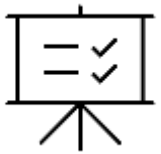
Advanced Algorithms & Data Structures

AIM OF THE SESSION

To familiarize students with the basic concept of Singly linked list

INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1. Demonstrate The Singly linked list
2. Describe the types of Singly linked list.
3. List out the operations possible on the Singly linked list.
4. Describe each operation
5. List out the advantages and applications of SLL.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define Singly linked list
2. Describe two types of Singly linked list. And their operations
3. Summarize definition, types and operations of circular linked list and its applications

Linked List

- The term ***list*** refers to a linear collection of data items.
- One way to store such data is by means of ***Arrays and Linked List***.
- The ***Array*** implementation uses static structures where as ***Linked lists*** implementation uses dynamically allocated structures.

Array Implementation:

The array implementation of a list has certain drawbacks. These are:

1. Operations like insertion and deletion at a specified location in a List requires a lot of movement of data, therefore, leading to inefficient and time consuming algorithm.
2. Memory storage space is wasted; very often the List is much shorter than the array size declared.
3. List cannot grow in its size beyond the size of the declared array if required during program execution.

Linked List Implementation:

Following are the advantages of Linked Lists over Arrays:

1. Linked list is a dynamic structure where as array is a static structure.
2. Insertion and deletion of nodes requires no data movement.

Types of Linked List :

Following are the 4 types of linked lists:

1. Singly Linked List
2. Circular Linked List
3. Doubly Linked List
4. Circular Doubly Linked List

Singly Linked List:

- A **Singly Linked List** is a linear collection of data elements, called **nodes.**
- Here each node is divided into two fields. The first field contains the information of the element, and the second field contains the address of the next node in the list.
- The first field is called **Data field** and the second field is called **Next field.**
- Following figure shows the structure of a node in a linked list. The nodes in a linked list are called **self-referential** structures.

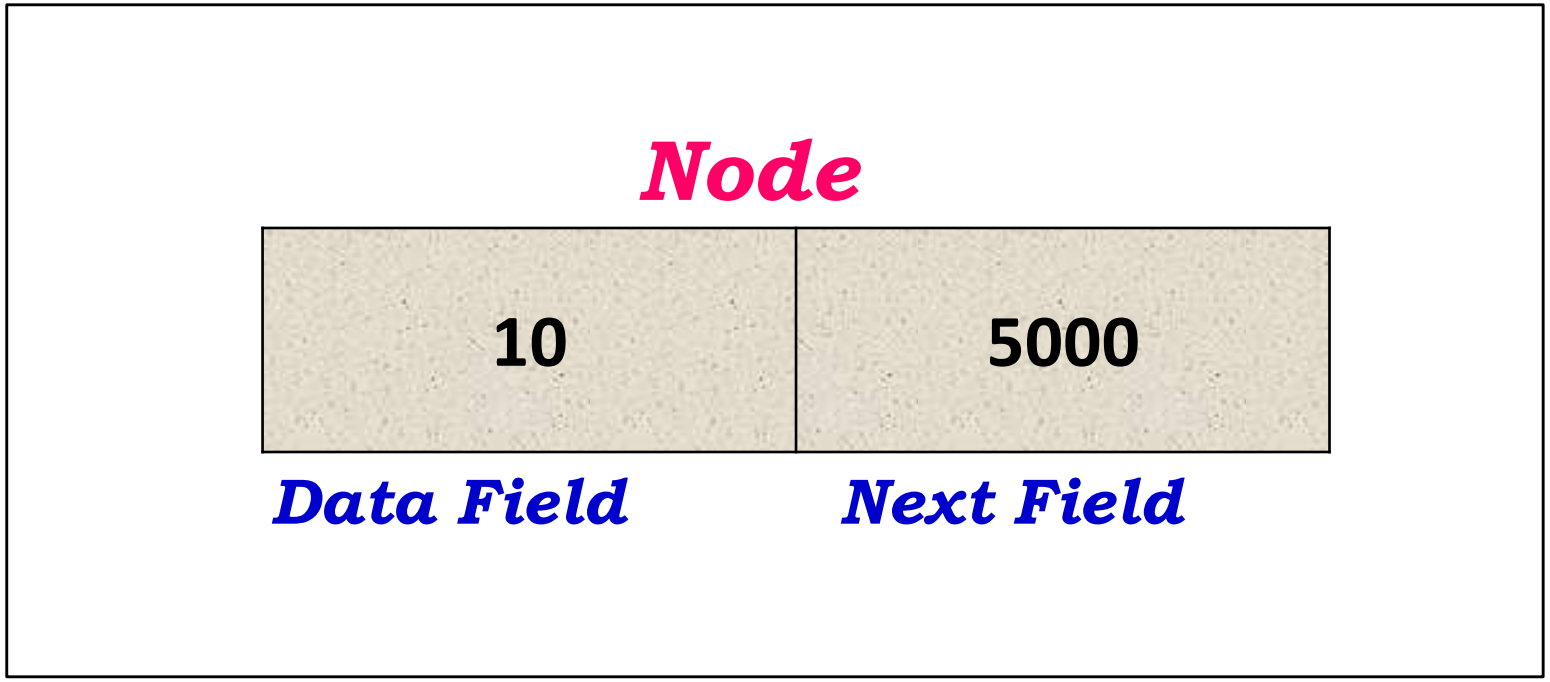
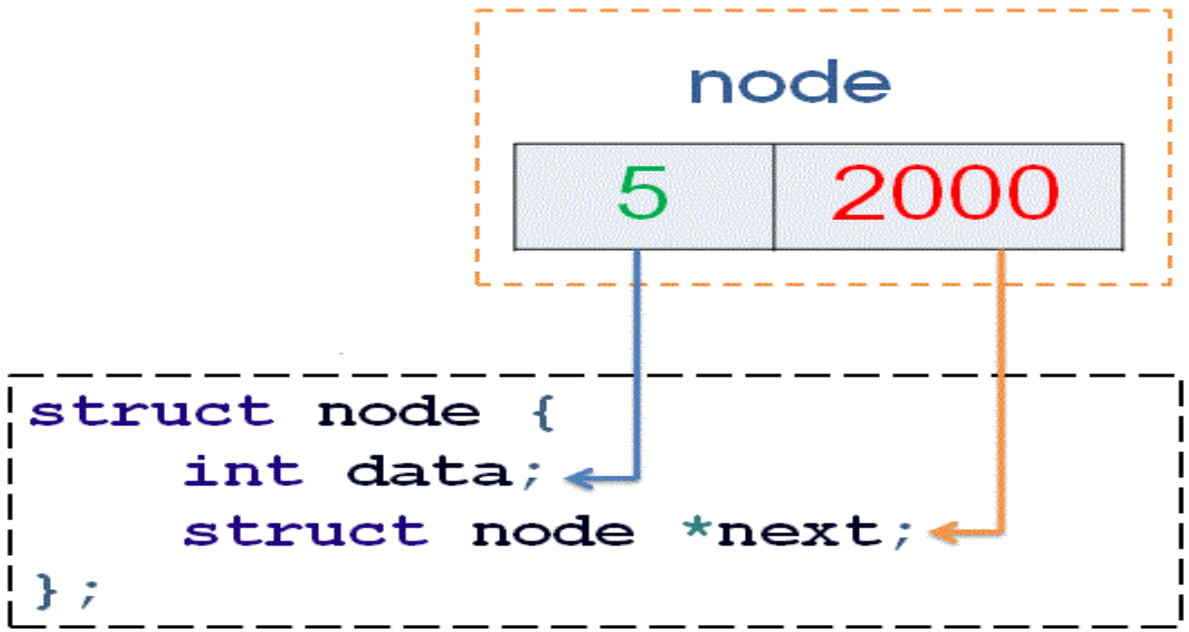


Figure: Structure of a node in a Linked List



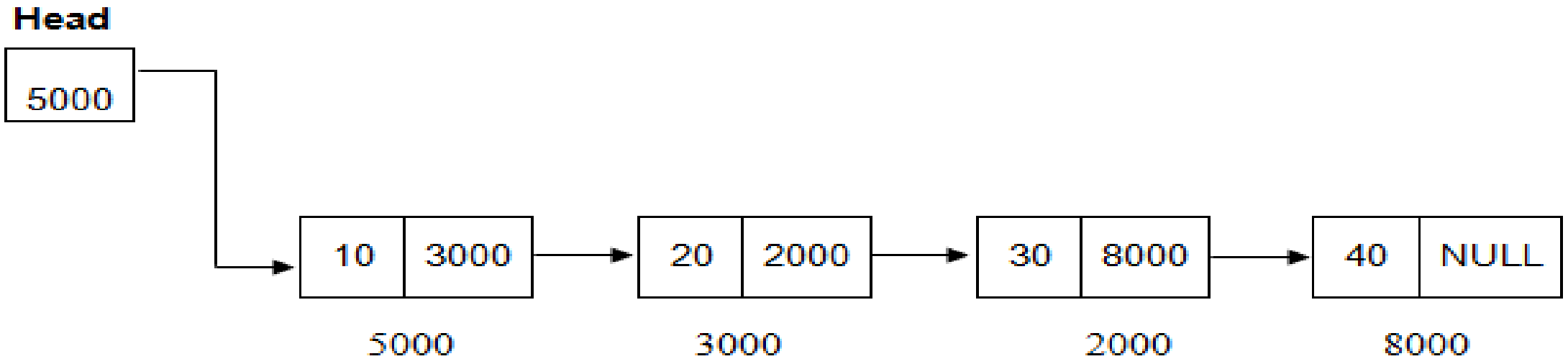
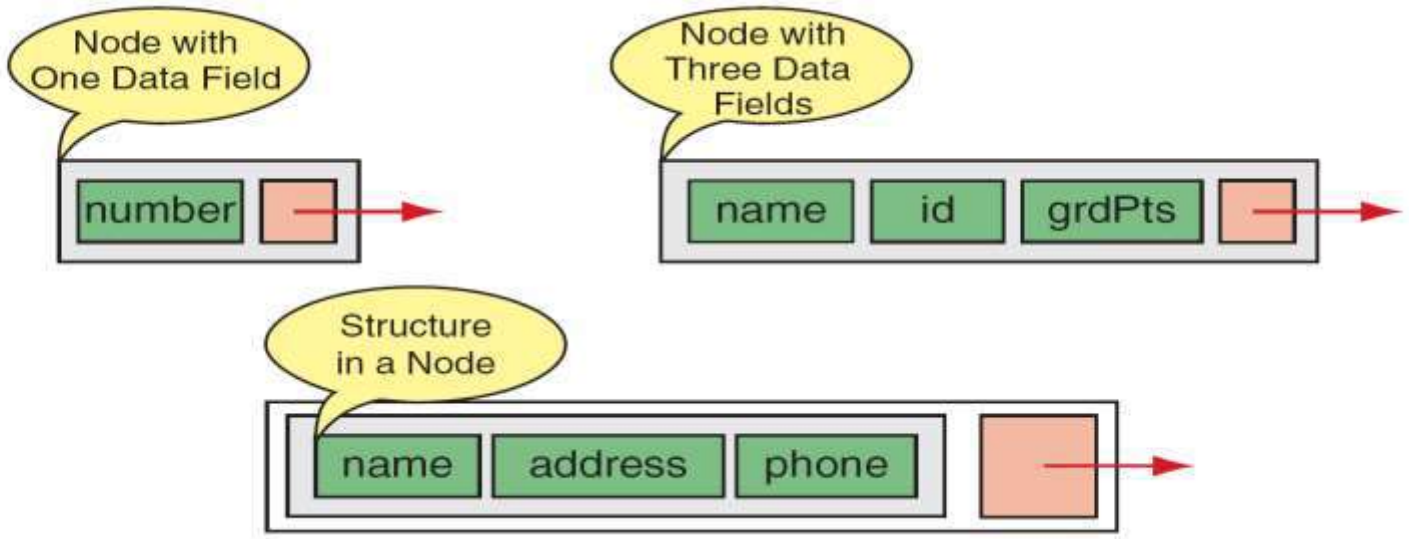


Figure: Singly Linked List with 4 Nodes

- There is an arrow from a node to the next node in the list.
- The pointer to the last node contains a special value called the ***null pointer*.**
- Linked list also contains a pointer variable, called ***HEAD*,** which points to the first node of the list.
- As nodes are added to a list, memory for a node is dynamically allocated. Thus the number of items that may be added to a list is limited only by the amount of memory available.
- If HEAD is NULL then list is said to be ***empty list*** or ***null list.***

- The data part in a node can be single field or multiple fields but the next part should be single field.

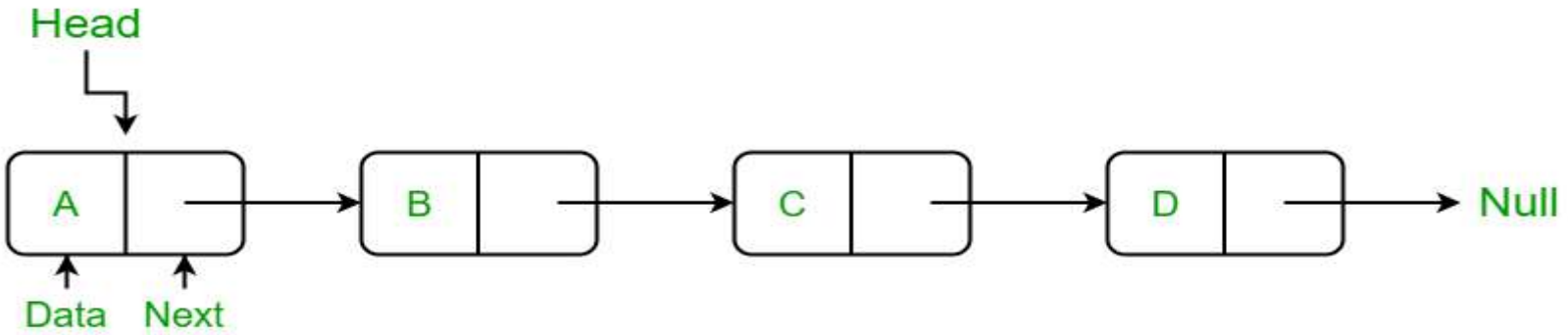


Singly Linked ListINTRODUCTION

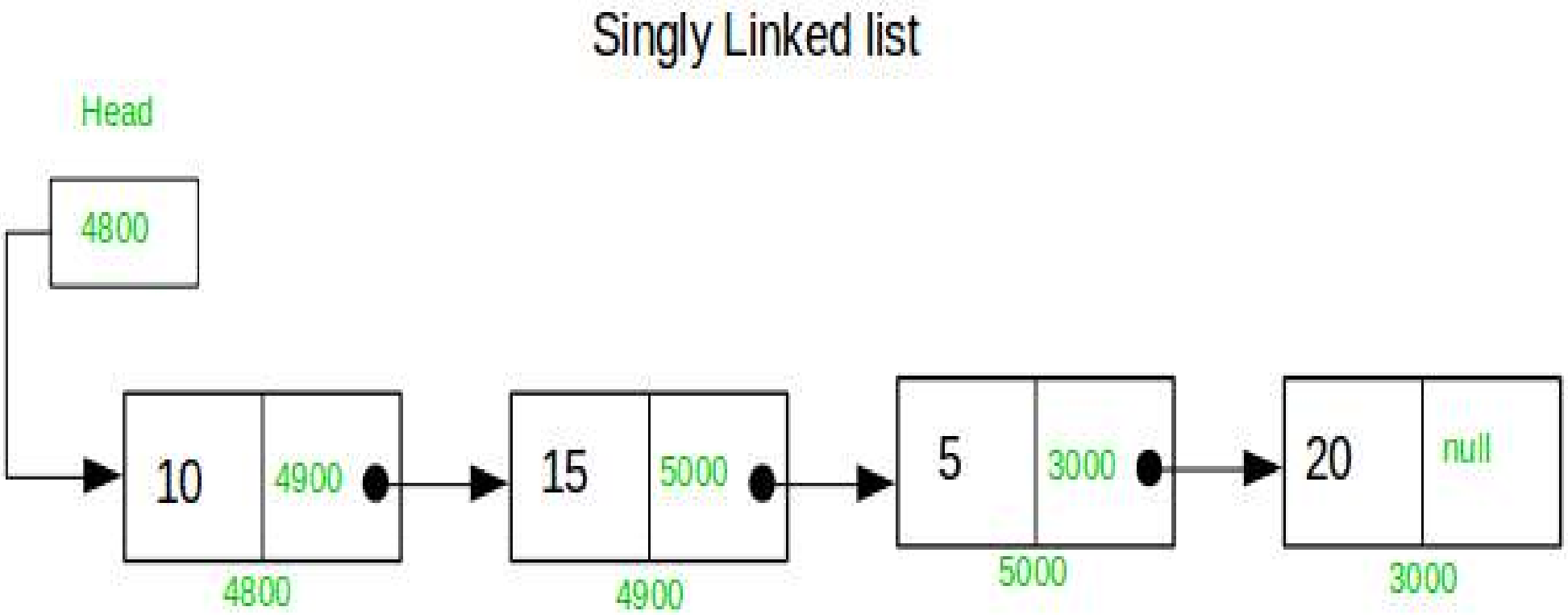
➢ A **singly linked list** is a linear data structure in which the elements are not stored in contiguous memory locations.

➢Each element is connected only to its next element using a pointer.

➢ Every node points to its next node in the sequence and the last node points NULL.



Example of Singly Linked list with 4 nodes



OPERATIONS ON SINGLY LINKED LIST

In a singly linked list, we can perform the following operations...

1. Insertion
2. Deletion
3. Display

1. Insertion:

i) Insertion At Beginning of the SLL
ii) Insertion At End of the SLL
iii) Insertion At Specific location in the SLL

2. Deletion:

i) Deletion At Beginning of the SLL
ii) Deletion At End of the SLL
iii) Deletion At Specific location in the SLL

Setup the Singly Linked List

Before we implement those operations, first we need to **setup empty list**.

- **Perform the following steps to setup Empty SLL:**.

Step 1 - Define a **Node** structure with two members/fields **data** and **next**

Step 2 - Define a Node pointer '**head**' and set it to **NULL**.

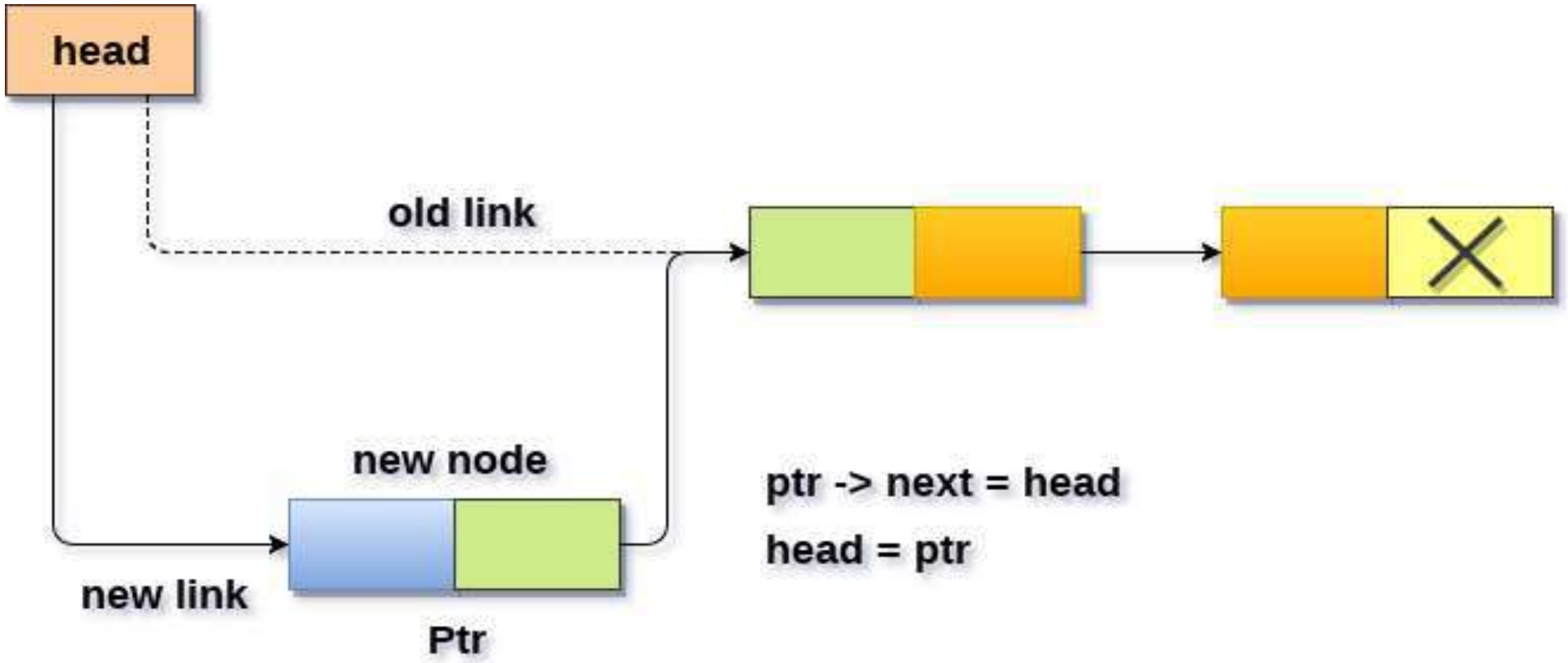
Step 3 – Then perform the Operations one by one.

The node is defined (represented) as:

```
struct Node
{
        int  data;
        struct Node  * next;
};
 struct Node *head=NULL;
```

i) Insertion At Beginning of the SLL

It involves inserting any element at the front of the list. We just need to a few link adjustments to make the new node as the head of the list.



i) Insertion At Beginning of the SLL

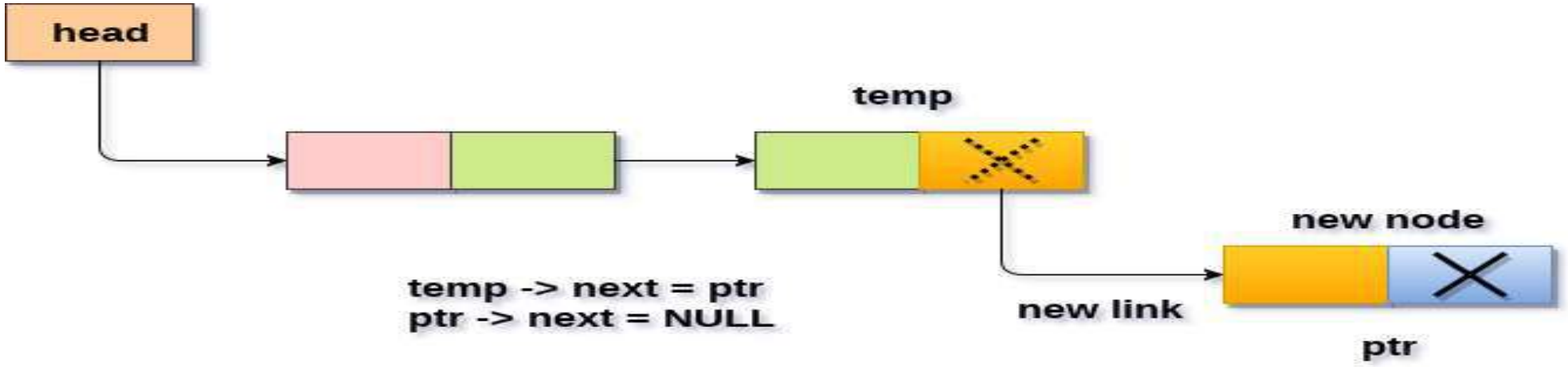
We can insert a new node at beginning(before head node) of the Singly linked list.

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

ii) Insertion At End of the SLL

It involves insertion at the last of the linked list. The new node can be inserted as the only node in the list or it can be inserted as the last one. Different logics are implemented in each scenario.



Inserting node at the last into a non-empty list

ii) Insertion At End of the SLL

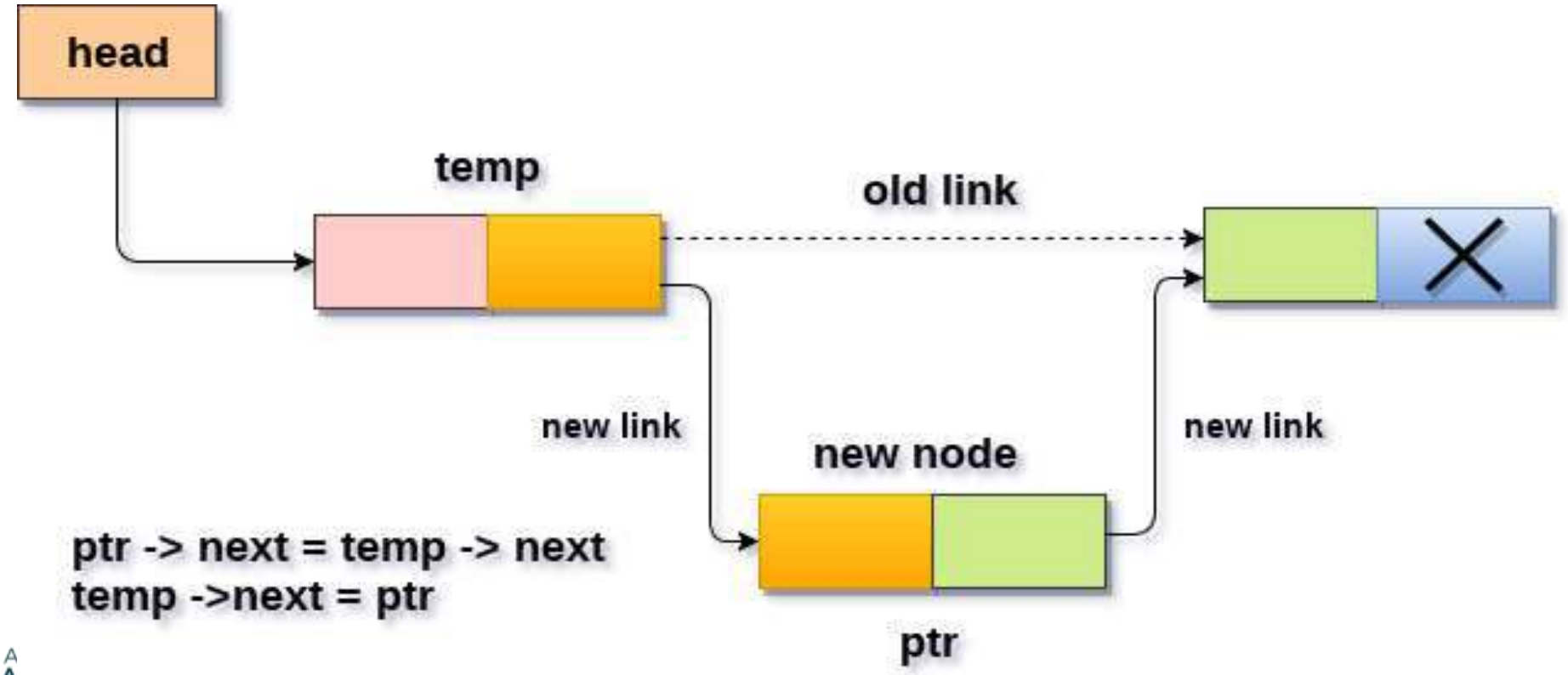
We can insert a new node at end(after tail/null node) of the singly linked list.

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

iii) Insertion At Specific location in the SLL

It involves insertion after the specified node of the linked list. We need to skip the desired number of nodes in order to reach the node after which the new node will be inserted.



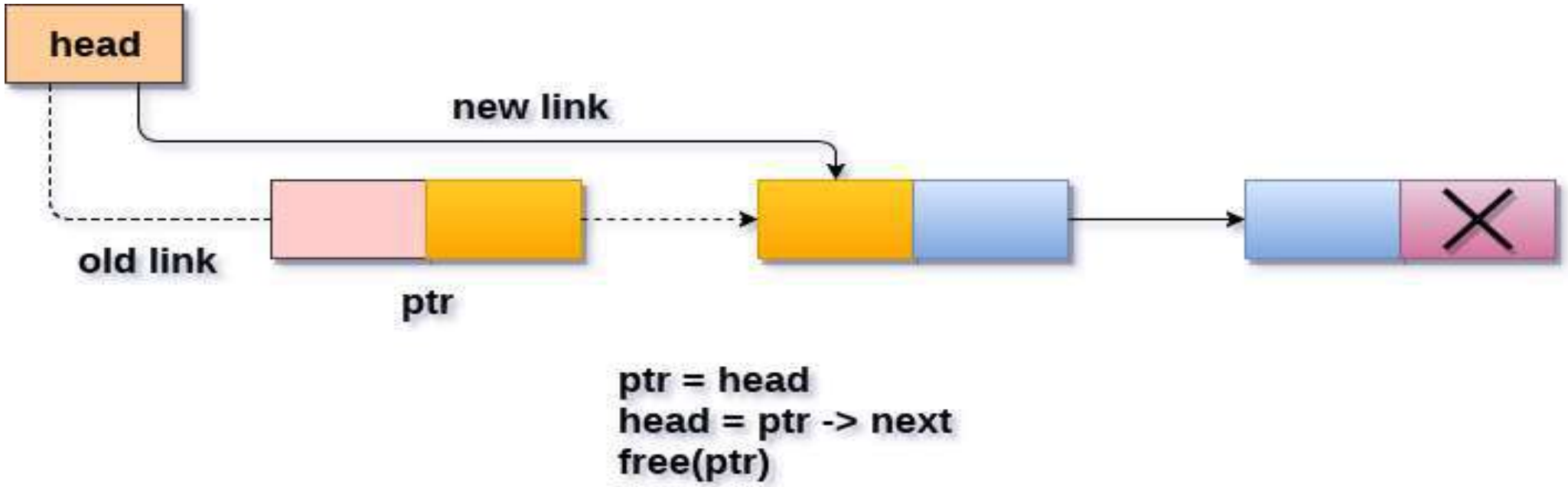
iii) Insertion At Specific location of the SLL

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

i) Deletion At Beginning of the SLL

It involves deletion of a node from the beginning of the list. This is the simplest operation among all. It just need a few adjustments in the node pointers.



Deleting a node from the beginning

i) Deletion At Beginning of the SLL

We can delete a node from the beginning of the list.

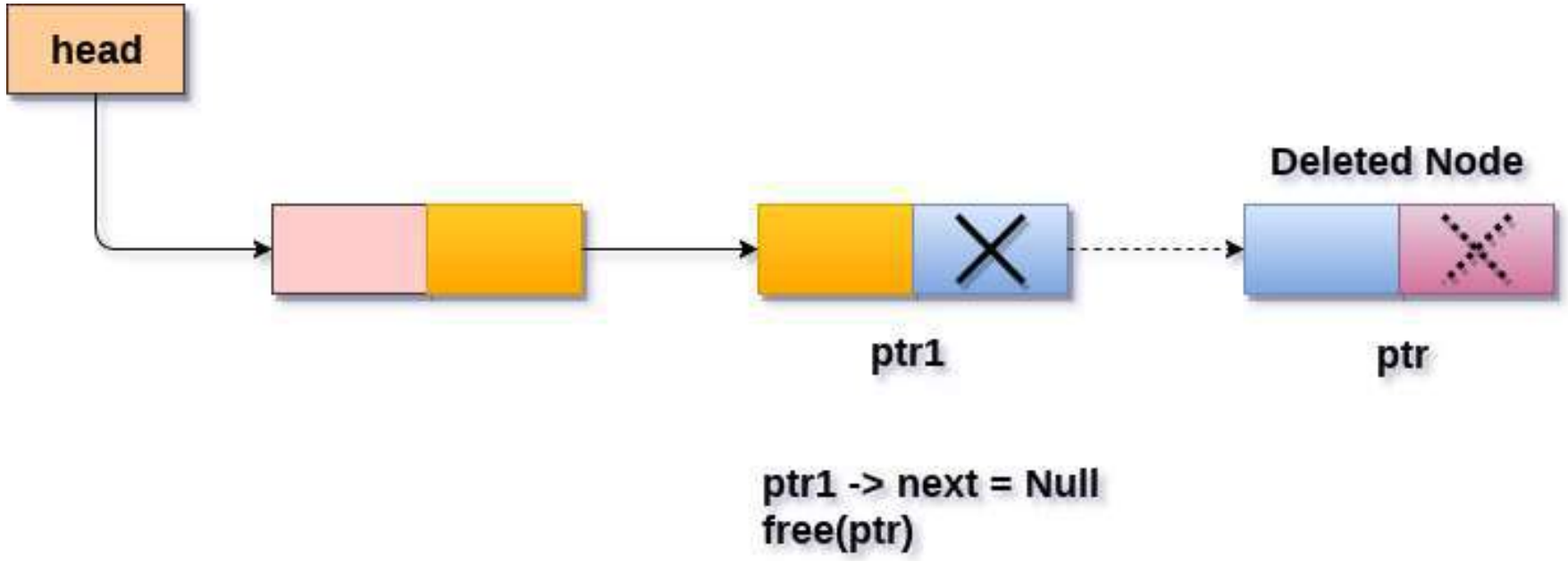
Algorithm:

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

ii) Deletion At End of the SLL

It involves deleting the last node of the list. The list can either be empty or full. Different logic is implemented for the different scenarios.



Deleting a node from the last

ii) Deletion At End of the SLL

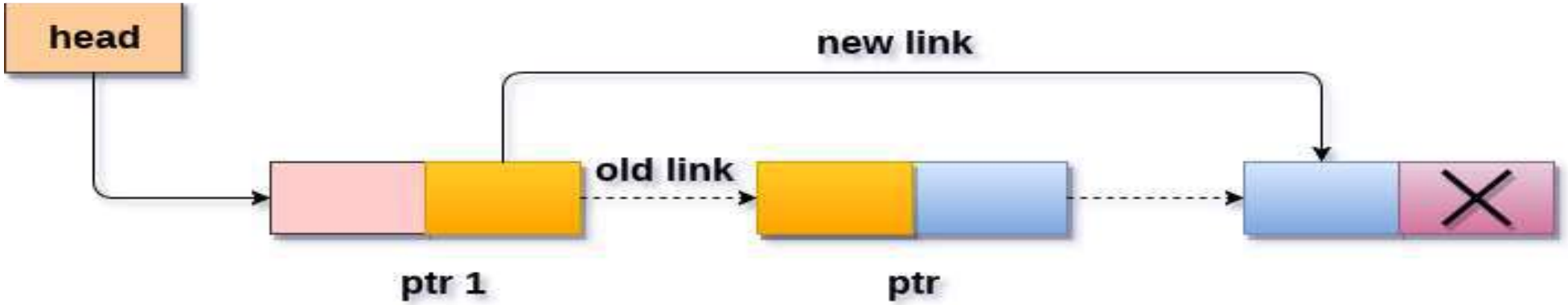
We can delete a node at end of the singly linked list.

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

iii) Deletion At Specific location in the SLL

It involves deleting the node after the specified node in the list. we need to skip the desired number of nodes to reach the node after which the node will be deleted. This requires traversing through the list.




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

Deletion a node from specified position

iii) Deletion At Specific location of the SLL

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: 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

Display Operation in Singly Linked List

We can use the following steps to display the elements of a single linked list

Algorithm:

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last node

Step 5 - Finally display temp → data with arrow pointing to NULL (temp → data ---> NULL).

SUMMARY

Singly linked list is a basic linked list type. Singly linked list is a collection of nodes linked together in a sequential way where each node of singly linked list contains a data field and an address field which contains the reference of the next node. Singly linked list can contain multiple data fields but should contain at least single address field pointing to its connected next node

SELF-ASSESSMENT QUESTIONS

A linear collection of data elements where the linear node is given by means of pointer is called?

a) Linked list

b) Node list

c) Primitive list

d) Unordered list

In linked list each node contains a minimum of two fields. One field is data field to store the data second field is??

a) Pointer to character

b) Pointer to integer

c) Pointer to node

d) Node

TERMINAL QUESTIONS

1.How will you represent a linked list in a graphical view?

2.Explain Singly Linked List in short.

3.How many pointers are necessary to implement a simple Linked List?

4.Which type of memory allocation is referred for Linked List

5.What are the main differences between the Linked List and Linear Array?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. Mark Allen Weiss, Data Structures and Algorithm Analysis in C, 2010 , Second Edition, PearsonEducation.
2. 2. Ellis Horowitz, Fundamentals of Data Structures in C: Second Edition, 2015
3. A.V.Aho, J. E. Hopcroft, and J. D. Ullman, “Data Structures And Algorithms”, Pearson Education, First Edition

   Reprint2003.

Sites and Web links:

1. https://nptel.ac.in/courses/106102064
2. https://in.udacity.com/course/intro-to-algorithms--cs215
3. https://www.coursera.org/learn/data-structures?action=enroll

(DEEMED TO BE UNIVERSITY)

THANK YOU