Unit-3

What is linked list?

- Simply a list is a sequence of data, and linked list is a sequence of data linked with each other.
- When we want to work with unknown number of data values, we use a linked list data structure to organize that data.
- Linked list is a **linear data structure** that **contains sequence of elements** such that each element links to its next element in the sequence.
- Each element in a linked list is called as "**Node**".

Advantages of linked lists :

1. Linked lists are dynamic data structures . i.e., they can grow or shrink during the execution of a program.
2. Linked lists have efficient memory utilization. Here, memory is not pre- allocated. Memory is allocated whenever it is required and it is de- allocated (removed) when it is no longer needed.
3. Insertion and Deletions are easier and efficient . Linked lists provide flexibility in inserting a data item at a specified position and deletion of the data item from the given position.

Disadvantages of linked lists:

1. It consumes more space because every node requires a additional pointer to store address of the next node.
2. Searching a particular element in list is difficult and also time consuming.

Types of Linked List:

Following are the various types of linked list.

1. **Singly Linked List (Uni-directional)**

2. **Doubly Linked List (Bi-directional)**

3. **Circular Linked List**

What is Single Linked List?

In any single linked list, the individual element is called as "**Node**". Every "**Node**" contains two fields, **data** and **next**. The **data** field is used to store actual value of that node and next field is used to store the address of the next node in the sequence.

The graphical representation of a node in a single linked list is as follows...

The formal definition of a single linked list is as follows...

Single linked list is a sequence of elements in which every element has link to its next element in the sequence.

NOTE

- In a single linked list, the address of the first node is always stored in a reference node known as "front" (Some times it is also known as "head").
- Always next part (reference part) of the last node must be NULL.

- **Next field** is also called **address field**

Example

Fig: Singly Linked List

Basic structure of a singly linked list

- Each node of a singly linked list follows a common basic structure. In a node we can store more than one data fields but we need at least single address field to store the address of next connected node.

```
struct node
{
        int data;                   // Data
        struct node * next;         // Address
};
```

Operations

In a single linked list we perform the following operations.

1. **Display**
2. **Insertion**
3. **Deletion**

1. Display

Program 1

- **WAP in C to create a Singly linked list and display elements of linked list.**

or

- **WAP in C to create and traverse a Singly Linked List.**

Note: type this link for code.

https://codeforwin.org/2015/09/c-program-to-create-and-traverse-singly-linked-list.html

//C program to create and traverse a Linked List. (list1.cpp)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
/* Structure of a node */
struct node
{
   int data;        // Data
   struct node *next; // Address
}*head;
// Functions to create and display list
void createList(int n);
void traverseList();

void main()
{
   int n;
   clrscr();
   printf("Enter the total number of nodes: ");
   scanf("%d", &n);
   createList(n);
   printf("\nData in the list are: \n");
   traverseList();
   getch();
}
//Create a list of n nodes
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;
   head = (struct node *)malloc(sizeof(struct node));
   if(head == NULL)
   {
                  printf("Unable to allocate memory.");
                  exit(0);
   }
   // Input data of node from the user
   printf("Enter the data of node 1: ");
   scanf("%d", &data);
   head->data = data;//Link data field with data
   head->next = NULL;//Link address field to NULL
   // Create n - 1 nodes and add to list
   temp = head;
   for(i=2; i<=n; i++)
   {
                  newNode = (struct node *)malloc(sizeof(struct node));
                  /* If memory is not allocated for newNode */
                  if(newNode == NULL)
                  {
                     printf("Unable to allocate memory.");
                     break;
                  }
                  printf("Enter the data of node %d: ", i);
                  scanf("%d", &data);
                  newNode->data = data;//Link data field of newNode
                  newNode->next = NULL;//Make sure new node points to NULL
                  temp->next = newNode;//Link previous node with newNode
                  temp = temp->next;   //Make current node as previous node
   }
}
// Display entire list
void traverseList()
{
   struct node *temp;
   // Return if list is empty
   if(head == NULL)
   {
                  printf("List is empty.");
                  return;
   }
   temp = head;
   while(temp != NULL)
   {
                  printf("Data = %d\n", temp->data);//Print data of current node
                  temp = temp->next;              //Move to next node
   }
}
```

Output:

```
Enter the total number of nodes: 5
Enter the data of node 1: 100
Enter the data of node 2: 200
Enter the data of node 3: 300
Enter the data of node 4: 400
Enter the data of node 5: 500

Data in the list are:
Data = 100
Data = 200
Data = 300
Data = 400
Data = 500
```

2. Insertion

- In a single linked list, the insertion operation can be performed in three ways. They are as follows.

a. Inserting a node at the beginning of list.

b. Inserting a node at the end of list.

c. Inserting node at the middle (or at any position) of Singly Linked List.

- **How to insert a new node at the beginning of a Singly Linked List.**

a. Inserting a node at the beginning of list:

Algorithm:

Step 1: Create a **newNode** with given value.

Step 2: Check whether list is **Empty** (**head** == **NULL**)

Step 3: If it is **Empty** then, set **newNode→next** = **NULL** and **head** = **newNode**.

Step 4: If it is **Not Empty** then, set **newNode→next** = **head** and **head** = **newNode**.

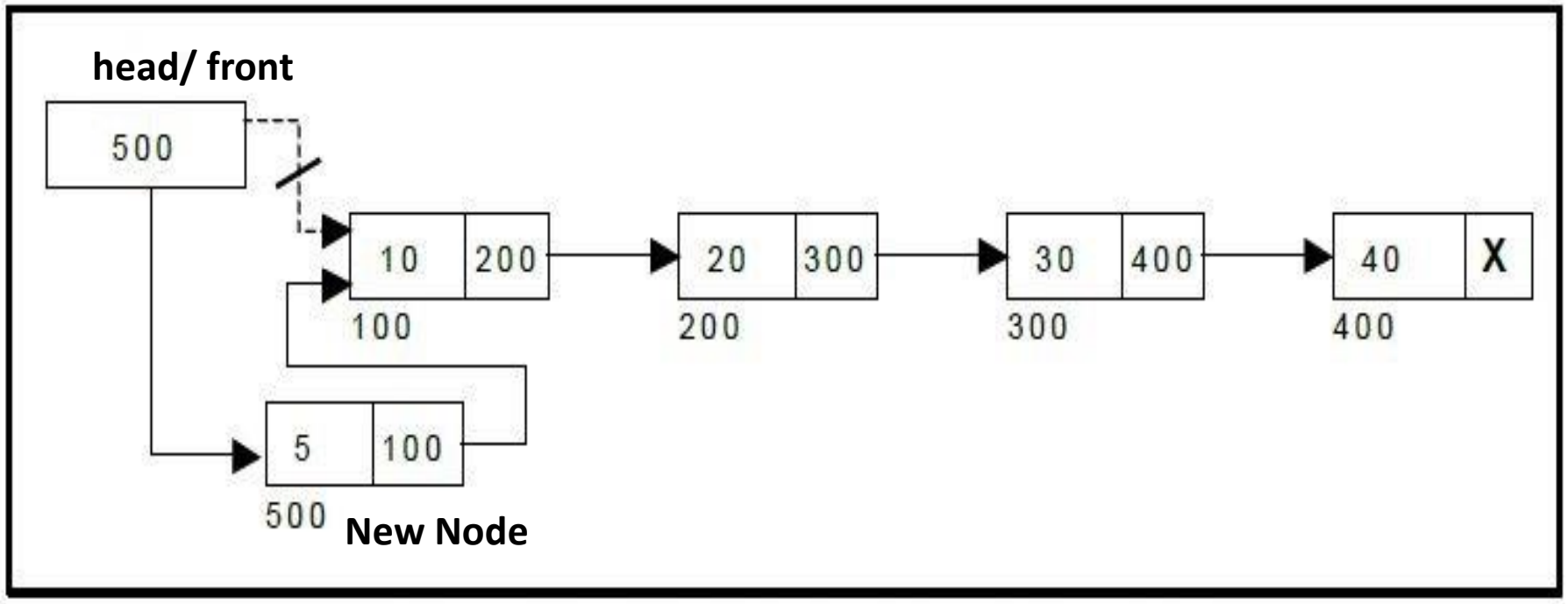
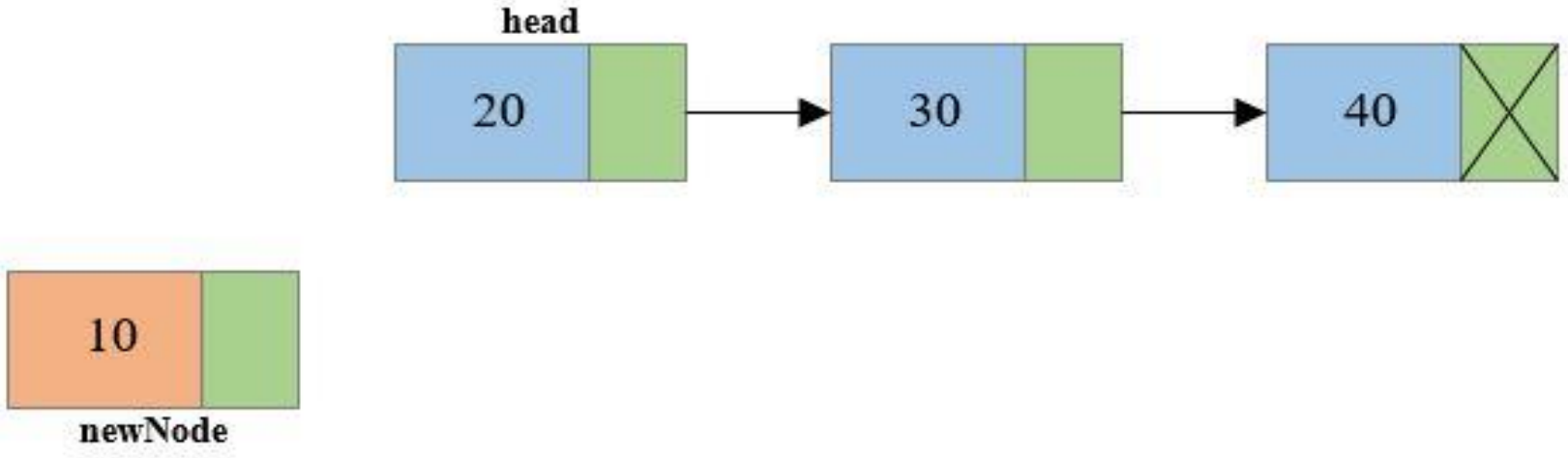


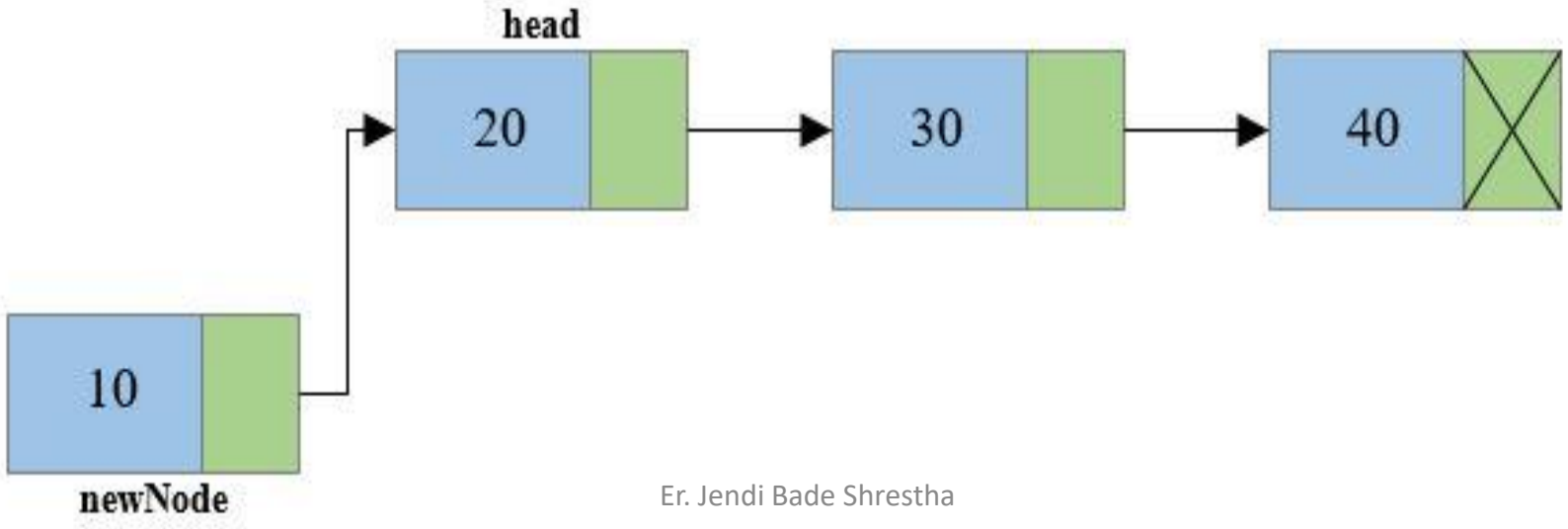
Figure → Inserting a node at the beginning

Steps to insert node at the beginning of singly linked list

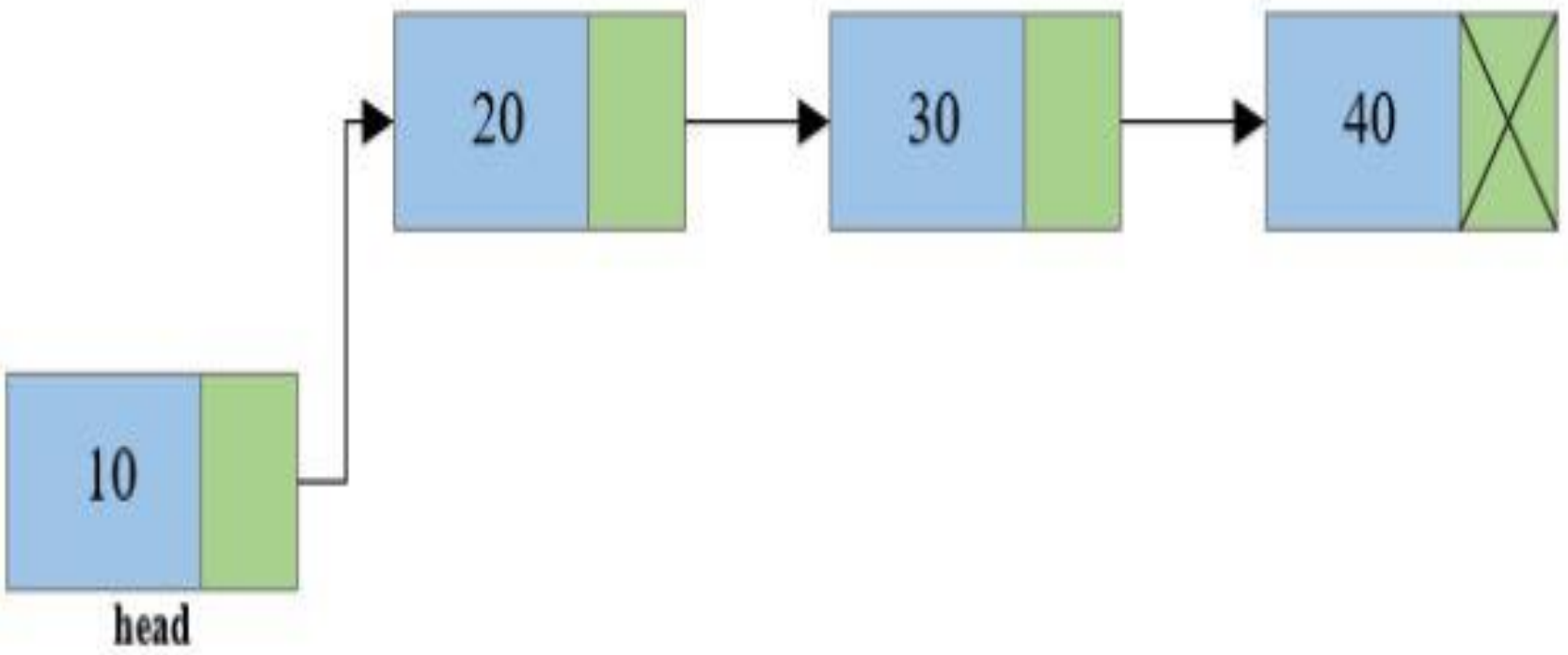
1. Create a new node, say newNode



2. Link the newly created node with the head node, i.e. the newNode will now point to head node.



3. Finally make the new node as the head node.



Program 2

WAP in C to create a singly linked list of n nodes and insert a new node in the beginning of the singly linked list.

Note: type this link for code.

https://codeforwin.org/2015/09/c-program-to-insert-node-at-beginning-of-singly-linked-list.html

```
1.  /* C program to insert a new node at the beginning
2.  of a Singly Linked List*/ (list2.cpp)
3.  #include <stdio.h>
4.  #include <conio.h>
5.  #include <stdlib.h>
6.  // Structure of a node
7.  struct node
8.  {
9.     int data;        // Data
10.    struct node *next; // Address
11. }*head;
12. void createList(int n);
13. void insertNodeAtBeginning(int data);
14. void displayList();
15. void main()
16. {
17.    int n, data;
18.    clrscr();
19.    // Create a singly linked list of n nodes
20.    printf("Enter the total number of nodes: ");
21.    scanf("%d", &n);
22.    createList(n);
23.    printf("\nData in the list \n");
24.    displayList();
25.    //Insert data at the beginning of the singly linked list
26.    printf("\nEnter data to insert at beginning of the list: ");
27.    scanf("%d", &data);
28.    insertNodeAtBeginning(data);
29.    printf("\nData in the list \n");
30.    displayList();
31.    getch();
32. }
33. //Create a list of n nodes
34. void createList(int n)
35. {
36.    struct node *newNode, *temp;
37.    int data, i;
38.    head = (struct node *)malloc(sizeof(struct node));
39.    // If unable to allocate memory for head node
40.    if(head == NULL)
41.    {
42.       printf("Unable to allocate memory.");
43.    }
44.    else
45.    {
46.       //Input data of node from the user
47.       printf("Enter the data of node 1: ");
48.       scanf("%d", &data);
49.       head->data = data; // Link data field with data
50.       head->next = NULL; // Link address field to NULL
51.       temp = head;
52.       // Create n nodes and adds to linked list
53.       for(i=2; i<=n; i++)
54.       {
55.          newNode = (struct node *)malloc(sizeof(struct node));
56.          // If memory is not allocated for newNode */
57.          if(newNode == NULL)
58.          {
59.                printf("Unable to allocate memory.");
60.                break;
61.          }
62.          else
63.          {
64.                printf("Enter the data of node %d: ", i);
65.                scanf("%d", &data);
66.                newNode->data = data; // Link data field of
   newNode with data
67.                newNode->next = NULL; // Link address field
   of newNode with NULL
68.                temp->next = newNode; // Link previous node
   i.e. temp to the newNode
69.                temp = temp->next;
70.          }
71.       }
72.       printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");
73.    }
74. }
```

```
//Create a new node and inserts at the beginning of the linked list.
void insertNodeAtBeginning(int data)
{
    struct node *newNode;
    newNode = (struct node*)malloc(sizeof(struct node));
    if(newNode == NULL)
    {
        printf("Unable to allocate memory.");
    }
    else
    {
        newNode->data = data; // Link data part
        newNode->next = head; // Link address part
        head = newNode;        // Make newNode as first node
        printf("DATA INSERTED SUCCESSFULLY\n");
    }
}
//Display entire list
void displayList()
{
    struct node *temp;
    //If the list is empty i.e. head = NULL
    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
            printf("Data = %d\n", temp->data); // Print data of current node
            temp = temp->next;              // Move to next node
        }
    }
}
```

Output:

```
Enter the total number of nodes: 4
Enter the data of node 1: 200
Enter the data of node 2: 300
Enter the data of node 3: 400
Enter the data of node 4: 500
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 200
Data = 300
Data = 400
Data = 500

Enter data to insert at beginning of the list: 100
DATA INSERTED SUCCESSFULLY

Data in the list
Data = 100
Data = 200
Data = 300
Data = 400
Data = 500
```

- **How to insert a new node at the end of a Singly Linked List.**

b. Inserting a node at the end of list.

Algorithm: [METHOD 1]

Step1: Create a **newNode** with given value.

Step2: Check whether list is **Empty (head==NULL)**

Step3: If it is **Empty** then, Set **newNode -> next =NULL** and **head = newNode.**

Step 4: If it is **Not Empty** then, Traverse to the last node of the linked list and connect the last node of the list with the new node, i.e. last node will now point to new node. **(lastNode ->next = newNode, newnode->next = null).**

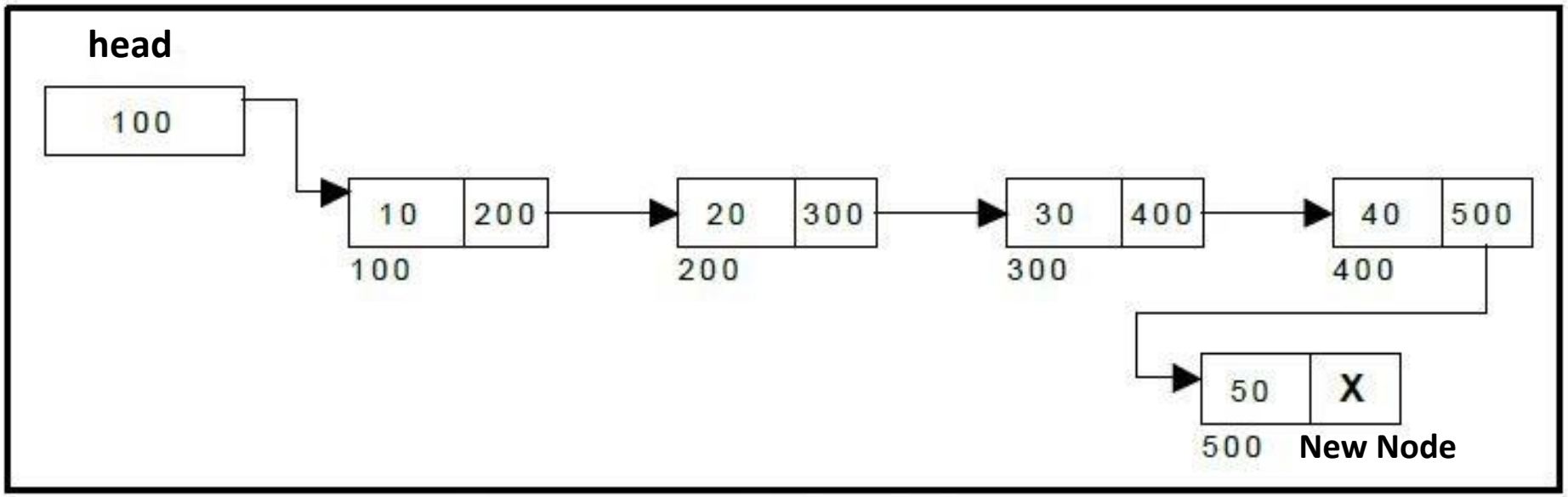


Figure → Inserting a node at the end.

b. Inserting a node at the end of list.

Algorithm: [METHOD 2]

Step 1: Create a **newNode** with given value and **newNode → next** as **NULL**.

Step 2: Check whether list is **Empty** (**head** == **NULL**).

Step 3: If it is **Empty** then, set **head** = **newNode**.

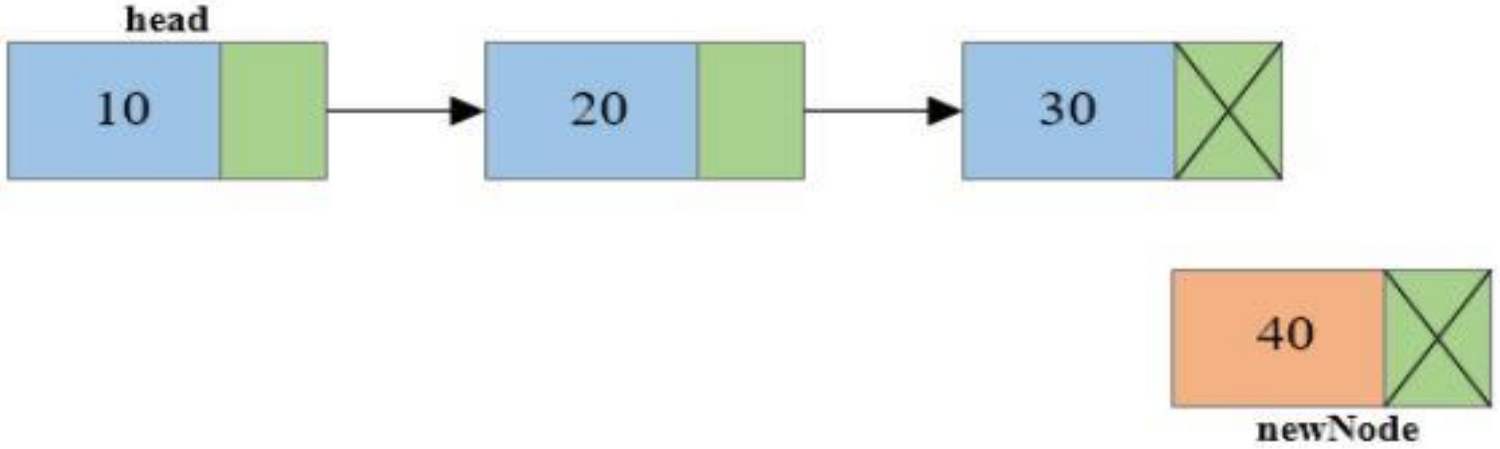
Step 4: If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.

Step 5: Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).

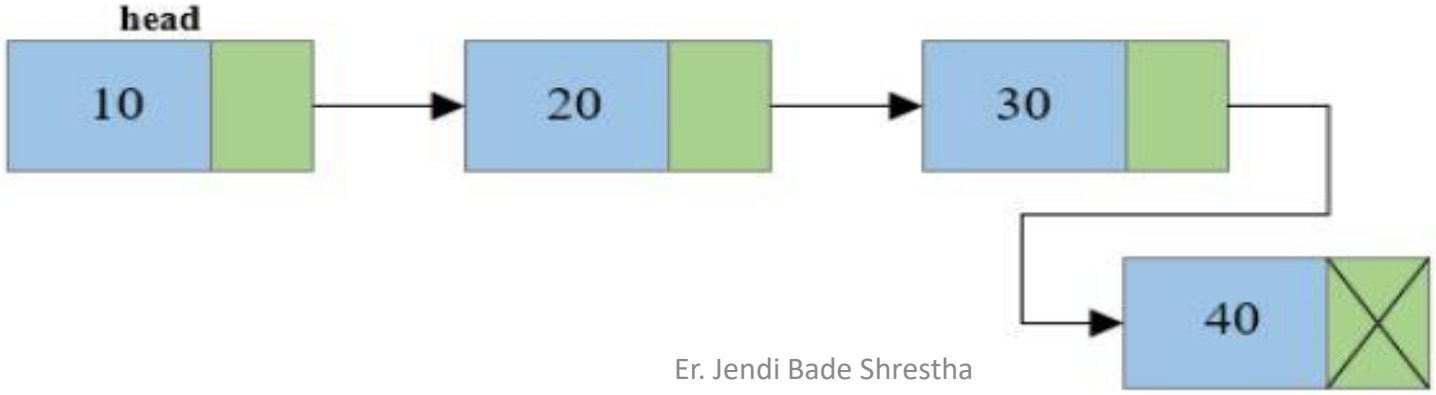
Step 6: Set **temp → next** = **newNode**.

Steps to insert node at the end of Singly linked list

1. Create a new node and make sure that the address part (next field) of the new node points to NULL i.e. **newNode->next=NULL**



2. Traverse to the last node of the linked list and connect the last node of the list with the new node, i.e. last node will now point to new node. **(lastNode->next = newNode).**



Program 3

WAP in C to create a singly linked list of n nodes and insert a new node at the end of the singly linked list.

Note: type this link for code.

https://codeforwin.org/2015/09/c-program-to-insert-node-at-end-of-singly-linked-list.html

```
1.  /*C program to insert new node at
2.  the end of a Singly Linked List*/
3.  #include <stdio.h>
4.  #include <conio.h>
5.  #include <stdlib.h>
6.  /* Structure of a node */
7.  struct node
8.  {
9.     int data;        // Data
10.    struct node *next; // Address
11. }*head;
12. void createList(int n);
13. void insertNodeAtEnd(int data);
14. void displayList();
15. void main()
16. {
17.    int n, data;
18.    clrscr();
19.    printf("Enter the total number of nodes:
    ");
20.    scanf("%d", &n);
21.    createList(n);
22.    printf("\nData in the list \n");
23.    displayList();
24.    printf("\nEnter data to insert at end of
    the list: ");
25.    scanf("%d", &data);
26.    insertNodeAtEnd(data);
27.    printf("\nData in the list \n");
28.    displayList();
29.    getch();
30. }
```

```
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;
   head = (struct node *)malloc(sizeof(struct node));
   if(head == NULL)
   {
      printf("Unable to allocate memory.");
   }
   else
   {
      printf("Enter the data of node 1: ");
      scanf("%d", &data);

      head->data = data; // Link the data field with data
      head->next = NULL; // Link the address field to NULL
      temp = head;
      for(i=2; i<=n; i++)
      {
         newNode = (struct node *)malloc(sizeof(struct node));
         if(newNode == NULL)
         {
                  printf("Unable to allocate memory.");
                  break;
         }
         else
         {
                  printf("Enter the data of node %d: ", i);
                  scanf("%d", &data);
                  newNode->data = data; // Link the data field of newNode with data
                  newNode->next = NULL; // Link the address field of newNode with NULL
                  temp->next = newNode; // Link previous node i.e. temp to the newNode
                  temp = temp->next;
         }
      }
      printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");
   }
}
```

```
void insertNodeAtEnd(int data)
{
   struct node *newNode, *temp;
   newNode = (struct node*)malloc(sizeof(struct node));
   if(newNode == NULL)
   {
      printf("Unable to allocate memory.");
   }
   else
   {
      newNode->data = data; // Link the data part
      newNode->next = NULL;
      temp = head;
      while(temp != NULL && temp->next != NULL)
         temp = temp->next;
      temp->next = newNode; // Link address part
      printf("DATA INSERTED SUCCESSFULLY\n");
   }
}
```

```
void displayList()
{
   struct node *temp;
   if(head == NULL)
   {
      printf("List is empty.");
   }
   else
   {
      temp = head;
      while(temp != NULL)
      {
         printf("Data = %d\n", temp->data); // Print data of current node
         temp = temp->next;          // Move to next node
      }
   }
}
```

Output:

```
Enter the total number of nodes: 4
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 30
Enter the data of node 4: 40
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40

Enter data to insert at end of the list: 50
DATA INSERTED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50
```

- **How to Insert a new node at the middle (or at any position) of Singly Linked List.**

c. Inserting node at the middle (or at any position) of Singly Linked List.

Algorithm:

Step 1: Create a **newNode** with given value and **newNode → next** as **NULL**.

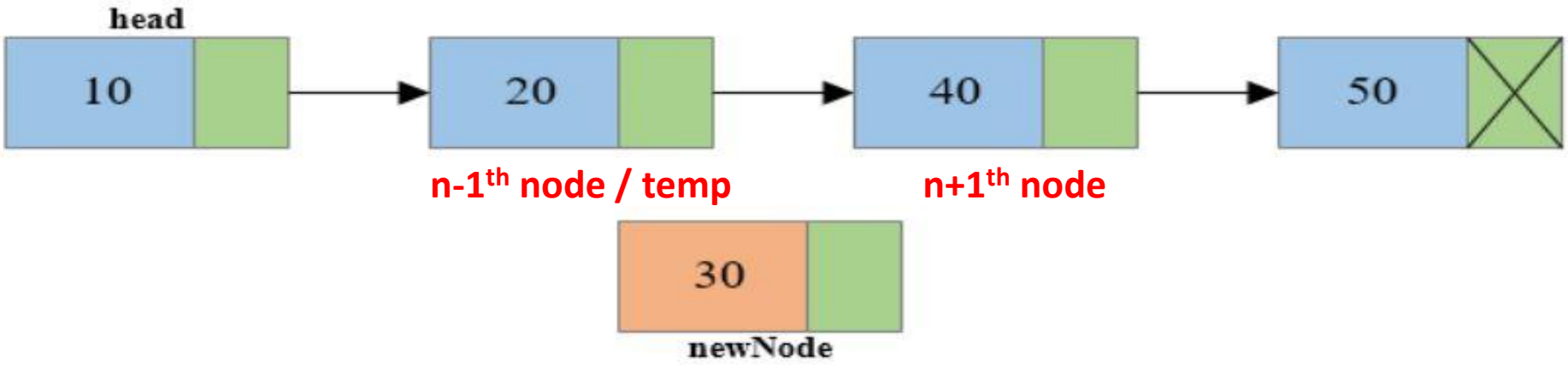
Step 2: Check whether list is **Empty** (**head** == **NULL**).

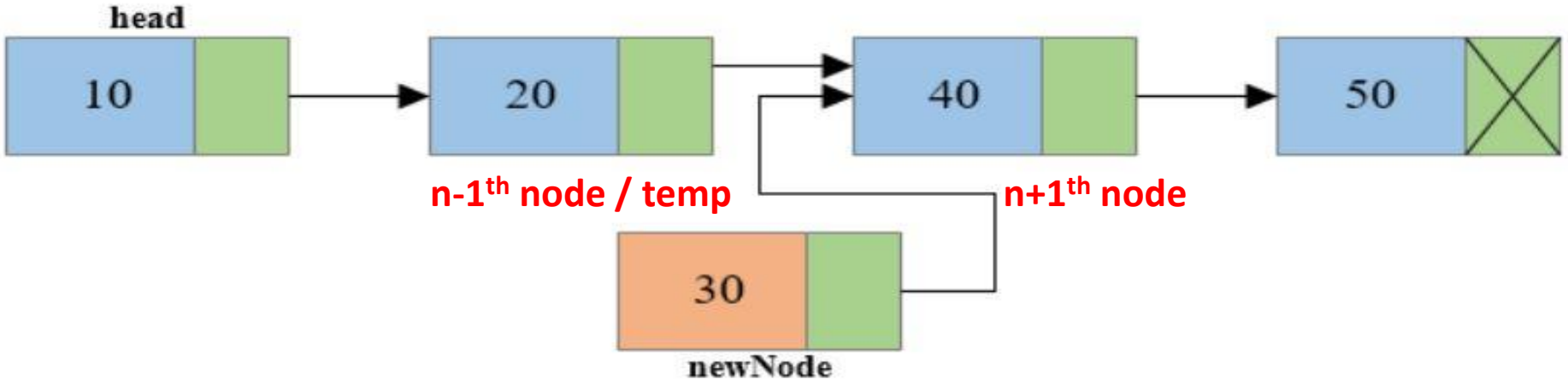
Step 3: If it is **Empty** then, set **head** = **newNode**.

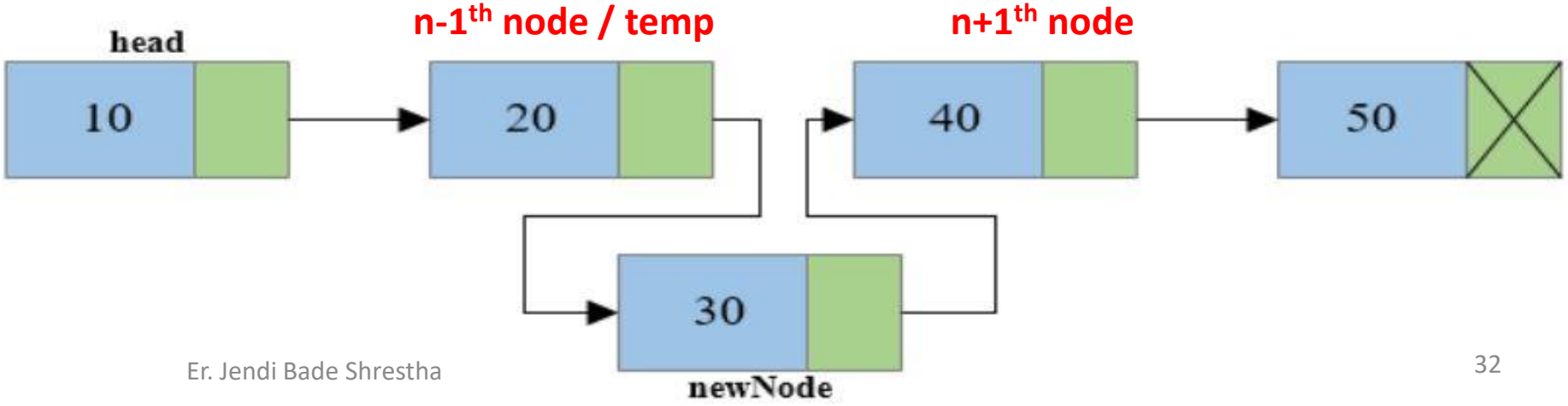
Step 4: If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.

Step 5: Traverse to the **n-1th position** of the linked list and connect the **new node** with the **n+1th node**. Means the **new node** should point to the same node that the **n-1th node** is pointing to. (**newNode->next = temp->next** where **temp** is the **n-1th node**).

Step 6: Now at last connect the **n-1th node** with the **new node** i.e. the **n-1th node** will now point to **new node**. (**temp->next = newNode** where **temp** is the **n-1th node**).


head
10
20
40
50
n-1th node / temp
n+1th node
30
newNode


head
10
20
40
50
n-1th node / temp
n+1th node
30
newNode


n-1th node / temp
n+1th node
head
10
20
40
50
30
newNode

Program 4

WAP in C to create a singly linked list of n nodes and insert a new node at the middle (or at any position) of the singly linked list.

Note: type this link for code.

https://codeforwin.org/2015/09/c-program-to-insert-node-at-middle-of-singly-linked-list.html

```
/**
 * C program to insert new node at the middle of Singly Linked List
 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>



/* Structure of a node */
struct node {
   int data;        // Data
   struct node *next; // Address
}*head;


void createList(int n);
void insertNodeAtMiddle(int data, int position);
void displayList();


int main()
{
   int n, data, position;
   clrscr();

   /*
    * Create a singly linked list of n nodes
    */
   printf("Enter the total number of nodes: ");
   scanf("%d", &n);
   createList(n);

   printf("\nData in the list \n");
   displayList();

   /*
    * Insert data at middle of the singly linked list
    */
   printf("Enter data to insert at middle of the list: ");
   scanf("%d", &data);
   printf("Enter the position to insert new node: " );
   scanf("%d", &position);
   insertNodeAtMiddle(data, position);

   printf("\nData in the list \n");
   displayList();
   getch();
   return 0;
}
```

```
40.       /*
 * Create a list of n nodes
 */
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;

   head = (struct node *)malloc(sizeof(struct node));

   /*
    * If unable to allocate memory for head node
    */
   if(head == NULL)
   {
                printf("Unable to allocate memory.");
   }
   else
   {
                /*
                 * Input data of node from the user
                 */
                printf("Enter the data of node 1: ");
                scanf("%d", &data);

                head->data = data; // Link the data field with data
                head->next = NULL; // Link the address field to NULL

                temp = head;

                /*
                 * Creates n nodes and adds to linked list
                 */
                for(i=2; i<=n; i++)
                {
                   newNode = (struct node *)malloc(sizeof(struct
node));

                   /* If memory is not allocated for newNode */
                   if(newNode == NULL)
                   {
                                printf("Unable to allocate memory.");
                                break;
                   }
                   else
                   {
                                printf("Enter the data of node %d: ", i);
                                scanf("%d", &data);

                                newNode->data = data; // Link the
data field of newNode with data
                                newNode->next = NULL; // Link the
address field of newNode with NULL

                                temp->next = newNode; // Link
previous node i.e. temp to the newNode
                                temp = temp->next;
                   }
                }

                printf("SINGLY LINKED LIST CREATED
SUCCESSFULLY\n");
   }
}
```

```
90.     /*
 * Creates a new node and inserts at middle of the linked
list.
 */
void insertNodeAtMiddle(int data, int position)
{
   int i;
   struct node *newNode, *temp;

   newNode = (struct node*)malloc(sizeof(struct node));

   if(newNode == NULL)
   {
            printf("Unable to allocate memory.");
   }
   else
   {
            newNode->data = data; // Link data part
            newNode->next = NULL;

            temp = head;

            /*
             * Traverse to the n-1 position
             */
            for(i=2; i<=position-1; i++)
            {
               temp = temp->next;

               if(temp == NULL)
                        break;
            }

            if(temp != NULL)
            {
               /* Link address part of new node */
               newNode->next = temp->next;

               /* Link address part of n-1 node */
               temp->next = newNode;

               printf("DATA INSERTED SUCCESSFULLY\n");
            }
            else
            {
               printf("UNABLE TO INSERT DATA AT THE
GIVEN POSITION\n");
            }
      }
}
```

```
/*
 * Display entire list
 */
void displayList()
{
   struct node *temp;

   /*
    * If the list is empty i.e. head = NULL
    */
   if(head == NULL)
   {
            printf("List is empty.");
   }
   else
   {
            temp = head;
            while(temp != NULL)
            {
               printf("Data = %d\n", temp->data); // Print data of current node
               temp = temp->next;              // Move to next node
            }
   }
}
```

Output:

```
Enter the total number of nodes: 4
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 40
Enter the data of node 4: 50
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 40
Data = 50
Enter data to insert at middle of the list: 30
Enter the position to insert new node: 3
DATA INSERTED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50
```

3. Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows.

a. **Deleting from Beginning of the list**

b. **Deleting from End of the list**

c. **Deleting a Specific Node(i.e middle node or at any position)**

- **How to delete first node from singly linked list**

NOTE:

[Good site for DSA]

- http://www.btechsmartclass.com/

a. Deleting from Beginning of the list

Algorithm: [Method 1]

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

Step 2: If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

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

Step 4: Check whether list is having only one node (**temp** → **next** == **NULL**)

Step 5: If it is **TRUE** then set **head** = **NULL** and delete **temp** (Setting **Empty** list conditions)

Step 6: If it is **FALSE** then set **head** = **temp** → **next**, and delete **temp**.

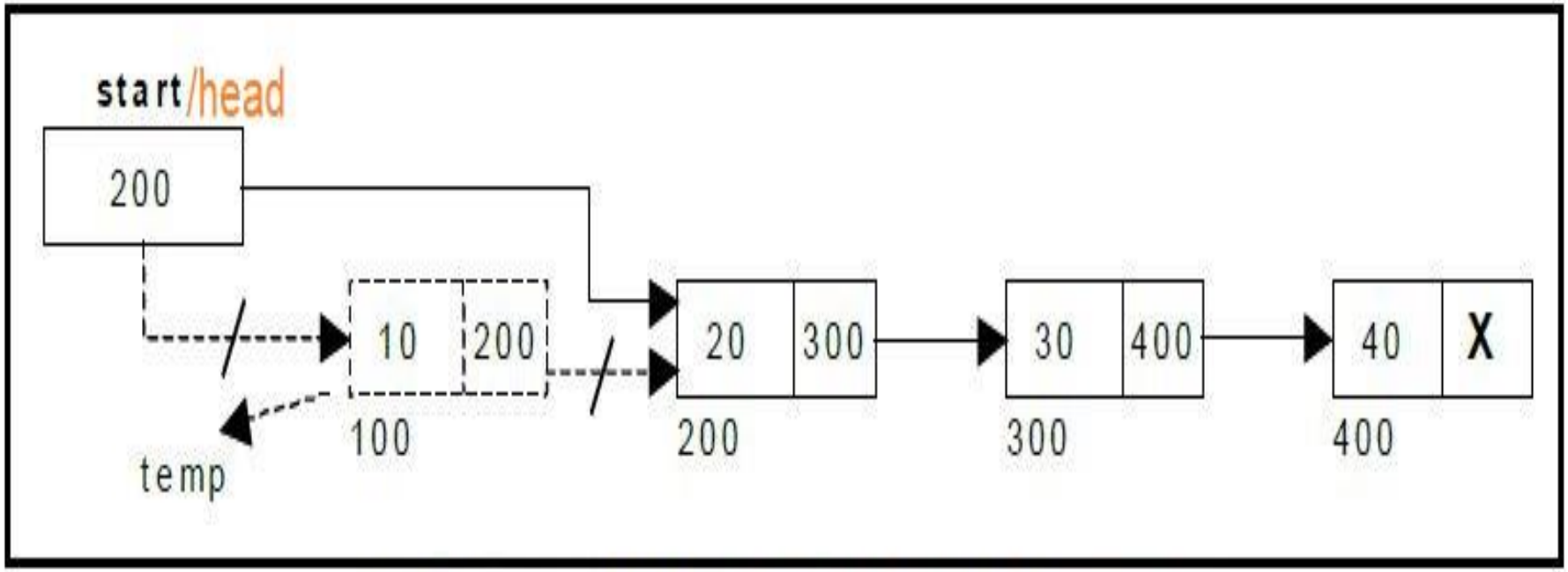
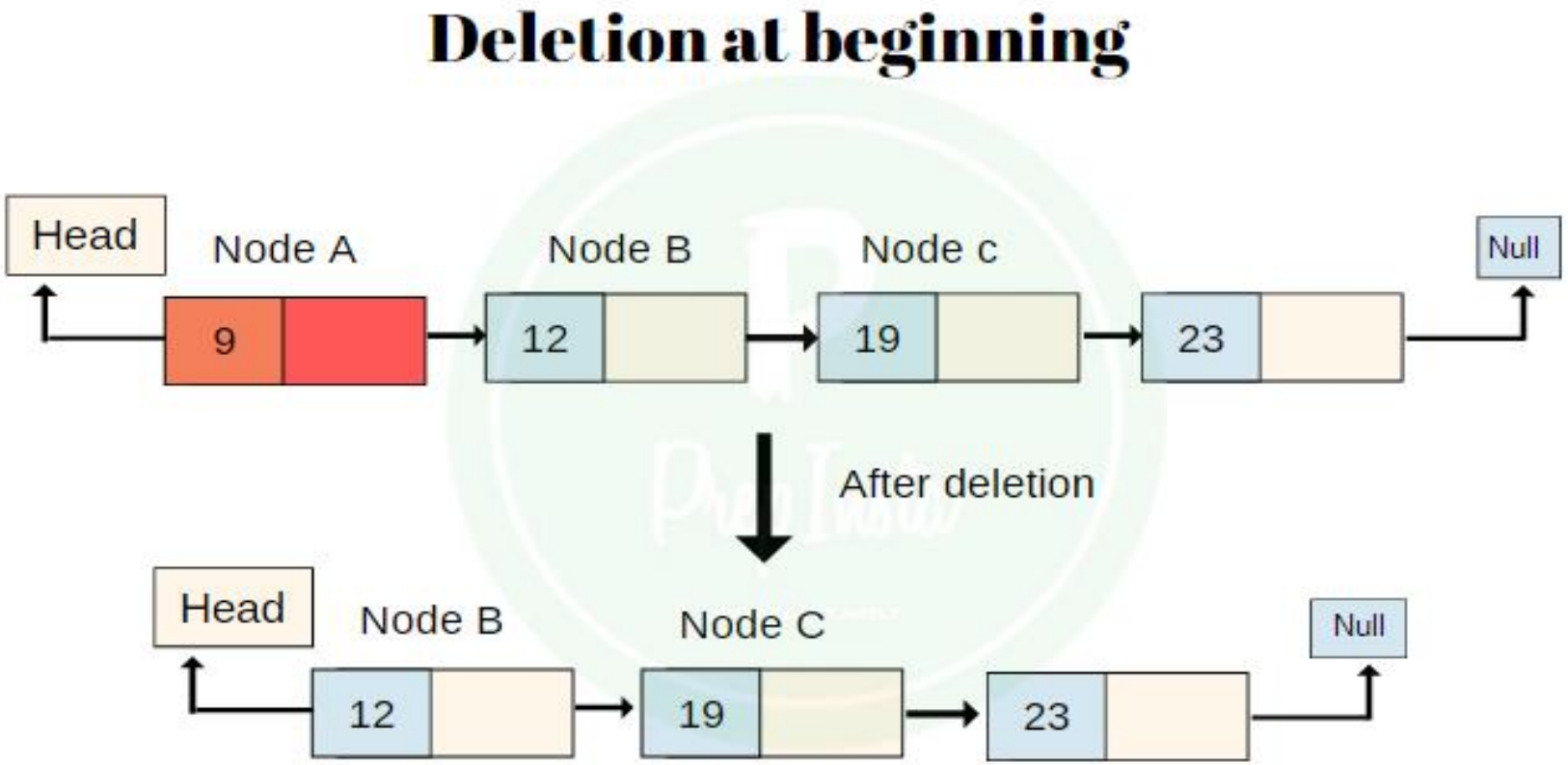


Figure → Deleting a node at the beginning.

Note:

We can also make a fig like this to delete a node form beginning.



a. Deleting from Beginning of the list

Algorithm: [Method 2]

1. Read the target node or first node

2. If the first node is not found
   Display message and stop

   else

3. Copy the address from the address field of first node and assign to the head.

4. Free the first node.

Program 5

Write a C program to create a singly linked list of n nodes and delete the first node or beginning node of the linked list.

https://codeforwin.org/2015/09/c-program-to-delete-first-node-of-singly-linked-list.html

```
/**
 * C program to delete first node
 from Singly Linked List
 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>


/* Structure of a node */
struct node {
   int data;        // Data
   struct node *next; // Address
}*head;


void createList(int n);
void deleteFirstNode();
void displayList();



int main()
{
   int n, choice;
   clrscr();

   /*
    * Create a singly linked list of n nodes
    */
   printf("Enter the total number of nodes: ");
   scanf("%d", &n);
   createList(n);

   printf("\nData in the list \n");
   displayList();

   printf("\nPress 1 to delete first node: ");
   scanf("%d", &choice);

   /* Delete first node from list */
   if(choice == 1)
            deleteFirstNode();

   printf("\nData in the list \n");
   displayList();
   getch();
   return 0;
}
```

```
/*
 * Create a list of n nodes
 */
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;

   head = (struct node *)malloc(sizeof(struct node));

   /*
    * If unable to allocate memory for head node
    */
   if(head == NULL)
   {
                printf("Unable to allocate memory.");
   }
   else
   {
                /*
                 * In data of node from the user
                 */
                printf("Enter the data of node 1: ");
                scanf("%d", &data);

                head->data = data; // Link the data field with data
                head->next = NULL; // Link the address field to NULL

                temp = head;

                /*
                 * Create n nodes and adds to linked list
                 */
                for(i=2; i<=n; i++)
                {
                   newNode = (struct node *)malloc(sizeof(struct
node));

                   /* If memory is not allocated for newNode */
                   if(newNode == NULL)
                   {
                                printf("Unable to allocate memory.");
                                break;
                   }
                   else
                   {
                                printf("Enter the data of node %d: ", i);
                                scanf("%d", &data);

                                newNode->data = data; // Link the
data field of newNode with data
                                newNode->next = NULL; // Link the
address field of newNode with NULL

                                temp->next = newNode; // Link
previous node i.e. temp to the newNode
                                temp = temp->next;
                   }
                }

                printf("SINGLY LINKED LIST CREATED
SUCCESSFULLY\n");
   }
}
```

```
/*
 * Deletes the first node of the linked list
 */
void deleteFirstNode()
{
   struct node *toDelete;

   if(head == NULL)
   {
            printf("List is already empty.");
   }
   else
   {
            toDelete = head;
            head = head->next;

            printf("\nData deleted = %d\n", toDelete-
>data);

            /* Clears the memory occupied by first
node*/
            free(toDelete);

            printf("SUCCESSFULLY DELETED FIRST NODE
FROM LIST\n");
   }
}
```

```
/*
 * Displays the entire list
 */
void displayList()
{
   struct node *temp;

   /*
    * If the list is empty i.e. head = NULL
    */
   if(head == NULL)
   {
            printf("List is empty.");
   }
   else
   {
            temp = head;
            while(temp != NULL)
            {
               printf("Data = %d\n", temp->data); // Print
data of current node
               temp = temp->next;          // Move to
next node
            }
   }
}
```

Output:

```
Enter the total number of nodes: 4
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 30
Enter the data of node 4: 40
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40

Press 1 to delete first node: 1

Data deleted = 10
SUCCESSFULLY DELETED FIRST NODE FROM LIST

Data in the list
Data = 20
Data = 30
Data = 40
```

- **How to delete last node from singly linked list**

b. Deleting from End of the list

Algorithm: [METHOD 1]

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

Step 2: If it is **Empty** then, display '**List is Empty!!! Deletion is not possible**' and terminate the function.

Step 3: If it is **Not Empty** then, define two Node pointers '**temp1**' and '**temp2**' and initialize '**temp1**' with **head**.

Step 4: Check whether list has only one Node (**temp1 → next == NULL**)

Step 5: If it is **TRUE**. Then, set **head = NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)

Step 6: If it is **FALSE**. Then, set '**temp2 = temp1** ' and move **temp1** to its next node. Repeat the same until it reaches to the last node in the list. (until **temp1 → next == NULL**)

Step 7: Finally, Set **temp2 → next = NULL** and delete **temp1**.

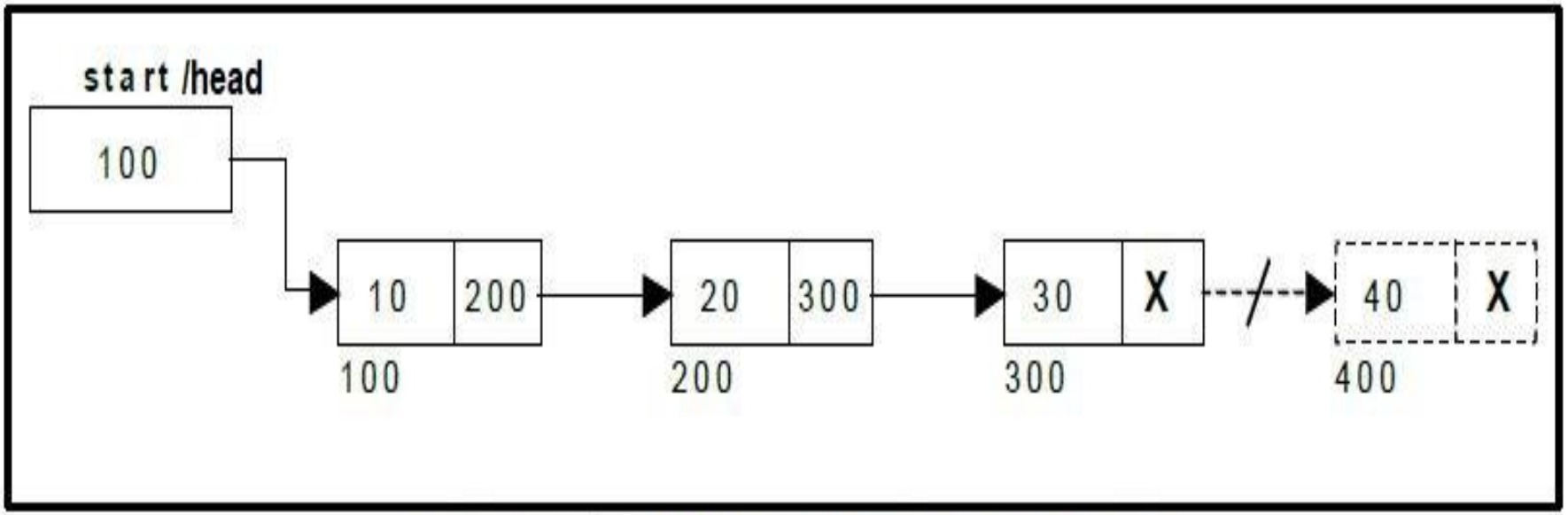
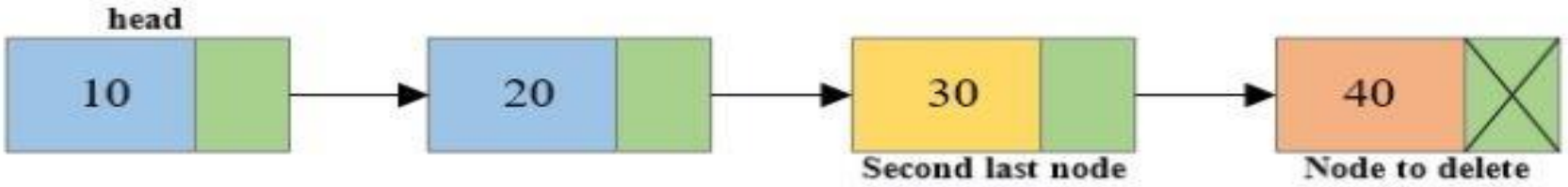
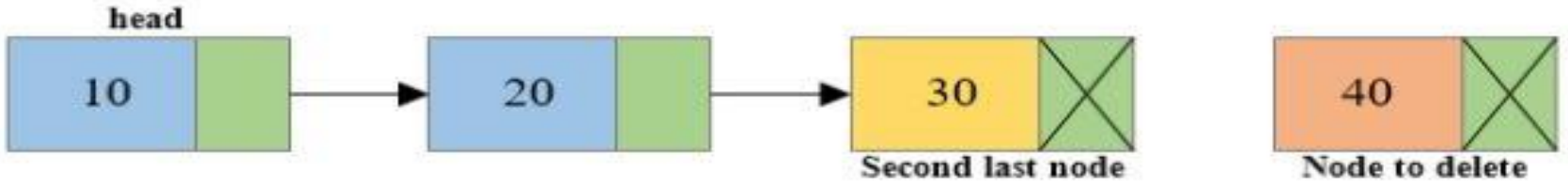


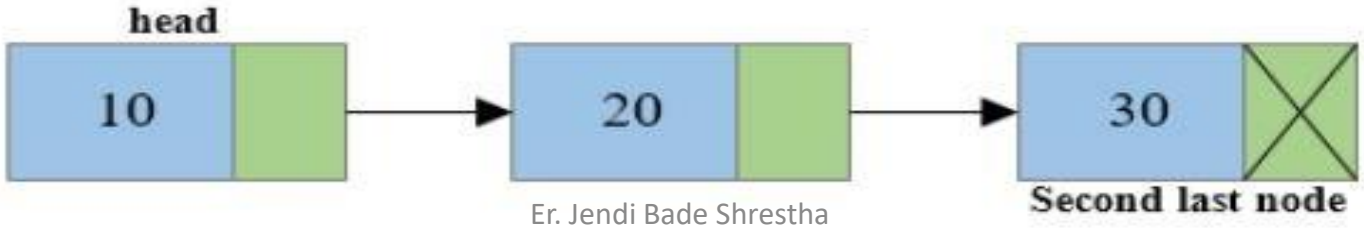
Figure → Deleting a node at the end.

Note:

We can also draw a figure like this for deletion of node at last.







b. Deleting from End of the list

Algorithm: [METHOD 2]

1. Read the target node

2. If target node is not found
   Display message and stop
   else

3. Traverse the list to the second last node.

4. Assign null value to the address field of second last node.

5. Free the last node.

PROGRAM 6

- **Write a C program to create a singly linked list of n nodes and delete the last node of the list.**

- https://codeforwin.org/2015/09/c-program-to-delete-last-node-of-singly-linked-list.html

```
/**
 * C program to delete last node of Singly Linked List
 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>


/* Structure of a node */
struct node {
    int data;        // Data
    struct node *next; // Address
}*head;

void createList(int n);
void deleteLastNode();
void displayList();


int main()
{
    int n, choice;
    clrscr();

    /*
     * Create a singly linked list of n nodes
     */
    printf("Enter the total number of nodes: ");
    scanf("%d", &n);
    createList(n);

    printf("\nData in the list \n");
    displayList();

    printf("\nPress 1 to delete last node: ");
    scanf("%d", &choice);

    /* Delete last node from list */
    if(choice == 1)
            deleteLastNode();

    printf("\nData in the list \n");
    displayList();
    getch();
    return 0;
}
```

```
/*
 * Create a list of n nodes
 */
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;

   head = (struct node *)malloc(sizeof(struct node));

   /*
    * If unable to allocate memory for head node
    */
   if(head == NULL)
   {
                printf("Unable to allocate memory.");
   }
   else
   {
                /*
                 * Input data of node from the user
                 */
                printf("Enter the data of node 1: ");
                scanf("%d", &data);

                head->data = data; // Link the data field with data
                head->next = NULL; // Link the address field to NULL

                temp = head;

                /*
                 * Create n nodes and adds to linked list
                 */
                for(i=2; i<=n; i++)
                {
                   newNode = (struct node *)malloc(sizeof(struct
node));

                   /* If memory is not allocated for newNode */
                   if(newNode == NULL)
                   {
                                printf("Unable to allocate memory.");
                                break;
                   }
                   else
                   {
                                printf("Enter the data of node %d: ", i);
                                scanf("%d", &data);

                                newNode->data = data; // Link the
data field of newNode with data
                                newNode->next = NULL; // Link the
address field of newNode with NULL

                                temp->next = newNode; // Link
previous node i.e. temp to the newNode
                                temp = temp->next;
                   }
                }

                printf("SINGLY LINKED LIST CREATED
SUCCESSFULLY\n");
   }
}
```

```
/*
 * Delete last node of the linked list
 */
void deleteLastNode()
{
   struct node *toDelete, *secondLastNode;

   if(head == NULL)
   {
                printf("List is already empty.");
   }
   else
   {
                toDelete = head;
                secondLastNode = head;

                /* Traverse to the last node of the list */
                while(toDelete->next != NULL)
                {
                   secondLastNode = toDelete;
                   toDelete = toDelete->next;
                }

                if(toDelete == head)
                {
                   head = NULL;
                }
                else
                {
                   /* Disconnect link of second last node with last
node */
                   secondLastNode->next = NULL;
                }
                /* Delete the last node */
                free(toDelete);

                printf("SUCCESSFULLY DELETED LAST NODE OF
LIST\n");
   }
}


/*
 * Display entire list
 */
void displayList()
{
   struct node *temp;

   /*
    * If the list is empty i.e. head = NULL
    */
   if(head == NULL)
   {
                printf("List is empty.");
   }
   else
   {
                temp = head;
                while(temp != NULL)
                {
                   printf("Data = %d\n", temp->data); // Print the
data of current node
                   temp = temp->next;              // Move to next node
                }
   }
}
```

Output:

```
Enter the total number of nodes: 5
Enter the data of node 1: 11
Enter the data of node 2: 22
Enter the data of node 3: 33
Enter the data of node 4: 44
Enter the data of node 5: 55
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 11
Data = 22
Data = 33
Data = 44
Data = 55

Press 1 to delete last node: 1
SUCCESSFULLY DELETED LAST NODE OF LIST

Data in the list
Data = 11
Data = 22
Data = 33
Data = 44
```

- **How to delete node from the middle (or at any position) of the singly linked list**

c. delete middle node (or at any position) from singly linked list

Algorithm

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

Step 2: If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

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

Step 4: Check whether list has only one Node (**temp1 → next** == **NULL**)

Step 5: If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)

Step 6: If it is **FALSE** then Traverse to the **n^{th} node** of the singly linked list and also keep reference of **n-1^{th} node** in some **temp** variable say **prevnode.**

Step 7: Reconnect the **n-1^{th} node** with the **n+1^{th} node**

i.e. **prevNode->next = toDelete->next** (Where **prevNode** is **n-1^{th} node** and **toDelete node** is the **n^{th} node** and **toDelete->next** is the **n+1^{th} node**).

Step 8: Free the memory occupied by the **n^{th} node** i.e. **toDelete node**

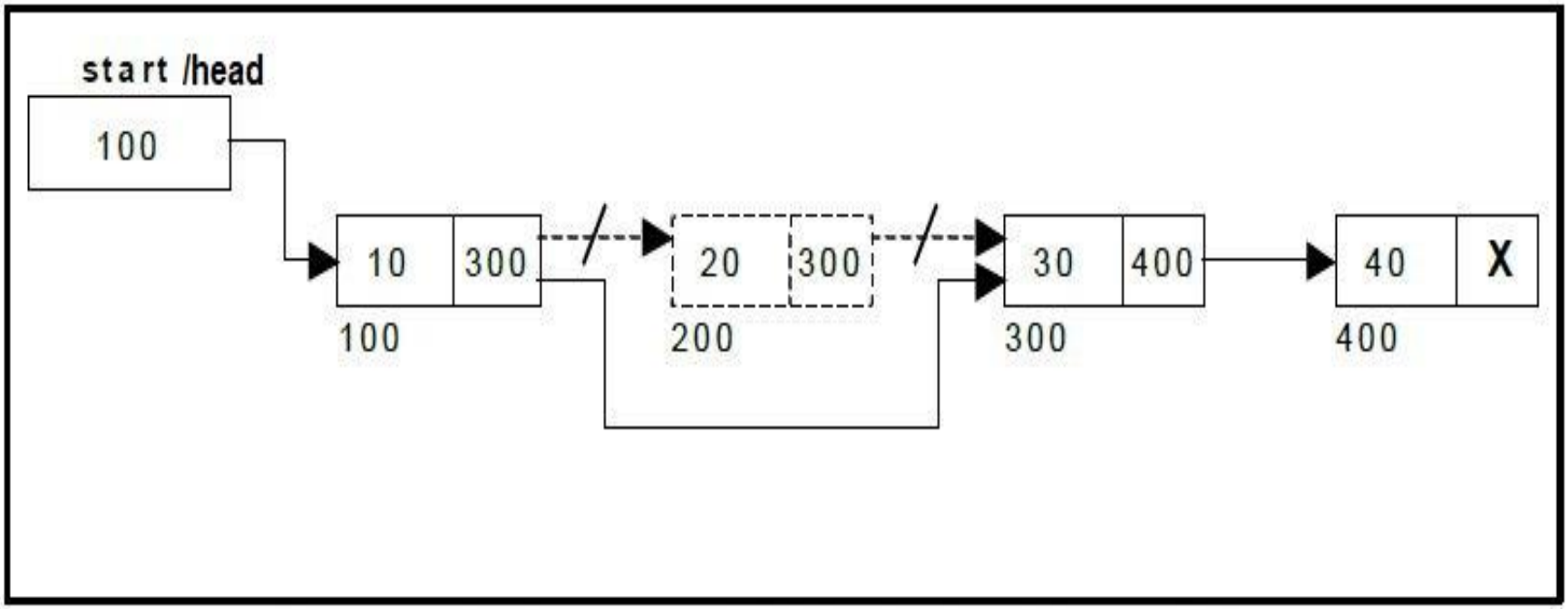
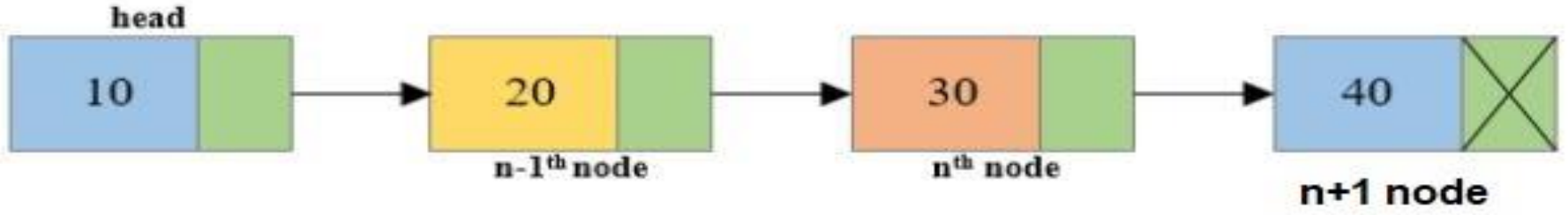
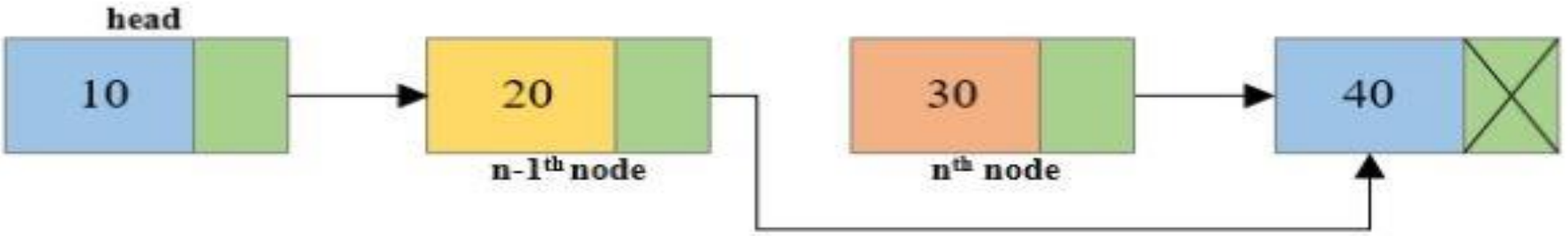


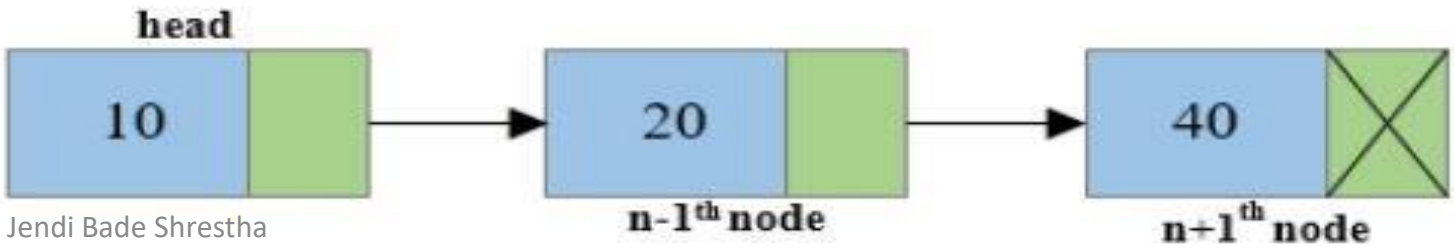
Figure ⟶ Deleting a node at an intermediate position.

Note:

We can also draw a figure like this for deletion of node at middle.







PROGRAM 7

- **Write a C program to create a singly linked list of n nodes and delete node from the middle (or at any position) of the linked list.**

- https://codeforwin.org/2015/09/c-program-to-delete-middle-node-of-singly-linked-list.html

```
/**
 * C program to delete middle node of Singly Linked List
 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>


/* Structure of a node */
struct node {
   int data;        // Data
   struct node *next; // Address
} *head;


/* Functions used in program */
void createList(int n);
void deleteMiddleNode(int position);
void displayList();


int main()
{
   int n, position;
   clrscr();

   /*
    * Create a singly linked list of n nodes
    */
   printf("Enter the total number of nodes: ");
   scanf("%d", &n);
   createList(n);

   printf("\nData in the list \n");
   displayList();

   printf("\nEnter the node position you want to delete:
");
   scanf("%d", &position);

   /* Delete middle node from list */
   deleteMiddleNode(position);

   printf("\nData in the list \n");
   displayList();
   getch();
   return 0;
}
```

```
/*
 * Create a list of n nodes
 */
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;

   head = (struct node *)malloc(sizeof(struct node));

   /*
    * If unable to allocate memory for head node
    */
   if(head == NULL)
   {
                printf("Unable to allocate memory.");
   }
   else
   {
                /*
                 * Read data of node from the user
                 */
                printf("Enter the data of node 1: ");
                scanf("%d", &data);

                head->data = data; // Link the data field with data
                head->next = NULL; // Link the address field to NULL

                temp = head;

                /*
                 * Create n nodes and adds to linked list
                 */
                for(i=2; i<=n; i++)
                {
                   newNode = (struct node *)malloc(sizeof(struct
node));

                   /* If memory is not allocated for newNode */
                   if(newNode == NULL)
                   {
                                printf("Unable to allocate memory.");
                                break;
                   }
                   else
                   {
                                printf("Enter the data of node %d: ", i);
                                scanf("%d", &data);

                                newNode->data = data; // Link the
data field of newNode with data
                                newNode->next = NULL; // Link the
address field of newNode with NULL

                                temp->next = newNode; // Link
previous node i.e. temp to the newNode
                                temp = temp->next;
                   }
                }

                printf("SINGLY LINKED LIST CREATED
SUCCESSFULLY\n");
   }
}
```

```
/*
 * Delete middle node of the linked list
 */
void deleteMiddleNode(int position)
{
   int i;
   struct node *toDelete, *prevNode;

   if(head == NULL)
   {
                printf("List is already empty.");
   }
   else
   {
                toDelete = head;
                prevNode = head;

                for(i=2; i<=position; i++)
                {
                   prevNode = toDelete;
                   toDelete = toDelete->next;

                   if(toDelete == NULL)
                                break;
                }

                if(toDelete != NULL)
                {
                   if(toDelete == head)
                                head = head->next;

                   prevNode->next = toDelete->next;
                   toDelete->next = NULL;

                   /* Delete nth node */
                   free(toDelete);
                                printf("SUCCESSFULLY DELETED NODE FROM MIDDLE OF LIST\n");
                }
      else
      {
         printf("Invalid position unable to delete.");
      }
   }
}


/*
 * Display entire list
 */
void displayList()
{
   struct node *temp;

   /*
    * If the list is empty i.e. head = NULL
    */
   if(head == NULL)
   {
      printf("List is empty.");
   }
   else
   {
      temp = head;
      while(temp != NULL)
      {
         printf("Data = %d\n", temp->data); // Print the data of current node
         temp = temp->next;              // Move to next node
      }
   }
}
```

Output:

```
Enter the total number of nodes: 5
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 30
Enter the data of node 4: 40
Enter the data of node 5: 50
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50

Enter the node position you want to delete: 3
SUCCESSFULLY DELETED NODE FROM MIDDLE OF LIST

Data in the list
Data = 10
Data = 20
Data = 40
Data = 50
```

Advantages of Singly linked list

- Singly linked list is probably the most easiest data structure to implement.
- Insertion and deletion of element can be done easily.
- Insertion and deletion of elements doesn't requires movement of all elements when compared to an array.
- Requires less memory when compared to doubly and circular linked list.
- Can allocate or de-allocate memory easily when required during its execution.
- It is one of most efficient data structure to implement when traversing in one direction is required.

Disadvantages of Singly linked list

- It uses more memory when compared to an array.
- Since elements are not stored sequentially hence requires more time to access each elements of list.
- Traversing in reverse is not possible in case of Singly linked list when compared to Doubly linked list.
- Requires O(n) time on appending a new node to end.

Double Linked List

What is Double Linked List?

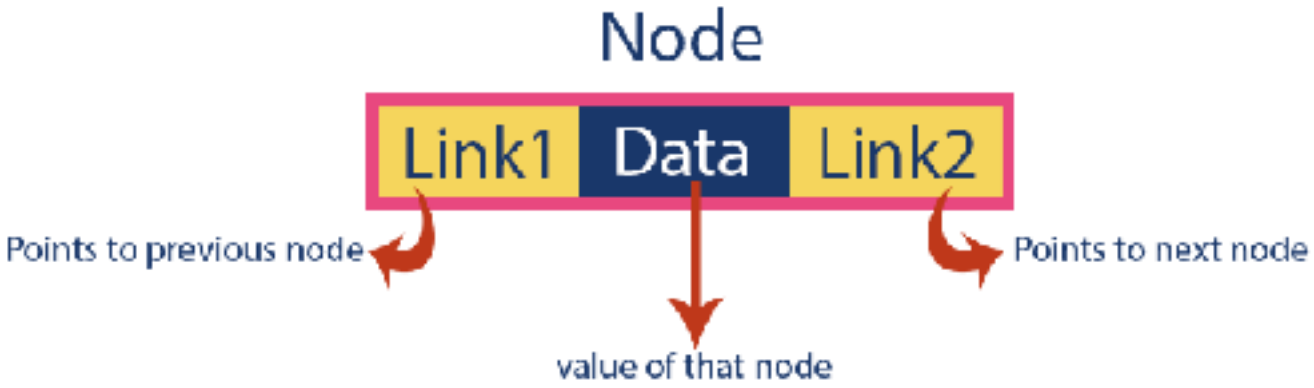
- In a single linked list, every node has link to its next node in the sequence. So, we can traverse from one node to other node only in one direction and we can not traverse back. We can solve this kind of problem by using **double linked list.** Double linked list can be defined as follows:

Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence.

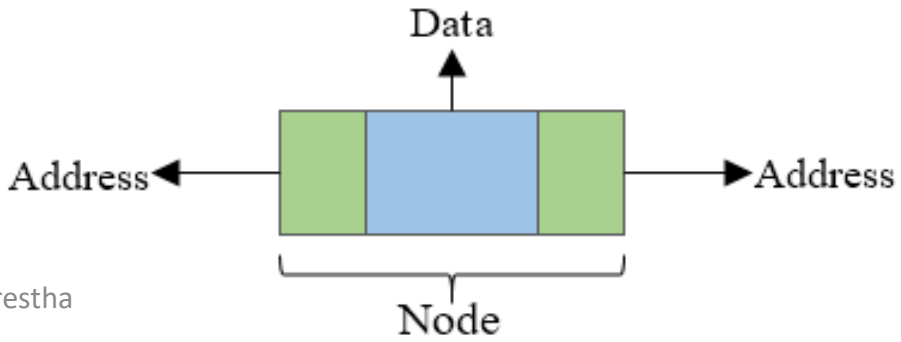
Or

- Doubly linked list is a collection of nodes linked together in a sequential way. Each node of the list contains two parts **data part** and the **reference or address part**.

- In double linked list, every node has link to its previous node and next node. So, we can traverse forward by using next field and can traverse backward by using previous field. Every node in a double linked list contains three fields and the **basic structure of node** is shown in the following figure.

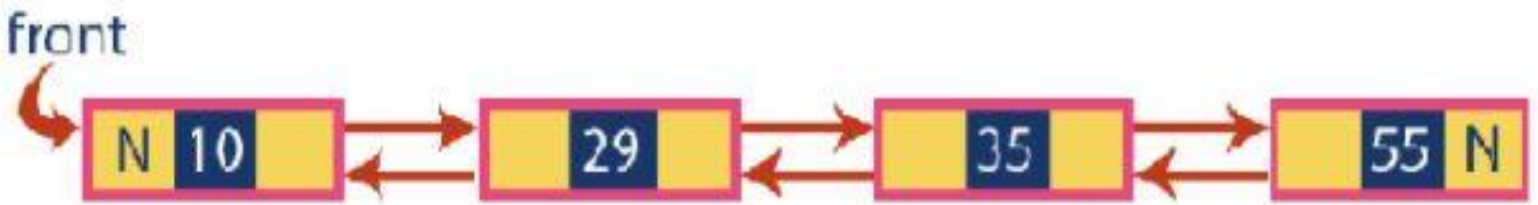


Or



- Here, '**link1**' field is used to store the address of the previous node in the sequence, '**link2**' field is used to store the address of the next node in the sequence and '**data**' field is used to store the actual value of that node.

- Doubly linked list is sometimes also referred as **bi-directional linked list** since it allows traversal of nodes in both direction. The **basic structure of a doubly linked list** is represented as:



Example:

A double linked list is shown in figure below:

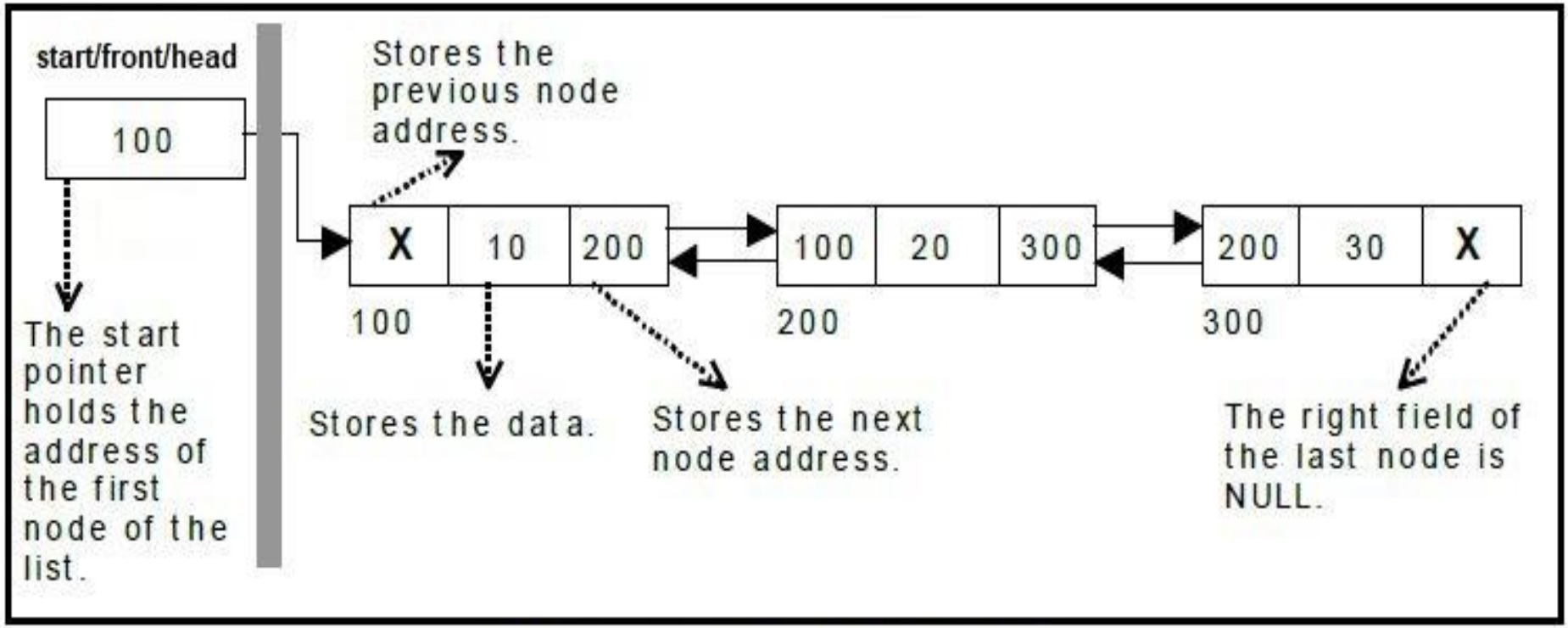


Figure → Double Linked List

NOTE

- In double linked list, the first node must be always pointed by **head.**
- Always the previous field of the first node must be **NULL.**
- Always the next field of the last node must be **NULL.**

Basic structure of a doubly linked list

- The basic structure of a doubly linked list contains a data field and two address fields.

```
struct node {
    int data;              // Data field
    struct node * prev; // Address of previous node
    struct node * next; // Address of next node
};
```

Operations

In a Doubly linked list we perform the following operations.

1. **Display**
2. **Insertion**
3. **Deletion**

1. Display

Program 8

- **Write a C program to create a doubly linked list and display all nodes of the created list.**

or

- **WAP in C to create and traverse a Doubly Linked List.**

Note: type this link for code.

https://codeforwin.org/2015/10/c-program-to-create-and-traverse-doubly-linked-list.html#creation-algorithm

```
/**
 * C program to create and display Doubly linked list.
[double1.cpp] */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

/*
 * Basic structure of Node
 */
struct node {
   int data;
   struct node * prev;
   struct node * next;
}*head, *last;


/*
 * Function used in this program
 */
void createList(int n);
void displayListFromFirst();
void displayListFromEnd();


int main()
{
   int n, choice;
   clrscr();
   head = NULL;
   last = NULL;

   printf("Enter the number of nodes you want to create:
");
   scanf("%d", &n);

   createList(n); // Create list of n nodes

   printf("\nPress 1 to display list from First:");
   printf("\nPress 2 to display list from End :\n");
   scanf("%d", &choice);

   if(choice==1)
   {
            displayListFromFirst();
   }
   else if(choice == 2)
   {
            displayListFromEnd();
   }
   getch();
   return 0;
}
```

```
/**
 * Create a doubly linked list of n nodes.
 * @n Number of nodes to be created
 */
void createList(int n)
{
   int i, data;
   struct node *newNode;

   if(n >= 1)
   {
      head = (struct node *)malloc(sizeof(struct node));

      if(head != NULL)
      {
         printf("Enter data of 1 node: ");
         scanf("%d", &data);

         head->data = data;
         head->prev = NULL;
         head->next = NULL;

         last = head;

         /*
          * Create rest of the n-1 nodes
          */
         for(i=2; i<=n; i++)
         {
            newNode = (struct node *)malloc(sizeof(struct node));

            if(newNode != NULL)
            {
               printf("Enter data of %d node: ", i);
               scanf("%d", &data);

               newNode->data = data;
               newNode->prev = last; // Link new node with the previous node
               newNode->next = NULL;

               last->next = newNode; // Link previous node with the new node
               last = newNode;         // Make new node as last/previous node
            }
            else
            {
               printf("Unable to allocate memory.");
               break;
            }
         }

         printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
      }
      else
      {
         printf("Unable to allocate memory");
      }
   }
}
```

```
/**
 * Displays the content of the list from beginning to end
 */
void displayListFromFirst()
{
   struct node * temp;
   int n = 1;

   if(head == NULL)
   {
      printf("List is empty.");
   }
   else
   {
      temp = head;
      printf("\n\nDATA IN THE LIST:\n");

      while(temp != NULL)
      {
         printf("DATA of %d node = %d\n", n, temp->data);

         n++;

         /* Move the current pointer to next node */
         temp = temp->next;
      }
   }
}
```

```
/**
 * Display the content of the list from last to first
 */
void displayListFromEnd()
{
   struct node * temp;
   int n = 0;

   if(last == NULL)
   {
      printf("List is empty.");
   }
   else
   {
      temp = last;
      printf("\n\nDATA IN THE LIST:\n");

      while(temp != NULL)
      {
         printf("DATA of last-%d node = %d\n", n, temp->data);

         n++;

         /* Move the current pointer to previous node */
         temp = temp->prev;
      }
   }
}
```

Output:

```
Enter the number of nodes you want to create: 5
Enter data of 1 node: 11
Enter data of 2 node: 22
Enter data of 3 node: 33
Enter data of 4 node: 44
Enter data of 5 node: 55

DOUBLY LINKED LIST CREATED SUCCESSFULLY

Press 1 to display list from First:
Press 2 to display list from End :
1


DATA IN THE LIST:
DATA of 1 node = 11
DATA of 2 node = 22
DATA of 3 node = 33
DATA of 4 node = 44
DATA of 5 node = 55
```

```
Enter the number of nodes you want to create: 5
Enter data of 1 node: 11
Enter data of 2 node: 22
Enter data of 3 node: 33
Enter data of 4 node: 44
Enter data of 5 node: 55

DOUBLY LINKED LIST CREATED SUCCESSFULLY

Press 1 to display list from First:
Press 2 to display list from End :
2


DATA IN THE LIST:
DATA of last-0 node = 55
DATA of last-1 node = 44
DATA of last-2 node = 33
DATA of last-3 node = 22
DATA of last-4 node = 11
```

2. Insertion

- In a doubly linked list, the insertion operation can be performed in three ways. They are as follows.

a. Inserting a node at the beginning of list.

b. Inserting a node at the end of list.

c. Inserting node at the middle (or at any position) of Doubly Linked List.

- **How to insert a new node at the beginning of a Doubly Linked List.**

a. Inserting a node at the beginning of list:

Algorithm: [METHOD 1]

Step 1: Create a **newNode** with given value and

newNode → previous as **NULL**.

Step 2: Check whether list is **Empty** (**head** == **NULL**)

Step 3: If it is **Empty** then, assign

NULL to **newNode →next** and **newNode** to **head**.

Step 4: If it is **not Empty** then,

assign **head** to **newNode →next** and **newNode** to **head**.

Step 5: Again assign address of newNode to the left link of node pointed by the right link of newNode.

Example:

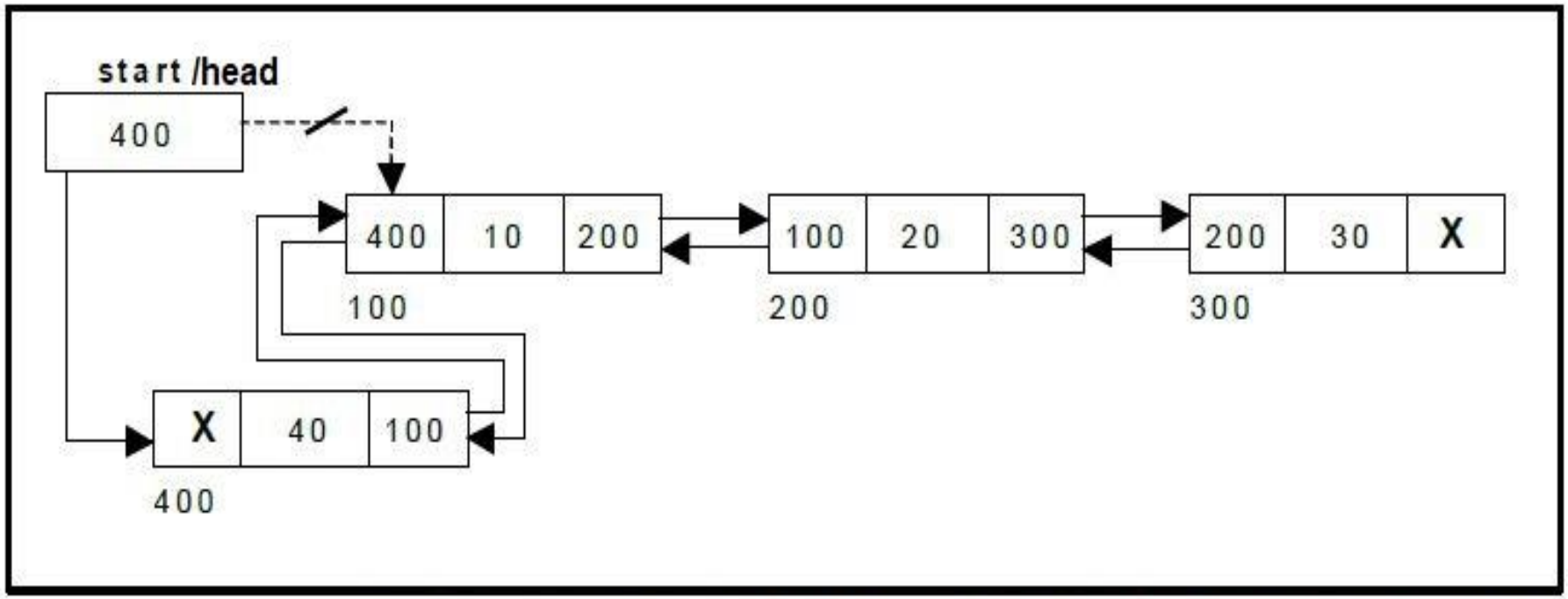
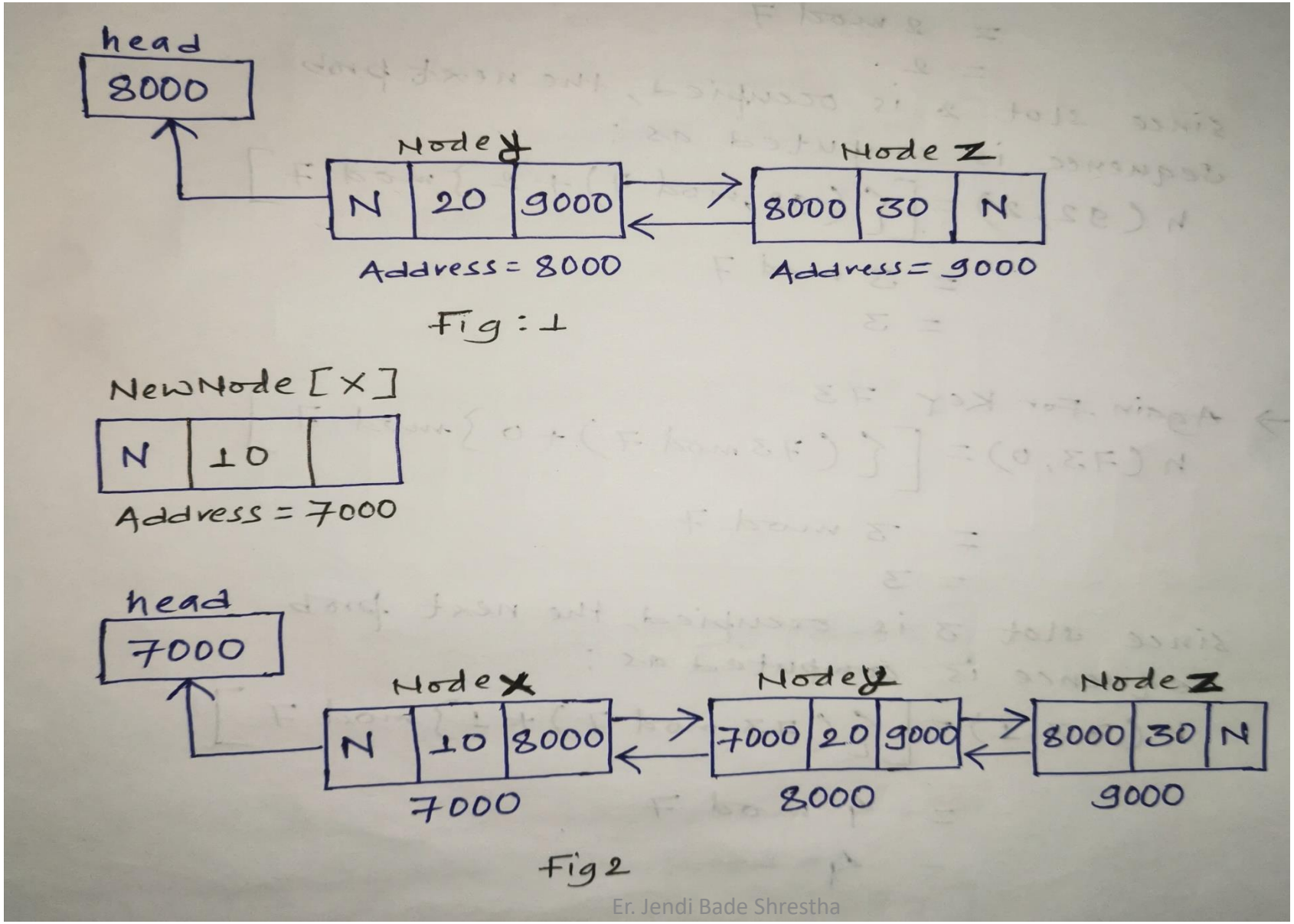


Figure ⟶ Inserting a node at the beginning **in doubly linked list**

We can draw a figure like this to insert a node at the beginning:



a. Inserting a node at the beginning of list:

Algorithm: [METHOD 2]

1. Create a new node.
2. Read the data.
3. Insert data to the data field.
4. Assign left field of new node with null value.
5. Assign value of head to the right link of new node.
6. Assign the address of new node to the left link of node pointed by right link of new node.
7. Assign the address of new node to head.

- **How to insert a new node at the end of a Doubly Linked List.**

b. Inserting a node at the end of list:

Algorithm: [METHOD 1]

Step 1: Create a **newNode** with given value and

newNode → previous as **NULL**.

Step 2: Check whether list is **Empty** (**head** == **NULL**)

Step 3: If it is **Empty** then, assign

NULL to **newNode →next** and **newNode** to **head**.

Step 4: If it is **not Empty**, then, define a node pointer **temp** and initialize with **head**.

Step 5: Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).

Step 6: Assign **newNode** to **temp → next** and

temp to **newNode → previous**.

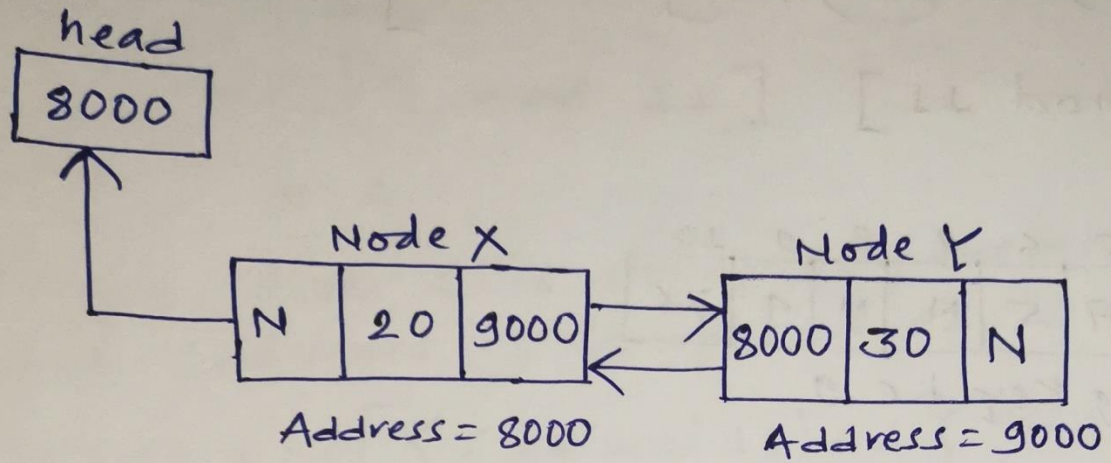
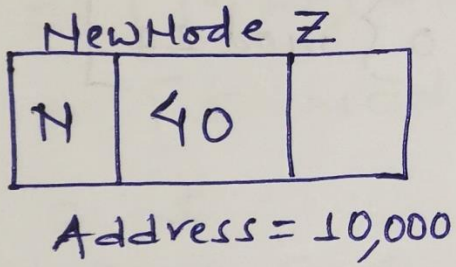

head
8000
Node X
N
20
9000
Address = 8000
Node Y
8000
30
N
Address = 9000

Fig 1


NewNode Z
N
40
Address = 10,000

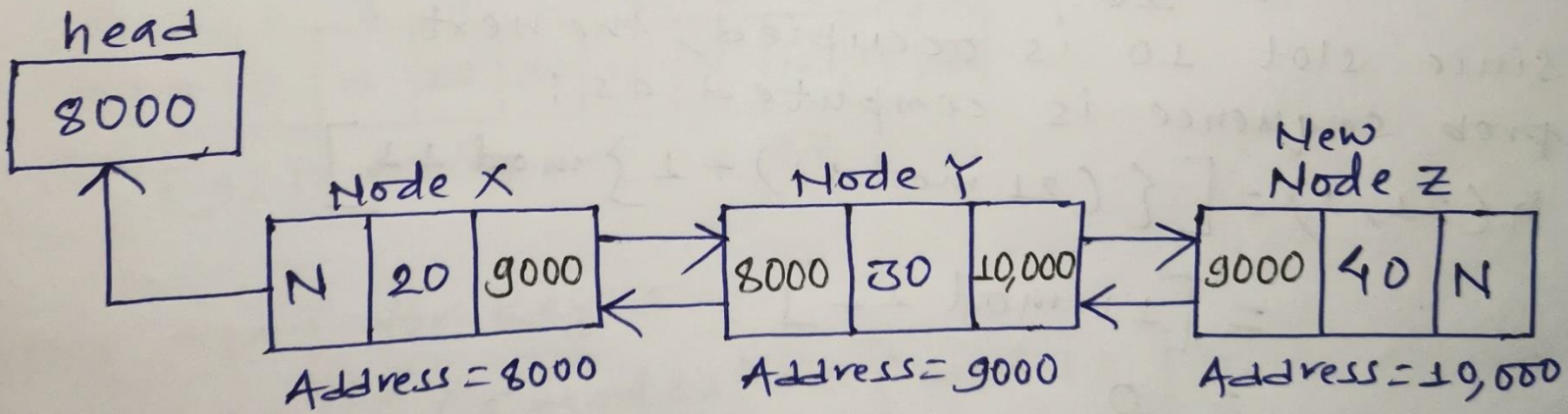

head
8000
Node X
N
20
9000
Address = 8000
Node Y
8000
30
10,000
Address = 9000
New
Node Z
9000
40
N
Address = 10,000

Fig 2

b. Inserting a node at the end of list:

Algorithm: [METHOD 2]

1. Create a new node.
2. Read the data.
3. Insert data to the data field.
4. Find the last node of linked list.
5. Assign the address of last node to the left link of new node.
6. Assign the address of new node to the right link of last node.
7. Assign the right link of new node with null value.

- **How to insert a new node at the middle (or at any position) of a Doubly Linked List.**

a. Inserting a node at the middle (or at any position) of list:

Algorithm:

Step 1: Create a **newNode** with given value and
newNode → previous as **NULL.**

Step 2: Check whether list is Empty **(head == NULL)**

Step 3: If it is Empty then, assign
NULL to **newNode →next** and **newNode** to **head.**

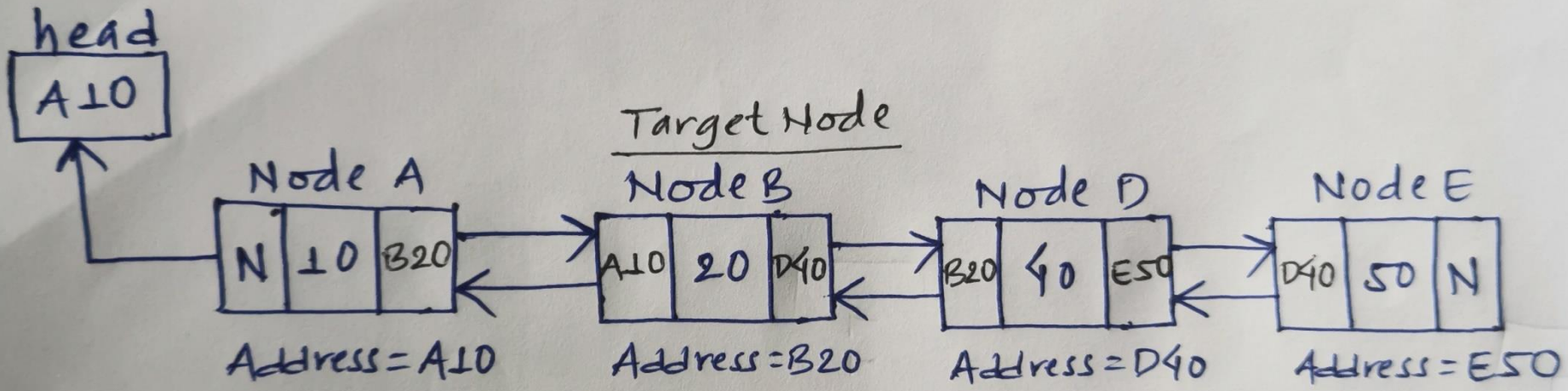
Step 4: If it is not empty Search the **target node.**

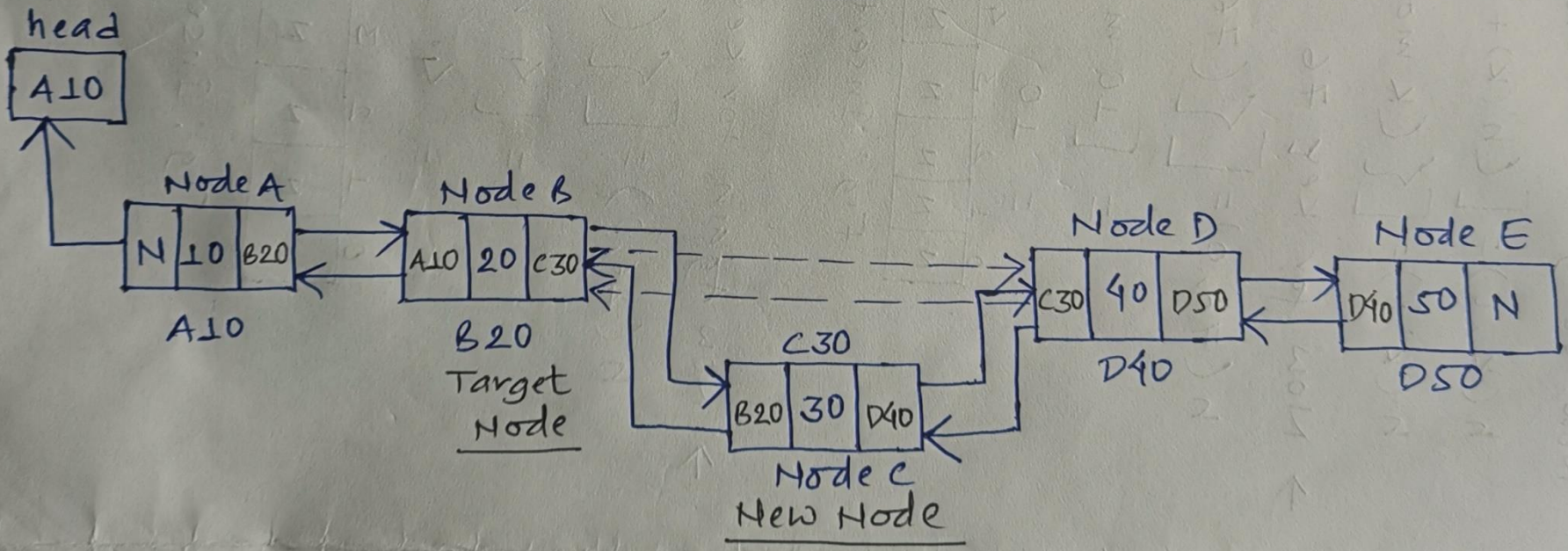
- If the target node is not found
  - Display the message and stop.
- else

Step 5: Copy the address of **right link** of **target node** to the **right link** of **new node.**

Step 6: Copy the content of left link of node pointed by right link of target node to the left link of new node.

Step 7: Assign the address of new node to the right link of target node and left link of node pointed by right link of target node.


head
A10
Target Node
Node A
Node B
Node D
Node E
N
10
B20
A10
20
D40
B20
40
E50
D40
50
N
Address = A10
Address = B20
Address = D40
Address = E50


head
A10
Node A
Node B
Node D
Node E
N
10
B20
A10
20
C30
C30
40
D50
D40
50
N
A10
B20
Target Node
C30
B20
30
D40
D40
D50
Node C
New Node

Program 9

- **Write a C program to create a doubly linked list and insert a new node in beginning, end or at any position in the list.**

Note : for code go through this link

- https://codeforwin.org/2015/10/c-program-to-insert-node-in-doubly-linked-list.html

```
1.  /**
2.  a C program to create a doubly linked list and insert a new node in
    beginning, end or at any position in the list. (double3.cpp)
 */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
/*
 * Basic structure of Node
 */
struct node {
   int data;
   struct node * prev;
   struct node * next;
}*head, *last;
/*
 * Function used in this program
 */
void createList(int n);
void displayList();
void insertAtBeginning(int data);
void insertAtEnd(int data);
void insertAtN(int data, int position);
```

```
int main()
{
   int n, data, choice=1;
   clrscr();
   head = NULL;
   last = NULL;

   /*
    * Run forever until user chooses 0
    */
   while(choice != 0)
   {
                /*
                 * Menu creation to use the program
                 */
                printf("=========================================\
n");
                printf("DOUBLY LINKED LIST PROGRAM\n");
                printf("=========================================\
n");
                printf("1. Create List\n");
                printf("2. Insert node - at beginning\n");
                printf("3. Insert node - at end\n");
                printf("4. Insert node - at N\n");
                printf("5. Display list\n");
                printf("0. Exit\n");
                printf("-----------------------------------------\n");
                printf("Enter your choice : ");

                scanf("%d", &choice);

                /*
                 * Choose from different menu operation
                 */
                switch(choice)
                {
                   case 1:
                                printf("Enter the total number of nodes in list:
");
                                scanf("%d", &n);
                                createList(n);
                                break;
                   case 2:
                                printf("Enter data of first node : ");
                                scanf("%d", &data);

                                insertAtBeginning(data);
                                break;
                   case 3:
                                printf("Enter data of last node : ");
                                scanf("%d", &data);

                                insertAtEnd(data);
                                break;
                   case 4:
                                printf("Enter the position where you want to
insert new node: ");
                                scanf("%d", &n);
                                printf("Enter data of %d node : ", n);
                                scanf("%d", &data);

                                insertAtN(data, n);
                                break;
                   case 5:
                                displayList();
                                break;
                   case 0:
                                break;
                   default:
                                printf("Error! Invalid choice. Please choose
between 0-5");
                }

                printf("\n\n\n\n\n");
   }

   getch();
   return 0;
}
```

```
/* Creates a doubly linked list of n nodes.
 * @n Number of nodes to be created
 */
void createList(int n)
{
   int i, data;
   struct node *newNode;

   if(n >= 1)
   {
      /*
       * Create and link the head node
       */
      head = (struct node *)malloc(sizeof(struct node));

      printf("Enter data of 1 node: ");
      scanf("%d", &data);

      head->data = data;
      head->prev = NULL;
      head->next = NULL;

      last = head;

      /*
       * Create and link rest of the n-1 nodes
       */
      for(i=2; i<=n; i++)
      {
         newNode = (struct node *)malloc(sizeof(struct node));

         printf("Enter data of %d node: ", i);
         scanf("%d", &data);

         newNode->data = data;
         newNode->prev = last; // Link new node with the previous node
         newNode->next = NULL;

         last->next = newNode; // Link previous node with the new node
         last = newNode;         // Make new node as last/previous node
      }

      printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
   }
}
```

```
/**
 * Display content of the list from beginning to end
 */
void displayList()
{
   struct node * temp;
   int n = 1;

   if(head == NULL)
   {
      printf("List is empty.\n");
   }
   else
   {
      temp = head;
      printf("DATA IN THE LIST:\n");

      while(temp != NULL)
      {
         printf("DATA of %d node = %d\n", n, temp->data);

         n++;

         /* Move the current pointer to next node */
         temp = temp->next;
      }
   }
}
```

```
void insertAtBeginning(int data)
{
   struct node * newNode;

   if(head == NULL)
   {
      printf("Error, List is Empty!\n");
   }
   else
   {
      newNode = (struct node *)malloc(sizeof(struct node));

      newNode->data = data;
      newNode->next = head; // Point to next node which is currently head
      newNode->prev = NULL; // Previous node of first node is NULL

      /* Link previous address field of head with newnode */
      head->prev = newNode;

      /* Make the new node as head node */
      head = newNode;

      printf("NODE INSERTED SUCCESSFULLY AT THE BEGINNING OF THE LIST\n");
   }
}
```

```
/**
 * Inserts a new node at the beginning of the doubly linked list
 * @data Data of the first node i.e. data of the new node
 */
```

```
/**
 * Inserts a new node at the end of the doubly linked list
 * @data Data of the last node i.e data of the new node
 */
void insertAtEnd(int data)
{
   struct node * newNode;

   if(last == NULL)
   {
      printf("Error, List is empty!\n");
   }
   else
   {
      newNode = (struct node *)malloc(sizeof(struct node));

      newNode->data = data;
      newNode->next = NULL;
      newNode->prev = last;

      last->next = newNode;
      last = newNode;

      printf("NODE INSERTED SUCCESSFULLY AT THE END OF LIST\n");
   }
}


/**
 * Inserts a node at any position in the doubly linked list
 * @data Data of the new node to be inserted
 * @position Position where to insert the new node
 */
void insertAtN(int data, int position)
{
   int i;
   struct node * newNode, *temp;

   if(head == NULL)
   {
      printf("Error, List is empty!\n");
   }
   else
   {
      temp = head;
      i=1;

      while(i<position-1 && temp!=NULL)
      {
         temp = temp->next;
         i++;
      }

      if(position == 1)
      {
         insertAtBeginning(data);
      }
      else if(temp == last)
      {
         insertAtEnd(data);
      }
      else if(temp!=NULL)
      {
         newNode = (struct node *)malloc(sizeof(struct node));

         newNode->data = data;
         newNode->next = temp->next; // Connect new node with n+1th node
         newNode->prev = temp;       // Connect new node with n-1th node

         if(temp->next != NULL)
         {
            /* Connect n+1th node with new node */
            temp->next->prev = newNode;
         }
         /* Connect n-1th node with new node */
         temp->next = newNode;

         printf("NODE INSERTED SUCCESSFULLY AT %d POSITION\n", position);
      }
      else
      {
         printf("Error, Invalid position\n");
      }
   }
}
```

3. Deletion

In a double linked list, the deletion operation can be performed in three ways. They are as follows.

a. **Deleting from Beginning of the list**

b. **Deleting from End of the list**

c. **Deleting a Specific Node(i.e middle node or at any position)**

- **How to delete first node from double linked list**

NOTE:

[Good site for DSA]

- http://www.btechsmartclass.com/

a. Deleting from Beginning of the list

Algorithm:

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

Step 2: If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

Step 3: If it is not Empty then, define a Node pointer **'temp'** and initialize with **head**.

Step 4: Check whether list is having only one node

(**temp → previous** and **temp → next** both are **NULL.**)

here temp is newNode

Step 5: If it is **TRUE**, then set **head** to **NULL** and delete **temp** (Setting **Empty** list conditions)

Step 6: If it is **FALSE**, then assign **temp → next** to **head** and **Assign the null to the left link of the node pointed by right link of temp node(or first node),** and finally delete **temp**.

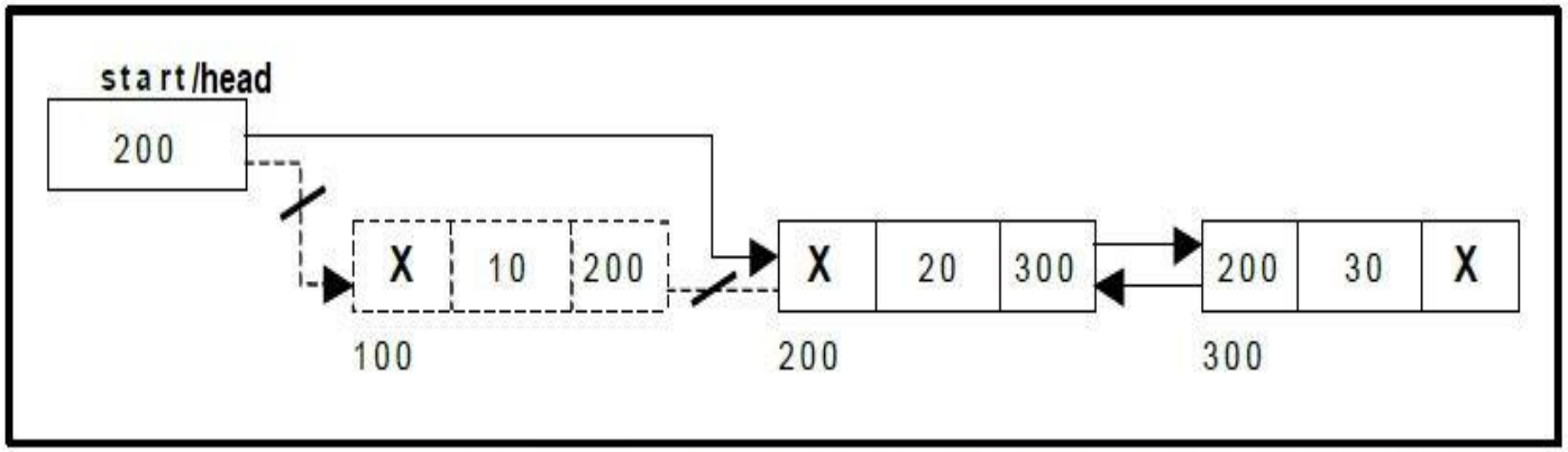


Figure —> Deleting a node at beginning **in doubly linked list**

- **How to delete last node from doubly linked list**

b. Deleting from End of the list

Algorithm: [METHOD 1]

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

Step 2: If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

Step 3: If it is not Empty then, define a Node pointer **'temp'** and initialize with **head**.

Step 4: Check whether list is having only one node

(**temp → previous** and **temp → next** both are **NULL.**)

here temp is newNode

Step 5: If it is **TRUE**, then set **head** to **NULL** and delete **temp** (Setting **Empty** list conditions)

Step 6: If it is **FALSE**, then keep moving **temp** until it reaches to the last node in the list. (until **temp → next** is equal to **NULL**)

Step 7: Assign **NULL** to **temp → previous → next** , again put the NULL value to the right address filed of node which is pointed by the left field of temp node and finally delete **temp**

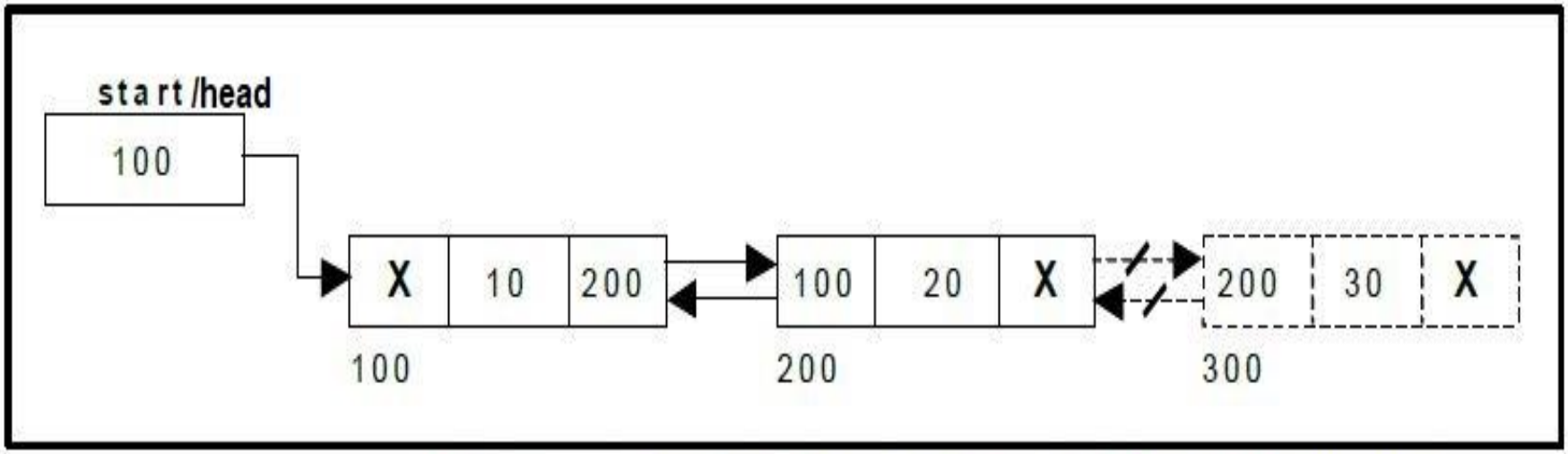


Figure → Deleting a node at the end **in doubly linked list**

- **How to delete a node at the middle (or at any position) of a Doubly Linked List.**

c. delete middle node (or at any position) from doubly linked list

Algorithm

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

Step 2: If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

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

Step 4: Check whether list has only one Node (**temp1 → next** == **NULL**)

Step 5: If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)

Step 6: If it is **FALSE** then Traverse the doubly linked list to search the **target node. (i.e.** exact node which we want to delete**)**

Step 7: If the target node is not found

Display the message and stop.

else

Step 8: Copy the content of right link of target node to the right link of a node which is pointed by the left link of target node.

Step 9: Copy the content of left link of target node to the left link of a node pointed by the right link of target node.

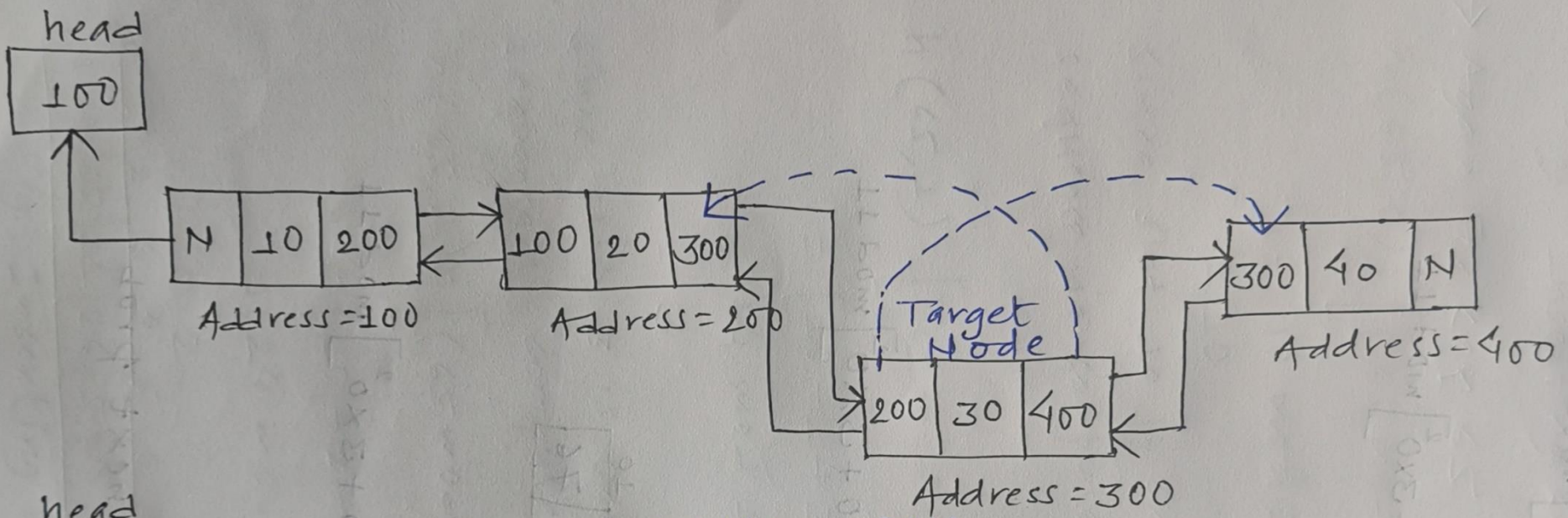


Fig: Initial condition

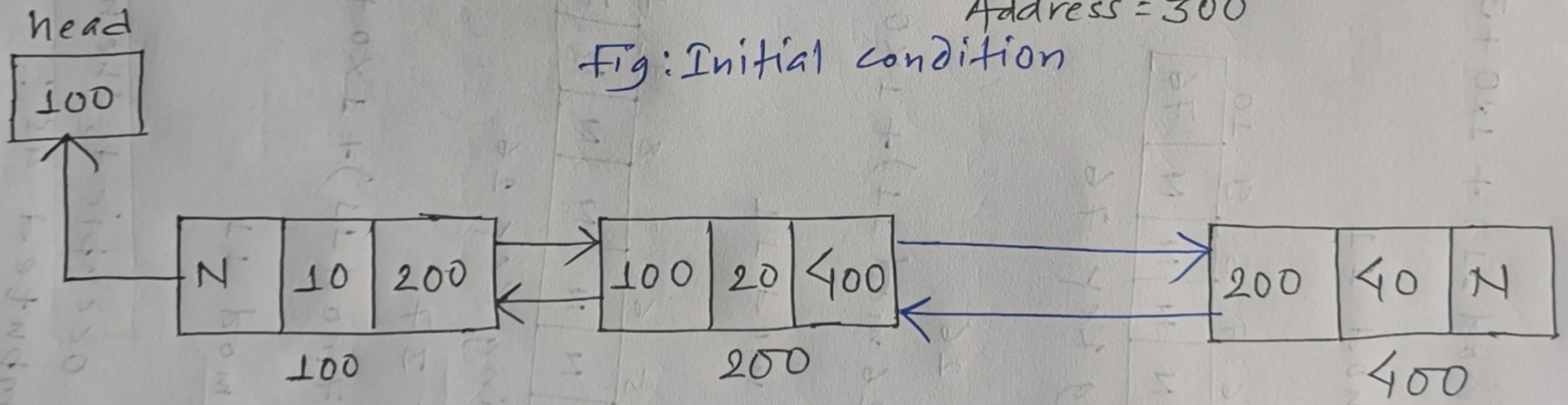


Fig: Final Linked List

Program 10

- **Write a C program to create a doubly linked list and delete a node from beginning, end or at any position of the linked list.**

Note: go through this link for code:

- https://codeforwin.org/2015/10/c-program-to-delete-node-from-doubly-linked-list.html

```
/**
 a C program to create a doubly linked list and delete a node from beginning, end or at any position of the linked list. (double2.cpp)
 */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
/*
 * Basic structure of Node
 */
struct node {
   int data;
   struct node * prev;
   struct node * next;
}*head, *last;
/*
 * Functions used in this program
 */
void createList(int n);
void displayList();
void deleteFromBeginning();
void deleteFromEnd();
void deleteFromN(int position);
```

```
int main()
{
   int n, data, choice=1;
   clrscr();
   head = NULL;
   last = NULL;

   /*
    * Run forever until user chooses 0
    */
   while(choice != 0)
   {
                printf("================================
=======\n");
                printf("DOUBLY LINKED LIST PROGRAM\n");
                printf("================================
=======\n");
                printf("1. Create List\n");
                printf("2. Delete node - from beginning\n");
                printf("3. Delete node - from end\n");
                printf("4. Delete node - from N\n");
                printf("5. Display list\n");
                printf("0. Exit\n");
                printf("----------------------------------------\n");
                printf("Enter your choice : ");

                scanf("%d", &choice);

                switch(choice)
                {
                   case 1:
                                printf("Enter the total number of
nodes in list: ");
                                scanf("%d", &n);
                                createList(n);
                                break;
                   case 2:
                                deleteFromBeginning();
                                break;
                   case 3:
                                deleteFromEnd();
                                break;
                   case 4:
                                printf("Enter the node position which
you want to delete: ");
                                scanf("%d", &n);
                                deleteFromN(n);
                                break;
                   case 5:
                                displayList();
                                break;
                   case 0:
                                break;
                   default:
                                printf("Error! Invalid choice. Please
choose between 0-5");
                }

                printf("\n\n\n\n\n");
   }
   getch();
   return 0;
}
```

```
/**
 * Creates a doubly linked list of n nodes.
 * @n Number of nodes to be created
 */
void createList(int n)
{
   int i, data;
   struct node *newNode;

   if(n >= 1)
   {
            /*
             * Creates and links the head node
             */
            head = (struct node *)malloc(sizeof(struct
node));

            printf("Enter data of 1 node: ");
            scanf("%d", &data);

            head->data = data;
            head->prev = NULL;
            head->next = NULL;

            last = head;

            /*
             * Create and link rest of the n-1 nodes
             */
                        for(i=2; i<=n; i++)
                        {
                           newNode = (struct node
*)malloc(sizeof(struct node));

                           printf("Enter data of %d node: ", i);
                  scanf("%d", &data);

                  newNode->data = data;
                  newNode->prev = last; // Link new node with the
previous node
                  newNode->next = NULL;

                  last->next = newNode; // Link previous node with
the new node
                  last = newNode; // Make new node as last node
               }

                        printf("DOUBLY LINKED LIST CREATED
SUCCESSFULLY\n");
   }
}
```

```
/**
 * Display the content of the list from beginning to end
 */
void displayList()
{
   struct node * temp;
   int n = 1;

   if(head == NULL)
   {
                printf("List is empty.\n");
   }
   else
   {
                temp = head;
                printf("DATA IN THE LIST:\n");

                while(temp != NULL)
                {
                   printf("DATA of %d node = %d\n", n, temp->data);

                   n++;

                   /* Move the current pointer to next node */
                   temp = temp->next;
                }
   }
}


/**
 * Delete or remove the first node of the doubly linked list
 */
void deleteFromBeginning()
{
   struct node * toDelete;

   if(head == NULL)
   {
                printf("Unable to delete. List is empty.\n");
   }
   else
   {
                toDelete = head;

                head = head->next; // Move head pointer to 2 node

                if (head != NULL)
                   head->prev = NULL; // Remove the link to previous
node

                free(toDelete); // Delete the first node from memory
                printf("SUCCESSFULLY DELETED NODE FROM
BEGINNING OF THE LIST.\n");
   }
}
```

```
/**
 * Delete or remove the last node of the doubly linked list
 */
void deleteFromEnd()
{
   struct node * toDelete;

   if(last == NULL)
   {
                printf("Unable to delete. List is empty.\n");
   }
   else
   {
                toDelete = last;

                last = last->prev; // Move last pointer to 2nd last
node

                if (last != NULL)
                   last->next = NULL; // Remove link to of 2nd last
node with last node

                free(toDelete);       // Delete the last node
                printf("SUCCESSFULLY DELETED NODE FROM END OF
THE LIST.\n");
   }
}
/**
 * Delete node from any position in the doubly linked list
 */
void deleteFromN(int position)
{
   struct node *current;
   int i;

   current = head;
   for(i=1; i<position && current!=NULL; i++)
   {
                current = current->next;
   }

   if(position == 1)
   {
                deleteFromBeginning();
   }
   else if(current == last)
   {
                deleteFromEnd();
   }
   else if(current != NULL)
   {
                current->prev->next = current->next;
                current->next->prev = current->prev;

                free(current); // Delete the n node

                printf("SUCCESSFULLY DELETED NODE FROM %d
POSITION.\n", position);
   }
   else
   {
                printf("Invalid position!\n");
   }
}
```

Program 11

- **Write a C program to create a doubly linked list and reverse the linked list. How to reverse the doubly linked list in C programming.**

Note: click this link for reverse the linked list.

- https://codeforwin.org/2015/11/c-program-to-reverse-doubly-linked-list.html

```
/**
 * C program to reverse a Doubly linked list
 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

/*
 * Basic structure of Node
 */
struct node {
   int data;
   struct node * prev;
   struct node * next;
}*head, *last;

/*
 * Functions used in this program
 */
void createList(int n);
void displayList();
void reverseList();
```

```
int main()
{
   int n, data, choice=1;
   clrscr();
   head = NULL;
   last = NULL;

   /*
    * Runs forever until user chooses 0
    */
   while(choice != 0)
   {
            printf("================================
============\n");
            printf("DOUBLY LINKED LIST PROGRAM\n");
            printf("================================
============\n");
            printf("1. Create List\n");
            printf("2. Reverse List\n");
            printf("3. Display list\n");
            printf("0. Exit\n");
            printf("----------------------------------------\n");
            printf("Enter your choice : ");

            scanf("%d", &choice);

            switch(choice)
            {
               case 1:
                        printf("Enter the total number of
nodes in list: ");
                        scanf("%d", &n);
                        createList(n);
                        break;
               case 2:
                        reverseList();
                        break;
               case 3:
                        displayList();
                        break;
               case 0:
                        break;
               default:
                        printf("Error! Invalid choice.
Please choose between 0-3");
                        }

                        printf("\n\n\n\n\n");
   }
   getch();
   return 0;
}
```

```
/**
 * Creates a doubly linked list of n nodes.
 * @n Number of nodes to be created
 */
void createList(int n)
{
   int i, data;
   struct node *newNode;

   if(n >= 1)
   {
      /*
       * Create and link head node
       */
      head = (struct node *)malloc(sizeof(struct node));

      printf("Enter data of 1 node: ");
      scanf("%d", &data);

      head->data = data;
      head->prev = NULL;
      head->next = NULL;

      last = head;

      /*
       * Create and link rest of the n-1 nodes
       */
      for(i=2; i<=n; i++)
      {
         newNode = (struct node *)malloc(sizeof(struct node));

         printf("Enter data of %d node: ", i);
         scanf("%d", &data);

         newNode->data = data;
         newNode->prev = last; // Link new node with the previous node
         newNode->next = NULL;

         last->next = newNode; // Link previous node with the new node
         last = newNode; // Make new node as last/previous node
      }

      printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
   }
}
```

```
/**
 * Display the content of the list from beginning to end
 */
void displayList()
{
   struct node * temp;
   int n = 1;

   if(head == NULL)
   {
      printf("List is empty.\n");
   }
   else
   {
      temp = head;
      printf("DATA IN THE LIST:\n");

      while(temp != NULL)
      {
         printf("DATA of %d node = %d\n", n, temp->data);

         n++;

         /* Move pointer to next node */
         temp = temp->next;
      }
   }
}


/**
 * Reverse order of the doubly linked list
 */
void reverseList()
{
   struct node *current, *temp;

   current = head;
   while(current != NULL)
   {
      /*
       * Swap the previous and next address fields of current
node
       */
      temp = current->next;
      current->next = current->prev;
      current->prev = temp;

      /* Move the current pointer to next node which is stored
in temp */
      current = temp;
   }

   /*
    * Swap the head and last pointers
    */
   temp = head;
   head = last;
   last = temp;

   printf("LIST REVERSED SUCCESSFULLY.\n");
}
```

Advantages of Doubly linked list

- Allows traversal of nodes in both direction which is not possible in singly linked list.
- Deletion of nodes is easy when compared to singly linked list, as in singly linked list deletion requires a pointer to the node and previous node to be deleted. Which is not in case of doubly linked list we only need the pointer which is to be deleted.
- Reversing the list is simple and straightforward.
- Can allocate or de-allocate memory easily when required during its execution.
- It is one of most efficient data structure to implement when traversing in both direction is required.

Disadvantages of Doubly linked list

- It uses extra memory when compared to array and singly linked list.

- Insertion and deletion take more time than linear linked list because more pointer operations are required than linear linked list.

Implementing Stacks and Queues with Linked Lists

- Because linked lists store data elements in linear sequences, they can be used to give alternative implementations of stacks and queues.
- One advantage to using linked lists is that we don't have to worry about filling up something like an array - we can just keep allocating cells as long as we need to (unless we run out of memory).

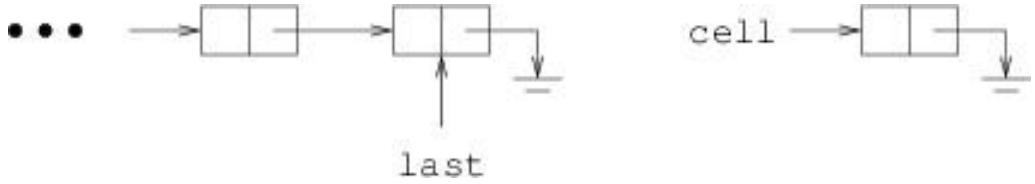
- Implementing a stack using a linked list is particularly easy because all accesses to a stack are at the top. One end of a linked list, the beginning, is always directly accessible. We should therefore arrange the elements so that the top element of the stack is at the beginning of the linked list, and the bottom element of the stack is at the end of the linked list. We can represent an empty stack with **null**.

- The **public** methods **Push**, **Peek**, and **Pop** are then fairly straightforward to implement. For **Push** we need to add the given element to a new cell at the beginning of the linked list, as shown in the previous section, and update the **Count**. To implement **Peek**, if the stack is nonempty, we simply return the **Data** property of the cell at the beginning of the linked list; otherwise, we throw an **InvalidOperationException**. To implement **Pop**:
- Using **Peek**, obtain the element to be returned;
- Remove the first element from the linked list as shown in the previous section;
- Update the **Count**; and
- Return the retrieved value.

- Implementing a queue is a little more involved because we need to operate at both ends of the linked list. For efficiency, we should keep a reference to the last cell in the linked list, as this will allow us to access both ends of the linked list directly.

We will therefore have the following:

- We now need to decide which end to make the front of the queue. As we saw in the previous section, both inserting and removing can be done efficiently at the beginning of a linked list. Likewise, it is easy to insert an element at the end if we have a reference to the last cell. Suppose, for example, that last refers to the last cell in a linked list, and that cell refers to a **LinkedListCell<T>** that we want to insert at the end. Suppose further that the linked list is not empty (that will be a special case that we'll need to handle).



We therefore need to:

Construct a new **LinkedListCell<T>**;

Assign it to the field denoting the front of the queue;

Assign it to the field denoting the back of the queue;

Store the given element in its **Data** property; and

Update the **Count**.

Note that there is no need to initialize the new cell's **Next** property, as it will automatically be initialized to **null**.

If the queue is nonempty, the only step that changes is Step 2. Because the queue is nonempty, we don't want to make the new cell the front of the queue; instead, we need to insert it at the end of the linked list, as outlined above. The implementations of the **Peek** and **Dequeue** methods are essentially the same as the implementations of the **Peek** and **Pop** methods, respectively, for a stack.

- The implementations described in this section are simpler than the implementations using arrays, mainly due to the fact that we don't need to rebuild the structure when we fill up the space available. While these implementations are also pretty efficient, it turns out that the array-based implementations tend to out-perform the linked-list-based implementations. This might be counterintuitive at first because rebuilding the structures when the array is filled is expensive. However, due to the fact that we double the size of the array each time we need a new one, this rebuilding is done so rarely in practice that it ends up having minimal impact on performance. Due to hardware and low-level software issues, the overhead involved in using arrays usually ends up being less.

Assignment

1. What is linked list? Write down the advantages and disadvantage of linked list.
2. What is singly linked list? Explain with an example.
3. Write down the basic structure of single linked list. What are the basic operation of single linked list?
4. Write down the algorithm to insert a node at the beginning of a single linked list.
5. Write down the algorithm to insert a node at the end of a single linked list.
6. Write down the algorithm to insert a node at the middle of a single linked list.
7. Write down the algorithm to delete a node at the beginning of a single linked list.
8. Write down the algorithm to delete a node at the end of a single linked list.
9. Write down the algorithm to delete a node at the middle of a single linked list.

10. What are the advantages and disadvantages of singly linked list?
11. What is doubly linked list? Explain with an example.
12. Write down the basic structure of double linked list. What are the basic operation of double linked list?
13. Write down the algorithm to insert a node at the beginning of a double linked list.
14. Write down the algorithm to insert a node at the end of a double linked list.
15. Write down the algorithm to insert a node at the middle of a double linked list.
16. Write down the algorithm to delete a node at the beginning of a double linked list.
17. Write down the algorithm to delete a node at the end of a double linked list.
18. Write down the algorithm to delete a node at the middle of a double linked list.

19. What are the advantages and disadvantages of doubly linked list?
20. **What are the advantages & disadvantages of doubly linked list over singly linked list? Justify with an example.**