The Shirpur Education Society's
R. C. Patel College of Engineering & Polytechnic, Shirpur
Department of Computer Science & Engineering

Course Title - Data Structure Using C

Course Code - 313301

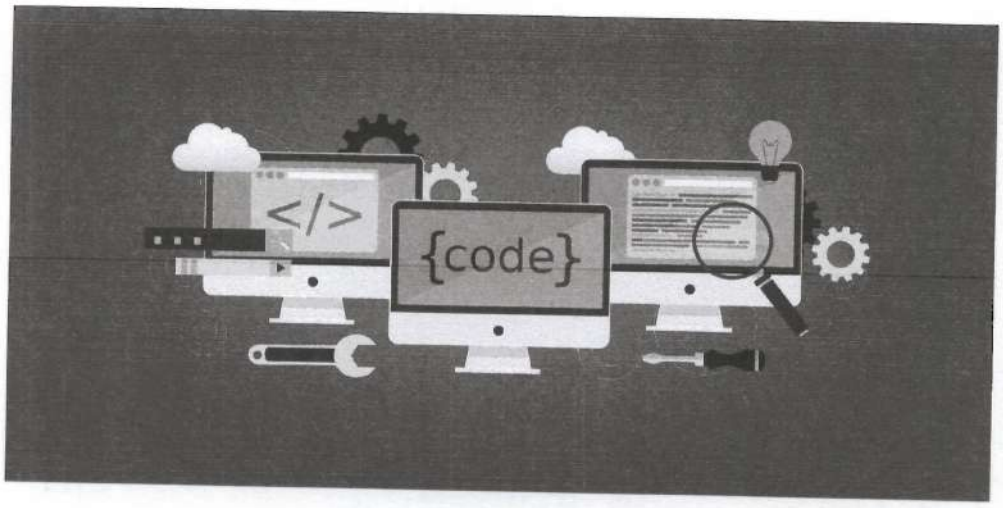
Programme Name - Computer Science &Engineering

Semester - Third

Unit - III Linked List

Total Marks:16

Prepared By: Mrs.Yogita D. Badgujar



Subject : Data Structure Using C

Subject code : 313301

- Syllabus

Unit - III

3.1 Difference between Static and Dynamic Memory Allocation.

3.2 Introduction to Linked List, Terminologies : Node, Address, Pointer, Information field / Data field, Next pointer, Null pointer, Empty List.

3.3 Type of Lists : Linear List, Circular List, Representation of Doubly Linked List.

3.4 Operations on a Singly Linked List : Creating a Linked List, Inserting a new node in a Linked List, Deleting a node from a Linked List, Searching a key in Linked List, Traversing a Singly Linked List.

3.5 Applications of Linked List.

Prepared By : Mrs. Y. D. Badgujar

Unit - III Linked List

Marks - 16

3.1 Difference between Static and Dynamic Memory Allocation

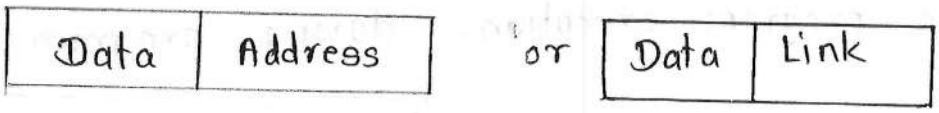
	Static Memory Allocation	Dynamic Memory Allocation
1.	Memory is allocated at the compile time.	Memory is allocated at the run time.
2.	Memory allocation is done before program execution.	Memory allocation is done during program execution.
3.	Once a memory is allocated it remains fixed during program execution.	Once a memory is allocated it can be changed during program execution by creating and releasing memory during program execution.
4.	Used in array	Used in Linked List
5.	Allocates memory from the stack	Allocates memory from the heap
6.	Memory is allocated automatically by the compiler	Memory is allocated only when there is an explicit call to malloc() and calloc() functions.
7.	Size of memory known at compile time.	size of memory determined at run time

3.2 Introduction to Linked List

i) For sequential memory allocation array is used. And when random memory is available then Linked List used.

ii) ~~List~~ Linked List consists of collection of nodes. Each node is divided into two parts first part is data part and second part is link part which stores the address of the next node.

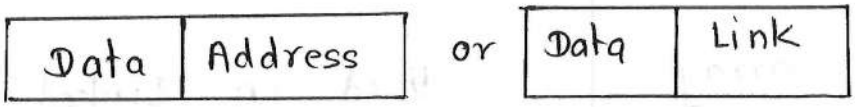
iii) Actual representation of a node is



Terminologies :

- Node : In a Linked list a node consist of a data and address part which stores address of the next node

Structure of a node



1) Data - Stores data or actual value

2) Address - Address of the next node.

Definition of node structure

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

- **Address :** Address is a memory location where node is stored. Represented by a pointer
  e.g. struct Node * next ;
  where it stores address of next node

- **Pointer :** Pointer is also called as link which stores the memory address of the next node.

Information field / Data field : Stores the actual data.
Example : int rollno ;
float marks ;

- **Next pointer :** Stores the memory address of the next node in singly linked list.
  In doubly linked list there are two pointer fields one for the next node and one for the previous node
  Example : Struct Node * next ;

- **Null pointer :** Is special pointer that does not point to any location which indicates end of the list or an empty list. Used in data structures like trees, linked lists etc. to indicate the end.
  Syntax :

```
ptr == NULL ;
```

Declaration of Null pointer

```
int * ptr = NULL;
```

- **Empty List :** Zero nodes in a list is called as ~~Empty~~ Empty list. Represented by a head pointer initialized to NULL.

Initialize the list as empty

```
struct Node* head = NULL;
```

3.3 Types of Lists

i) Linear List

ii) Circular List

iii) Doubly linked list

i) **Linear List :** Each node points at the successive node. The last node points to NULL. Nodes are linked with each other in sequential manner. It is also called as singly Linked List. List traversal is only in forward direction.

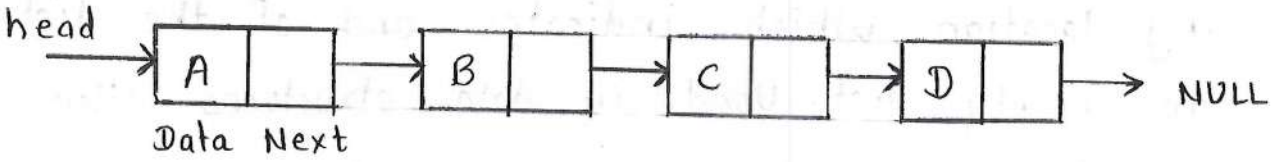


Figure 3.1 Linear List

ii) **Circular List :** First and last node are logically connected with each other. It creates a circular structure. Next part of the last node stores the address of the first node which connects rear end of list to its frond end.

Circular list allows continuous traversal of the list. There is no null to end the list. We can traverse a circular linked list from any node i.e. it has no beginning and no end.

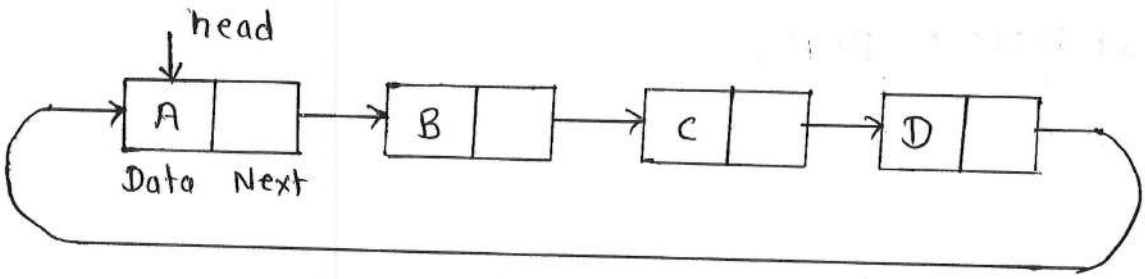


Figure 3.2 Circular Linked List

iii) Representation of Doubly Linked List

Also called as two-way linked list which is a more complex. Pointer points to the next as well as previous node. It contains three parts data, pointer to the next node and pointer to the previous node. List traversal is in both direction forward as well as backward direction.

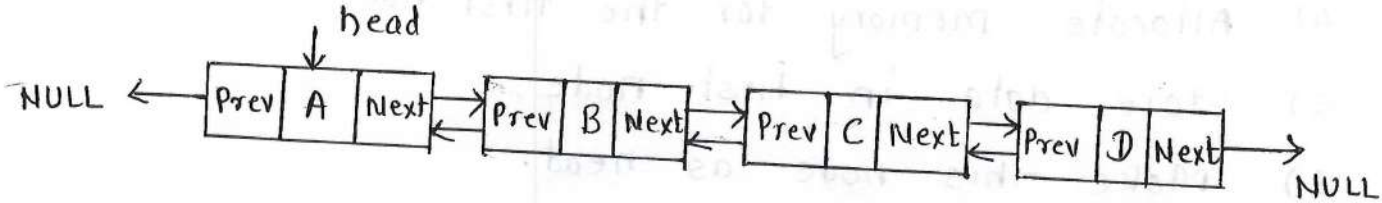


Figure 3.3 Doubly Linked List

In above figure

Data part stores actual data or value.

Prev part stores the address of previous node.

Next part stores the address of next node.

There are two NULL at the end of list as well as at the beginning of the list.

Definition of the node structure

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

3.4 Operations on a Singly Linked List :

- Creating a Linked List

Algorithm for creating a Linked List of size 3

Steps

~~1) Define node structure~~

1) Start
2) Define the node structure
3) Create the first node
4) Allocate memory for the first node
5) Store data in first node
6) Make this node as head.
7) To create a second node allocate memory for second node and store data in second node.
8) Link the head node's next to the second node.
9) To create a third node allocate memory for third node and store data in third node.
10) Link the second node's next to the third node and set third node's next to NULL
11) Stop

Node structure

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

Memory allocation to head

```
struct Node* head = (struct Node*) malloc
                    (sizeof(struct Node));
```

Data storage in head

```
head → data = 10;
```

• Inserting a new node in a Linked List

• There are three ways to insert a new node in a Linked List.

1. Insert at the beginning
2. Insert at the end
3. Insert at the middle

1. Insert at the beginning

Algorithm

i) Start
ii) Allocate memory for new node
iii) Store data in new node
iv) Make a next of new node to point to head.
v) Make a head to point to newly inserted node.
vi) Stop

Piece of code to insert node at the beginning

```
struct node *newnode;
newnode = malloc(sizeof(struct node));
newnode->data = 4;
newnode->next = head;
head = newnode;
```

2. Insert at the end

Algorithm

i) Start

ii) Allocate memory for new node

iii) Store data in new node.

iv) Traverse the list to the last node

v) Make next of last node to newly inserted node

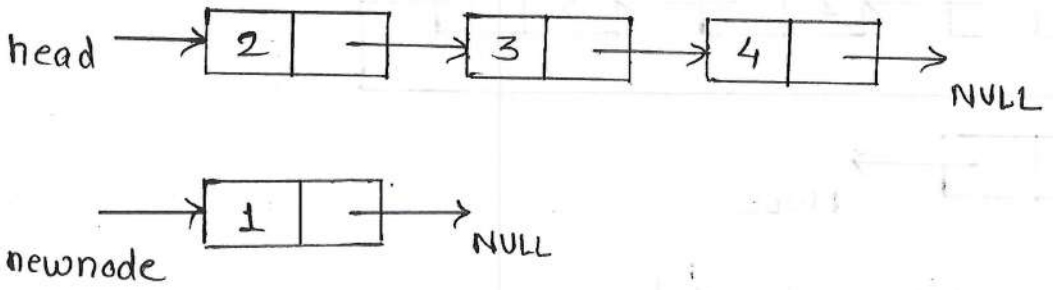
vi) Stop

Piece of code to insert node at the end

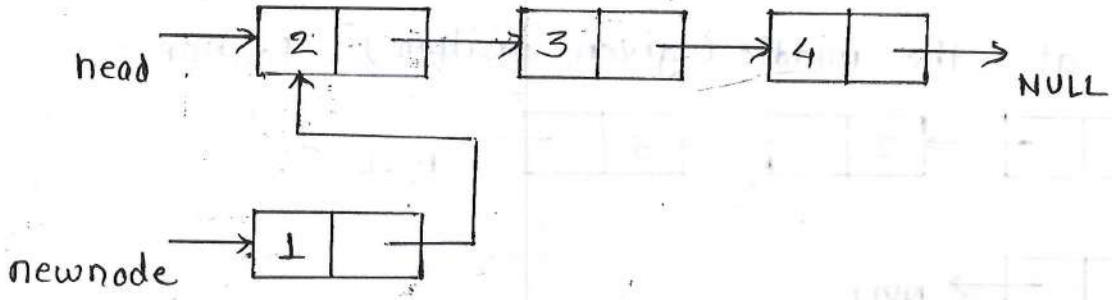
```
struct node *newnode;
newnode = malloc(sizeof(struct node));
newnode->data = 4;
newnode->next = NULL;
struct node *temp = head;
while(temp->next != NULL)
{
  temp = temp->next;
}
temp->next = newnode;
```

Examples

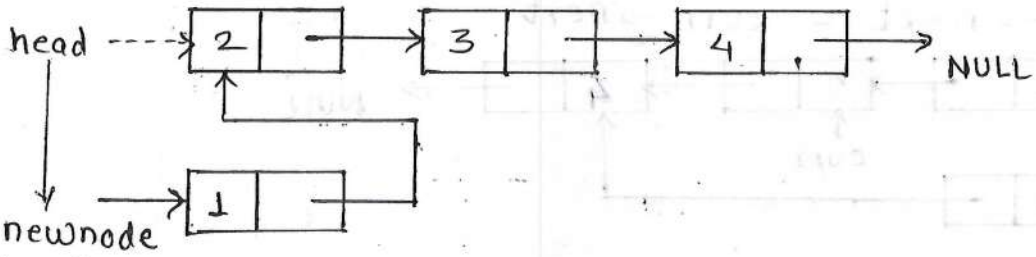
1. Insert at the beginning



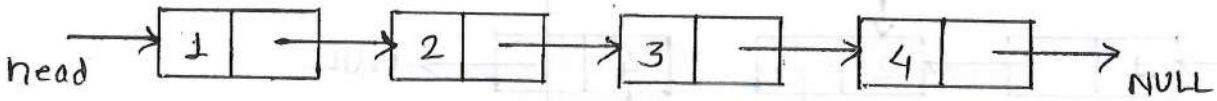
Assign newnode → next = head



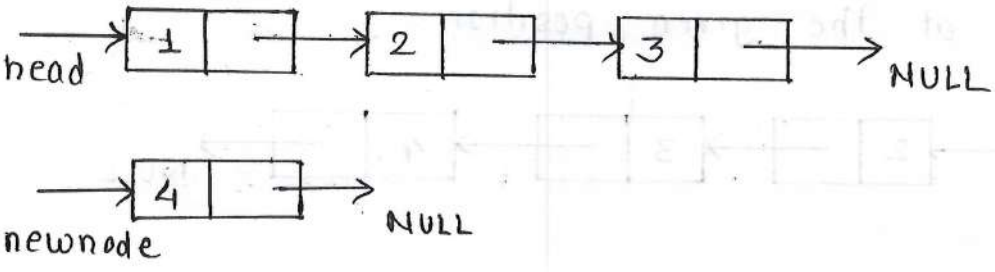
Assign head = newnode (newhead)



After insertion

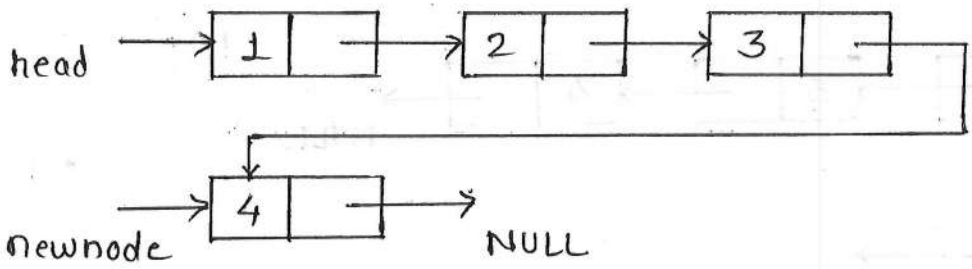


2. Insert at the end

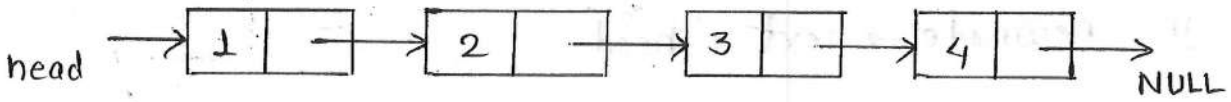


Traverse till the end of Linked list and assign

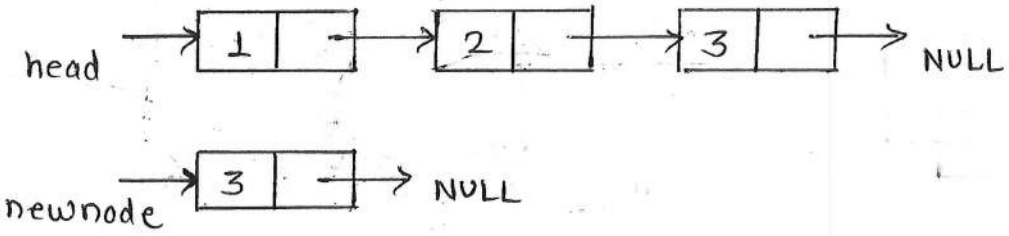
lastnode → next = newnode



After insertion at end

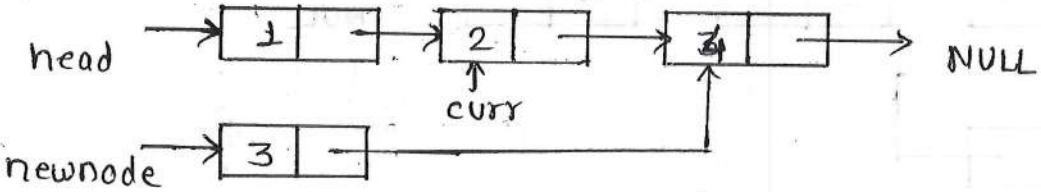


3. Insert at the middle (given position) Position = 3

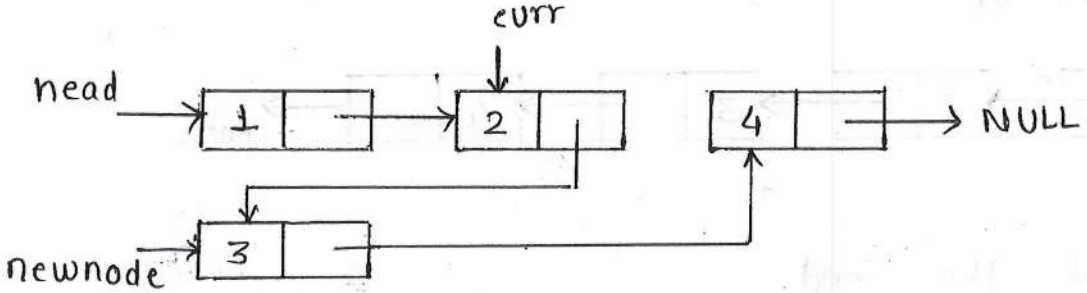


Traverse till (position - 1) node and assign

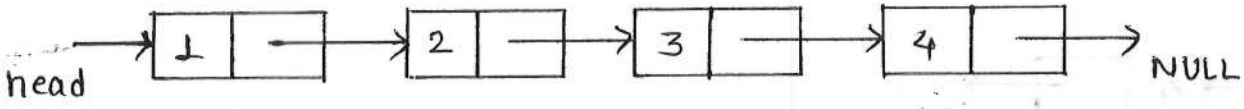
newnode → next = curr → next



Assign curr → next = newnode



After insertion at the given position



3. Insert at the Middle

Algorithm

i) Start

ii) Allocate memory for new node.

iii) Store data in new node

iv) Traverse the list to just before the given position of new node

v) Change next pointers to insert new node in betwee

vi) Stop

Piece of code to insert node at the middle

```
struct node *newnode;
newnode = malloc(sizeof(struct node));
newnode -> data = 4;
struct node *temp = head;
For( int i = 2; i< position ; i++)
{
  if( temp -> next != NULL)
    {
      temp = temp -> next;
    }
}
newnode -> next = temp -> next;
temp -> next = newnode;
```

- Deleting a node from a Linked List
  - There are three ways to delete a node from Linked List

1. Delete from beginning
2. Delete from end
3. Delete from middle

1. Delete from beginning

Algorithm

i) Start

ii) Check if head is NULL if it is true it returns NULL. If it is false then store the current head in a temporary pointer

iii) Move the head pointer to the next node

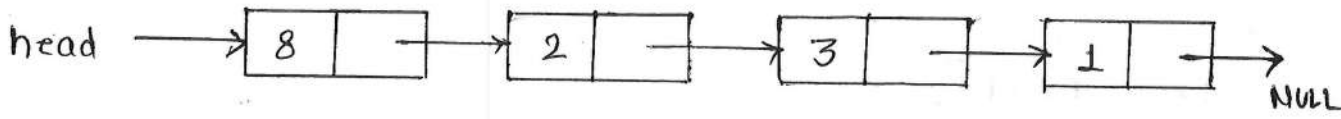
iv) Free the memory allocated for the old head node

v) Stop

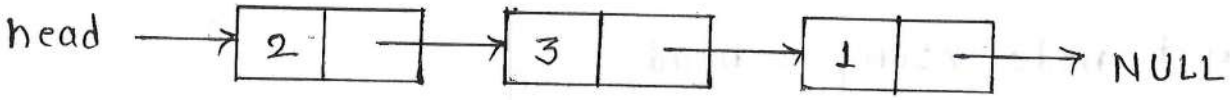
Piece of code to delete node from the beginning

```
if (head == NULL)
{
  printf("List is empty, deletion not possible.\n");
  return NULL;
}
struct node *temp = head;
 head = head -> next;
 free(temp);     // free memory of old head
```

Example



After deleting from beginning



2. ~~Detefro~~

2. Delete from end

Algorithm

i) Start

ii) Case 1 : The list is already empty check if head is NULL then cannot delete node

iii) Case 2: The list has only one node. check if head. next is NULL then free head.

iv) Case 3 : The list has multiple nodes then initialize temp = head and previous = NULL. Traverse the list until temp reaches the last node. Store the second last node in one variable and last node in temp variable. Unlink the last node and free last node memory

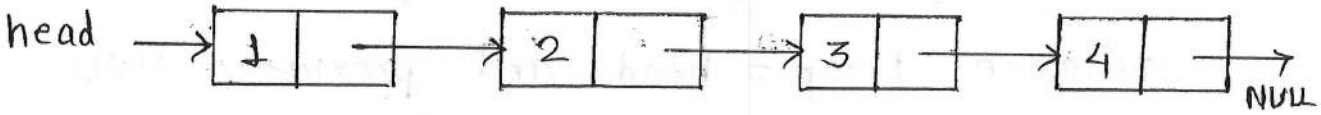
v) Stop

Piece of code to delete node from the end

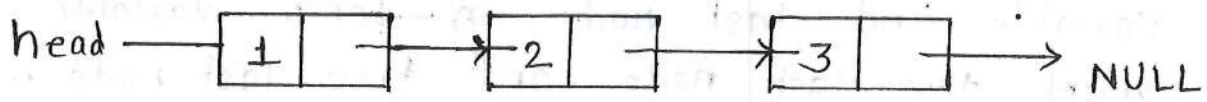
```
if (head == NULL)
{ printf("List is empty, deletion not possible");
  return NULL;
}
```

```
if( head→next == NULL)
{
  free(head);
  return NULL;
}
struct node *temp = head;
struct node * previous = NULL;
while( temp→next != NULL)
 {
  previous = temp;
  temp = temp→next;
 }
  previous→next = NULL;
  free(temp);
  return head;
```

Example



After deleting from end



3. Delete from middle

Algorithm

i) start

ii) Case 1 : The list is already empty check if head is NULL then cannot delete node.

iii) Case 2 : The list has only one node. Check if head next is NULL then free head

iv) Case 3 : The list has multiple nodes then traverse the element before the element to be deleted. Then change next pointers to delete the node from the list.

v) Stop

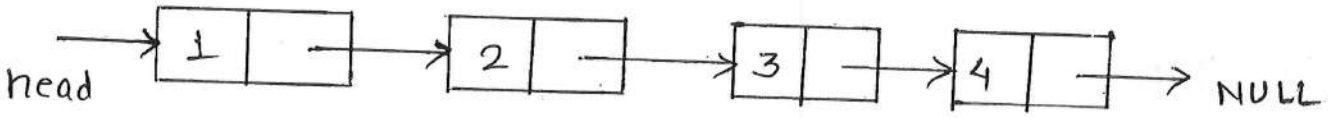
Piece of code to delete node from given position

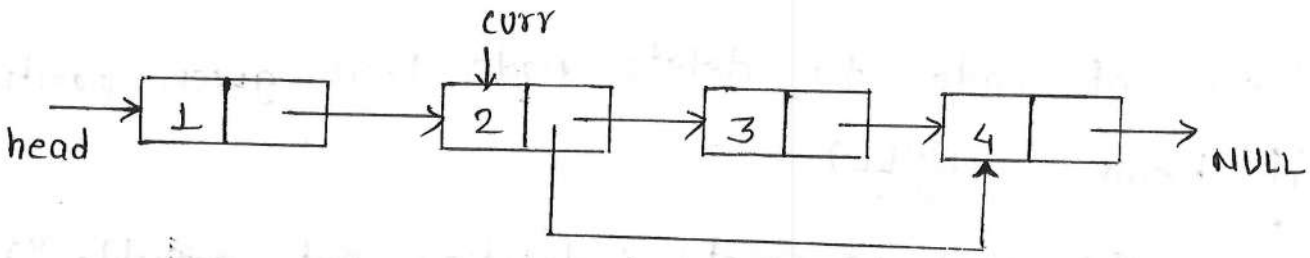
```
if (head == NULL)
 {
   printf("List is empty, deletion not possible");
   return NULL;
 }
if( head → next == NULL)
{
  free(head);
  return NULL;
}
struct node *temp = head;
for(int i=0; temp!= NULL && positi
```

```
for(int i=0; temp!=NULL && i< position-1; i++)
{
 temp= temp->next;
}
struct node* *node_delete = temp->next;
temp->next = node_delete->next;
free(node_delete);
return head;
```

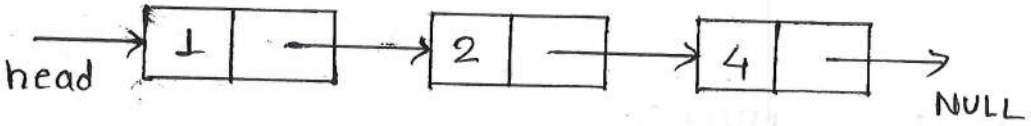
Example

Position = 3





After deletion



- Searching a key in Linked List

Algorithm

i) Start

ii) Initialize head = NULL

iii) Insert a nodes to the Linked list

iv) Take input to search the item

v) Initialize a node pointer temp = head

vi) Traverse the list while temp is NOT NULL.

vii) For each node check if its data equals to item wants to search.

viii) If match is found then return true or return index of node where item was found

ix) If match is not found move the pointer to the next node (temp = temp → next).

x) If the traverse ends and temp becomes NULL as well as ~~key~~ item is not found then return false or -1.

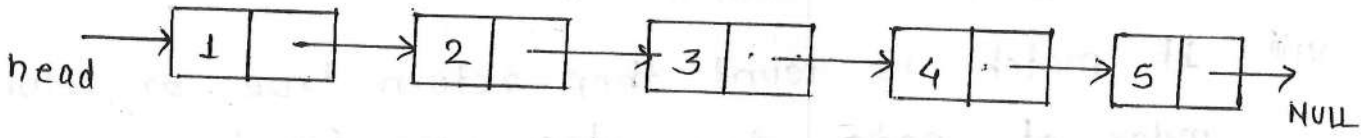
xi) Stop

Piece of code to search a key in Linked List

```
bool searchKey( struct node *head , int key)
{
  struct node *temp = head ;
  while( temp != NULL)
  {
     if( temp → data == key)
        return true ;
     temp = temp → next ;
  }
  return false ;
}
```

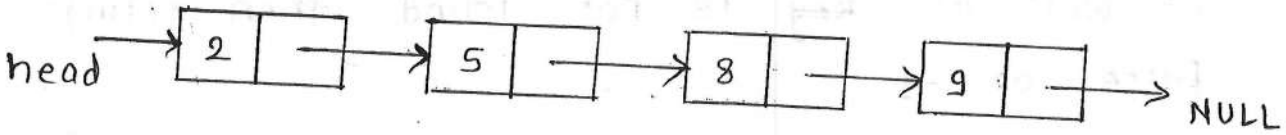
Example

1) Given head and key = 5, determine whether the key exists by traversing through the nodes sequentially.



Output : true

2) Given head and key = 27



Output : False

- Traversing a Singly Linked List

Traversing is a process of visiting each node exactly once, starting from the head and ending at the last node. It is perform with the help of next pointers till lastnode → next becomes NULL. Traversing involves printing the value of each node.

Algorithm

i) Start

ii) Check if list is empty if it is true then it will not print anything.

iii) Initialize temporary pointer with head like
temp = head

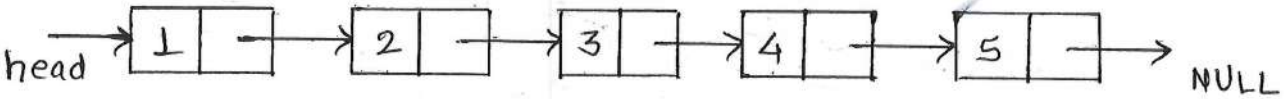
iv) Traverse the list until temp is not equal to NULL and print each node i.e. temp→data
Then move to the next node again print then move to next till end of list i.e. while temp != NULL

v) Stop

Piece of code to traverse a list in singly Linked List

```
if(head == NULL)
  printf(" List is empty ");
  return;
struct node *temp = head;
printf(" List elements are \n");
while( temp != NULL)
{
  printf(" %d --> ", temp→data);
  temp = temp→next;
}
```

Example



Output : 1→2→3→4→5

3.5 Applications of Linked List

i) Used as a building block for many other data structures, such as stacks, queues, trees, graphs etc

ii) Used as a polynomial manipulation

iii) Used to represent other data structure

iv) Used as a dynamic memory management.

v) Utilize a doubly linked list for browsers like chrome or firefox to manage page history. The "Back" and "Forward" buttons simply traverse backward and forward through node pointers

vi) Media Player Playlists move immediately to the next track, skip the previous track or append new tracks to the queue

vii) Task scheduling, process management

viii) Undo/Redo represent actions in applications

ix) Image viewers use a linked sequence to cycle through images when you click the left or right navigation arrows.

x) File & Directory Management

• Advantages of linked list over arrays

i) Linked list allow efficient insertion and deletion operations without shifting elements while arrays requires element shifting.

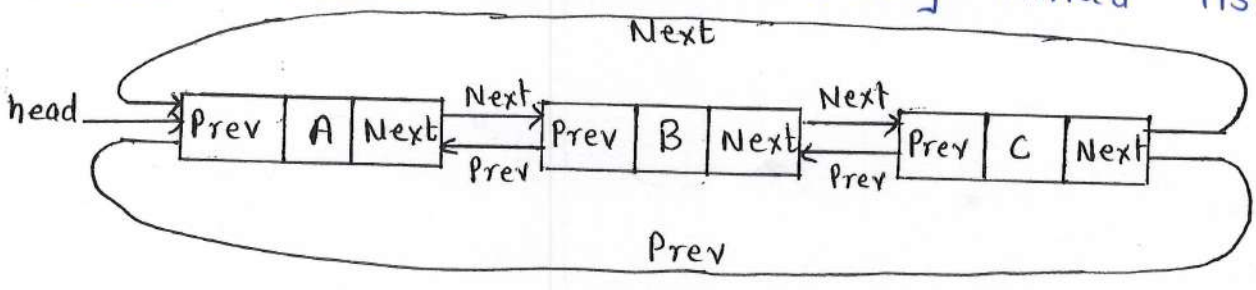
ii) Linked Lists can grow or shrink during runtime while array is fixed after creation.

iii) Linked Lists provide efficient insertion and deletion at both ends making the implementation of Queues and Deques simple and straightforward.

iv) No memory wasted in Linked List because memory is allocated exactly when a node is added while in array memory wastage is done because memory allocation is done at compile time.

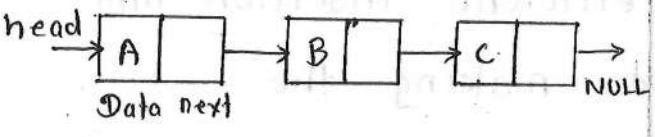
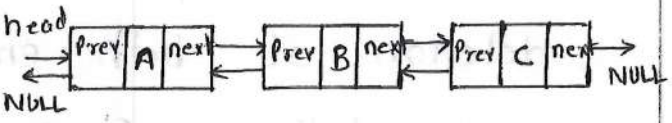
v) When data is added to an linked list memory addresses remain stable while in an array entire memory structure can be reallocated.

• Draw structure of Circular doubly linked list



Circular doubly linked list

· Differentiate between singly linked list and ~~doubly~~ doubly linked list.

	Singly linked list	Doubly linked list
1.	SLL nodes contains 2 field- data field and next link field	DLL nodes contains 3 fields- data field, a previous link field and a next link field.
2.		
3.	Traversal only in forward direction	Traversal in both directions forward and backward.
4.	Requires less memory than DLL as it has only two fields	Requires more memory than SLL as it has 3 fields.
5.	SLL is less used ~~in~~ due to limited number of operations.	DLL is more used due to wider number of operations.
6.	Used in stacks, queues or when memory is strictly limited.	Used in Undo/Redo functionality or building complex data structures

• Advantages of circular linked list over linear linked list

1) Circular linked list provides continuous traversal while linear linked list stop at one node.

2) No NULL pointers for circular linked list i.e no need to check for NULL at the end of list while linear linked has NULL pointer

3) Can traverse from any node to any node in circular linked list which was not possible in a singly linked list if we reached the last node.

4) Easily can go to head from the last node.

5) In a circular list, any node can be starting point means we can traverse each node from any point - which is not possible in singly linked list.

6) Uses memory efficiently with no extra pointer needed for the list end - while singly linked list requires a memory for NULL pointer.

Bodg 8/6/26

8/6/26.

Question Bank

Subject : Data Structure using C Marks : 16

Subject code : 313301

1) Differentiate between Static and Dynamic Memory location. (Minimum two points)

S-26, W-25 Marks - 02

2) State the syntax of declaration of doubly linked list.

W-24 Marks - 02

3) List advantages of linked list over arrays.

S-26 Marks - 02

4) Describe the concept of linked list with the terminologies : node, next pointer, null pointer and empty list.

W-24, S-25, W-25 Marks - 04

5) List any four applications of linked list.

W-24, Marks - 04

6) Write an algorithm to insert an element at the beginning of singly linked list.

S-25, W-25, S-26, Marks - 04

7) Create a singly linked list using data fields :- 15, 30, 90, 60, 75. Search a node 60 from singly linked list and show procedure step-by-step with the help of dia. start to end

W-24, W-25, Marks - 06

8) Draw structure to following :

i) Singly linked list ii) Doubly linked list

iii) Circular singly linked list

iv) Circular doubly linked list

W-24, S-26, Marks - 04

9) Differentiate between singly linked list and doubly linked list. (minimum six points)

W-24, S-25, Marks - 06

10) Describe circular linked list with suitable diagram. Also state the advantages of circular linked list over linear linked list.

W-25, S-25, Marks - 06

11) Write a C program to implement singly linked list with operation

i) insert at end ii) Display the list

S-25, Marks - 04

12) Explain different operations on singly linked list.

W-25, Marks - 04

13) Write C program to delete a node from linked list from last position

S-26, Marks - 06

14) Write a C program for inserting a new node at random position in singly linked list.

S-26, Marks - 06

15) Write an algorithm for searching a node in linked list.

8/6/26

8/6/26