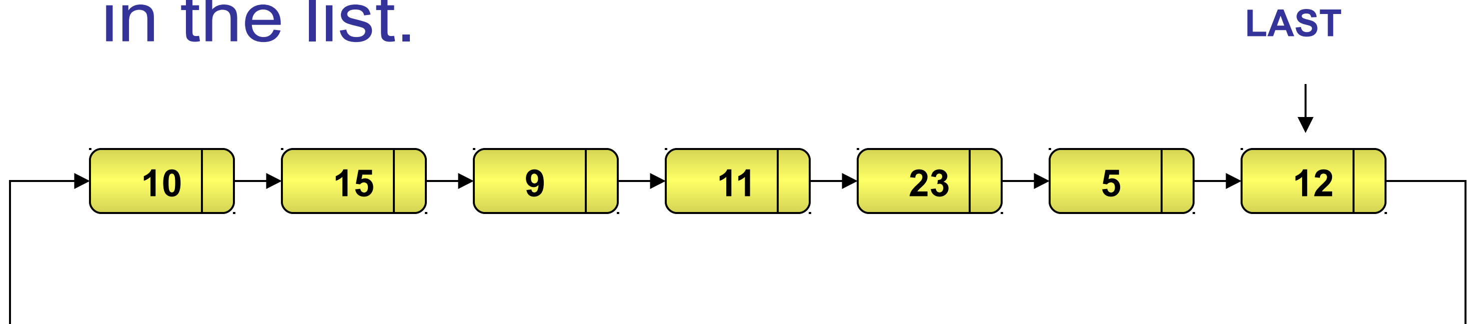
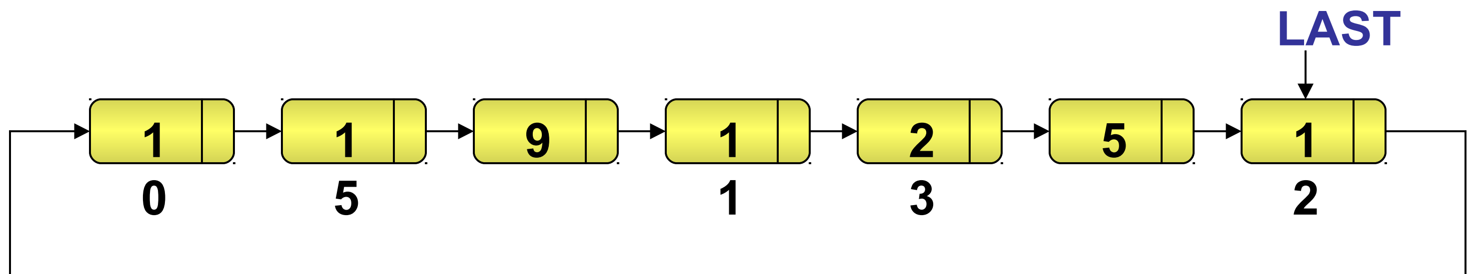
Circular Linked Lists

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

- You can implement a circular linked list by linking the last node back to the first node in the list.



- In a circular linked list, the last node holds the address of the first node.



- In a circular linked list, you need to maintain a variable/pointer, LAST, which always point to the last node.

- Write an algorithm to traverse a circular linked list.
- Algorithm to traverse a circular linked list.

1. Make currentNode point to the successor of the node marked as LAST, such that currentNode points to the first node in the list.

2. Repeat step 3 and 4 until currentNode = LAST.

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

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

5. Display the information contained in the node marked as LAST.

TRAVERSING IN LIST

```
Algorithm traversing()
{
1.new1 = Last -> next
2.while(new1 != Last)
        2.1 Print  new1 -> info
        2.2 new1 = new1 -> next
3. Print Last -> info
}
```

- In a circular linked list, you can insert a node at any of the following positions:
  - Beginning of the list
  - At specific position
  - End of the list

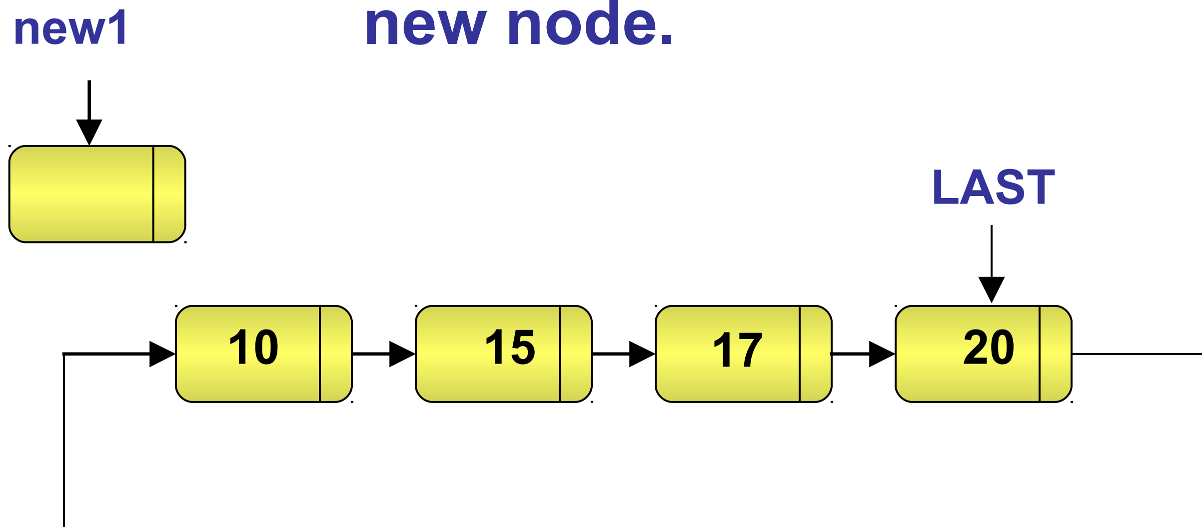
- Write an algorithm to insert a node in the beginning of a circular linked list.

1. Allocate memory for the new node.

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

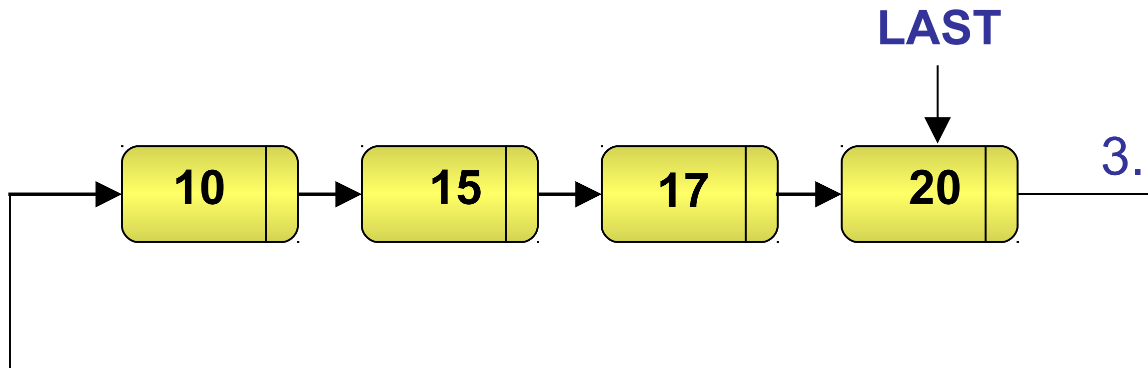
3. Make the next field of the new node point to the successor of LAST.

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



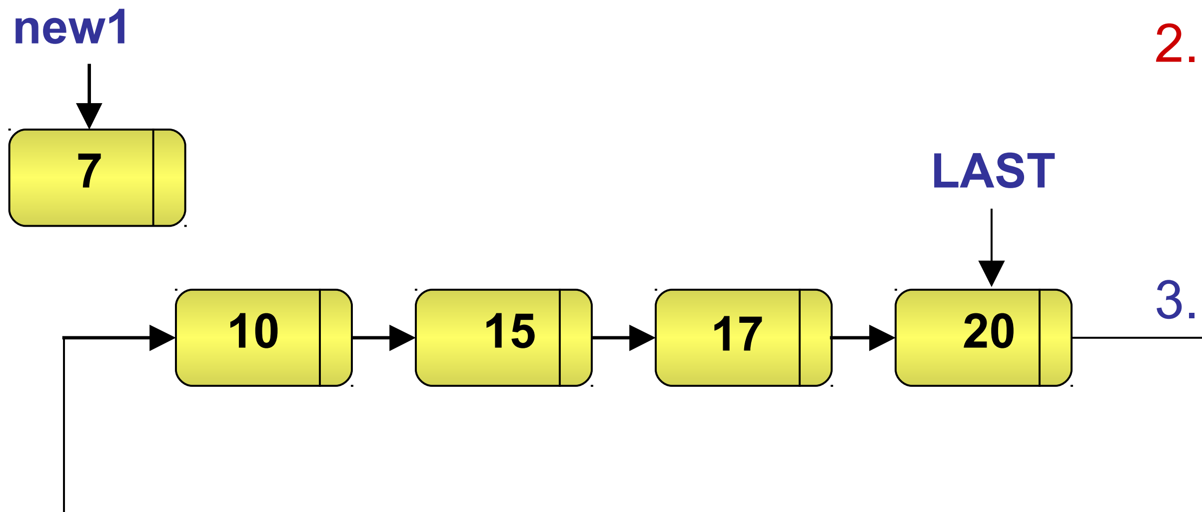
Inserting a Node at the Beginning of the List (Contd.)

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



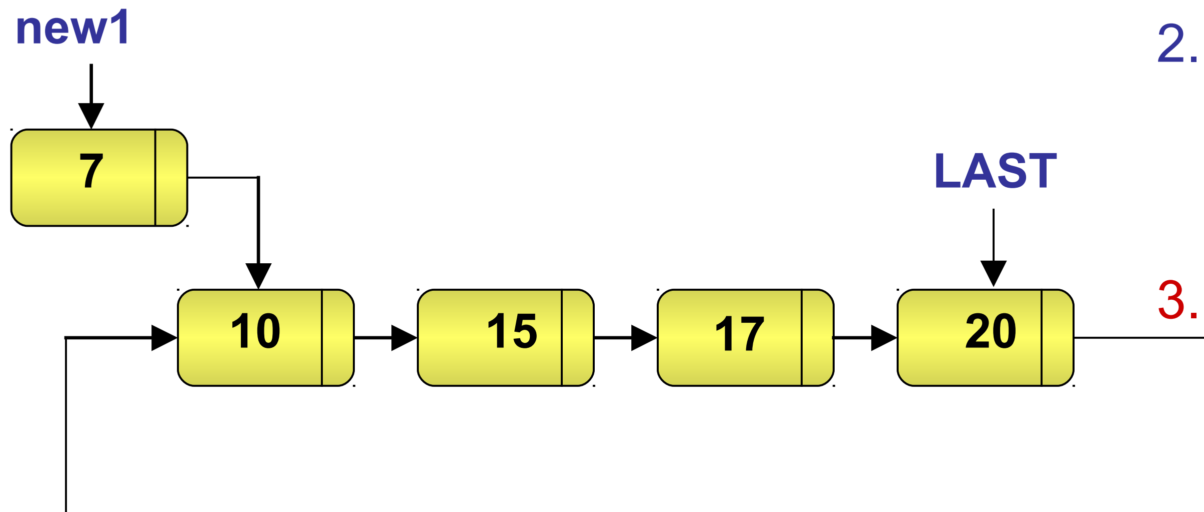
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the successor of LAST.
4. Make the next field of LAST point to the new node.

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 successor of LAST.
4. Make the next field of LAST point to the new node.

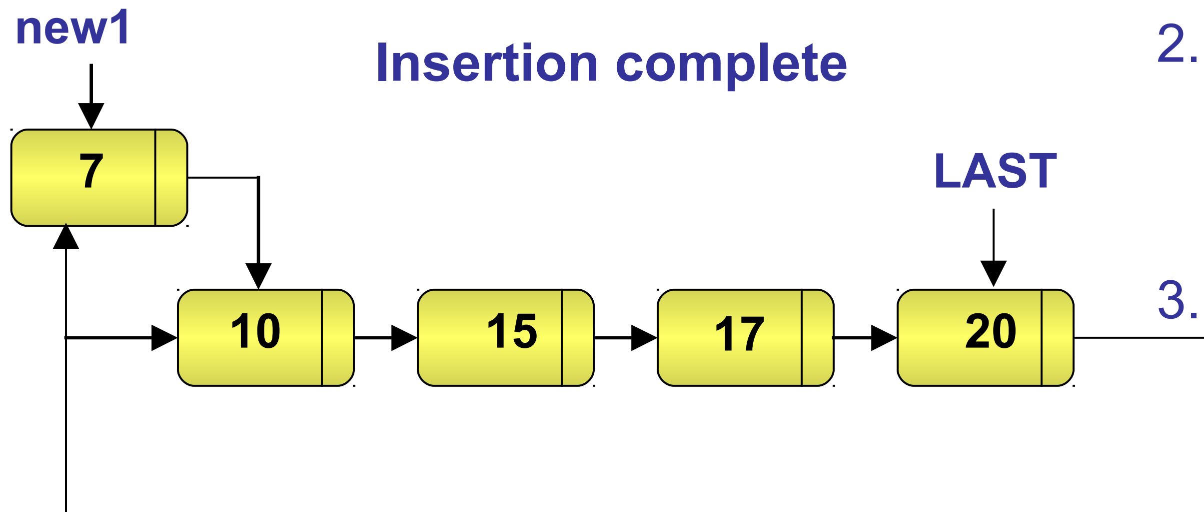
Inserting a Node at the Beginning of the List (Contd.)



new1 -> next = LAST ->next

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 successor of LAST.
4. Make the next field of LAST point to the new node.

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



LAST -> next = 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 successor of LAST.
4. Make the next field of LAST point to the new node.

ALGORITHM TO INSERT NODE AT BEGINING

Last=NULL

Algorithm InsertAtBEG()

{

1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info = data]
3. If (Last == NULL)

    3.1 new1 -> next = new1

    3.2 Last=new1

  else

    3.1 new1 -> next = Last -> next

    3.2 Last -> next = new1

}

- The algorithm to insert a node between two nodes in a circular linked list is same as that of a singly-linked list.
- However, the algorithm to search for the nodes between which the new node is to be inserted is a bit different in a circular linked list.

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

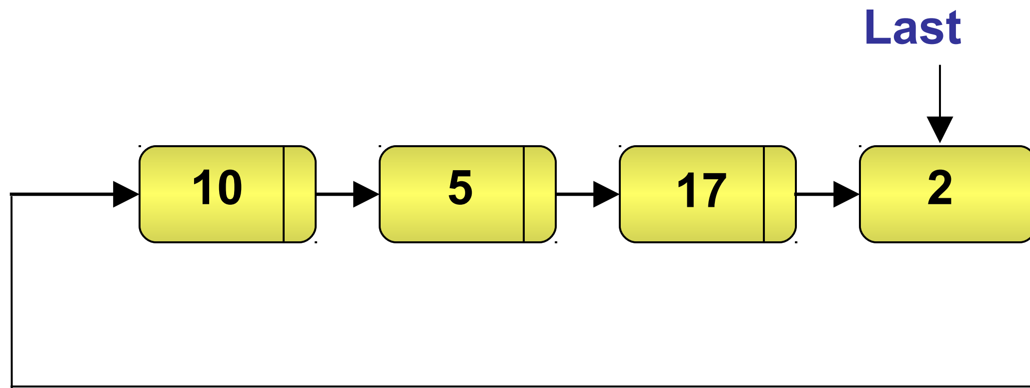
1.Allocate memory for the new node.

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

Inserting a Node in a Singly-Linked List (Contd.)

Insert 16 At LOC = 3

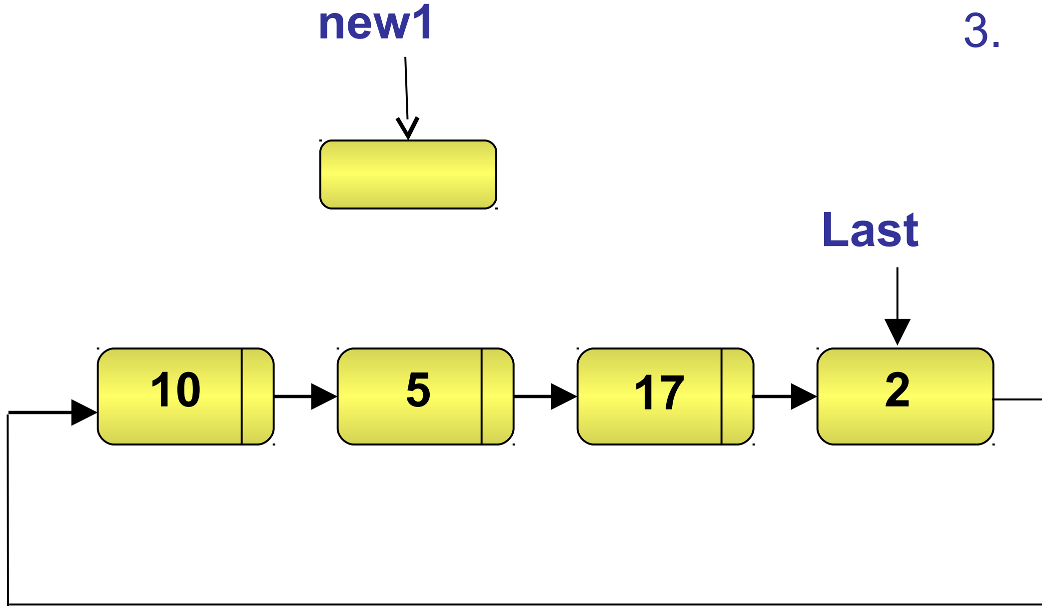
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence



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

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

Inserting a Node in a Singly-Linked List (Contd.)



1. Allocate memory for the new node.

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

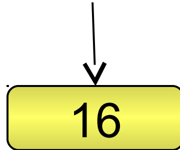
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

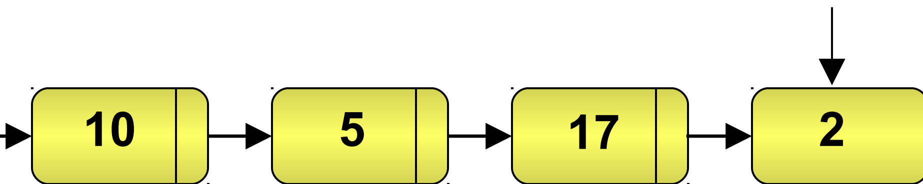
4. Make the next field of the new node point to the next of previous node

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

Inserting a Node in a Singly-Linked List (Contd.)

Insert 16
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

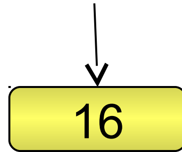
4. Make the next field of the new node point to the next of previous node

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

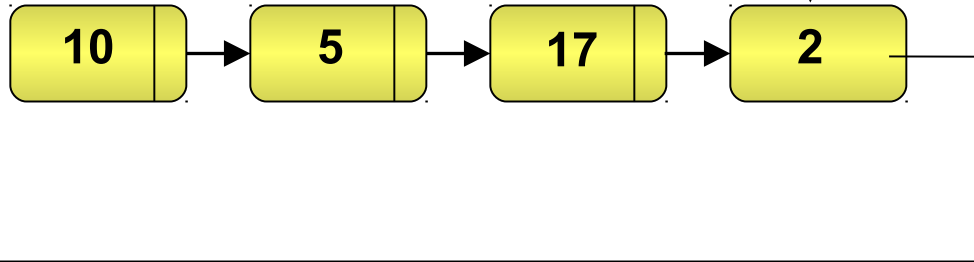
Inserting a Node in a Singly-Linked List (Contd.)

Insert 16
LOC = 3

new1



Last



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 next field of previous point to the new node.

Inserting a Node in a Singly-Linked List (Contd.)

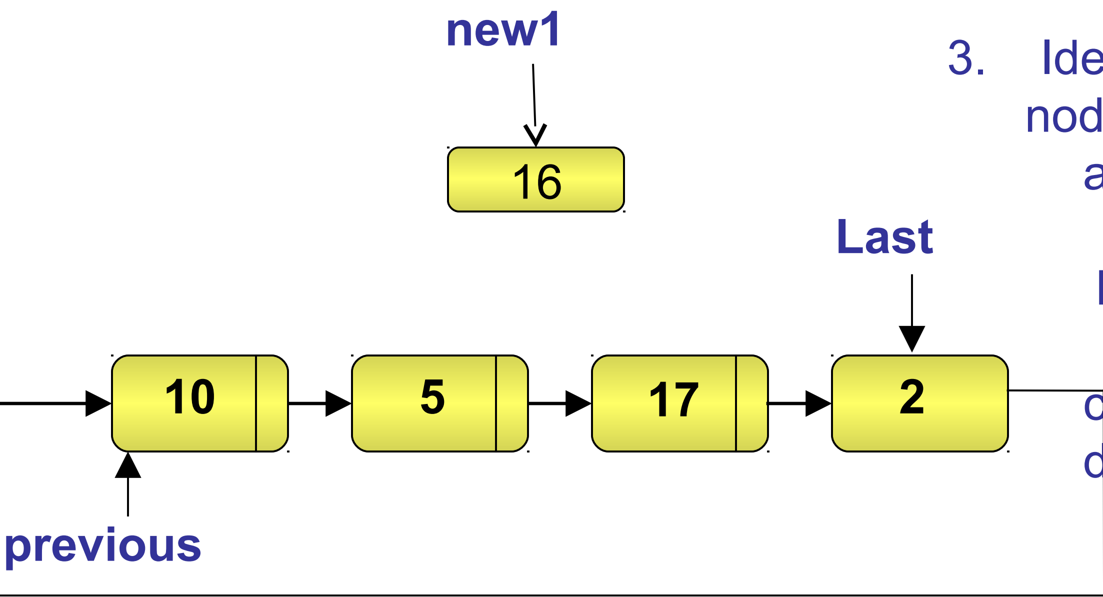
1. Allocate memory for the new node.

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

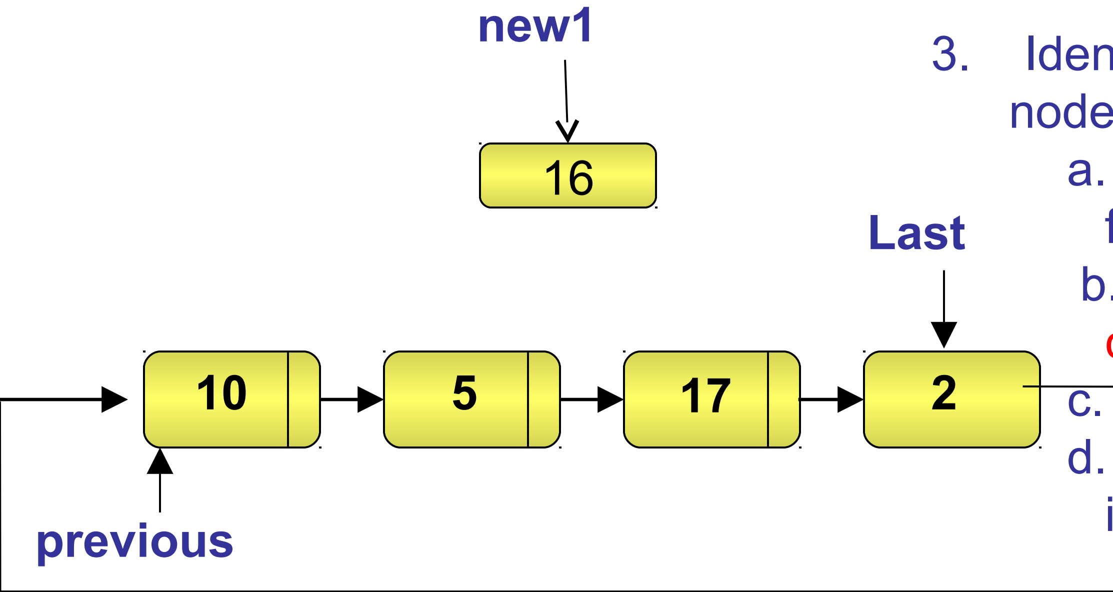
4. Make the next field of the new node point to the next of previous node

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



Count=1

Inserting a Node in a Singly-Linked List (Contd.)



Count=1

1. Allocate memory for the new node.

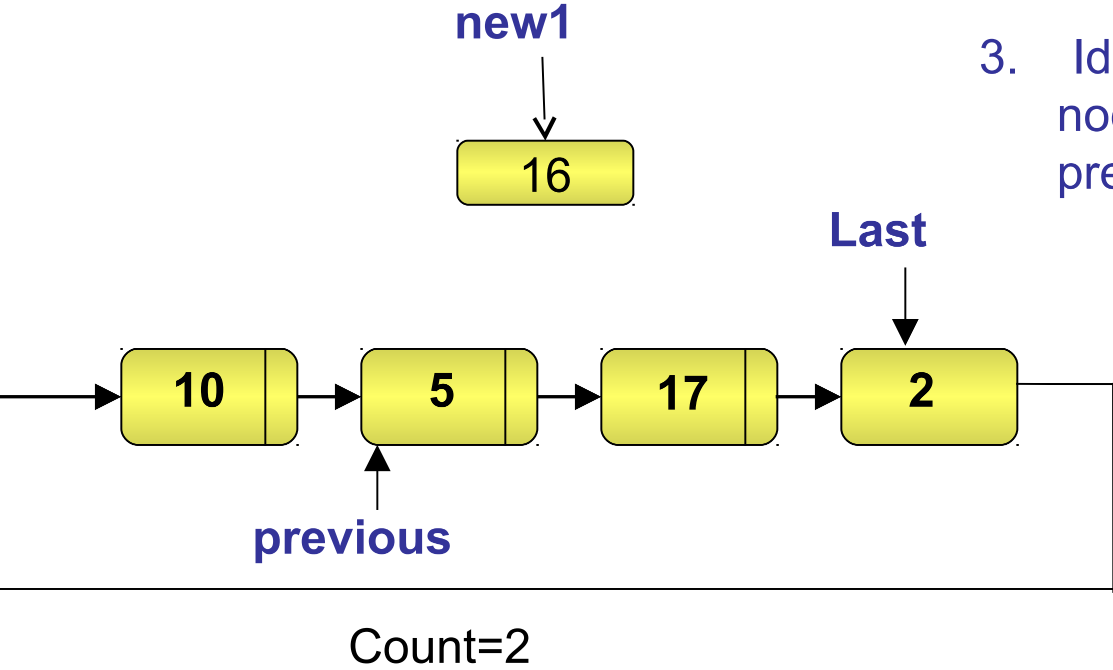
2. Assign value to the data field of the new node.

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 next field of previous point to the new node.

Inserting a Node in a Singly-Linked List (Contd.)



1. Allocate memory for the new node.

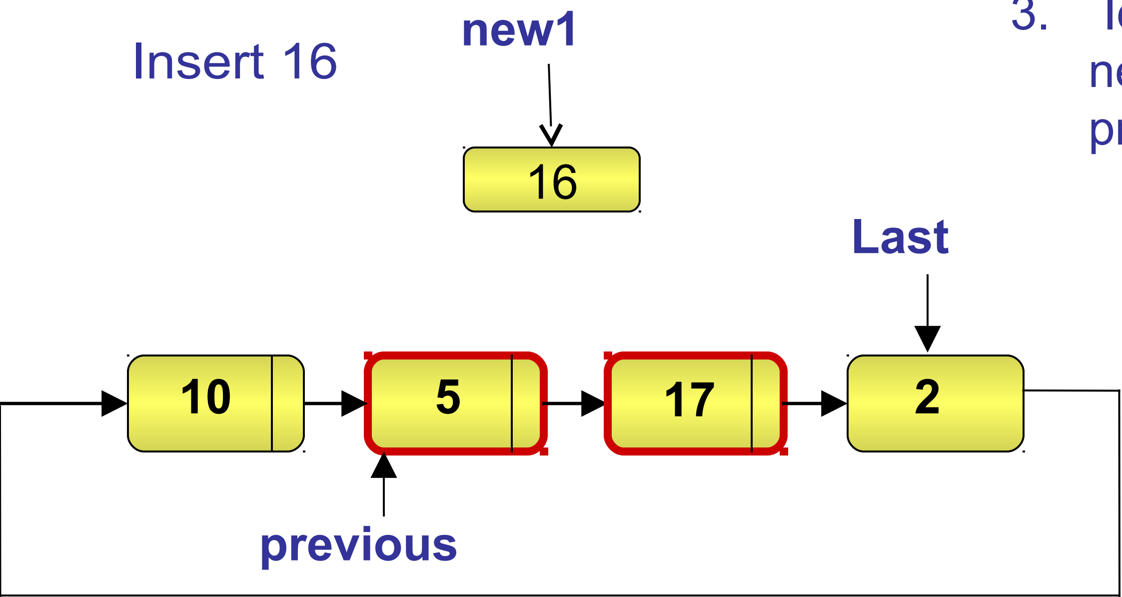
2. Assign value to the data field of the new node.

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 next field of previous point to the new node.

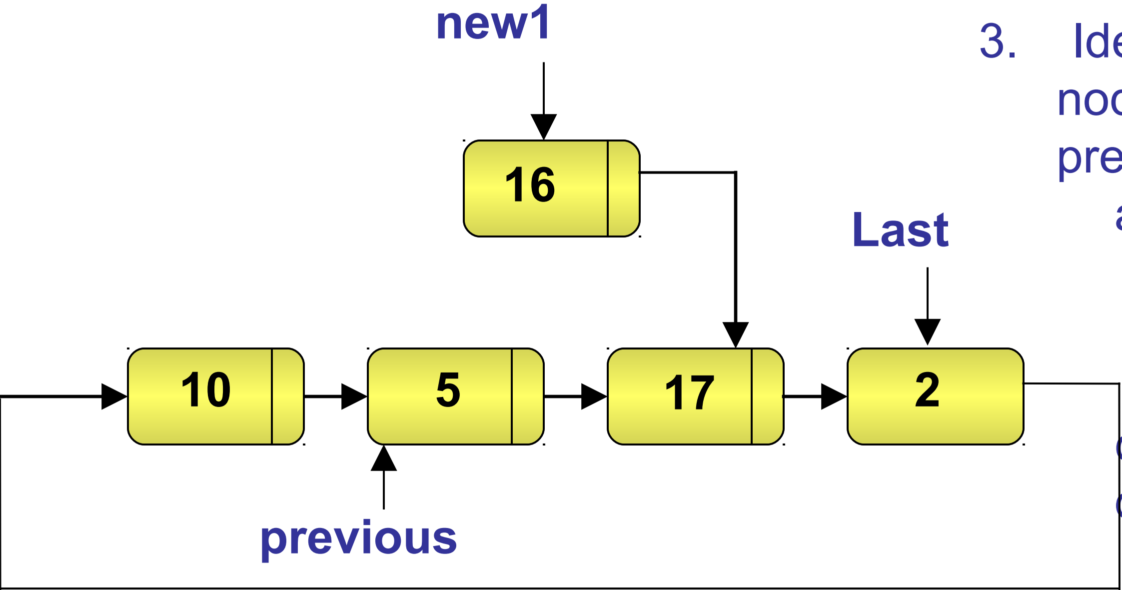
Inserting a Node in a Singly-Linked List (Contd.)



Nodes located

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 next field of previous point to the new node.

Inserting a Node in a Singly-Linked List (Contd.)



1. Allocate memory for the new node.

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

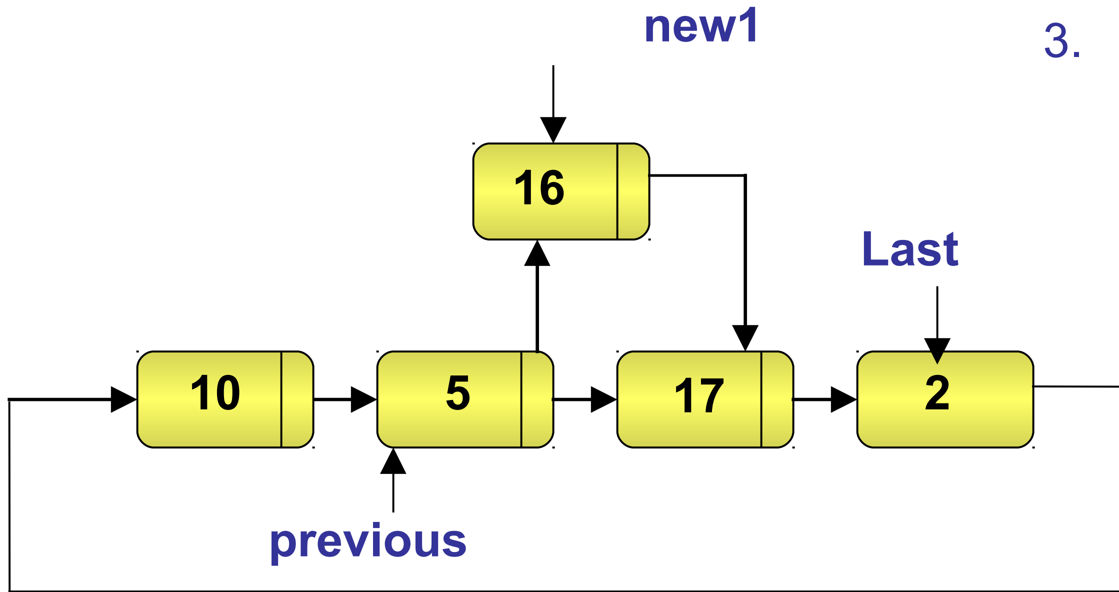
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

new1 -> next = previous->next

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

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

Inserting a Node in a Singly-Linked List (Contd.)

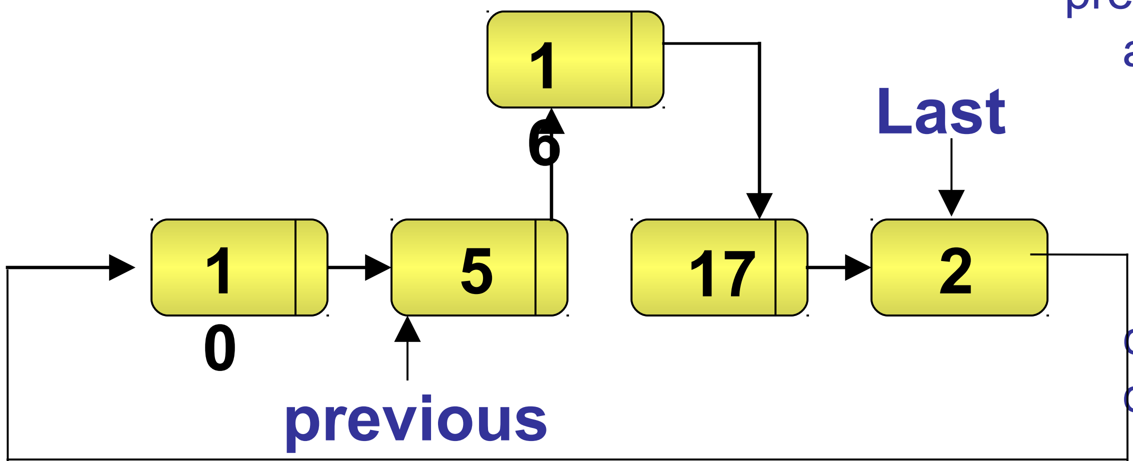


new1 -> next = previous->next

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 next field of previous point to the new node.

Inserting a Node in a Singly-Linked 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 next field of previous point to the new node.

Insertion complete

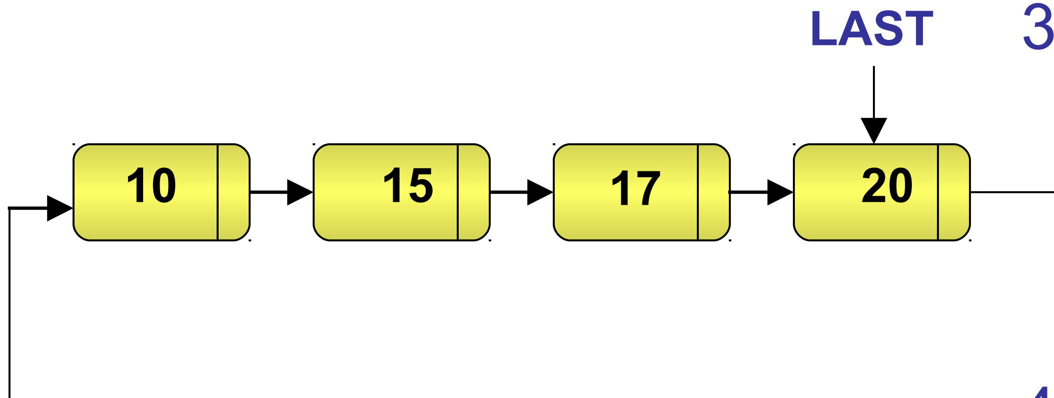
ALGORITHM TO INSERT NODE At PARTICULAR POSITION IN A LINKED LIST

```
Algorithm InsertAtSpec()
{
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter Data and Location
3. new1->info=Data
4. If (Location == 1)
        4.1 new1 -> next = Last -> next
        4.2 Last -> next = new1
   Else
        4.1 Previous = Last -> next
        4.2 Count = 1
        4.3 While( Count <= Location - 1)
                        4.3.1 Previous = Previous -> next
                        4.3.2 Count++
        4.4 if (Prev == Last)
                        4.4.1 new1 -> next = Last -> next
                        4.4.2 Last ->next=new1;
                        4.4.3 Last = new1;
            else
                        4.4.1 new1 -> next = Previous -> next
                        4.4.2 Previous -> next = new1
}
```

- Write an algorithm to insert a node at the end of a circular linked list.

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

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

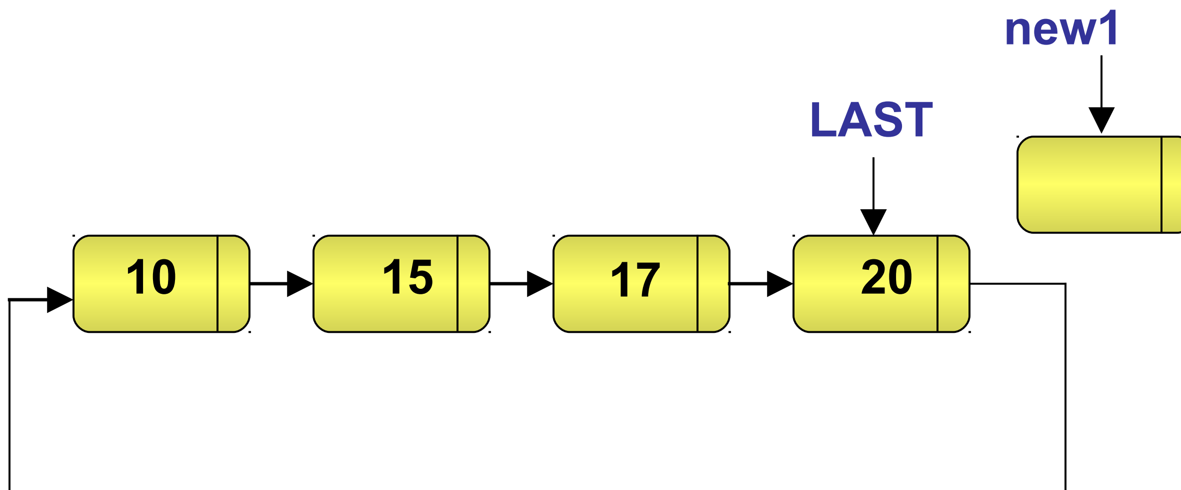


- Let us insert a node after node 20

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 successor of LAST.
4. Make the next field of LAST point to the new node.
5. Mark LAST point to the new node.

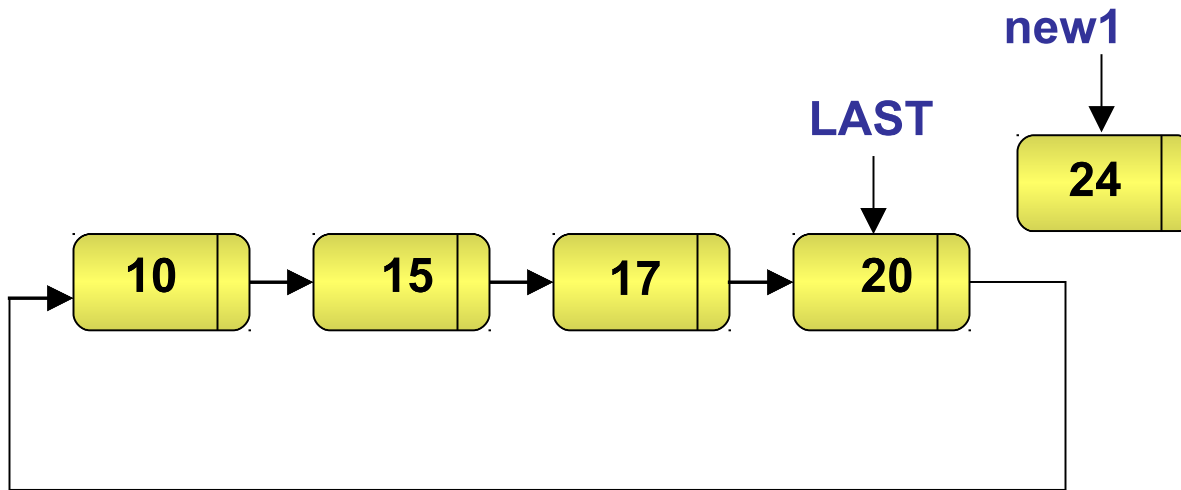
Inserting a Node at the End of the List (Contd.)

- Let us insert a node after node 20



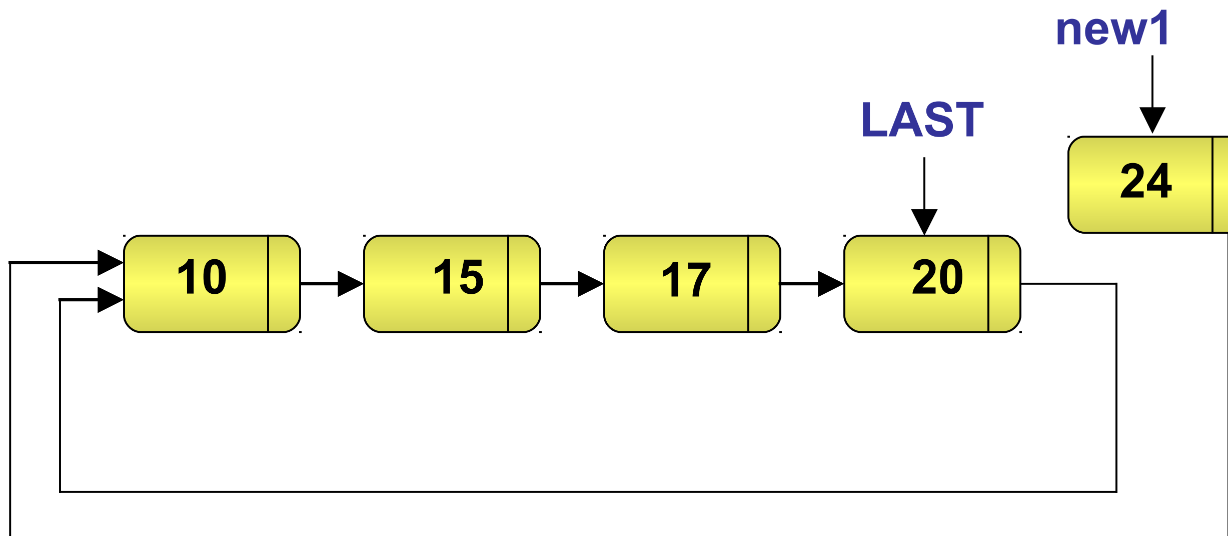
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. Make the next field of the new node point to the successor of LAST.
4. Make the next field of LAST point to the new node.
5. Mark LAST point to the new node.

Inserting a Node at the End 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 successor of LAST.
4. Make the next field of LAST point to the new node.
5. Mark LAST point to the new node.

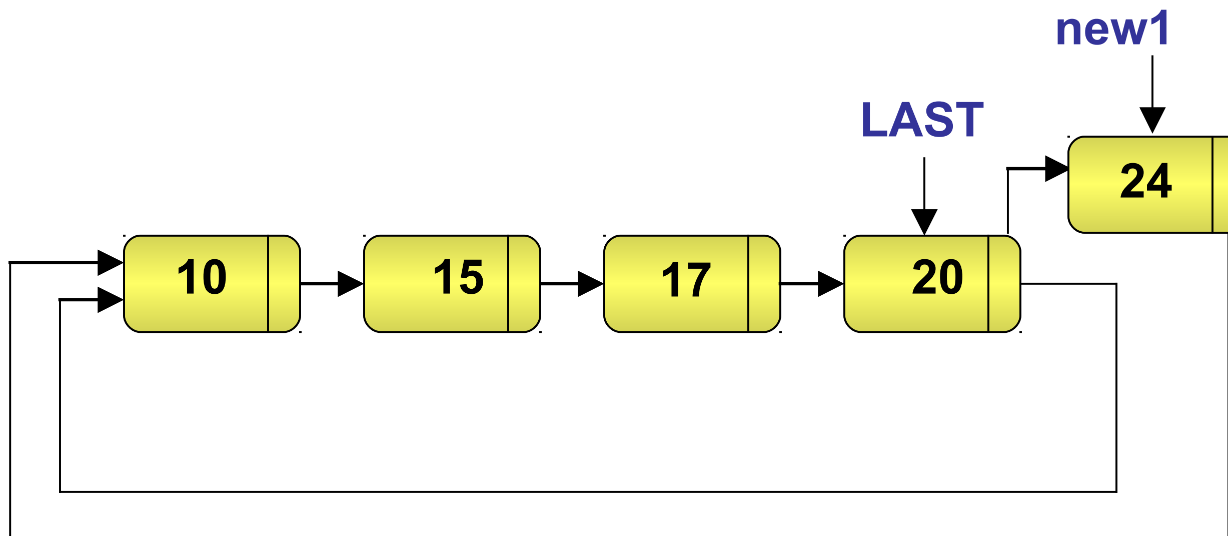
Inserting a Node at the End of the List (Contd.)



new1 -> next = LAST -> next

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 successor of LAST.
4. Make the next field of LAST point to the new node.
5. Mark LAST point to the new node.

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

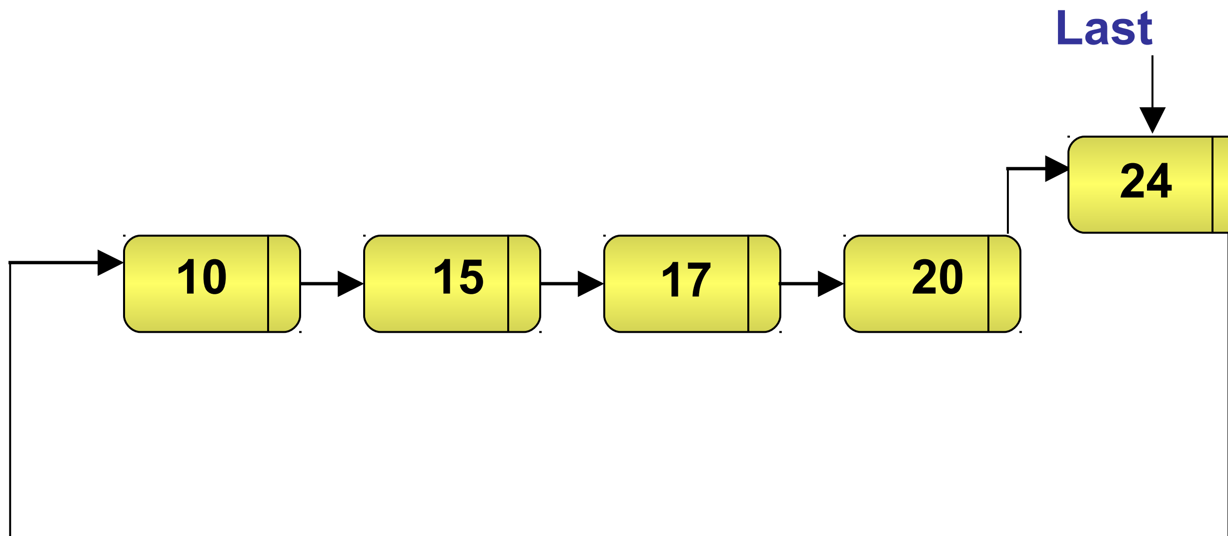


LAST -> next = new1

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 successor of LAST.
4. Make the next field of LAST point to the new node.
5. Mark LAST point to the new node.

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

Insertion complete



LAST = newnode

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 successor of LAST.
4. Make the next field of LAST point to the new node.
5. Mark LAST point to the new node.

ALGORITHM TO INSERT NODE AT END

Last=NULL

Algorithm InsertAtEnd()

{

1. Create node [(new1 = (struct node*) malloc(sizeof(struct node))]

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

3.if(Last == NULL)

    3.1 new1 -> next = new1

    3.2 Last=new1

else

    3.1 new1 -> next = Last -> next

    3.2 Last -> next = new1

    3.3 Last = new1

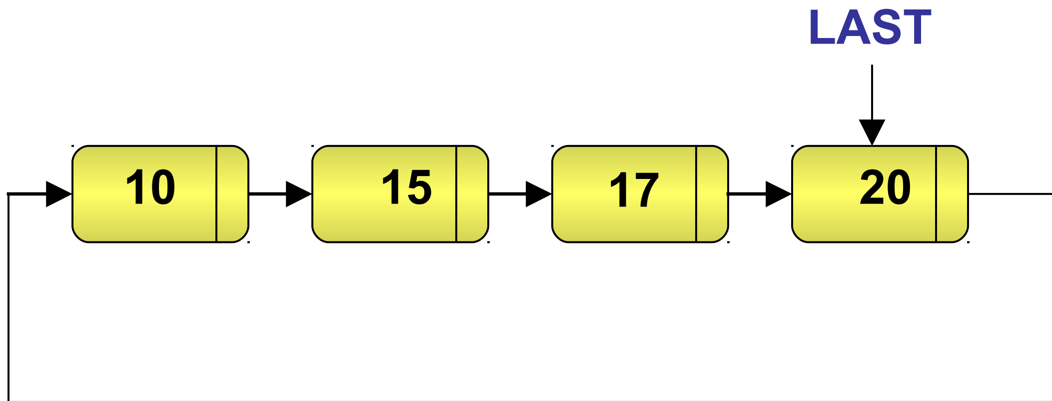
}

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

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

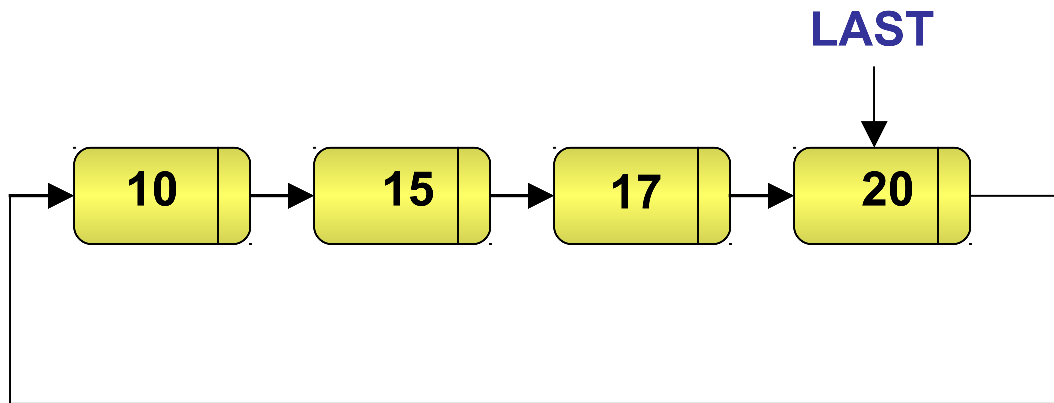
Deleting a Node From the Beginning of the List (Contd.)

- Algorithm for deleting a node from the beginning of a circular linked list.



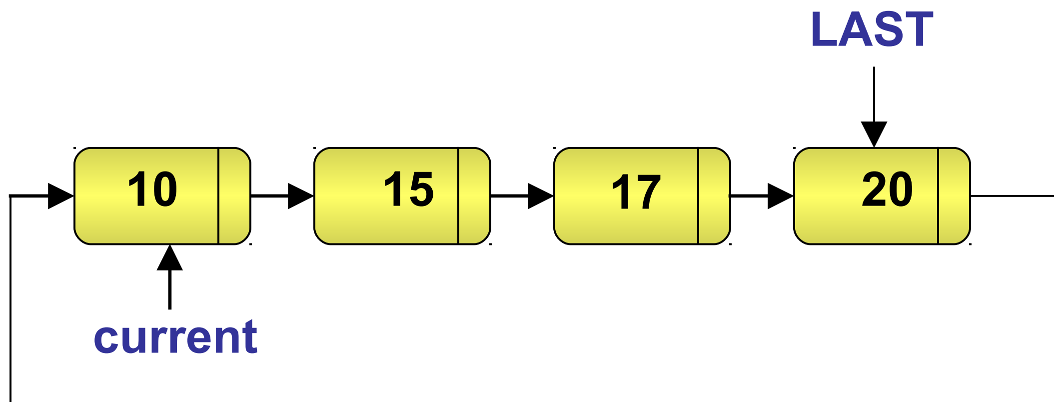
1. If the node to be deleted is the only node in the list: **// If LAST points to itself**
   a. Mark LAST as NULL
   b. Exit
2. Make current point to the successor of LAST
3. Make the next field of LAST point to the successor of current
4. Release the memory for the node marked as current

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



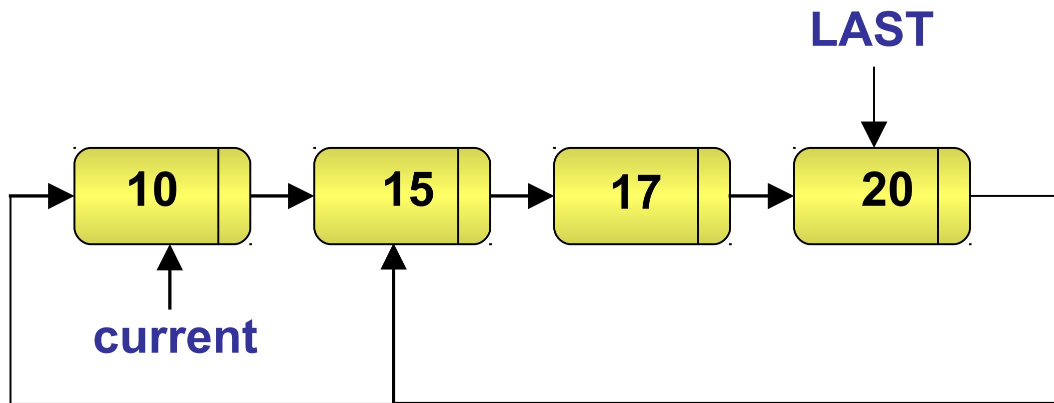
1. If the node to be deleted is the only node in the list: **// If LAST points to itself**
   a. Mark LAST as NULL
   b. Exit
2. Make current point to the successor of LAST
3. Make the next field of LAST point to the successor of current
4. Release the memory for the node marked as current

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



1. If the node to be deleted is the only node in the list: **// If LAST points to itself**
    a. Mark LAST as NULL
    b. Exit
2. Make current point to the successor of LAST
3. Make the next field of LAST point to the successor of current
4. Release the memory for the node marked as current

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

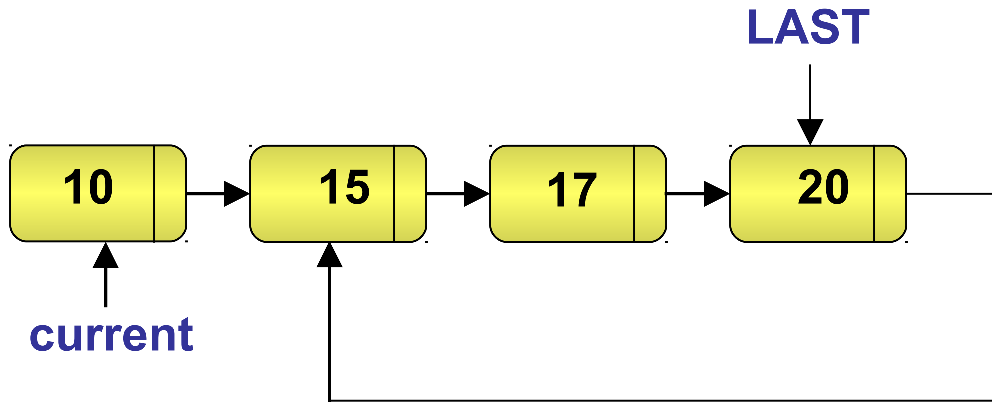


1. If the node to be deleted is the only node in the list: **// If LAST points to itself**
   a. Mark LAST as NULL
   b. Exit
2. Make current point to the successor of LAST
3. Make the next field of LAST point to the successor of current
4. Release the memory for the node marked as current

LAST -> next = current -> next

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

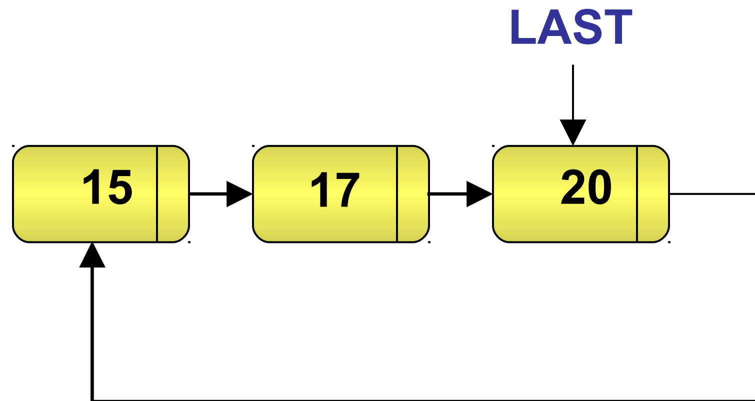
Deletion Complete



1. If the node to be deleted is the only node in the list: **// If LAST points to itself**
   a. Mark LAST as NULL
   b. Exit
2. Make current point to the successor of LAST
3. Make the next field of LAST point to the successor of current
4. Release the memory for the node marked as current

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

Deletion Complete



1. If the node to be deleted is the only node in the list: **// If LAST points to itself**
   a. Mark LAST as NULL
   b. Exit
2. Make current point to the successor of LAST
3. Make the next field of LAST point to the successor of current
4. Release the memory for the node marked as current

ALGORITHM TO DELETE A NODE FROM THE BEGINING

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

- Delete operation in between two nodes in a circular linked list is same as that of a singly-linked list.
- However, the algorithm to locate the node to be deleted is a bit different in a circular linked list.

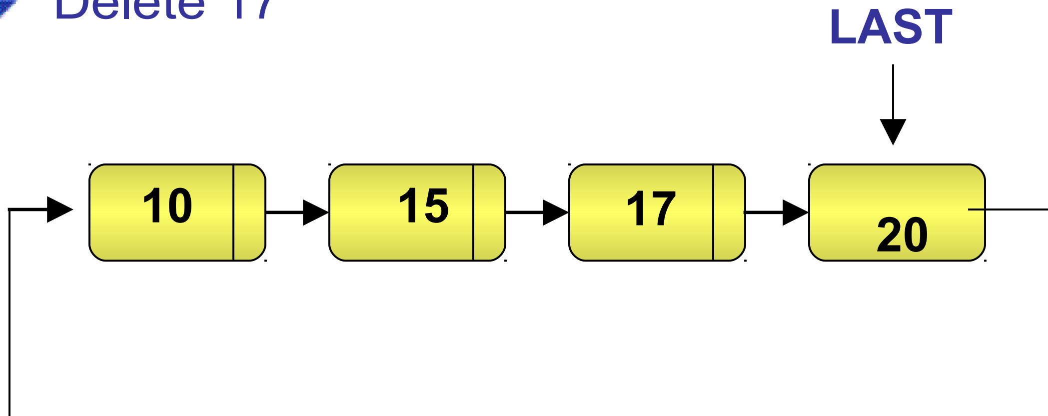
- Write an algorithm to delete a node from a specific position in a linked list.

Deleting a Node

Between two Nodes in the List (Contd.)

- Algorithm to delete a node from a specific position.

- Delete 17



1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
   a. Set previous = NULL
   b. Set current = first node
   c. Repeat step d and e until either the node is found or previous
   d. becomes Last.
   e. Make previous point to current .
   f. Make current point to the next node in sequence.

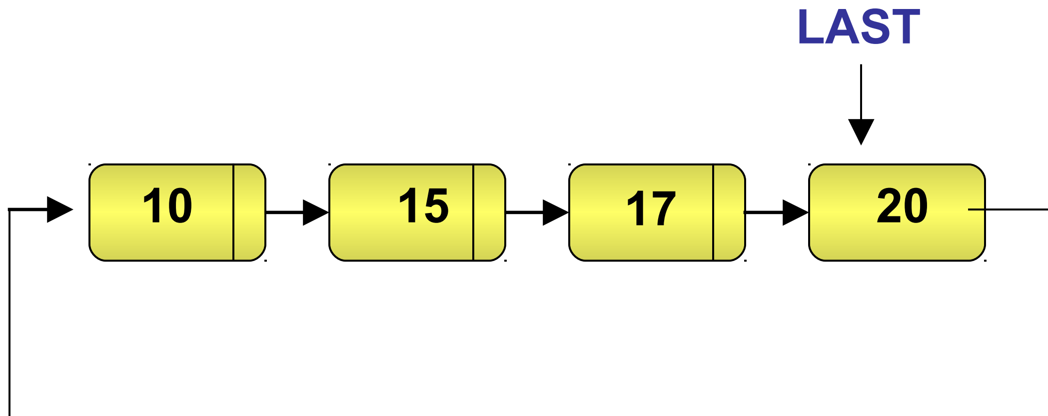
2. Make the next field of previous point to the successor of current.

3. Release the memory for the node marked as current.

Deleting a Node

Between two Nodes in the List (Contd.)

- Delete 17



1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
   a. Set previous = NULL
   b. Set current = first node
   c. Repeat step d and e until either the node is found or current becomes .
   d. Make previous point to current.
   e. Make current point to the next node in sequence.

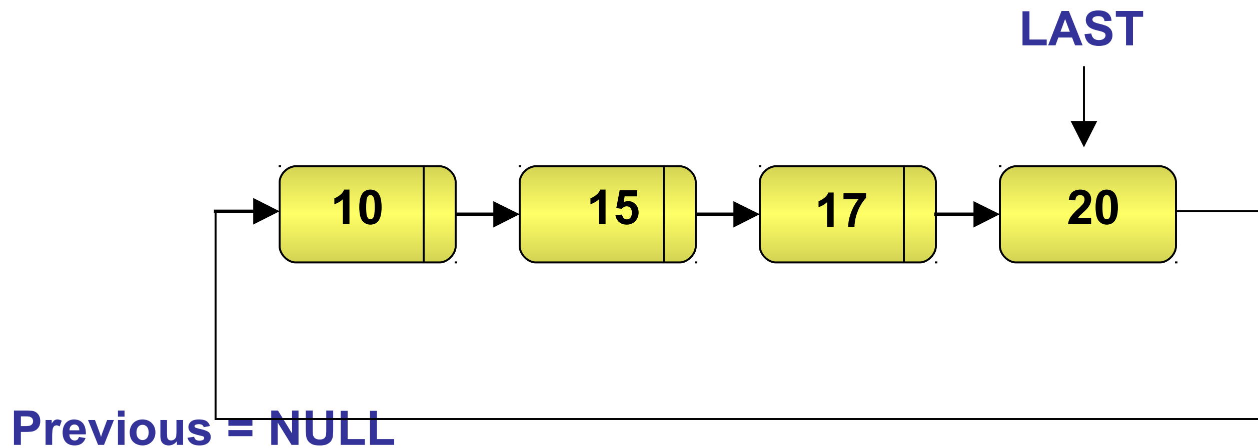
2. Make the next field of previous point to the successor of current.

3. Release the memory for the node marked as current.

Deleting a Node

Between two Nodes in the List (Contd.)

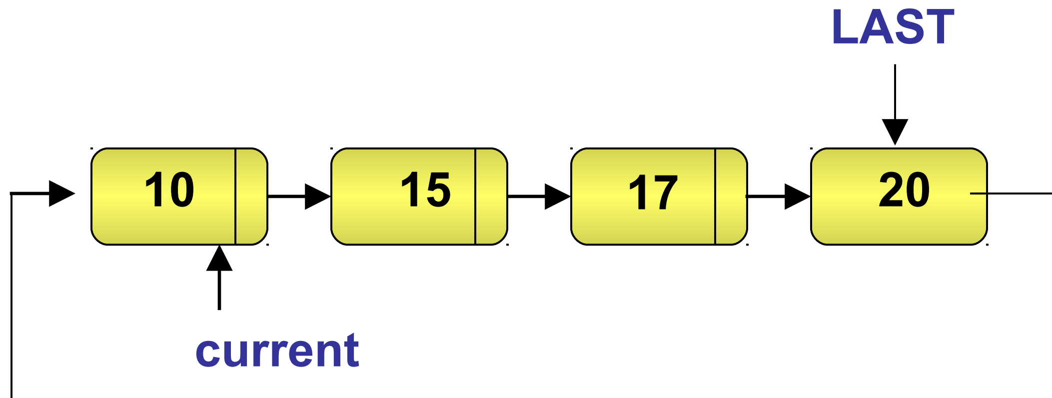
- Delete 17



Deleting a Node

Between two Nodes in the List (Contd.)

- Delete 17

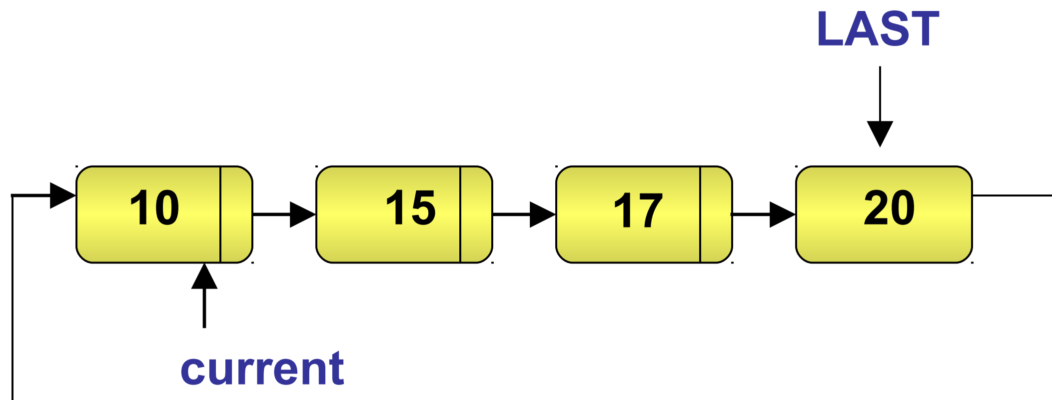


Previous = NULL

Deleting a Node

Between two Nodes in the List (Contd.)

- Delete 17

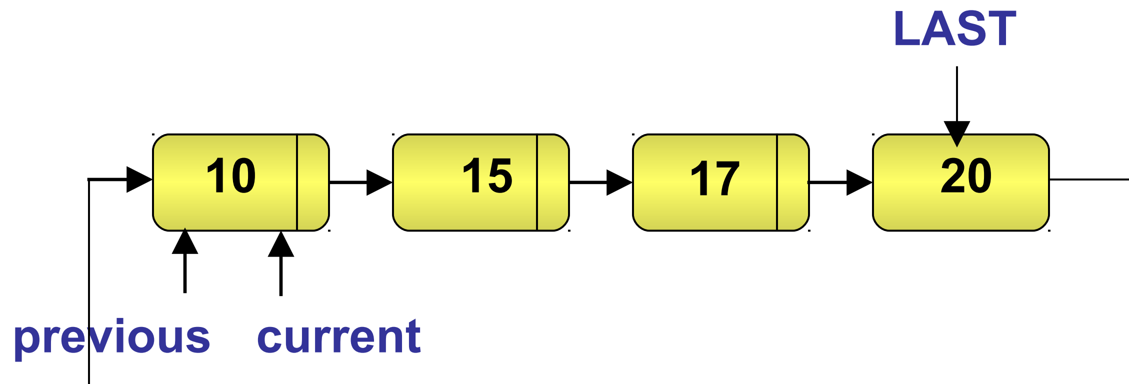


Previous = NULL

Deleting a Node

Between two Nodes in the List (Contd.)

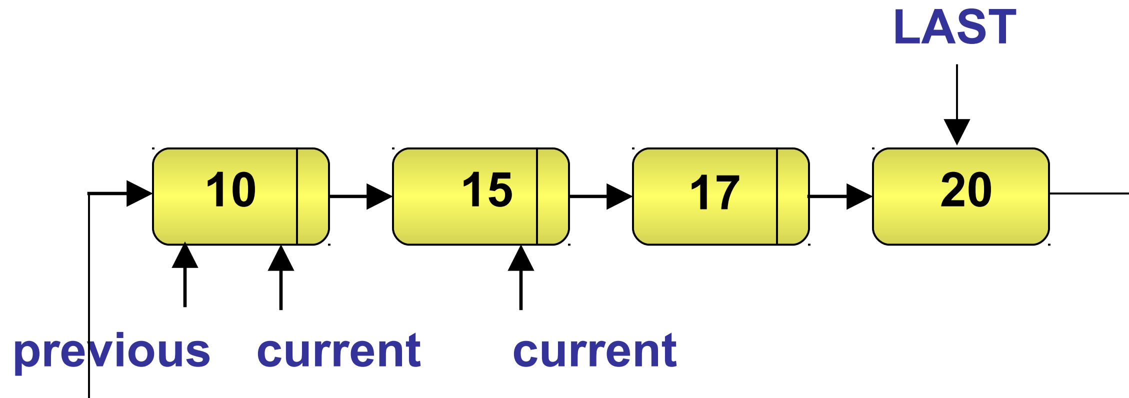
- Delete 17



Deleting a Node

Between two Nodes in the List (Contd.)

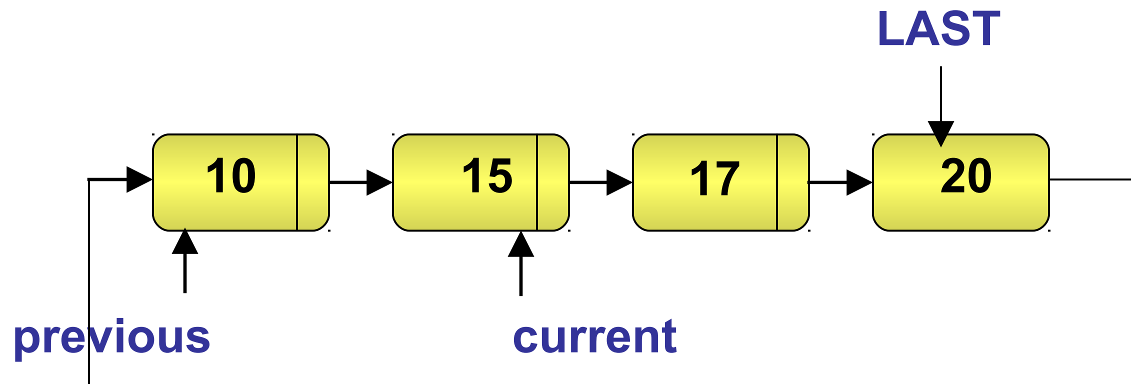
- Delete 17



Deleting a Node

Between two Nodes in the List (Contd.)

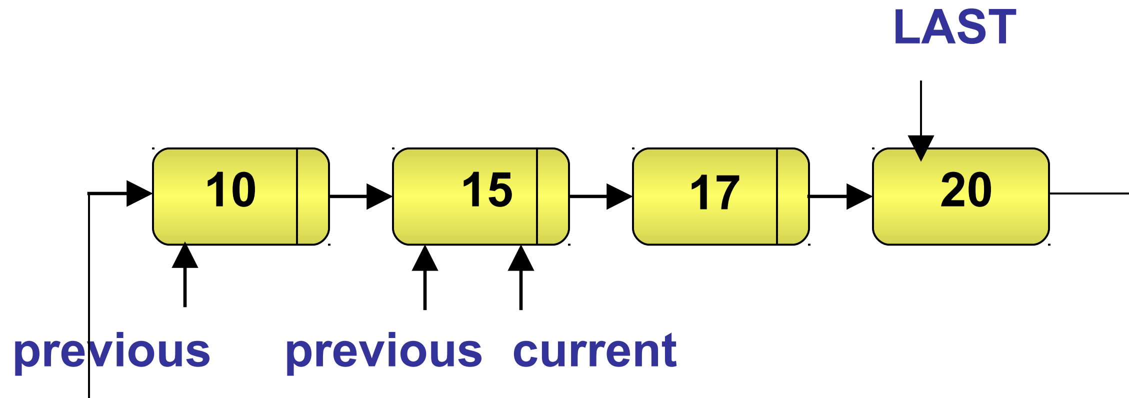
- Delete 17



Deleting a Node

Between two Nodes in the List (Contd.)

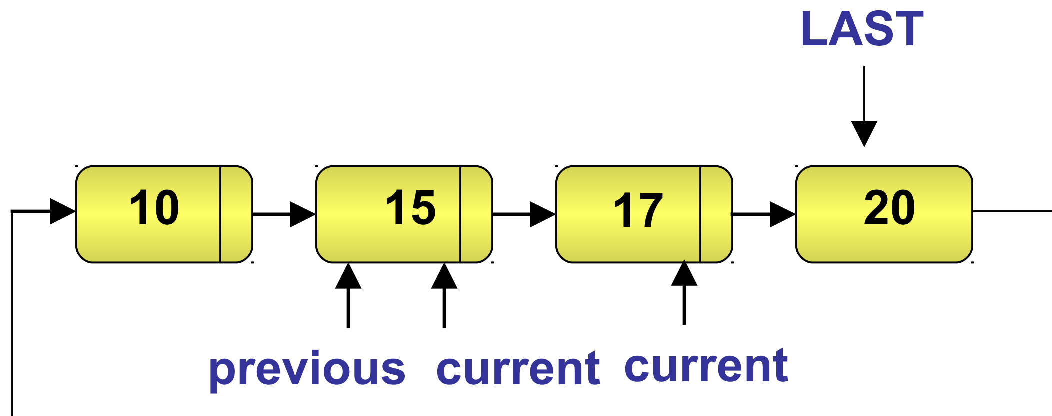
- Delete 17



Deleting a Node

Between two Nodes in the List (Contd.)

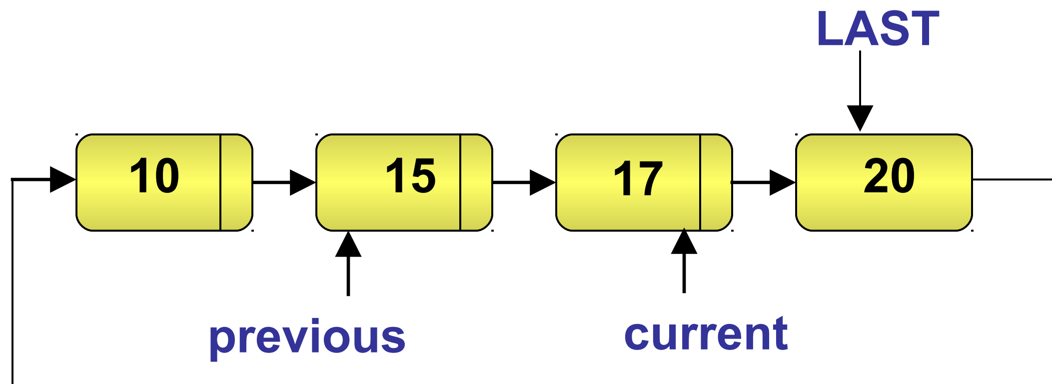
- Delete 17



Deleting a Node

Between two Nodes in the List (Contd.)

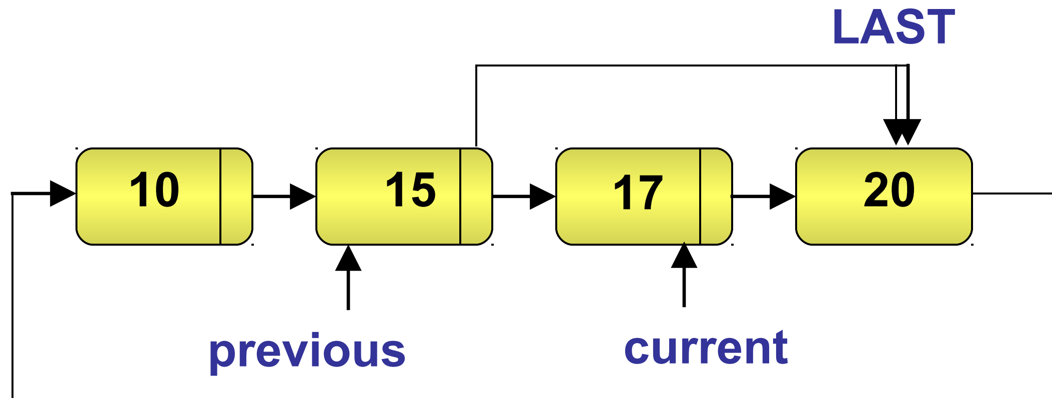
- Delete 17



Deleting a Node

Between two Nodes in the List (Contd.)

- Delete 17



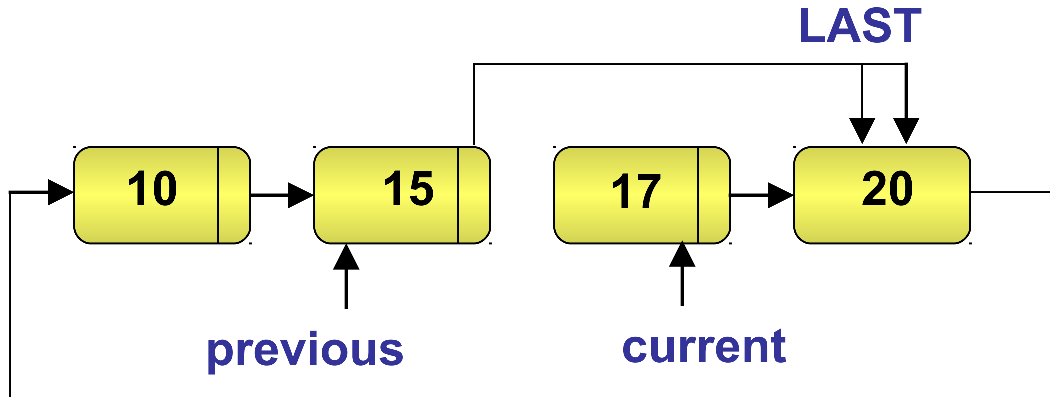
Previous -> next = current -> next

Deleting a Node

Between two Nodes in the List (Contd.)

- Delete 17

Delete operation complete



Previous -> next = current -> next

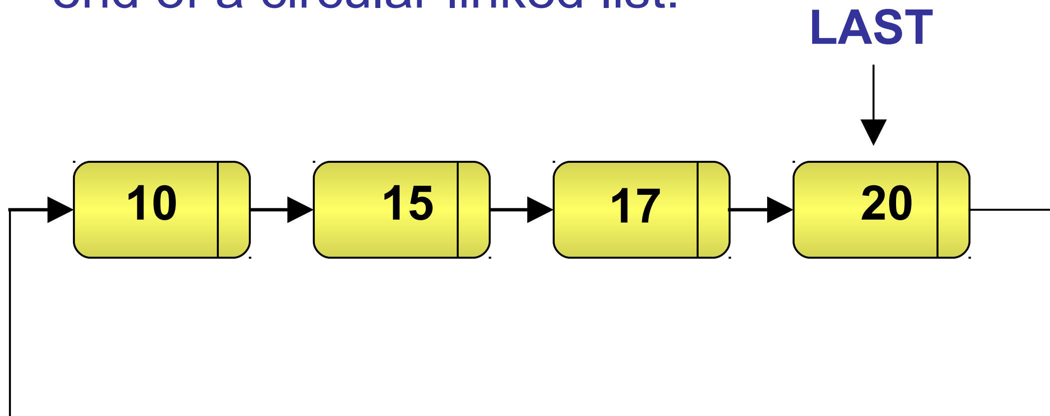
ALGORITHM TO DELETE A NODE FROM THE SPECIFIC POSITION

```
Algorithm DeleteAtSpec()
{
1. Enter the Location
2. Current = Last -> next
3. Previous = NULL
4. If (Last == NULL)
            4.1 Print "underflow“
   else If ( Location == 1)
            4.1 Last -> next = Current -> next
            4.2 Current -> next = NULL
            4.3 Release the memory [ free (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 == Last)
                        4.2.1 prev -> next = Current -> next
                        4.2.2 Last = Prev
                else
                        4.2.1 Previous -> next = Current -> next
            4.3 Current -> next = NULL
            4.4 Release the memory [ free (Current) ]
}
```

- Write an algorithm to delete a node from the end of a circular linked list.

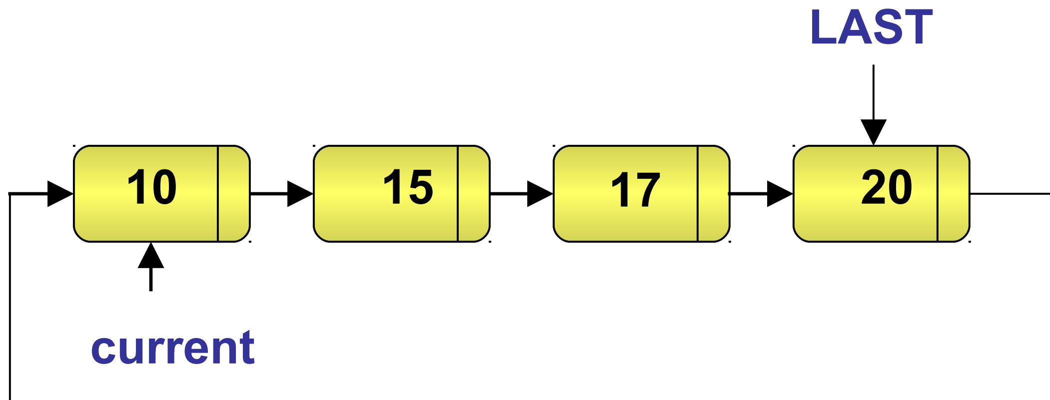
Deleting a Node From the End of the List (Contd.)

- Algorithm for deleting a node from the end of a circular linked list.



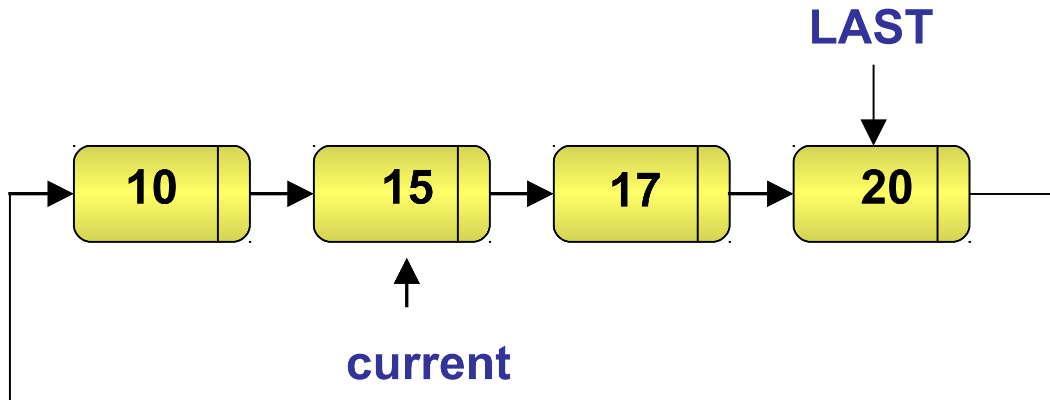
1. Make current point to first node.
2. Repeat step c until the successor of current becomes LAST.
   a. Make current point to the next node in its sequence.
3. Make the next field of current point to the successor of LAST.
4. Release the memory for the node marked as LAST.
5. Mark current as LAST.

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



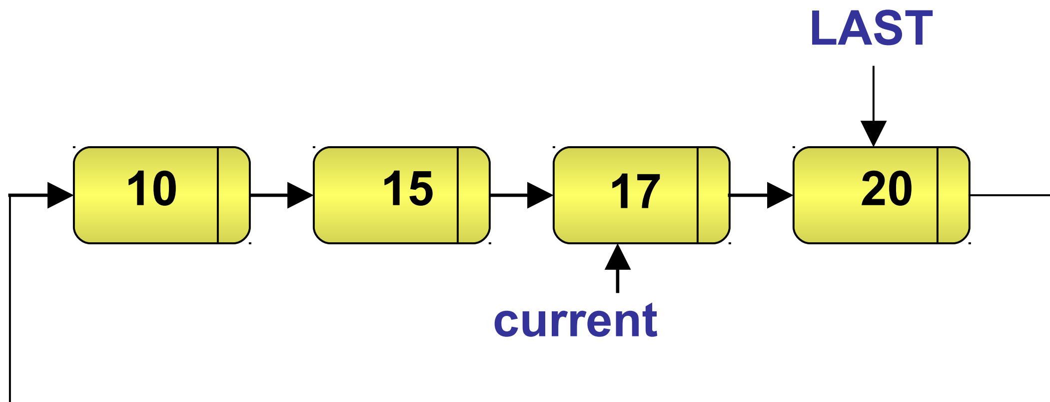
1. Make current point to LAST.
2. Mark the predecessor of LAST as previous. To locate the predecessor of LAST, execute the following steps:
   a. Make previous point to the successor of LAST.
   b. Repeat step c until the successor of previous becomes LAST.
   c. Make previous point to the next node in its sequence.
3. Make the next field of previous point to the successor of LAST.
4. Mark previous as LAST.
5. Release the memory for the node marked as current.

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



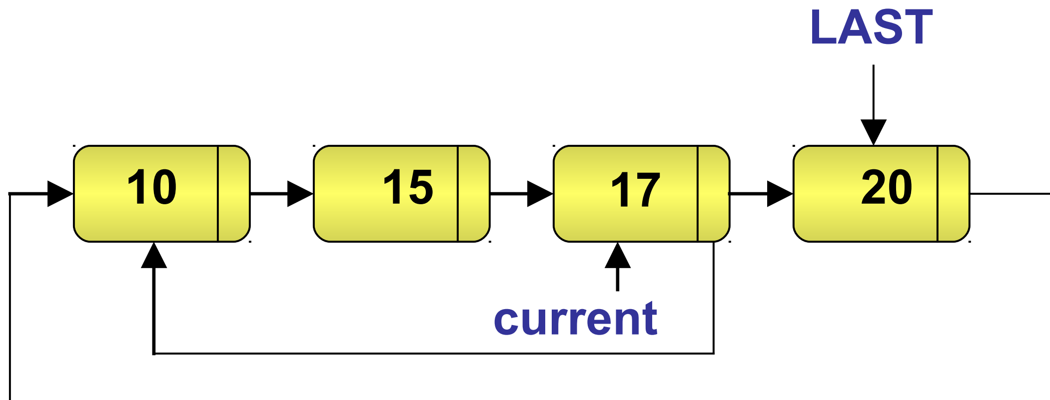
1. Make current point to LAST.
2. Mark the predecessor of LAST as previous. To locate the predecessor of LAST, execute the following steps:
   a. Make previous point to the successor of LAST.
   b. Repeat step c until the successor of previous becomes LAST.
   c. Make previous point to the next node in its sequence.
3. Make the next field of previous point to the successor of LAST.
4. Mark previous as LAST.
5. Release the memory for the node marked as current.

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



1. Make current point to LAST.
2. Mark the predecessor of LAST as previous. To locate the predecessor of LAST, execute the following steps:
   a. Make previous point to the successor of LAST.
   b. Repeat step c until the successor of previous becomes LAST.
   c. Make previous point to the next node in its sequence.
3. Make the next field of previous point to the successor of LAST.
4. Mark previous as LAST.
5. Release the memory for the node marked as current.

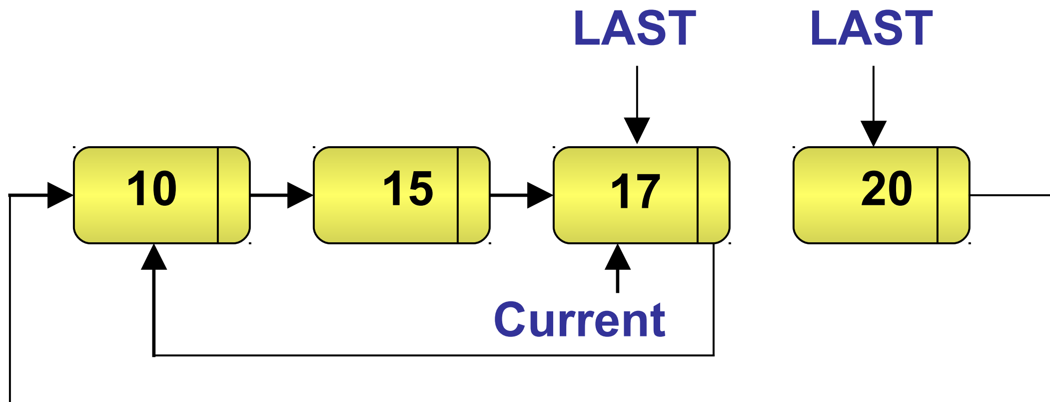
Deleting a Node From the End of the List (Contd.)



Current -> next = LAST -> next

1. Make current point to LAST.
2. Mark the predecessor of LAST as previous. To locate the predecessor of LAST, execute the following steps:
   a. Make previous point to the successor of LAST.
   b. Repeat step c until the successor of previous becomes LAST.
   c. Make previous point to the next node in its sequence.
3. Make the next field of previous point to the successor of LAST.
4. Mark previous as LAST.
5. Release the memory for the node marked as current.

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



LAST = Current

1. Make current point to LAST.
2. Mark the predecessor of LAST as previous. To locate the predecessor of LAST, execute the following steps:
    a. Make previous point to the successor of LAST.
    b. Repeat step c until the successor of previous becomes LAST.
    c. Make previous point to the next node in its sequence.
3. Make the next field of previous point to the successor of LAST.
4. Mark previous as LAST.
5. Release the memory for the node marked as current.

ALGORITHM TO DELETE A NODE FROM THE END

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

}
```