
K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

# Data Structure

ENCS205

*School of Engineering & Technology (SOET)*
*K.R. Mangalam University*

UNIT-2

Session 18: SINGLY LINKED LIST Operations II

# Recap

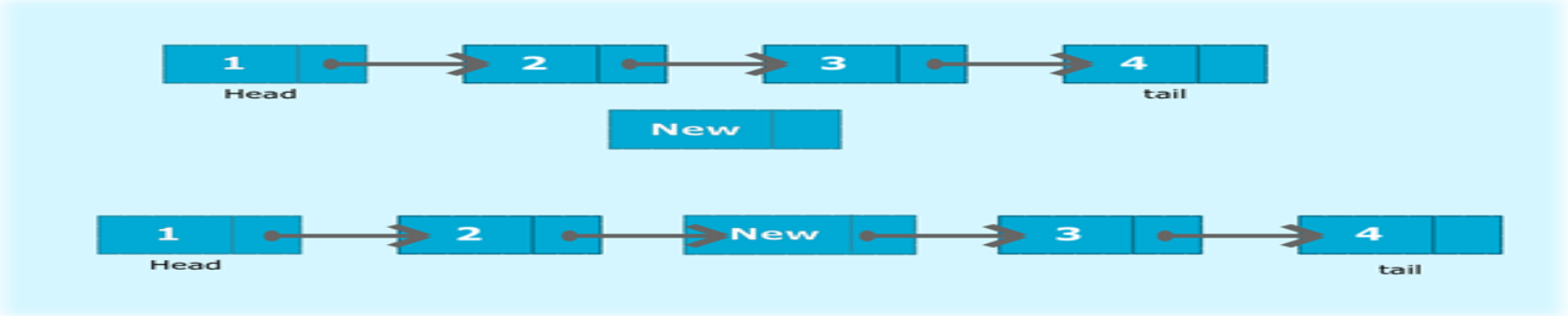
**Definition**: Linked list is a linear data structure with nodes containing data and pointers to the next node.
**Components**: Nodes, head pointer (points to first node), optional tail pointer (points to last node).
**Representation**: Nodes linked via pointers, dynamic memory allocation for nodes, head pointer for access.
**Advantages**: Dynamic memory allocation, efficient insertion/deletion.
**Disadvantages**: Higher memory overhead, no direct access to elements, potentially slower access compared to arrays.

# Sessions 18 Outlook

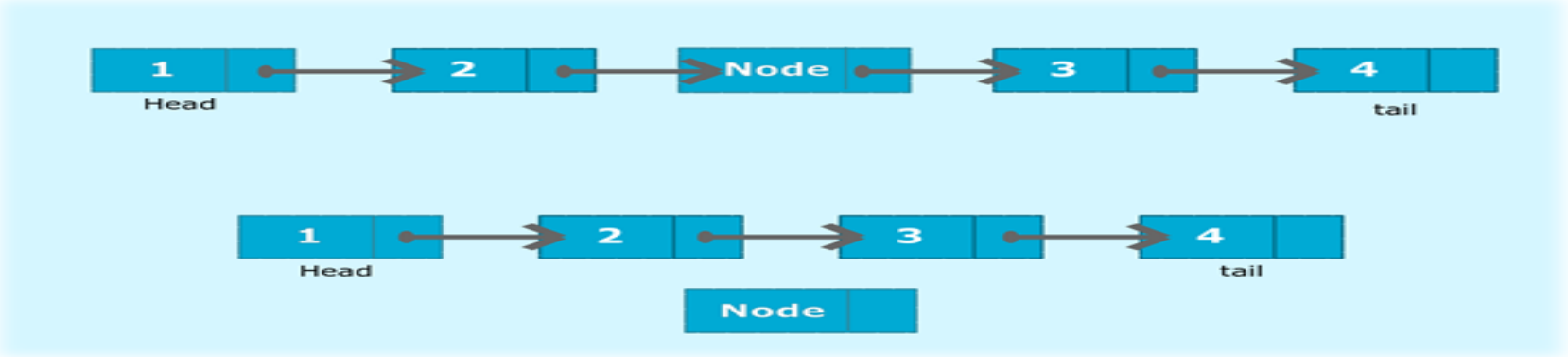
- Traversing a list
- Inserting a Node at different position
- Deleting a Node at different position

# Operations on Singly Linked List

**Insertion**




**Deletion**

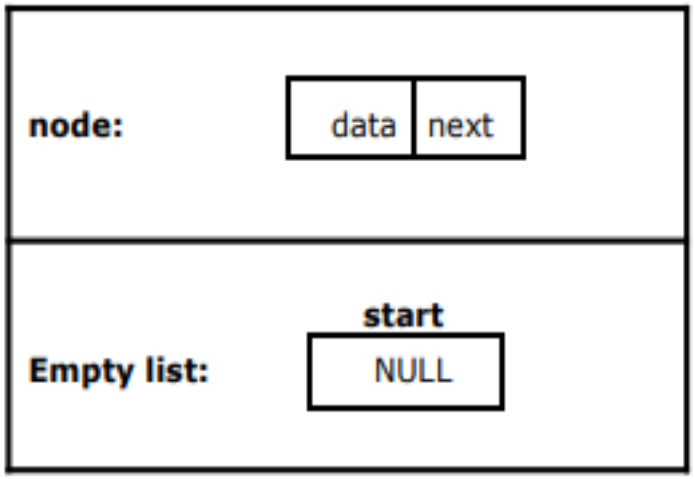


# Implementation of Single Linked List

we need to create a start node, used to create and access other nodes in the linked list. The following structure definition will do

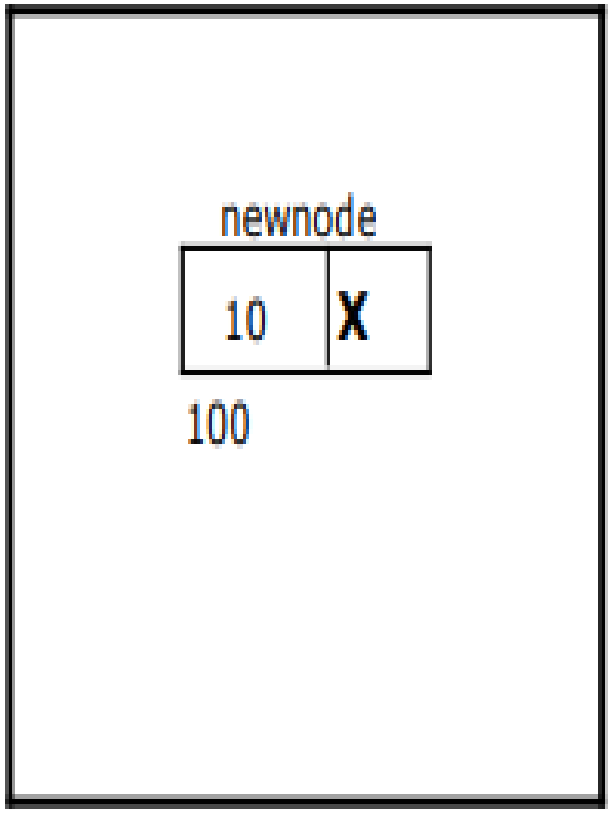
- Creating a structure with one data item and a next pointer, which will be pointing to next node of the list. This is called as self-referential structure.
- Initialise the start pointer to be NULL

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

typedef struct slinklist node;

node *start = NULL;
```



# Creating a node for Single Linked List

```
node* getnode()
{
      node* newnode;
      newnode = (node *) malloc(sizeof(node));
      printf("\n Enter data: "); scanf("%d",
      &newnode -> data);
      newnode -> next = NULL;
      return newnode;
}
```



# Traversing in singly linked list

**STEP 1:** SET PTR = HEAD
**STEP 2:** IF PTR = NULL
WRITE "EMPTY LIST"
GOTO STEP 7
END OF IF
**STEP 4:** REPEAT STEP 5 AND 6 UNTIL PTR != NULL
**STEP 5:** PRINT PTR→ DATA
**STEP 6:** PTR = PTR → NEXT
[END OF LOOP]
**STEP 7:** EXIT

# Insertion in singly linked list at beginning

•Allocate the space for the new node and store data into the data part of the node.

**ptr = (struct node *) malloc(sizeof(struct node *));**

**ptr → data = item**

•Make the link part of the new node pointing to the existing first node of the list.

**ptr->next = head;**

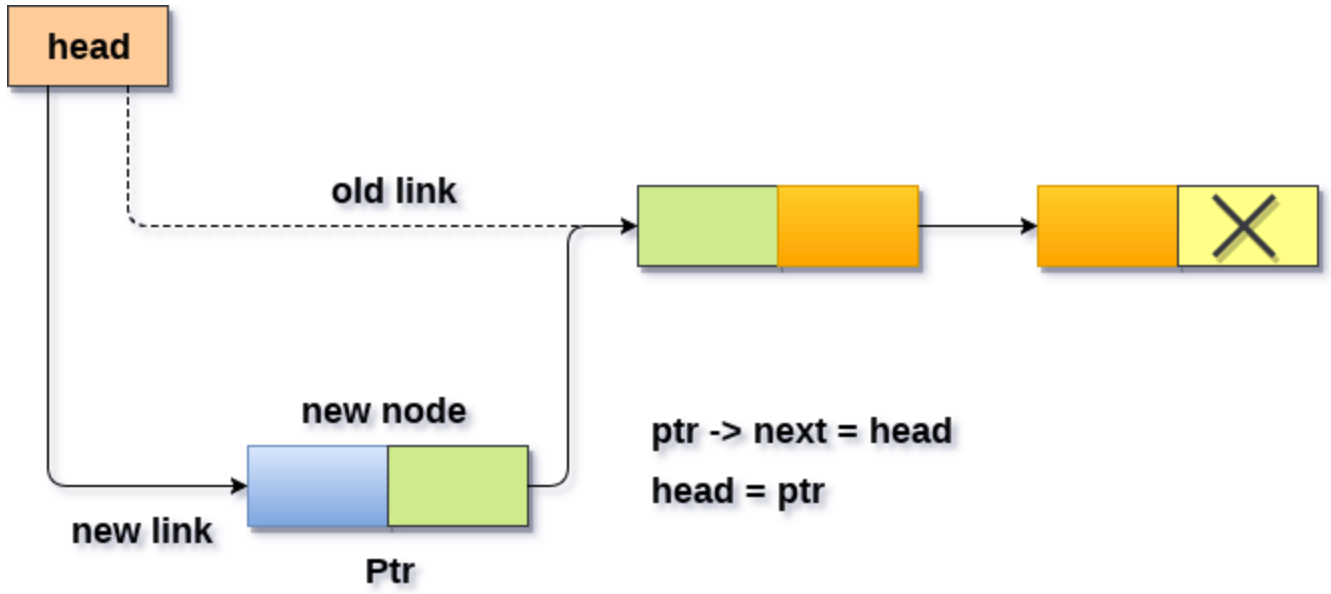
•At the last, we need to make the new node as the first node of the list

**head = ptr;**

# Algorithm for Insertion in singly linked list at beginning

**Step 1:** IF PTR = NULL
Write OVERFLOW
Go to Step 7
[END OF IF]
**Step 2:** SET NEW_NODE = PTR
**Step 3:** SET PTR = PTR → NEXT
**Step 4:** SET NEW_NODE → DATA = VAL
**Step 5:** SET NEW_NODE → NEXT = HEAD
**Step 6:** SET HEAD = NEW_NODE
**Step 7:** EXIT

# Algorithm for Insertion in singly linked list at beginning




**Reference:** https://www.tpointtech.com/insertion-in-singly-linked-list-at-beginning

# Insertion in singly linked list at the end

To insert a node at the last

- The node is being added to an empty list
- The node is being added to the end of the linked list

**First case,**

```
ptr->data = item;
        ptr -> next = NULL;

Head = ptr
```

# Insertion in singly linked list at the end

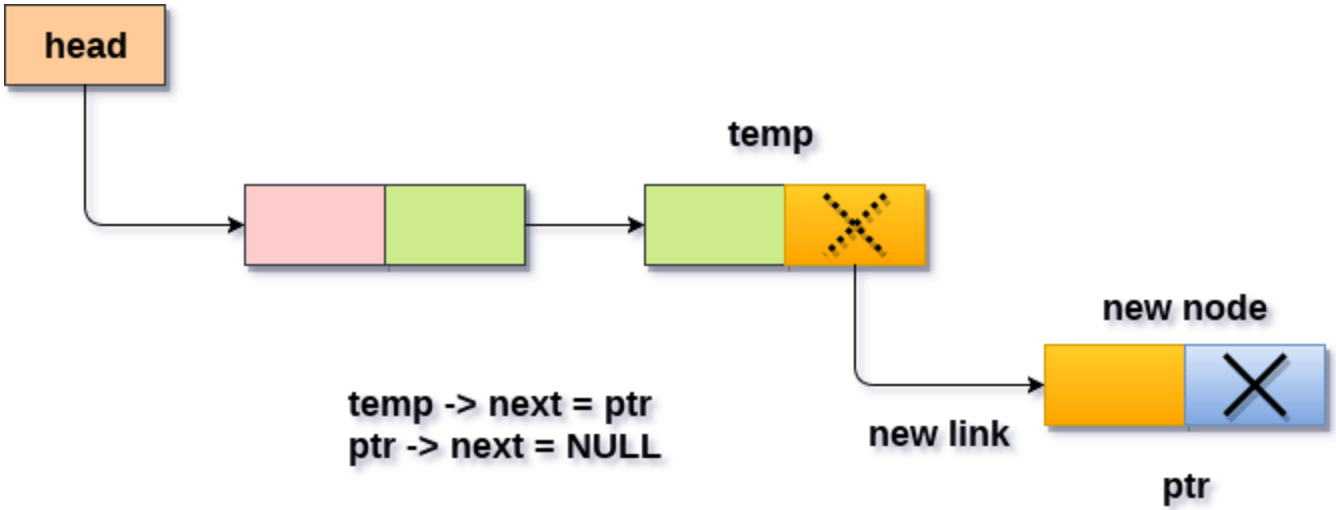
**2nd case,**

```
temp = head;
      while (temp -> next != NULL)
      {
         temp = temp -> next;
      }
      temp->next = ptr;
      ptr->next = NULL;
```

# Algorithm for Insertion in singly linked list at end

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

# Algorithm for Insertion in singly linked list at end




Inserting node at the last into a non-empty list

# Algorithm for Insertion in singly linked list at specific position

- **STEP 1:** IF PTR = NULL

WRITE OVERFLOW

GOTO STEP 12

END OF IF

- **STEP 2:** SET NEW_NODE = PTR
- **STEP 3:** NEW_NODE → DATA = VAL
- **STEP 4:** SET TEMP = HEAD
- **STEP 5:** SET I = 0
- **STEP 6:** REPEAT STEP 5 AND 6 UNTIL I
- **STEP 7:** TEMP = TEMP → NEXT
- **STEP 8:** IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"

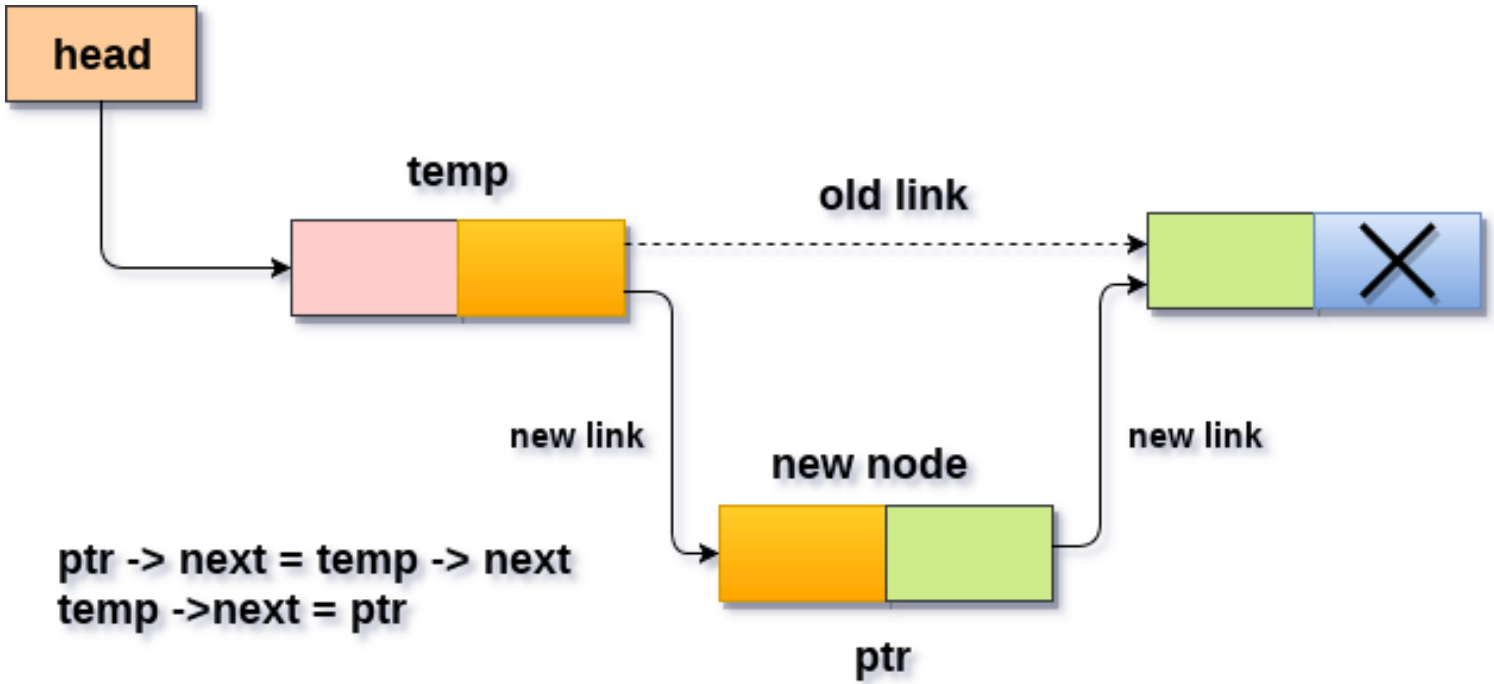
GOTO STEP 12

END OF IF

END OF LOOP

- **STEP 9:** PTR → NEXT = TEMP → NEXT
- **STEP 10:** TEMP → NEXT = PTR
- **STEP 11:** SET PTR = NEW_NODE
- **STEP 12:** EXIT

# Algorithm for Insertion in singly linked list at the specific position



# Algorithm for Deletion in singly linked list at beginning

- **Step 1:** IF HEAD = NULL

Write UNDERFLOW

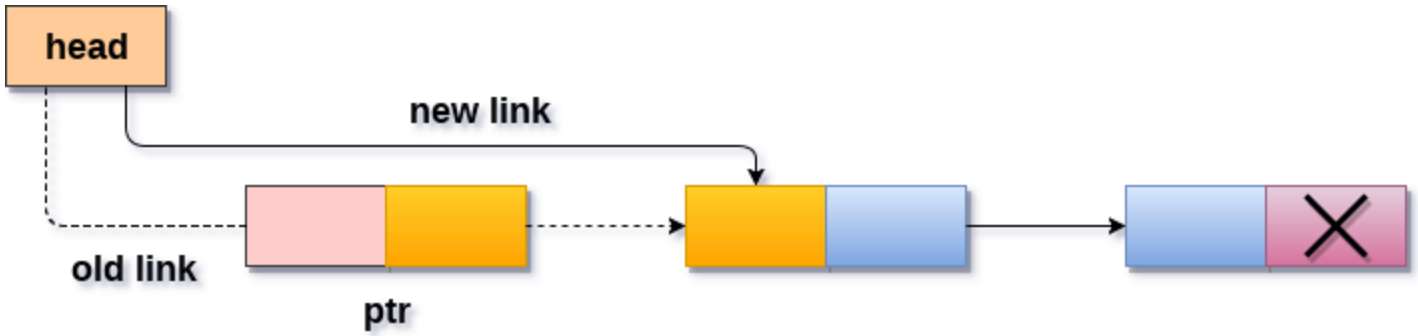
Go to Step 5

[END OF IF]

ELSE

- **Step 2:** SET PTR = HEAD
- **Step 3:** SET HEAD = HEAD -> NEXT
- **Step 4:** FREE PTR
- **Step 5:** EXIT

# Algorithm for Deletion in singly linked list at beginning



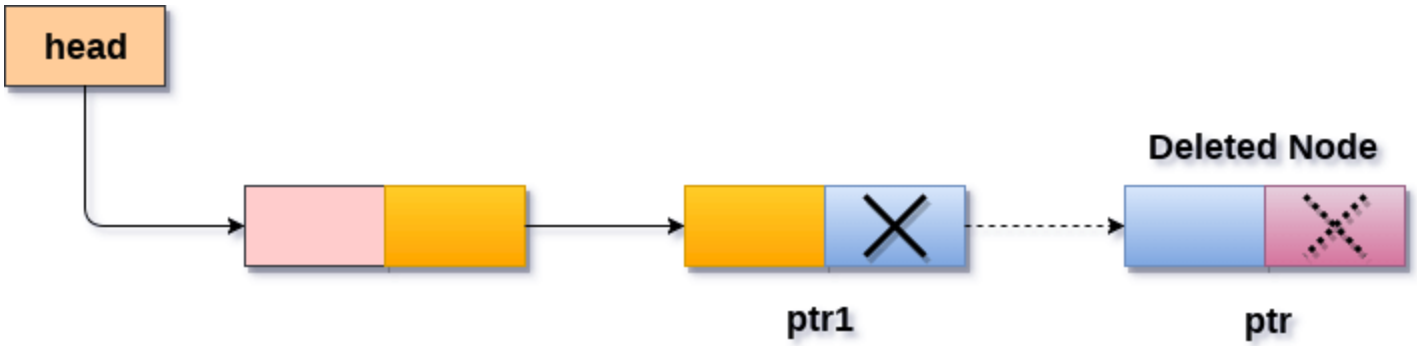

```
ptr = head
head = ptr -> next
free(ptr)
```

Deleting a node from the beginning

# Algorithm for Deletion in singly linked list at end

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

# Algorithm for Deletion in singly linked list at end




Deleting a node from the last

# Algorithm for Deletion in singly linked list after specific node

- **STEP 1:** IF HEAD = NULL

WRITE UNDERFLOW

GOTO STEP 10

END OF IF

- **STEP 2:** SET TEMP = HEAD
- **STEP 3:** SET I = 0
- **STEP 4:** REPEAT STEP 5 TO 8 UNTIL I
- **STEP 5:** TEMP1 = TEMP
- **STEP 6:** TEMP = TEMP → NEXT
- **STEP 7:** IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"

GOTO STEP 12

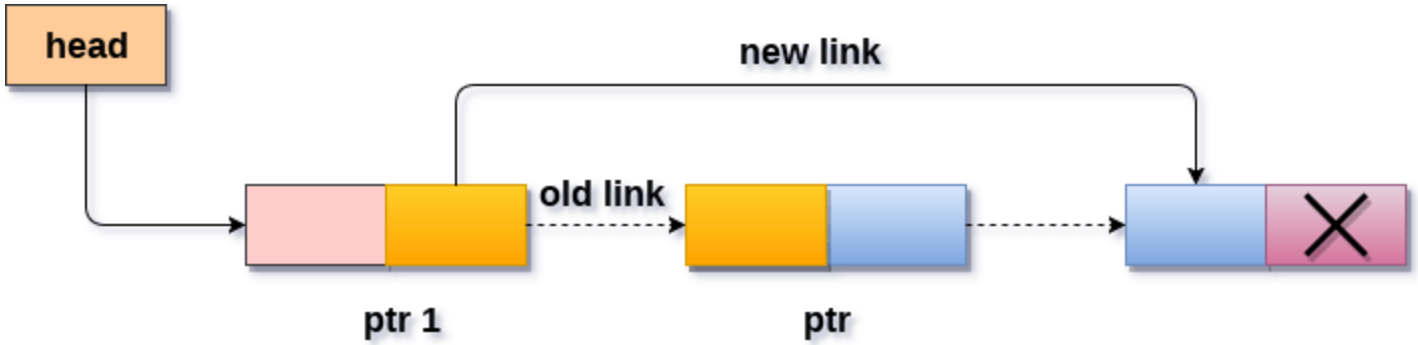
END OF IF

- **STEP 8:** I = I+1

END OF LOOP

- **STEP 9:** TEMP1 → NEXT = TEMP → NEXT
- **STEP 10:** FREE TEMP
- **STEP 11:** EXIT

# Algorithm for Deletion in singly linked list after specific node




ptr1 -> next = ptr -> next
free(ptr)

**Deletion a node from specified position**

# Algorithm for Searching in singly linked lis

- **Step 1:** SET PTR = HEAD
- **Step 2:** Set I = 0
- **STEP 3:** IF PTR = NULL
  WRITE "EMPTY LIST"
  GOTO STEP 8
  END OF IF
- **STEP 4:** REPEAT STEP 5 TO 7 UNTIL PTR != NULL
- **STEP 5:** if ptr → data = item
  write i+1
  End of IF
- **STEP 6:** I = I + 1
- **STEP 7:** PTR = PTR → NEXT

[END OF LOOP]

- **STEP 8:** EXIT

# Test Your self

**1. As we have the memory address of Nodes can we traverse backwards in a linked list ?**

a: Yes, we can.

b: No, we can't.

**2. What are the advantages of a linked list?**

a: Dynamic Memory Allocation

b: They require less memory than an array to store the same data.

c: We can easily traverse back to previous elements.

d: None of the above.

**3. What is the time complexity for insertion of an element into linked list?**

a: O(1)

b: O(log n)

c: O(n)

d: O(n^2)

# Quiz Answers

1. **Answer: b**
2. **Answer :a**
3. **Answer : c**

https://questions.examside.com/past-years/gate/question/pconsider-the-problem-of-reversing-a-singly-linked-list-t-gate-cse-theory-of-computation-finite-automata-and-regular-language-zpfpkf4re1g8xuff

# Review

**Definition:**

Singly linked list: linear structure with nodes containing data and pointers.

**Components:**

Nodes: data and next pointer.

Head Pointer: points to first node.

Tail Pointer: optional, points to last node.

**Representation:**

Nodes linked via pointers, dynamic memory allocation.

Access via head pointer, traversal for manipulation.

# Review

**Advantages:**

Efficient insertion/deletion.

Dynamic size, flexibility.

**Disadvantages:**

No direct access, traversal needed.

Higher memory overhead.

Inefficient reverse traversal.


THANK
YOU