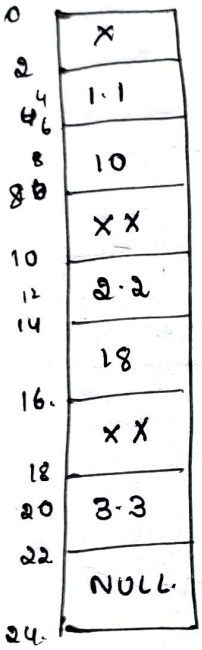
M-3

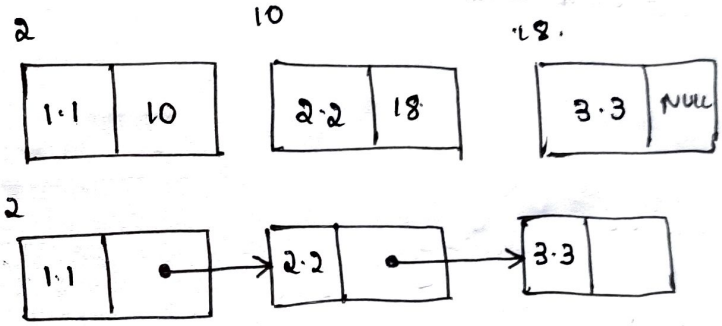
→ Linked Lists:

float (1.1) 4 + 2 → to store the address of next memory allocated.

info → data.

link → refers next data location / NULL.





→ Non sequential collection of elements.

Disadvantage:

→ for x bytes it takes x+2 bytes.

→ What is fragmentation? → internal
→ external.

```
struct LL
{
 float cgpa;
 struct LL* link;
};
```

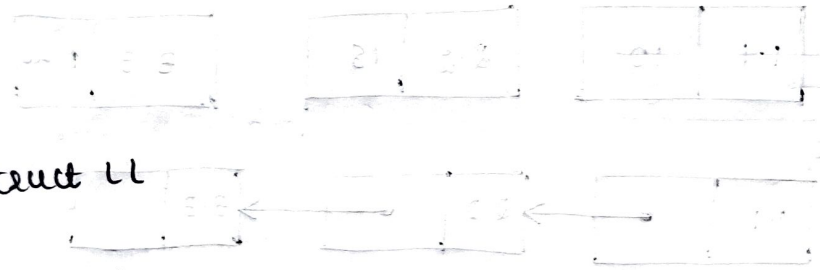
struct LL * getnode.

getnode is used to reserve room of specified bytes.

~~struct~~ OR

```
typedef struct LL
{
 float cgpa;
 struct LL* link;
};
```

typedef struct LL



```
struct LL
{
 float c;
 struct LL * link;
};
typedef struct LL Node;
```

```
typedef struct LL Node;
Node * getnode( )
{
```

```
Node * getnode()
{       float m;
Node * ptr;
ptr = (Node *) malloc (size of (Node)).
printf("      ");
scanf("%f", &m);

ptr -> c = m;
ptr -> link = NULL;
return ptr;
}
```

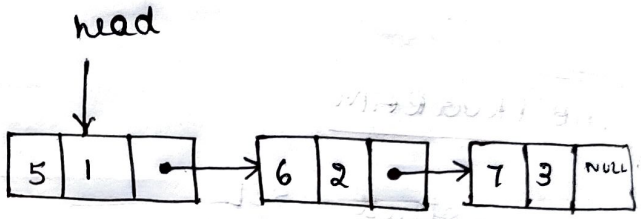
GLOBAL DECLARATION:

```
Node *first = NULL;
```

LAB PROGRAM

```
struct SLL
{ float avg;
  int sum;
  struct SLL * Ae link;
};
typedef struct SLL Node.;
Node * head = NULL;
Node * getnode()
{       float m; int s;
  Node *ptr;
  ptr = (Node *) malloc (size of (Node));
  printf("Enter m & sum\n");
  scanf("%f %d", &m, &s);
```

```
    ptr -> avg = m;
    ptr -> sem = s;
    ptr -> link = NULL;
  } return ptr;

void create( )
{
  Node *temp;
  temp = getnode( );
  if(head == NULL)
      head = temp;
  else
    temp -> link = head;
    head = temp;
  }
```



Display

```
void display( )
{   int c=0;
  Node *p;
  p=head;
  if(head == NULL)
   {printf("No records to display");
    exit(0);
   }
  while(p != NULL)
   {  c++;
      printf("Student %d details\n"); : Marks = %f Sem = %d
        \n", c, p -> avg, p -> sem);
        p = p -> link;
   }}
```

```
void inef()
{
 create();
}

void insr()
 {
  Node *temp, p;
  temp = getnode();
  if (head == NULL)
    head = temp;
  else
    { while (p->link != NULL).
       {
         p = p -> link;
       }
      p -> link = temp;
    }
  }

void delf()
  { Node *temp;
   if(head == NULL)
   exit(0);
   else
     { temp = head;
       head = head -> link;
     }
   } free(temp);
```

I.

bl = NULL

p = b

II

cl = NULL.

p = c;

print & delete,

if Prakyath sir comes to lab.

diagram.

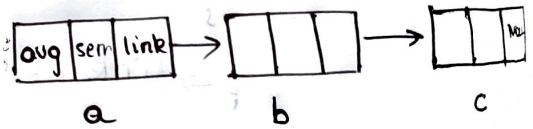
```
void dell()
{ Node *p, *temp = head
  if(head == NULL)
  exit(0);
  else
  {
    while(p -> link != NULL)
    {
      p = p -> link;
    }   temp = p -> link;
    p -> link = NULL;
    free(temp);
  }.
```

I
p, a
b != NULL.
p = b

II.
c != NULL.
p = c

III.
NULL != NULL,

stack { push() / pop() } { inf, delf / inq, dulq. } → single DL. compul. → D.LL

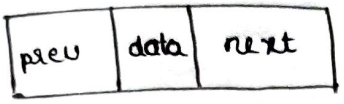


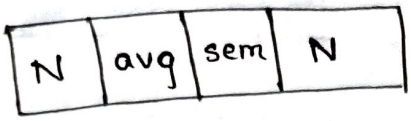
queue [inf / dulq] or [inq / delf]

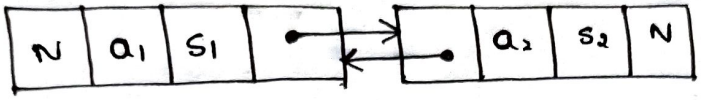
delete rear

```
    else if(head -> link == NULL)
    {
      head = NULL;
    }
  }
```

Doubly Linked Lists:







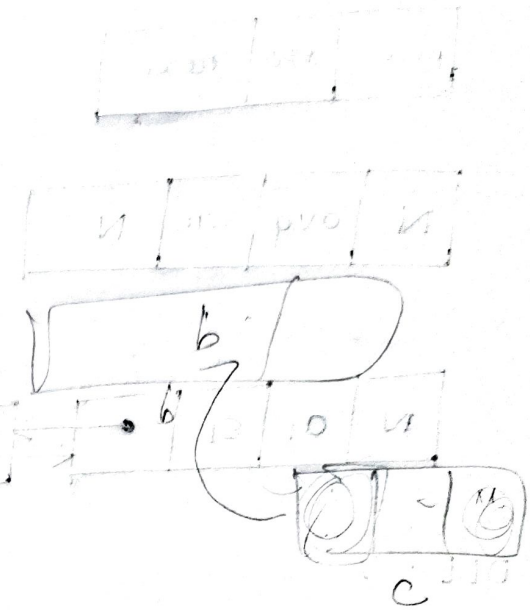

```
struct DLL
 {
  float avg;
  int sem;
  struct DLL *next, *prev;
 };
 typedef struct DLL Node;
  Node *head = NULL;

  Node *getnode()
  {
    float m, int s;
    Node *ptr;
    ptr = (Node *) malloc (sizeof (Node));
    // read
    ptr -> avg = m;
    ptr -> sem = s;
    ptr -> prev = NULL;
    return ptr;
  }
```

Node *getnode ( )

i) case :

head = NULL

```
void create()
{
  Node *t;
  t = getnode();
  if(head == NULL)
    { head = t;
    }
else { while(p -> next != NULL)
    {
      p = p -> next;
    }
      p -> next = t;
      t -> prev = p;
    }
}

void ins()
 {
  create();
 }
 void ins()
 {
   Node *t;
    t = getnode()
     if(head == NULL)
       {
        head = t;
       }
     else
       { head -> prev = t;
```



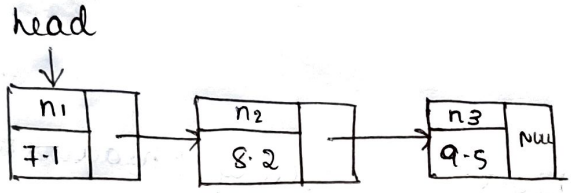
```
t -> next = head;
head = t;  // head = head -> prev.
}
}

void delf()
{ Node *t;
 if(head == NULL)
   exit(0);
 else
   t = head;
   head = head -> next;
   free(t);
 }
}

void delr()
{
  Node *t; *p = head;
  if(head == NULL)
    exit(0)
  else { while(p -> next != NULL)
         { p = p -> next;
         }
         p -> prev = NULL; -> next = NULL;
         p -> prev = NULL;
         free(p);
       }
```

⇒ head -> prev == NULL;

```
else if(head -> next == NULL)
{ head = NULL;
}
```

```
void display()
{
  Node *t; *p=head;
  if(head == NULL)
    exit(0);
  else
    while(p != NULL)
    {
    c++;
    //print
    p=p->next;
  }
```

*



key 8.2.

```
struct SLL
{
 char name[25];
 float cgpa;
 struct SLL *link;
};
typedef struct SLL Node;

 void read()
 {
```

```
char n; float c; char ch;
{ do
  {
    printf("    ");   //name & cgpa.
    scanf("%f %f", n,&c);
    getnode(&n,c);
if(head == NULL)
{
 head = ptr;
}
else
{
 while(p->link != NULL)
 {
   p = p->link;
 }
   p->link = ptr;
}
print// y to c & N to S).
} while(ch == y);
```

```
void getnode(char nm[], float f)
{
 Node *ptr; *p = head;
 ptr = (Node *) malloc(sizeof
 (Node));
 ptr -> name = nm;
 ptr -> cgpa = f;
 ptr -> link = NULL;
}
```

pseudocode to search

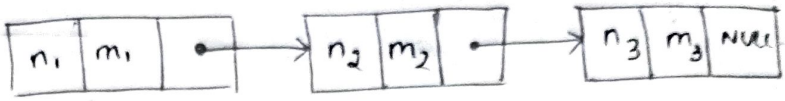
```
(head == NULL).
  Key not found.

else
 while(p != NULL).
 {
 if(p->cgpa == key)
 // key found;
   break;
 p = p -> link;
```

without header nodes:

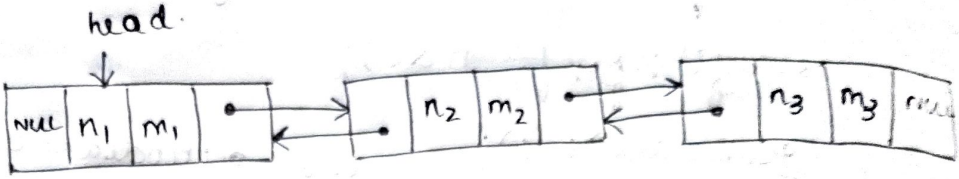
student

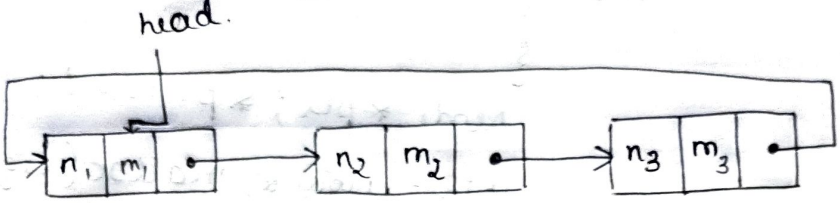
name cgpa

SLL:



DLL

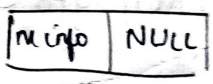




CLL.

header node

```
struct SLL
{
 char n[];
 float m;
 struct SLL *link;
};

struct header
{
  int count;
  float sum, avg;
  struct SLL *next;
};
```

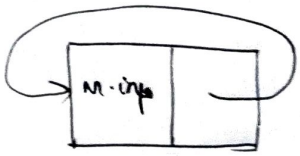


CHN

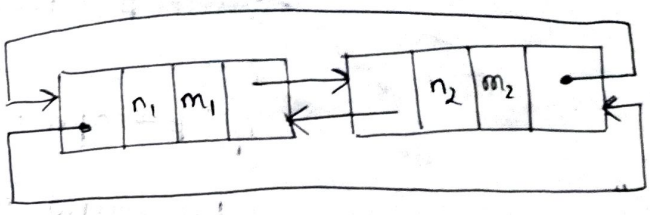
↓

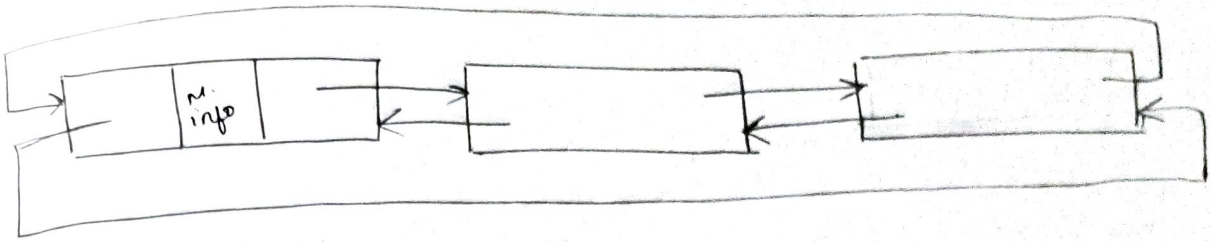
only for SLL/DLL.

for circular:

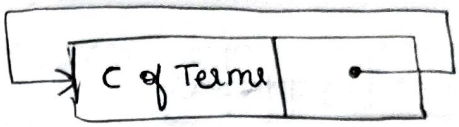


circular Doubly linked lists:





SCLL with header node:




```
struct SCLL
{
 float cf;
 int px, py, pz;
 struct SCLL * next;
};

struct HN
 {
  int ct;
  struct SLL * next;
 };
```

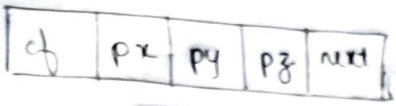
→ In circular list, HN should be same as that of nodes.

→ HN can be of different types but in CL it should be of same type.

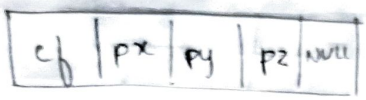
→ problem: last node will point to the HN, which is of different type.

~~Node * head.~~

```
Node * getnode()
{
 Node * ptr;
 ptr = (Node *) malloc (sizeof (Node));
 if (ptr == NULL)
  { standard error:
```




```
ptr = NULL;
return ptr;
}
```




```
Node * head = getnode();
head → next = head;
```

Mathematics in adding 2 polynomials:

$p_1 = 2x^2yz + 2xy^3z + xyz$

$p_2 = 2xy^2z - 2x^2yz + 2xz$

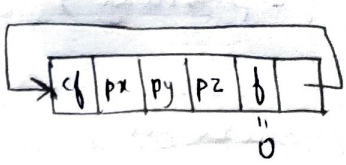
flag = 1.

ans $-2xy^3z + 3xyz + 2xy^2z$

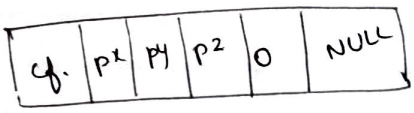
x=1
y=1
z=0
0-0+1

```
struct SCLL
{
  float cf;
  int px, py, pz, flag;
  struct SLL * next;
};
```

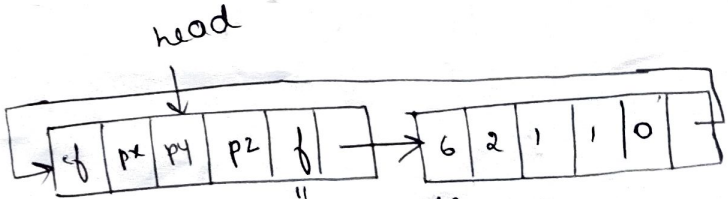
```
Node * create (Node * head)
{   float c; int x,y,z, ch;
  do {
     printf("          ");
     scanf("%f %d %d %d", &c, &x, &y, &z);
head = ins(head, c, x, y, z);
printf("          ");
scanf("%d", &ch);
} while (ch);
return head
};
```



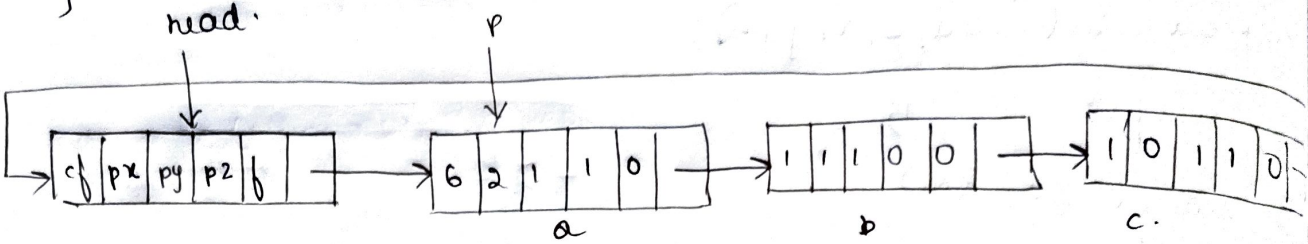

```
Node * ins (Node * head, float c, int x, int y, int z)
{
  Node *t, *p;
  t = getnode();
  t→cf = c;
  t→px = x;
  t→py = y;
  t→pz = z;
  if (head → next == head)
    {
      head → next = t;
      t → next = head;
    }
  else
    { p = head → next;
      while (p → next != head)
```



ptr
t



```
{ p = p->next;
{
p->next = t;
t->next = head;
} return head;
```




```
void display (Node *head)
{ Node *p;
 if(head->next == head)
  {
  }
 else
  {
   p = head->next;
   while(p != head)
   {
    printf("%f %d. x^%d y^%d z^%d  ", p->cf, p->px, p->py, p->pz);
if(p->next != head)
    printf("+ ");
    p = p->next;
   }
  }
```

Tracing

p=a	p=b	p=c	p=h
I	II	III	h != h
a != h.	b != h	c != h	X.

6x^2y^1z^1 + 1x^1y^1z^0 + -1x^0y^1z^1.

$6x^2y^1z + xy - yz$

6 2 1 1	1 1 1 0	-1 0 1 1

6 * pow(x, px)

```
void evaluate(Node *head)
{       float x,y,z; sum=0;
  Node *P;
  if(head->next == head)
  {

  }
  else
  { p = head->next;
    printf("      ");
    scanf("%f%f%f", &x, &y, &z);
    while(p != head)
    { sum = sum + p->cf * pow(x, p->px) * pow(y, p->py) *
        pow(z, p->pz);
      p = p->next;
    }
  }
```

x=1 y=1 z=1 s=0 p=a

sum = 0 + 6 * x^2 * y^1 * z^1

p=b.

II

sum = 0 + $6x^2y^1z^1$ + 1 * $x^1y^1z^0$

$p_1 = 6x^2yz - xy + yz$

$p_2 = xyz + xy + 2yz + z$

$6x^2yz + 3yz + xyz + z$

6211 3011 1111 1001

```
Node *pa(Node *p1, Node *p2)
{
  Node *rp = getnode();
  rp->next = rp;
  p1 = h1->next;
  while(p1 != h1)
  {
    p2 = h2->next;
    while(p2 != h2)
    {
      if(p1->px == p2->px && p1->py == p2->py && p1->pz ==
        p2->pz)
        break;
    }
    if(p2 != h2)
    {
      rp = inv(rp, p2->f = 1);
      if((p1->f + p2->f))
        rp = inv(rp, (p1->f + p2->f), p1->px, p2->p
          p->pz);
```

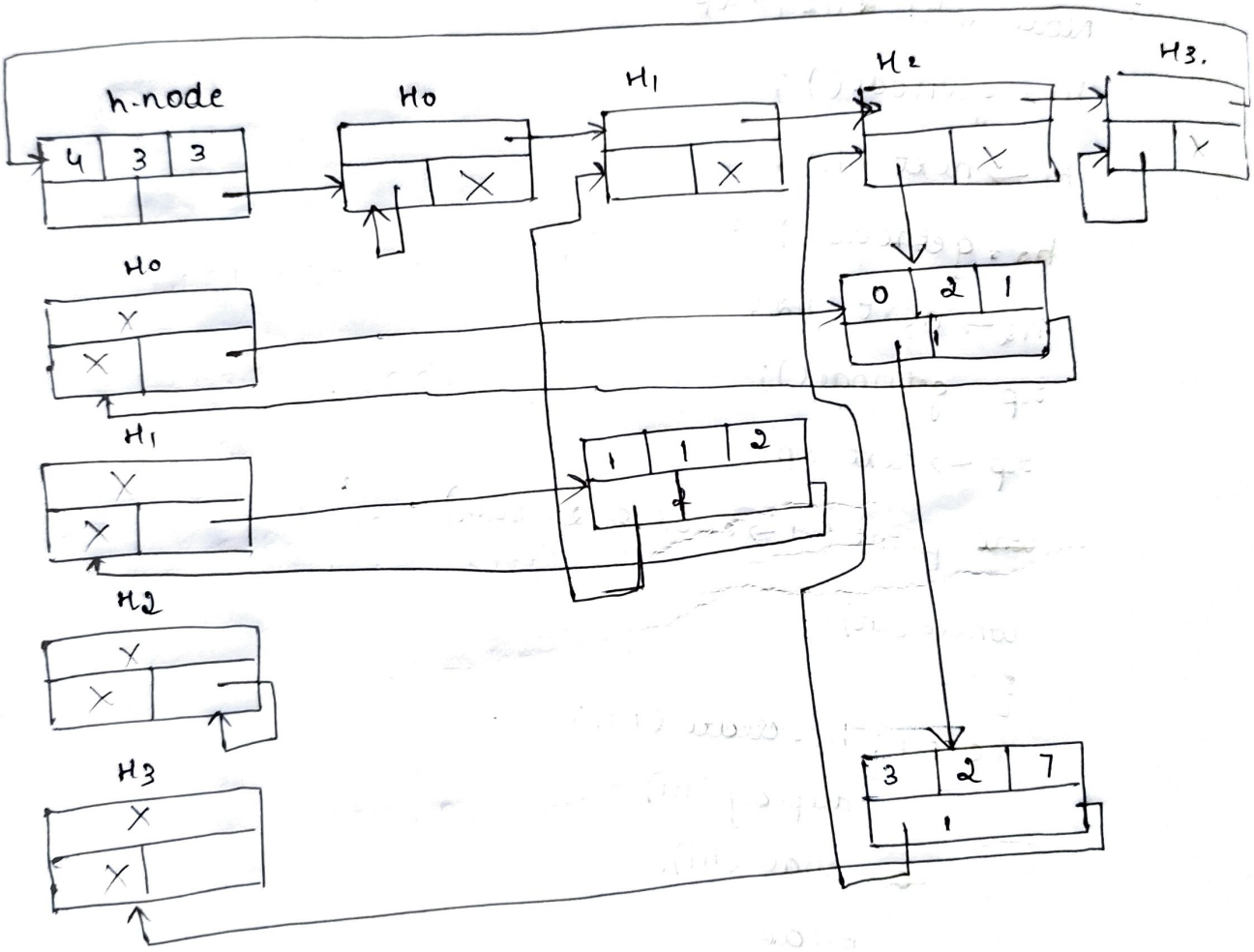
```
else
{
rp = insr(rp, p1->cf, p1->px, p1->py, p1->pz); p1=p1->next;
}
p2 = h2->next;
while(p2 != h2)
{
if(p2->f != 1)
{
rp = insr(rp, p2->cf, p2->px, p2->py, p2->pz);
}
p2->f = 1; p1 = p1->next; p2->f = 1;
}
p2 = h2->next;
```

```
main()
{
Node *h1, *h2, *rp;
h1 = getnode();
h1->next = h1;
h2 = getnode();
h2->next = h2;
rp = getnode();
rp->next = rp;
print(-1->eval & 2 = sum)
while(ch)
{
case 1: h1 = create(h1);
display(h1)
eval(h1).
break;
case 2: h1 = create(h1);
h2 = create(h2);
rp = addpoly(h1, h2, rp)
```

```
        display (&p);
        break;
    default:
}
```

Sparse Matrix Representation

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 7 \end{bmatrix} \qquad \begin{matrix} 4 & 3 & 3 \end{matrix} \begin{bmatrix} 0 & 2 & 1 \\ 1 & 1 & 2 \\ 3 & 2 & 7 \end{bmatrix}$$

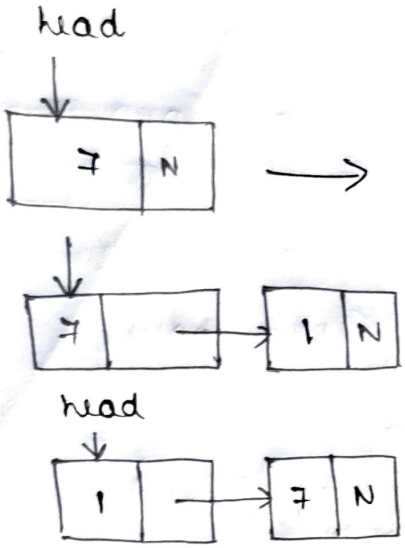


```
typedef struct
{ int r;
  int c;
  int v;
} entrynode;

typedef struct
{ int r;
  int c;
  int v;
} entrynode;

typedef struct
{
 struct sm * left;
 struct sm * down;

 union
  {
   struct sm * next;
   entrynode n;
  } u;
 } sm;

typedef enum{ head, entry } e;
```

Total no of nodes :

max(r, c) + 1 + non zero terms.

To reverse a singly linked list. Imp




```
typedef struct SLL Node;

struct SLL
{
  int v;
  struct SLL * link;
}
```

```
Node *rev (Node *head)
{   Node *prev = NULL, *next = NULL, *p;
 if (head == NULL)
  {
   exit(0);
  }
 if (head -> link == NULL)
  {
   return head;
  }
 else
  { p = head;
    while (p != NULL)
     {
       next = p -> link;
       p -> link = prev;
       prev = p;
       p = next;
     }
    head = prev;
```

→ To combine marks & arrange them in descending order.

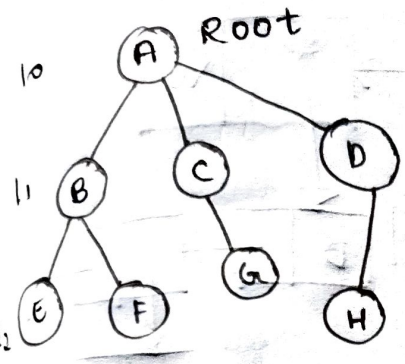
```
struct SLL
{
 char n[ ];
 float m;
 struct SLL *next;
};
float max( node *head)
{ node *p= head, max;
   if(head == NULL)
   {//exit (0);
   }
   max = p → m;
   while(p != NULL)
   { p=p→next;
     if(p → m > max)
     {max = p → m;
     }
   } p = p → next;
  }
```

→ to find average:

```
float avg(Node *head)
{
  int c=0; float sum=0;
  if(head==NULL)
  {
    //exit(0);
  }
  while(p!=NULL)
  {
    sum = sum+p->n;
  }
  avg = sum/c;
}.
```

MOD - 4

TREES.



Degree (A) = 3.

Degree (B) = 2.

Degree (C) = 1

Degree (E) = 0.

Leaf Nodes : E, F, G, H.

Subnodes are called as Siblings.

Ancestors of E is A & B.

The no of paths from depth leaf is height of a tree. ↓

Height : 2

Depth : how deep an element is from the root. ↑

Representation of a Tree:

→ List

→ Left child Right sibling

→ Degree - 2.

1) LIST REPRESENTATION: