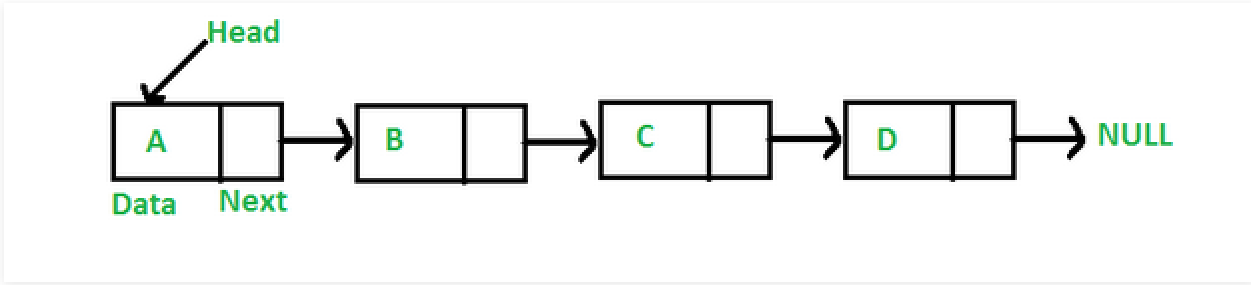
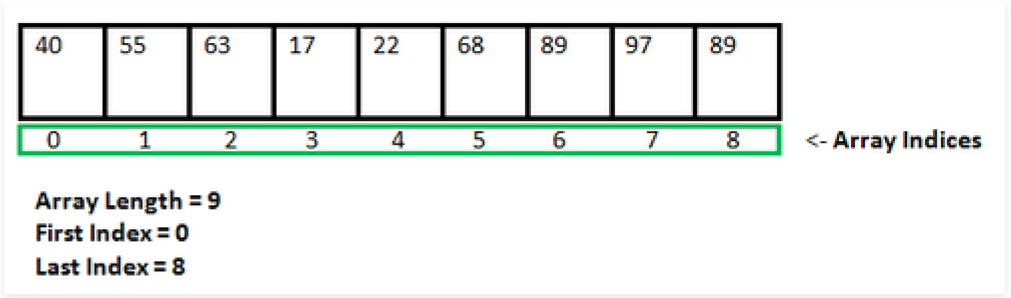
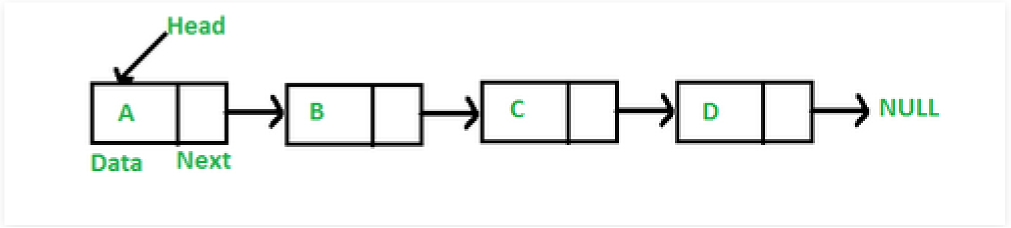
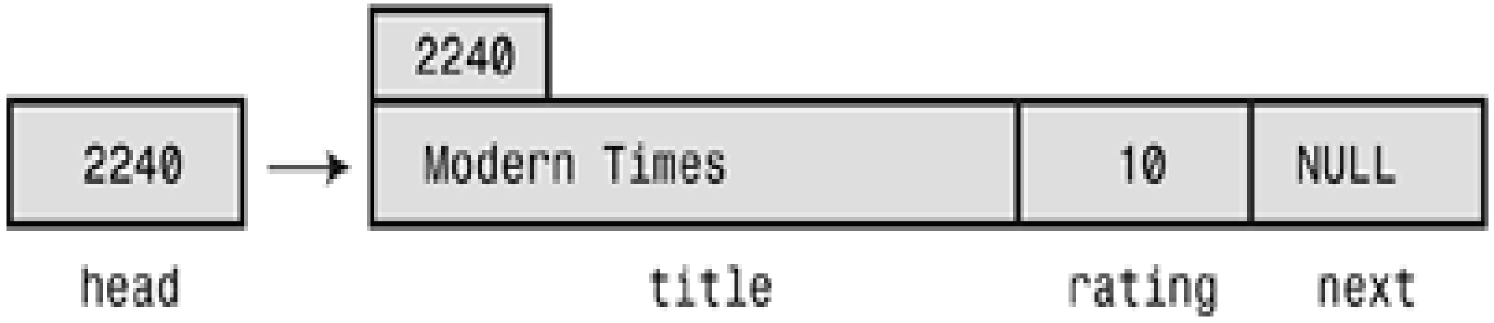
Linked lists

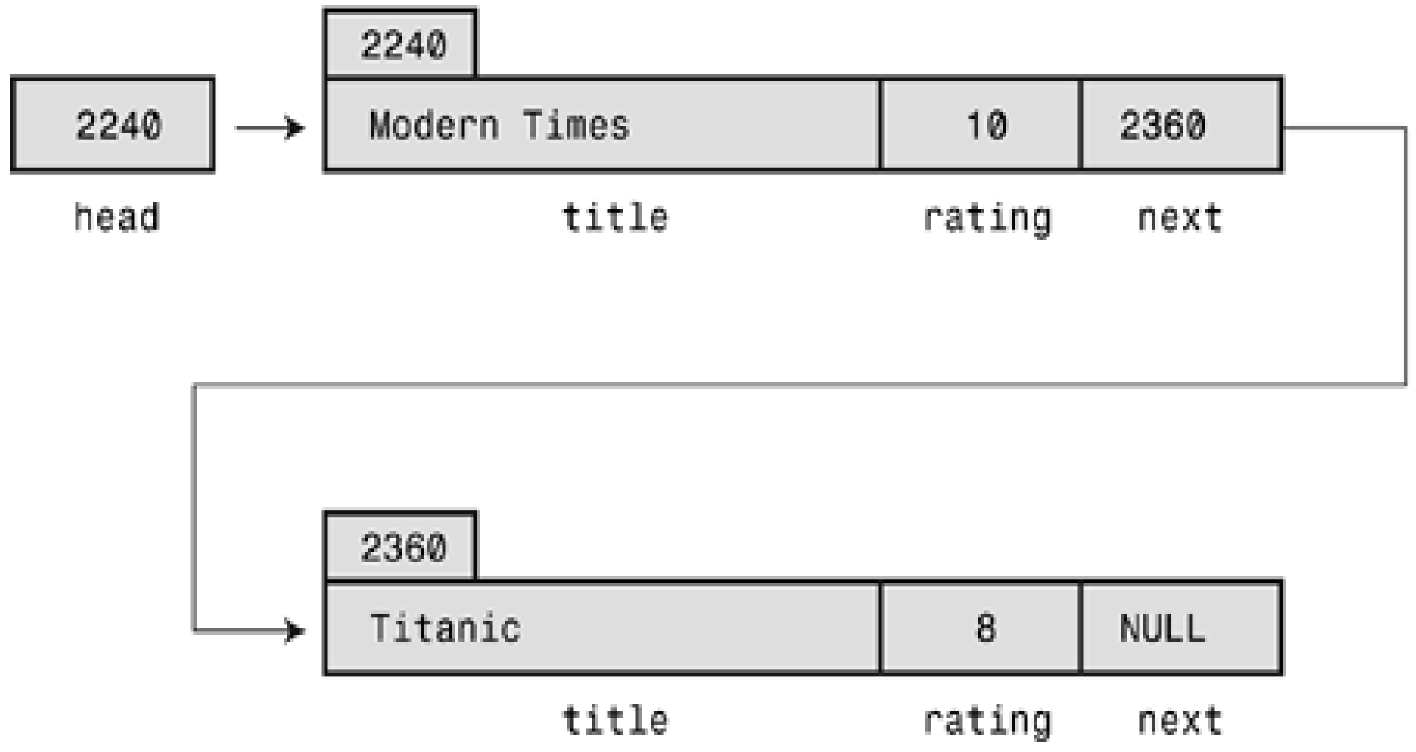


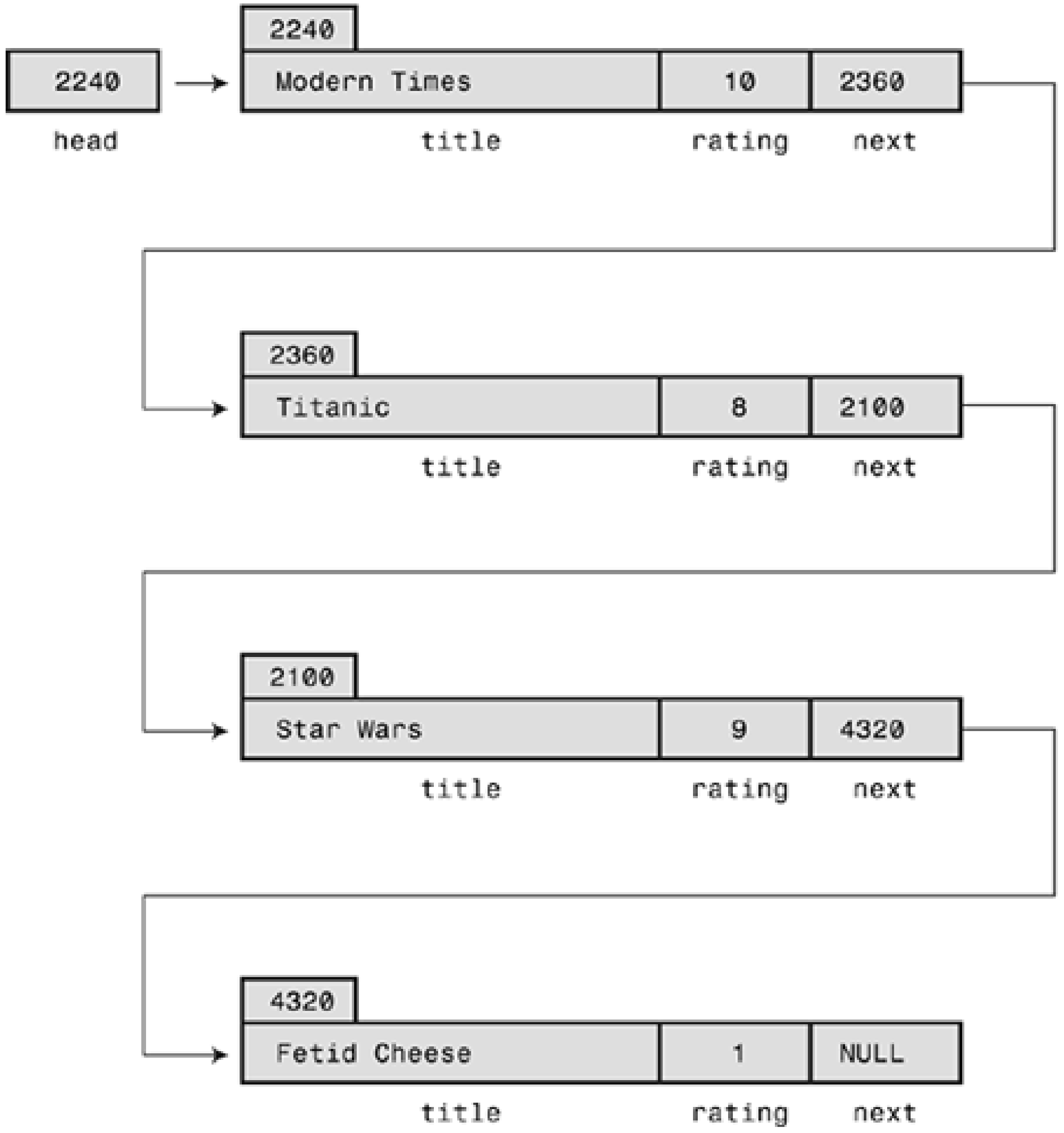




```
#define TSIZE 45
struct film {
   char title[TSIZE]
   int rating;
   struct film * next;
};
struct film * head;
```




2240
2240
Modern Times
10
2360
head
title
rating
next
2360
Titanic
8
NULL
title
rating
next


2240
2240
Modern Times
10
2360
head
title
rating
next
2360
Titanic
8
2100
title
rating
next
2100
Star Wars
9
4320
title
rating
next
4320
Fetid Cheese
1
NULL
title
rating
next

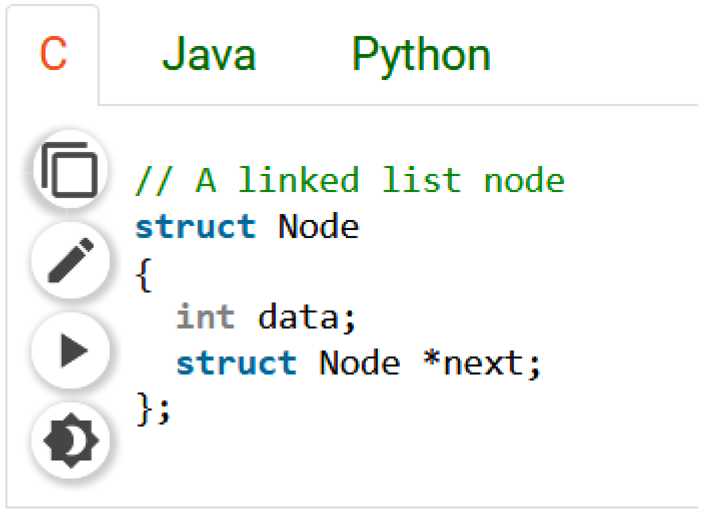
Avantaj & dezavantaj

Advantages over arrays

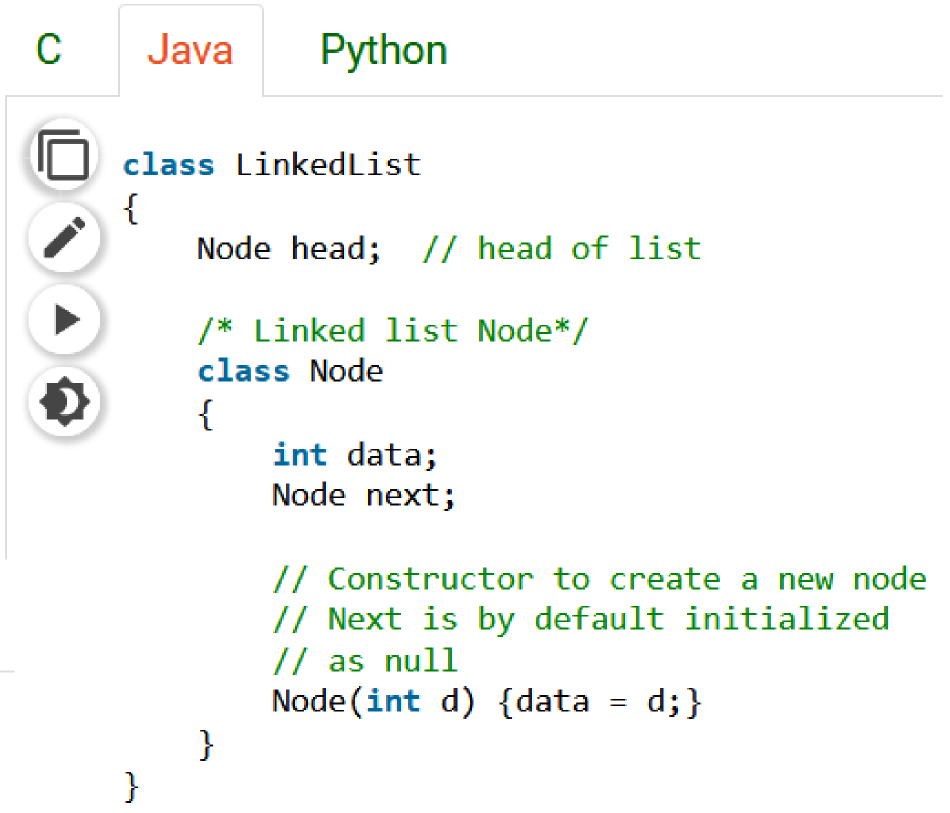
- 1) Dynamic size
- 2) Ease of insertion/deletion

Drawbacks:

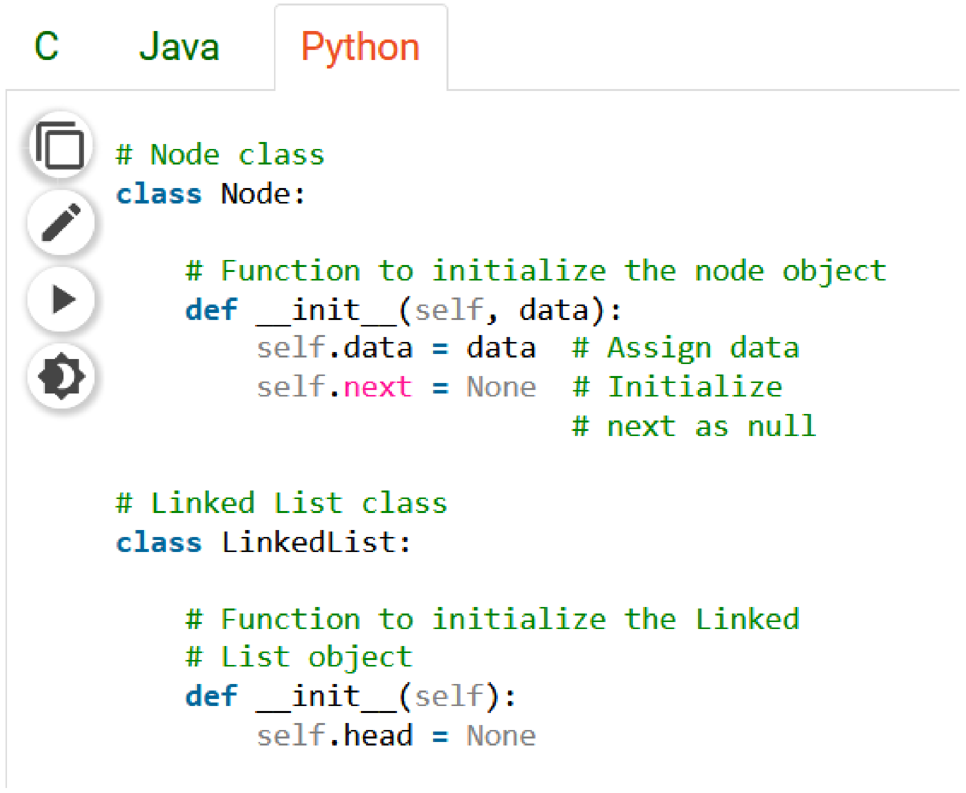
- 1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation.
- 2) Extra memory space for a pointer is required with each element of the list.
- 3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.



```c
// A linked list node
struct Node
{
  int data;
  struct Node *next;
};
```



```java
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;

        // Constructor to create a new node
        // Next is by default initialized
        // as null
        Node(int d) {data = d;}
    }
}
```



```python
# Node class
class Node:

    # Function to initialize the node object
    def __init__(self, data):
        self.data = data  # Assign data
        self.next = None  # Initialize
                          # next as null

# Linked List class
class LinkedList:

    # Function to initialize the Linked
    # List object
    def __init__(self):
        self.head = None
```

Lists oluşturma

- 3 Aşamalı bir işlemdir:

  malloc()
- kullanarak structure için yer ayrılır
- Structure adresi depolanır.
- Eldeki veriler structure kaydedilir

```
struct film {
    char title[TSIZE];
    int rating;
    struct film * next;  /* points to next struct
in list */
};
```

Veri girişi var mı?

```
while (gets(input) != NULL && input[0] != '\0')
```

Eğer var ise program hafızada yer açar ve adresi current pointerine yollar:

```
current = (struct film *) malloc(sizeof(struct film));
```

- İlk structure adresi **head** pointerinde saklanır. Sonraki sıradaki structure adresleri **next** pointerinde saklanır.
- Program içinde ilk pointeri ile mi yoksa sonrakiler ilemi işlem yaptığınızı bilmelisiniz. Bunun yolu head pointerini NULL yapmak ve program içinde head pointerının değerini değiştirmektir.

```
if (head == NULL)               /* first structure */
head = current;
else                            /* subsequent structures */
 prev->next = current;
```

Prev pointeri önceki ayrılmış structure yerini gösterir.

- Structure üyelerinin değerleri atanır: Next pointerının değeri NULL atanır ki bu structure listedeki son structure. Bundan sonra film ismi ve verilen puanlar girilir.

```
current->next = NULL;
strcpy(current->title, input);
puts("Enter your rating <0-10>:");
scanf("%d", &current->rating);
```

- Program ikinci dalga girişler için hazırlanır: Prev pointerı şimdi current pointerına eşitlenir ki yeni giriş yapıldığında bu current bir önceki giriş olacaktır.

```
    prev = current;
```

- Program malloc() tarafından kullanılan yeri serbest bırakır.

```
current = head;
while (current != NULL)
 {
free(current);
current = current->next;
 }
```

```
#include <stdio.h>
#include <stdlib.h>      /* has the malloc prototype      */
#include <string.h>      /* has the strcpy prototype      */
#define TSIZE    45      /* size of array to hold title   */


struct film {
    char title[TSIZE];
    int rating;
    struct film * next;  /* points to next struct in list */
};
```

```
int main(void)
{
    struct film * head = NULL;
    struct film * prev, * current;
    char input[TSIZE];
/* Gather  and store information          */
    puts("Enter first movie title:");
    while (gets(input) != NULL && input[0] != '\0')
    {
        current = (struct film *) malloc(sizeof(struct film));
        if (head == NULL)                          /* first structure       */
            head = current;
        else                                   /* subsequent structures */
            prev->next = current;
        current->next = NULL;
strcpy(current->title, input);
        puts("Enter your rating <0-10>:");
        scanf("%d", &current->rating);
        while(getchar() != '\n')
            continue;
        puts("Enter next movie title (empty line to stop):");
        prev = current;
    }
/* Show list of movies                    */
    if (head == NULL)
        printf("No data entered. ");
    else
        printf ("Here is the movie list:\n");
    current = head;
```

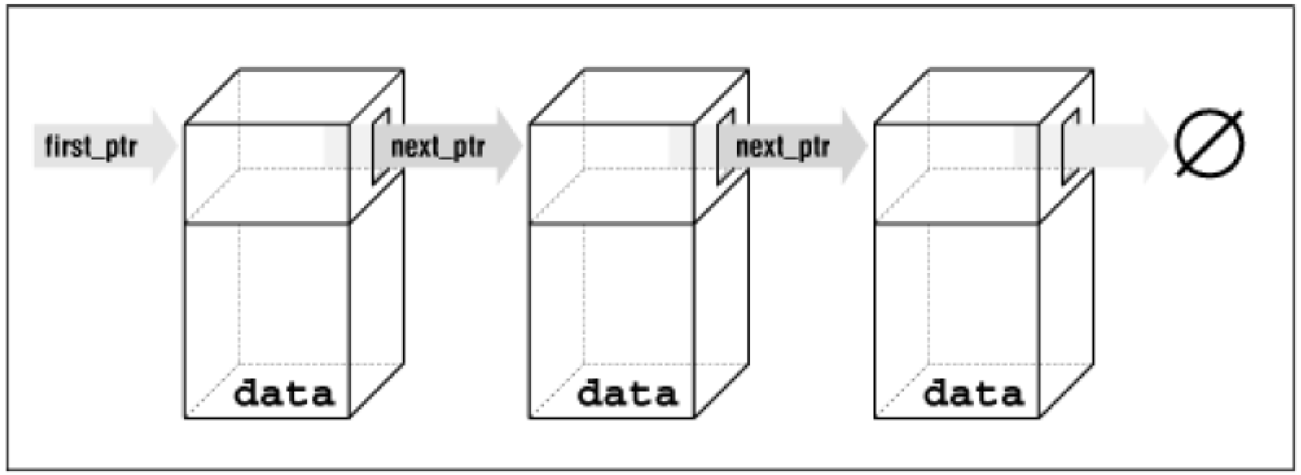
```
while (current != NULL)
  {
    printf("Movie: %s  Rating: %d\n",
           current->title, current->rating);
    current = current->next;
  }
/* Program done, so free allocated memory */
  current = head;
  while (current != NULL)
  {
    free(current);
    current = current->next;
  }
  printf("Bye!\n");
  return 0;}
```

```
// A simple C program for traversal of a linked list
#include<stdio.h>
#include<stdlib.h>
struct Node
{
        int data;
        struct Node *next;
};
// This function prints contents of linked list starting from
// the given node
void printList(struct Node *n){
        while (n != NULL)
        {
                printf(" %d ", n->data);
                n = n->next;
        }
}
```

```
int main()  {
        struct Node* head = NULL;        struct Node* second = NULL;
        struct Node* third = NULL;
        // allocate 3 nodes in the heap
        head = (struct Node*)malloc(sizeof(struct Node));
        second = (struct Node*)malloc(sizeof(struct Node));
        third = (struct Node*)malloc(sizeof(struct Node));

        head->data = 1; //assign data in first node
        head->next = second; // Link first node with second

        second->data = 2; //assign data to second node
        second->next = third;
        third->data = 3; //assign data to third node
        third->next = NULL;

        printList(head);
        return 0;}
```

Linked Lists (Sıralı listeler)

Linked list



```
struct linked_list {
        char data[30];                  /* data in this element */
        struct linked_list *next_ptr;   /* pointer to next element */
};
struct linked_list *first_ptr = NULL;
```

Liste başına eleman ekleme

1. İlk eleman (veri) için structure oluştur.

```
new_item_ptr = malloc(sizeof(struct linked_list));
```

2. Veriyi yeni oluşturulan elemana depola.

```
(*new_item_ptr).data = item;
```

3. Listenin yeni elamanı eski ilk elamanı işaret etsin

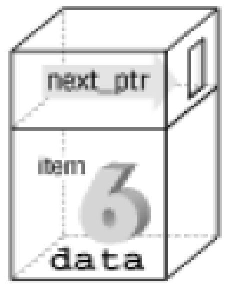
```
(*new_item_ptr).next_ptr = first_ptr;
```

4. Yeni eleman artık ilk eleman

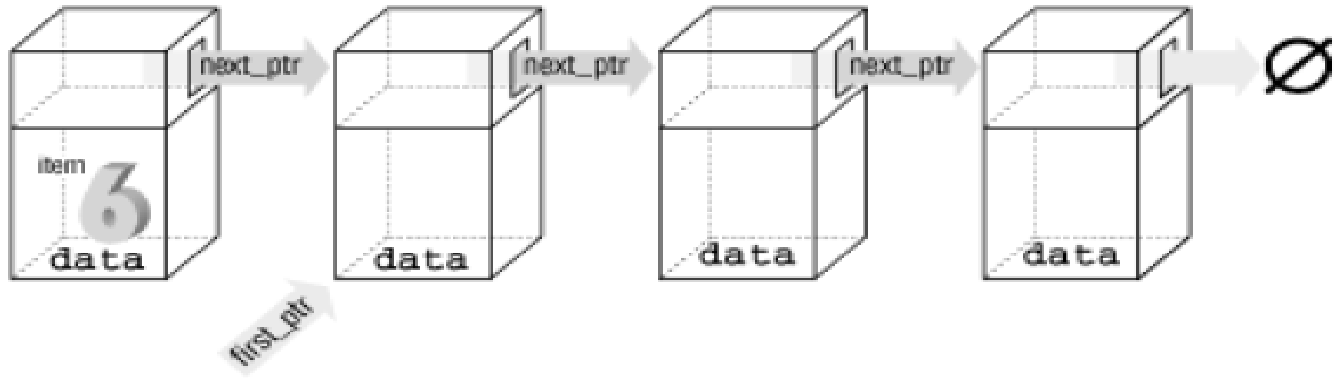
```
first_ptr = new_item_ptr;
```

```
void add_list(char *item)
{
struct linked_list *new_item_ptr;  /* pointer to the next item in the list */
new_item_ptr = malloc(sizeof(struct linked_list));
strcpy((*new_item_ptr).data, item);
(*new_item_ptr).next_ptr = first_ptr;
first_ptr = new_item_ptr;
}
```

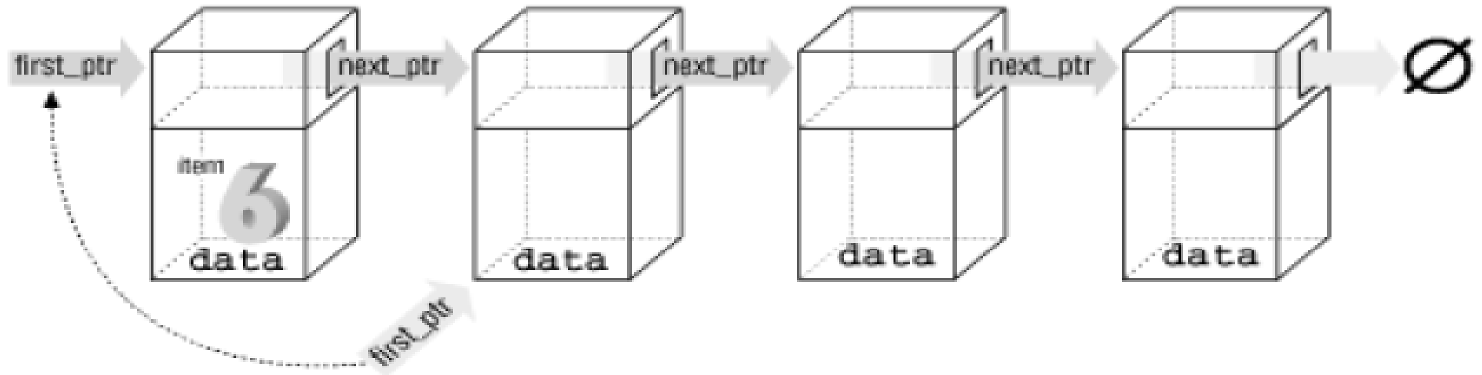
1. *Create new element.*
2. *Store item in new element.*

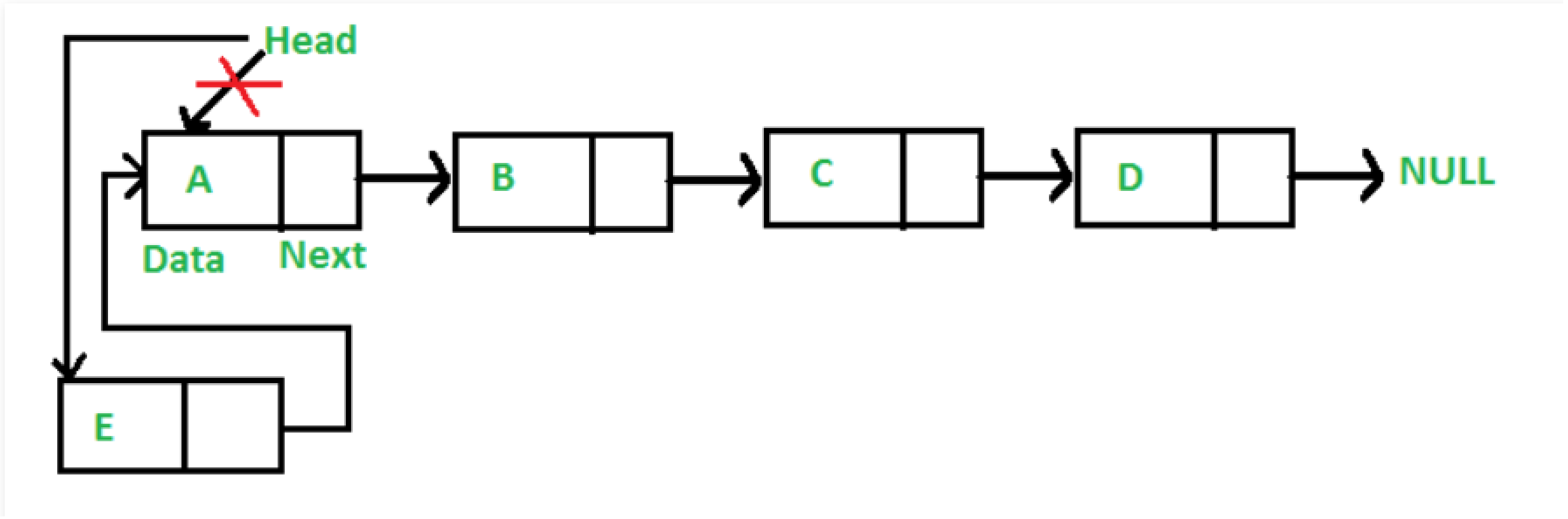


3. *Make* **`next_ptr`** *point to the first element.*



4. *Change* **`first_ptr`** *to point to the new element, thus breaking the link between* **`first_ptr`** *and the old first element.*



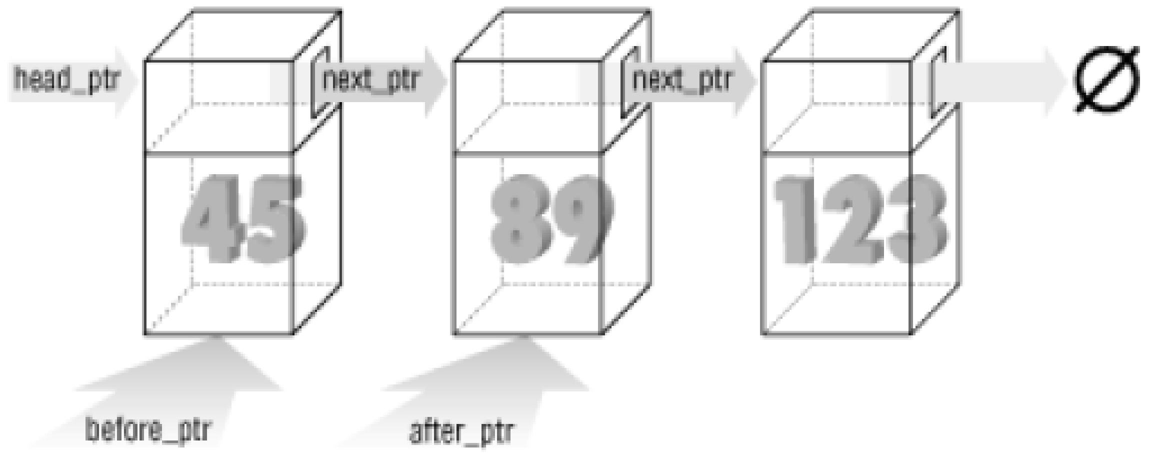

Head
A
B
C
D
NULL
Data
Next
E

```
/* Given a reference (pointer to pointer) to the head of a list
and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
          /* 1. allocate node */
struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

          /* 2. put in the data */
          new_node->data = new_data;

          /* 3. Make next of new node as head */
          new_node->next = (*head_ref);

          /* 4. move the head to point to the new node */
          (*head_ref) = new_node;
}
```

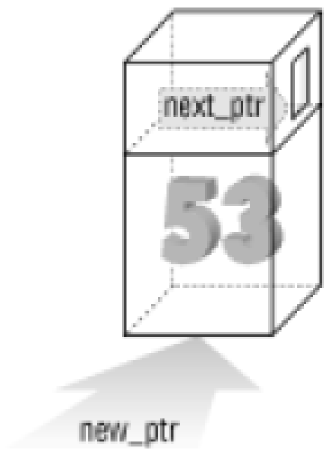
Listede sıralı olacak şekilde eleman ekleme

```
void enter(struct item *first_ptr, const int value)
{
struct item *before_ptr;                    /* Item before this one */
struct item *after_ptr;                     /* Item after this one */
struct item *new_item_ptr;                  /* Item to add */
/* Create new item to add to the list */
before_ptr = first_ptr;                     /* Start at the beginning */
after_ptr = before_ptr->next_ptr;
while (1) {
        if (after_ptr == NULL)
        break;
                if (after_ptr->value >= value)
                break;
                        /* Advance the pointers */
                        after_ptr = after_ptr->next_ptr;
                        before_ptr = before_ptr->next_ptr;
  }
/* Uygun yer bulundu*/
  new_item_ptr = malloc(sizeof(struct item));
  new_item_ptr->value = value;        /* Set value of item */
  before_ptr->next_ptr = new_item_ptr;
  new_item_ptr->next_ptr = after_ptr;
}
```

❶ **`before_ptr`** *points to the elements before the insertion point,* **`after_ptr`** *points to the element after the insertion point.*

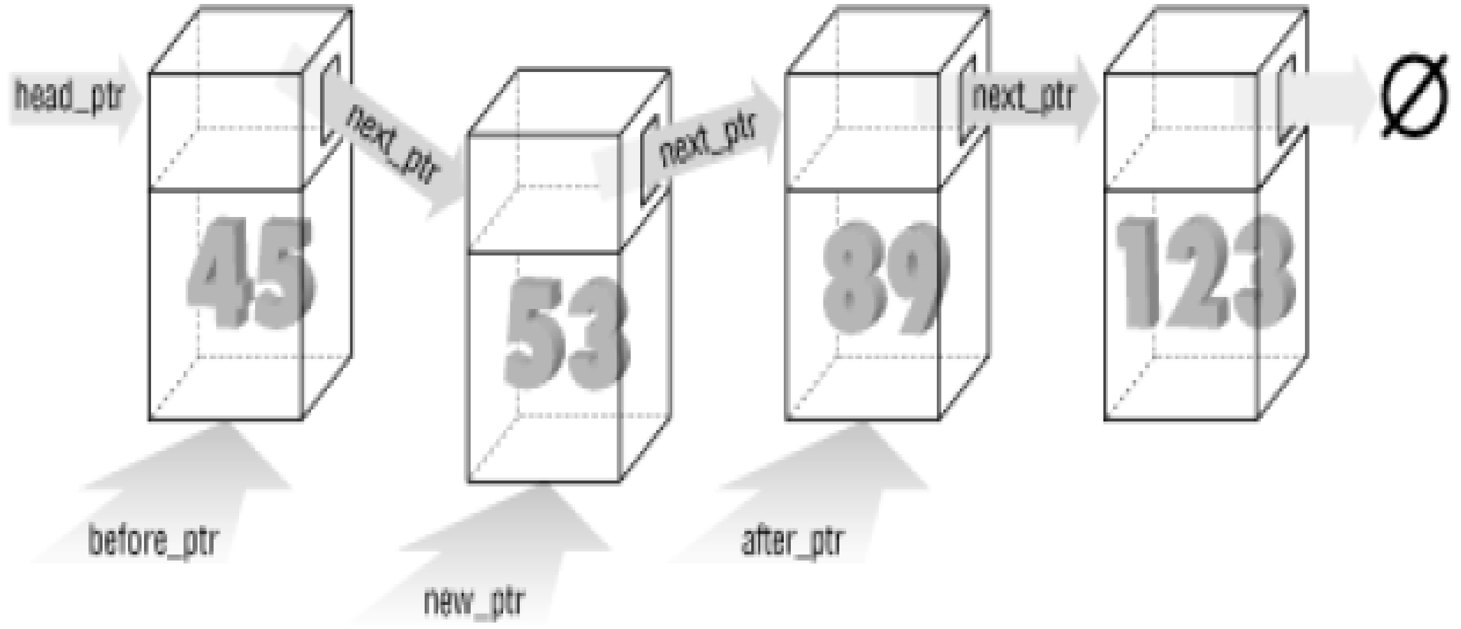


❷ *Create new element.*



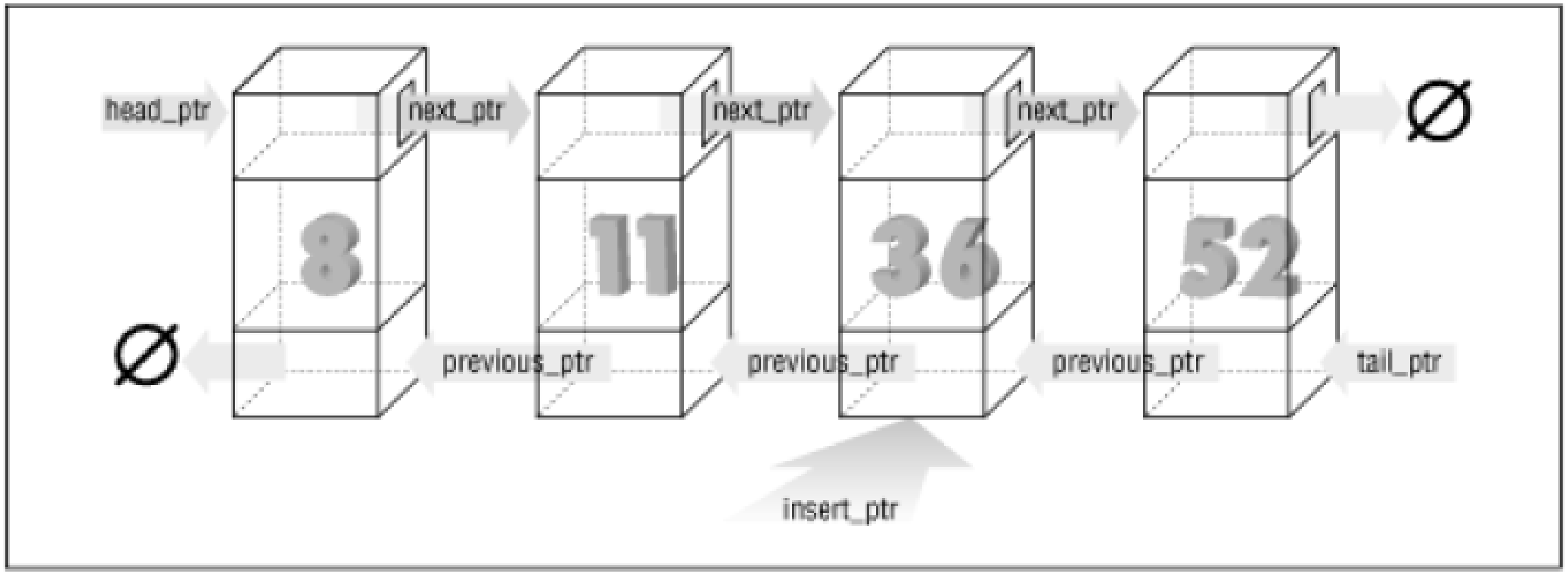
❸ *Make the* **`next_ptr`** *of the new element point to the same element as* **`after_ptr`**.

❹ *Link the element pointed to by* **`before_ptr`** *to our new element by changing* **`before_ptr->next_ptr`**.



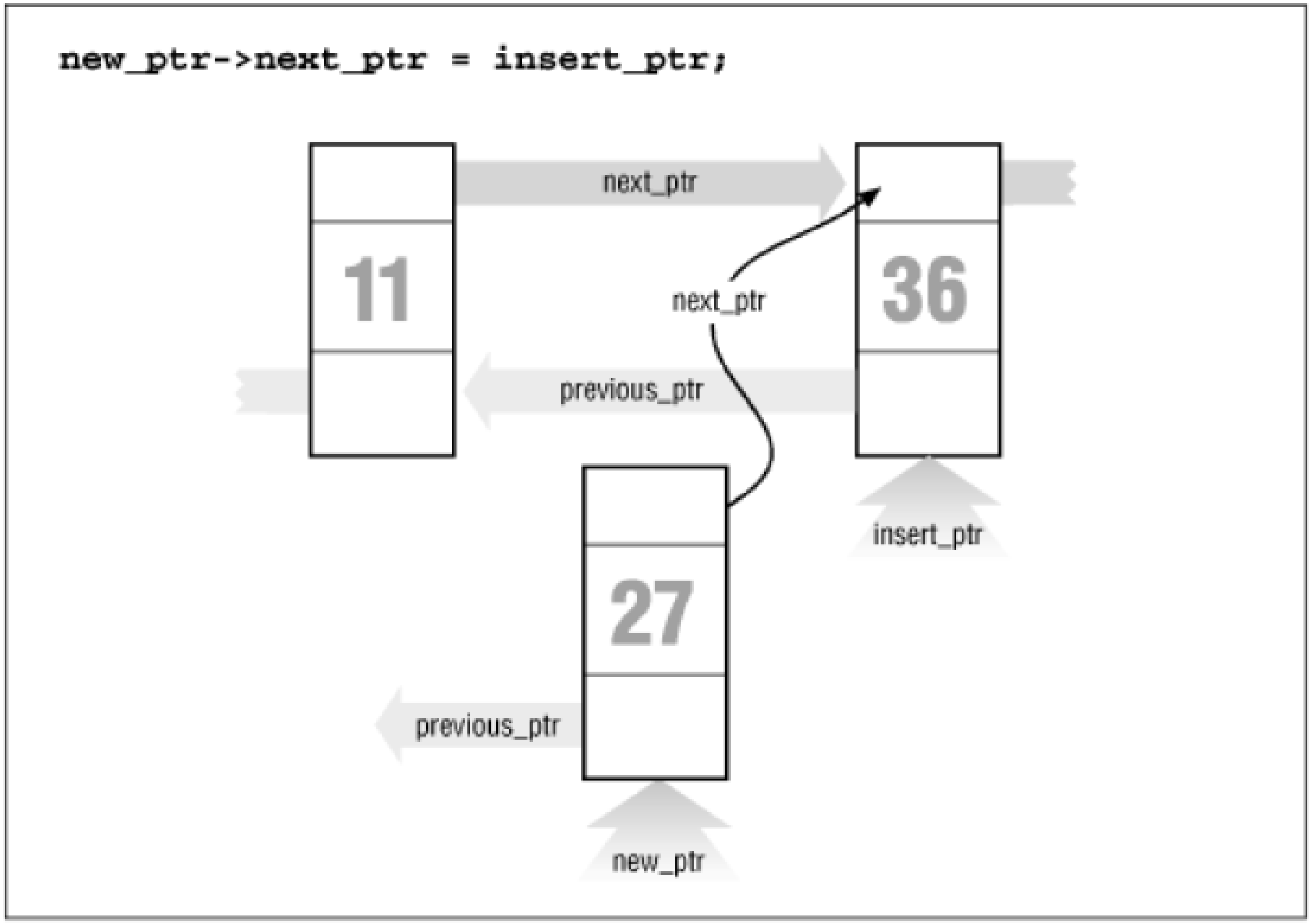
Double-Linked Lists

```
struct double_list {
    int data;                                /* data item */
     struct double_list *next_ptr;           /* forward link */
     struct double_list *previous_ptr;       /* backward link */
};
```

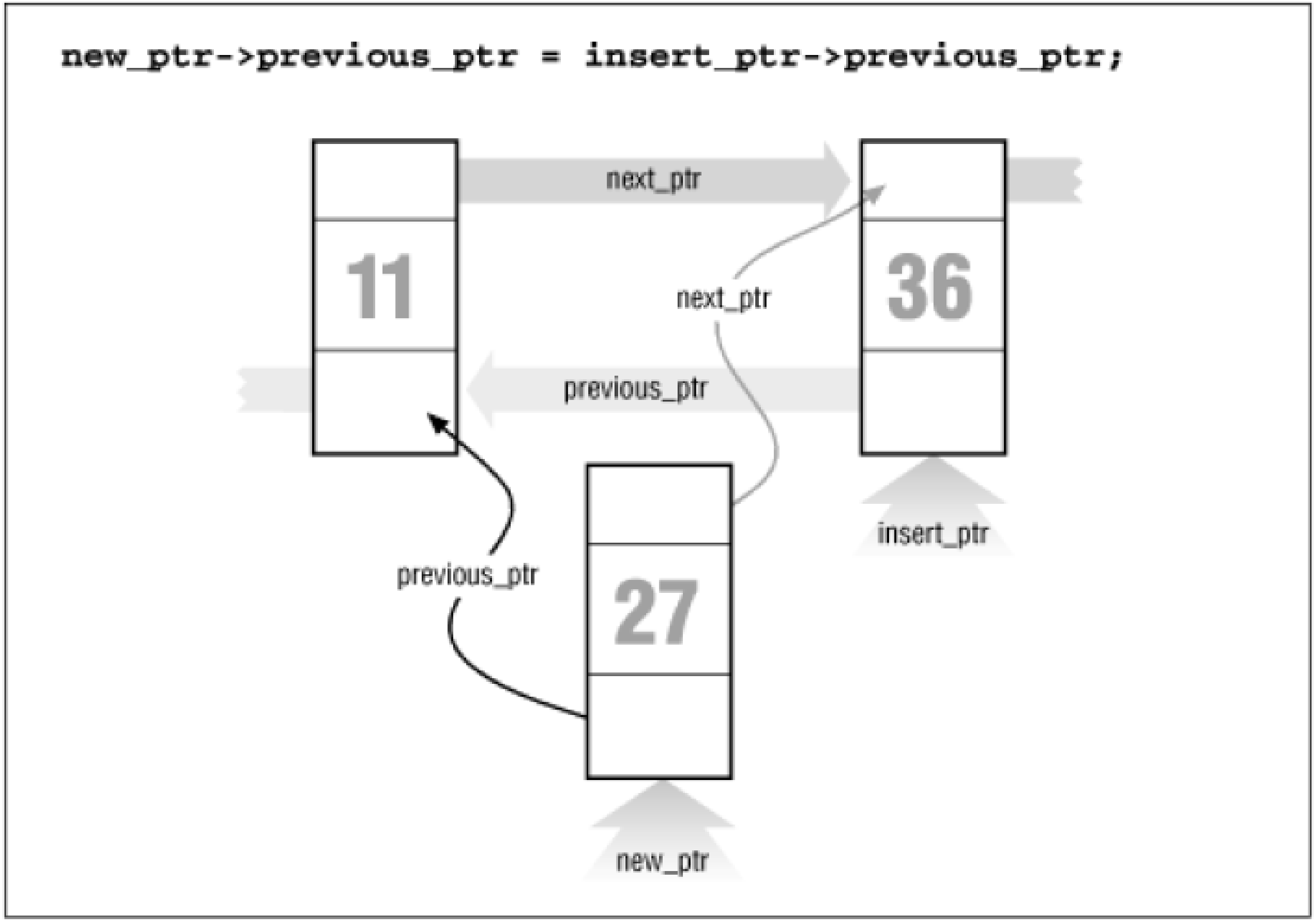


Eleman ekleme

```
void double_enter(struct double_list *head_ptr, int item)
{
struct list *insert_ptr;                    /* insert before this element */
/*****Liste başına ve sonuna eklemeler gözardı edilmiştir*******/
insert_ptr = head_ptr;
   while (1) {
            insert_ptr = insert_ptr->next;
            if (insert_ptr == NULL) /* have we reached the end */
            break;
                if (item >= insert_ptr->data) /* have we reached the right place */
                break;
   }
new_item_ptr->next_ptr = insert_ptr;
new_item_ptr->previous_ptr = insert_ptr->previous_ptr;
insert_ptr->previous_ptr->next_ptr = new_ptr;
insert_ptr->previous_ptr = new_item_ptr;
```

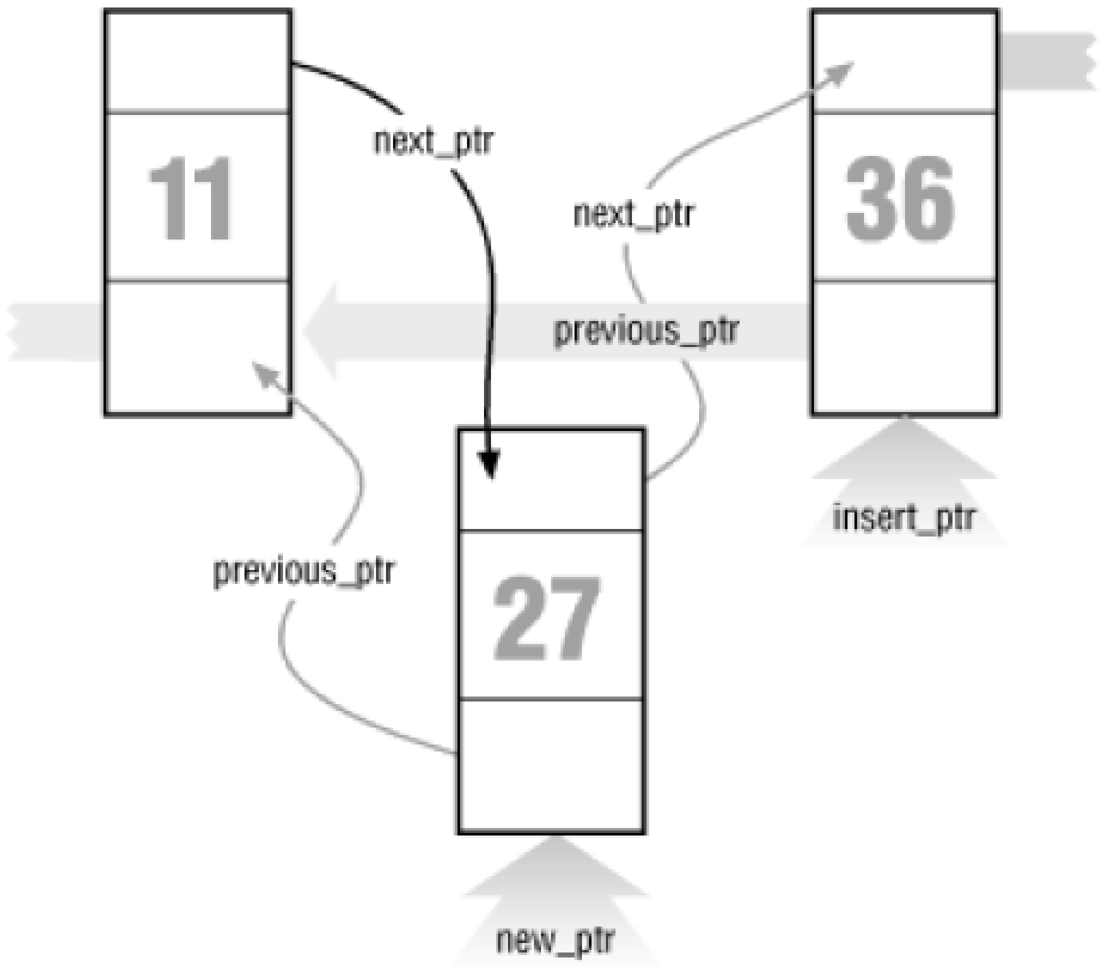


```
new_ptr->next_ptr = insert_ptr;
```

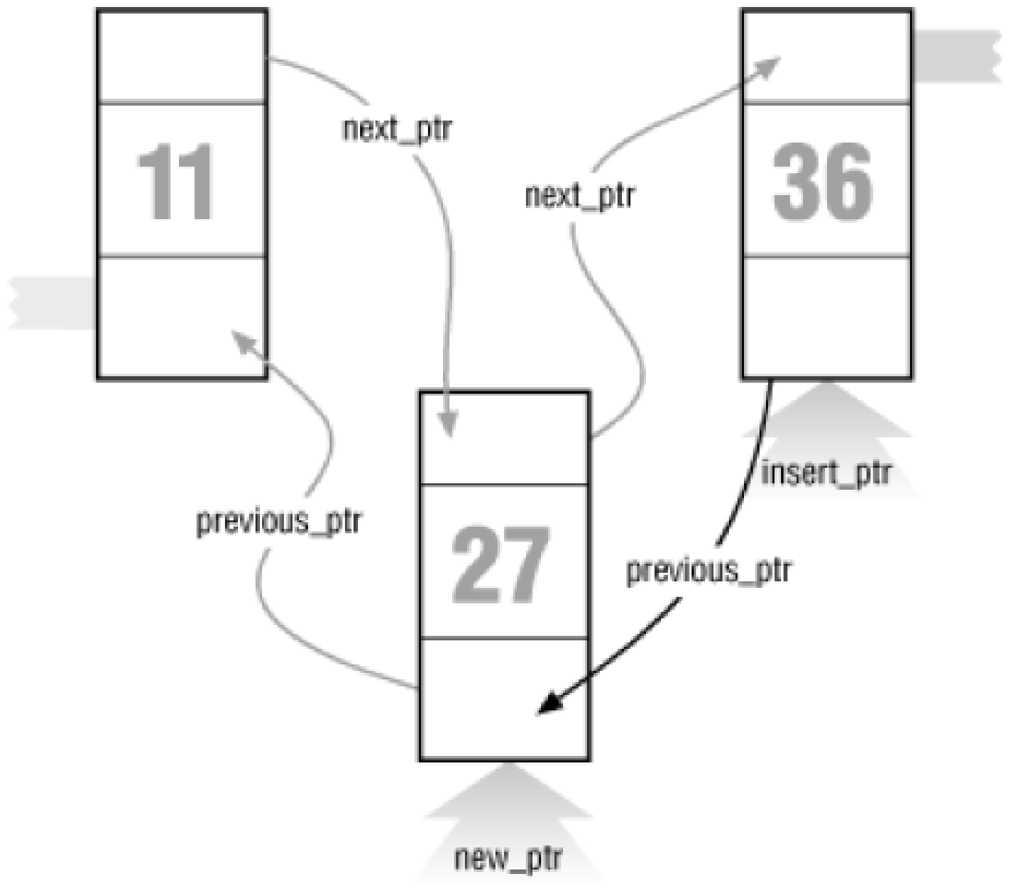


```
new_ptr->previous_ptr = insert_ptr->previous_ptr;
```

```
insert_ptr->previous_ptr->next_ptr = new_ptr;
```



```
insert_ptr->previous_ptr = new_ptr;
```



- **DELETING THE SPECIFIED NODE IN A SINGLY LINKED LIST**

To delete a node, first we determine the node number to be deleted (this is based on the assumption that the nodes of the list are numbered serially from 1 to n). The list is then traversed to get a pointer to the node whose number is given, as well as a pointer to a node that appears before the node to be deleted. Then the link field of the node that appears before the node to be deleted is made to point to the node that appears after the node to be deleted, and the node to be deleted is freed

```
# include <stdio.h>
# include <stdlib.h>
struct node *delet ( struct node *, int );
int length ( struct node * );
struct node {
int data;
struct node *link;
 };
```

```
struct node *insert(struct node *p, int n) {
struct node *temp;
       if(p==NULL) {
      p=(struct node *)malloc(sizeof(struct node));
                        if(p==NULL) {
                        printf("Error\n");
                        exit(0);
                        }
       p-> data = n;
       p-> link = NULL;
       }
                        else {
                        temp = p;
                                                 while (temp-> link != NULL)
                                                 temp = temp-> link;
                                                  temp-> link = (struct node *)malloc(sizeof(struct node));
                                                                          if(temp -> link == NULL) {
                                                                         printf("Error\n");
                                                                         exit(0);
                                                                         }
                         temp = temp-> link; temp-> data = n;
                         temp-> link = NULL;
                        }
 return (p);
 }
```

```
void printlist ( struct node *p )
{
printf("The data values in the list are\n");
    while (p!= NULL)
   {
            printf("%d\t",p-> data);
             p = p-> link;
   }
 }
void main()
 {
int n;
 int x;
 struct node *start = NULL;
 printf("Enter the nodes to be created \n");
 scanf("%d",&n);
```

```
    while ( n- > 0 )
    {
    printf( "Enter the data values to be placed in a node\n");
    scanf("%d",&x);
    start = insert ( start, x );
    }
            printf(" The list before deletion id\n");
            printlist ( start );
            printf("% \n Enter the node no \n");
            scanf ( " %d",&n);
            start = delet (start , n );
            printf(" The list after deletion is\n");
            printlist ( start );
}
```

```
/* a function to delete the specified node*/
    struct node *delet ( struct node *p, int node_no )
    {
     struct node *prev, *curr ;
    int i;
                if (p == NULL )
                {
                printf("There is no node to be deleted \n");
                }
                        else
                        {
                                if ( node_no > length (p))
                                {
                                printf("Error\n");
                                }
                                        else
                                        {
                                        prev = NULL;
                                        curr = p;
                                        i = 1 ;
```

```
                    while ( i < node_no )
                    {
                    prev = curr;
                    curr = curr-> link;
                    i = i+1;
                    }
                            if ( prev == NULL )
                            {
                            p = curr -> link;
                            free ( curr );
                             }
                                    else
                                    {
                                    prev -> link = curr -> link ;
                                    free ( curr );
                                    }
            }
    }
 return(p);
}
```

```
/* a function to compute the length of a linked list */
int length ( struct node *p )
{
int count = 0 ;
   while ( p != NULL )
   {
   count++;
   p = p->link;
    }
return ( count ) ;
}
```

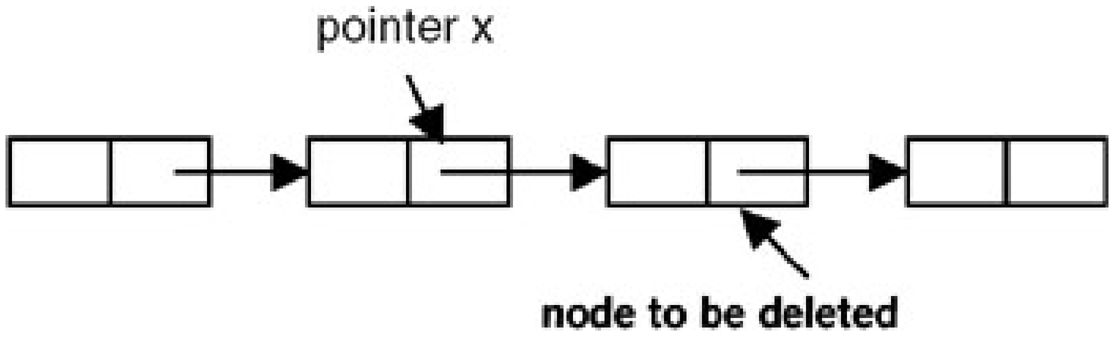
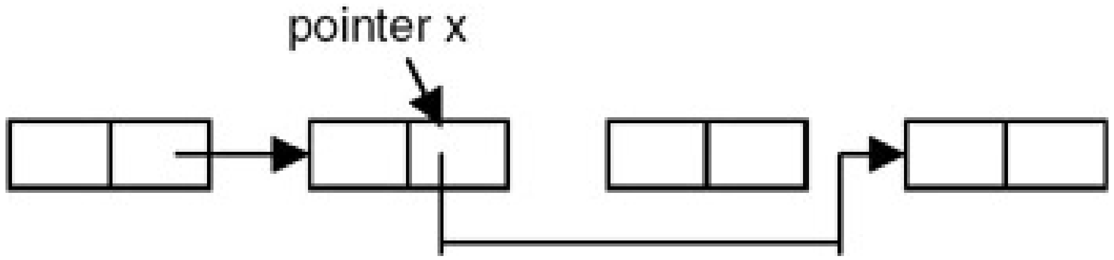


Figure 20.5: Before deletion.



```c
#include<stdio.h>        #include<stdlib.h>        #include<stdbool.h>
struct test_struct{
int val;
struct test_struct *next;
};
struct test_struct *head = NULL;
struct test_struct *curr = NULL;

struct test_struct* create_list(int val){
printf("\n creating list with headnode as [%d]\n", val);
struct test_struct *ptr = (struct test_struct*)malloc(sizeof(struct test_struct));
if (NULL == ptr){
printf("\n Node creation failed \n");
return NULL;
}
ptr->val = val;
ptr->next = NULL;

head = curr = ptr;
return ptr;
}
```

```c
struct test_struct* add_to_list(int val, bool add_to_end){
if (NULL == head){
return (create_list(val));
}
        if (add_to_end)
printf("\n Adding node to end of list with value [%d]\n", val);
        else
printf("\n Adding node to beginning of list with value [%d]\n", val);

struct test_struct *ptr = (struct test_struct*)malloc(sizeof(struct test_struct));
                if (NULL == ptr){
                printf("\n Node creation failed \n");
                return NULL;      }
        ptr->val = val;
        ptr->next = NULL;
                if (add_to_end){
                curr->next = ptr;
                curr = ptr;    }
                        else{
                        ptr->next = head;
                        head = ptr;}
return ptr;}
```

```c
struct test_struct* search_in_list(int val, struct test_struct **prev){
struct test_struct *ptr = head;
struct test_struct *tmp = NULL;
bool found = false;
printf("\n Searching the list for value [%d] \n", val);

while (ptr != NULL){
if (ptr->val == val){
found = true;
break;
}
else{
tmp = ptr;
ptr = ptr->next;
}
}
if (true == found){
if (prev)
*prev = tmp;
return ptr;
}
else{
return NULL;
}
}
```

```
int delete_from_list(int val){
struct test_struct *prev = NULL;
struct test_struct *del = NULL;
printf("\n Deleting value [%d] from list\n", val);

del = search_in_list(val, &prev);
if (del == NULL){
return -1;
}
else{
if (prev != NULL)
prev->next = del->next;
if (del == curr){
curr = prev;
}
else if (del == head){
head = del->next;
}
}
free(del);
del = NULL;

return 0;
}
```

```c
void print_list(void)
{
struct test_struct *ptr = head;

printf("\n -------Printing list Start------- \n");
while (ptr != NULL)
{
printf("\n [%d] \n", ptr->val);
ptr = ptr->next;
}
printf("\n -------Printing list End------- \n");

return;
}
```

```
int main(void){
int i = 0, ret = 0;
struct test_struct *ptr = NULL;
print_list();
for (i = 5; i<10; i++)
add_to_list(i, true);
print_list();
for (i = 4; i>0; i--)
add_to_list(i, false);
print_list();
for (i = 1; i<10; i += 4){
ptr = search_in_list(i, NULL);
if (NULL == ptr){
printf("\n Search [val = %d] failed, no such element found\n", i);
}
else{
printf("\n Search passed [val = %d]\n", ptr->val);
}
print_list();
ret = delete_from_list(i);
if (ret != 0){
printf("\n delete [val = %d] failed, no such element found\n", i);
}
else{
printf("\n delete [val = %d]  passed \n", i);
}
print_list();
}
return 0;}
```