1. Write a program to insert and delete an element at the n^{th} and k^{th} position in a linked list where n and k is taken from the user.

code :-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 struct node *next;
};
struct node * curr *temp;
void input (struct node*)
 void delete (struct node*)
  void main(void)
  {
   struct node * s;
   int n;
   s=NULL;
  do
  {
   printf ("1.enter the element to insert; \n");
   printf ("2. enter the element to delete \n");
    printf("3. exit \n")
    printf ("enter the choice:");
    scanf("%d",&n);
    switch(n)
    {
     case 1 :input (s);
             break;
     case 2: delete (s);
             break;
      } while (n!=3)
```

```
}
void input(struct node* z)
{
  int p, c=1
  curr z;
 Printf("enter the element to be inserted:");
 scanf("%d", &p);
  while (curr->next != NULL)
  {
   c++;
  if (c==p)
  {
  temp=(struct node*)(malloc(size (struct node)));
  Printf("enter the numbers:");
  scanf("%d", &temp->n);
  temp->next = curr->next;
  curr->next =temp
   break;
  }
}
}
void delete (struct node* z)
 {
 int p, c=1;
  curr=z;
 printf("enter the element to be deleted:");
 scanf("%d",&p);
  while(curr->next != NULL)
  {
  c++;
 if(c==p)
```

```
{
 temp = current->next;
 curr->next = curr->next->next;
 free (temp)
 }
curr=curr->next;
}
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
}
int main( )
{
 struct node* p=NULL, *q=NULL
  push (&p, 1);
  push(&p, 2);
  push(&p, 3);
```

```
Printf ("first linked list : \n");
 Print (ist (R);
 push (&q, 4);
 push (&q, 5);
 Push (&q, 6);
 Printf ("second linked list : \n"),
 Print list (q);
 merge (P, &q);
 Printf ("Modified first linked list = \n");
 Print List (P);
 Printf ("Modified second linked list = \n");
 Print List (q);
 return 0;
}
```

2. Construct a new linked list by Merging alternate nodes of two lists for example in list 1 we have {1,2,3} and in list 2 we have {4,5,6} in the new list we should have {1,4,2,5,3,6}.

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct Node*next;
};
void printlist (struct Node*head)
{
 struct Node*ptr=head;
 while (ptr)
 {
 printf("%d->", ptr->data);
 ptr =ptr->next;
 }
 printf ("NULL\n");
 }
void push(struct Node** head, int data)
{
 struct Node* newNode =(struct Node*)malloc(sizeof(struct Node));
 new Node ->data =data;
 new Node ->next =*head;
 *head =new Node;
}
 struct Node* shuffle Merge (struct Node*a, struct Node*b)
 {
```

```
struct Node dummy;
struct Node* tail = &dummy;
dummy.next = NULL;
while (1)
{
 if (a == NULL)
 {
  tail->next = b;
  break;
 }
 else if (b == NULL)
 {
  tail->next = a;
  break;
 }
 else
 {
tail->next = a;
tail = a;
a = a->next;
tail->next = b;
tail = b;
b = b->next;
 }
}
 return dummy.next;
 }
 int main(void)
 {
 int keys[] = {1,2,3,4,5,6,7};
 int n = sizeof(keys) / sizeof(keys[0]);
 struct Node *a = NULL, *b = NULL;
 for (int i=n-1; i>=0; i=i-2)
```

```
    Push (&a,keys[i]);
for (int i=n-2; i>=0; i=i-2)
    Push(&b, keys[i]);
Printf ("first list:");
Print list'(a);
Printf ("second list:");
print list(b);
Struct Node* head = shuffle Merge(a,b);
Printf ("After Merge:");
printlist(head);
return 0;
}
```

3. Find all the elements in the stack whose sum is equal to k. (where k is given from user).

code:

```
#include <stdio.h>
int top=-1;
int x;
char stack[100];
void push (int x);
char pop();
int main()
{
int i, n, a, t, k, f, sum=0, count=1;
printf ("enter the number of elements in the stack");
scanf ("%d", &n);
for(i=0; i<n; i++)
 {
 printf ("enter next element");
 scanf ("%d,", &a);
```

```
push(a);
}
printf("enter the sum to be checked");
scanf("%d",&k);
for(i=0;i<n;i++)
{
t=pop();
sum+=t;
count+=1;
if(sum==k)
{
for(int j=0;j<count;j++)
printf("%d",stack[j]);
f=1;
break;
}
push(t);
}
if(f!=1)
printf("The elements in the stack dont add up to the sum");
}
void push(int x)
{
if (top==99)
{
printf("\n stack is FULL!!!\n");
return;
}
top=top+1;
stack[top]=x;
}
```

```
char pop()
{
if(stack[top]==-1)
{
printf("\n stack is EMPTY!!!\n");
return 0;
}
x=stack[top];
top=top-1;
return x;
}
```

4. write a program to print the elements in a queue
i, in reverse order ii, in alternate order.

Code:-

i)

```
#include<stdio.h>
#include "stack.h"
#include "QQ.h"
int main()
{
int n, a[20], i, j=0
struct stack s;
init stack (&s);
printf("enter number");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("enter values :");
scanf("%d", &a[i]);
}
```

```
for (i=0; i<n ; i++)
{
 insert (a[i]);
 }
  while (j!=n)
  {
     Push (&s, del( ));
     j ++;
  }
   printf("Reverse order ");
   while (stop!=-1)
   {
    Printf ("%od", pop(&s));
     }
     Printf ("\n");
 return 0;
 }
```

(ii)

```
#include <stdio.h>
#include <stdlib.h>
 struct node
 {
  int data;
   struct Node* next;
 }
 Void print Nodes (struct Node* head)
 {
   int count = 0;
```

```
while (head! =NULL){
 if (count %2 ==0)
 {
  printf("%d", head->data);
  }
   count++;
   head = head->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_node;
       }
      int main( )
      {
      struct node* head=NULL;
       push(&head,12);
       push(&head,29);
       push(&head,11);
       push(&head,23);
       push(&head,8);
       Print node(head);
return 0;
}
```

5. (i) How array is different from the linked list?

Array is a collection of elements of similar data type. Linked list is an ordered collection of elements of same type, which are connected to each other using pointers. Elements in array are sorted in contiguous memory location or consecutive manner in the memory. In linked list, new elements can be stored anywhere in the memory. In array, Insertion and deletion operation takes more time. In linked lists Insertion and deletion operations are fast in linked list. In array memory is allocated at compile time and known as static Memory Allocation. In linked list memory is allocated at run time and known as Dynamic Memory Allocation. Array and linked list regards to their structural Difference.

(ii) write a program to add first element of one list to another list of example we have {1, 2, 3} in list 1 and {4, 5, 6} in list 2 we have to get {4, 1, 2, 3} as output for list 1 and {5, 6} for list 2.

code:

```
#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 (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");
}
```