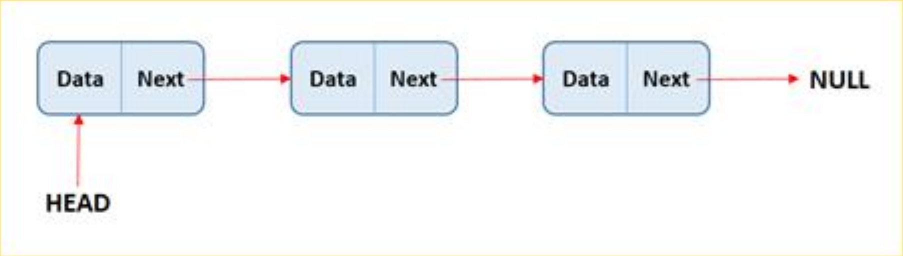
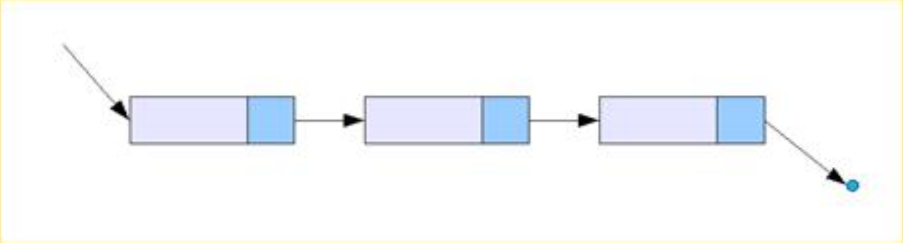
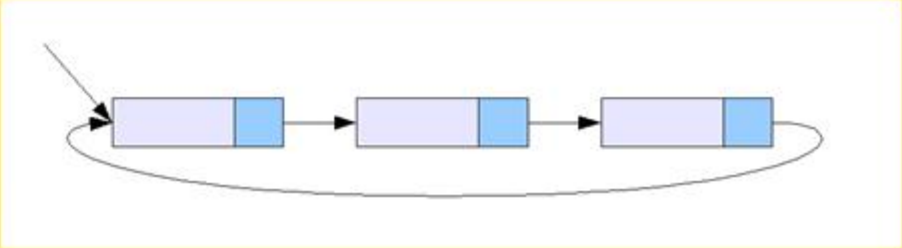
Md. Mehedi Hassan

Lecturer, Department of CIS, DIU







Linked List

Basic of Linked List

Linked List

A linked list is a linear collection of data elements called nodes, where the linear order is given by means of 'pointer'. Each node has two parts:

- Value
- Information address

The head is a special pointer variable which contains the address of the first node of the list. If head is NULL then the list is empty. NULL pointer also used to represent end of list.

Advantage of Linked List

- A linked list is a appropriate when the number of data element are unpredictable.
- It is also appropriate for frequently insertion and deletion of data.
- Linked list are dynamic. So the length of a list can be increased or decreased easily.

Types of Linked List

- One way linked list
- Two way linked list
- Circular linked list (One way and Two way)

Operation of Linked List

- Traversing
- Searching
- Insertion/Deletion into/from a linked list

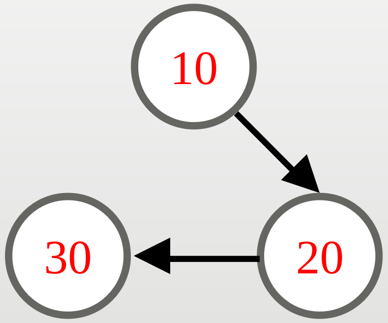
Disadvantage of Linked List

- In linked list, if we want to access any node it is difficult.
- It is occupying more memory.

❑ One way linked list

Head = 1x

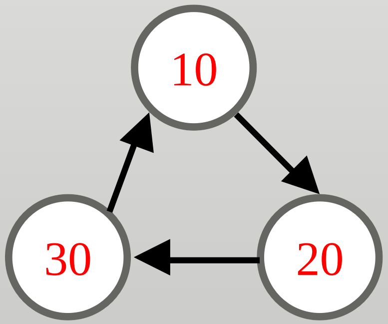
10	**2x**	→	**20**	**3x**	→	**30**	**Null**
1x			2x			3x	



❑ One way circular linked list

Head = 1x

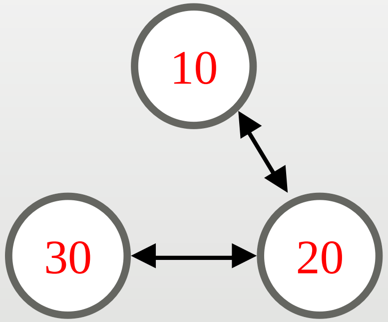
10	**2x**	→	**20**	**3x**	→	**30**	**1x**
1x			2x			3x	



❑ Two way linked list

Head = 1x

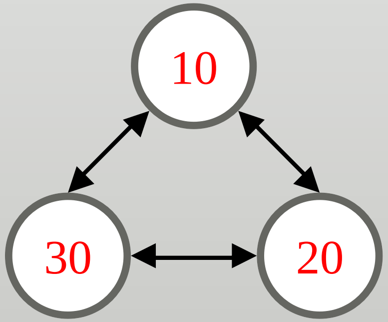
Null	**10**	**2x**	⟷	**1x**	**20**	**3x**	⟷	**2x**	**30**	**Null**
	1x				2x				3x	



❑ Two way circular linked list

Head = 1x

3x	**10**	**2x**	⟷	**1x**	**20**	**3x**	⟷	**2x**	**30**	**1x**
	1x				2x				3x	

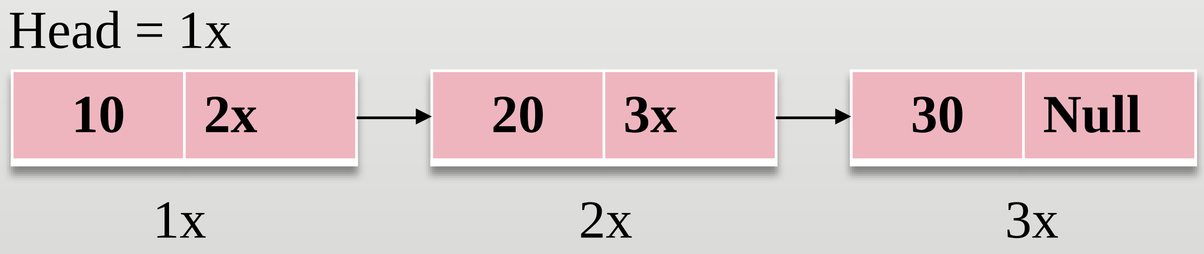


❑ Write an simple algorithm to traverse a linked list.

Pseudocode:

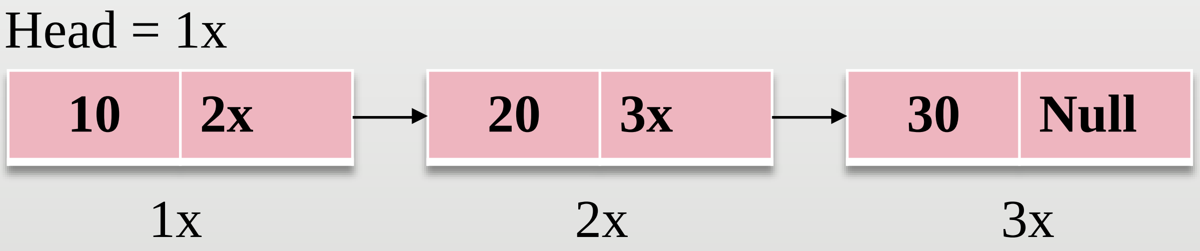
```
Linked list_Traversing (Head)

Step 1: Set PTR = Head
Step 2: while (PTR!=Null)
            {
            Printf (PTR → Value)
            PTR = PTR → Next
            }
Step 3: Exit
```



❑ Write an simple algorithm to search a linked list.

Pseudocode:




```
Linked list_Searching (Head, Svalue (20))

Step 1: Set PTR = Head
            Value found =0
Step 2: while (PTR!=NULL && Value found!=1)
            {
            If (PTR → Value == Svalue)
            Value found =1;
            PTR = PTR → Next;
            }
Step 3: Exit
```

❑ Write an simple algorithm to split a linked list.

Pseudocode:

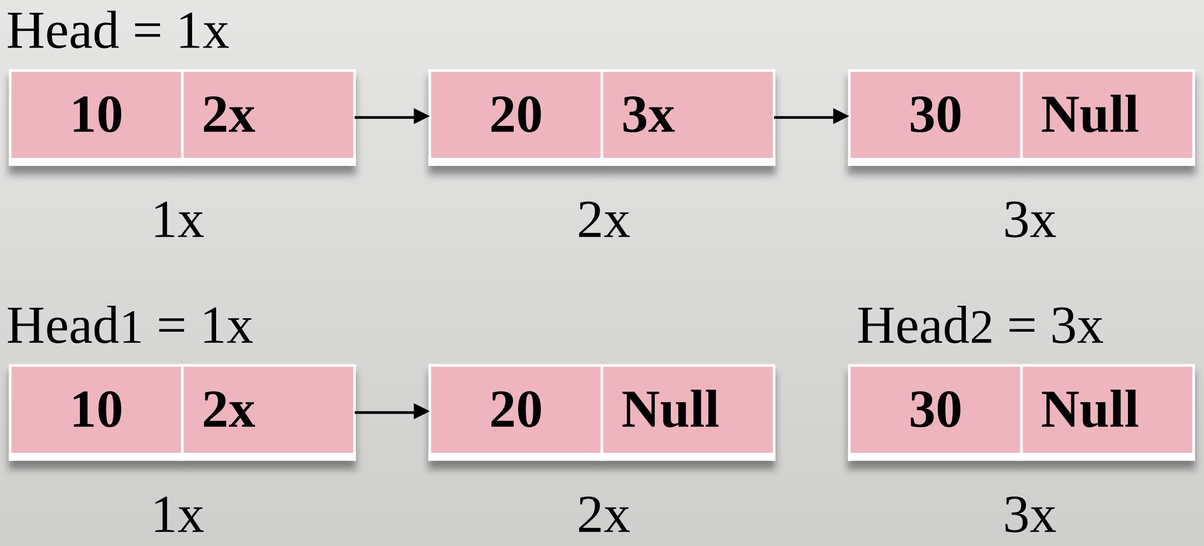
Split_linked list (Head, Position)

Step 1: Set Head1 = Head
Head2 = NULL
i = 1
PTR = Head

Step 2: while (i!=Position)
i = i+1
PTR = PTR → Next

Step 3: Head2 = PTR → Next
PTR → Next = NULL

Step 4: Exit



❑ Write an simple algorithm to merge two linked lists.

Pseudocode:

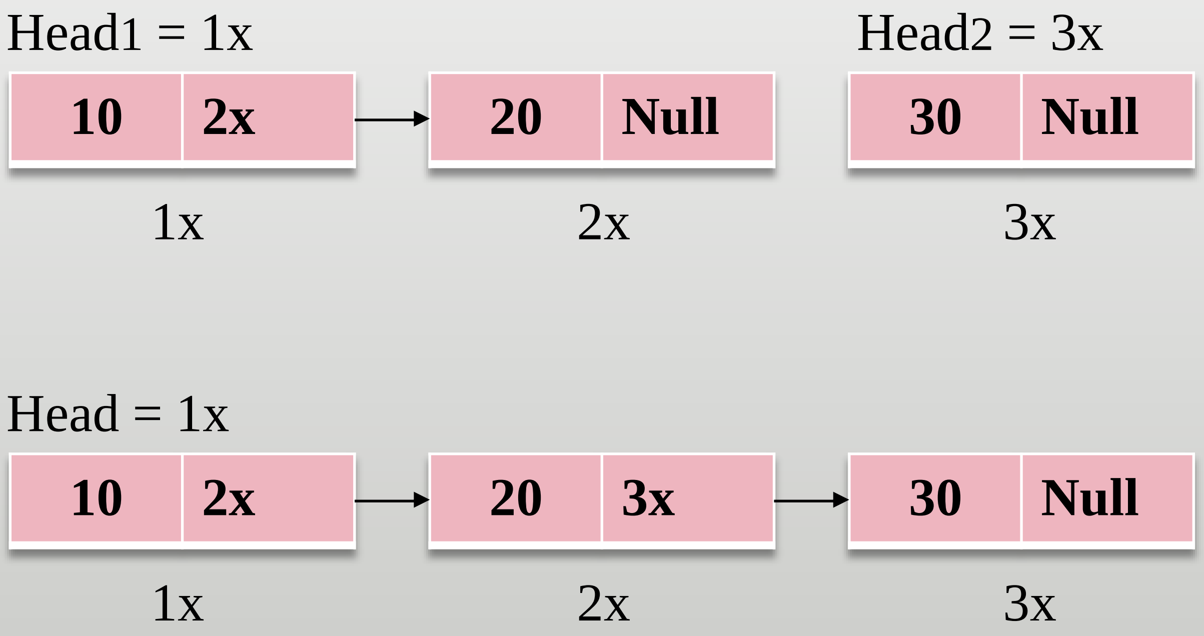
Merge_linked list (Head1, Head2)

Step 1: Set Head = Head1
PTR = Head1

Step 2: while (PTR → Next!=NULL)
PTR = PTR → Next

Step 3: PTR → Next = Head2

Step 4: Exit

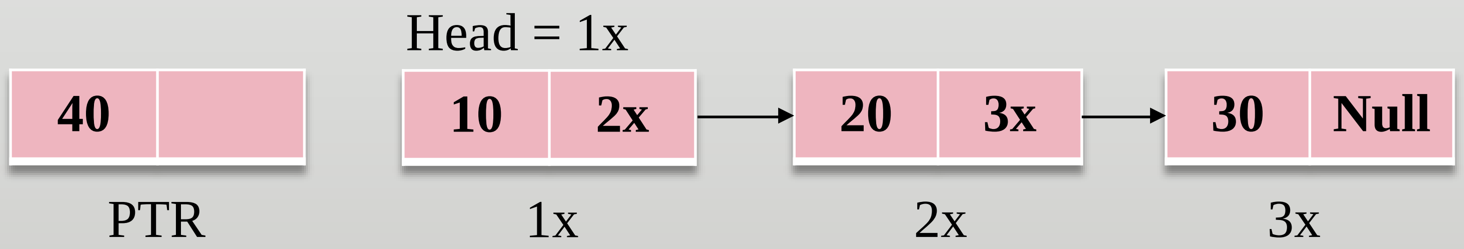


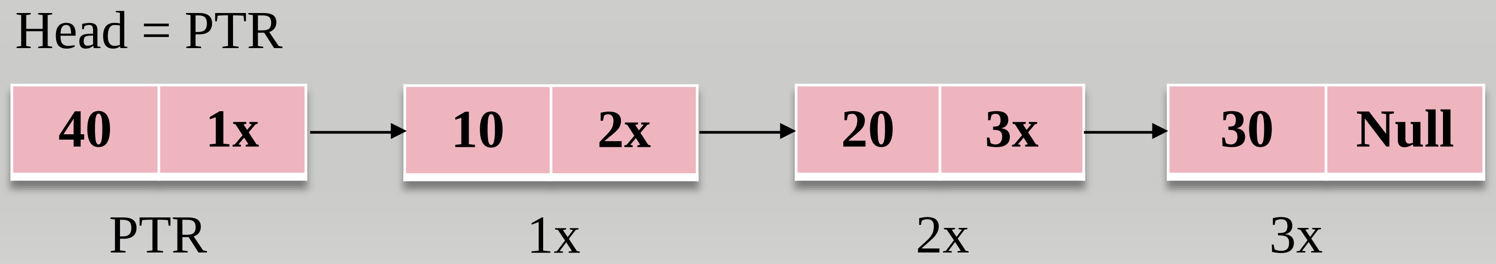
Insert into a linked list (One way)

Inserting a new item in simple linked list has three situations:

- ➢ Insert at first
- ➢ Insert at middle
- ➢ Insert at last

❑ Insert at first





Pseudocode:

Insert_first (Head, Invalue (40))

Step 1: PTR = Address of the insert value

Step 2: PTR → Value = Invalue

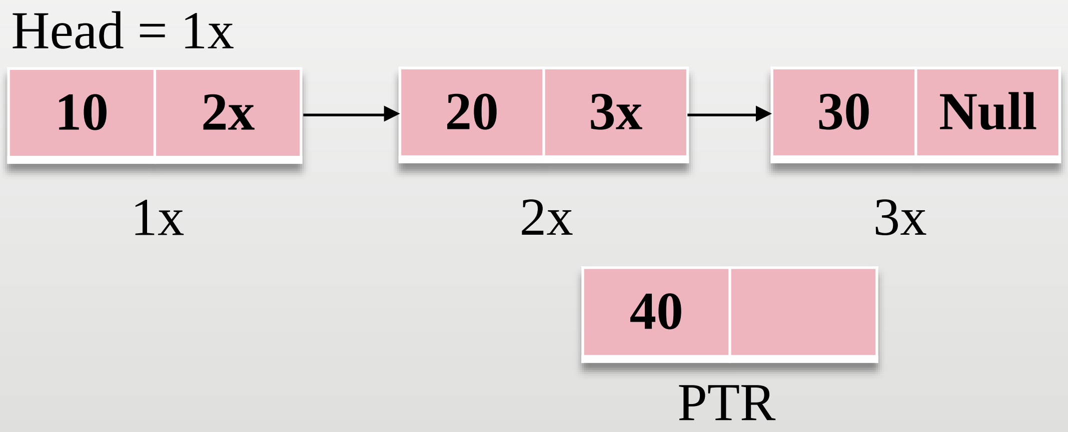
PTR → Next = Head

Head = PTR

Step 3: Exit

Insert into a linked list (One way)

❑ Insert at middle



Pseudocode:

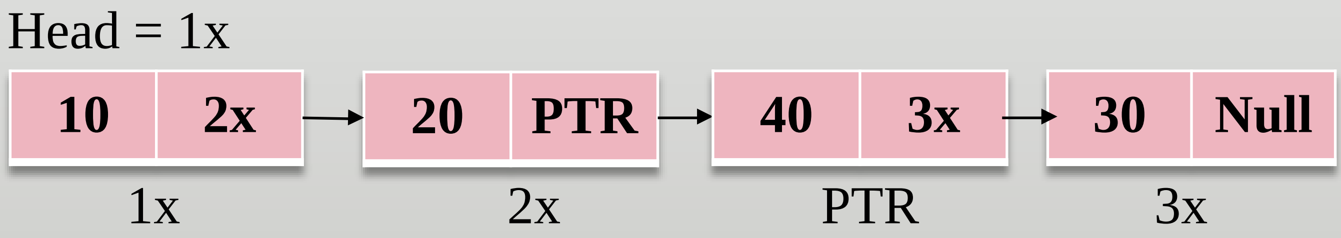
Insert_middle (Head, Invalue (40), Svalue(20))

Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR1 = Head

Step 3: while (PTR1 → Value!=Svalue)
PTR1 = PTR1 → Next

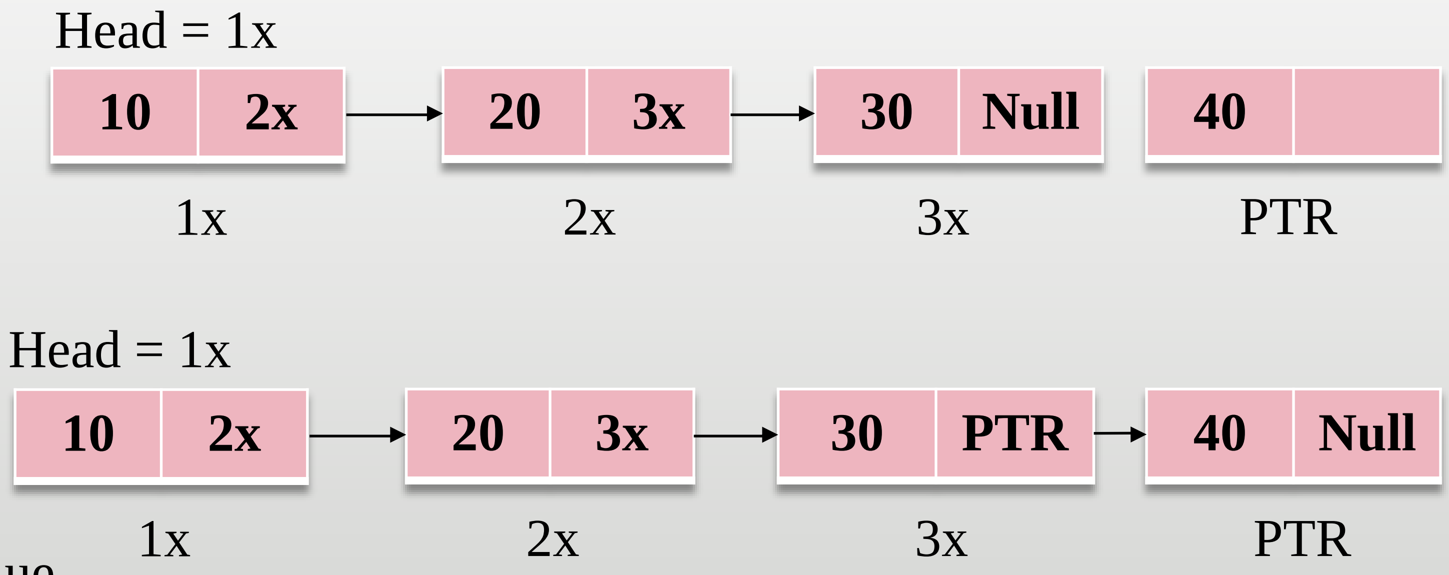
Step 4: PTR1 → Next = PTR
PTR → Next = PTR1 → Next

Step 5: Exit



Insert into a linked list (One way)

❑ Insert at last



Pseudocode:

Insert_last (Head, Invalue (40))

Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR → Next = Null
PTR1 = Head

Step 3: while (PTR1 → Next!=Null)
PTR1 = PTR1 → Next

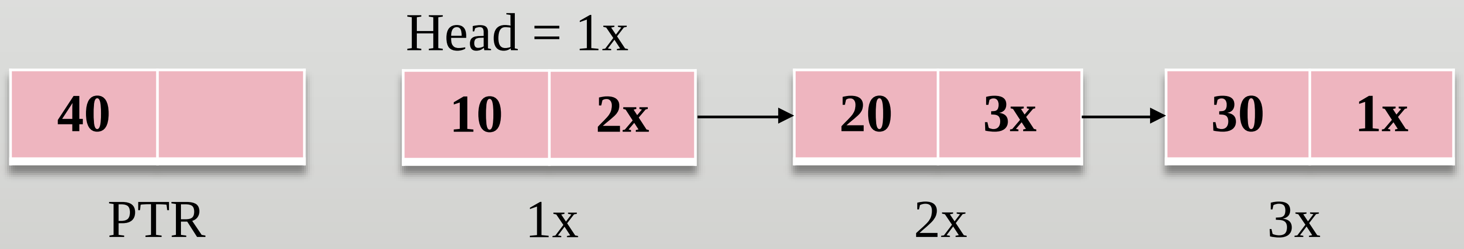
Step 4: PTR1 → Next = PTR
Step 5: Exit

Insert in circular linked list (One way)

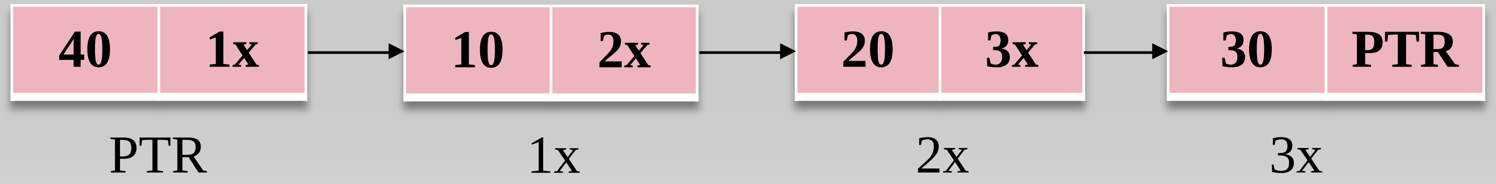
Inserting a new item in circular linked list has three situations:

- Insert at first
- Insert at middle
- Insert at last

❑ Insert at first



Head = PTR



Pseudocode:

Insert_first (Head, Invalue (40))

Step 1: PTR = Address of the insert value

Step 2: PTR → Value = Invalue

    PTR → Next = Head

    PTR1 = Head

Step 3: while (PTR1 → Next!=Head)

    PTR1 = PTR1 → Next

Step 4: PTR1 → Next = PTR

    Head = PTR

Step 5: Exit

Insert in circular linked list (One way)

❑ Insert at middle

Pseudocode:

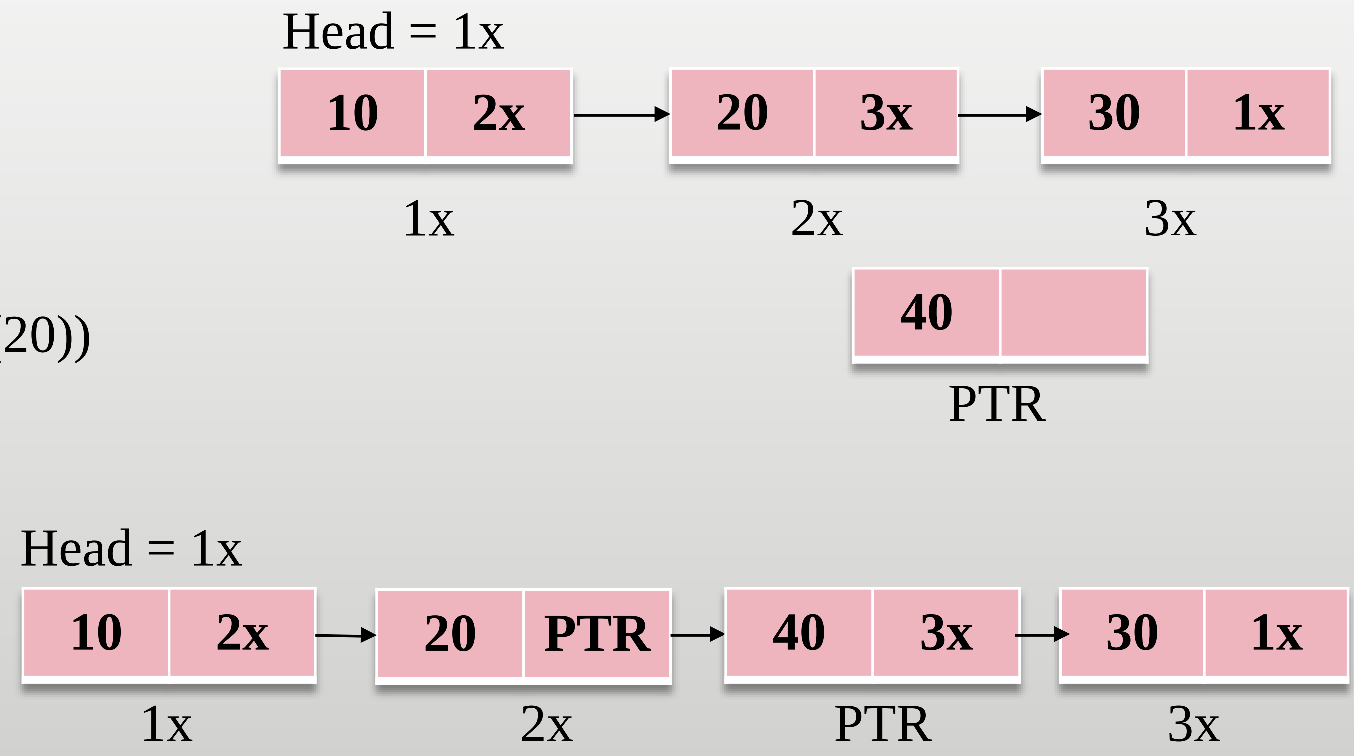
Insert_middle (Head, Invalue (40), Svalue(20))

Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR1 = Head

Step 3: while (PTR1 → Value!=Svalue)
PTR1 = PTR1 → Next

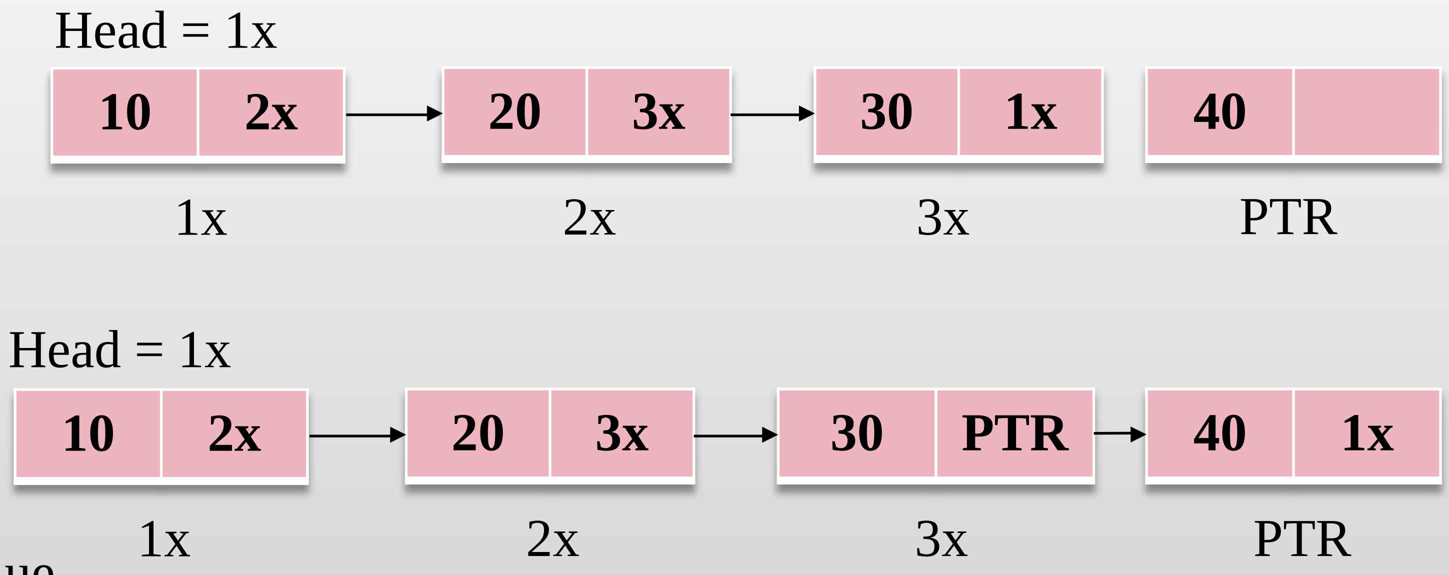
Step 4: PTR1 → Next = PTR
PTR → Next = PTR1 → Next

Step 5: Exit



Insert in circular linked list (One way)

❑ Insert at last



Pseudocode:

Insert_last (Head, Invalue (40))

Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR → Next = Head
PTR1 = Head

Step 3: while (PTR1 → Next!=Head)
PTR1 = PTR1 → Next

Step 4: PTR1 → Next = PTR
Step 5: Exit

Insert into a linked list (Two way)

❑ Insert at first

Pseudocode:

Insert_first (Head, Invalue (40))
Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR → Next = Head
PTR → Previous = Null
Head → Previous = PTR
Head = PTR
Step 3: Exit

Head = 1x

	40			**Null**	**10**	**2x**	↔	**1x**	**20**	**3x**	↔	**2x**	**30**	**Null**
	PTR				1x				2x				3x	

Head = PTR

Null	**40**	**1x**	↔	**PTR**	**10**	**2x**	↔	**1x**	**20**	**3x**	↔	**2x**	**30**	**Null**
	PTR				1x				2x				3x	

Insert into a linked list (Two way)

❑ Insert at middle

Pseudocode:

Insert_middle (Head, Invalue (40), Svalue(20))
Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR1 = Head
Step 3: while (PTR1 → Value!=Svalue)
PTR1 = PTR1 → Next
Step 4: PTR2 = PTR1 → Next
Step 5: PTR → Next = PTR1 → Next
PTR → Previous = PTR1
PTR1 → Next = PTR
PTR2 → Previous = PTR
Step 6: Exit

Head = 1x

Null	**10**	**2x**	↔	**1x**	**20**	**3x**	↔	**2x**	**30**	**Null**
	1x				2x				3x	

	40	
	PTR	

Head = 1x

Null	**10**	**2x**	↔	**1x**	**20**	**PTR**	↔	**2x**	**40**	**3x**	↔	**PTR**	**30**	**Null**
	1x				2x				PTR				3x	

Insert into a linked list (Two way)

- Insert at last

Pseudocode:

Insert_last (Head, Invalue (40))
Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR → Next = Null
PTR1 = Head
Step 3: while (PTR1 → Next!=Null)
PTR1 = PTR1 → Next
Step 4: PTR1 → Next = PTR
PTR → Previous = PTR1
Step 5: Exit

Head = 1x

Null	**10**	**2x**	↔	**1x**	**20**	**3x**	↔	**2x**	**30**	**Null**			**40**	
	1x				2x				3x				PTR	

Head = 1x

Null	**10**	**2x**	↔	**1x**	**20**	**3x**	↔	**2x**	**30**	**PTR**	↔	**3x**	**40**	**Null**
	1x				2x				3x				PTR	

Insert in circular linked list (Two way)

- Insert at first

Pseudocode:

```
Insert_first (Head, Invalue (40))
Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
            PTR → Next = Head
            PTR → Previous = Head → Previous
            Head → Previous = PTR
            PTR1 = Head
Step 3: while (PTR1 → Next!=Head)
            PTR1 = PTR1 → Next
Step 4: PTR1 → Next = PTR
            Head = PTR
Step 5: Exit
```

Head = 1x

	40		**3x**	**10**	**2x**	**1x**	**20**	**3x**	**2x**	**30**	**1x**
	PTR			1x			2x			3x	

Head = PTR

3x	**40**	**1x**	**PTR**	**10**	**2x**	**1x**	**20**	**3x**	**2x**	**30**	**PTR**
	PTR			1x			2x			3x	

Insert in circular linked list (Two way)

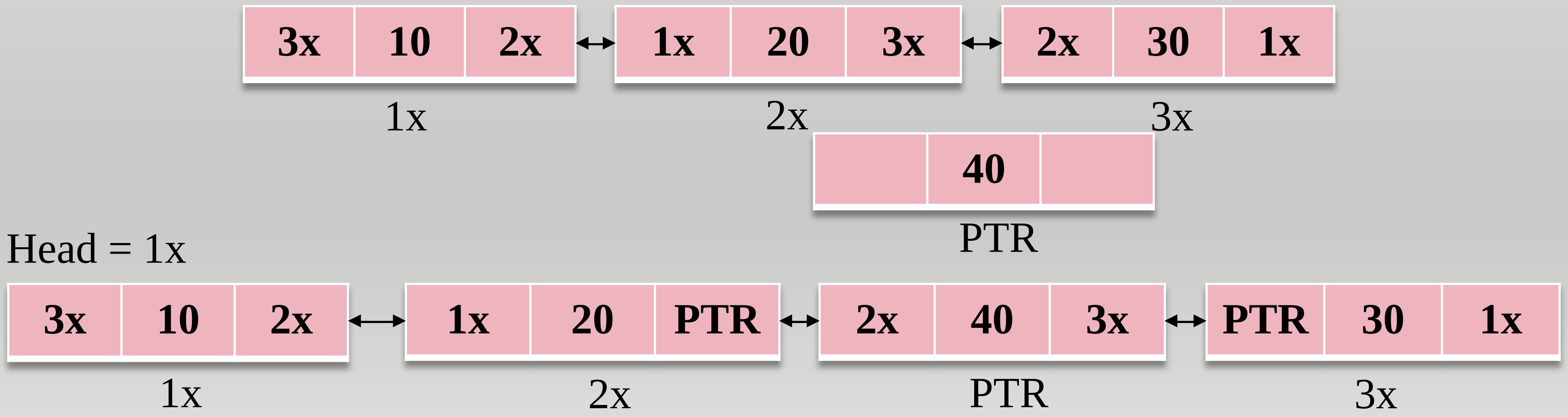
❑ Insert at middle

(Same as simple linked list)

Pseudocode:

Insert_middle (Head, Invalue (40), Svalue(20))
Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
    PTR1 = Head
Step 3: while (PTR1 → Value!=Svalue)
    PTR1 = PTR1 → Next
Step 4: PTR2 = PTR1 → Next
Step 5: PTR → Next = PTR1 → Next
    PTR → Previous = PTR1
    PTR1 → Next = PTR
    PTR2 → Previous = PTR
Step 6: Exit

Head = 1x



Insert in circular linked list (Two way)

❑ Insert at last

Pseudocode:

Insert_last (Head, Invalue (40))
Step 1: PTR = Address of the insert value
Step 2: PTR → Value = Invalue
PTR → Next = Head
Head → Previous = PTR
PTR1 = Head
Step 3: while (PTR1 → Next!=Head)
PTR1 = PTR1 → Next
Step 4: PTR1 → Next = PTR
PTR → Previous = PTR1
Step 5: Exit

Head = 1x

3x	**10**	**2x**	↔	**1x**	**20**	**3x**	↔	**2x**	**30**	**1x**			**40**	
	1x				2x				3x				PTR	

Head = 1x

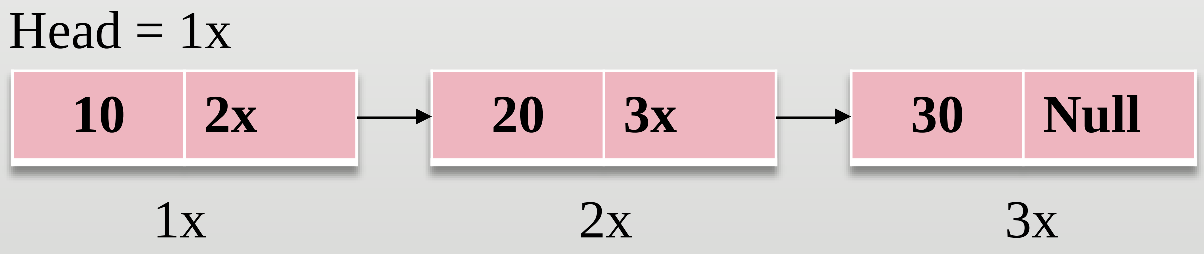
PTR	**10**	**2x**	↔	**1x**	**20**	**3x**	↔	**2x**	**30**	**PTR**	↔	**3x**	**40**	**1x**
	1x				2x				3x				PTR	

❑ Write an simple algorithm to count number of nodes.

Pseudocode:

Linked list_Nodes Count (Head)

Step 1: Set PTR = Head
    Count = 0
Step 2: while (PTR!=Null)
    Count = Count + 1
    PTR = PTR → Next
Step 3: Printf (Count)
Step 4: Exit



Thank You

Any Question ?