PDS LAB – 8 (Section-5) Date: 27th March 2017
Structures & Linked Lists
Tutorial Problems

1. Write a C program to input a student record which contains the following details: {int roll, char name[20], int age}. Define the structure to represent the details of the student, input the attributes of the student record through keyboard and print the same.

2. Modify the above program to enter and print multiple student records say "n" taken from the keyboard. First try with arrays and later replace array with pointer. (Hint: in case of pointers, you have to allocate the memory).
   Example Output:
   Student [1] = 25 Krish 18
   Student [2] = 28 Ram 19
   Student [3] = 22 Satish 17

3. Create 3 nodes, where each node holds one student record. Let the student records be S1, S2 and S3. Each node carries the details of the student (mentioned in the Tut-1 problem) and pointer to the next student. Create a linked list with S1 as the 1st node and S3 as the last node. First enter the details of the student in each node and then provide the link from one node to other. Once the linked list is created, demonstrate the traversal by printing the records in sequence.

Assignment Problems (For All Students)

1. Tut-2 problem

2. Write a C program to read (input) multiple student records from keyboard. Write C functions separately for sorting the student records based on each attribute. Demonstrate the above functions by printing the attribute-based sorted records.
   Example Output:

Original student records (before sorting)	*Sorted records based on student roll*
Student [1] = 22 Satish 17	*Student [1] = 22 Satish 17*
Student [2] = 28 Ram 19	*Student [2] = 25 Krish 18*
Student [3] = 25 Krish 18	*Student [3] = 28 Ram 19*
Sorted records based on student name	*Sorted records based on student age*
Student [1] = 25 Krish 18	*Student [1] = 22 Satish 17*
Student [2] = 28 Ram 19	*Student [2] = 25 Krish 18*
Student [3] = 22 Satish 17	*Student [3] = 28 Ram 19*

3. Write a C program to read (input) multiple student records from keyboard. The attributes of the student record consists of {int roll, char name[20], int age, struct marks M}, marks {float sub1, float sub2, float sub3, float sub4, float total, char result}. Enter the number of

students' records (say "n") through keyboard. Then call a function to enter the attributes of the students records. Write a C function to sort the records based on total marks and provide the grade as per IITKGP grading scheme.

Example Output:

Original student records (before sorting)										
Student [1]	=	22	Satish	17	66	70	87	74	297	B
Student [2]	=	28	Ram	19	55	90	99	95	339	A
Student [3]	=	25	Krish	18	22	33	56	90	201	D
Sorted records based on student total marks										
Student [1]	=	25	Krish	18	22	33	56	90	201	D
Student [2]	=	22	Satish	17	66	70	87	74	297	B
Student [3]	=	28	Ram	19	55	90	99	95	339	A

4. A linked list, where the node of the linked list (data component) holds the student record {int roll, char name[20], int age}. Write C functions to (i) create the list and fill the nodes with student records and (ii) print the student records by traversing the list.

5. Continuation to the above problem: write C functions to (i) insertion of a node (at the beginning, at the end, in between based on some attribute of the student record), (ii) deletion of a node (at the beginning, at the end, in between based on some attribute of the student record) and (iii) print the student records in reverse order.

Assignment Problems (For those who completed 5 assignment problems)

1. Write C program to compute the interest and total amount (principle + interest) for all the account holders of a bank. The details of the account holder are as follows: {int acc_no, char name[20], float principle, struct date deposit_date, float interest, float total}. The attributes of Date are {int date, int month, int year}. Define appropriate structures and input the number of account holders, rate of interest through keyboard. Write C functions to (i) enter the details of account holders, (ii) compute the difference between deposit date and target date, (iii) compute the interest and total amount for each account holder and (iv) print the records in systematic manner.

2. Assume that the linked list holds the data consisting of student records mentioned in Tut-2 problem. Write separate C function to sort the records based on each attribute of the student record. Print the student records in the original order (before sorting) and in sorted order as per each specific attribute.

Defining a Structure

- The composition of a structure may be defined as

```
struct tag {
            member 1;
            member 2;
            :
            member m;
        };
```

- **`struct`** is the required keyword
- **`tag`** is the name of the structure
- **`member 1,member 2,..`** are individual member declarations

Defining a Structure

- Example

```
struct student {
                    char name[30];
                    int roll_number;
                    int total_marks;
                    char dob[10];
                };
```

- Defining structure variables

```
struct student s1, sList[100];
```

A new data-type

Point to be Noted

- The individual members can be ordinary variables, pointers, arrays, or other structures.
  - The member names within a particular structure must be distinct from one another.
  - A member name can be the same as the name of a variable defined outside of the structure.
- Once a structure has been defined, the individual structure-type variables can only be declared then.
- Each member in a structure can be accessed with (.) operator called scope resolution operator

```
struct student s1, sList[100];

s1.name;  sList[5].roll_number;
```

Example: Complex Number Addition

```
#include <stdio.h>
main()
{
      struct complex
      {
          float real;
          float complex;
      } a, b, c;

      scanf ("%f %f", &a.real, &a.complex);
      scanf ("%f %f", &b.real, &b.complex);
      c.real = a.real + b.real;
      c.complex = a.complex + b.complex;
}
```

Scope restricted within main()

Structure definition and Variable Declaration

Reading a member variable

Accessing members

//Program to input and print 4 student records using pointers

```
#include <stdio.h>
#include <stdlib.h>
struct std
{
   char name[20];
   float marks;
   int age;
};

int main()
{
struct std *x;
int i,n;

printf("Enter the number of Students : \n");
scanf("%d",&n);

x=(struct std*)malloc(n*sizeof(struct std));

printf("Enter the details of students: \n");

for(i=0;i<n;i++)
{
scanf("%s",(x+i)->name);
scanf("%f",&(x+i)->marks);
scanf("%d",&(x+i)->age);
}
printf("The details of students Entered are: \n");
for(i=0;i<n;i++)
{
printf("%s\t",(x+i)->name);
printf("%f\t",(x+i)->marks);
printf("%d\t",(x+i)->age);
printf("\n");
}

return (0);

}
```

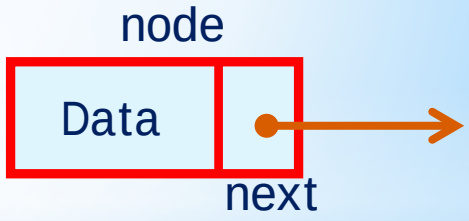
Defining a Node of a Linked List

Each structure of the list is called a node, and consists of two fields:

- Item (or) data
- Address of the next item in the list (or) pointer to the next node in the list

How to define a node of a linked list?

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



Note:

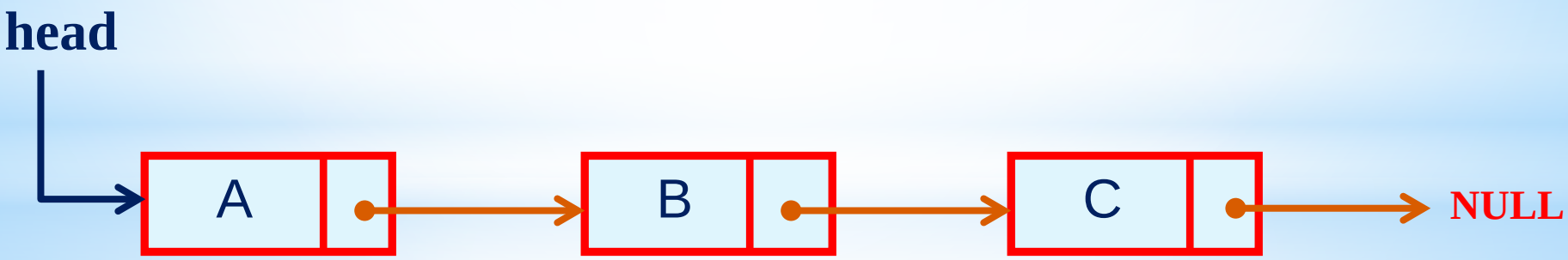
Such structures which contain a member field pointing to the same structure type are called self-referential structures.

Types of Lists: Single Linked List

Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.

Single linked list (or simply **linked list)**

- A head pointer addresses the first element of the list.
- Each element points at a successor element.
- The last element has a link value NULL.



Example 1: Creating a Single Linked List

Linked list to store and print roll number, name and age of 3 students.

```
#include <stdio.h>
struct stud
{
   int roll;
   char name[30];
   int age;
   struct stud *next;
};
main()
{
   struct stud n1, n2, n3;
   struct stud *p;
   scanf (“%d %s %d”, &n1.roll, n1.name, &n1.age);
   scanf (“%d %s %d”, &n2.roll,n2.name, &n2.age);
   scanf (“%d %s %d”, &n3.roll,n3.name, &n3.age);
```

Example 1: Creating a Single Linked List

```
    n1.next = &n2 ;
    n2.next = &n3 ;
    n3.next = NULL ;
/* Now traverse the list and print the elements */
    p = &n1 ;          /* point to 1st element */
    while (p != NULL)
    {
        printf (“\n %d %s %d”, p->roll, p->name, p->age);
        p = p->next;
    }
}
```

Example 1: Illustration

The structure:

```
struct stud
{
  int roll;
  char name[30];
  int age;
  struct stud *next;
};
```

Also assume the list with three nodes n1, n2 and n3 for 3 students.

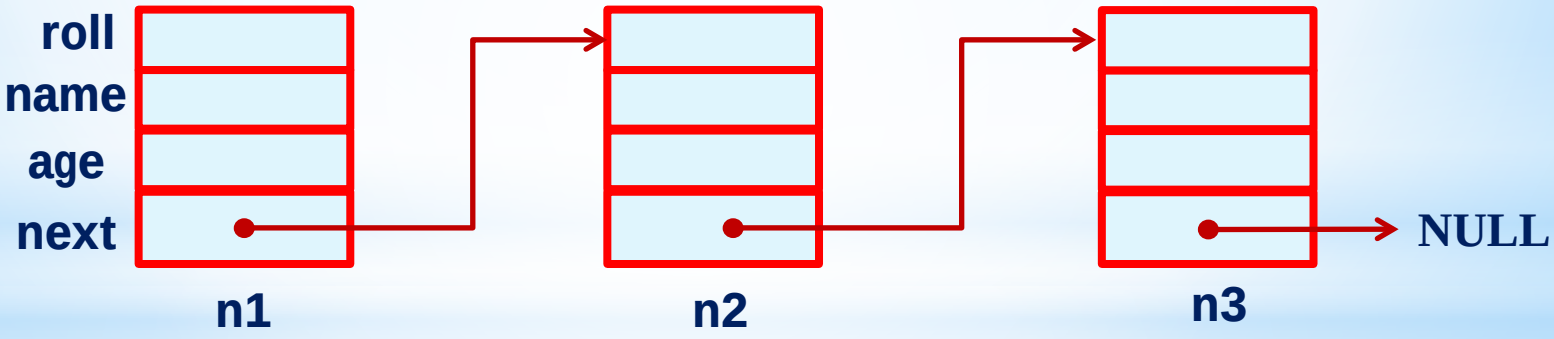
```
struct stud n1, n2, n3;
```

Example 1: Illustration

To create the links between nodes, it is written as:

```
n1.next = &n2 ;
n2.next = &n3 ;
n3.next = NULL ;  /* No more nodes follow */
```

- Now the list looks like:



Example 2: Creating a Single Linked List

C-program to store 10 values on a linked list reading the data from keyboard.

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

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

void createList(int n);          /* Functions to create a list*/

int main()
{
    int n;
    printf("Enter the total number of nodes: ");
    scanf("%d", &n);
    createList(n);
    return 0;
}
```

Example 2: Creating a Single Linked List

```
void createList(int n)
{
    struct node *newNode, *temp;
    int data, i;

    /* A node is created by allocating memory to a structure */
    newNode = (struct node *)malloc(sizeof(struct node));

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

        newNode->data = data; //Links the data field with data
        newNode->next = NULL; //Links the address field to NULL
        header = newNode;     //Header points to the first node
        temp = newNode;       //First node is the current node
```

Example 2: Illustration

- To start with, we have to create a node (the first node), and make header point to it.

```
newNode = (struct node *)malloc(sizeof(struct node));
newNode->data = data;      //Links the data field with data
newNode->next = NULL;      //Links the address field to NULL
header = newNode;
temp = newNode;
```

It creates a single node. For example, if the data entered is 100 then the list look like

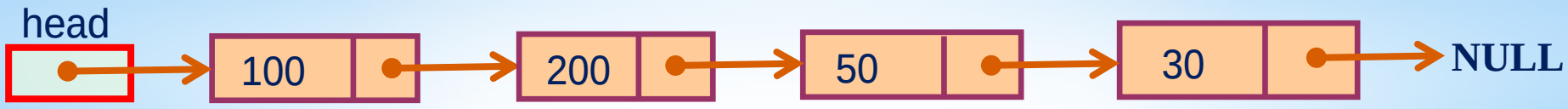
Creating a single linked list

If we need n number of nodes in the linked list:

- Allocate n newNodes, one by one.
- Read in the data for the newNodes.
- Modify the links of the newNodes so that the chain is formed.

```
newNode = (struct node *)malloc(sizeof(struct node));
newNode->data = data;     //Links the data field of newNode with data
newNode->next = NULL;     //Links the address field of newNode with NULL

temp->next = newNode;     //Links previous node i.e. temp to the newNode
temp = temp->next;
```

It creates n number of nodes . For e.g. if the data entered is 200, 50, 30 then the list look like



Example 3: Creating a Single Linked List

C-program to copy an array to a single linked list.

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

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

int main()
{
    struct node *header, *newNode, *temp;
    int data, i, n, a[100];
    printf("Enter the total number of data: ");
    scanf("%d", &n);
    // Write code here to initialize the array a with n elements //
                                                                ...
```

Example 2: Creating a Single Linked List

```
/* A node is created by allocating memory to a structure */
newNode = (struct node *)malloc(sizeof(struct node));

/* If unable to allocate memory for head node */
if(newNode == NULL)
{
    printf("Unable to allocate memory.");
}
else
{
    newNode->data = a[0]; //Links the data field with data
    newNode->next = NULL; //Links the address field to NULL
    header = newNode;     //Header points to the first node
    temp = header;
```

Example 2: Creating a Single Linked List

```
    for(i = 1; i <= n; i++)
    {
      /* A newNode is created by allocating memory */
      newNode = (struct node *)malloc(sizeof(struct node));

      if(newNode == NULL)
      {
         printf("Unable to allocate memory.");
         break;
      }
      else
      {
         newNode->data = a[i]; //Links the data field of newNode with a[i]
         newNode->next = NULL; //Links the address field of newNode with NULL

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

Traversing a Linked List

Single Linked List: Traversing

Once the linked list has been constructed and `header` points to the first node of the list,

- Follow the pointers.
- Display the contents of the nodes as they are traversed.
- Stop when the `next` pointer points to NULL.

The function **`traverseList`**`(struct Node *)` is given in the next slide. This function to be called from **`main`**`()` function as:

```
int main()
{
    // Assume header, the pointer to the linked list is given as an input
    printf("\n Data in the list \n");
    traverseList(header);
    return 0;
}
```

Single linked list: Traversing

```
void traverseList(struct Node *header)
{
  struct node *temp;

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

Insertion in a Linked List

Single Linked List: Insertion

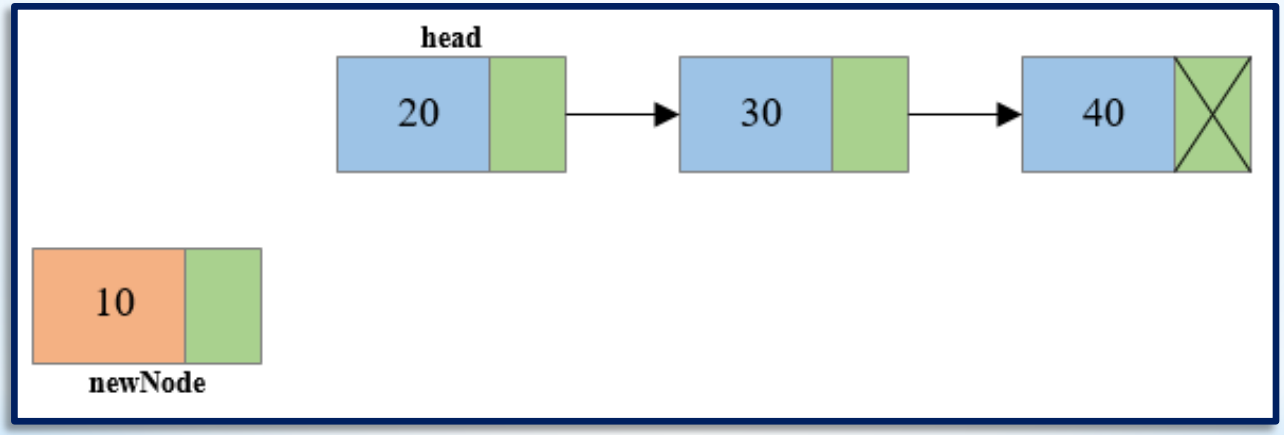
Insertion steps:

- Create a new node
- Start from the header node
- Manage links to
  - Insert at front
  - Insert at end
  - Insert at any position

Insertion at Front

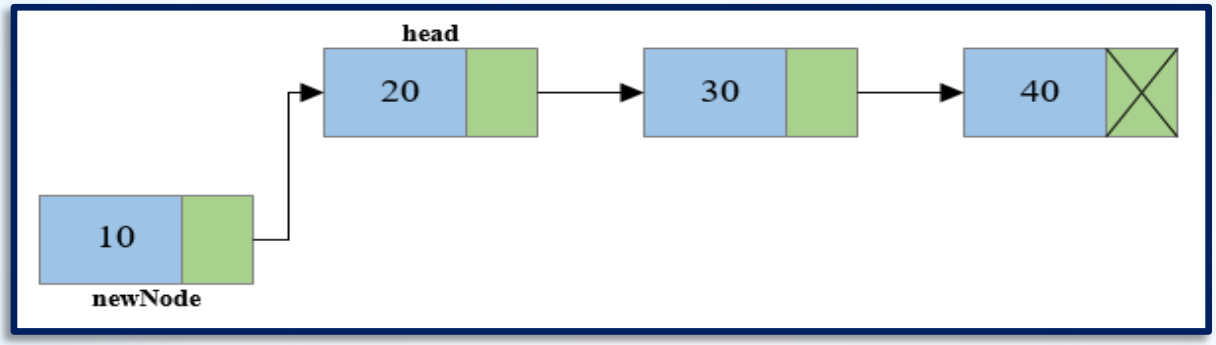
Steps to insert node at the beginning of singly linked list

Step 1: Create a new node.

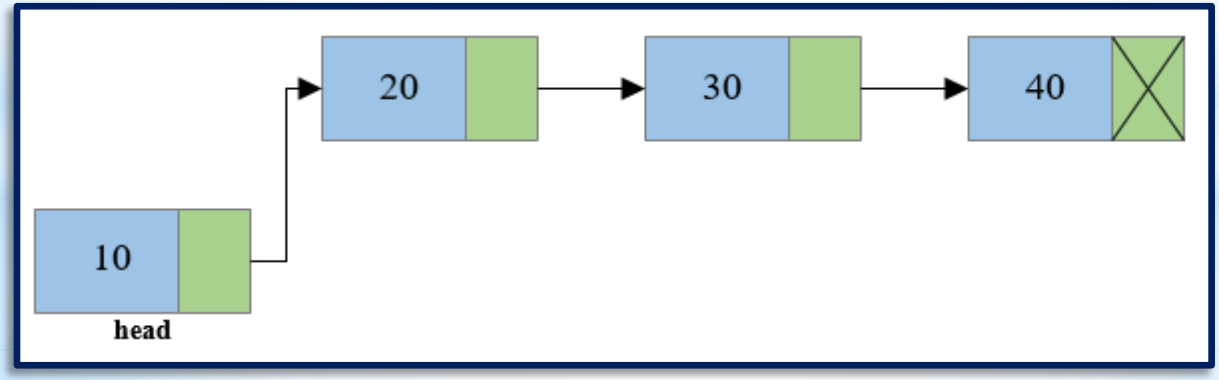


Insertion at Front

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



Step 3: Make the new node as the head node, i.e. now **head** node will point to **newNode**.



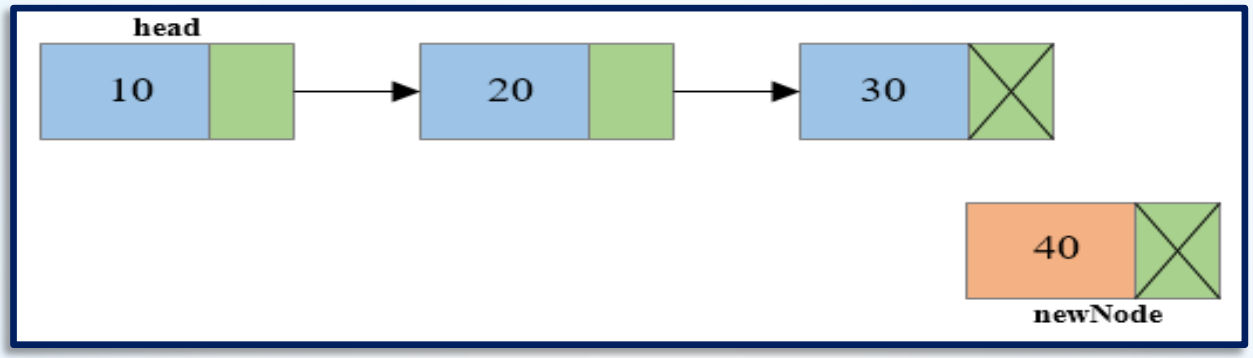
Insertion at front

```
/*Create a new node and insert 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;   //Links the data part
        newNode->next = head;   //Links the address part

        head = newNode;         //Makes newNode as first node

        printf("DATA INSERTED SUCCESSFULLY\n");
    }
}
```

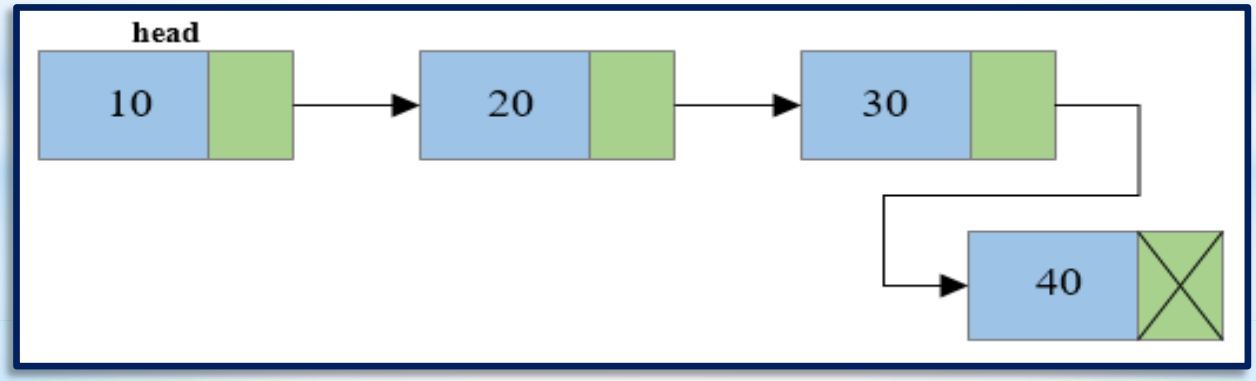
Single Linked List: Insertion at End

Steps to insert node at the end of Singly linked list

Step 1: Create a new node and make sure that the address part of the new node points to NULL. i.e. `newNode->next=NULL`



Step 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`).



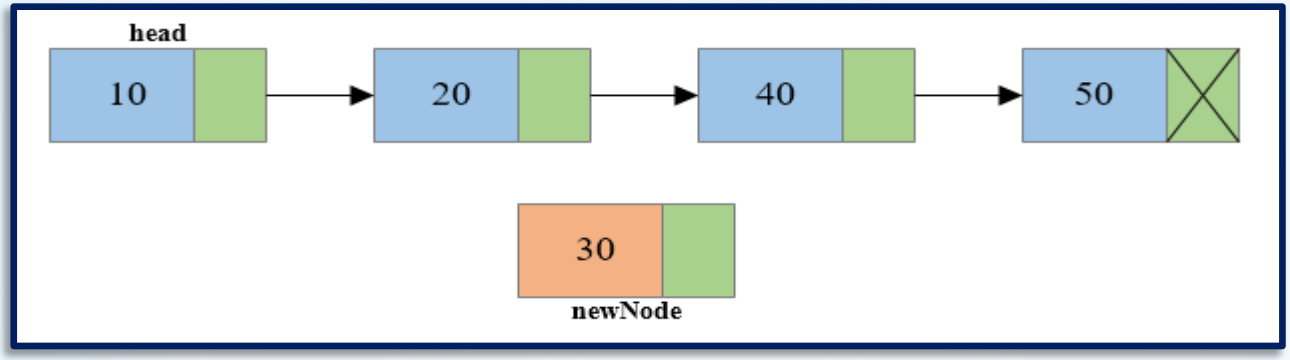
Insertion at End

```
/* Create a new node and insert at the end of the linked list. */
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; //Links the data part
        newNode->next = NULL;
        temp = head;

        while(temp->next != NULL) //Traverse to the last node
            temp = temp->next;

        temp->next = newNode; //Links the address part
        printf("DATA INSERTED SUCCESSFULLY\n");
    }
}
```

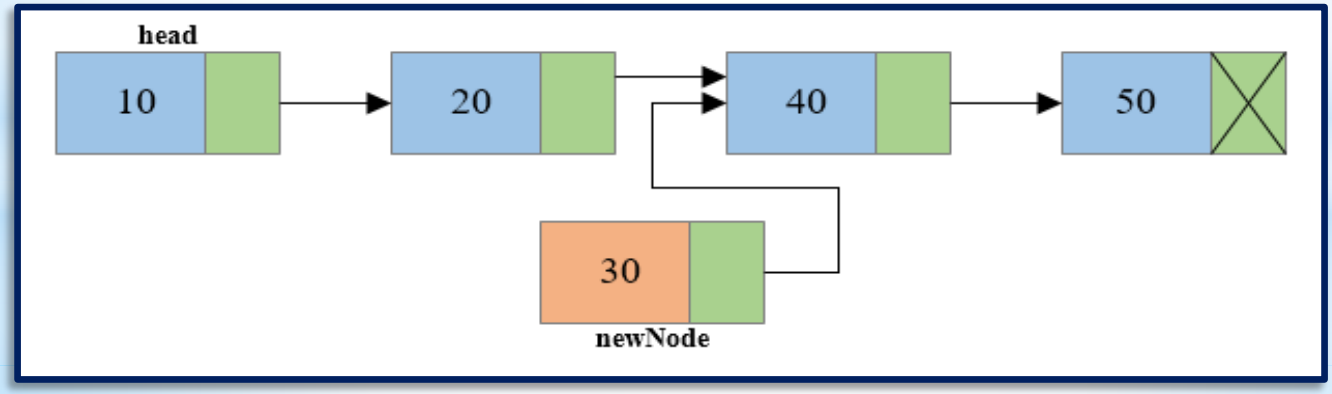
Single Linked List: Insertion at any Position

Steps to insert node at any position of Singly Linked List

Step 1: Create a new node.

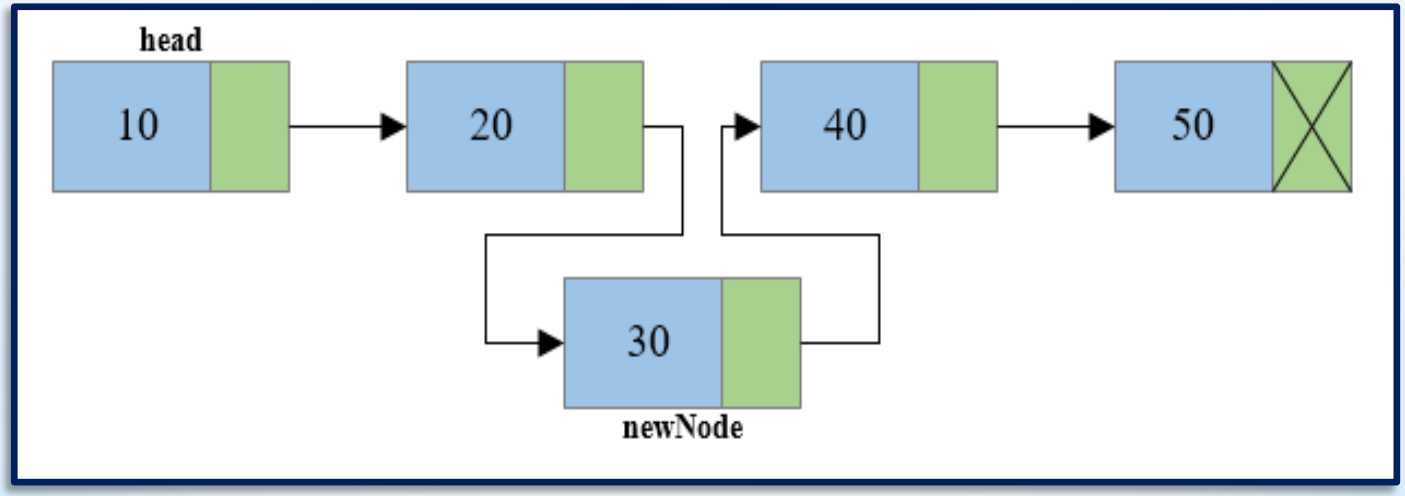


Step 2: Traverse to the n-1th position of the linked list and connect the new node with the n+1th node. (`newNode->next = temp->next`) where temp is the n-1th node.



Single Linked List: Insertion at any position

Step 3: 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.



Insertion at any Position

```
/* Create a new node and insert 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;   //Links the data part
        newNode->next = NULL;

        temp = head;
```

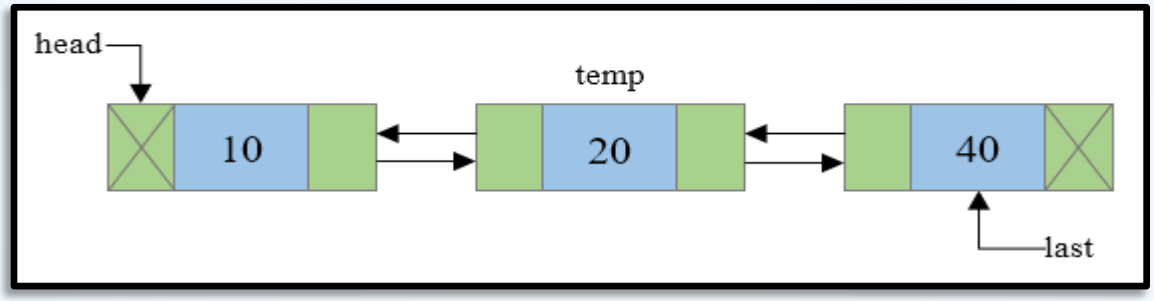
Insertion at any Position

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

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

            /* Links the 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");
        }
    }
}
```

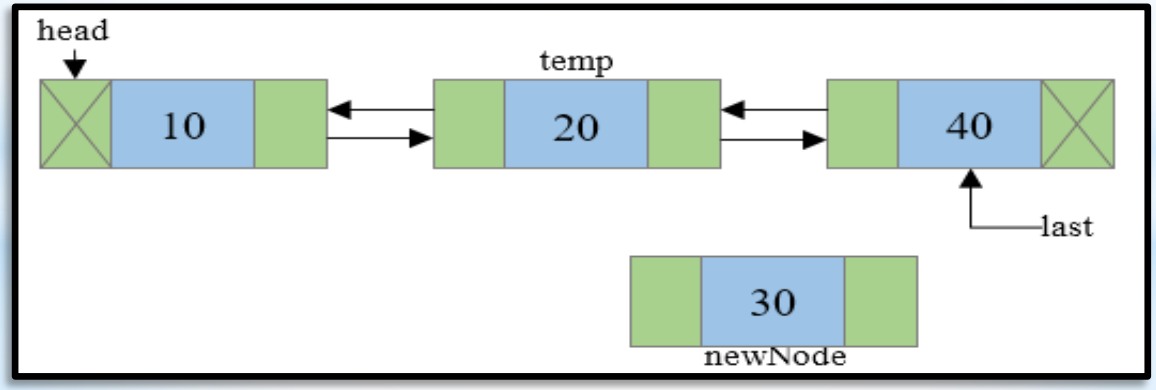
Double Linked List: Insertion at any Position

Steps to insert a new node at n^{th} position in a Doubly linked list.

Step 1: Traverse to N-1 node in the list, where N is the position to insert. Say **`temp`** now points to $N-1^{th}$ node.



Step 2: Create a **`newNode`** that is to be inserted and assign some data to its data field.



Deletion from a Linked List

Single Linked List: Deletion

Deletion steps

- Start from the header node
- Manage links to
  - Delete at front
  - Delete at end
  - Delete at any position
- freeingup the node as free space.

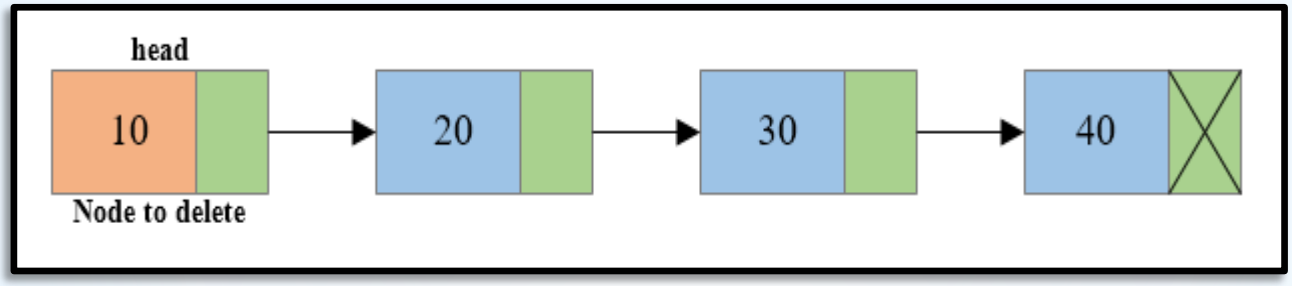
Free Memory after Deletion

- Do not forget to free() memory location dynamically allocated for a node after deletion of that node.
- It is the programmer's responsibility to free that memory block.
- Failure to do so may create a dangling pointer – a memory, that is not used either by the programmer or by the system.
- The content of a free memory is not erased until it is overwritten.

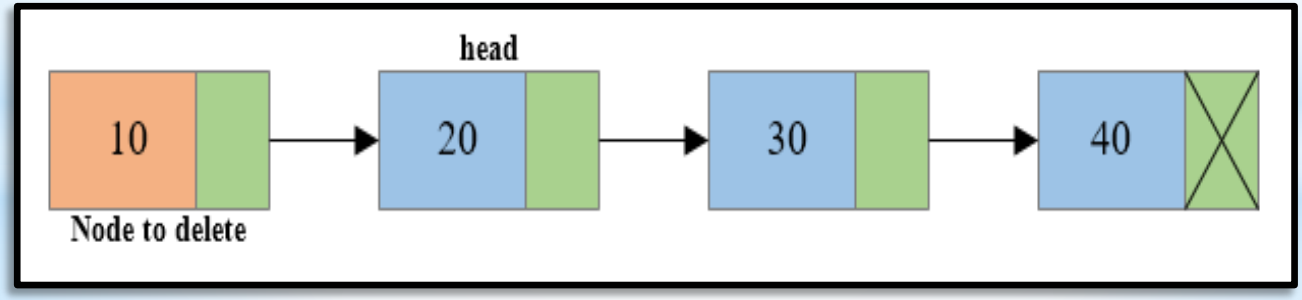
Single Linked List: Deletion at Front

Steps to delete first node of Singly Linked List

Step 1: Copy the address of first node i.e. **head** node to some temp variable say **toDelete**.

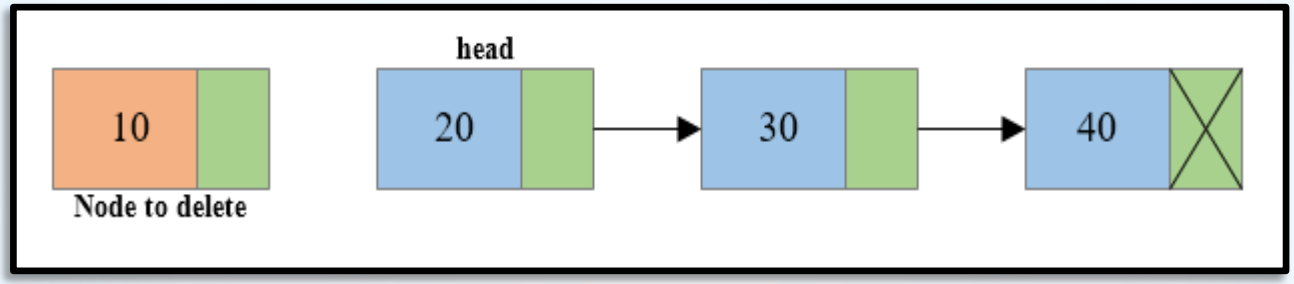


Step 2: Move the head to the second node of the linked list (`head = head->next`).

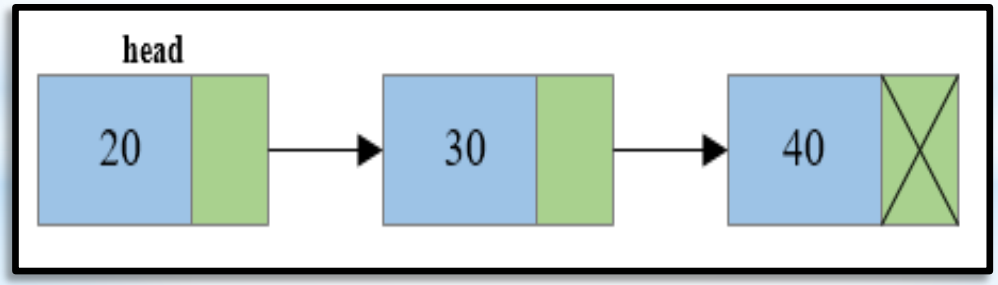


Single linked list: Deletion at front

Step 3: Disconnect the connection of first node to second node.



Step 4: Free the memory occupied by the first node.



Deletion at Front

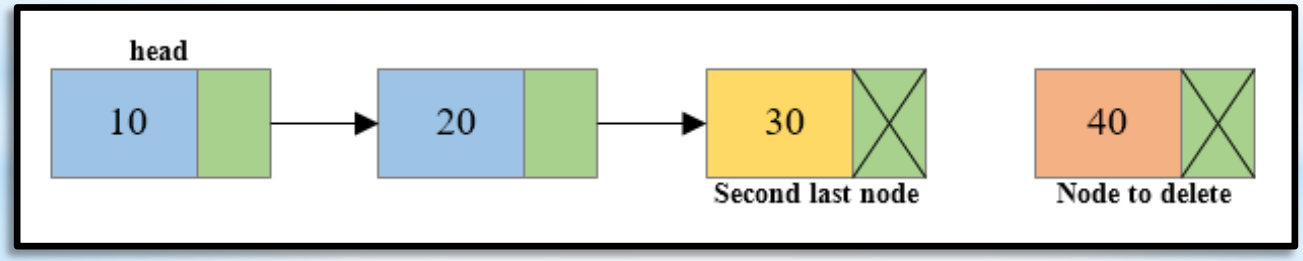
```
/* Delete the first node of the linked list */
void deleteFirstNode()
{
    struct node *toDelete;

    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");
    }
}
```

Single linked list: Deletion at End

Steps to delete last node of a Singly Linked List

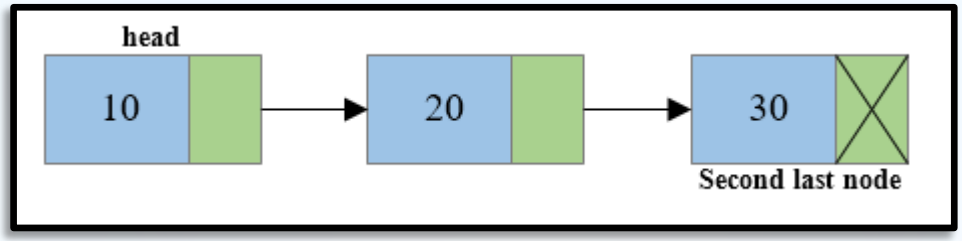
Step 1: Traverse to the last node of the linked list keeping track of the second last node in some temp variable say **secondLastNode**.

Step 2: If the last node is the head node then make the head node as NULL else disconnect the second last node with the last node i.e. `secondLastNode->next = NULL`



Single linked list: Deletion at End

Step 3: Free the memory occupied by the last node.



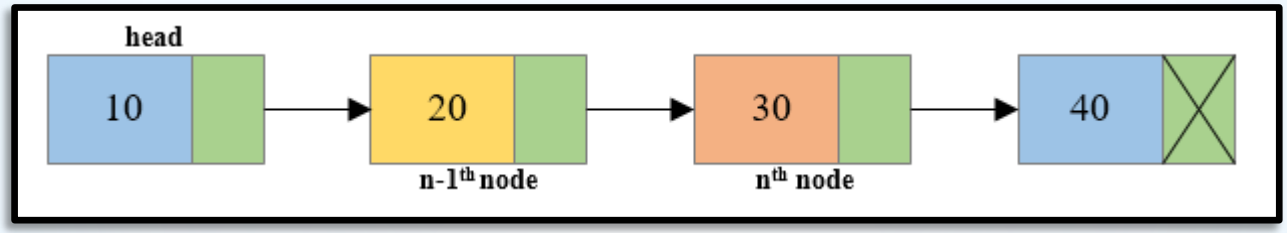
Deletion at End

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

    while(toDelete->next != NULL) /* Traverse to the last node of the list*/
     {
       secondLastNode = toDelete;
       toDelete = toDelete->next;
     }
    if(toDelete == head)
     {
       head = NULL;
     }
    else
     {
      /* Disconnects the link of second last node with last node */
       secondLastNode->next = NULL;
     }
      /* Delete the last node */
    free(toDelete);
}
```

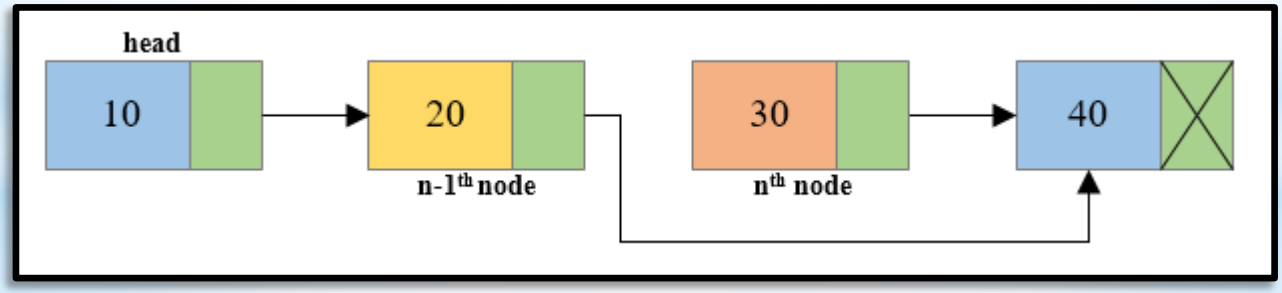
Single Linked List: Deletion at any Position

Steps to delete a node at any position of Singly Linked List

Step 1: 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**.

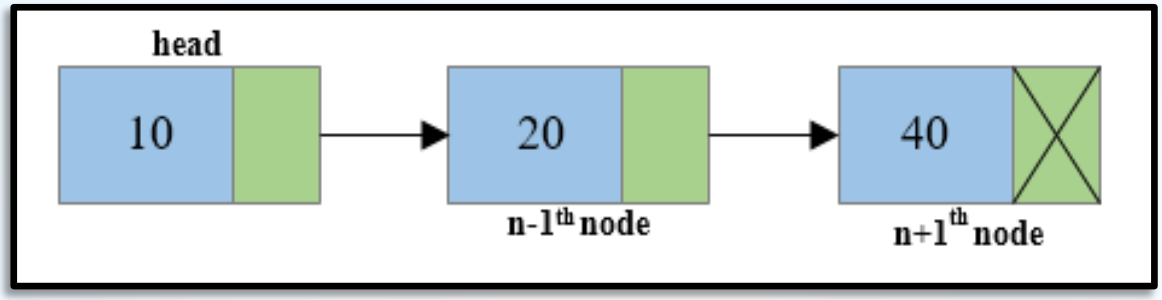


Step 2: Reconnect $n-1^{th}$ node with the $n+1^{th}$ node i.e. `prevNode->next = toDelete->next` (Where **prevNode** is $n-1^{th}$ node and **toDelete** node is the n^{th} node and `toDelete->next` is the n+1th node).



Single Linked List: Deletion at any Position

Step 3: Free the memory occupied by the n^{th} node i.e. **toDelete** node.



Deletion at any Position

```
/* Delete the node at any given position 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;
        }
```

Deletion at any Position

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

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

            /* Deletes the n node */
            free(toDelete);

            printf("SUCCESSFULLY DELETED NODE FROM MIDDLE OF LIST\n");
        }
        else
        {
            printf("Invalid position unable to delete.");
        }
    }
}
```