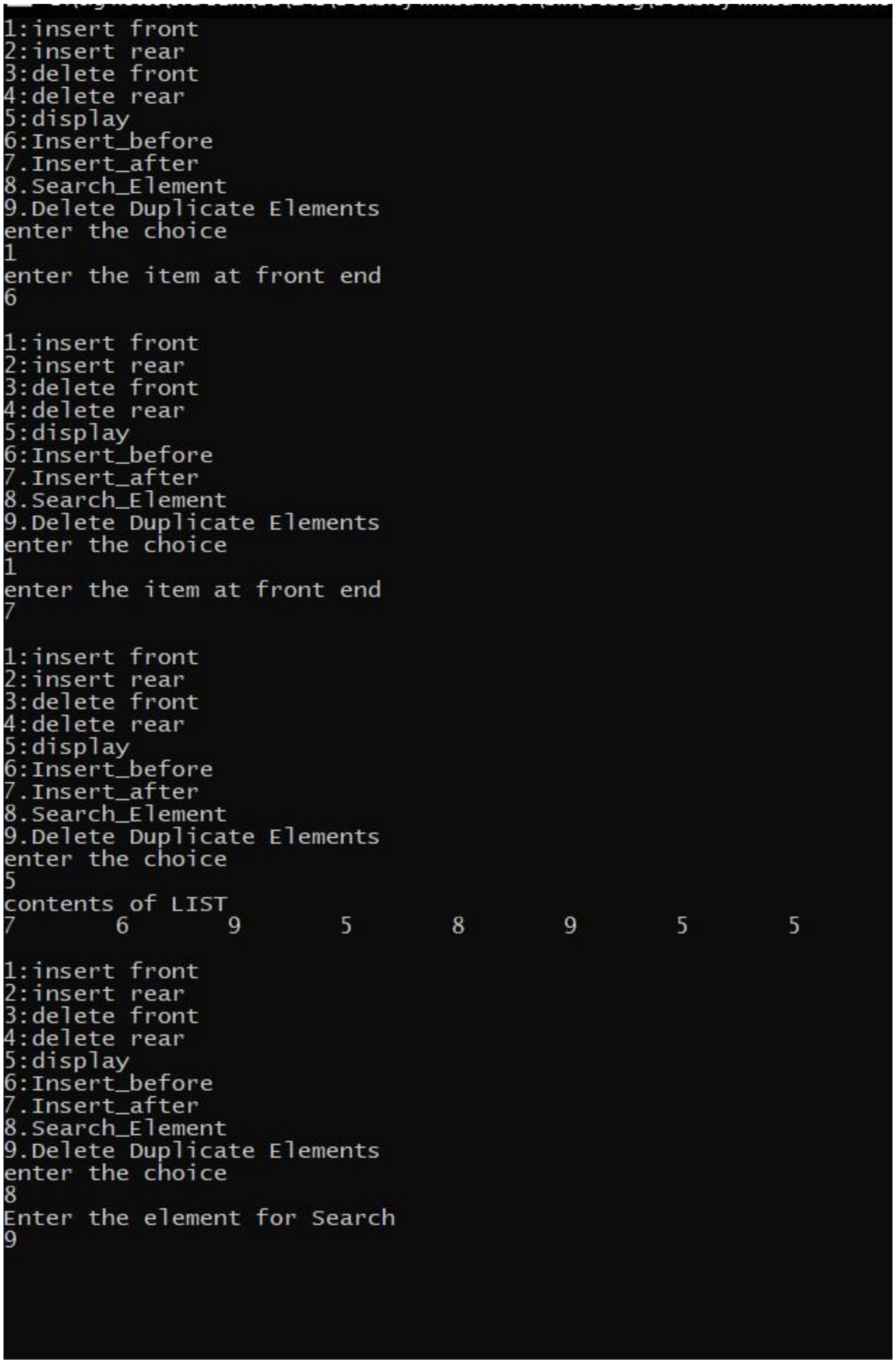
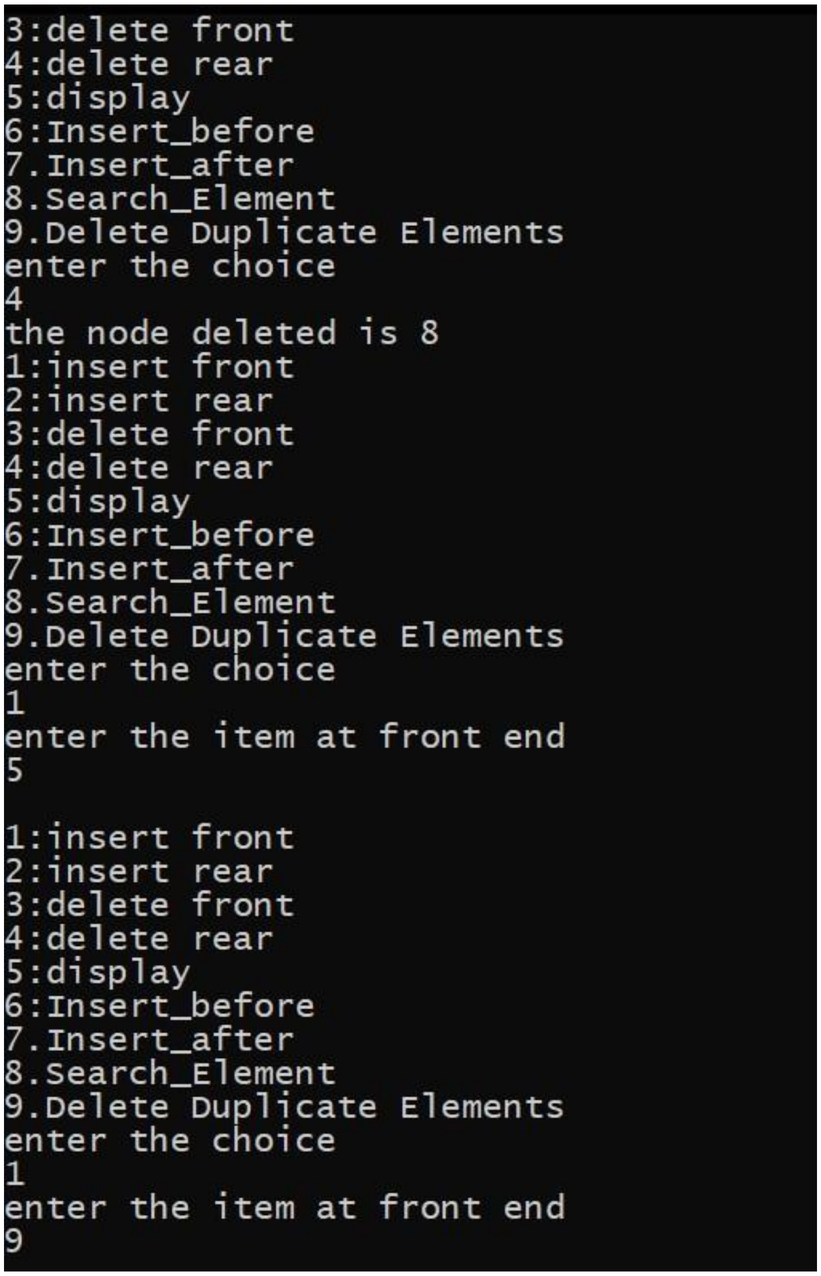
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
       int info;
       struct node *llink;
       struct node *rlink;
       };
typedef struct node *NODE;
NODE getnode()
{
      NODE x;
      x=(NODE)malloc(sizeof(struct node));
      if(x==NULL)
      {
           printf("mem full\n");
           exit(0);
           }
      return x;
      }
void freenode(NODE x)
{
      free(x);
}

NODE dinsert_front(int item,NODE head)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
cur=head->rlink;
head->rlink=temp;
temp->llink=head;
temp->rlink=cur;
cur->llink=temp;
return head;
}
NODE dinsert_rear(int item,NODE head)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
cur=head->llink;
head->llink=temp;
temp->rlink=head;
temp->llink=cur;
cur->rlink=temp;
return head;
}
NODE ddelete_front(NODE head)
{
NODE cur,next;
```

```
if(head->rlink==head)
{
printf("list empty\n");
return head;
}
cur=head->rlink;
next=cur->rlink;
head->rlink=next;
next->llink=head;
printf("the node deleted is %d",cur->info);
freenode(cur);
return head;
}
NODE ddelete_rear(NODE head)
{
NODE cur,prev;
if(head->rlink==head)
{
printf("list empty\n");
return head;
}
cur=head->llink;
prev=cur->llink;
head->llink=prev;
prev->rlink=head;
printf("the node deleted is %d",cur->info);
freenode(cur);
return head;
}


NODE insert_before(int item,NODE head)
{
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("list empty\n");
return head;
}
cur=head->rlink;
while(cur!=head)
{
if(item==cur->info)break;
cur=cur->rlink;
}
if(cur==head)
{
 printf("key not found\n");
 return head;
 }
 prev=cur->llink;
 printf("enter element before %d=",item);
 temp=getnode();
 scanf("%d",&temp->info);
 prev->rlink=temp;
 temp->llink=prev;
 cur->llink=temp;
 temp->rlink=cur;
 return head;
}


NODE insert_after(int item,NODE head)
{
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("list empty\n");
return head;
}
cur=head->rlink;
while(cur!=head)
{
if(item==cur->info)break;
cur=cur->rlink;
}
if(cur==head)
{
 printf("key not found\n");
 return head;
 }
 prev=cur->rlink;
 printf("enter element after %d = ",item);
 temp=getnode();
 scanf("%d",&temp->info);
 cur->rlink = temp;
 temp->llink = cur;
 temp->rlink = prev;
 prev->llink = temp;
 return head;
}

NODE search(int item,NODE head){

    if(head->rlink==head)
{
printf("list empty\n");
return head;
}

NODE cur;
cur = head->rlink ;
while(cur!=head){
    if(item==cur->info)break;
    cur = cur->rlink;
}
if(cur==head){
    printf("\nElement not found \n");
    return head;
}
printf("\nSearch Successful Element found \n");
return head;
}

/*
NODE DeleteALL(NODE head){

    NODE cur ,prev;

    if(head->rlink==head)
 {
 printf("list empty\n");
 return head;
 }
 while(cur!=head){
    prev = cur;
    cur = cur->
 }

}
*/


void delete_dup(NODE head)
{
    NODE cur,temp,ptr,prev;
    if(head->rlink==head)
    {
        printf("List is empty\n");
        return ;
    }
    temp=head->rlink;
    cur=head->rlink;

    while(temp!=head)
    {
        prev=cur;
        cur=temp->rlink;
        while(cur!=head){

        if(temp->info==cur->info)
        {
          ptr=cur->rlink;ptr->llink=cur->llink;
          ptr=cur->llink;ptr->rlink=cur->rlink;
          freenode(cur);
        }
        cur=cur->rlink;
        }
        temp=temp->rlink;
     }
     return ;
     }


void display(NODE head)
{
NODE temp;
if(head->rlink==head)
{
printf("List empty\n");
return;
}
printf("contents of LIST\n");
temp=head->rlink;
while(temp!=head)
{
printf("%d\t",temp->info);
temp=temp->rlink;
}
printf("\n");
}


int main()
{
NODE head,last;
int item, choice;
head=getnode();
head->rlink=head;
head->llink=head;

for(;;)
{
      printf("\n1:insert front\n2:insert rear\n3:delete front\n4:delete
rear\n5:display\n6:Insert_before\n7.Insert_after\n8.Search_Element\n9.
Delete Duplicate Elements\n");
      printf("enter the choice\n");
      scanf("%d",&choice);
      switch(choice)
      {
           case 1: printf("enter the item at front end\n");
                scanf("%d",&item);
                last=dinsert_front(item,head);
                break;
           case 2: printf("enter the item at rear end\n");
                scanf("%d",&item);
                last=dinsert_rear(item,head);
                break;
```

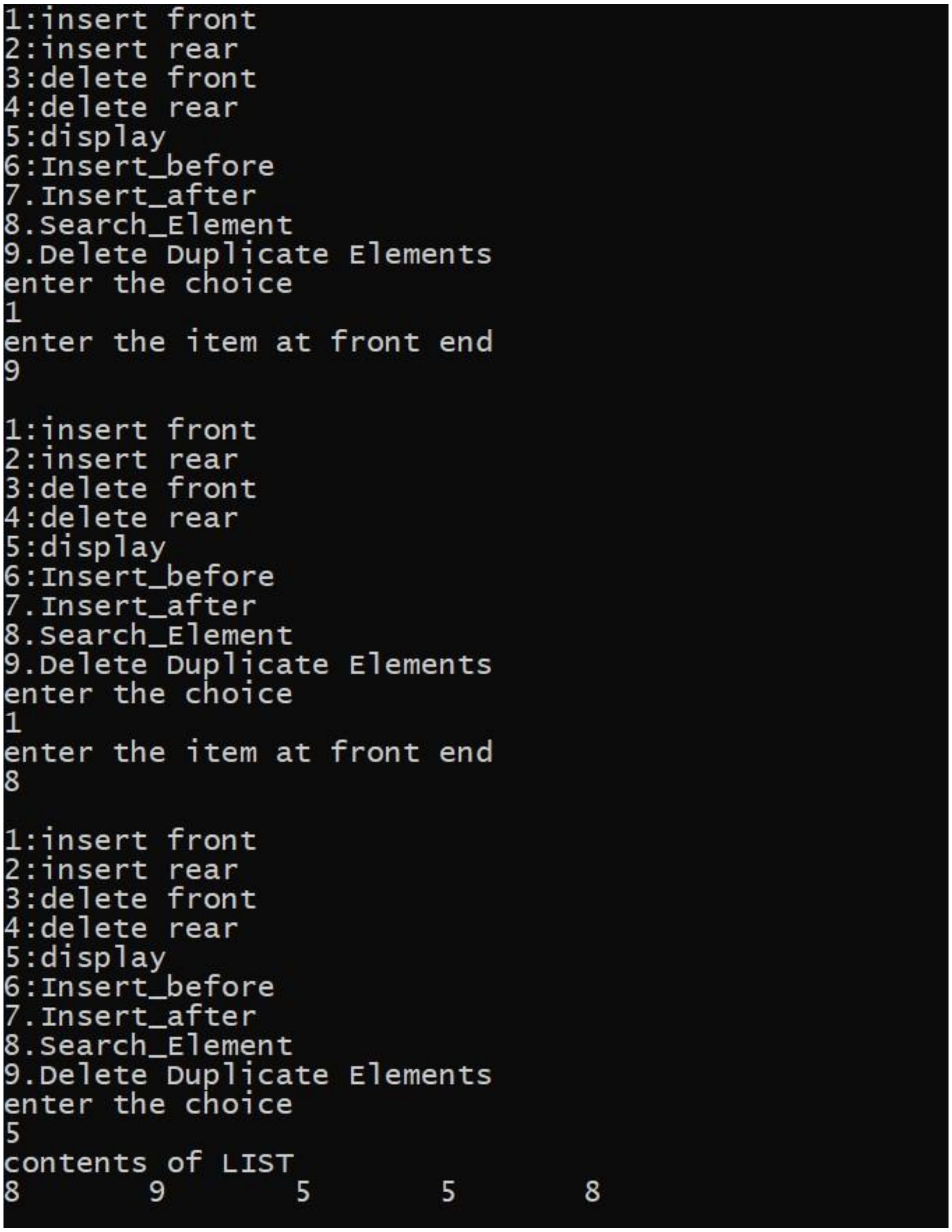
```
          case 3:last=ddelete_front(head);
               break;
          case 4: last=ddelete_rear(head);
               break;
          case 5: display(head);
               break;
       case 6:printf("enter the key item\n");
           scanf("%d",&item);
           head=insert_before(item,head);
           break;
            case 7:printf("enter the key item\n");
           scanf("%d",&item);
           head=insert_after(item,head);
           break;
           case 8 : printf("Enter the element for Search\n");
           scanf("%d",&item);
           head = search(item,head);
           break;

           case 9:delete_dup(head);
                printf("\nList after deleting duplicate elements\n");

           display(head);
           break;
          default:exit(0);
          }
     }
}
```

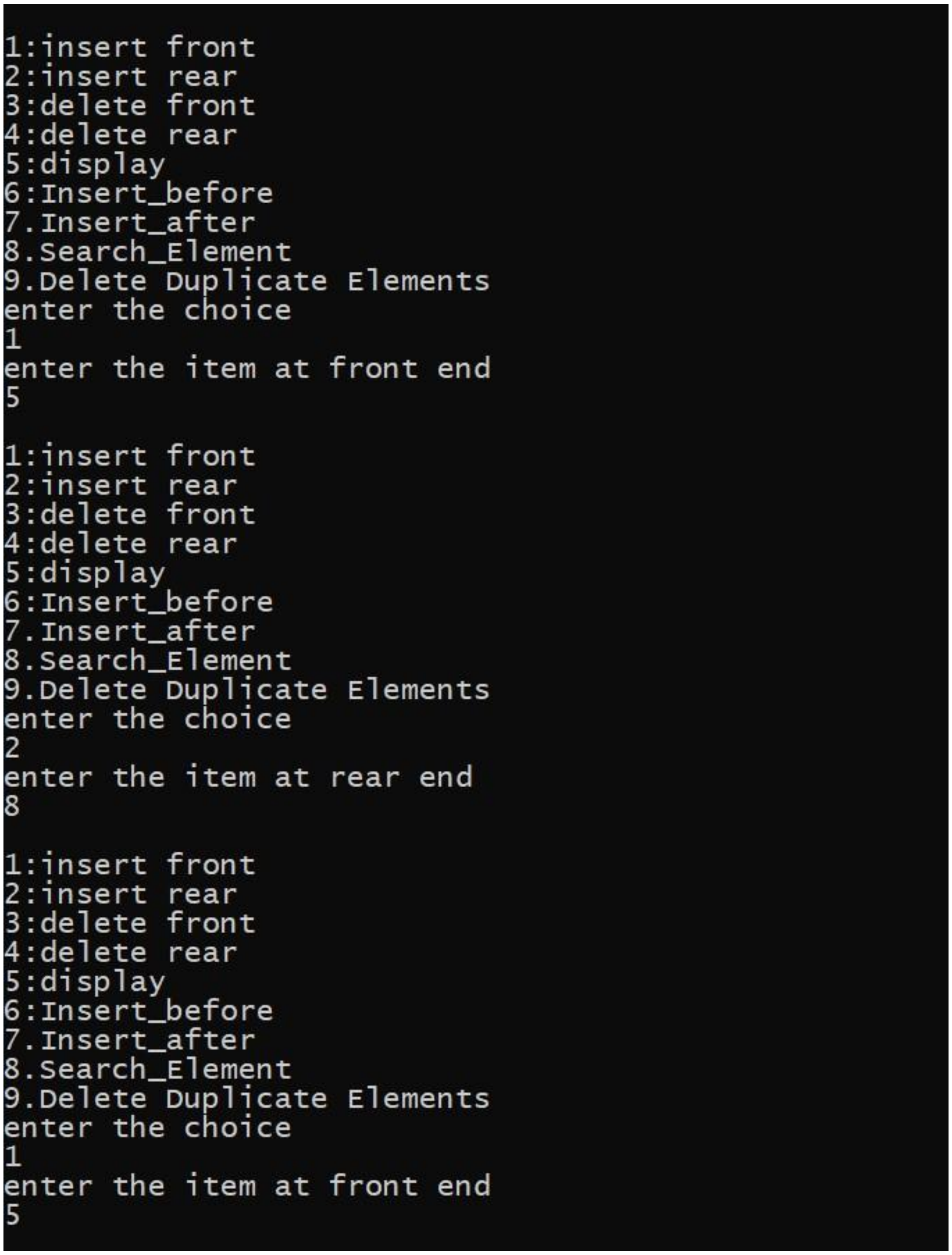


```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
6

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
7

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
5
contents of LIST
7       6       9       5       8       9       5       5

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
8
Enter the element for Search
9
```



```
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
4
the node deleted is 8
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
5

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
9
```


```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
9

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
8

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
5
contents of LIST
8       9       5       5       8
```



```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
5

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
2
enter the item at rear end
8

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
5
```