SYLLABUS OF SEMESTER-I, MCA (Artificial Intelligence and Machine Learning)

Course Code: 24CS60TR1177 Course: Data Structures

UNIT-II

Linked List - Concept of Linked Lists, Types, Operations on Linked lists, concept of Doubly Linked List, Header Linked List. Other Operation & Applications: Reversing a Linked List, Concatenation of Two Lists.

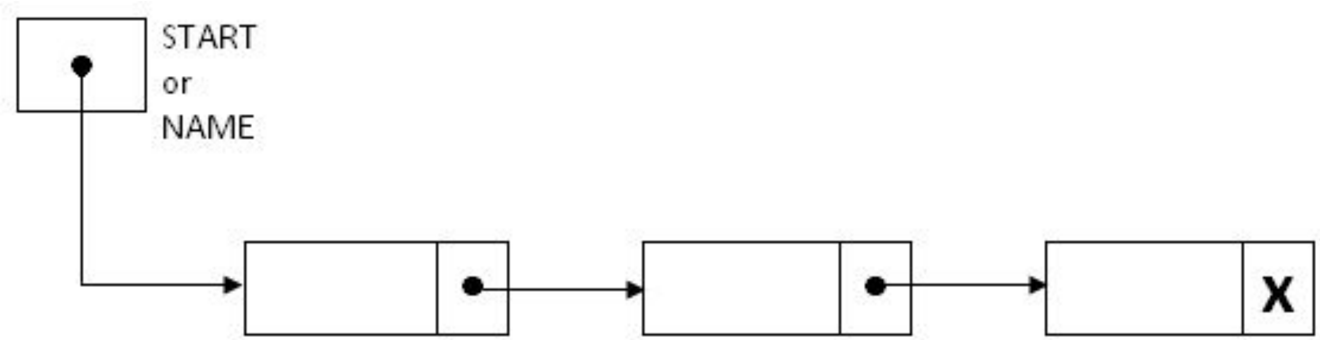
Linked List

❑A linked list is a linear collection of data elements.

❑Data elements are called nodes.

❑Node divided in two parts: first part; ***info*** and second; ***link*** field.

INFO	**LINK**




```
struct Node
{
   int info;
   struct Node *link;
};
```

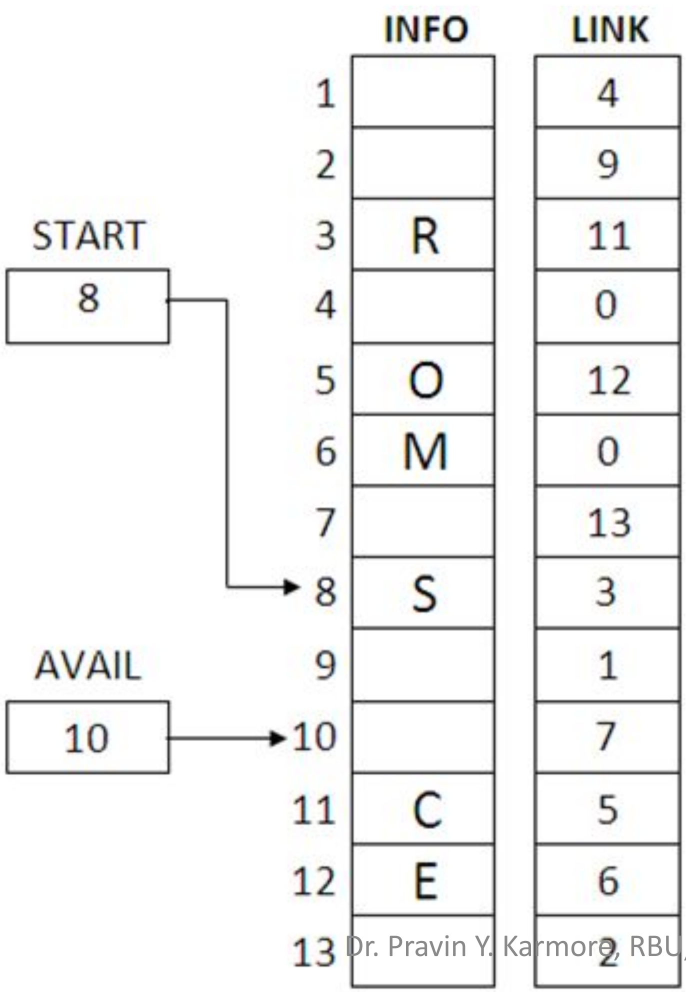
Representation of Linked List in Memory using Linear array

START=8, so INFO[8]=S LINK[8]=3

INFO[3]=R LINK[3]=11

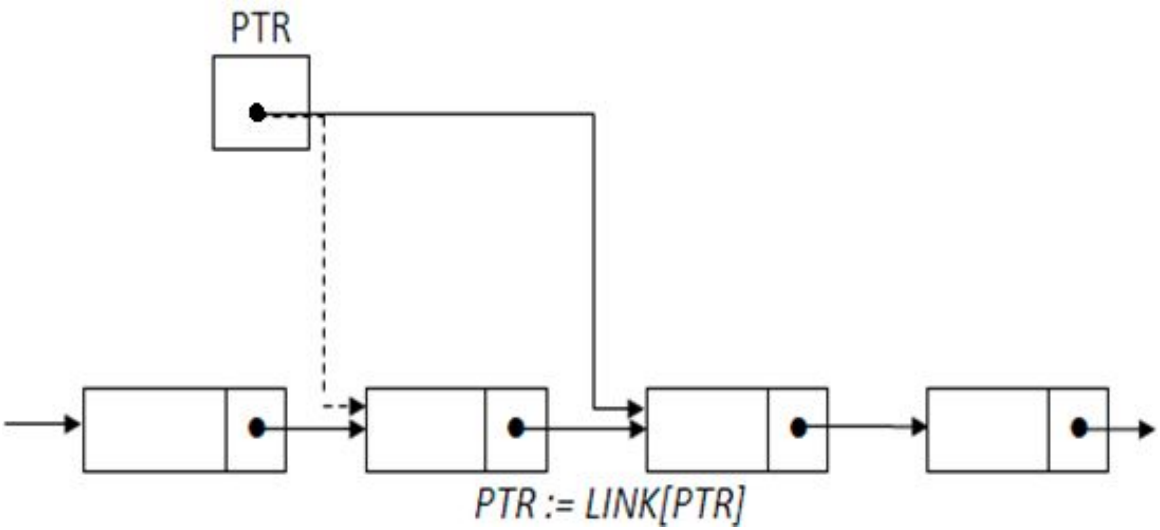
. . . . . .

INFO[6]=M LINK[6]=NULL



Traversing a Linked List

- **LIST** be a linked list stored in linear arrays **INFO** and **LINK**.
- **START** pointing to the first element and **NULL** indicating end of **LIST**.
- **PTR** pointer variable points to the currently processing node .
- **LINK[PTR]** points to the next node to be processed.



Algorithm:

1.Set PTR := START.

2.Repeat steps 3 and 4 while PTR $\neq$ NULL.

3. Apply PROCESS to INFO[PTR].

4. Set PTR := LINK[PTR].

5.Exit.

Searching an Element in Unsorted and Sorted Linked List

SEARCHUSL(INFO, LINK, START, ITEM, LOC)

1.Set PTR := START.

2.Repeat steps 3 while PTR $\neq$ NULL.

3. If ITEM = INFO[PTR], then:

   Set LOC := PTR and Exit.

   Else:

   Set PTR := LINK[PTR].

4.Set LOC := NULL.

5.Exit.

SEARCHSL(INFO, LINK, START, ITEM, LOC)

1.Set PTR := START.

2.Repeat steps 3 while PTR $\neq$ NULL.

3. If ITEM > INFO[PTR], then:

   Set PTR := LINK[PTR].

   Else if ITEM = INFO[PTR], then:

   Set LOC := PTR and Exit.

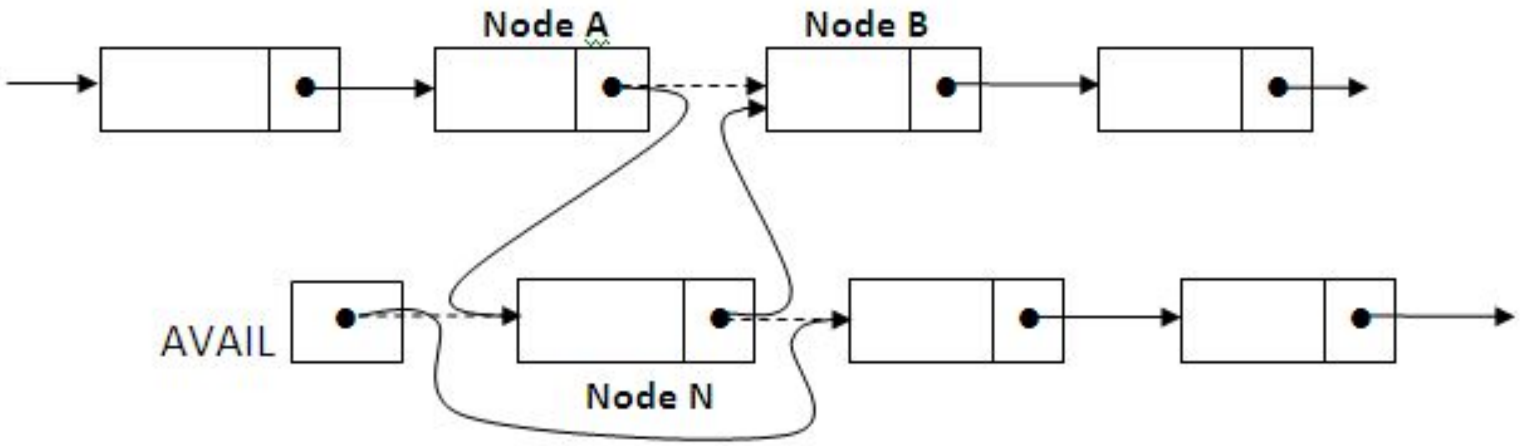
   Else:

   Set LOC := NULL and Exit.

4.Set LOC := NULL.

5.Exit.

Inserting Node into a Linked List



Insert at the Beginning of List:

INSFIRST(INFO, LINK, START, AVAIL, ITEM)

1. If AVAIL = NULL, then Write : OVERFLOW and Exit.
2. Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM .
4. Set LINK[NEW] := START.
5. Set START := NEW.
4. Exit.

Inserting Node into a Linked List …

Inserting after a Given Node:

INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

1. If AVAIL = NULL, then Write : OVERFLOW and Exit.
2. Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM.
4. If LOC = NULL, then:

   Set LINK[NEW] := START and START := NEW.

   Else:

   Set LINK[NEW] := LINK[LOC] and LINK[LOC] := NEW.
5. Exit.

Inserting Node into a Linked List …

Inserting into a Sorted Linked List:

INSERT(INFO, LINK, START, AVAIL, ITEM)

1. Call FINDA(INFO, LINK, START, ITEM, LOC)
2. Call INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)
3. Exit.

FINDA(INFO, LINK, START, ITEM, LOC)

1. If START = NULL, then: Set LOC := NULL and Return
2. If ITEM < INFO[START], then: Set LOC := NULL and Return
3. Set SAVE := START and PTR := LINK[START]
4. Repeat Steps 5 and 6 while PTR $\neq$ NULL
5. If ITEM $\leq$ INFO[PTR], then:
   Set LOC := SAVE, and Return
6. Set SAVE := PTR and PTR := LINK[PTR]
7. Set LOC := SAVE
8. Return

```
struct node
{   int data ;
    struct node *link;
}*start;

/* adds a node at the end of a linked list */
void append (int num ) {
   struct node *temp, *r ;
   if ( start == NULL ){
        temp = malloc ( sizeof ( struct node ) ) ;
        temp -> data = num ;
        temp -> link = NULL ;
        start = temp ;
}
else {
         temp = start ;
        /* go to last node */
        while ( temp -> link != NULL )
        temp = temp -> link ;
        /* add node at the end */
        r = malloc ( sizeof ( struct node ) ) ;
        r -> data = num ;
        r -> link = NULL ;
        temp -> link = r ;
} }
```

```
/* adds a new node at the beginning of the linked list */
void addatbeg (int num )
{
        struct node *t ;
        t = malloc ( sizeof ( struct node ) ) ;
        t -> data = num ;
        t -> link = start ;
        start = t ;
}
```

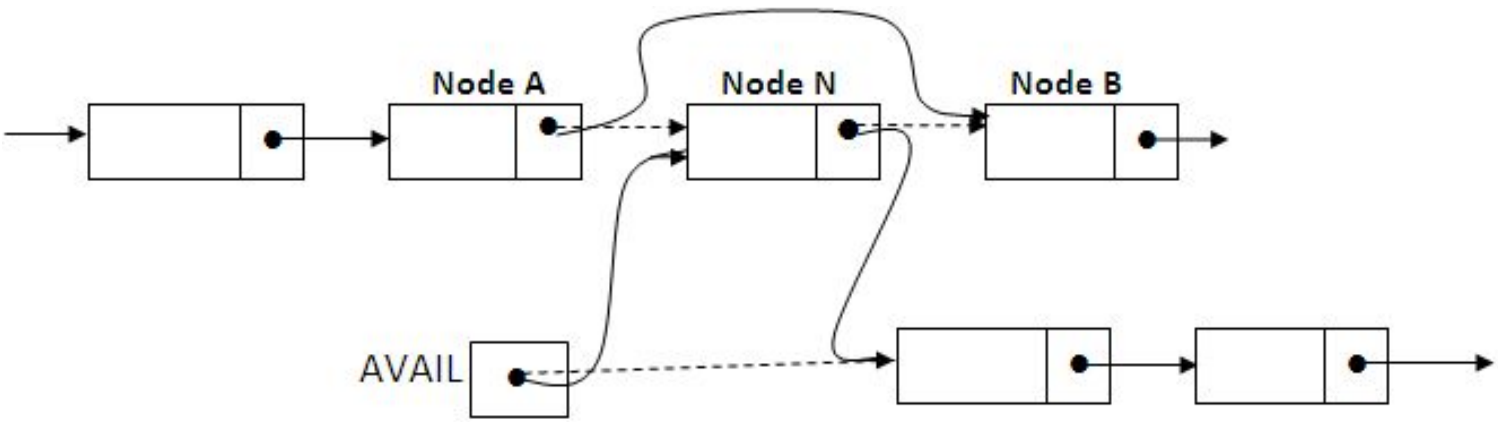
```c
/* adds a new node after the specified number of nodes */
void addafter (int num, int loc ) {
        struct node *temp, *r ;
        int i ;
        temp = start ;

        for ( i = 0 ; i < loc ; i++ )
        {
                temp = temp -> link ;
                    /* if end of linked list is encountered */
            if ( temp->link == NULL )
            {
            printf ( "\nThere are less than %d elements in list", loc ) ;
            break;
            }
        }
        /* insert new node */
        r = malloc ( sizeof ( struct node ) ) ;
        r -> data = num ;
        r -> link = temp -> link ;
        temp -> link = r ;
}
```

```c
/* deletes the specified node from the linked list */
void delnode ( int num )
{
  struct node *old, *temp ;
  temp = start ;
  while ( temp != NULL )
  {
if ( temp -> data == num )
{
/* if node to be deleted is the first node in the linked list */
          if ( temp == start )
              start = temp -> link ;
          else
              old -> link = temp -> link ;

   free ( temp ) ;
   printf ( "\nElement successfully deleted") ;
   return;
    }
else
/* traverse the linked list till the last node is  reached */
{
        old = temp ;
         /* old points to the previous node */
        temp = temp -> link ;
         /* go to the next node */
 }
}
printf ( "\nElement %d not found", num ) ;
}
```

Deleting Node from a Linked List



Deleting the Node following a Given Node:

DEL(INFO, LINK, START, AVAIL, LOC, LOCP)

1.If LOCP = NULL, then:

Set START := LINK[START]

Else:

Set LINK[LOCP] := LINK[LOC]

2.Set LINK[LOC] := AVAIL and AVAIL := LOC

3.Exit

Deleting Node from a Linked List …

Deleting the Node with a Given ITEM of Information:

DELETE(INFO, LINK, START, AVAIL, ITEM)

1. Call FINDB(INFO, LINK, START, ITEM, LOC, LOCP)
2. If LOC = NULL, then: Print “ITEM not in list”
3. If LOCP = NULL, then:

   Set START := LINK[START]

   Else:

   Set LINK[LOCP] := LINK[LOC]
4. Set LINK[LOC] := AVAIL and AVAIL := LOC
5. Exit

Deleting Node from a Linked List …

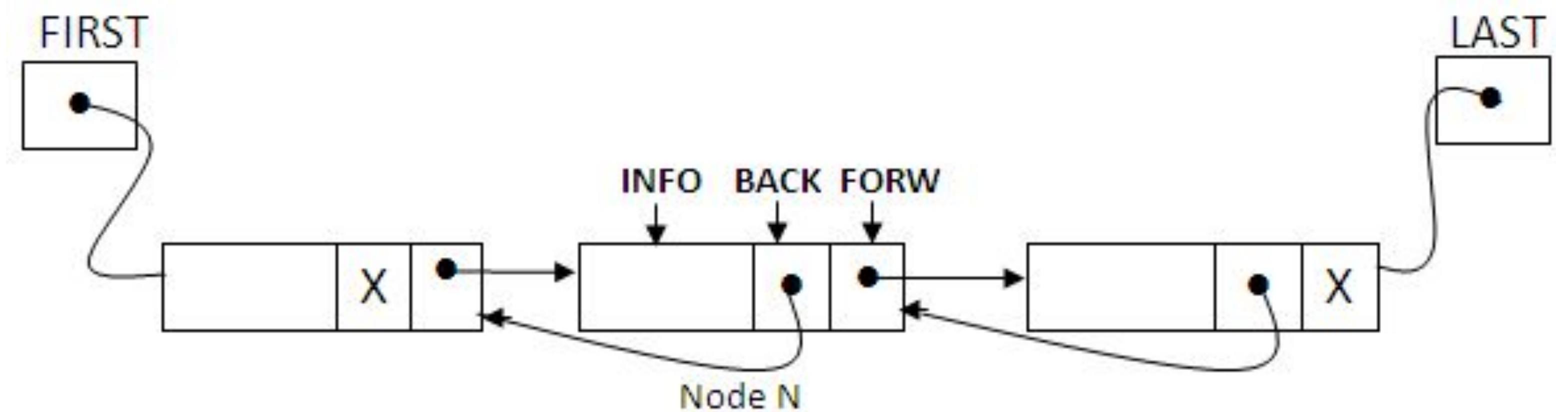
FINDB(INFO, LINK, START, ITEM, LOC, LOCP)

1. If START = NULL, then: Set LOC := NULL , LOCP := NULL and Return
2. If INFO[START] = ITEM, then: Set LOC := START , LOCP := NULL and Return
3. Set SAVE := START and PTR := LINK[START]
4. Repeat Steps 5 and 6 while PTR $\neq$ NULL
5. If ITEM = INFO[PTR], then:

   Set LOC := PTR, LOCP := SAVE and Return
6. Set SAVE := PTR and PTR := LINK[PTR]
7. Set LOC := NULL
8. Return

Two-way Linked List (Doubly Linked List)

- A **two-way** is a linear collection of data elements each node is divided into three parts:
  1. **INFO** field contains the data.
  2. **FORW** field contains the location of the next node in the list.
  3. **BACK** field contains the location of the preceding node in the list.



Algorithm:

1.Set PTR := FIRST.

2.Repeat steps 3 and 4 while PTR $\neq$ NULL.

3. Apply PROCESS to INFO[PTR].

4. Set PTR := FORW[PTR].

5.Exit.

Algorithm:

1.Set PTR := LAST.

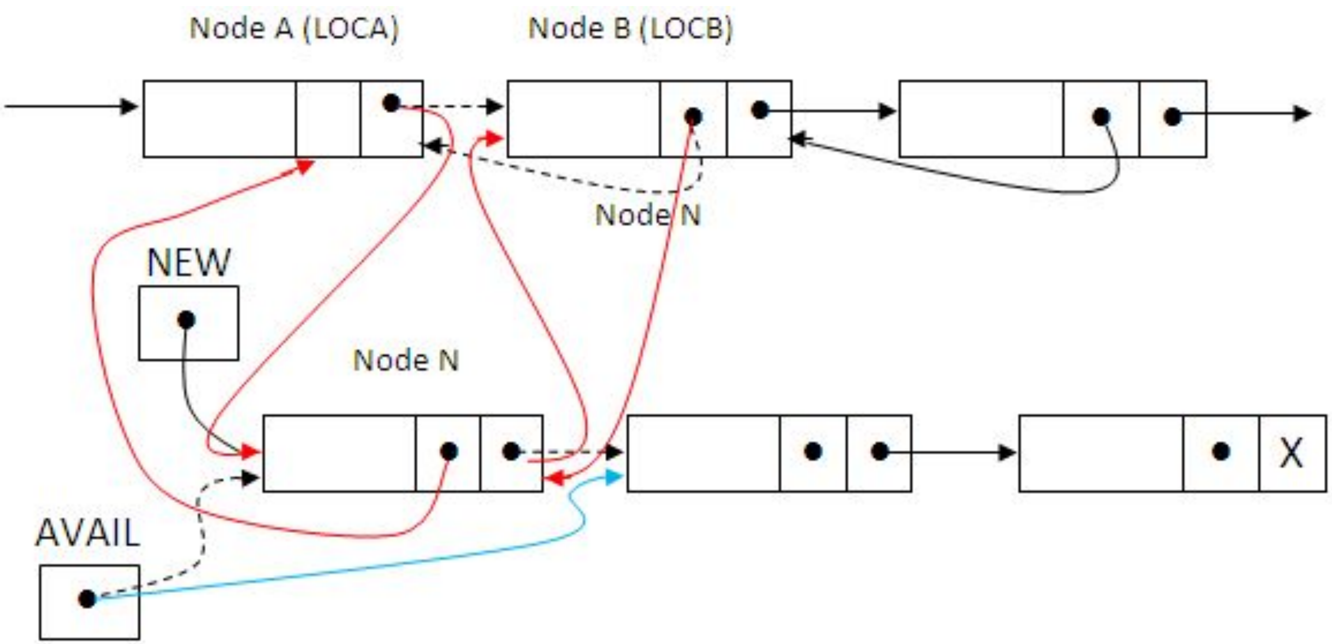
2.Repeat steps 3 and 4 while PTR $\neq$ NULL.

3. Apply PROCESS to INFO[PTR].

4. Set PTR := BACK[PTR].

5.Exit.

Inserting Node into a Two-way Linked List



Insert after a Given Location:

INSTWL(INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

1.If AVAIL = NULL, then Write : OVERFLOW and Exit.

2.Set NEW := AVAIL and AVAIL := FORW[AVAIL].

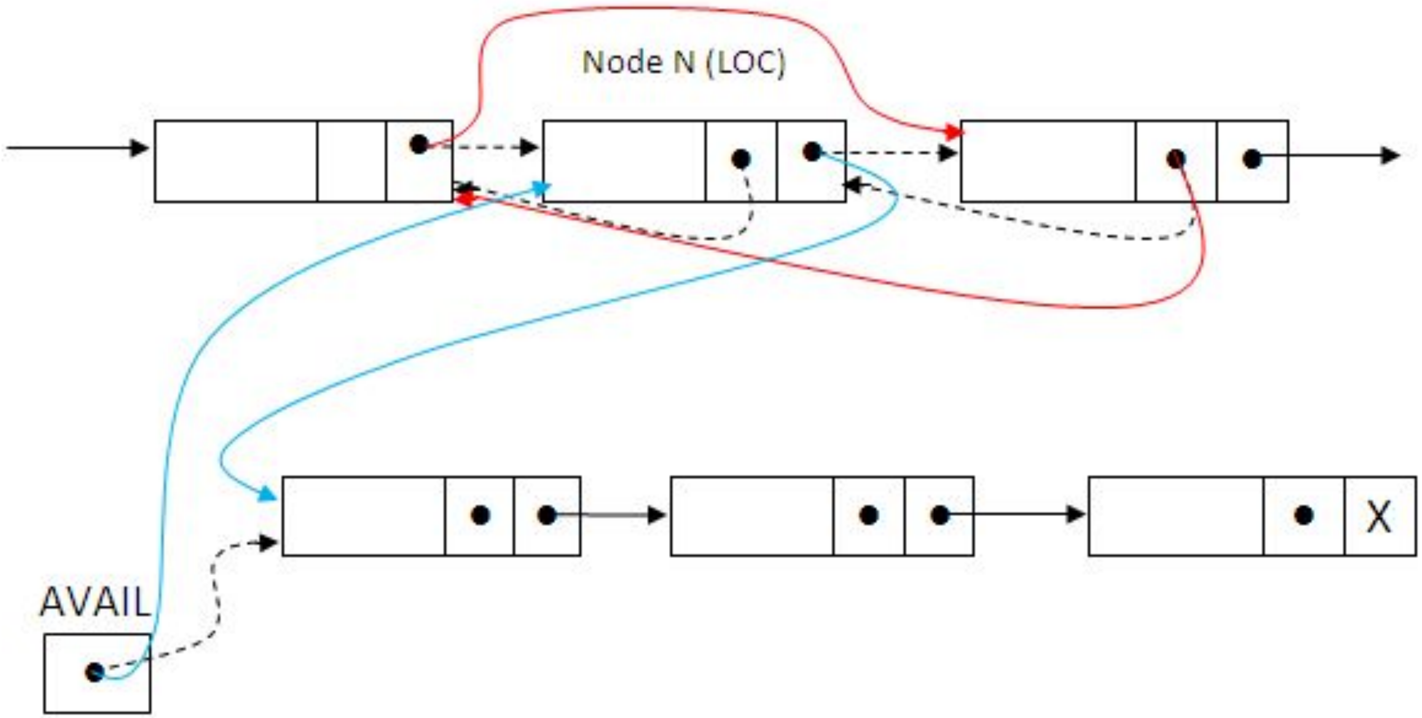
3.Set INFO[NEW] := ITEM .

4.Set FORW[LOCA] := NEW , FORW[NEW] := LOCB

BACK[LOCB] := NEW, BACK[NEW] := LOCA

5.Exit.

Deleting Node from a Two-way Linked List



Deleting Node from a Given Location:

DELTWL(INFO, FORW, BACK, START, AVAIL, LOC)

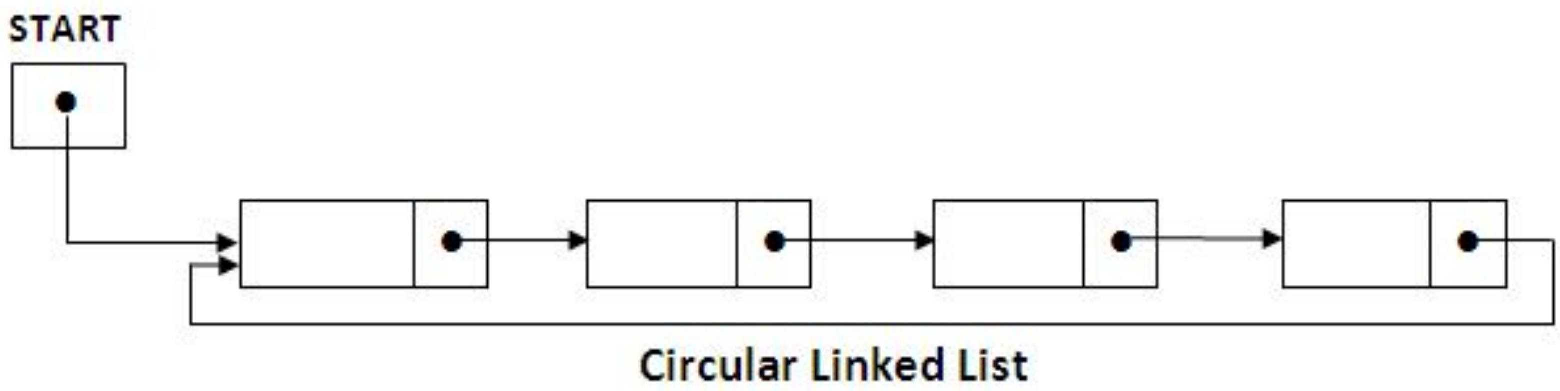
1.Set FORW[BACK[LOC]] := FORW[LOC]

BACK[FORW[LOC] := BACK[LOC]

2.Set FORW[LOC] := AVAIL and AVAIL := LOC

3.Exit

Type of Linked List



Traversing Algorithm:

1.Apply PROCESS to INFO[START].

2.Set PTR := LINK[START].

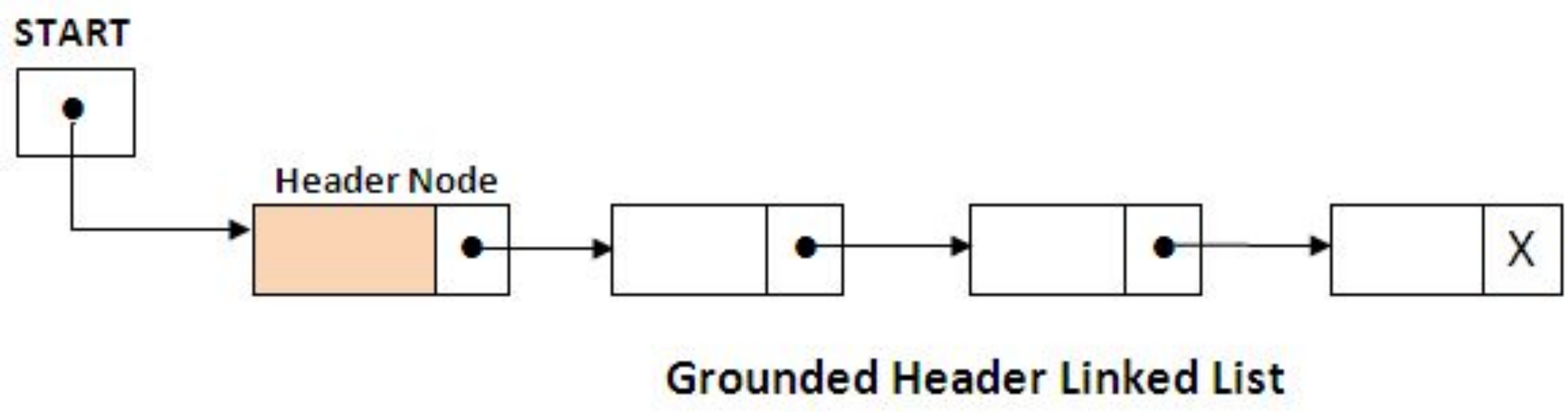
3.Repeat steps 3 and 4 while PTR $\neq$ START.

4. Apply PROCESS to INFO[PTR].

5. Set PTR := LINK[PTR].

6.Exit.

Types of Header Linked List



Traversing Algorithm:

1.Set PTR := LINK[START].

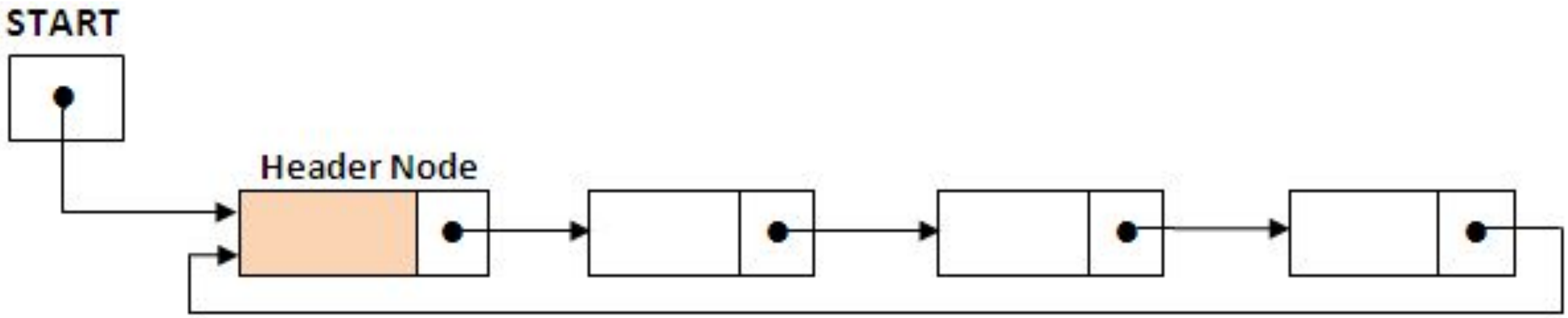
2.Repeat steps 3 and 4 while PTR ≠ NULL.

3. Apply PROCESS to INFO[PTR].

4. Set PTR := LINK[PTR].

5.Exit.

Types of Header Linked List



Circular Header Linked List

Traversing Algorithm:

1.Set PTR := LINK[START].

2.Repeat steps 3 and 4 while PTR $\neq$ START.

3.   Apply PROCESS to INFO[PTR].

4.   Set PTR := LINK[PTR].

5.Exit.

Application of Linked List

Reversing a Linked List

REVERSE(INFO, LINK, START, TEMP, AVAIL)

1. Set TEMP:=NULL, PTR := START.
2. Repeat steps 3 to 8 while PTR $\neq$ NULL.
3. If AVAIL = NULL, then Write : OVERFLOW and Exit.
4. Set NEW := AVAIL and AVAIL := LINK[AVAIL].
5. Set INFO[NEW] := INFO[PTR] .
6. Set LINK[NEW] := TEMP.
7. Set TEMP := NEW.
8. Set PTR := LINK[PTR].
9. Set START := TEMP.
10. End.

Reversing a Linked List

REVERSE1(INFO, LINK, START, TEMP, AVAIL)

1. Set TEMP:=NULL.
2. Repeat steps 3 to 8 while START$\neq$ NULL.
3. If AVAIL = NULL, then Write : OVERFLOW and Exit.
4. Set NEW := AVAIL and AVAIL := LINK[AVAIL].
5. Set INFO[NEW] := INFO[START]
6. Set LINK[NEW] := TEMP and TEMP := NEW
7. Set PTR:= START and START := LINK[START].
8. Set LINK[PTR] := AVAIL and AVAIL := PTR.
9. Set START := TEMP.
10. End.

Concatenation of Two Unsorted Lists

CONCATENATE(INFO, LINK, LIST1, LIST2, LIST3)

1.Set PTR := LIST1, LIST3:=LIST1.

2.Repeat step 3 while PTR $\neq$ NULL.

3. Set SAVE:=PTR and PTR:=LINK[PTR], .

4.Set LINK[SAVE]:=LIST2.

5.End.

Concatenation of Two Sorted Linked Lists

CONCATSORTED(INFO, LINK, LIST1, LIST2, LIST3)

1.If INFO[LIST1] < INFO[LIST2], then

Set LIST3 := LIST1 and LIST1 := LINK[LIST1]

Else

Set LIST3 := LIST2 and LIST2 := LINK[LIST2]

2.Set PTR:=LIST3

3.Repeat steps 4 and 5 while LIST1 ≠ NULL and LIST2 ≠ NULL:

4. If INFO[LIST1] < INFO[LIST2], then

Set LINK[PTR] := LIST1 and LIST1 := LINK[LIST1]

Else

Set LINK[PTR] := LIST2 and LIST2 := LINK[LIST2]

5. Set PTR := LINK[PTR].

6.IF LIST1 ≠ NULL , then: Set LINK[PTR] := LIST1.

7.IF LIST2 ≠ NULL , then: Set LINK[PTR] := LIST2.

8.Exit.

Assignment

OCCUR(INFO, LINK, START, ITEM)

1.Set COUNT:=0

2.Set PTR:= START

3.Repeat Steps 4 and 5 while PTR $\neq$ NULL

4. If ITEM = INFO[PTR], then:

   Set COUNT := COUNT + 1

5. Set SAVE := PTR and PTR := LINK[PTR]

6.PRINT “ Element occurred = “, COUNT, “ times.”

7.Return

Assignment…

ELLIMINATE(INFO, LINK, START, ITEM)

1.Set X:=START , ITEM := INFO[START], SAVE := START and PTR:= LINK[START]

2.Repeat Steps 3 and 4 while PTR ≠ NULL

3. If ITEM = INFO[PTR], then:

a) TEMP:= PTR, LINK[SAVE] := LINK[PTR] and PTR := LINK[PTR]

b) LINK[TEMP] := AVAIL, AVAIL:=TEMP and goto Step 2

4. Set SAVE := PTR and PTR := LINK[PTR]

5. Set PTR:= LINK[X] and ITEM := INFO[PTR]

6.IF PTR ≠ NULL, then: X:= PTR, SAVE:=PTR, PTR := LINK[PTR] and goto Step 2.

7.Return