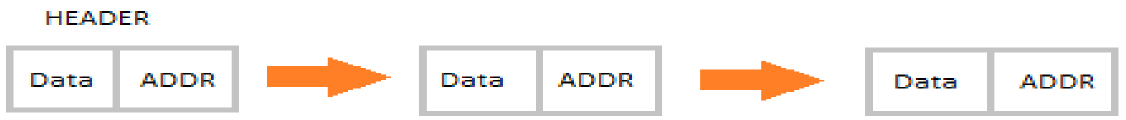
GLS UNIVERSITY

# DATA STRUCTURES.  UNIT– II

## Part 1 – Single Linked List

# Introduction to Linked List

- Linked List is a very commonly used linear data structure which consists of group of nodes in a sequence.
- Linked list is a ADT (Abstract Data Type) consisting of group of nodes in a connected way forming a sequence.
- Each node holds its own data and the address of the next node hence forming a chain like structure.



- Data part of the node hold the actual data while the address part of the node holds the reference or address to its next connected node.

## Advantages of Linked List

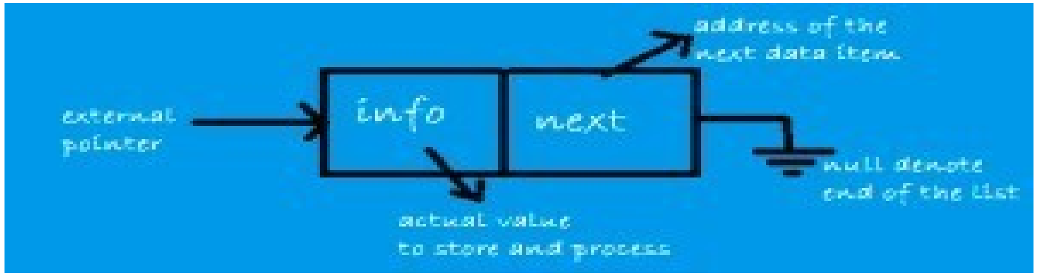
- They are a dynamic in nature which allocates the memory when  required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

## Disadvantages of Linked List

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

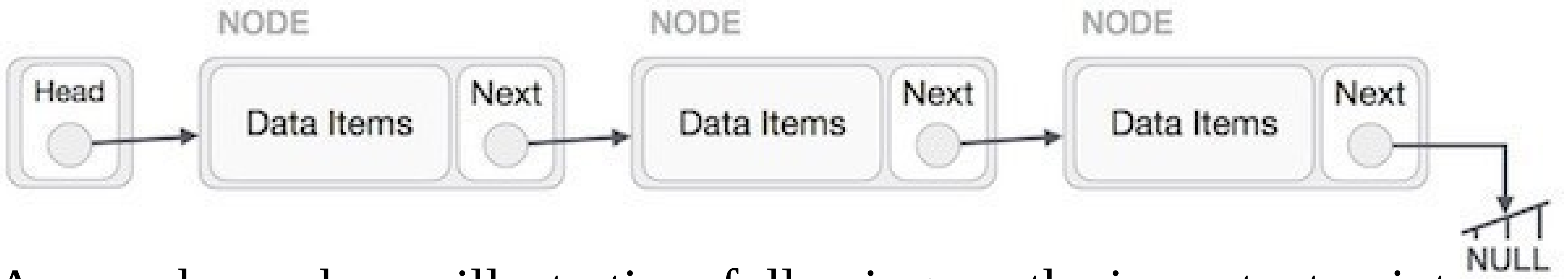
## Structure of Linked list

- A  list consists of linearly connected train of nodes. Each node can be divided into two parts:
- Data Field:It is the actual value that is stored and processed.
- Linked Field: It is the address of the next data item in the Linked



# LINKED LIST Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per above shown illustration, following are the important points to be considered.

- LinkedList contains an link element called first.
- Each Link carries a data field(s) and a Link Field called next.
- Each Link is linked with its next link using its next link.
- Last Link carries a Link as null to mark the end of the list.

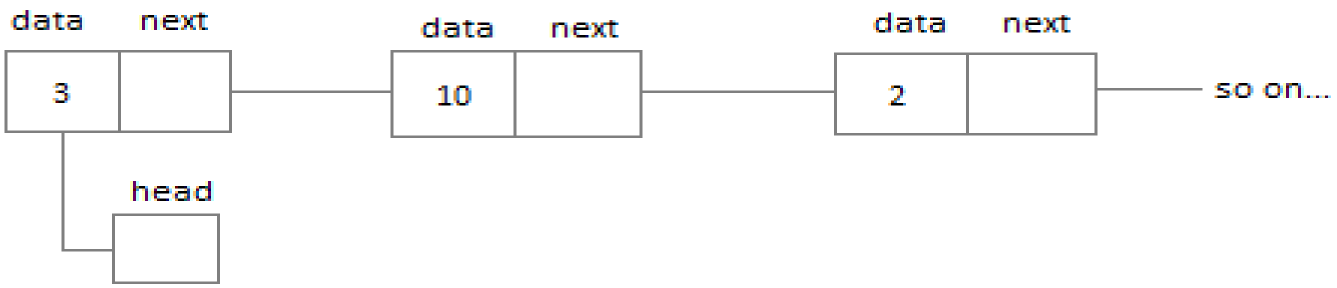
## TYPES OF LINKED LIST

Following are the various flavours of linked list.

- **Singly Linked List −** Item Navigation is forward only.
- **Doubly Linked List −** Items can be navigated forward and backward way.
- **Circular Linked List −** Last item contains link of the first element as next and and first element has link to last element as prev.

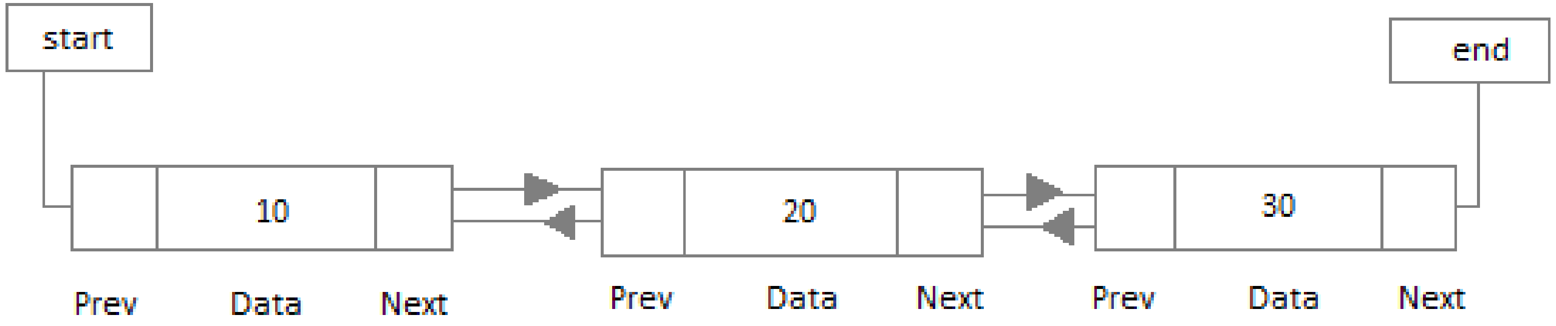
## Singly Linked List

- Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in the sequence of nodes.
- The operations we can perform on singly linked lists are insertion, deletion and traversal.



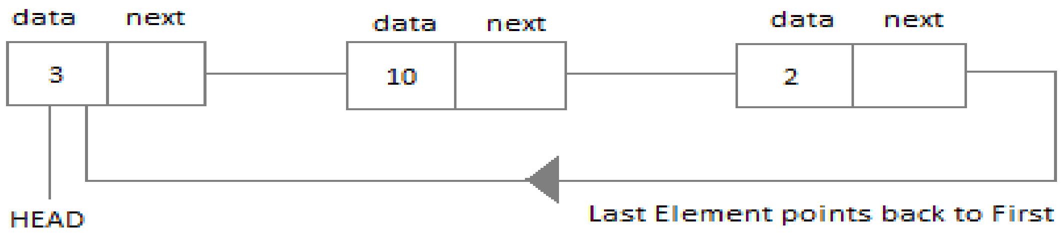
## Doubly Linked List

- In a doubly linked list, each node contains a data part and two addresses, one for the previous node and one for the next node.



## Circular Linked List

- In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.



## Operations on Linked List

- Creation of node
- Inserting node(beginning, end, between)
- Deletion (beginning, end, between)
- Display Linked List

## Creating a Node

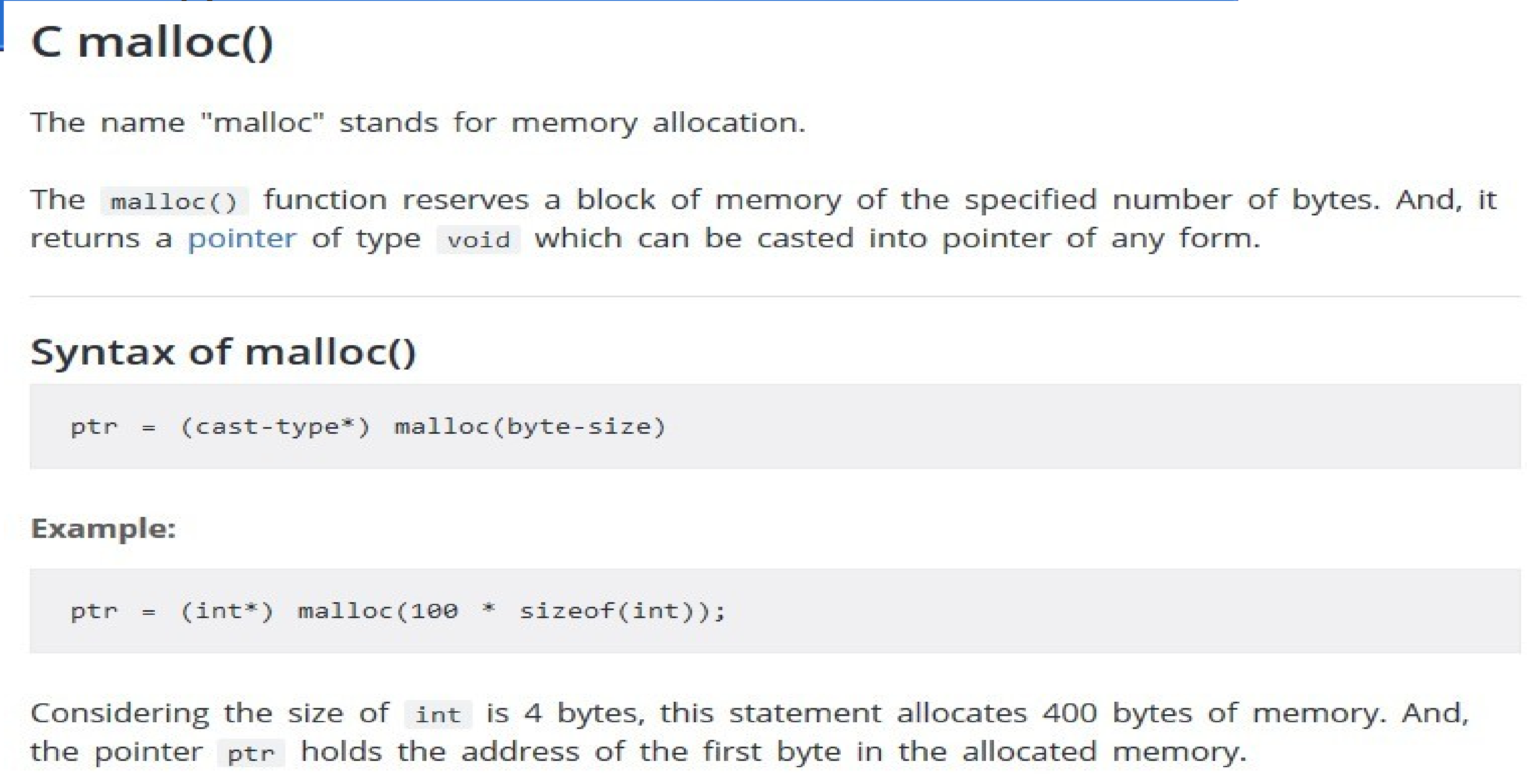
- A node is the main structure of the linked list.
- It contains all details about the data and the address of next node.
- A node can contain many data fields and many address fields, but should contain at least one address field.
- Address part of last node contains a NULL value specifying end of the list.
- A node can be represented either by structures (struct in C or C++) or by classes (class in C++ or Java)

## Creating a Node

- A basic structure of a node in C programming language.

```
// Basic structure of a node
struct node
 {
   int data;   // Data
   struct node * next; // Address
};
```

# Creating a Node



## C malloc()

The name "malloc" stands for memory allocation.

The `malloc()` function reserves a block of memory of the specified number of bytes. And, it returns a pointer of type `void` which can be casted into pointer of any form.

---

### Syntax of malloc()

```
ptr = (cast-type*) malloc(byte-size)
```

**Example:**

```
ptr = (int*) malloc(100 * sizeof(int));
```

Considering the size of `int` is 4 bytes, this statement allocates 400 bytes of memory. And, the pointer `ptr` holds the address of the first byte in the allocated memory.

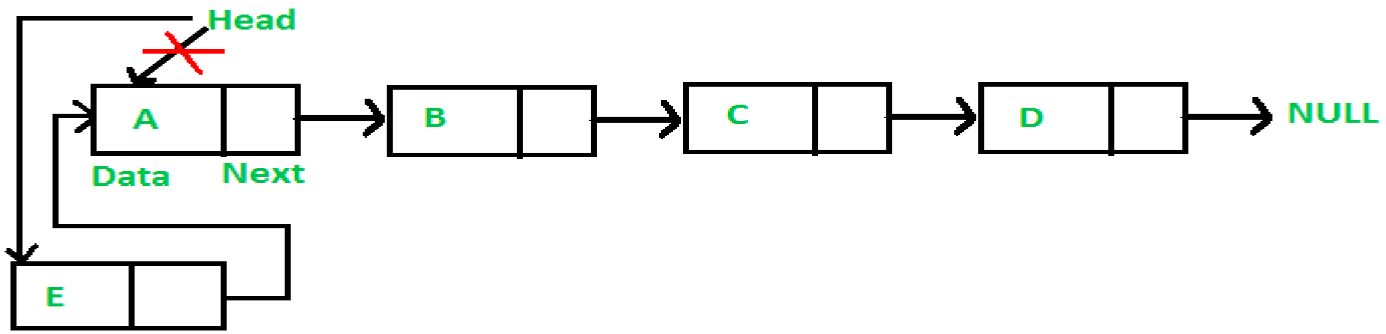
## Inserting a node in the linked list

A node can be added in three ways

1) At the front of the linked list
2) After a given node.
3) At the end of the linked list.

**Add a node at the front:** (A 4 steps process)
The new node is always added before the head of the given Linked List. And newly added node becomes the new head of the Linked List. For example if the given Linked List is 10->15->20->25 and we add an item 5 at the front, then the Linked List becomes 5->10->15->20->25. Let us call the function that adds at the front of the list is push(). The push() must receive a pointer to the head pointer, because push must change the head pointer to point to the new node

## Inserting a node in the linked list

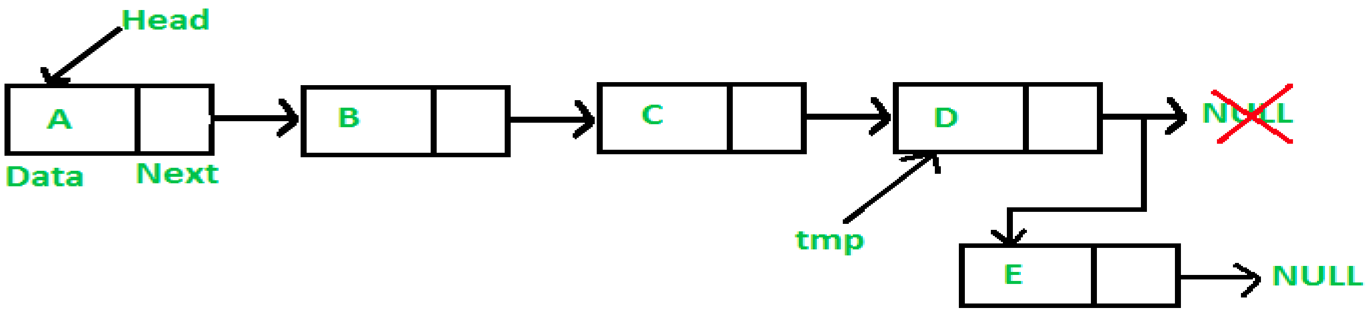


## Algorithm to insert a node in starting

```
Step1: If AVAIL= NULL, then  Write
                 OVERFLOW
                     GO TO STEP 6
Step2: SET New_Node = AVAIL
Step3: SET New_Node -> DATA = VAL  Step4:
SET New_Node -> NEXT = START
 Step5: SET START = New_Node
Step6: EXIT
```

# Add a node at the end

- The new node is always added after the last node of the given Linked List. For example if the given Linked List is 5->10->15->20->25 and we add an item 30 at the end, then the Linked List becomes 5->10->15->20->25->30.
- Since a Linked List is typically represented by the head of it, we have to traverse the list till end and then change the next of last node to new node.



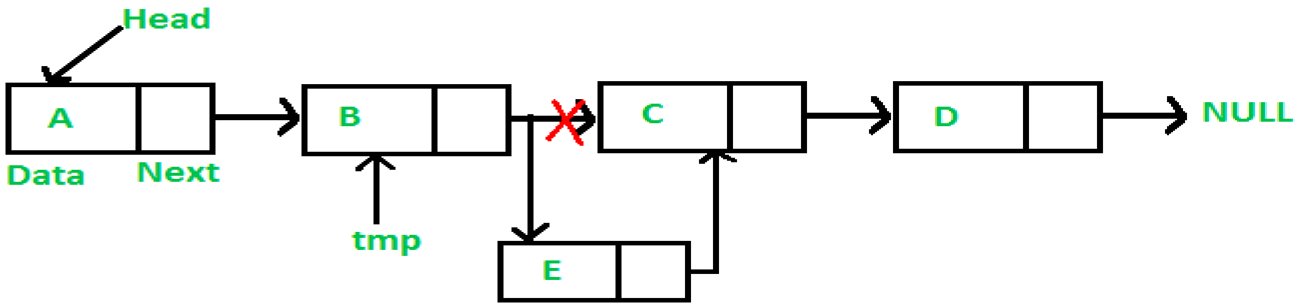
## Algorithm to insert a node in the end

```
Step1: If AVAIL= NULL, then  Write OVERFLOW
GO TO STEP 9
Step2: SET New_Node = AVAIL
Step3: SET New_Node -> DATA = VAL
Step4: SET New_Node -> Next = NULL
Step5: SET PTR = START
Step6: Repeat Step 8 while PTR -> NEXT != NULL
Step7:    SET PTR = PTR -> NEXT
[END OF LOOP]
Step8: SET PTR->NEXT = NEW_NODE
 Step9: EXIT
```

# Inserting a node in the linked list

**Add a node after a given node:**

We are given pointer to a node, and the new node is inserted after the  given node.



## Algorithm to insert a node after given node

```
Step1: If AVAIL= NULL, then  Write OVERFLOW
GO TO STEP 11  [END IF]
Step2: SET New_Node = AVAIL
Step3: SET New_Node -> DATA = VAL
Step4: SET PTR = START
Step5:  SET PREPTR = PTR
```

## Algorithm to insert a node after given node

```
Step6: Repeat Step 7 and 8 while PREPTR -> DATA != NUM
Step7: SET PREPTR = PTR
Step 8: SET PTR = PTR -> NEXT
[END OF LOOP]
Step9: PREPTR->NEXT = NEW_NODE
  Step10: SET NEW_NODE-> NEXT =PTR
Step11: EXIT
```

## Deleting a node in the linked list

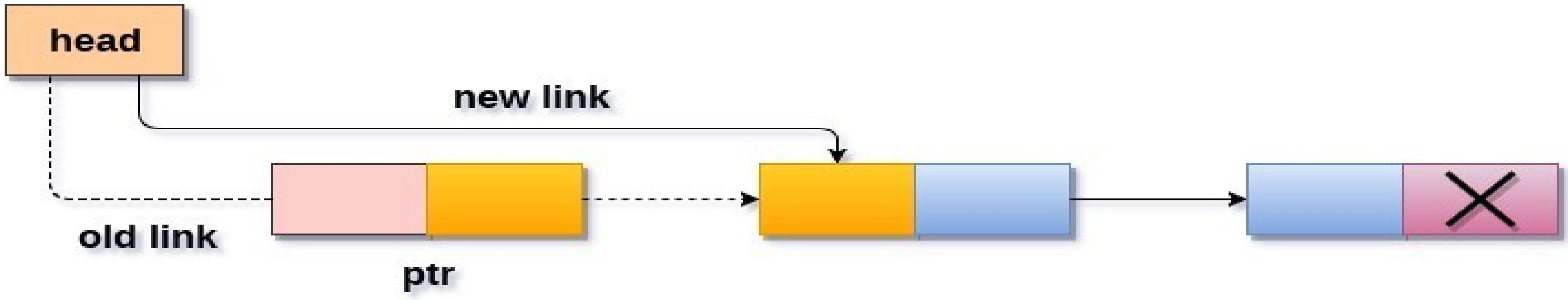
A node can be deleted in three ways

1) The first node is deleted
2) The last node is deleted
3) The node after a given node is deleted.

Underflow:

UNDEEFLOW is a condition that occurs when we try to delete a node from a linked list that is empty. This happens when start=NULL or when there are no more nodes to delete and we try to delete.

## Delete the first node in the linked list



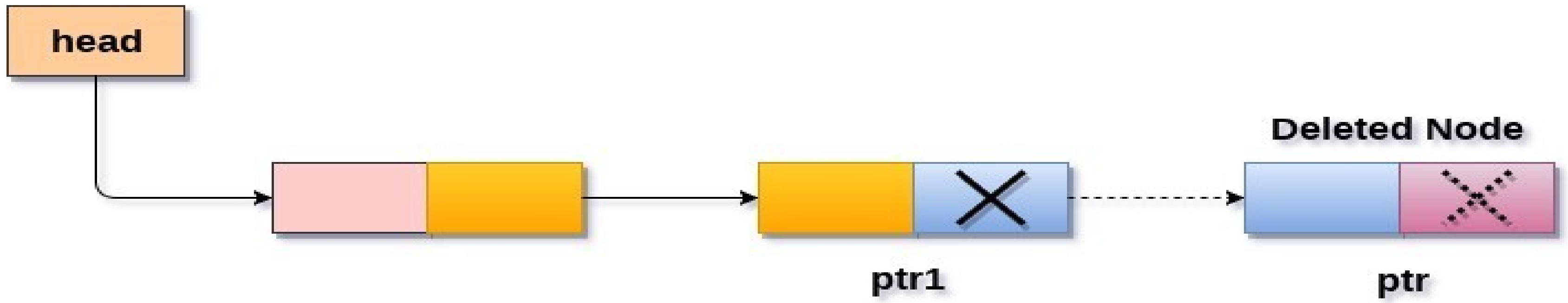
```
ptr = head
head = ptr -> next
free(ptr)
```

Deleting a node from the beginning

## Delete the first node in the linked list

```
Step1: IF START=NULL then
            Write UNDERFLOW
            Go to Step 5  [END OF IF]
   Step2: SET PTR = START
   Step3: SET START= START->NEXT
   Step4: FREE PTR
   Step5:  EXIT
```

# Delete the last node in the linked list



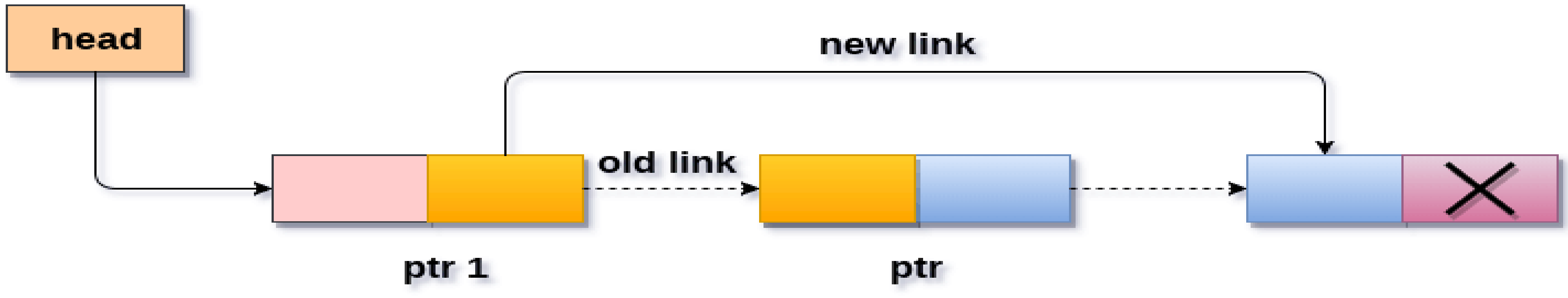
```
ptr1 -> next = Null
free(ptr)
```

Deleting a node from the last

## Delete the last node in the linked list

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
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
```

## Delete the node after a given node in the linked list



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

**Deletion a node from specified position**

## Delete the node after a given node in the linked list

```
Step1: IF START=NULL then
            Write UNDERFLOW
            Go to Step 9  [END OF IF]
Step2: SET PTR = START
Step3: SET PREPTR=PTR
Step4: Repeat Step 5 and 6 while PREPTR->DATA!=NUM
Step5:          SET PREPTR=PTR
Step6:          SET PTR=PTR->NEXT
      [END OF LOOP]
```

## Delete the node after a given node in the linked list

```
Step7: SET PREPTR->NEXT = PTR->NEXT
 Step8: FREE PTR
Step9: EXIT
```