Parul® University

Data Structure

Dr. Ghanshyam Rathod, Assistant Professor
IT and Computer Science

Parul®
University

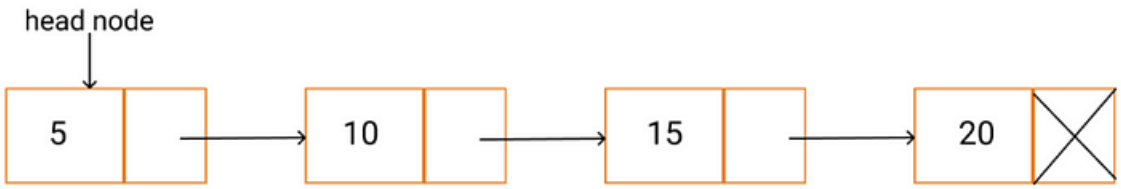
DIGITAL LEARNING CONTENT

CHAPTER-3

Linked List

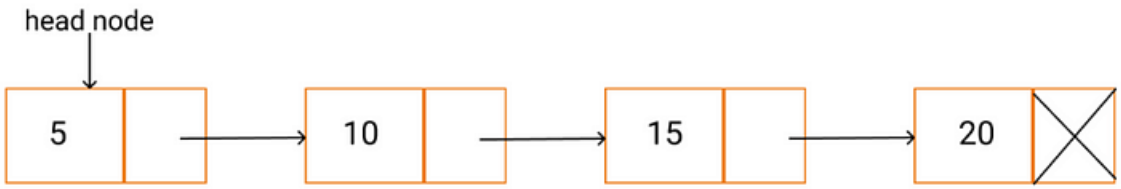
Linked List

- "A data structure that not only contains data field but also one pointer to the other nodes in the 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 is divided into two parts: the first part called ***Info*** contains the information of the element and the second part called ***Link*** or ***Next*** pointer field , contains the address of the next node in the list.



Linked List

- Each node is pictured with two parts : The left part represents the Info part of the node which contains one or more than one field. The right part represents the Link or next pointer field , which contains the address of the next node to which it points.
- START or Head contains the address of the first node in the list.
- The pointer at the last node contains a special value called NULL pointer which is any invalid address.



Linked List: Static Vs Dynamic Memory Allocation

Static Memory Allocation	Dynamic Memory Allocation
Eg. Array, Stack, Queue etc.	Linked List, Graph, Tree etc.
Linear memory allocation	Linked memory allocation
Storage requirement must be known at the time of creation	Not required
Insertion and deletion operations require large no. of movement of data	Less movement of date is needed
We can directly compute the address of a particular element with help of address calculating formula	It's difficult to calculate the address of a particular node by using formula.
Size of each element in a data structure is same	Size may be different
All the elements are logically as well as physically adjacent to each other in the memory.	All the elements are logically adjacent but need not be physically in the memory.

Types of Linked List

- **Single 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 and the first element has a link to the last element as previous.

Operations of Linked List

Creation:	This operation is used to create a linked list. Here, the constituent node is created as and when it is required and linked to the list to preserve the integrity of the list.
Insertion:	This operation is used to insert a new node in the linked list at the specified position. A new node may be inserted. - At the beginning of a linked list. - At the end of a linked list. - At the specified position in a linked list. - If the list itself is empty, then the new node is inserted as a first node.
Deletion:	This operation is used to delete an item (a node) from the linked list. A node may be deleted from the - Beginning of a linked list. - End of a linked list. - Specified position in the list.

Operations of Linked List

Traversing:	It is a process of going through all the nodes of a linked list from one end to the other end. If we start traversing from the very first node towards the last node, it is called forward traversing. If the desired element is found, we signal operation "SUCCESSFUL". Otherwise, we signal it as "UNSUCCESSFUL".
Concatenation:	It is a process of appending (joining) the second list to the end of the first list consisting of m nodes. When we concatenate two lists, the second list has n nodes, and then the concatenated list will be having (m + n) nodes. The last node of the list is modified so that it is now pointing to the first node in the second list.
Display:	This operation is used to print each and every node's information. We access each node from the beginning (or the specified position) of the list and output the data housed there.

Representation of Linked List in memory

Link list can be implemented in two ways :

1. Physical representation using ***array***
2. Logical representation using ***pointer***

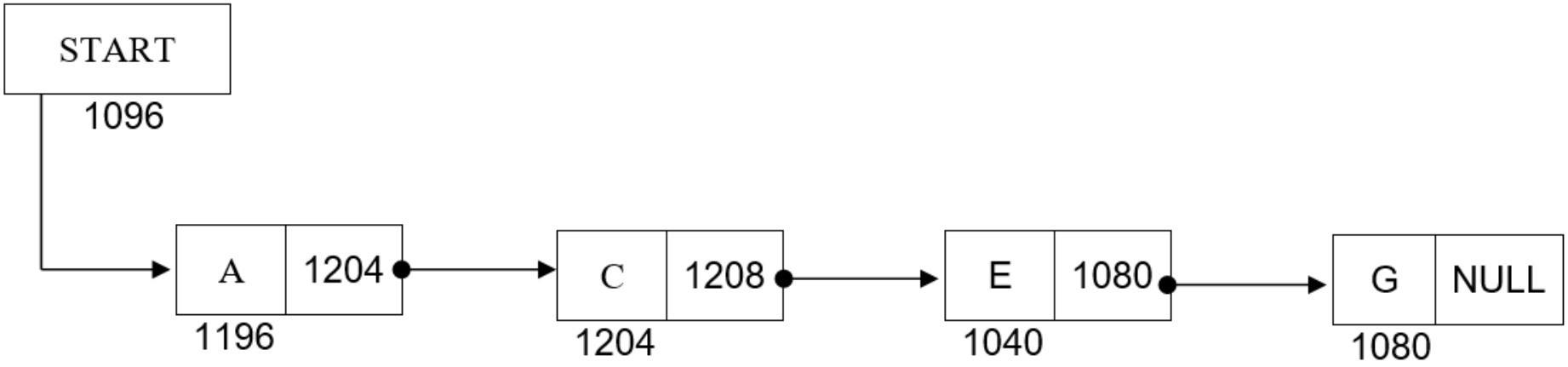
1. **Linked List using Array :** An Array Linked List uses an array to simulate link list. It requires two arrays - INFO and LINK such that INFO contains the Information of the element and LINK stores the index to the next value in array link list.

HEAD(START) →

INFO	LINK
A	6
C	4
L	1
B	2
M	NULL

Representation of Linked List in memory

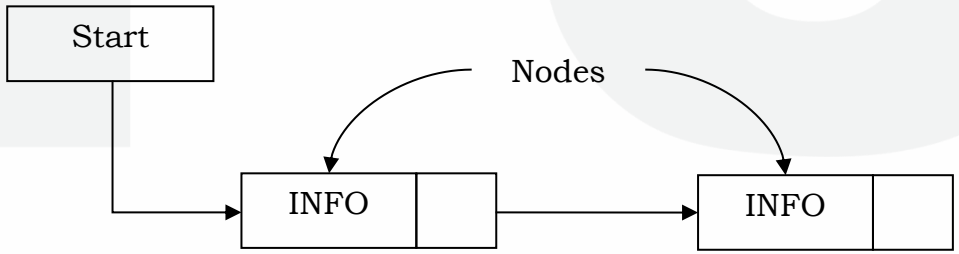
2. **Linked List using Pointer :** A pointer linked list is a dynamic representation of linked list created during the execution of a program. The memory is allocated in form of nodes. Each node has an INFO field that contains data and LINK pointer that is an address to the next node in the memory.



Single Linked List

Linked lists are special list of some data elements linked to one another. The logical ordering is represented by having each element pointing to the next element. Each element is called a *node*, which has two parts ***INFO*** part which stores the information and ***POINTER*** which points to the next element.

Following figure shows both types of lists (singly linked list and doubly linked lists).



Singly Linked Lists

Avail (Availability) List

- Together with the linked list, a special list is maintained in memory which consist of unused memory cells.
- This list, which has its own pointer, is called the list of available space or the free storage list or the free pool.
- This list is called the Avail List.
- During insertion operation, new nodes are taken from this avail list which is maintained just like normal data linked list using its own pointer.
- Similar to START, Avail list also has its own start pointer named as AVAIL which stores the address of the first free node of avail list.

START 5

AVAIL 10

1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8	Green	1
9	Samuels	0
10		2
11	Fields	8
12	Nelson	9

Algorithm for insert NEW node at BEGINNING of linked list

Function : INSERT(X,FIRST). Given X, a new element , and FIRST , a pointer to the first element of a linked list whose typical node contains INFO and LINK fields as previously described , this function inserts X.AVAIL is a pointer to the top element of the availability stack; NEW is a temporary pointer variable. It is required that X precede the node whose address is given by FIRST.

1. [Underflow?]
   If AVAIL = NULL
   then Write('AVAILABILITY STACK UNDERFLOW')
   Return(FIRST)
2. [Obtain address of next free node]
   NEW ← AVAIL

Algorithm for insert NEW node at BEGINNING of linked list

3. [Remove free node from availability stack]

   AVAIL ← LINK(AVAIL)

4. [Initialise fields of new node and its link to the list]

   INFO(NEW) ← X

   LINK(NEW) ← FIRST

5.[Return address of new node]

   Return(NEW)

Algorithm for insert NEW node at END of linked list

Function: INSEND(X,FIRST).Given X, a new element , and FIRST , a pointer to the first element of a linked linear list whose typical node contains INFO and LINK fields as previously described , this function inserts X.AVAIL is a pointer to the top element of the availability stack; NEW and SAVE are temporary pointer variables. It is required that X be inserted at the end of the list.

1.[Underflow?]
    If AVAIL = NULL
    then Write('AVAILABILITY STACK UNDERFLOW')
        Return(FIRST)
2. [Obtain address of next free node]
    NEW ← AVAIL
3. [Remove free node from availability stack]
    AVAIL ← LINK(AVAIL)

Algorithm for insert NEW node at END of linked list

4. [Initialise fields of new node]

   INFO(NEW) ← X

   LINK(NEW) ← NULL

5. [Is the list empty?]

   If FIRST = NULL

   then Return(NEW)

6. [Initiate search for the last node]

   SAVE ← FIRST

7. [Search for end of list]

   Repeat while LINK(SAVE) ≠ NULL

   SAVE ← LINK(SAVE)

8. [Set LINK field of last node to NEW]

   LINK(SAVE) ← NEW

9. [Return first node pointer]

   Return(FIRST)

Algorithm for DELETE node from linked list

Procedure DELETE (X, FIRST). Given X and FIRST, pointer variables whose values denote the address of a node in a linked list and the address of the first node in the linked list, respectively, this procedure deletes the node whose address is given by X. TEMP is used to find the desired node, and PRED keeps track of the predecessor of TEMP. Note, that FIRST is changed only when X is the first element of the list.

1. [Empty list?]

    If FIRST = NULL

    then write ('UNDERFLOW')

    Return

2. [Initialize search for X]

    TEMP ← FIRST

Algorithm for DELETE node from linked list

3. [Find X]

   Repeat thru step 5 while TEMP ≠ X and LINK(TEMP) ≠ NULL

4. [Update predecessor marker]

   PRED ← TEMP

5. [Move to next node]

   TEMP ← LINK(TEMP)

6. [End of the list?]

   If TEMP ≠ X

   then Write('NODE NOT FOUND')

   Return

Algorithm for DELETE node from linked list

7. [Delete X]

    If X = FIRST (Is X the first node?)

    then FIRST ← LINK(FIRST)

    else LINK(PRED) ← LINK(X)

8. [Return node to availability area]

    LINK(X) ← AVAIL

    AVAIL ← X

    Return

How to create Linked List and Node in ‘C’:

In C, a linked list is created using structures, pointers and dynamic memory allocation function malloc. We consider head as an external pointer. This helps in creating and accessing other nodes in the linked list.

Consider the following structure definition and head creation. [1]

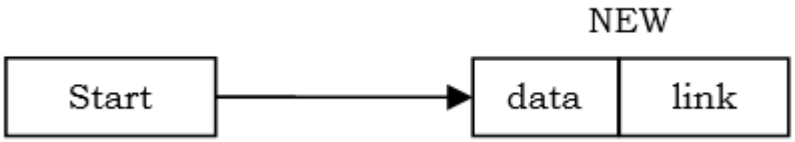
```
struct node
{
      int data;
      struct node *link;       /*pointer to node*/
};

struct node *start, *newnode;              /*pointer to the node of linked list*/
newnode = (struct node*) malloc (sizeof(struct node));
```

How to create Linked List and Node in 'C':

When the statement,

```
newnode = (struct node *) malloc(sizeof (struct node) );
```

is executed, a block of memory sufficient to store the NEW is allocated and assign start as the starting address of the NEW. This activity can be pictorially shown as follows:

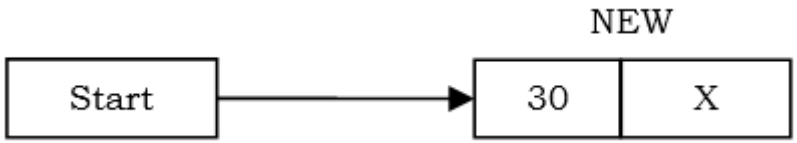


Now, we can assign values to the respective fields of NEW.

```
newnode -> data = 30        /* Data fields contains value 30 */
newnode -> link = NULL;     /* Null pointer assignment */
start = newnode;
```

How to create Linked List and Node in 'C':

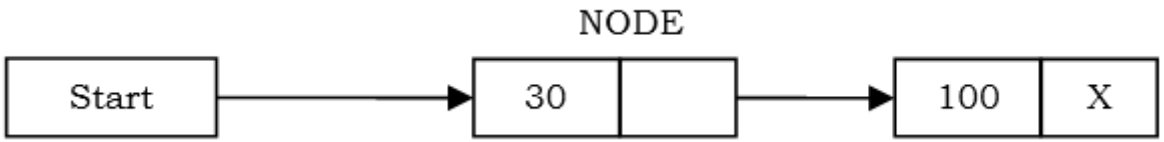
Thus, new vision of the newnode would be like this:



Any number of nodes can be created and linked to the existing node. Suppose we want to add another node to the above list, then the following statements are required.

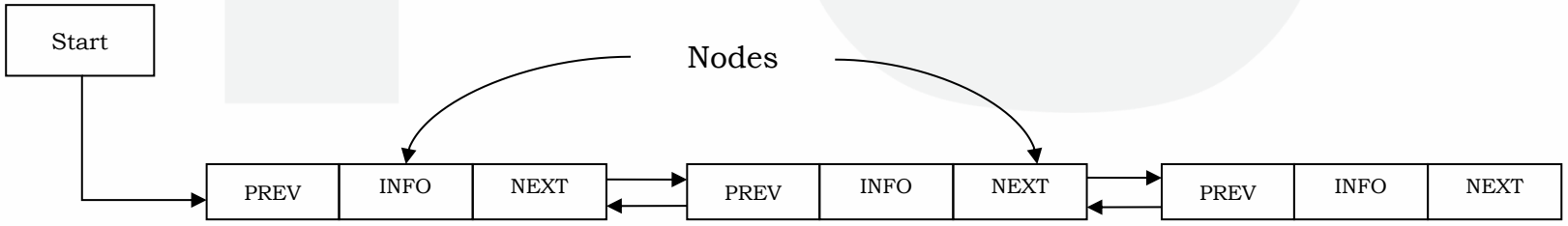
```
newnode = (struct node *) malloc (sizeof (struct node) );
newnode -> data = 100;          /* Data fields contains value 30 */
newnode -> link = NULL;         /* Null pointer assignment */
start -> link = newnode;
```

Thus, the inserted list would have the following view



Double Linked List

- A Linked list containing two pointers, one to its predecessor and another to its successor, thus allowing to traverse the list in both forward and backward direction is called doubly linked list or Two-way chain.
- In singly linked list nodes have one pointer (NEXT) pointing to the next node, whereas nodes of doubly linked lists have two pointers (PREV and NEXT). PREV points to the previous node and NEXT points to the next node in the list.



Doubly Linked Lists

Double Linked List

Problem with single Linked list : To go back to the previous element in a Linked list one had to keep track by using a PRED pointer or traverse again from FIRST to come to that position.

Doubly linked list eliminates this task by using the link to the preceding as well as succeeding node. Besides that one can traverse in both directions in a linked list. This provides faster searching and deletion of an element if its position is known.

Disadvantage of Doubly Linked List : It uses extra memory pointers to store each element. This increases the overhead associated with it. i.e. To store a single element of data like ‘A’, two extra pointers are required. However, this technique is fruitful when each element is a large string or requires more space. i.e. To store a string like “Sportsmanship” only two pointers are required and hence reduce the overhead.

Algorithm for *Inserting* an element in a Doubly Link List

Terms Used in Double Linked List

1. PLINK: - Pointer to the Preceding element in the list.
2. NLINK: - Pointer to the Succeeding element in the list.
3. NEW, AVAIL is similar to previous definitions.

DBLYINSERT(X, PL, NL, M)

X - The Value of the element to be inserted in a doubly linked list

M - is the node near which we want to insert the element X

PL and NL are pointers which indicate whether the node is inserted preceding or succeeding to M.

Step 1: [Check for Memory available ?]

if AVAIL = NULL then

write ('NO MEMORY AVAILABLE)

Return

Algorithm for *Inserting* an element in a Doubly Link List

Step 3: [Update AVAIL to next free node]

AVAIL ← LINK(AVAIL)

Step 4: [Initialize NEW node]

INFO(NEW) ←X

Step 5: [Check For Empty list ?]

if M=NULL then

PLINK(NEW) ← NLINK(NEW) ← NULL

Return

Algorithm for *Inserting* an element in a Doubly Link List

Step 6: [Check insertion in the Preceding position?]

if PL = M then

PLINK(NEW)=M

NLINK(NEW)= NLINK(M)

NLINK(M)= NEW

PLINK(NLINK(NEW)) = NEW

Return

Step 7: [Insertion in the Next position]

NLINK(NEW)=M

PLINK(NEW)= PLINK(M)

PLINK(M)= NEW

NLINK(PLINK(NEW)) = NEW

Return

Algorithm for *Deleting* an element in a Doubly Link List

DBLYDELETE(X,HEAD)

X - The Value of the element to be deleted

PRED - pointer to preceding node

SUCD - pointer to succeeding node

Step 1: [Check for Empty List ?]

if HEAD = NULL then

write ('EMPTY LIST')

Return

Step 2: [Search Element X]

TEMP ← HEAD

Repeat while INFO(TEMP) ≠ X and TEMP ≠ NULL

TEMP ← NLINK(TEMP)

Algorithm for *Deleting* an element in a Doubly Link List

Step 3: [Verify for element found ?]

if INFO(TEMP) ≠ X then

write ('ELEMENT NOT FOUND')

Step 4: [Check if X is at HEAD]

if TEMP=HEAD then

HEAD ← NLINK(HEAD)

Return

Step 5: [Delete X from list]

PRED = PLINK(TEMP)

SUCD = NLINK(TEMP)

NLINK(PRED) = SUCD

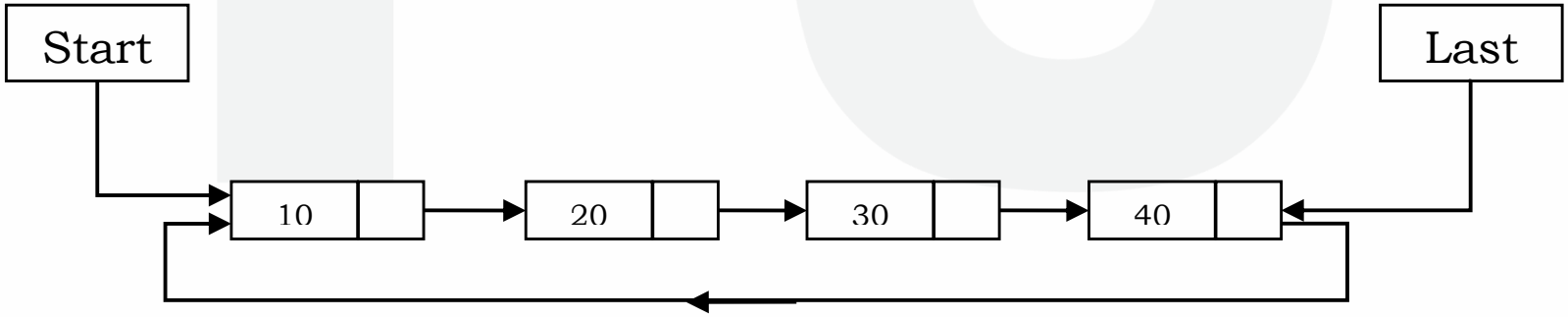
PLINK(SUCD) = PRED

Return

Circular Linked List

It is just a singly linked list in which the link field of the last node contains the address of the first node of the list. That is, the link field of the last node does not point to NULL rather it points back to beginning of the linked list.

A circular linked list has ***no*** end. Therefore, it is necessary to establish the ***FIRST*** and ***LAST*** nodes in such a linked list. It is useful if we the external pointer (i.e., head pointer) to point to the last node in this list. From this conversion we can easily locate the FIRST node of the list.

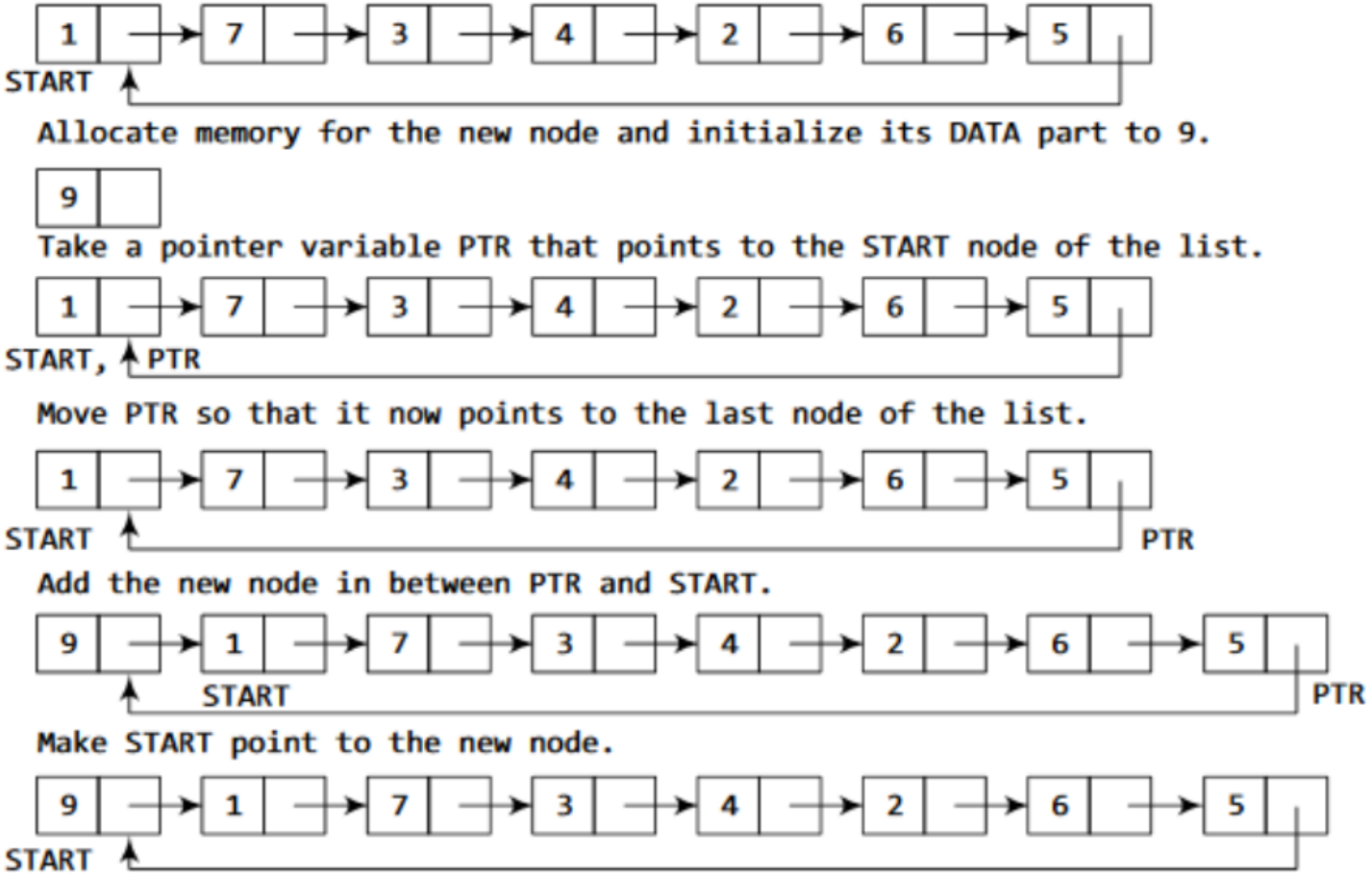


Circular Linked List

We declare the structure for the circular linked list in the same way as we declare it for the linear linked lists.

```
struct node
{
    int data;
    struct node *link;
} *start=NULL, *last=NULL;
```

Insert a NEW node at BEGINNING of a Circular Linked List



Insert a NEW node at BEGINNING of a Circular Linked List

Step 1 : IF AVAIL = NULL

Write OVERFLOW

Go to Step 11

[END OF IF]

Step 2 : SET NEW_NODE = AVAIL

Step 3 : SET AVAIL = AVAIL -> NEXT

Step 4 : SET NEW_NODE -> DATA = VAL

Step 5 : SET PTR = START

Insert a NEW node at BEGINNING of a Circular Linked List

Step 6 : Repeat Step 7 while PTR -> NEXT != START [START OF LOOP]

Step 7 : PTR = PTR -> NEXT

[END OF LOOP]

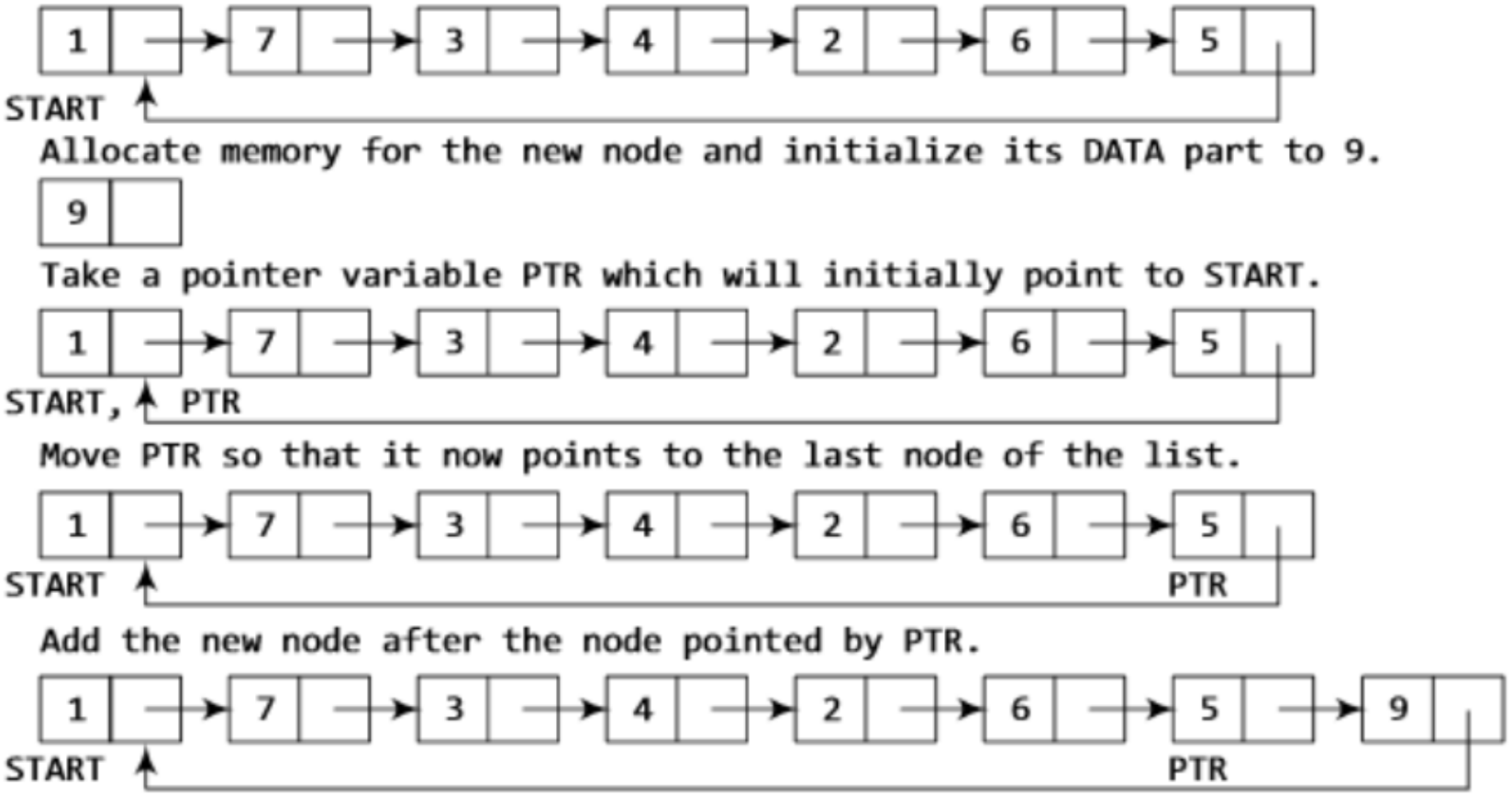
Step 8 : SET NEW_NODE -> NEXT = START

Step 9 : SET PTR -> NEXT = NEW_NODE

Step 10: SET START = NEW_NODE

Step 11: EXIT

Insert a NEW node at END of a Circular Linked List



Insert a NEW node at END of a Circular Linked List

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 1

[END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET AVAIL = AVAIL -> NEXT

Step 4: SET NEW_NODE -> DATA = VAL

Step 5: SET NEW_NODE -> NEXT = START

Step 6: SET PTR = START

Step 7: Repeat Step 8 while PTR -> NEXT != START

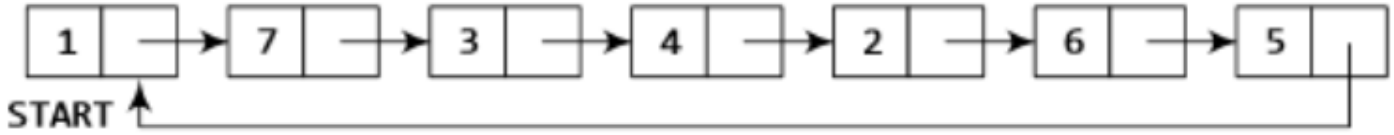
Step 8: SET PTR = PTR -> NEXT

[END OF LOOP]

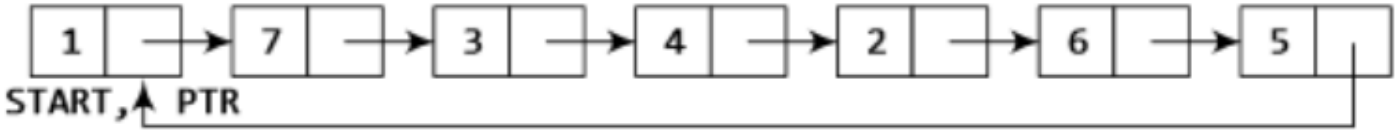
Step 9: SET PTR -> NEXT = NEW_NODE

Step 10: EXIT

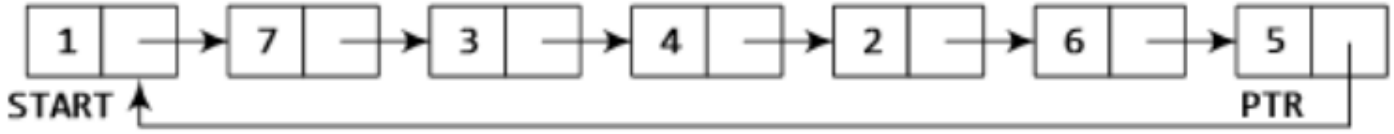
Deleting the FRIST node from a Circular Linked List



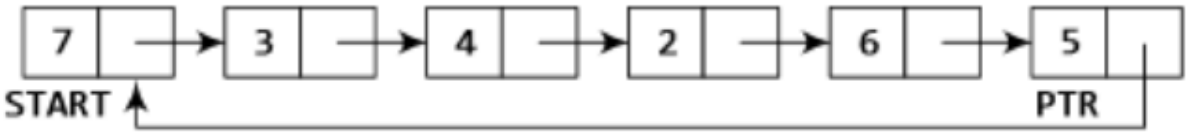
Take a variable PTR and make it point to the START node of the list.



Move PTR further so that it now points to the last node of the list.



The NEXT part of PTR is made to point to the second node of the list and the memory of the first node is freed. The second node becomes the first node of the list.



Deleting the FRIST node from a Circular Linked List

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 8

[END OF IF]

Step 2: SET PTR = START

Step 3: Repeat Step 4 while PTR -> NEXT != START

Step 4: SET PTR = PTR -> NEXT

[END OF LOOP]

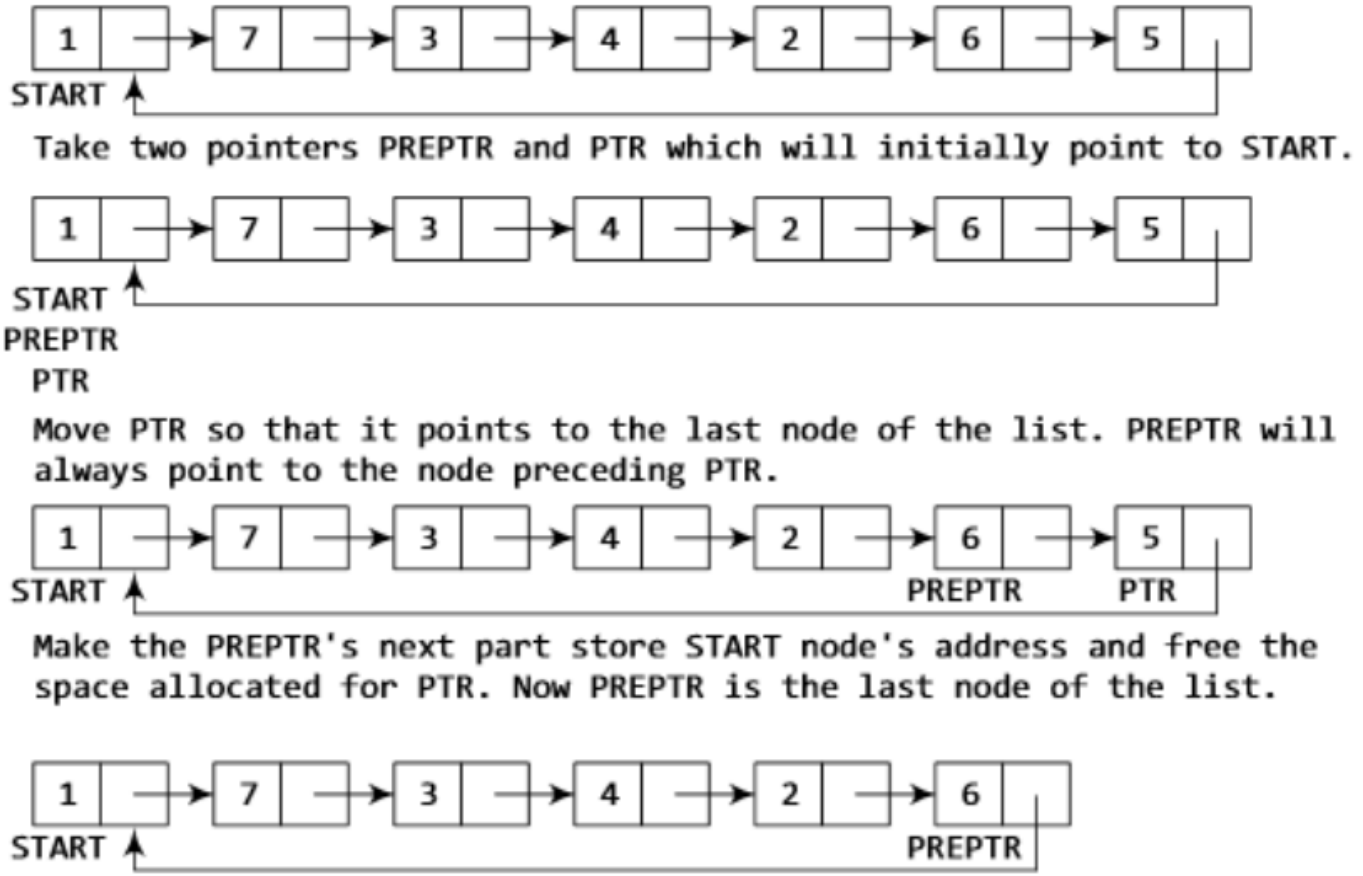
Step 5: SET PTR -> NEXT = START -> NEXT

Step 6: FREE START

Step 7: SET START = PTR -> NEXT

Step 8: EXIT

Deleting the LAST node from a Circular Linked List



Deleting the FRIST node from a Circular Linked List

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 8 [END OF IF]

Step 2: SET PTR = START

Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START

Step 4: SET PREPTR = PTR

Step 5: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 6: SET PREPTR -> NEXT = START

Step 7: FREE PTR

Step 8: EXIT

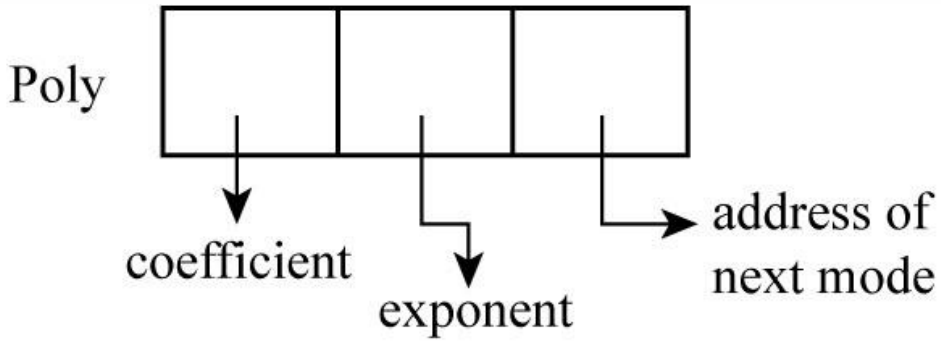
What is Polynomials?

- **Polynomials** are mathematical expressions consisting of **variables and coefficients,** that involve only the operations of addition, subtraction, multiplication, and **positive-integer powers of variables**.
- They are used in various fields of mathematics, astronomy, economics, etc.
- On the basis of number of terms there are different types of polynomials such as **monomial, binomial, trinomial** etc.
- Some examples of polynomials are **$2x + 3$, $x^2 + 4x + 5$, etc.**

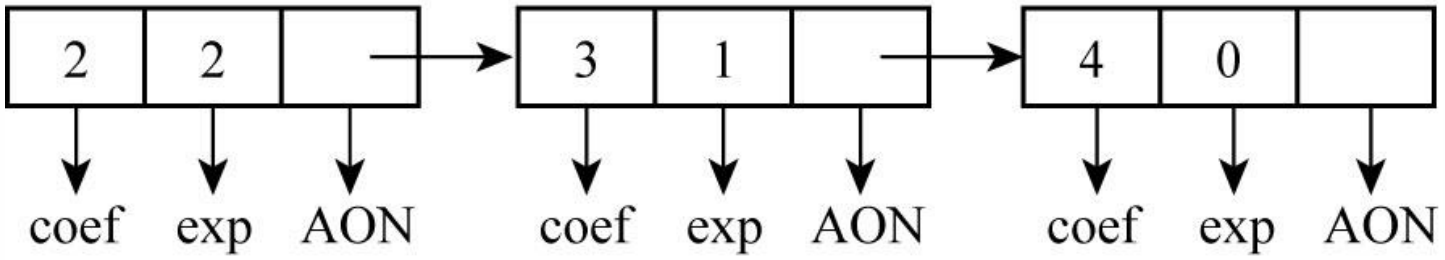
Representation of Polynomial using Linked List

- An ordered list of non-zero terms can be thought of as a polynomial. Each non-zero term consists of three sections namely coefficient part, exponent part, and then a pointer pointing to the node containing the next term of the polynomial.
- Let's take an example-
- If the polynomial is $2x^2+3x+4$, then it is written in the form of $2x^2+3x^1+4x^0$
- and represented it using a linked list. In the diagram, AON means “Address of next Node".

Representation of Polynomial using Linked List

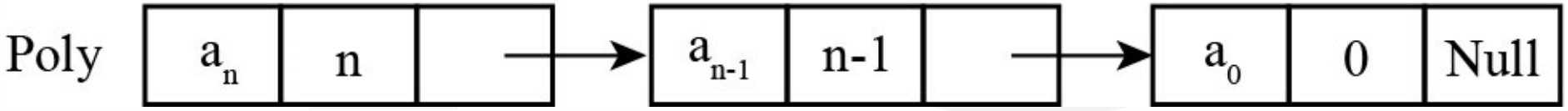


$2x^2 + 3x^2 + 4x^0$



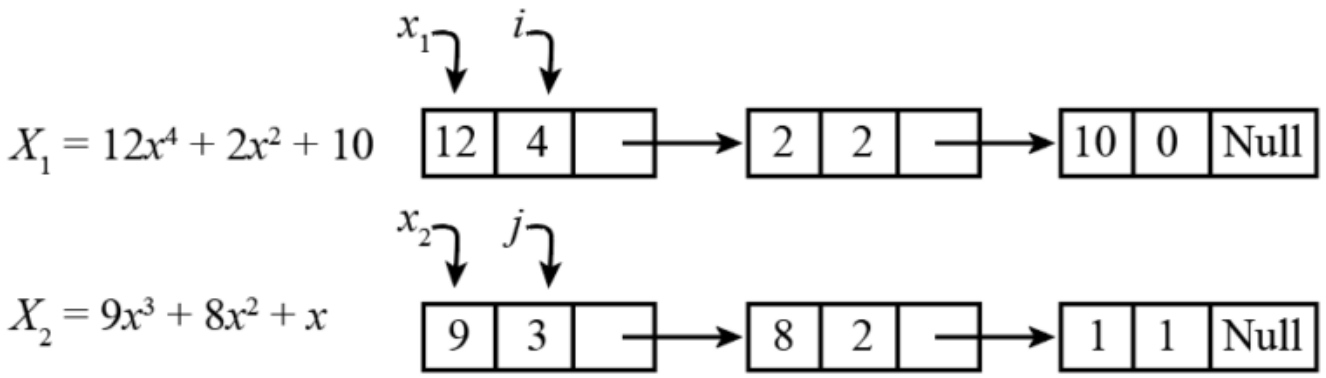
Representation of Polynomial using Linked List

- The above diagram shows the linked list representation of polynomial. A polynomial of a single variable can be written as $a_0+a_1X^1+a_2X^2+a_3X^3+.....+a_nX^n$
- where $a_n \neq 0$ and degree of A(X) is n. Here a_0, a_1, a_2, ...a_n is the coefficient of respective terms.

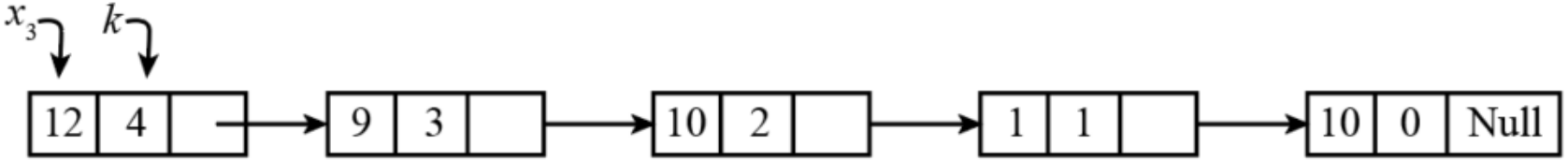


Addition of polynomials represented as linked lists

- Consider the polynomials $12x^4 + 2x^2 + 10$ and $9x^3 + 8x^2 + x$.



The resultant liked list :-



DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in