1) To insert an element & delete at a specific posit. in linked list wher n & k is taken from user

A) Programme

```
#include <stdio.h>
#include <stdlib.h>
void insert_after(struct node* head, int value, int after)
  int a printf("Enter the int-after,")
      scanf("%d" &int_after)
{
  struct node* new_node = NULL;
  struct node* tmp = head;
  while(tmp){
  if(tmp->val == after) {/* found the node */
  new_node = (struct node*)malloc(sizeof(struct node));
  if(new_node == NULL) {
    printf("Failed to insert element. Out of memory");
  }
  new_node->val = value;
  new_node->next = tmp->next;
  tmp->next = new_node;
  return;
  }
  tmp = tmp->next;
  }
}
```

```
ete Node( struct Node **head_ref, int position)
{
    if(*head_ref == NULL)
        return;

    Struct Node *temp = *head_ref;
    If(Position == 0)
    *head_ref = temp->next;
    free(temp);
    return;
for(int i=0; temp!=NULL && i< position-1; i++)
        temp = temp->next;
if (temp == NULL || temp->next == NULL)
    return;
struct Node *next = temp->next->next;
free(temp->next);
tem->next.next;
```

2) Merge a linked list at alternate position

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node *next;
};

void push(struct Node ** head_ref, int new_data)
{
    struct Node * new_node =
        (struct Node *) malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_nodes;

void printlist(struct Node *head)
{
    struct Node *temp = head;
    while(temp != NULL)
    { printf("%d", temp->data);
      temp: temp->next;
```

```
printf("\n");
}

void merge(struct Node *p, struct Node *q)
{
    struct Node * p_curr = p, *q_curr = q;
    struct Node * p_next; *q_next;

    while (p_curr != NULL && q_curr != NULL)
    {
        p_next = p_curr->next;
        q_next = q_curr->next;

        q_curr->next = p_next;
        p_curr->next = q_curr;

        p_curr = p_next;
        q_curr = q_next;
    }
    *q = q_curr;
}
```

4) Reversing elements in a que

```
# include <stdio.h>
# include <stdlib.h>
  Struct stackrecord
  {
    int *array;
    int capacity;
    int tos;
  };
  typedef struct stackrecord *stack;
  Stack createstack(int max)
  {
   Stack S;
   S = malloc (size of (struct stackrecord));
   if (S == NULL)
   {
     Printf("out of space");
   }
   S->array = malloc ((sizeof (int)) * max);
   if (S->array == NULL)
   {
     Printf ("out of space");
    }
    S->array = malloc((sizof (int)) * max);
    if (S->array == NULL)
    { Printf("out of space");
```

```
S->capacity= max-1;
 S->tos = -1;
 return(s);
}
int is emptys (stack s)
{
  return S->tos == -1;
}
  int is full s (stack s)
  {
   return S->tos == S->capacity;
   }
    void push (int x, stack s)
   {
    if (is fulls(s))
     Printf("Overflow");
   else
   {
    Printf("\n %d is Pushed", x);
     s->tos++;
     s->array[s->tos] = x;
   }
   }
   int top and pop (stack s)
   {
   if (is emptys(s))
   {
   Printf("\n empty stack")
   return;
```

```
}
else
{
printf("\n %d is popped", s->array[s->tos]);
return s->array[s->tos--];
}
}
struct queuerecord
{
int *array;
int front;
int rear;
int capacity;
};
typedef struct queuerecord *queue;
queue creatqueue(int max)
{
queue q;
q = malloc(sizeof(struct queuerecord));
if(q==NULL)
Printf("Error")
q->array = malloc(sizeof(int)*max);
if(q->array==NULL)
Printf("Error");
q->capacity = max-1;
q->front = -1;
q->rear = -1;
```

```
}
int isfullq(queue q)
{
 return (q->rear==q->capacity);
}
 int isemptyq(queue q)
{
  return (q->front==-1);
 }
 void enqueue(queue q, int x)
 {
  if (isfullq(q))
  Printf("overflows");
  else
{
Printf("\n %d is enqued ", x);
q->rear++;
 q->array[q->rear] = x;
if (q->front==-1)
 q->front++;
 }
 }
int frontanddelete(queue q)
 {
  int p;
```

```
if (is empty
{ printf("under Flow");
 return;
}
else
{
p = q->array [q ->front];
Printf ("\n %d is foont & deleted ", p);
 q ->foont++;
 return p;
 }
}
void display (queue q)
{
 int i;
 if (is empty q (q))
{
 Point f("under Flow");
  return;
}
for (i = q->front; i<rear; i++)
 Printf("%d \t", q->array [i]);
}
 int main ()
 {
 int max, ele, i, choice, n=0, x=2;
 queue q;
 stack s;
```

```
Point f ("\n\n Enter the max elements:");
scan F ("%d", &max);
q = Coeate queue (max);
s = Coeate stack (max);
while(1)
{ point F ("\n Menu : 1. Insert 2. Display reverse order
                3. exit");
Point f ("\n Enter the choice");
scan f ("%d" &choice);
switch (choice)
{
case 1;
Point f ("\n Enter the element:");
scan F ("%d ", &ele);
enqueue (q, ele);
n++;
break;
case 2:
Point f ("\n Contents of the queue:");
display (q);
for (i=0; i < capacity; i++)
{
```

```
push(2,5);
}
q->front = -1;
q->rear = -1;
for(i=0; i<capacity; i++)
{y = top and pop(s);
enqueue(q,y);
}
Printf("\n Reversed contents are: ");
display(q);
bocak;
case3;
exit(0);
}
}
}
```

5) Array	Linked List
1) It consistent set of a fixed number of data items	It is an ordered set comprising a variable number of data items
Specified during declaration	No need to specify; grow and shrink during execution
stored consecutively	Stored Randomly
Direct or Random access	Sequentially accessed
Slow Relatively as shifting is required	Easier, fast & efficient
Both Binary & linear search	Linear search
less required memory	More memory required
Ineffective	Efficient

ii)

```
#include <stdio.h>
#include <stdlib.h>

{ int data;
  struct Node* next;
};

{ struct Node * new_node = (struct Node*) malloc (size of (struct
                                                    Node));

  new_node ->data = new_data;
  new_node ->next = (*head_ref);

(*head_ref) = new_node,
 }
  void printList (struct Node * head)
 {
   struct Node * temp = head;
   while (temp != NULL)
   {
     Printf ("%d", temp->data);
      temp = temp->next;
    }
     Printf ("\n"),
 }
```

```
{ Struct Node *p_curr = p, *q_curr = *q;
  Struct Node* p_next, *q_next;
  {
    p_next = p_curr -> next;
    q_next = q_curr -> next;

    q_curr -> next = p_next;
    p_curr -> next = q_curr;

    p_curr = p_next;
    q_curr = q_next;
  }
  *q = q_curr;
}
int main()
{
  struct Node * p = NULL; *q = NULL;
  Push(&p, 3);
  Push(&p, 2);
  Push(&p, 1);
  Point("First List:\n");
  Point List(q);
```

```
merge(P,&Q);
Printf("Merged List:\n");
PrintList(P);
PrintList(Q);
 getchar();
 return 0;
}
```