DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

COURSE NAME : 19CS307- DATA STRUCTURES

II YEAR / III SEMESTER

UNIT I – LINEAR STRUCTURES-LIST

Singly Linked List Operations

Printing each node data in a linked list Implementation

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

int main()
{
  //node structure
  struct node
  {
     int data;
     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->data   = 10;
  middle->data = 20;
  last->data   = 30;

  //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("%d->",temp->data);
     temp = temp->next;
  }
  printf("NULL");

  return 0;
}
```

Inserting a node at the beginning of a linked list

The new node will be added 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 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;
    }
```

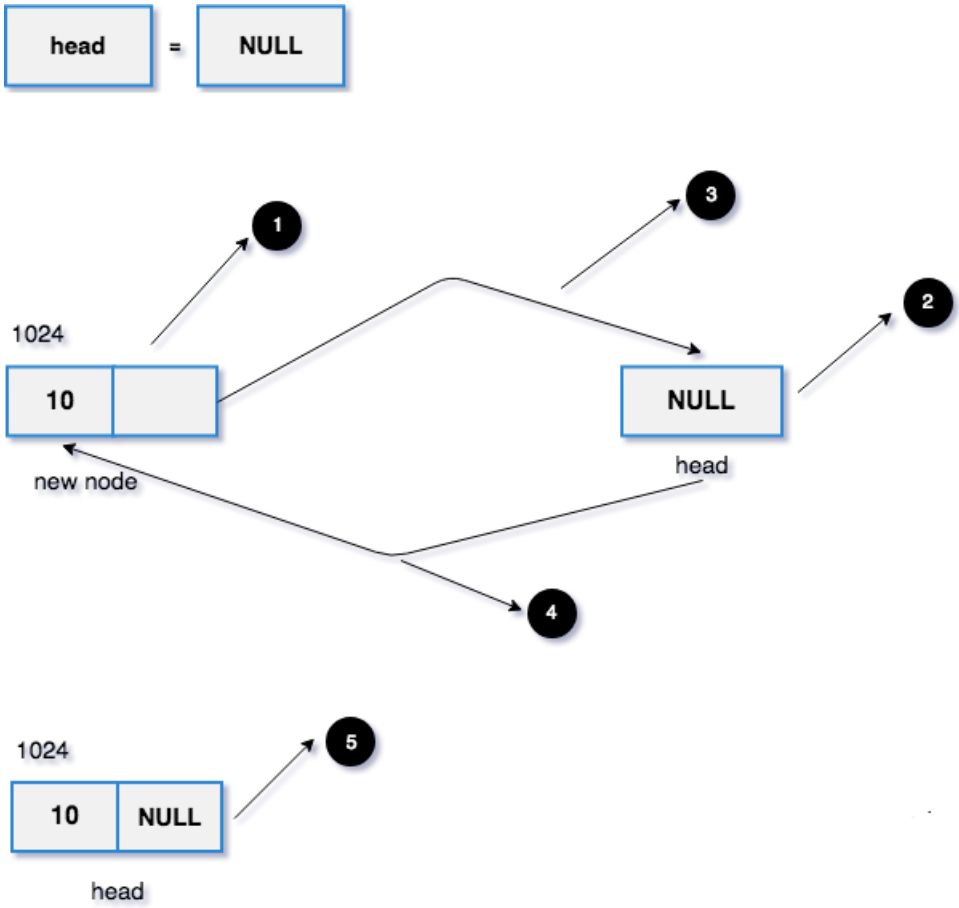
4.Make the new node as the head node

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

     *head = newNode;

}
```

Example

Insert 10

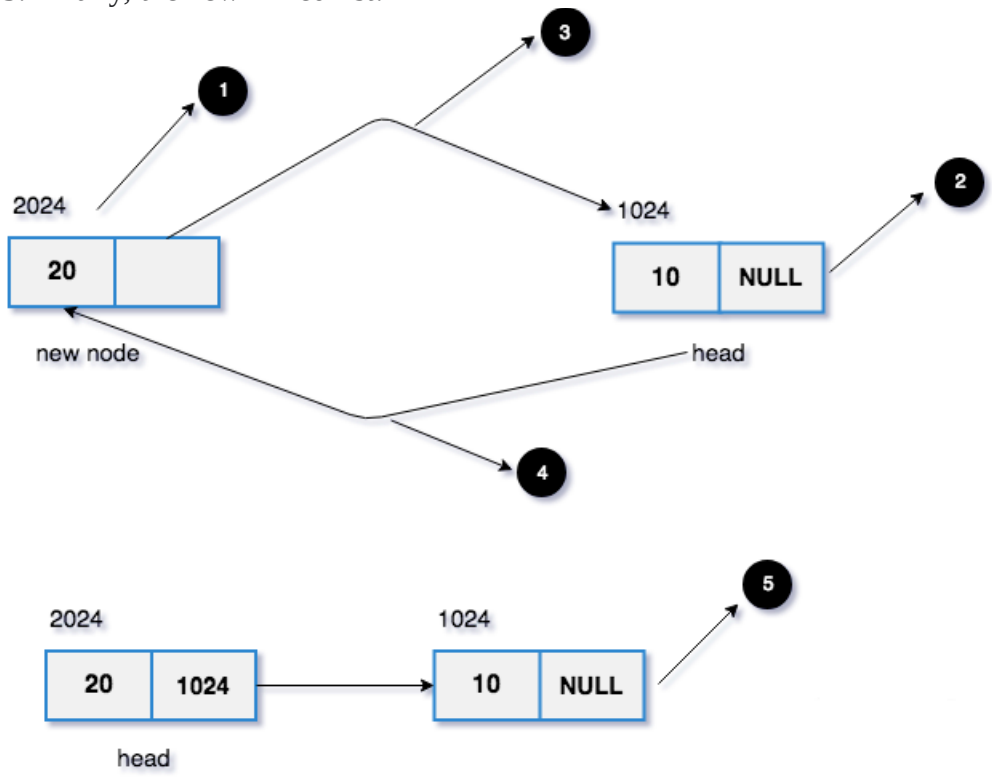
Steps
1. A newly allocated node with data as 10.
2. Head points to NULL.
3. New node -> next points to the head which is NULL. So newnode->next = NULL.
4. Make the head points to the new node. Now, the head will hold the address of the new node w hich is 1024.
5. Finally, the new linked list.



Insert 20

Steps
1. A newly allocated node with data as 20.
2. Head points to the memory address 1024 (It has only one node. 10->NULL).
3. New node -> next points to the head which is 1024. So newnode->next = 1024 (10->NULL) w ill be added back to the new node.

4. Make the head points to the new node. Now, the head will hold the address of the new node w hich is 2024.
5. Finally, the new linked list.



Program to insert a node at the beginning of linked list

```
#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,10);
    addFirst(&head,20);
    printList(head);

    return 0;
}
```

Inserting a node at the end of a linked list

The new node will be added at the end of the linked list.

Example
Input
Linked List : 10 20 30 40 NULL.

50

Output
Linked List : 10 20 30 40 50 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;
}
```

4. Otherwise, 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

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

After that, we have to make the last node-> next as the new node. i.e. last node->next = new node;

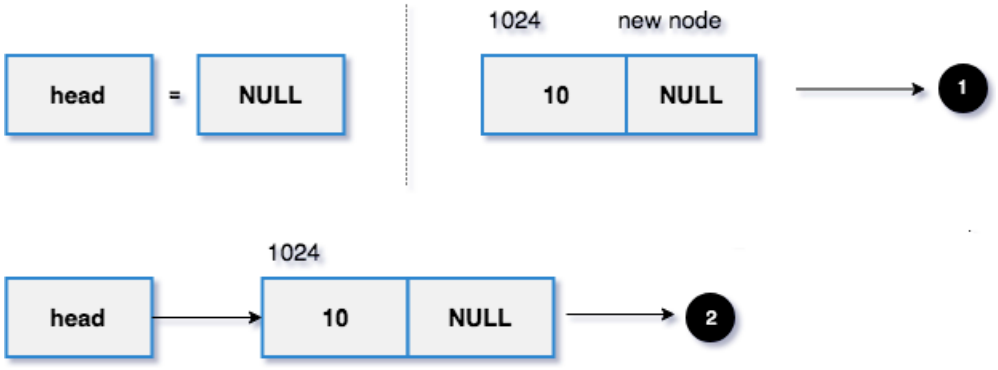
Example

Insert data 10.

The head is NULL initially.

1. The new node with data as 10 and reference is NULL (address 1024).

2. Since it is the first node, make the head node points to the newly allocated node.



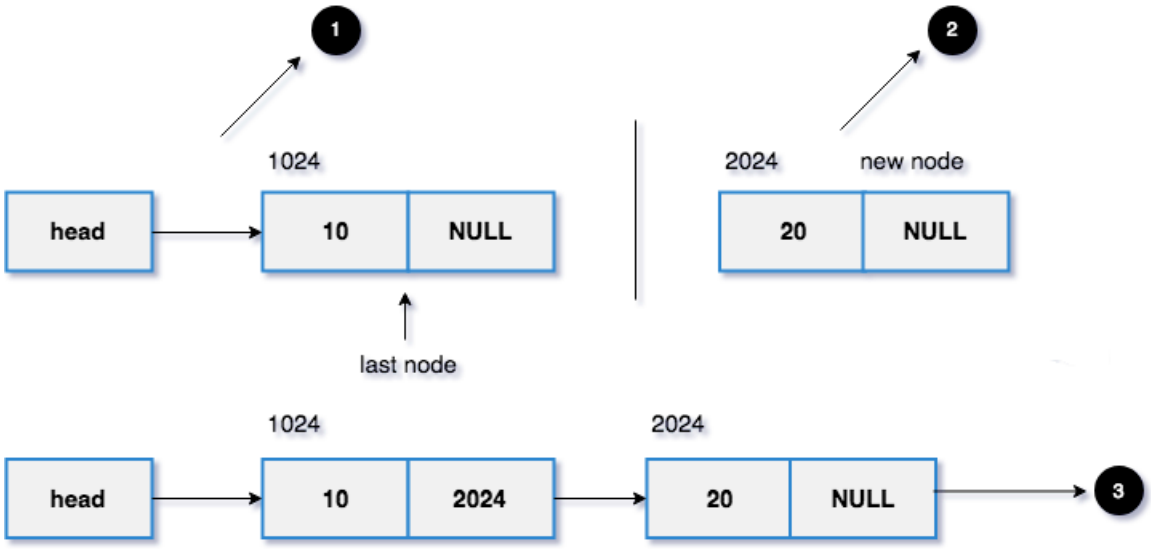
Insert data 20.

1. The head points to the memory address 1024 and it is the last node.

2. The new node with data as 20 and reference is NULL (address 2024).

set last node =>next = new node. The new node added at the end of the linked list.

3. Finally, the new linked list.



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,10);
addLast(&head,20);
printList(head);
return 0;
}
```

Searching a node in singly linked list

Check whether the given key is present or not in the linked list.

Example

Linked List : 10 20 30 40 NULL.

Input

20

Output

Search 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;
    }
}
```

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;

}
```

Example

Linked List : 10 20 30 NULL

Key : 100

Steps:

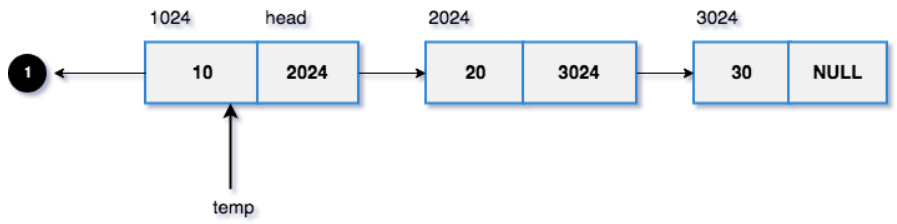
1. temp->data = 10. key = 100. temp->data != key. Hence move the temp variable to the next node.

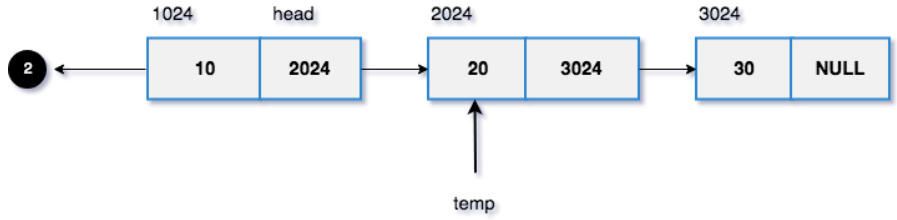
2. temp->data = 20. key = 100. temp->data != key. Hence move the temp variable to the next node.

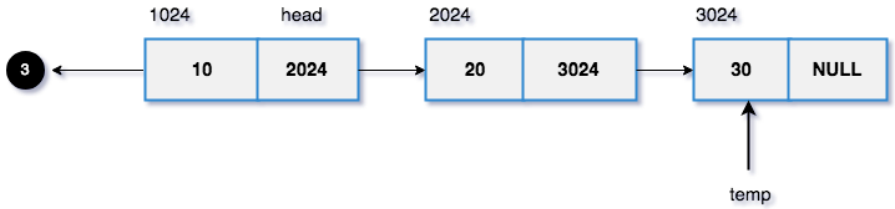
3. temp->data = 30. key = 100. temp->data != key. Hence move the temp variable to the next node which is NULL.

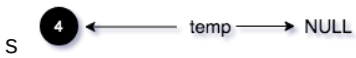
4. Finally, the program execution will come out of the loop. So, it will return -1.

"Search Not Found".









s

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.
    if(searchNode(head,20) == 1)
        printf("Search Found\n");
    else
        printf("Search Not Found\n");
    return 0;
}
```

Deleting a node in linked list

Delete a given node from the linked list.

Example

Linked List : 10 20 30 NULL

Input

20

Output

10 30 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,

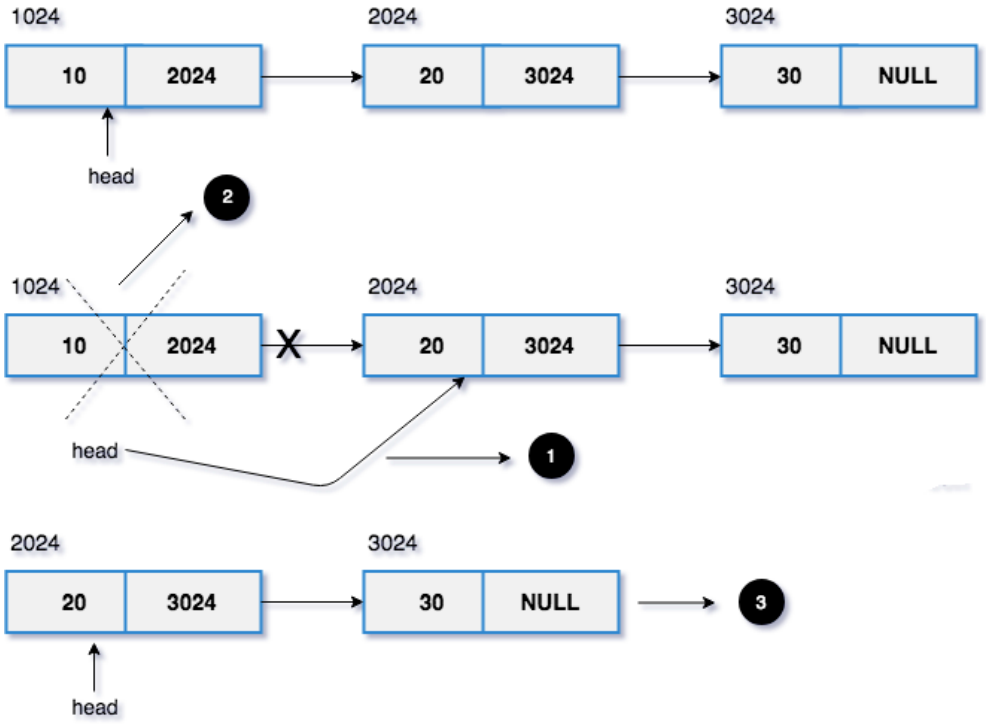
From the current node, check whether the next node has the given key

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);
} }
```



1. Make the head points to the next node.

2. Free the head node's memory.

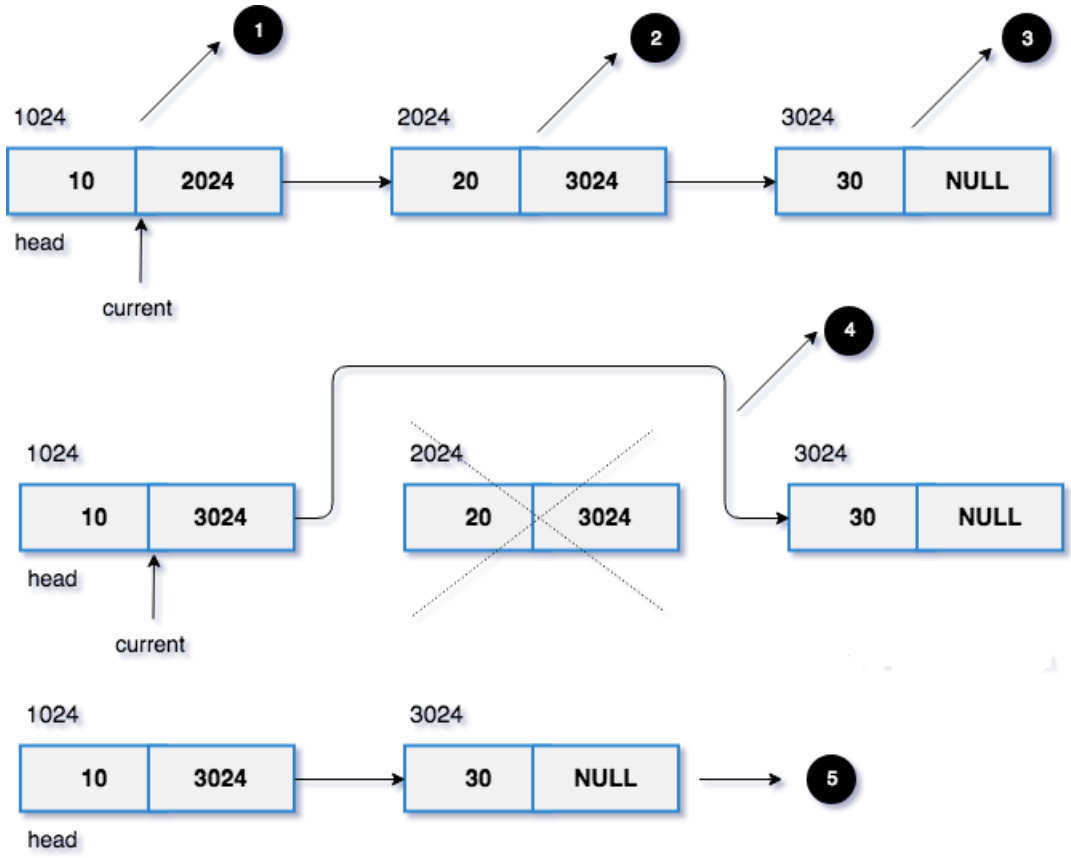
3. Finally, the new linked list.

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;
    }     }}
```

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.

Implementation of deleting a node in linked list

Example

```
#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);
    return 0;
}
```

Advantage of Singly Linked list:-

1) Insertions and Deletions can be done easily.

2) It does not need movement of elements for insertion and deletion.

3) It space is not wasted as we can get space according to our requirements.

4) Its size is not fixed.

5) It can be extended or reduced according to requirements.

6) Elements may or may not be stored in consecutive memory available; even then we can store the data in computer.

7) It is less expensive.

Disadvantage of Singly Linked list:-

1) It requires more space as pointers are also stored with information.

2) Different amount of time is required to access each element.

3) If we have to go to a particular element then we have to go through all those elements that come before that element.

4) We cannot traverse it from last & only from the beginning.

5) It is not easy to sort the elements stored in the linear linked list.

Applications of Singly Linked list:-

- One of the applications of singly linked list is in applications like a photo viewer, for watching similar photos in a continuous manner in the form of a slide show.
- Strategy for file allocation schemes by Operating System. Singly Linked List can be used to keep track of free space in the secondary disk. All the free spaces can be linked together