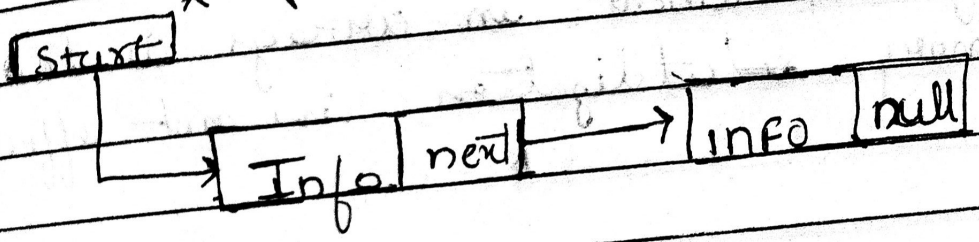
CHAPTER-6

Linked Lists :-

Linked lists are special type of some data elements linked to one another.

Linked list contains data elements in non consecutive manner unlike arrays.



nodes

- External pointers are use to traverse list.

Types of linked list :-

- Singly ll
- Doubly ll
- Circular ll
- Doubly circular ll

3d Applets

DATE
PAGE

★ Arrays v/s linked list

→ Cost of accessing an element :-

(I) We can access element randomly if we have base address O(1)

(II) We can access data randomly O(n)

→ Memory requirement and memory utilization :-

(I) Memory requirement in arrays is less. But memory utilization is not efficient.

(II) Memory requirement is more, but memory utilization is efficient.

→ Cost of insertion and deletion :-

(I) insertion at begginning - O(n)
at end - O(1) // cuz of shifting
at i^{th} position - O(n)

3rd Apr/23

DATE
PAGE

☆ Arrays v/s linked list

→ Cost of accessing an element :-

(I) We can access element randomly if we have base address O(1)

(II) We can access data randomly O(n)

→ Memory requirement and memory utilization :-

(I) Memory requirement in arrays is less. But memory utiligation is not efficient.

(II) Memory requirement is more, But memory utiligation is efficient

→ cost of insertion and deletion :-

(I) insertion at begginning - O(n)
at end - O(1) // cus of shifting
at i^{th} position - O(n)

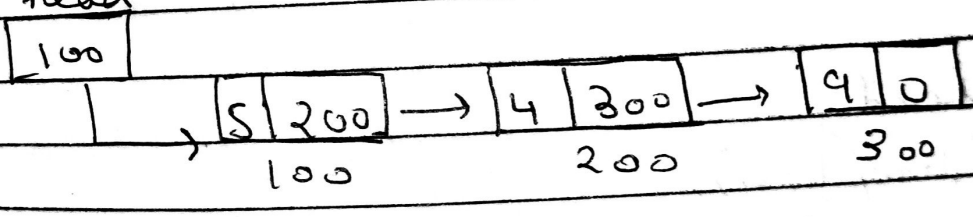
(ii) at beggining - O(1)
at End - O(n)
at i^{th} position - O(n)

The cost of deletion is same.

→ Easy to use → (A)

→ searching operation:
(A) linear and binary
(ii) only linear.

Implementation of linked list




```
Void main () {
    struct node
    { int data;
      struct node *next;
    };
    struct node *head, *newnode, *temp;
    head = 0; int choice; while (choice) {
    newnode = (struct node *) malloc (sizeof (struct node));
```

//malloc return void pointer and allocates memory
So we typecasts it to get address of that block.

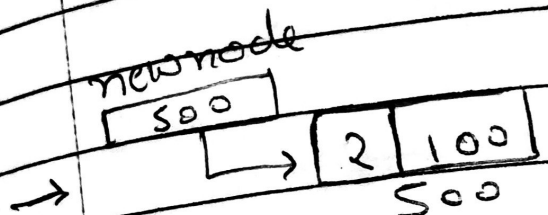
cout <<

```
cout<<"Enter data"<<endl;
cin>>newnode->data;

newnode->next=0;
if(head==0)
{head=temp=newnode;}
else
{
temp->next=newnode;
temp=newnode;
}
cout<<"Do you want to cont."<<endl;
cin>>choice;
}
//Display
temp=head
while(temp!=0)
  {cout<<temp->data<<endl;
    temp=temp->next;
  }
}
// Three pointers are required *temp for
storing address of current node.
// We can create function to implement
this.
```

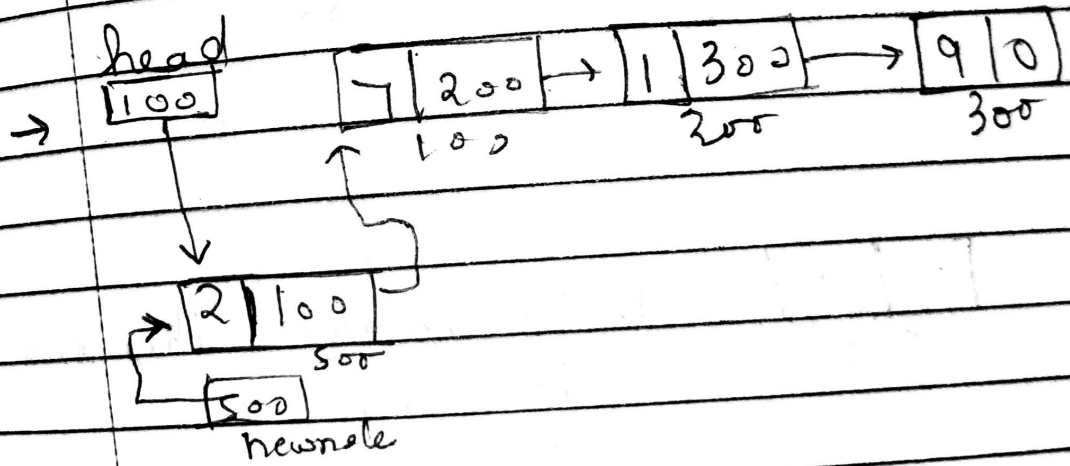
27/11/23

Insertion of node in linked list (at beginning and specific position)

(1) Insert at begin :-



// To insert this node at beginning

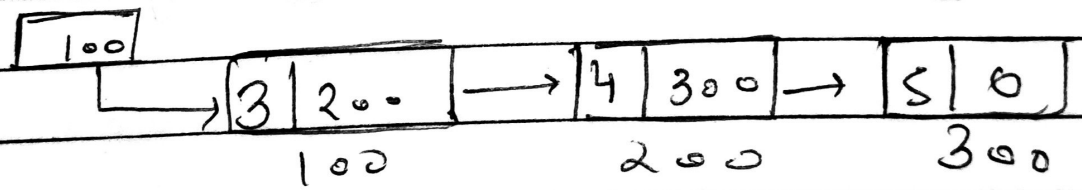


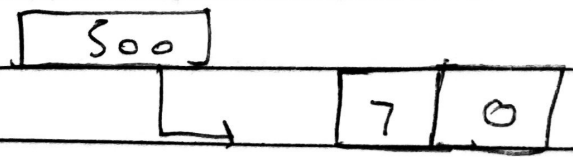
// code to insert at beginning.

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

```
struct node *head, *newnode;
newnode = new(struct node);
cout << "Enter data" << endl;
cin >> newnode -> data;          // entering data
// to place node at beginning
newnode -> next = head;
head = newnode;
}
```

(2) At the end

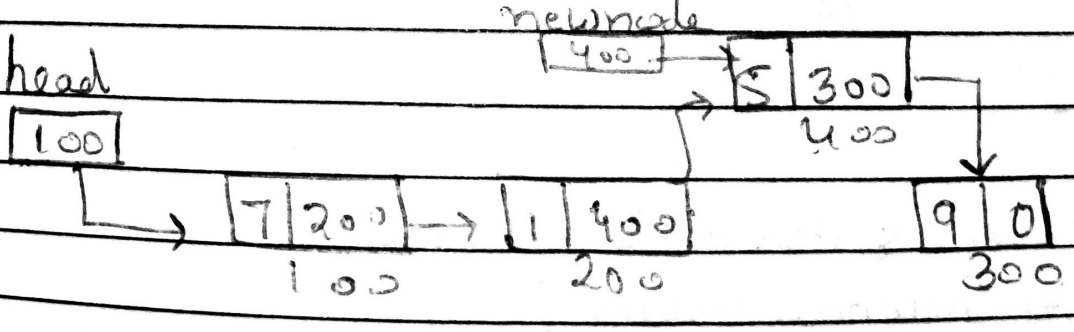




// To insert this e
at end.

```
struct node {
int data;
struct node * next;
};
struct node *head, *temp, *newnode;
head = 0;

newnode = new(struct node);
cin>> newnode -> data;      // entering data.
newnode -> next = 0;
// To place node at end.
temp = head;
while (temp != 0) (temp -> next != 0)
{ temp = temp -> next;
}
temp -> next = newnode;
temp = newnode;
```

(3) Insertion at Specific position :-

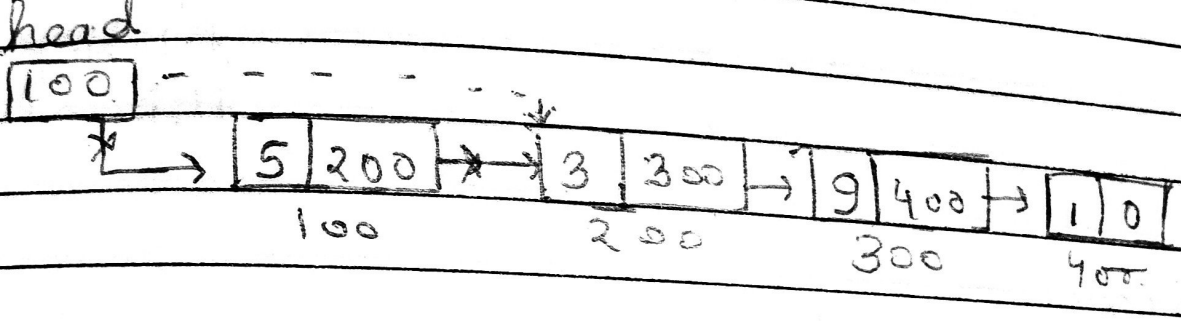


Code

```
int Pos, i=1;
struct node
{ int data;
  struct node *next;
};
struct node *head, *newnode, *temp;

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

Cout<< "Enter position" <<endl;
Cin>> Pos;
if(Pos > count) Cout<< "Invalid" <<endl;

// placing node
Else {
temp = head;
While (i<Pos) {
    temp = temp -> next;
        i++;
}

Cout << "Enter data" <<endl;
Cin >> newnode -> data;
newnode -> next = temp -> next;
temp -> next = newnode;
```

Deletion of a node from linked list (beggin, end, specific postion)

1) Delete from beginning :



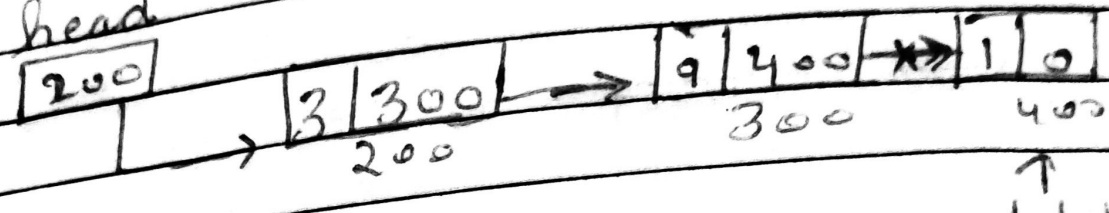
// code

```
Struct node {
    int data;
    Struct node *next;
};
Struct node *head;
```

→ DeleteFromBeg()

```
{ Struct node *temp;
    temp = head;
    head = head -> next;
    free(temp);
}.
```

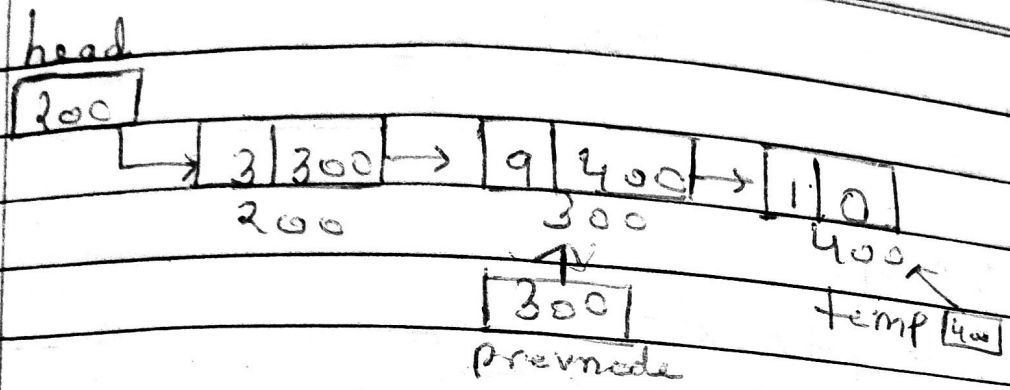
(2) Deletion from end :-



// Code
struct node {
int data;
struct node *next, *temp;
};

⇒ DeleteFromEnd() {
// to delete node we have to use two pointers.

Struct node *prevnode;
temp = node;
while (temp->next != 0)
{ prevnode = temp;
temp = temp->next;
}
if (temp == head) head = 0; // only one node is there
free temp;



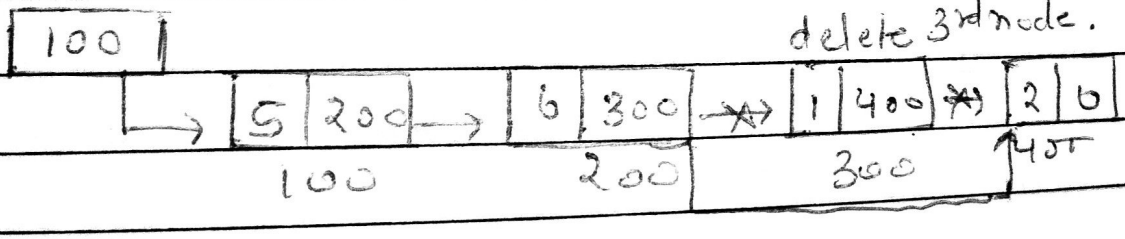
ELSE {

Prevnode → next = 0;

free (temp);

}

(3) Delete from specified position:-



// code

Struct node {

int data;

Struct node * next;

};

struct node *head, *temp;

⇒ DeleteFromPos() {

Struct node * nextnode;

```
int pos, i=1;          Cin>> Pos;
temp = head;

While (i < Pos-1)
{ temp = temp->next;
    i++;
}
nextnode = temp->next;
temp->next = newnode->next;

free(nextnode);
}
```