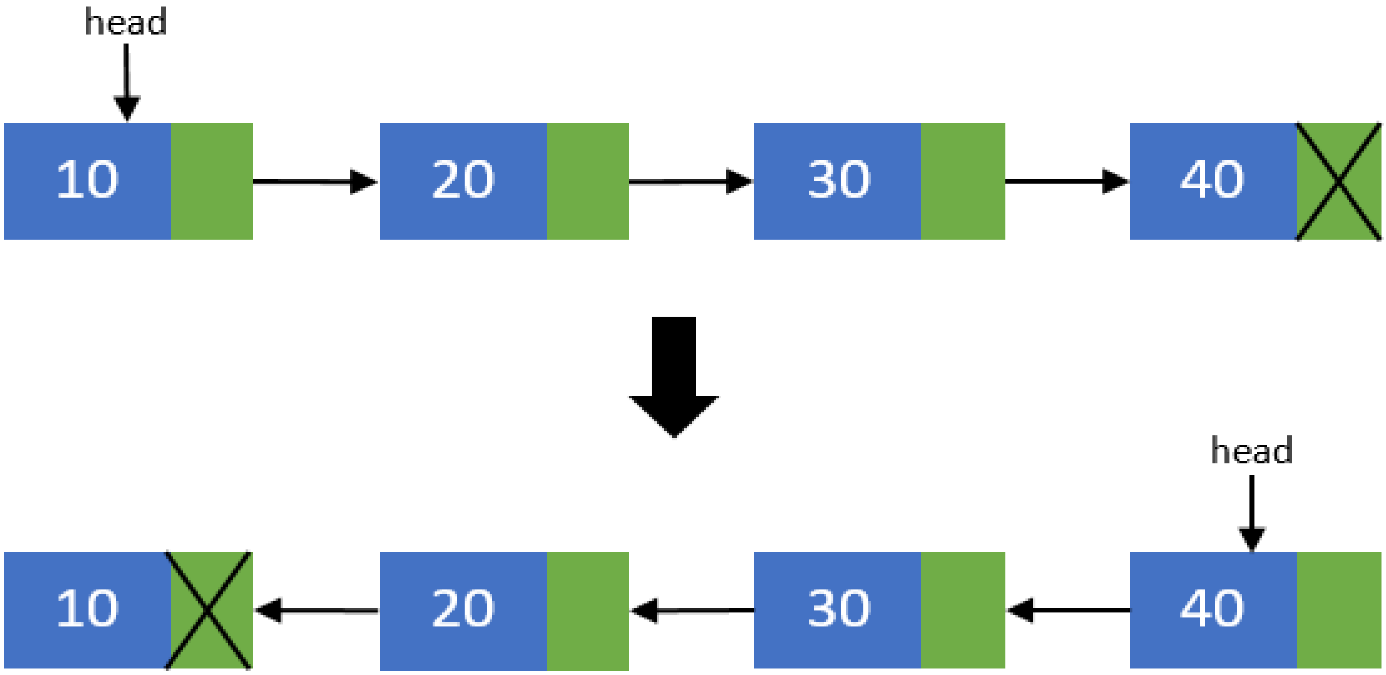
Deletion in Single Linked List (from position)

```
# print method for the linked list
def printLL(self):
  current = self.head
  if(current!=None):
      print("The List
    Contains:",end="\n")
      while(current):
          print(current.data)
          current = current.next
  else:
      print("List is Empty.")
```

```
# Singly Linked List with deletion and print methods
LL = LinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.create(7)
LL.create(8)
LL.printLL()
LL.del_position(4)
LL.printLL()
```

Reverse of a Single Linked List

If the linked list has two or more elements, we can use three pointers to implement an iterative solution..



Reverse of a Single Linked List

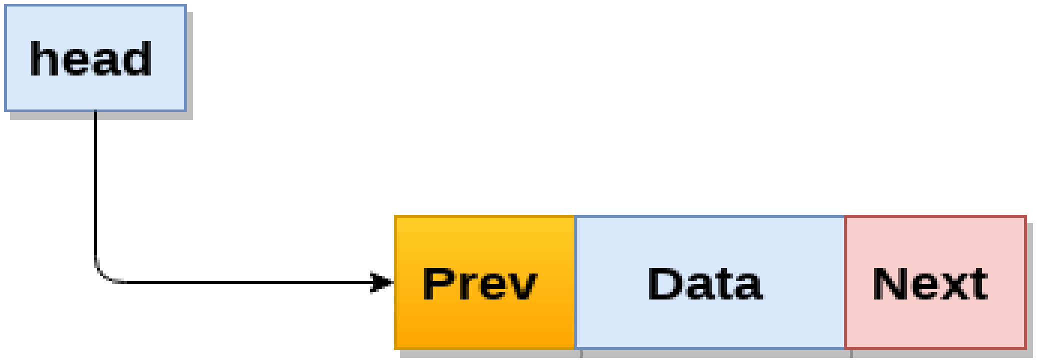
```
# Method to Reverse the linked list
def reverse(self):
    if(self.head==None):
        print("List is Empty.")

    elif(self.head.next==None):
        print("Only one node is present in list")
    else:
        temp1 = self.head
        temp2=temp1.next
        temp3=temp2.next
        temp1.next=None
        while(temp3!=None):
            temp2.next=temp1
            temp1=temp2
            temp2=temp3
            temp3=temp3.next

        temp2.next=temp1
        self.head=temp2
```

Doubly Linked List

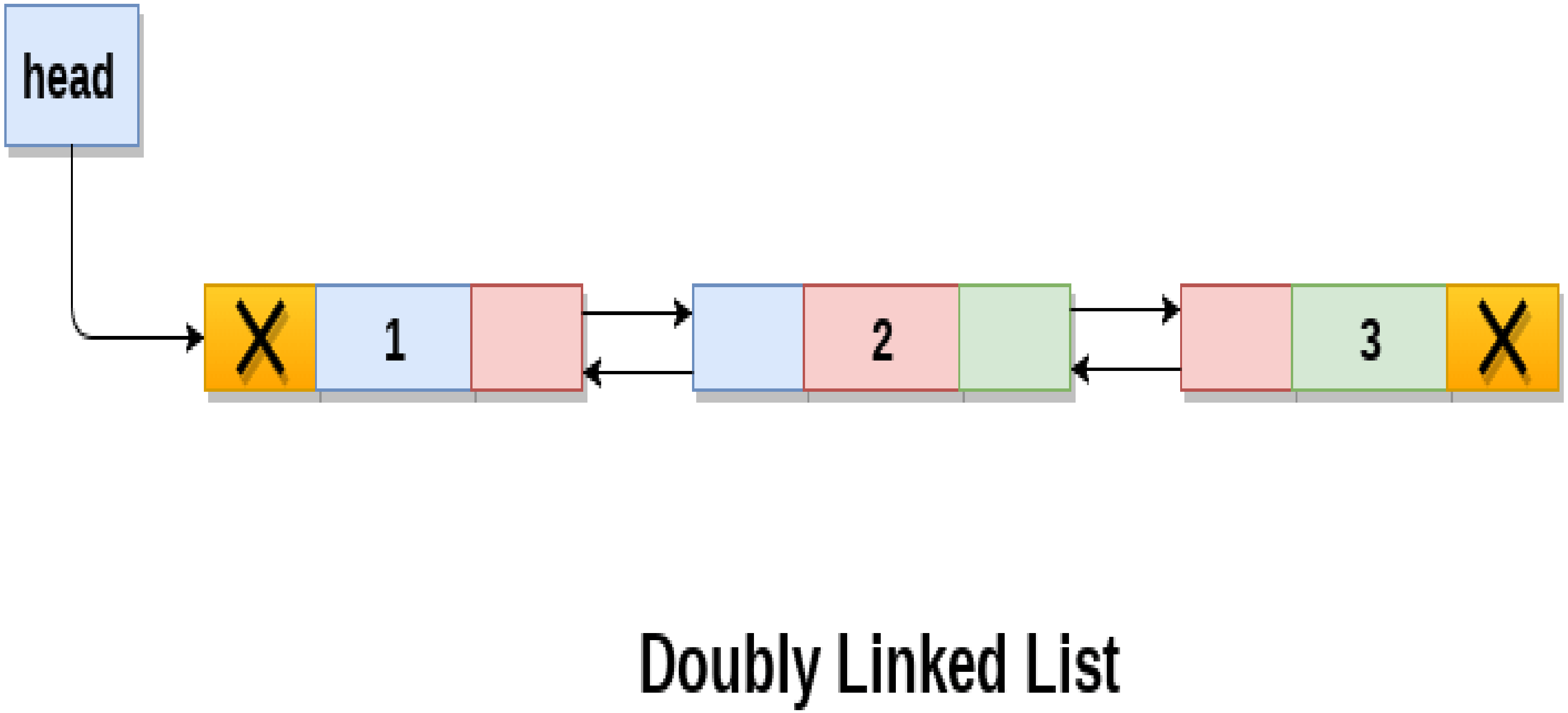
- Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence.
- Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).
- A sample node in a doubly linked list is shown in the figure.



Node

Doubly Linked List

A doubly linked list containing three nodes is shown in the following image.



Creating a Node of Doubly Linked List

- Create a class for creating a node in a doubly linked list, with three attributes: the data, previous pointer and next pointer. The code looks like this:

```
class Node:
    def __init__(self, data):
        self.prev = None
        self.item = data
        self.next = None
```

Creating Doubly Linked List Class

Create a doublyLinkedList class, that contains different functions to insert, delete and display elements of doubly linked list.

```
class doublyLinkedList:
    def __init__(self):
        self.start_node = None
```

Creating a Doubly linked list with single node

```
class Node:
    def __init__(self, data):
        self,prev=None
        self.data = data
        self.next = None

class DoublyLinkedList:
    def __init__(self):
       self.head = None

LL = DoublyLinkedList()
LL.head = Node(3)
print(LL.head.data)
```

Creation and Traversal of Doubly linked list

```
# A single node of a doubly linked list
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None


# A Linked List class with a single head node
class DoublyLinkedList:
    def __init__(self):
        self.head = None
```

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

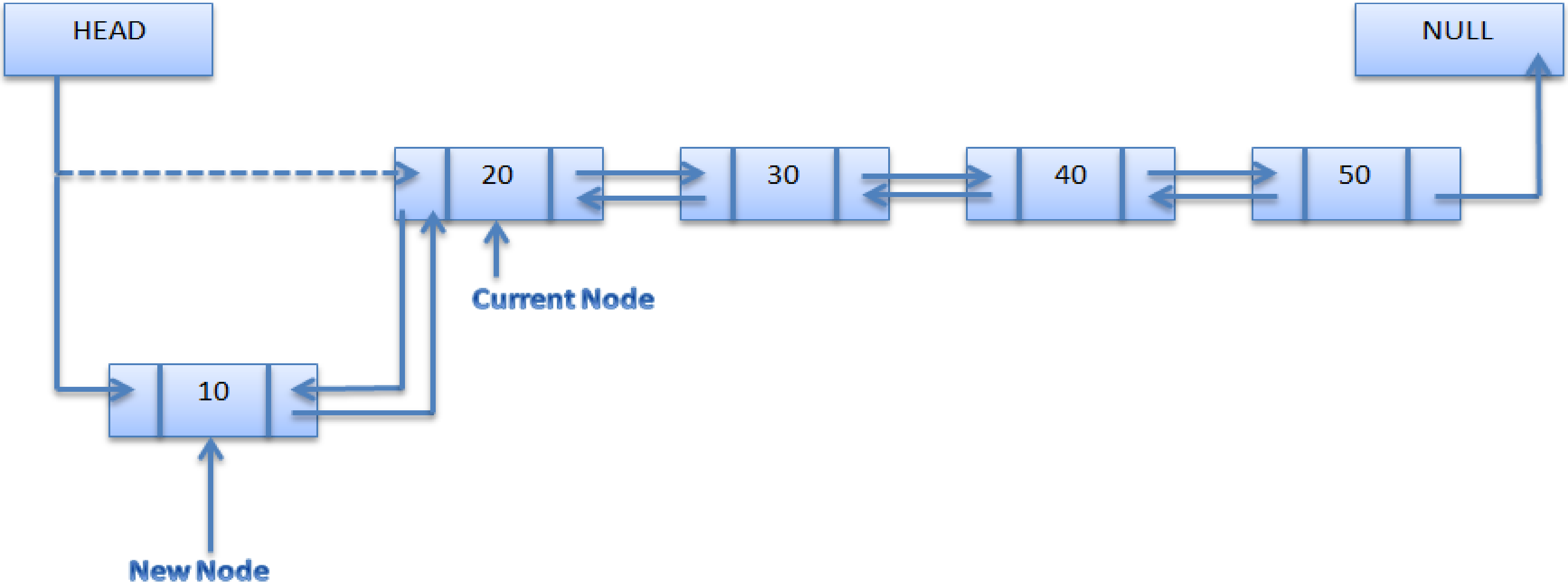
Creation and Traversal of doubly linked list (contd..)

```
# print method for the linked list
 def printLL(self):
    current = self.head
    if(current!=None):
       print("The List Contains:",end="\n")
       while(current!=None):
          print(current.data)
          current = current.next
    else:
       print("List is Empty.")
```

```
# Singly Linked List with creation and
     print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.printLL()
```

Insertion at the Beginning Linked List

- Insertion at the Beginning of Doubly Linked List



Insertion at Beginning in Doubly linked list

```
# A single node of a doubly linked list
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None

# A Linked List class with a single head
    node
class DoublyLinkedList:
    def __init__(self):
        self.head = None
```

```
# Insertion method for the doubly
    linked list at  beginning
    def insert_beg(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            newNode.next=self.head
            self.head.prev=newNode
            self.head=newNode
```

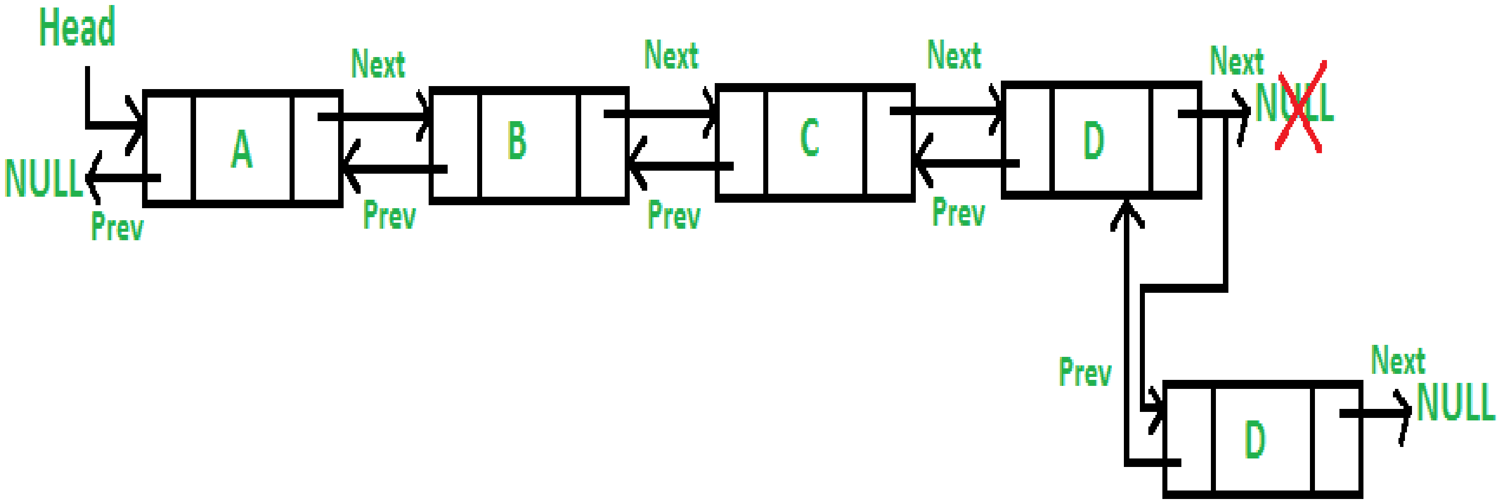
Insertion at Beginning in Doubly linked list (contd..)

```
# print method for the linked list
  def printLL(self):
    current = self.head
    if(current!=None):
      print("The List Contains:",end="\n")
      while(current!=None):
        print(current.data)
        current = current.next
    else:
      print("List is Empty.")
```

```
# Singly Linked List with creation and
    print methods
LL = DoublyLinkedList()
LL.insert_beg(6)
LL.insert_beg(5)
LL.insert_beg(4)
LL.insert_beg(3)
LL.printLL()
```

Insertion at the end of Doubly Linked List

- Insertion at the end of Doubly Linked List



Insertion at end in Doubly linked list

```
# A single node of a doubly linked list
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None


# A Linked List class with a single head node
class DoublyLinkedList:
    def __init__(self):
        self.head = None
```

```
# Insertion method for the doubly linked list at end
def insert_end(self, data):
    newNode = Node(data)
    if(self.head==None):
        self.head = newNode

    else:
        temp=self.head
        while(temp.next!=None):
            temp=temp.next

        temp.next=newNode
        newNode.prev=temp
```

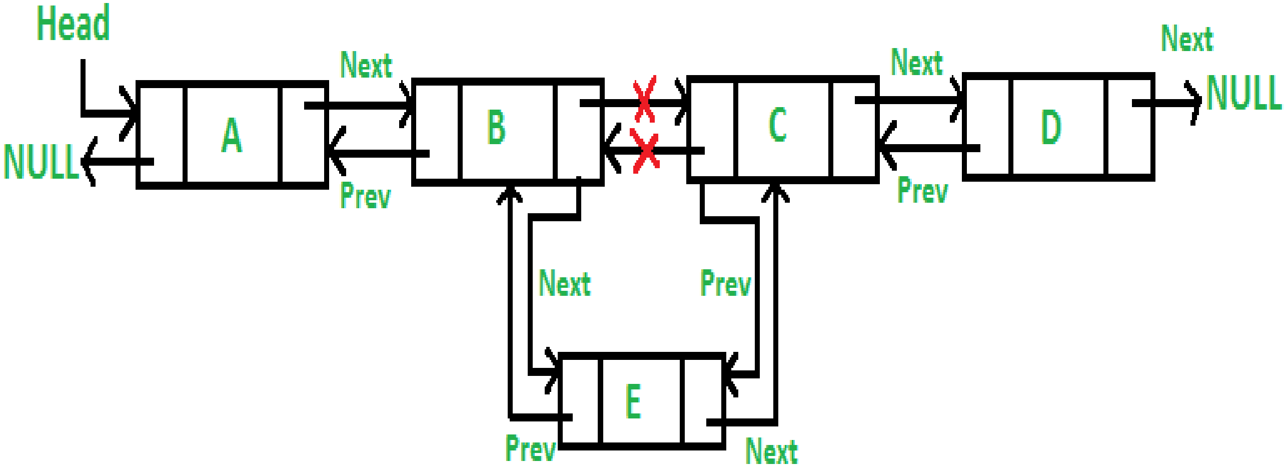
Insertion at end in Doubly linked list (contd..)

```
# print method for the linked list
  def printLL(self):
    current = self.head
    if(current!=None):
      print("The List Contains:",end="\n")
      while(current!=None):
        print(current.data)
        current = current.next
    else:
      print("List is Empty.")
```

```
# Singly Linked List with creation and
      print methods
LL = DoublyLinkedList()
LL.insert_end(3)
LL.insert_end(4)
LL.insert_end(5)
LL.insert_end(6)
LL.printLL()
```

Insertion in Doubly Linked List (at position)

- Insertion at given position in Doubly Linked List



Insertion in Doubly Linked List (at position)

```
# A single node of a doubly linked list
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None


# A Linked List class with a single head node
class DoublyLinkedList:
    def __init__(self):
        self.head = None
```

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Insertion in Doubly Linked List (at position) (contd..)

```
# insertion method for the doubly
  linked list at given position

def insert_position(self, data, pos):
  newNode = Node(data)
  if(pos<1):
    print("\nPosition should be >=1.")

  elif(pos==1):
    newNode.next=self.head
    self.head=newNode
  else:
    current=self.head
    for i in range(1, pos-1):
      if(current!=None):
        current=current.next

    if(current!=None):
      newNode.next=current.next
      current.next.prev=newNode
      current.next=newNode
      newNode.prev=current

    else:
      print("\nThe previous node is null.")
```

Insertion in Doubly Linked List (at position) (contd..)

```
# print method for the linked list
  def printLL(self):
     current = self.head
     if(current!=None):
        print("The List Contains:",end="\n")
        while(current!=None):
           print(current.data)
           current = current.next
     else:
        print("List is Empty.")
```

```
# Singly Linked List with creation and
     print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.create(7)
LL.printLL()
LL.insert_position(4, 9)
LL.printLL()
```

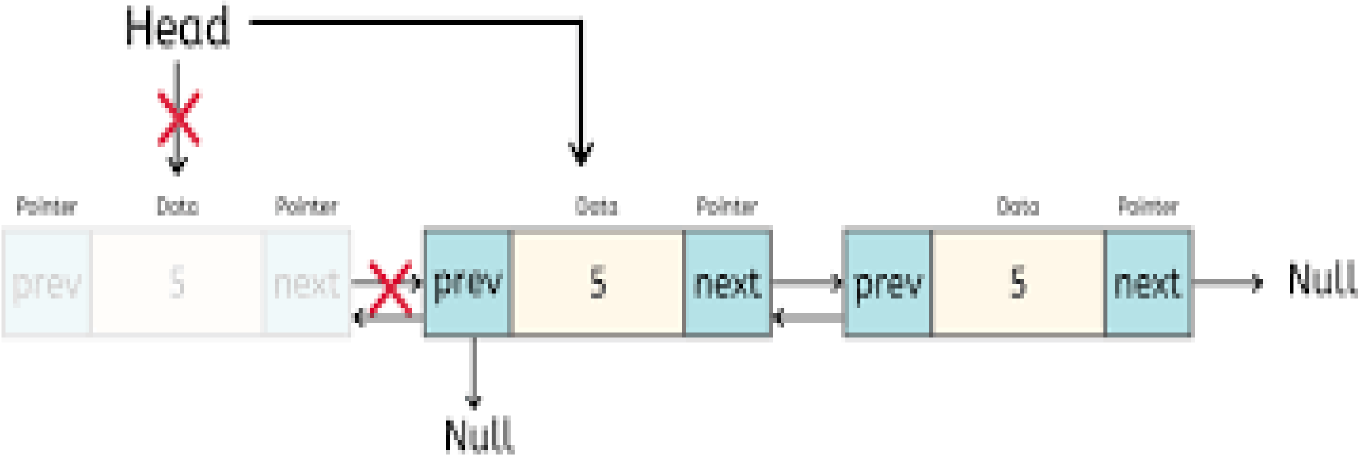
Deletion in a Doubly Linked List

- Similar to single linked list there are three possible positions where we can enter a new node in a doubly linked list –
  - **Deletion at beginning**
  - **Deletion at end**
  - **Deletion from given position**

- Deleting new node in linked list is a more than one step activity.

Deletion in Doubly Linked List (from beginning)

- **Deletion from beginning**



Deletion in Doubly Linked List (from beginning)

```
# A single node of a doubly linked list
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None

# A Linked List class with a single head node
class DoublyLinkedList:
    def __init__(self):
        self.head = None
```

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Deletion in Doubly Linked List (from beginning) (contd..)

```
#Delete first node of the list
 def del_beg(self):
    if(self.head == None):
       print("Underflow-Link List is empty")
    else:
       temp = self.head
       self.head = self.head.next
       self.head.prev=None
       print("the deleted element is", temp.data)
       temp = None
```

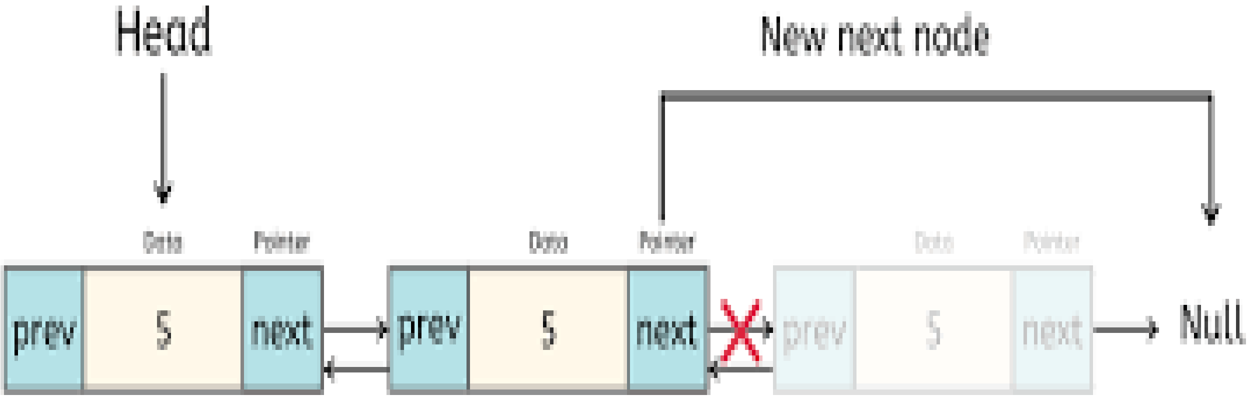
```
# print method for the linked list
   def printLL(self):
      current = self.head
      if(current!=None):
         print("The List
     Contains:",end="\n")
         while(current!=None):
            print(current.data)
            current = current.next
      else:
         print("List is Empty.")
```

Deletion in Doubly Linked List (from beginning) (contd..)

```
# Doubly Linked List with creation, deletion and print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.printLL()
LL.del_beg()
LL.printLL()
```

Deletion in Doubly Linked List (from end)

- **Deletion from end**



Deletion in Doubly Linked List (from end)

```
# A single node of a doubly linked list
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None

# A Linked List class with a single head node
class DoublyLinkedList:
    def __init__(self):
        self.head = None
```

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Deletion in Doubly Linked List (from end) (contd..)

#Delete last node of the list

```
def del_end(self):
    if(self.head == None):
        print("Underflow-Link List is empty")

    else:
      temp = self.head
      while(temp.next!=None):
          prev=temp
          temp=temp.next

      prev.next=None
      print("The deleted element is", temp.data)
      temp = None
```

# print method for the linked list

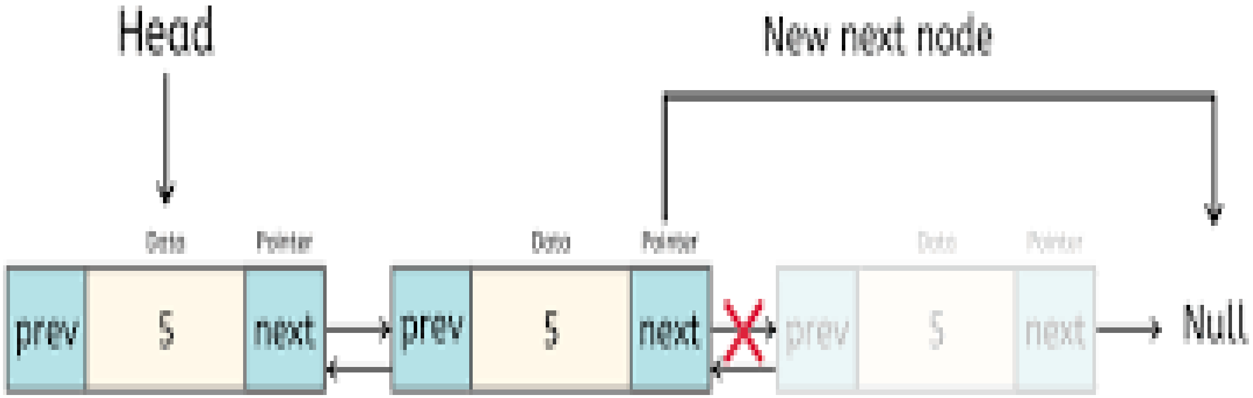
```
def printLL(self):
    current = self.head
    if(current!=None):
        print("The List
  Contains:",end="\n")
        while(current!=None):
            print(current.data)
            current = current.next
    else:
        print("List is Empty.")
```

Deletion in Doubly Linked List (from end) (contd..)

```
# Doubly Linked List with creation, deletion and print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.printLL()
LL.del_end()
LL.printLL()
```

Deletion in Doubly Linked List (from position)

- **Deletion from end**



Deletion in Doubly Linked List (from position)

```
# A single node of a doubly linked list
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None


# A Linked List class with a single head node
class DoublyLinkedList:
    def __init__(self):
        self.head = None
```

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Deletion in Doubly Linked List (from position) (contd..)

```
# Deletion method from the linked list at given position
def del_position(self, pos):
  if(pos<1):
    print("\nPosition should be >=1.")

  else:
    temp=self.head
    for i in range(1, pos):
      if(temp!=None):
        current=temp
        temp=temp.next
    if(temp!=None):
      current.next=temp.next
      temp.next.prev=current
      print("the deleted element is", temp.data)
      temp=None

    else:
      print("\nThe position does not exist in link list.")
```

Deletion in Doubly Linked List (from position) (contd..)

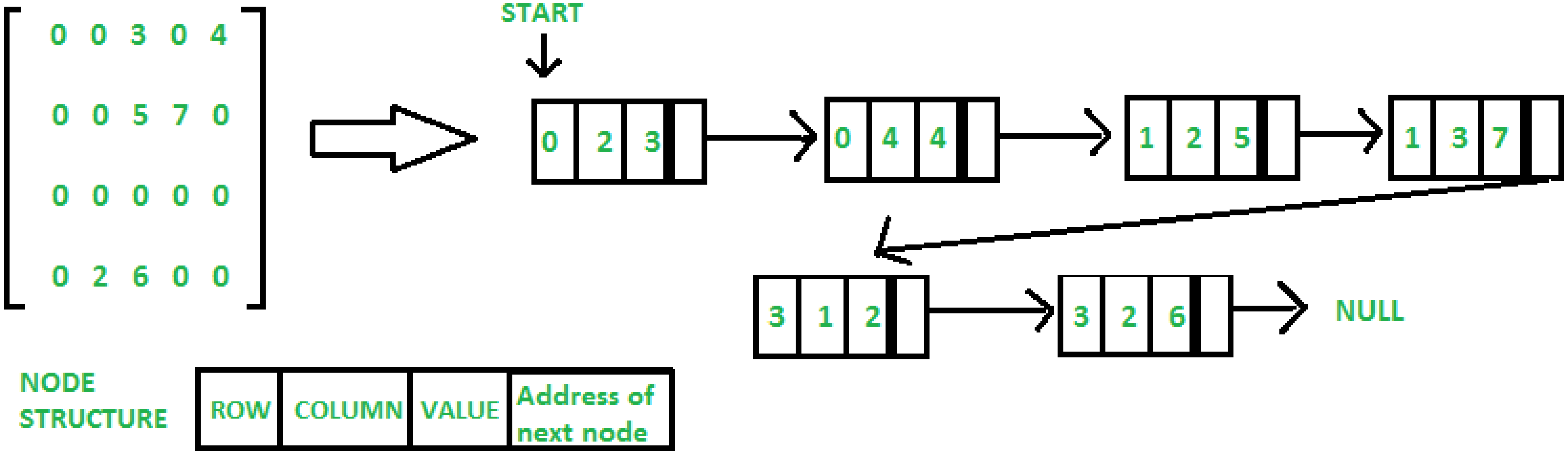
```
# print method for the linked list
  def printLL(self):
    current = self.head
    if(current!=None):
      print("The List Contains:",end="\n")
      while(current!=None):
        print(current.data)
        current = current.next
    else:
      print("List is Empty.")
```

```
# Doubly Linked List with creation, deletion and print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.create(7)
LL.create(8)
LL.printLL()
LL.del_position(4)
LL.printLL()
```

Linked Representation of Sparse Matrix

In linked list, each node has four fields. These four fields are defined as:

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row , column)
- **Next node:** Address of the next node



$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Polynomial Representation

Polynomials

Polynomials are the algebraic expressions which consist of exponents and coefficients.

Example -

$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 is its exponential value.

Polynomial can be represented in the various ways. These are:

- By the use of arrays
- By the use of Linked List

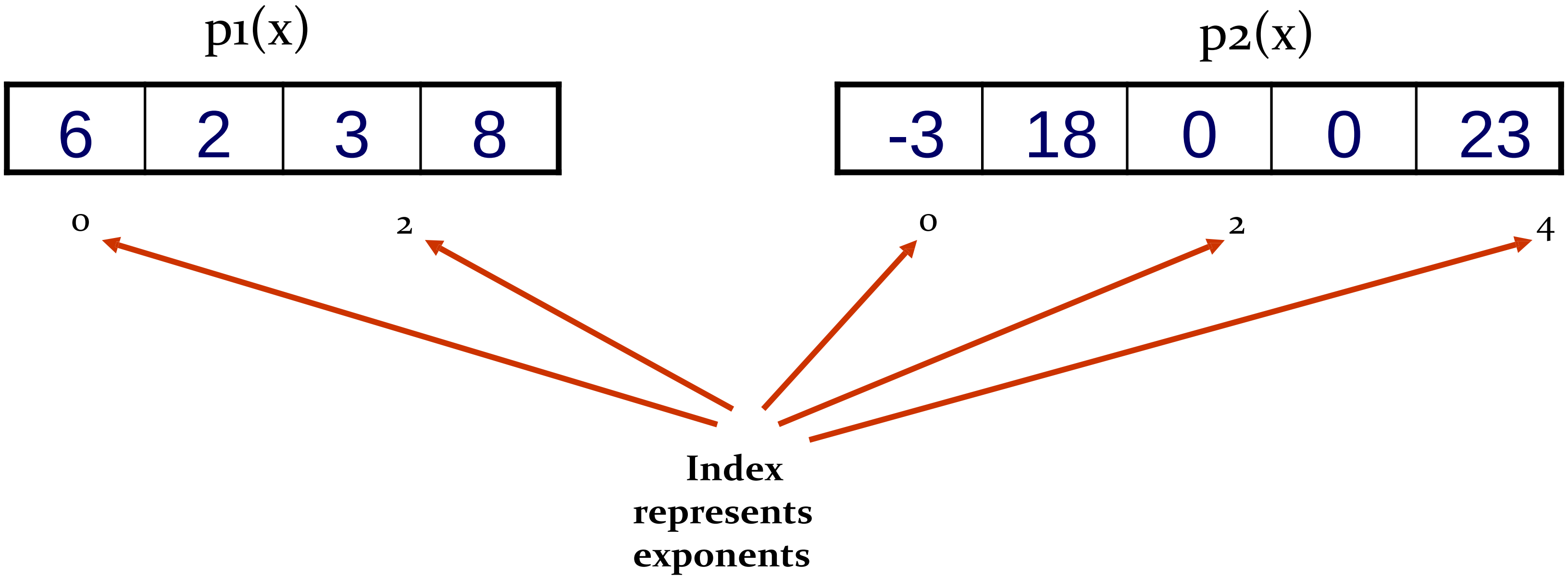
Polynomial Representation

Polynomial can be represented

- By the use of arrays
- By the use of Linked List

Polynomial (Array Representation)

- Array Representation:
- $p1(x) = 8x^3 + 3x^2 + 2x + 6$
- $p2(x) = 23x^4 + 18x - 3$



p1(x)

6	2	3	8

p2(x)

-3	18	0	0	23

Index represents exponents

Polynomial (Array Representation)

- This is why arrays aren't good to represent polynomials:

- $p3(x) = 16x^{21} - 3x^{5} + 2x + 6$

6	2	0	0	-3	0	…………	0	16

WASTE OF SPACE!

Add Two Polynomials Using Arrays

Input: A[] = {5, 0, 10, 6} B[] = {1, 2, 4}
Output: sum[] = {6, 2, 14, 6}

The first input array represents "5 + 0x^1 + 10x^2 + 6x^3“

The second array represents "1 + 2x^1 + 4x^2"

And Output is "6 + 2x^1 + 14x^2 + 6x^3“

Polynomial (Array Representation)

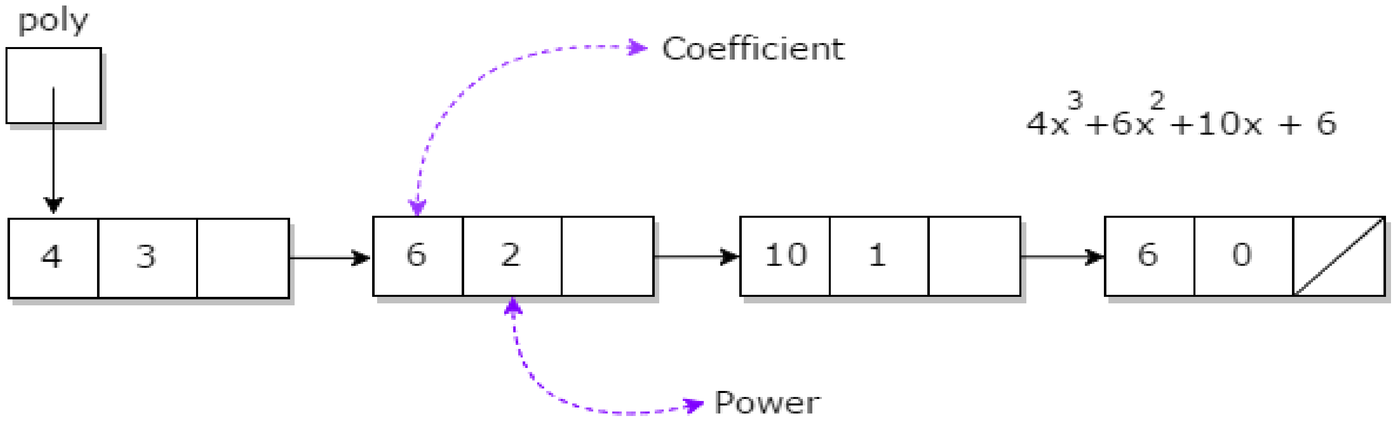
- Advantages of using an Array:
  - Only good for non-sparse polynomials.
  - Ease of storage and retrieval.
- Disadvantages of using an Array:
  - Have to allocate array size priorly.
  - Huge array size required for sparse polynomials. Waste of space and runtime.

Polynomial Representation (using Linked List)

- A polynomial p(x) is the expression in variable x which is in the form (ax^n + bx^{n-1} + …. + jx+ k), where a, b, c …., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.
- An essential characteristic of the polynomial is that each term in the polynomial expression consists of two parts:
  - one is the coefficient
  - other is the exponent
- <u>Example:</u>
  - $10x^2 + 26x$,
    - here 10 and 26 are coefficients and 2, 1 is its exponential value.

Polynomial Representation (using Linked List)

- Points to keep in Mind while working with Polynomials:
- The sign of each coefficient and exponent is stored within the coefficient and the exponent itself
- Additional terms having equal exponent is possible one
- The storage allocation for each term in the polynomial must be done in ascending and descending order of their exponent



Polynomial Representation (using Linked List)

- Addition of polynomial

1) Input: 1st number = $5x^2 + 4x^1 + 2x^0$

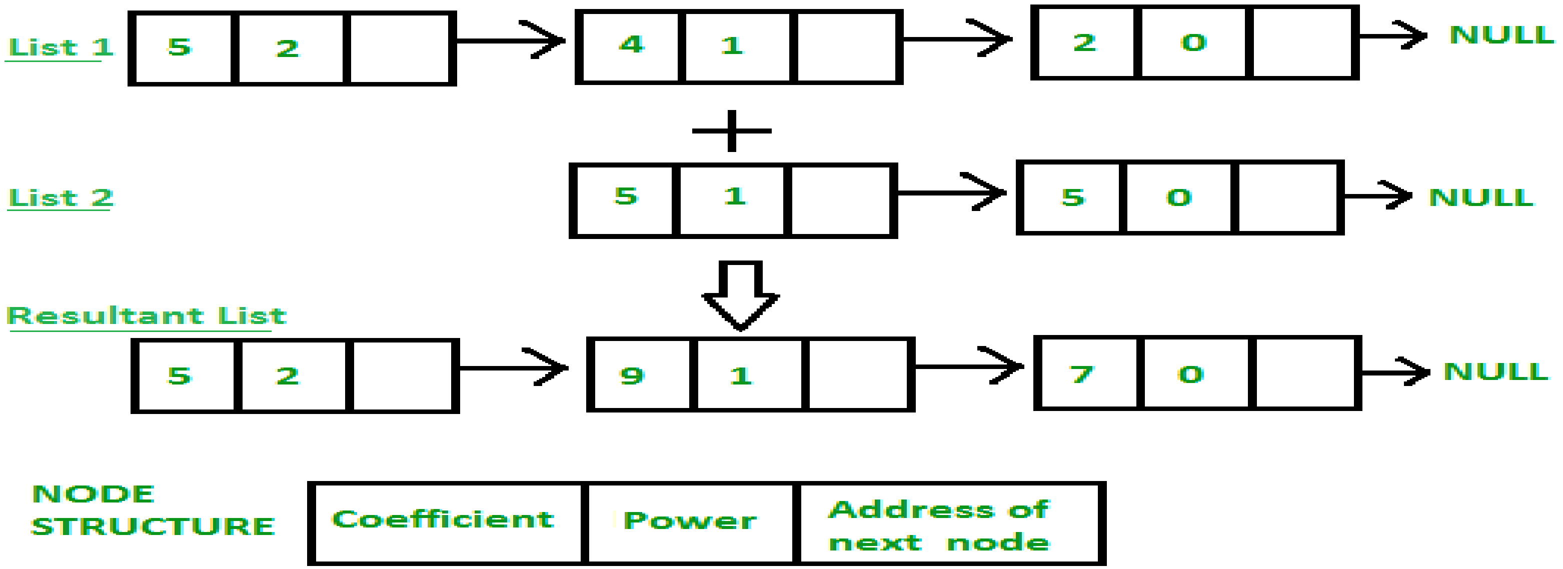
2nd number = $-5x^1 - 5x^0$

Output: $5x^2-1x^1-3x^0$

2) Input: 1st number = $5x^3 + 4x^2 + 2x^0$

2nd number = 5x^1 - 5x^0

Output: $5x^3 + 4x^2 + 5x^1 - 3x^0$



Polynomial Representation (using Linked List)

Subtraction of polynomial

Input: 1st number = $5x^2 + 4x^1 + 2x^0$

2nd number = $-5x^1 - 5x^0$

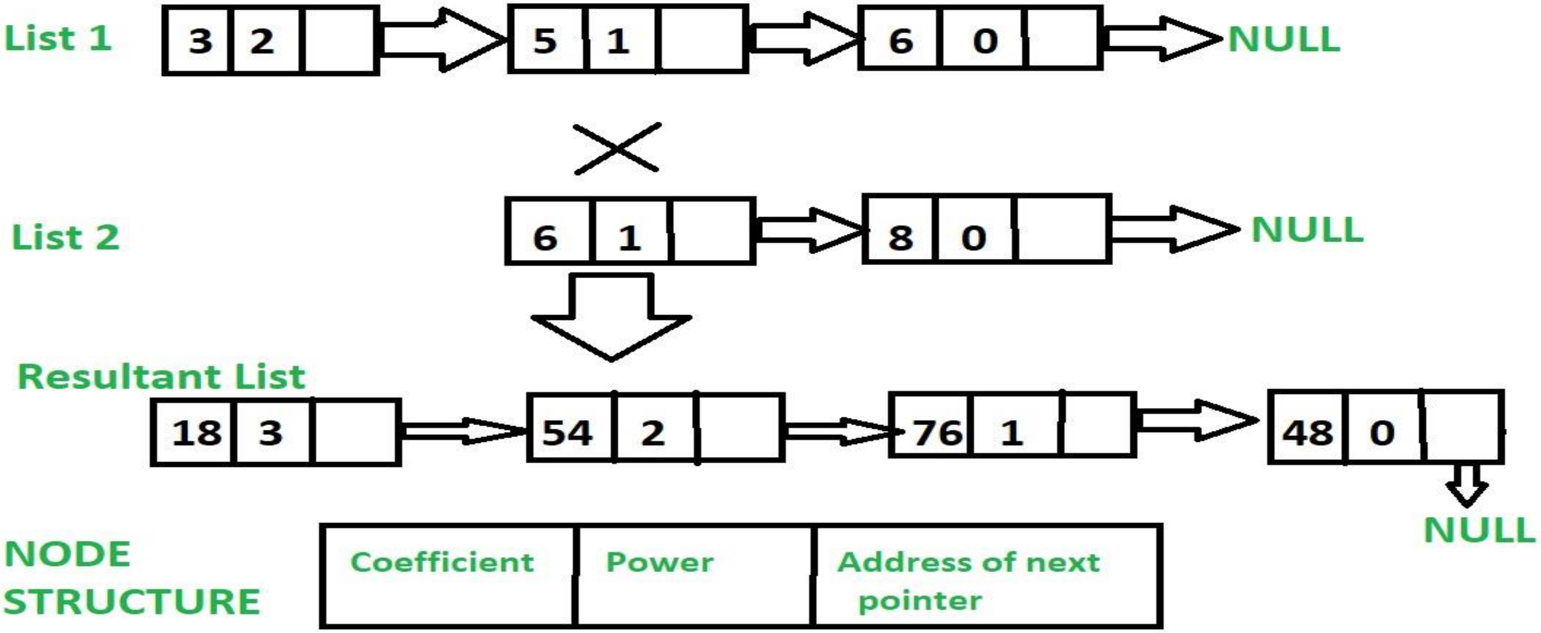
Output: $5x^2 + 9x^1 + 7x^0$

Polynomial Representation (using Linked List)

Multiplication of polynomial

Input: Poly1: 3x^2 + 5x^1 + 6, Poly2: 6x^1 + 8

Output: 18x^3 + 54x^2 + 76x^1 + 48



Stack Implementation using Linked List

- Using List as Stack and Queues in Python (tutorialspoint.com)

Queue Implementation using Linked List

- Using List as Stack and Queues in Python (tutorialspoint.com)

Daily Quiz

- Which of the following is not a disadvantage to the usage of array?

  a) Fixed size

  b) There are chances of wastage of memory space if elements inserted in an array are lesser than the allocated size

  c) Insertion based on position

  d) Accessing elements at specified positions

- What is the time complexity to count the number of elements in the linked list?

  a) O(1)

  b) O(n)

  c) O(logn)

  d) O(n2)

Daily Quiz

- What is the space complexity for deleting a linked list?

  a) O(1)

  b) O(n)

  c) Either O(1) or O(n)

  d) O(logn)

- What differentiates a circular linked list from a normal linked list?

  a) You cannot have the ‘next’ pointer point to null in a circular linked list

  b) It is faster to traverse the circular linked list

  c) You may or may not have the ‘next’ pointer point to null in a circular linked list

  d) Head node is known in circular linked list

Weekly Assignment

- Define data structure and explain types of data structure with suitable examples
- What are the primitive operations in any data structure
- Explain the asymptotic notations with suitable examples
- Differentiate between space and time complexity of an algorithm
- Write an algorithm for merging two already sorted arrays
- Write an algorithm/program for insert a node at between any position of a linked list
- Write an algorithm/program to reverse a linked list
- Write an algorithm/program to sort two linked lists
- Write an algorithm/program to add two polynomials by using linked list
- what are the applications of a single linked list, doubly linked list and circular linked list

MCQ s

- Which of these best describes an array?
  a) A data structure that shows a hierarchical behaviour
  b) Container of objects of similar types
  c) Arrays are immutable once initialised
  d) Array is not a data structure

- How do you initialize an array in C?
  a) int arr[3] = (1,2,3);
  b) int arr(3) = {1,2,3};
  c) int arr[3] = {1,2,3};
  d) int arr(3) = (1,2,3);

MCQ s

In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is (GATE CS 2002)
A log 2 n
B n/2
C log 2 n – 1
D n

You are given pointers to first and last nodes of a singly linked list, which of the following operations are dependent on the length of the linked list?
A Delete the first element
B Insert a new element as a first element
C Delete the last element of the list
D Add a new element at the end of the list

MCQ s

- What are the advantages of arrays?

  a) Objects of mixed data types can be stored
  b) Elements in an array cannot be sorted
  c) Index of first element of an array is 1
  d) Easier to store elements of same data type

- Elements in an array are accessed _______________

  a) randomly
  b) sequentially
  c) exponentially
  d) logarithmically

MCQ s

- In linked list each node contain minimum of two fields. One field is data field to store the data second field is?

  A. Pointer to character

  B. Pointer to integer

  C. Pointer to node

  D. Node

- Linked list data structure offers considerable saving in

  A. Computational Time

  B. Space Utilization

  C. Space Utilization and Computational Time

  D. None of the mentioned

Glossary Question

i. Linked list ii. Abstract Data type iii.Asymptotic Analysis iv. Linear order

Answer the questions.

a. Another term for total order.

b. A process followed to solve a problem.

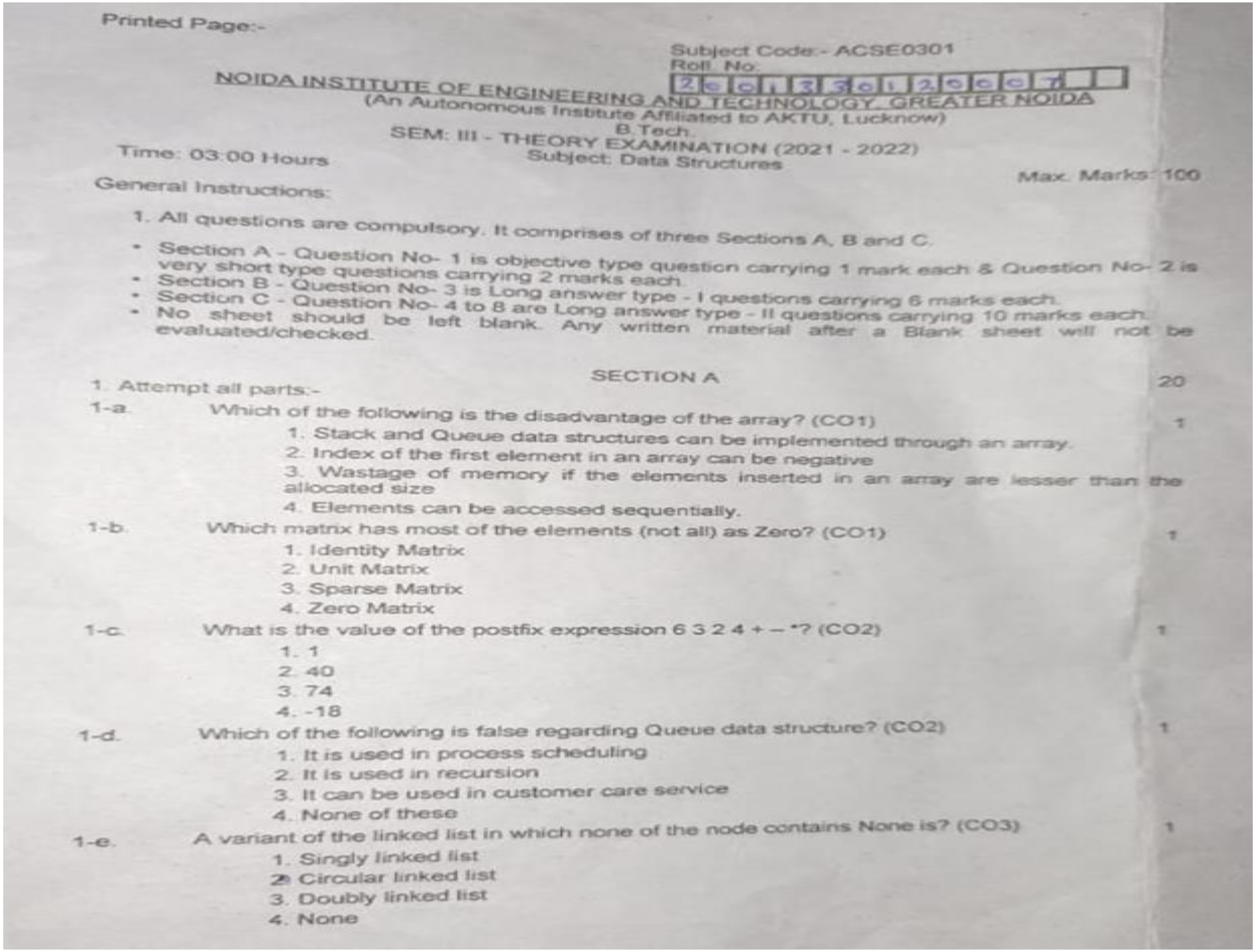
c. The specification of a data type with some language.

d. Dynamic allocation of link nodes to store the list of elements.

Old Question Papers

- http://www.aktuonline.com/papers/btech-cs-3-sem-data-structures-kcs301-2020.html
- http://www.aktuonline.com/papers/btech-cs-3-sem-data-structures-rcs-305-2018-19.html
- http://www.aktuonline.com/papers/btech-cs-3-sem-data-structures-rcs-305-2017-18.html
- http://www.aktuonline.com/papers/btech-cs-3-sem-data-structures-using-c-ncs-301-2016-17.html
- https://www.aktuonline.com/papers/btech-cs-3-sem-data-structures-kcs301-2020.html
- https://firstranker.com/fr/frdA290120A1345373/download-aktu-b-tech-3rd-sem-2018-2019-data-structures-rcs-305-question-paper

Old Question Papers



Printed Page:-

Subject Code:- ACSE0301
Roll No: 2 0 0 1 3 3 0 1 2 0 0 0 7

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA
(An Autonomous Institute Affiliated to AKTU, Lucknow)
B.Tech.
SEM: III - THEORY EXAMINATION (2021 - 2022)
Subject: Data Structures

Time: 03:00 Hours

Max. Marks: 100

General Instructions:

1. All questions are compulsory. It comprises of three Sections A, B and C.

- Section A - Question No- 1 is objective type question carrying 1 mark each & Question No- 2 is very short type questions carrying 2 marks each.
- Section B - Question No- 3 is Long answer type - I questions carrying 6 marks each.
- Section C - Question No- 4 to 8 are Long answer type - II questions carrying 10 marks each.
- No sheet should be left blank. Any written material after a Blank sheet will not be evaluated/checked.

SECTION A

1. Attempt all parts:- 20

1-a. Which of the following is the disadvantage of the array? (CO1) 1

1. Stack and Queue data structures can be implemented through an array.
2. Index of the first element in an array can be negative
3. Wastage of memory if the elements inserted in an array are lesser than the allocated size
4. Elements can be accessed sequentially.

1-b. Which matrix has most of the elements (not all) as Zero? (CO1) 1

1. Identity Matrix
2. Unit Matrix
3. Sparse Matrix
4. Zero Matrix

1-c. What is the value of the postfix expression 6 3 2 4 + – *? (CO2) 1

1. 1
2. 40
3. 74
4. -18

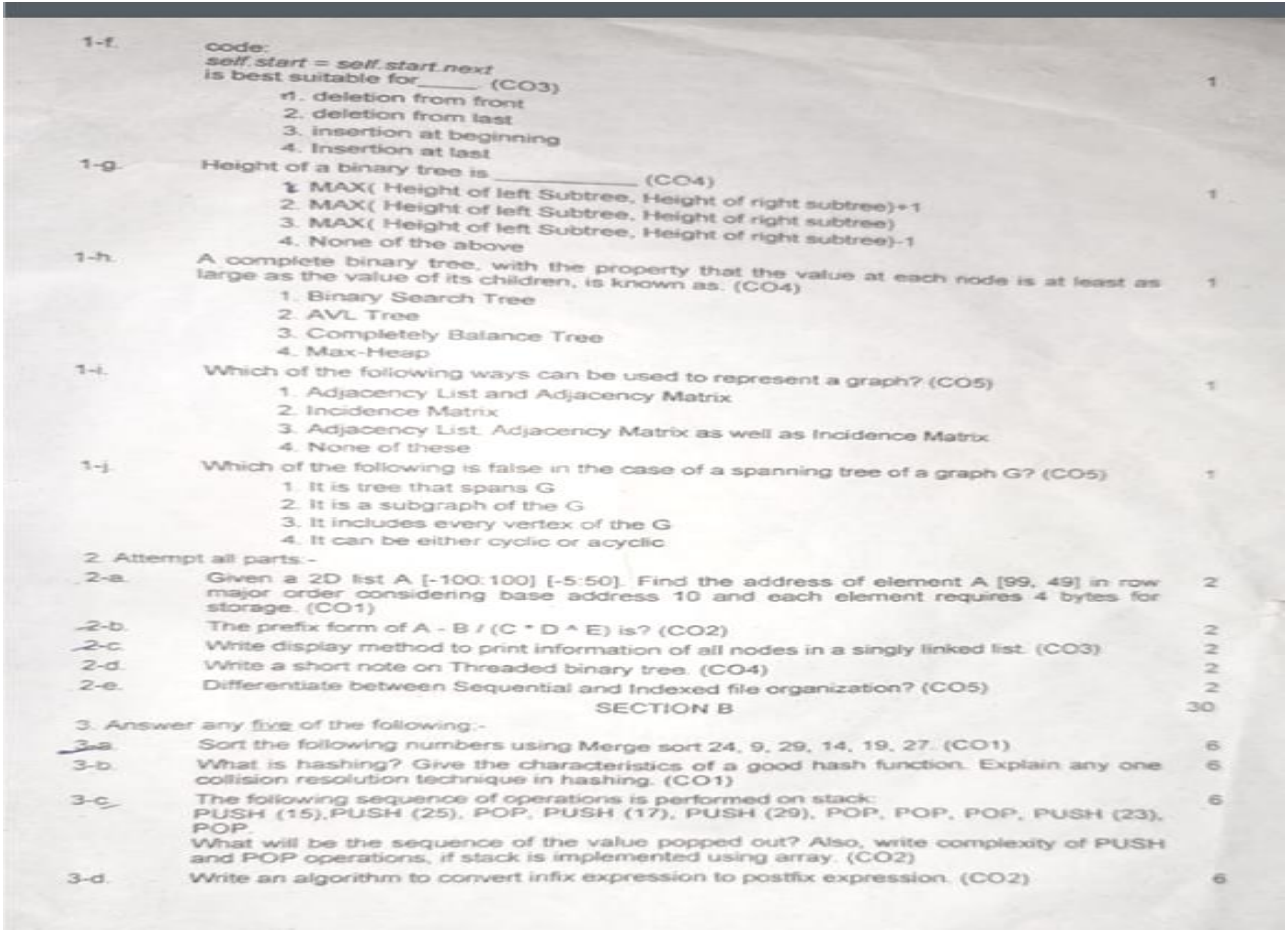
1-d. Which of the following is false regarding Queue data structure? (CO2) 1

1. It is used in process scheduling
2. It is used in recursion
3. It can be used in customer care service
4. None of these

1-e. A variant of the linked list in which none of the node contains None is? (CO3) 1

1. Singly linked list
2. Circular linked list
3. Doubly linked list
4. None

Old Question Papers



1-f. code:
self.start = self.start.next
is best suitable for______ (CO3) 1

1. deletion from front
2. deletion from last
3. insertion at beginning
4. Insertion at last

1-g. Height of a binary tree is ____________ (CO4) 1

1. MAX(Height of left Subtree, Height of right subtree)+1
2. MAX(Height of left Subtree, Height of right subtree)
3. MAX(Height of left Subtree, Height of right subtree)-1
4. None of the above

1-h. A complete binary tree, with the property that the value at each node is at least as large as the value of its children, is known as. (CO4) 1

1. Binary Search Tree
2. AVL Tree
3. Completely Balance Tree
4. Max-Heap

1-i. Which of the following ways can be used to represent a graph? (CO5) 1

1. Adjacency List and Adjacency Matrix
2. Incidence Matrix
3. Adjacency List, Adjacency Matrix as well as Incidence Matrix
4. None of these

1-j. Which of the following is false in the case of a spanning tree of a graph G? (CO5) 1

1. It is tree that spans G
2. It is a subgraph of the G
3. It includes every vertex of the G
4. It can be either cyclic or acyclic

2. Attempt all parts:-

2-a. Given a 2D list A [-100:100] [-5:50]. Find the address of element A [99, 49] in row major order considering base address 10 and each element requires 4 bytes for storage. (CO1) 2

2-b. The prefix form of A - B / (C * D ^ E) is? (CO2) 2

2-c. Write display method to print information of all nodes in a singly linked list. (CO3) 2

2-d. Write a short note on Threaded binary tree. (CO4) 2

2-e. Differentiate between Sequential and Indexed file organization? (CO5) 2

SECTION B 30

3. Answer any five of the following:-

3-a. Sort the following numbers using Merge sort 24, 9, 29, 14, 19, 27. (CO1) 6

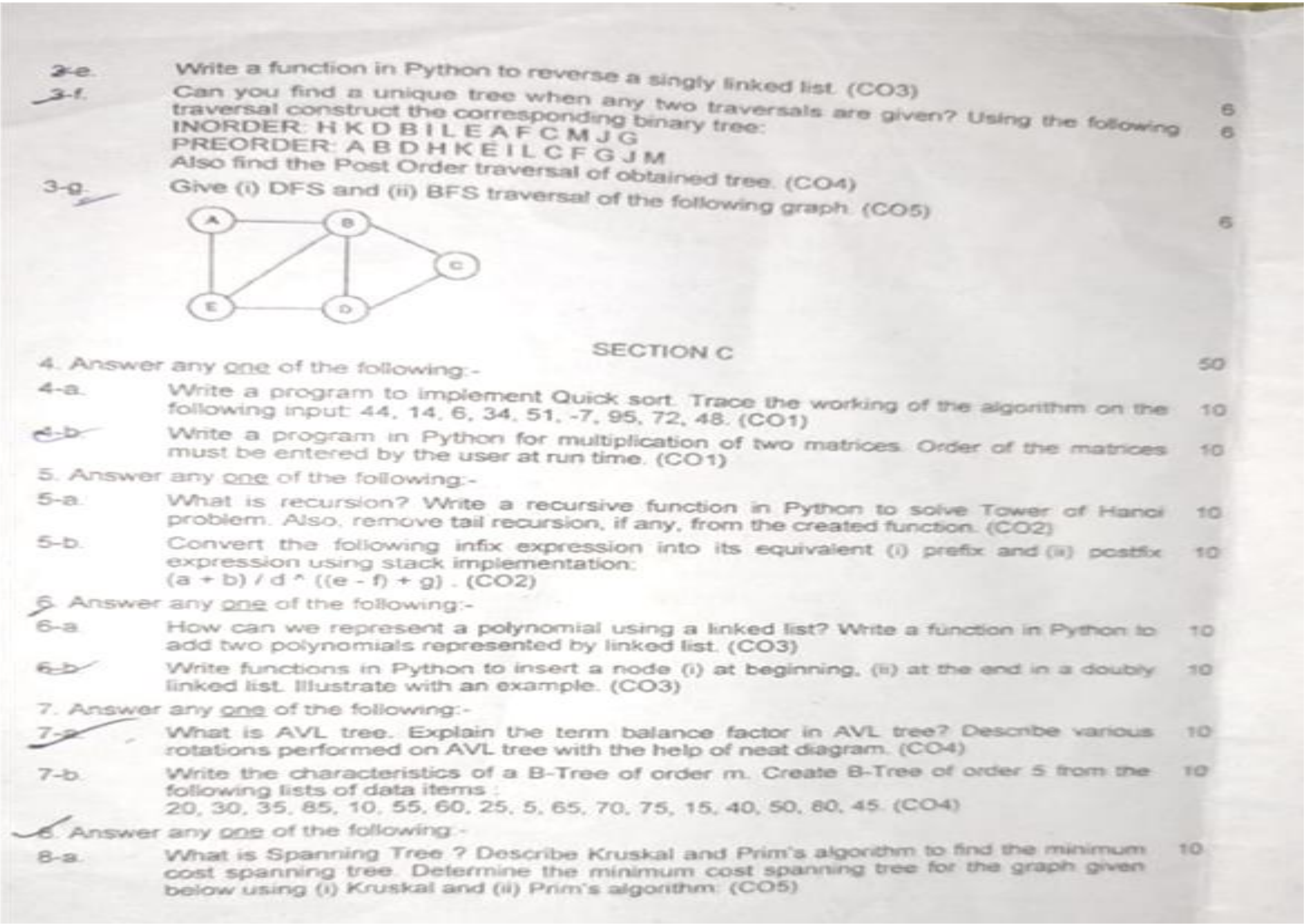
3-b. What is hashing? Give the characteristics of a good hash function. Explain any one collision resolution technique in hashing. (CO1) 6

3-c. The following sequence of operations is performed on stack: 6
PUSH (15),PUSH (25), POP, PUSH (17), PUSH (29), POP, POP, POP, PUSH (23), POP.
What will be the sequence of the value popped out? Also, write complexity of PUSH and POP operations, if stack is implemented using array. (CO2)

3-d. Write an algorithm to convert infix expression to postfix expression. (CO2) 6

NIET Greater Noida
GET FUTURE READY
Estd.2001
AN AUTONOMOUS INSTITUTE

Old Question Papers



3-e. Write a function in Python to reverse a singly linked list. (CO3) 6

3-f. Can you find a unique tree when any two traversals are given? Using the following traversal construct the corresponding binary tree: 6
INORDER: H K D B I L E A F C M J G
PREORDER: A B D H K E I L C F G J M
Also find the Post Order traversal of obtained tree. (CO4)

3-g. Give (i) DFS and (ii) BFS traversal of the following graph. (CO5) 6

SECTION C

4. Answer any one of the following:- 50

4-a. Write a program to implement Quick sort. Trace the working of the algorithm on the following input: 44, 14, 6, 34, 51, -7, 95, 72, 48. (CO1) 10

4-b. Write a program in Python for multiplication of two matrices. Order of the matrices must be entered by the user at run time. (CO1) 10

5. Answer any one of the following:-

5-a. What is recursion? Write a recursive function in Python to solve Tower of Hanoi problem. Also, remove tail recursion, if any, from the created function. (CO2) 10

5-b. Convert the following infix expression into its equivalent (i) prefix and (ii) postfix expression using stack implementation:
(a + b) / d ^ ((e - f) + g) . (CO2) 10

6. Answer any one of the following:-

6-a. How can we represent a polynomial using a linked list? Write a function in Python to add two polynomials represented by linked list. (CO3) 10

6-b. Write functions in Python to insert a node (i) at beginning, (ii) at the end in a doubly linked list. Illustrate with an example. (CO3) 10

7. Answer any one of the following:-

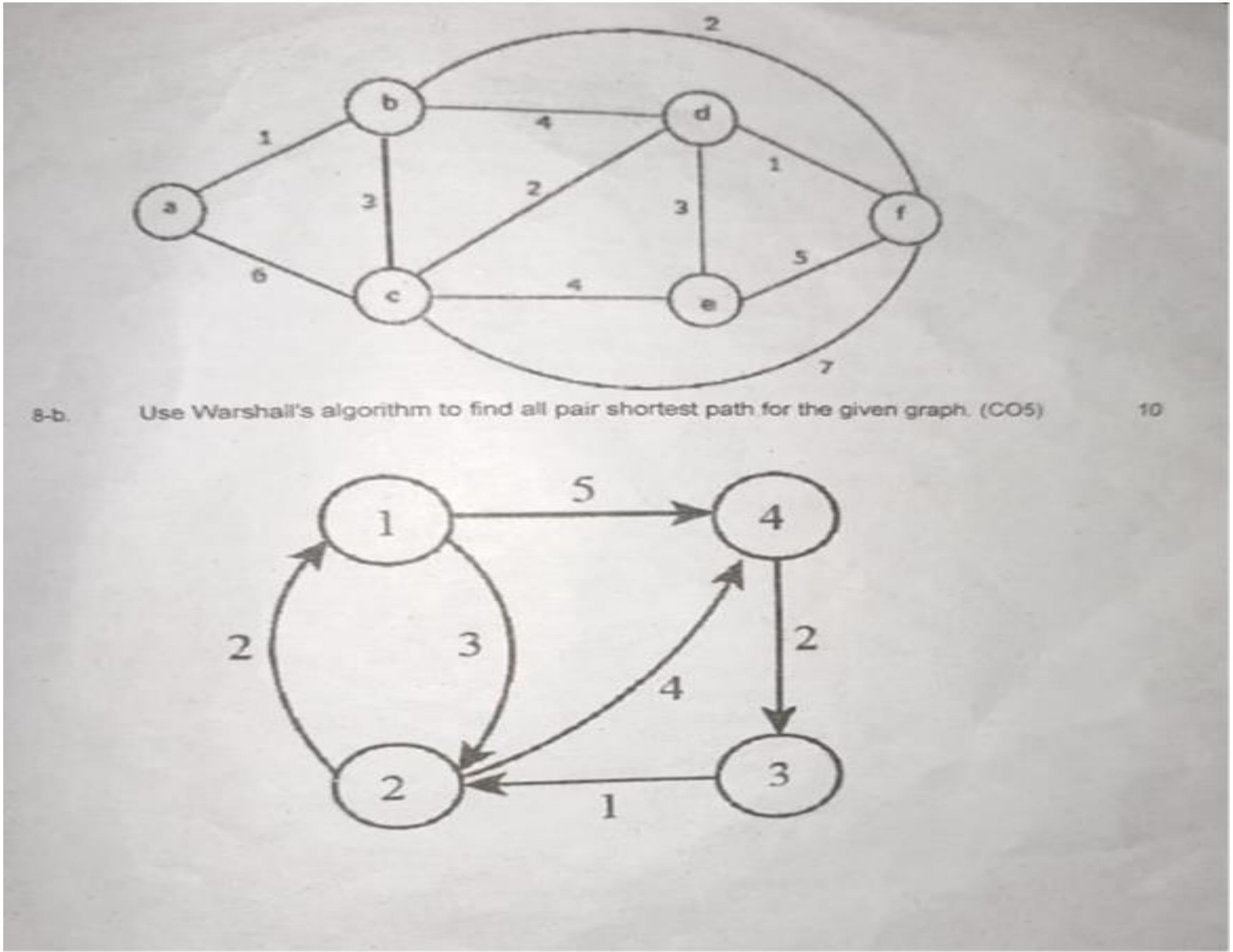
7-a. What is AVL tree. Explain the term balance factor in AVL tree? Describe various rotations performed on AVL tree with the help of neat diagram. (CO4) 10

7-b. Write the characteristics of a B-Tree of order m. Create B-Tree of order 5 from the following lists of data items :
20, 30, 35, 85, 10, 55, 60, 25, 5, 65, 70, 75, 15, 40, 50, 80, 45. (CO4) 10

8. Answer any one of the following:-

8-a. What is Spanning Tree ? Describe Kruskal and Prim's algorithm to find the minimum cost spanning tree. Determine the minimum cost spanning tree for the graph given below using (i) Kruskal and (ii) Prim's algorithm. (CO5) 10

Old Question Papers


2
b
d
1
4
1
a
3
2
3
f
5
6
c
4
e
7

8-b. Use Warshall's algorithm to find all pair shortest path for the given graph. (CO5) 10

1
5
4
2
3
4
2
2
3
1

Expected Questions for University Exam

- What is doubly linked list? What are its applications? Explain how an element can be deleted from doubly linked list using C program.
- What are the merits and demerits of array? Given two arrays of integers in ascending order, develop an algorithm to merge these arrays to form a third array sorted in ascending order.
- How can you represent a sparse matrix in memory?
- List the various operations on linked list.
- What do you understand by time and space trade off?
- Define the various asymptotic notations. Derive the O-notation for linear search.
- Write a program in c to delete a specific element in single linked list. Double linked list takes more space than single linked list for storing one extra address. Under what condition, could a double linked list more beneficial than single linked list.

Summary

We consider two fundamental data types for storing collections of objects: the stack and the queue.

We implement each using either a singly-linked list or a resizing array.

We introduce two advanced Java features—generics and iterators—that simplify client code.

Finally, we consider various applications of stacks and queues ranging from parsing arithmetic expressions to simulating queueing systems.

References

- Aaron M. Tenenbaum, Yedidyah Langsam and Moshe J. Augenstein, “Data Structures Using C and C++”, PHI Learning Private Limited, Delhi India
- Horowitz and Sahani, “Fundamentals of Data Structures”, Galgotia Publications Pvt Ltd Delhi India.
- Lipschutz, “Data Structures” Schaum’s Outline Series, Tata McGraw-hill Education (India) Pvt. Ltd.
- Thareja, “Data Structure Using C” Oxford Higher Education.
- AK Sharma, “Data Structure Using C”, Pearson Education India.
- Michael T. Goodrich, Roberto Tamassia, David M. Mount “Data Structures and Algorithms in C++”, Wiley India.
- . P. S. Deshpandey, “C and Data structure”, Wiley Dreamtech Publication.

Thank You