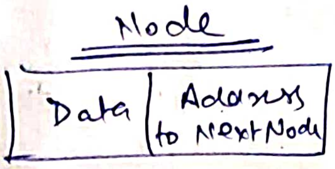
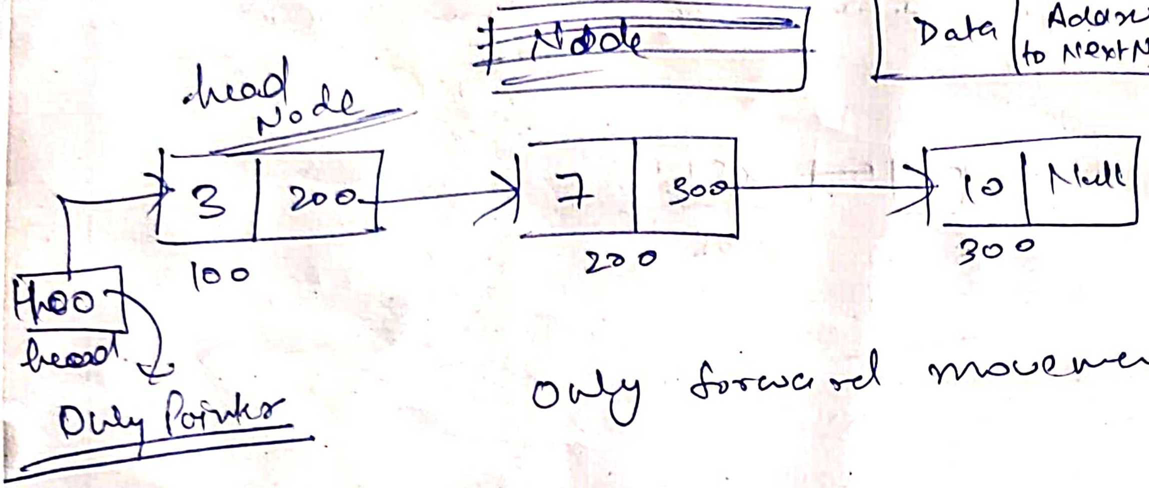
Types of linked list

Singly linked list





Only forward movement is possible

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

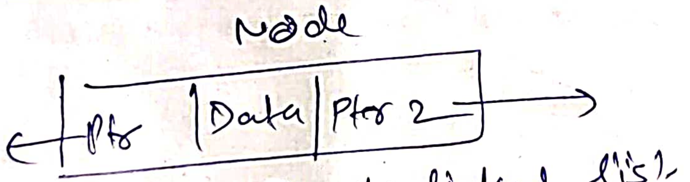
Doubly linked list

each node will have

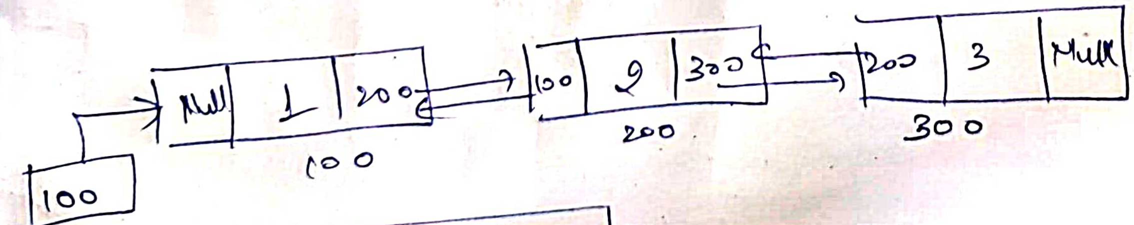
data Part

Address of Previous Node

Address of Next Node



of Doubly linked list



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

Representation of Doubly linked list in C

S14 10:05-11:00

001	12	41	353
002	13	44	355
003	14	45	[illegible]
04	18	47	[illegible]
005	22	48	Vibhor
006	24	49	Akhil
007	25	50	Bham
009	36		Pratap
010			Abhisheek
			Sumit
			Aryan

05, 38, 36

42, 43, 35

025, 036

021, 1016

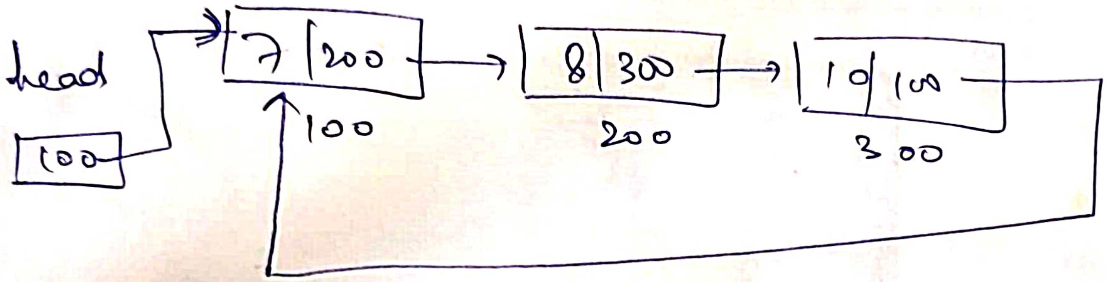
020, 023

06

2

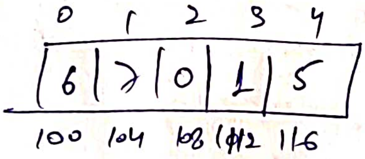
③ Circular linked list

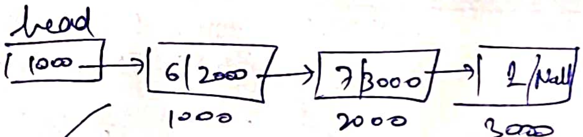
is only single linked list only with one difference that the last node's Address part will point back to ~~head~~ first node.



Array Vs linked list

① Cost of Accessing an element



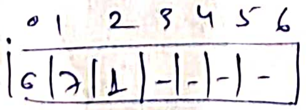


$a[2] = 100 + 4 \times 2$
$= 108$

taking O(1) time

Since only sequential access is possible & unlike array contiguous allocation is not there to access a particular element it will take $O(\frac{n}{2})$ time $= O(n)$ [depends on no. of nodes in linked list]

② Memory Requirement & Memory utilization



~~total size required~~
total memory allotted
= 7×4 = 28 bytes.

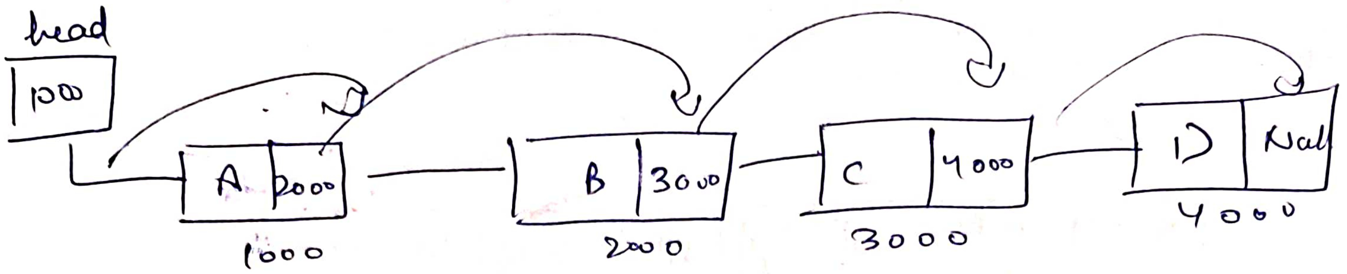
~~for~~ upto 7 data it will still require = 28 bytes

data + Address
Ex (4+4) = 8 bytes for one node
⇒ total size = 8×3 = 24 bytes.
but in case of 7 nodes linked list would be = 8×7 = 56 bytes.

Hence Memory requirement is less in Array but Memory utilization is efficient in linked list.

Traversal in linked list

the process of visiting each node of the list once is Called traversig.

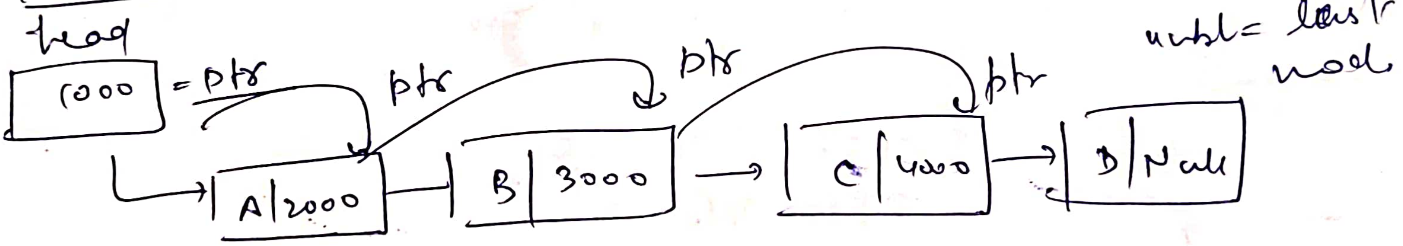


to traverse the link list we need to have one pointer before our first node, coz we can't use head pointer for traversal.

$\boxed{\text{head} \leftarrow \text{head} \rightarrow \text{next}}$ will lose !

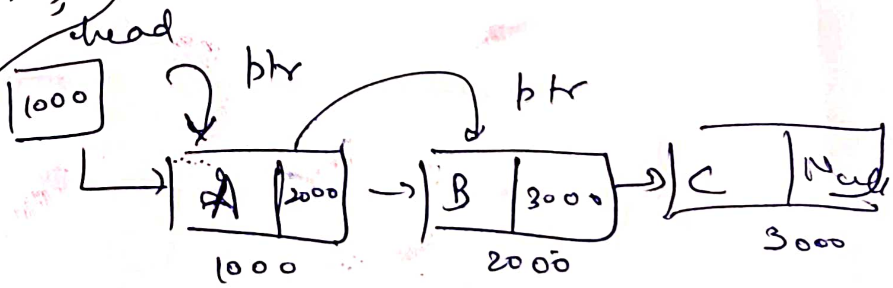
$\boxed{\text{head} = 2000}$ than first node will be lost.

$\boxed{\text{struct node *ptr}}$



```
traverse(struct node* head)
{ if (head==null)
  printf("linked list is empty");
  struct node * ptr;
     ptr=head;  (ptr=1000)
  while (ptr!=null)
   { ptr=ptr→next;

   }
}
```



malloc( ) in Programming

Syntax of malloc()

→ void * malloc(sizeof ( ));

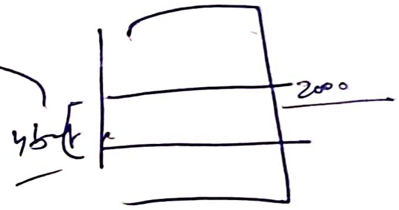
<stdlib.h>

return void Pointer.

malloc(2)

block of 2 bytes is available.

```
~~include~~
#include <stdio.h>
# include <stdlib.h>
  void main( )
  int *p;
  p=(int*) malloc (sizeof(int));
  *p=100
  p = (int*) malloc ( 10* size of (int));
```

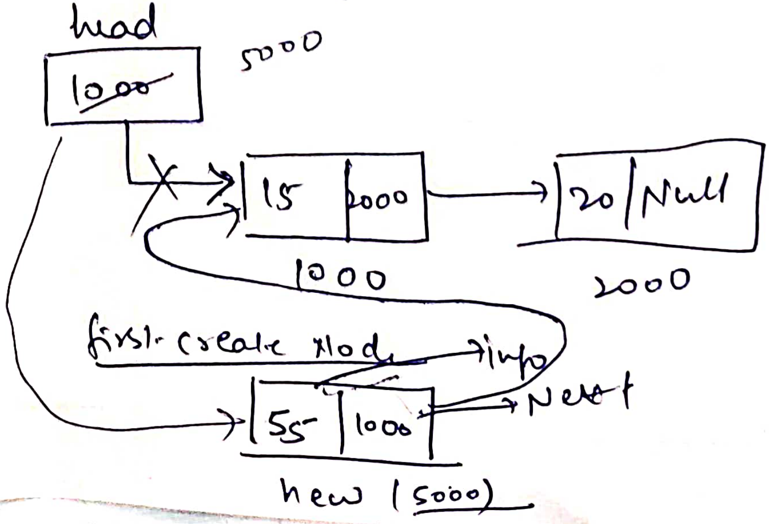


Inserting a node at the beginning

→ Pointing to the first node of link list.

```
insertbeg (struct node* head, int info)
{
 struct node *new;
 new = (struct node *) malloc ( size of (struct node)
  new → data = info ;
  new → next = head;
  head = new;
  return head;
}.
```

→ info of new node to be inserted.

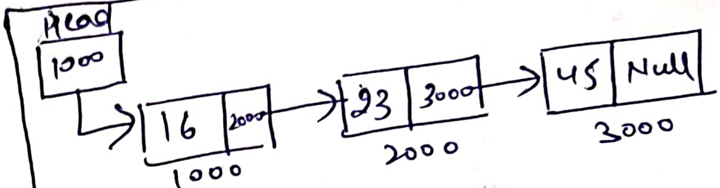


Inserting a node at the end of linked list

```
insert_end (struct node * head, int info)
{
 struct nod *ptr, *new;
 new = (struct node *) malloc ( sizeof (struct node));
  new -> data = info;
  new -> next = null;
  ptr = head;
  if (ptr ! = null)
  {
   while (ptr -> next ! = null)
        ptr = ptr -> next;
        ptr -> next = new;
   }
  else
    head = new;
  return head;
}
```

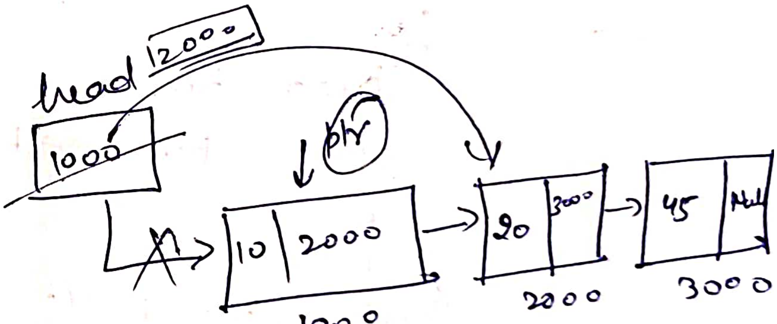
Inserting after a specific node in a linked list

```
insert_after (struct node * head, int x, int info)
 { struct node *ptr, * new;
  new = (struct node *) malloc (size of( struct node));
  new → data = info;
  ptr = head;
  while ( ptr → data ! = x && ptr ! = null)
   {  ptr == ptr → next;
   }
  if(ptr → data = x)
   { new → next = ptr → next;
     ptr → next = new;
   }
  return head;
 }
```



Deletion from the Beginning of the linked list

```
del_first (struct node * head)
 {
  struct node * ptr
  if (head == null)
     printf("list is empty");
  else
  {
     ptr = head;
     head = head → Next;
     free (ptr);
  }
  return head;
 }
```



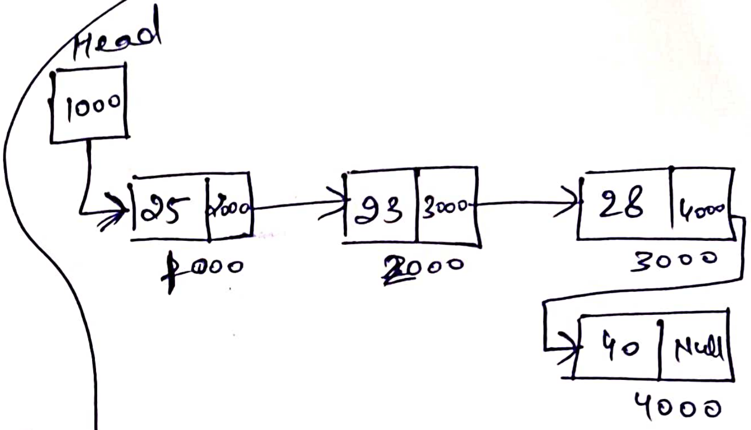
(7)

Deletion from the end of linked list

```
del_last ( struct node * head)
{
 struct node * ptr, * prep;
  If ( head==null)
     printf ("list is empty");
  elseif (head→next = null)
     {
      free(head);
      head = null;
     }
   else
     { ptr=head;
      while (ptr→next!=null()
        {
         prep = ptr;
         ptr= ptr→next;
        }
       prep →next = null;
      free(ptr);
     }
    return head;
 }
```

3 etion after a given node of the linked list

```
delete_after (struct node * head, int key)
{
  struct node * ptr1, * ptr2;
  ptr1 = head;
  while (ptr1 → next != null)
   {
     if(ptr1 → data == key)
       {
         ptr2 = ptr → next;
         ptr1 → next = ptr2 → next;
         free(ptr2);
         break;
       }
     ptr1 = ptr → next;
   }
  retur head;
}.
```


Head
1000
25
2000
1000
93
3000
2000
28
4000
3000
40
Null
4000