Assignment - 4

1. Write a program to insert and delete an element at the nth and kth position in a linked list where n and k are taken from user.

```
# include <stdio.h>
# include <stdlib.h>
Struct Node{
        int data;
        struct Node *next;
};
Struct Node *delete (struct Node *head, int n);
Struct Node *insert (Struct Node *head, int n);
Struct Node *Create_list ();

Void main()
{       int k;
        int p;
        Struct Node *head;
        head = Create_list ();
        display (head);
        printf ("Enter index where you want to enter:");
        scanf ("%d", &k);
        printf ("Enter index that you want to delete:");
        Scanf ("%d", &p);
        head = insert (head, k);
        display {head);
        head = delete(head, p);
        display (head);
}.
```

Scanned with CamScanner

```
Void   display(struct Node *head)
{
        Struct Node *p;
        for (p=head; p!=NULL; p=p->next)
        {
              printf("\n Node : %d", p->data);
        }
}
Struct Node * Create_list ()
{
        int k,n;
        struct Node *p, *Head;
        Printf("\n Enter the number of elements:");
        Scanf("%d", &n);
        for ( k=0; k<n; k++)
        {     if (k==0)
              {
                   Head =(Struct Node*)malloc(sizeof(struct Node));
                   p=Head;
              }
              else
              {
                   p->next=(struct Node*) malloc(sizeof(struct Node));
                   p=p->next;
              }
              ~~printf("Enter element:\n"~~
              printf("\n Enter element:");
              Scanf("%d",&p->data);
        }
}.
```

```
Struct Node *insert( Struct Node *head, int n)
{       int i=0;
        Struct Node *p, *temp;
        p=head;
        temp=(struct Node*)malloc( sizeof(struct Node));
        while (i!=n)
        {    p=p->next;
             i++;
             if (i==n)
             {
                  printf("Enter the element that you
                              want to enter");
                  Scanf ( "%d", &temp->data);
                  temp->next = p->next;
                  p->next=temp;
             }
        }
        return (head);
}
Struct Node *delete ( struct Node *head, int n)
{       int i=0;
        Struct Node *p, *temp;
        p=head;
        while (i!=n-1)
        {    p=p->next;
             i++;
             if (i==n-1)
             {     p->next=(p->next)->next;
             }
        }
        return (head);
}.
```

Output:-

```
Enter the number of elements: 5
Enter element : 1
Enter element : 2
Enter element : 3
Enter element : 4
Enter element : 5

Node: 1
Node : 2
Node : 3
Node : 4
Node : 5
Enter the index where you want to enter 2
Enter the element that you want to insert 66

Node 1
Node 2
Node 3
Node 66
Node 4
Node 5.
Enter index ~~where~~ that you want to delete: 2

Node 1
Node 2
Node 66
Node 4
Node 5.
```

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}.

```
/* We are using the same functions which were
 used in first question like create_list() display() instead
 of writing again */
#include <stdio.h>
Struct Node
{
        int data;
        Struct Node *Next;
}; Struct Node * merge (struct Node *a , struct node *b);
int main()
{
        struct Node *head 1, *head2, *head;
        head1 = Create_list();
        head2 = create_ list();
        printf ("\n The elements in first linked list:");
        display(head 1);
        printf ("\n The elements in second linked list:");
        display( head2);
        printf ("\n The List after merging:\n");
        head = |
        display(head);
}
Struct Node*  Merge ( struct Node *a, struct Node *b)
{
        Struct Node dummy;
        Struct Node *new list = &dummy;
        dummy.next = NULL;
while (1)
{       if (a==NULL);
        {
              newlist->next=b;
              break;
        }
        else if ( b==NULL)
        {
               new list-> next=a;
               break;
        }
        else
        {
               newlist->next=a;
               newlist=a;
               a=a->next;
               newlist-> next=b;
               newlist=b;
               b=b->next.
        }
}
return dummy.next;
}
```

Output:-

```
Enter the number of elements:3
Enter element : 1
Enter element :2
Enter element :3
Enter the number of elements:3
Enter element: 4
Enter element: 5
Enter element: 6
```

```
The elements in first linked list:

Node: 1
Node: 2
Node: 3
The elements in second linked list:

Node: 4
Node: 5
Node: 6
The list after merging

Node: 1
Node: 4
Node: 2
Node: 5
Node: 3
Node: 6
```

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

```
~~# include <stdio.h>
Void Sum( int arr[ ], int n; int p)
{              int S=0, l=0, h=0;
               for (l=0; l<n; l++)
               {     while ( s<p && h<n)
                     {
                           S+ = arr[h];
                           h++
                     }
                     if (Sum==S)
                     {     printf ('~~
```

3. Find all the elements in the stack whose sum is equal to k

```
#include <stdio.h>
# include <stdlib.h>
Void sum (int arr[], int s, int n)
{          int i,j,k;
           for (i=0; i<n; i++){
                for ( int j=i+1; j<n ; j++)
                     if (arr[i]+arr[j] == s)
                     {
                          printf("{%d,%d,%d}\t", arr[i],
                                                  arr[j]);
                     }

           }
           for (i=0; i<n; i++ {
                for ( ~~j=0~~ j=i+1; j<n; j++)
                     for ( k=j+1; j<n; j++)
                          if ( arr[i]+arr[j]+arr[k]==s)
                          {
                               printf ("{%d, %d, %d}\t",
                                       arr[i],arr[j],arr[k]);
                          }
           }
}
Void main( )
{          int arr[10] = {1,2,3,4,5,6}, Sum;
           printf (" Enter number");
           Scanf (" %d", &Sum);
           Sum( arr, Sum, 6);

}
```

Output:-

```
Enter number 10
{4,6} {1,3,6} {1,4,5} {2,3,5}
Enter number 12
{1,5,6} {2,4,6} {3,4,5}.
Enter number 7
{1,6} {2,5} {3,4} {1,2,4}
```

4. Write a program to print the elements in a queue
(i) in reverse order
(ii) in alternate order.
Both (i) & (ii) are excecuted in the same code.

```
#include <stdio.h>
#include<stdlib.h>
#define SIZE 10
Void enQueue (int);
Void deQueue();
Void reverse_display();
Void alternate_display();
int queue[10], front=-1, rear=-1;
Void main(){ while(1) {
        int Value, choice;
        printf("Enter your choice:\n 1.Insertion \n2.Deletion\n
                3.Displaying in reverse order\n 4.Displaying
                 alternate element\n 5.Exit");

        Scanf("%d", &choice);
        Switch (choice){
         case 1:
                printf("\nEnter the element You want to insert");
                Scanf("%d", &Value);
                en Queue(Value);
                break;
         Case 2:
                deQueue();
                break;
         Case 3:
                 reverse_display()
                 break;
         Case 4:
                 alternate_display()
                break;
        default:
                printf("\n Wrong Selection! Try again!");
        }
}}
Void  enQueue (int value) {
        if (rear == SIZE-1) {
                printf("\n Queue is full"); }
        else{
                if (front == -1) {
                        front = 0; }
                rear++;
                queue[rear] = value; }
}.
Void  de Queue() {
        if (front == -1 || front > rear) {
                printf(" Queue is empty\n"); }
        else {
                printf(" \n Deleted: %d", queue[front]);
                front++;
                if (front == rear+1) {
                        front = rear = -1; }
                }
}
Void  reverse_display() {
        if (rear == -1) {
                printf("Queue is empty\n");
        else {
                int i;
                printf(" The elements are:\n");
        for(i=rear; i>=front ;i--) {
                printf("%d\t", queue[i]); }
        }
}
Void   alternate-display().{
    if  (rear==-1){
            printf("\n Queue is Empty"); }

    else{
          int i;
          printf("\n Queue elements1);
          for(i=front; i<=rear; i+=2).
          {      printf("%d \t", queue[i]);
          }
    }
}.
```

Output:-

```
Enter your choice:
1. Insertion
2. Deletion
3. Displaying in reverse order
4. Displaying alternate elements
5. Exit. 1
Enter the element you want to
insert : 11
Enter your choice:
1. Insertion
2. Deletion
3. Displaying in reverse order
4. Displaying alternate elements
5. Exit 1
Enter the element you want
to incert.22
```

```
Enter your choice:
1. Insertion
2. Deletion
3. Displaying in reverse order
4. Displaying alternate elements
5. Exit 1
Enter the element you want
to insert: 33
Enter your choice:
1. Insertion
2. Deletion
3. Displaying in reverse order
4. Displaying alternate elements
5. Exit 1
Enter the element you want
to insert: 44
```

```
Enter your choice:
1. Insertion
2. Deletion
3. Displaying elements in reverse order
4. Displaying alternate elements
5. Exit . 3
~~Deleted~~ :
The elelements are:
44  33  22  11
Enter your choice:
1. Insertion
2. Deletion
3. Displaying elements in reverse order
4. Displaying alternate elements
5. Exit. 4
The elements are:
11   33
Enter your choice:
1. Insertion
2. Deletion
3. Displaying elements in reverse order
4. Displaying alternate elements
5. Exit 5.
```

(5) (i). How array is different from linked list
(ii) Write a program to add the first element of one list to another list for 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

(i)

Array	Linked List
(1) Fixed size: resizing expensive	(1) Dynamic size
(2) Elements need to be Shifted for insertion and deletion	(2) No shifting is ~~requ~~ takes place in while inserting or deleting
(3) Random access is possible (indexing)	(3) No random access Not suitable for operations requiring accessing elements by index such as Sorting
(4) It results huge memory waste except the array is full.	(4) Since memory is allocated dynamically, there is no memory waste
(5) Sequential access is faster (Elements in Contiguous memory locations	(5) Sequential access is slow (Elements are not in contiguous memory locations).

```
(ii) # include <stdio.h>
# include <stdlib.h>
int  len (int a[]) // To find the length easily: defining function
{       int i=0, an=0;
        while(1)
        {      if  (a[i])
               {      an++, i++;
               }
               else
               {    break;  }
          }
}.
Void   changinglist (int a[], int b[])
{       for (int i= len(a)-1 ; i>=0; i--)
         {        a[i+1] = a[i] ;
         }
         a[o] = b[o];
         for  (int j=0; j< len(b) ; j++)
          {         b[i] = b[i+1];
          }
          printf("\n The elements of first array:\n");
          for (int i=0; i< len(a) ; i++)
          {
                  printf ( " %d  ", a[i]);
           } printf ("\n The elements of second array: \n"):
          for (int i=0; i<len(b); i++)
           {
                  printf ( " %d  ", b[i]);
           }
}
```

```
int  main()
{
       int  a[10] = {1,2,3} , b[10] = {4,5,6}
       Changinglist(a,b);
}
```

Output :-

```
The elements of first array
 4     1     2 3  3
The   elements of second array

 5    6
```