ASSIGNMENT- 4

1)
```
# include <stdio.h>
# include < stdlib.h >
void insert (node*, int ; int)
int  size = 0;
struct node{* ~~node next~~;
int data ;
struct node *next ;
}
node *get node (int data)
{
  node *newnode = (struct node*)malloc (newnode );
  new node → data = data;
  new node → next = null ;
  return new node;
 }
 void insert (node *current, int post, int data)
 {
   if (post < 1 || post > size+1)
   printf ("Invalid");
   else
  { while (post --)
   { if (post == 0)
     { node* temp = get node (data);
       temp → next = * current ;
       * current = temp;
     }
 else
   { current = & (*current) → next;
   }
  size++; }}
```

```
void printf (struct node* head)
 { while (head != null)
   { printf ("%d", head → data);
     head = head → next;
   }
    printf ("\n");
   }
 Void del (struct node* head_f, int post)
 { if (head_f == null)
   return;
   temp = head_f;
   if (post =0)
   {
    * head_f = temp → next;
    free (temp → next);
     temp → next = next;
   }}
   int main ()
   {
    struct node * head = Null;
    push (& head, 7);
    push (& head, 8);
    push (& head, 6);
    insert (& head, 7, 15);
    del (& head, 4);
    print list (head);
    return (0);
   }
```

Scanned with CamScanner

```
2) # include <stdio.h>
 # include <stdlib.h>
  struct node {
  int data;
  struct node* next;
  }
 void print list (structnode* 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* new = (struct node*) malloc (size of
                                              (struct node*));
     new -> data = data;
     new -> next = * head;
    * head = news;
   }
    struct node * 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;
}}
return dummy.next;
}
void main ()
{
int keys [] = {1,2,3,4,5,6,7};
 int n = size of (keys) / size of key [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, key [i]);
 struct node* head = merge (a,b);
   print-list (head);
}
```

3)
```
# include <stdio.h>
void find (int arr[], int n, int s) {
 int sum = 0;
 int l = 0, h = 0;
 for (l=0; l<n; l++) {
   while (sum<s && h<n)
     sum+ = arr [h];
      h++;
   if (sum == s)
   { printf ("found");
     return; }
      sum - = arr[l];
   }}
   int main (void) {
   int arr [] = {2,6,0,9,7,3}
   int s = 15;
   int n = size of arr/ size of (arr [0]);
   find (arr, n, s)
   return 0;
   }
```

```
4) # include <stdio.h>
   # include < stdlib.h>
   struct node
   { int data;
     struct node * next;
   }
    void print rev (struct node *head)
    { if (head == Null)
         return;
      print rev (head->next);
      print f ("%d", head-> data);
  void push (struct node *node_new = (struct node *) malloc
                                             (size of (struct node));

   node_new -> data = new;
   node_new -> next = (head *_ref);
   (head_ref) = node_new;
  } int main ()
    struct node *head = Null;
      push (& head, 4);
      push (& head, 3);
      push (& head, 2);
    print new (head); print alternate (head);
    return 0;
   }
   void print alternate (struct node* head)
   { int count = 0;
     while (head != Null)
     { if (count % 2 == 0)
            printf ("head->data");
            count ++;
            head = head-> next;
     }
```

5) Differences are:

i) An array can store similar type of data type whereas linked list can store different data type.

ii) In array, elements belong to indexes. whereas in linked list you have to start with head if you want to get some element.

iii) Accessing an element in array is fast whereas in linked list it is linear so, it is slow.

iv) Operations like insertion and deletion in array consume a lot of time whereas in linked list it is quite fast.

v) In array memory is assigned before, which wastes memory whereas in linked list it is allocated in runtime.

```
ii) # include <stdio.h>
  # include <stdlib.h>
  int len (int a[ ])
  { int i = 0, am = 0;
     while(1)
     { if (a[i])
        { am++, i++;
        }
        else
         { break;
          }
       }
      return am;
   }
   void changinglist(int a[ ], int b[ ])
   { for ( int i = len (a) - 1; i >= 0, i--)
     { a[i+1] = a[i];
      }
     a[0] = b[0];
   printf ("\n the elements of first array : \n");
   for ( int i = 0; i < len (a); i++)
     { printf (" %d", a[i]);
     }
   for ( int i = 0; i < len(b); i++)
     { b[i] = b[i+1]; }
  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);
```

Scanned with CamScanner