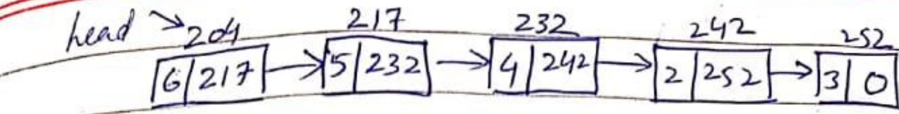
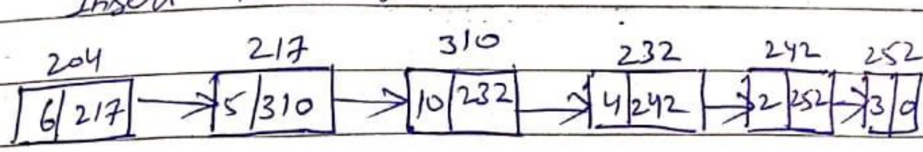
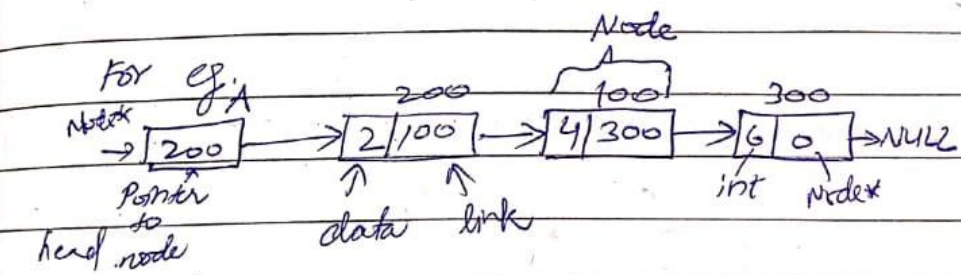
Linked List



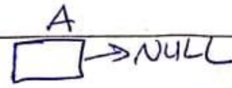

```
struct Node
{ int data;
  Node * next;
}
```

Insert 10 in list

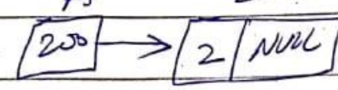





```
struct Node
{
  int data;
  Node * link;
};
```

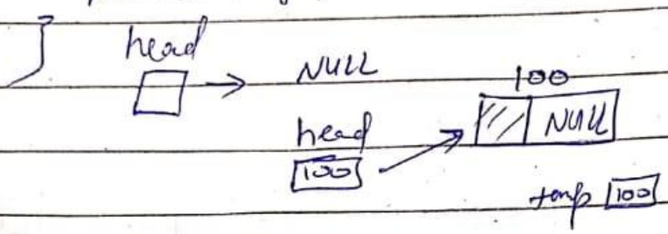


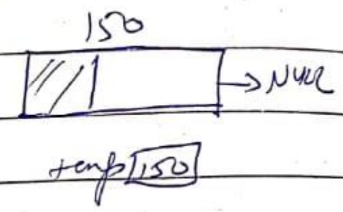

```
Node* A;
A= NULL;
Node * temp = new Node();
temp -> data = 2;
temp -> link = NULL;
A = temp;
```



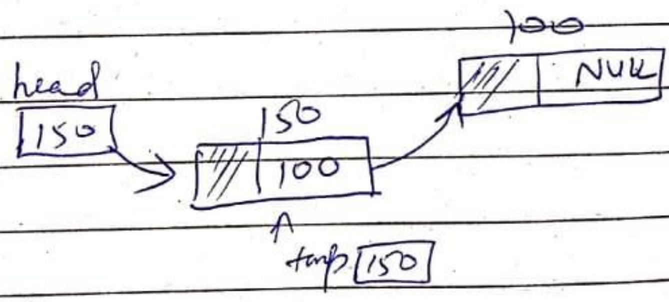
Linked List : Insert a node at beginning.

```
void Insert (int x)
{ Node * temp = new Node();
  temp -> data = x;
  temp -> next = NULL;
  head = temp;
}
```





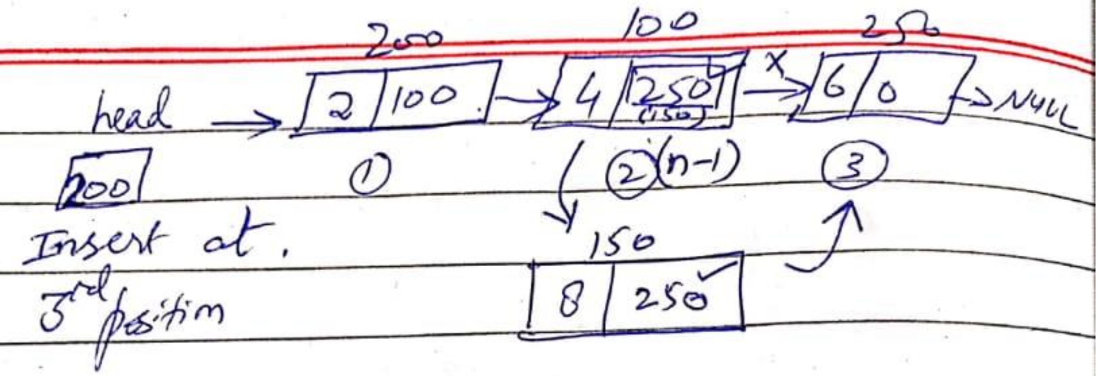

```
if (head != NULL)
   temp -> next = head;
```




```
void Print()
{  Node* temp = head;
   while (temp != NULL)
   {  cout << temp -> data;
      temp = temp -> next;
   }
}
```

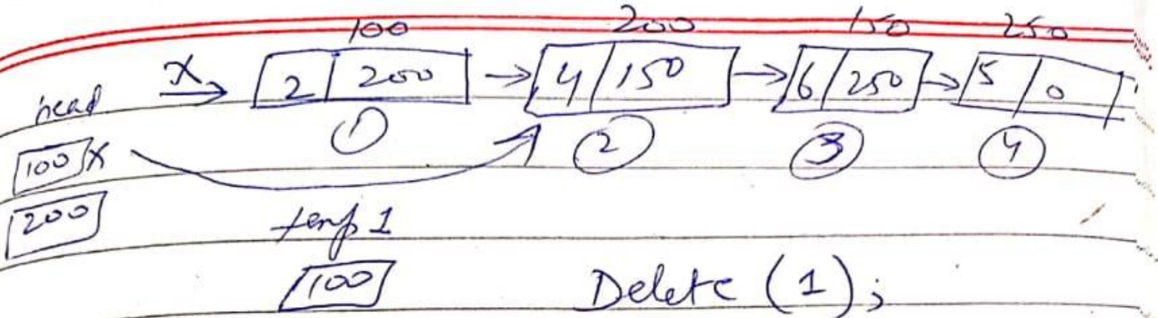
Linked List : Inserting Node at nth position.

```
int main()
{ head = NULL;
  Insert(2,1);    2
  Insert(3,2);    2,3
  Insert(4,1);    4,2,3
  Insert(5,2);    4,5,2,3
  Print();
}

void Insert (int data, int n)
{ Node* temp1 = new Node();
  temp1 -> data = data;
  temp1 -> next = NULL;
  if (n == 1)
  { temp1 -> next = head;
    head = temp1;
    return;
  }
  Node* temp2 = head;
  for(int i=0; i<n-2; i++)
  { temp2 = temp2 -> next;
  }
  temp1 -> next = temp2 -> next;
  temp2 -> next = temp1;
}
```

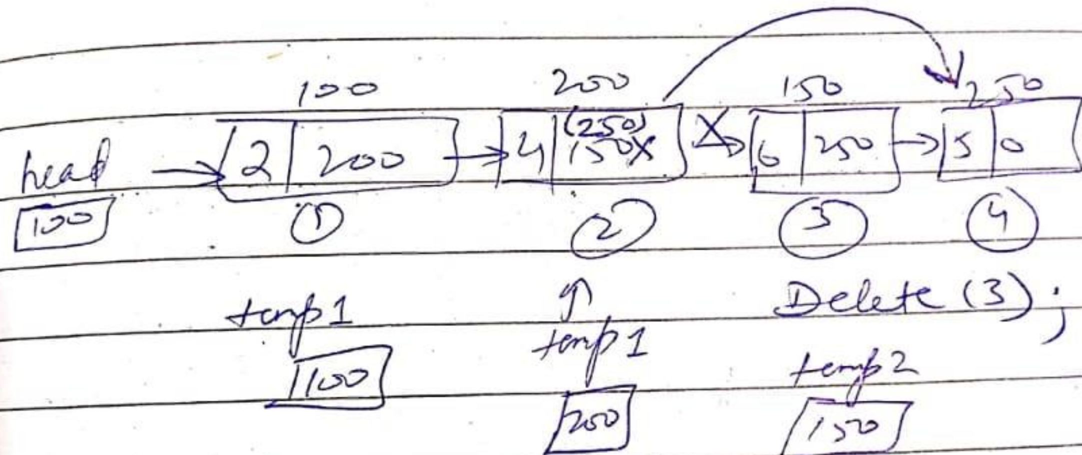


Delete node at position n

```
void Delete (int n)
{ Node* temp1 = head;
  if (n==1)
  { head = temp1 -> next;
     delete temp1;
     return;
  }
  int i;
  for (i=0; i<n-2; i++)     temp1 points
    temp1 = temp1 -> next;  //(n-1) node
  Node* temp2 = temp1 -> next; //nth node
  temp1 -> next = temp2 -> next; //(n+1)th node
    delete temp2;
}
```



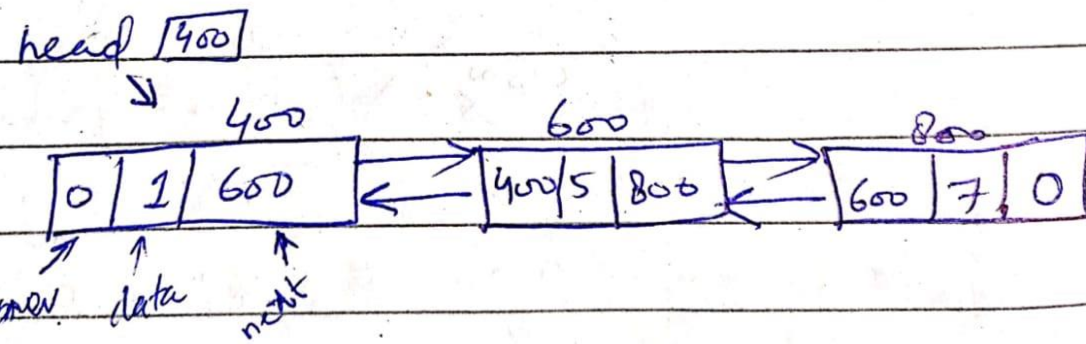
Delete (1);



Delete (3);

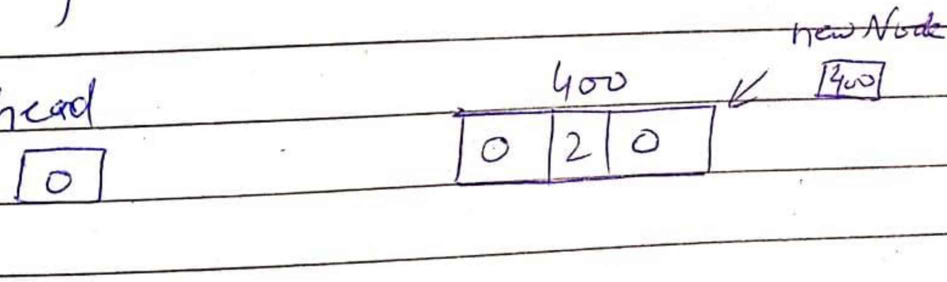
Doubly Linked List

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

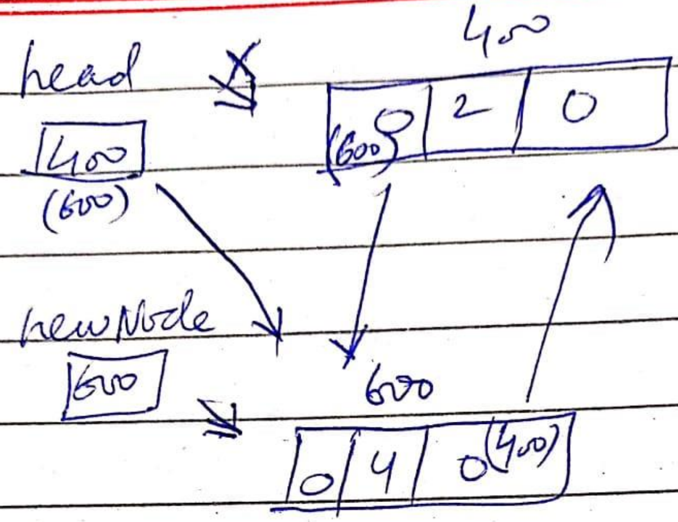


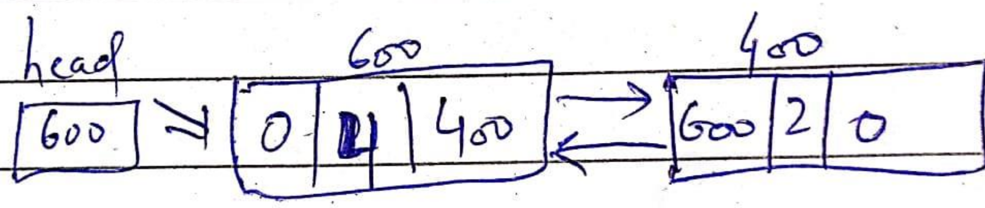

```
Node* head;
```

```
void Insertathead (int x)
{ Node * newNode = GetNewNode (x);
  if (head == NULL)
  { head = newNode;
      return;
  }
  head -> prev = newNode;
  newNode -> next = head;
  head = newNode;
}
```



Insertathead(2);


head
400
(600)
400
(600)
2
0
newNode
600
600
0
4
0(400)

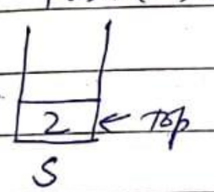

head
600
600
0
4
400
400
600
2
0

Stacks (LIFO)

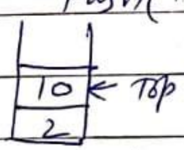
A list with the restriction that insertion and deletion can be performed only from one end, called the top.

Operations
Push(x);
Pop();

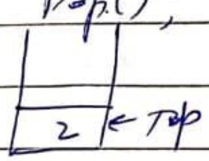
eg Push(x) = Push(2);



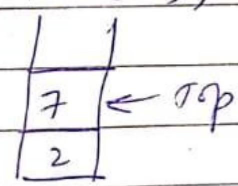
Push(10);



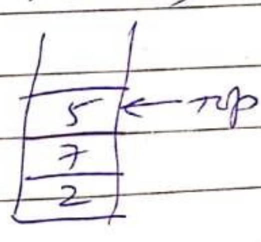
Pop();



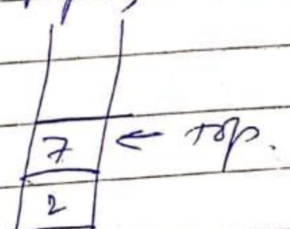
Push(7);



Push(5);



Pop();



Application :-

- Undo in an editor.
- Balanced Parenthesis.

Implement stacks using :-

- Arrays
- Linked List.

Stack - Array based Implementation.

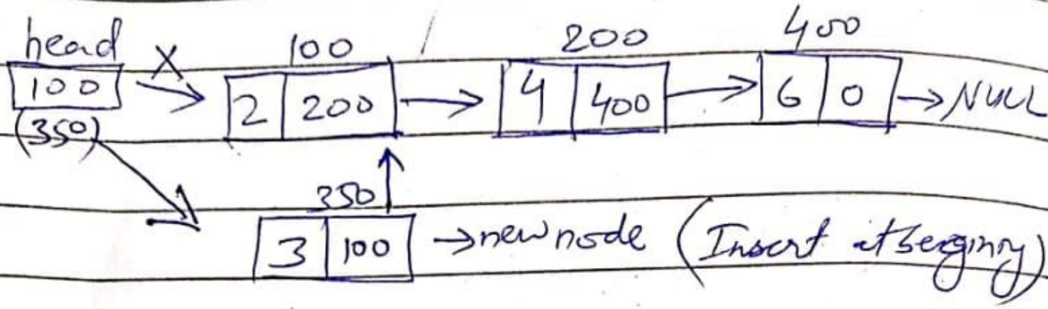
```
void Push(int x)
{   top++;
    A[top] = x;
}
```

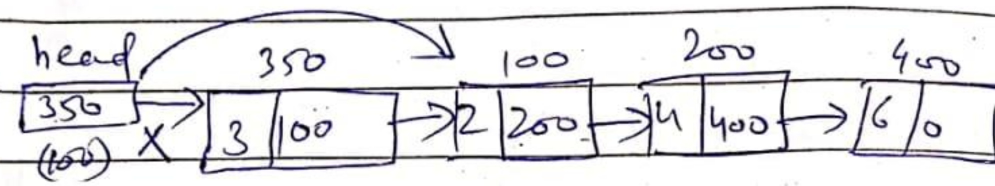
```
if (top == Max-1)
{
    cout<< "stack full";
    return;
}
```

```
void Pop()
{
    if (top == -1)
    {   cout<< "No element to pop";
        return;
    }
    top--;
}
```

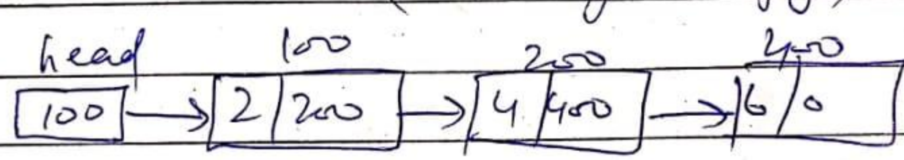
```
void Print()
{ for (int i=0; i<=top; i++)
    cout << A[i];
}

int main()
{ Push(2); Print();
  Push(10); Print();
  Push(20); Print();
  Pop(); Print();
  Print(30); Print();
}

#define Max 100
int A[Max];
int top = -1;
```

Stack - Linked List based Implementation





(Delete from beginning)

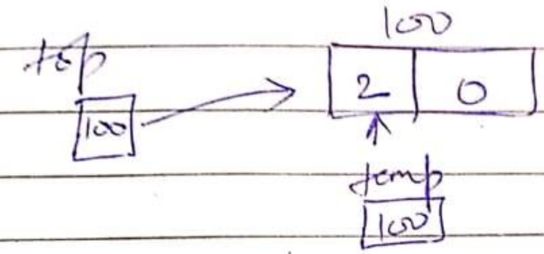



```
struct Node
{ int data;
  Node* link;
};
```

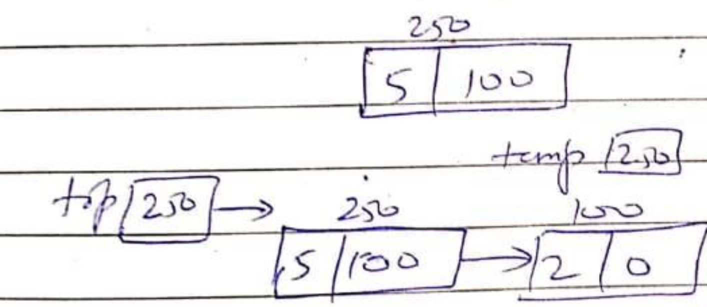
`Node* top = NULL;`   top [0] stack is empty.

```
void Push (int x)
{
  Node * temp = new Node ();
  temp -> data = x;
  temp -> link = top;
  top = temp;
}
```

Push(2);

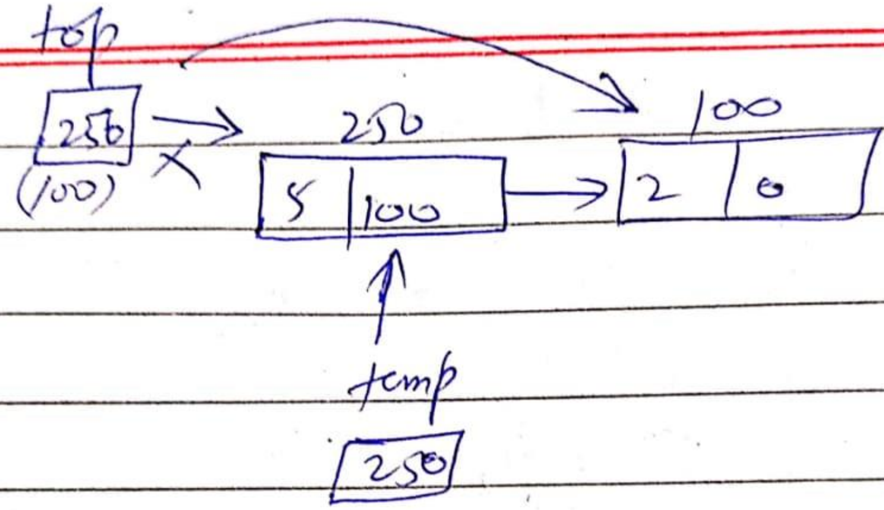


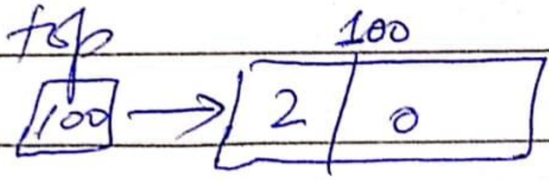
Push(5);




```
void Pop()
{ Node * temp;
  if (top = = NULL) return ;
  temp = top;
  top = top -> link;
  free (temp);
}
```

Pop();


top
250
(100)
250
5
100
100
2
0
temp
250


top
100
100
2
0

Queues

FIFO (First In First out)

A list with the restriction that insertion can be performed at one end (rear) & deletion can be performed at other end (front).

Enqueue → ______ → Dequeue

Q

Enqueue(2);

2

Q

rear front

Enqueue (5);

rear ↘ ↙ front

5	2

Q

Enqueue(3);

rear ↘ ↙ front

3	5	2

Q

Dequeue ();

rear ↘ ↙ front

3	5

Application :-

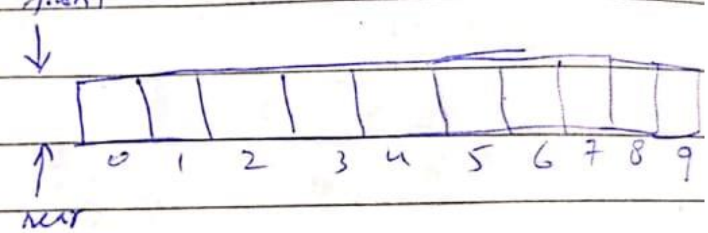
- Printer queue
- Process scheduling

Implement Queue using :-

- Array
- Linked List.

Queue - Array based Implementation.

int A[10];

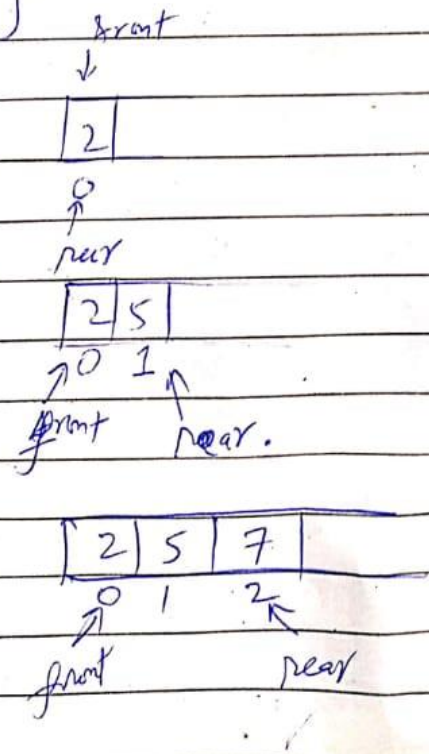


front ← -1
rear ← -1

```
IsEmpty ()
{ if front == -1 && rear == -1
      return true.
  else
      return false.
}
```

```
Enqueue (x)
{ if rear == size(A) - 1
    cout << Queue is full;
    return;
  else if IsEmpty()
  {
    front ← rear ← 0
    A[rear] ← x ;
  }
  else
  { rear ← rear + 1
    A[rear] ← x
  }
}
```

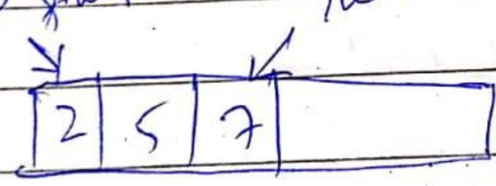
Enqueue(2);
Enqueue(5);
Enqueue(7);

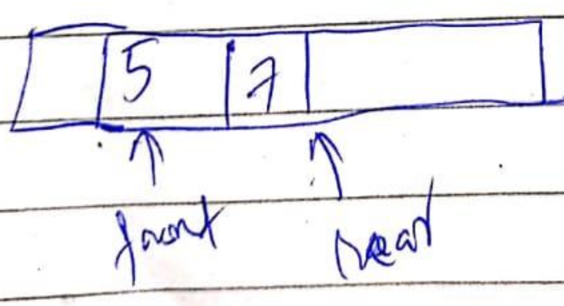


Dequeue ()
{ if IsEmpty ()
return;
elseif front == rear
front ← rear ← -1

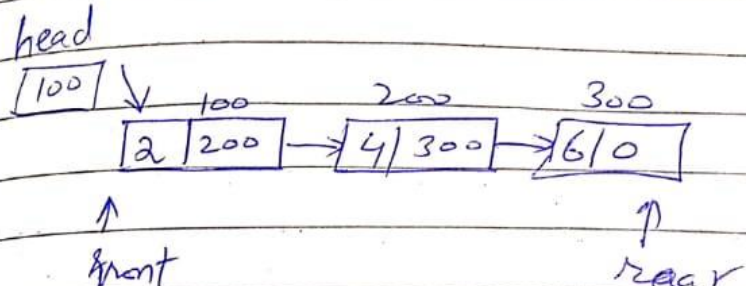
else
front ← front + 1
}

Dequeue ();

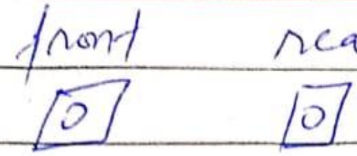




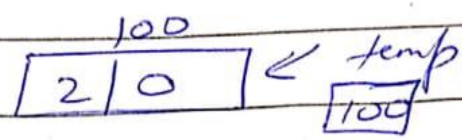
Queue - Linked List Implematation

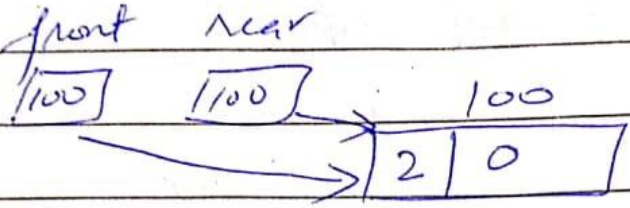



```
struct Node{
    int data;
    Node* next;
};
Node* front= NULL;
Node* rear= NULL;

void Enqueue (int x)
{
    Node* temp= new Node();
    temp -> data = x;
    temp -> next= NULL;
    if (front == NULL && rear == NULL)
    { front= rear = temp;
        return;
    }
    rear -> next = temp;
    rear = temp;
}
```

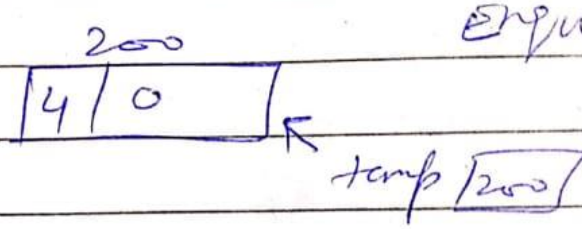


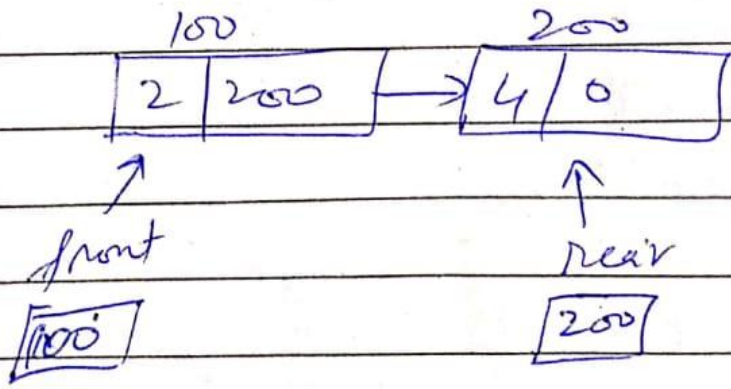
Enqueue(2);





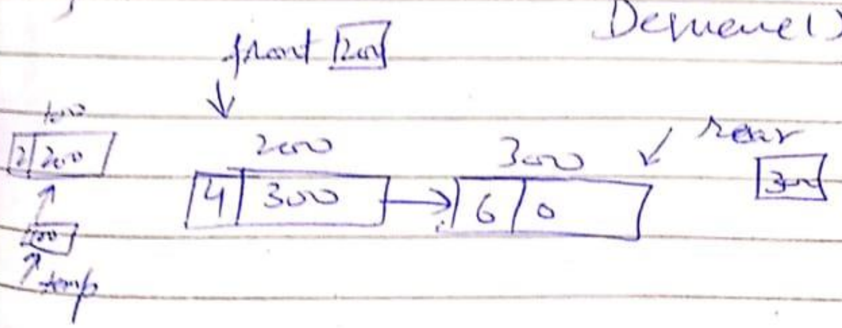
Enqueue(4);





```
int Dequeue ( )
{
    Node* temp = front;
    if (front == NULL) return;
    if (front == rear)
    {
        front = rear = NULL;
    }
    else
    { front = front -> next;
    }
}
```

Dequeue();




```
free (temp);
delete temp;
```