UNIT - II

* Arrays
* Operations on Arrays - Insertion & Deletion
* Applications on Arrays.
* Multidimensional Arrays - Sparse Matrix
* Linked List Implementation - Insertion
* Linked List : Deletion & Search
* Application of Linked List
* Polynomial Arithmetic
* Cursor Based Implementation - Methodology
* Cursor Based Implementation
* Circular Linked List
* Circular Linked List - Implementation
* Applications of Circular List
* Doubly Linked List
* Doubly Linked List - Insertion
* Doubly Linked List - Insertion variation
* Doubly Linked List Deletion
* Doubly Linked List Search

UNIT-II

ARRAYS

* An array is a collection of similar data elements.
* These data elements have the same data type.
* The elements of the array are stored in consecutive memory locations & are referenced by an index

DECLARATION OF ARRAYS

* Data type - kind of values stored eg: int, char, float, double
* Name - identify the array.
* Size - maximum number of values that the array can hold.
* Syntax: type name[size];
  int marks[10];

OPERATIONS ON ARRAYS

* Traversing
* Inserting
* Searching
* Deleting
* Merging
* Sorting

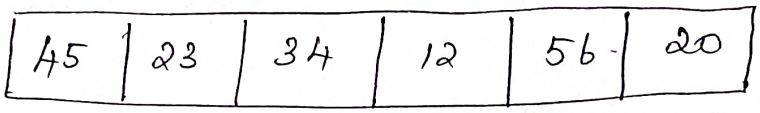
INSERTING AN ELEMENT IN AN ARRAY

* Simple - If insertion is done to the end of the existing array..
* Average - If insertion is done in the middle of the array as half of the elements have to be moved from their positions in order to accomodate space for the new element.

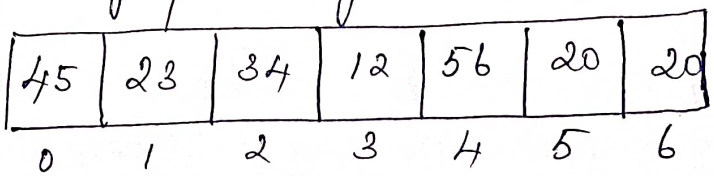
* Algorithm to insert an Element in the Middle of an Array

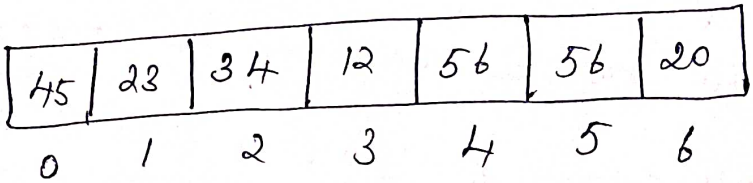
- INSERT(A, N, POS, VAL).
- The arguments are
  (i) A, the array in which the element has to be inserted.
  (ii) N, the number of elements in the array
  (iii) POS, the position at which the element has to be inserted.
  (iv) VAL, the value that has to be inserted.

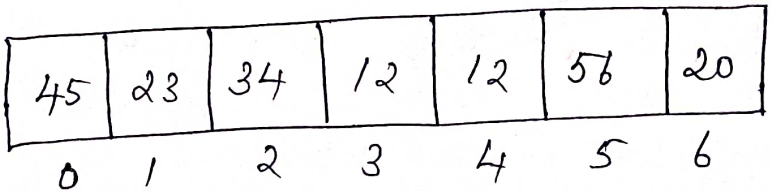
Eg: Data[6]

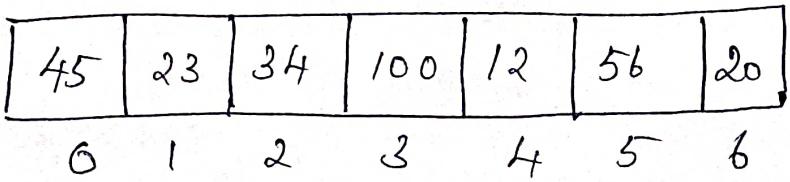


Calling INSERT(Data, 6, 3, 100) will lead to the following processing in the array :









DELETING AN ELEMENT FROM AN ARRAY

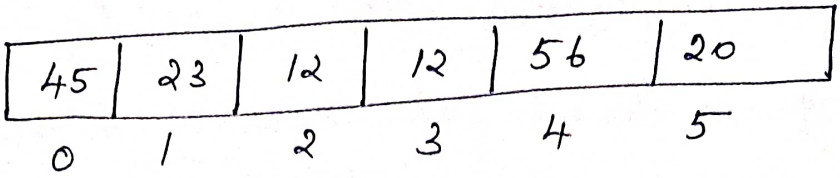
* Deleting an element from an array means removing a data element from an already existing array.
* If the element has to be deleted from the end of the existing array, one element has to be subtracted from the upper bound.
* If the element has to be deleted from the middle of the array, on an average, we might have to move as much as half of the elements from their position in order to occupy the space of the deleted element

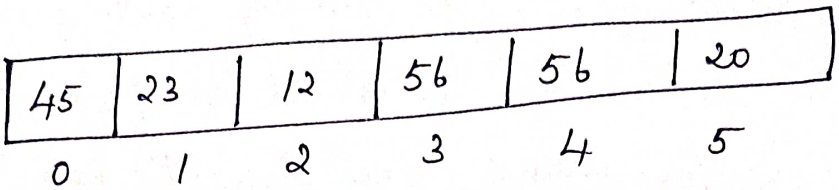
Algorithm to Delete an element from the middle of an Array

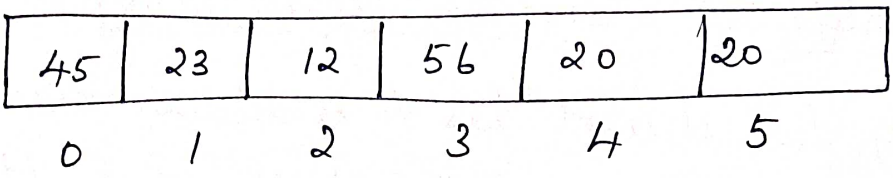
* The algorithm will be declared as
  DELETE (A, N, POS) where
  (a) A - array from which the element has to be deleted
  (b) N - number of elements in the array
  (c) POS - position from which the element has to be deleted.
* Eg: DELETE (Data, 6, 2)

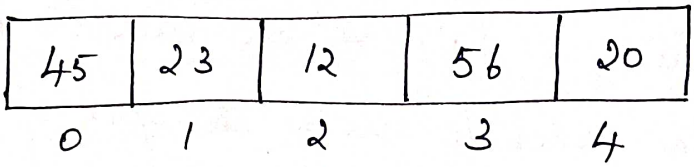
int Data[] = {45, 23, 34, 12, 56, 20}

45	23	34	12	56	20
0	1	2	3	4	5









* Program to insert a number at a given location in an array.

```
# include <stdio.h>
# include <conio.h>
int main()
{
    int i, n, num, pos, arr[10];
    clrscr();
    printf("\n Enter the number of elements in
            the array : ");
    scanf("%d", &n);
```

```
for (i=0; i<n; i++)
{
    printf("\n arr[%d]=", i);
    scanf("%d", &arr[i]);
}
printf("\n Enter the number to be inserted : ");
scanf("%d", &num);
printf("\n Enter the position at which the number
        has to be added : ");
scanf("%d", &pos);
for (i=n-1; i>=pos; i--)
    arr[i+1] = arr[i];
arr[pos] = num;
n = n+1;
printf("\n The array after insertion of %d is :", num);
for (i=0; i<n; i++)
    printf("\n arr[%d] = %d", i, arr[i]);
getch();
return 0;
}
```

* Program to delete a numbers from a given location in an array

```
# include <stdio.h>
# include <conio.h>
int main ()
{
    int i, n, pos, arr [10];
    clrscr ();
    printf ("\n Enter the number of elements in the array: ");
    scanf ("%d", &n);
    for (i=0; i<n; i++)
    {
        printf ("\n arr [%d] = ", i);
        scanf ("%d", &arr [i]);
    }
    printf ("\n Enter the position from which the number
                has to be deleted : ");
    scanf ("%d", &pos);
    for (i=0; i<n-1; i++)
        arr [i] = arr [i+1];
    n--;
    printf ("\n The array after deletion is : ");
    for (i=0; i<n; i++)
        printf ("\n arr [%d] = %d", i, arr [i]);
    getch ();
    return 0;
```

APPLICATIONS OF ARRAYS

* Arrays - used to implement mathematical vectors, matrices, and rectangular tables
* Databases include one-dimensional arrays whose elements are records.
* It is also used to implement other data structures like strings, stacks, queues, heaps & hash tables.
* It is used for sorting elements in ascending or descending order.

MULTIDIMENSIONAL ARRAYS (MD)

* MD Array is an array of arrays
* one index in a 1-D array, 2 indices in a 2-D array & n indices in an n-D array / multidimensional array
* A particular element is specified by n subscripts
  A[I1][I2][I3]...[In] where
  $I_1 <= M_1$, $I_2 <= M_2$, ... $I_n <= M_n$

SPARSE MATRICES

* It has large numbes of elements with a zero value.
* In order to efficiently utilize memory, specialized algorithms & data structures that take advantage

of the sparse structure should be used.

* If the operations used in standard matrix structures and algorithms are applied to sparse matrices, then the execution will slow down & the matrix will consume large amount of memory.
* Sparse data can be easily compressed, which in turn can significantly reduce memory usage.
* There are 2 types of sparse matrices

(i) all elements above the main diagonal have a zero value - (lower) triagonal matrix

$$\begin{bmatrix} 1 & & & & \\ 5 & 3 & & & \\ 2 & 7 & -1 & & \\ 3 & 1 & 4 & 2 & \\ -9 & 2 & -8 & 1 & 7 \end{bmatrix}$$

- $A_{i,j} = 0$ where $i < j$
- 1 non-zero element in 1st row
- 2 non-zero " 2nd row
- n non-zero " nth row

- Mapping between the 2-D matrix & a 1-D array can be done like in any of 2 ways:

(a) Row-wise mapping : A[] = {1, 5, 3, 2, 7, -1, 3, 1, 4, 2, -9, 2, -8, 1, 7}

(b) Column-wise mapping : A[] = {1, 5, 2, 3, -9, 3, 7, 1, 2, -1, 4, -8, 2, 1, 7}

(ii) all elements below the main diagonal have a zero value - (upper) triagonal matrix.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 6 & 7 & 8 & \\ & -1 & 9 & 1 & \\ & & 9 & 2 & \\ & & & 7 & \end{bmatrix}$$

- $A_{i,j} = 0$ where $i > j$
- n non-zero elements in 1st row
- $n-1$ non-zero " 2nd row
- 1 non-zero " n^{th} row

* Another variant of a sparse matrix, has non-zero elements appearing only on the diagonal or above/below the diagonal. - tri-diagonal matrix

$$\begin{bmatrix} 4 & 1 & & & \\ 5 & 1 & 2 & & \\ & 9 & 3 & 1 & \\ & & 4 & 2 & 2 \\ & & & 5 & 1 & 9 \\ & & & & 8 & 7 \end{bmatrix}$$

- $A_{i,j} = 0$ where $|i-j| > 1$

* In a tridiagonal matrix if elements are present on
  (i) the main diagonal, it contains non-zero elements for $i=j$. n elements in all.
  (ii) below the main diagonal, it contains non-zero elements for $i = j+1$. $(n-1)$ elements in all.
  (iii) above the main diagonal, it contains non-zero elements for $i = j-1$. $(n-1)$ elements in all.

* To store a TD-matrix in memory, we can used 1-D array that stores only non-zero elements.

* The mapping between a 2-D matrix and a 1-D array can be done in one of following ways.

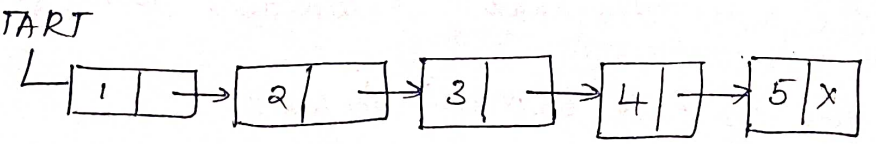
(a) Row-wise mapping: A[] = {4,1, 5,1,2, 9,3,1, 4,2,2, 5,1,9, 8,7}

(b) Column-wise mapping: A[] = {4,5, 1,1,9, 2,3,4, 1,2,5, 2,1,8, 9,7}

(c) Diagonal-wise mapping: A[] = {5,9,4,5,8, 4,1,3,2,1,7, 1,2,1,2,9}

LINKED LIST IMPLEMENTATION.

* A linked list is a linear collection of data elements called nodes.
* Each node contains one or more data fields and a pointer to the next node
* Eg:



Here each node has 2 parts: an integer + a pointer to the next node

* Since in a linked list, every node contains a pointer to another node which is of the same type, it is also called a self-referential data type.
* In C, it is implemented as

```
struct node
{
    int data;
    struct node *next;
};
```

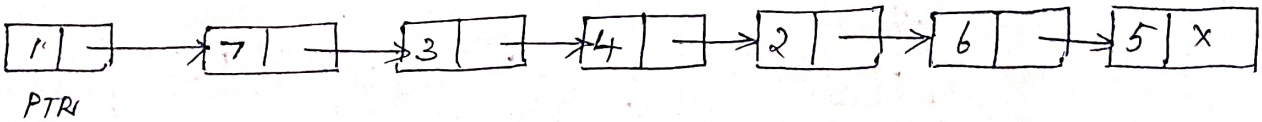
TRAVERSING A LINKED LIST

1. [INITIALIZE] SET PTR = START
2. Repeat steps 3 & 4 while PTR != NULL
3. Apply Process to PTR → DATA
4. SET PTR = PTR → NEXT
   [END OF LOOP]
5. EXIT

SEARCHING FOR A VALUE IN A LINKED LIST

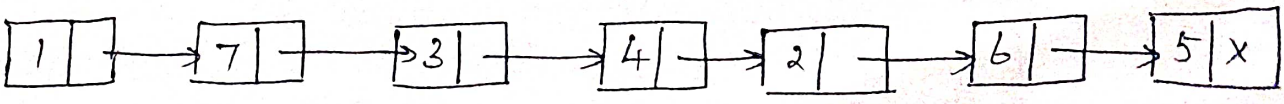
1. [INITIALIZE] SET PTR = START
2. Repeat Step 3 while PTR != NULL
3. IF VAL = PTR → DATA
   SET POS = PTR
   Go To Step 5
   ELSE
   SET PTR = PTR → NEXT
   [END OF IF]
   [END OF LOOP]
4. SET POS = NULL
5. EXIT

Eg: VAL = 4

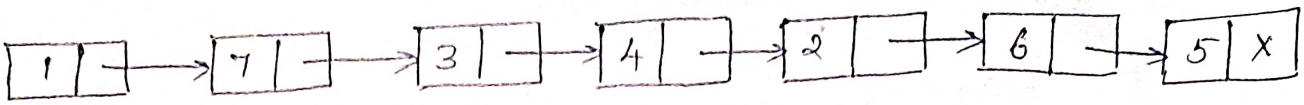


① PTR → DATA = 1 ② VAL != PTR → DATA ③ PTR = PTR → NEXT

① PTR → DATA = 7 ② VAL != PTR → DATA ③ PTR = PTR → NEXT



① PTR → DATA = 3 ② VAL != PTR → DATA ③ PTR = PTR → NEXT



① PTR → DATA = 4 ② VAL = PTR → DATA ③ POS = PTR

④ EXIT.

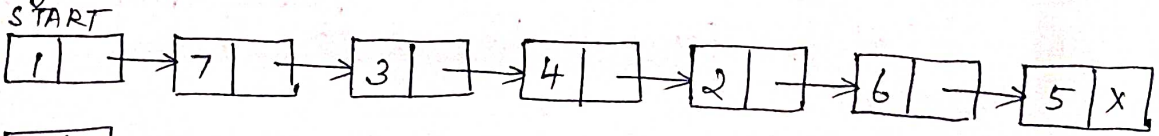
INSERTING A NEW NODE IN A LINKED LIST

* 4 Cases of Insertion
  (i) New node is inserted at the beginning
  (ii) New node is inserted at the end
  (iii) New node is inserted after a given node
  (iv) New node is inserted before a given node

(i) 1. IF AVAIL = NULL
Write OVERFLOW
Go to Step 7
[END OF IF]

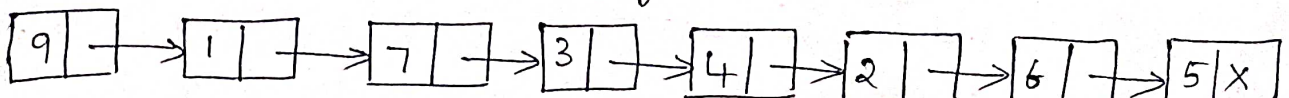
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL → NEXT
4. SET NEW_NODE → DATA = VAL
5. SET NEW_NODE → NEXT = START
6. SET START = NEW_NODE
7. EXIT

Eg: Insert 9. (VAL = 9)



New Node Memory Allocation

SET NEW_NODE = AVA
NEW_NODE → DATA = VAL



START

NEW_NODE → NEXT = START

SET START = NEW_NODE

(ii) 1. IF AVAIL = NULL

Write Overflow

Goto Step 10

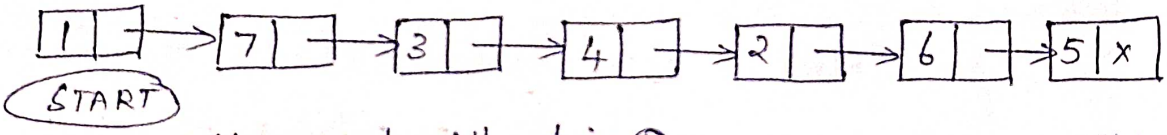
[END OF IF]

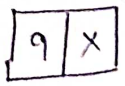
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL → NEXT
4. SET NEW_NODE → DATA = VAL
5. SET NEW_NODE → NEXT = NULL
6. SET PTR = START
7. Repeat Step 8 while PTR → NEXT != NULL
8. SET PTR = PTR → NEXT

[END OF LOOP]

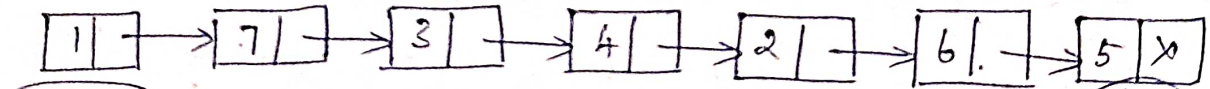
9. SET PTR → NEXT = NEW_NODE
10. EXIT

Eg: Insert 9 (VAL = 9)

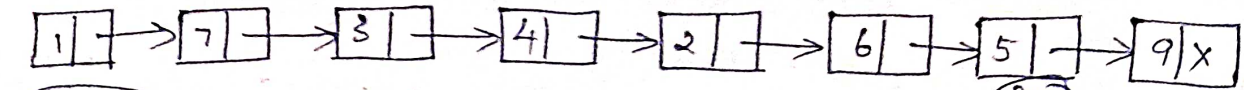




① New Node Allocation ② NEW_NODE → DATA = VAL
NEW_NODE = AVAIL
AVAIL = AVAIL → NEXT ③ NEW_NODE → NEXT = NULL



① SET PTR = START ② Move PTR, PTR = PTR → NEXT till end



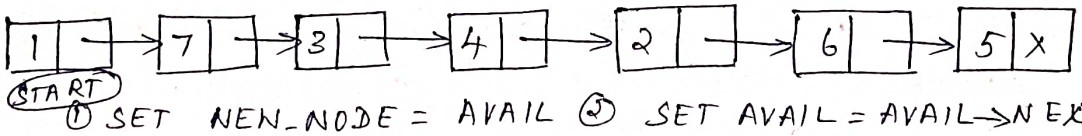
PTR → NEXT = NEW_NODE

(iii)

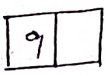
1. IF AVAIL = NULL
   Write OVERFLOW
   Go to Step 12
   [END OF IF]
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL → NEXT
4. SET NEW_NODE → DATA = VAL
5. SET PTR = START
6. SET PREPTR = PTR
7. Repeat Steps 8 & 9 while PREPTR → DATA != NUM
8. SET PREPTR = PTR
9. SET PTR = PTR → NEXT
   [END OF LOOP]
10. PREPTR → NEXT = NEW_NODE
11. SET NEW_NODE → NEXT = PTR
12. EXIT

PTR → NEXT
PREPTR → NEXT

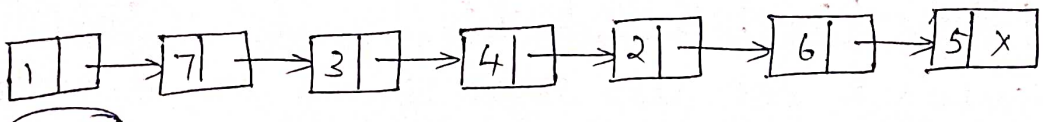
Eg: Insert 9 after node containing data 3 (VAL = 9)



① SET NEW_NODE = AVAIL ② SET AVAIL = AVAIL → NEXT

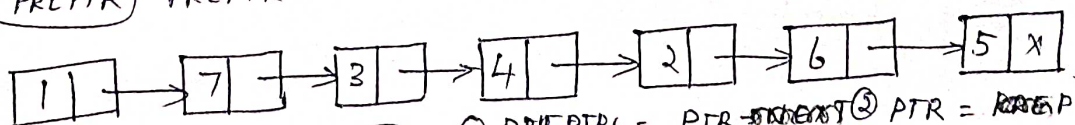


③ SET NEW_NODE → DATA = VAL



START
PTR
PREPTR

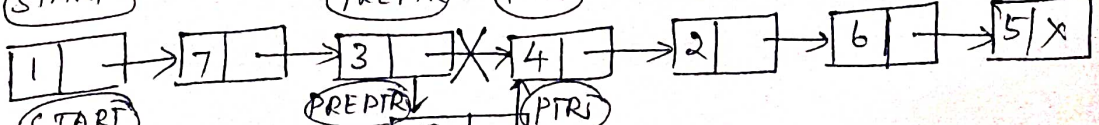
PTR = START.
PREPTR = PTR



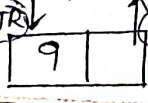
① PREPTR = PTR ② PTR = PREPTR → NEXT

START PREPTR PTR

START PREPTR PTR



START PREPTR PTR



NEW_NODE

(iv) 1. IF START = NULL

Write UNDERFLOW

Go To Step 5

[END OF IF]

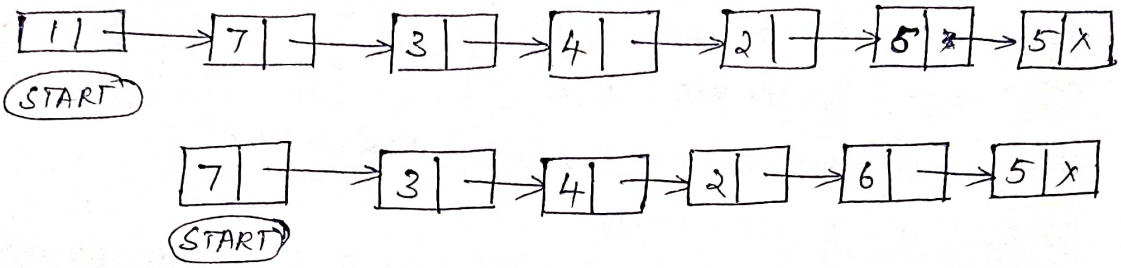
2. SET PTR = START

3. SET START = START → NEXT

4. FREE PTR

5. EXIT

// Deleting from start



1. IF START = NULL

Write UNDERFLOW

Go To Step 8

[END OF IF]

2. SET PTR = START

3. Repeat Steps 4 and 5 while PTR → NEXT != NULL

4. SET PREPTR = PTR

5. SET PTR = PTR → NEXT

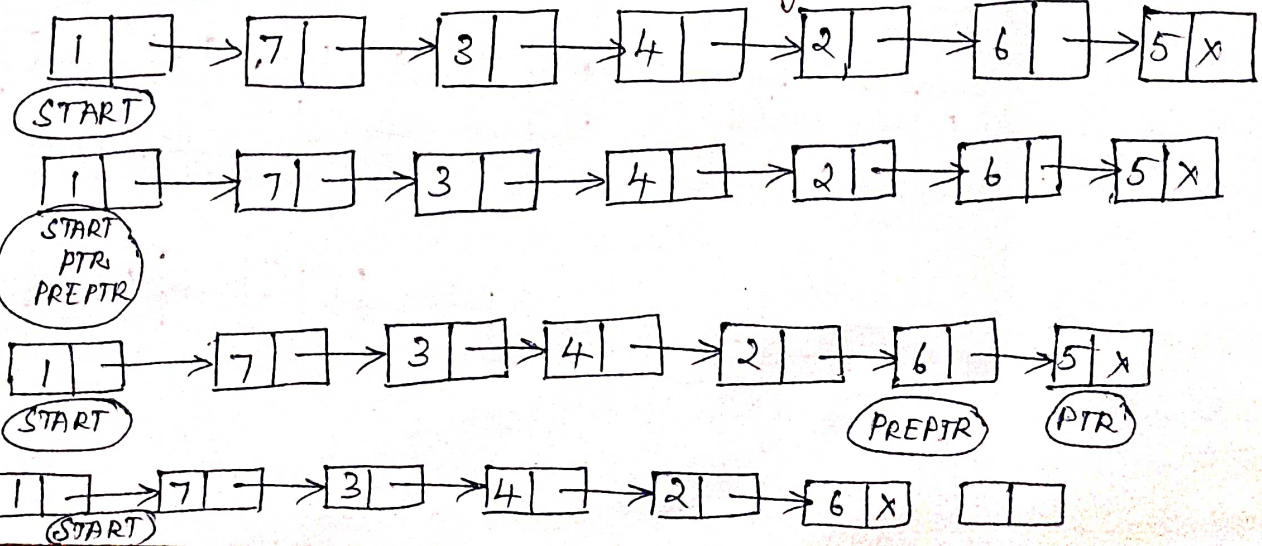
[END OF LOOP]

6. SET PREPTR → NEXT = NULL

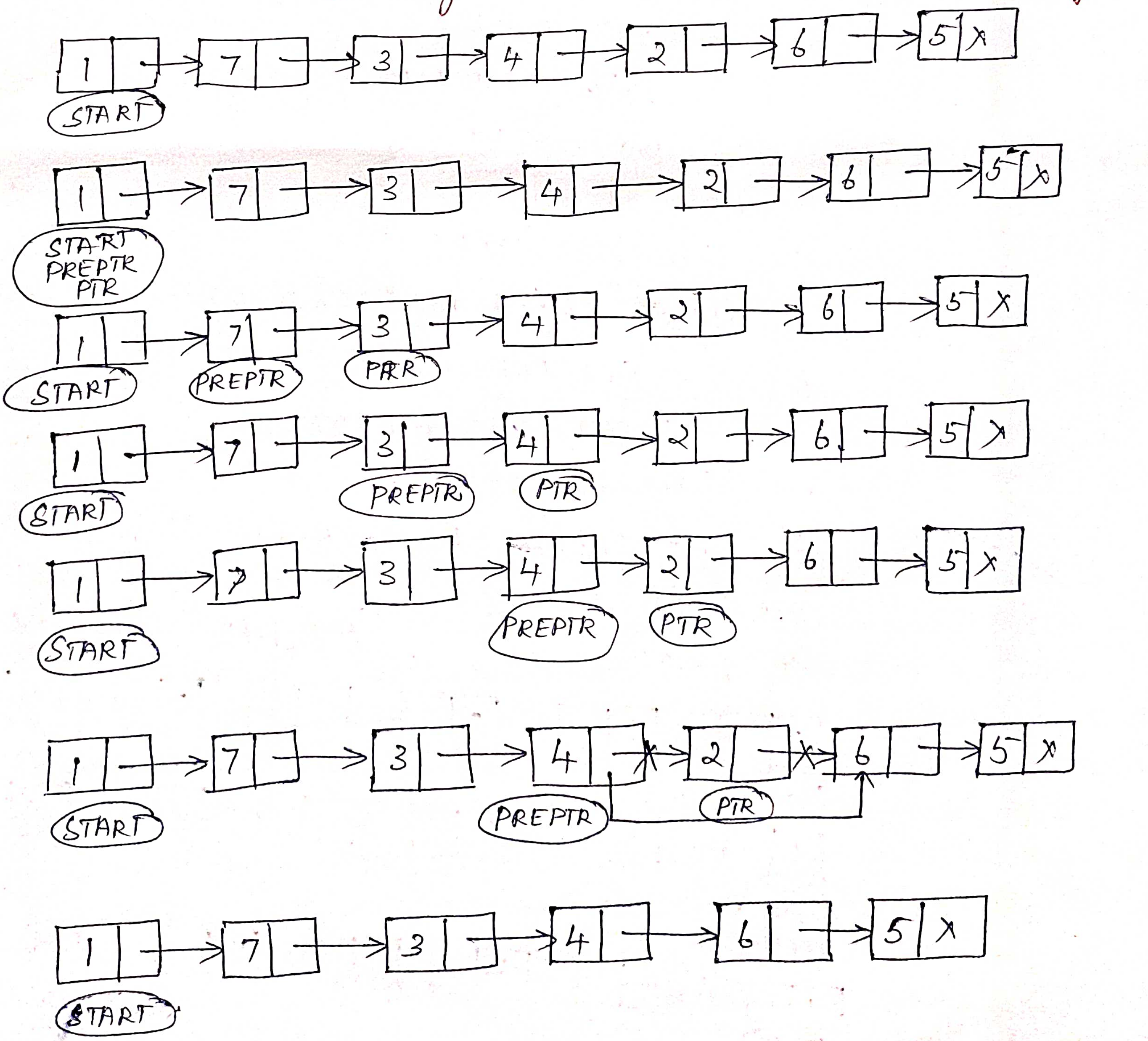
7. FREE PTR

8. EXIT

// Deleting from end

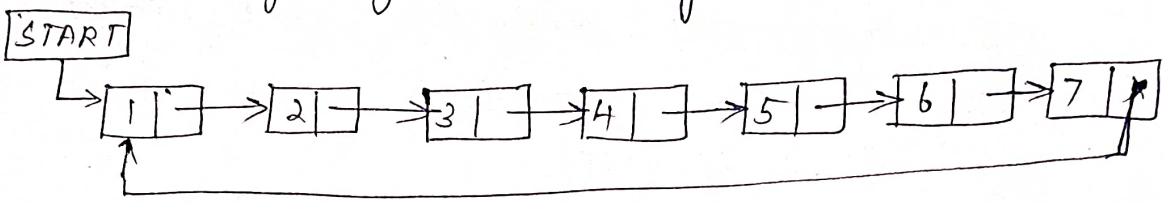


1. IF START = NULL
   Write UNDERFLOW
   Go To Step 10
   [END OF IF]
2. SET PTR = START
3. SET PREPTR = PTR
4. Repeat steps 5 and 6 while PREPTR -> DATA != NUM
5. SET PREPTR = PTR
6. SET PTR = PTR -> NEXT
   [END OF LOOP]
7. SET TEMP = PTR
8. SET PREPTR -> NEXT = PTR -> NEXT
9. FREE TEMP.
10. EXIT // Deleting the node that succeeds the node containing 4

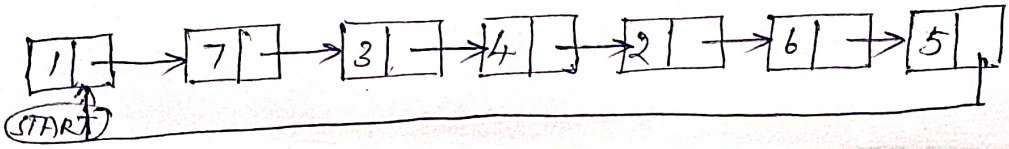


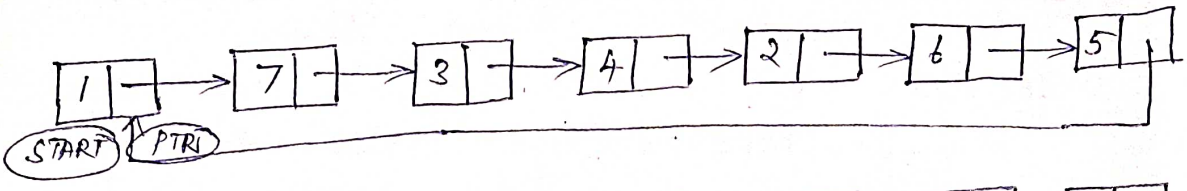
CIRCULAR LINKED LISTS (CLL)

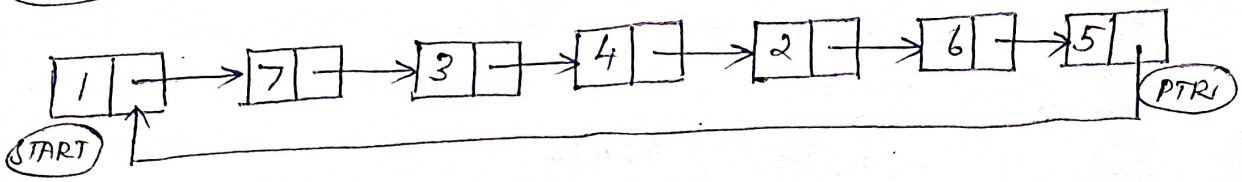
* The last node contains a pointer to the first node of the list.
* While traversing a CLL, traversing can begin at any node, in any direction, forward or backward until the same start node is reached.
* It has no beginning and no ending.

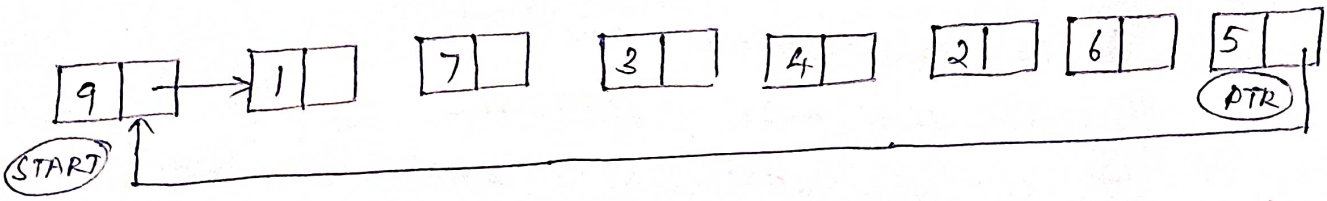


INSERTING A NEW NODE IN A CIRCULAR LINKED LIST







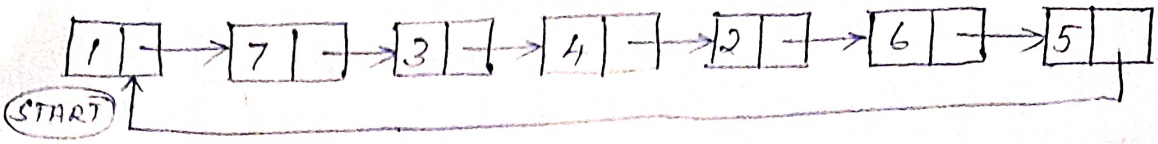


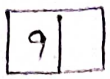
// Inserting a node at the beginning of a Circular Linked List.

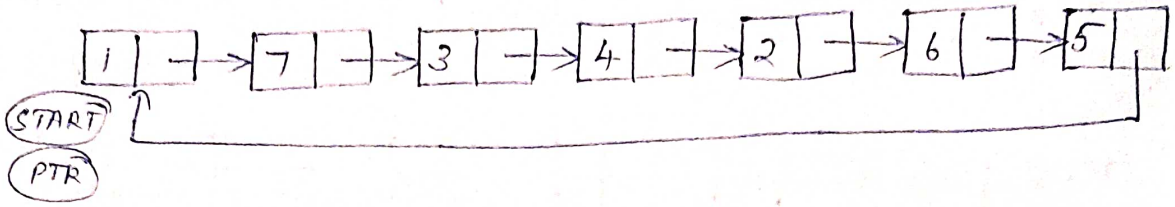
1. IF AVAIL = NULL
   Write OVERFLOW
   Go To Step 11
   [END OF IF]
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL -> NEXT
4. SET NEW_NODE -> DATA = VAL
5. SET PTR = START
6. Repeat Step 7 while PTR -> NEXT != START
7. PTR = PTR -> NEXT
   [END OF LOOP]
8. SET NEW_NODE -> NEXT = START
9. SET PTR -> NEXT = NEW_NODE
10. SET START = NEW_NODE
11. EXIT.

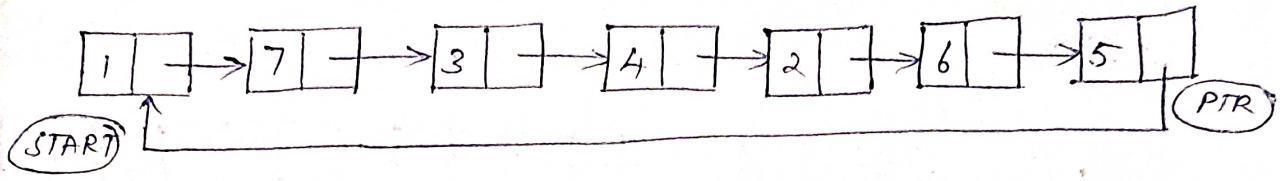
// Inserting a Node at the End of a Circular Linked List

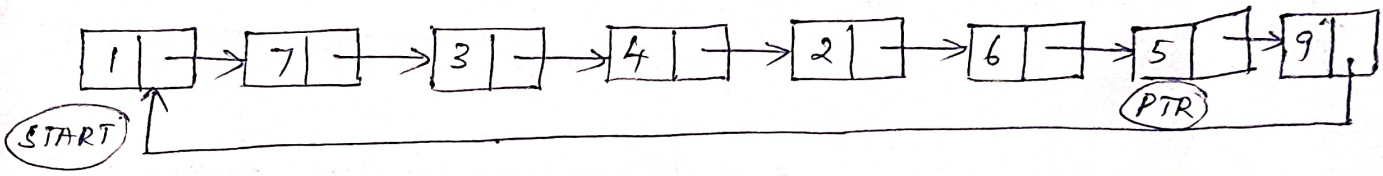
1. IF AVAIL = NULL.
   Write OVERFLOW
   Go To Step 10
   [END OF IF]
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL -> NEXT
4. SET NEW_NODE -> DATA = VAL
5. SET NEW_NODE -> NEXT = START
6. SET PTR = START
7. Repeat Step 8 while PTR -> NEXT != START
8. SET PTR = PTR -> NEXT
   [END OF LOOP]
9. SET PTR -> NEXT = NEW_NODE
10. EXIT







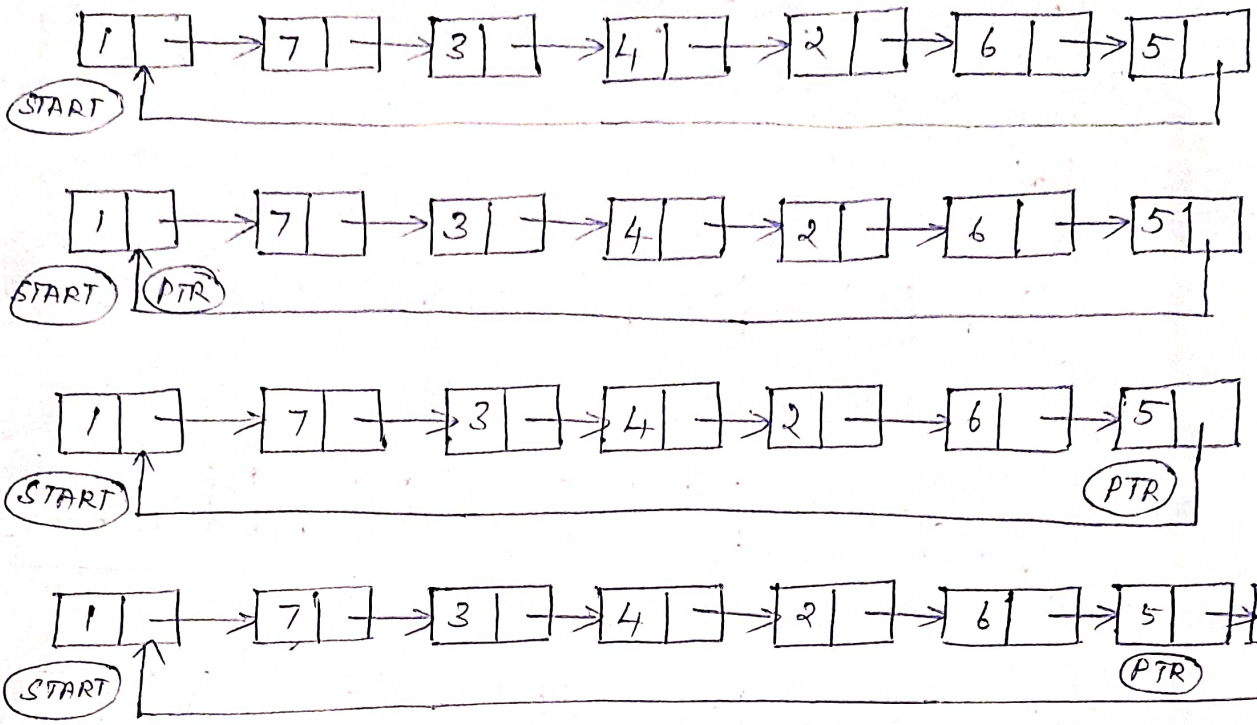




DELETING A NODE FROM A CIRCULAR LINKED LIST

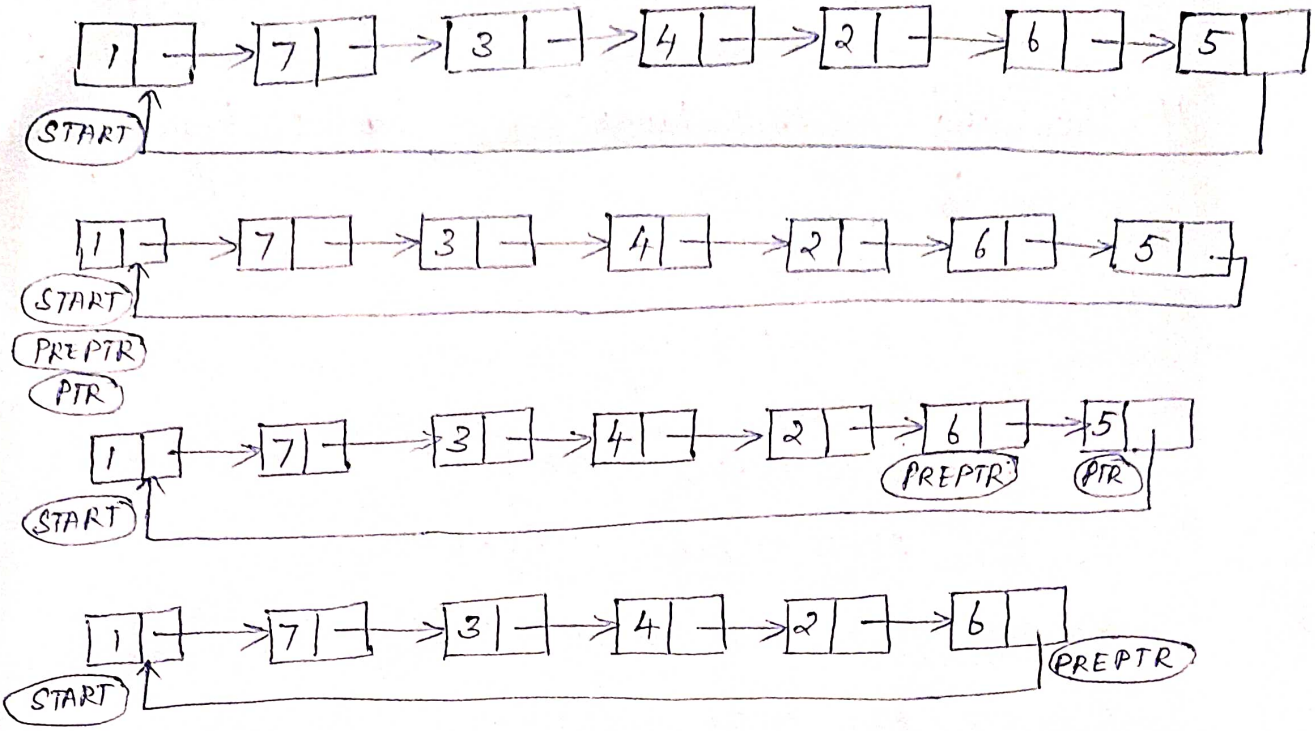
// Deleting the first node from a Circular Linked List

1. IF START = NULL
   Write UNDERFLOW
   Go to Step 8
   [END OF IF]
2. SET PTR = START
3. Repeat Step 4 while PTR → NEXT != START
4. SET PTR = PTR → NEXT
   [END OF LOOP]
5. SET PTR → NEXT = START → NEXT
6. FREE START
7. SET START = PTR → NEXT
8. EXIT



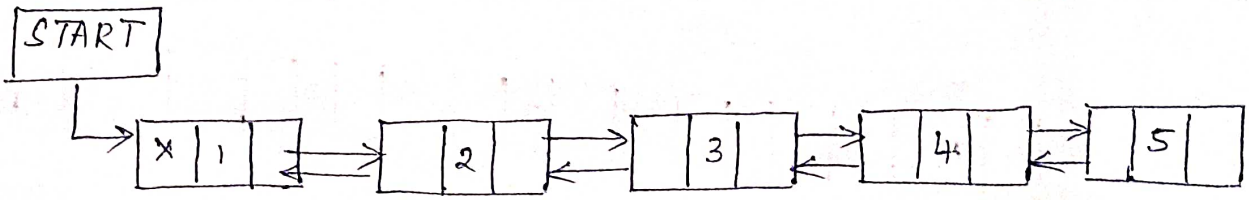
// Deleting the Last Node from a Circular Linked List

1. IF START = NULL
   Write UNDERFLOW
   Go to Step 8
   [END OF IF]
2. SET PTR = START
3. Repeat Steps 4 and 5 while PTR → NEXT != START
4. SET PREPTR = PTR
5. SET PTR = PTR → NEXT
   [END OF LOOP]
6. SET PREPTR → NEXT = START
7. FREE PTR
8. EXIT.



DOUBLY LINKED LISTS

* It is a two way linked list which contains a pointer to the next as well as the previous node in the sequence.
* It consists of 3 parts : data, pointer to the next node and a pointer to the previous node.



* In C, the structure of a doubly linked list is.

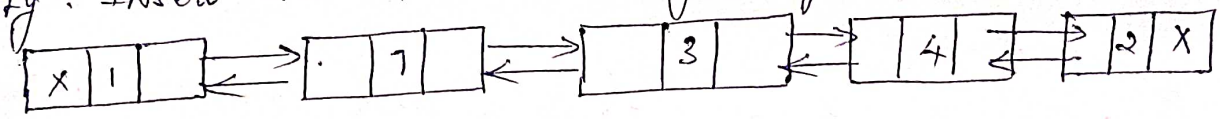
```
struct node
{
    struct node * prev;
    int data;
    struct node * next;
};
```

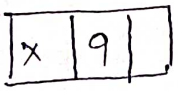
INSERTING A NEW NODE IN A DOUBLY LINKED LIST (DLL)

// Inserting a New Node in a DLL - Beginning

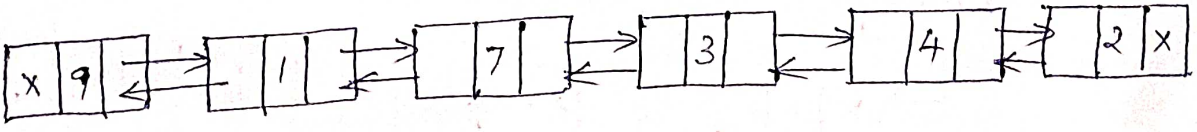
1. IF AVAIL = NULL
   Write OVERFLOW
   Go to Step 9
   [END OF IF]
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL → NEXT
4. SET NEW_NODE → DATA = VAL
5. SET NEW_NODE → PREV = NULL
6. SET NEW_NODE → NEXT = START
7. SET START → PREV = NEW_NODE
8. SET START = NEW_NODE
9. EXIT

Eg : Insert Val = 9 in the beginning.





New Node Allocation

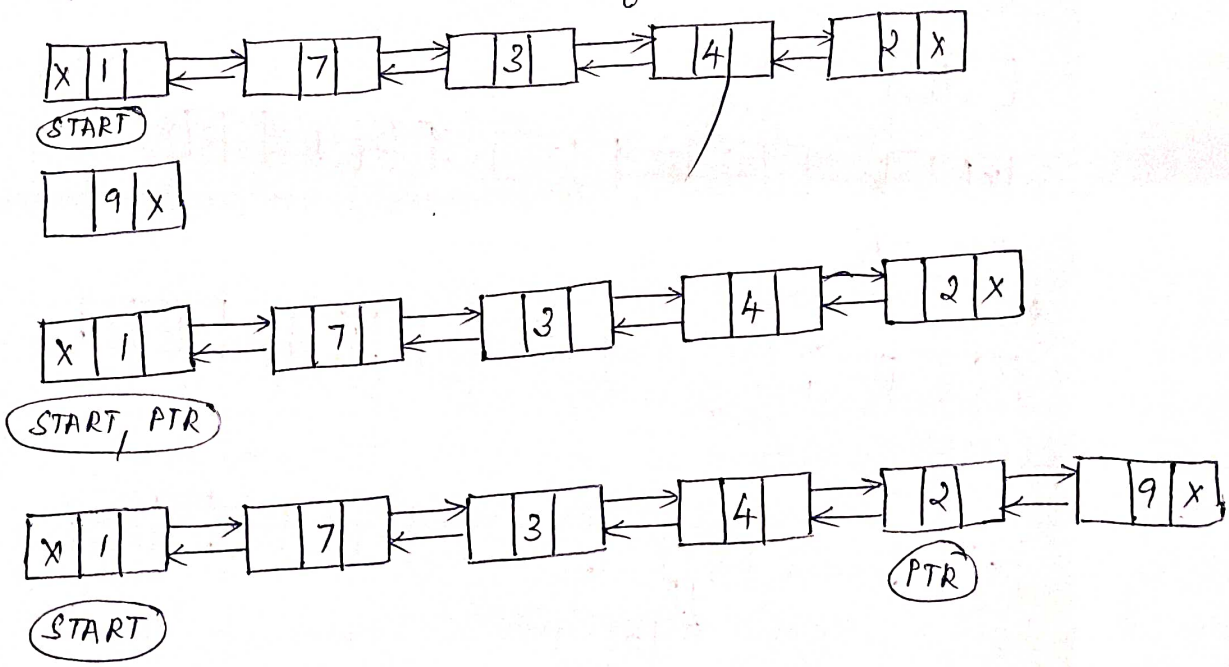


// Inserting a Node at the end of DLL

1. IF AVAIL = NULL
   Write OVERFLOW
   Go to Step 11
   [END OF IF]
2. SET NEW_NODE = AVAIL

3. SET AVAIL = AVAIL → NEXT
4. SET NEW_NODE → DATA = VAL
5. SET NEW_NODE → NEXT = NULL
6. SET PTR = START
7. Repeat Step 8 while PTR → NEXT != NULL
8. SET PTR = PTR → NEXT
   [END OF LOOP]
9. SET PTR → NEXT = NEW_NODE
10. SET NEW_NODE → PREV = PTR
11. EXIT

Eg: Insert 9 at the end of DLL



11 Inserting a node after a given node in DLL

1. IF AVAIL = NULL
   Write OVERFLOW
   Go to Step 12
   [END OF IF]
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL → NEXT
4. SET NEW_NODE → DATA = VAL

5. SET PTR = START

6. Repeat Step 7 while PTR → DATA != NUM

7. SET PTR = PTR → NEXT

[END OF LOOP]

8. SET NEW_NODE → NEXT = PTR → NEXT

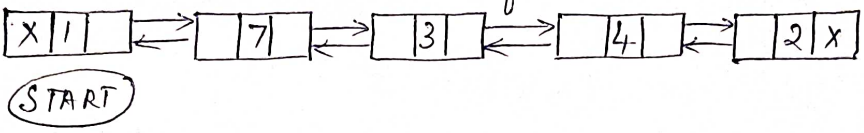
9. SET NEW_NODE → PREV = PTR

10. SET PTR → NEXT = NEW_NODE

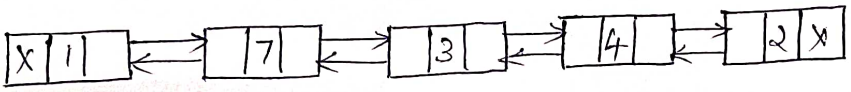
11. SET PTR → NEXT → PREV = NEW_NODE

12. EXIT

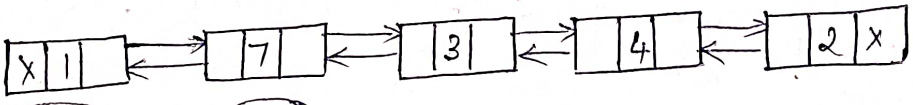
Insert 9 after 7.



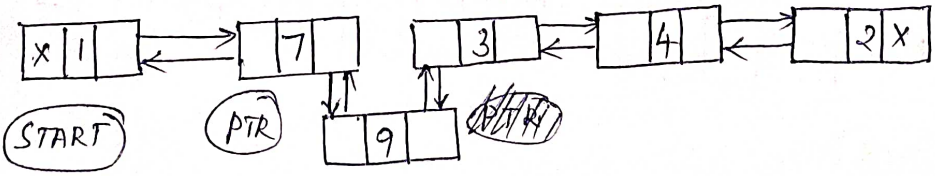
START



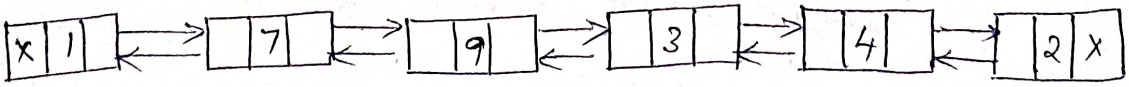
START, PTR



START PTR



START PTR

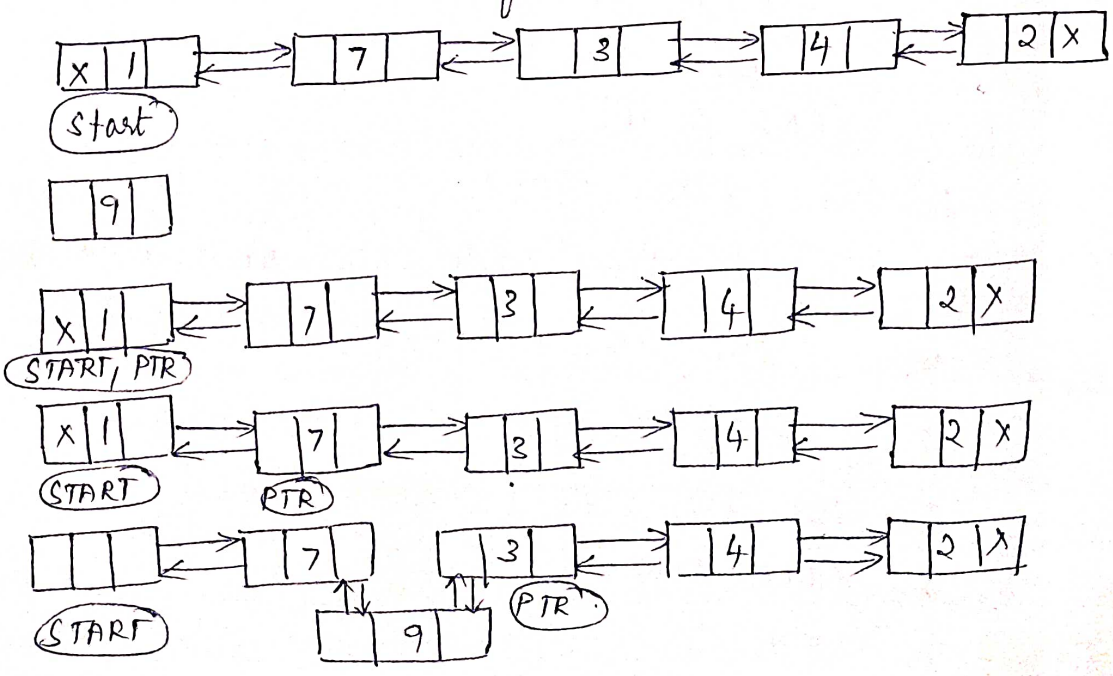


START

// Inserting a node before a given node in DLL

1. IF AVAIL = NULL
   Write OVERFLOW
   Go to Step 12
   [END OF IF]
2. SET NEW_NODE = AVAIL
3. SET AVAIL = AVAIL → NEXT
4. SET NEW_NODE → DATA = VAL
5. SET PTR = START
6. Repeat Step 7 while PTR → DATA != NUM
7. SET PTR = PTR → NEXT
   [END OF LOOP]
8. SET NEW_NODE → NEXT = PTR
9. SET NEW_NODE → PREV = PTR → PREV
10. SET PTR → PREV = NEW_NODE
11. SET PTR → PREV → NEXT = NEW_NODE
12. EXIT

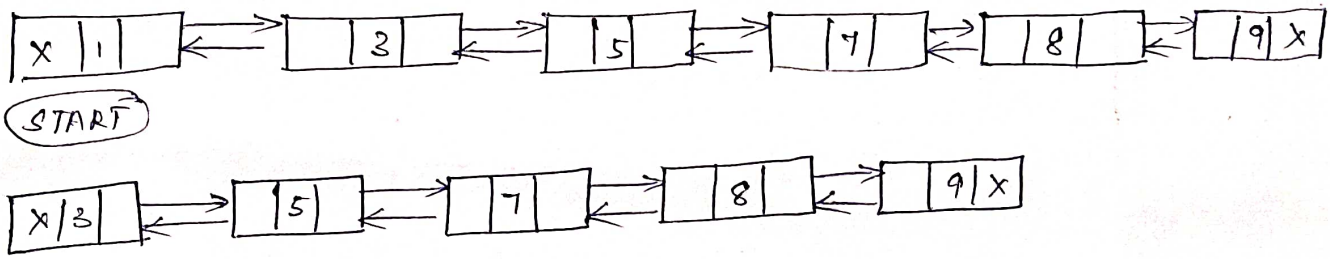
Insert a new node before 3 ,



DELETING A NODE FROM A DOUBLY LINKED LIST

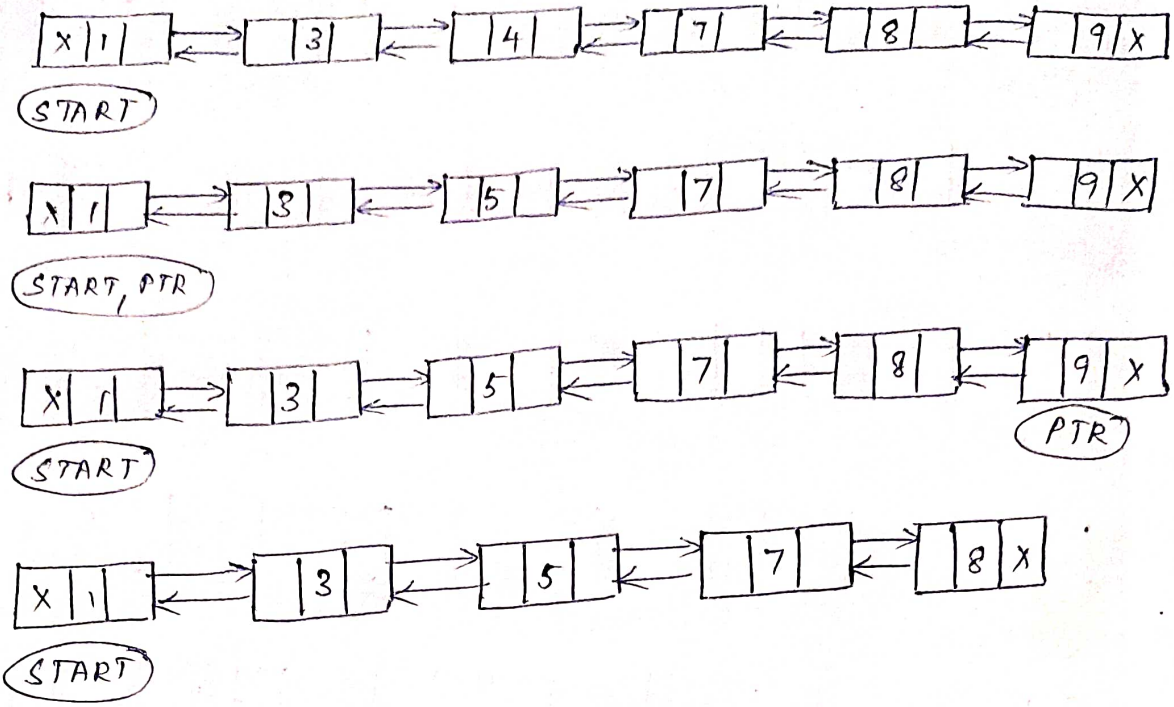
// Deleting the first node from a DLL

1. IF START = NULL
   Write UNDERFLOW
   Go to Step 6
   [END OF IF]
2. SET PTR = START
3. SET START = START → NEXT
4. SET START → PREV = NULL
5. FREE PTR
6. EXIT



// Deleting the last node from a DLL

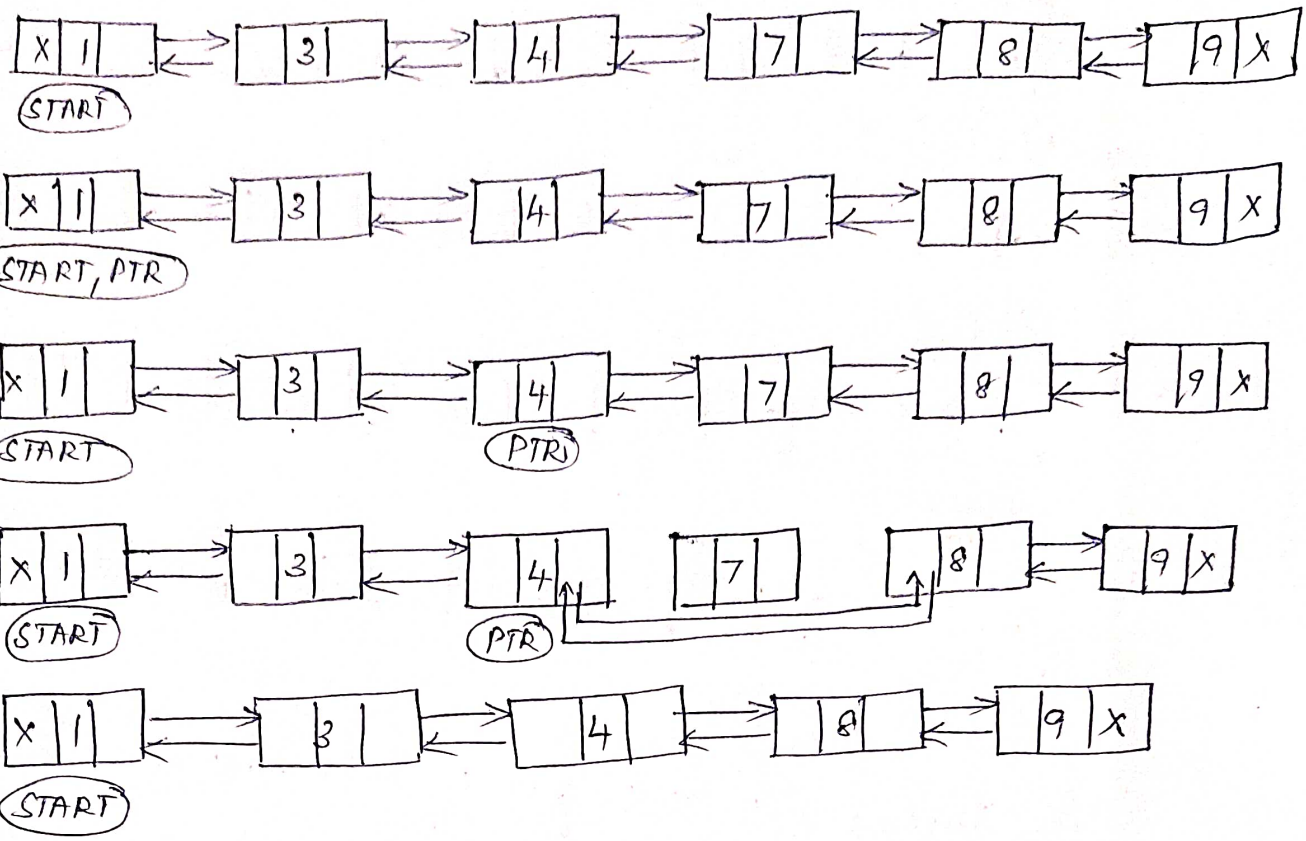
1. IF START = NULL
   Write UNDERFLOW
   Go to Step 7
   [END OF IF]
2. SET PTR = START
3. Repeat Step 4 while PTR → NEXT != NULL
4. SET PTR = PTR → NEXT
5. [END OF LOOP]
6. SET PTR → PREV → NEXT = NULL
7. FREE PTR
8. EXIT



// Deleting the Node after a given node in a DLL

1. IF START = NULL
   Write UNDERFLOW
   Go to Step 9
   [END OF IF]
2. SET PTR = START
3. Repeat Step 4 while PTR → DATA != NUM
4. SET PTR = PTR → NEXT
   [END OF LOOP]
5. SET TEMP = PTR → NEXT
6. SET PTR → NEXT = TEMP → NEXT
7. SET TEMP → NEXT → PREV = PTR
8. FREE TEMP
9. EXIT

Eg: Delete the node that succeeds 4.



// Deleting the node before a given node in a DLL

1. IF START = NULL
   Write UNDER FLOW
   Go to Step 9
   [END OF IF]
2. SET PTR = START
3. Repeat Step 4 while PTR → DATA != NUM
4. SET PTR = PTR → NEXT
   [END OF LOOP]
5. SET TEMP = PTR → PREV
6. SET TEMP → PREV → NEXT = PTR
7. SET PTR → PREV = TEMP → PREV
8. FREE TEMP
9. EXIT

Delete the node proceeding 4.