LAB PROGRAM 6(SINGLY LINKED LIST DELETE FRONT, DELETE REAR DELETE AT POS) EXECUTION

```
#include <stdio.h>

#include <conio.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 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 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("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 prev, cur;
    int count;
    if (first == NULL || pos <= 0)
    {
        printf("Invalid position\n");
        return NULL;
```

```
}
if (pos == 1)
{
    cur = first;
    first = first->link;
    printf("Item deleted is %d", cur->info);
    freenode(cur);
    return first;
}
prev = NULL;
cur = first;
count = 1;
while (cur != NULL)
{
    if (count == pos)
    {
        break;
    }
    prev = cur;
    cur = cur->link;
```

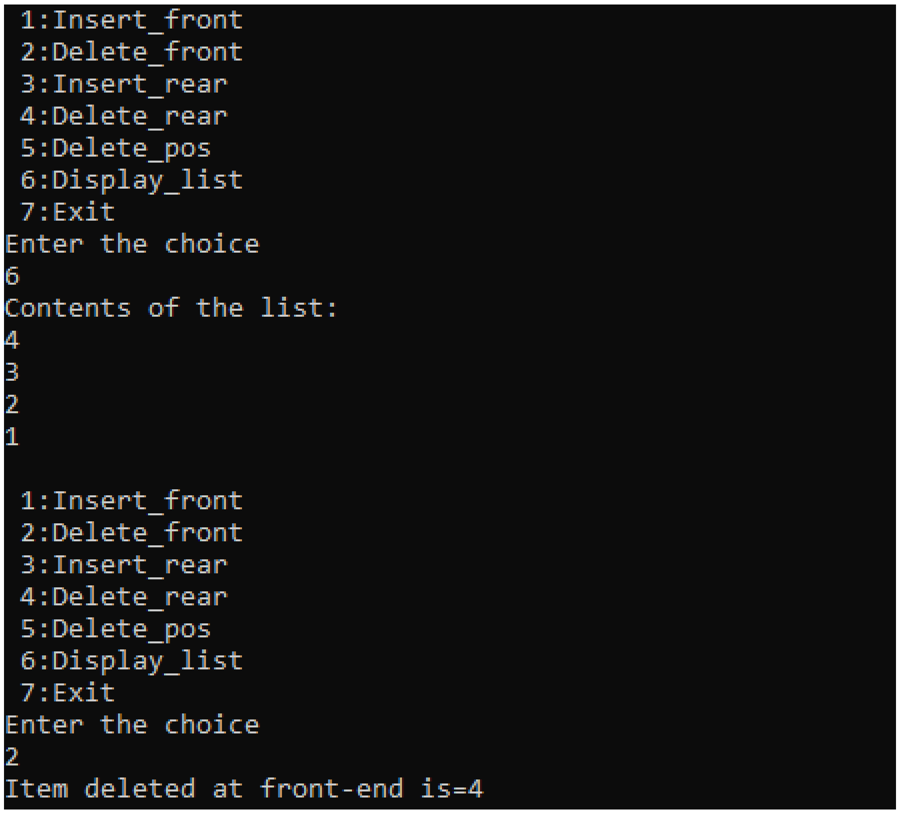
```
        count++;
    }
    if (count != pos)
    {
        printf("Invalid position\n");
        return first;
    }
    prev->link = cur->link;
    printf("Item deleted is %d", cur->info);
    freenode(cur);
    return first;
}
void display(NODE first)
{
    NODE temp;
    if (first == NULL)
        printf("List empty cannot display items\n");
    else
        printf("Contents of the list:\n");
    for (temp = first; temp != NULL; temp = temp->link)
    {
        printf("%d\n", temp->info);
    }
}
void main()
{
    int item, choice, pos;
    NODE first = NULL;

    for (;;)
    {
        printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n
4:Delete_rear\n 5:Delete_pos\n 6:Display_list\n 7: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:
            first = delete_front(first);
            break;
        case 3:
            printf("Enter the item at rear-end\n");
            scanf("%d", &item);
            first = insert_rear(first, item);
            break;
        case 4:
            first = delete_rear(first);
            break;
        case 5:
            printf("Enter the position:\n");
            scanf("%d", &pos);
            first = delete_pos(pos, first);
            break;
        case 6:
            display(first);
```

```
            break;
        case 7:
            exit(0);
            break;
            default:printf("Invalid choice\n");
        }
    }
}
```

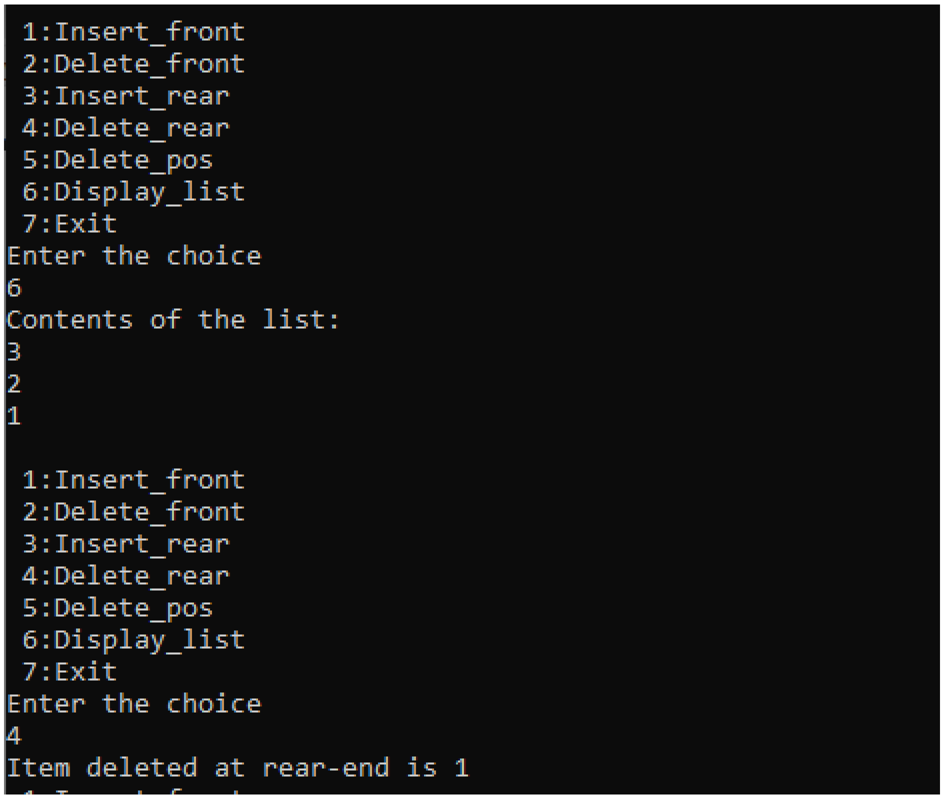
OUTPUT:

1.delete front and delete rear



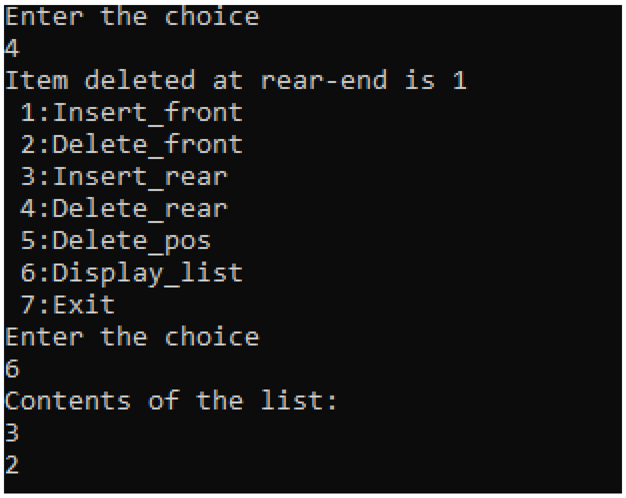
```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
6
Contents of the list:
4
3
2
1

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
2
Item deleted at front-end is=4
```



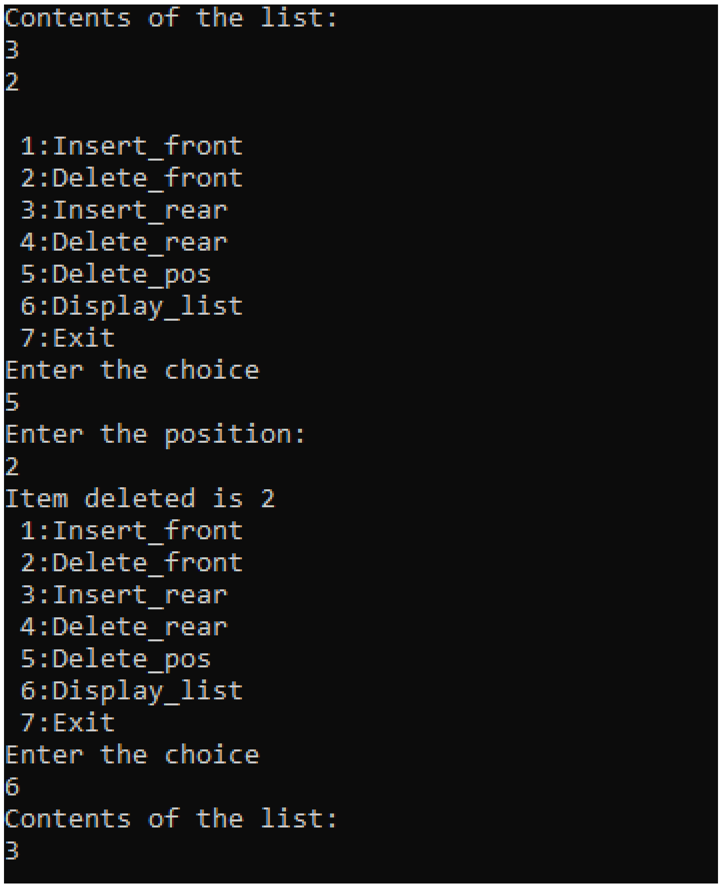
```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
6
Contents of the list:
3
2
1

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
4
Item deleted at rear-end is 1
```



```
Enter the choice
4
Item deleted at rear-end is 1
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
6
Contents of the list:
3
2
```

2.delete pos



```
Contents of the list:
3
2

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
5
Enter the position:
2
Item deleted is 2
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
6
Contents of the list:
3
```