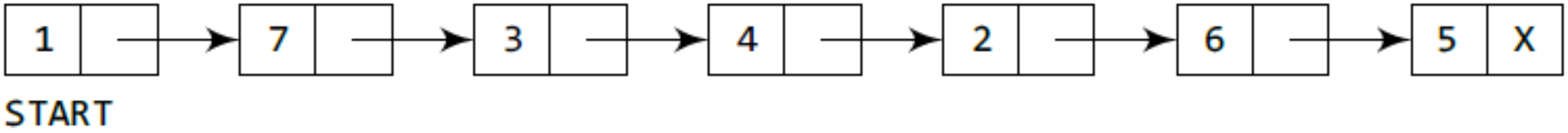
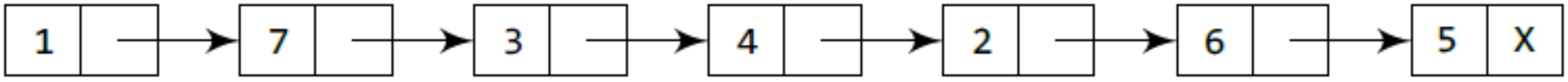
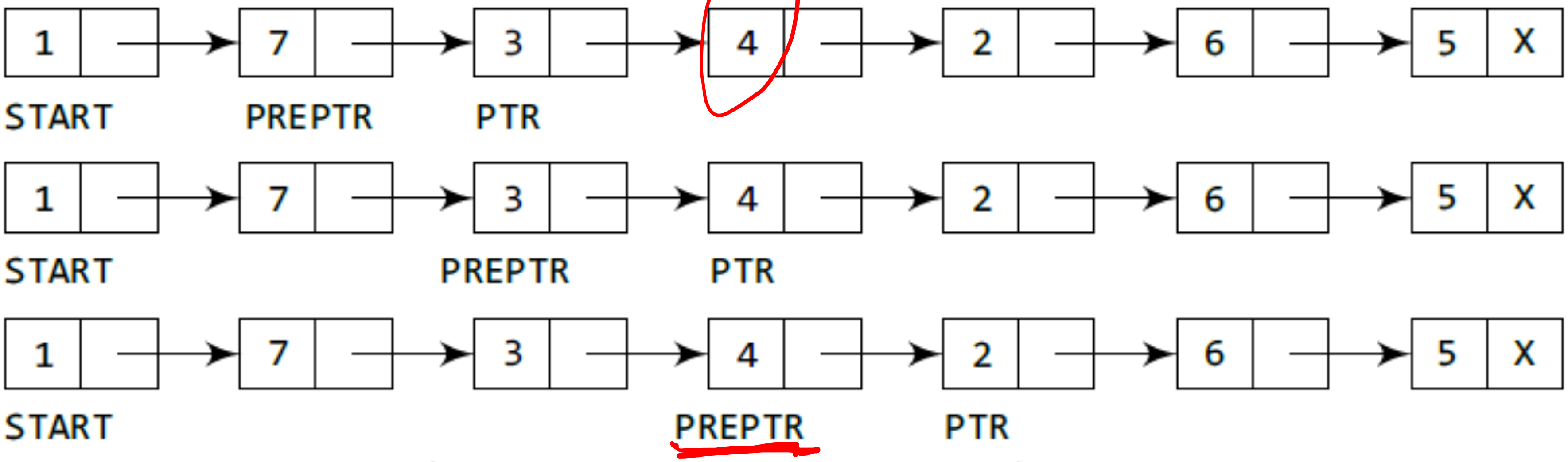


Take pointer variables PTR and PREPTR which initially point to START.

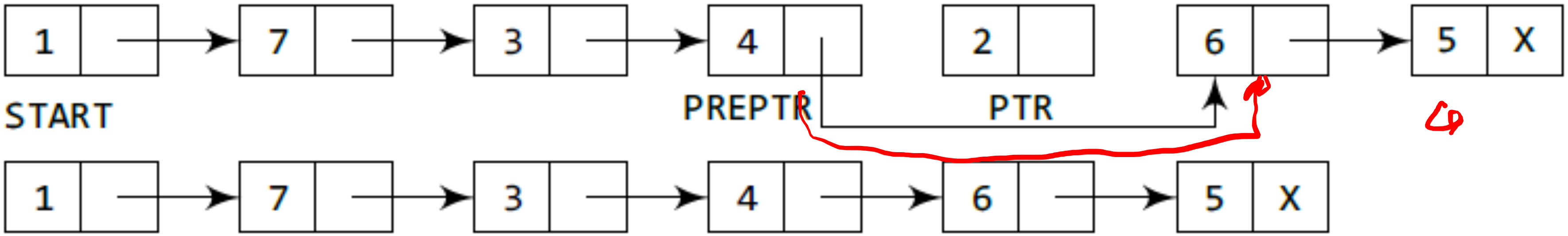


Deleting the Node After a Given Node (4)

Move PREPTR and PTR such that PREPTR points to the node containing VAL and PTR points to the succeeding node.



Set the NEXT part of PREPTR to the NEXT part of PTR.



Delete after a given node (i.e. 4)

Step 1: // check underflow

```
if (START == NULL)
{ printf("underflow\n");          -> O(1)
  return;
}
```

Step 2: find the given node.

```
struct Node *p = START;
struct Node *pre = START;
while (pre->data != 4 && p != NULL)
{     pre = p;                                  -> O(n)
      p = p->next;
}
```

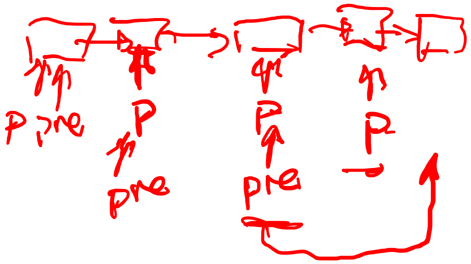
Step 3: Deletion.

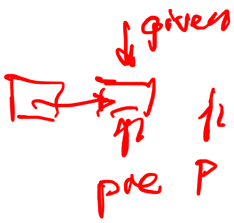
```
if (pre->data == 4)
{ // Found the given node
    if (pre->next == NULL) // give node at the tail
          return;

    pre->next = p->next;        -> O(1)

    free(p);
    p = NULL;
}   return START;
else { printf("Not found data\n");
}  return START;
```

Time Complexity: O(n)





Time Complexity – Worst Case

	Linked List	Array
Access	O(n)	O(1)
Search	O(n)	O(n)
Insertion	O(1)	O(n)
Deletion	O(1) without traversal	O(n)

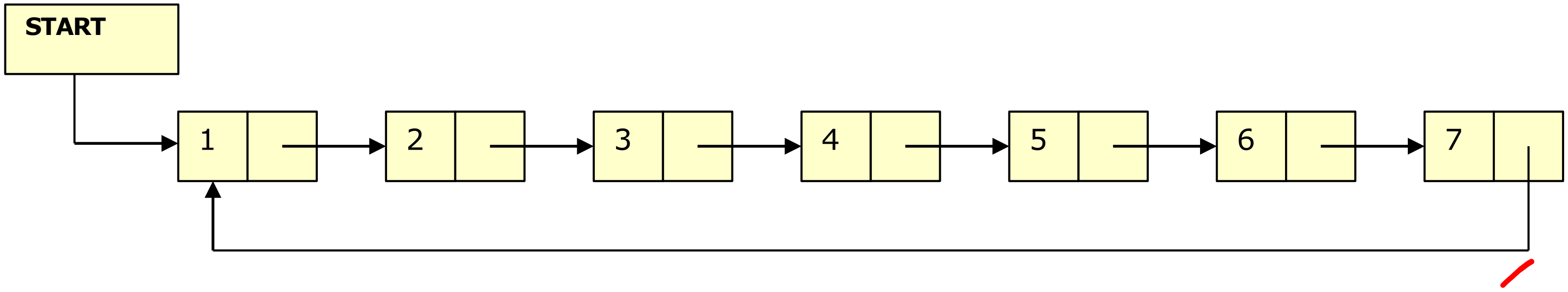
1 2 ~~3~~ 4 5

Circular Linked List

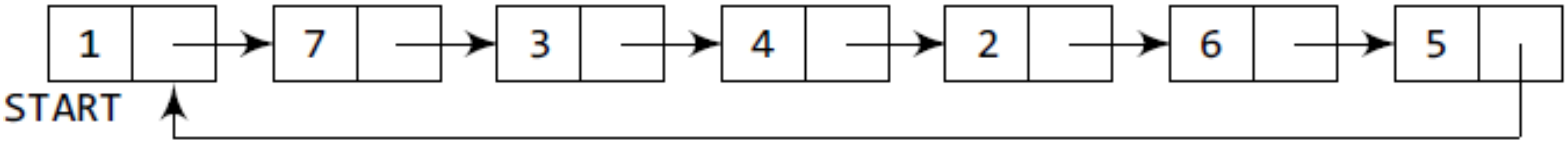
- In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly listed list as well as circular doubly linked list.
- While it looks like a circular linked list has no beginning and no ending, we use START to mark the beginning of the list. We can traverse the list until we find the NEXT entry that contains the address of the first node of the list.
- Circular linked lists are widely used in operating systems for task maintenance.

p->next=NULL

p->next=START

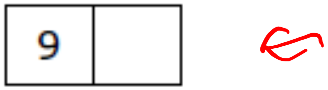


Circular Linked List – Insert at Beginning

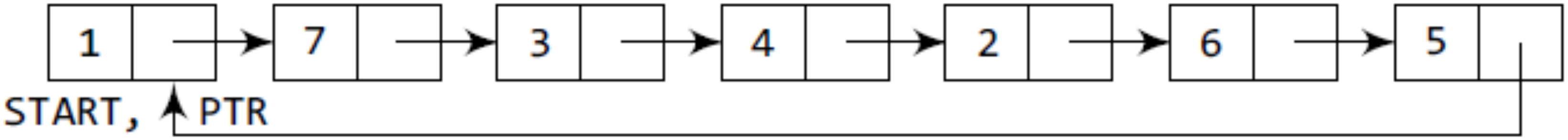


given

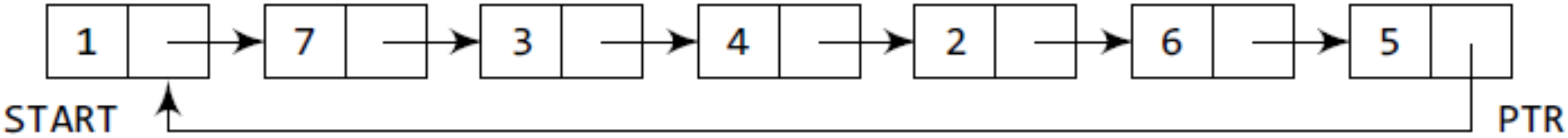
Allocate memory for the new node and initialize its DATA part to 9.



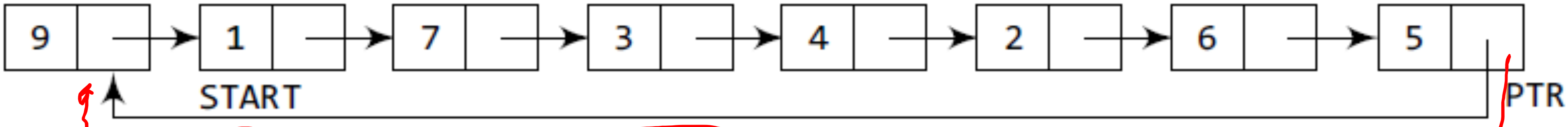
Take a pointer variable PTR that points to the START node of the list.



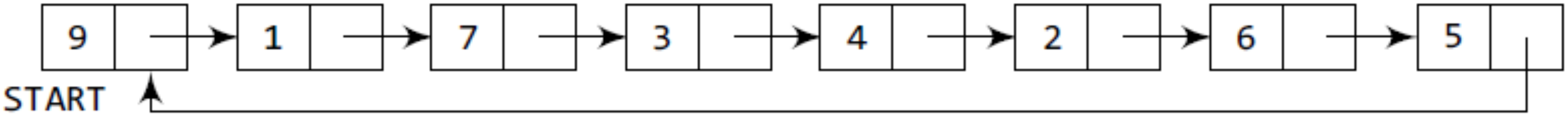
Move PTR so that it now points to the last node of the list.



Add the new node in between PTR and START.



Make START point to the new node.



Circular Linked List – Insert at Beginning

Algorithm to insert a new node in the beginning of the **circular** linked list

```
Step 1: IF AVAIL = NULL, then
                Write OVERFLOW
                Go to Step 10
        [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET New_Node->DATA = VAL
Step 5: SET PTR = START
Step 6: IF PTR == NULL, then
              SET New_Node->Next = New_Node
              Go to Step 10
         [END OF IF]
Step 7: Repeat Step 8 while PTR->NEXT != START
Step 8:       PTR = PTR->NEXT
Step 9: SET New_Node->Next = START
        SET PTR->NEXT = New_Node
Step 10: SET START = New_Node
Step 11: EXIT
```

O(1)

O(n)

O(1)

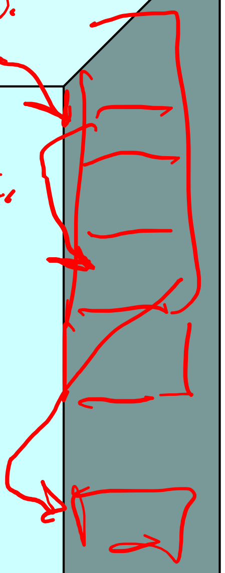
O(1)

05

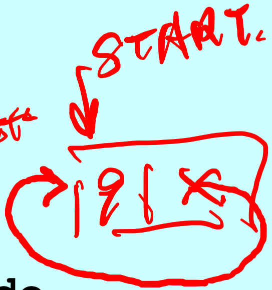
val.

malloc.

newNode == NULL.



START.

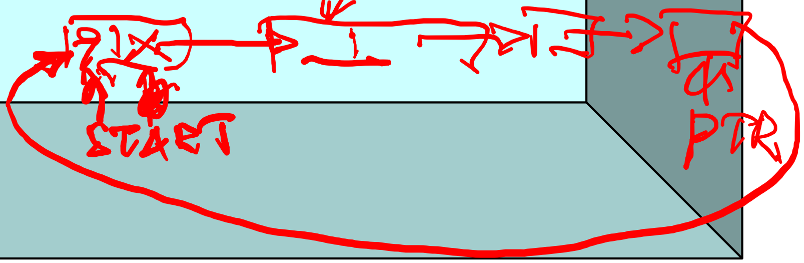


→ empty linked list

Time Complexity O(n).

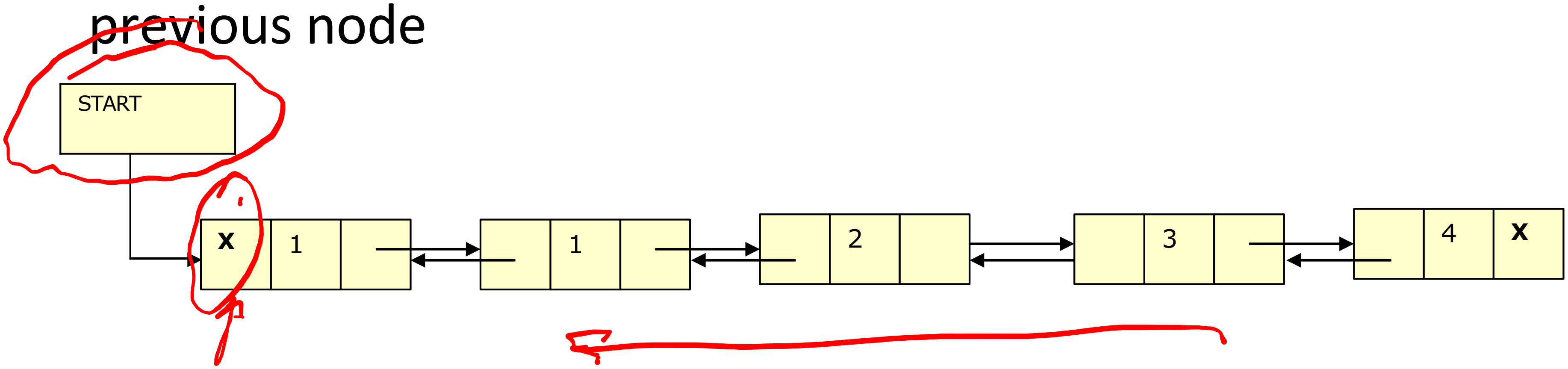
// fixed

p tail START



Doubly Linked List

- A doubly linked list or a two-way linked list is a more complex type of linked list which contains a pointer to the next as well as previous node in the sequence. Therefore, it consists of three parts and not just two. The three parts are data, a pointer to the next node and a pointer to the previous node



Doubly Linked List

- In C language, the structure of a doubly linked list is given as,

```
struct node

{
    struct node *prev;
    int data;
    struct node *next;
};
```

- The prev field of the first node and the next field of the last node will contain NULL. The prev field is used to store the address of the preceding node. This would enable to traverse the list in the backward direction as well.

Homework 1

📌 **Content:** Time Complexity & Linked List

📖 Exercises from the 1st textbook

⏳ **Deadline:** 11:59 PM, Feb 18 (Tuesday), 2025

⚠️ **Late Submission Penalty:** 10% per day