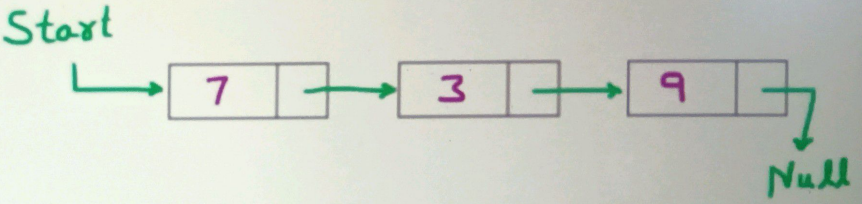
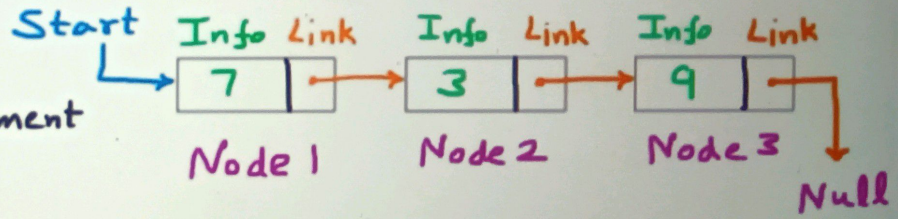
Linked List (One Way List)



- Linear collection of data elements called Nodes
- Each node contain two parts:



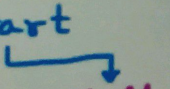
Information (Info)
- Contain information of element

Next (Link) Pointer
- Contain address of next node

Null Pointer
- Signals end of list
- Denoted by X or NULL

Start (Name) Pointer
- Contain address of first node

Null (Empty) List



- List has no node
- Denoted by Null Pointer in Start

Linked List (One Way List)

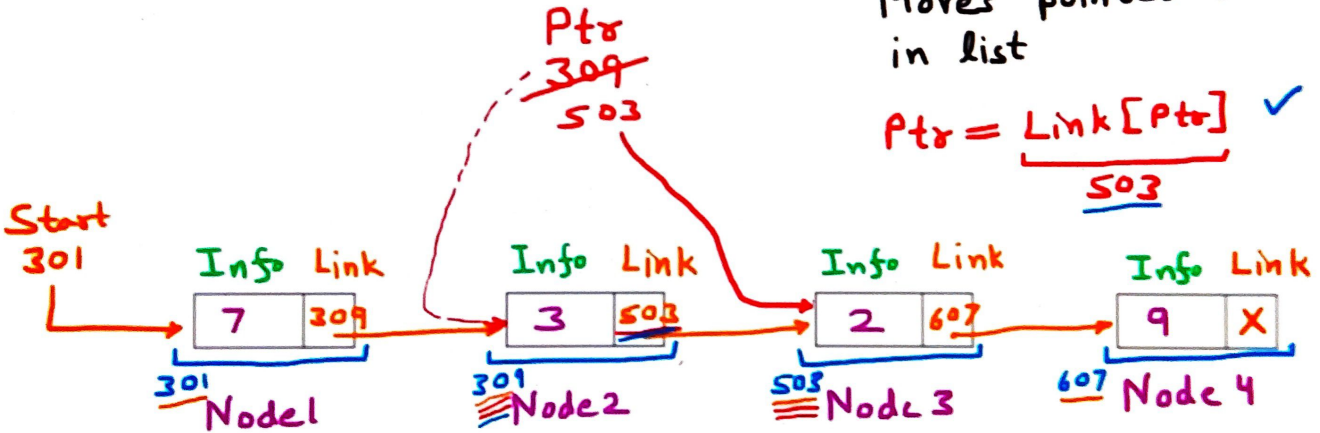
Traversing Linked List

Ptr = Link [Ptr]

Moves pointer to next node in list

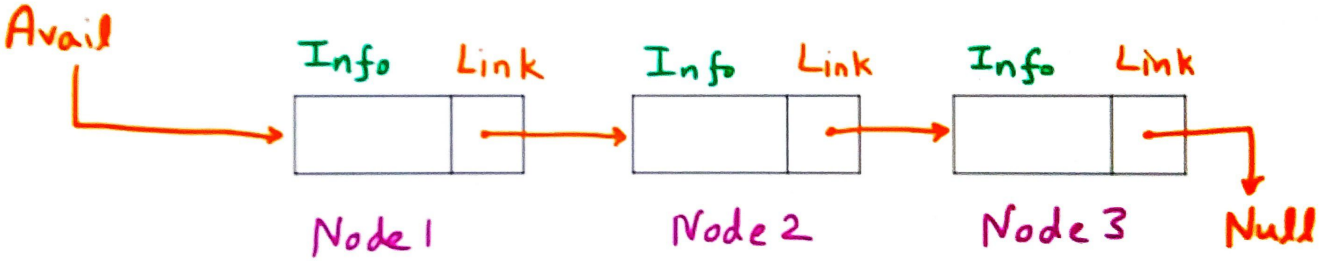
Ptr = Link [Ptr] ✓

503



Available List

- Linked List which contain available memory spaces



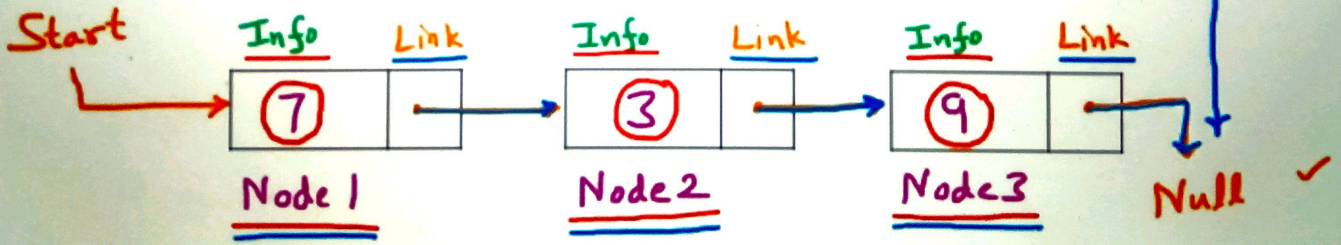
ALGORITHM: Traversing Linked List

Let LIST be a Linked List in Memory. This algorithm traverse LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to node currently being processed.

1. Set PTR := START.
2. Repeat Steps 3 and 4 while PTR != NULL.
3. Apply PROCESS to INFO[PTR].
4. Set PTR := LINK[PTR].
5. Exit.

7 3 9

PTR



ALGORITHM: SEARCH_SORTED_LL (INFO, LINK, START, ITEM, LOC)

LIST is the linked list in memory. The algorithm finds the location LOC of the node where ITEM first appears in LIST, or sets LOC=NULL.

1. Set PTR := START
2. Repeat Step 3 while PTR != 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.

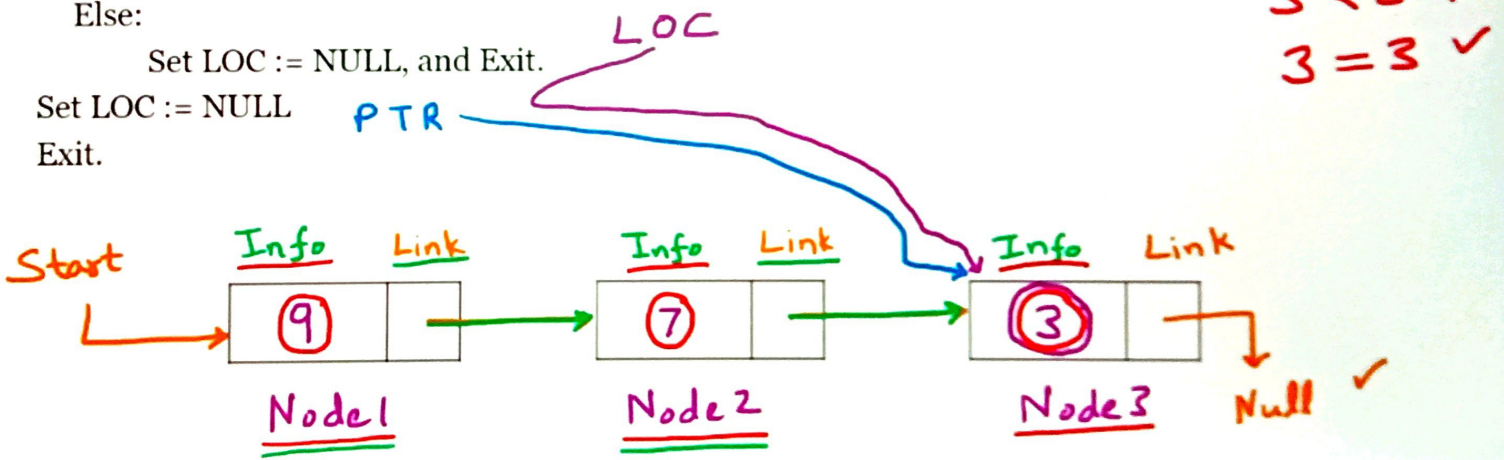
Item ③

3 < 9

3 < 7

3 < 3 X

3 = 3 ✓



ALGORITHM: SEARCH_UNSORTED_LL (INFO, LINK, START, ITEM, LOC)

LIST is the linked list in memory. The algorithm finds the location LOC of the node where ITEM first appears in LIST, or sets LOC=NULL.

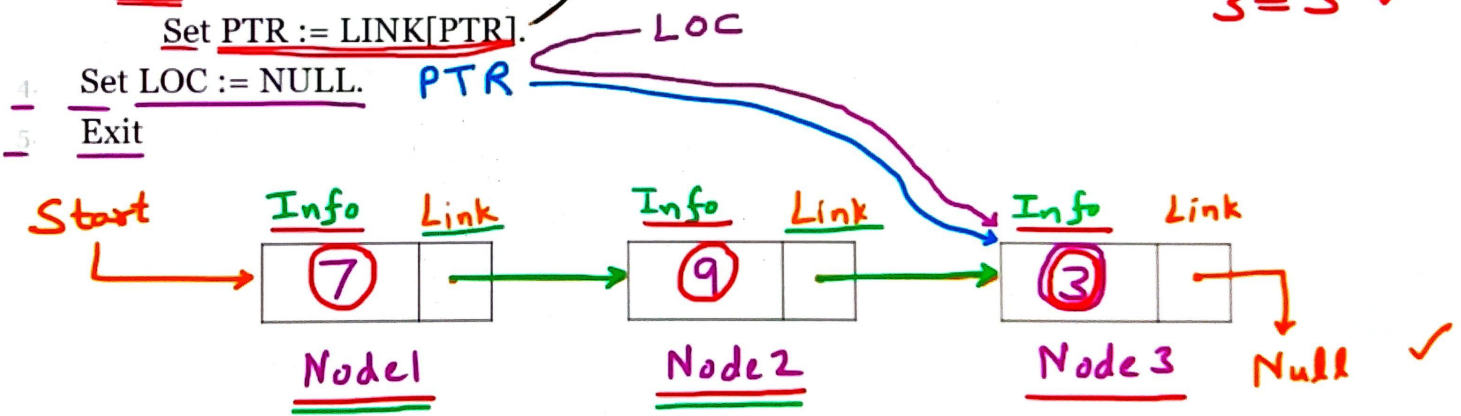
1. Set PTR := START.
2. Repeat Step 3 while PTR != NULL:
3. If ITEM = INFO[PTR], then:
   Set LOC := PTR, and Exit.
   Else:
   Set PTR := LINK[PTR].
4. Set LOC := NULL.
5. Exit

Item (3) ✓

3=7 X

3=9 X

3=3 ✓

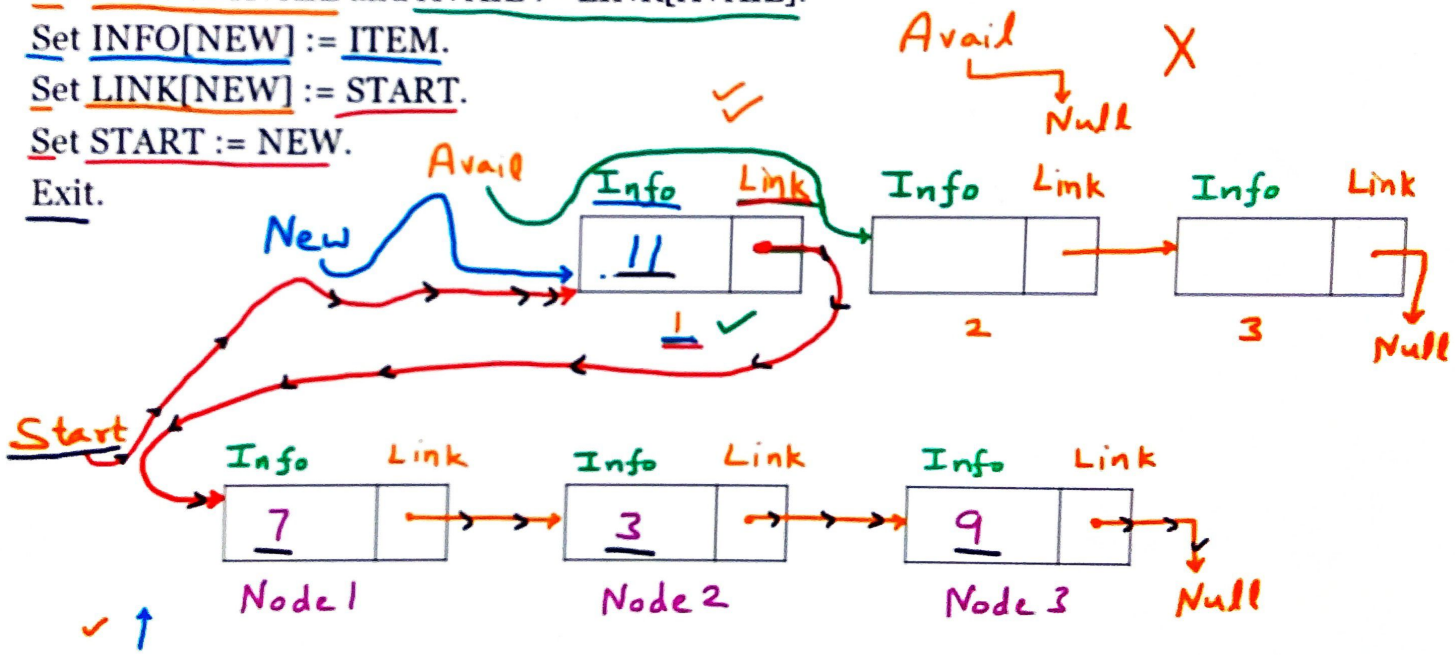


ALGORITHM: INSFIRST (INFO, LINK, START, AVAIL, ITEM)

This algorithm inserts ITEM as the first node in the List.

Item
11

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.
6. Exit.

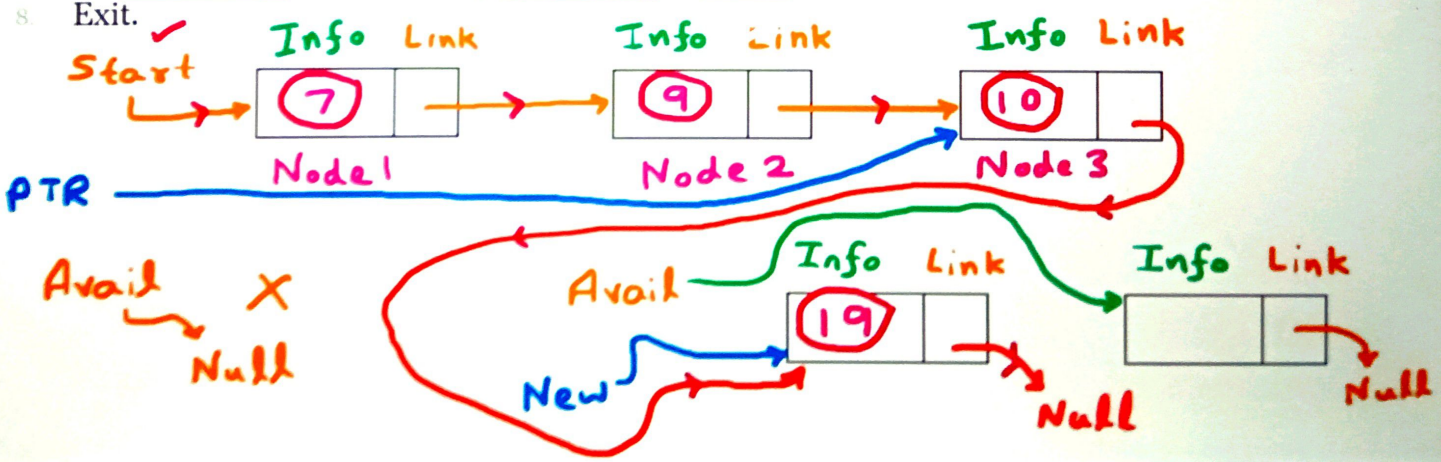


ALGORITHM: INSERT END (INFO, LINK, START, AVAIL, ITEM)

This algorithm inserts ITEM as the END node in the List.

Item
19

1. If AVAIL = NULL, then: Write OVERFLOW and Exit.
2. Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM.
4. Set PTR := START.
5. Repeat Step 6 while LINK[PTR] != NULL
6. Set PTR := LINK[PTR].
7. Set LINK[PTR] := NEW and LINK[NEW] := NULL.
8. Exit.



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

This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.

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:

Case 1 Set LINK[NEW] := START and START := NEW.

Else:

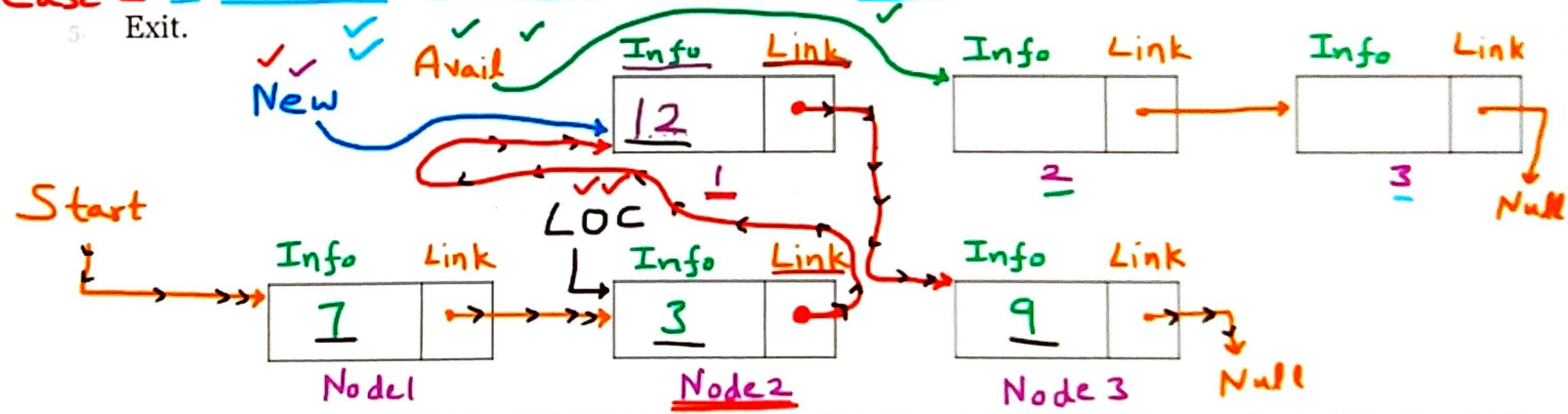
Case 2 Set LINK[NEW] := LINK[LOC] and LINK[LOC] := NEW.

5. Exit.

Item 12

7 3 12 9

Avail → X → Null



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

This algorithm Deletes the node N with location LOC. LOCP is the location of node which precedes N or, when N is the first node, LOCP = NULL.

If LOCP = NULL, then:

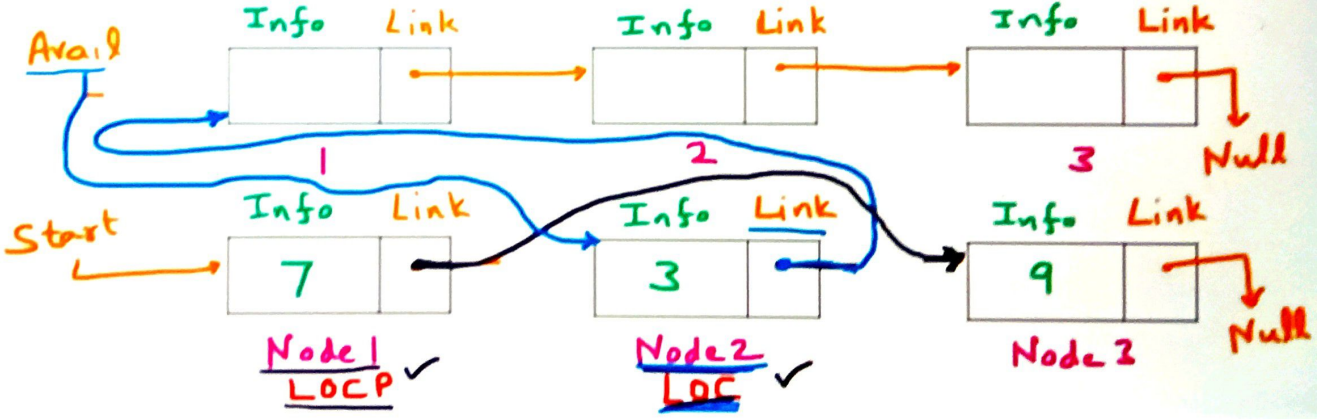
Case 1: Set START := LINK[START].

Else:

Case 2: Set LINK[LOCP] := LINK[LOC].

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

Exit.



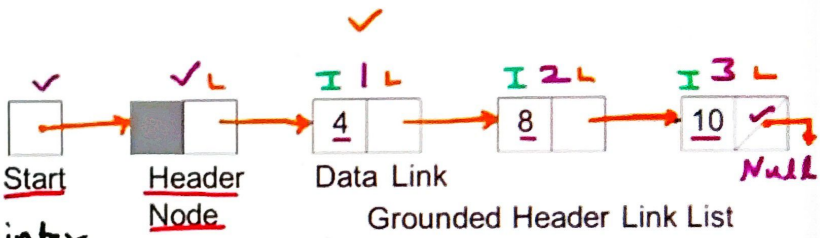
Header Linked List

- Contain special node called Header Node at begining of list

2 Types of Header Linked List

Grounded Header List

- Last node contain NULL pointer
- Term grounded indicate NULL pointer
- Start always point to Header Node
- LINK[START] = NULL means Grounded Header List is Empty



Circular Header List

- Last node point back to Header Node
- LINK[START] = START means Circular Header List is Empty
- Header list always be circular, unless stated
- Header node acting as sentinel indicating end of list