CME 112- Programming Languages II
Lecture 4: Linked Lists

Assist.Prof.Dr. Ümit ATİLA

LINKED LISTS

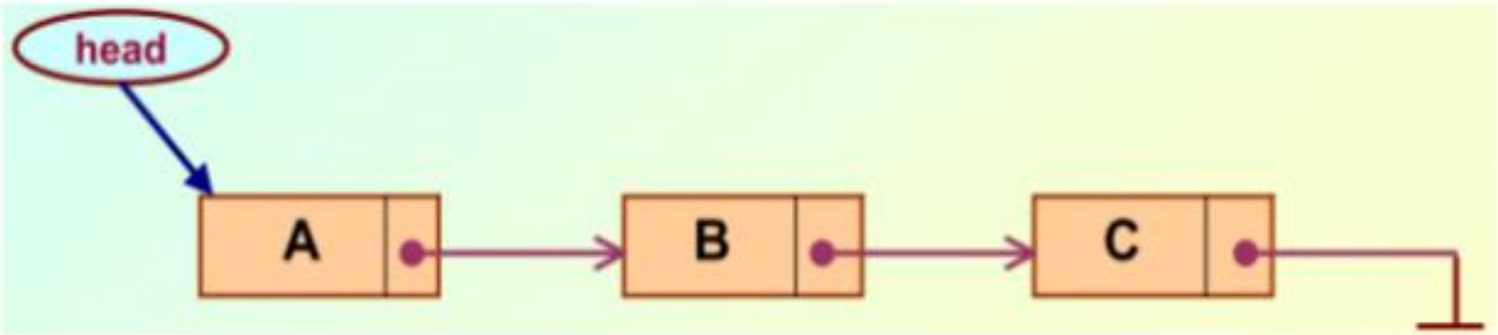
- Linked lists are useful to study for two reasons.
  - Most obviously, linked lists are a data structure. Seeing the strengths and weaknesses of linked lists will give you an appreciation of the some of the time, space, and code issues which are useful to thinking about any data structures in general.
  - Somewhat less obviously, linked lists are great way to learn about pointers. In fact, you may never use a linked list in a real program, but you are certain to use lots of pointers

LINKED LISTS

- Linked list problems are a nice combination of algorithms and pointer manipulation.
- Traditionally, linked lists have been the domain where beginning programmers get the practice to really understand pointers

LINKED LISTS

- A linked list is a data structure which can change during execution.
  - Successive elements are connected by pointers.
  - Last element points to NULL
  - It can grow or shrink in size during execution of a program.
  - It can be made just as long as required.
  - It doesn't made waste memory

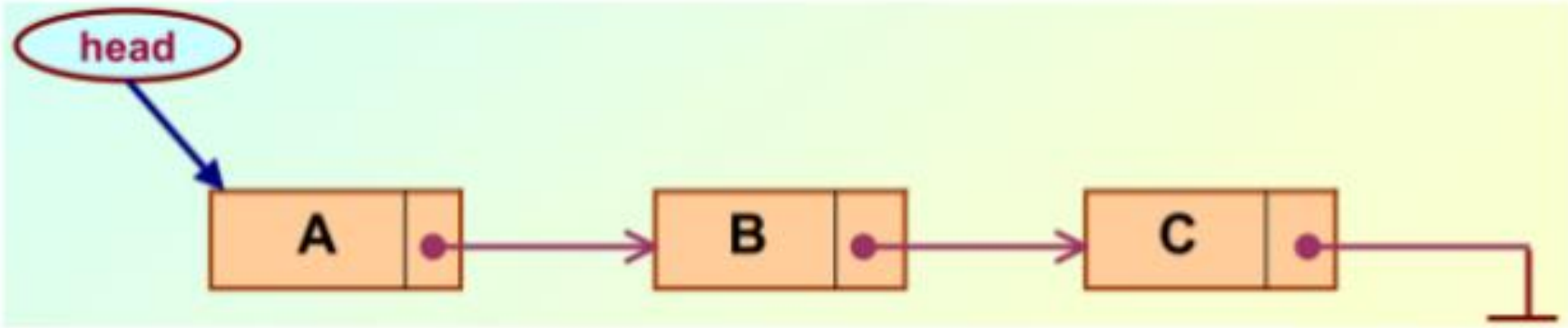


ARRAYS vs. LINKED LISTS

- Arrays are suitable for:
  - Inserting/deleting an element at the end.
  - Randomly accessing any element.
  - Searching the list for a particular value.
- Linked lists are suitable for:
  - Inserting an element.
  - Deleting an element.
  - Applications where sequential access is required.
  - In situations where the number of elements can not be predicted beforehand.

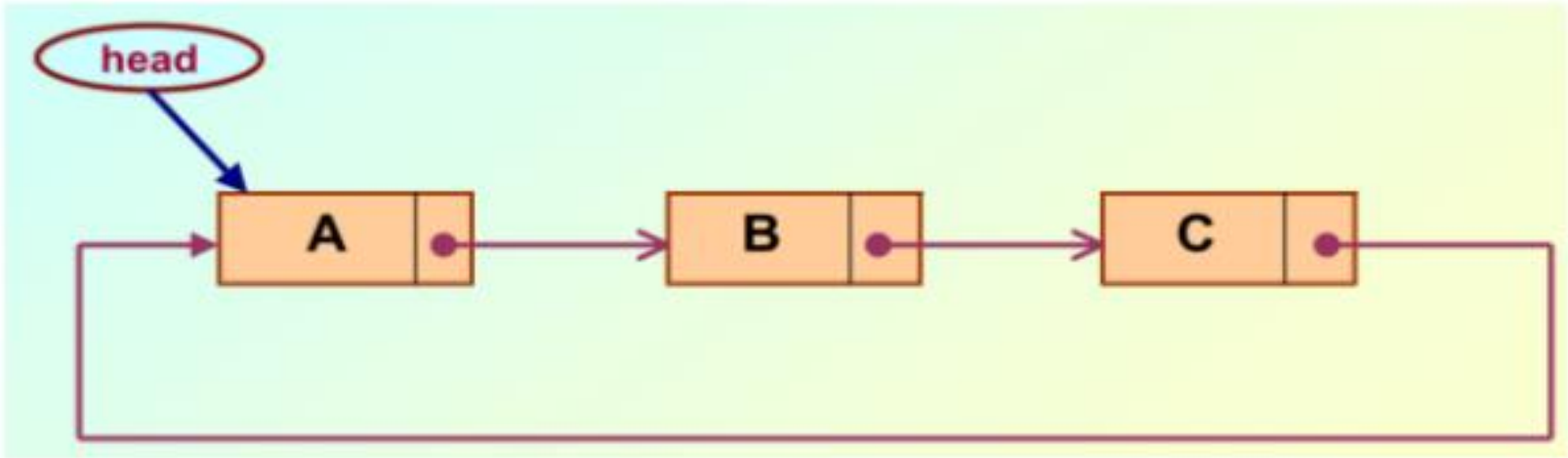
TYPES of LISTS

- Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.
  - **1. Linear singly linked list (Linear List)**



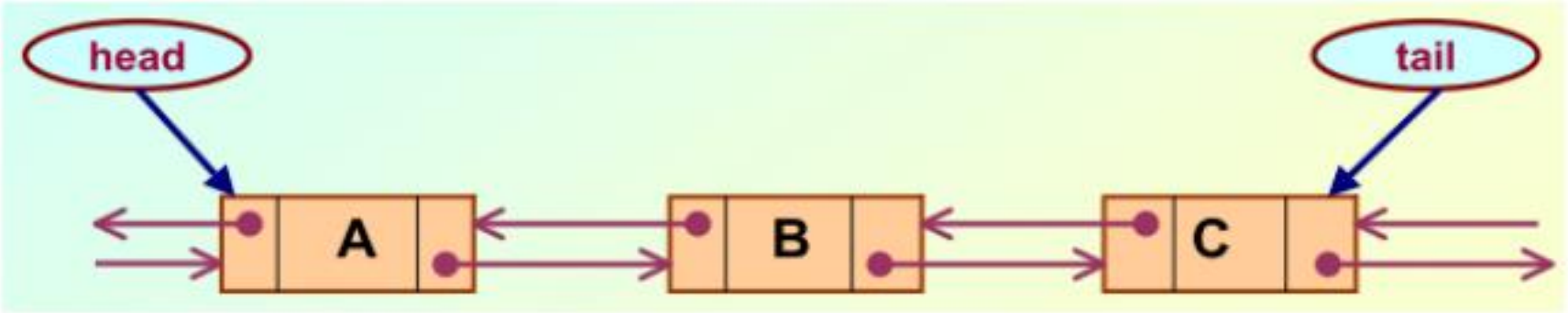
Liste Tipleri

- **2.Circular linked list**
  - The pointer from the last element in the list points back to the first element.



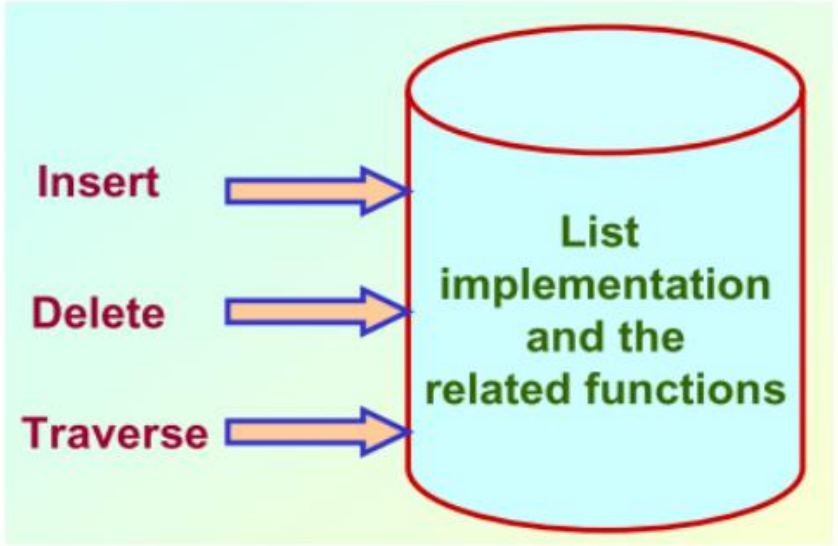
Liste Tipleri

- **3.Doubly linked list**
  - Pointers exist between adjacent nodes in both directions.
  - The list can be traversed either forward or backward.



LINKED LISTS

- List is an abstract data type
  - What is an abstract data type?
  - It is a data type defined by the user.
  - Typically more complex than simple data types like int, float, etc.



Basic Operations on a List

- Creating a list
- Traversing the list
- Inserting an item in the list
- Deleting an item in the list
- Concatenating two lists into one

Working with Linked Lists

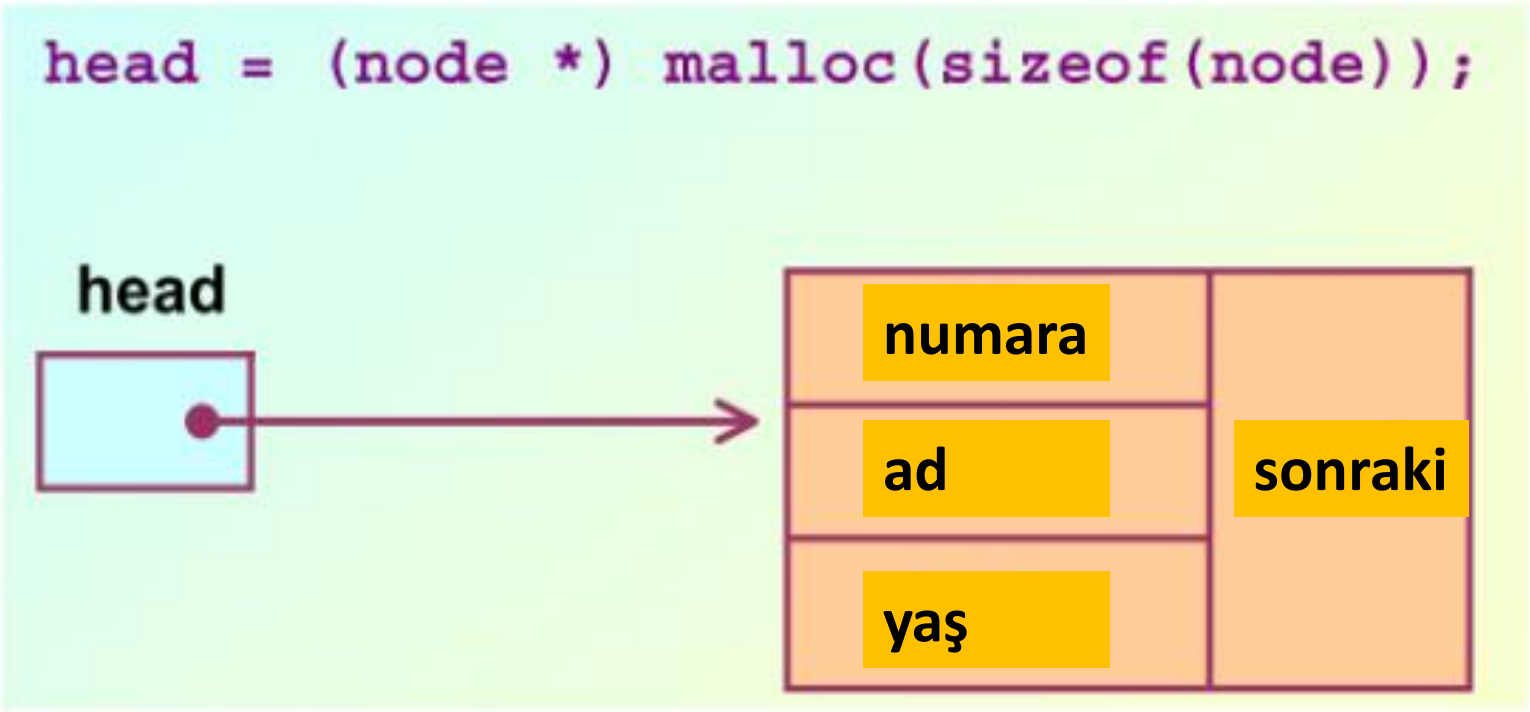
- Consider the structure of a node as follows:

```
 3  struct personel {
 4          int numara;
 5          char ad[25];
 6          int yas;
 7          struct personel *sonraki;
 8   };
 9  //Kullanıcı tanımlı veri tipi "dugum"
10   typedef struct personel dugum;
```

Creating a Linear List

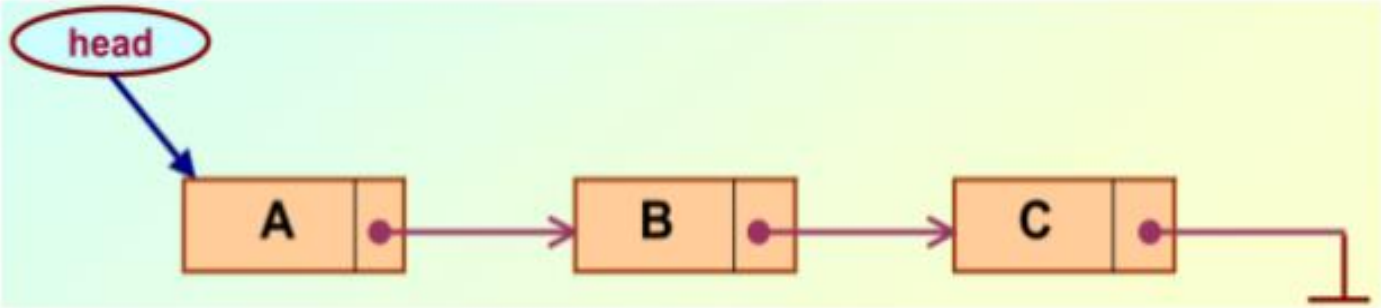
- To start with, we have to create a node (the first node), and make head point to it

```
head = (node *) malloc(sizeof(node));
```



Creating a Linear List

- If there are n number of nodes in the initial linked list:
  - Allocate n records, one by one.
  - Read in the fields of the records.
  - Modify the links of the records so that the chain is formed.



Creating a Linear List

```
16  dugum * listeOlustur()
17  {
18      int k,n;
19      dugum *p, *head;
20      printf("Kaç eleman gireceksiniz");
21      scanf("%d",&n);
22      for(k=0;k<n;k++)
23      {
24          if(k==0)
25          {
26              head = (dugum *)malloc(sizeof(dugum));
27              p = head;
28          }
29          else
30          {
31              p->sonraki = (dugum *)malloc(sizeof(dugum));
32              p=p->sonraki;
33          }
34          scanf("%d %s %d",&p->numara,p->ad,&p->yas);
35      }
36      p->sonraki = NULL;
37      return head;
38  }
```

Traversing the List

- Once the linked list has been constructed and head points to the first node of the list,
  - Follow the pointers.
  - Display the contents of the nodes as they are traversed.
  - Stop when the next pointer points to NULL

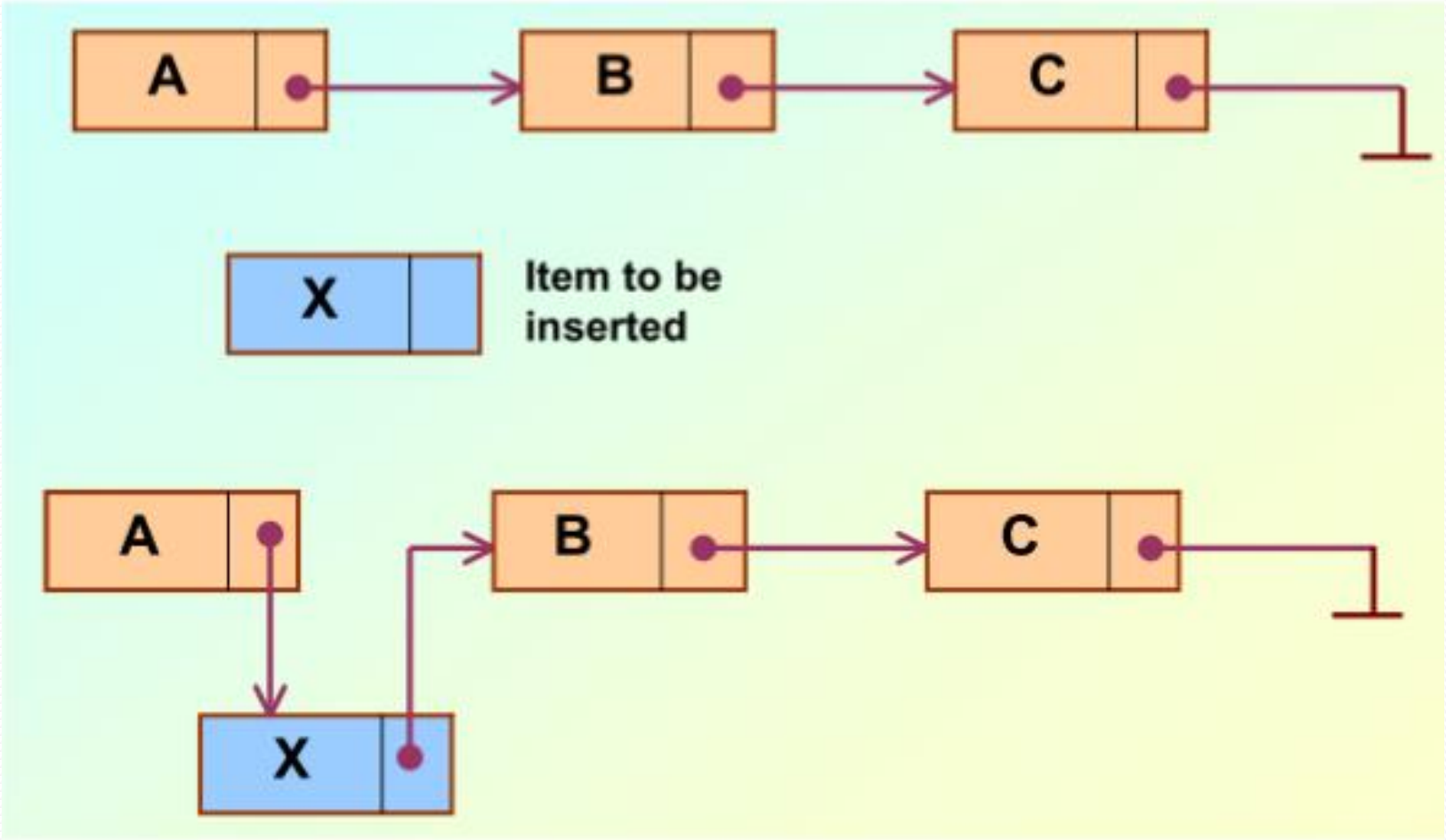
Traversing the List

```
39  void listeDolas(dugum *head)
40  {
41      int sayac =1;
42      dugum *p;
43      p = head;
44      while (p!= NULL)
45      {
46          printf("Dugum %d:%d %s %d",sayac,p->numara,p->ad,p->yas);
47          sayac++;
48          p = p->sonraki;
49          printf("\n");
50      }
51  }
```

Inserting a Node in a List

- For insertion:
  - A record is created holding the new item
  - The next pointer of the new record is set to link it to the item which is to follow it in the list.
  - The next pointer of the item which is to precede it must be modified to point to the new item.
- The problem is to insert a node before a specified node.
  - Specified means some value is given for the node (called key).
  - In this example, we consider it to be roll

Inserting a Node in a List



Inserting a Node in a List

- When a node is added at the beginning,
  - Only one next pointer needs to be modified.
    - Head is made to point to the new node.
    - New node points to the previously first element.
- When a node is added at the end,
  - Two next pointers need to be modified.
    - Last node now points to the new node.
    - New node points to NULL
- When a node is added in the middle,
  - Two next pointers need to be modified.
    - Previous node now points to the new node.
    - New node points to the next node.

Inserting a Node in a List

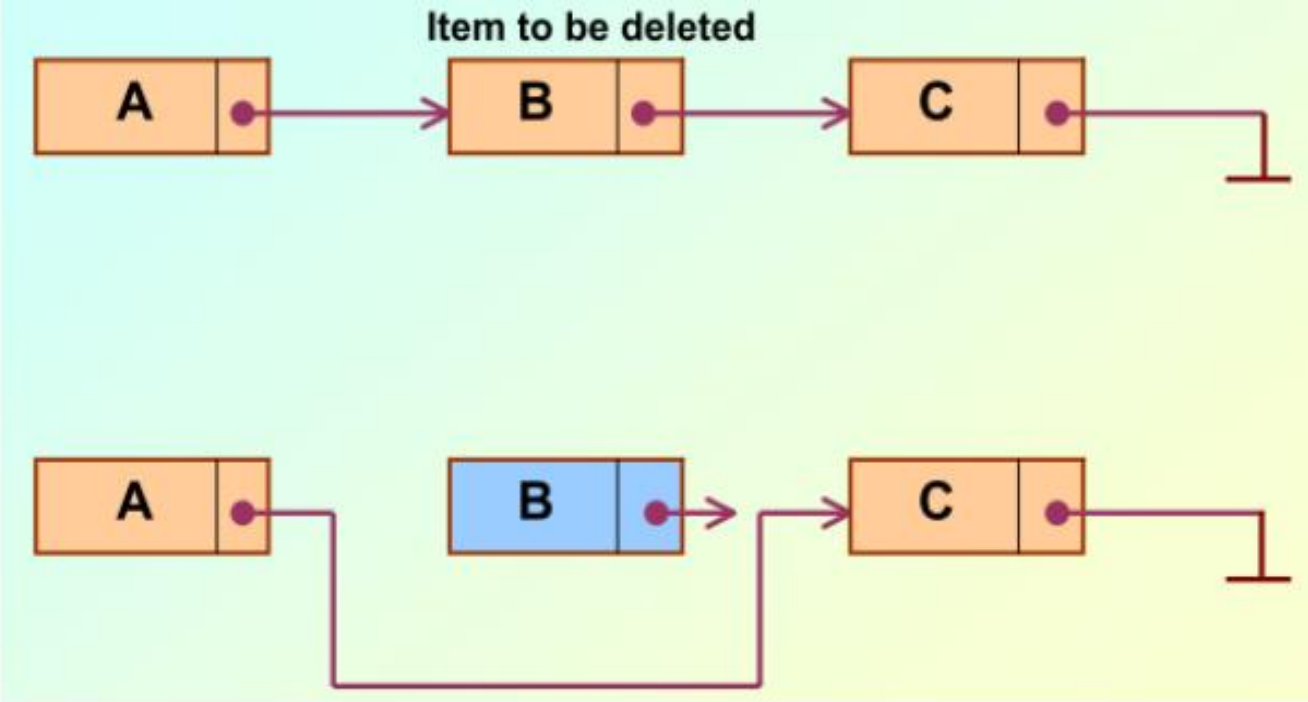
```
49 void dugumEkle(dugum **head)
50 {
51     int kayitNo;
52     dugum *p, *q, *yeni;
53     yeni = (dugum *) malloc(sizeof(dugum));
54     printf("\nEklenecek veriyi gir.Numara?Ad?Yas?\n");
55     scanf("%d %s %d",&yeni->numara,yeni->ad,&yeni->yas);
56
57     printf("Hangi kayit no'dan once eklenecek");
58     scanf("%d",&kayitNo);
59
60     p= *head;
61     if(p->numara==kayitNo) //başa eklenecekse
62     {
63         yeni->sonraki = p;
64         *head = yeni;
65     }
```

Inserting a Node in a List

```
69      else
70      {
71          while(p->sonraki!=NULL && p->numara!=kayitNo)
72          {
73              q = p;
74              p= p->sonraki;
75          }
76          if(p==NULL) //Sona eklenecekse
77          {
78              q->sonraki = yeni;
79              yeni->sonraki = NULL;
80          }
81          else if(p->numara == kayitNo) //Araya eklenecekse
82          {
83              yeni->sonraki = p;
84              q->sonraki = yeni;
85          }
86      }
87  }
```

Deleting a Node from the List

- For deletion:
  - The next pointer of the item immediately preceding the one to be deleted is altered, and made to point to the item following the deleted item.



Deleting a Node from the List

- To delete a specified node.
  - Say, the node whose roll field is given.
- Three conditions arise:
  - Deleting the first node.
  - Deleting the last node.
  - Deleting an intermediate node.

Deleting a Node from the List

```
85  void dugumSil(dugum **head)
86  {
87      int kayitNo;
88      dugum *p, *q;
89      printf("Hangi kayit no silinecek");
90      scanf("%d",&kayitNo);
91
92      p= *head;
93      if(p->numara==kayitNo)//ilk düğüm siliniyorsa
94      {
95          head = p->sonraki;
96          free(p);
97      }
```

Deleting a Node from the List

```
101         else
102         {
103             while(p->sonraki!=NULL && p->numara!=kayitNo)
104             {
105                 q = p;
106                 p= p->sonraki;
107             }
108             if(p==NULL)
109             {
110                 printf("Uygun kayit bulunamadi.Silme basarisiz!");
111             }
112             else if(p->numara == kayitNo) //Aradaki silinecekse
113             {
114                 q->sonraki = p->sonraki;
115                 free(p);
116             }
117     }
```

Singly Linked List Application-1

```
116 int main(void)
117 {
118     int secim=0;
119     dugum * aktif;
120
121     printf("1-Liste Olustur\n2-Liste Dolas\n3-Dugum Sil\n4-Dugum Ekle\n5-Cikis");
122     while(1)
123     {
124         printf("\nSecim [1-5]?\n");
125         scanf("%d",&secim);
126         switch(secim)
127         {
128             case 1 : aktif = listeOlustur();
129                      listeDolas(aktif);
130                      break;
131             case 2 : listeDolas(aktif);break;
132             case 3 : dugumSil(&aktif);
133                      listeDolas(aktif);
134                      break;
135             case 4 : dugumEkle(&aktif);
136                      listeDolas(aktif);
137                      break;
138             case 5 : exit(0);break;
139             default: printf("Yanlis secim");break;
140         }
141     }
142     while( getchar() != '\n' ) { /*do nothing*/} ;
143         getchar() ; /* wait */
144     return 0;
145 }
```

Singly Linked List Application-2

- A linear list application that has capability of listing nodes in alphabetical order, inserting nodes, deleting a specified node and finding the record that has maximum number of characters in the list.

Singly Linked List Application-2

```
 5  struct listyapi
 6  {
 7    char   adi[21];
 8    struct listyapi *sonraki;
 9  };
10
11  //artık listyapi yerine listnode kullanılacak
12  typedef struct listyapi listnode;
13  // *headnode listnode tipinde bir işaretçidir.
14  //Herzaman listenin başını gösterecek
15  listnode *headnode;
```

Singly Linked List Application-2

```
17  void ara(char *searchthis, listnode **prevnode) /
18  {
19    listnode *c;
20    c = headnode;
21    *prevnode = c;
22    while ((c->sonraki != NULL))
23      {
24        c = c->sonraki;
25        if (strcmp(c->adi, searchthis) >= 0) break;
26        *prevnode = c;
27      }
28  }
```

Singly Linked List Application-2

```
29 void kayit(char *s)
30 /* prevnode kayidi newnode kayidini,
31 newnode kayidi prevnode'nin daha once gosterdigi kayidini gosterir. */
32 {
33   listnode *newnode, *prevnode;
34   newnode = (listnode *) malloc(sizeof(listnode)); /* yeni kayida yer at */
35   strcpy(newnode->adi, s);              /*bilgiyi yeni kayida yaz */
36   ara(newnode->adi, &prevnode);  /* listedeki yerini bul */
37   newnode->sonraki = prevnode->sonraki;  /* listeye ekle */
38   prevnode->sonraki = newnode;
39 }
```

Singly Linked List Application-2

```
40  void iptal(char *s)
41  /* newnode kayidi silinir.
42  prevnode kayidi newnode kayidinin
43  gosterdigi kayidini gosterir. */
44  {
45    listnode *newnode, *prevnode;
46    ara(s, &prevnode);
47    newnode = prevnode->sonraki;
48    prevnode->sonraki = newnode->sonraki;
49    free(newnode);
50  }
```

Singly Linked List Application-2

```
51 void listlist(void)
52 {
53   listnode *currentnode;
54   currentnode = headnode;
55   if (currentnode != NULL) currentnode = currentnode->sonraki;
56   while (currentnode != NULL)
57     {
58       printf("%s  ",currentnode->adi);
59       currentnode = currentnode->sonraki;
60     }
61   printf("\n");
62 }
```

Singly Linked List Application-2

```
63 void enUzunBul(void)
64 {
65   listnode *currentnode, *enuzun;
66   currentnode = headnode;
67   if (currentnode != NULL)
68     {
69         currentnode = currentnode->sonraki;
70     }
71   enuzun=currentnode;
72   while (currentnode != NULL)
73     {
74       if (strlen(currentnode->adi) >= strlen(enuzun->adi))
75         {
76           enuzun = currentnode;
77         }
78       currentnode = currentnode->sonraki;
79     }
80   printf("%s  %d",enuzun->adi,strlen(enuzun->adi));
81   getchar();
82 }   /* En uzun isim; */
```

Singly Linked List Application-2

```
83  void main()
84  {
85    char      sec;
86    char      s[21];
87    headnode = (listnode *) malloc(sizeof(listnode));
88    strcpy(headnode->adi," listenin basi");
89    headnode->sonraki = NULL;
90    do
91      {
92        system("cls");
93        listlist();
94        printf("\n\n1 - Ekle\n2 - iptal\n3 - En Uzun isim\n4 - Çıkış\n\nSec :");
95        sec = getche();
96        switch (sec)
97          {
98          case '1':printf("\nAdi :"); gets(s);
99            kayit(s);break;
100         case '2':printf("Adi \n");  gets(s);
101           iptal(s);break;
102         case '3':enUzunBul();break;
103         case '4':exit(0);break;
104         }
105     }
106   while (1);
107 }
```