(14) WAP to implement single Linked list the following operations
A, Create a linked list
b, insertion of a node at first position.
* End position.
* display.

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

struct node {
    int data;
    struct node *next;
};
struct * Node *head = Null;

struct node* createNoode(int data)
{
    struct node* newNode = (struct Node*)
malloc(sizeof(struct node));
        newNode -> data = data;
        newNode -> next = Null;
        return newNode; }

void insertAtBeggining(int data)
{ struct Node* newNoode = createNode(data
    newNode -> next = head;
    head = newNode;
}
void insertAtPosition(int data, int position){
struct Node* newNoode = createNode(data);
if(Position == 1) {
insertAtBeginning(data);
return;
}
```

```
struct Node* temp = head;
for(int i=1; i< position-1 && temp != NULL; i++)
  { temp = temp->next; }
  if (temp == null)
    {
      printf("position out of bounds\n");
      return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
  }*/

void insertAtEnd(int data) {
   struct Node* newNode = CreateNode(data
   if (head == NULL) {
    head = newNode;
    return; }
  struct Node* temp = head;
  while (temp->next != NULL)
  { temp = temp->next;
    }
      temp->next = newNode;
  }

void deleteFirst()
  {
    if (head == NULL) {
      printf("List is empty\n");
      return;
    }
  struct Node* temp = head;
    head = head->next;
    free(temp);
  }
```

```
/* void deleteAtPosition(int position)
{ if (head == NULL) {
    printf("List is empty\n");
    return; }
  struct Node* temp = head;
  if(position == 1) {
    head = temp->next;
    free(temp);
    return; }
  for(int i = 1; i < position - 1 && temp != NULL;
  ) { temp = temp->next; }
    if (temp == NULL || temp -> next == NULL)
      printf("Position out of bounds\n");
      return;
    }
    struct Node * nodeToDelete = temp->ne
    temp -> next = nodeToDelete -> next;
    free(nodeToDelete);
  } */
  void deleteLast() {
  if (head == NULL) {
    printf("List is empty\n");
    return; }
  if (head -> next == NULL) {
    free(head)
    head = NULL;
    return; }
  struct Node* temp = head;
  while (temp -> next -> next != NULL){
    temp = temp -> next; }
    free(temp->next);
    temp -> next = NULL;
}
```

```
void displayList()
{
 Struct Node* temp = head;
 if (temp==NULL) {
  printf ("List is empty\n");
  return; }
 while (temp != NULL) {
  Printf("%d ->", temp -> data);
   temp = temp -> next;
 }
 Print ("NULL\n");
}

int main()
{
 int choice, data, position;
while (1)
{
   printf(" Enter 1 to insert the element
      at front");
   Printf (" Enter 2 to insert the ele
     -ent of End:");
   Printf (" Enter 3 to Delete the ele
     -ment at front:");
   Printf(" Enter 4 to Delete the elem
     -ent at the end:");
   Printf (" Enter 5 to Display the ele
   -ments);
   Printf (" Exit");

   Scanf ("%d", &choice);
```

```
switch(choice)
{
 case 1:
 printf(" Enter data: ");
 scanf("%d", &data);
 insertatBeggining(data);
 break;
 case 2;
 Printf(" Enter data: ");
 scanf("%d", &data);
 insertatEnding(data);
 break;
 case 3:
 deletefirst();
 printf("deleted list: %d");
 break;
 case 4:
 deleteLast();
 printf("deleted list: %d",
 break;
 case 5:
 display();
 Printf("
 break;
 case 6:
 exit();
 default:
 printf(" Please enter Correct option");
 }
 return 0;
}
```

Proud

28/10

O/P

Menu:
1. Inserting at beggining
2. Insert at ending
3. Delete at beggining
4. Delete at ending
5. Display the list
6. exit

Enter your choice: 1
Enter data to insert at beggining

Enter your choice: 2
Enter data to insert at Ending

Enter your choice: 5
Linked List: 11 -> 22 ->

Enter your choice: 3
First node deleted

Enter your choice: 4
Last node deleted

O/P

Enter your choice: 5
Linked list: List is Empty

Enter your choice: 8
Please enter correct choice