ASSIGNMENT-4

1. Program to inserting a not at any Given position of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
Struct node{
  int data;
Struct node * next;
}*head;
Void Create_list (int n);
Void insert_at_given_Position (int data, int Position);
Void displaylist ();
int main()
{
  int n, data, pos;
  printf ("\n Enter the total number of nodes:");
  Scanf ("%d", &n);
Create_list (n);
Print f("\n The list is\n");
display list ();
```

```
printf("\n Enter the position : ");
scanf("%d", &pos);
printf("\n Enter the data to insert at position %d
        of the list: ", pos);
scanf("%d ", &data);
insert_at_given_Position(data, pos);
printf("\n\n The list is \n");
display list();
return 0;
}
void create_list(int n)
{
struct node *newnode, *temp;
int data, i;
head = (struct node *) malloc(sizeof(struct node));
if(head == NULL)
{
printf("unable to allocate memory.");
}
```

```
else
{
printf("\n Enter the data of node 1 : ");
scanf("%d", &data);
head->data = data;
head->next = NULL;
temp = head;
for(i=2; i<=n; i++)
{
newnode = (struct node *) malloc (sizeof(struct node));
if(newnode == NULL)
{
printf("unable to allocate memory.");
break;
}
else
{
printf("\n Enter the data of node %d:", i);
scanf("%d", &data);
newnode->data = data;
newnode->next = NULL;
temp->next = newnode;
```

```
        temp= temp->next;
        }
      }
    }
  }
Void insert_at_given_position (int data, int position)
{
  int count = 0;
    Struct node * new Node * temp;
  new Node = (Struct node)* malloc (Size_of (Struct node));
  if (new Node == Null)
  {
    print f ("unable to allocate memory.");
  }
  else
  {
    temp = head;
  while (temp-> next != Null and ((position-1) != count))
  {
    temp = temp -> next;
  } count = count+1;
```

```
if (position-1)== count)
{
newnode->data = data;
new node ->next = temp->next;
temp->next = new node;
}
}
}
void display list()
{
struct node * temp;
if (head == NULL)
{
printf(" list is empty!!");
}
else
{
temp =head;
while (temp != NULL)
{
printf("%d \t", temp->data);
temp = temp->next;
} printf("\n")
```

(2) deleting a node from the beginning of the linked

```
#include <stdio.h>
#include <stdlib.h>
Struct node {
    int data;
    Struct node* next;
} *head;
Void Create List (int n);
Void delete firstnode ();
Void display list();
int main()
{
int n, choice
printf ("Enter the total number of nodes: ");
Scanf ("%d, &n);
Create List (n);
Print ("In Data in the list \n");
display List ();
```

```
printf("\n Press 1 to delete first node:");
scanf("%d", &choice);
if (choice == 1)
    delete first node();
printf("\n Data in the list \n");
display list();
return 0;
}
void create list(int n)
{
struct node * new node * temp;
int data, i;
head = (struct node *) malloc (sizeof (struct node));
if (head == NULL)
{
  printf ("unable to allocate memory.");
}
else
{
   printf(" Enter the data of node 1:");
   scanf ("%d", &data);
```

```
head->data=data;
head->next=NULL;
temp=head;
for(i=2;i<=n;i++)
{
newnode=(struct node*)malloc(sizeof(struct node));
if(newnode==NULL)
{
printf("unable to allocate memory.");
break;
}
else
{
printf("Enter the data of node %d",i);
scanf("%d",&data);
newNode->data=data;
newnode->next=NULL;
temp->next=newnode;
temp=temp->next;
}
}
```

```
printf("Singly Linked List Created Successfully\n");
    }
}
void deleteFirstNode()
{
struct node *toDelete;
if(head == NULL)
{
printf("List is already empty.");
}
else
{
toDelete = head;
head = head->next;
printf("\n Data deleted = %d\n", toDelete->data);
free(toDelete);
printf("Successfully deleted first node from U2\n");
    }
}
```

```
void displaylist()
{
  struct node *temp;

  if (head == NULL)
  {
    printf("list is empty.");
  }
  else
  {
    temp = head;
    while (temp != NULL)
    {
      printf("Data = %d\n", temp->data);
      temp = temp->next;
    }
  }
}
```

(3) deleting a node from the end of the LinkedList.

```
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node* next;
} *head

void create_list(int n);
void delete_lastnode();
void displaylist();

int main()
{
    int n, choice;
    printf("Enter the total number of nodes:");
    scanf("%d", &n);
    create_list(n);

printf("\nData in the list \n");
displaylist();
```

```
printf("\n Press 1 to delete last node: ");
Scanf("%d", &choice);

if (choice == 1)

    deletelastnode();

printf("\n Data in the list \n");

displaylist();

    getchar();

}

void createlist(int n)

{

  Struct node *newnode, *temp;

    int data, i;

    head = (Struct node *) malloc (sizeof(struct node));

if (head == NULL)

{

    printf("unable to allocate memory.");

}

else

{
```

```
printf ("Enter the data of node 1:");
scanf ("%d", &data);

head -> data = data;
head -> next = NULL;

temp = head;

for (i=2; i<=n; i++)
{
newNode = (struct node *) malloc (sizeof (struct node));

if (newNode == NULL)
{
printf (" unable to allocate memory.");
break;
}
else
{
printf ("Enter the data of node %d:", i);
scanf ("%d", &data);

newNode -> data = data;
newNode -> next = NULL;
temp -> next = newNode;
temp = temp -> next;
```

```
    }
}
printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");
    }
}
Void deletelastNode()
{
  Struct node *toDelete, *SecoundlastNode;
  if (head == NULL)
  {
    printf("list is already empty.");
  }
  else
  {
    toDelete = head;
    SecoundlastNode = head;
  while (toDelete->next != NULL)
  {
      Secoundlastnode = toDelete;
    }  toDelete = toDelete->next;
```

```
if (toDelete == head)
{
    head = NULL;
}
else
{
    secondlastNode->next = NULL;
}
free(toDelete);
printf("Successfully Deleted last node of list\n");
}
}
void displayList()
{
struct node *temp;
if (head == NULL)
{
    printf("List is empty.");
}
else
{
    temp = head;
    while (temp != NULL)
```

```
{
    printf("data = %d\n", temp->data);
    temp = temp->next;
    }
  }
}
```

(4) I am very sure that everyone is able to find middle index of array once you know start index and end index of array, but there are certain benefits of using start + (end-start)/2 over (start+end)/2,

The very first way of finding index is

mid = (start + end)/2

But there is problem with this approach, what if value of start or end or both is INT_MAX, it will cause integer overflow.

the better way of calculating mid index is

mid = start + (end - start)/2

Algorithm :

(3) ternary Search (array, start, end, key)

Begin

if start <= end then

mid first := start + (end - start)/3

mid second := mid first + (end - start)/3

if array [mid first] = key then

return mid first

if array [mid second] = key then

return mid second

if key < array [mid first] then

Call ternarySearch (array, start, mid first - 1, key)

if key > array [mid, second] then

Call ternary search (array, mid first + 1, end, key)

else

Call ternary search (array, mid first + 1, mid second - 1, key)

else

return invalid location

End.