Unit - 2

Lists

D D M M Y Y Y Y

Que- Explain the Basic Insertion operation in Singly Linked List with suitable C program example.

⇒

- In a singly linked list, the basic insertion operation involves adding a new node to the list.
- There are typically three main cases for insertion:-
  1. Insertion at the beginning of the list
  2. Insertion at the end of the list
  3. Insertion at a specific position in the list.
- A singly linked List is a data structure consisting of a sequence of elements, where each element is represented by a node.
- Each node in a singly linked list contains two fields:-
  1. Data :- This field stores the actual value or data of the node.
  2. Next :- This field is a reference or pointer to the next node in the sequence.

Program:-

```
#include <stdio.h>
#include <stdlib.h>
 struct Node
 {
    int data;
    struct Node *next;
 };
int main()
{
```

```
//creating nodes
struct Node* head = NULL;
struct Node* second = NULL;
struct Node* third = NULL;

head = (struct Node*) malloc (sizeof (struct Node));
second = (struct Node*) malloc (sizeof (struct Node));
third = (struct Node*) malloc (sizeof (struct Node));

//Assigning values and linking nodes
head -> data = 1;
head -> next = second;

second -> data = 2;
second -> next = third;

third -> data = 3;
third -> next = NULL; // Indicates the end of the List

//Traversing and printing the Linked List
struct Node* current = head;
while (current != NULL)
{
printf("%d -> ", current -> data);
current = current -> next;
}
printf("NULL\n");
return 0;
}
```

que- Describe the following operations of doubly linked List, delete at Front, delete at Any, Delete at End, search.

⇒

-In a doubly linked list, you can perform various operations, including delete at the front, delete at any position, delete at the end and search for a specific element.

1] Delete at Front:-

-To delete a node at the front of a doubly linked List, you need to follow these steps:-

- update the 'next' pointer of the new head node to point to 'NULL' (indicating the new head).
- update the 'Prev' pointer of the previous head node to 'NULL'.
- Free the memory of the old head node.

2] Delete at Any Position:-

-To delete a node at any specific position in a doubly linked list, you need to:-

- Traverse to the node at the desired position.
- update the 'next' pointer of the node before the target node to point to the node after the target node.
- Update the 'Prev' pointer of the node after the target node to point to the node before the target node.
- Free the memory of the target node.

3] Delete at End :-
- To delete a node at the end of a doubly linked list, follow these steps:-
- Traverse to the last node
- update the 'next' pointer of the node before the last node to point to "NULL"
- free the memory of the last node.

4] Search :-
- Searching for a specific element in a doubly linked list involves traversing the list until you find the desired value.
- You can traverse the list from either end (front or back) depending on your specific needs.
- When you find the element you're looking for, you can return it's node or an indication that it was found.

Program:-

```
Struct Node
{
    int data;
    struct Node * next;
    Struct Node * prev;
};
// Delete at Front
void deleteAtFront (struct Node ** head)
{
    if (*head == NULL)
```

```
    {
        // List is empty
        return;
    }
    struct Node* temp = *head;
    *head = (*head) -> next;
    (*head) -> Prev = NULL;
    free (temp);
}

// Delete at Any Position
void deleteAtPosition (struct Node** head, int Position)
{
    // Implement this based on position
}

// Delete at End
void deleteAtEnd (struct Node** head)
{
    // Implement this to delete the last node
}

// Search
struct Node* search (struct Node* head, int key)
{
    while (head != NULL)
    {
        if (head -> data == key)
        {
            return head;
        }
        head = head -> next;
    }
```

```
    return NULL; // Element not found
}
```

Que- Write a C program for following operations of Doubly Linked List, attach a node in the beginning of the linked list, Detach the last node of linked list.

⇒

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* next;
    struct Node* prev;
};
// Function to create a new node
struct Node* createNode(int data)
{
struct Node* newNode = (struct Node*) malloc
                        (sizeof(struct Node));
newNode -> data = data;
newNode -> next = NULL;
newNode -> prev = NULL;
 return newNode;
}
// Function to attach a node at the beginning of
 the linked List.
Void attachAtBeginning (struct Node** head, int data)
{
struct Node* newNode = createNode(data);
```

```
if (*head == NULL)
{
  *head = newNode;
}
else
{
  newNode->next = *head;
  (*head)->Prev = newNode;
  *head = newNode;
}
}
// Function to detach the last node From the
   linked List
Void detachLastNode (struct Node**head)
{
  if (*head == NULL)
  {
  Printf("The List is empty. cannot detach Last
        node.\n");
  return;
}
if ((*head)->next == NULL)
{
  // There is only one node in the list
  free (*head);
  *head = NULL;
}
else
{
  Struct Node* current = *head;
  While (current->next != NULL)
```

```
{
  current = current ->next;
}
current -> prev ->next = NULL;
  free (current);
}

// Function to print the doubly linked list
void PrintList (struct Node* head)
{
  struct Node* current = head;
  while (current != NULL)
  {
    printf("%d <-> ", current -> data);
    current = current ->next;
  }
  Printf("NULL\n");
}
int main()
{
  struct Node* head = NULL;
  attachAtBeginning (&head, 3);
  attachAtBeginning (&head, 2);
  attachAtBeginning (&head, 1);
Printf("Linked List after attaching nodes at
    the beginning :\n");
PrintList(head);
detachLastNode (&head);
detachLast Node (& head);
```

```
printf("Linked List after detaching the last
        nodes:\n");
printList(head);
return 0;
}
```

Imp Que- Explain the terms List, stack, queue, deque.

⇒

1) List :-

- A list is a common data structure that stores a collection of element.
- It can be implemented using arrays, linked lists, or other data structures.
- Lists may allow duplicate elements and usually maintain the order in which elements are added.
- Lists provide flexibility for various operations like adding, deleting, or modifying elements at different positions.

2) Stack :-

- A stack is a linear data structure that follows the last-In-First-out (LIFO) principal.
- It has two primary operations: "Push", which adds an element to the top of the stack, and "Pop", which removes the top element.
- Stacks are often used for tasks that involve tracking the most recent elements such as function call management, expression

evaluation and undo functionality.

3] Queue :-

- A queue is a linear data structure that follows the First-in-First-out (FIFO) principle.
- It has two main operations: "enqueue" (or "push"), which adds an element to the rear of the queue, and "dequeue", which removes an element from the front.
- Queues are used in situations where the order of processing matters, like task scheduling, print job management and breadth-first search algorithms.

4] Deque (Double-Ended Queue) :-

- A deque is a generalized form of a queue that allows elements to be added or removed from both ends.
- It can be considered a hybrid of stacks and queues.
- Deques offer more flexibility than stacks and queues and can be used in various scenarios, including implementing both stack and queue operations in a single data structure.

Que- Elaborate the operations of circular singly Linked insert, Delete search of a value at any given Position.

⇒

- circular singly linked lists are a type of linked list where the last node points back to the first node, creating a loop.

1) Insertion at a Given Position :-

- To insert a node with a specific value at a given position in a circular singly linked list:
  - Create a new node with the desired value.
  - Traverse the list to the node just before the target Position.
  - update the pointers to insert the new node:
  - set the next pointer of the new node to the node at the target position.
  - Update the next pointer of the node at Position -1 to point to the new node.

2) Deletion at a Given position :-

- To delete a node at a specific Position in a circular singly linked list :-
  - Traverse the list to the node just before the target Position.
  - update the next pointer of the node at Position -1 to skip the node at the target position and point directly to the node after the target position.

3) Search for a value at a Given position:-
- To search for a specific value at a given position in a circular singly linked list:-
- Start at the head node and traverse the list node by node.
- Keep track of the position while comparing each node's value with the target value.
- If the target value is found at the desired position, return the position or the node containing the value.

Que- Explain Types of Linked List.

⇒

1) singly Linked List:-
- Each node has data and a reference (or a link) to the next node.
- Traversal is possible in only one direction, from the head (First node) to the tail (Last node).
- It is relatively simple and memory-efficient.

2) Doubly Linked List:-
- Each node has data and references to both the next and the previous node.
- Allows for easy traversal in both directions.
- Typically uses more memory than a singly linked list due to the extra references.

3) circular Linked List:-
- A variation of singly or doubly linked lists

where the last node's next pointer points back to the first node.
- Used in applications where you need to continuously cycle through the list.

i) Singly circular Linked List:-
- In this type of circular linked list, each node has a reference only to the next node.
- The last node's reference points back to the first node, creating a continuous loop.
- Singly circular linked lists are often used when you need to traverse a list continuously or when you want a more memory-efficient representation of a circular structure.

ii) Doubly circular Linked List:-
- In a doubly circular linked list, each node has references to both the next and the previous node.
- Like the singly circular linked list, it forms a loop, with the last node's reference pointing back to the first node.
- Doubly circular linked lists allow for easy traversal in both directions, which can be useful in certain applications.

IMP

Que- Construct algorithm for following operations on a circular Linked List.

i) Create at start
ii) Delete at End
iii) Traverse and display the elements of data.

⟹

Algorithm :-

1] Create at start :-

a] create a new node with the given data.
b] If circular Linked List is empty :
  i] set the new node's next reference to itself (circular)
  ii] Update circular Linked List to Point to the new node.
c] If circular Linked List is not empty :-
  i] Find the last node by starting at circular Linked List and following the next reference until you reach the last node.
  ii] set the new node's next reference to circular Linked List (the first node).
  iii] update the last node's next reference to the new node.
  iv] Update circular Linked List to Point to the new node.

2] Delete at End :-

a] If circular Linked List is empty, return an error (List is already empty).
b] If circular Linked List has only one node :
  i] Set circular Linked List to None (List is

now empty).
c) IF circular Linked List has more than one node:-
i) Find the node before the last node by starting at circular Linked List and following the next references until you reach the node whose next node is the last node.
ii) Update the last node's next reference to None.
iii) Update the previous node's next reference to circular Linked List (connecting the last node's predecessor to the new last node).

3) Traverse and Display the Elements:-
a) IF circular Linked List is empty, return "circular Linked List is empty.".
b) Initialize a current node to circular Linked List.
c) Loop until you complete a full traversal of the circular List (back to the starting point):-
i) Display the data of the current node.
ii) Move to the next node by updating current to the next node.
iii) stop when you reach circular Linked List again.

Imp

Que- List out the various Applications of the following ADTs.

1] singly Linked List
2] Doubly Linked List
3] circular singly Linked List
4] circular Doubly List

⇒ Applications :-

1] singly Linked List :-
- Dynamic memory allocation for data storage
- Implementing a stack or a queue.
- used in many algorithms like traversal and manipulation of data.

2] Doubly Linked List :-
- Implementing data structures like a double-ended queue (deque).
- Efficient backward and forward traversal of data.
- Undo/redo functionality in applications.

3] circular singly Linked List :-
- Implementing a round-robin scheduling algorithm.
- creating circular buffers for data storage and processing.
- Representing a ring or loop structure in certain applications.

4) circular Doubly Linked List :-
- Efficient implementation of certain data structures like a circular double-ended queue.
- Applications where both forward and backward traversal in loop are required.
- circular navigation in software interfaces, e.g. image galleries.

imp Que- What is Doubly Linked List? List various operations of Doubly Linked List and explain any one operation.

⇒

- A Doubly Linked List is a data structure that consists of nodes, where each node contains data and two pointers, one pointing to the previous node and one pointing to the next node.
- This bidirectional linking allows for easy traversal in both directions, which is a key feature distinguishing it from a singly linked List.

operations :-

1) Insertion :-
- Insertion at beginning :- Add a new node at the front of the list.
- Insertion at the end :- Add a new node at the end of the list.
- Insertion at a specific Position :- Insert a new

node at a given position in the list.

2] Deletion :-
- Deletion at the beginning :- Remove the first node.
- Deletion at the end :- Remove the last node.
- Deletion of a specific node :- Remove a node from any Position in the List.

3] Traversal :-
- Traverse forward :- visit each node in the list from the beginning to the end.
- Traverse backward :- visit each node in the list from the end to the beginning.

4] Search :-
- search for a specific element in the list.

- Insertion at the Beginning :-

steps :-
1] Create a new node with the data you want to insert.
2] set the 'next' Pointer of the new node to Point to the current head of the List.
3] set the 'previous' pointer of the current head to point to the new node (making it the new node's next node).
4] Update the head pointer to Point to the new node.

Que- Explain Deletion in doubly Linked List in specified Node with Pseudo code and example.

⇒

Deletion at specified Node in doubly Linked List:-

Algorithm :-

Input :- A doubly Linked list, and the node to be deleted.

Output :- The doubly linked list with the specified node removed.

1) Check if the input node to delete is null. If it's null, return, as there's nothing to delete.
2) If the node to delete has a previous node, update the 'next' pointer of the previous node to point to the 'next' node of the node to be deleted.
3) If the node to delete has a next node, update the 'prev' pointer of the next node to point to the 'prev' node of the node to be deleted.
4) If the node to delete is the head of the list, update the head pointer to point to the 'next' node.
5) If the node to delete is the tail of the list, update the tail pointer to point to the 'prev' node.
6) optionally, deallocate memory for the node to be deleted if it's no longer needed.

Example:-
Suppose we want to delete the node with the value 15.

1) check if the input node to delete (15) is not null.
2) update the 'next' pointer of the previous node (10) to point to the 'next' node (20) of the node to be deleted (15).
3) update the 'prev' pointer of the next node (20) to point to the 'prev' node (10) of the node to be deleted (15).
4) Since we are deleting the node with value 15, we don't need to update the head pointer in this case.
5) update the tail pointer to point to the 'prev' node (10) of the node to be deleted (15) since the 'tail' is also getting updated.
6) optionally, deallocate memory for the node with value 15.

Que- Singly circular Linked List with insertion and deletion with Psedou code and example.

→ -A singly circular linked list is a data structure in which each element, called a node, is connected to the next node through a "next" Pointer, and the last node is connected back to the first node, Forming a loop.

Node structure :-
Each node in the circular linked list contain two components :-

1) Data :- the actual data or value stored in the node.
2) Next pointer :- A reference to the next node in the list.

Insertion Algorithm :-

1) Create a new node with the data you want to insert.
2) If the list is empty, set the new node as the head, and make it point to itself (new_node.next = new_node).
3) If the list is not empty, follow these steps :-
a) Make the new node's next pointer point to the current head node (new_node.next = head).
b) Update the current tail node's next pointer to point to the new node (tail.next = new_node).
c) set the new node as the new tail (tail = new_node).

Deletion Algorithm:-

1] To delete a specific node with data x:-

a] start at the head and traverse the list until you find a node whose next node has data x (i.e. node.next.data == x).

b] update the current node's next pointer to skip the node to be deleted.
(node.next = node.next.next)

c] IF the node to be deleted is the head or tail, update the head and tail accordingly.

d] IF the list becomes empty after deletion, set both head and tail to null.

Example:-

create a singly circular linked list with values [1,2,3] and then perform some insertions and deletions.

1] Initial List:- 1 → 2 → 3 → 1 (the last node points back to the first).

2] Insertion:-

- Insert 4 at the beginning:-
  List:- 4 → 1 → 2 → 3 → 4
- Insert 5 at the end:-
  List:- 4 → 1 → 2 → 3 → 4 → 5

3] Deletion:-

- Deletion node with data 2:-
  List:- 4 → 1 → 3 → 4 → 5
- Deletion node with data 4:-
  List:- 1 → 3 → 4 → 5.

Que- Explain Doubly circular linked List with insertion and deletion with algorithm and example.

⇒ - A doubly circular linked list is similar to a singly circular linked list, with the main difference being that each node has both a "next" and a "previous" pointer, allowing for traversal in both directions.

Node Structure :-

Each node in the doubly circular linked list contains three components :-

1] Data :- The actual data or value stored in the node.

2] Next Pointer :- A reference to the next node in the list.

3] Previous Pointer :- A reference to the previous node in the list.

Insertion Algorithm :-

1] Create a new node with the data you want to insert.

2] If the list is empty, set the new node as the head and tail, and make it point to itself in both the next and previous directions.

3] If the list is not empty, follow these steps :-

a] Make the new node's next pointer point to the current head node (new_node.next = head).

b] Make the new node's previous pointer Point

to the current tail node
(new_node.prev = tail)
c] update the current head node's previous pointer to point to the new node
(head.prev = new_node)
d] update the current tail node's next pointer to point to the new node (tail.next = new_node)
e] set the new node as the new tail
(tail = new_node).

Deletion Algorithm:-

1] To delete a specific node with data x:-
a] start at the head and traverse the list until you find a node whose data matches x.
b] update the next pointer of the previous node to skip the node to be deleted.
(prev_node.next = node.next)
c] update the previous pointer of the next node to skip the node to be deleted.
(next_node.prev = node.prev)
d] If the node to be deleted is the head or tail, update the head and tail accordingly.
e] If the list becomes empty after deletion, set both head and tail to null.

Example :-
create a doubly circular linked list with values [1,2,3]

1] Initial List :- '1 <-> 2 <-> 3 <-> 1' (both the last and first nodes point to each other).

2] Insertion :-
- Insert 4 at the beginning :-
- List :- 4 <-> 1 <-> 2 <-> 3 <-> 4

- Insert 5 at the end :-
List :- '4 <-> 1 <-> 2 <-> 3 <-> 4 <-> 5'

3] Deletion :-
- Delete node with data 2 :-
List :- 4 <-> 1 <-> 3 <-> 4 <-> 5'

- Delete node with data 4 :-
List :- '1 <-> 3 <-> 5'

Que- Write Pseudo code for insertion and deletion in singly Linked List with diagram.

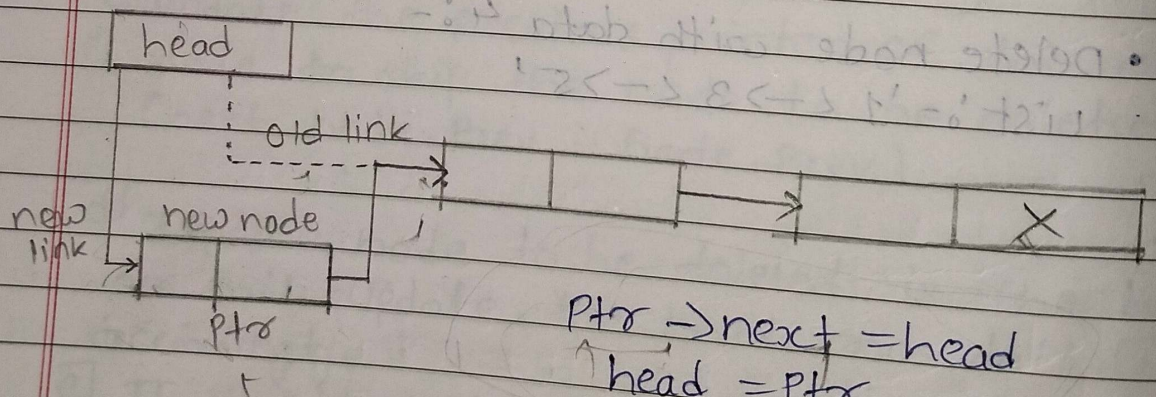
⟹

- Insertion at beginning :-

Algorithm :-

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

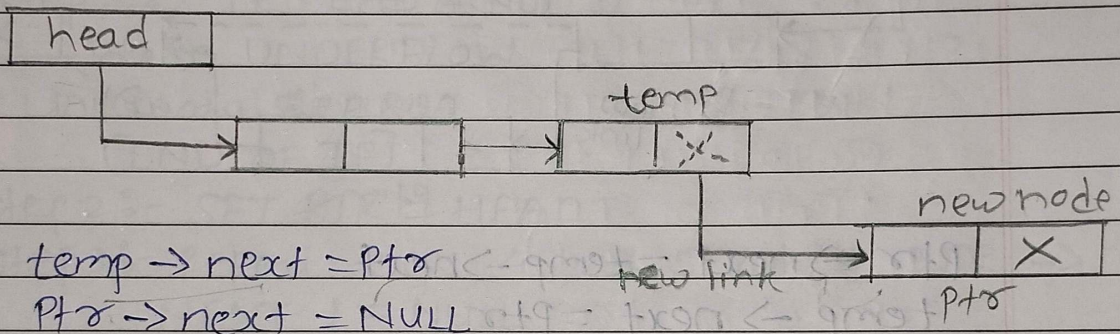


Ptr → next = head
head = ptr

- Insertion at end / Last :-

Algorithm :-
step 1 :- IF PTR = NULL write OVERFLOW
Go to step 1
[END OF IF]

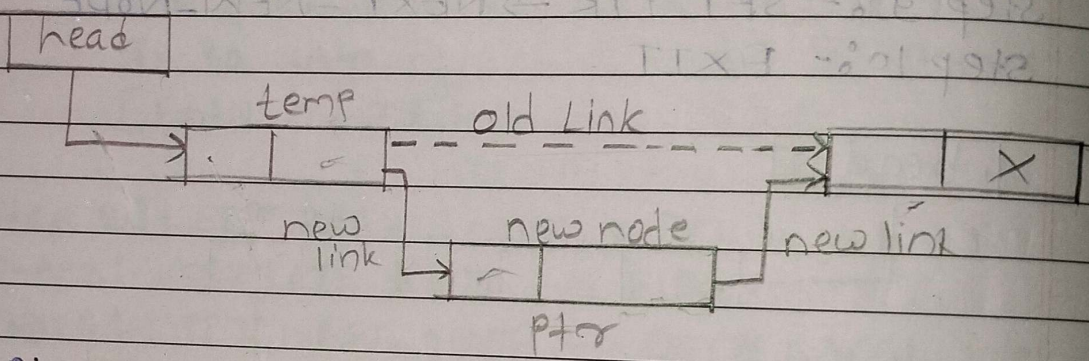
Step 2:- SET NEW_NODE = PTR
Step 3:- SET PTR = PTR -> NEXT
Step 4:- SET NEW_NODE -> DATA = VAL
Step 5:- SET NEW_NODE -> NEXT = NULL
Step 6:- SET PTR = HEAD
Step 7:- Repeat Step 8 while PTR -> NEXT != NULL
Step 8:- SET PTR = PTR -> NEXT
[END OF LOOP]
Step 9:- SET PTR -> NEXT = NEW_NODE
Step 10:- EXIT



- Insertion at specific Node:-

Algorithm:-
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

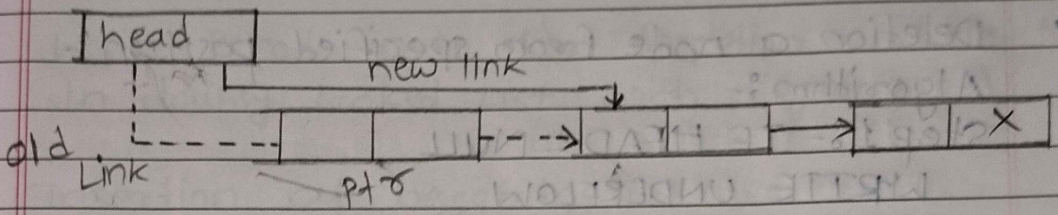
WRITE "DESIRED NODE NOT PRESENT"
GOTO STEP 12
END OF IF
END OF LOOP
step 9 :- PTR -> NEXT = TEMP -> NEXT.
step 10 :- TEMP -> NEXT = PTR
step 11 :- SET PTR = NEW_NODE
step 12 :- EXIT



Ptr -> next = temp -> next
temp -> next = ptr

- Deletion at beginning :-

Algorithm :-
Step 1 :- IF HEAD = NULL
write UNDERFLOW
Go to step 5
[END OF IF]
Step 2 :- SET PTR = HEAD
Step 3 :- SET HEAD = HEAD -> NEXT
Step 4 :- FREE PTR
Step 5 :- EXIT



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

- Deletion at end / Last :-

Algorithm :-

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

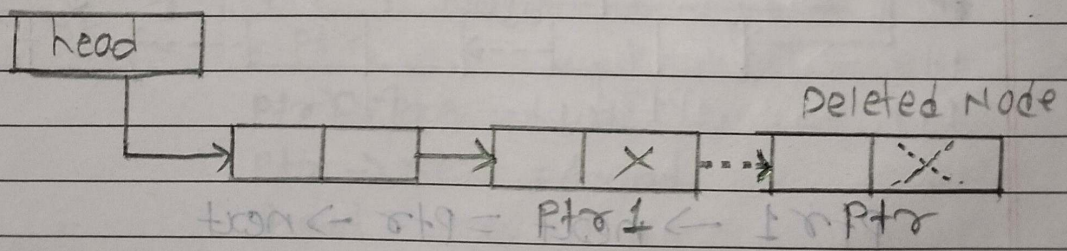
step 4 :- SET PREPTR := PTR

step 5 :- SET PTR = PTR -> NEXT
[END OF LOOP]

step 6 :- SET PREPTR -> NEXT = NULL

step 7 :- FREE PTR

step 8 :- EXIT

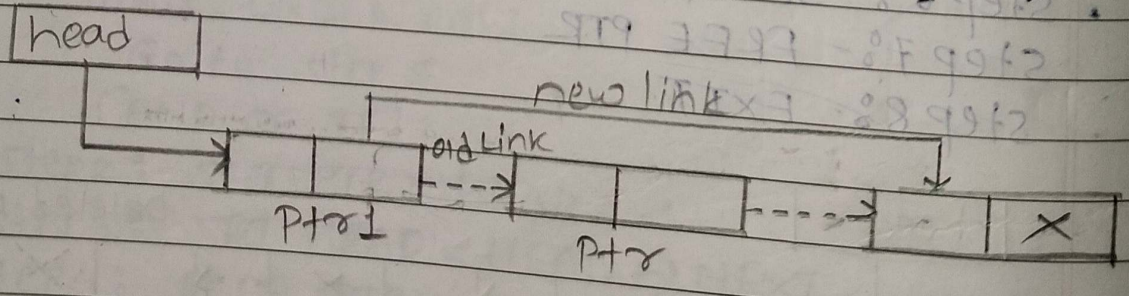


Ptr1 -> next = NULL
free (Ptr)

- Deletion a node from specified position :-

Algorithm :-

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 :- TEMP 1 = 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 :- TEMP 1 → NEXT = TEMP → NEXT
Step 10 :- FREE TEMP
Step 11 :- EXIT



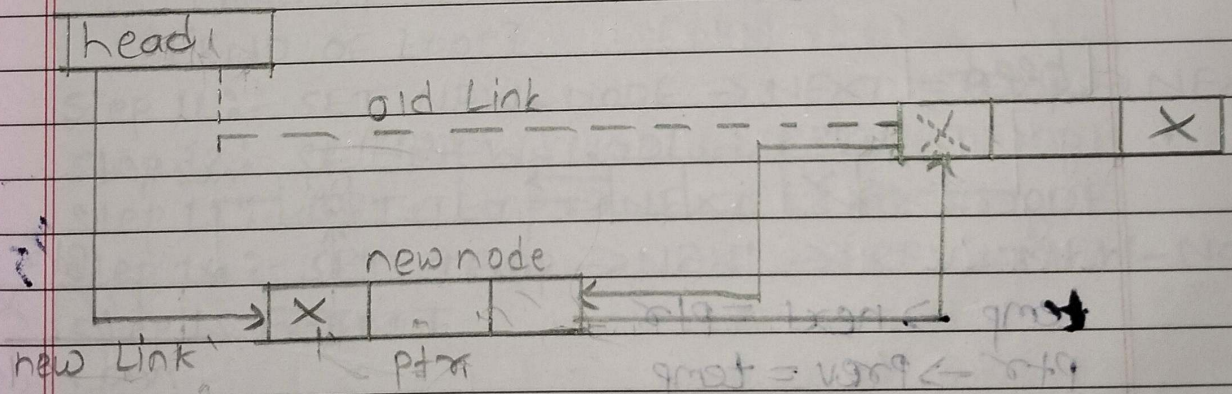
Ptr1 → next = Ptr → next
free (Ptr)

Imp Que- Write Pseodu code for insertion and deletion in doubly Linked List with diagram.

⇒

Insertion at beggining :-

Step : 1 :- IF Ptr = NULL
Write OVERFLOW
Go to step 9
[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 -> PREV = NULL
step 6 :- SET NEW_NODE -> NEXT = START
step 7 :- SET head -> PREV = NEW_NODE
step 8 :- SET head = NEW_NODE
step 9 :- EXIT



Ptr -> prev = NULL
ptr -> next = head
head -> Prev = Ptr
head = Ptr

- Insertion at END / Last :-

Algorithm :-

Step 1 :- IF PTR = NULL
Write OVERFLOW
Go to Step 11
[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 TEMP = START

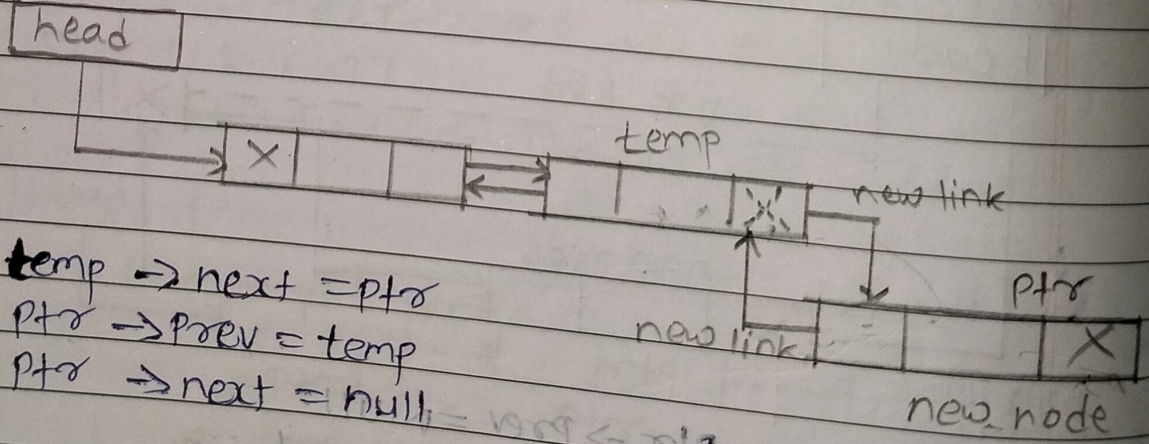
Step 7 :- Repeat Step 8 while TEMP -> NEXT != NULL

Step 8 :- SET TEMP = TEMP -> NEXT
[END OF LOOP]

Step 9 :- SET TEMP -> NEXT = NEW_NODE

Step 10 :- SET NEW_NODE -> PREV = TEMP

Step 11 :- EXIT

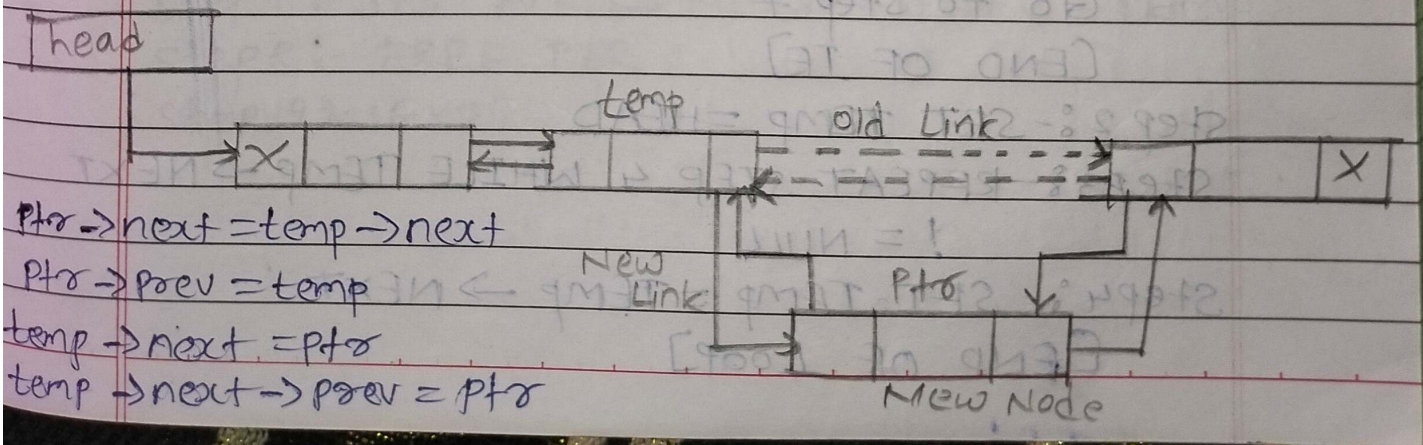


temp -> next = ptr
Ptr -> Prev = temp
Ptr -> next = null;

- Insertion at specific node:-

Algorithm:-

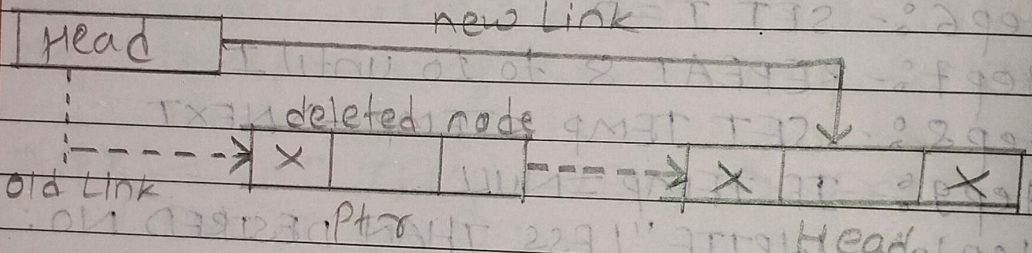
step 1:- IF PTR = NULL
    Write OVERFLOW
    Go to step 15
    [END OF IF]
step 2:- SET NEW_NODE = PTR
step 3:- SET PTR = PTR → NEXT
step 4:- SET NEW_NODE → DATA = VAL
step 5:- SET TEMP = START
step 6:- SET I = 0
step 7:- REPEAT 8 to 10 until I
step 8:- SET TEMP = TEMP → NEXT
step 9:- IF TEMP = NULL
step 10:- WRITE "LESS THAN DESIRED NO. OF ELEMENTS"
    GOTO STEP 15
    [END OF IF]
    [END OF LOOP]
Step 11:- SET NEW_NODE → NEXT = TEMP → NEXT
Step 12:- SET NEW_NODE → PREV = TEMP
Step 13:- SET TEMP → NEXT = NEW_NODE
Step 14:- SET TEMP → NEXT → PREV = NEW_NODE
Step 15:- EXIT



- Deletion at beggining:-

Algorithm:-

STEP 1:- IF HEAD = NULL
WRITE UNDERFLOW
GoTo step 6
Step 2:- SET PTR = HEAD
Step 3:- SET HEAD = HEAD → NEXT
Step 4:- SET HEAD → PREV = NULL
Step 5:- FREE PTR
Step 6:- EXIT

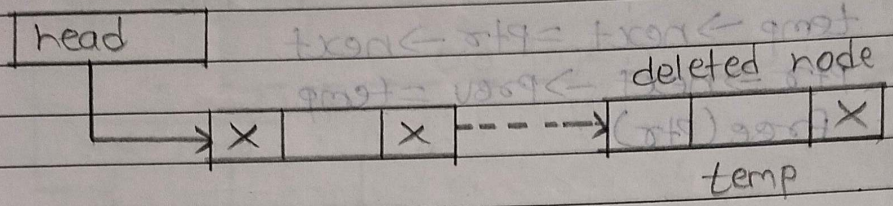


ptr = head
head = head → next
head → prev = NULL
free (Ptr)

- Deletion at End / Last:-

Algorithm:-
Step 1:- IF HEAD = NULL
Write UNDERFLOW
Go to step 7
[END OF IF]
Step 2:- SET TEMP = HEAD
Step 3:- REPEAT STEP 4 WHILE TEMP → NEXT
!= NULL
Step 4:- SET TEMP = TEMP → NEXT
[END OF LOOP]

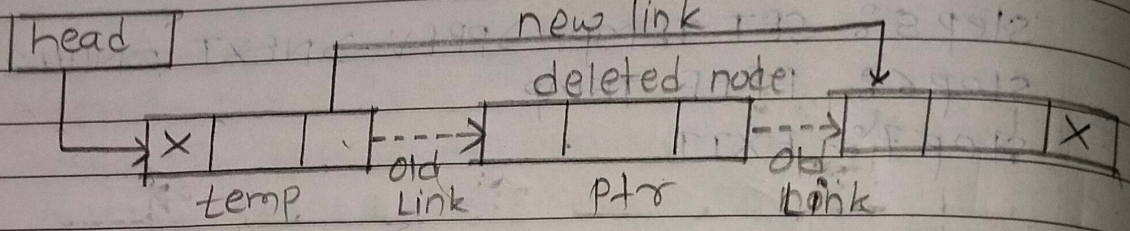
Step 5 :- SET TEMP -> PREV -> NEXT = NULL
Step 6 :- FREE TEMP
Step 7 :- EXIT



temp -> prev -> next = NULL
Free (temp)

- Deletion at specified node :-

Algorithm :-
Step 1 :- IF HEAD = NULL
Write UNDERFLOW
Go to step 9
[END OF IF]
Step 2 :- SET TEMP = HEAD
Step 3 :- Repeat step 4 while TEMP -> DATA != ITEM
Step 4 :- SET TEMP = TEMP -> NEXT
[END OF LOOP]
Step 5 :- SET PTR = TEMP -> NEXT
Step 6 :- SET TEMP -> NEXT = PTR -> NEXT
Step 7 :- SET PTR -> NEXT -> PREV = TEMP
Step 8 :- FREE PTR
Step 9 :- EXIT



temp -> next = ptr -> next
ptr -> next -> prev = temp
free (ptr)

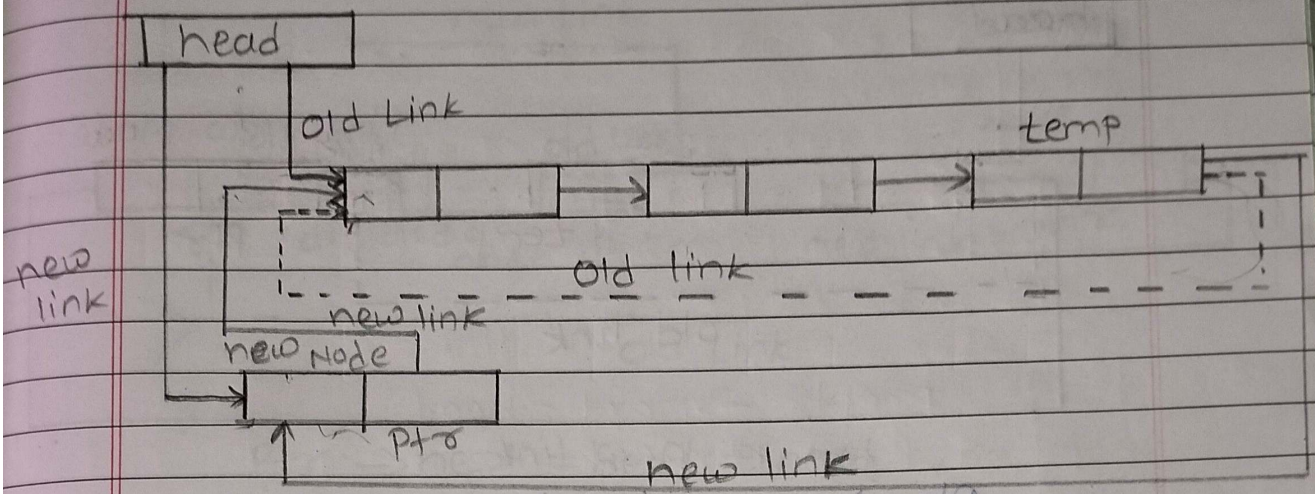
Que- Write Psedou code for insertion and deletion in circular singly Linked List with diagram.

⇒

- ~~Insert~~ Insertion at beggining :-

Algorithm :-

step1 :- IF PTR = NULL
Write OVERFLOW
Go to step 11
[END OF IF]
step 2 :- SET NEW_NODE = PTR
step3 :- SET PTR = PTR -> NEXT
step4 :- SET NEW_NODE -> DATA = VAL
step5 :- SET TEMP = HEAD
step6 :- Repeat step 8 while TEMP -> NEXT != HEAD
step7 :- SET TEMP = TEMP -> NEXT
[END OF LOOP]
step8 :- SET NEW_NODE -> NEXT = HEAD
step9 :- SET TEMP -> NEXT = NEW_NODE
step10 :- SET HEAD = NEW_NODE
step11 :- EXIT

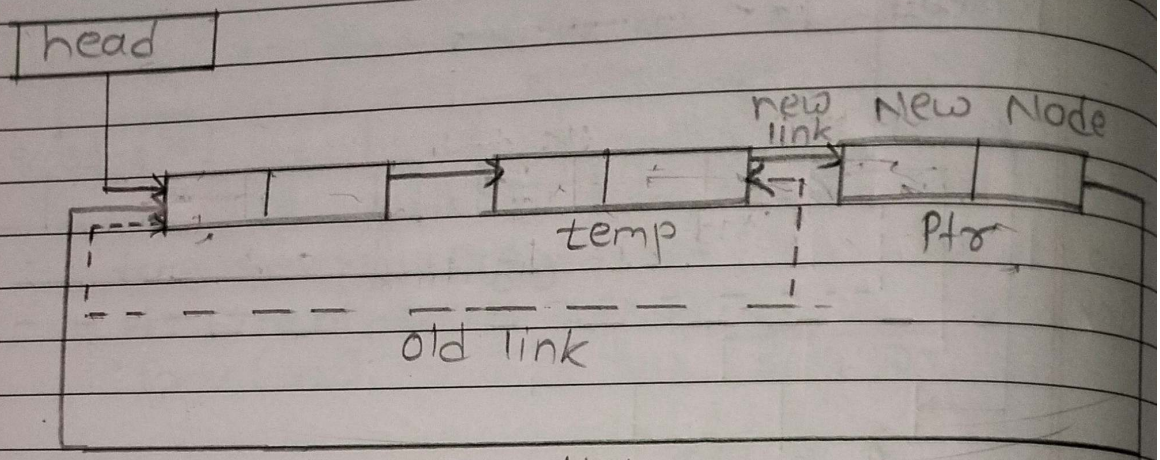



```
temp -> next = ptr
ptr -> next = head
head = ptr
```

- Insertion at end / Last :-

Algorithm :-

```
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 = HEAD
Step 6 :- SET TEMP = HEAD
Step 7 :- Repeat step 8 while TEMP -> NEXT != HEAD
Step 8 :- SET TEMP = TEMP -> NEXT
          [END OF LOOP]
Step 9 :- SET TEMP -> NEXT = NEW_NODE
Step 10 :- EXIT
```

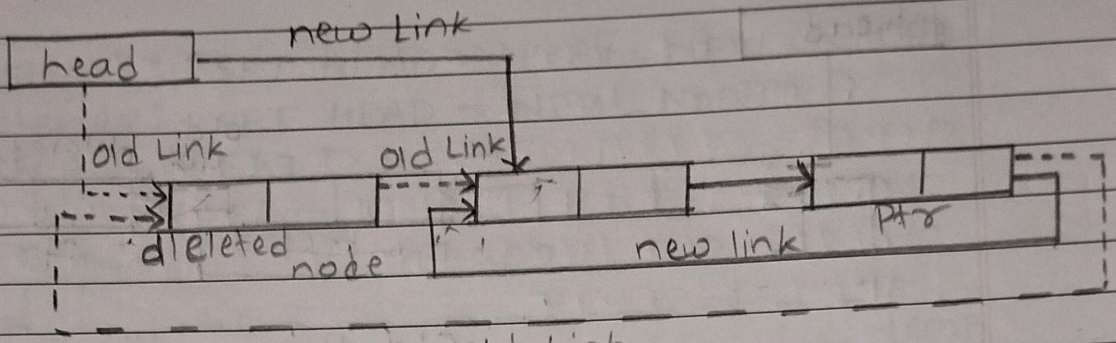


ptr -> next = head
temp -> next = ptr

- Deletion at beginning :-

Algorithm :-

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



Ptr -> next = head -> next
free head
head = Ptr -> next

- Deletion at End / Last :-

Algorithm :-

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 != HEAD

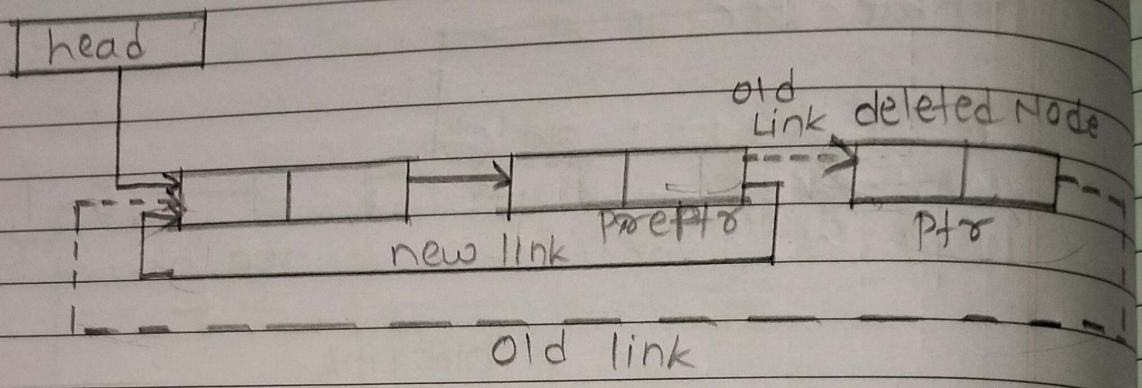
step 4 :- SET PREPTR = PTR

step 5 :- SET PTR = PTR -> NEXT
[END OF LOOP]

step 6 :- SET PREPTR -> NEXT = HEAD

Step 7 :- FREE PTR

step 8 :- EXIT.



preptr -> next = head
free ptr.

Que- Write Pseodu code for insertion and deletion in circular doubly Linked List with diagram

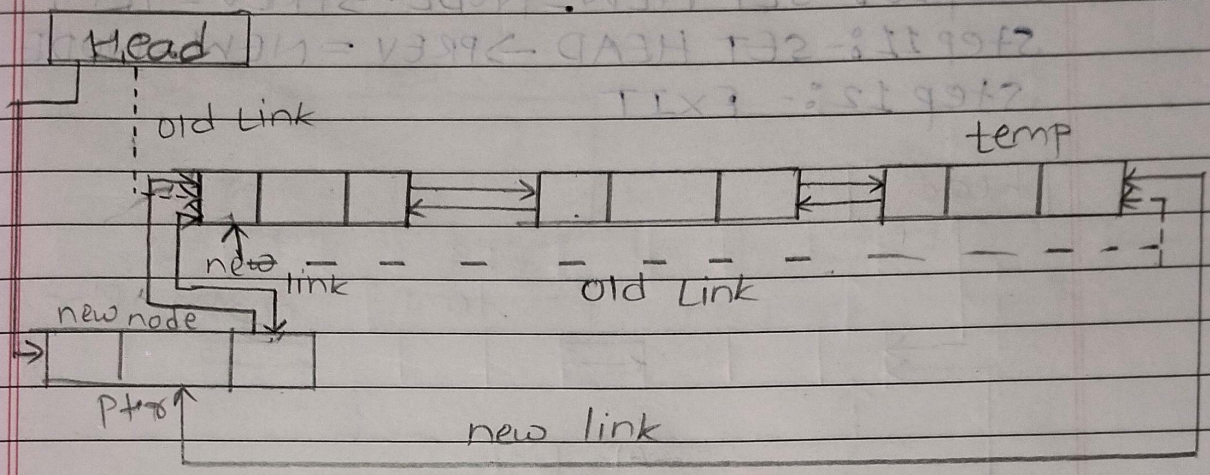
⇒

- Insertion at beginning :-

Algorithm :-

step1 :- IF PTR = NULL
Write OVERFLOW
Go to step 13
[END OF IF]
step2 :- SET NEW_NODE = PTR
step3 :- SET PTR = PTR -> NEXT
step4 :- SET NEW_NODE -> DATA = VAL
step5 :- SET TEMP = HEAD
step 6 :- Repeat step 7 while TEMP -> NEXT != HEAD
step 7 :- SET TEMP = TEMP -> NEXT
[END OF LOOP]
step 8 :- SET TEMP -> NEXT = NEW_NODE
step 9 :- SET NEW_NODE -> PREV = TEMP
step 10 :- SET NEW_NODE -> NEXT = HEAD

Step 11:- SET HEAD -> PREV = NEW_NODE
Step 12:- SET HEAD = NEW_NODE
Step 13:- EXIT

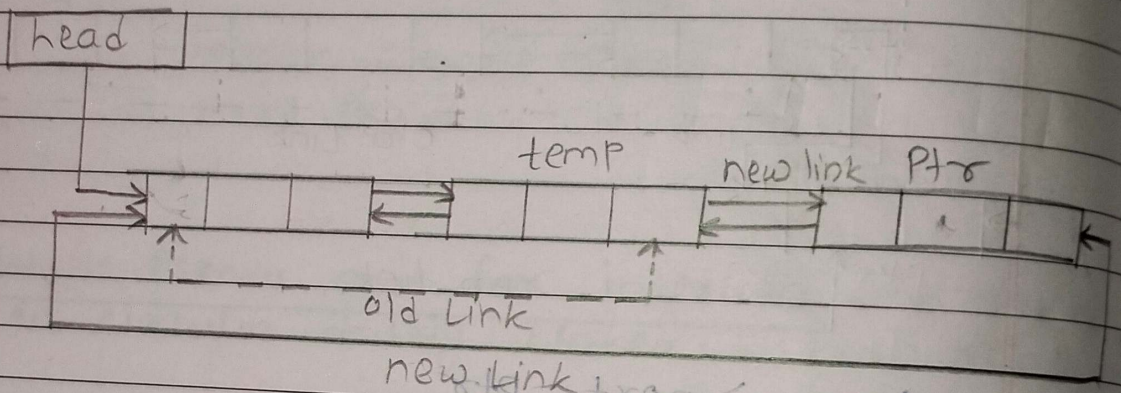


temp -> next = ptr
ptr -> prev = temp
ptr -> next = head
head -> prev = ptr
head = ptr

- Insertion at End:-

Algorithm:-
Step 1:- IF PTR = NULL
Write OVERFLOW
Go to step 12
[END OF IF]
Step 2:- SET NEW_NODE = PTR
Step3:- SET PTR = PTR -> NEXT
step4:- SET NEW_NODE -> DATA = VAL
Step5:- SET NEW_NODE -> NEXT = HEAD
step 6:- SET TEMP = HEAD
step 7:- Repeat Step 8 while TEMP -> NEXT
!= HEAD

step 8:- SET TEMP = TEMP → NEXT
[END OF LOOP]
step 9:- SET TEMP → NEXT = NEW_NODE
step 10:- SET NEW_NODE → PREV = TEMP
step 11:- SET HEAD → PREV = NEW_NODE
step 12:- EXIT

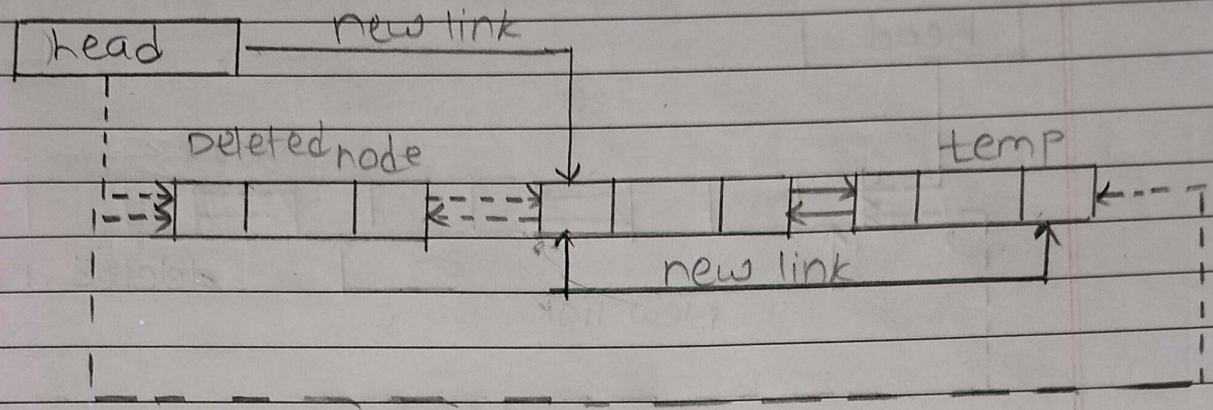


temp → next = Ptr
Ptr → Prev = temp
head → prev = Ptr
Ptr → next = head

- Deletion at beginning:-

Algorithm:-
step 1:- IF HEAD = NULL
write UNDERFLOW
Go to step 8
[END OF IF]
step 2:- SET TEMP = HEAD
Step 3:- Repeat step 4 While TEMP → NEXT != HEAD
step 4:- SET TEMP = TEMP → NEXT
[END OF LOOP]
step 5:- SET TEMP → NEXT = HEAD → NEXT

Step 6:- SET HEAD -> NEXT -> PREV = TEMP
Step 7:- FREE HEAD
Step 8:- SET HEAD = TEMP -> NEXT

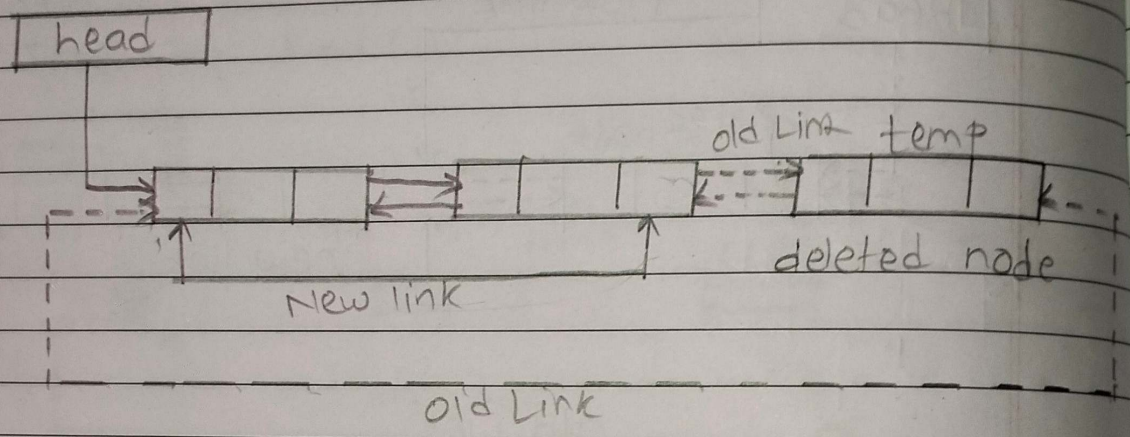


temp -> next = head -> next
head -> next -> prev = temp
free head
head = temp -> next

- Deletion at End:-

Algorithm:-
Step 1:- IF HEAD = NULL
Write UNDERFLOW
Go to step 8
[END OF IF]
Step 2:- SET TEMP = HEAD
Step 3:- Repeat step 4 while TEMP -> NEXT != HEAD
Step 4:- SET TEMP = TEMP -> NEXT
[END OF LOOP]
Step 5:- SET TEMP -> PREV -> NEXT = HEAD

Step 6 :- SET HEAD -> PREV = TEMP -> PREV
Step 7 :- FREE TEMP
Step 8 :- EXIT



temp -> Prev -> next = HEAD
HEAD -> Prev = temp -> Prev
Free temp.