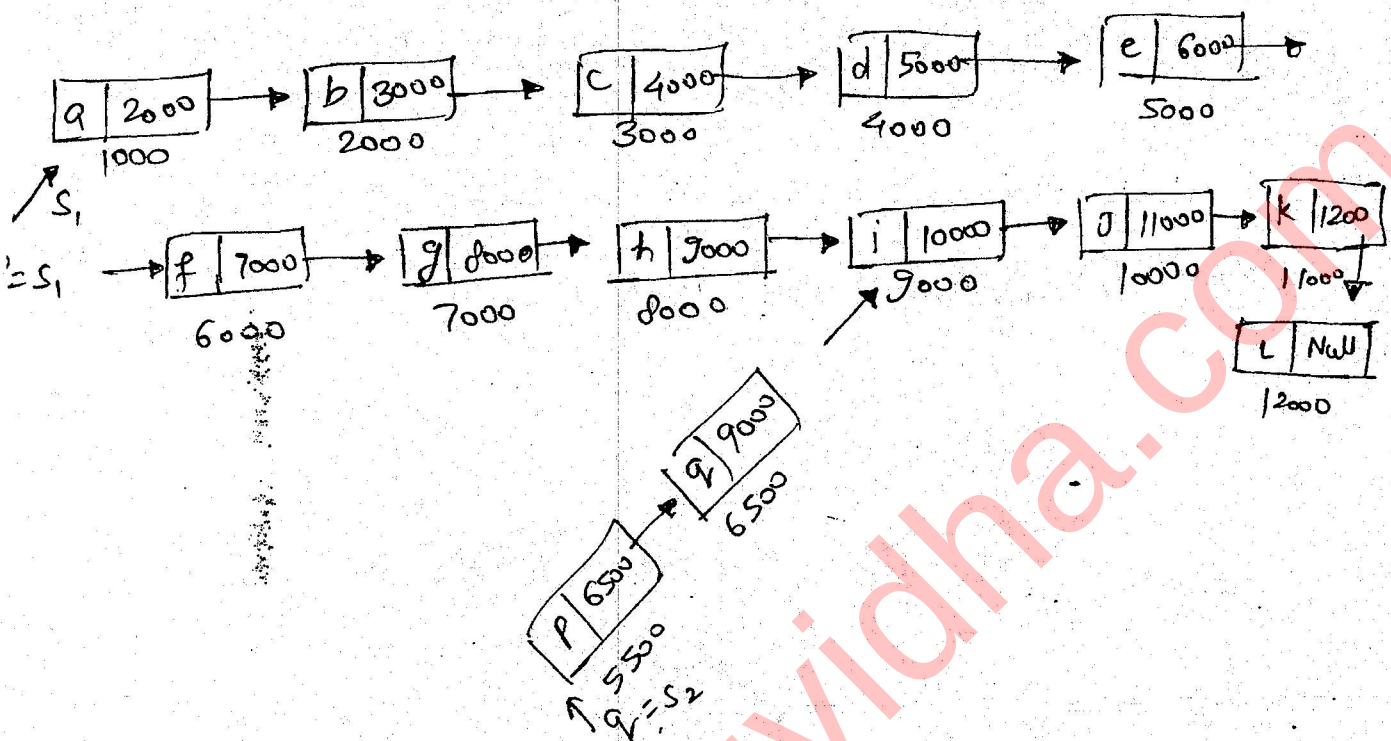
Que :- write a 'C' program to find Address of first intersecting node of given two link-list.



$p = S_1$ S_2

$C_1 = 0;$

① while (S_1 != Null) ↑

{ $S_1 = S_1 \rightarrow Next;$ O(m)

$C_1 = C_1 + 1;$ ↓

}

$q = S_2$ $C_2 = 0$

② while (S_2 != null) ↑

{ $S_2 = S_2 \rightarrow Next;$

$C_2 = C_2 + 1;$ O(n)

} ↓

$S_1 = p$

$S_2 = q$

③ if ($c_1 > c_2$)

$d = c_1 - c_2$

else

$d = c_2 - c_1$

$m > n$

④ while (d != 0)

{

$s_1 = s_1 \rightarrow next;$

$d = d - 1;$

}

m

~~max(m,n)~~

⑤ while ($s_1 != s_2$)

{

$s_1 = s_1 \rightarrow next;$

$s_2 = s_2 \rightarrow next;$

}

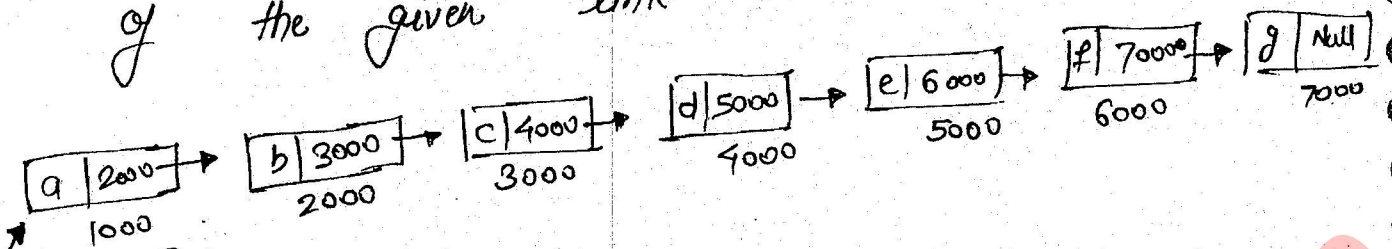
Time Complexity = $O(m) + O(n) + O(max)$

$= O(m+n)$

Questions :-

1. Program to perform merge sort on given Linklist
2. Implement stack using Link list
3. implement queue using link list.
4. Program to reverse given link list
5. Program to find cycle in given link list

ue write a 'C' program to find middle node (Addr.) of the given link-list



Algo ① :-

$S_1 = S$ $C = 0$;

① while (S_1 != null) } O(n)

{ C = C+1;

$S_1 = S_1 \rightarrow$ next;

}

$\frac{3n}{2} \Rightarrow O(n)$

② $C = \lceil C/2 \rceil$

③ while (C != 1) } n/2

{ C = C-1;

S = S → next;

}

Algo ② :-

p
1000
2000
3000
4000
5000
:
n/2

q
1000
3000
5000
7000
9000
:
n

Algo ② :- :- finding mid element

```
mid (S)
{
① P = q = S;
② while ( q != null && q -> next != null &&
          q -> next -> next != null )
  {
    q = q -> next -> next;
    P = P -> next;
  }
  return (P);   /* returning address of middle Node */
}
```

n/2 ⇒ O(n)

★ Que :- write a 'C' program to perform Binary Search on the given sorted link list.

```
BS (S, x)        <- element to search
{
  if ( S -> next == null)
  {
    if (S -> data == x)
      return (S);
    else
      return (null);
  }
  else
  {
    m = mid (S);   ⇒ O(n)
    if ( m -> data == x)   ⇒ O(1)
      return (m)
```

O(n)

```
if (x < m->data)  ⟹ O(1)
else {
    m->next = null
    BS(s, x);   ⇒ T(n/2)
}
else {
    BS(m->next, x);  ⇒ T(n/2)
}
```

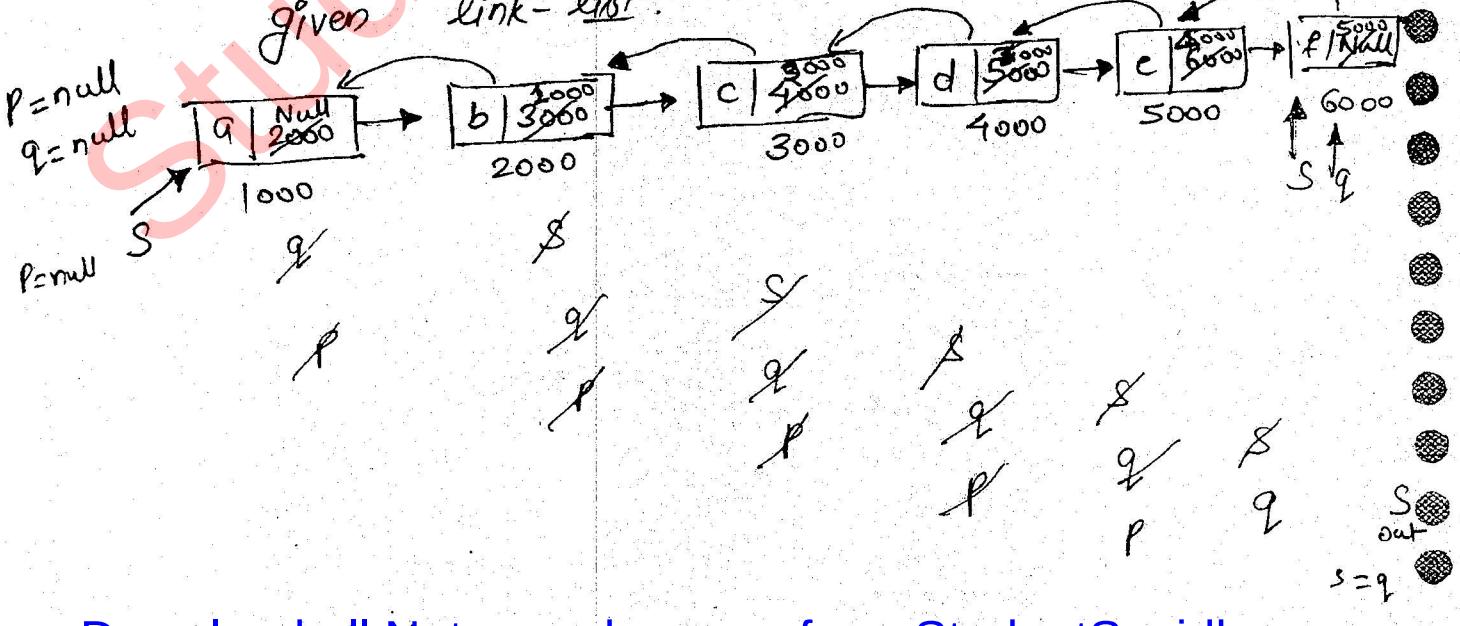
Time complexity ⇒ $T(n) = T(n/2) + n$

⇒ $O(n)$ {Every case} {on Link list}

Note :- On Link-list Binary search is possible but not Efficiently because :

	Linear search	Binary search
Time :	$O(n)$	$O(n)$
space :	$O(1)$	$O(\log n)$

Que :- write a 'C' Program to reverse the given link-list.



```
Reverse (s)
{
  p = q = null;
  while (s != null)
  {
    p = q
    q = s
    s = s → next;
    q → next = p;
  }
  s = q
  return (s);
}
```

/* reverse */

O(n) [every case]

Que :- write a 'c' program to perform merge sort on the given link-list.

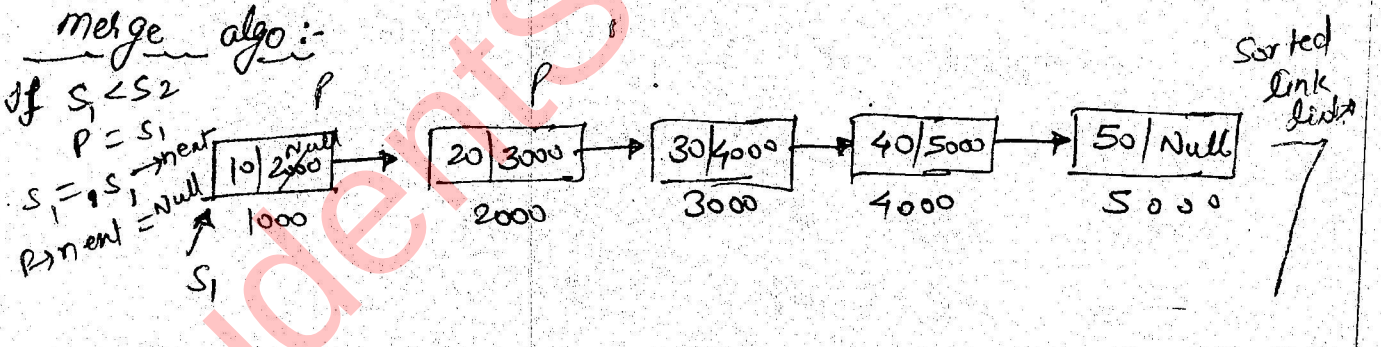
merge algo :-

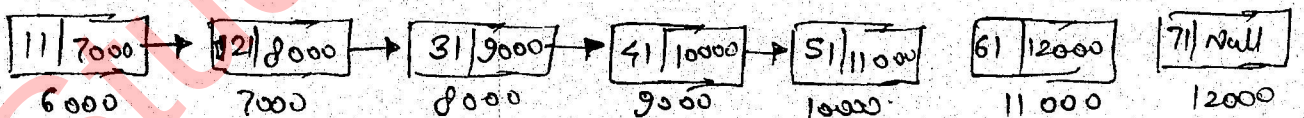
if $s_1 < s_2$

$p = s_1$

$s_1 = s_1 \rightarrow next$

$p \rightarrow next = Null$



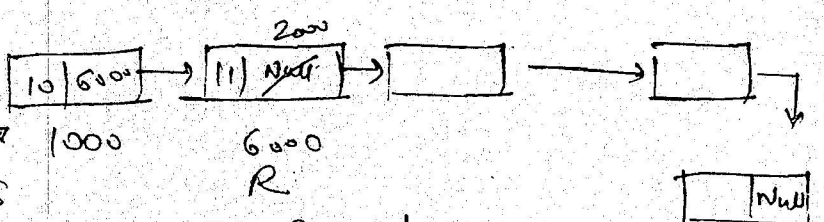


if $s_2 < s_1$

$p = s_2$

$s_2 = s_2 \rightarrow next$

$p \rightarrow next = Null$



$R \rightarrow next = p$

$R = p$

merged link list

In merging list - list, Procedure is same as merging Arrays.

Note :-

merging two Sorted link list (each of size (m,n)) will take O(m+n) time [worst case].

It is a inplace algo, as no m/y created just modifying links.

Best case (Time) = O[min(m,n)]

★ Two Link lists with S_1, S_2 starting address. will be merged with the two extra pointers P, R.

If (S_1 → data < S_2 → data) then P = S_1

otherwise P = S_2. then set P → next = null

and S_1 or S_2 = $S_{1,2}$ → next respectively

move this P to merging link list.

so R → next = P & R = P.

merge - Algo :-

mergesort :-

```
mergesort (S)
{
  if ( S → next == Null )
      return (S);
  else
  {
    m = mid (S);   ⇒ O(n)
    P = m → next;
    m → next = null;
    S1 = mergesort (S);
    S2 = mergesort (P);   } ⇒ 2T(n/2)
    S = merge algo (S1, S2);  ⇒ O(n)
    return (S);
  }
}
```

Time complexity (Link list) ⇒ $T(n) = 2T(n/2) + 2n$

$= 2T(n/2) + n$

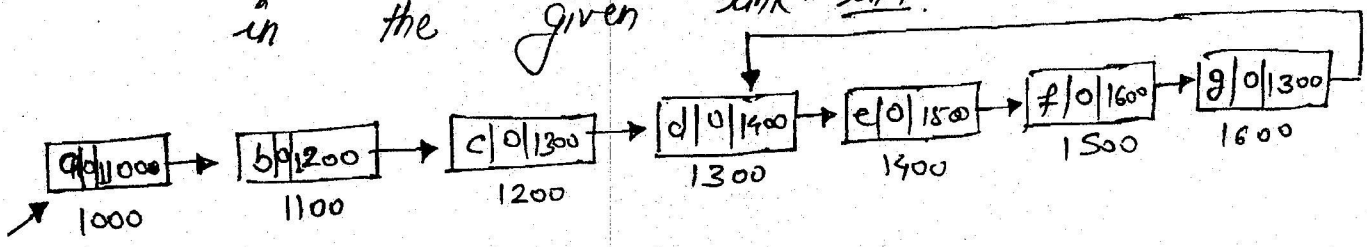
$T(n) = \underline{O(n \log n)}$ { Inplace merge sort with Link-List }

mergesort :

Time complexity (Array) $T(n) = 2T(n/2) + n$

$= \underline{O(n \log n)}$ { But outplace Algo (Arrays) }

Que :- write a 'C' program to find cycle (loop) in the given link-list.



Algo (i) :- (or) BFT

(i) Assume that every node contain 1-more extra field "flag" which is 'zero' (0) initially.

(ii) visit everynode and check flag

```
if (flag == 0)
    flag = 1;
else
    Printf(loop found);
```

Time = O(n)

Extra space = O(n) for 'flag'

Algo (ii) :-

p (increament by 1)	q (increament by 2)
1000	1000
1100	1200
1200	1400
1300	1600 (after 1600 → 1300)
1400 ⟷ matched	1400 (cycle is there)

[p will go only once from start to end, q may go for many rounds, anywhere it can meet p. so worst case Time = O(n)]

Algo :- ① $P = q = s$

② P-by-1
q-by-2

③ while ($P != q$)
{
P-by-1
q-by-2
}

$O(n)$ time

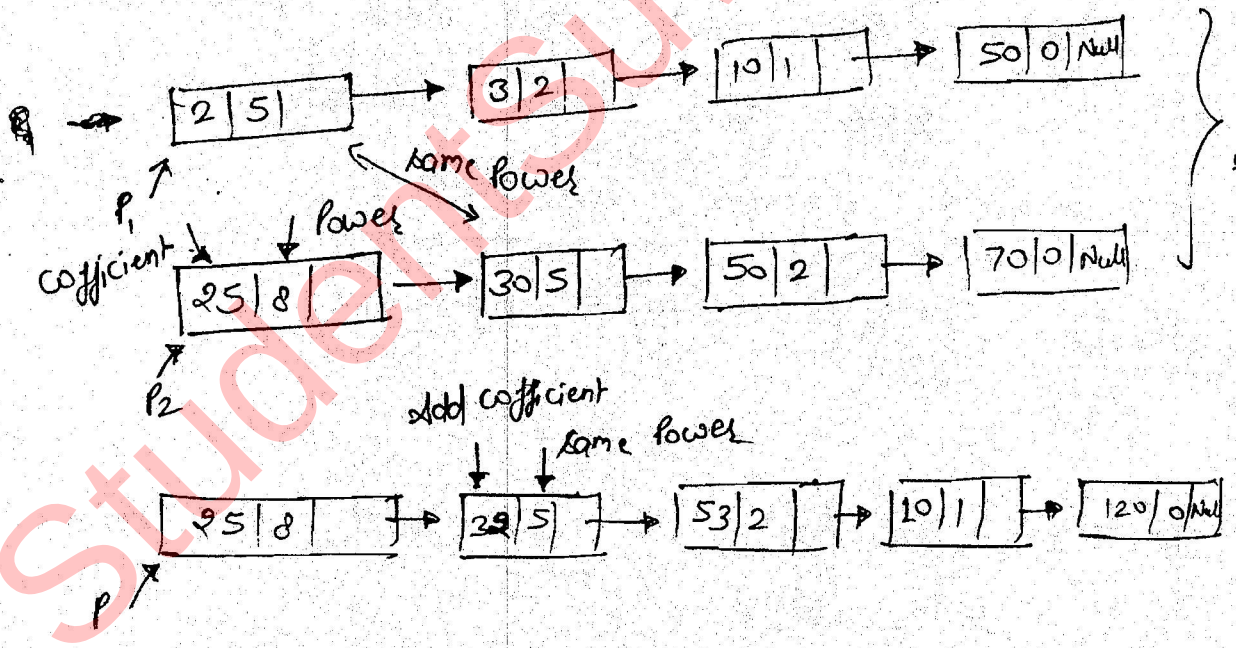
Extra space = $O(1)$ [2 Pointers]

:- Polynomial Addition :-

$P_1 = 2x^5 + 3x^2 + 10x + 50$

$P_2 = 25x^8 + 30x^5 + 50x^2 + 70$

$P_1 + P_2 = 25x^8 + 32x^5 + 53x^2 + 10x + 120$



★ Note :- using link-list Polynomial Addition & multiplication is possible :

Poly Addition ⇒ T = $O(m+n)$ [worst case]

★ Poly multiplicatn ⇒ T = $O(m \cdot n)$ [every case]

Drawbacks of Single link-list :-

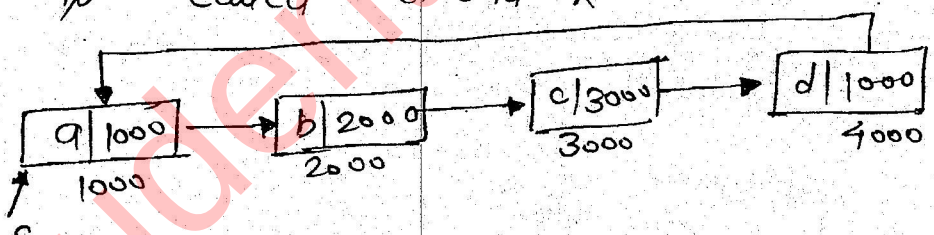
* i) we can not go back, because Every Node contain only one pointer and that pointer pointing to next Node.

* ii) Single link-list last Node Next part always "Null" that means, we are not utilizing it properly.

[To Eliminate above two drawbacks, we are going to circular link-list].

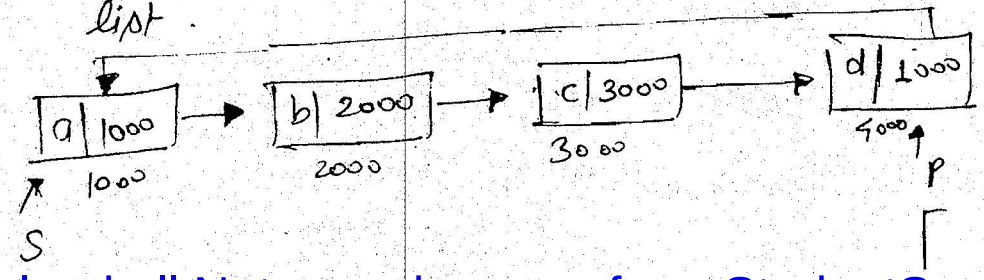
"Circular Single Link-list" :-

* i) In single link-list last Node Next part if we store first Node address then it is called circular single link-list.



* ii) we can go back, But it will take more time (O(n) time).

* * How to findout Last Node of circular single link list.



code :

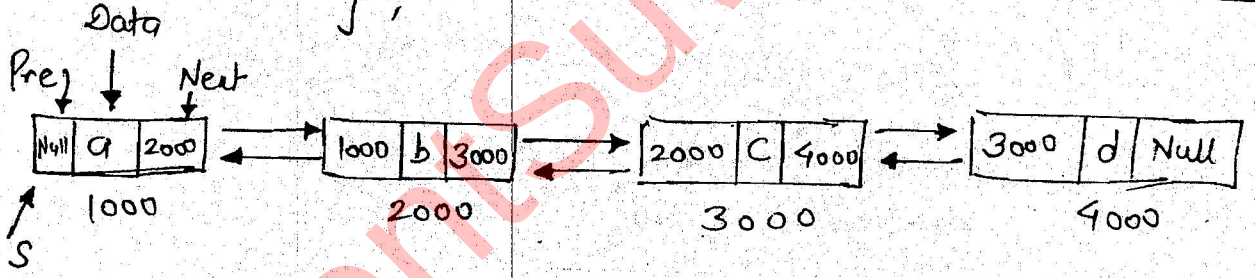
```
P = S
while ( P → next != S)
{
    P = P → next;
}
```

"Double Link List" :-

```
Struct DLL
{
    Struct DLL * Pre;
    int data;
    Struct DLL * next;
};
```

which : containing Pointer Pointing to same type of Data, so called "self Referencing Data Structure"

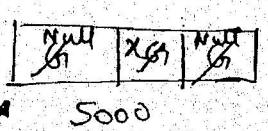
18 B



Que :- write a C Program to insert a node with Data 'x' at the end of the given double link-list.

① create Node :- Struct DLL *P;

P = (Struct DLL *) malloc (size of (Struct DLL))



P → Pre = Null ; P → data = x.

② while (S→next != null) } go to last Node.
{ S = S→next;
}

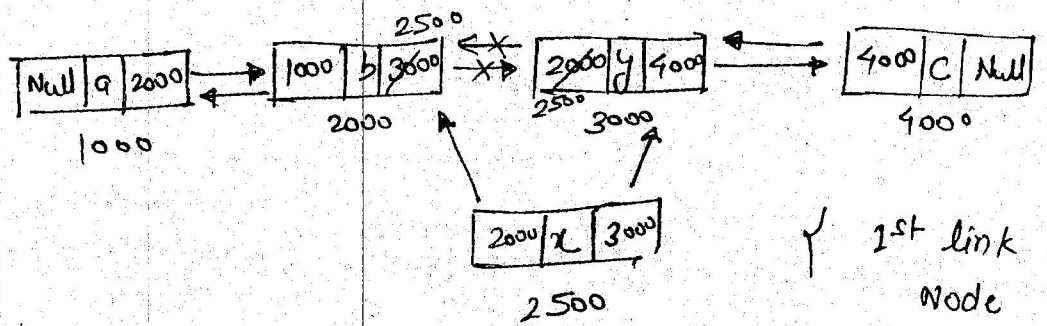
③ S→next = P; } linking
P→Pre = S;

Que :- Inserting a Node with data 'x' before a Node with data 'y'

① Create Node :

Struct DLL *P;
P = (Struct DLL *) malloc (sizeof (struct DLL));
P→Pre = Null;
P→Next = Null;
P→data = x;

② while (S→~~next~~data != y && S→next != null)
S = S→next;
If (S→data == y)
{
1.) P→Prev = S→Prev; } Linking New Node
2.) P→Next = S;
3.) S→Prev = P;
4.) P→Prev→Next = P }
}



1st link New Node then Remove old links

Que :- write a C program to delete a Node which contain data 'x' in the given double link-list.

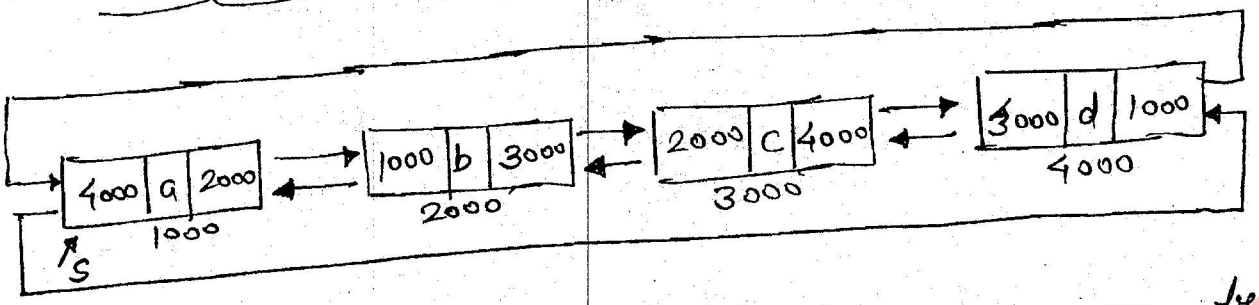
~~while (S → data != x)~~

~~{~~

~~S = S → next;~~

```
while ( S→data != x && S→next != null)
{
S = S→next;
}
if ( S→data == x)
{
S→Prev→next = S→next;
S→next→Prev = S→Prev;
free (S);
S = Null;
}
```

"Circular Double Link-List" :-



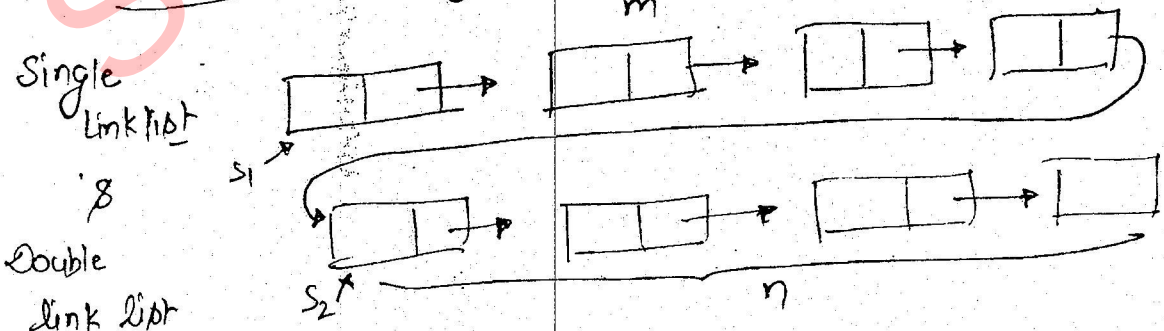
:- from starting node to last Node can be reached in constant time O(1).

:- So Adding Node at end [Time = O(1)
Adding Node at start [Time = O(1))
At start & end deletion also = O(1)

:- But in middle, same as double link list = O(n)

★ Note :- The greatest advantage with the double link-list or circular double link-list is If you lost one pointer, with help of another pointer you can get it back. (there may be possibility).

Concatenation of two link list:



O(m) :- go to last node & Put S_2 in last node next.

*) Concatenation of two circular single Linklist

$O(m+n)$

**) Circular double link list concatenation

$O(1)$

:- Intersection & union what will be Time complexity in single link-list.

[1st sort (merge sort) = $O(n \log n)$

2nd merge algo apply = $O(m+n)$ or $O(n)$]

so Time = $O(n \log n) + n$

$= O(n \log n)$