Assignment-4

. K.LAHARI
AP19110010548
CSE-G.

1. write a program to insert and 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;
};
display (struct node *head)
{
if (head == NULL)
{
printf ("NULL \n");
}
else
{printf ("%d \n", head->data);
display (head->next);
}
}
del (struct node* before_delete)
{
struct node* temp;
temp = before_delete->next;
```

```
before_delete->next = temp->next;
free(temp);
}
struct node *front (struct node* head,int value)
{
struct node *f;
f=malloc (size (structnode));
f -> data=value;
f -> next=head;
return (f);
}
end (struct node*head, int value)
{
struct node *f, *q;
f=malloc (size (struct node));
f -> data=value;
f -> next=Null;
q =head;
while (q->next!=NULL)
{
q=q->next;
}
q->next=f;
}
after (structnode *b, int value)
{
if (b->next!=NULL).
```

```
{
struct node *f;
f = malloc (size (struct node));
f -> data = value;
f -> next = b -> next;
b -> next = f;
}
else
{
printf ("use end function to insert.\n");
}
}
int main()
{
struct node *prev, *head, *f;
int b, i
printf ("No. of elements");
scanf ("%d", &b);
head = NULL;
for (i=0; i<b; i++)
{
f = malloc (size of (struct node));
scanf ("%d", &f -> data);
f -> next = NULL;
if (head = NULL)
head = f;
```

```
else
prev -> next = f;
prev = f;
}
head = front (head, 10);
end (head, 15);
after (head -> next -> next, 20);
delete (head -> next);
delete (head -> next -> next);
display (head);
return 0;
}
```

Out Put:

(Enter) No. of Elements : 4.

1
2
3
4
10
1
20
3
4
15.
NULL.

2. New linked list by merging Alternate Nodes

```
#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
                                    (struct Node));
 new_node -> data = new_data;
 new node -> next = (* head_ref);
 (* head_ref) = new code;
}
void printlist (struct Node* head)
{
struct Node * temp = head;
 while (temp! = NULL)
{
printf ("%d", temp -> data);
 temp = temp -> next;
}
printf ("\n");
}
```

```
void merge (struct Node *p, struct Node **q)
{
struct Node *f_curr = f, *q_curr = *q;
struct Node *p_curr, *q_next;
while (f_curr != NULL && q_curr != NULL)
f_next = f_curr -> Next;
q_next = q_curr -> next;
q_curr -> next = f_next;
f_curr -> next = q_curr;
f_curr = q_next;
q_curr = q_next;
}
*q = q_curr;
}
int main()
{
struct Node *p = NULL; *q = NULL;
push (&f, 5);
push (&f, 2);
push (&f, 1);
printf (" ist linked list \n");
printf list (f);
push (&q, 2);
push (&q, 1);
push (&q, 0);
```

```
printf ("2nd linked list \n");
Print list (g);
merge (f, &g);
Printf ("changed linked list \n");
Print list (f);
return 0;
}
```

Out Put:

1st linked list

1 2 5

2nd linked list

0 1 2

changed linked list

1 0 2 1 5 2

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

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
define max 1000
```

```
type of struct stack{
    int arr[max];
    int top;
}stack;
void push (stack*s , int data){
    if (s->top>=max-1){
        exit (0);
    }
  s->top++;
  s->arr[s->top]=data;
}
int POP(stack*s) {
   if (s->top<0) return INT_MIN;
   int temp=s->arr[s->top];
   s->top--;
   return temp;
}
Void display (stack s){
 int i;
 for (i=s.top;i>-1;i--){
  Printf ("%d", s.arr[i]);
}
 printf ("\n");
}.
```

```
void sumk (stack s1, stack v, int k){
if (k==0){
 display (v);
return;
}
if (s1.top == -1). return;
int temp = pop (&s1);
 sumk (s1, v, k);
stack v1 = v;
push (&v1, temp);
sumk (s1, v1, k-temp);
}
int main (int argc, char const *argv[]){
 stack arr, v;
 arr.top = -1;
 v.top = -1;
 int expected, n, num
 printf ("enter the number of element in stack\n");
  scanf ("%d", &n);
  while (n-) {
```

```
Printf("numbe \n");
scanf("%d", &num);
 Push (&arr, num);
}
Printf ("enter expected value \n");
scanf ("%d", &expected);
 n = arr.top+1;
 sumk (arr, v, expected);
 return 0;
}
```

out put

enter the number of element you want in the stack.

3

r

number 0 1 2

expected value : 3

0 3 2 1

1 2 3 0

3.

4. i) Elements in a queue in Reverse order.
ii) Alternative order.

```
#include <stdio.h>
# include <stdlib.h>
struct node
{.
```

```
    int data;
    Struct Node *next;
};node;
Struct Queue{
    node *front, *rear;
};queue;
node* newnode (int k)
{
    node* temp = (node*) malloc(sizeof(node));
    temp->data =k;
    temp -> next =NULL;
    return temp;
}
queue create queue()
{
    queue q;
    q->front = q->rear =NULL;
    return q;
}
void enqueue (queue *q, int k).
{
    node* temp = new Node (k);
    if (q -> rear ==NULL) {
        q ->front = q -> rear = temp;
        return;
```

```
}
q->rear->next = temp;
q->rear = temp;

}
void displayAlt(queue q){
    while(q.front != NULL){
        printf("%d->->", q.front->data);
        if(q.front->next != NULL)
        q.front->next->next;
        else break;
    }
    printf("NULL\n");

}void displayRev(queue q){
    if(q.front == NULL){
        printf("NULL");
        return;

    }
    int temp = q.front->data;
    q.front = q.front->next;
    displayRev(q);
    printf("<-%d", temp);

}int main()
{.
```

```
Queue q = create queue();
int n, num;
printf("enter the no. of element in the queue \n");
scanf("%d", &n);
while (n--){
        printf("number\n")
        scanf("%d & num);
        enque (&q, num);
}
display Rev(q);
printf("\n");
display Alt(q);
return 0;
}
```

out put

Enter the no. of element in the queue: 5

number
0

number
1

number
2

number
3

number
4

NULL ← 4 ← 3 ← 2 ← 1 ← 0

0 → 2 → 4 → NULL.

5. difference b/w array and linked list:

Array	linked list.
1. fixed size, Resizing is expensive	1. Dynamic size.
2. Sequential access is fast	2. sequential aces is slow
3. Random access	3. No random acess
4. Insertion and deletion take more time	4. Ingertion and deletion process take less time.
5. Deleting an element from an array is not possible.	5. Deleting an element is possible.
6. occupies less memory.	6. occupies more memory.

5(ii) Add the first element of 1st list to another list.

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
 int data;
 struct Node *next;
};
void push (struct Node **head_ref, int new_data)
{
struct Nod* new_data = (struct Node*) malloc (size
                                    (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");
}
void merge (struct Node *f, struct Node **q)
{
Struct Node *f_curr=f, *q_curr= *q;
struct Node *f_next, *q_next;
while (f_curr != NULL && q_curr != NULL)
f_next = f_curr -> next
q_next = q_curr -> next;
q_curr -> next = f_next;
f_curr -> next = q_curr;
f_curr = f_next;
q_curr = q_next;
}
*q = q_curr;
}
int main()
{
struct node *f = NULL, *q = NULL;
push (&f, 2);
push (&f, 8);
push (&f, 7);
push (&f, 1);
printf ("1st linked list\n");
```

```
Printlist(f);
Push(&q,0);
Push(&q,1);
Push(&q,3);
Push(&q,4);
Push(&q,3);
Printf("2nd linked list \n");
Printlist(q);
merge(f,q);
Printf("changed 1st linked list \n");
printlist(f);
Print("changed 2nd linked list \n");
print list(q);
```

output

1st linked list.

1 7 8 2 $

2nd linked list.

4 3 1 0 3

changed 1st linked list.

1 4 7 3 8 1 2 0 $

changed 2nd linked list.

3.