AP19110010380
CSE-H

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

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node* next;
};
struct node* head;
void Insert(int data, int n)
 {
  Node* temp= newnode();
  temp->data = data;
  temp-> next = NULL;
  if (n==1)
  {
  temp->next = head;
    head = temp;
  return;
}
```

```
Void Delete (int k)
{
 Struct Node *temp = head;
 if (k==1)
 {
  head = temp->next;
 free(temp);
  return;
 }
 Node * temp = head;
 for (int i=0; i<n-2; i++)
 {
  temp= temp -> next;
  }
  temp->next = temp->next;
   temp -> next = temp;
   }
void print();
{
 for (int i=0, i<K-2, i++)
     temp = temp->next;
     free (temp);
 }
```

```
int main()
{
 int n,x,k;
  head = NULL;
  printf ("Enter the position:");
  scanf ("%d", &n);
  scanf ("%d", &x);
  Insert (x,n);
  printf ("Enter the position to delete :");
  scanf ("%d", &k);
   Delete (k);
    printf(x);
     return;
}
```

Output :-

1. Insert
2. Delete
3. display
4. exit

Enter ch: 1

Enter inserting position: 1

Enter value: 5

Enter ch: 1

Enter inserting position: 2

Enter value: 10

Enter ch: 3

5

10.

2, Construct a new linked list by merging alternate nodes of two lists for example in list 1 we have {1,2,3} & in list2 we have {4,5,6} in the new list we should have {1,4,2,5,3,6}.

```
#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");
 }
void push(struct node **head, int data)
{
 struct node*new node =(struct node*)
                    malloc(sizeof(struct node));
```

```
newnode->data=data;
newnode->next=*head;
*head=newnode;
}
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()
{
  int keys[] = {1,4,2,5,3,6}
   int n = sizeof(Keys) / sizeof(Keys[0]);
   struct node *a = null, *b = null;
   for (i=n-1; i>=0; i=i-2)
         push(&b, Keys[i]);
   printf("First list:");
    printlist(a);
    printf("Secondlist:");
    printlist(b);
     Structnode* head = shuffle merge(a,b);
    Printf("After merge:");
     print list (head);
```

```
    return 0;
}
```

Output :~

```
Merged list is

1   4   2   5   3   6.
```

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

```
#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 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 stack dont add up to 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;
}
```

Output:-

```
Enter the size of stack : 6
Enter elements of stack: 2 6 8 10 14
Enter Sum to be checked : 60.
```

Output:-

The elements in stack do not add up to sum.

4, Write a program to print elements in a queue

i, reverse order

ii, Alternate order

i, #include<stdio.h>

#include "stack.h"

```
int main()
{
  int n, arr[50], i, j;
   Stuct stacks;
   int stack(S);
   Printf("Enter the number of elements : ");
   Scanf("%d",&n);
   for(i=0; i<n; i++)
   {
     printf("Enter values: ");
     Scanf("%d", &arr[i]);
   }
   for(i=0; i<n; i++)
   {
     insert(arr[i]);
   }
   while(i!=n)
   {
```

```
    push(&s, def(i));
    i++;
}
printf("The reverse order is :");
while (stop != -1)
{
    printf("%d", pop(&s));
}
printf("\n");
return 0;
}
```

```
#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;
}
```

Output :-

Queue Elements are : 2 5

1. Enque
2. Dequeue
3. Alternate
4. Reverse
5. Exit

Enter your choice : 4.

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

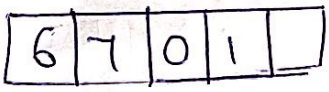
ii, Write a program to add the first element of one list to another list for example we have {1,2,3} in list 1 and {4,5,6} in list 2 we have to get {4,1,2,3} as output for list 1 and {5,6} for list 2.

i, Array Vs Linked list

1. Both are data structures and are used to store the data.

2. Cost of accessing the elements

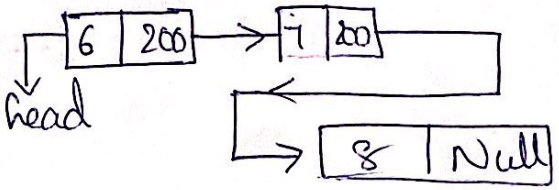
Arrays



=> It takes at constant time

Time complexeity is O(1).

Linked list



=> It depends on number of nodes in the linked list i.e O(n).

3. Memory Requirement and Utilization

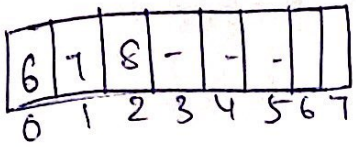
Array

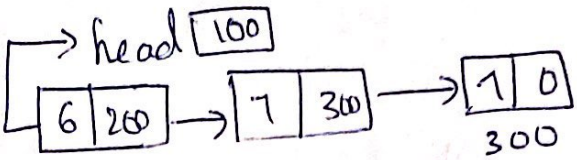
=> Ineffective in memory utilization

Linked list

=> It is in dynamic size.

Ex:-





8 X 4 = 32 bytes
used = 12

⇒ Requires less memory

8 x 3 = 24 bytes

⇒ Require more memory.

4. Cost of insertion and delete

	Array	Linked list
Beginning	O(n)	O(1)
At end	O(1)	O(n)
ith position	O(n)	O(n)

5. Easy Use and operations

Array

⇒ Easier to use

⇒ Linear and Binary

Linked lists

⇒ Less Easier

⇒ Linear

ii)

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node*next;
};
void push(struct node**head_ref; int new_data

{
 struct node* newnode = (struct node*)
                  malloc(sizeof(struct node));

 newnode->data =newdata;
 newnode-> next = (&head_ref);
 (*head_ref) = newnode;
}
void printlist(struct node* head)
{
 struct node* temp=head;
 while (temp!=NULL)
 {
  printf("%d",temp->data);
   temp=temp->next;
 }
 printf("\n");
}
```

Output :-

Enter number of nodes in list 1, list 2 : 2,3

First list is: 3 → 2 → 1 → Second list is: 2 → 1 →