Lab 7:

```
#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("mem 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;
    temp -> link = first;
    first = temp;
    return first;
}
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 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 ("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_front (NODE first)
{
  NODE temp;
  if (first == NULL)
  {
    printf("list is empty cannot delete\n");
    return first;
  }
  temp = first;
  temp = temp -> link;
  printf("item deleted at front-end is = %d\n", first->info);
```

```
    free (first);
    return temp;
}
NODE delete_rear (NODE first)
{
    NODE cur, prev;
    if (first == NULL)
    {
        printf ("List is empty cannot delete\n")
        return first;
    }
    if (first -> link == NULL)
    {
        printf (" 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 ("item deleted at rear-end is %d", cur->info);
```

```
        free(cur);
        prev->link = NULL;
        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)
    printf (" list empty cannot display items \n");
    for (temp = first; temp != NULL; temp = temp -> link)
    {
        printf ("%d \n", temp -> info);
```

```
    }
}

int main()
{
    int item, choice, pos;
    NODE first = NULL;
    for(;;)
    {
        printf("\n1. Insert Front\n2. Insert Rear\n3. Insert
            Pos\n4. Delete Front\n5. Delete Rear\n6. Delet
            Pos\n7. Display\n8. Exit\n");
        printf("Enter the choice\n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("enter the item at front-end\n");
                scanf("%d", &item);
                first = insert_front(first, item);
                break;
            case 2: printf("enter the item at rear-end\n");
                scanf("%d", &item);
                first = insert_rear(first, item);
                break
            case 3: printf("enter the item to be inserted at
                    given pos\n");
```

```
            scanf("%d", &item);
            printf("enter pos\n");
            scanf("%d", &pos);
            first = insert_pos(item, pos, first);
            break;
        case 4: first = delete_front(first);
            break;
        case 5: first = delete_rear(first);
            break;
        case 6: printf("enter the position\n");
            scanf("%d", &pos);
            first = delete_pos(pos, first);
            break;
        case 7: display(first);
            break;
        default: exit(0);
            break;
    }
  }
}
```