Chapter 2.

Linked List

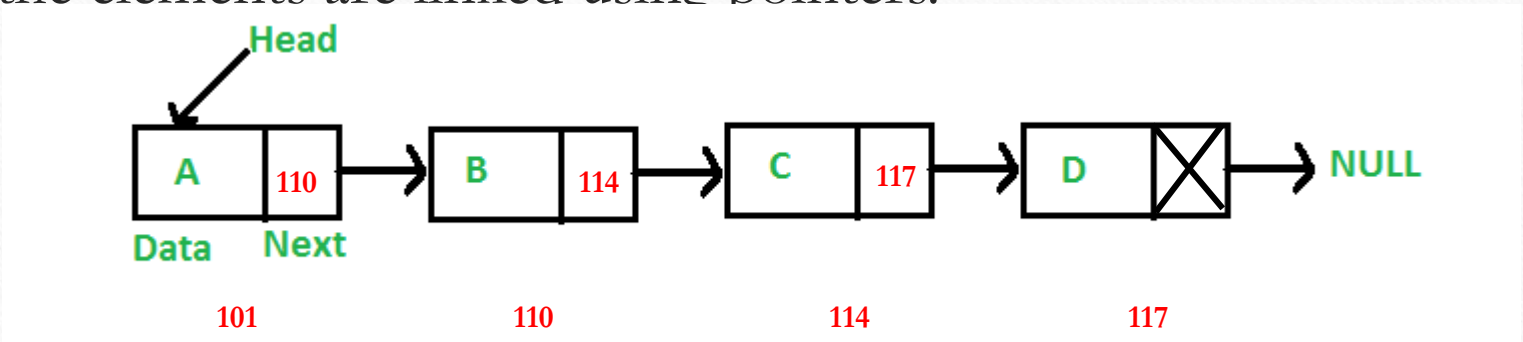
(08 periods)

Content:

1. Introduction to Linked list,
2. Representation of linked list in memory,
3. Traversing,
4. Searching in Unsorted linked list,
5. Overflow and Underflow,
6. Inserting at the beginning of a list,
7. deleting node following a given Node.

Introduction to Linked list

- Like arrays, Linked List is a linear data structure.
- Unlike arrays, linked list elements are not stored at a contiguous location;
- the elements are linked using pointers.



Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.

2) Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

For example

- in a system, if we maintain a sorted list of IDs in an array id[].

  id[] = [1000, 1010, 1050, 2000, 2040].

- And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).
  Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

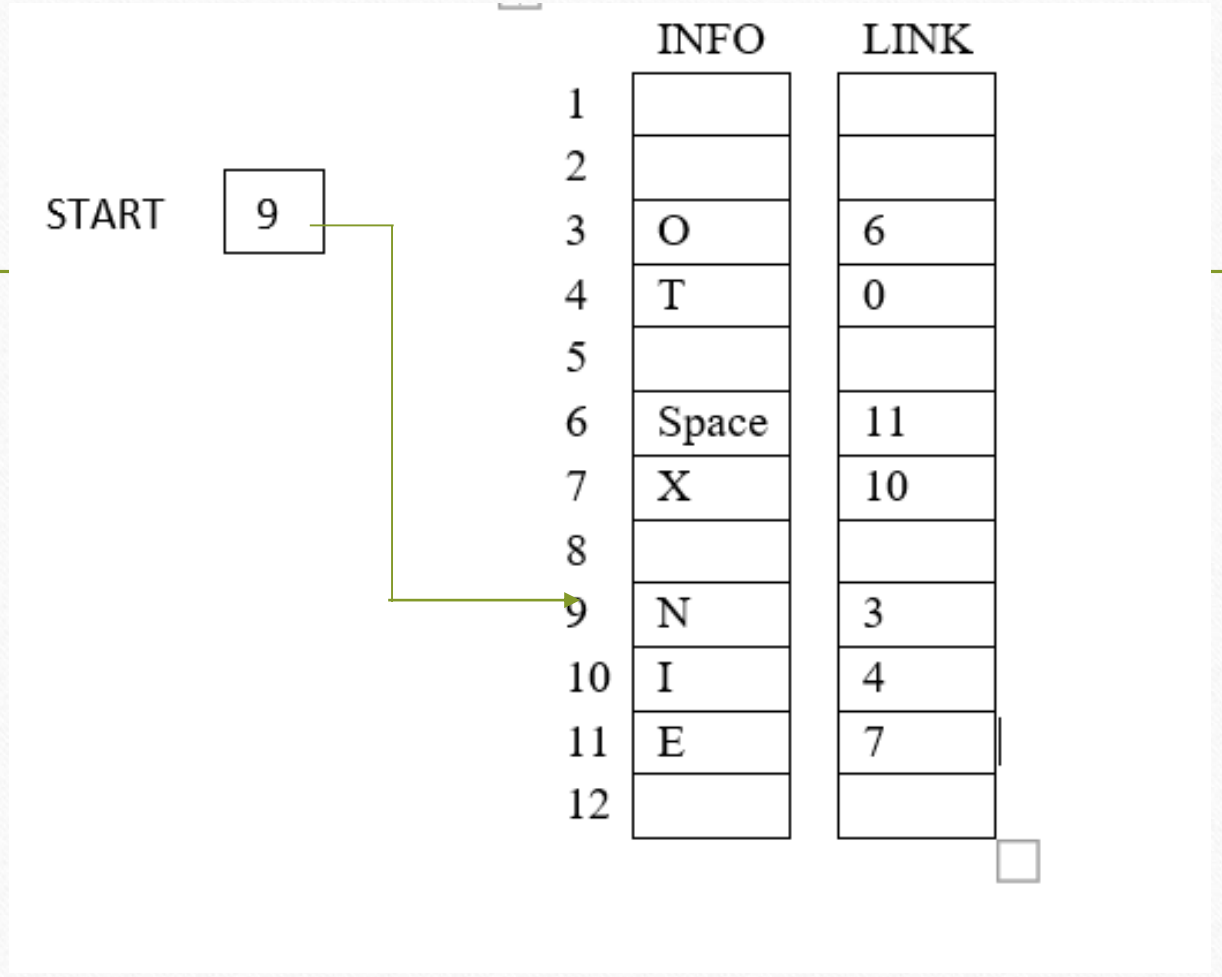
Representation:

- A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.
  Each node in a list consists of at least two parts:
  1) data
  2) Pointer (Or Reference) to the next node
  In C, we can represent a node using structures. Below is an example of a linked list node with integer data.

Representation:

- Let LIST be a linked list. Then LIST will be maintained in memory specified as follows. First of all, LIST requires two linear arrays we will call them here INFO and LINK- such that INFO[K] and LINK[K] contain, respectively, the information part and the *nextpointer* field of a node LIST. As noted above LIST also requires a variable name- such as START. START contains the location of the beginning of the list, and a nextpointer sentinel -denoted by NULL- which indicate the end of the list. Since the subscripts of the array INFO and LINK are usually positive, we will choose NULL=0.

- The following example of linked list indicate that the nodes of a list need not occupy adjacent elements in the array INFO and LINK, and that more than one list may be maintained in the same linear array INFO and LINK. However, each list must have its own pointer variable giving the location of its first node.


START
9
INFO
LINK
1
2
3
O
6
4
T
0
5
6
Space
11
7
X
10
8
9
N
3
10
I
4
11
E
7
12

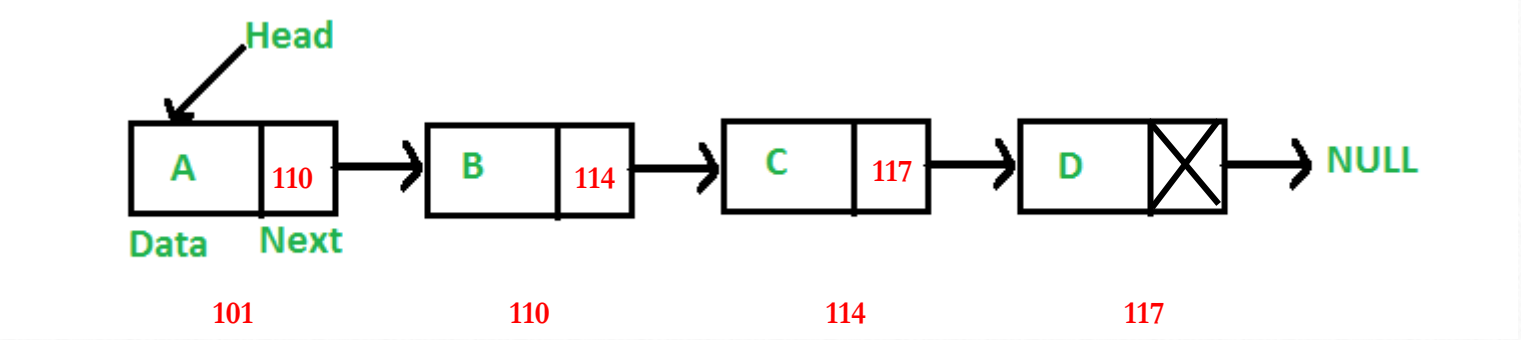
Representation

- Above picture is of linked list in memory where each node of the list contains a single character. We can obtain the actual list of characters as follows.
- **Algorithm**
- START= 9, so INFO[9]=N is the first character.
- LINK[9]=3, so INFO[3]=O is the second character.
- LINK[3]=6, so INFO[6]= (Blank) is the third character.
- LINK[6]=11, so INFO[11]=E is the fourth character.
- LINK[11]=7, so INFO[7]=X is the fifth character.
- LINK[7]=10, so INFO[10]=I is the sixth character.
- LINK[10]=4, so INFO[4]=T is the seventh character.
- LINK[4]=0, the NULL value, so the List has ended.
- **In other words, NO EXIT is the character string.**

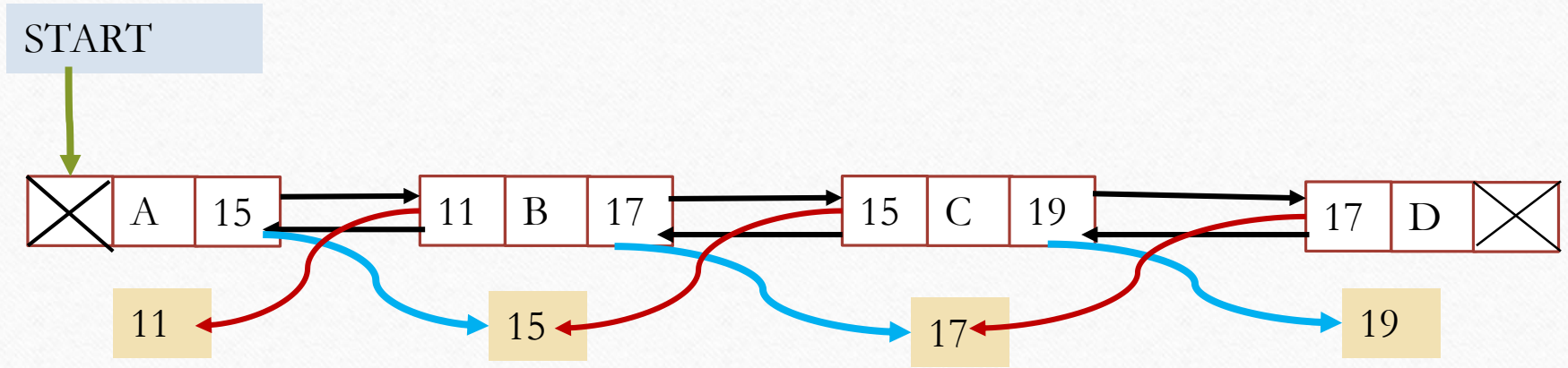
Types of Linked List

- Following are the various types of linked list.
- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

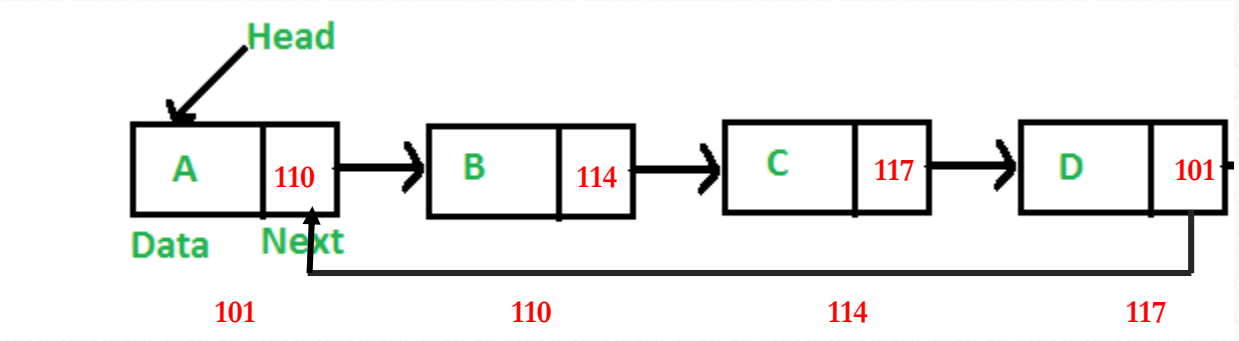
Simple Linked List



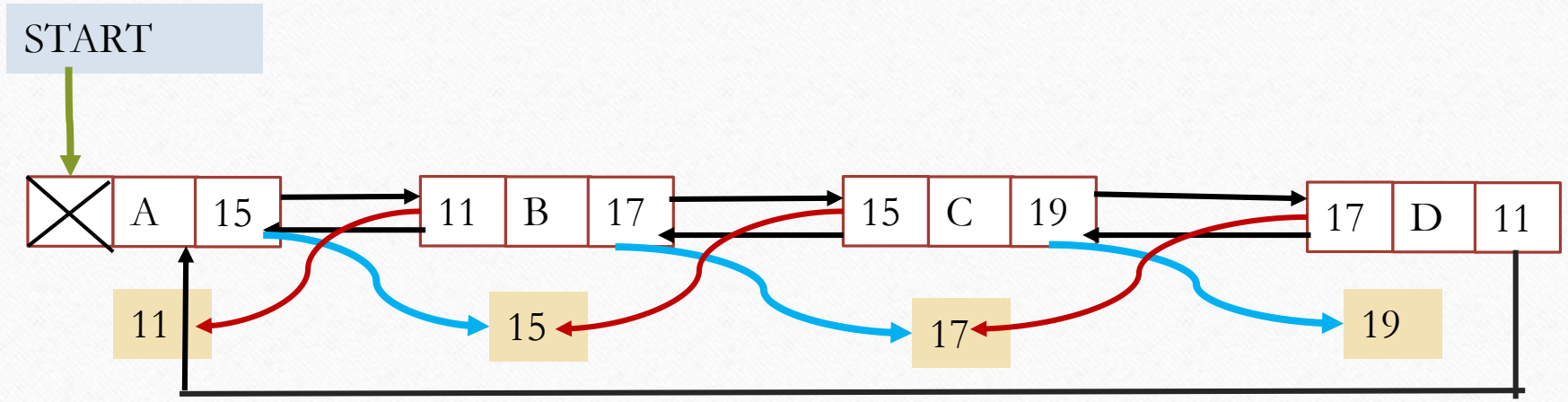
Doubly Linked List



Circular Linked List (Simple)



Circular Linked List (Double)



2.3 Traversing

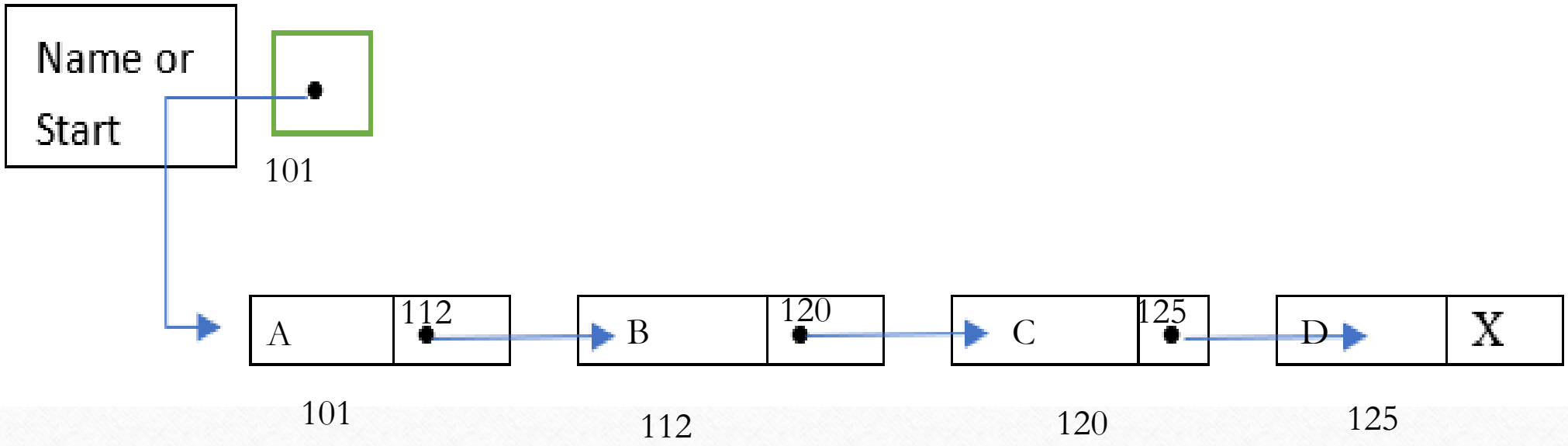
A linked list is a linear data structure that needs to be traversed starting from the head node until the end of the list. Unlike arrays, where random access is possible, linked list requires access to its nodes through sequential traversal. Traversing a linked list is important in many applications. For example, we may want to print a list or search for a specific node in the list. Or we may want to perform an advanced operation on the list as we traverse the list. The algorithm for traversing a list is fairly trivial.

a. Start with the head of the list. Access the content of the head node if it is not null.

b. Then go to the next node(if exists) and access the node information

c. Continue until no more nodes (that is, you have reached the last node)

Let LIST be a linked list in memory stored in linear array INFO and LINK with START pointing to the first element and NULL indicating the end of LIST. Suppose we want to traverse LIST in order to Process each node exactly once. This section presents an algorithm that does so and then uses the algorithm in some applications.


Name or
Start
101
A
112
B
120
C
125
D
X
101
112
120
125

Algorithm: 1. Set PTR:=START.[Initializes pointer PTR]

2. Repeat Step 3 and 4 while PTR≠ NULL.

3. Apply PROCESS to INFO[PTR].

4. Set PTR:= LINK[PTR]. [PTR now points to the next node.]

[End of Step 2 loop.]

5. Exit.

Algorithm details

Initialize PTR or START.

Then process INFO[PTR], the information at the first node.

Update PTR by the assignment PTR:=LINK[PTR], so that PTR points to the second node.

Then Process INFO[PTR], the information at the second node.

Again update PTR by the assignment operator PTR:=LINK[PTR], and then process INFO[PTR], the information at the third node.

And so on. Continue until PTR=NULL, which signals the end of the list.

Traversing Linear Array

- LA- Array, Lower Bound= LB, Upper Bound=UB, S-> Process each element

- Algorithm: 1) Initialize Counter I=LB

2) Loop (While I<=UB)

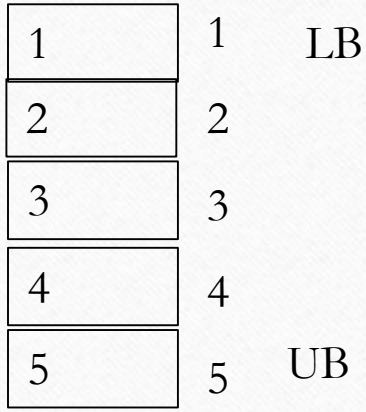
3) Apply S to LA[I]

4) I=I+1

5) End Loop

6) Exit

I=LB	LA[i] *2	I=I+1
1	1*2=2	I=1+1=2
2	2*2=4	I=2+1=3
3	3*2=6	I=3+1=4
4	4*2=8	I=4+1=5
5	5*2=10	I=5+1=6



```
#include <stdio.h>
#include<conio.h>
 void main()
{ int LA[] = {2,4,6,8,9};
int i, n = 5;
printf("The array elements are:\n");
for(i = 0; i < n; i++)
{
 printf("LA[%d] = %d \n", i, LA[i]);
}
getch();
}
```

Command Prompt

The array elements are:

LA[0] = 2

LA[1] = 4

LA[2] = 6

LA[3] = 8

LA[4] = 9

2.4 Searching Unsorted linked list

- Let LIST be a linked list in memory which is not sorted, then one searches for ITEM in LIST by traversing through the list using a pointer variable PTR and comparing ITEM with the contents INFO[PTR] of each node, one by one, of LIST. Before we update the pointer PTR by PTR:=LINK[PTR]
- We require two tests. First we have to check to see whether we have reached the end of list i.e.
- PTR= NULL
- Then we check to see whether
- INFO[PTR]=ITEM

Algorithm:

SEARCH (INFO, LINK, START, ITEM, LOC)

LIST is a linked list in memory. This algorithm finds the location LOC of the node where ITEM first appear in LIST, or set LOC=NULL.

set PTR:=START.

Repeat step 3 while PTR ≠ NULL:

If ITEM =INFO[PTR], then:

Set LOC:=PTR, and Exit.

Else:

Set PTR:=LINK[PTR].[PTR now points to the next node.]

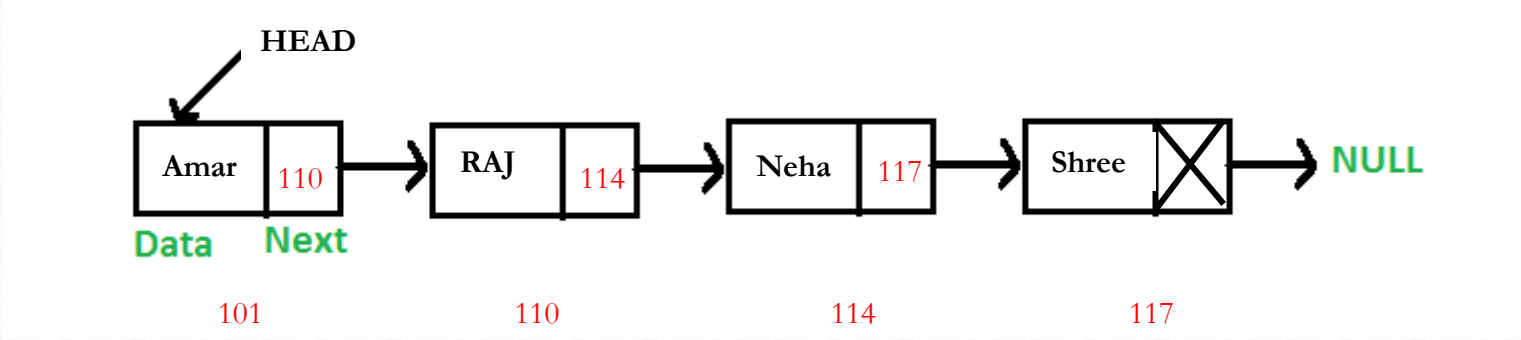
[End of IF structure]

[End of step 2 loop]

[Search is unsuccessful.] set LOC:=NULL.

Exit.

- Want to Search For Shree then



Overflow

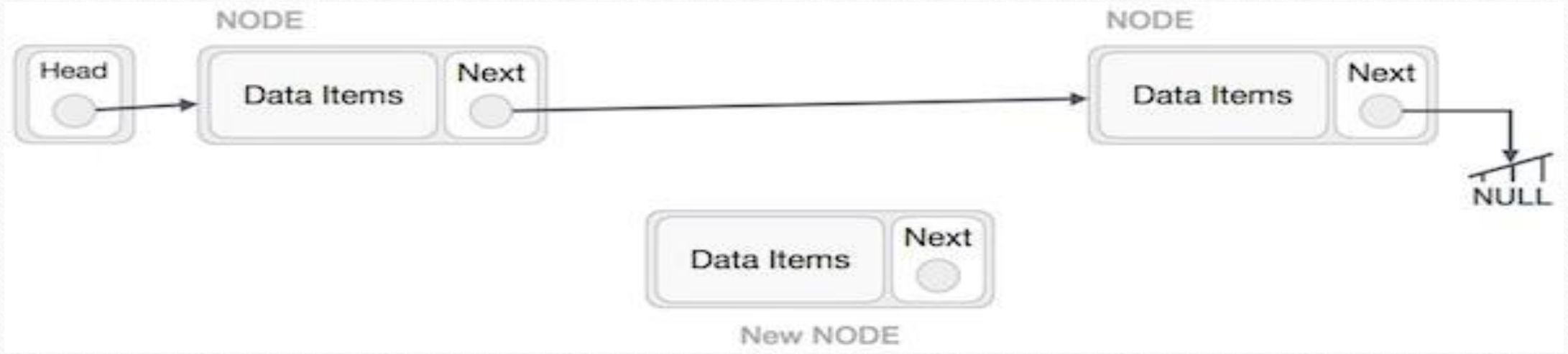
- Sometimes new data are to be inserted into a data structure but there is no available space, i.e. the free-storage list is empty. This situation is usually called overflow.

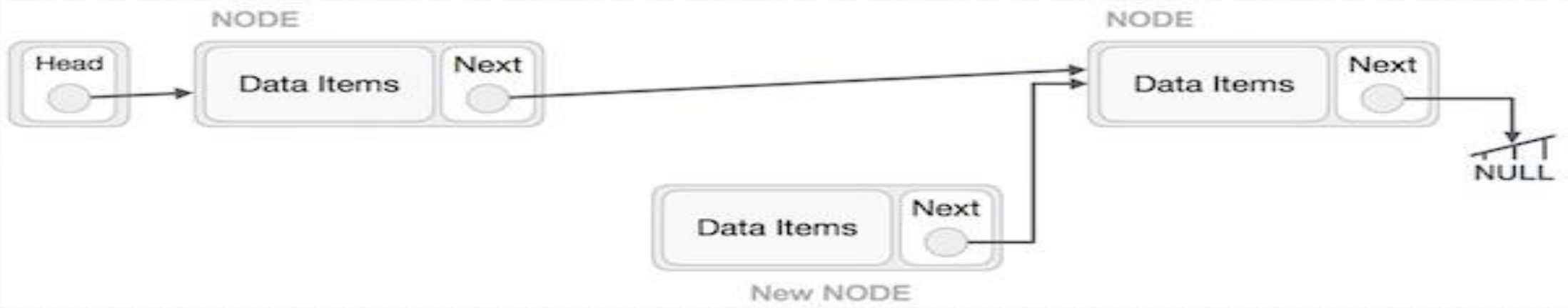
- The programmer may handle overflow by printing the message OVERFLOW.
- In such a case, the programmer may then modify the program by adding space to the underlying arrays.
- Observe that overflow will occur with our linked lists when AVAIL=NULL and there is an insertion.

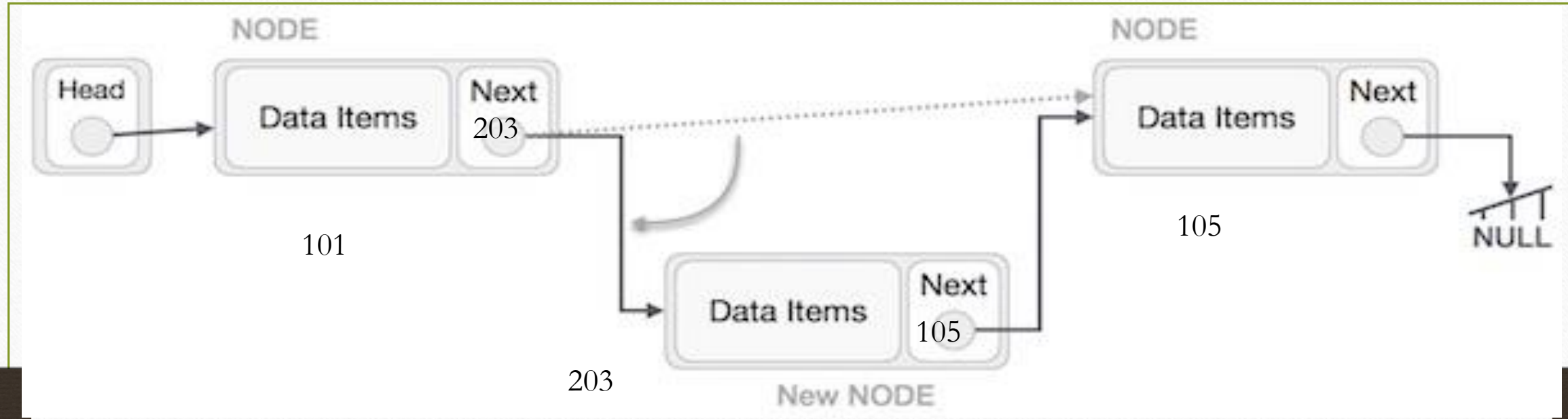
UNDERFLOW

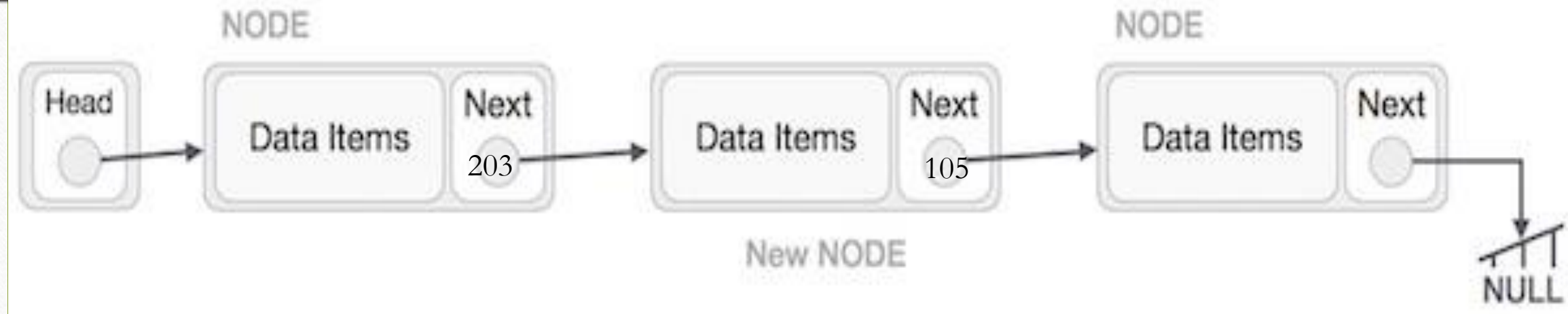
- The term underflow refers to the situation where one wants to delete data from a data structure that is empty.
- The programmer may handle underflow by printing the message UNDERFLOW.
- Observe that underflow will occur with our linked when START = NULL and there is a deletion.

Inserting at the beginning of a list,

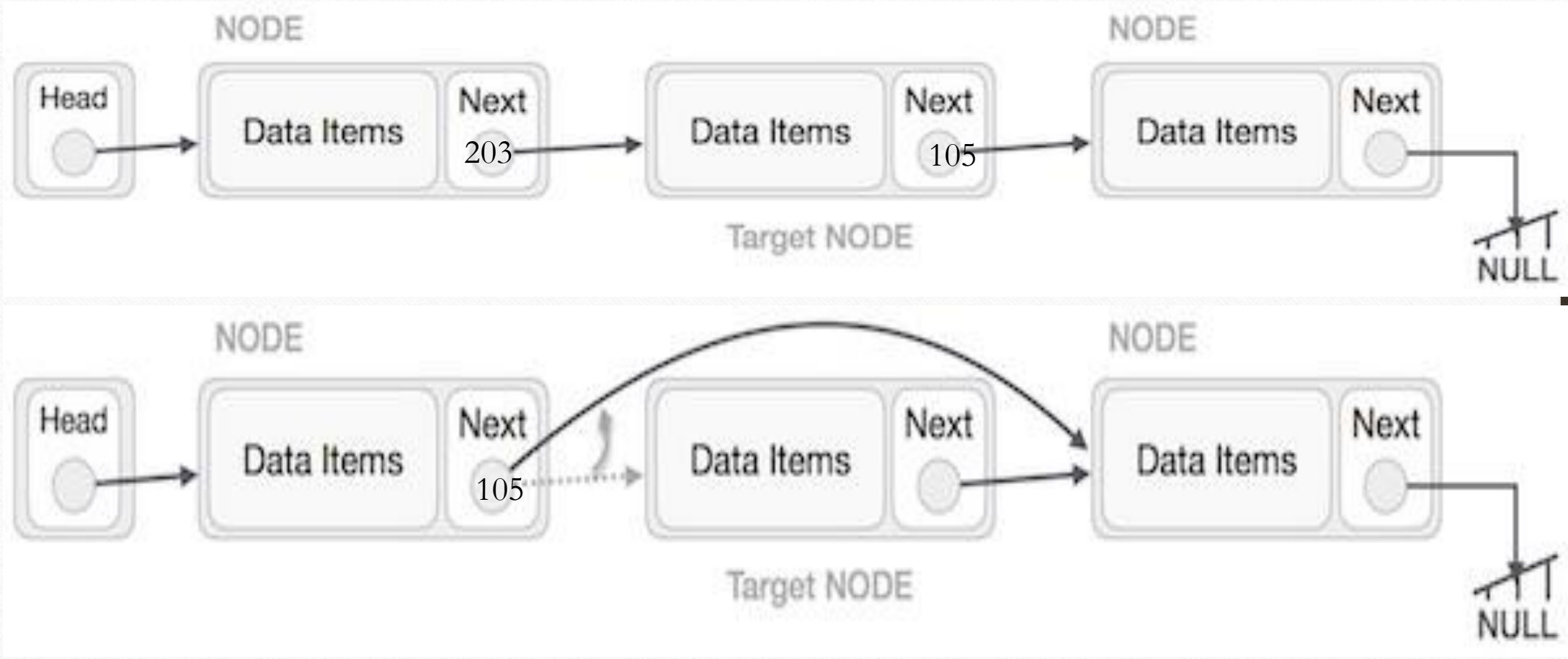


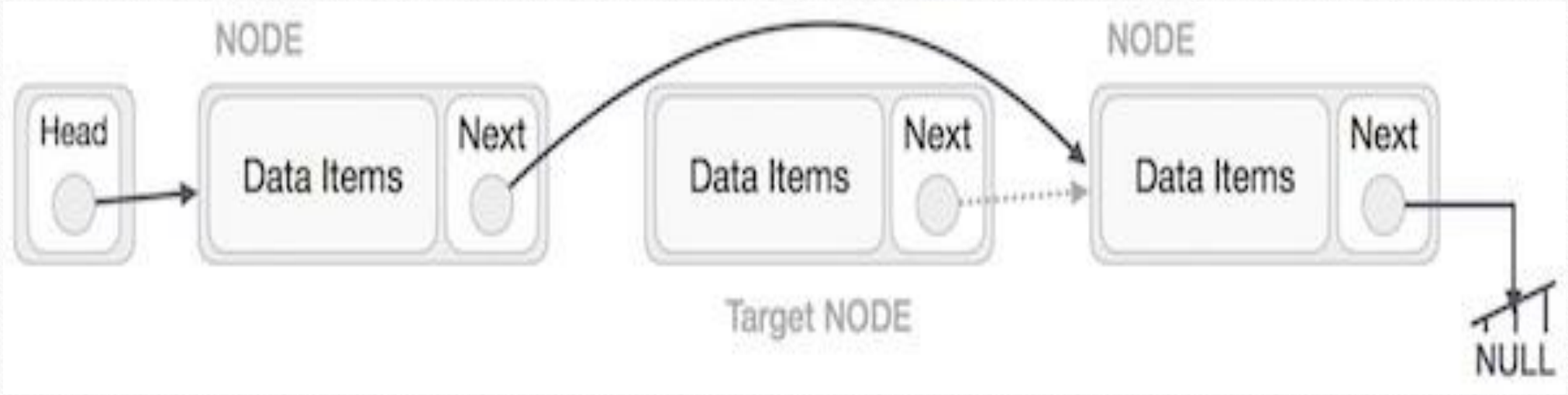


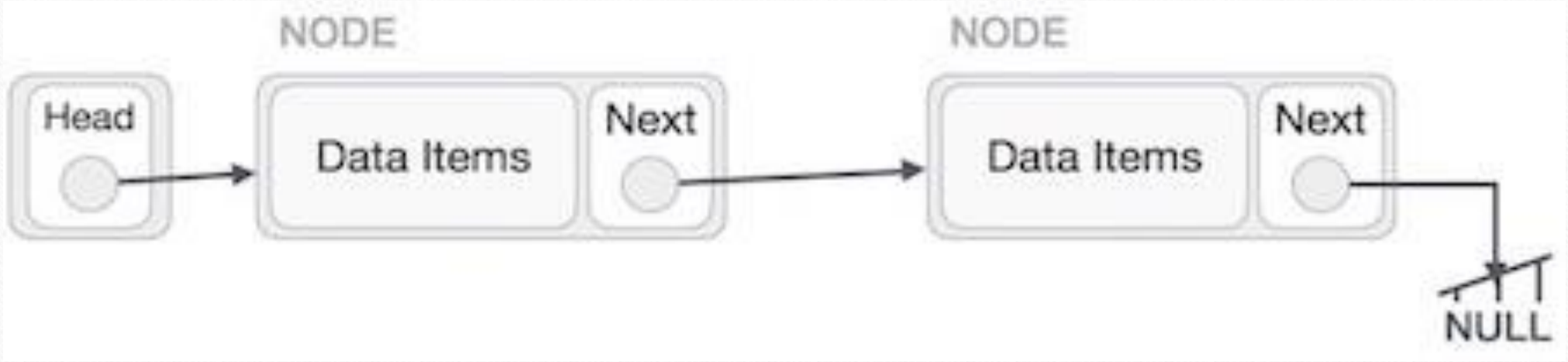

NODE
Head
Data Items
Next
203
101
NODE
Data Items
Next
105
NULL
Data Items
Next
105
203
New NODE


NODE
Head
Data Items
Next
203
Data Items
Next
105
NODE
Data Items
Next
New NODE
NULL

Deleting Node following a given Node.




NODE
NODE
Head
Data Items
Next
Data Items
Next
Data Items
Next
Target NODE
NULL


NODE
NODE
Head
Data Items
Next
Data Items
Next
NULL

Question Bank

1. What is Linked list? Explain it with suitable example.
2. Explain the representation of linked list in memory.
3. Explain Traversing of linked list with suitable example.
4. Explain searching in Unsorted linked list.
5. Write a short note on overflow and underflow.
6. Write a procedure for inserting element in a linked list
7. Explain deleting node in a linked list.