```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
   struct node*prev;
   int data;
   struct node*next;
}node;
void doublylinklisttraverse(node*ptr){
   while(ptr !=NULL){
   printf("%d\n",ptr->data);
   ptr=ptr->next;}
}
void reverseprint(node * head)
{
   node * p= head;
   while(p->next!=NULL)
      p=p->next;
   while(p!=head){
   printf("%d\n",p->data);
   p=p->prev;}
   printf("%d",p->data);

}
node * Initlist(int n)
{
   node * head,* p,* q;
   head=(node * )malloc(sizeof(node));
   head->prev=NULL;
   head->next=NULL;
   scanf("%d",&(head->data));
   p=head;
```

```
   for(int i=1;i<n;i++)
   {
      q=(node*)malloc(sizeof(node));
      scanf("%d",&(q->data));
      q->prev=p;
      q->next=NULL;
      p->next=q;
      p=q;
   }
   return head;

}
node * append(node *head)//adding element at last
{
   node * nn,*p;
   nn=(node * )malloc(sizeof(node));
   printf("Enter data for new node to append ");
   scanf("%d",&(nn->data));
   p=head;
   while(p->next!=NULL)
   {
      p=p->next;
   }
   p->next=nn;
   nn->prev=p;
   nn->next=NULL;
   return head;

}
node * insert(node * head) //Element insertion between first and last node
{
```

```
    int data,index;
    node * nn,*p,*q;
    p=head;
    q=p->next;
    nn=(node * )malloc(sizeof(node));
    printf("Enter the index of node ");
    scanf("%d",&index);
    printf("Enter the data of node ");
    scanf("%d",&data);
    int i=0;
    while(i!=index-1)
    {
       p=p->next;
       q=q->next;
       i=i+1;
    }
    nn->data=data;
    nn->prev=p;
    nn->next=q;
    q->prev=nn;
    p->next=nn;
    return head;

}
node * insertatfirst(node * head)
{
    node * ptr=(node *)malloc(sizeof(node));
    printf("Enter data to add at start of list ");
    scanf("%d",&(ptr->data));
    ptr->next=head;
    ptr->prev=NULL;
```

```
    head->prev=ptr;
    return ptr;
}
node * delete_at_first(node*head)
{
    node *p=head;
    head=head->next;
    head->prev=NULL;
    free(p);
    printf("Element deleted from first node\n");
    return head;
}
node * delete_at_index(node * head)
{   int index;
    node * p= head;
    node * q = head->next;
    node * r= q->next;
    int i=0;
    printf("Enter the index you want to delete ");
    scanf("%d",&index);
    for (i=0;i<index-1;i++)
    {
        p=p->next;
        q=q->next;
        r=r->next;
    }
    p->next=r;
    r->prev=p;
    free(q);
    return head;
}
```

```
node * delete_at_last(node * head)
{
   node * p= head;
   node * q = head->next;
   while(q->next!=NULL)
   {
      p=p->next;
      q=q->next;
   }
   free(q);
   printf("Element deleted from last node \n");
   p->next=NULL;
   return head;
}
node* search(node* head,int data)
{  int a=0;
   node *p = head;
   while(p!=NULL)
   {
      if(p->data == data)
         a=1;
      p = p->next;
   }
   if (a==1)
   {
      printf("Element present");
   }
   else
   {
      printf("Element not present");
   }
```

```
}
void main()
{
   int n;
   int i;
   int a;
   node * head;
   head=NULL;
   printf("Enter the number of nodes ");
   scanf("%d\n",&n);
   head=Initlist(n);
   printf("1. Press 1 to append \n2. Press 2 to insert element at particular index\n3. Press 3 to insert element at first\n");
   printf("4. Press 4 to remove element at first\n5. Press 5 to remove element at particular index\n6. Press 6 to remove element at last\n");
   printf("7. Press 7 to search element \n8. Press 8 to traverse backward \n");
   scanf("%d",&i);
   switch(i)
   {
   case 1:
      head=append(head);
      doublylinklisttraverse(head);
      break;
   case 2:
      head=insert(head);
      doublylinklisttraverse(head);
      break;
   case 3:
      head=insertatfirst(head);
      doublylinklisttraverse(head);
      break;
   case 4:
```

```
        head=delete_at_first(head);
        doublylinklisttraverse(head);
        break;
    case 5:
        head=delete_at_index(head);
        doublylinklisttraverse(head);
        break;
    case 6:
        head = delete_at_last(head);
        doublylinklisttraverse(head);
        break;
    case 7:
        printf("Enter Element ");
        scanf("%d",&a);
        head=search(head,a);
    case 8:
        reverseprint(head);
    }
   // doublylinklisttraverse(head);
   // head=append(head);
   // doublylinklisttraverse(head);
   // head=insert(head);
   // doublylinklisttraverse(head);
   // head=insertatfirst(head);
   // doublylinklisttraverse(head);
   // head=delete_at_first(head);
   // head=delete_at_index(head);
   // head = delete_at_last(head);
    //searchNode(4);

}
```