:— "Structures & union" —:

"Structures" :—

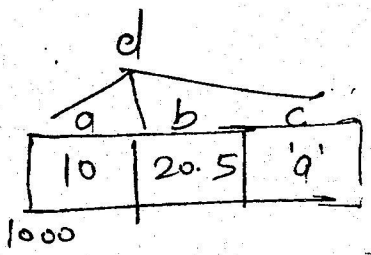
```
main()
{
2. struct node
   {
     int a;   ⇒ 2B
     float b; ⇒ 4B      } 7B
     char c;  ⇒ 1B
   };
```

variable of struct node type

3. struct node d = {10, 20.5, 'a'};

// user defined data type //

```
Pf ("%d", d.a);   10
Pf ("%c", d.c);   'a'
```



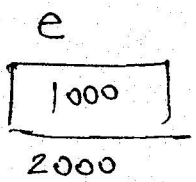
4. struct node *e = &d;

Pf(e) ⇒ 1000

Pf("%d", (*e).a) ⇒ 10 → Inside d.a

or

e → a ⇒ 10



Pf("%c", (*e).c) ⇒ 'a'

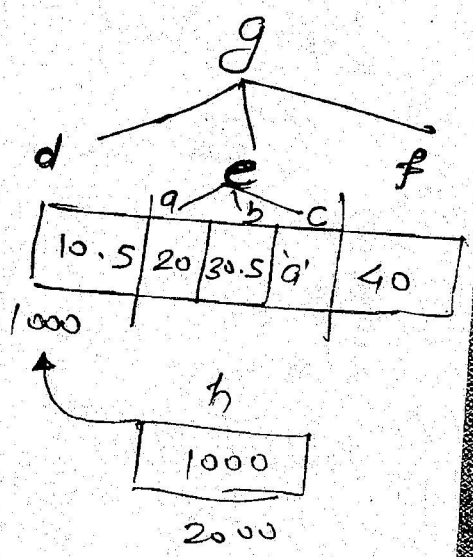
e → c ⇒ 'a'

/* By default w/o '.' operator Pf(d) will print 1st value i.e 10 */

}

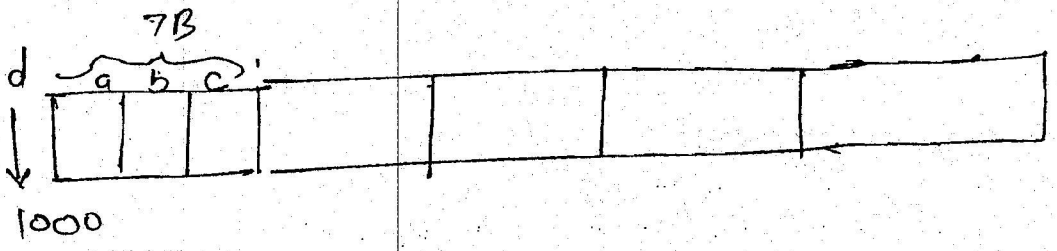
Ex : ② :-

```
main()
{
① struct s1
  {
    int a; ⇒ 2B
    float b; ⇒ 4B      } 7B
    char c; ⇒ 1B
  };
② struct s2
  {
    float d; ⇒ 4B
    struct s1 e; ⇒ 7B  } 13B
    int f; ⇒ 2B
  }
③ struct s2 g = {10.5, {20, 30.5, 'a'}, 40};
  struct s2 *h = &g;
④ pf("%d", (*h).e.a); ⇒ 20
                or
                (h→e).a
  pf("%f", (*h).f); ⇒ 40
                or
                h→f
}
```



Structure Array :-

Struct node d[5] = { 10 - - - - - }



:- union $2

{
float d; ⇒ 4B
Struct $1 e; ⇒ 7B } 7B (maximum size among all)
int f; ⇒ 2B
};

Note :-
In the case of structure, every member having its own memory but in the case of union all members are storing same m/y.

"Type casting" :- [coercion]

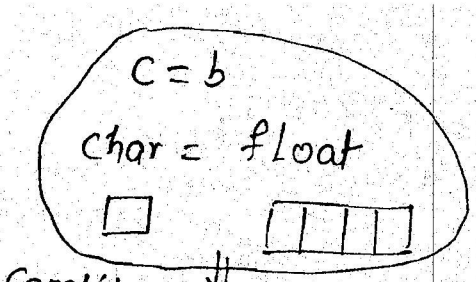
Data

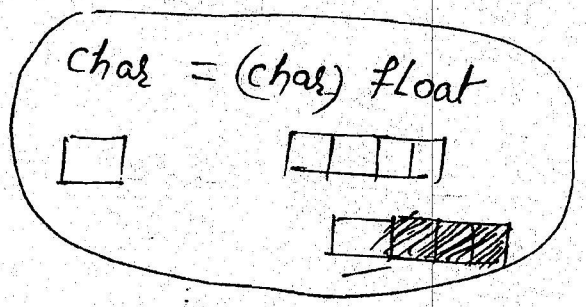
```
int a;
float b;
char c;
```

float = char
b = c

float = (float) char (0's padded)

Implicit Type casting (or) Coercion ← By compiler



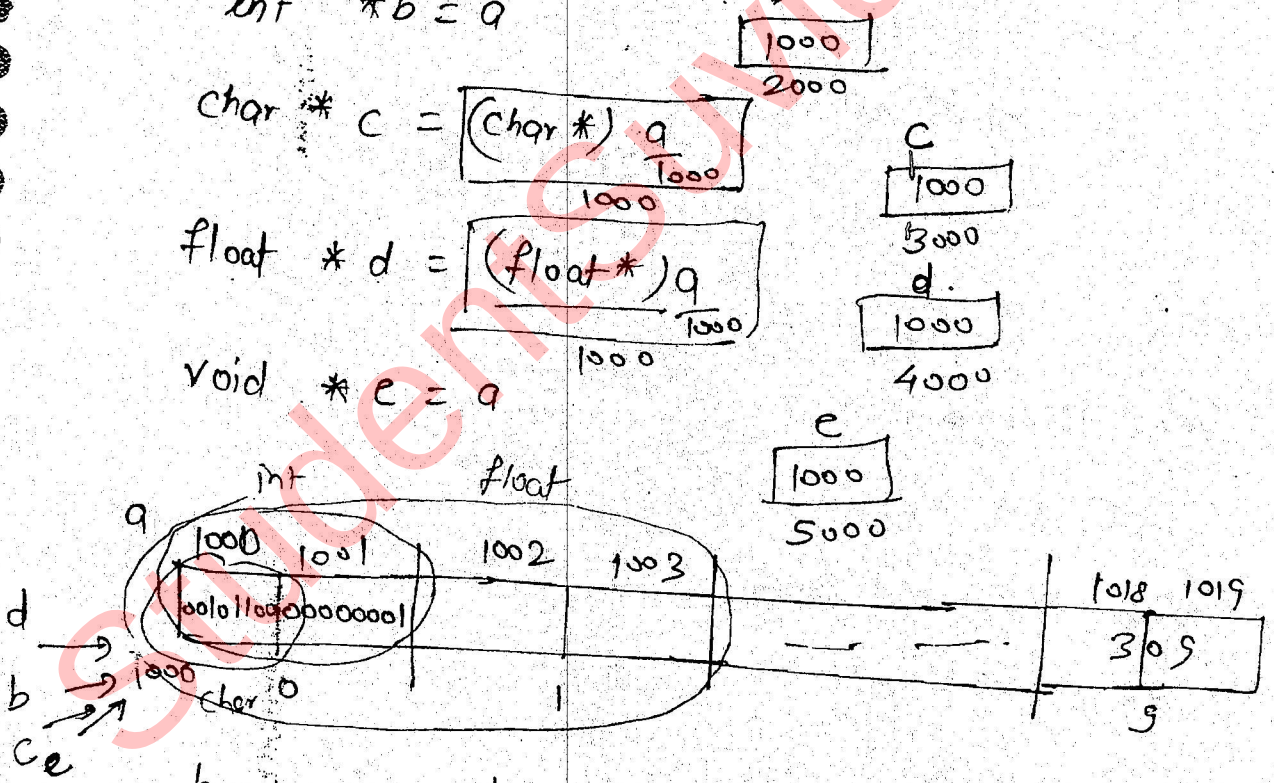


} Explicit type casting (Done by user)

Pointer

Type casting in Pointer ~

```
int a[10] = {300, 301, 302, ---- 309}
int *b = a
char *c = (char*) a
float *d = (float*) a
void *e = a
```



b = 1000

*b = 2B data

300

b+1 ⇒ 1002

c ⇒ 1000

*c ⇒ 1B data

44

c+1 ⇒ 1001

d ⇒ 1000

*d ⇒ 4B data

large

d+1 ⇒ 1004

300 = 00000001 00101100

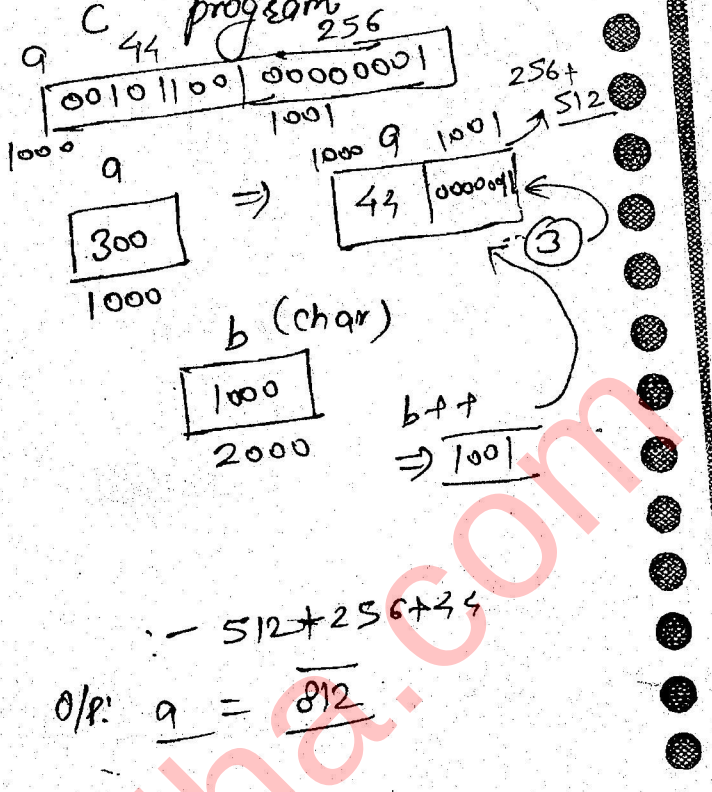
e ⇒ 1000

*e ⇒ error

e+1 ⇒ error

Que :- Consider the following C program

```
main()
{
  int a = 300
  char *b = (char *) &a;
    b++;
    *b = 3;
  Printf("%d", a);
}
```

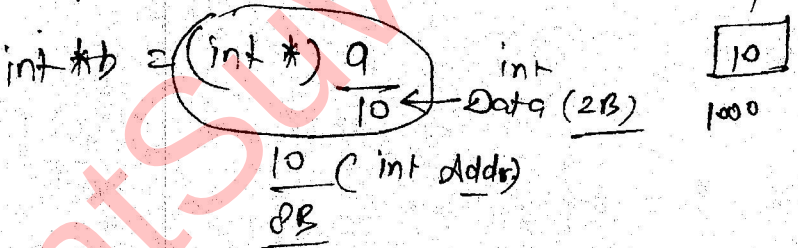


∴ 512+256+44

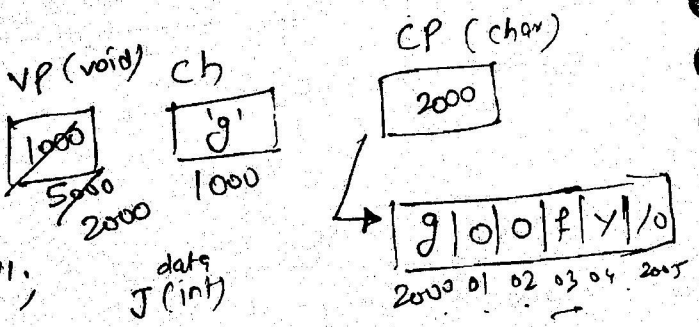
O/P: a = 812

Que :- Consider the following C Program

```
int a = 10
int **b = (int *) a;
```




```
main()
{ void *vp;
  char ch = 'g';
  char *cp = "goofy";
  int J = 20;
  vp = &ch;
  Printf("%c", *(char*)vp);
  vp = &J;
  Printf("%d", *(int*)vp);
  vp = cp;
  Printf("%s", (char*)vp + 3);
```



'g'

20

2000+3 ⇒ 2003

char(vp)

fy

O/p :- g20fy

Date 09/oct/2017

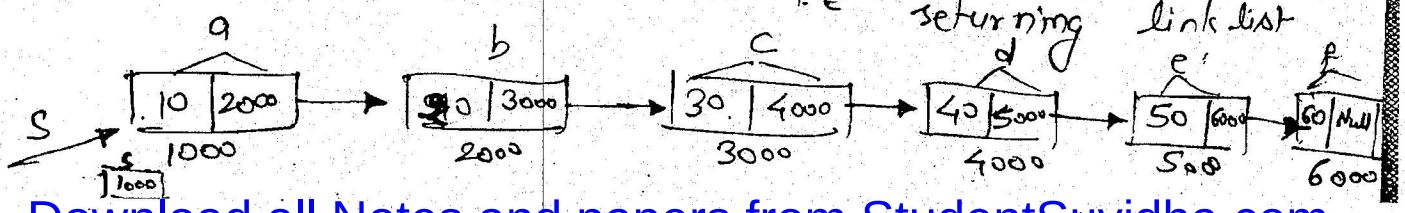
:- Link_List :-

```
struct node* main()          // returning Address of struct node type data */
{
    struct node
    {
        int data;            => 2B
        struct node *next;   => 8B
    };                       } 10B

(2.) struct node a = {10, null};      /* creation of nodes */
     struct node b = {20, null};
     struct node c = {30, null};
     struct node d = {40, null};
     struct node e = {50, null};
     Stuct node f = {60, null};

(3) a.next = &b;                      /* creating Link List & returning link-list */
    b.next = &c;
    c.next = &d;
    d.next = &e;
    e.next = &f;                      /* Linking the nodes */

(4) struct node *s = &a;   // Address of starting node
    return(s);             // returning Addr. of 1st node i.e returning link list
}
```



Previous Program will create link list & return it.

*/ malloc () will create m/y in "heap" area in runtime. */

Above program link-list nodes will be allocated in main(), stack area so after end of main() link-list nodes will be popped out so link-list will be gone.

Que write a C program to return the address of last node in the given link-list.

Struct node *s = &a; /* s=1000 */ given

```
Struct node * findlastnodeaddress ( Struct node *s)
{
① if ( s == null)        /* If not Present segmentation error */
      return (s);
② while ( s -> next != "null")
  [ {
      s = s -> next;
  } ]
  return (s);
}
```

O(n) every case

Que :- write a C Program to return Address of 2nd last node in the given link list.

```
Struct node * secondlastnodeAddr ( Structnode *s)
{   Struct node *P;
①  If (s==null)
       return (s);
②  If (s->next ==null)
       return (null);
③  while (s->next !=null)
   {
       P=s;
       s=s->next;
   }
   return (P);
}
```

(or)

```
Struct node * secondlast node addr.( Structnode *s)
{
①  If (s==null)
       return(s);
②  If (s->next ==null)
       return (null);
③  while (s->next->next!= null)
   {
       s=s->next;
   }
   return(s);
}
```

Que :- Consider the following C Program

i) struct node *P ;

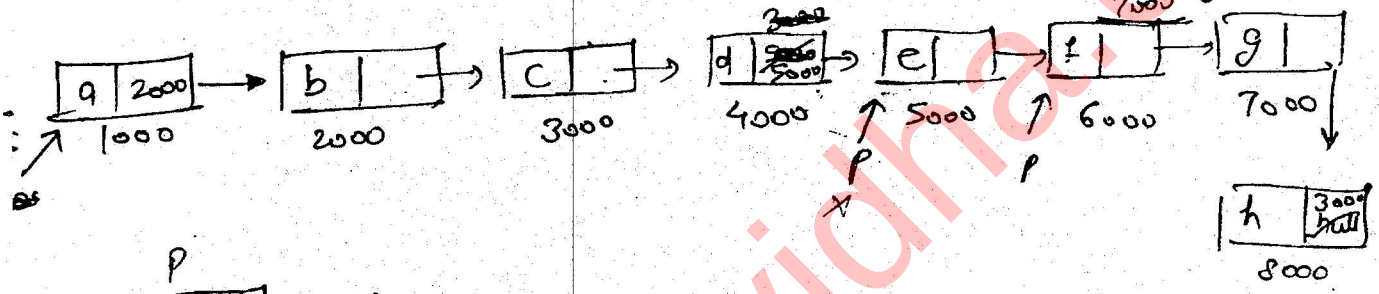
ii) P = S → next → next → next → next ;
1000 2000 3000 4000 5000

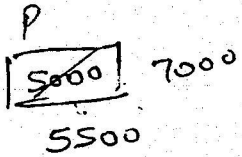
P = 5000

iii) P → next → next → next → next = S → next → next ;
5000 6000 7000 8000 null 1000 2000 3000

iv) P = P → next → next → next → next → next → next → next → next ;
5000 6000 7000 8000 3000 4000 5000 6000 7000

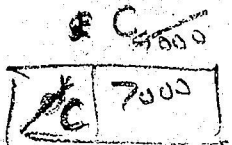
O/P:- g v) Pf(P → next → next → next → next → next → next → data) ;
7000 8000 3000 4000 5000 6000 7000



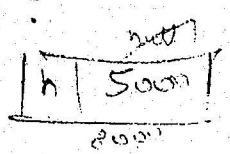


Que. what will be the output If you execute above C Program on given link-list.

Pf(7000 → data) = g



Que :- Consider the following C Program.



i) struct node *P;

ii) P = S → next → next → next → next → next ;
1000 2000 3000 4000 5000

iii) P → next → next → next = S → next → next → next → next ;
6000 7000 8000 null 1000 2000 3000 4000 5000

O/P :- e iv) S → next → next → next = P → next → next → next → next → next ;
1000 2000 3000 4000 6000 7000 8000 5000 6000

v) P = S → next → next → next → next → next → next → next → next ;
1000 2000 3000 4000 5000 6000 7000 8000 5000

O/P :- h vi) Pf(P → next → next → next → next → next → next → data) ;

Eg:-

```
f( struct node * P)
{
return ((P==null) || (P->next==null) || ((P->data <= P->next->data) && f(P->next)))
}
```

termination conditions for recursion

The above function on the given link-list 'P' always return '1' then P should be

a) empty
b) 1 element ✓
c) the data in P is in ascending order ✓
d) data in P is in descending order.

c is Answer. ∴ option a & b are also in 'c'

1 = f(P) (Recursion) option (c) ✓

0 || 0 || ((10 ≤ 20) && f(P))

0 || 0 || (20 ≤ 30 && f(P))

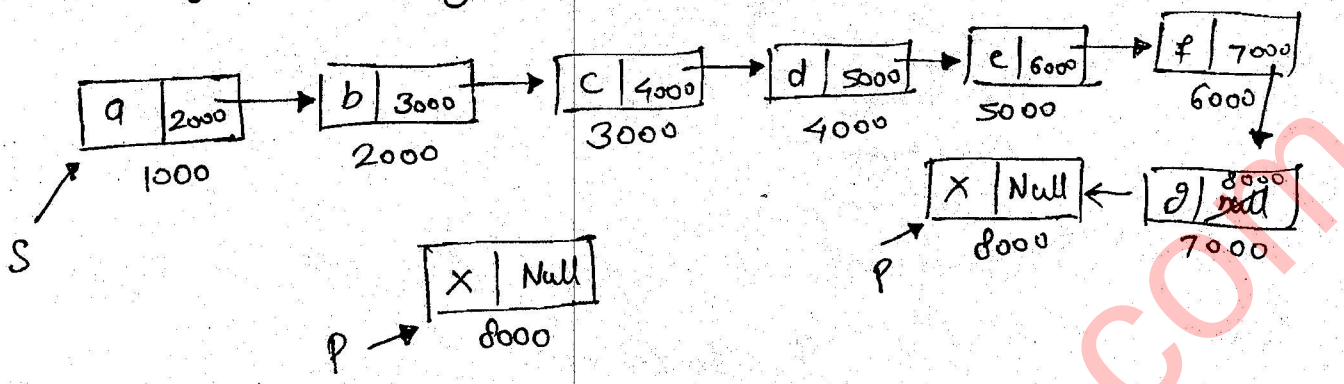
0 || 1 ⇒ 1

ii)

```
f( struct node * P)
{
return ((P==null) || (P->next==null) || ((P->data <= P->next->data) || f(P->next)))
}
```

:- all (or) operator. so o/p :- always '1'

Que :- write a C program to add a Node with data 'x' at the End of the given single link-list.




```
Insert_x_at_end ( struct node *S, int x)
{
i) struct node *P;
   P = (struct node *) malloc ( size of (struct node) );
       /* type casting */
```

8000 ← Void Pointer

Node Creation with data 'x'.

```
   (*P).data = x  (or)  P -> data = x
   (*P).next = null  (or)  P -> next = null
ii) If (S == Null)
      { S = P
        return (S) }
iii) S1 = S;
```

iii) $s_1 = s$

```
while (s1 → next != null)
{
    s1 = s1 → null;
}
```

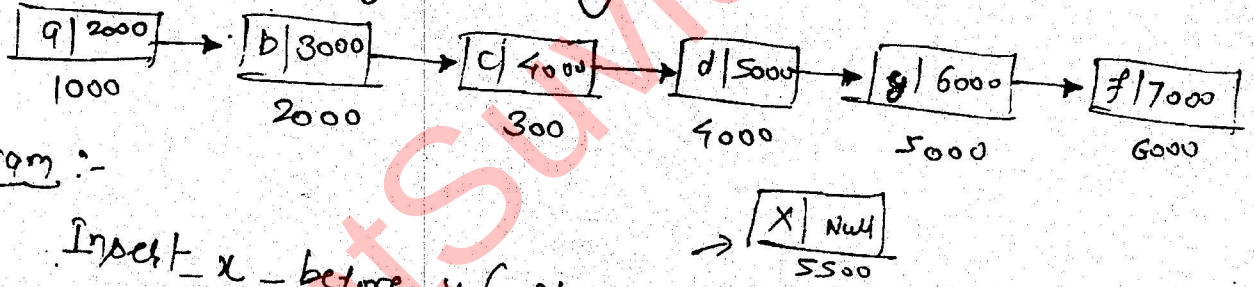
↑ O(n) { every case }
↓

iv) `s1 → next = p` /* linking P (Node) at last */

`return (s);`

}

Que:- Write a 'C' Program to add a Node with data 'x' before a node with data 'y' in the given Single link-list.



Program :-

```
Insert_x_before_y (struct node *s, int x, int y)
{
①   struct node *p;
    p = (struct node *) malloc (sizeof (struct node));
②   if (s == null)
    empty
    return;
    (*p).data = x  (or)  p → data = x
③   (*p).next = Null  (or)  p → next = Null
    while (s → data != y && s → next != null)
    {   q = s;
        s = s → next;
    }
```

```
if (s -> data == y)
    { q -> next = p
      p -> next = s
    }
else
Pf ("No 'y' so can't add 'x");
}
```

memory leak :-

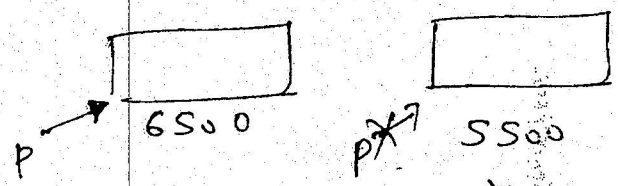
P = (struct node *) malloc (size of (struct node))
5500

5500

P → 5500

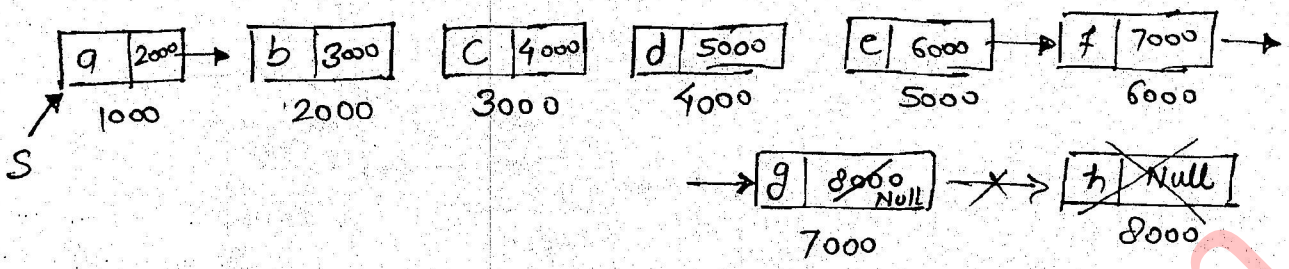
P = (struct node *) malloc (size of (struct node))
6500

6500

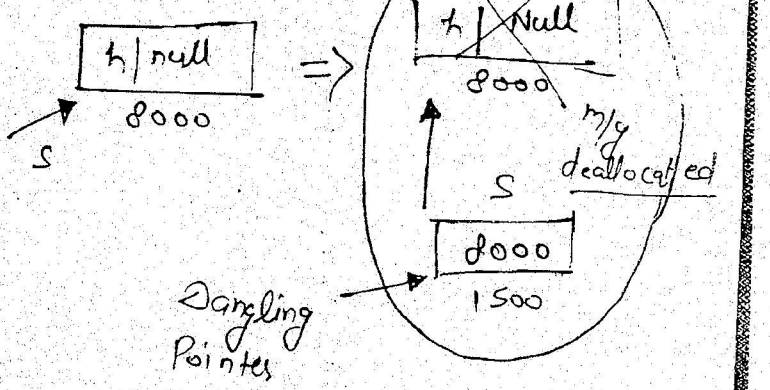


Operating system thinking still using 5500, this is called as "memory Leak."

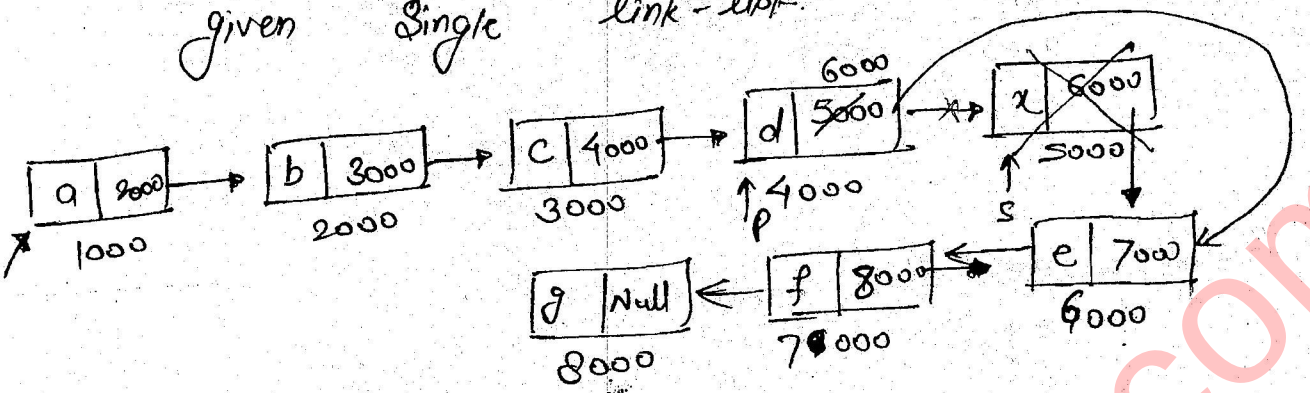
Que :- write a 'C' program to delete last node of the given link-list.




```
Delete_at_end ( struct node *s)
{
① Empty
② while ( s→next != null)
  {
    P = S
    S = S→next;
  }
③ P→next = Null;
/* At this Point
   S is pointing to 8000
   which is allocated to you,
   " Valid Pointer "
   Pf(S) = 8000
   Pf (S→data) = your value */
free (S);
S = Null;
/* At this Point
   Pf(S) = 8000
   Pf(S→data) = garbage }
   S is pointing to 8000
   which is deallocated.
   " Dangling Pointer " */
```

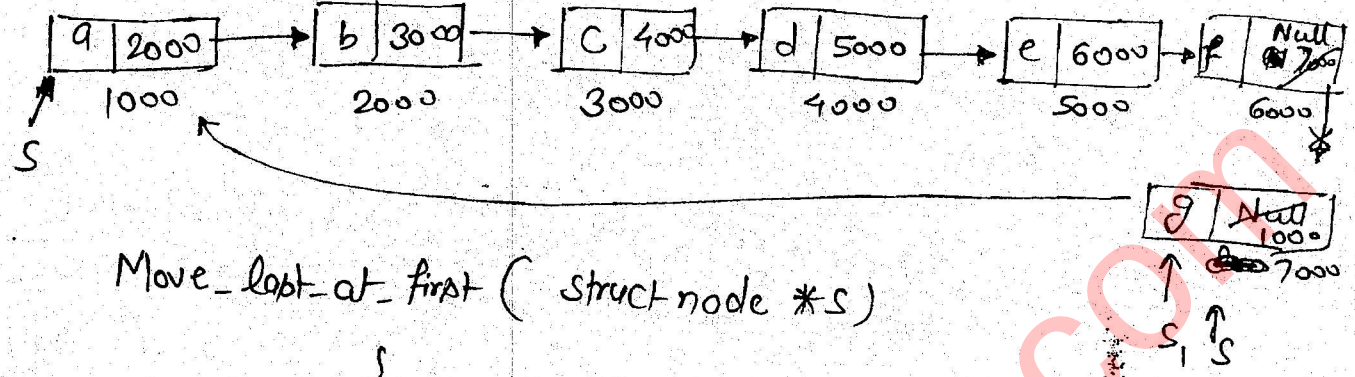


Que:- write a 'C' program to delete a node which contain data 'x' in the given Single link-list.




```
Delete_x ( Struct node *s, x)
{
① Empty
② while ( s→data != x && s→next != Null)
  {
    p = s;
    s = s→next;
  }
③ if ( s→data == x)
  { p→next = s→next;
    free (s);
    s = Null;
  }
}
```

Que :- write a C program to move last node of the link-list to the front of the link-list.




```
Move_last_at_first ( struct node *s)
{
① Empty
② If (s→next == null)
   {
    return (s);
   }
③ Struct node *s1; s1 = s;
  while (s1→next != null)
  {
    p = s1;
    s1 = s1→next;
  }
  i) p→next = null;
  ii) *s1→next = s
                 or
      s1→next = s;
  iii) s = s1
}
```

In same order