# ASSIGNMENT - 4

N. Akhila
AP19110010322
CSE-H

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

-Ans
```
# 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 display (struct Node *);
void main ()
{ int k;
    struct Node * head;
    head = create_list();
    display (head);
    printf("enter the index where you want to enter!");
    scanf ("%d",&k);
    head = insert (head, k);
    display (head);
    head =delete (head, 3);
    display (head);
}
```

```
void display (struct Node* head)
{
Struct Node * P;
for (P=head; P! =NULL;  P=P->next)
 {
  Printf(" \n Node data %d", P->data);
 }
 Printf("\n");
}
Struct Node* create_list ()
{
 int k, n;
 struct Node * P, * Head;
 printf (" \n How many elements to enter ?");
  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 (size of struct Node));
     P=P->next;
   }
    printf("\n Enter an '%dth element", k);
     scanf ("%d", &P->data);
    }
     P->next=NULL;
    return (Head);
  }
```

```
Struct Node *insert (struct Node* head, int n)
{
 int i=0;
 Struct Node * P, *temp;
  P=head;
 temp=(struct Node*) malloc (size of (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);
}
```

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 we should have {1,4,2,5,3,6}.

Ans

```
# include <stdio.h>
# include <stdlib.h>
Struct node {
  int data;
  struct node* next;
}
void print list ( structnode * head)
{
   structnode * 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 =new;
 }
 Struct node* merge (struct node* a, struct node* b)
 {
    struct node dummy;
    Struct node* fail =dumy;
    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, keys[j]);
 struct node* head =merge (a,b);
 print list (head);
}
```

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

Ans

```
# 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. Write a program to print all elements in a queue.

i) in reverse order ii) in alternate order

Ans i)
```
# include <stdio.h>
# include <stdlib.h>
Struct node
 {
  int data;
  Struct node * next;
 }
  void print rev(struct node* head)
 {
  if (head = =NULL)
     return;
   printf rev(head -> next);
   printf ("%d", head-> data);
void push (struct node* head rev, char new)
{
 Struct node* node_new = (struct node*) malloc (site 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);
   printf new (head);
   printf alternative (head);
  return 0;
}
void Print alternate (struct node* head)
{
  int count =0;
  While ( head! =NULL)
  {
     if (count % 2 ==0)
        count <<head -> data << " ";
        count ++;
        head =head -> next;
  }

i) # 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 stack record)];
   if (s==NULL)
   {
   printf ("out of space");
   }
   s->array = malloc ((size of (int))* max);
   if (s->array == NULL)
   {
   printf ("out of space");
   }
```

```
    S->capacity =max-1;
    S->tos = -1;
    return(s);
  }
int isemptys (stack *s)
{
  return s->tos==-1;
}
void Push(int x, stack s)
{
  if (isfulls (s))
     printf ("overflow");
  else
   {
     Printf ("n %d is Pushed",x);
     S->tos++;
     S->array[S->tos] =x;
   }
}
int topand POP (stack s)
{
  if (isemptys(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;
  };
typef struct queuerecord *queue;
queue createqueue(int max)
 {
   queue q;
   q=malloc(size of(struct queuerecord));
   if (q==NULL)
   printf("Error");
   q->array=malloc(size of(int)* max);
   if (q->array==NULL)
   printf("Error");
   q->capacity=max-1;
   q->front =-1;
   q->rear=-1;
   return q;
}
int is fullq(queue q)
{
  return (q->rear ==q->capacity);
}
int isempty q(queue q)
 {
  return (q->front==-1);
 }
void enqueue (queue q, int x)
{
 if (is-fullq(q))
 printf("overflows");
 else
 {
 printf("n %d is enqueued", x);
 q->rear++;
```

```
    q->array[q->rear]=x;
    if  (q->front==-1)
    q->front++;
    }
  }
void display (queue q)
{
int i;
if (isempty q(q))
{
Printf ("under-flow");
return;
for (i=q->front; irear; i++)
    Printf("%dt", q->array[i]);
}
int main ( )
{
int max, ele, i, choice, n=0, y, z;
queue q;
Printf(" n Enter the maximum elements: ");
Scanf ("%d", &max);
While (1)
{
 Printf (" 1.Insert 2.Display reversed order 3.exit");
 Printf ("n Enter the choice: ");
 Scanf ("%d", & choice);
 Switch (choice)
 {
 case 1:
 printf ("n Enter the element: ");
 Scanf ("%d", &ele);
 enqueue (q, ele);
 n++;
 break;
case 2:
printf ("n contents of the queue: ");
display (q);
for (i=0; i<capacity; i++)
{
 z = frontand delete (q), s;
  push (z, s);
 }
 q->front =-1;
 q->rear =-1;
 for (i=0; i<capacity; i++)
 {
   y = topand pop (s);
   enqueue (q, y);
 }
 printf ("n Reversed contents are : ");
 display (q);
 break;
 case 3:
   exit (0);
 }
}
}
```

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

Ans key differences between Array and Linked list.

1) An array is a data structures that contains a collection of Similar type data elements where as the linked list is considered as non-Primitive datastructure contains a collection of unordered linked elements known as nodes.

2) In the array the elements belong to indexes, i.e., if you want to get into the fourth element you have to write the variable name with its index or location with in the square bracket.

3) In a linked list through, you have to start from the head and work your way through until you get to the fourth element.

4) Accessing an element in an array is fast, while in linked list takes linear time, so it is quite a list slower.

5) Operations like insertion and deletion in array consume a lot of time. on the other hand the performance of these operations in linked list is fast.

6) In a array, memory is assigned during compile time while in linked list it is allocated during execution of runtime.

ii) write a Program to add the ~~differena~~ 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 list1 and {5,6} for list 2.

```
# include <stdio.h>
# include <stdlib.h>
int len (int a[ ])
 { int i = 0, an = 0;
   While (1)
   {
     if (a[i])
     {
       an++, i++;
     }
     else
     {
       break;
     }
   }
 return an;
}
```

```
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);
}
```