Union node (In union m/m is created as much as the larger one)

```
{
member1  int a[5];    ——10
member2  float b[20]; ——80
member3  char c[50];  ——50
};
```

80 bytes — This is shared by all the ~~pointer~~ members (one member is used at a time)

- Previous content is overwritten.
- In union M/m Sharing is present.

V.Imp

LINKED LIST (Interview)

☆ Struct node (6bytes)

```
{
    int data;          // 2bytes
    Struct node *next; // (pointer) → 4bytes
};
```

means address which is pointing to same kind of data type (i.e. Struct node)

// Self Referential data structure

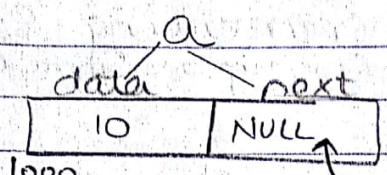
printf("%d", sizeof(structnode)) = 6bytes

☆ Struct node

```
{
    int data;
    struct node next; // wrong
};
```

It is not a pointer

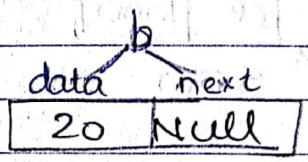
Error (Not a self Referential data structure)



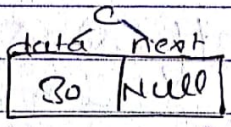
object created

Structnode a={10, Null};

Structnode b={20, Null};



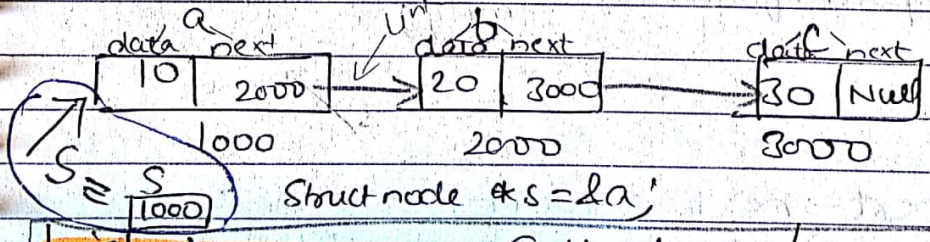
Struct node c={30, Null};



Linking (statically) (we are ourself specifying what should we put in next)

a.next = &b;

b.next = &c;



Structnode *s = &a;

List ⇒ means Collection / Structure of data.

printf(s) = 1000.

(*s).data = 10 ≡ s→data = 10

(*s).next = 2000 ≡ s→next = 2000

(*s) inside 1000

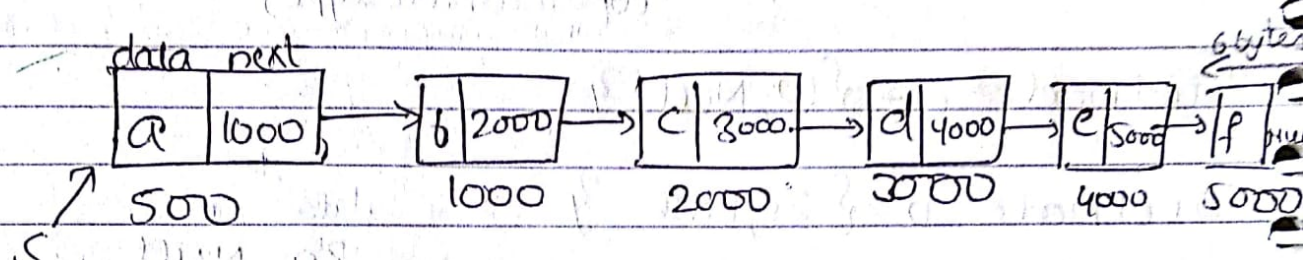
dynamic M/m allocation

malloc(6) = allocate M/m of 6 bytes

free(1000) = free M/m of 1000 bytes

★ S = S → next (incrementing S)
(pointing everytime other node.

Ques. Consider the following Linked List

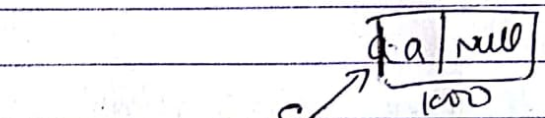


S
500

★ printf(S) = 500

Conditions (Important)

★ if (S → next == NULL)
printf (S is last node)



"LL has one element"

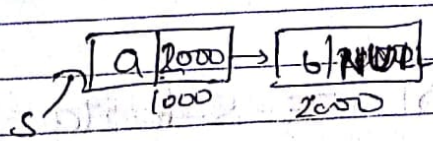
★ if (S == NULL)
printf (Linked List is empty)

Linked List is empty

S

★ if (S → next → next == NULL)
printf (LL has 2 elements)

LL has 2 elements



★ If S = NULL
then *S = *(NULL)
// you can't go inside Null

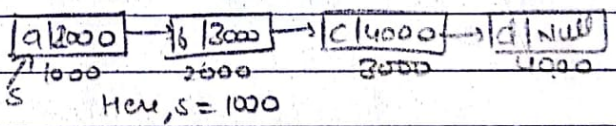
Segmentation Error
(Accessing some M/m location which doesn't exist)

Ques: Write a C program to print data part of every node in the given linked list.

S is a address of structnode type

```
PDL (Structnode *S)        (Starting node)
{
    if (S==NULL) return; // LL over
    else
    {  while (S!=NULL)
       {
          printf (S→data);
          S=S→next;
       }
    }
}
```

Time complexity ⇒ O(n)



output: a b c d e f

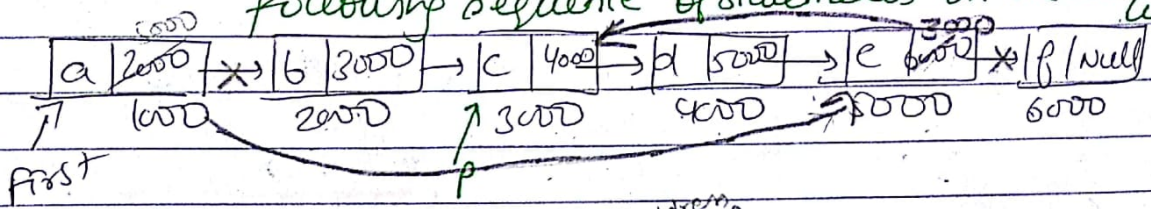
Ques: Write a C program to find the last node of the given linked list. O(n)

```
Structnode* fLL (Structnode *S)
{
    if (S==NULL)  // L.L Over
        return;
    else
    {  while (S→next!=NULL)
       {
          S=S→next;
       }
       return (S);
    }
}
```

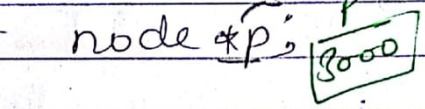
fLL should return structnode type value

address of last node

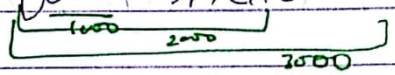
Ques: Consider the following Linked List what is the output after executing following sequence of statements on the above given linked list.



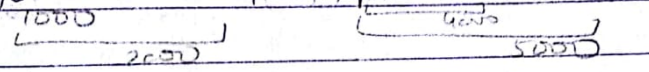
(i) Struct node *p; store address in p



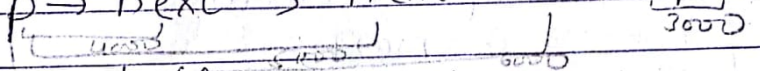
(ii) p = first → next → next;



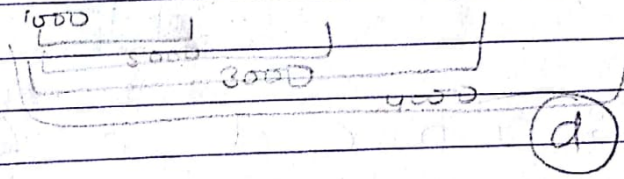
(iii) first → next = p → next → next;



(iv) p → next → next → next = p

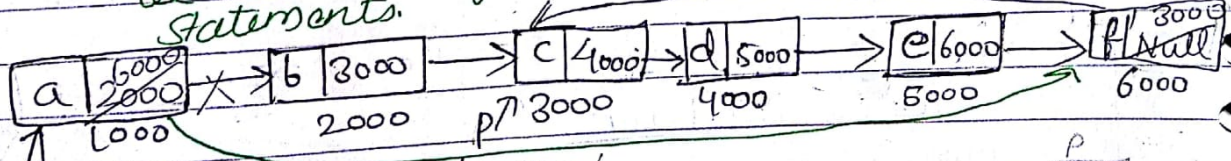


(v) printf (first → next → next → next → data);



Ans: d

Ques: Consider the following Linked List. what will be the o/p if we execute the following statements.



(i) Struct node *p;

(ii) p = first → next → next;

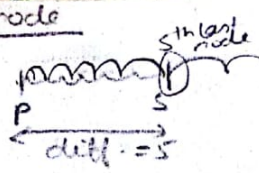
(iii) first → next = p → next → next → next;

(iv) p → next → next → next = p;

(v) printf (first → next → next → next → next → data);

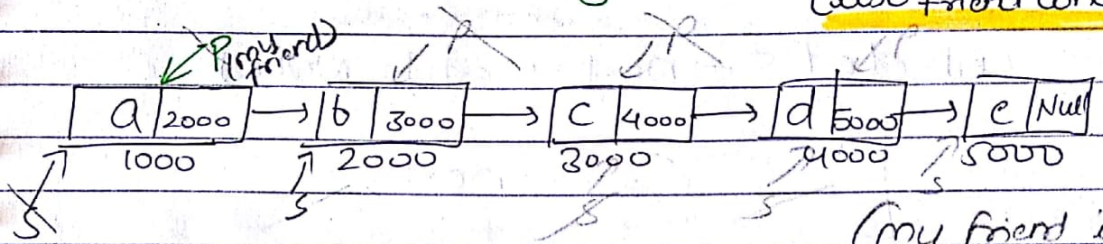
Ans: e

5th last node



(after S reached 5th node say P to work from first node)

Ques: Write a C program to find second last node in the given Linked List. (use Friend concept)



(my friend is previous to me always)

```
FindSecondLast(Structnode *S)
{
1. if (S==NULL) return;   //Empty L.L.
                          or
                          L.L. over
2. else
      if (S→Next==NULL) return;
                          //only one node in L.L.
      else //more than one node in L.L.
        while (S→Next!=NULL)
        {
            P=S;
            S=S→next;
        }
      return P;
}
```

```
FindSecondLast(Structnode *S)
{
1. if (S==null) return;
```

else

2. if (s → next == NULL) return;

3. while (s → next → next != NULL)

```
{
    s = s → next;
}
```

return (s);

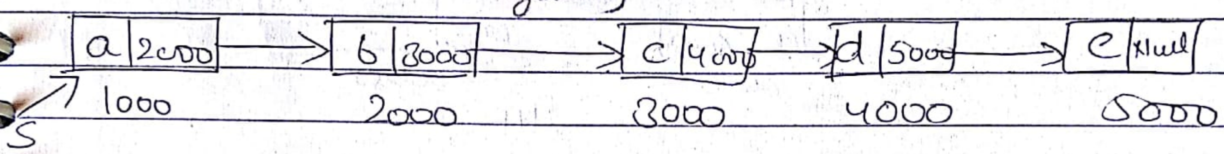
- Most of the problems in single linked list can be solved by the friend node

15th / June / 2013

Lecture 17

Ques: Write a C program to Insert a node with data x at the end of the given Linked List.

↳ (Starting address given)




```
Structnode        (6 bytes)
{
  int data;        //2 bytes
  Structnode *next;  //4 bytes
};
```

↑ address of (same structnode type)

// Structnode *S = 1000;   (S is 1000)
*S = Inside the node

(those m/m which are created dynamically those can only deallocate dynamically (using free))

func returning address

data | next — Garbage | null Garbage (P → 6000) — data for node

Structnode* ← This func is finally returns address & structnode type.

```
Insertatend (Structnode *S, int x)
{
    Structnode *p, *q;   q = S;
    p = (Struct node *) malloc (sizeof(Structnode));
              Type Casting                    (P is 6000)
    (*p).data = x;  (*p).next = null;         (*P = Inside P)

    1. If (S == null)  S→ (or) [NULL] S
    {
        S = p;
        return S;  // returning the address of st node / the linked list
    }
```

p [6000] 3000 (pointer) Contain 6000 structnode pointer

Getnode(x); // Create the node with data x & returns the address where it is created.

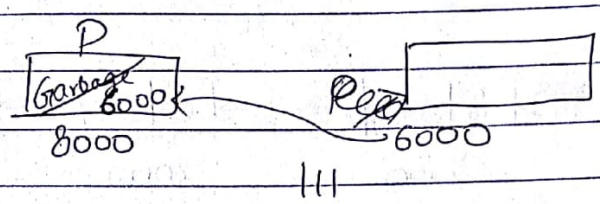
★ Continue on the next to next page

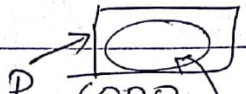
Explanations :→

```
struct node *p;
p = malloc(6Bytes);
```

(struct node pointer) (Type Conversion) (6000 (Void pointer))

= Somewhere 6 Bytes m/m will be created for you. (The address of that m/m is given to us).





(p is a pointer pointing to 6000)

6Bytes data is stored here (Struct node type data is stored).

p = (Structnode *) malloc (6Bytes) = 6000

Type casting — Void pointer

Structnode node pointer (6000) — Sizeof(Struct

p = (structnode*) malloc (sizeof (structnode));

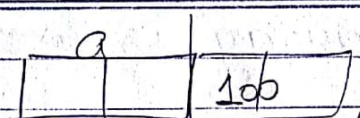
"Creation of M/m"

Note :- ☆ If M/m is not allocated to you (you can only read, not write in the Memory)

☆ If M/m is allocated to you (you can read & write in it).

* Null = Segmentation Error (as we can't go inside the non-available M/m)
Non available M/m

int a=10;
printf(a) ⇒ 10

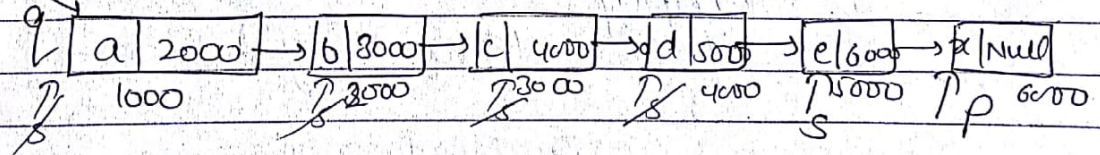


*(&a+1) = 100;
1000
1002

printf(*(&a+1)) ⇒ Garbage (∵ You can't write in the M/m not related to you)

2. while(S->Next != Null)
↳ this node is not the last node
// L.L is not empty, there is atleast one element
{
S = S->Next;
}
// coming out of while loop means, you are at the last node

3. S->next = p;
return(q); // return the starting address of the Linked list.
}



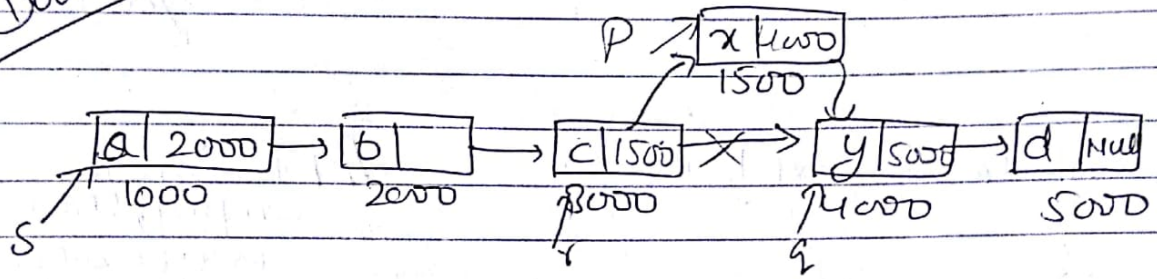
(int*) f(int*, int*); // returns Integer pointer as O/P.
2 Integer Pointers as I/P

(int *) f(float *, (char *)g(int, int *))
↓ returns Integer pointer O/P.
↳ Integer pointer
↳ character pointer
↳ Integer
↳ Integer pointer
func which will takes 2 parameters as I/P. 1st is float Pointer & 2nd parameter is character pointer.

• Size of array is fixed (you can't insert any element in it)

Question Write a C program to insert a node with data 'x' before a node with data 'y'.
(use friend concept)

Solution:




```
Structnode* Insertxbeforey(int x, Structnode *S, int y)
{
    Structnode *p;
    p=(Structnode*) malloc(sizeof(Structnode));
    p->data = x;
    p->Next = null;
1. if empty (I can't do anything if L.L is empty)
2. if (S->data == y)        (first node with y)
   {
       p->Next = S;
       S=P; return(S);
   }
3. q=S, r=Null;            (r = my friend)
   while(q->data != y && q->next != null)
   {                       (curr data is not my data   next node is available)
```

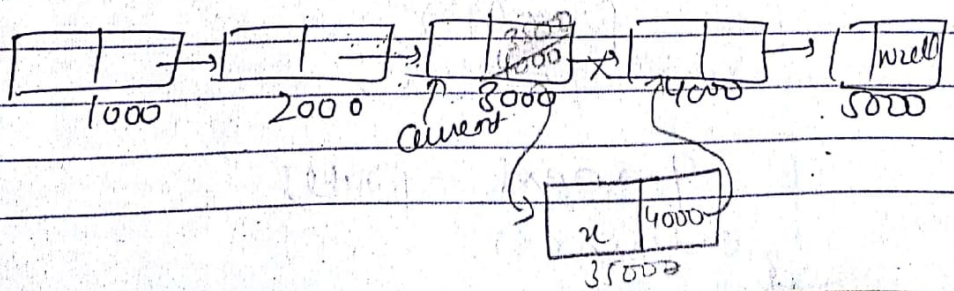
(only 1 node in L.L with data 'y')

```
x = q; // put the friend before me
q = q -> next;
} // why I came out of the for loop
if (q -> data == 'y') // if this will fail means L.L. is over & 'y' is not there in list so we don't add 'x'
{
    x -> next = p;
    p -> next = q;
} else {printf("L.L. is over, y is not there so don't add x"); }
return(s);
}
```

. Time Complexity = O(n). (Even then O(n) is not at all possible ∵ random access is not allowed).

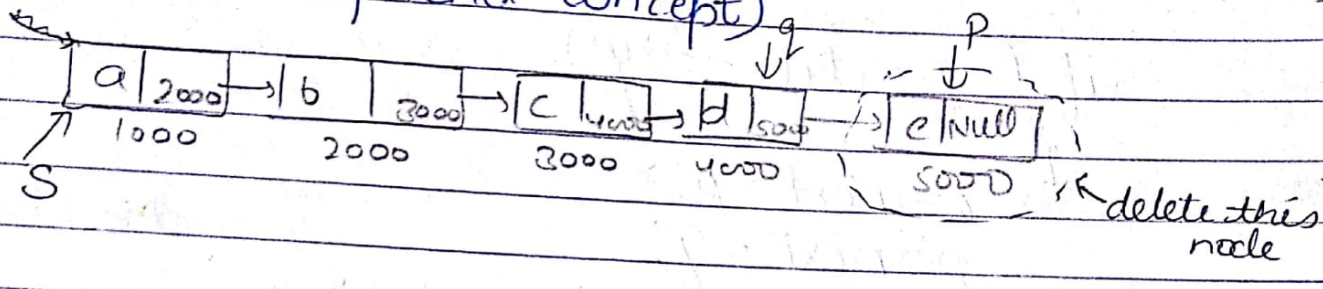
Question Which one of the following statement- Inserts an element x after position current ?

a) current = newnode(x, current)
b) current = newnode(x, current->next);
c) current->next = newnode(x, current);
d) current->next = newnode(x, current->next);

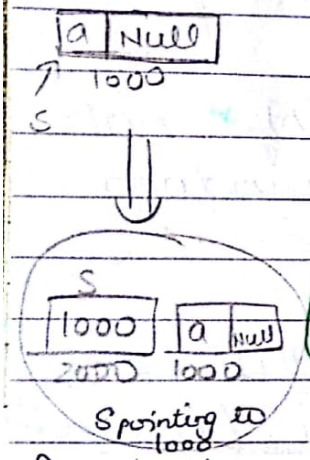


Ques: Write a C program to delete a node at the end of given linked list. (use free also to deallocate the M/m, (use friend Concept)

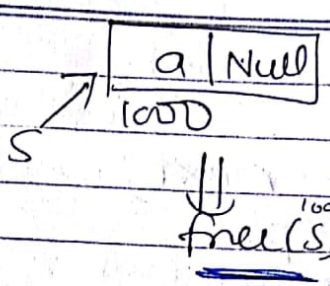
remove the link




```
Structnode * DeleteLast(Structnode *S)
{
    1. empty L.L. (don't do anything)
    2. if(S->next == Null)   (exactly 1 node)
       {
           free(S);
           S = Null;
       }
    3. p = S, q = Null;   (My friend.)
       while(p->next != Null)  // You are not last node
       {
           q = p;
           p = p->next;
       }
       free(p);
       p = Null;  // Now, p is not a dangling pointer
       q->next = Null;
       return(S);
}
```



Ex.

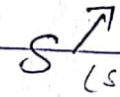


(S is pointing to 1000 which is allocated)

(*S).data = a

printf (S) = 1000 (1000 which is allocated to you)

(dangling Pointer)



(S is still pointing 1000 but 1000th location is not allocated to you)

(S is pointing to 1000 which is deallocated, so S is called dangling pointer)

printf (S) = 1000 (1000 which is not allocated to you)

(*S).data = Garbage.

→ dangling pointer gives wrong answer but not the Segmentation Error.

- we should write program in such a way that it doesn't create the dangling pointer.

Eg. 1. free(S);

2. S = null ← this will remove the dangling pointer

← after free S is dangling pointer

Ques. Write a C program to delete a node with data x from the given linked list.

Solution

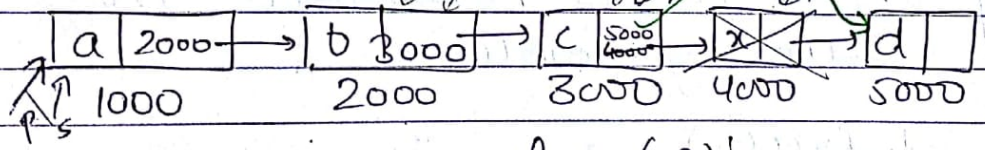
```
Structnode* Deletex(Structnode*S, int x)
{
    1. empty
```

2. if (s→data == x)
{
p=s;
s=s→next;
free(p);
p=NULL;
}

p=s, q=null; (my friend)

while (p→data != x && p→next != NULL)
{
q=p;
p=p→next;
}

if (p→data == x)
{ q→next=p→next;

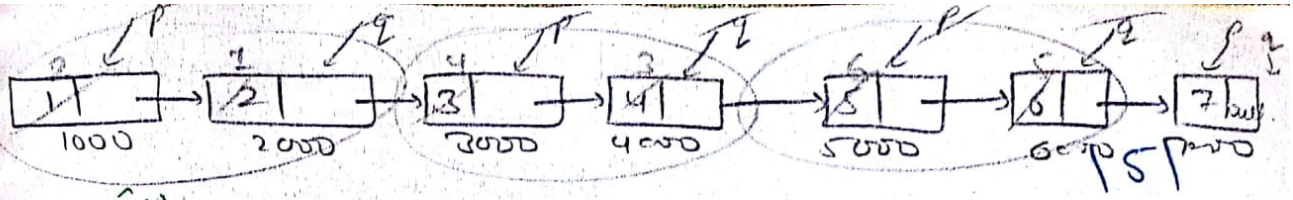


free(p);
p=NULL;
}

else
{
printf("No x found");
}

return(s);

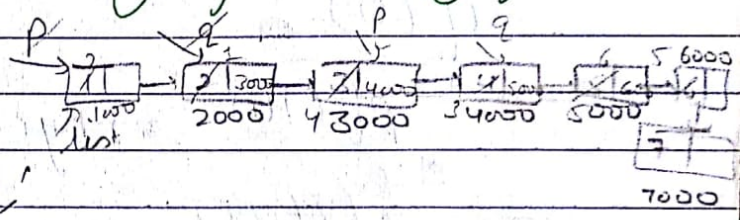
}



Q. Question:

The following C program func: takes single L.L. as a parameter & rearranges the elements of the list. The function is called with the list containing the integers 1,2,3,4,5,6,7 in given order, Then what will be the contents of list after executing following func":

Solution:




```
Struct node
{
    int value;
    Structnode *next;
};
```

temp

```
void rearrange ( struct node *list)
{
    Structnode *p, *q;
    int temp;
    if (!list || !(list->next)) return;
    // if(list->next == null)
    // true when L.L. is empty
    // true when L.L. has 1 element
    else // link list has min. 2 nodes.
    {
        p = list; q = list->next;
        while (q)
        {
            swap(p->value, q->value);
            p = q->next;
            q = p->next;
        }
    }
}
```

o/p: 2, 1, 4, 3, 6, 5, 7

if (Nonzero) → always True
{ ≡ }

if (Zero) → always false
{ ≡ }

Note:

if (1) ← nonzero
printf (Hi) ⇒ Hi
else
pf (Bye)

if (10) ← Nonzero
{
printf (Hi) ⇒ Hi
}
else
pf (Bye)

if (0)
{
Pf (Hi) ⇒ Bye
}
else
pf (Byte)

True = 1 Nonzero
False = Zero

x=10, y=0
if (x==y)
pf (Hi) ⇒ Bye
else
pf (Bye)

x=10, y=20
if (x=y)
pf (Hi) ⇒ Hye
else
pf (Bye)

x = 0, y=20
if (x=y)
pf (Hi) ⇒ Hi
else
pf (Bye)

x = 10, y=0
if (x=y)
pf (Hi) ⇒ Bye
else
pf (Bye);

float ⇒ 0.0000--- 0
char ⇒ '\0'
Integer ⇒ 0
⇒ Null
⇒ Zeros

if (0) ⇒ fail

- printf("%d", Null) ⇒ 0 (ASCII value of Null is 0).

*Null = Segmentation Error.

if(!Null) ⇒ True.
   ||
if(1) or if(!0)

Question

Consider the function f defined below :⇒

```
Struct item
{ int data;
  Struct item *next;
};
int f (Struct item *p)
{
return ((p==null) (OR) (p->next==Null) (OR)
((p->data <= p->next->data) && f(p->next)));
}
```

for given L.L. p the func: f returns 1 iff

a) p contains 0 element or 1 element (also correct)

b) the elements in list P are sorted in non decreasing order. (Increasing of all same elements) (Test also contains duplicates also) ⇒ True for this also [10]→[10]→[10]

c) the elements in list P are sorted in non increasing order

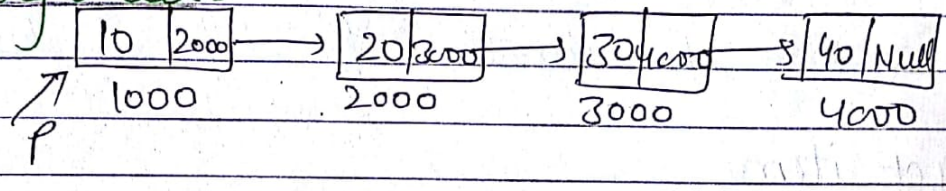
d) None

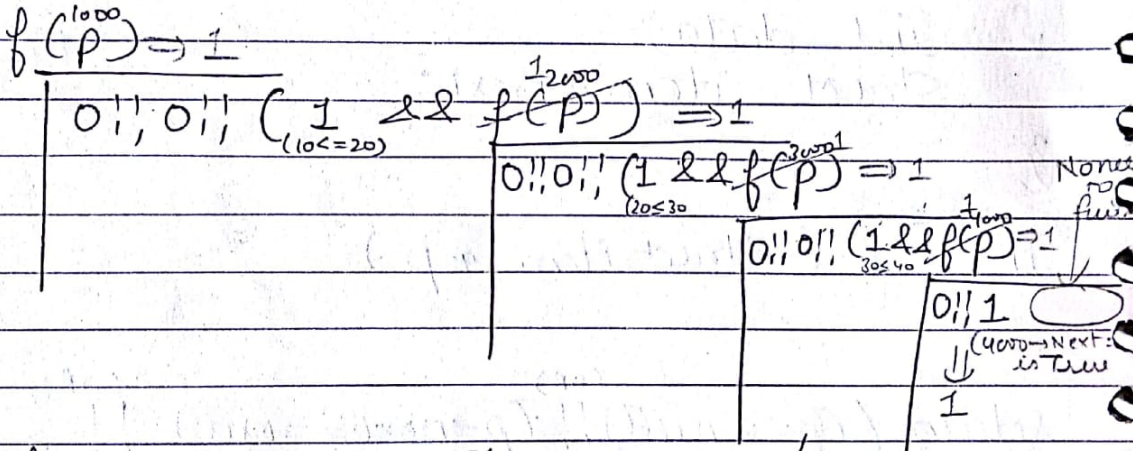
OR

if (a || b || c)

if a itself is true then don't goto b & c. But if a fails goto b & if a, b fails goto c, if all a, b & c fails means if (0).

Increasing order :-





Link List contain Zero element / one element it is still non-decreasing element.
Option (b) contains option (a) also,

So, option (b) is Correct.

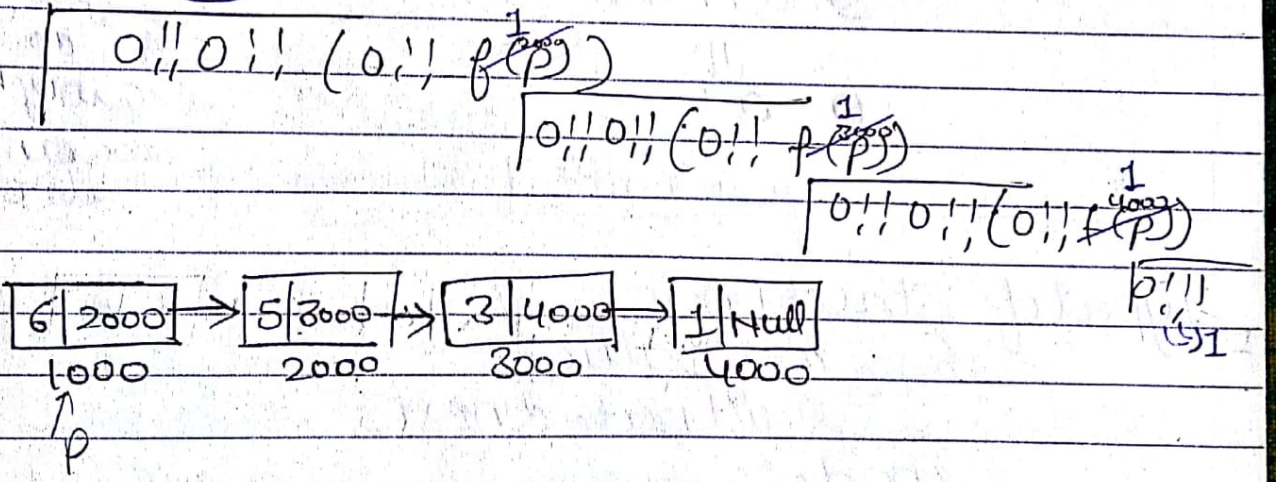
Case 2:

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

```
int f(struct item *p)
{
return((p==Null)|| (p->next==Null)|| ((p->data<=
p->next->data)|| f(p->next)));
}
```

(|| replaced from &&)

f(p) ⇒ 1



- This func^n will return 1 for all non-decreasing or non-increasing elements.