Linked list implementation – Singly linked lists

Linked Lists Versus Arrays

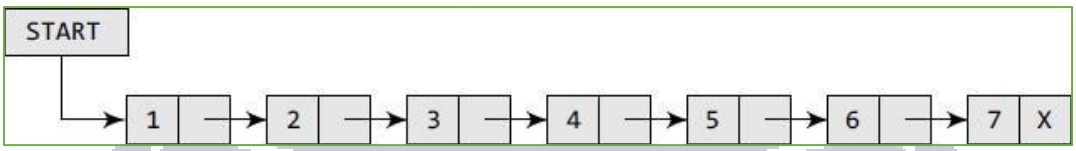
Arrays and linked lists are a linear collection of data elements

Array	Linked Lists
It stores its nodes in consecutive memory locations	It does not store its nodes in consecutive memory locations
It allows random access of data	It does not allow random access of data. Nodes in a linked list can be accessed only in a sequential manner
It cannot add any number of elements in the array	It can add any number of elements in the list

SINGLY LINKED LISTS

Definition

A singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. By saying that the node contains a pointer to the next node, we mean that the node stores the address of the next node in sequence. A singly linked list allows traversal of data only in one way



Traversing a Linked List

Accessing the nodes of the list in order to perform some processing on them. Remember a linked list always contains a pointer variable START which stores the address of the first node of the list. End of the list is marked by storing NULL or –1 in the NEXT field of the last node. For traversing the linked list, we also make use of another pointer variable PTR which points to the node that is currently being accessed.

Algorithm for traversing a linked list

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:            Apply Process to PTR->DATA
Step 4:            SET PTR = PTR->NEXT
       [END OF LOOP]
Step 5: EXIT
```

- In this algorithm, we first initialize PTR with the address of START. So now, PTR points to the first node of the linked list.
- Then in Step 2, a while loop is executed which is repeated till PTR processes the last node, that is until it encounters NULL.
- In Step 3, we apply the process (e.g., print) to the current node, that is, the node pointed by PTR.
- In Step 4, we move to the next node by making the PTR variable point to the node whose address is stored in the NEXT field.

Algorithm to print the number of nodes in a linked list

```
Step 1: [INITIALIZE] SET COUNT = 0
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4:            SET COUNT = COUNT + 1
Step 5:            SET PTR = PTR->NEXT
       [END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT
```

We will traverse each and every node of the list and while traversing every individual node, we will increment the counter by 1. Once we reach NULL, that is, when all the nodes of the linked list have been traversed, the final value of the counter will be displayed.

Searching for a Value in a Linked List

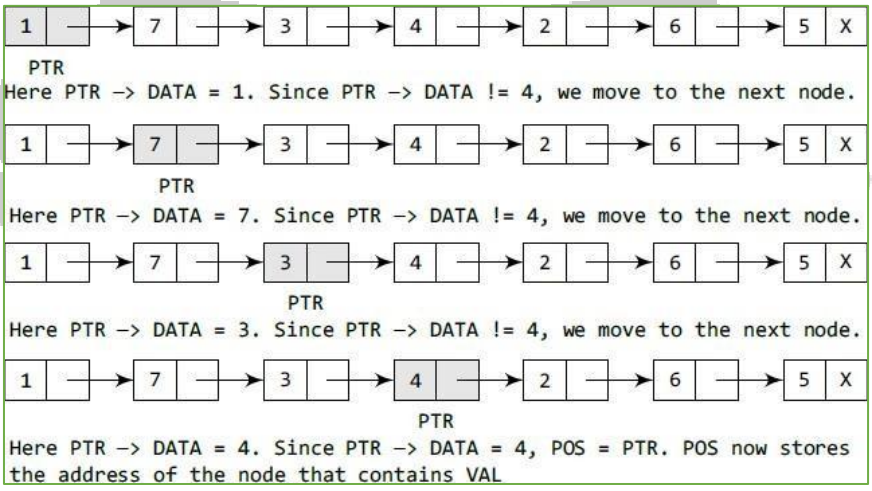
Searching a linked list means finding whether a given value is present in the information part of the node or not. If it is present, the algorithm returns the address of the node that contains the value.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:     IF VAL = PTR -> DATA
                SET POS = PTR
                Go To Step 5
            ELSE
                SET PTR = PTR -> NEXT
            [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

- In Step 1, we initialize the pointer variable PTR with START that contains the address of the first node.
- In Step 2, a while loop is executed which will compare every node's DATA with VAL for which the search is being made.
- If the search is successful, that is, VAL has been found, then the address of that node is stored in POS and the control jumps to the last statement of the algorithm.
- However, if the search is unsuccessful, POS is set to NULL which indicates that VAL is not present in the linked list.

Example: Illustration of Searching algorithm

Consider the linked list shown in Figure. If we have VAL = 4, then the flow of the algorithm can be explained as shown in the figure.



Inserting a New Node in a Linked List

How a new node is added into an already existing linked list? We will take four cases and then see how insertion is done in each case.

Case 1: The new node is inserted at the beginning.

Case 2: The new node is inserted at the end.

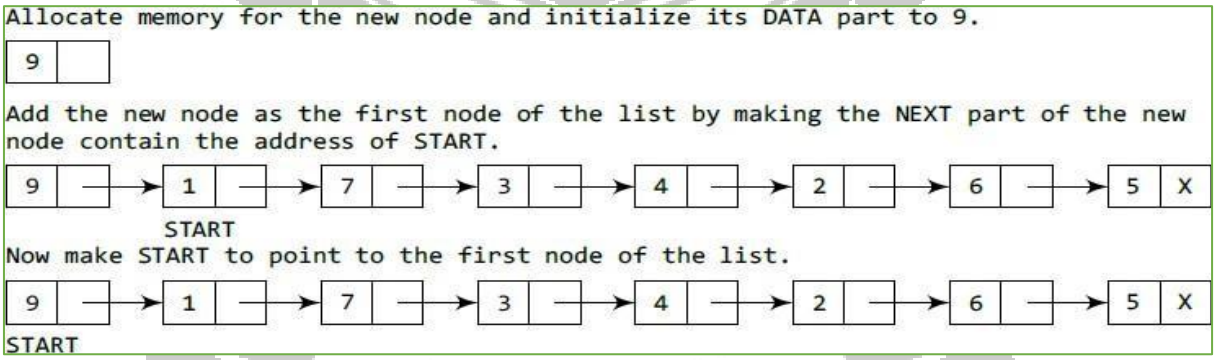
Case 3: The new node is inserted after a given node.

Case 4: The new node is inserted before a given node. OVERFLOW

Overflow is a condition that occurs when AVAIL = NULL or no free memory cell is present in the system. When this condition occurs, the program must give an appropriate message.

Case 1: Inserting a Node at the Beginning of a Linked List

Add a new node with data 9 and add it as the first node of the list.



Algorithm to insert a new node at the beginning

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [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 NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

- In Step 1, we first check whether memory is available for the new node. If the free memory has exhausted, then an OVERFLOW message is printed.
- Otherwise, if a free memory cell is available, then we allocate space for the new node. Set its DATA part with the given VAL and the next part is initialized with the address of the first node of the list, which is stored in START.
- Now, since the new node is added as the first node of the list, it will now be known as the START node, that is, the START pointer variable will now hold the address of the NEW_NODE.

Note the following two steps:

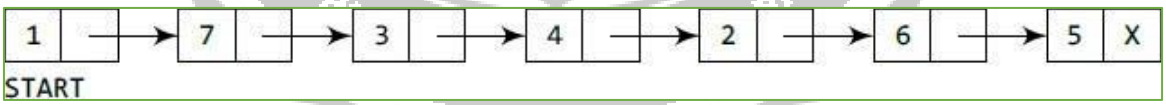
Step 2: SET NEW_NODE = AVAIL

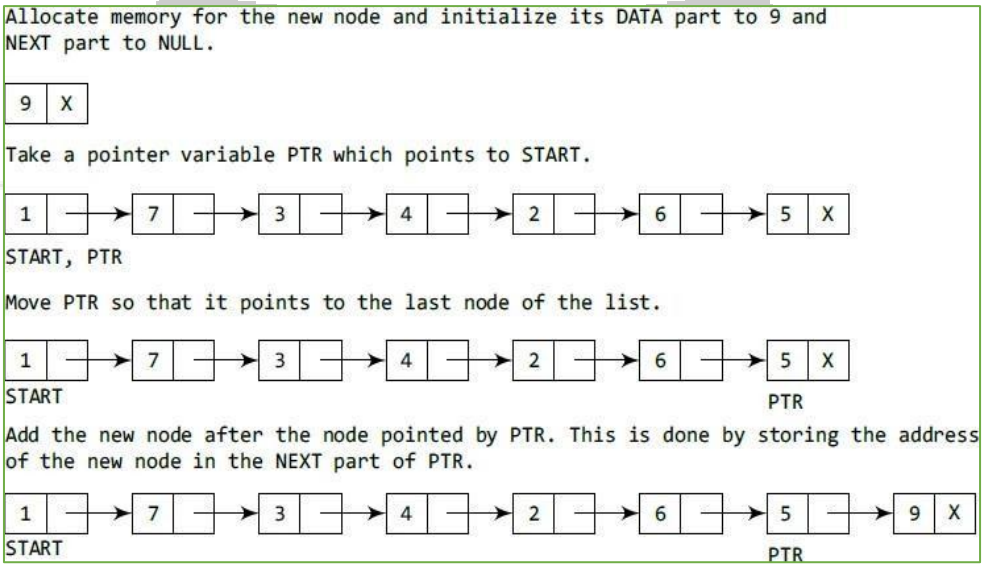
Step 3: SET AVAIL = AVAIL -> NEXT

These steps allocate memory for the new node. In C, there are functions like malloc(), alloc, and calloc() which automatically do the memory allocation on behalf of the user.

Case 2: Inserting a Node at the End of a Linked List

Add a new node with data 9 as the last node of the list





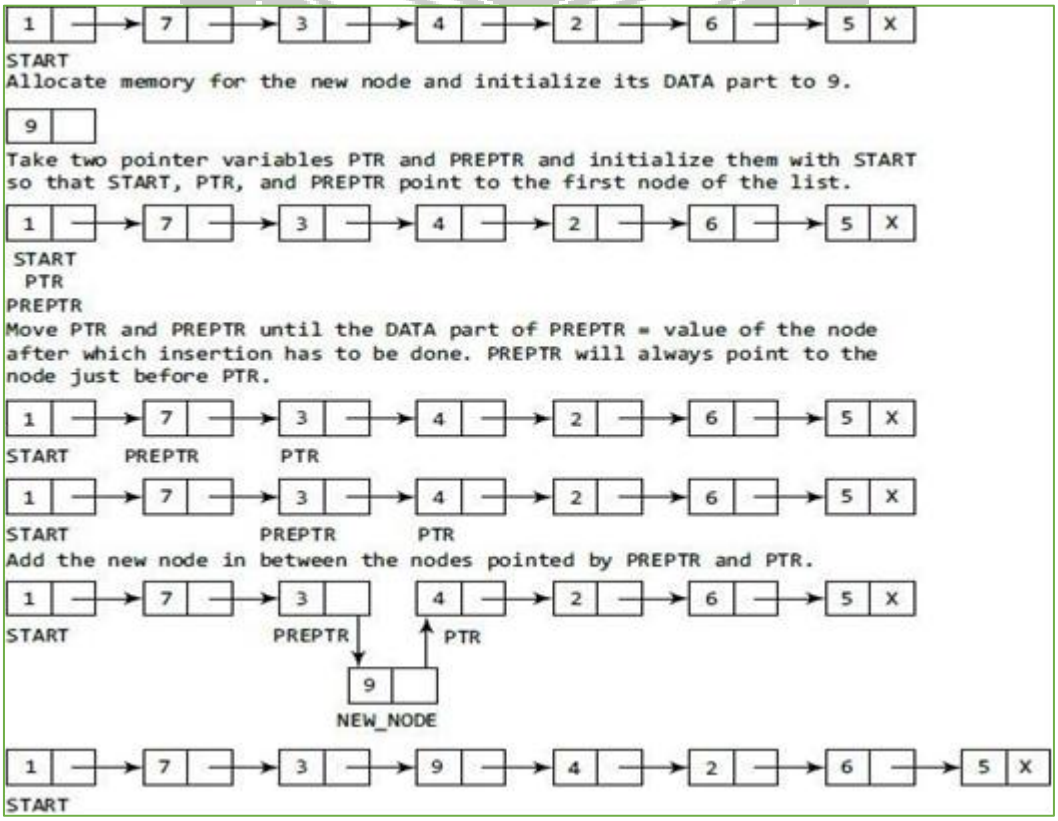
Algorithm to insert a new node at the end

```
Step 1: IF AVAIL = NULL
            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 NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
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
```

This algorithm to insert a new node at the end of a linked list. In Step 6, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list. In the while loop, we traverse through the linked list to reach the last node. Once we reach the last node, in Step 9, we change the NEXT pointer of the last node to store the address of the new node. Remember that the NEXT field of the new node contains NULL, which signifies the end of the linked list.

Case 3: Inserting a Node After a Given Node in a Linked List

Add a new node with value 9 after the node containing data 3.



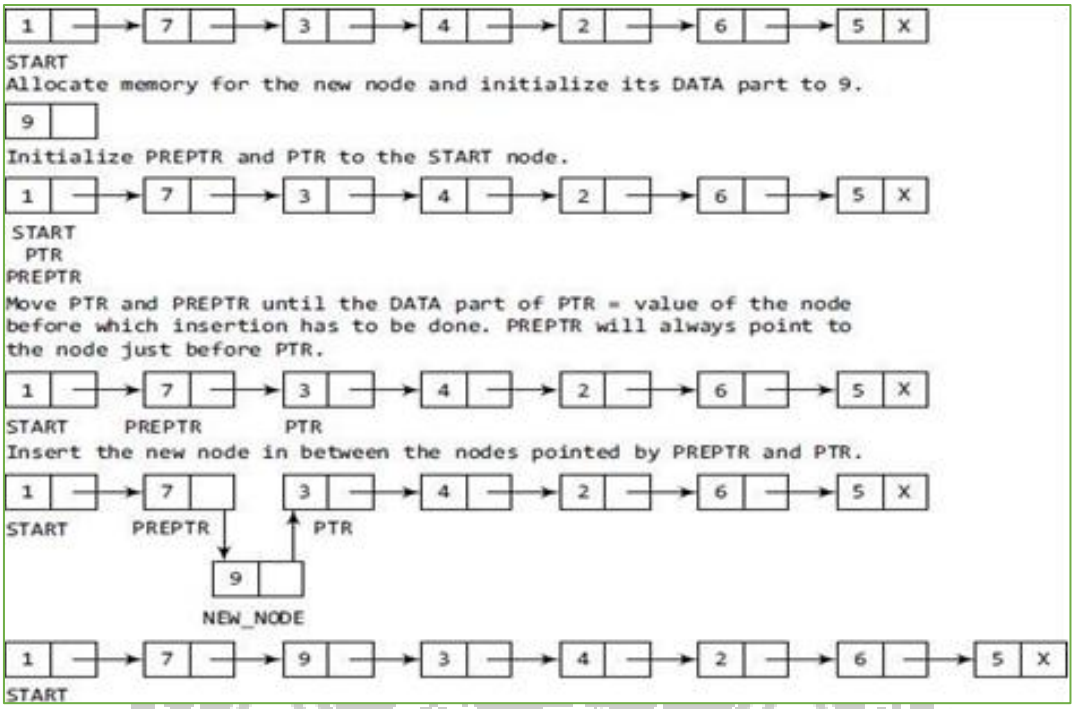
Algorithm to insert a new node after a node that has value NUM

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [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: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA
         != NUM
Step 8:      SET PREPTR = PTR
Step 9:      SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step 12: EXIT
```

- In Step 5, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list.
- Then we take another pointer variable PREPTR which will be used to store the address of the node preceding PTR. Initially, PREPTR is initialized to PTR.
- So now, PTR, PREPTR, and START are all pointing to the first node of the linked list.
- In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM. We need to reach this node because the new node will be inserted after this node.
- Once we reach this node, in Steps 10 and 11, we change the NEXT pointers in such a way that new node is inserted after the desired node.

Case 4: Inserting a Node Before a Given Node in a Linked List

Add a new node with value 9 before the node containing 3.



Algorithm to insert a new node before a node that has value NUM

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [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: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

- In Step 5, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list.
- Then, we take another pointer variable PREPTR and initialize it with PTR. So now, PTR, PREPTR, and START are all pointing to the first node of the linked list.
- In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM.
- We need to reach this node because the new node will be inserted before this node.

- Once we reach this node, in Steps 10 and 11, we change the NEXT pointers in such a way that the new node is inserted before the desired node.

Deleting a Node from a Linked List

We will discuss how a node is deleted from an already existing linked list. We will consider three cases and then see how deletion is done in each case.

Case 1: The first node is deleted.

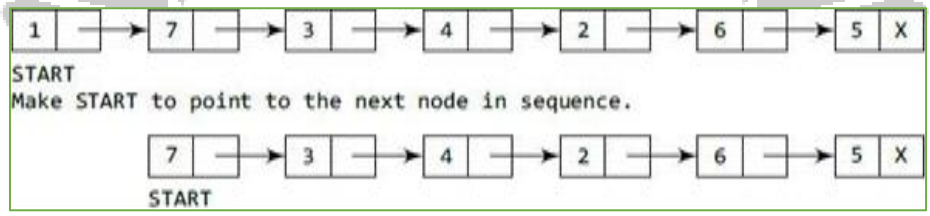
Case 2: The last node is deleted.

Case 3: The node after a given node is deleted.

- Underflow 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.
- Note that when we delete a node from a linked list, we actually have to free the memory occupied by that node.
- The memory is returned to the free pool so that it can be used to store other programs and data.
- Whatever be the case of deletion, we always change the AVAIL pointer so that it points to the address that has been recently vacated.

Case 1: The first node is deleted.

When we want to delete a node from the beginning of the list, then the following changes will be done in the linked list.



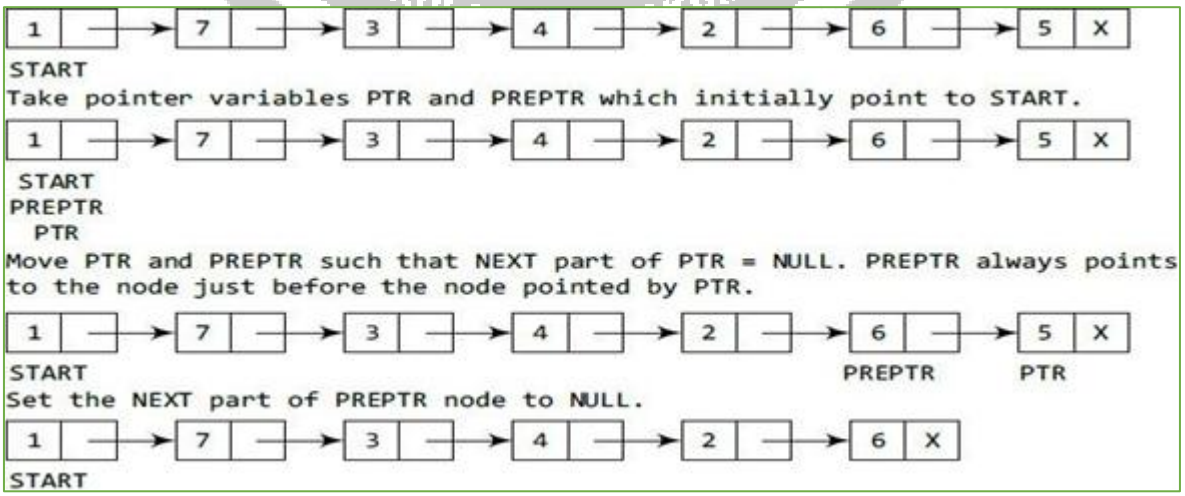
Algorithm to delete the first node

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

- In Step 1, we check if the linked list exists or not. If START = NULL, then it signifies that there are no nodes in the list and the control is transferred to the last statement of the algorithm.
- However, if there are nodes in the linked list, then we use a pointer variable PTR that is set to point to the first node of the list.
- For this, we initialize PTR with START that stores the address of the first node of the list.
- In Step 3, START is made to point to the next node in sequence and finally the memory occupied by the node pointed by PTR (initially the first node of the list) is freed and returned to the free pool.

Case 2: The last node is deleted.

We want to delete the last node from the linked list, then the following changes will be done in the linked list.



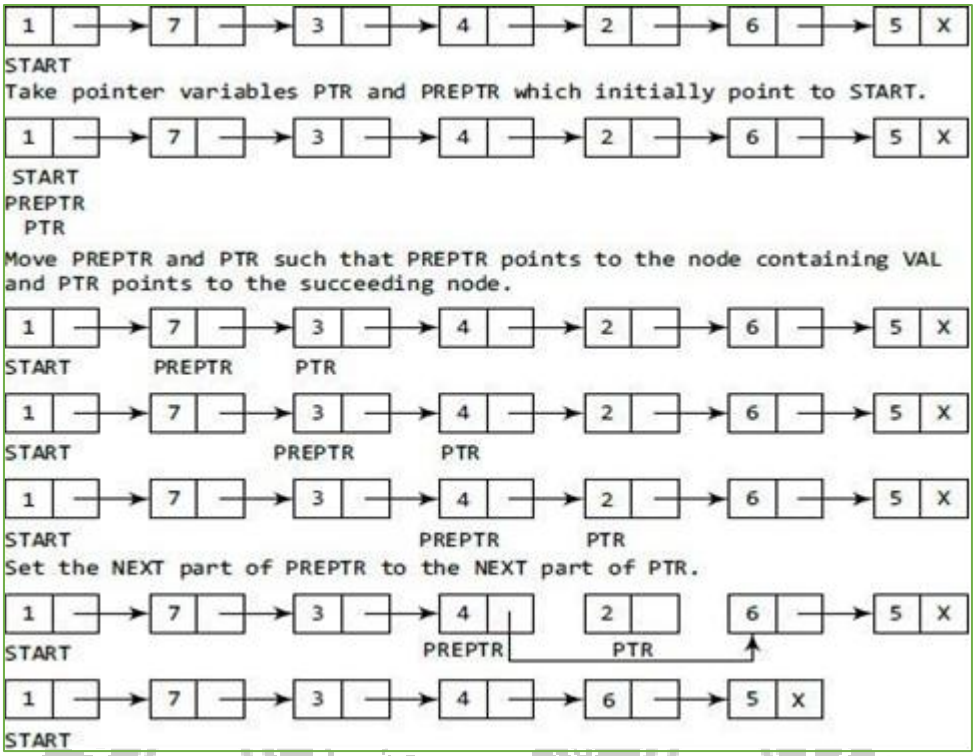
Algorithm to delete the last node

```
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
```

- In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list.
- In the while loop, we take another pointer variable PREPTR such that it always points to one node before the PTR.
- Once we reach the last node and the second last node, we set the NEXT pointer of the second last node to NULL, so that it now becomes the (new) last node of the linked list.
- The memory of the previous last node is freed and returned back to the free pool.

Case 3: The node after a given node is deleted.

We want to delete the node that succeeds the node which contains data value 4. Then the following changes will be done in the linked list.



Algorithm to delete the node after a given node

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

- In Step 2, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list. In the while loop, we take another pointer variable PREPTR such that it always points to one node before the PTR.
- Once we reach the node containing VAL and the node succeeding it, we set the next pointer of the node containing VAL to the address contained in next field of the node succeeding it.
- The memory of the node succeeding the given node is freed and returned back to the free pool.