Data Structures
Unit III

Kousalya. G

Professor

Department of Computer Science and Engineering

Dayananda Sagar University, Harohalli, Karnataka.

Email-id: kousalya-cse@dsu.edu.in

Faculty Cabin: A439

Outline – Unit III

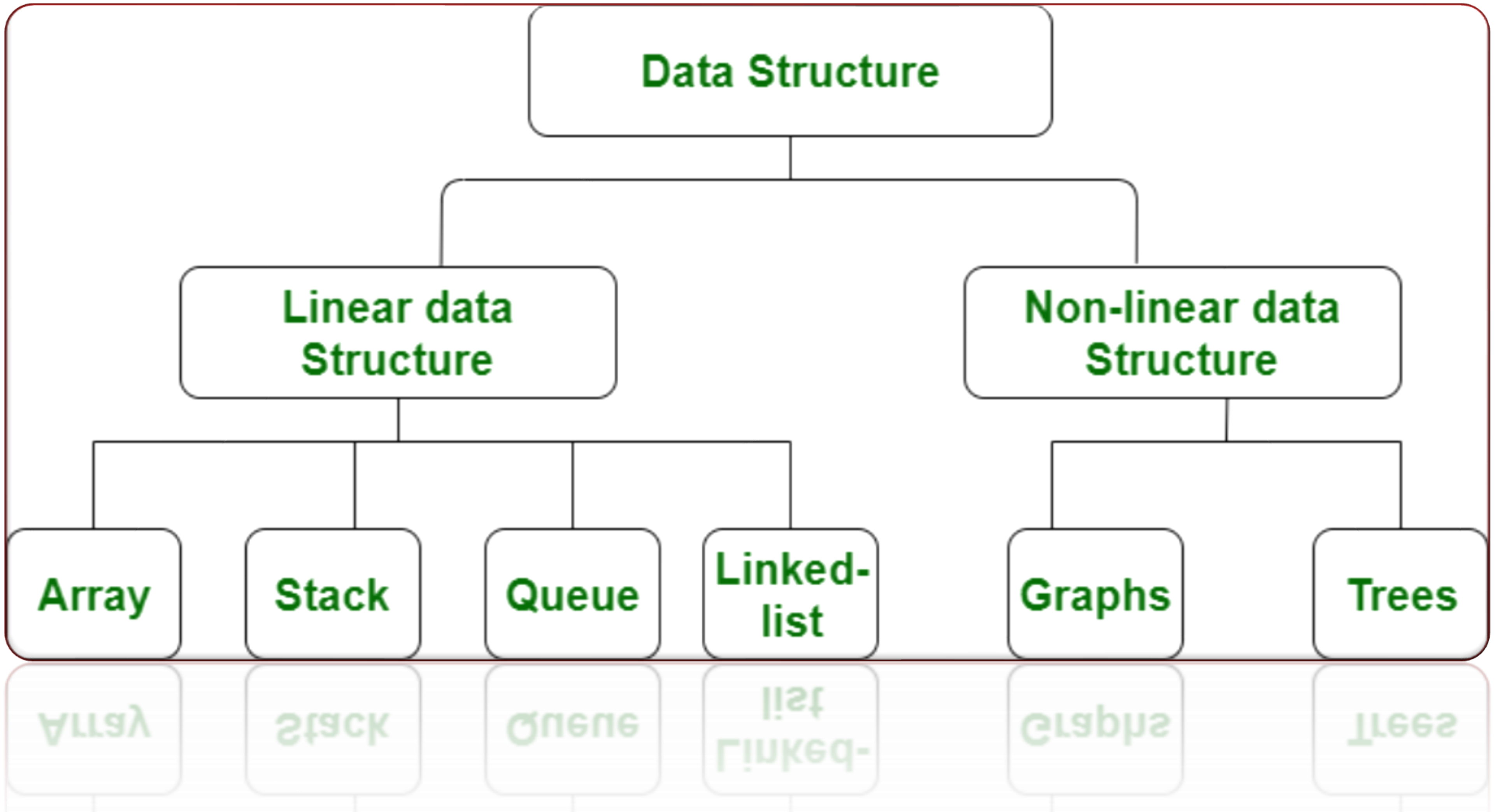
- ✓ Linked List
  - Representation of Linked Lists in Memory.
  - Insertion
  - Searching
  - Traversing
  - Deletion
- ✓ Circular List
- ✓ Doubly Linked List
- ✓ Operations on Doubly Linked List

  (Insertion, Deletion, Traversal).
- ✓ Applications

  Stack & Queue Implementation using Linked Lists.
- ✓ Case Study

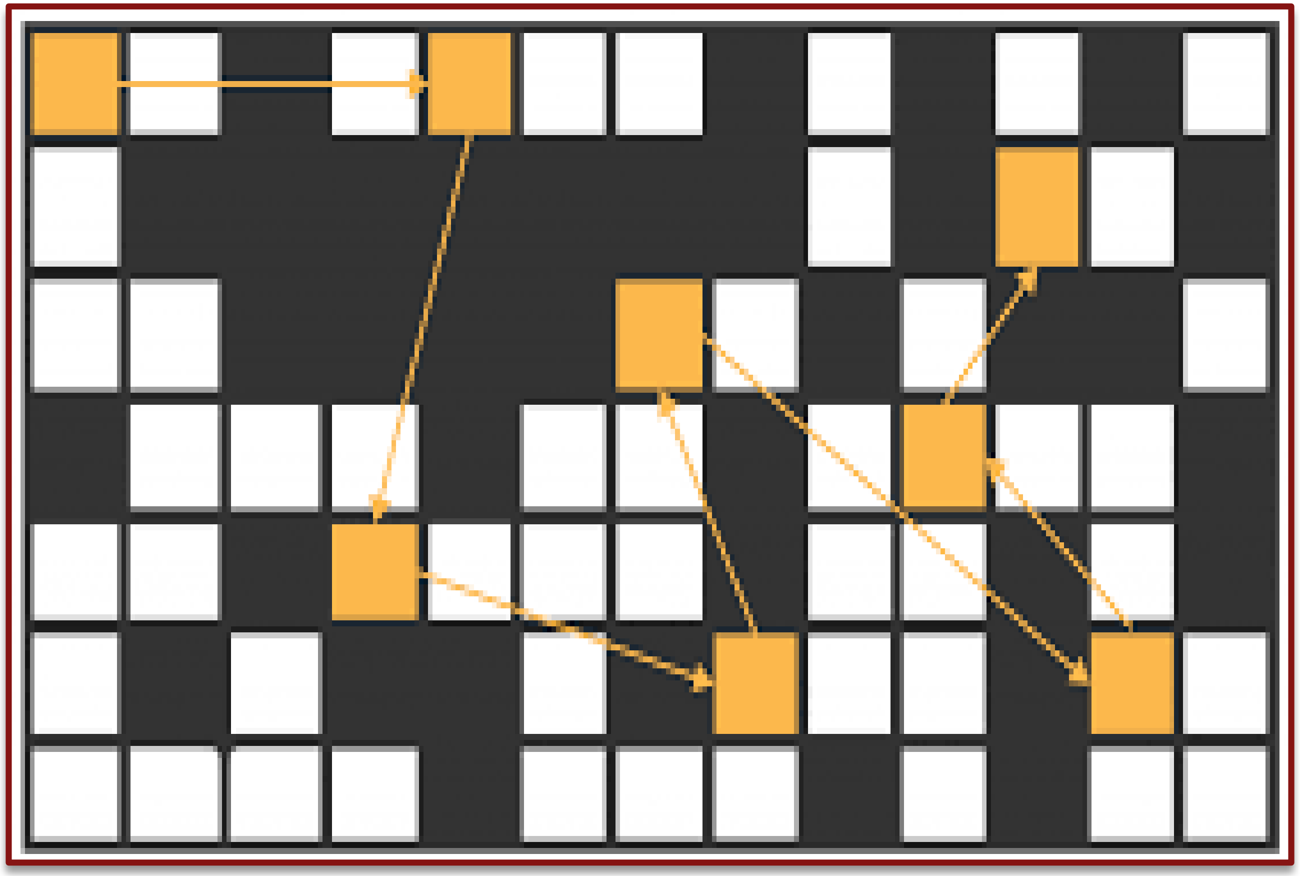
  Josephus problem.

Data Structure-Classification



How to do?

- Store every element in its own block of memory ?.
- How to link the individual blocks?
- How to link in any order? (in front or at the end or in the middle?

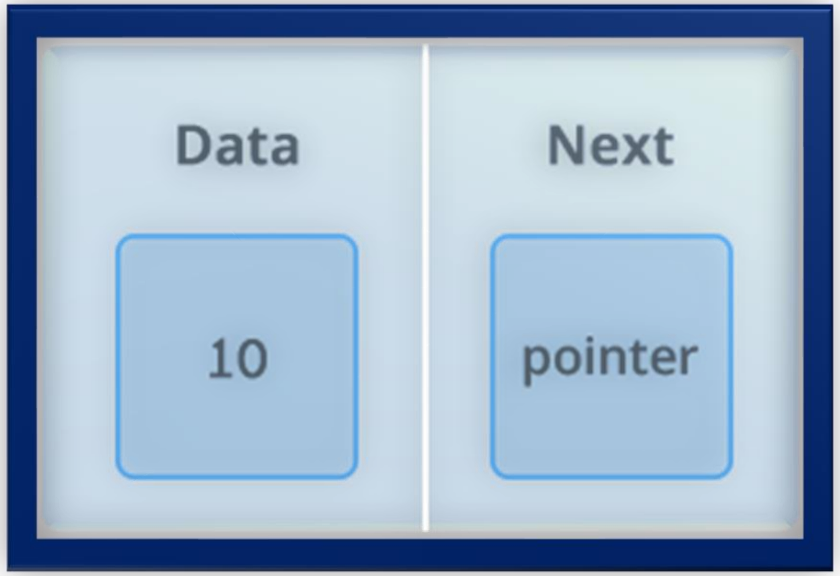


Types of Linked List

- Singly Linked List
- Doubly Linked List
- Circular Linked List
- Circular Doubly Linked List
- Header Linked List

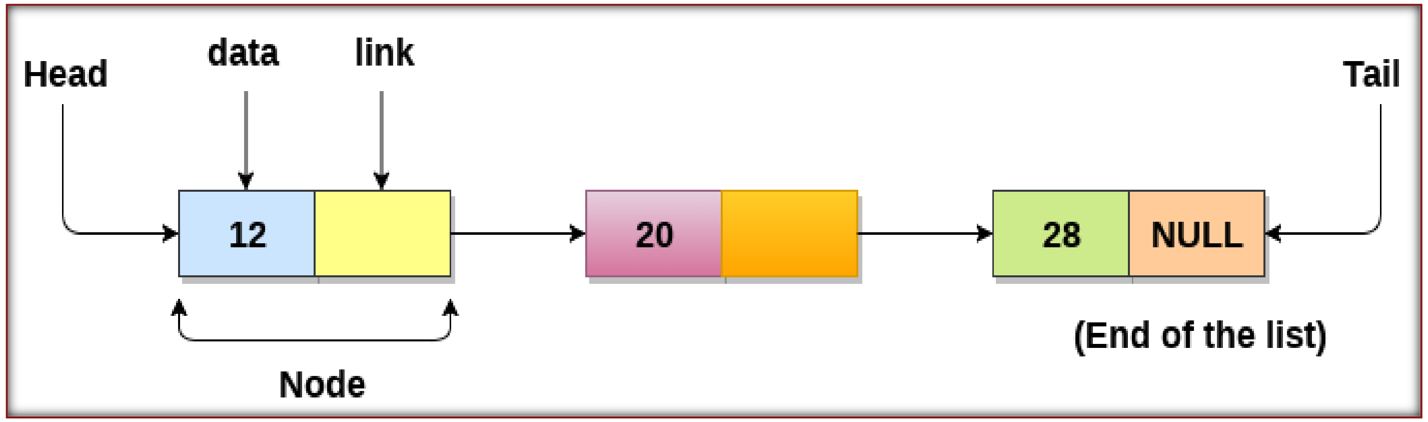
Linked List (Dynamic Data Structure)

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
- For every data item in a linked list, there is an associated pointer that would give the memory location of the next data item in the linked list.

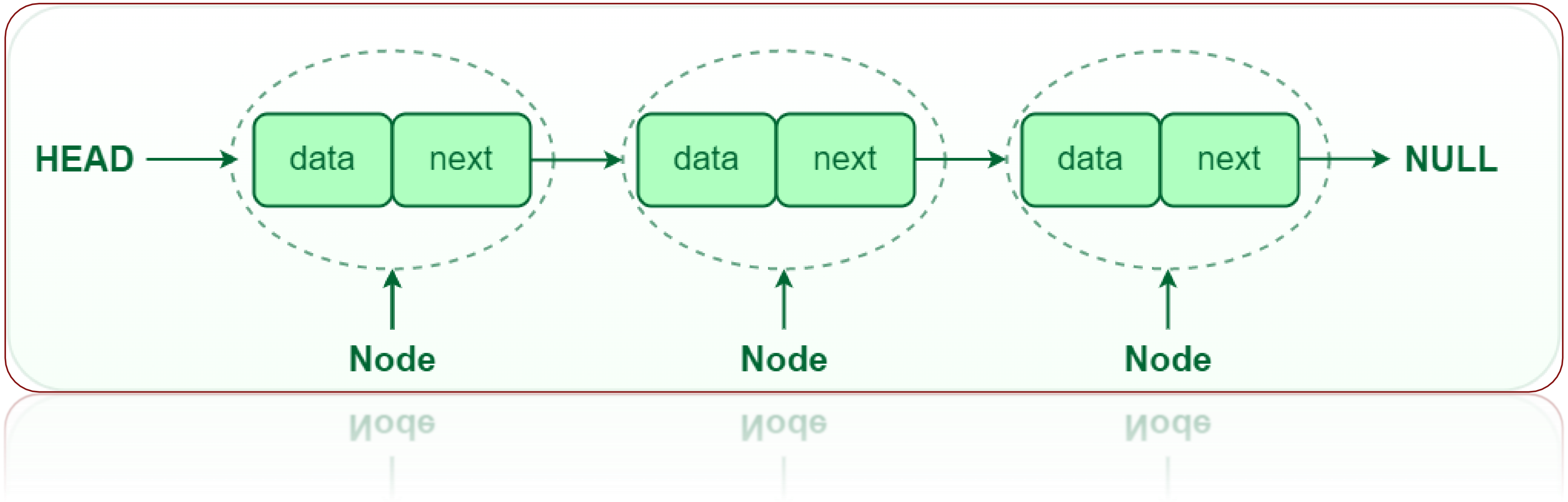


Linked List

- A list can be defined as an ordered collection.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



Linked List Representation



Node

Data + Pointer to the next node

Dynamic Data Structure

A linked list can change during execution (Dynamic Data Structure)

- Successive elements are connected by pointers.
- Last element points to NULL.
- It can grow or shrink in size during execution of a program.
- It can be made just as long as required.
- It does not waste memory space.

Basic operations of Linked List

• Creation	- Creates a linked list
• Insertion	– Adds an element to the list.
• Deletion	– Deletes an element to the list.
• Display	– Displays the complete list.
• Search	– Searches an element using the given key.
• Delete	– Deletes an element using the given key.

Algorithm (Inserting a node at the beginning of a linked list)

1. Declare head pointer and make it as NULL.

2. Create a new node with the given data.
   And make the new node => next as NULL.
   (Because the new node is going to be the last node.)

3. If the head node is NULL (Empty Linked List),
   make the new node as the head.

4. If the head node is not NULL , (Linked list already has some elements),
   find the last node.
   make the last node => next as the new node.

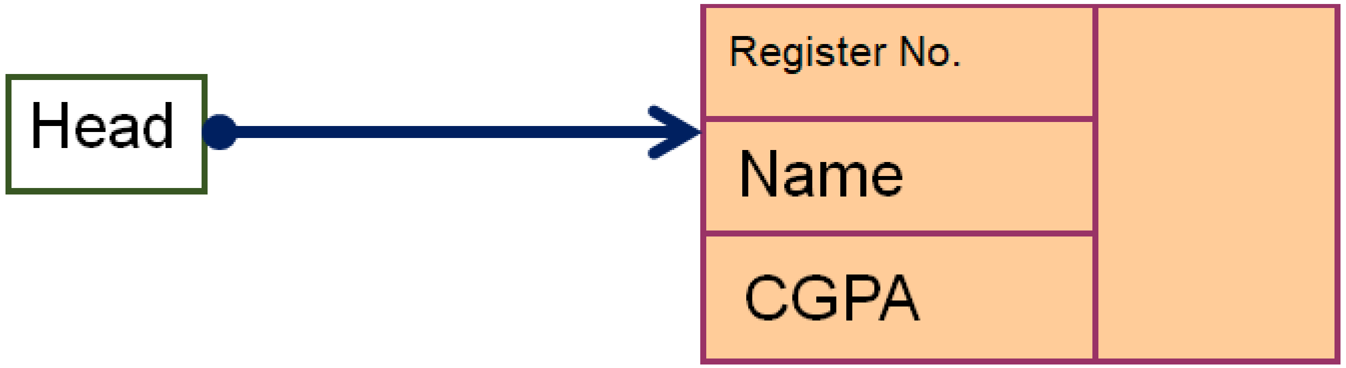
https://onlinegdb.com/MPl6x3D-Y

Creation

To start with

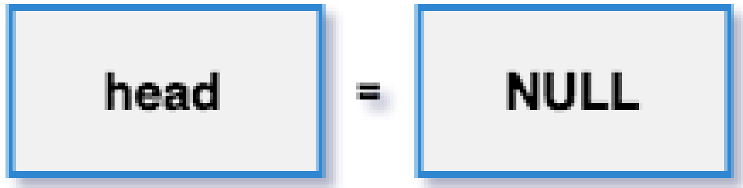
create a node (the first node), and make Head point to it.

```
 struct Node
{
   int data;
   struct Node* next;
};
```



1. Declare a head pointer

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

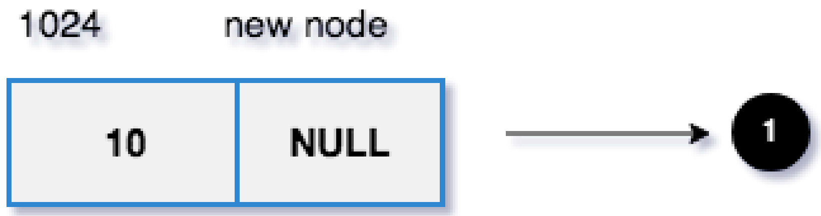


Make it as NULL

https://www.onlinegdb.com/online_c_compiler#

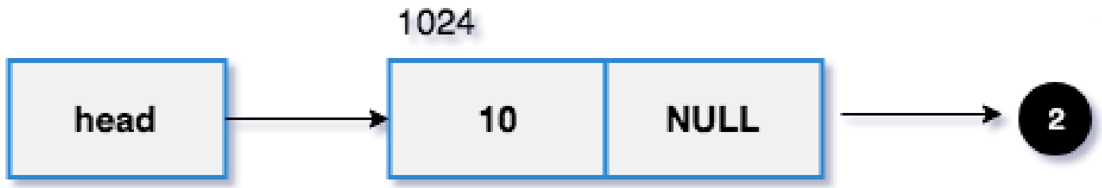
2.Create a new node with the given data.

```
void addFirst(struct node **head, int val)
{
    //Creating a New Node
     struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;

}
```



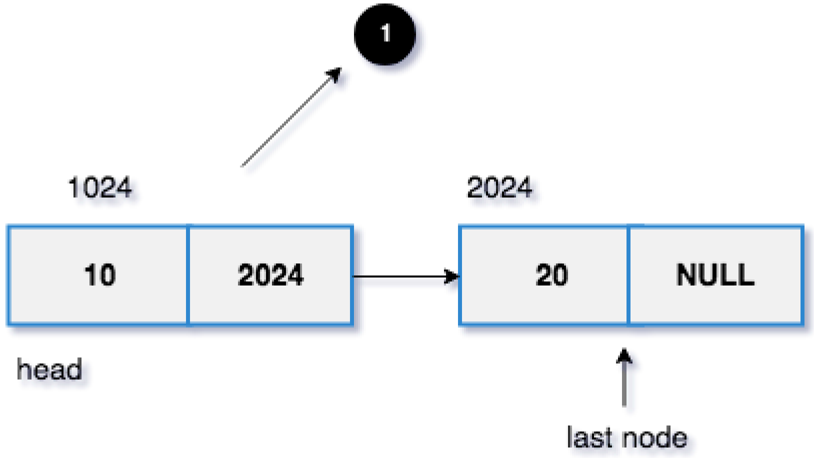
3 .Make the new node points to the head node

```
void addFirst(struct node **head, int val)
{
    //create a new node
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;


    newNode->next = *head;


}
```

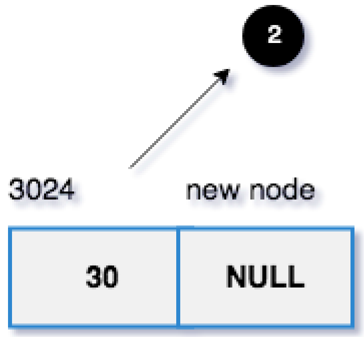


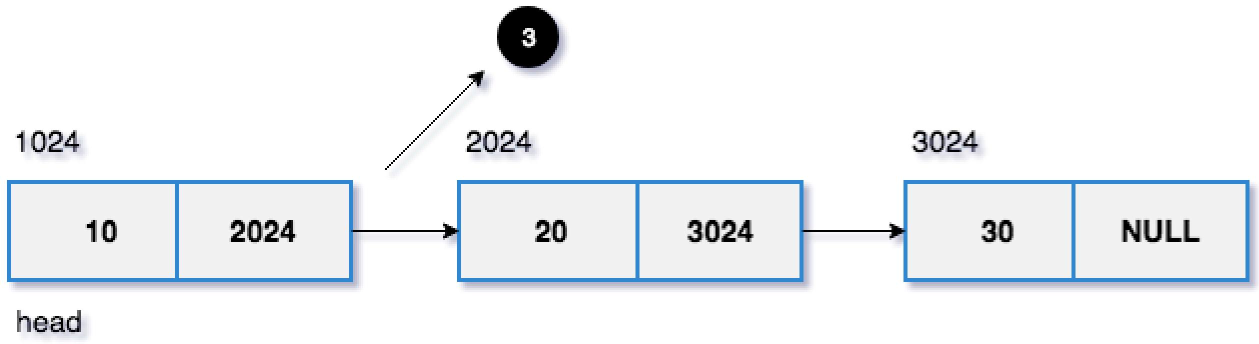
4. Make the new node as the head node.

```
void addFirst(struct node **head, int val)
{
        //creating  a new node
        struct node *newNode = malloc(sizeof(struct node));
        newNode->data = val;
        newNode->next = *head;
       *head = newNode;
 }
```



URL: https://www.onlinegdb.com/myfiles/I5WFRzSFa





Inserting node at the end of the list

Algorithm

1. Declare head pointer and make it as NULL.
2. Create a new node with the given data. And make the new node => next as NULL.

    (Because the new node is going to be the last node.)

3. If the head node is NULL (Empty Linked List),

        make the new node as the head.

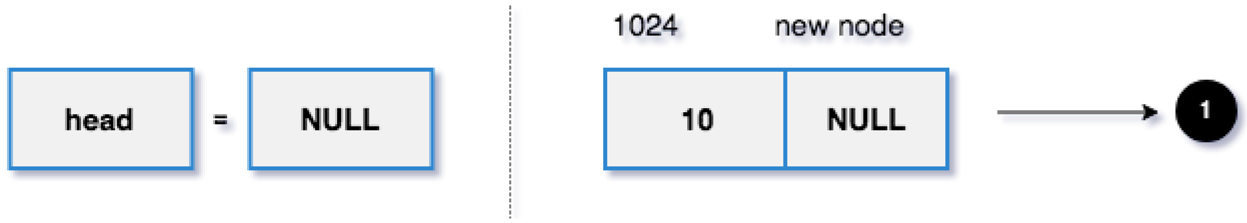
4. If the head node is not null, (Linked list already has some elements),

        find the last node.

        Make the last node => next as the new node.

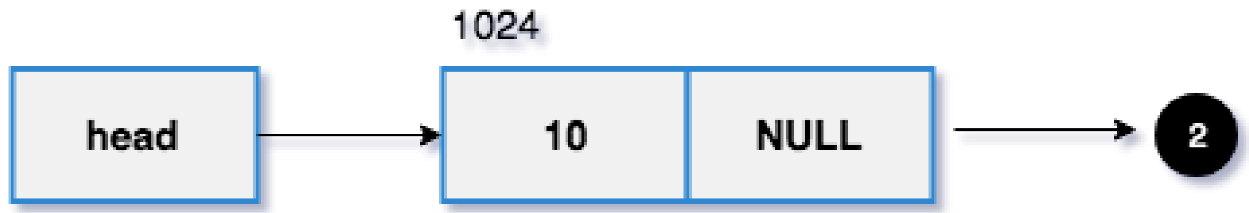
1. Declare head pointer and make it as NULL.

```
struct node
{
    int data;
    struct node *next;
};

struct node *head = NULL;
```



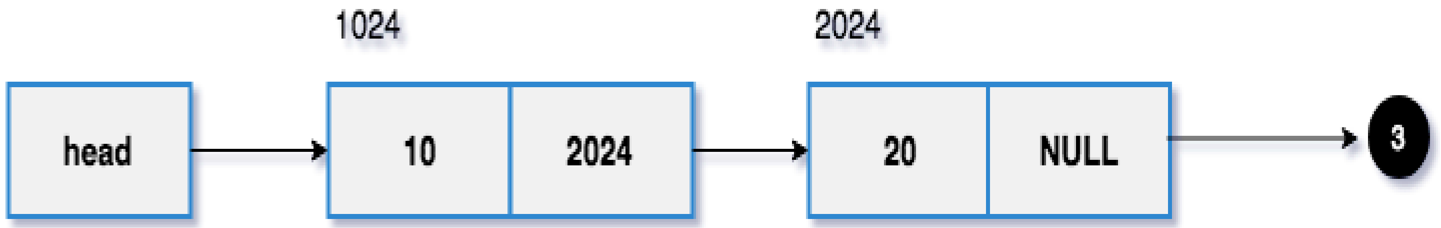
2. Create a new node

```
void addLast(struct node **head, int val)
{
    //create a new node
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next = NULL;
}
```



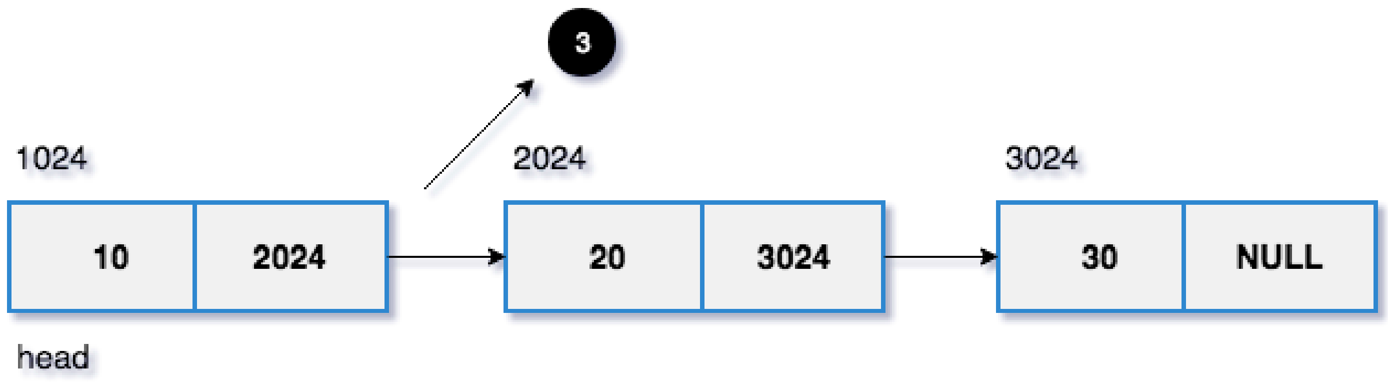
3. If the head node is NULL, make the new node as head

```
void addLast(struct node **head, int val)
{
    //create a new node
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next     = NULL;
     //if head is NULL, it is an empty list
    if(*head == NULL)
        *head = newNode;
    }
```



4. Find the last node and set last node => new node

```
while(node->next != NULL)
{
    node = node->next;
}
```



Sample Linked List Implementation

Example

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  //node structure
  struct node
  {
      int regno;
      char name[20];
      struct node *next;
  };
;
```

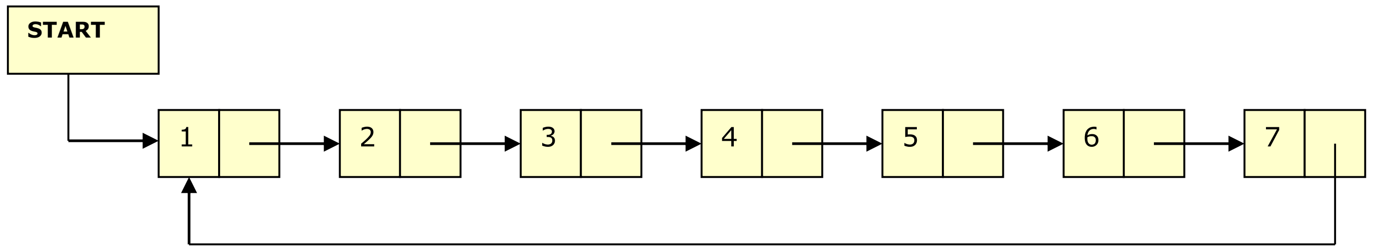
```
//declaring nodes
  struct node *head,*middle,*last;
  //allocating memory for each node
  head   = malloc(sizeof(struct node));
  middle = malloc(sizeof(struct node));
  last   = malloc(sizeof(struct node));

  //assigning values to each node
  head->regno   = 10;
  head->name=" DSU";
  middle->regno = 20;
  head->name ="DSE";
  last->regno  = 30;
 head->name="DSI"
 //connecting each nodes. head->middle->last
  head->next    = middle;
  middle->next = last;
  last->next    = NULL;
   //temp is a reference for head pointer.
  struct node *temp = head;
   //till the node becomes null, printing each nodes data
  while(temp != NULL)
  {
      printf("\n Regno: %d   Name: %s => ",temp->regno,temp->na
me );
      temp = temp->next;
  }
  printf("NULL");
   return 0;
}
```

Circular Linked List

*The **circular linked list** is a linked list where all nodes are connected to form a circle.*

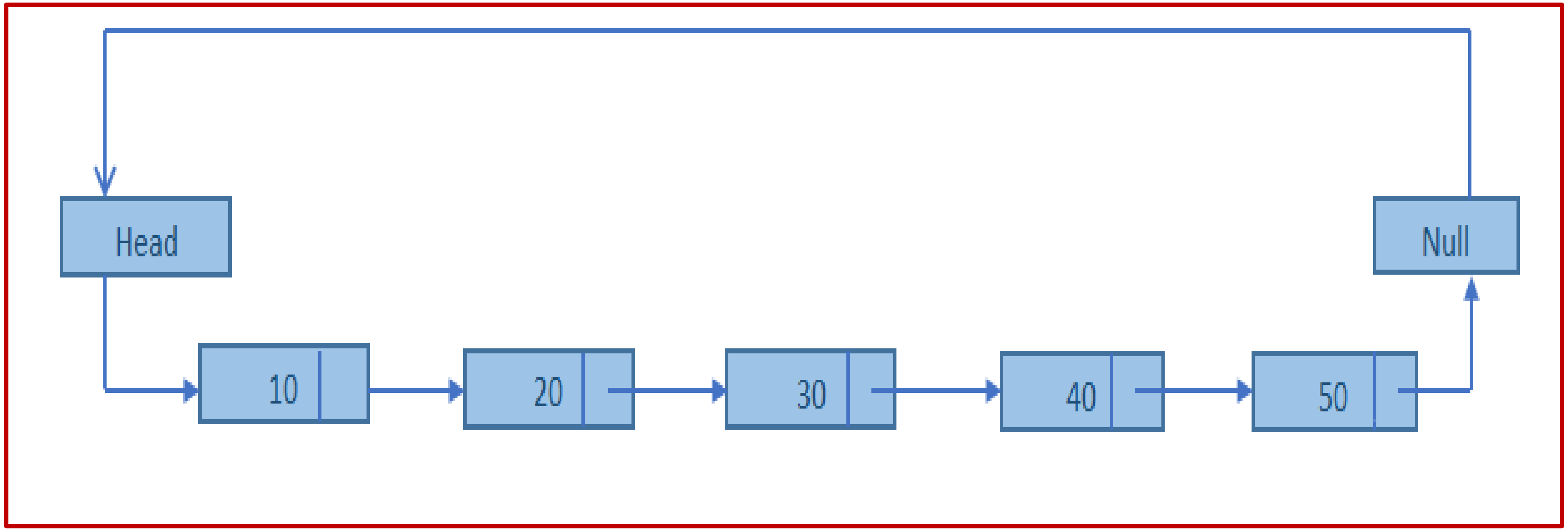
In a circular linked list, the first node and the last node are connected to each other which forms a circle.

There is no NULL at the end.



Circular linked list

Circular list is a list in which the link field of the last node is made to point to the start/first node of the list.



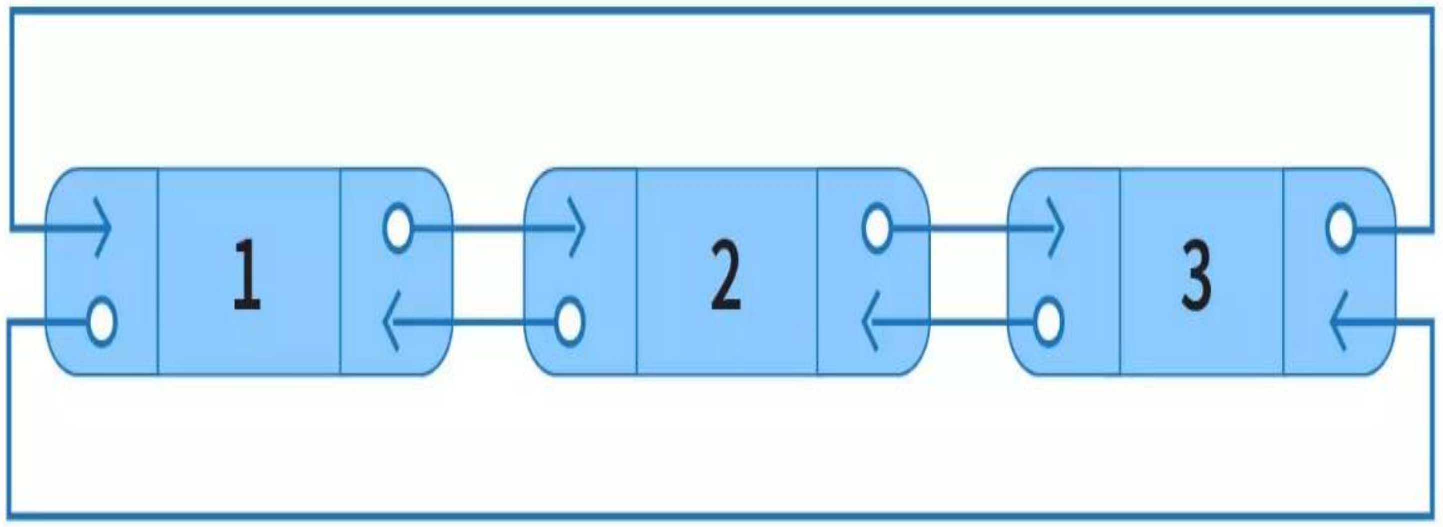
1.A circular linked list is a linked list in which the head element's previous pointer points to the tail element and the tail element's next pointer points to the head element.

2.A circularly linked list node looks exactly the same as a linear singly linked list.

Circular Linked List

In a circular linked list, each node has a data element and a pointer/reference to the next node in the sequence.

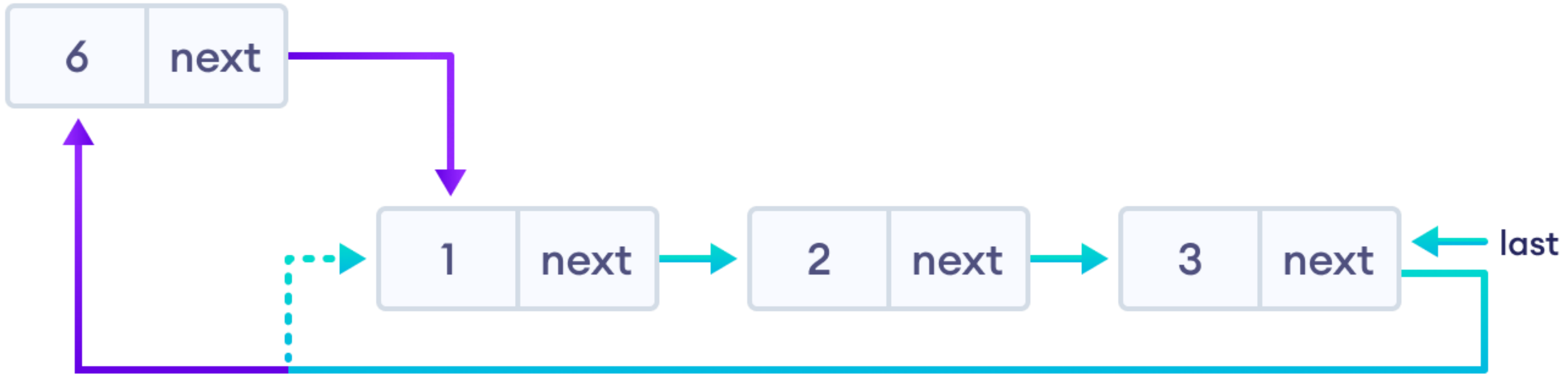
The last node in the list points back to the first node, and the traversal of the list can continue indefinitely in a loop.



1. Insertion at the Beginning

store the address of the current first node in the newNode (i.e. pointing the newNode to the current first node)

point the last node to newNode (i.e making newNode as head)
Insert at the beginning



Insertion at beginning in linked list the steps are followed :-

1.Make the linked list.

2.Then we have to take an extra pointer which points to the end node of the circular linked list.

3.Then we have a pointer that is pointing to the end node, then end node-> next will point to the first node.

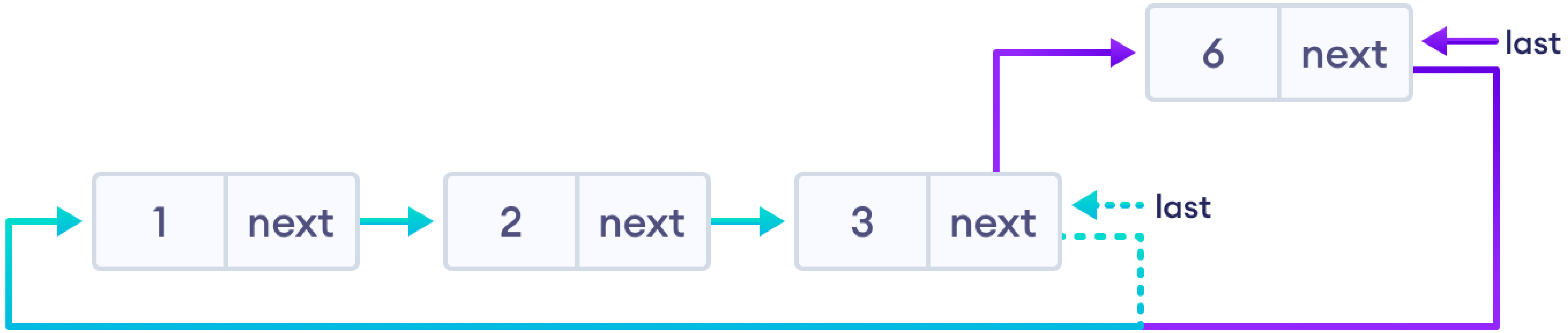
4.At last follow the algorithm for insertion at beginning in circular linked list given below

```
void insertStart (struct Node **head, int data)
{
  struct Node *newNode = (struct Node *) malloc (sizeof (struct
Node));
  newNode->data = data;
  // if its the first node being entered
  if (*head == NULL)
    {
      *head = newNode;
      (*head)->next = *head;
      return;
    }
// if LL already as >=1 node
  struct Node *curr = *head;
  // traverse till last node in LL
  while (curr->next != *head)
    {
      curr = curr->next;
    }
  // assign LL's last node's next as this new node
  curr->next = newNode;
  // assign newNode's next as current head
  newNode->next = *head;
  // change head to this new node
  *head = newNode;
}
```

Insertion at the end

store the address of the head node to next of newNode (making newNode the last node)

point the current last node to newNode make newNode as the last node



Insertion at end in linked list the steps are followed :-

1.Make a new node.
2.Assign the new node next to circular list.
3.If the list is empty then return new node.
4.Assign the new node next to the front of the list.
5.Assign tail next to the new node.
6.Return the end node of the circular linked list.

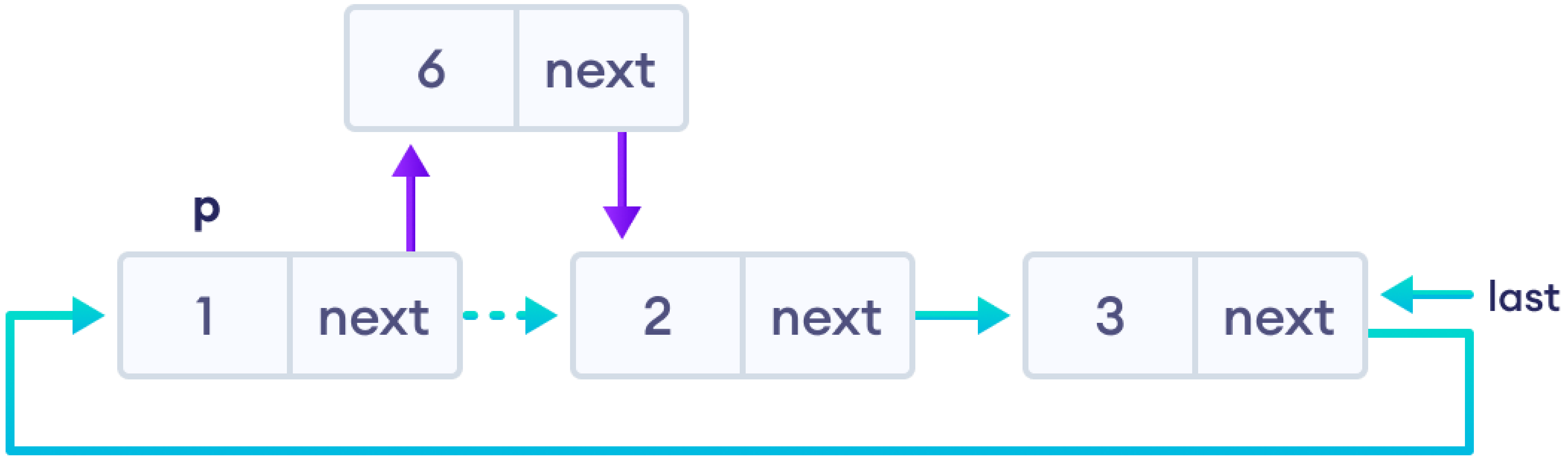
```
void insertLast (struct Node **head, int data)
{
  struct Node *newNode = (struct Node *) malloc (sizeof (struct
Node));
  newNode->data = data;

  // if its the first node being entered
  if (*head == NULL)
    {
      *head = newNode;
      (*head)->next = *head;
      return;
    }
```

```
  // if LL already as >=1 node
  struct Node *curr = *head;

  // traverse till last node in LL
  while (curr->next != *head)
    {
      curr = curr->next;
    }
  // assign LL's current last node's next as
this new node
  curr->next = newNode;
  // assign this new node's next as current
head of LL
  newNode->next = *head;
}
```

Insertion in between two nodes

Let's insert newNode after the first node. travel to the node given (let this node be p)

point the next of newNode to the node next to p store the address of newNode at next of p



Insertion in between the nodes in linked list the steps are followed :-

1.Make a new node and set the data.
2.Move to pos-1 position in the circular linked list.
3.Now link the next pointer of new node with the node pointed by the next pointer of current(pos-1) node.
4.After that join the next pointer of current node with the newly created node which means that the next pointer of current node will point to new node.
5.Now print the linked list.
6.Learn algorithm given below to understand better.

```
void insertPosition (int data, int pos, struct Node **head)
//function to insert element at specific position
{
  struct Node *newnode, *curNode;
  int i;
  if (*head == NULL)
    {
      printf ("List is empty");
    }
  if (pos == 1)
    {
      insertStart (head, data);
      return;
    }
```

```
else
   {
     newnode = (struct Node *) malloc (sizeof (struct Node));
     newnode->data = data;
     curNode = *head;
     while (--pos > 1)
       {
         curNode = curNode->next;
       }
     newnode->next = curNode->next;
     curNode->next = newnode;
   }
}
```

https://www.alphacodingskills.com/ds/notes/circular-singly-linked-list-insert-a-new-node-at-a-given-position.php

Module – 3 Linked List

The linked list is a **linear data structure** where each node has two parts.

1. Data

2. Reference to the next node

Data

In this section, we can store the required information. It can be any data type.

Example

```
int age;
char name[20];
```

Reference to the next node

It will hold the next nodes address. (Pointer variable is required)

Head Node - Starting node of a linked list.

Last Node - Node with reference pointer as NULL.

Data - it can be any data type. int,char,float,double etc.

Reference Part - It will hold the address of the next node. So, it is a type pointer.

To group two different data types (heterogeneous).

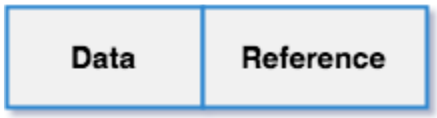
Which data type is used to group the different data types?

That is a structure. So, every node in a linked list is a structure data type. Sample Linked List Node

```
struct node
{
   int data;
   struct node *next;
};
```

data - used to store the integer information.

struct node *next - It is used to refer to the next node. It will hold the address of the next node.

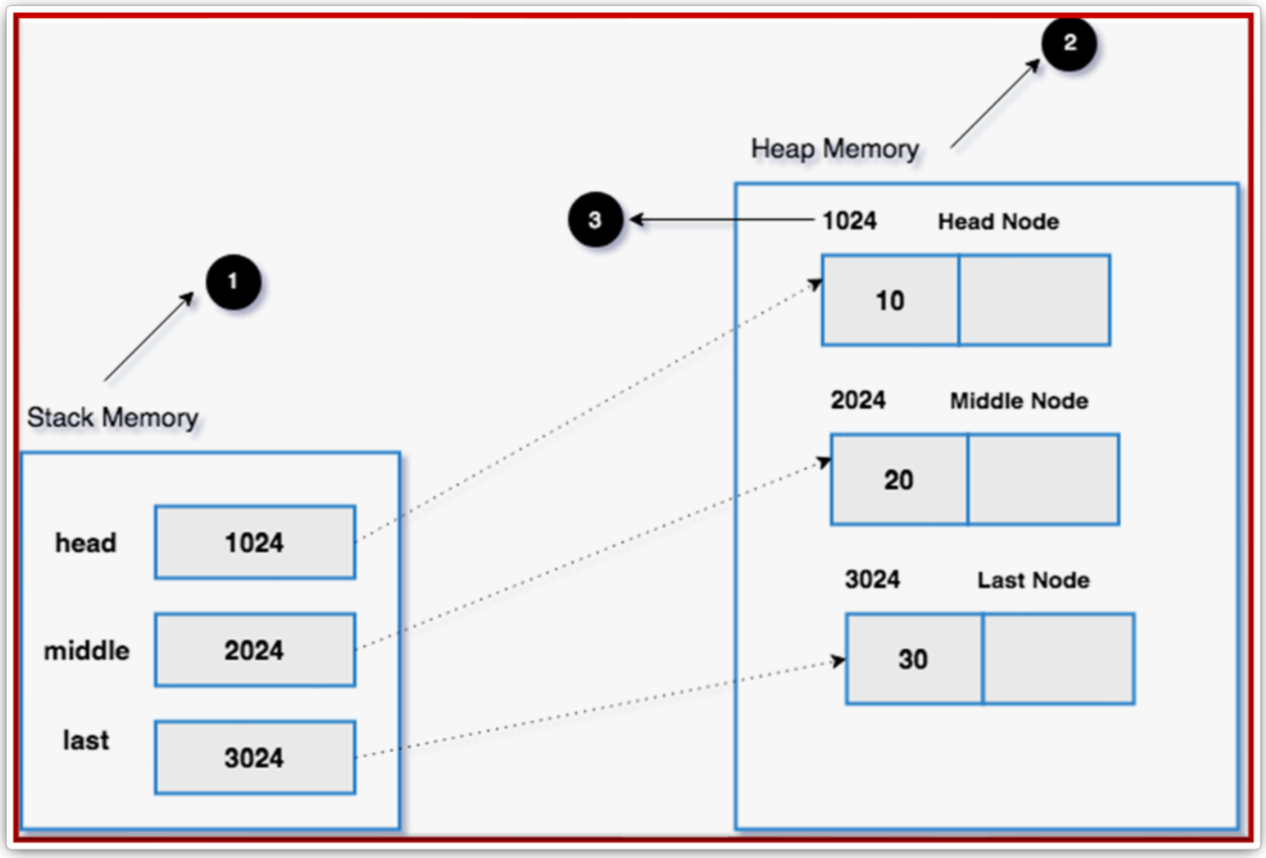


Create and allocate memory for 3 nodes

```
struct node
{
   int data;
   struct node *next;
};

struct node *head,*middle,*last;

head   = malloc(sizeof(struct node));
middle = malloc(sizeof(struct node));
last   = malloc(sizeof(struct node));
```

Assign values to each node

```
head->data   = 10;
middle->data = 20;
last->data   = 30;
```



1. Stack memory stores all the local variables and function calls (static memory).

Example: int a = 10;

2. Heap memory stores all the dynamically allocated variables.

Example: int *ptr = malloc(sizeof(int));

Here, memory will be allocated in the heap section.

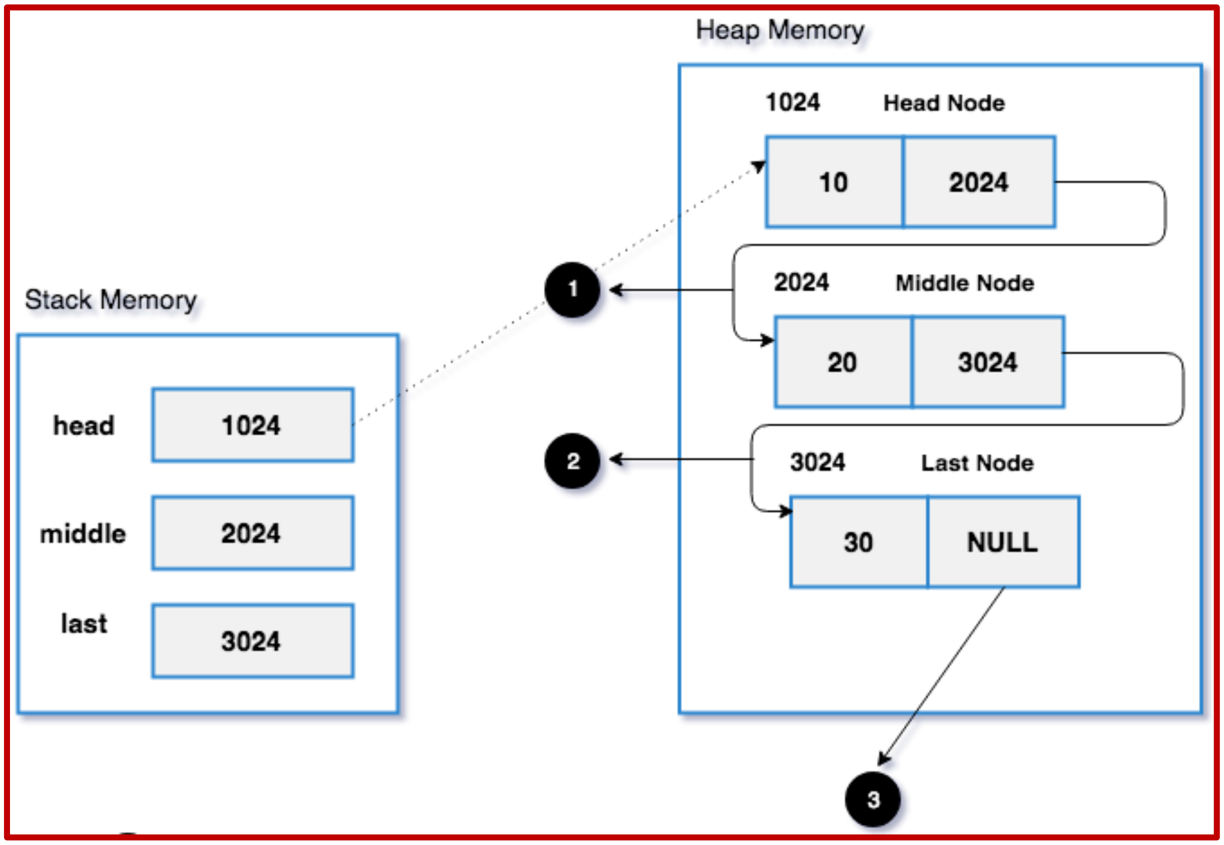
And the ptr resides in the stack section and receives the heap section memory address on successful memory allocation.

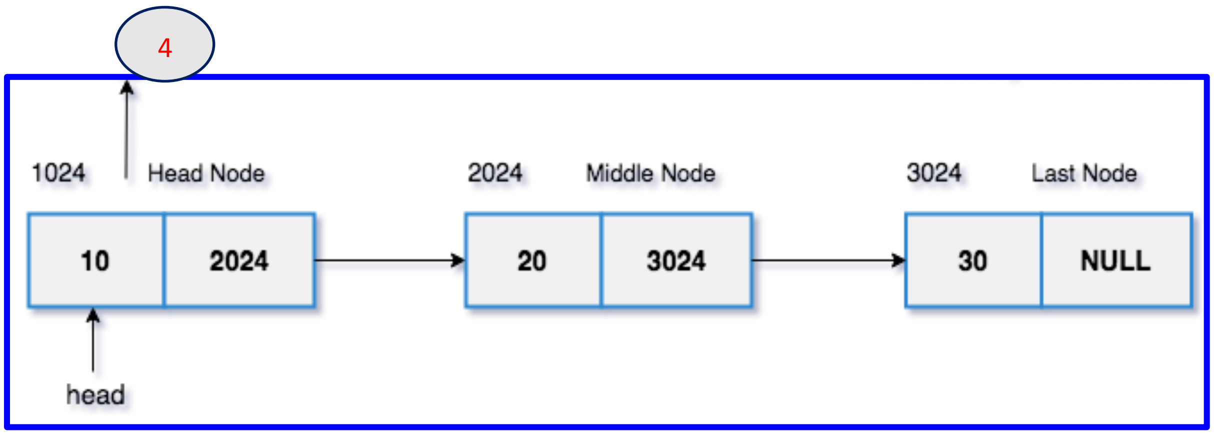
3. Address of the dynamic memory which will be assigned to the corresponding variable.

Linking each node

headnode middlenode lastnode NULL

```
head->next   = middle;
middle->next = last;
last->next   = NULL;     //NULL indicates the end of the linked list
```





1. head => next = middle. Hence head => next holds the memory address of the middle node (2024).

2. middle => next = last. Hence middle => next holds the memory address of the last node (3024).

3. last => next = NULL which indicates it is the last node in the linked list.

4. The simplified version of the heap memory section.

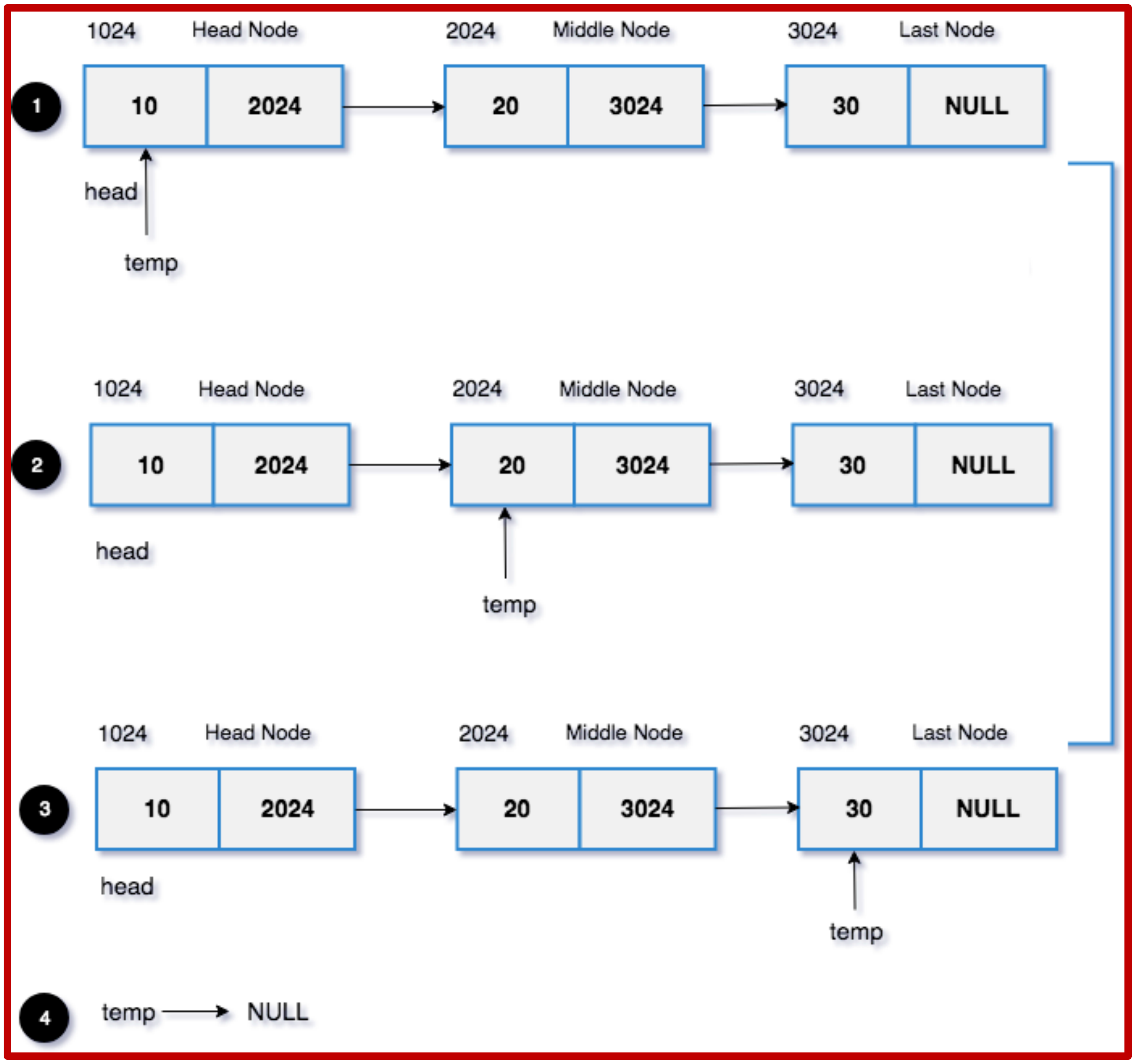
Display the content of each node

To print each node's data, we have to traverse the linked list till the end.

Algorithm

1. Create a temporary node(temp) and assign the head node's address.
2. Print the data which present in the temp node.
3. After printing the data, move the temp pointer to the next node.
4. Do the above process until we reach the end.

Visual Representation



1. temp points to the head node. temp => data = 10 will be printed. temp will point to the next node (Middle Node).

2. temp != NULL. temp => data = 20 will be printed. Again temp will point to the next node (Last Node).

3. temp != NULL. temp => data = 30 will be printed. Again temp will point to the next node which is NULL.

4. temp == NULL. Stop the process we have printed the whole linked list.

Code

```
struct node *temp = head;

while(temp != NULL)
{
   printf("%d ",temp->data);
   temp = temp->next;
}
```

What is the reason for opting for the temp node over the head node?

- If we use the head pointer to print the linked list, we might lose track of where it starts because the head node will point to nothing (NULL) after printing the data.
- To avoid this, we need to avoid changing the address of the head node while working with the linked list.
- It's better to always use a temporary node to handle the linked list.

Sample Linked List Implementation

Example

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  //node structure
  struct node
  {
      int regno;
      char name[20];
      struct node *next;
  };
  //declaring nodes
  struct node *head,*middle,*last;
  //allocating memory for each node
  head   = malloc(sizeof(struct node));
  middle = malloc(sizeof(struct node));
  last   = malloc(sizeof(struct node));

  //assigning values to each node
  head->regno   = 10;
  head->name=” DSU”;
  middle->regno = 20;
  head->name =”DSE”;
  last->regno  = 30;
 head->name=”DSI”;

  //connecting each nodes. head->middle->last
  head->next   = middle;
  middle->next = last;
  last->next   = NULL;

  //temp is a reference for head pointer.
  struct node *temp = head;

  //till the node becomes null, printing each nodes data
  while(temp != NULL)
  {
      printf("\n Regno: %d   Name: %s => ",temp->regno,temp->na
me );
      temp = temp->next;
  }
  printf("NULL");

  return 0;
}
```

Inserting a node at the beginning of a linked list

Example

Assume that the linked list has elements: 20 30 40 NULL

If we insert 100, it will be added at the beginning of a linked list.

After insertion, the new linked list will be

100 20 30 40 NULL

Algorithm

1. Declare a head pointer and make it as NULL.
2. Create a new node with the given data.
3. Make the new node points to the head node.
4. Finally, make the new node as the head node.

1. Declare a head pointer and make it as NULL

```
struct node
{
    int data;
    struct node *next;
};

struct node *head = NULL;
```

2.Create a new node with the given data.

```
void addFirst(struct node **head, int val)
{
      //create a new node
```

```
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
```

3 .Make the newnode points to the head node

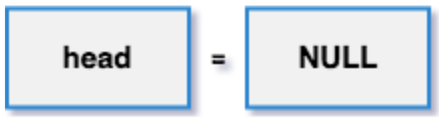
```
void addFirst(struct node **head, int val)
{
    //create a new node
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;

    newNode->next = *head;


}
```

4. Make the new node as the head node.

```
void addFirst(struct node **head, int val)
{
      //create a new node
      struct node *newNode = malloc(sizeof(struct node));
      newNode->data = val;

      newNode->next = *head;


      *head = newNode;


}
```

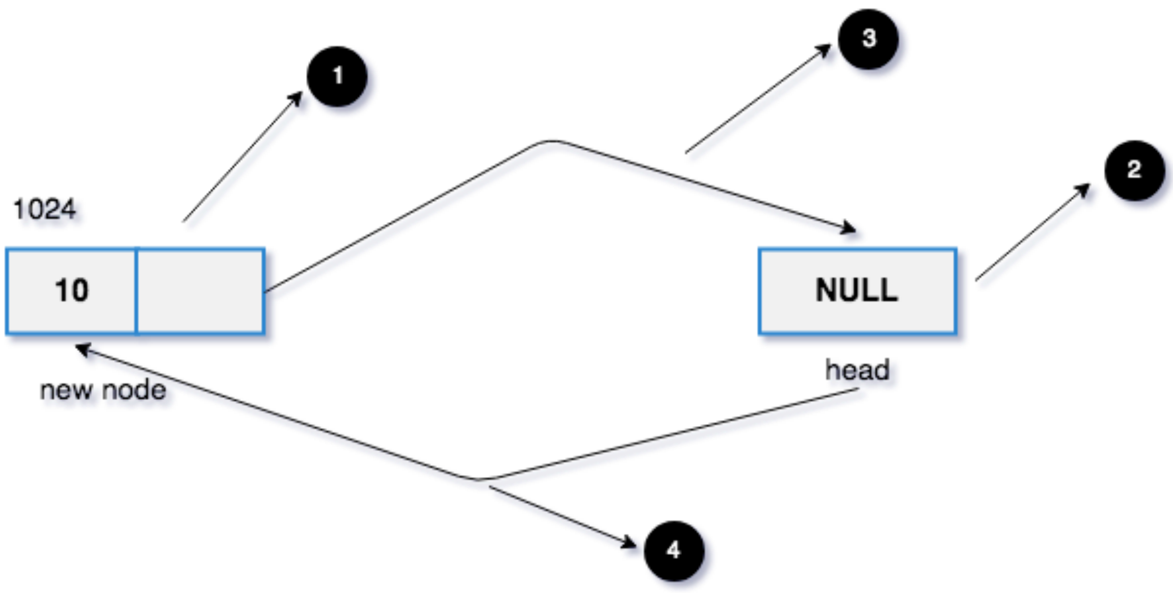
Visual Representation

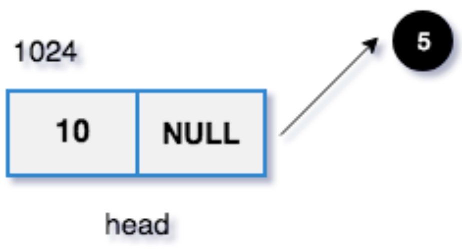
head = NULL



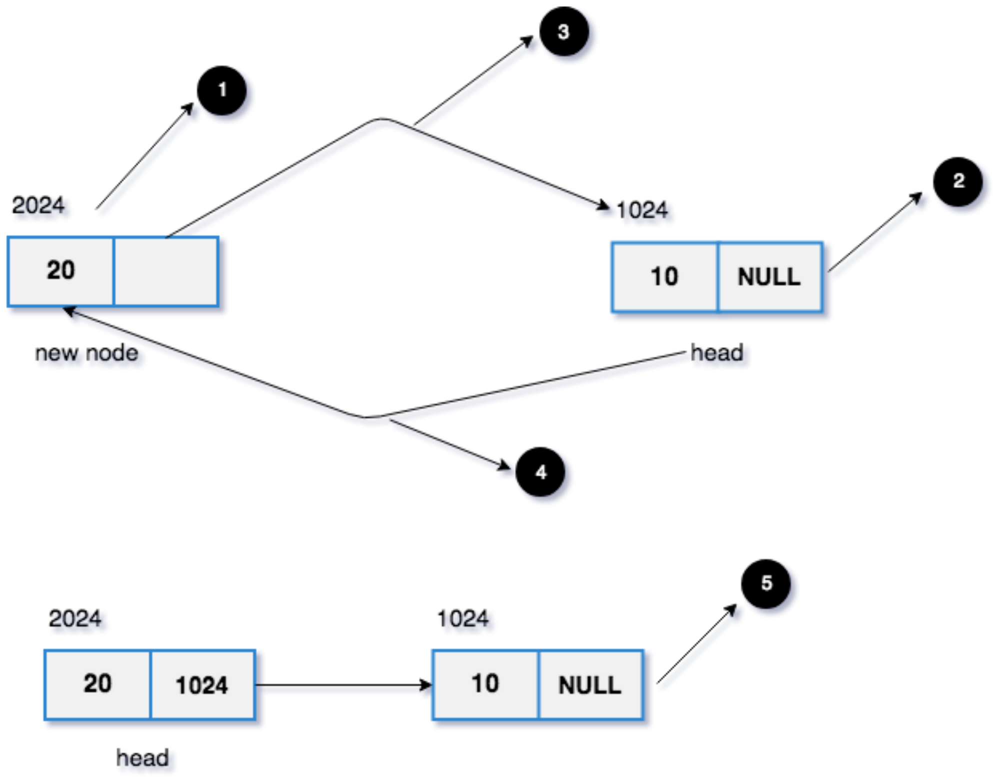
head initially set to NULL.

Let's insert data 10.

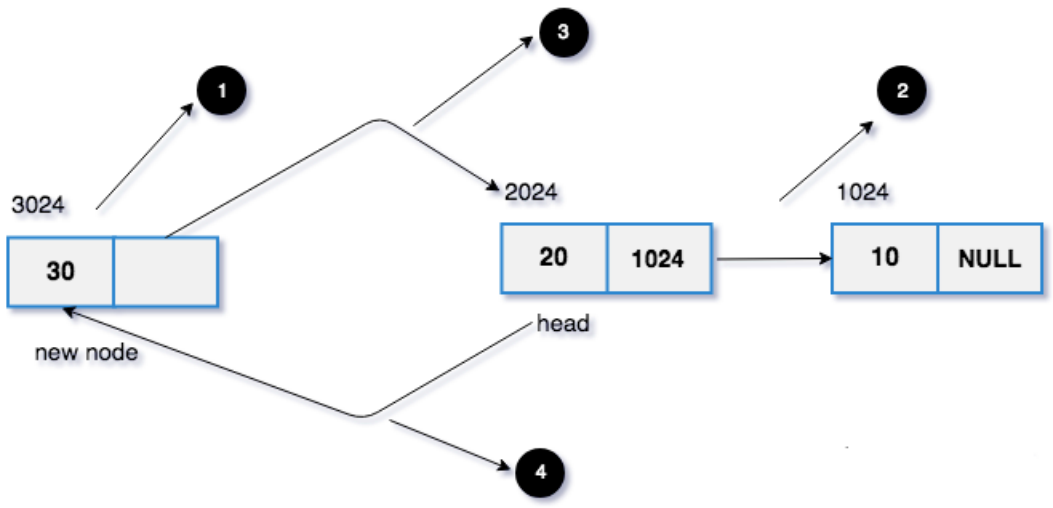


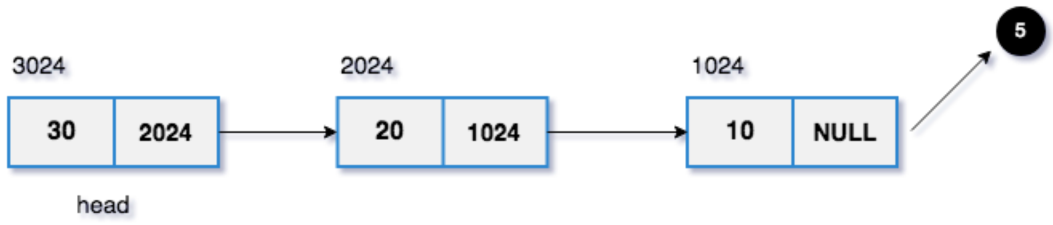


Let's insert data 20.



Let's insert data 30.





Program to insert a node at the beginning of linked list

Example

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

struct node
{
    int data;
    struct node *next;
};
```

```
void addFirst(struct node **head, int val)
{
      //create a new node
      struct node *newNode = malloc(sizeof(struct node));
      newNode->data = val;

      newNode->next = *head;

      *head = newNode;
}

void printList(struct node *head)
{
    struct node *temp = head;

    //iterate the entire linked list and print the data
    while(temp != NULL)
    {
         printf("%d->", temp->data);
         temp = temp->next;
    }
    printf("NULL\n");
}

int main()
{
     struct node *head = NULL;

     addFirst(&head,101);
     addFirst(&head,201);
     addFirst(&head,301);

     printList(head);

     return 0;
}
```

Output

```
301->201->101->NULL
```

Inserting a node at the end of a linked list

Example

Input

Linked List : 10 20 30 40 NULL.

50

Output

Linked List : 10 20 30 40 50 NULL.

Input

Linked List : NULL

10

Output

Linked List : 10 NULL

Algorithm

```
1. Declare head pointer and make it as NULL.
2. Create a new node with the given data. And make the new node => next as NULL.
   (Because the new node is going to be the last node.)
3. If the head node is NULL (Empty Linked List),
       make the new node as the head.
4. If the head node is not null, (Linked list already has some elements),
       find the last node.
       make the last node => next as the new node.
```

1. Declare head pointer and make it as NULL.

```
struct node
{
    int data;
    struct node *next;
};

struct node *head = NULL;
```

2. Create a new node

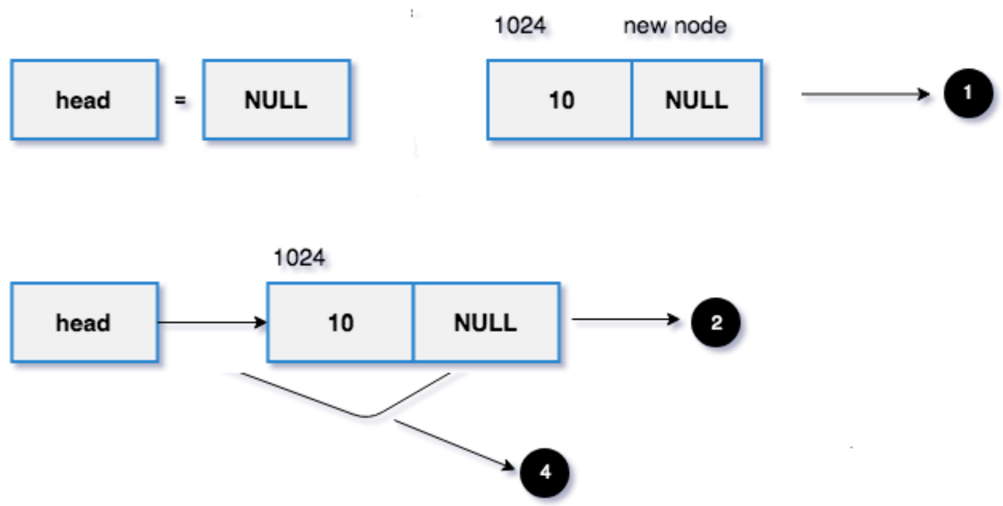
```
void addLast(struct node **head, int val)
{
    //create a new node
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next = NULL;
}
```

3. If the head node is NULL, make the new node as head

```
void addLast(struct node **head, int val)
{
    //create a new node
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next     = NULL;



    //if head is NULL, it is an empty list
    if(*head == NULL)
        *head = newNode;


}
```

Visual Representation

Let's insert data 10.



Alternate way

find the last node and set last node => new node

The last node of a linked list has the reference pointer as NULL. i.e. node=>next = NULL.

To find the last node, we have to iterate the linked till the node=>next != NULL.

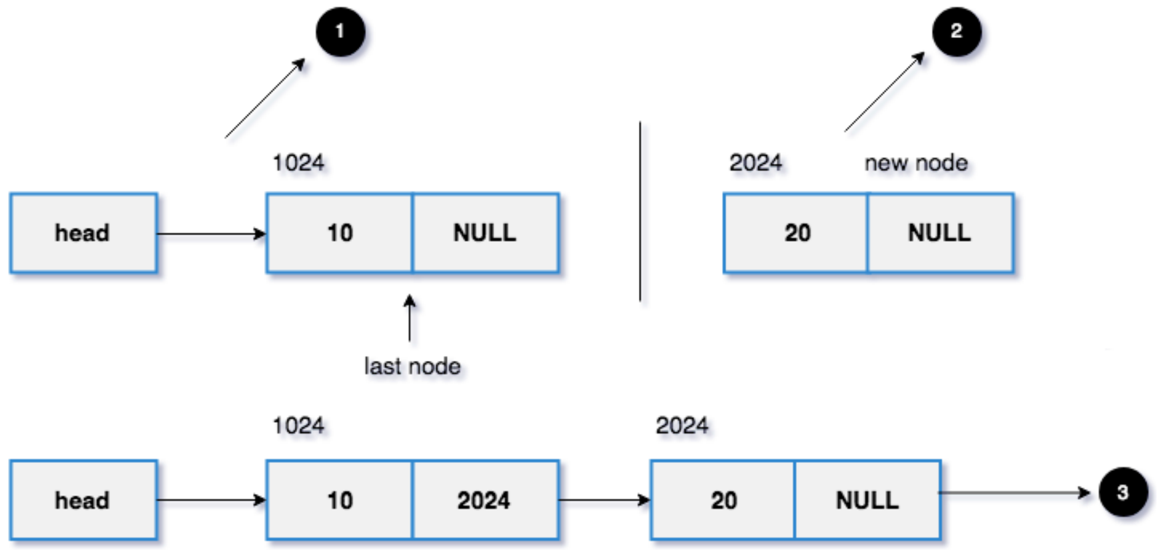
Pseudocode

```
while(node->next != NULL)
{
    node = node->next;
}

void addLast(struct node **head, int val)
{
    //create a new node
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next     = NULL;

    //if head is NULL, it is an empty list
    if(*head == NULL)
         *head = newNode;
    //Otherwise, find the last node and add the newNode
    else
    {
        struct node *lastNode = *head;


        //last node's next address will be NULL.
        while(lastNode->next != NULL)
        {
            lastNode = lastNode->next;
        }

        //add the newNode at the end of the linked list
        lastNode->next = newNode;


    }
```

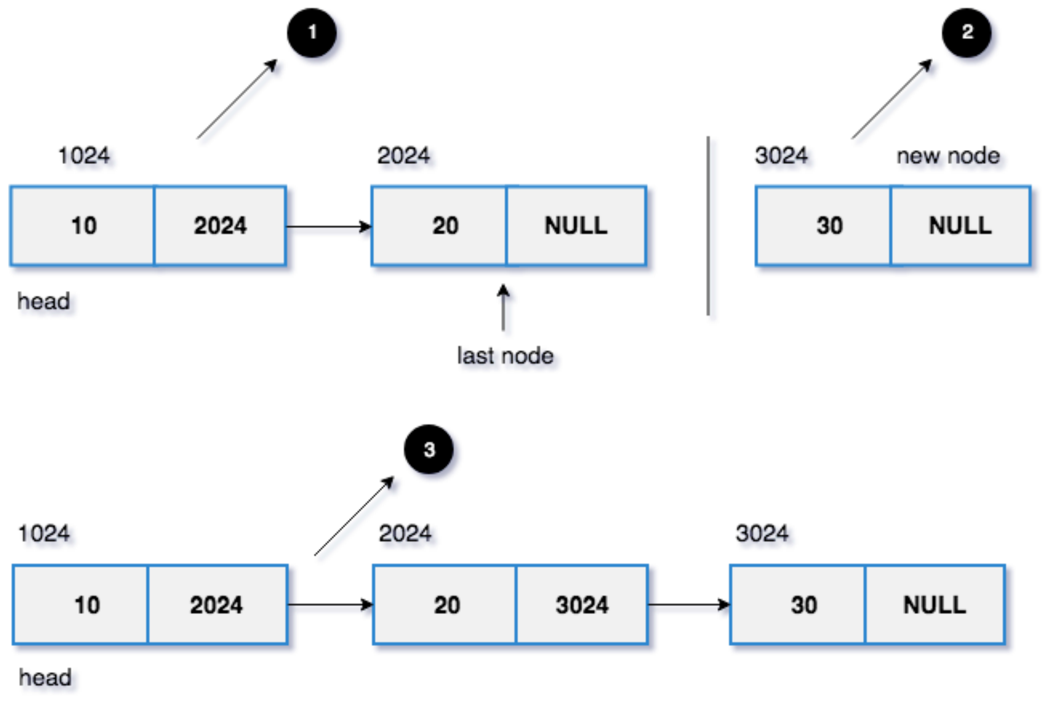
```
}
```

Visual Representation

Let's insert data 20.



Let's insert data 30.



Implementation of inserting a node at the end of a linked list

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node *next;

};
```

```
void addLast(struct node **head, int val)

{

    //create a new node

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

    newNode->data = val;

    newNode->next     = NULL;

    //if head is NULL, it is an empty list

    if(*head == NULL)

         *head = newNode;

    //Otherwise, find the last node and add the newNode

    else

    {

        struct node *lastNode = *head;

        //last node's next address will be NULL.

        while(lastNode->next != NULL)

        {

            lastNode = lastNode->next;

        }
        //add the newNode at the end of the linked list
        lastNode->next = newNode;
    }
}
void printList(struct node *head)
{
    struct node *temp = head;
    //iterate the entire linked list and print the data
    while(temp != NULL)
    {
         printf("%d->", temp->data);
         temp = temp->next;
    }
    printf("NULL\n");
}
int main()
{
     struct node *head = NULL;
     addLast(&head,10111);

     addLast(&head,20111);

     addLast(&head,30111);

     printList(head);

     return 0;

}
```

Output

```
10111->20111->30111->NULL
```

Searching a node in singly linked list

Example

Linked List : 10 20 30 40 NULL.

Input

20

Output

Search Found

Input

100

Output

Search Not Found

Algorithm

1. Iterate the linked list using a loop.
2. If any node has the given key value, return 1.
3. If the program execution comes out of the loop (the given key is not present in the linked list), return -1.

Search Found   => return 1.

Search Not Found => return -1.

1. Iterate the linked list using a loop.

```
int searchNode(struct node *head, int key)
{
    struct node *temp = head;


    while(temp != NULL)
    {
        temp = temp->next;
    }


}
```

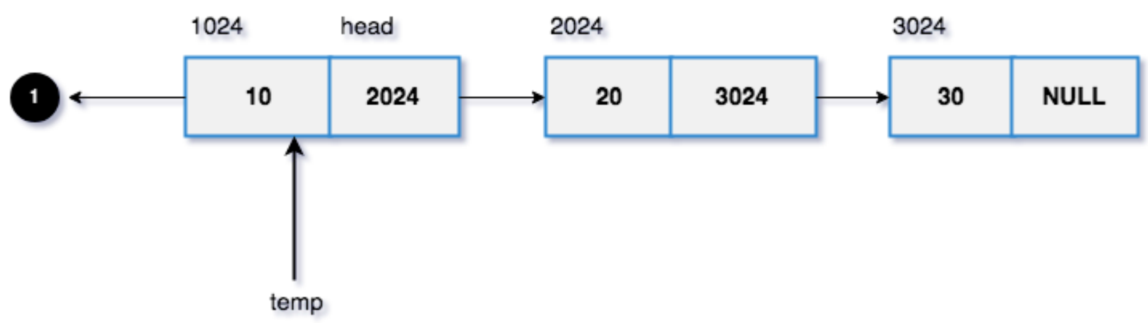
2. Return 1 on search found

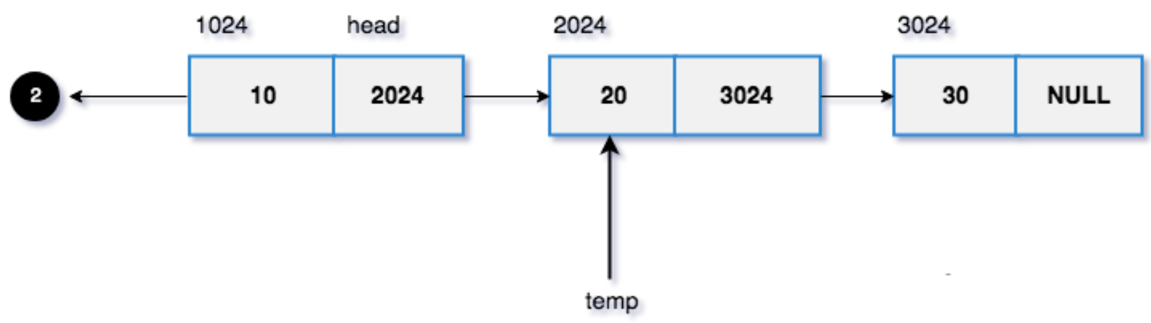
```
int searchNode(struct node *head, int key)
{
    struct node *temp = head;

    while(temp != NULL)
    {


        if(temp->data == key)
            return 1;


        temp = temp->next;
    }
}
```

Visual Representation

Linked List : 10 20 30 NULL

Key : 20





1. temp holding the address of the head node. temp->data = 10. key = 20. temp->data != key, so move the temp variable to the next node.

2. Now, temp->data = 20. key = 20. temp->key == key. "Seach Found".

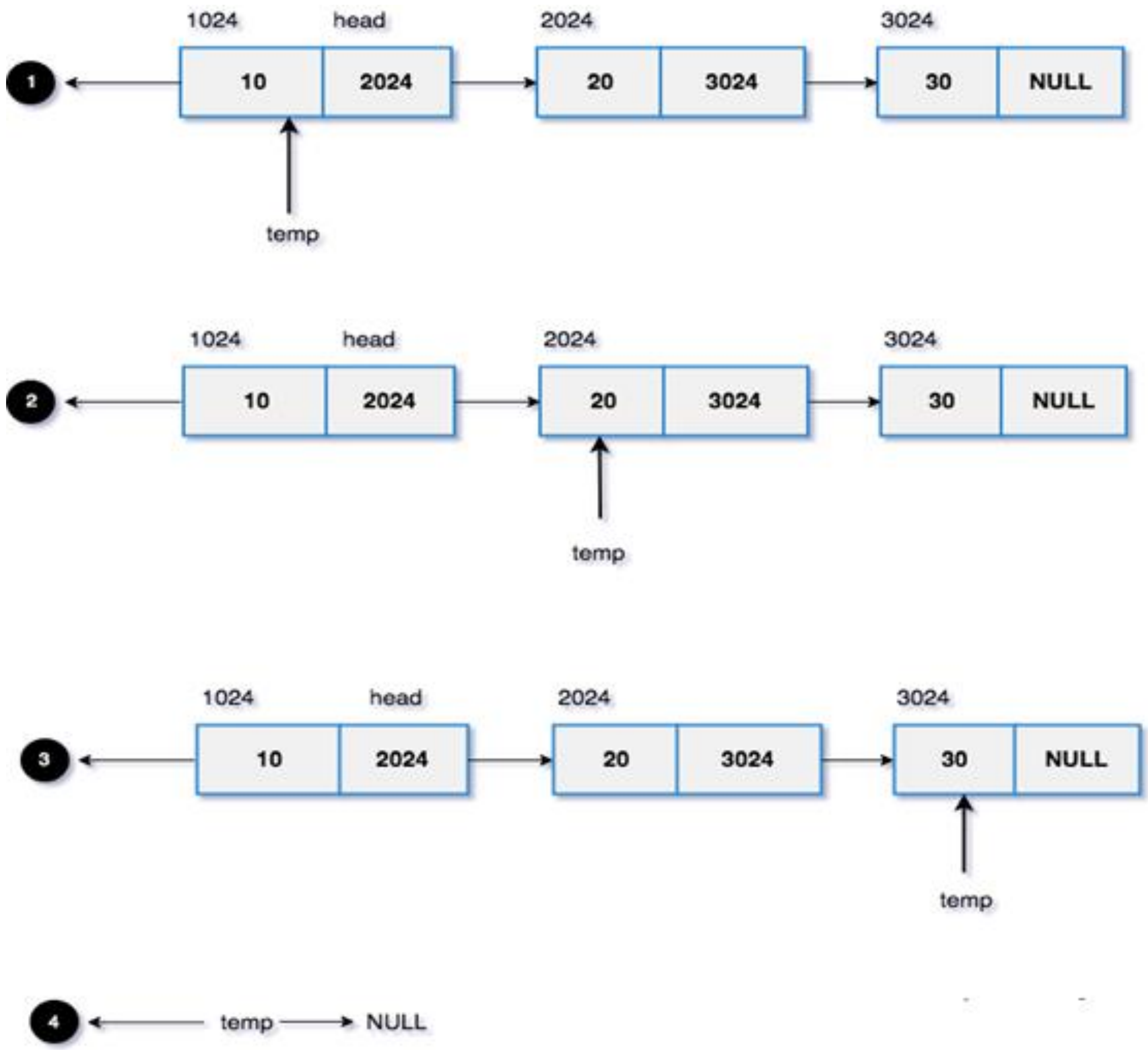
3. Return -1 on search not found

```
int searchNode(struct node *head, int key)
{
    struct node *temp = head;

    while(temp != NULL)
    {
        if(temp->data == key)
            return 1;
        temp = temp->next;
    }

    return -1;


}
```

Visual Representation

Linked List : 10 20 30 NULL

Key : 100



Implementation of searching a node in singly linked list

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node *next;

};

void addLast(struct node **head, int val)

{

    //create a new node

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

    newNode->data = val;

    newNode->next     = NULL;

    //if head is NULL, it is an empty list

    if(*head == NULL)

         *head = newNode;

    //Otherwise, find the last node and add the newNode
    else
    {
        struct node *lastNode = *head;
        //last node's next address will be NULL.
        while(lastNode->next != NULL)
        {
            lastNode = lastNode->next;
        }
        //add the newNode at the end of the linked list
        lastNode->next = newNode;
    }
}
int searchNode(struct node *head,int key)
{
    struct node *temp = head;
    //iterate the entire linked list and print the data
    while(temp != NULL)
    {
        //key found return 1.
        if(temp->data == key)
            return 1;
        temp = temp->next;
    }
    //key not found
    return -1;
}
int main()
{
     struct node *head = NULL;
     addLast(&head,10);
     addLast(&head,20);
     addLast(&head,30);
     //change the key and try it yourself.
     int no;
     printf("Enter the number to be searched \n");
     scanf(" %d",&no);
```

```
         if(searchNode(head,no) == 1)
        printf("Search Found\n");
    else
        printf("Search Not Found\n");
    return 0;
}
```

Output

Case 1

```
Enter the number to be searched
23
Search Not Found
```

Case2

```
Enter the number to be searched
10
Search Found
```

Deleting a node in linked list

Example

Linked List : 10 20 30 NULL

Input

20

Output

10 30 NULL

Input

30

Output

10 20 NULL

Algorithm

1. If the head node has the given key,

   make the head node points to the second node and free its memory.

2. Otherwise,

   From the current node, check whether the next node has the given key

   if yes, make the current->next = current->next->next and free the memory.

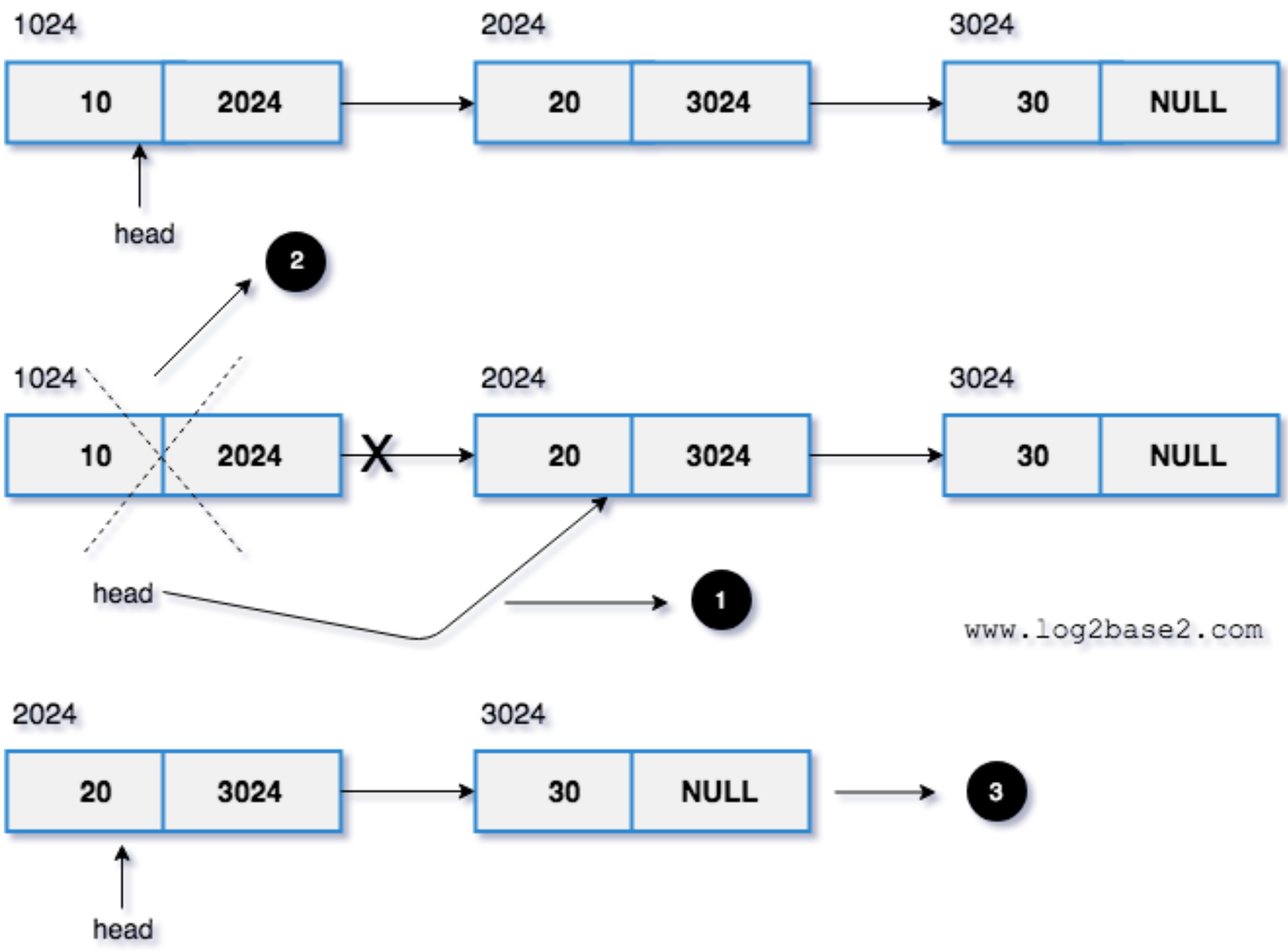
   else, update the current node to the next and do the above process (from step 2) till the last node.

1. Head node has the given key

```
void deleteNode(struct node **head, int key)
{
      //temp is used to freeing the memory
      struct node *temp;


      //key found on the head node.
      //move to head node to the next and free the head.
      if(*head->data == key)
      {
          temp = *head;    //backup the head to free its memory
          *head = (*head)->next;
          free(temp);
      }


}
```

Visual Representation

Let's delete data 10 (head node).



1. Make the head points to the next node.
2. Free the head node's memory.
3. Finally, the new linked list.

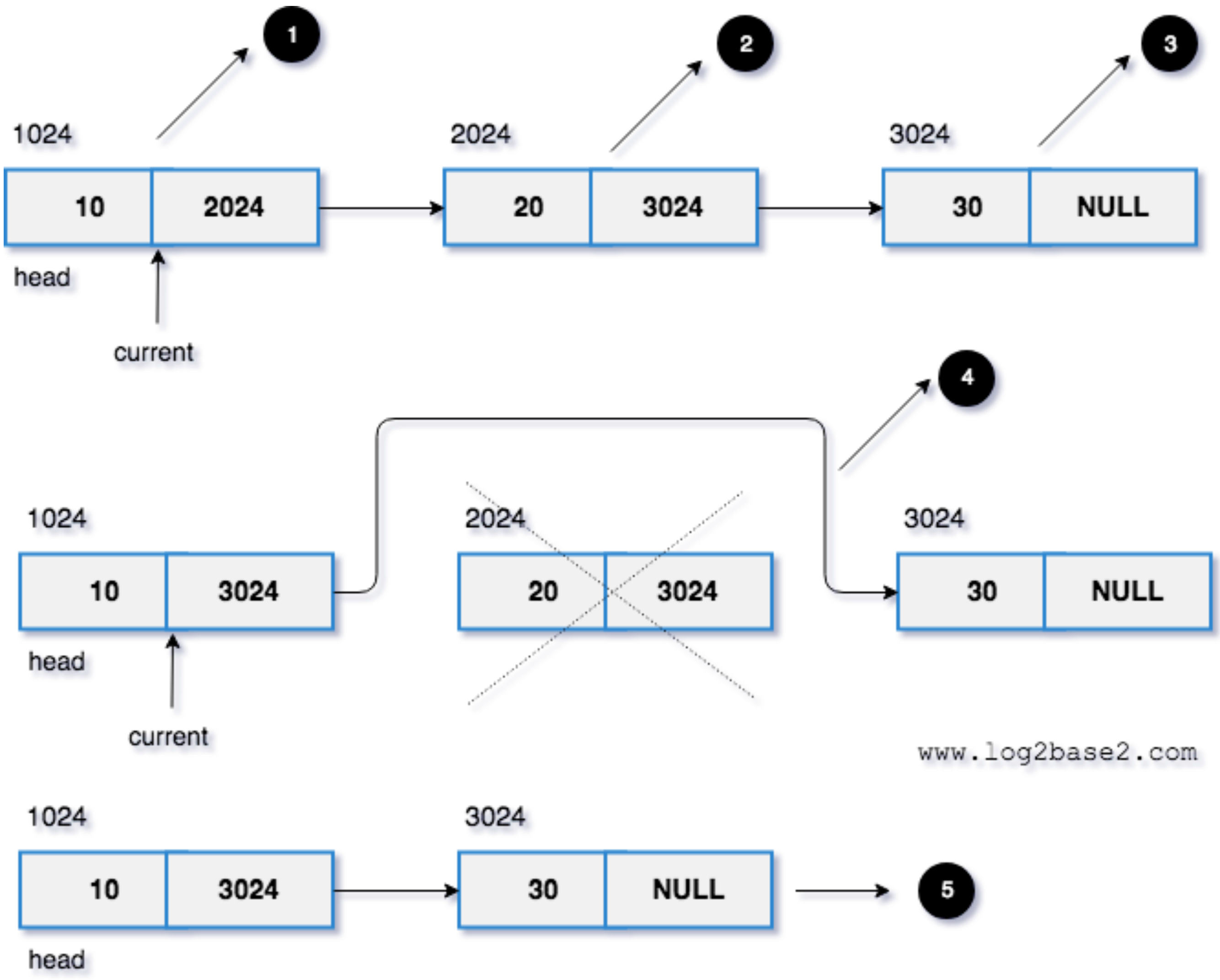
2. For other nodes (Non-Head)

```
void deleteNode(struct node **head, int key)
{
      //temp is used to freeing the memory
      struct node *temp;

      //key found on the head node.
      //move to head node to the next and free the head.
      if((*head)->data == key)
      {
          temp = *head;    //backup to free the memory
          *head = (*head)->next;
          free(temp);
      }


      else
      {
          struct node *current  = *head;
          while(current->next != NULL)
          {
              //if yes, we need to delete the current->next node
```

```
            if(current->next->data == key)
            {
                temp = current->next;
                //node will be disconnected from the linked list.
                current->next = current->next->next;
                free(temp);
                break;
            }
            //Otherwise, move the current node and proceed
            else
                current = current->next;
        }
    }


}
```

Visual Representation

Let's delete data 20.



1. Make the current node points to the head node. (current => data = 10).

2. current => next. (current=>next=>data = 20).

3. current => next => next. (current=>next=>next=>data = 30).

4. We have to remove the node 20. Since current => next = 20 we can remove the node by setting current => next = current =>next => next. And then free the node.

5. Finally, the new linked list.

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node *next;

};

void addLast(struct node **head, int val)

{

    //create a new node

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

    newNode->data = val;

    newNode->next     = NULL;
```

```
//if head is NULL, it is an empty list
if(*head == NULL)
     *head = newNode;
//Otherwise, find the last node and add the newNode
else
{
    struct node *lastNode = *head;

    //last node's next address will be NULL.
    while(lastNode->next != NULL)
    {
        lastNode = lastNode->next;
    }

    //add the newNode at the end of the linked list
    lastNode->next = newNode;
}
```

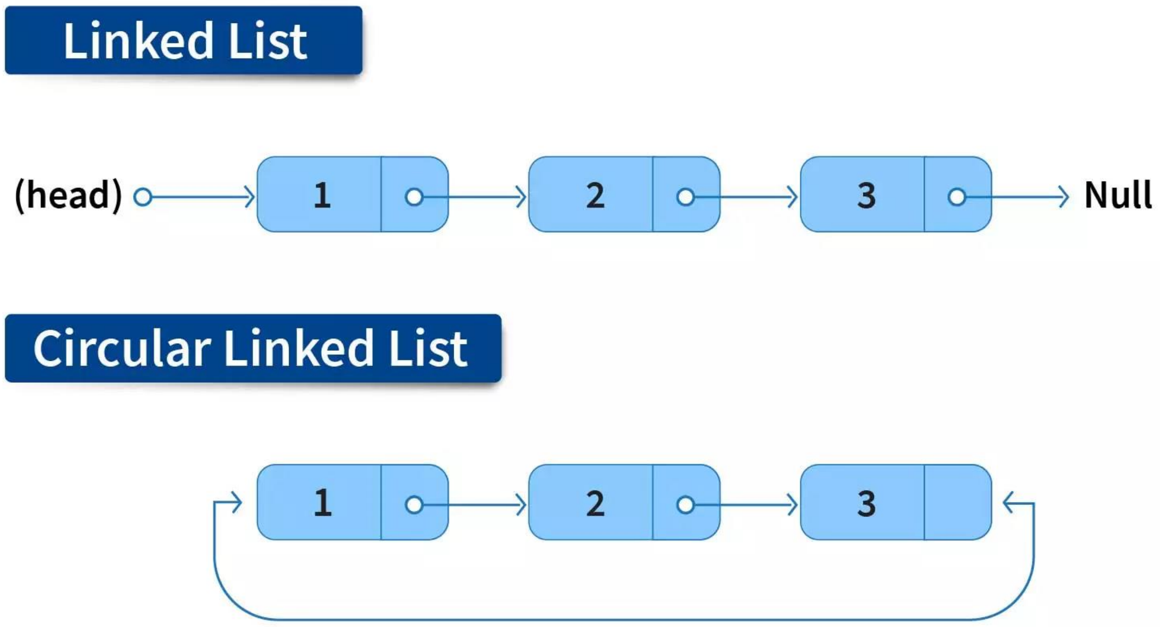
```
}

void deleteNode(struct node **head, int key)
{
    //temp is used to freeing the memory
    struct node *temp;

    //key found on the head node.
    //move to head node to the next and free the head.
    if((*head)->data == key)
    {
        temp = *head;    //backup to free the memory
        *head = (*head)->next;
        free(temp);
    }
    else
    {
        struct node *current  = *head;
        while(current->next != NULL)
        {
            //if yes, we need to delete the current->next node
            if(current->next->data == key)
            {
                temp = current->next;
                //node will be disconnected from the linked list.
                current->next = current->next->next;
                free(temp);
                break;
            }
            //Otherwise, move the current node and proceed
            else
                current = current->next;
        }
    }
}
```

```
void printList(struct node *head)

{

    struct node *temp = head;


    //iterate the entire linked list and print the data

    while(temp != NULL)

    {

         printf("%d ->", temp->data);

         temp = temp->next;

    }

    printf("NULL\n");

}

int main()

{

     struct node *head = NULL;

     addLast(&head,10);

     addLast(&head,20);

     addLast(&head,30);
     printf("Linked List Elements:\n");
     printList(head);
     //delete first node
     deleteNode(&head,10);
     printf("Deleted 10. The New Linked List:\n");
     printList(head);
     //delete last node
     deleteNode(&head,30);
     printf("Deleted 30. The New Linked List:\n");
     printList(head);
     //delete 20
     deleteNode(&head,20);
     printf("Deleted 20. The New Linked List:\n");
     printList(head);
}
```

Output

```
20 ->30 ->NULL
Deleted 30. The New Linked List:
```

```
20 ->NULL

Deleted 20. The New Linked List:

NULL
```


Linked List
(head)
1
2
3
Null
Circular Linked List
1
2
3

Example -1

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
  char data;
  struct Node *next;
};
void insertStart (struct Node **head, char data)
{
  struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));
  newNode->data = data;

  // if its the first node being entered
  if (*head == NULL)
    {
      *head = newNode;
      (*head)->next = *head;
      return;
    }

  // if LL already as >=1 node
  struct Node *curr = *head;

  // traverse till last node in LL
  while (curr->next != *head)
    {
      curr = curr->next;
    }
  // assign LL's last node's next as this new node
  curr->next = newNode;

  // assign newNode's next as current head
  newNode->next = *head;

  // change head to this new node
  *head = newNode;
```

```
}

void insertLast (struct Node **head, char data)
{
  struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));
  newNode->data = data;

  // if its the first node being entered
  if (*head == NULL)
    {
      *head = newNode;
      (*head)->next = *head;
      return;
    }

  // if LL already as >=1 node
  struct Node *curr = *head;

  // traverse till last node in LL
  while (curr->next != *head)
    {
      curr = curr->next;
    }

  // assign LL's current last node's next as this new node
  curr->next = newNode;

  // assign this new node's next as current head of LL
  newNode->next = *head;
}

void insertPosition (char data, int pos, struct Node **head)
//function to insert element at specific position
{
  struct Node *newnode, *curNode;
  int i;
  if (*head == NULL)
    {
      printf ("List is empty");
    }
  if (pos == 1)
    {
      insertStart (head, data);
      return;
    }
  else
    {
      newnode = (struct Node *) malloc (sizeof (struct Node));
      newnode->data = data;
      curNode = *head;
      while (--pos > 1)
          {
            curNode = curNode->next;
          }
      newnode->next = curNode->next;
      curNode->next = newnode;
    }
}

void display (struct Node *head)
{
  // if there are no node in LL
  if (head == NULL)
    return;

  struct Node *temp = head;

  //need to take care of circular structure of LL
  do
    {
      printf ("%c ", temp->data);
      temp = temp->next;
    }
  while (temp != head);
  printf ("\n");
}

int main ()
{

  struct Node *head = NULL;

  printf("Insert at beginning: ");
  insertStart (&head, 'A');
  insertStart (&head, 'B');
  display (head);

  printf("Insert at End: ");
  insertLast (&head, 'Y');
  insertLast (&head, 'Z');;
  display (head);

  printf("Insert at Specific Position: ");
  insertPosition ('E', 3, &head);
  display (head);

  return 0;
}
```

Output

```
Insert at beginning: B A
Insert at End: B A Y Z
Insert at Specific Position: B A E Y Z


...Program finished with exit code 0
Press ENTER to exit console.
```

Example -2

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
  int data;
  struct Node *next;
};
void insertStart (struct Node **head, int data)
{
  struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));
  newNode->data = data;

  // if its the first node being entered
  if (*head == NULL)
    {
      *head = newNode;
      (*head)->next = *head;
      return;
    }

  // if LL already as >=1 node
  struct Node *curr = *head;

  // traverse till last node in LL
  while (curr->next != *head)
    {
      curr = curr->next;
    }
  // assign LL's last node's next as this new node
  curr->next = newNode;

  // assign newNode's next as current head
  newNode->next = *head;

  // change head to this new node
  *head = newNode;
```

```
}

void insertLast (struct Node **head, int data)
{
  struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));
  newNode->data = data;

  // if its the first node being entered
  if (*head == NULL)
    {
      *head = newNode;
      (*head)->next = *head;
      return;
    }

  // if LL already as >=1 node
  struct Node *curr = *head;

  // traverse till last node in LL
  while (curr->next != *head)
    {
      curr = curr->next;
    }

  // assign LL's current last node's next as this new node
  curr->next = newNode;

  // assign this new node's next as current head of LL
  newNode->next = *head;
}

void insertPosition (int data, int pos, struct Node **head)
//function to insert element at specific position
{
  struct Node *newnode, *curNode;
  int i;
  if (*head == NULL)
    {
      printf ("List is empty");
    }
  if (pos == 1)
    {
      insertStart (head, data);
      return;
    }
  else
    {
      newnode = (struct Node *) malloc (sizeof (struct Node));
      newnode->data = data;
      curNode = *head;
      while (--pos > 1)
           {
             curNode = curNode->next;
           }
      newnode->next = curNode->next;
      curNode->next = newnode;
    }
}

void display (struct Node *head)
{
  // if there are no node in LL
  if (head == NULL)
    return;

  struct Node *temp = head;

  //need to take care of circular structure of LL
  do
    {
      printf ("%d ", temp->data);
      temp = temp->next;
    }
  while (temp != head);
  printf ("\n");
}

int main ()
{

  struct Node *head = NULL;

  printf("Insert at beginning: ");
  insertStart (&head, 2);
  insertStart (&head, 1);
  display (head);

  printf("Insert at End: ");
  insertLast (&head, 30);
  insertLast (&head, 40);
  display (head);

  printf("Insert at Specific Position: ");
  insertPosition (5, 3, &head);
  display (head);

  return 0;
}
```

Output

```
Insert at beginning: 1 2
Insert at End: 1 2 30 40
Insert at Specific Position: 1 2 5 30 40
```

Module -3 Doubly Linked List

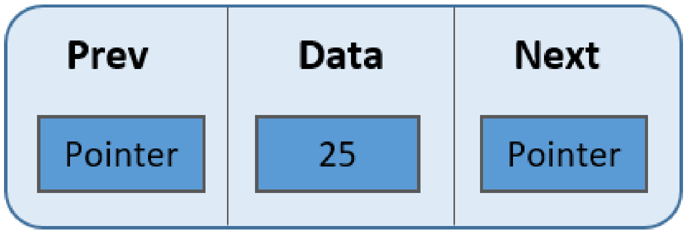
Linked List

A **linked list** is a linear data structure in which data is stored in a non-contiguous memory location. Every node contains data and the address of the other node.

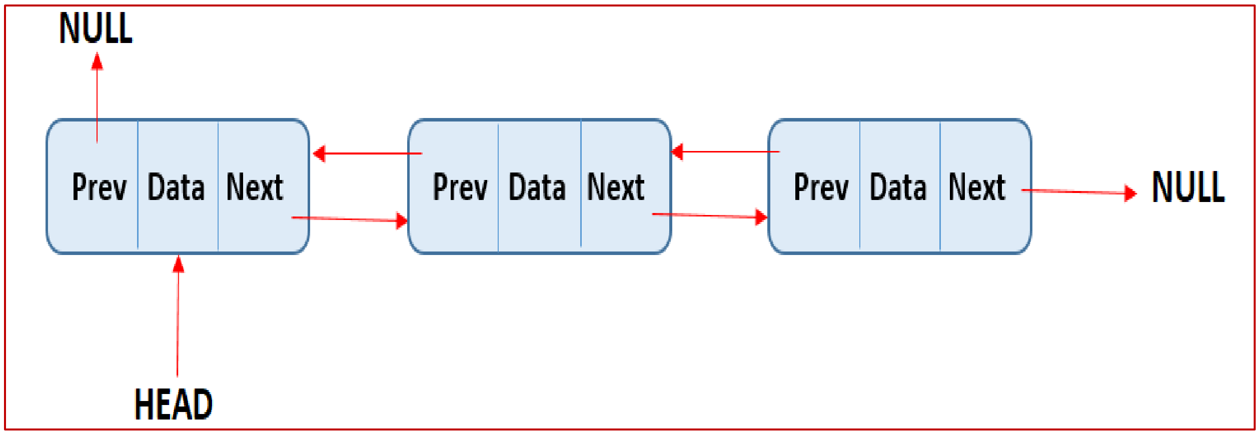
Doubly Linked List

A doubly linked list is a type of linked list in which each node consists of 3 components:

***prev - address of the previous node**
data - data item
***next - address of next node**



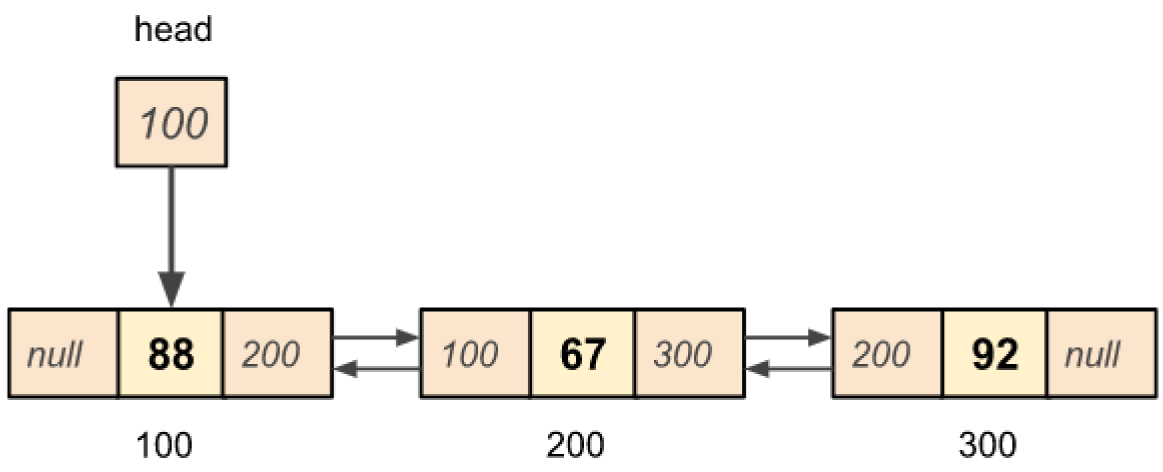
```
//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};
```



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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};
int main() {

  //create the head node with name MyList
  struct Node* MyList = NULL;

  //Add first node.
  struct Node* first;
  //allocate second node in the heap
  first = (struct Node*)malloc(sizeof(struct Node));
  first->data = 10001;
  first->next = NULL;
  first->prev = NULL;
  //linking with head node
  MyList = first;
  //Add second node.
  struct Node* second;
  //allocate second node in the heap
  second = (struct Node*)malloc(sizeof(struct Node));
  second->data = 20001;
  second->next = NULL;
  //linking with first node
  second->prev = first;
  first->next = second;

  //Add third node.
  struct Node* third;
  //allocate third node in the heap
  third = (struct Node*)malloc(sizeof(struct Node));
  third->data = 30001;
  third->next = NULL;
  //linking with second node
  third->prev = second;
  second->next = third;

  return 0;
}
```

Traversal

```
void PrintList(struct Node* head_ref) {

  //1. create a temp node pointing to head

  struct Node* temp = head_ref;

  //2. if the temp node is not null continue
  //   displaying the content and move to the
  //   next node till the temp becomes null

  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {

    //3. If the temp node is null at the start,
    //   the list is empty

    printf("The list is empty.\n");
  }
}
```



```
#include <stdio.h>
#include <stdlib.h>
//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};
//Add new element at the end of the list
void push_back(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    while(temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}

//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}
int main() {
  struct Node* MyList = NULL;

  //Add three elements at the end of the list.
  push_back(&MyList, 110);
  push_back(&MyList, 210);
  push_back(&MyList, 310);

  //traverse to display the content of the list.
  PrintList(MyList);

  return 0;
}
```

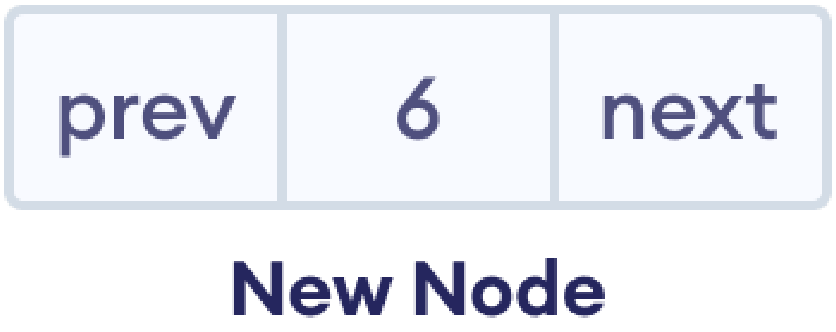
Doubly Linked List - Insert a new node at the beginning

1. Insertion at the Beginning

Let's add a node with value **6** at the beginning of the doubly linked list we made above.

1. **Create a new node**

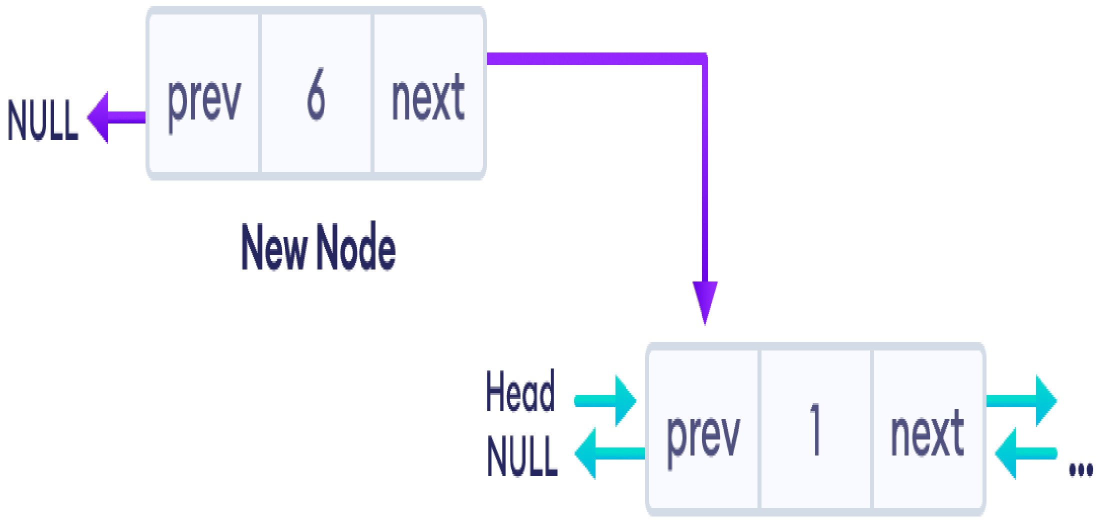
```
allocate memory for newNode
assign the data to newNode.
```



2. Set prev and next pointers of new node

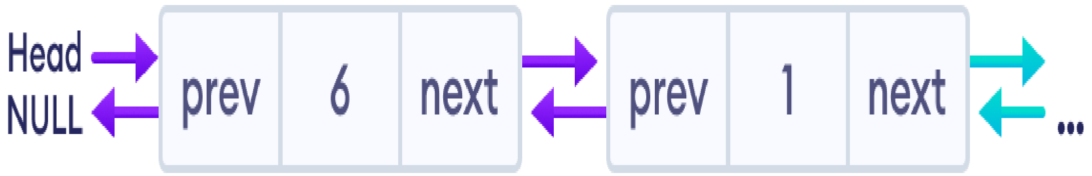
point next of newNode to the first node of the doubly linked list

point prev to null



3. Make new node as head node

- **Point prev of the first node to newNode (now the previous head is the second node)**
- **Point head to newNode**



New Node

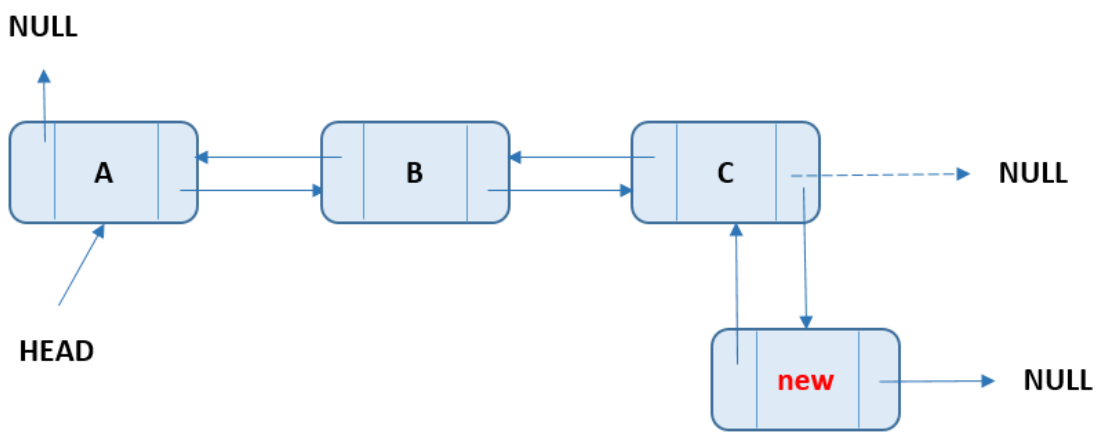
```
// insert node at the front

void insertFront(struct Node** head, int data) {
   // allocate memory for newNode

   struct Node* newNode = new Node;
   // assign data to newNode
   newNode->data = data;

   // point next of newNode to the first node of the doubly linked list
   newNode->next = (*head);
   // point prev to NULL
   newNode->prev = NULL;

   // point previous of the first node (now first node is the second
node) to newNode
   if ((*head) != NULL)
      (*head)->prev = newNode;

   // head points to newNode
   (*head) = newNode;
}
```

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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

//Add new element at the start of the list
void push_front(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    (*head_ref)->prev = newNode;
    newNode->next = *head_ref;
    *head_ref = newNode;
  }
}

//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The doubly linked  list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}
int main() {
  struct Node* MyList = NULL;

  //Add three elements at the start of the list.
  push_front(&MyList, 10);
  push_front(&MyList, 20);
  push_front(&MyList, 30);
  PrintList(MyList);

  return 0;
}
```

Output

```
The doubly linked  list contains: 30 20 10
```

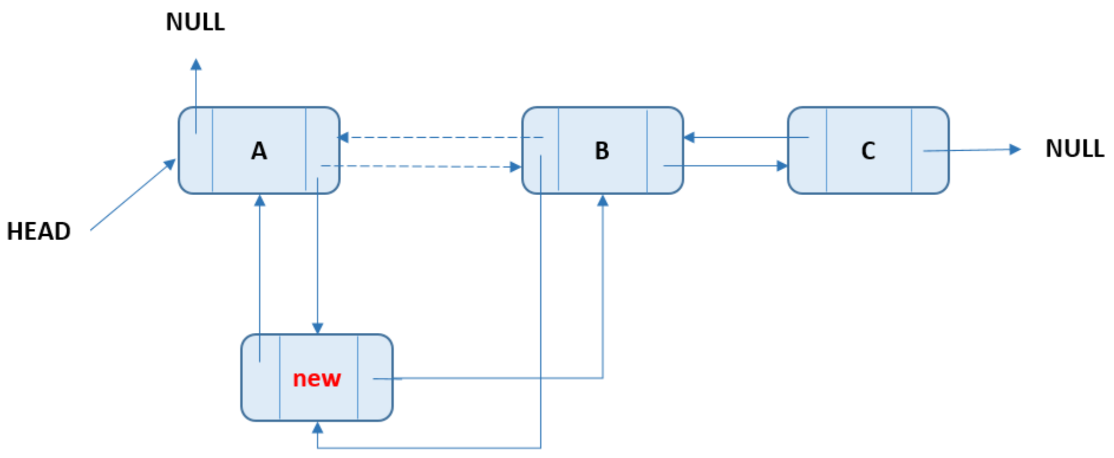
Doubly Linked List - Insert a new node at the end



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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

//Add new element at the end of the list
void push_back(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    while(temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}

//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}
int main() {
  struct Node* MyList = NULL;

  //Add three elements at the end of the list.
  push_back(&MyList, 10);
  push_back(&MyList, 20);
  push_back(&MyList, 30);
  PrintList(MyList);

  return 0;
}
```

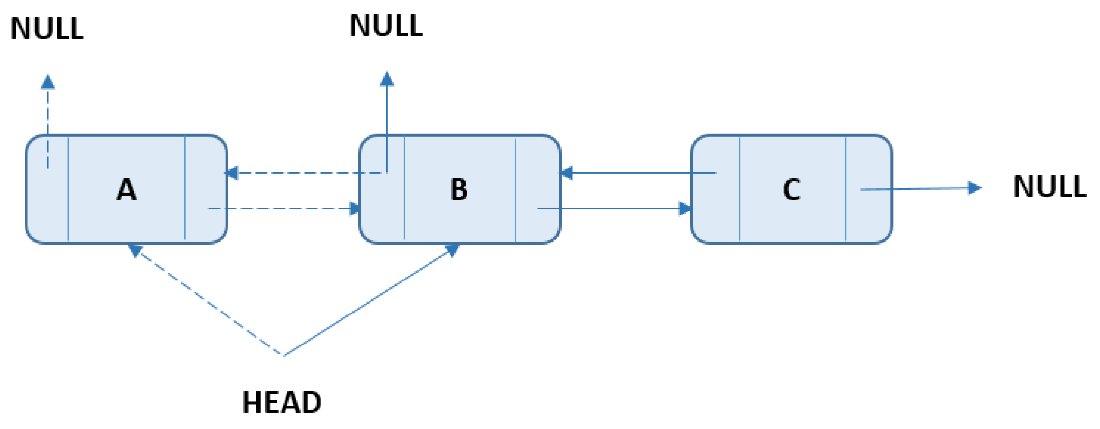
Doubly Linked List - Insert a new node at the given position

https://visualgo.net/en/list



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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

//Add new element at the end of the list
void push_back(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    while(temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}

//Inserts a new element at the given position
void push_at(struct Node** head_ref, int newElement, int position) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(position < 1) {
    printf("\nposition should be >= 1.");
  } else if (position == 1) {
    newNode->next = *head_ref;
    (*head_ref)->prev = newNode;
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    for(int i = 1; i < position-1; i++) {
      if(temp != NULL) {
        temp = temp->next;
      }
    }
    if(temp != NULL) {
      newNode->next = temp->next;
      newNode->prev = temp;
      temp->next = newNode;
      if(newNode->next != NULL)
        newNode->next->prev = newNode;
    } else {
      printf("\nThe previous node is null.");
    }
  }
}

//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}

int main() {
  struct Node* MyList = NULL;

  //Add three elements in the list.
  push_back(&MyList, 10);
  push_back(&MyList, 20);
  push_back(&MyList, 30);
  PrintList(MyList);

  //Insert an element at position 2
  push_at(&MyList, 100, 2);
  PrintList(MyList);

  //Insert an element at position 1
  push_at(&MyList, 200, 1);
  PrintList(MyList);
  return 0;
}
```

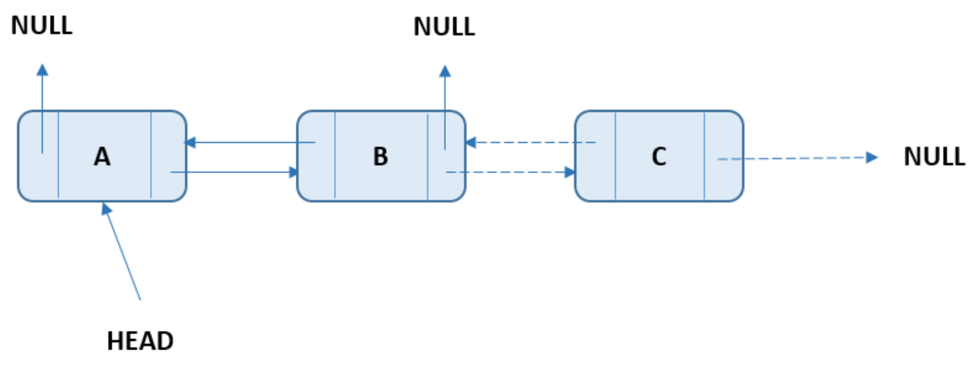
Doubly Linked List - Delete the first node



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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

//Add new element at the end of the list
void push_back(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    while(temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}

//Delete first node of the list
void pop_front(struct Node** head_ref) {
  if(*head_ref != NULL) {
    struct Node *temp = *head_ref;
    *head_ref = (*head_ref)->next;
    temp = NULL;
    if(*head_ref != NULL)
      (*head_ref)->prev = NULL;
  }
}

//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}
```

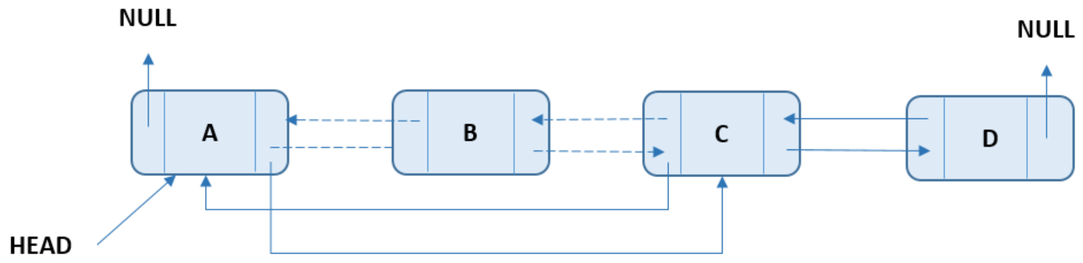
```
int main() {
  struct Node* MyList = NULL;

  //Add four elements in the list.
  push_back(&MyList, 10);
  push_back(&MyList, 20);
  push_back(&MyList, 30);
  push_back(&MyList, 40);
  PrintList(MyList);

  //Delete the first node
  pop_front(&MyList);
  PrintList(MyList);

  return 0;
}
```

Doubly Linked List - Delete the last node



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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

//Add new element at the end of the list
void push_back(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    while(temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}

//Delete last node of the list
void pop_back(struct Node** head_ref) {
  if(*head_ref != NULL) {
    if((*head_ref)->next == NULL) {
      *head_ref = NULL;
    } else {
      struct Node* temp = *head_ref;
      while(temp->next->next != NULL)
        temp = temp->next;
      struct Node* lastNode = temp->next;
      temp->next = NULL;
      free(lastNode);
    }
  }
}

//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}
int main() {
  struct Node* MyList = NULL;

  //Add four elements in the list.
  push_back(&MyList, 10);
  push_back(&MyList, 20);
  push_back(&MyList, 30);
  push_back(&MyList, 40);
  PrintList(MyList);
  //Delete the last node
  pop_back(&MyList);
  PrintList(MyList);

  return 0;
}
```

Doubly Linked List - Delete a node at the given position



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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

//Add new element at the end of the list
void push_back(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    while(temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}

//Delete an element at the given position
void pop_at(struct Node** head_ref, int position) {
  if(position < 1) {
    printf("\nposition should be >= 1.");
  } else if (position == 1 && *head_ref != NULL) {
    struct Node* nodeToDelete = *head_ref;
    *head_ref = (*head_ref)->next;
    free(nodeToDelete);
    if(*head_ref != NULL)
      (*head_ref)->prev = NULL;
  } else {
    struct Node *temp;
    temp = *head_ref;
    for(int i = 1; i < position-1; i++) {
      if(temp != NULL) {
        temp = temp->next;
      }
    }
    if(temp != NULL && temp->next != NULL) {
      struct Node* nodeToDelete = temp->next;
      temp->next = temp->next->next;
      if(temp->next->next != NULL)
        temp->next->next->prev = temp->next;
      free(nodeToDelete);
    } else {
      printf("\nThe node is already null.");
    }
  }
}
```

```
//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}

// test the code
int main() {
  struct Node* MyList = NULL;

  //Add three elements at the end of the list.
  push_back(&MyList, 10);
  push_back(&MyList, 20);
  push_back(&MyList, 30);
  PrintList(MyList);

  //Delete an element at position 2
  pop_at(&MyList, 2);
  PrintList(MyList);

  //Delete an element at position 1
  pop_at(&MyList, 1);
  PrintList(MyList);

  return 0;
}
```

Doubly Linked List - Count nodes

```
int countNodes(struct Node* head_ref) {

  //1. create a temp node pointing to head
  struct Node* temp = head_ref;

  //2. create a variable to count nodes
  int i = 0;

  //3. if the temp node is not null increase
  //   i by 1 and move to the next node, repeat
  //   the process till the temp becomes null
  while (temp != NULL) {
    i++;
    temp = temp->next;
  }

  //4. return the count
  return i;
}
```

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

//node structure
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

//Add new element at the end of the list
void push_back(struct Node** head_ref, int newElement) {
  struct Node *newNode, *temp;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = newElement;
  newNode->next = NULL;
  newNode->prev = NULL;
  if(*head_ref == NULL) {
    *head_ref = newNode;
  } else {
    temp = *head_ref;
    while(temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}

//count nodes in the list
int countNodes(struct Node* head_ref) {
  struct Node* temp = head_ref;
  int i = 0;
  while (temp != NULL) {
    i++;
    temp = temp->next;
  }
  return i;
}

//display the content of the list
void PrintList(struct Node* head_ref) {
  struct Node* temp = head_ref;
  if(head_ref != NULL) {
    printf("The list contains: ");
    while (temp != NULL) {
      printf("%i ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  } else {
    printf("The list is empty.\n");
  }
}

int main() {
  struct Node* MyList = NULL;

  //Add four elements in the list.
  push_back(&MyList, 10);
  push_back(&MyList, 20);
  push_back(&MyList, 30);
  push_back(&MyList, 40);

  //Display the content of the list.
  PrintList(MyList);
  //number of nodes in the list
  printf("No. of nodes: %i",countNodes(MyList));
  return 0;
}
```