LAB Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int info;
    struct node *link;
};
typedef struct node *NODE;
NODE getnode(){
 NODE x;
 x = (NODE) malloc (sizeof (struct node));
 if (x == NULL){
     printf("ee memory full\n");
     exit(0);
 }
 return x;
}
void freenode (NODE x){
    free (x);
}
NODE insert_front (NODE first, int item){
 NODE temp;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
  if (first == NULL)
      return temp;
  else
  temp -> link = first;
  first = temp;
  return first;
}
NODE delete_front (NODE first){
   NODE temp;
```

```
if (first == NULL) {
    printf("\n List is empty Cannot decline\n");
    return first;
}
temp = first;
temp = temp -> link;
printf("\n Item deleted at front end is %d\n",
                                    first->info);
free(first);
return temp;
}
NODE insert_rear (NODE first, int item) {
  NODE temp, cur;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
  if (first == NULL)
    return temp;
  cur = first;
  while (cur -> link != NULL)
  cur = cur -> link;
  cur -> link = temp;
  return first;
}
NODE delete_rear (NODE first) {
  NODE cur, prev;
  if (first == NULL) {
      printf ("\n List is empty Cannot delete\n");
      return first;
  }
  if (first -> link == NULL) {
      printf ("\n Item deleted is %d\n", first->info);
      free (first);
      return NULL;
  }
```

```
    prev = NULL;
    cur = first;
    while (cur -> link != NULL){
    prev = cur;
     cur = cur -> link;
   }
   printf ("\n Item deleted at rear end is %d",
                                    cur->info);
    free (cur);
    prev -> link = NULL;
    return first;
}
NODE insert_pos (int item, int pos, NODE first){
  NODE temp, cur, prev;
  int count;
  temp = getnode ();
  temp -> info = item;
  temp -> link = NULL;
  if (first == NULL && pos == 1){
        return temp;
  }
  if (first == NULL){
        printf("\n Invalid position\n");
        return first;
  }
  if (pos == 1){
     temp -> link = first;
     first = temp;
     return temp;
   }
  count = 1;
   prev = NULL;
   cur = first;
  while (cur != NULL && count != pos){
```

```
        prev = cur;
        Cur = cur -> link;
        Count ++;
    }
    if (
        Count == pos) {
        prev -> link = temp;
        temp -> link = cur;
        return first;
    }
    printf ("Invalid position \n");
        return first;
}
NODE delete_pos (int pos, NODE first) {
        NODE cur;
        NODE prev;
        int count, flag = 0;
        if (first == NULL || pos < 0) {
            printf ("Invalid position \n");
            return NULL;
        }
        if (pos == 1) {
            Cur = first;
            first = first -> link;
            freenode (cur);
            return first;
        }
        prev = NULL;
        Cur = first;
        Count = 1;
        while (Cur != NULL) {
            if (count == pos) {
                flag = 1;
                break;
```

```
        count++;
        prev = cur;
        cur = cur -> link;
    }
    if (flag == 0){
        printf("Invalid position\n");
        return first;
    }
    printf("Item deleted at given position
                is %d\n", cur -> info);
    prev -> link = cur -> link;
    freenode(cur);
    return first;
}
void display(NODE first){
    NODE temp;
    if (first == NULL)                    item\n
        printf("List empty Cannot display
    for(temp = first; temp != NULL; temp = temp
        printf("%d\n", temp -> info);
    }
}
void main()
{
    int item, choice, key, pos;
    int count = 0;
    NODE first = NULL;
    for(;;){
        printf("\n1. Insert rear \n2. Delete rear \n3.
            Insert front \n4. Delete front \n5: Insert
            info position \n6. Delete info position
            \n7. Display list \n8. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice){
```

```
Case 1: printf ("Enter the item at rear
                end \n");
        scanf ("%d", &item);
        first = insert_rear (first, item);
        break;
Case 2: first = delete_rear (first);
        break;
Case 3: printf ("In Enter the item at
                front end \n");
        scanf ("%d", &item);
        first = insert_front (first, item);
        break;
Case 4 = first = delete_front (first);
        break;
Case 5: printf ("Enter the item to be
inserted at any given position \n");
        scanf ("%d", &item);
        printf ("Enter the position \n");
        scanf ("%d", &pos);
        first = insert_pos (item, pos, first);
        break;
Case 6: printf ("Enter the position \n");
        scanf ("%d", &pos);
        first = delete_pos (pos, first);
        break;
    }
}
}
```