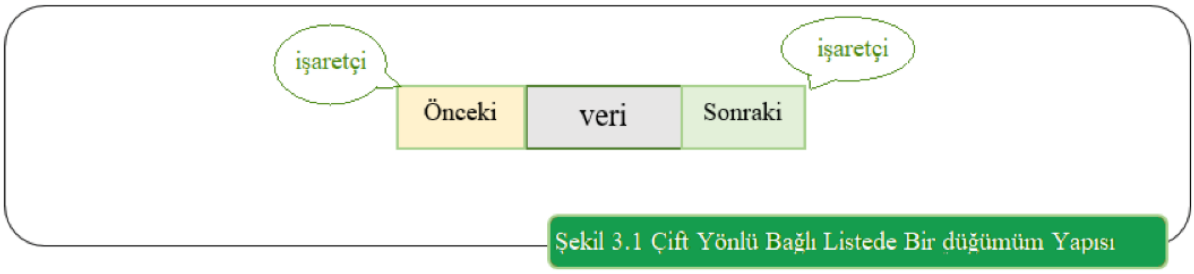
3. ÇİFT YÖNLÜ BAĞLI LİSTELER

Giriş

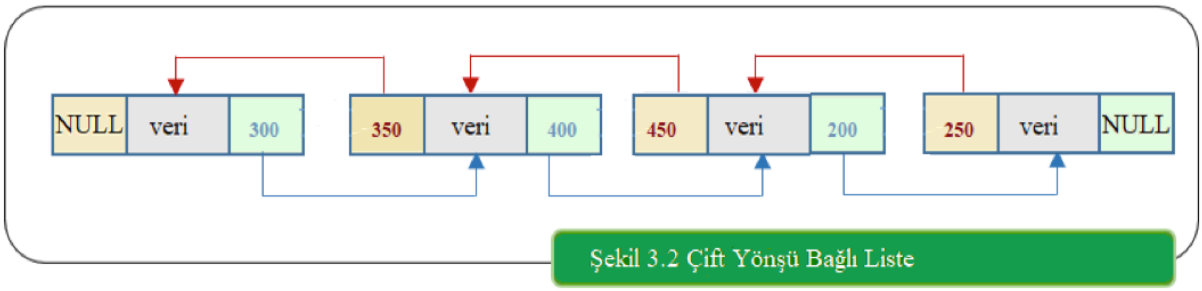
Bağlı listelerde bir düğüm kendisinden sonra gelen düğümün adres bilgisini ve kendisinden önceki düğümün adres bilgisini içerebilir. Bu şekilde bir yapıya sahip bağlı listelere çift yönlü bağlı liste (**Doubly Linked List**) denir. Başka bir ifade ile kendisinden önceki ve sonraki düğümlerin adreslerini saklamak üzere iki adet işaretçisi olan bağlı listelere çift yönlü bağlı liste veya iki yönlü bağlı liste denir.

Çift yönlü bağlı listelerde bir düğümde bulunan iki işaretçiden önceki düğümün adresini gösteren işaretçi **önceki**, sonraki düğümün adresini gösteren işaretçi **sonraki** olarak isimlendirilebilir. Şekil 3.1 'de çift yönlü bağlı listede bir düğümün yapısı gösterilmiştir.



Şekil 3.1 Çift Yönlü Bağlı Listede Bir düğümüm Yapısı

Şekil 3.2'de çift yönlü bağlı listenin doğrusal gösterimi verilmiştir. Şekilden de anlaşılacağı gibi Çift yönlü bağlı listeler (DLL), tek bağlantılı listede bulunan sonraki işaretçi ve verilerle birlikte, önceki işaretçi olarak adlandırılan ekstra bir işaretçiye de sahiptir. Çift Yönlü Bağlı Listeler doğrusal olarak tasarlanabileceği gibi dairesel olarak da tasarlanabilir Dairesel Bağlı Listeler Bölüm 4 'de ele alınacaktır.



Şekil 3.2 Çift Yönşü Bağlı Liste

Çift Yönlü Bağlı Listelerde son düğümden sonra liste başına geri gitmek mümkündür. Bunun için son düğümden itibaren listenin

elemanları geriye doğru taranır. Bu işlemin n-1 elemana erişmek anlamına geldiğini söyleyebiliriz.

C programlama dili kullanılarak, işaretçi yaklaşımı ile bir yapı aşağıdaki şekilde tanımlanır;

```
struct cbagliListe{
    int veri1;
    int veri2;
    struct cbagliListe *onceki;
    struct cbagliListe *sonraki;
};
struct cbagliListe* ilkDugum=NULL;
struct cbagliListe* sonDugum=NULL;
```

Yukarıdaki tanımı yapılan çift yönlü bağlı listenin her düğümünde tamsayı türünde iki veri saklanabileceği *(tanıma bağlı olarak her düğümde bir veya birden çok, aynı tipte veya farklı tipte veri/veriler saklanacak tanımlamalar yapılabilir.)* ve her düğümün biri önceki diğeri sonraki olmak üzere iki işaretçiye sahip olduğu görülmektedir. Yapı tanımlandıktan hemen sonra ilkDugum ve sonDugum adli iki adet global yapı değişkeni tanımlandığına da dikkat ediniz.

Çift Yönlü Bağlı Listelere, Tek Yönlü Bağlı Listelerde olduğu gibi listenin başına, sonuna veya düğümler arasına eleman eklenebilir. Aynı şekilde listenin başından sonundan veya düğümlerin arasından eleman silme işlemi gerçekleştirilebilir.

Çift Yönlü Bağlı Listelerin Avantajları ve Dezavantajları

Avantajları

- Liste üzerinde çift yönlü hareket edilebilir,
- Ekleme, Silme gibi bazı işlemler daha kolaydır.

Dezavantajları

- Önceki işaretçi için bellekte fazladan yer kaplar,
- Her düğümün önceki (prev) ve sonraki (next) adında iki işaretçisi olduğu için liste işlemleri daha yavaştır,

- Hata yapılma ihtimali yüksektir. Örneğin listeye eleman ekleme sırasında, önceki işaretçileri sonraki işaretçilerle birlikte değiştirmemiz gerekir. Herhangi bir işlemin atlanması hataya neden olur.

3.1. Çift Yönlü Bağlı Liste Oluşturmak

Çift yönlü bağlı liste oluşturmak için, tek yönlü bağlı liste oluştururken yaptığımız gibi bir fonksiyon yazacağız. Oluşturacağımız fonksiyon, tanımlanan yapı tipinde olacak ve düğümlerde veri için ayrılacak alanlarda tutulacak bilgiler karşılamak amacıyla tamsayı tipinde iki parametre (***int veri1, int veri2***) alacak. Aşağıda çift yönlü bağlı listelerde düğüm oluşturan ve oluşturulan düğümü geri döndüren fonksiyon verilmiştir.

```
struct cbagliListe *dOlustur(int veri1, int veri2){
    struct cbagliListe* yeniDugum=
    (struct cbagliListe*)malloc(sizeof(struct cbagliListe));
    yeniDugum->veri1=veri1;
    yeniDugum->veri2=veri2;
    yeniDugum->onceki=NULL;
    yeniDugum->sonraki=NULL;
    return yeniDugum;
}
```

Yukarıda verilen fonksiyondan da görülebileceği gibi, önce ***dOlustur()*** fonksiyonu içerisinde önceden tanımlanmış olan yapı tipinde ***yeniDugum*** ismiyle bir düğüm tanımlanıyor ve malloc() fonksiyonu kullanılarak bu düğüm için bellekte yer ayrılıyor. Bellekte düğüm için alan ayrıldıktan sonra **yeniDugum** ‘e bilgiler aktarılır. Bunun için fonksiyon gövdesinde aşağıdaki satırlar yazılmıştır;

```
yeniDugum->veri1=veri1;
yeniDugum->veri2=veri2;
yeniDugum->onceki=NULL;
yeniDugum->sonraki=NULL;
```

Bilgilerin yeniDugum ‘e aktarılmasından sonra ***return yeniDugum;*** ifadesiyle düğümün geri dönmesi sağlanmıştır.

Çift yönlü bağlı liste oluşturmak için yazılan programın tamamı aşağıda, program 3.1 ‘ de verilmiştir. Program 3.1 ‘de dOluştur() fonksiyonundan başka, düğümlerde saklanan verilerin ve işaretçilerin değerlerini yazdırmak amacıyla listeYazdir() isimli bir fonksiyon daha yazılmıştır. ***listeYazdir(struct cbagliListe****

ilkDugum, int yon) fonksiyonu iki farklı parametre almıştır. Bu parametrelerden ilki yazdırma işleminin başlangıç noktasını tespit etmek için, ikincisi ise yazdırma işleminin listede hangi yönde ilerlenerek yapılacağını belirlemek için kullanılmıştır.

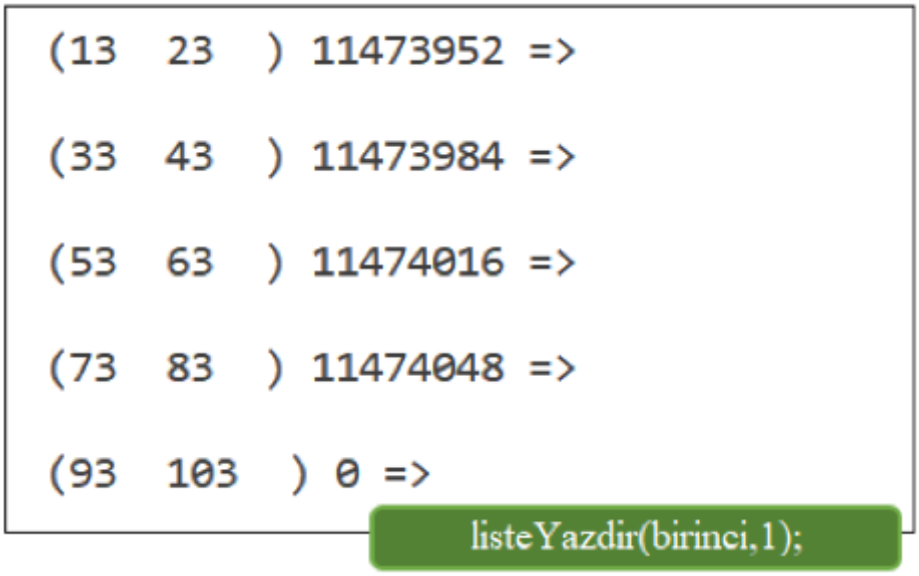
```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
struct cbagliListe{
    int veri1;
    int veri2;
    struct cbagliListe *onceki;
    struct cbagliListe *sonraki;
};
struct cbagliListe* ilkDugum=NULL;
struct cbagliListe* sonDugum=NULL;

struct cbagliListe *dOlustur(int veri1, int veri2){
    struct cbagliListe* yeniDugum=
    (struct cbagliListe*)malloc(sizeof(struct cbagliListe));
    yeniDugum->veri1=veri1;
    yeniDugum->veri2=veri2;
    yeniDugum->onceki=NULL;
    yeniDugum->sonraki=NULL;
    return yeniDugum;
}
void listeYazdir(struct cbagliListe* ilkDugum, int yon){
    struct cbagliListe* temp=ilkDugum;
    while(temp!=NULL)
    {
        printf("\n(%d  %d  )",temp->veri1,temp->veri2);
        if(yon==1){
          printf(" %d =>\n",temp->sonraki);
          temp=temp->sonraki;
            }
          else{
          printf(" %d =>\n",temp->onceki);
          temp=temp->onceki;
                }
}   }

int main() {
    struct cbagliListe *birinci=dOlustur(13,23);
    struct cbagliListe *ikinci=dOlustur(33,43);
    struct cbagliListe *ucuncu=dOlustur(53,63);
    struct cbagliListe *dorduncu=dOlustur(73,83);
    struct cbagliListe *besinci=dOlustur(93,103);
    ilkDugum=birinci;
    sonDugum=besinci;
    birinci->sonraki=ikinci;
    ikinci->sonraki=ucuncu;
    ucuncu->sonraki=dorduncu;
    dorduncu->sonraki=besinci;
    besinci->onceki=dorduncu;
    dorduncu->onceki=ucuncu;
    ucuncu->onceki=ikinci;
    ikinci->onceki=birinci;
    listeYazdir(birinci,1);
    return 0;
}
```

Program 3.1

Şimdi İki Yönlü Bağlı Listelerin özelliklerini daha iyi anlayabilmek için Program 3.1 ‘de yer alan ***void listeYazdir(struct cbagliListe****

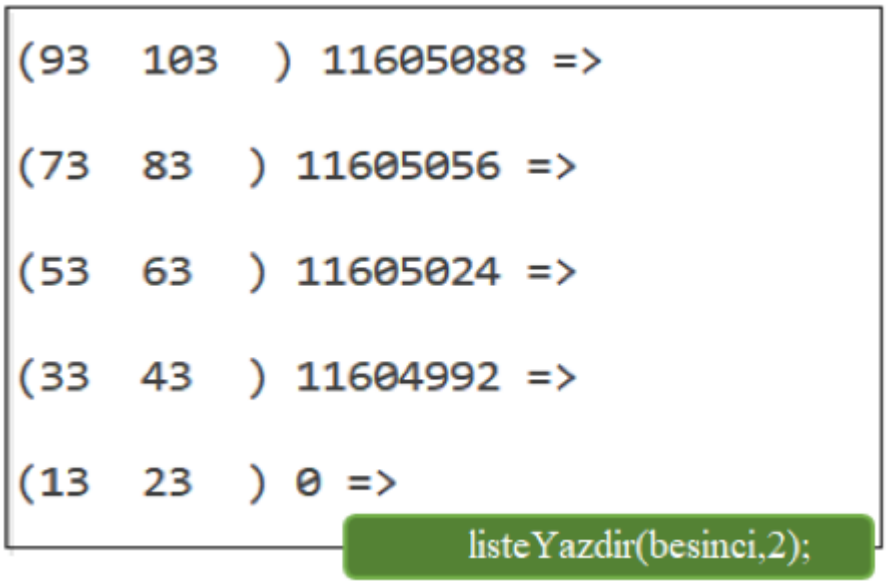
ilkDugum, int yon) fonksiyonunu farklı parametrelerle çalıştıralım ve her defasında elde edilen ekran çıktılarını inceleyelim. Biz bu işlem için fonksiyonun parametrelerini aşağıdaki şekilde değiştirdik ve farklı ekran çıktıları elde ettik.

1. listeYazdir(birinci,1);



Yazdırma işlemi ilk düğümden başlayıp son düğüme kadar gerçekleştirilmiştir. Bölüm 2'de, Tek Yönlü Bağlı Listelerde de bu fonksiyonun parametre almadan aynı çıktıya benzer bir çıktı sağladığını hatırlayınız.

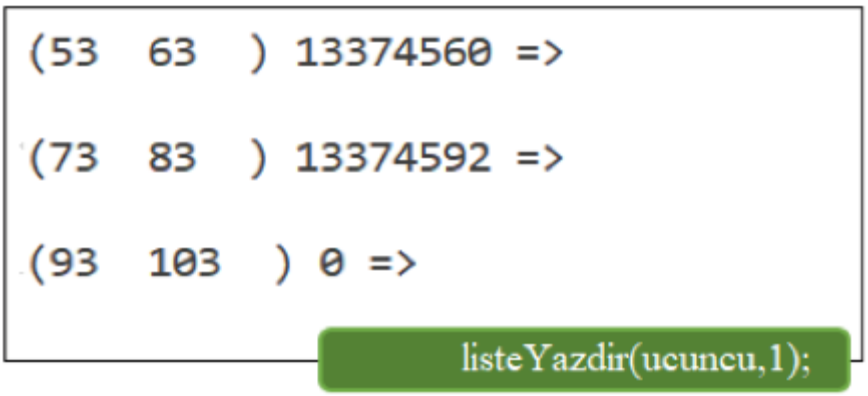
2. listeYazdir(besinci,2);



Yazdırma işlemi son düğümden başlayıp ilk düğüme kadar devam etmiştir. Çünkü İki Yönlü Bağlı listelerde önceki düğümü işaret eden

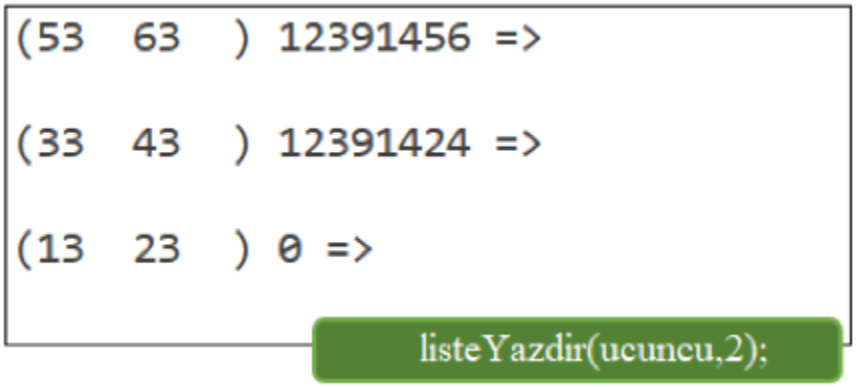
bir işaretçi de bulunduğu içim düğüm üzerinde soldan sağa doğru ilerlenebildiği gibi sağdan sola doğru da kolayca ilerlenebilmektedir.

3. listeYazdir(ucuncu,1);



Yazdırma işlemi üçüncü düğümden başlatılıp sağa doğru son düğüme kadar ilerletilmiştir.

4. listeYazdir(ucuncu,2);



Yazdırma işlemi üçüncü düğümden başlatılıp sola doğru son düğüme kadar ilerletilmiştir.

Sizde listeYazdir() fonksiyonunun parametrelerini değiştirerek farklı ekran çıktıları oluşturabilirsiniz.

3.2. Çift Yönlü Bağlı Listede Düğüm Sayısını Bulmak

C programlama dilinde yazılan ve Çift Yönlü Bağlı Listedeki eleman sayısını fonksiyon aşağıda verilmiştir.

```
69  void Say(){
70      int kontrol = 0,sayac=1;
71      if(ilkDugum == NULL){
72          printf("\n ==>Listede Herhangi Bir eleman yok \n");
73          return;
74      }
75       struct cbagliListe* temp = ilkDugum;
76       while(temp->sonraki != NULL){
77          sayac+=1;
78          kontrol=1;
79          temp = temp->sonraki;
80      }
81      if(kontrol == 1){
82          printf("\n ==>Listede %d Dügüm var \n",sayac);
83          return;
84      }
85  }
```

Çift Yönlü Bağlı Listelerde Eleman Sayısını Bulan Fonk

Say() fonksiyonunda, önce listede düğümün olmadığı durumda kullanıcıya mesaj verilmesi ve fonksiyondan çıkılmasını sağlayan ***if*** bloğu (71 -74. Satırlar) yazılmıştır. Sonra ***while*** bloğu içerisinde traverse (dolaşma) işlemi sırasında ***sayac*** arttırılmış, ***kontrol*** değişkeninin değeri değiştirilmiş ve daha sonra ***while*** dışında ***sayac*** değeri düğüm sayısı olarak ekrana yazdırılmıştır.

3.3. Çift Yönlü Bağlı Listede Aranan Elemanı Bulmak

```
42  void Ara(int arananSayi){
43      int kontrol = 0,sayac=1;
44      if(ilkDugum == NULL){
45          printf("\n ==>Listede Herhangi Bir eleman yok \n");
46          return;
47      }
48      if((ilkDugum->veri1 == arananSayi)||(ilkDugum->veri2==arananSayi) ){
49          printf("\n ==>Aranan sayi %d inci düğümde bulundu ( %d  %d )\n"
50          ,sayac,ilkDugum->veri1,ilkDugum->veri2);
51          return;
52      }
53       struct cbagliListe* temp = ilkDugum;
54       while(temp->sonraki != NULL){
55          sayac+=1;
56          if((temp->sonraki->veri1 == arananSayi)||(temp->sonraki->veri2 == arananSayi)){
57              kontrol = 1;
58              printf("\n ==>Aranan sayi %d inci düğümde bulundu ( %d  %d )\n"
59              ,sayac,temp->sonraki->veri1,temp->sonraki->veri2);
60              break;
61          }
62          temp = temp->sonraki;
63      }
64      if(kontrol == 0){
65          printf("\n ==>Arama Tamamlandı Listede Aranan Eleman Bulunamadı\n");
66          return;
67      }
68  }
```

Çift Yönlü Bağlı Listede Aranan Elamanı Bulan Fonk

Ara() fonksiyonu, listede aranan değeri parametre olarak almıştır. Fonksiyon içerisinde önce listede eleman olmaması durumu kontrol edilmiş ve eğer listede eleman bulunmuyorsa kullanıcıya mesaj verilip fonksiyonun çalışması sonlandırılmıştır (44-47. satırlar). Sonra aranan elemanının ilk düğümde olması durumu değerlendirilmiş ve kontrolden "True" dönmesi halinde kullanıcıya uygun mesaj verilip fonksiyonun çalışması sonlandırılmıştır (48-52. satırlar). Daha sonra bu iki kontrolden de "False" değeri geri dönmesi halinde diğer düğümlerde arama yapılabilmesi için while bloğu (53, 63. satırlar) oluşturulmuştur. while bloğu içerisinde traverse (dolaşma) işlemi ile düğümler ileriye doğru taranmış ve karşılaştırma yapılarak aranan değerin düğümlerde olup olmadığı araştırılmıştır. Aranan sayı traverse işlemi sırasında herhangi bir düğümde bulunmuşsa sonuç ekrana yazdırılmış ve **break** ile döngüden çıkılmıştır. Fonksiyonda, son olarak son düğüme kadar ilerlenmesi ve aranan sayının bulunamaması durumu değerlendirilerek kullanıcıya mesaj verilmiştir (64-67. satırlar).

3.4. Çift Yönlü Bağlı Listelere Eleman Eklemek

3.4.1. Çift Yönlü Bağlı Listenin Sonuna Eleman Eklemek

C programlama dilinde yazılan ve Çift Yönlü Bağlı Listenin Sonuna eleman ekleyen fonksiyon aşağıda verilmiştir.

```
21  void sonaEkle(int veri1,int veri2){
22      if(ilkDugum == NULL) {
23      ilkDugum = (struct cbagliListe *)malloc(sizeof(struct cbagliListe));
24      ilkDugum -> veri1 = veri1;
25      ilkDugum -> veri2 = veri2;
26      ilkDugum -> sonraki = NULL;
27      ilkDugum -> onceki = NULL;
28      }
29      else {
30            struct cbagliListe *temp1 = ilkDugum;
31            struct cbagliListe *temp2 =
32            (struct cbagliListe *)malloc(sizeof(struct cbagliListe));
33            while(temp1 -> sonraki != NULL)
34            temp1 = temp1 -> sonraki;
35            temp2 -> veri1 = veri1;
36            temp2 -> veri2 = veri2;
37            temp2 -> sonraki = NULL;
38            temp2 -> onceki = temp1;
39            temp1 -> sonraki = temp2;
40  }
41      }
```

Çift Yönlü Bağlı Listenin Sonuna Eleman Ekleyen Fonksiyon

sonaEkle() fonksiyonunda önce ***if(ilkDugum == NULL)*** ifadesiyle kontrol yapılır ve bu kontrolün sonunda geriye ***"True"*** dönerse bellekte ***ilkDugum*** için alan ayrılır. Sonra, düğüme ***veri, veri2*** değerleri atanır,
düğümün ***sonraki*** ve ***onceki*** işaretçileri ***NULL*** yapılır. Kontrol sonucunda ***"False"*** değeri geri dönerse bu durumda bağlı listede eleman vardır ve bu bloktaki ifadeler
çalıştırılmayıp, ***if*** bloğunun ***else*** kısmındaki ifadeler çalıştırılır.

Çift Yönlü Bağlı Listede ***if*** bloğunun ***else*** kısmındaki ifadelerin çalışacağı koşul oluşursa (Bağlı listede düğüm/düğümler bulunuyorsa) ***ilkDugum temp1***'e atanır (30. satırdaki ifade) ve bellekte ***temp2*** için alan ayrılır (31. Satırdaki ifade). Sonra traverse (dolaşma) işlemi ile son düğüme gidilir (33,34. satırlardaki ifadeler), ***veri1*** ve ***veri2*** değerleri ***temp2***'ye atanır (35,36. satırlardaki ifadeler). ***temp2***'nin sonraki işaretçisi ***NULL*** yapılır (37. satırdaki
ifadeler). ***temp2***'nin ***onceki*** işaretçisinin ***temp1***'i, ***temp1***'in ***sonraki*** i şaretçisinin de ***temp2***'yi işaret etmesi sağlanır. (38, 39. satırlardaki ifadeler)

3.4.2. Çift Yönlü Bağlı Listenin Başına Eleman Eklemek

C programlama dilinde yazılan ve Çift Yönlü Bağlı Listenin Sonuna eleman ekleyen fonksiyon aşağıda verilmiştir.

```
43  void basaEkle(int veri1,int veri2){
44      if(ilkDugum==NULL)
45      {
46          ilkDugum = (struct cbagliListe *)malloc(sizeof(struct cbagliListe));
47          ilkDugum -> veri1 = veri1;
48          ilkDugum -> veri2 = veri2;
49          ilkDugum -> sonraki = NULL;
50          ilkDugum -> onceki = NULL;
51          }
52          else {
53                  struct cbagliListe *temp =
54                  (struct cbagliListe *)malloc(sizeof(struct cbagliListe));
55                   temp -> veri1 = veri1;
56                   temp -> veri2 = veri2;
57                   temp -> sonraki = ilkDugum;
58                   temp -> onceki = NULL;
59                   ilkDugum -> onceki = temp;
60                   ilkDugum = temp;
61  }
62  }
```

Çift Yönlü Bağlı Listelerin Başına Eleman Ekleyen Fonksiyon

basaEkle() fonksiyonunda önce ***if(ilkDugum == NULL)*** ifadesiyle kontrol yapılır ve bu kontrolün sonunda geriye ***"True"*** dönerse bellekte ***ilkDugum*** için alan ayrılır. Sonra, düğüme ***veri, veri2*** değerleri atanır, düğümün ***sonraki*** ve ***onceki*** işaretçileri ***NULL*** yapılır. Kontrol sonucunda ***"False"*** değeri geri dönerse bu durumda bağlı listede eleman vardır ve bu bloktaki ifadeler çalıştırılmayıp, ***if*** bloğunun ***else*** kısmındaki ifadeler çalıştırılır.

Çift Yönlü Bağlı Listede ***if*** bloğunun ***else*** kısmındaki ifadelerin çalışacağı koşul oluşursa (Bağlı listede düğüm/düğümler bulunuyorsa) öncelikle ***temp*** için bellekte yer ayrılır. ***veri1*** ve ***veri2*** değerleri ***temp*** 'e atanır (55, 56. satırlardaki ifadeler). ***temp '***in ***sonraki*** işaretçisinin ***ilkDugum '***e işaret etmesi sağlanır (57. Satırdaki ifade). ***temp***'in ***onceki*** işaretçisi NULL yapılır (58. satırdaki ifade). ***ilkDugum*** 'ün ***onceki*** işaretçisi temp 'i işaret etmesi sağlanır (59. Satırdaki ifade). Fonksiyonda son satırda ***ilkDugum=temp*** güncellemesi yapılarak listenin başına eleman ekleme işlemi gerçekleştirilir.

3.4.3. Çift Yönlü Bağlı Listenin Arasına Eleman Eklemek

C programlama dilinde yazılan ve Çift Yönlü Bağlı Listelerde listenin arasına eleman ekleyen fonksiyon aşağıda verilmiştir.
Fonksiyon ***veri1***, ***veri2*** ve ***sira*** isimli tamsayı tipinde üç parametre almaktadır. ***sira*** isimli parametre araya eklenecek düğümün listenin neresine ekleneceğini belirlemek için kullanılacaktır.

```
64  void arayaEkle(int veri1,int veri2,int sira){
65      struct cbagliListe* ArayaEklenecek = dOlustur(veri1,veri2);
66
67      if(sira == 0){
68          basaEkle(veri1,veri2);
69          return;
70      }
71      if(ilkDugum == NULL && sira > 0){
72          printf("\n Ekleme işlemi yapılamaz ");
73          return;
74      }
75      int sayac = 0,kontrol=0;
76      struct cbagliListe* temp = ilkDugum;
77      while(temp != NULL){
78          if(sayac == sira){
79              kontrol = 1;
80              break;
81          }
82          temp = temp->sonraki;
83          sayac++;
84      }
85      if(kontrol == 0){
86          printf("\n Eklenecek Pozisyon Yok\n");
87          return;
88      }
89      if(temp->sonraki == NULL){
90          sonaEkle(veri1,veri2);
91          return;
92      }
93      struct cbagliListe* OncekiDugum = temp->onceki;
94      OncekiDugum->sonraki = ArayaEklenecek;
95      ArayaEklenecek->onceki = OncekiDugum;
96      ArayaEklenecek->sonraki = temp;
97      temp->onceki = ArayaEklenecek;
98  }
```

Çift Bağlı Listede elemanların arasına Yeni Eleman Ekleyen Fonk.

1.

```
struct cbagliListe* ArayaEklenecek = dOlustur(veri1,veri2);
```

Yukarıdaki ifade ile fonksiyonda ArayaEklenecek() isimli bir düğüm oluşturulur.

2.

```
if(sira == 0){
    basaEkle(veri1,veri2);
    return;
}
```

sira sıfır (0) ise eklenecek düğüm listenin başına gelecektir. Bu nedenle ***basaEkle(veri1, veri2);*** fonksiyonu çağırılıp eklenecek

eleman listenin başına eklenir. Daha sonra ***return*** ile fonksiyonun çalışması sonlandırılır.

3.

```
if(ilkDugum == NULL && sira > 0){
    printf("\n Ekleme işlemi yapılamaz ");
    return;
}
```

ilkDugum NULL ve **sira> 0** ise eklenecek eleman için listede anlamlı bir konum yoktur. Bu durumda kullanıcıya uygun mesaj verilip return ile fonksiyonun çalışması sonlandırılır.

4.

```
struct cbagliListe* temp = ilkDugum;
int sayac = 0,kontrol=0;
while(temp != NULL){
    if(sayac == sira){
        kontrol = 1;
        break;
    }
    temp = temp->sonraki;
    sayac++;
}
```

Fonksiyonda yer alan yukarıdaki kod bloğunda önce ***temp*** adında bir düğüm oluşturuluyor ve ***temp=ilkDugum*** güncellemesi yapılıyor. Sonra traverse (dolaşma) işleminde kaçıncı sıradaki düğüme gelindiği bilgisini tutmak için ***sayac*** değişkeni ve ***sayac*** 'ın ***sira*** 'ya eşitlendiği durumu tespit etmek için ***kontrol*** değişkeni tanımlanıyor.

Daha sonra while bloğunda ***sayac == sira*** eşitliği sağlandığında ***kontrol*** değişkeninin değeri ***bir (1)*** yapılıp ***break*** ile döngüden çıkılıyor.

sayac == sira eşitliğinin sağlanmadığı her adımda ise traverse (dolaşma) işlemi devam ediyor ve sayac değişkeninin değeri bir (1) arttırılıyor.

5.

```
if(kontrol == 0){
    printf("\n Eklenecek Pozisyon Yok\n");
    return;
}
```

while döngüsünün çalışması sona erdiğinde ***sayac == sıra*** eşitliği sağlanmamış, yani ***kontrol*** değişkenin değeri ***sıfır (0)*** kalmış ise, yukarıdaki ***kontrol == 0*** karşılaştırması "***True"*** değeri geri döndürür. Bu, eleman eklemek için uygun sıra olmadığı anlamına gelir ve o nedenle kullanıcıya mesaj verilerek fonksiyonun çalışması sonlandırılır.

6.

```
if(temp->sonraki == NULL){
    sonaEkle(veri1,veri2);
    return;
}
```

Yukarıdaki blok, traverse (dolaşma) işlemi sırasında son düğüme gelinme halidir. Bu durumda elemanın sona eklenmesi gerekir. Bu işlem için ***sonaEkle()*** fonksiyonu çağırılmıştır. İşlem tamamlandıktan sonra fonksiyonun çalışması ***return*** ifadesi ile sonlandırılmıştır.

7.

```
struct cbagliListe* OncekiDugum = temp->onceki;
OncekiDugum->sonraki = ArayaEklenecek;
ArayaEklenecek->onceki = OncekiDugum;
ArayaEklenecek->sonraki = temp;
temp->onceki = ArayaEklenecek;
```

Yukarıdaki bloktaki ifadeler, fonksiyonda ***basaEkle()*** çağırılarak çalıştırılıp veya ***sonaEkle()*** çağırılarak çalıştırılıp veya eleman ekleme işleminin gerçekleştirilmeden fonksiyondan çıkıldığı bu üç farklı durumdan herhangi birinin gerçekleşmediği halde çalışacak ifadelerdir.

3.5. Çift Yönlü Bağlı Listelerden Eleman Silmek

3.5.1. Çift Yönlü Bağlı Listelerin Son Elemanını Silmek

C programlama dilinde yazılan ve Çift Yönlü Bağlı Listelerde listenin sonundan eleman silen fonksiyon aşağıda verilmiştir.

```
127  void bastanElemanSil()
128  {
129      if(ilkDugum == NULL)
130          return;
131      if(ilkDugum->sonraki == NULL)
132      {
133          ilkDugum = NULL;
134          return;
135      }
136      struct cbagliListe* Ikinci = ilkDugum->sonraki;
137      Ikinci->onceki = NULL;
138      free(ilkDugum);
139      ilkDugum = Ikinci;
140  }
```

sondanElemanSil() fonksiyonunda önce ***if(ilkDugum == NULL)*** ile listede düğüm olup olmadığı kontrol ediliyor (112. Satır). Karşılaştırmanın sonucu ***"True"*** ise listede silinecek eleman yoktur. Bu durumda ***return*** (113. Satır) ile fonksiyondan çıkılıyor.

114 ve 118. Satırlar arasında yazılan ***if*** bloğunda, listede sadece bir eleman olup olmadığı kontrol ediliyor ve listede sadece bir elemanın olması durumunda ***bastanElemanSil()*** fonksiyonu çağırılarak listedeki eleman siliniyor.

120. satırda temp isimli düğüm oluşturuluyor ve ***temp = ilkDugum*** ataması yapılıyor.

121 ve 125. Satırlar arasında kalan ***while*** bloğunda önce traverse (dolaşma) işlemi ile son düğüm bulunuyor.
Sonra, ***sondanOnceki*** isimli bir düğüm tanımlanıp ***sondanOnceki = temp->onceki*** ile sondan bir önceki düğüme gidiliyor ve 124. Satırda bu düğümün sonraki işaretçisi NULL yapılıp 125. Satırda listenin en sonundaki düğüm ***(temp)*** siliniyor.

3.5.2. Çift Yönlü Bağlı Listelerin Başındaki Elemanı Silmek

C programlama dilinde yazılan ve Çift Yönlü Bağlı Listelerde listenin başından eleman silen fonksiyon aşağıda verilmiştir.

```
void bastanElemanSil()
{
    if(ilkDugum == NULL)
        return;
    if(ilkDugum->sonraki == NULL)
    {
        ilkDugum = NULL;
        return;
    }
    struct cbagliListe* Ikinci = ilkDugum->sonraki;
    Ikinci->onceki = NULL;
    free(ilkDugum);
    ilkDugum = Ikinci;
}
```

Çift Yönlü Bağlı Listelerde Baştan Eleman Silen Fonksiyon

bastanElemanSil() Fonksiyonunda;

1.

```
if(ilkDugum == NULL)
    return;
```

İlk önce ifadeleri kullanılarak, bağlı listede eleman olmadığı durumda fonksiyondan çıkılması sağlanmıştır.

2.

Daha sonra;

```
if(ilkDugum->sonraki == NULL)
{
    ilkDugum = NULL;
    return;
}
```

İfadeleri ile bağlı listede tek bir düğüm olduğu durumda ilkDugum=NULL güncellemesi yapılmış ve fonksiyondan çıkılmıştır.

3.

```
struct cbagliListe* Ikinci = ilkDugum->sonraki;
Ikinci->onceki = NULL;
free(ilkDugum);
ilkDugum = Ikinci;
```

İfadeleri ile önce ***Ikinci*** isimli bir düğüm oluşturulmuş, ***Ikinci*** 'nin ***onceki*** işaretçisi ***NULL*** yapılmış, ***ilkDugum*** silinmiş ve ***ilkDugum = Ikinci*** güncellemesi yapılarak silme işlemi gerçekleştirilmiştir.

3.5.3. Çift Yönlü Bağlı Listelerde Aradan Eleman Silmek

C programlama dilinde yazılmış, Çift Yönlü Bağlı Listelerde listenin düğümleri arsından eleman silen fonksiyon aşağıda verilmiştir.

Not: Fonksiyon yazılırken, Çift Yönlü Bağlı Listede listenin düğümleri arasından eleman silmek için, hangi sıradaki elemanın silinmesi istendiği bilgisinin fonksiyona parametre olarak gönderileceği varsayılmıştır.

```
void aradanElemanSil(int sira){
    if(sira == 0){
        if(ilkDugum != NULL){
            bastanElemanSil();
            return;
        }
    }
    struct cbagliListe* temp = ilkDugum;
    int sayac = 0,kontrol=0;
    while(temp != NULL){
        if(sayac == sira){
            kontrol = 1;
            break;
        }
        temp = temp->sonraki;
        sayac++;
    }
    if(kontrol == 0){
        printf("\n Girilen Sıra Yok ");
        return;
    }
    if(temp->sonraki == NULL){
        sondanElemanSil();
        return;
    }
    struct cbagliListe* Onceki = temp->onceki;
    struct cbagliListe* Sonraki = temp->sonraki;
    Onceki->sonraki = Sonraki;
    Sonraki->onceki = Onceki;
    free(temp);
}
```

Çift Yönlü Bağlı Listelerde Aradan Eleman Silmek

aradanElemanSil() fonksiyonunda;

1.

```
if(sira == 0){
    if(ilkDugum != NULL){
        bastanElemanSil();
        return;
    }
}
```

İfadeleri ile silinecek düğümün sıra numarası sıfır (0), ***ilkDugum!=NULL*** ise ***bastanElemanSil()*** fonksiyonu ile listenin ilk düğümü silinir ve fonksiyondan çıkılır.

2.

```
struct cbagliListe* temp = ilkDugum;
int sayac = 0,kontrol=0;
while(temp != NULL){
    if(sayac == sira){
        kontrol = 1;
        break;
    }
    temp = temp->sonraki;
    sayac++;
}
```

Fonksiyonda yer alan yukarıdaki kod bloğunda önce ***temp*** adında bir düğüm oluşturuluyor ve ***temp=ilkDugum*** güncellemesi yapılıyor. Sonra traverse (dolaşma) işleminde kaçıncı sıradaki düğüme gelindiği bilgisini tutmak için ***sayac*** değişkeni ve ***sayac*** 'ın ***sira*** 'ya eşitlendiği durumu tespit etmek için ***kontrol*** değişkeni tanımlanıyor.

Daha sonra while bloğunda ***sayac == sira*** eşitliği sağlandığında ***kontrol*** değişkeninin değeri ***bir (1)*** yapılıp ***break*** ile döngüden çıkılıyor.

Sayac == sira eşitliğinin sağlanmadığı her adımda ise traverse (dolaşma işlemi devam ediyor ve sayac değişkeninin değeri bir (1) arttırılıyor.

3.

```
if(kontrol == 0){
    printf("\n Girilen Sıra Yok ");
    return;
}
```

while döngüsünün çalışması sona erdiğinde ***sayac == sıra*** eşitliği sağlanmamış, yani ***kontrol*** değişkenin değeri ***sıfır (0)*** kalmış ise, yukarıdaki ***kontrol == 0*** karşılaştırması "***True"*** değeri geri döndürür.

Bu, eleman silmek için giriş yapılan sırada düğümün olmadığı anlamına gelir ve o zaman kullanıcıya mesaj verilerek fonksiyonun çalışması sonlandırılır.

4.

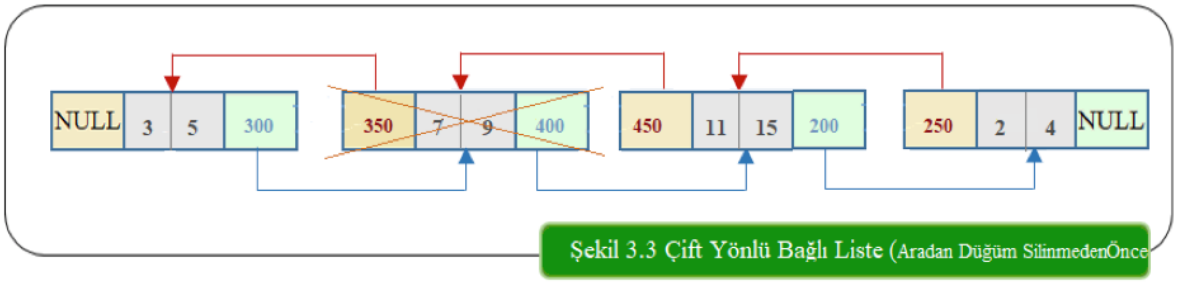
```
if(temp->sonraki == NULL){
    sondanElemanSil();
    return;
}
```

Yukarıdaki blok traverse (dolaşma) işlemi sırasında son düğüme gelinme halidir. Bu durumda son düğümün silinmesi gerekir. Bu işlem için ***sondanElemanSil()*** fonksiyonu çağırılmıştır. İşlem tamamlandıktan sonra fonksiyonun çalışması ***return*** ifadesi ile sonlandırılmıştır.

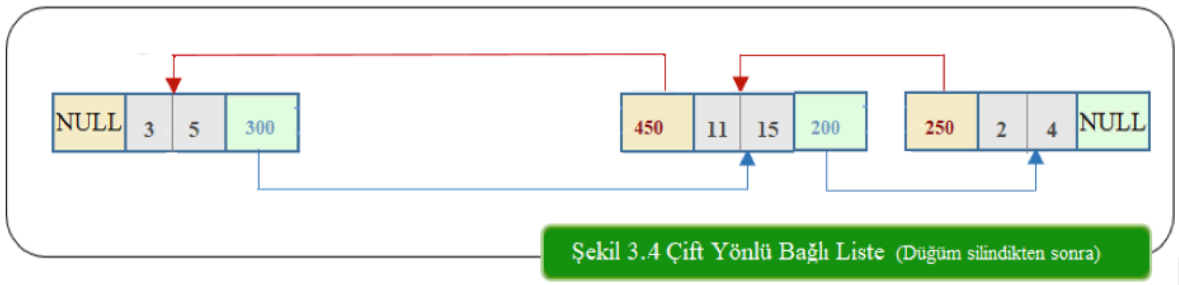
5.

```
struct cbagliListe* Onceki = temp->onceki;
struct cbagliListe* Sonraki = temp->sonraki;
Onceki->sonraki = Sonraki;
Sonraki->onceki = Onceki;
free(temp);
```

Yukarıdaki bloktaki ifadeler,
fonksiyonda ***bastanElemanSil()*** çağırılarak çalıştırılıp veya ***sondanElemanSil()*** çağırılarak çalıştırılıp veya eleman silme işleminin gerçekleştirilmeden fonksiyondan çıkıldığı üç farklı durumdan herhangi birinin gerçekleşmediği halde çalışacak ifadelerdir. ***aradanElemanSil()*** Fonksiyonunda, çalışma sırasının bu ifadelere geldiği noktada, aradaki silinecek düğüm ***temp*** 'te tutulmakta olup temp 'in önündeki ve ***temp***' ten sonra gelen düğümlerin işaretçileri güncellenerek ki yukarıdaki ifadelerle bu işlemler yapılmıştır. Fonksiyon aradaki düğümün silinmesini sağlar.



Şekil 3.3 Çift Yönlü Bağlı Liste (Aradan Düğüm SilinmedenÖnce



Şekil 3.4 Çift Yönlü Bağlı Liste (Düğüm silindikten sonra)

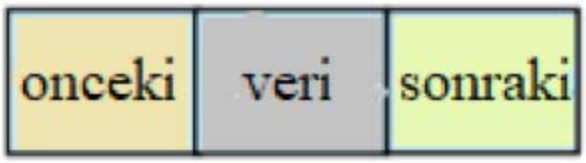
Bölüm Özeti

Çift Yönlü Bağlı listelerin tanımını yapabilecek,

Kendisinden önceki ve sonraki düğümlerin adreslerini saklamak üzere iki adet işaretçisi olan doğrusal bağlı listelere çift yönlü bağlı liste (**Doubly Linked List**) veya iki yönlü bağlı liste denir.

Çift Yönlü Bağlı Listelerin yapısını açıklayabilecek,

Çift yönlü bağlı listelerde bir düğümde bulunan iki işaretçiden önceki düğümün adresini gösteren işaretçi ***önceki (prev),*** sonraki düğümün adresini gösteren işaretçi ***sonraki (next)*** olarak isimlendirilebilir. Çift bağlı listede bir düğümün yapısı aşağıdaki gibidir. Hatırlanacağı gibi, Tek Yönlü Bağlı Listelerde bir düğümde sadece bir işaretçi (*sonraki*) bulunabiliyordu.



Çift Yönlü Bağlı Listelerde işaretçiler bakımından struct yapısı da Tek Yönlü Bağlı Listelerden farklıdır. Aşağıda Tek Yönlü Bağlı Listelerin struct yapısı gösterilmiştir.

```
struct cbagliListe{
    int veri1;
    int veri2;
    struct cbagliListe *onceki;//Bu işaretçi Tek Yönlü Bağlı Listelerde Yoktu
    struct cbagliListe *sonraki;
};
```

Çift Yönlü Bağlı Listelerin avantaj ve dezavantajlarını açıklayabilecek,

Çift Yönlü Bağlı listelerin avantajları, bu listeler üzerinde hem ileriye hem de geriye doğru hareket edilebilir, listeye eleman ekleme, listeden eleman silme gibi işlemlerinin daha kolay yapılabilir şeklinde sıralanabilir.

Çift Yönlü Bağlı listelerin avantajları ise, önceki (***prev***) işaretçisi de bulundurduğu için bellekte fazladan yer kaplar, her düğümün önceki (prev) ve sonraki (next) adında iki işaretçisi olduğu için liste işlemleri daha yavaştır, hata yapılma ihtimali yüksektir şeklinde sıralanabilir.

Çift Yönlü Bağlı Liste Oluşturabilecek, listeye eleman ekleme, eleman silme işlemlerini yapabilecek

Çift Yönlü Bağlı Liste oluşturma işlemi birçok bakımdan Tek Yönlü Bağlı Liste oluşturmaya benzemektedir. Tahmin edilebileceği gibi en önemli fark, Çift Yönlü Bağlı Listeye ilk düğüm eklenirken iki işaretçi tutacak şekilde düzenleme yapılması zorunluluğudur.

Çift Yönlü Bağlı Listelerde eleman ekleme ve eleman silme işlemleri önceki ve sonraki işaretçilerin güncellenmesi yoluyla yapılır. Burada unutulmaması gereken en önemli husus, silme işlemi yapılacaksa silinecek düğüm için bellekte ayrılmış olan adreslerin serbest bırakılması hususudur.