Unit - 2

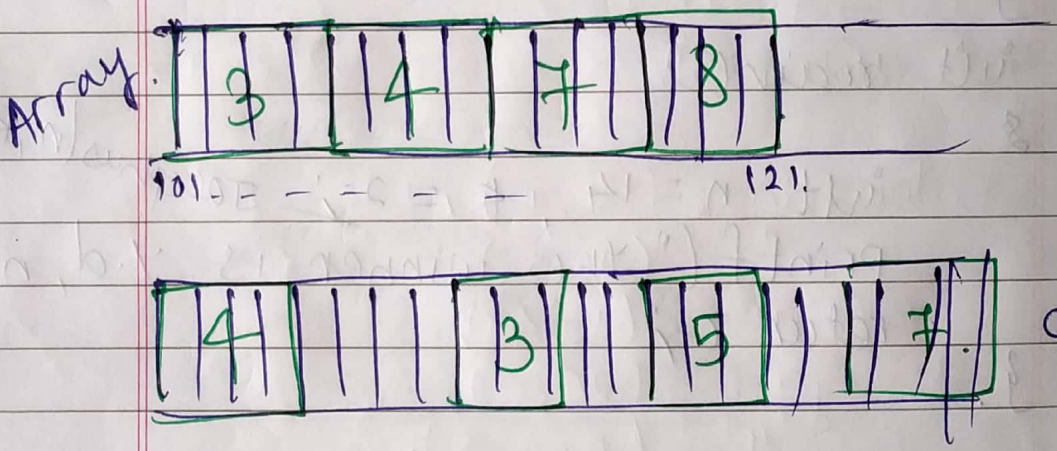
Linked List.

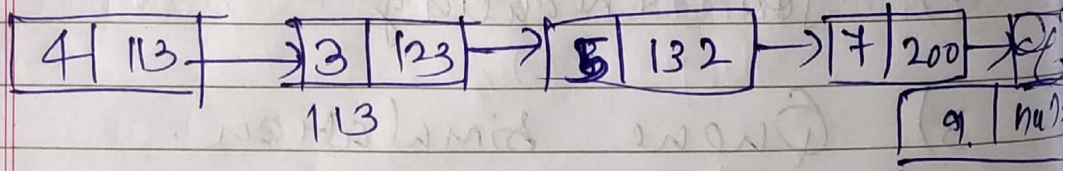
* Dynamic memory allocation.

SMA - at compile time
DMA - at run time.

* Concept of link organization.

```
int x;
int a[3];
```



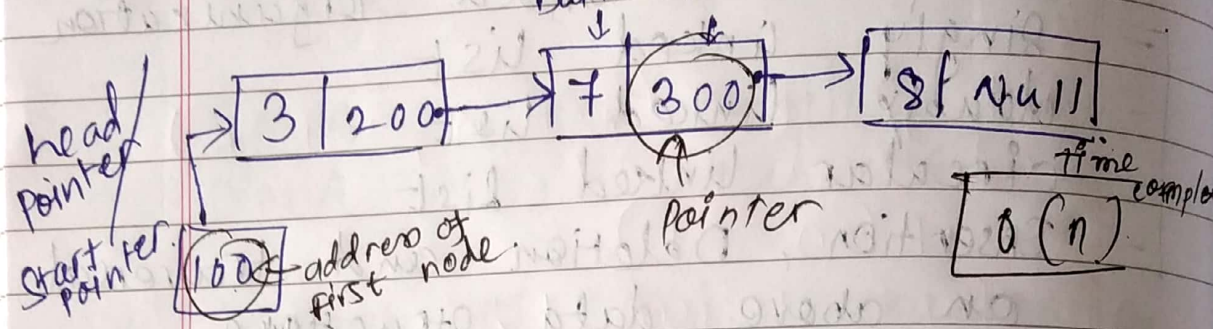


- Dynamic memory allocation
- concept of linked organization
- Singly linked list
- doubly linked list
- circular linked list
- Insertion, Deletion and traversal on above data structure.
- Representation and Manipulations of polynomials using linked lists.
- Implementation of Linked list for stacks and queues.
- Generalized linked list,
- operations on GLL like copy, Equality.

⇒ Types of linked list.

1) Singly linked list
2) Doubly linked list
3) circular linked list.

* Singly linked list.



- forward sequencial movement is possible.

* representation of node:

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

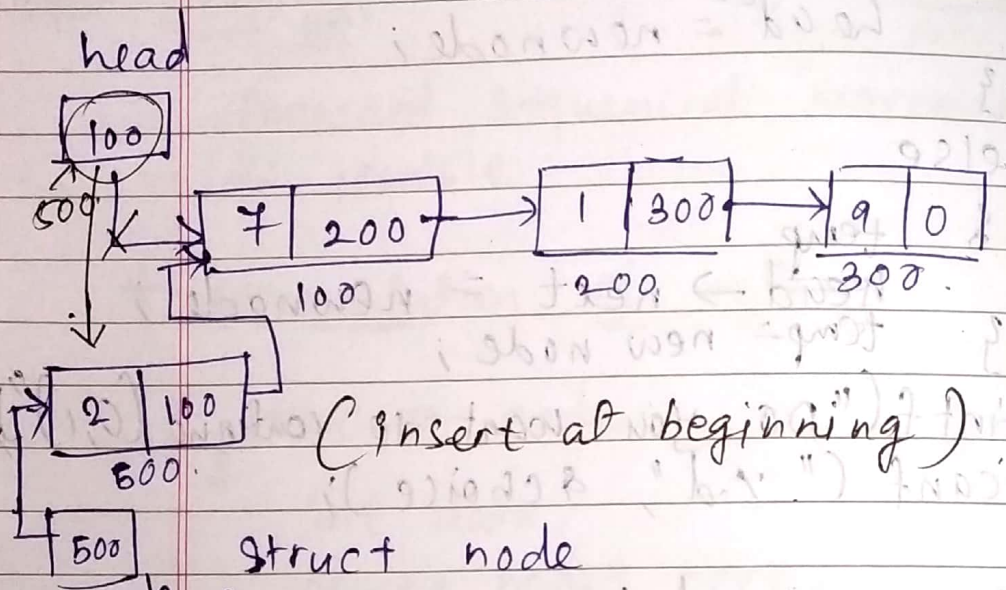
* Implementation of Singly linked list.

```
void main()
{ struct node.
  {
      int data;
      struct node * next;
  };
                          *temp;
struct node * head, *newnode;
 head = 0 ; int choice ; while (choice
{ newnode = (struct node*) malloc (size of
                          (struct node)).
```

```
printf("Enter Data");
scanf("%d", &newnode->data);
newnode->next = 0;
head = new node;
if(head==0)
{
    head = temp = newnode;
}
else
{   temp
    head->next = newnode;
}   temp = new node;
printf("Do you want to continue (0;1)");
scanf("%d", &choice);
}

temp = head;
while(temp != 0)
{
    printf("%d", temp->data);
    temp = temp->next;
}
getch();
}
```

* Insertion in linked list

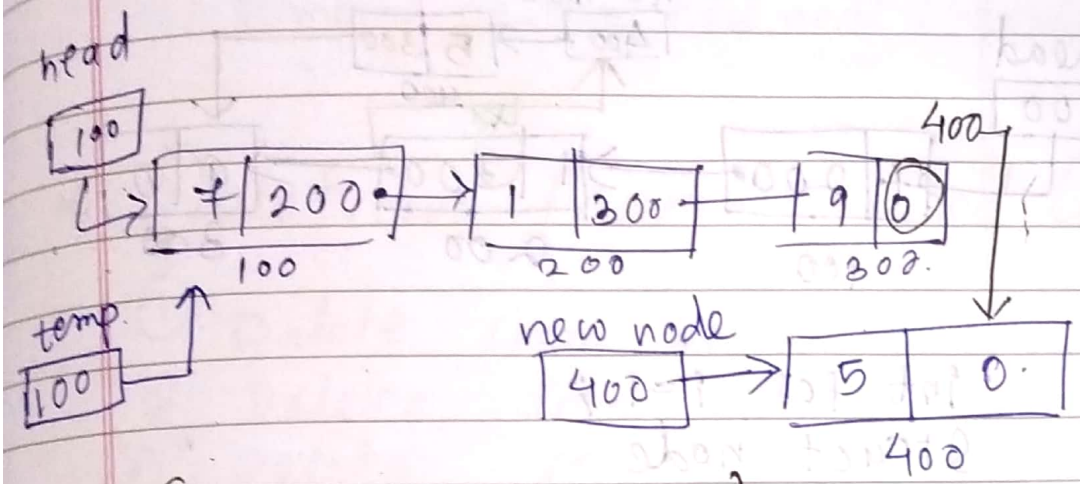
① insert at beginning
② insert at end
③ insert after a given location



(insert at beginning)

```
struct node
{
    int data;
    struct node *next;
};
struct node *head, *newnode;
newnode = (struct node *) malloc
          (sizeof (struct node));
printf("Enter data you want to
                    insert");
scanf("%d", &newnode->data);
newnode->next = head;
head = newnode;
```

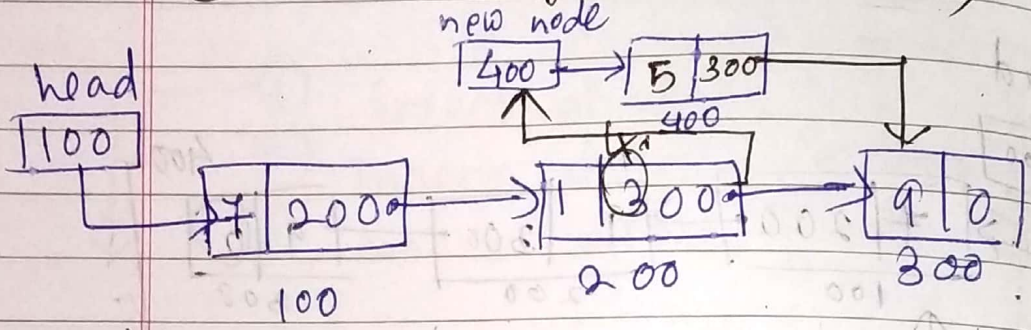
(insert at end)



(insert at end)

```
Struct node
{
    int data;
    struct node *next;
};
struct node *head, *newnode, *temp;
newnode = (struct node *) malloc
          (sizeof (struct node));
printf (" Enter data you want to
                    insert");
Scanf (" %d", &newnode -> data);
  newnode -> next = 0;
   temp = head;
while ( temp -> next != 0 )
{
    temp = temp -> next;
}
  temp -> next = newnode;
```

(insert after given location)

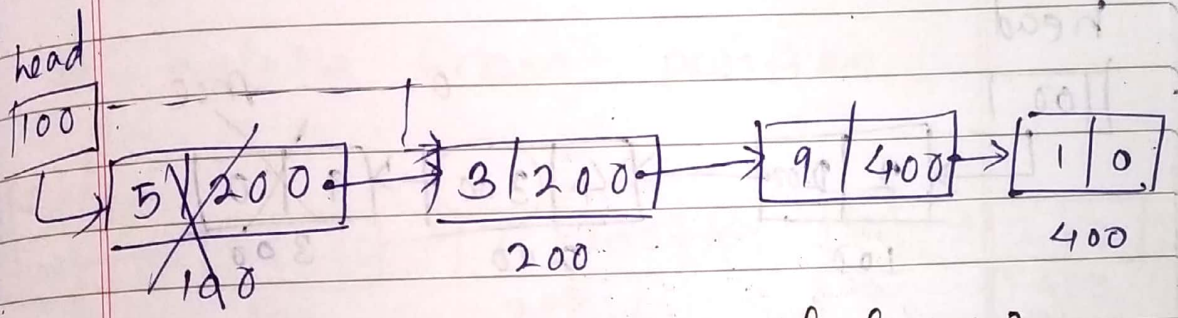



```
int pos, i=1;
Struct node
{   int data;
    Struct node *next;
};

Struct node *head, *new node, *temp
newnode = (Struct node) malloc (size of
          (Struct node))
printf ("Enter the position");
Scanf ("%d", &pos);
if (pos > Count)
{   printf ("Invalid position");
}
else
{   temp = head;
    while (i < pos)
    {
        temp = temp -> next;
        i++;
    }
    Printf ("Enter data");
    Scanf ("%d", & newnode -> data);
```

```
new node → next = temp → next;
temp → next = new node;
```

* Deletion from Linked List.

1. Delete from beginning.
2. Delete from end
3. Delete from specified position

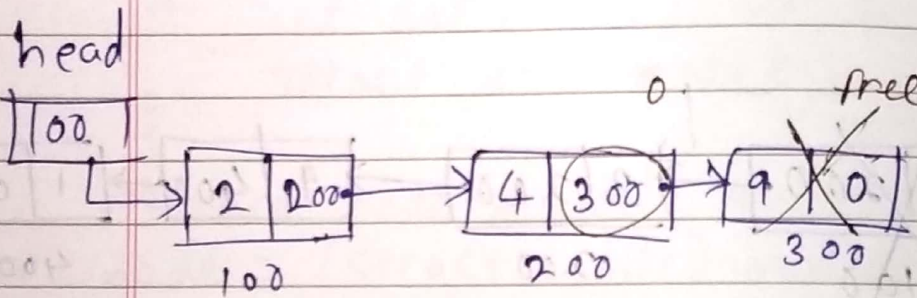



```
create list();
display();
Del_at_beg();
Del_at_end();
Del_at_pos();
```

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

```
Struct node *head;
Delete from Beg ( )
{
    struct node *temp;
    temp = head;
    head = head → next;
    free (temp);
}
```

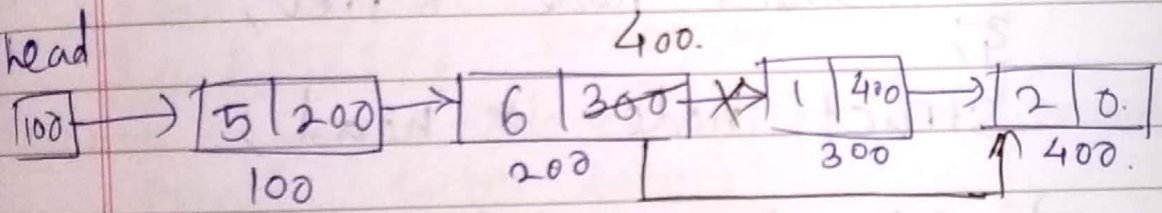
* Delete from end,




```
Struct node
{   int data;
    struct node *next;
};
Struct node *head, *temp;
Delete_from_end ( )
{
    Struct node *prev;
    temp = head;
    while (temp → next != 0)
    {   prev = temp;
        temp = temp → next;
    }
```

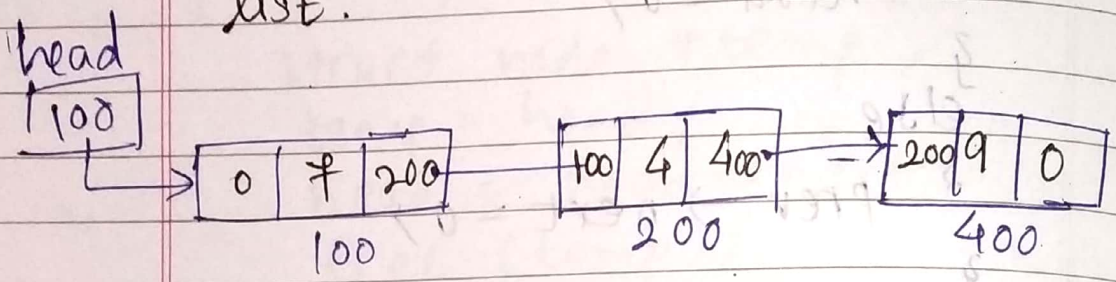
```
if (temp == head)
{   head = 0;
}
else
{   prev -> next = 0;
}
free (temp);
}
```

* Delete from position.




```
Struct node *head, *temp;
Delete Frompos ()
{
    struct node *nextnode;
    int pos, i = 1;
    temp = head;
    printf ("Enter pos");
    scanf ("%d", &pos)
    while (i < pos - 1)
    {   temp = temp -> next;
        i++;
    }
    next node = temp -> next;
    temp -> next = next node -> next;
}   free (nextnode);
```

* Introduction to Doubly linked list.




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

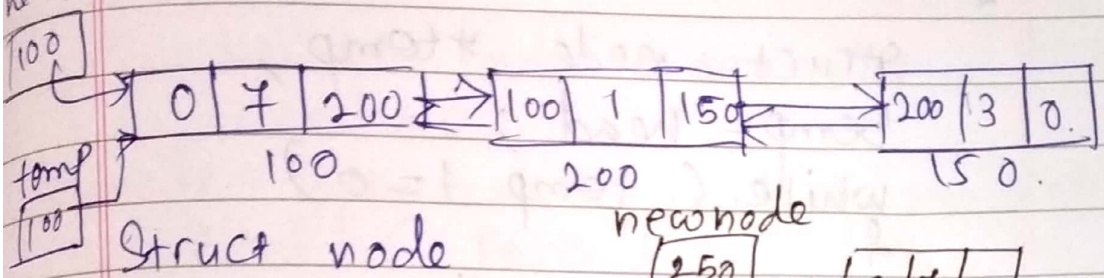
⇒ Advantages of Doubly

- You can traverse linked list in forward, Backward direction.
- insertion / Deletion is easy.

⇒ Disadvantages

- We are maitaining two pointers, so memory consumption is more.

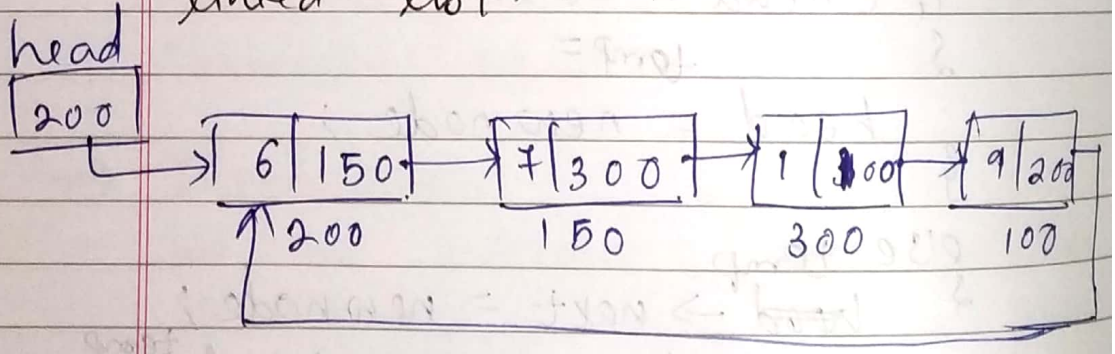
* Implementation Of Doubly Linked List




```
struct node
{   int data;
    struct node *next;
    struct node *prev;
};
struct node *head, *newnode

void create()
{   head=0;   struct node *temp;   int choice;
while(choice){
newnode = (struct node *) malloc (sizeof(struct node));
 printf(" Enter data );
 scanf("%d" newnode -> data );
 newnode -> prev = 0;
 newnode -> next = 0;
 if (head == 0)
 {          temp=
    head = newnode;
 }
 else temp
 {  head -> next = newnode;
    newnode -> prev = head;  temp
     temp = newnode;
 }
printf("Do you want to continue ");
scanf("%d", &choice);
}
```

```
void display()
{
    struct node *temp;
    temp = head;
    while (temp != 0)
    {
        printf("%d", temp->data);
        temp = temp->next;
    }

void main()
{
    create();
    display();
    getch();
}
```

* Implementation of circular linked list.




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

```
void createCLL()
{
    struct node *newnode, *temp;
    head = 0;
newnode = (struct node *) malloc (sizeof(struct node));
    printf("Enter data");
    scanf("%d", &newnode->data);
    newnode->next = 0;
    if(head == 0)
    {
        temp = head = newnode;
    }
    else
    {
        temp->next = newnode;
        temp = newnode;
    }
    temp->next = head;

void display()
{
    struct node *temp;
    if(head == 0)
    {
        printf("List is empty");
    }
    else
    {
        temp = head;
        while(temp->next != head)
        {
            printf("%d", temp->data);
            temp = temp->next;
        }
    printf("%d", temp->data);
```

Appli^{n} of Linked list.

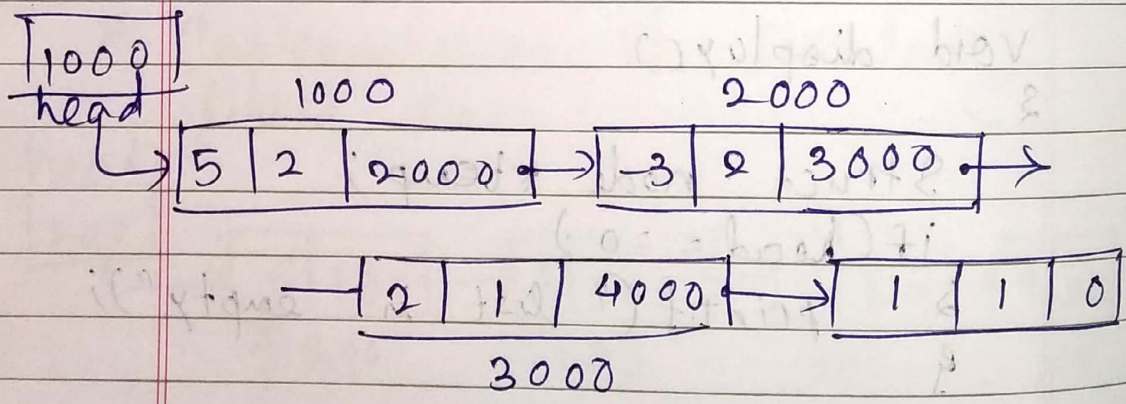
* Representation & manipulation of polynomial using linked list.
  ↑
  which has many terms.

4 term → $5x^2 - 3x^2 + 2x + 1$

Co-efficient, exponent

⇒ Representation using L.L.

```
struct node {
    float coefficient;
    int exponent;
    struct node *next;
};
```



Step 1: Creating main( )
Step 2: Enter number of terms.
Step 3: Enter coefficient & exponent.
Step 4: Insert new node in the LL

→ Polynomial should be sorted in descending order.

$5x + 3x^2 + 2x^3 + 1$

$2x^3 + 3x^2 + 5x + 1$.

```
int main()
{
    struct node *head=0;
    printf("Enter the polynomial");
    head = create(head);

struct node *create(struct node
                    *head)
{
    int n;
    int i;
    float coefficient;
    int exponent;

    printf("Enter the number of
                        terms");
    scanf("%d",&n);

    for(i=0; i<n; i++)
    {
        scanf("%d", &coeffi.);
        scanf("%d", &expo);
```

```
    head = insert(head, coff, exp);
}
struct node * insert (struct node
        *head int coe, int ex)
{
    struct node *temp;
    struct node * new =
      malloc(sizeof) struct node);

      newp → coeff = co;
      new p → expo = ex;
      new p → link = Null;

if(head == Null || ex > head → exp)
{
      newp → link = head;
      head = newp;
}
else
{     temp = head;
      while (temp → link != Null &&
          temp → link → expo > ex)

      temp = temp → link;
      new → link = temp → link;
      temp → link = newp;
}
  return head;
}
```

* Addition of two polynomials.

$3x^2 + 2x + 1$ — (1)

$5x^2 - x + 2$ — (2)

- addition of two polynomials involves combining like terms (same variables & same exponents)

⇒ Algorithm:

```
if(ptr1 → expo == ptr2 → expo)
{
    add the coefficient and
    insert the newly created node
    in the resultant RLL.
    and make ptr1 & ptr2 point
    to the next node.

else if(ptr1 → expo > ptr2 → expo)
{
    add ptr1 in the RLL. and
    ptr1 to next node.
}
else if(ptr1 → expo < ptr2 → expo)
{
    add ptr2 into LL and
    ptr2 point to the next node;
}
```

* Multiplication of two multipliers -

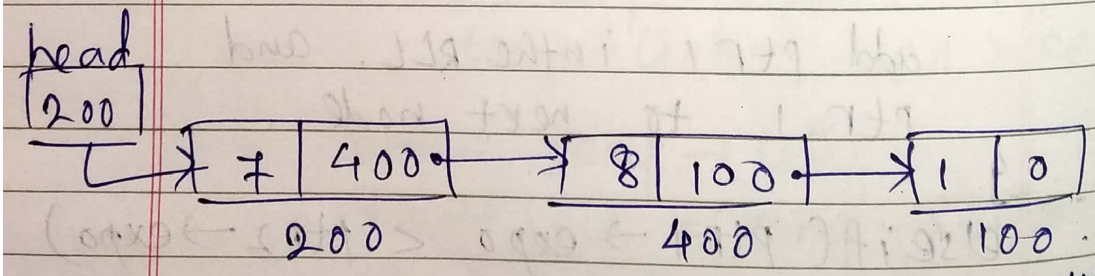
$2x^2 + 4x + 3$ — ①

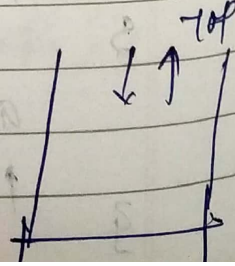
$3x^2 + 2x + 4$ — ②

$(2x^2 * 3x^2) + (2x^2 * 2x) + (2x^2 * 4)$
$+ (\ldots\ldots$

```
while (p1 != null)
{   while (p2 != null)
    {
        Res.coff = (p1.coff * p2.coeff) +
                   ---
        Res.exp = p1.exp + p2.exp.
        p2++;
    }
    p1++;
}
```

* Implementation Stack using LL.

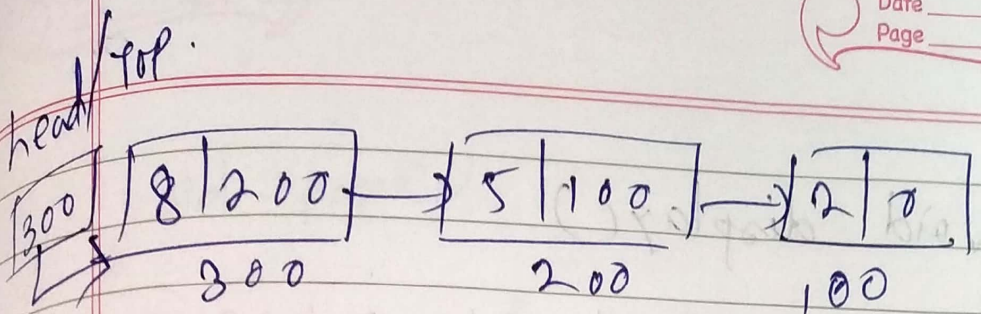




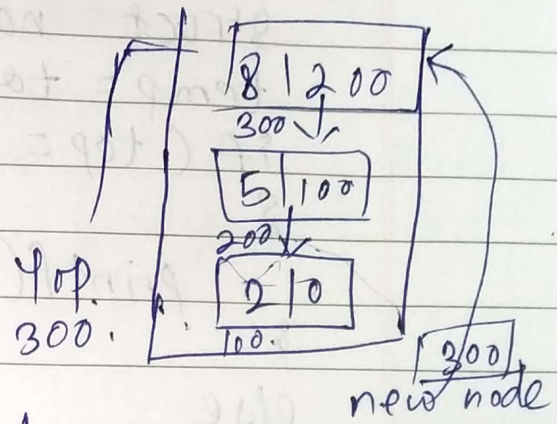
LIFO.

push

pop



push(2)
push(5)
push(8)



→

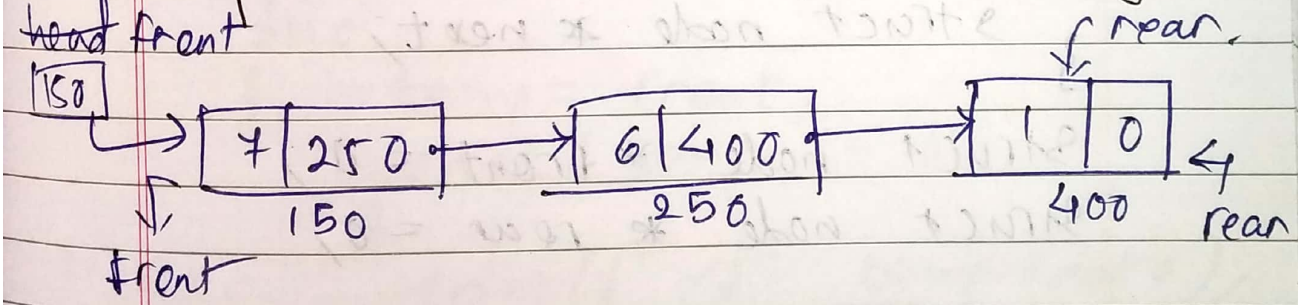
```
Struct node
{
  int data;
  struct node *link;
};

Struct node *top=0;

void push (int x)
{
    struct node *newnode;
newnode = (struct node *) malloc
                (sizeof(struct node));
  newnode -> data = x;
  newnode -> next = top;
   top = newnode;
}
```

```
Main()
{ push(2);
  push(3);
  push(10);
display();
peek();
pop();
peek();
display();
```

```
void display()
{
    struct node *temp;
    temp = top;
    if (top == 0)
    {
        printf("List is empty");
    }
    else
    {
        while(temp != 0)
        {
            printf("%d", temp->data);
            temp = temp->link;
        }
}
void peek()
{
    if(top == 0)
    {
        printf("Empty");
    }
    else
    {
        printf("top element is %d",
                                top->data);
    }
}
```

```
void pop()
{
    struct node *temp;
    temp = top;
    if (top == 0)
    {
        printf("Empty");
    }
    else
    {
        printf("%d", top->data);
        top = top->link;
    } free(temp);
```

* Implementation of Queue using LL

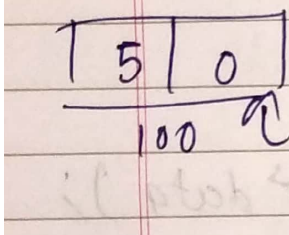


FIFO. ⇒ time complexity O(1)

- need to maintain two pointer.
  - head pointer (front)
  - tail pointer. (rear)

```
void main()
{
    enqueue(5);
    enqueue(0);
    enqueue(-3);
    display();
    dequeue();
    peek();
}
```

front ~~head~~ rear ~~tail~~




```
Struct node
{
    int data;
    struct node * next;
};
Struct node *front = 0;
Struct node * rear = 0;

void enqueue (int x)
{
    Struct node *new node;
new node = (Struct node *) malloc
                (sizeof (struct node));
    new node -> data = x;
    new node -> next = 0;
```

```
    if(front==0 && rear==0)
    {
        front=rear=newnode;
    }
    else
    {
        rear->next=newnode;
        rear=newnode;
}   }
void display()
{
    struct node *temp;
    if(f==0 && r==0)
    {
        empty;
    }
    else
    {   temp=front;
        while(temp!=0)
        {
            printf("%d", temp->data);
            temp=temp->next;
        }
    }

void dequeue()
{
    struct node *temp;
    temp=front;
```

```
    if(f==0 && R==0)
    {
        empty;
    }
    else
    { printf("%d" front->data);
      front = front->next;
      free(temp);
    }
}

void peek()
{
    if(f==0 && R==0)
    {
        empty;
    }
    else
    {
      printf("%d", front->data);
    }
```