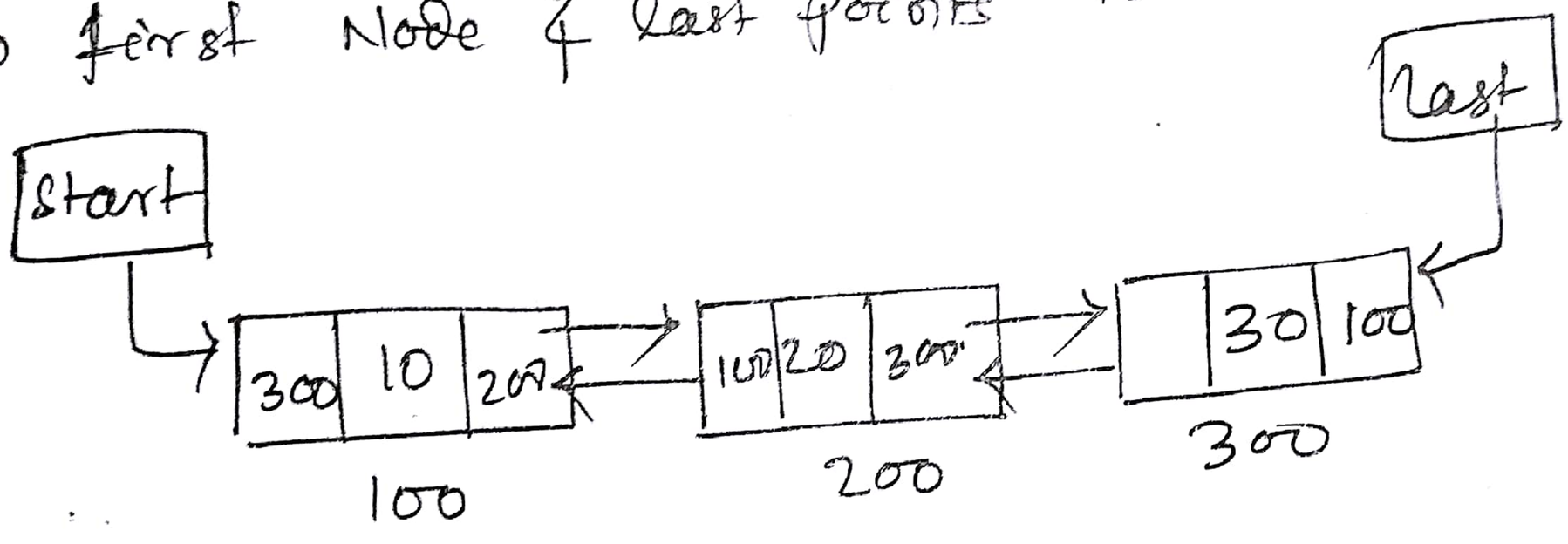
CIRCULAR DOUBLE LINKED LIST:

- circular double linked list node has 3 parts previous, info or data, next.
- out of 3, two are link fields & one is data field
- in circular the previous field of first Node holds the address of last node.
- Like that next fields of last node holds the address of first Node.
- Two external pointer start & last present, start points to first Node & last points to last Node.



* Insertion:

1) Insertion at begin:

```
void dll_ins_beg()
 { node *ptr;
   ptr = (node *) malloc(sizeof(node));
  if(ptr == NULL)
  { printf("Memory Not allocated");
    getch();
    exit(0);
  }
else
  { printf("Enter element");
    scanf("%d", &ptr->info);
   if(start == NULL)
     {ptr->next = ptr;
      ptr->prev = ptr;
```

```
        last = ptr;
        start = ptr;
    }
else
    {
        ptr->next = start;
        ptr->prev = last;
        start->prev = ptr;
        start = ptr;
        last->next = ptr;
    }
}
```

2) Insertion at end :-

```
void cdl_ins_end()
{
node *ptr;
ptr = (node *) malloc(sizeof(node));
if(ptr == NULL)
  {
   printf("Memory Not allocated");
    getch();
    exit(0);
  }
else
  {
printf("Enter element");
scanf("%d", &ptr->info);
if(start == NULL)
   {
ptr->next = ptr;
ptr->prev = ptr;
start = ptr;
last = ptr;
   }
```

```
else
{
ptr->next = start;
ptr->prev = last;
last->next = ptr;
last = ptr;
start->prev = ptr;
}

}
```

```
}
```

deletion :-

① deletion at begin :-

```
void edl_del_beg()
{
node *ptr;
ptr = start;
if(start == NULL)
{
pf("list is empty");
getch();
exit(0);
}
else
{
if(start->next == ptr)
{
start = NULL;
last = NULL;
}
else
{
(ptr->next)->prev = last;
last->next = ptr->next;
start = ptr->next;
}
free(ptr);
}
```

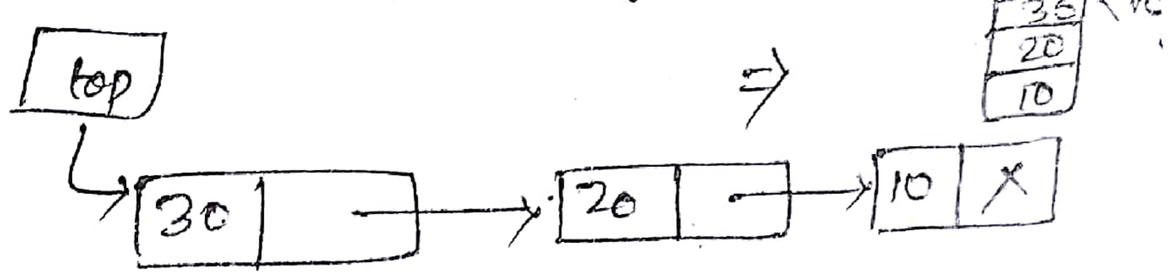
2) deletion at end :-

```
void  cdl_del_end()
{
 node *ptr;
   ptr = last;
   if(last == NULL)
     {
      printf("list is empty");
      getch();
      exit(0);
     }
else
   {
    if(last->prev == ptr)
      {
        start = NULL;
        last = NULL;
      }
   else
     {
      last->next = start;
     (last->prev)->next = start;
      start->prev = last->prev;
       last = last->prev;
     }
free(ptr);
}
```

Representing a stack using a linked list →

Disadvantage of array based stack :-

1) size of stack must be known in advance
2) There is situation of overflow arises which is not beneficial
3) Representation stack in an array prohibits growth of stack beyond the finite number of elements.



Menu driven prog :-

```
typedef struct NODE
{
int info;
struct NODE *next;
}node;
node *top=NULL;
void push(int);
int pop();
void display();
void main()
{
int ch,ele,e;
while(1)
{
printf("1. Push\n");
printf("2. pop\n");
printf("3. Display\n");
printf("4. Exit\n");
printf("Enter your choice");
scanf("%d",&ch);
switch()
{
case 1:
printf("Enter element");
scanf("%d",&ele);
push(ele);
break;
case 2: e=pop();
printf("Element deleted is %d",e);
break;
case 3: display();
break;
case 4: exit(0);
default:
printf("Invalid choice");
}
}
getch();
}
```

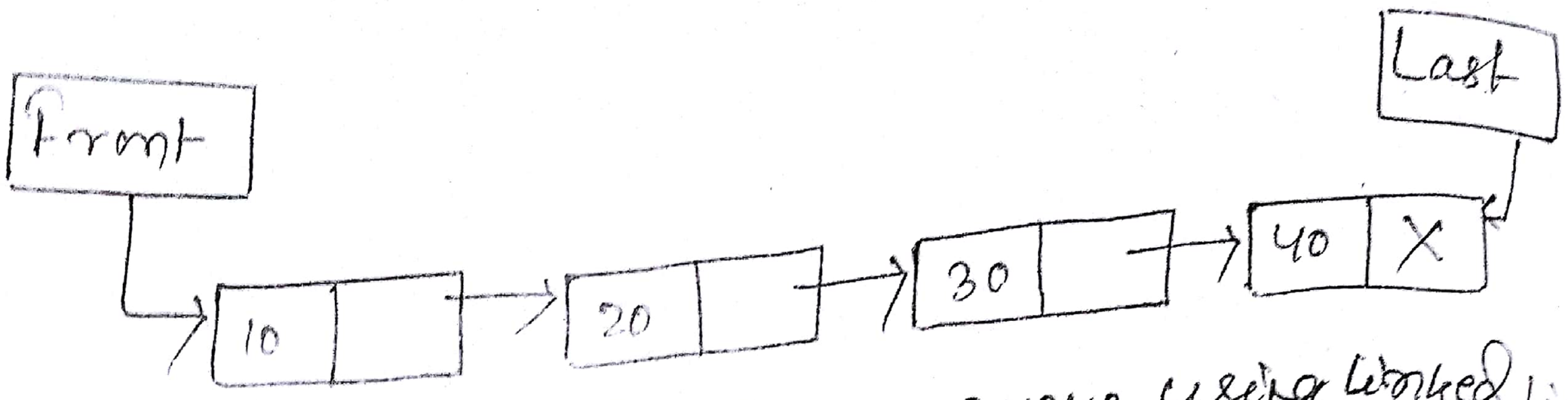
77

```
void push(int ele)
{
  node *ptr;
  ptr = (node *) malloc(sizeof(node));
  if(ptr == NULL)
  {
   printf("memory full");
    getch();
    exit(0);
  }
 else
  {
    ptr->info = ele;
    ptr->next = top;
    top = ptr;
  }
}
int pop()
{ int ele;
  node *ptr;
  ptr = top;
  if(top == NULL)
  {
   printf("List is empty");
    getch();
    exit(0);
  }
else
  {
  ele = ptr->info;
  top = ptr->next;
  free(ptr);
  }
}
```

```
void display()
{
 node *ptr;
  if(top == NULL)
   {printf("List is empty");
     getch();
     exit(0);
   }
  else
   {ptr = top;
    while(ptr != NULL)
    {printf("%d", ptr->info);
     ptr = ptr->next;
    }
  }
}
```

Linked Representation of Queue :

- One major drawback of representing a queue by using array is that a fixed amount of storage remains allocated even when the queue structure is actually using a small amount or possibly no storage at all.
- There is no fixed amount of storage allocation is done in case of linked list representation so there is not any possibility of overflow.
- Each node is divided into two fields.
  i) info field holds data element
  ii) next field holds the address of successor node.
- Apart from that two more variables are required
  - i) front which holds the address of the first node
  - ii) rear which holds the address of last node



Menu driven Program for linear queue using linked list

```
typedef struct NODE
{
int info;
 struct NODE *next;
 }node;
node *front=NULL;
node *rear=NULL;
void insertion();
void del();
void display();
void main()
{
int ch;
while(1)
 {
 printf("1. insertion 2. Deletion 3. Display 4. Exit");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1: insertion();
         break;
  case 2: del();
          break;
  case 3: display();
          break;
  case 4: exit(0);
  default:
          printf("Invalid choice");
  }
 }
getch();
}
```

```
void insertion()
{node *ptr;
ptr=(node *)malloc(sizeof(node));
printf("Enter info");
if(ptr==NULL)
{printf("queue overflow");
 getch();
 exit(0);
}
else
{printf("Enter info");
 scanf("%d",&ptr->info);
 ptr->next=NULL;
 if(front==NULL)
  {front=ptr;
   rear=ptr;
  }
 else
  {rear->next=ptr;
   rear=ptr;
  }
}
void del()
{node *ptr;
 if(ptr=front;
 if(ptr==NULL)
{printf("queue underflow");
 getch();
 exit(0);
}
else
{ptr=front;
printf("element deleted is %d",ptr->info)
 if(ptr->next==NULL)
  {front=NULL;
   rear=NULL;
```

```
else
    front = front -> next;
free(ptr);
}
}
void display()
{
node *ptr;
ptr = front;
while(ptr != NULL)
{
printf("%d", ptr->info);
ptr = ptr -> next;
}
}
```

Application of linked list -:

Polynomials -:

> Linked lists are widely used to represent and manipulate polynomials.

> Polynomials are the expressions containing number of terms with non zero co-efficients exponents.

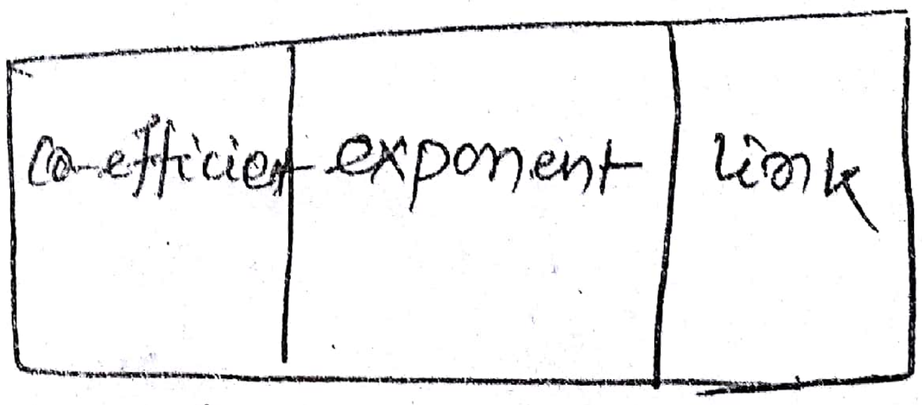
$$P(x) = a_n x_n^e + a_{n-1} n_{n-1}^e + \ldots + a_1 x_1^e + a$$

a = non zero co-efficient.

e = exponent.

> In linked list representation of polynomials each term is considered as a node of such node contains 3 fields.

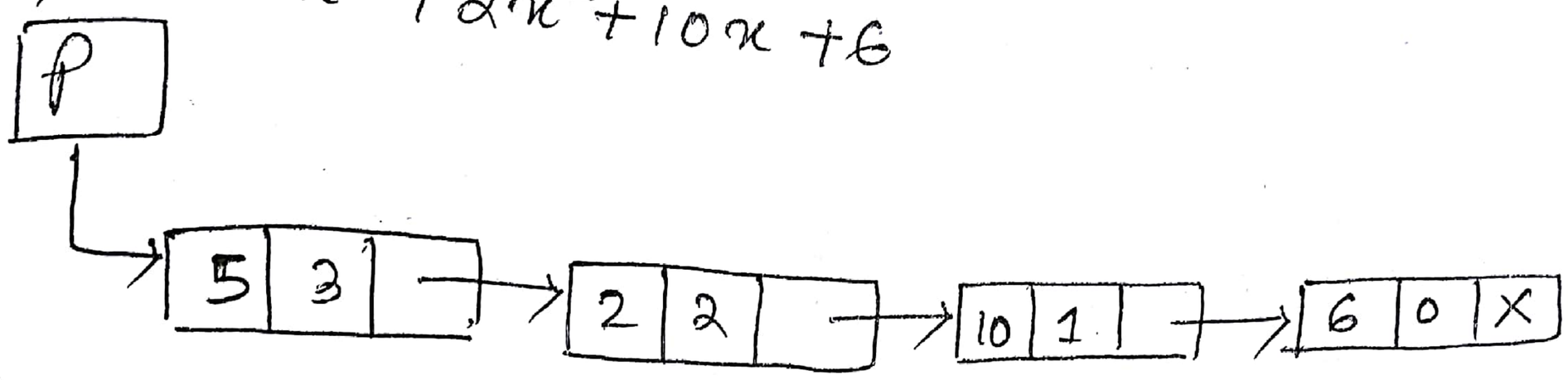
1) co-efficient field
2) Exponent field
3) link field



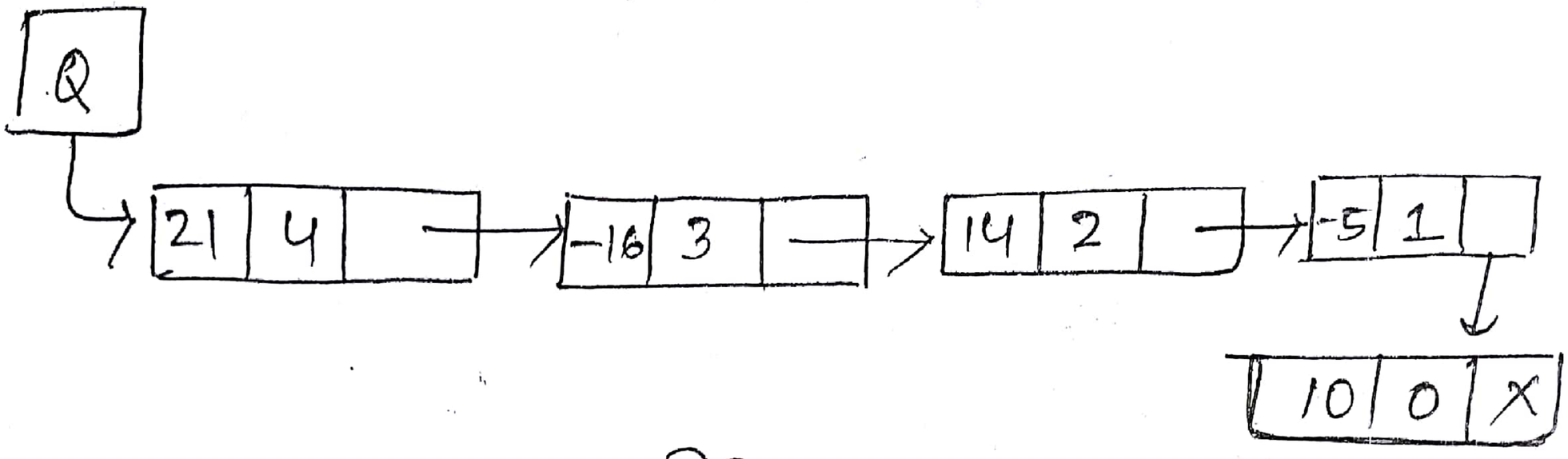
Structure Def^n :-

```
struct polynode
{
    int coeff;
    int exp;
    struct polynode *next;
};
```

Ex :- 1) $P = 5x^3 + 2x^2 + 10x + 6$



2) $Q = 21x^4 - 16x^3 + 14x^2 - 5x + 10$



Steps for polynomial addition :-

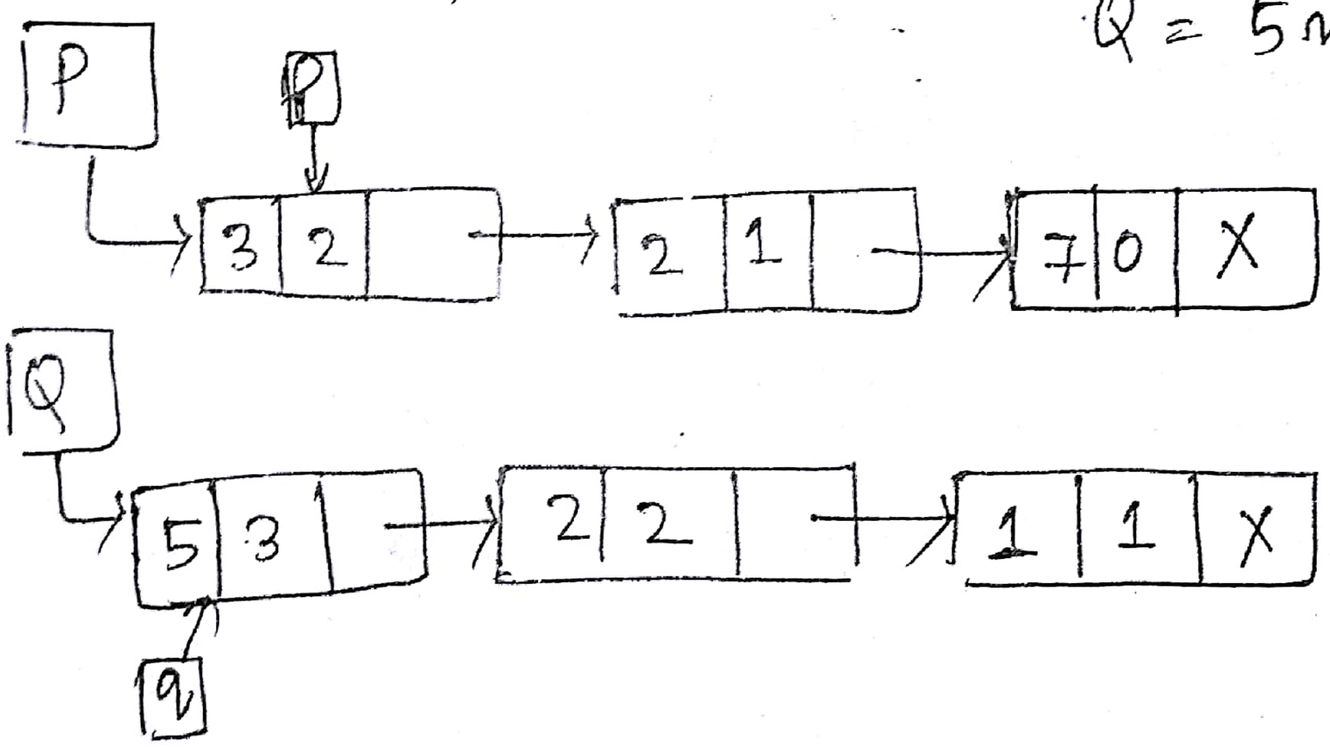
1) Read the number of terms in the first polynomial.
2) Read the co-efficient & exponent of the first polynomial
3) Read the number of terms in the second polynomial.
4) Read the co-efficient & exponent of the second polynomial
5) Set temporary pointer 'p' & 'q' to traverse the two polynomials respectively.
6) compare the exponent of the polynomials starting from the first node.
7) a) ~~compare the exponent of the polynomials starting from the first~~
   a) If both exponents are equal then add the co-efficients and store it in the resultant linked list.

b) if the exponent of the current term in the first polynomial P is less than the exponent of the current term in the 2nd polynomial Q. Then current term of Q is added to resultant linked list & move the temporary pointer 'q' to point to the next node in the 2nd polynomial Q.

c) if the exponent of the current term in the first polynomial P is ~~less~~ greater than the exponent of the current term in the 2nd polynomial Q. Then current term of P is added to resultant linked list & move the temporary pointer 'p' to point to the next node in the 1st polynomial P.

d) Append the remaining nodes of either of the polynomial to the resultant linked list.

Qs-: illustration of addition of $P = 3x^2 + 2x + 7$
$Q = 5x^3 + 2x^2 + x$

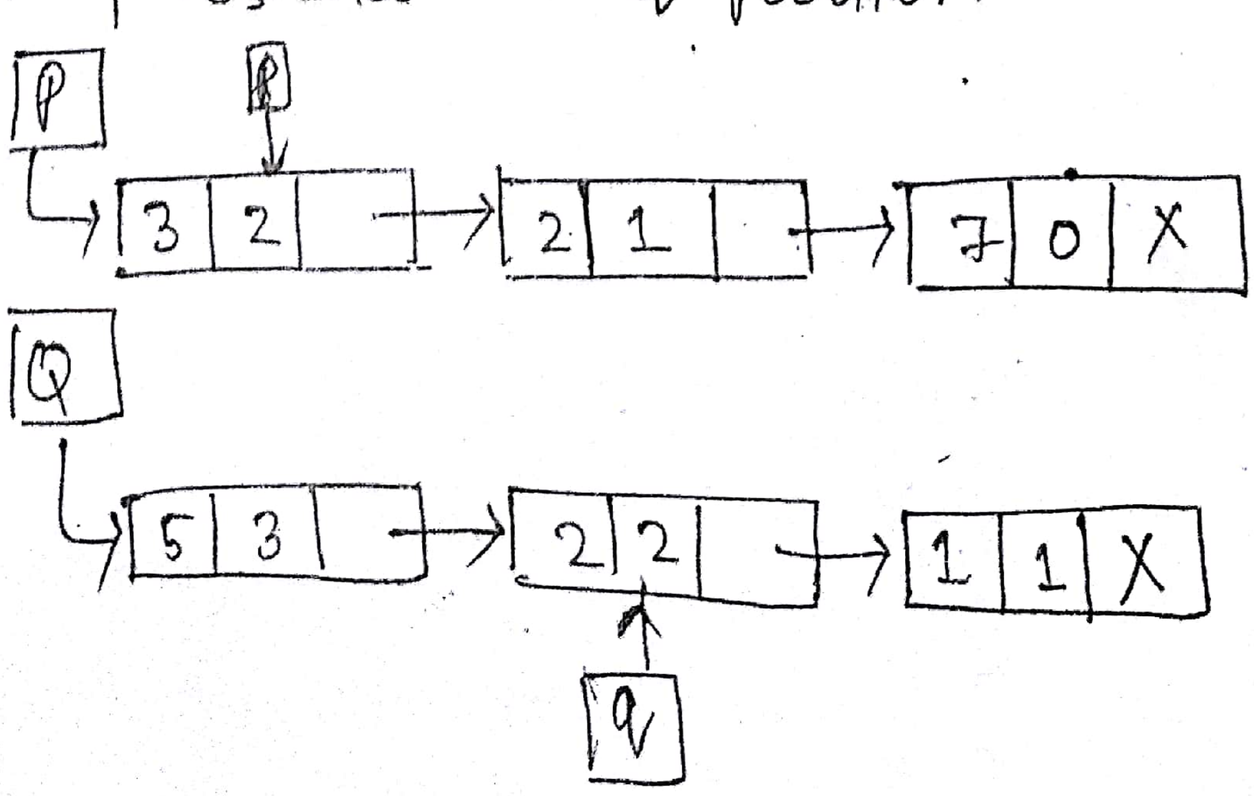


Step 1 -: comparing the exponent of p & corresponding exponent of Q.

Here exp(P) < exp(q)

So add the current term of Q to the resultant linked list & advance the q pointer.

Step 2 -:

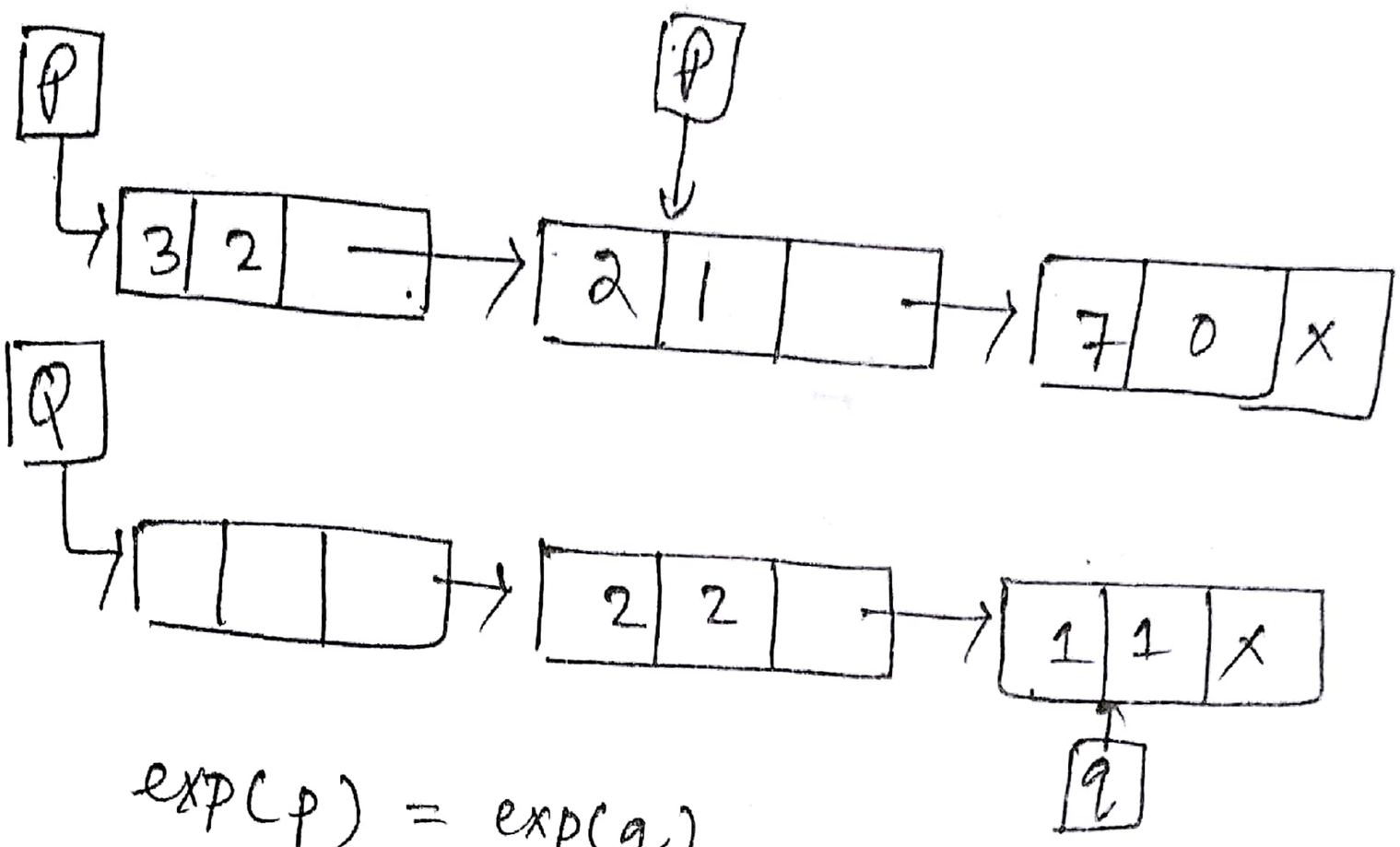


comparing the exponent of current term of p & q

exp(p) = exp(q)

So add the co-efficient of these 2 terms & advance p & q pointer.

Step 3:

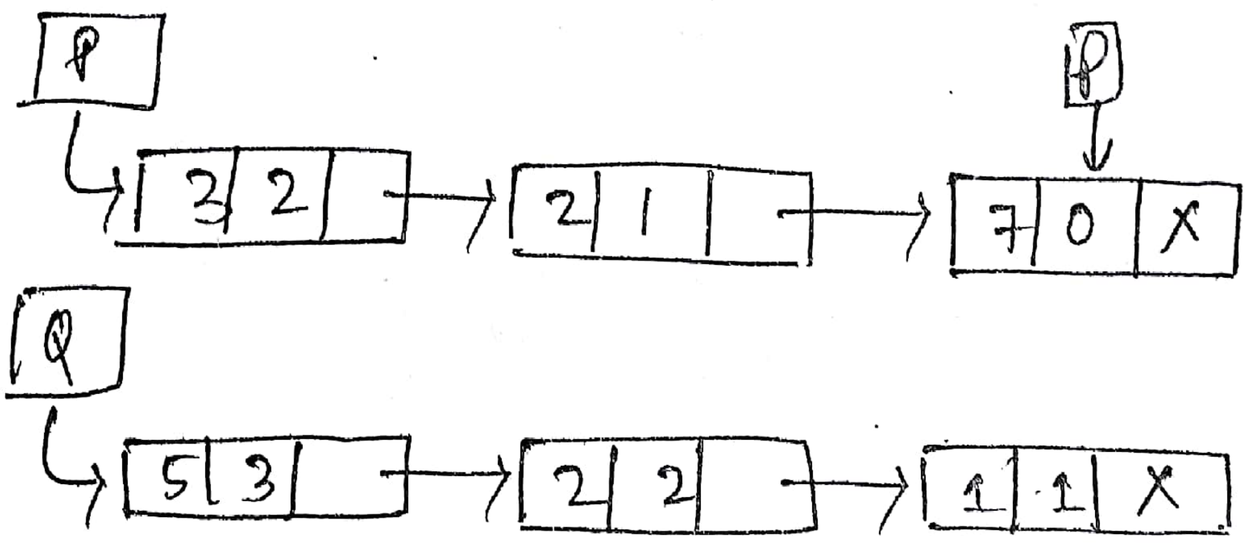


~~Step 4:~~

exp(p) = exp(q)

So add the coefficient of these 2 terms & advance p & q pointer.

Step 4:



There is no node in the 2nd polynomial to compare with so rest node in 1st polynomial is simply added to resultant linked list.

Step 5: Display Resultant Linked list.