Doubly Linked List (DLL)

Prof. Pujashree Vidap

Objectives

- In this Session, you will learn to:
  - Implement a doubly-linked list
    - Traversing
    - Insertion
    - Deletion

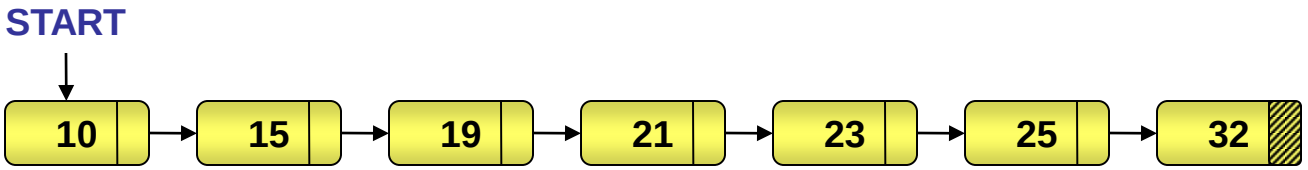
Now consider a case in which you need to display these numbers in a descending order.

How will you solve this problem?

- Consider a sorted list of 100000 numbers stored in a linked list. If you want to display these numbers in ascending order, what will you do?
  - Traverse the list starting from the first node.
  - Each node is linked to the next node in sequence.
  - This means that you can traverse the list in the forward direction only.
  - Such a linked list is called a singly-linked list.
  - To display the numbers in the descending order, you need to reverse the linked list.

Implementing a Doubly-Linked List (Contd.)

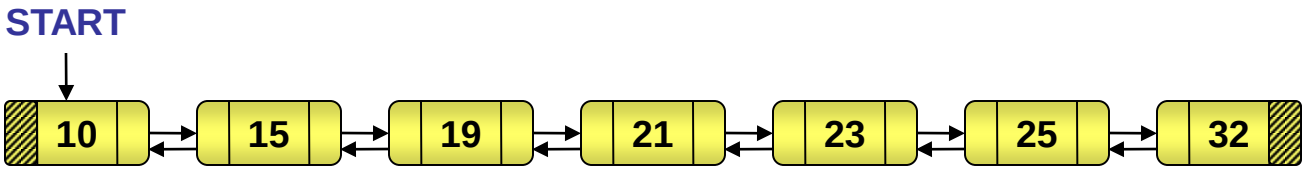
- What is the problem with the previous algorithm?
  - You need to adjust the links
  - Disadvantage of this approach:
    - This approach is inefficient and time consuming for large lists.



- How can you solve this problem?
  - This problem can be solved if each node in the list holds the reference of its preceding node in addition to its next node in sequence.
  - Consider the following list:

Implementing a Doubly-Linked List (Contd.)

- You can introduce an additional field in each node of a singly-linked list, which would hold the address of its previous node.
- Such a type of list is known as a doubly-linked list.



Structure of a doubly-linked list

Representing a Doubly-Linked List

- A Linked list is represented in a program by defining two classes:
  - Node class： In a doubly-linked list, each node needs to store:
    - The information
    - The address of the next node in sequence
    - The address of the previous node
  - DLL class :
    - The address of first node

Traversing a Doubly-Linked List

- Write an algorithm to traverse a doubly linked list in the forward direction.

1. Mark the first node in the list as currentNode.

2. Repeat steps 3 and 4 until currentNode becomes NULL.

3. Display the information contained in the node marked as currentNode.

4. Make currentNode point to the next node in sequence.

Traversing a Doubly-Linked List (Contd.)

- Write an algorithm to traverse a doubly linked list in the backward direction.

1. Mark the last node in the list as currentNode.
2. Repeat steps 3 and 4 until currentNode becomes NULL.
3. Display the information contained in the node marked as currentNode.
4. Make currentNode point to the node preceding it.

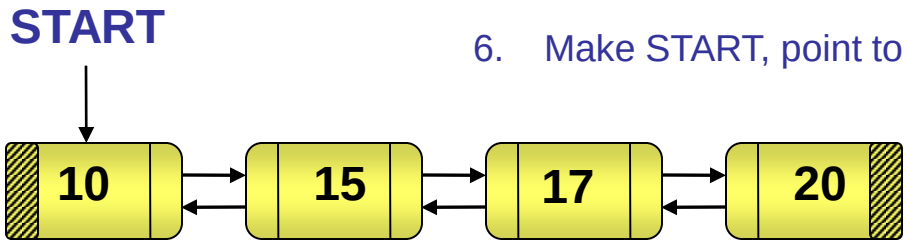
Inserting Nodes in a Doubly-Linked List

- A node can be inserted at any of the following positions in a doubly-linked list:
  - Beginning of the list
  - Between two nodes in the list
  - End of the list

Inserting a Node at the Beginning of the List (Contd.)

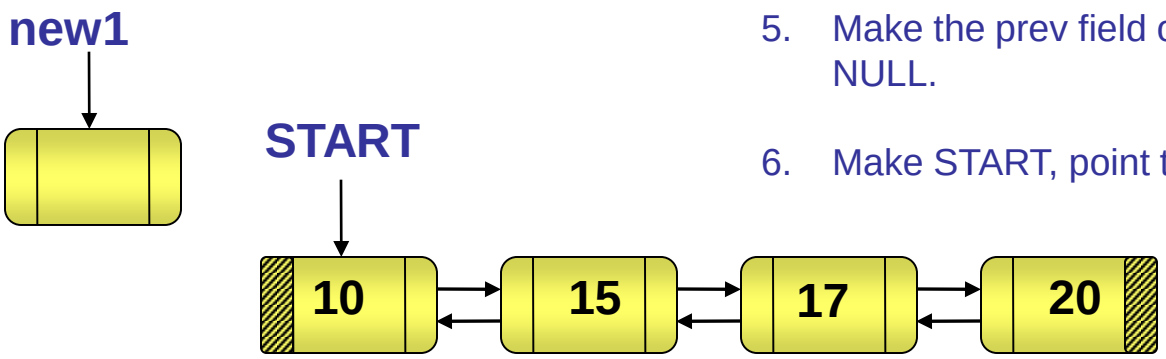
- Algorithm to insert a node in the beginning of a doubly-linked list.

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.



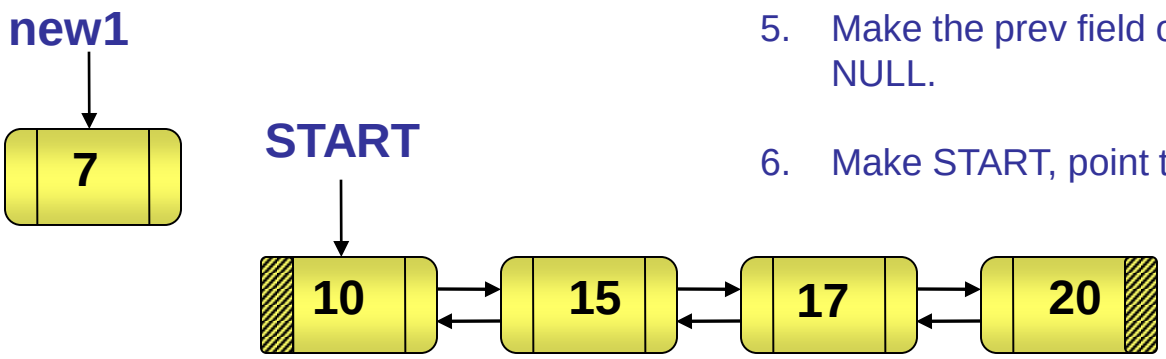
Inserting a Node at the Beginning of the List (Contd.)

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.



Inserting a Node at the Beginning of the List (Contd.)

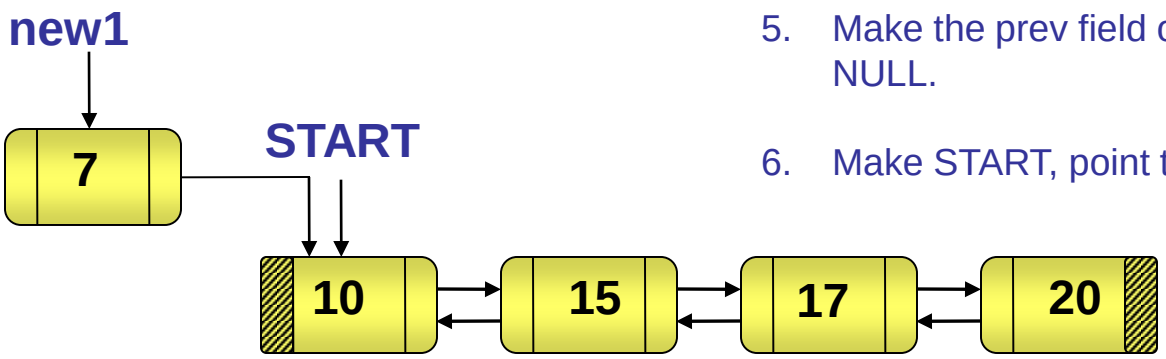
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.



Inserting a Node at the Beginning of the List (Contd.)

new1 -> next = START

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.

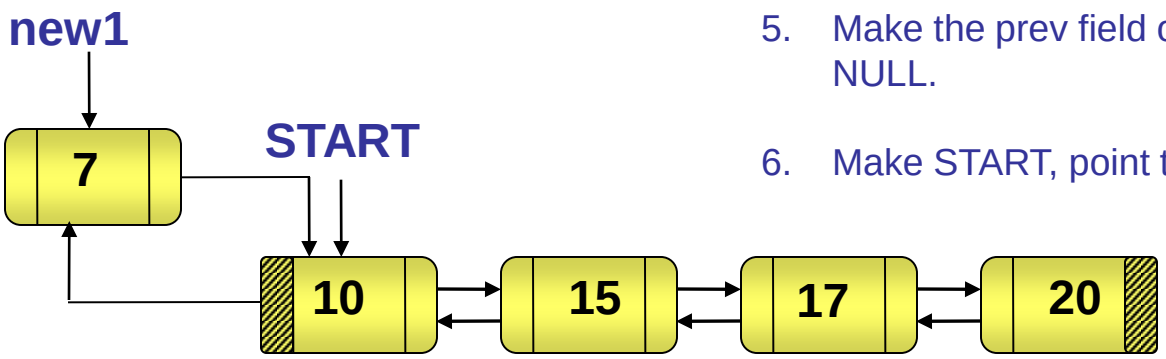


Inserting a Node at the Beginning of the List (Contd.)

new1 -> next = START

START -> prev = new1

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.



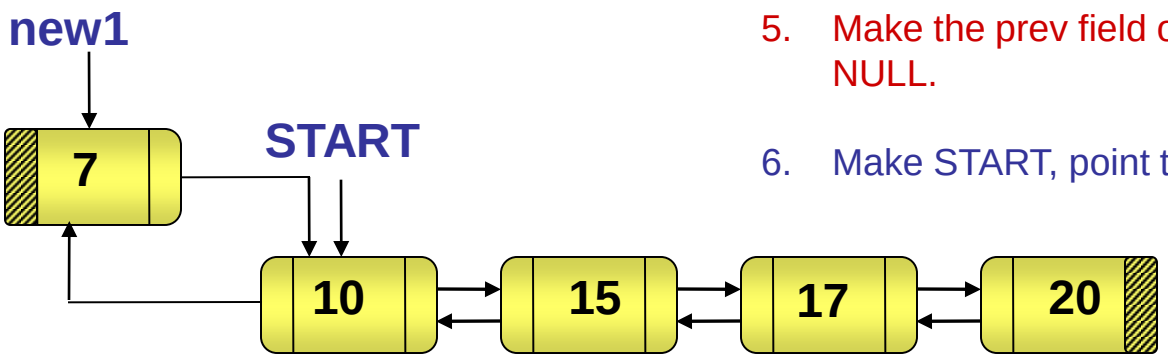
Inserting a Node at the Beginning of the List (Contd.)

new1 -> next = START

START -> prev = new1

new1 -> prev = NULL

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.



Inserting a Node at the Beginning of the List (Contd.)

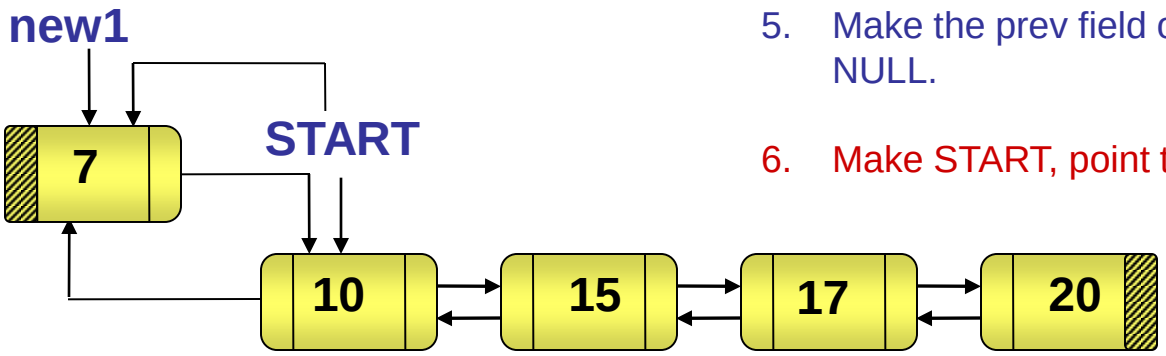
new1 -> next = START

START -> prev = new1

new1 -> prev = NULL

START = new1

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.



Insertion complete

Inserting a Node at the Beginning of the List (Contd.)

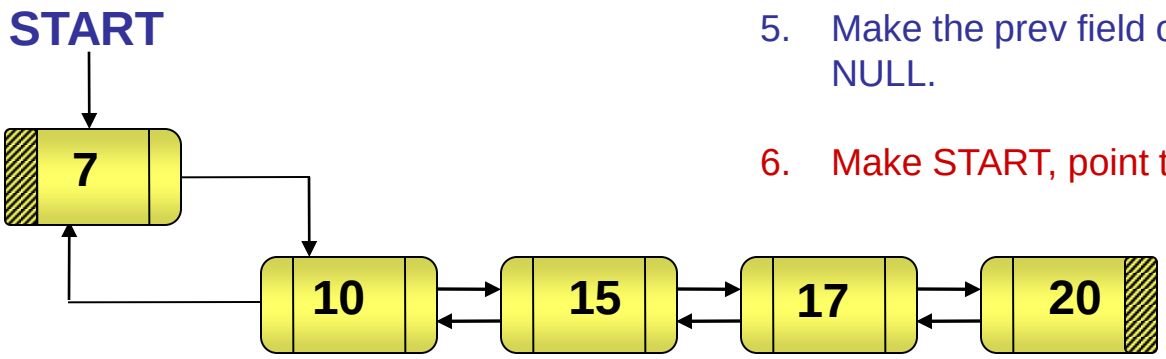
new1 -> next = START

START -> prev = new1

new1 -> prev = NULL

START = new1

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the first node in the list.
4. Make the prev field of START point to the new node.
5. Make the prev field of the new node point to NULL.
6. Make START, point to the new node.



Insertion complete

ALGORITHM TO INSERT NODE AT BEGINING

Start=NULL

Algorithm InsertAtBEG()

{

1. Create node (new1=new node)
2. Enter data [new1 -> info = data]
3. If (Start == NULL)

    3.1 new1 -> next = NULL

    3.2 new1 -> prev = NULL

    3.3 Last=new1

    3.4 Start=new1

else

    3.1 new1 -> next = Start

    3.2 Start -> prev = new1

    3.3 new1 -> Prev = NULL

    3.4 Start = new1

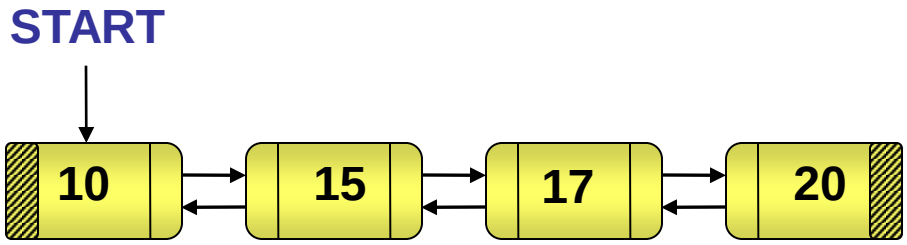
}

Inserting a Node at the end of the List

- Algorithm to insert a node at the end of a doubly-linked list

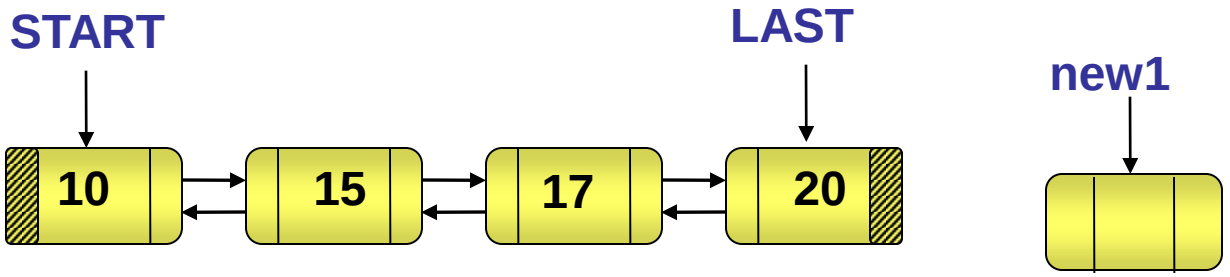
- Write an algorithm to insert a node at the end of a doubly-linked list that contains a variable, LAST, to keep track of the last node of the list.

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the node marked as LAST point to the new node.
4. Make the prev field of new node point to node marked LAST.
5. Make the next field of the new node point to NULL.
6. Mark the new node as LAST.



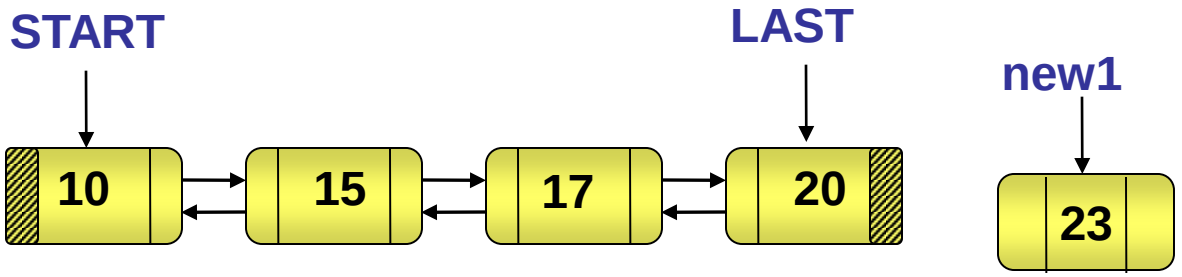
Inserting a Node at the end of the List

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the node marked as LAST point to the new node.
4. Make the prev field of new node point to node marked LAST.
5. Make the next field of the new node point to NULL.
6. Mark the new node as LAST.



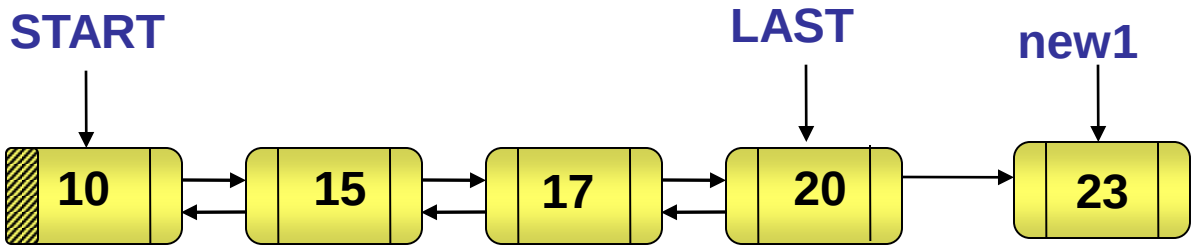
Inserting a Node at the end of the List

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the node marked as LAST point to the new node.
4. Make the prev field of new node point to node marked LAST.
5. Make the next field of the new node point to NULL.
6. Mark the new node as LAST.



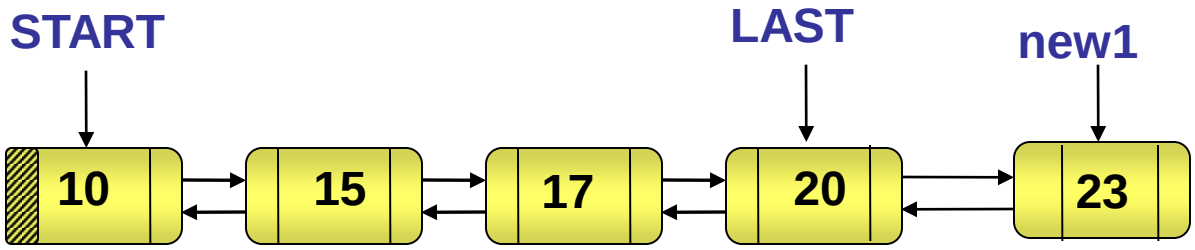
Inserting a Node at the end of the List

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the node marked as LAST point to the new node.
4. Make the prev field of new node point to node marked LAST.
5. Make the next field of the new node point to NULL.
6. Mark the new node as LAST.



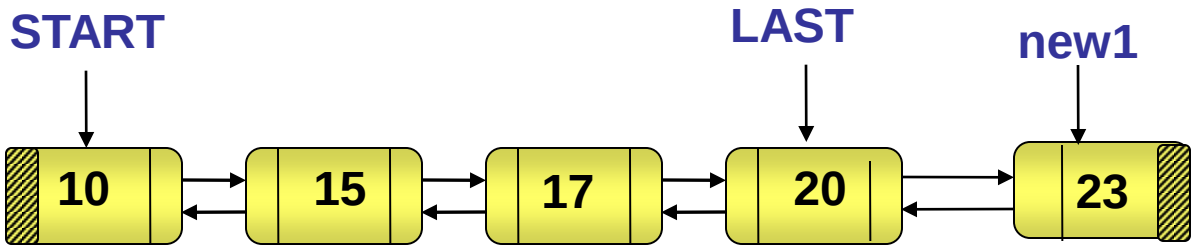
Inserting a Node at the end of the List

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the node marked as LAST point to the new node.
4. Make the prev field of new node point to node marked LAST.
5. Make the next field of the new node point to NULL.
6. Mark the new node as LAST.



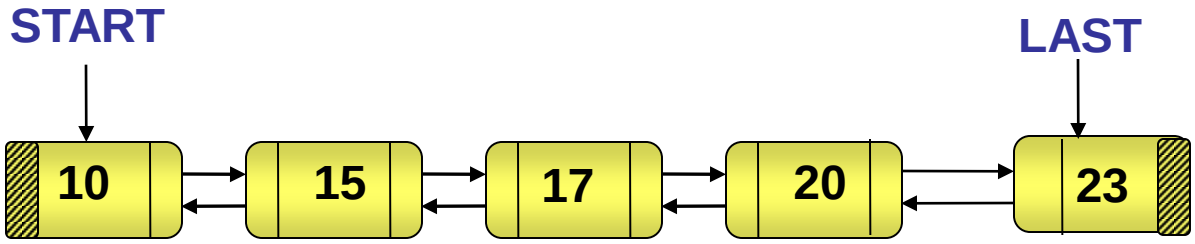
Inserting a Node at the end of the List

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the node marked as LAST point to the new node.
4. Make the prev field of new node point to node marked LAST.
5. Make the next field of the new node point to NULL.
6. Mark the new node as LAST.



Inserting a Node at the end of the List

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the node marked as LAST point to the new node.
4. Make the prev field of new node point to node marked LAST.
5. Make the next field of the new node point to NULL.
6. Mark the new node as LAST.



ALGORITHM TO INSERT NODE AT END

Start=NULL

Algorithm InsertAtEnd()

{

1. Create node [new1 = new node]

2. Enter data [new1 -> info =data]

3.if(Start == NULL)

    3.1 new1 -> next = NULL

    3.2 new1 -> prev = NULL

    3.3 Last=new1

    3.4 Start=new1

  else

    3.1 Last -> next = new1

    3.2 new1 -> prev = Last

    3.3 new1 -> next = NULL

    3.4 Last = new1

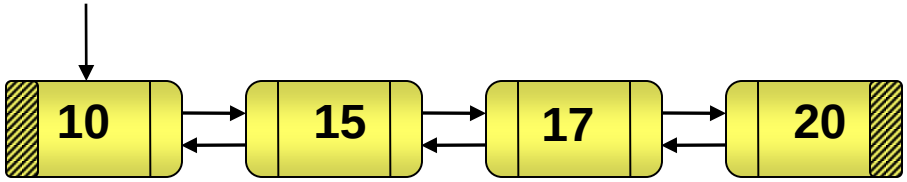
}

Inserting a Node Between Two Nodes in the List

- **Write an algorithm to insert a node at the particular position in a doubly-linked list.**

Insert 16

START



1. Allocate memory for the new node.

2. Assign value to the data field of the new node.

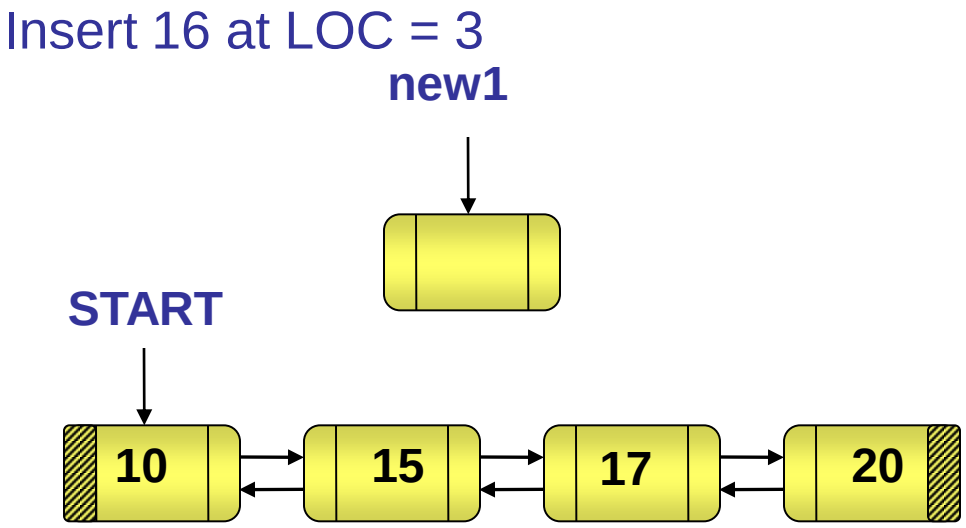
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5. Make the prev field of newnode point to the previous node.

6. Make the prev field of the successor node of current point to the new node. Make the next field of previous point to the new node.

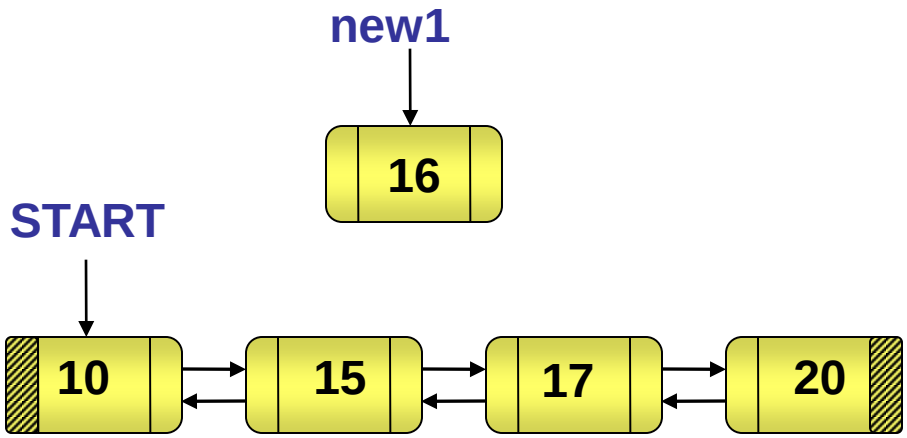
Inserting a Node Between Two Nodes in the List (Contd.)



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

Inserting a Node Between Two Nodes in the List (Contd.)

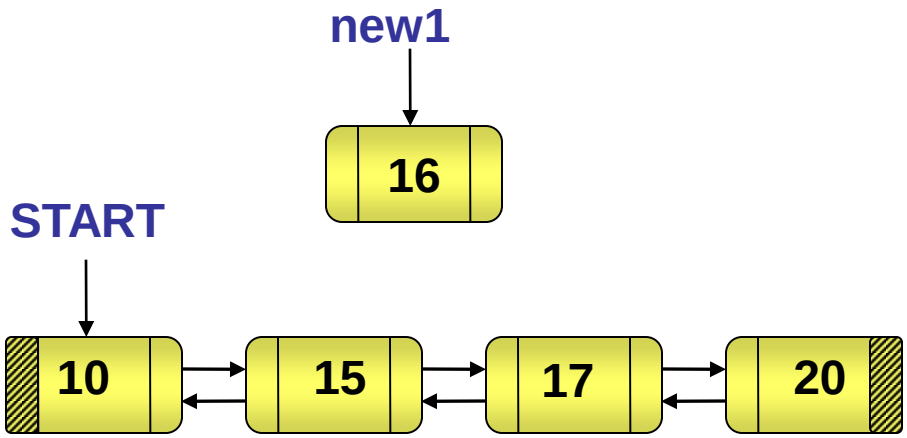
Insert 16 at LOC = 3



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

Inserting a Node Between Two Nodes in the List (Contd.)

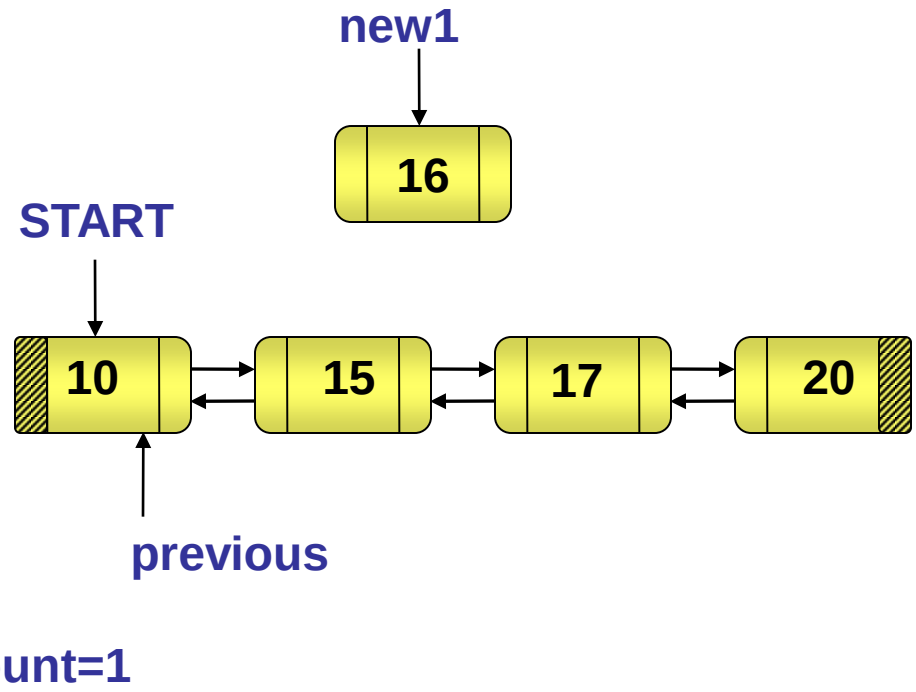
Insert 16 at LOC = 3



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
    1. Make previous node point to the first node and set count=1
    2. Repeat step c and step d until count becomes equal to location-1
    3. Count=count+1.
    4. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

Inserting a Node Between Two Nodes in the List (Contd.)

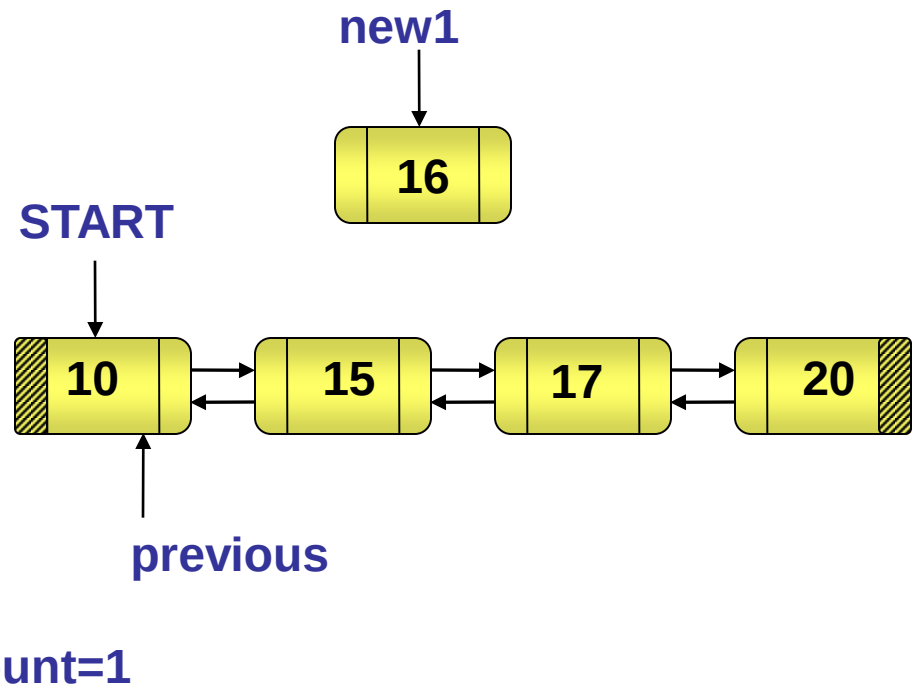
Insert 16



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

Inserting a Node Between Two Nodes in the List (Contd.)

Insert 16

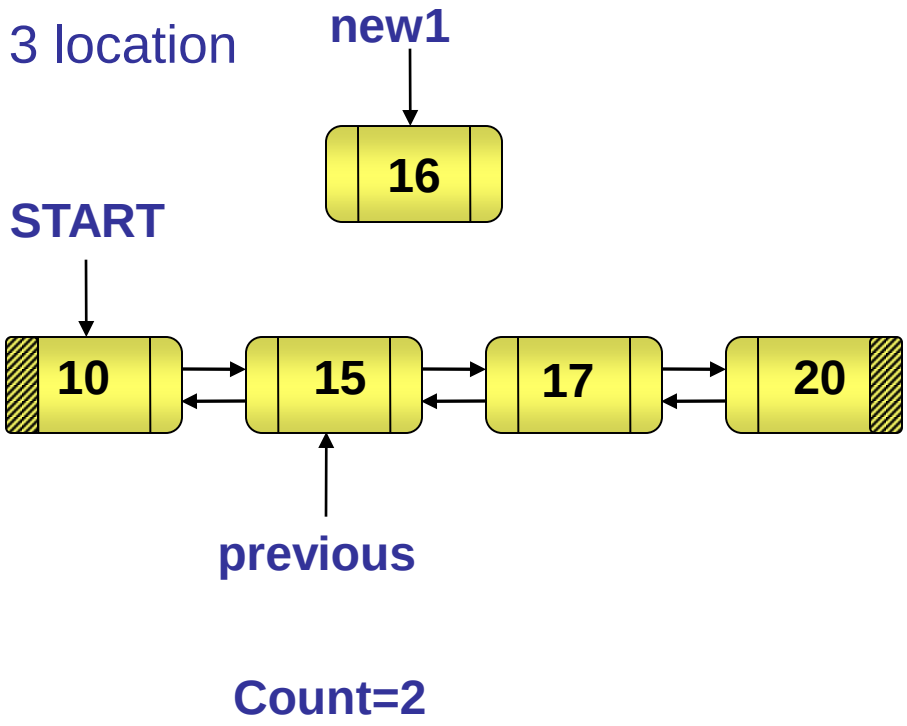


Count=1

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

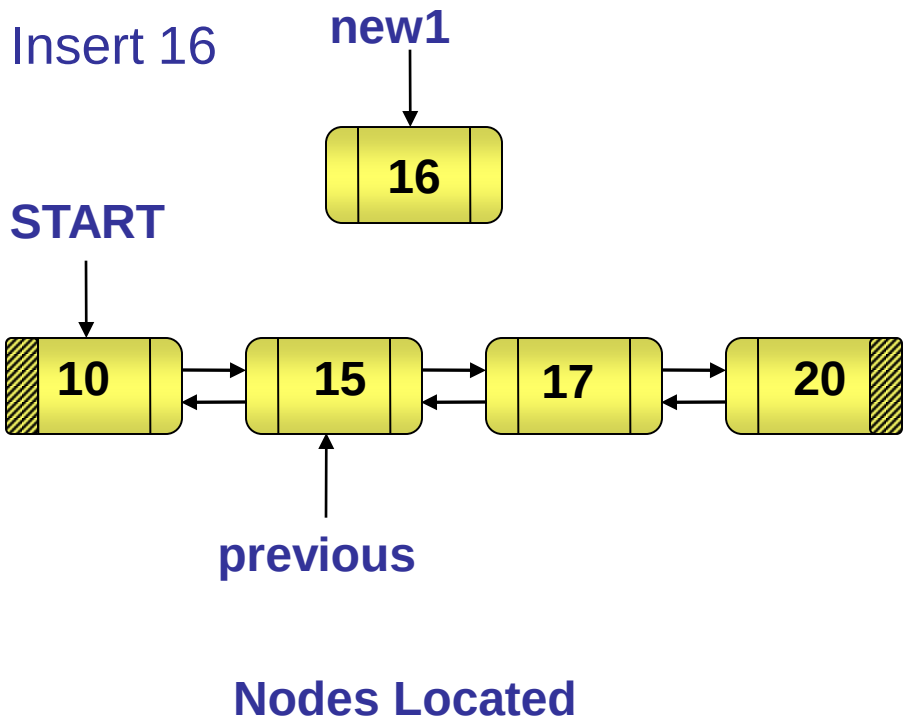
Inserting a Node Between Two Nodes in the List (Contd.)

Insert 16 at 3 location



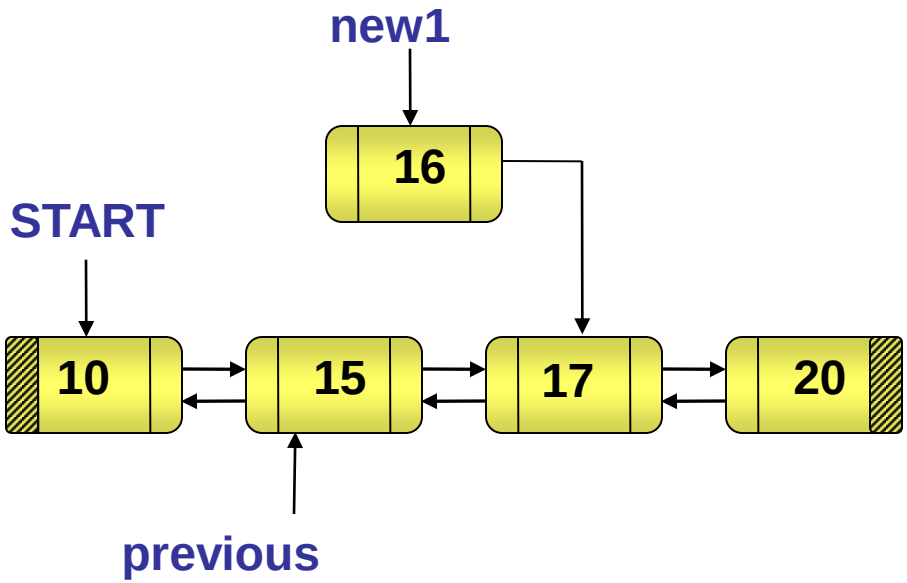
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

Inserting a Node Between Two Nodes in the List (Contd.)



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

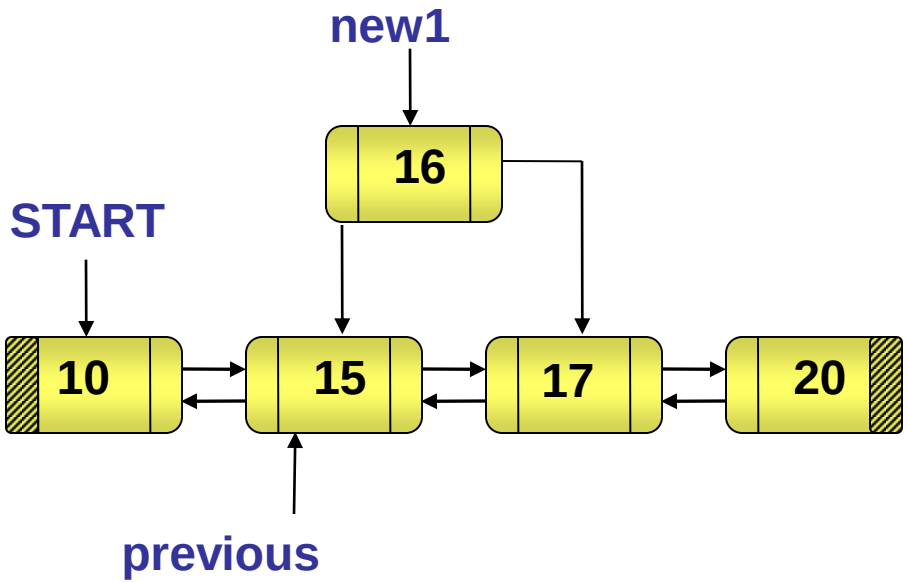
Inserting a Node Between Two Nodes in the List (Contd.)



new1 -> next = previous ->next

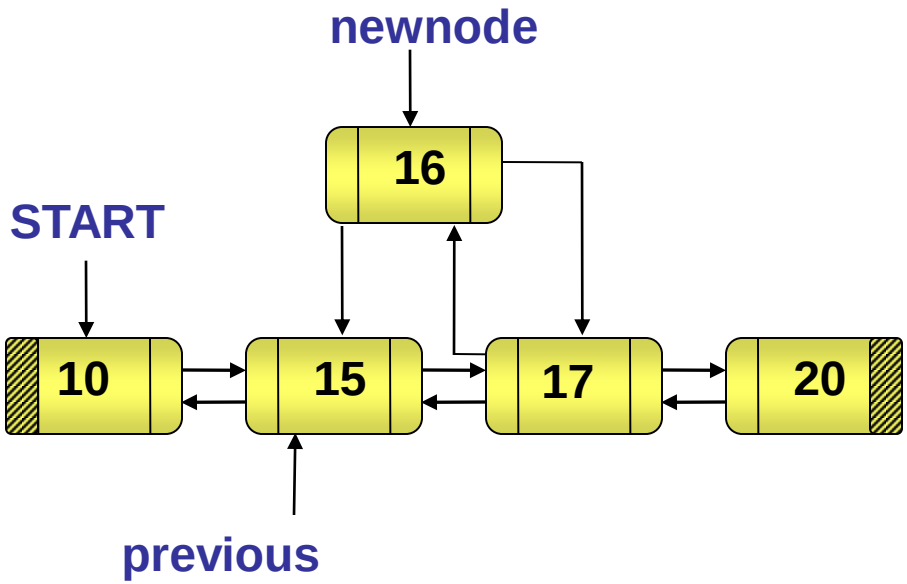
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

Inserting a Node Between Two Nodes in the List (Contd.)



new1 -> prev = previous

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

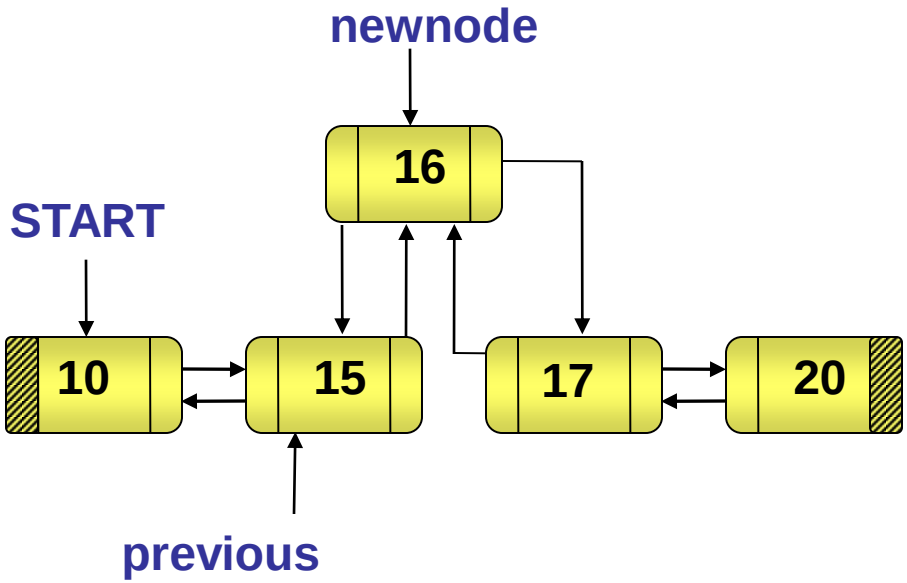


previous -> next -> prev = new1

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

Inserting a Node Between Two Nodes in the List (Contd.)

Insertion complete



previous -> next = new1

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence
4. Make the next field of the new node point to the next of previous node
5. Make the prev field of newnode point to the previous node.
6. Make the prev field of the successor node of current point to the new node.
7. Make the next field of previous point to the new node.

ALGORITHM TO INSERT NODE At PARTICULAR POSITION

Algorithm InsertAtSpec()

{

1. Create node [new1=new node]
2. Enter Data and Location
3. new1->info=Data
4. If (Location == 1)

4.1 new1 -> next = Start

4.2 Start -> prev = new1

4.3 new1 -> Prev = NULL

4.4 Start = new1

Else

4.1 Previous = Start

4.2 Count = 1

4.3 While(Count < Location - 1 && Previous !=NULL)

4.3.1 Previous = Previous -> next

4.3.2 Count++

4.4 new1 -> prev = Previous

4.5 If (Previous -> next == NULL)

4.5.1 new1 -> next = NULL; 4.5.2 Previous -> next = new1

4.5.3 Last = new1

Else

4.5.1 new1 -> next = Previous -> next 4.5.2 Previous -> next -> prev = new1

4.5.3 Previous -> next = new1

}

Deleting Nodes from a Doubly-Linked List

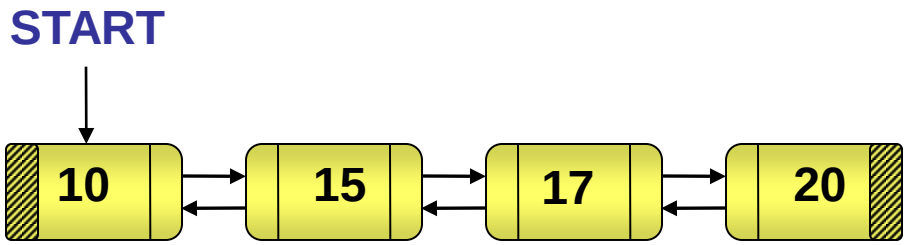
- You can delete a node from one of the following places in a doubly-linked list:
  - Beginning of the list
  - Between two nodes in the list
  - End of the list

Deleting a Node From the Beginning of the List

- Write an algorithm to delete a node from the beginning of a doubly-linked list.

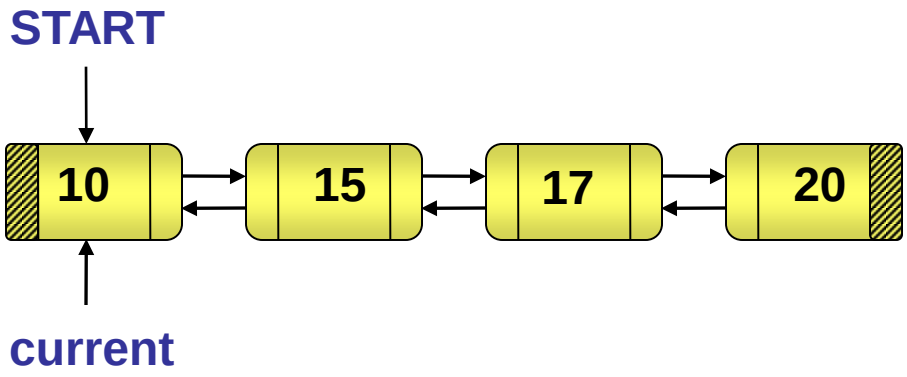
Deleting a Node From the Beginning of the List (Contd.)

- Algorithm to delete a node from the beginning of a doubly-linked list.



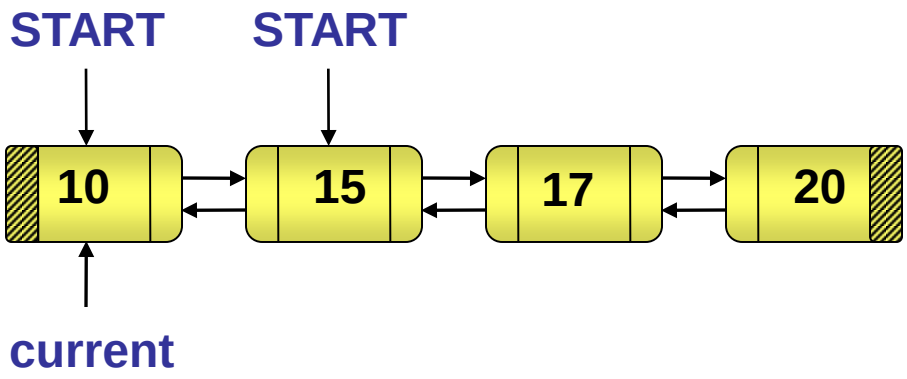
1. Mark the first node in the list as current.
2. Make START point to the next node in sequence.
3. If START is not NULL: **/* If the node to be deleted is not the only node in the list */**
    1. Assign NULL to the prev field of the node marked as START.
4. Release the memory of the node marked as current.

Deleting a Node From the Beginning of the List (Contd.)



1. Mark the first node in the list as current.
2. Make START point to the next node in sequence.
3. If START is not NULL: **/* If the node to be deleted is not the only node in the list */**
    1. Assign NULL to the prev field of the node marked as START.
4. Release the memory of the node marked as current.

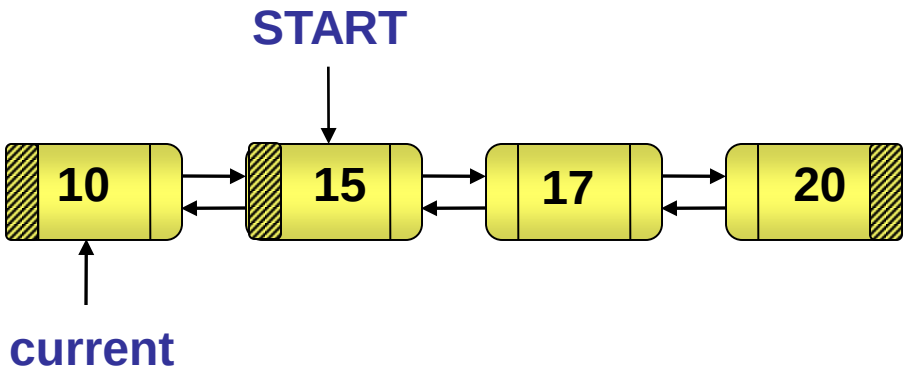
Deleting a Node From the Beginning of the List (Contd.)



START = START -> next

1. Mark the first node in the list as current.
2. Make START point to the next node in sequence.
3. If START is not NULL: **/* If the node to be deleted is not the only node in the list */**
    1. Assign NULL to the prev field of the node marked as START.
4. Release the memory of the node marked as current.

Deleting a Node From the Beginning of the List (Contd.)

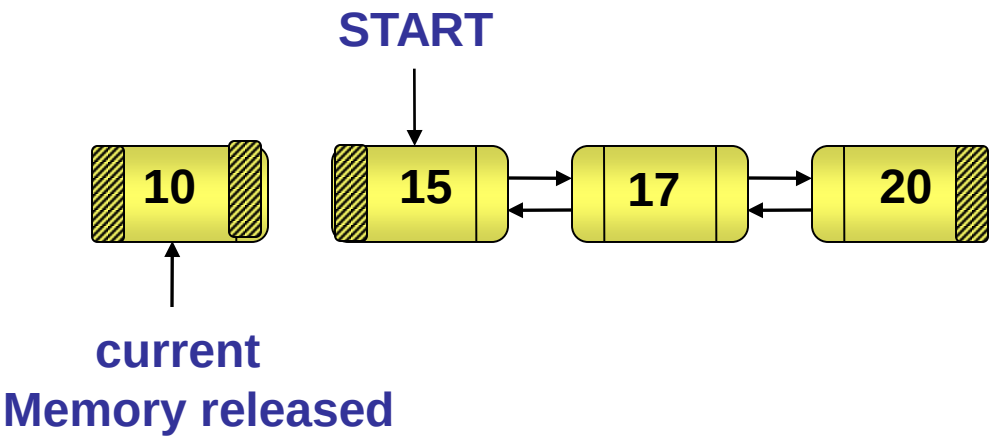


1. Mark the first node in the list as current.
2. Make START point to the next node in sequence.
3. If START is not NULL: **/* If the node to be deleted is not the only node in the list */**
    1. Assign NULL to the prev field of the node marked as START.
4. Release the memory of the node marked as current.

START -> prev = NULL

Deleting a Node From the Beginning of the List (Contd.)

Delete operation complete



1. Mark the first node in the list as current.
2. Make START point to the next node in sequence.
3. If START is not NULL: **/* If the node to be deleted is not the only node in the list */**
    1. Assign NULL to the prev field of the node marked as START.
4. Release the memory of the node marked as current.

ALGORITHM TO DELETE A NODE FROM THE BEGINING

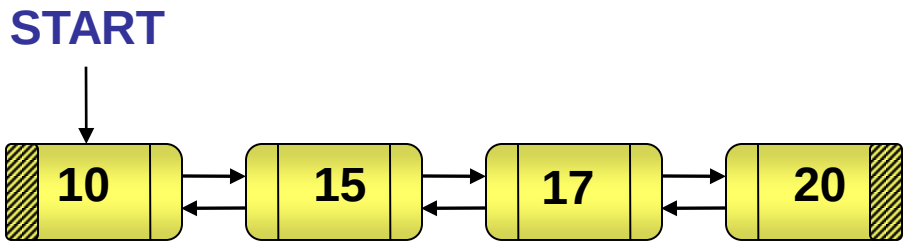
```
Algorithm DeleteAtBeg()
{
1. If (Start == NULL)
            1.1 Print "underflow“
   else
            1.1 Current = Start
            1.2 Start = Start -> next
            1.3 Start -> prev = NULL
            1.3 Current -> next = NULL
            1.4 Current -> prev = NULL
            1.4 Release the memory [delete(Current) ]
}
```

Deleting a Node Between Two Nodes in the List

- Write an algorithm to delete a Particular node in a doubly-linked list.

Deleting a Node Between Two Nodes in the List (Contd.)

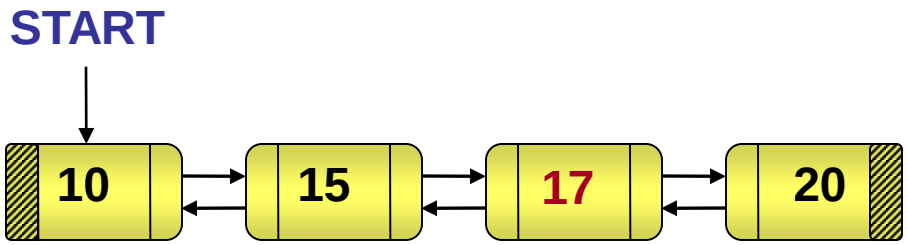
- Algorithm to delete a node between two nodes in a doubly-linked list.
  - Delete 17



1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
   1. Make previous point to NULL. **// Set previous = NULL**
   2. Make current point to the first node in the linked list. **// Set current = START**
   3. Repeat steps d and e until either the node is found or current becomes NULL.
   4. Make previous point to current.
   5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

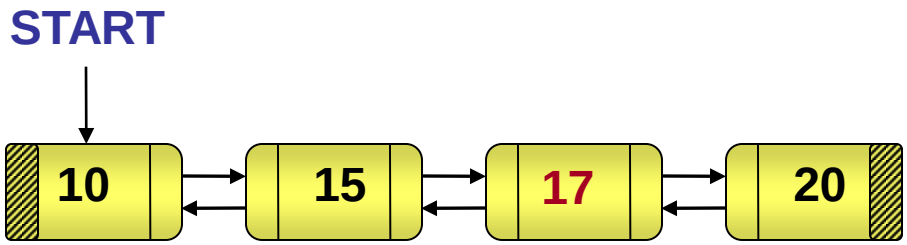
Deleting a Node Between Two Nodes in the List (Contd.)

- Delete 17



1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

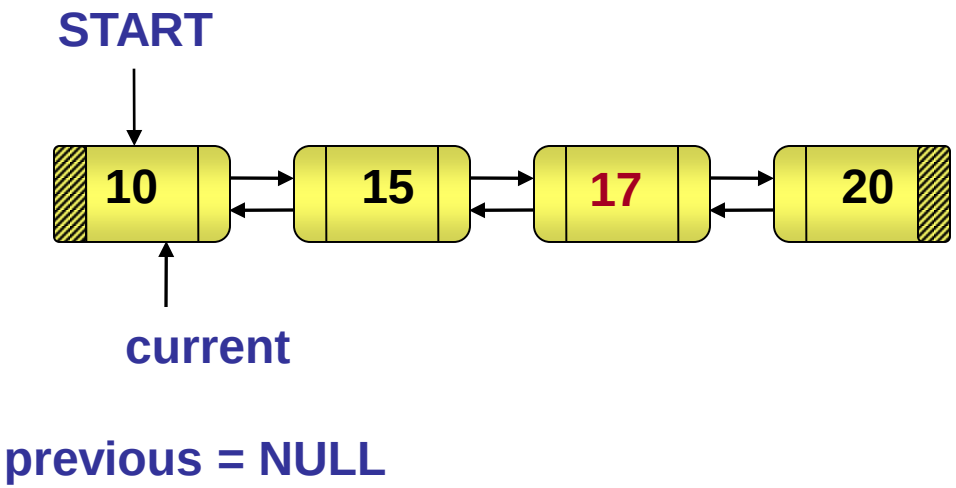
Deleting a Node Between Two Nodes in the List (Contd.)



previous = NULL

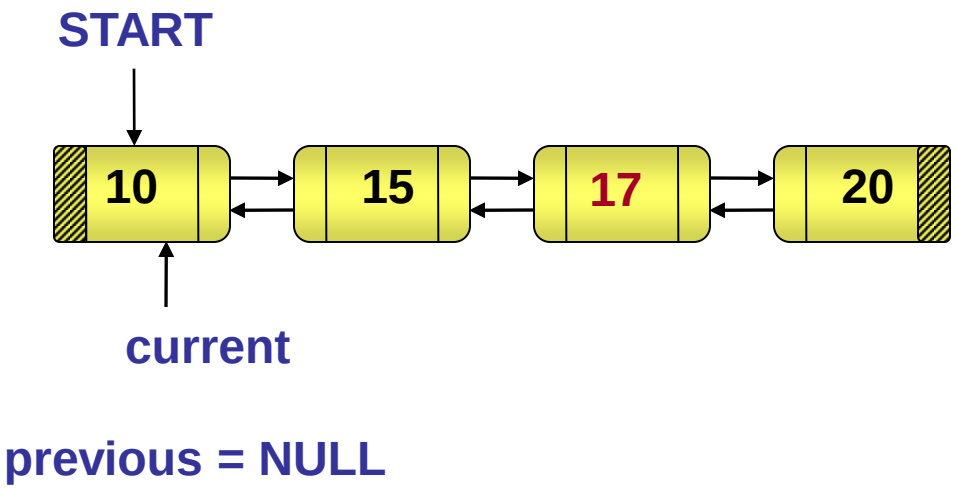
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. ***// Set previous = NULL***
    2. Make current point to the first node in the linked list. ***// Set current = START***
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



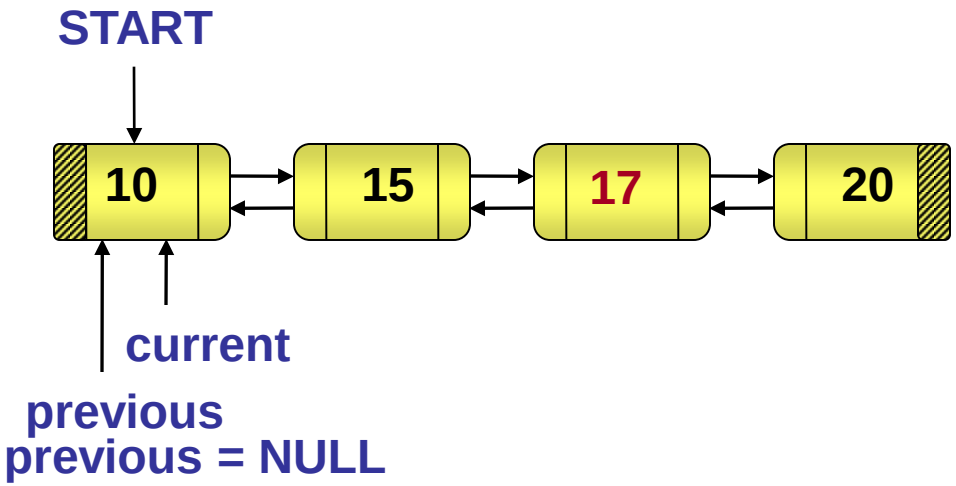
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



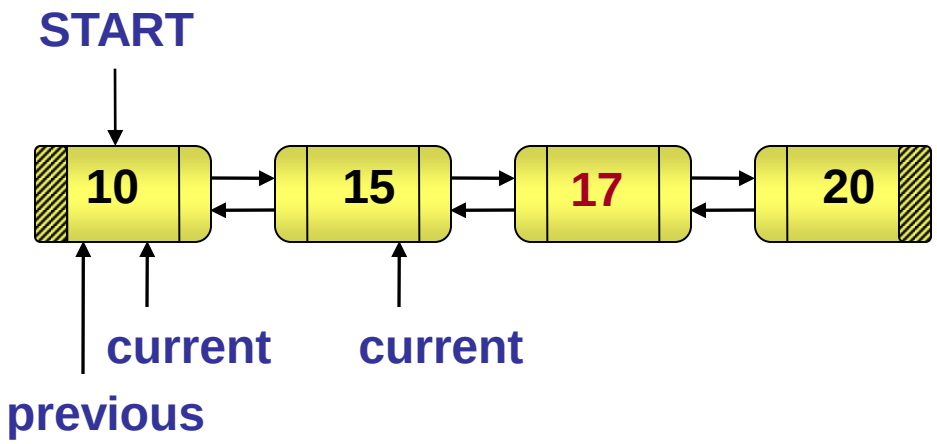
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



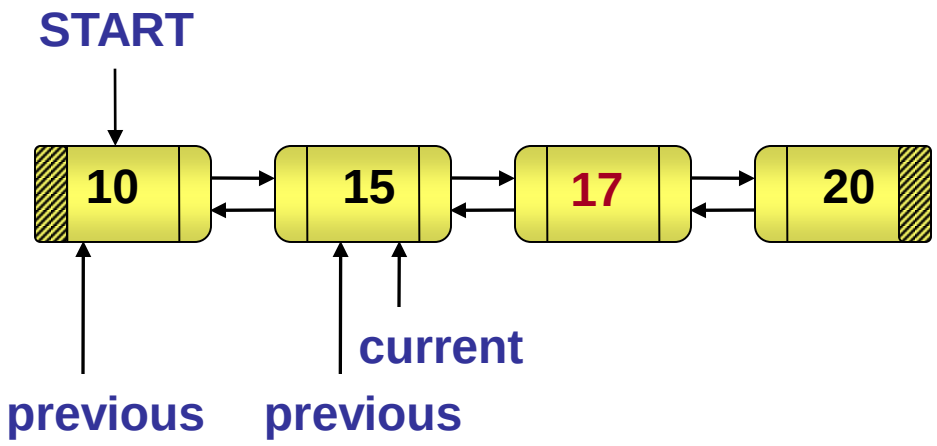
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. ***// Set previous = NULL***
    2. Make current point to the first node in the linked list. ***// Set current = START***
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



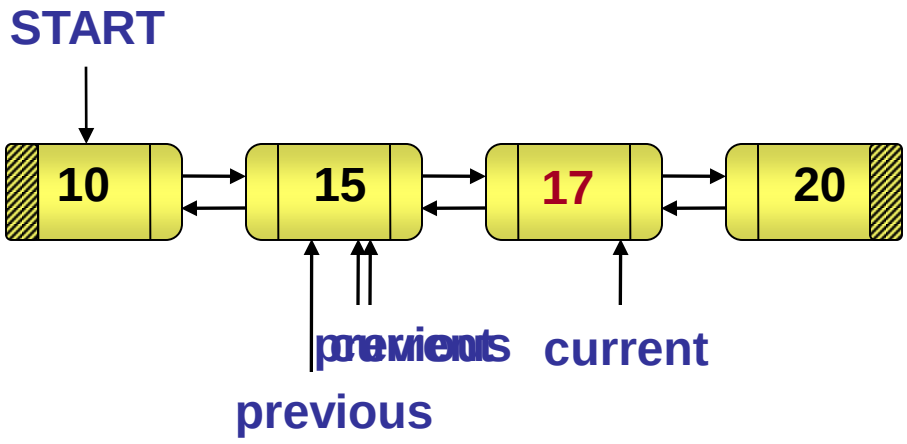
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



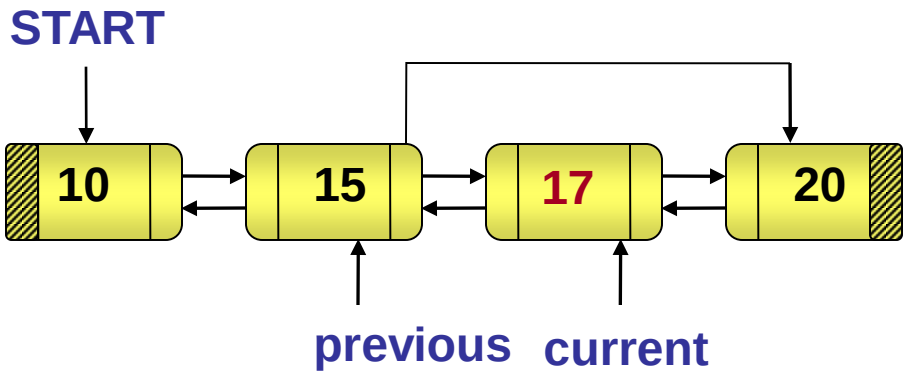
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



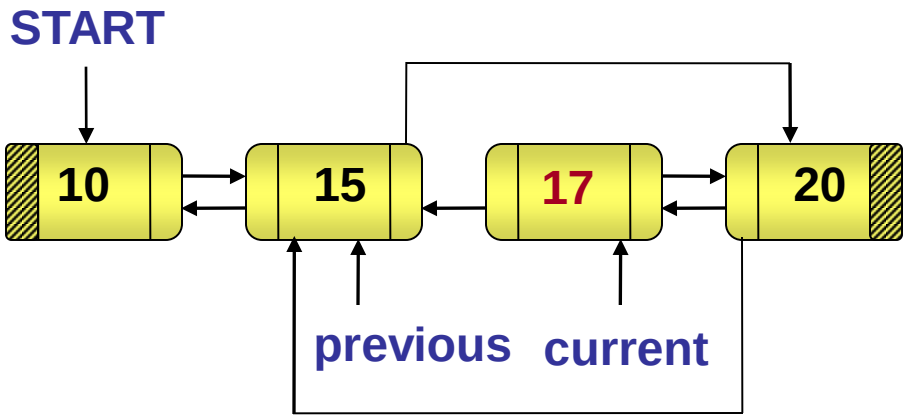
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



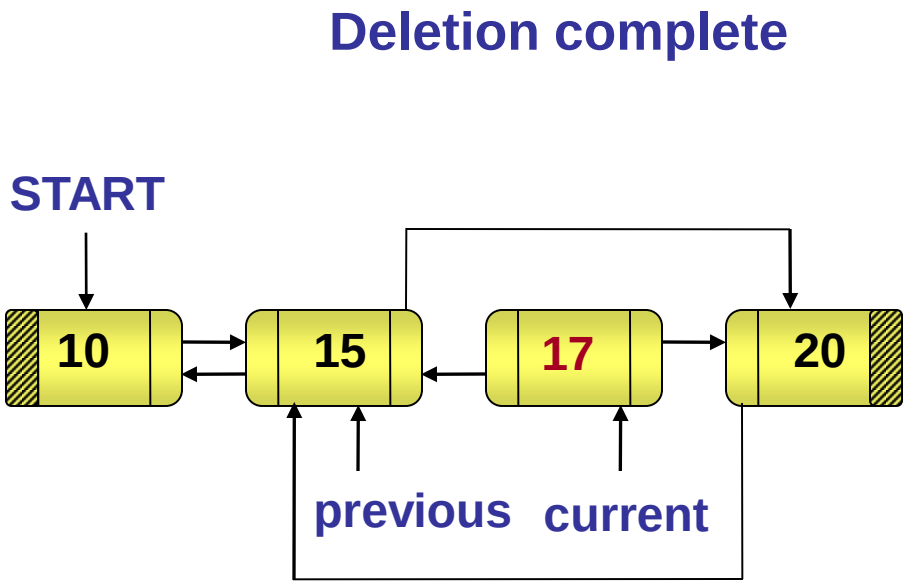
1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

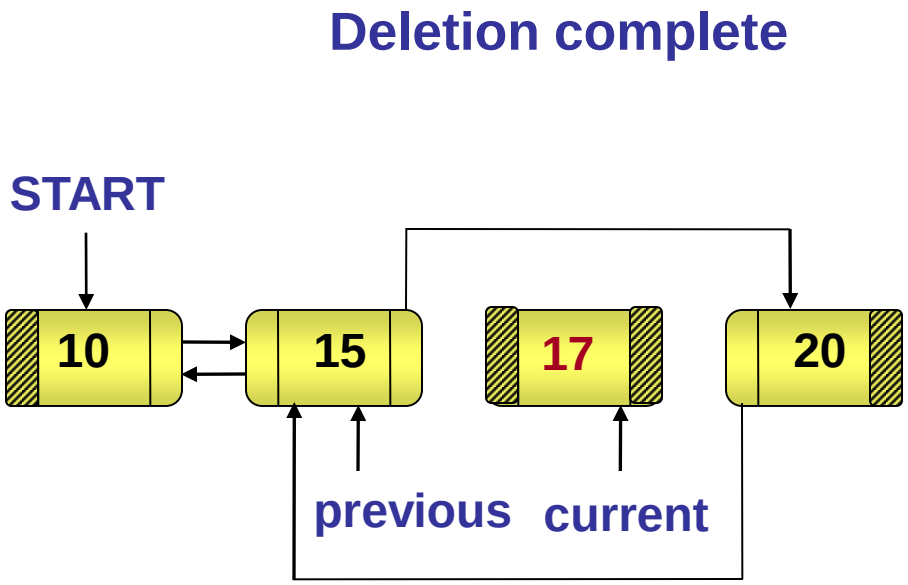
Deleting a Node Between Two Nodes in the List (Contd.)



current -> next -> prev = previous

1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. **// Set previous = NULL**
    2. Make current point to the first node in the linked list. **// Set current = START**
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

Deleting a Node Between Two Nodes in the List (Contd.)



current -> next -> prev = previous

Deletion Completed

1. Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:
    1. Make previous point to NULL. ***// Set previous = NULL***
    2. Make current point to the first node in the linked list. ***// Set current = START***
    3. Repeat steps d and e until either the node is found or current becomes NULL.
    4. Make previous point to current.
    5. Make current point to the next node in sequence.
2. Make the next field of previous point to the successor of current.
3. Make the prev field of the successor of current point to previous.
4. Release the memory of the node marked as current.

ALGORITHM TO DELETE A NODE FROM THE SPECIFIC POSITION

Algorithm DeleteAtSpec()

{

1. Enter the Location
2. Current = Start
3. Previous = NULL
4. If (Start == 0)

        4.1 Print "underflow“

   else If (Location == 1)

        4.1 Start = Start -> next

        4.2 Start -> prev = NULL

        4.3 Current -> next = NULL

        4.4 Release the memory [delete(Current)]

   else

        4.1 for (i=1; i<Location; i++)

            4.1.1 Previous = Current

            4.1.2 Current = Current -> next

        4.2 if (Current -> next == NULL)

              4.2.1 Previous -> next = NULL

              4.2.2 Last = Previous

            else

              4.2.1 Previous -> next = Current -> next

              4.2.2 Current -> next -> prev = Previous

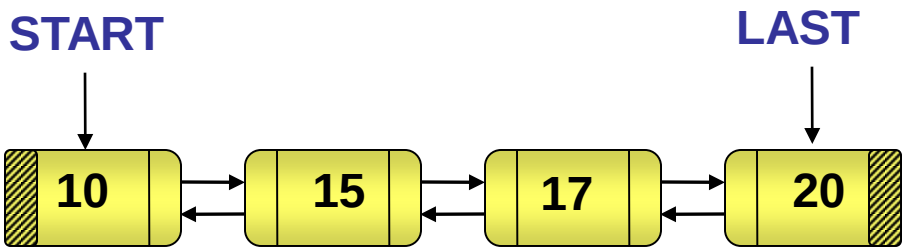
        4.3 Current -> next = NULL

        4.4 Current -> prev = NULL

        4.5 Release the memory [delete(Current)]

Deleting a Node Between Two Nodes in the List (Contd.)

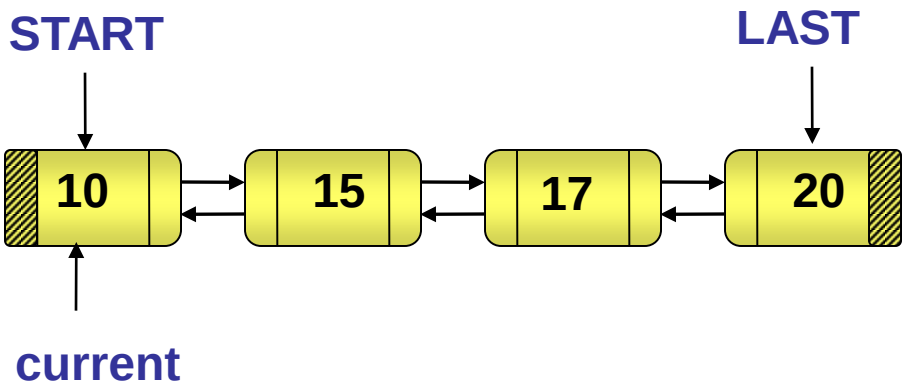
- Refer to the modified algorithm
  - Delete 20



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.
5. Release the memory for the node marked as Last.
6. Make Current point to the last node

Deleting a Node Between Two Nodes in the List (Contd.)

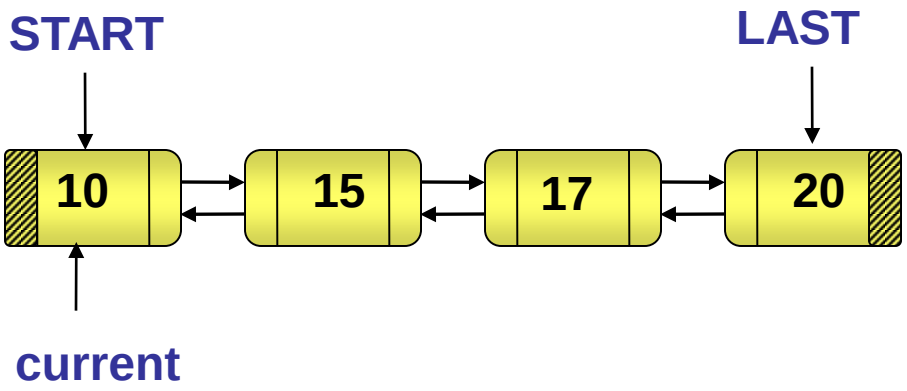
- Delete 20



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.
5. Release the memory for the node marked as Last.
6. Make Current point to the last node

Deleting a Node Between Two Nodes in the List (Contd.)

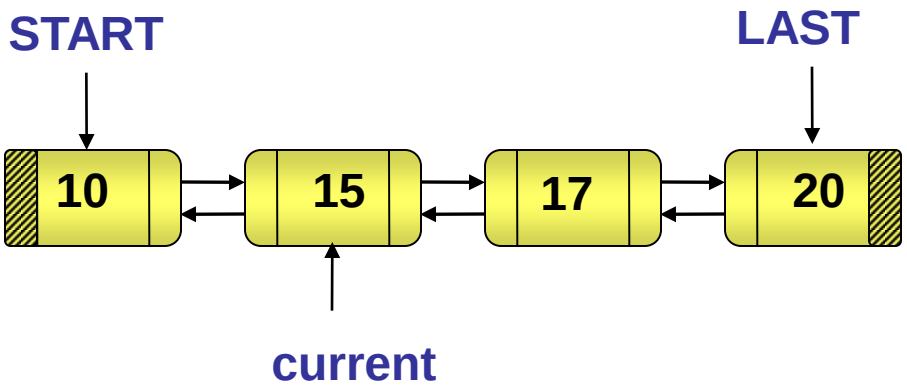
- Delete 20



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.

    1. Release the memory for the node marked as Last.
    2. Make Current point to the last node

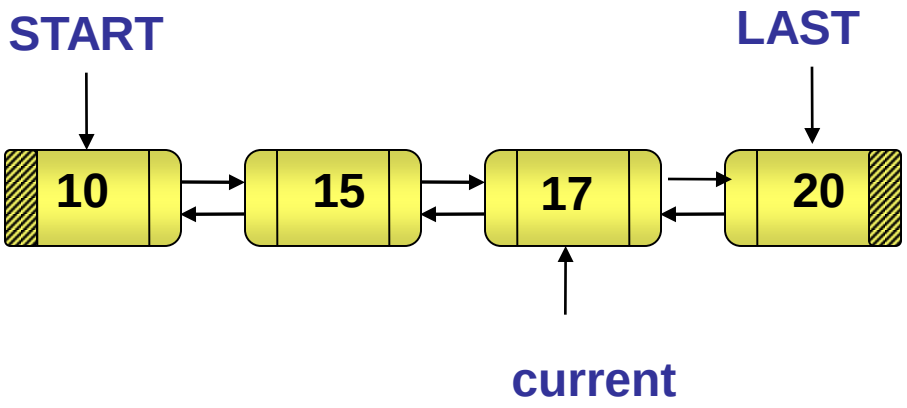
Deleting a Node Between Two Nodes in the List (Contd.)



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.

    1. Release the memory for the node marked as Last.
    2. Make Current point to the last node

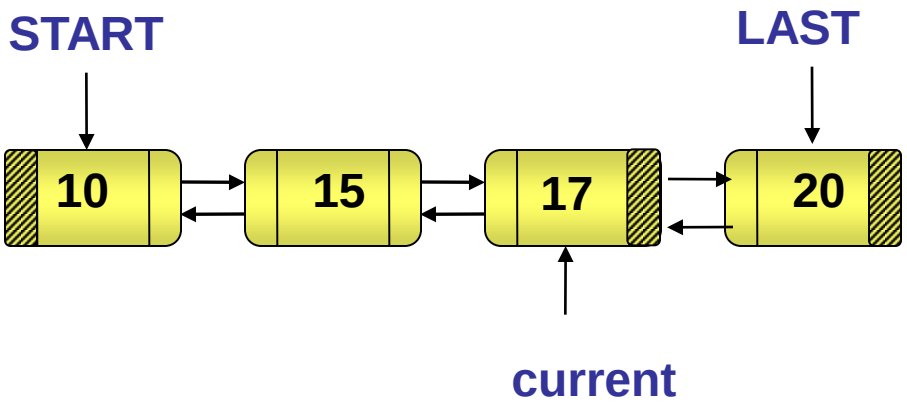
Deleting a Node Between Two Nodes in the List (Contd.)



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.

    1. Release the memory for the node marked as Last.
    2. Make Current point to the last node

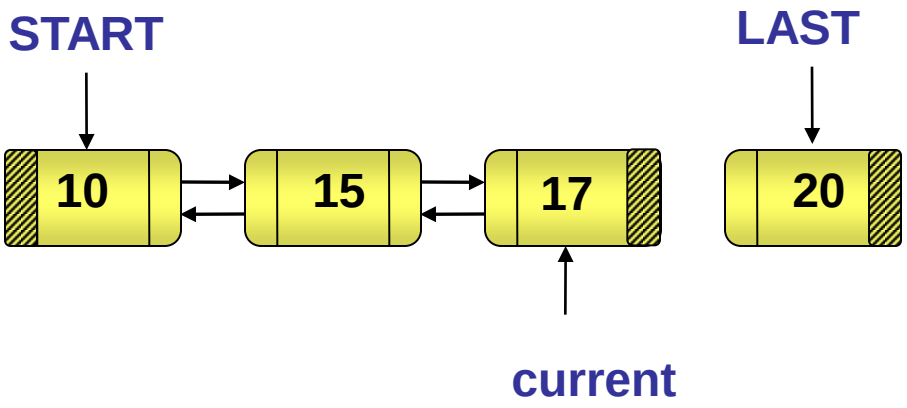
Deleting a Node Between Two Nodes in the List (Contd.)



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.

1. Release the memory for the node marked as Last.
2. Make Current point to the last node

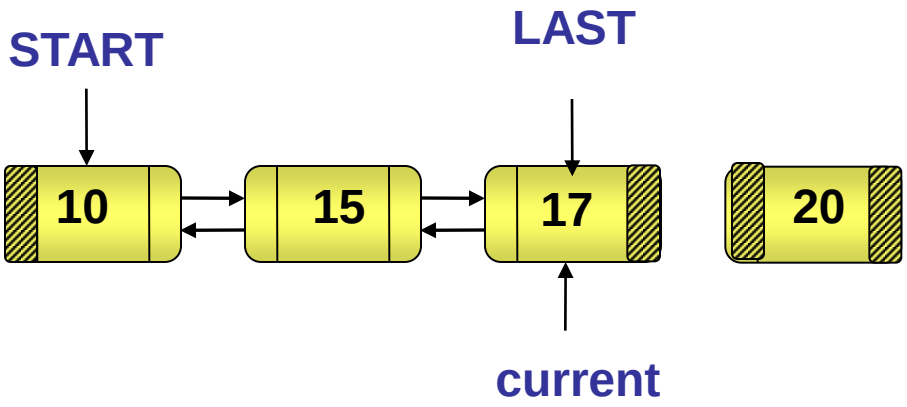
Deleting a Node Between Two Nodes in the List (Contd.)



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.

1. Release the memory for the node marked as Last.
2. Make Current point to the last node

Deleting a Node Between Two Nodes in the List (Contd.)



1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Make the next field of current point to the null.
5. Release the memory for the node marked as Last.
6. Make Current point to the last node

Delete operation complete

ALGORITHM TO DELETE A NODE FROM THE END

```
Algorithm DeleteAtEnd()
{
1. If (Start == NULL)
            1.1 Print "underflow“
   else If (Start -> next == NULL )
            1.1 Release the memory [delete(Start) ]
            1.2 Start == NULL
    else
            1.1 Current = Start
            1.2 while ( Current -> next != Last )
                  1.2.1 Current = Current -> next
            1.3 Current -> next = NULL
            1.4 Last -> prev = NULL
            1.4 Release the memory [ delete (Last) ]
            1.5 Last = Current
}
```

Advantages and disadvantages

Advantages

1) A DLL can be traversed in both forward and backward direction.
2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
3) We can quickly insert a new node before a given node.
4) It is easy to reverse the linked list.
5) If we are at a node, then we can go to any node. But in linear linked list, it is not possible to reach the previous node.
6) Most efficient data structure to implement when traversing in both direction is required.

Disadvantages

1) It uses extra memory for per node for previous pointer when compared to array and singly linked list for

2)All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers

Applications of DLL

1) Doubly linked list can be used in navigation systems where both front and back navigation is required.

2) It is used by browsers to implement backward and forward navigation of visited web pages i.e. back and forward button.

3) It is also used by various application to implement Undo and Redo functionality. (Ex word,photoshop)

4) It can also be used to represent deck of cards in games.

5) It is also used to represent various states of a game.

6) A stack, hash table, and binary tree can be implemented using a doubly linked list.