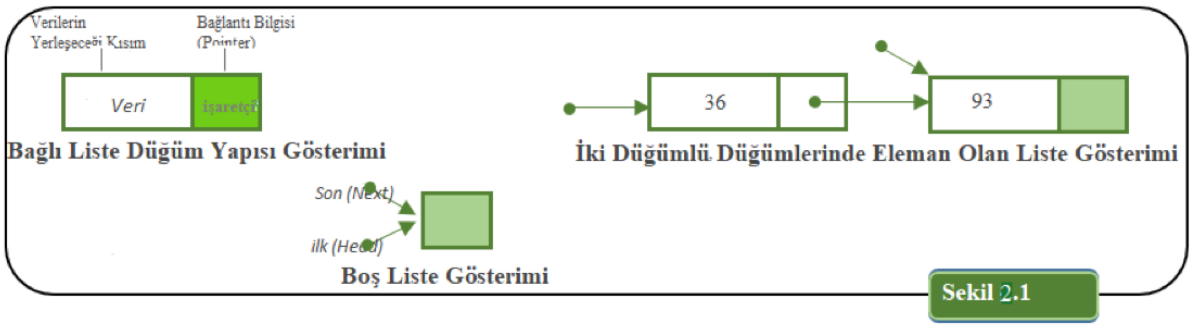
2. TEK YÖNLÜ BAĞLI LİSTELER

Giriş

Bağlı listeler dizilerden sonra en çok kullanılan veri yapısıdır ve doğrusal olma özellikleri ile dizilere benzerler. Fakat bağlı liste öğeleri (düğümler), dizilerden farklı olarak bilgisayar belleğinde birbirini takip eden adreslerde saklanmaz. Onun yerine bağlı listelerin her düğümünde, düğümdeki verinin yanında bir sonraki düğümü işaret eden işaretçi (pointer) bulunur.

Bağlı listelerde liste üzerindeki düğüm sayısı o listenin **uzunluk** bilgisine karşılık gelir. Liste üzerinde arama, sıralama vb. işlemler sırasında kullanılacak veri parçası bağlı listenin **anahtar** sözcüğüdür. Örneğin insan kaynakları uygulamasında liste veri yapısı kullanılmış ise T.C. kimlik numarası anahtar sözcük olabilir. Bağlı listelerde listede herhangi bir eleman bulunmuyorsa o listeye **boş liste** denir.

Şekil 2.1'de bağlı liste düğüm yapısı, boş liste ve iki düğümlü (liste uzunluğu=2) bağlı liste gösterilmiştir.



Bağlı listelerde bir sonraki düğüme işaret eden işaretçiler kullanılarak liste elemanları üzerinde kolaylıkla listeye eleman ekleme, silme, arama, liste elemanlarını yazdırma gibi işlemler gerçekleştirilebilir. Bağlı listeler genellikle tek yönlü bağlı listeler, çift yönlü bağlı listeler ve dairesel (çevrimsel) bağlı listeler olmak üzere üç başlık altında incelenir.

Tek yönlü bağlı listeler sadece düğümlerinden bir sonraki düğüme bağlantı kurularak oluşturulan listelerdir. Çift yönlü bağlı listeler ise bir

düğümden hem önceki düğüme hem de sonraki düğüme bağlantı kurularak oluşturulur. Tek Yönlü ve çift yönlü bağlı listelerde en son düğümün başka bir düğüme bağlantısı yoktur ve bu düğümdeki işaretçinin değeri NULL 'dır. Dairesel (çevrimsel) bağlı listeler oluşturulurken en son düğümden ilk düğüme bağlantı kurulur.

2.1. Diziler ve Bağlı Listeler Arasındaki Temel Farklılıklar

1. Bir dizi, benzer tipte veri elemanlarının bir koleksiyonunu içeren veri yapısıdır, buna karşılık bağlı listeler düğümler olarak bilinen sıralı olmayan bağlantılı elemanların bir koleksiyonunu içeren veri yapısıdır.
2. Dizinin elemanlarına ulaşmak için dizinin adı ve ulaşılmak istenilen dizi elemanının sırasını kullanmak yeterlidir. Buna karşılık, bağlantılı bir listede, bir düğümdeki veri üzerinde işlem yapabilmek için her zaman baştaki düğümden okumaya başlayıp okunması gereken verinin düğümüne kadar ilerlemek gerekir.
3. Bir dizideki bir elemana erişim hızlıdır, Bağlantılı listede bir düğüme ulaşmak doğrusal zaman alır, bu yüzden biraz daha yavaştır.
4. Dizilerde ekleme ve silme gibi işlemler çok zaman alır. Öte yandan Bağlantılı listelerde bu işlemlerin performansı dizilere göre daha hızlıdır.
5. Diziler sabit boyuttadır. Bağlı listelerin boyutu ise çalışma sırasında genişletip daraltılabilir.
6. Bir dizi için bellek tahsisi derleme sırasında yapılırken, bağlı listeler için bellek tahsisi yürütme veya çalıştırma sırasında yapılır.
7. Dizilerde öğeler (elemanlar) ardışık olarak saklanırken, bağlı listelerde öğeler rastgele saklanır.
8. Diziler bellekte ardışık adresleri kullandığı için dizi üzerinde işaretçi bilgisine ihtiyaç duyulmaz, Buna karşılık, bağlı listelerde, her düğümde verinin yanında bir sonraki düğüme işaret edecek işaretçi için yer ayrılması gerekir.

2.2. Bağlı Listelerin Dizilere Göre Avantaj ve Dezavantajları

Bağlı Listelerin Dizilere göre iki önemli avantajı vardır. Bu avantajlardan birincisi; bağlı listelerin boyutları dizilerin aksine dinamiktir. Yani öğe sayısı derleme zamanından sonra da ihtiyaca göre arttırılıp eksiltilebilir.

Bu özelliğinden dolayı bağlı listelerin, gereksiz bellek kullanımının önüne geçilmesini sağladığını söyleyebiliriz.

Bağlı Listelerin ikinci önemli avantajının ise bağlı listeye yeni öğe eklemenin veya bağlı listeden eleman silmenin, dizilere eleman eklemek ve dizilerden eleman silmeye göre daha hızlı, daha kolay ve daha düşük maliyetle yapılabilmesidir. Biz daha önceki bölümlerden dizilere yeni bir eleman eklemek/silmek için diğer elemanların tamamının yer değiştirmesine ihtiyaç duyulduğunu biliyoruz. Bağlı listelere eleman ekleme/silme konusunu işlemeye sıra geldiğinde bu işlemin bağlı listelerde çok daha kolay yapıldığını göreceğiz.

Bağlı listelerin dizilere göre yukarıda sayılan avantajları yanında dezavantajları da vardır. Örneğin, bağlı listelerde bir düğüme ulaşmak için rastgele bir erişime izin verilmez, düğüme ulaşabilmek için her zaman listenin ilk düğümünden okunacak verinin bulunduğu düğüme kadar okuma işleminin sürdürülmesi gerekir. Bilindiği gibi dizilerde dizi elemanlarına rastgele erişilebilir.

Bağlı listelerin dizilere göre ikinci önemli dezavantajı ise listenin her düğümünde işaretçi için fazladan bellek alanının kullanılması zorunluluğudur. Diziler zaten sıralı olduğu için diziye ait elemanlara ulaşabilmek için ayrıca bir işaretçiye gerek yoktur.

2.3. Bağlı Listeler Üzerinde Yapılabilecek İşlemler

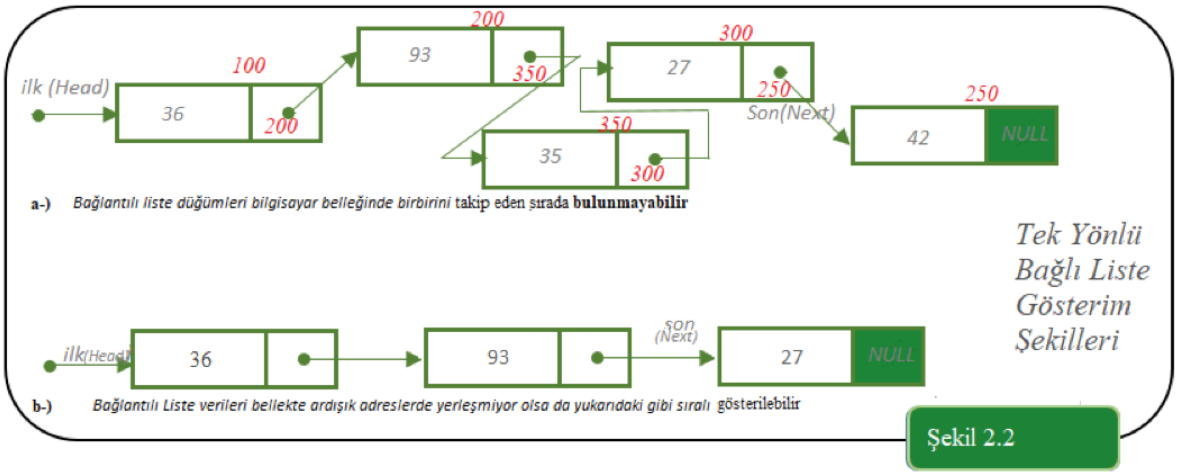
1. Liste oluşturmak
2. Listedeki verileri yazdırmak
3. Listeye eleman eklemek
4. Listeden eleman silmek
5. Liste elemanlarını saydırmak

Biz bu bölümde tek yönlü bağlı listeleri ele alacağız. Bu bölümden sonra üçüncü bölümde çift bağlı listeler, dördüncü bölümde de dairesel (çevrimsel) bağlı listeler anlatılacaktır.

2.4. Tek Yönlü Bağlı Listeler

Tek yönlü bağlı liste, bağlı listelerin en temel, en basit ve anlaşılması en kolay örneğidir. Tek yönlü bağlantılı listeler, bir elemanın kendinden sonraki elemanın adresini de bulunduran düğümlerden oluşan liste yapısıdır. Bu listenin her bir düğümü, veri alanı ve sonraki düğümün referansını içeren bir adres alanından (pointer) oluşur. Tek Yönlü Bağlı Listelerin düğümleri birden fazla veri alanı içerebilir, buna karşılık bir sonraki düğüme işaret eden tek bir adres alanı içerebilir. Tek yönlü bağlı listelerin düğümlerinde sadece kendisinden sonra gelen düğümü işaret eden işaretçi bulunur. Bu veri yapısında düğümde kendisinden önce gelen düğümü işaret eden işaretçi yoktur. Bu nedenle bu listelere tek yönlü bağlı listeler denilmiştir. Şekil 2.2'de tek yönlü bağlı listelerin mantıksal yapısı gösterilmiştir.

Şekil 2.2'den de görülebileceği gibi, tekyönlü bağlı listelerin düğümleri, dizilerin elemanlarının belleğe yerleşme biçiminden farklı olarak, birbirini takip eden adreslere yerleşmek yerine bellekte birbirinden farklı noktalardaki adreslere yerleşebilir. Listenin düğümleri arasındaki ilişki, düğüm içerisinde yer alan işaretçinin (pointer) bir sonraki düğümü işaret etmesiyle kurulur.



Şekil 2.2

Şekil 2.2.a veya şekil 2.2.b'yi dikkatli bir şekilde incelediğinizde tek yönlü bağlı listenin, birçok düğümün bir zincir oluşturmak için birbirine bağlandığı durumda oluştuğunu göreceksiniz. Tek yönlü bağlı listelerde her düğüm, sıradaki bir sonraki düğümü işaret eder ve ilk düğüm her zaman düğümler (nodes) arasında geçiş yapmak için bir referans olarak kullanılır.

Tek yönlü bağlı listenin ilk düğümünün işaretçisi HEAD olarak adlandırılır. Bu yapıdaki bir listeye ulaşabilmek için listenin ilk elemanını gösteren işaretçiye (HEAD) ihtiyaç vardır. Tek yönlü bağlı listede herhangi bir düğüm bulunmuyorsa bu listeye boş liste (null list) adı verilir. Eğer HEAD=NULL ise liste boştur.

Tek yönlü bağlı listelerin sonuna gelinip gelinmediğini kontrol etmek için listenin en son elemanının sonraki kısmını gösteren bir özel işaretçiye daha ihtiyaç vardır. Bu işaretçinin değeri her zaman NULL 'dır.

Şekil 2.2.a'da konuyu açıklayabilmek için düğümlerdeki işaretçilere sembolik adres değerleri atanmıştır. Buna göre ilk düğümün adresi 100, ikinci düğümün adresi 200, üçüncü düğümün adresi 350, dördüncü düğümün adresi 300, beşinci düğümün adresi 250'dir. Beşinci düğümün listenin son düğümü olduğuna ve kendisinden sonra işaret edeceği herhangi bir düğüm olmadığına, işaretçisinin değerinin NULL olduğuna dikkat ediniz.

Tek yönlü bağlı listelerin avantajlarını aşağıdaki şekilde sıralayabiliriz:

1. Uygulanması en kolay veri yapısıdır.
2. Eleman ekleme ve silme işlemleri kolaylıkla yapılabilir.
3. Düğümlerin eklenmesi ve silinmesi sırasında tüm düğümlerin hareket etmesi gerekmez.
4. İki yönlü bağlı listelere veya dairesel dairesel bağlı listelere kıyasla daha az bellek gerektirir.
5. Yürütülmesi sırasında gerektiğinde ihtiyaç duyulan bellek kolayca tahsis edebilir veya serbest bırakabilir.
6. Bir yönde hareket etmek gerektiği durumlarda uygulanacak en verimli veri yapısıdır.

Tek yönlü bağlı listelerin dezavantajlarını aşağıdaki şekilde sıralayabiliriz:

1. Bir diziye göre daha fazla bellek kullanır.

2. Düğümler sıralı olarak depolanmadığından, listenin her öğesine erişmek için daha fazla zaman gerekir.
3. Tek Yönlü bağlı listelerde tek yönde hareket edilebilir, ters yönde hareket etmek mümkün değildir.

Tek Yönlü Bağlı Listelerin Genel Yapısı

Bağlı listeler, genel olarak dizi veri yapısı üzerinde veya her bir düğüm (node) için bellekten **malloc()** fonksiyonu ile dinamik şekilde bellek alanı ayırarak ve bunları işaretçi değişken alanları üzerinden birbirine bağlayarak tutulabilir.

Biz bu bölümde bağlı listeleri anlatırken bellekten dinamik şekilde bellek alanı ayrılan ve bu alanların işaretçi değişken üzerinden birbirine bağlandığı yöntemi kullanacağız. Program 2.1'de bağlı listenin veri yapısını tanımlayan C kodu verilmiştir.

```
struct dugum {
    int veri;                       // Data
    struct dugum *sonrakiDugum;     // Adres
};
```

Program 2.1

Yukarıdaki kod parçası incelendiğinde, kod ile C programlama dilinde dugum isimli bir yapı tanımlandığı görülmektedir. Böyle bir yapıda istenilen türde istenildiği kadar eleman tanımlanabilir. Burada konunun anlaşılmasını zorlaştırmamak için yalnızca tamsayı türünde **veri** isimli tek bir eleman tanımlanmıştır. Tanımlanan yapıda **veri** isimli elemanın tanımından sonra gelen ifadede ile **sonrakiDugum** isimli bir işaretçi (pointer) türünde bir yapı değişkeninin tanımlandığı görülmektedir. **sonrakiDugum** isimli bu işaretçi bir sonraki düğümü işaret edecektir.

2.5. Tek Yönlü Bağlı Liste Oluşturmak ve Listeye Eleman Eklemek

Bağlı listelere her zaman eleman eklenebilir veya listeden eleman çıkartılabilir. Bu nedenle bu listeler dinamik bir veri yapısına sahiptir.

Bağlı listelere eleman eklemek dizilere eleman eklemekten görece daha kolaydır. Çünkü dizilere eleman eklemek gerektiğinde bilindiği gibi, dizi elemanları toplu olarak hareket ettirme zorunluluğu vardır. Bağlı listelere eleman eklemek için ise sadece bazı işaretçileri güncellemek yeterli olacaktır.

Program 2.2 'de verilen C programında önce **main()** fonksiyonu içerisinde dokuzuncu satırda ***ilkDugum** tanımlanıyor ve onuncu satırda bu düğüm için **malloc()** fonksiyonu kullanılarak bilgisayar belleğinde **ilkDugum** için yer ayrılıyor. On birinci satırda, **ilkDugum** 'ün veri alanına (sayi değişkeni) 23 değeri atanıyor. On ikinci satırda işaretçi değeri **NULL** yapılıyor. On üçüncü satırda bilgisayar belleğinde ikinci düğüm için yer ayrılıyor ve daha öce birinci düğüm için yapılan işlemler ikinci düğüm için tekrarlanıyor. On yedinci – yirmi ikinci satırlar arasında daha önce oluşturulan ve düğümlerine değer atanan bağlı listenin değerleri ekrana yazdırılıyor.

Tek Yönlü Bağlı Liste Oluşturan ve Listede bulunan iki düğüme eleman ekleyen C kodu

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  struct bListe
4  {
5      int sayi;
6      struct bListe *sonrakiDugum;
7  };
8  int main() {
9      struct bListe *ilkDugum;
10     ilkDugum=(struct bListe*)malloc(sizeof(struct bListe));
11     ilkDugum->sayi=23;
12     ilkDugum->sonrakiDugum=NULL;
13     ilkDugum->sonrakiDugum=(struct bListe*)malloc(sizeof(struct bListe));
14     ilkDugum->sonrakiDugum->sayi=36;
15     ilkDugum->sonrakiDugum->sonrakiDugum=NULL;
16
17     struct bListe* temp=ilkDugum;
18         while(temp!=NULL)
19     {
20         printf("%d ",temp->sayi);
21         temp=temp->sonrakiDugum;
22     }
23     return 0;
24 }
```

```
23 36
--------------------------------
```

Program 2.2'nin Çıktısı

//Tek Yönlü Bağlı Liste Elemanlarının Yazdırılması

Program 2.2

Program 2.2'de verilen C programında düğümleri oluşturmak için işlemlerin tekrar tekrar yapıldığına dikkat ediniz. Bunun yerine program 2.3'te verilen C programında, bu işlemler için bir fonksiyon (**dugumOlustur()**) yazılmış ve yazılan bu fonksiyon ile tekrar eden işlemler için aynı ifadelerin yazılmasının önüne geçilmiştir.

Program 2.3'te önce düğümleri tanımlayan bir yapı oluşturulmuştur. Yapı içerisinde, düğümdeki verilerin saklanması için tamsayı türünde bir öğe

(*int veri*) ve bir sonraki düğümleri işaret etmesi için struct bagliListe tipinde bir işaretçi (*struct bagliListe *sonrakiDugum;*) tanımlanmıştır. Bağlı listelerde bir düğüm içerisinde aynı veya farklı tiplerde birden fazla veri saklanabilir. Fakat düğümün kendisinden sonraki düğümü işaret etmesi için bir tek işaretçi yeterli olur. Aşağıda program 2.3'te düğümleri tanımlayan yapı gösterilmiştir.

Programda, düğümleri tanımlayan yapı oluşturulduktan sonra bağlı listenin ilk düğümünün tutulabilmesi için (s*truct bagliListe* ilkDugum=NULL;*) ifadesi yazılmıştır. İlk düğüm tanımlandığında, listede düğümün işaret edeceği herhangi başka bir düğüm olmadığı için, düğüme *NULL* değeri atandığına dikkate ediniz. Bazı geliştirme ortamlarında bu atama varsayılan olarak gerçekleştiğinden ilk düğüm oluşturulurken NULL değerinin atanması zorunlu değildir.

Bu ilk düğümün tanımlanması işleminden sonra sıra, bellekte listenin düğümleri için yer ayıracak (listeyi oluşturacak) fonksiyonun (dugumOlustur()) yazılmasına gelmiştir. Fonksiyonda önce bir düğüm (*yeniDugum*) oluşturulması gerekmektedir. Düğüm oluşturmak için *struct bagliListe * yeniDugum = (struct bagliListe*) malloc(sizeof(struct bagliListe));* ifadesi kullanılır. Bu ifadedeki malloc() fonksiyonu bellekte düğüme yer ayrılması için, sizeof() fonksiyonu ise bellekte ayrılacak yerin büyüklüğünü belirlemek için kullanılmıştır. Ayrıca bu ifadenin önüne Casting işlemi için (*struct bagliListe**) ifadesi yazılmıştır ki bu işlem tip dönüşümü yapılmasını sağlayarak tip uyuşmazlığının önüne geçilmesini sağlar.

yeniDugum için bellekte yer ayrıldıktan sonra fonksiyonun sayi parametresinin değeri yeniDugum için bellekte ayrılan sayi alanına *yeniDugum->veri=veri;* ifadesi ile atanmıştır.

Daha sonra *yeniDugum->sonrakiDugum=NULL;* ifadesi ile bağlı listeye en son eklenen düğümünün işaretçi değişkeninin değeri NULL yapılır ve *return yeniDugum;* ifadesi ile *yeniDugum* 'un geri dönmesi sağlanır. Aşağıda program 2.3'te yazılan dugumOlustur() fonksiyonu verilmiştir;

```
struct bagliListe* dugumOlustur(int veri){
    struct bagliListe* yeniDugum =
    (struct bagliListe*)malloc(sizeof(struct bagliListe));
    yeniDugum->veri=veri;
    yeniDugum->sonrakiDugum=NULL;
    return yeniDugum;
}
```

dugumOlustur() fonksiyonun main() içerisinden her çağırıldığında bellekte, bağlı liste için bir düğüm oluşmasının sağlanacağına dikkat ediniz.

Program 2.3'te dugumOlustur() fonksiyonundan sonra, düğümdeki veri ve işaretçileri yazdırmak amacıyla dugumYazdir() isimli bir fonksiyon daha yazılmıştır. dugumYazdir() fonksiyonu 2.5.1'de verilmiştir.

Yukarıda açıklanan ve düğümleri tanımlayan yapı (***bagliListe***), düğümlerin oluşturulabilmesi için ***dugumOlustur()*** fonksiyonu, düğümleri yazdırmak için dugumYazdir() fonksiyonları hazırlandıktan sonra main fonksiyonu içerisinde önce;

ilkDugum=dugumOlustur(90);

İfadesi ile ilk düğüme doksan (90) değeri atanmıştır.

Sonra aşağıdaki ifade çoğaltılarak listeye yeni düğümler eklenmiş ve düğümlerdeki veri değişkenine değerler atanmıştır;

struct bagliListe *ikinciDugum=dugumOlustur(83);

Düğümlerin eklenmesi işlemi tamamlandıktan sonra düğümlerin birbiri ile bağlanması için aşağıdaki ifade yazılmıştır;

ilkDugum->sonrakiDugum=ikinciDugum;

dugumOlustur() fonksiyonu içerisindeki **yeniDugum->sonrakiDugum=NULL;** ifadesinden dolayı son düğümün işaretçisi her zaman NULL olacağından main fonksiyonu içerisinde en son düğüme NULL atanmasına gerek kalmamıştır.

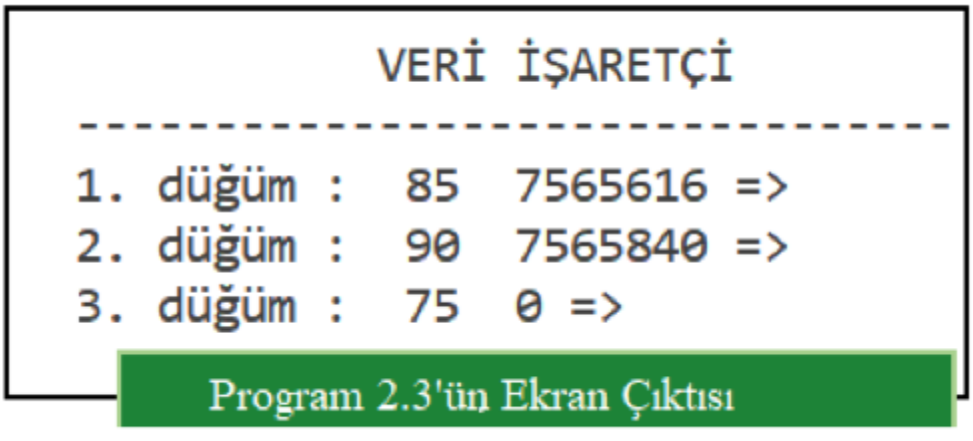
Yukarıdaki işlemlerden sonra program 2.3 tamamlanmış ve program ve programın çıktısı aşağıda verilmiştir.

Tek Yönlü Bağlı Liste Oluşturan ve Listeye üç düğüm ekleyen C programı

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
struct bagliListe{
 int veri;
 struct  bagliListe *sonrakiDugum;
};
struct bagliListe* ilkDugum=NULL;
struct bagliListe* dugumClustur(int veri){
    struct bagliListe* yeniDugum =
    (struct bagliListe*)malloc(sizeof(struct bagliListe));
    yeniDugum->veri=veri;
    yeniDugum->sonrakiDugum=NULL;
    return yeniDugum;
}
void dugumYazdir(){
    int n=0;
    struct bagliListe* temp = ilkDugum;
    printf("\n            VERİ İŞARETÇİ\n");
    printf("--------------------------------\n");
    while(temp != NULL){
        n++;
        printf("%d. düğüm : ",n);
        printf(" %d  ", temp->veri);
        printf("%d =>\n",temp->sonrakiDugum);
        temp = temp->sonrakiDugum;
    }
    printf("\n");
    }

int main() {
    setlocale(LC_ALL,"Turkish");
    ilkDugum=dugumOlustur(85);
    struct bagliListe* ikinciDugum=dugumOlustur(90);
    struct bagliListe* ucuncuDugum=dugumOlustur(75);
    ilkDugum->sonrakiDugum=ikinciDugum;
    ikinciDugum->sonrakiDugum=ucuncuDugum;
    dugumYazdir();
    return 0;
}
```

Program 2.3



Program 2.3'ün Ekran Çıktısı

2.5.1. Bağlı Listenin Elemanlarını Ekrana Yazdıran Fonksiyon

```
void dugumYazdir(){
    int n=0;
    struct bagliListe* temp = ilkDugum;// ilkDugum oluşturulan temp düğümüne atanmıştır.
    printf("\n            VERİ İŞARETÇİ\n");
    printf("--------------------------------\n");
    while(temp != NULL){  // while(temp!=NULL) döngüsü temp NULL olana kadar döner.
        n++;
        printf("%d. düğüm : ",n);
        printf(" %d  ", temp->veri);// temp 'in veri değişkeninin değeri ekrana yazdırılır.
        printf("%d =>\n",temp->sonrakiDugum);// işaretçi ekrana yazdırılır.
        temp = temp->sonrakiDugum;// Traversal (dolaşma) işlemi ile düğümler baştan
                                  // sona dolaşılır.
    }
    printf("\n");
    }
```

2.5.2. Tek Yönlü Bağlı Listenin Sonuna Eleman Eklemek

Tek yönlü bağlı listenin sonuna eleman eklenmesini sağlayan fonksiyon aşağıda verilmiştir.

```
void SonaEkle(int veri1,int veri2){
    struct bagliListe *eklenecekDugum = dugumOlustur(veri1,veri2);
    if(ilkDugum == NULL){
        ilkDugum = eklenecekDugum;
    }
    else{
        struct bagliListe* temp = ilkDugum;
        while(temp->sonrakiDugum != NULL){
            temp = temp->sonrakiDugum;
        }
        temp->sonrakiDugum = eklenecekDugum;
    }
}
```

Tek yönlü bağlı listenin sonuna eleman eklemek için önce **void** türünde ***SonaEkle(int veri, int veri2)*** şeklinde bir fonksiyon tanımlandı ve fonksiyon gövdesinde ilk ifade olarak ***struct bagliListe *eklenecekDugum = dugumOlustur(veri1,veri2);*** yazılarak eklenecek düğüm oluşturuldu. Bundan sonra karşımıza değerlendirmemiz gereken farklı iki durum çıkıyor. Birinci durum tek yönlü bağlı listede herhangi bir düğümün bulunmadığı durumdur.

```
if(ilkDugum == NULL){
    ilkDugum = eklenecekDugum;
}
```

Programı ilk çalıştırdığımızda tek yönlü bağlı listede herhangi bir düğüm olmayacaktır ve bu durumda fonksiyon gövdesinde yer alan yukarıdaki ifadeler çalışacaktır. Bu ifadelerin çalışması sonucunda daha önceden oluşturulmuş olan **eklenecekDugum** bağlı listenin **ilkDugum**'ü yapılacaktır.

Program çalıştırılmaya devam ettiğinde, yani listede **ilkDugum** oluştuktan sonra değerlendirilmesi gereken ikinci durum ortaya çıkar ve bu durumda **if** bloğunun **else** kısmındaki ifadeler çalışır.

```
else{
    struct bagliListe* temp = ilkDugum;
    while(temp->sonrakiDugum != NULL){
        temp = temp->sonrakiDugum;
    }
    temp->sonrakiDugum = eklenecekDugum;
```

Yukarıdaki ifadelerin çalışması sonucunda listedeki düğümlerin tamamı traversal işlemi ile sıra ile dolaşılır ve sona gelindiğinde (işaretçisi **NULL** olan düğüme gelindiğinde) while bloğunda çıkılır ve ***temp->sonrakiDugum = eklenecekDugum;*** ifadesi ile tek yönle bağlı listenin sonuna yeni bir düğüm eklenmiş olur.

Program 2.4 te, tek yönlü bağlı listenin sonuna yeni bir düğüm eklenmesini sağlayan C programının kaynak kodunun tamamı verilmiştir. Program 2.4 ile oluşturulan her düğüme iki veri girilebildiğine ve programın bu yönüyle bu bölümde verilen diğer programlardan farklı olduğuna dikkat ediniz.

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
struct bagliListe{
    int veri1;
    int veri2;
 struct  bagliListe *sonrakiDugum;
};
struct bagliListe* ilkDugum=NULL;
struct bagliListe* dugumOlustur(int veri1,int veri2){
    struct bagliListe* yeniDugum =
    (struct bagliListe*)malloc(sizeof(struct bagliListe));
    yeniDugum->veri1=veri1;
    yeniDugum->veri2=veri2;
    yeniDugum->sonrakiDugum=NULL;
    return yeniDugum;
}
void SonaEkle(int veri1,int veri2){
    struct bagliListe *eklenecekDugum =
    dugumOlustur(veri1,veri2);
    if(ilkDugum == NULL){
        ilkDugum = eklenecekDugum;
    }
    else{
        struct bagliListe* temp = ilkDugum;
        while(temp->sonrakiDugum != NULL){
            temp = temp->sonrakiDugum;
        }
        temp->sonrakiDugum = eklenecekDugum;
    }
}
```

```c
void dugumYazdir(){
    int n=0;
    struct bagliListe* temp = ilkDugum;
    printf("\n               VERİ          İŞARETÇİ\n");
    printf("-------------------------------------\n");
    while(temp != NULL){
        n++;
        printf("%d. düğüm : ",n);
        printf("( %d    ", temp->veri1);
        printf(" %d )=>  ", temp->veri2);
        printf("%d =>\n",temp->sonrakiDugum);
        temp = temp->sonrakiDugum;
    }
    printf("\n");
    }
int main() {
    int secim, veri1,veri2;
    setlocale(LC_ALL,"Turkish");
    while(1){
        printf("\n 1- Düğüm Sonuna Eleman Ekleme");
        printf("\n");
        printf("\n Seçiminiz : ");
        scanf("%d", &secim);
        switch(secim){
        case 1:
            printf("\n Düğümün ilk    elemanı : ");
            scanf("%d", &veri1);
            printf("\n Düğümün ikinci elemanı : ");
            scanf("%d", &veri2);
            SonaEkle(veri1,veri2);
            dugumYazdir();
            break; } }
    return 0;
}
```

Program 2.4

```
 1- Dugum Sonuna Eleman Ekleme

 Seçiminiz : 1

 Dügümün ilk    elemanı : 24

 Dügümün ikinci elemanı : 48
```

```
                VERİ          İŞARETÇİ
-------------------------------------
1. düğüm : ( 24    48 )=>  0 =>
```

```
 1- Düğüm Sonuna Eleman Ekleme

 Seçiminiz : 1

 Dügümün ilk    elemanı : 51

 Dügümün ikinci elemanı : 72
```

```
                VERİ          İŞARETÇİ
-------------------------------------
1. düğüm : ( 24    48 )=>  10907984 =>
2. düğüm : ( 51    72 )=>  0 =>
```

```
 1- Düğüm Sonuna Eleman Ekleme

 Seçiminiz : 1

 Dügümün ilk    elemanı : 84

 Dügümün ikinci elemanı : 96
```

```
                VERİ          İŞARETÇİ
-------------------------------------
1. düğüm : ( 24    48 )=>  10907984 =>
2. düğüm : ( 51    72 )=>  10908688 =>
3. düğüm : ( 84    96 )=>  0 =>
```

Progran 2.4'ün Ekran Görüntüsü

2.5.3. Tek Yönlü Bağlı Listenin Başına Eleman Eklemek

Tek yönlü bağlı listenin başına eleman ekleyen fonksiyon aşağıda verilmiştir.

```
void basaEkle(int veri1,int veri2)
{
    struct bagliListe* eklenecekDugum = dugumOlustur(veri1,veri2);
    if(ilkDugum == NULL)
    {
        ilkDugum = eklenecekDugum;
        return;
    }
    struct bagliListe* oncekiBirinci = ilkDugum;
    ilkDugum = eklenecekDugum;
    ilkDugum->sonrakiDugum = oncekiBirinci;
}
```

Yukarıda kaynak kodu verilen **basaEkle()** fonksiyonunun ilk ifadesinde (*struct bagliListe* eklenecekDugum=dugumOlustur(veri1,veri2);*) dugumOlustur() fonksiyonu çağırılarak eklenecekDugum oluşturulmuştur. Daha sonra listedeki ilk düğümün değerinin *NULL* olup olmadığı kontrol edilerek *NULL* olması durumunda eklenecekDugum ilkDugum'e atanmış ve fonksiyondan çıkılmıştır.

```
if(ilkDugum == NULL)
{
    ilkDugum = eklenecekDugum;
    return;
}
```

ilkDugum 'ün NULL olmaması durumunda ilk düğümü tutabilmek için oncekiBirinci isimli bir düğüm oluşturulmuş ve ilkDugum oncekiBirinci düğümüne atanmıştır. Daha sonra ilkDugum 'ün sonrakiDugum isimli işaretçisi oncekiBirinci yapılmıştır.

```
struct bagliListe* oncekiBirinci = ilkDugum;
ilkDugum = eklenecekDugum;
ilkDugum->sonrakiDugum = oncekiBirinci;
```

Bu şekilde eklenen düğüm başa gelmiştir.

2.5.4. Tek Yönlü Bağlı Listelerde Araya Eleman Eklemek

Tek yönlü bağlı listede araya eleman ekleyen fonksiyon aşağıda verilmiştir.

```
void arayaEkle(int veri1, int veri2, int esira){
    struct bagliListe* arayaEklenecek = dugumOlustur(veri1,veri2);
    if(esira == 0){
        basaEkle(veri1,veri2);
        return;
    }
    struct bagliListe* temp = ilkDugum;
    int sayac = 0;
    while(temp != NULL){
        if(sayac== esira-1)
            break;
        temp = temp->sonrakiDugum;
        sayac++;
    }
    if(sayac + 1 != esira){
        printf("\n Boyle bir pozisyon yok...");
        return;
    }
    if(temp->sonrakiDugum == NULL){
        SonaEkle(veri1,veri2);
        return;
    }
    struct bagliListe* sonrakiGecici = temp->sonrakiDugum;
    temp->sonrakiDugum = arayaEklenecek;
    arayaEklenecek->sonrakiDugum = sonrakiGecici;
}
```

Tek yönlü bağlı listelerde araya eleman ekleyen fonksiyon tamsayı tipinde üç parametre almıştır. Bu parametrelerden ikisi düğümde saklanan verilere karşılık gelirken üçüncüsü düğümün listede hangi sıraya ekleneceğini belirlemek için kullanılacaktır. Fonksiyonun gövdesindeki ilk ifade araya eklenecek düğümü oluşturmak için yazılmıştır;

struct bagliListe arayaEklenecek = dugumOlustur(veri1, veri2);*

Araya ekleme işleminde Eklenecek düğümün sırası sıfır (0) olarak belirlenmiş ise arayaEklenecek düğüm aslında listenin başına eklenecektir. Bu işlem için fonksiyon gövdesinde bir if bloğu oluşturulmuş ve blok içerisinden daha önce yazılan basaEkle() fonksiyonu çağırılmış ve düğümün listenin başına eklenmesi sağlanmıştır. Bu işlem için arayaEkle() fonksiyonu gövdesinde yazılan ifadeler aşağıda verilmiştir:

```
if(esira == 0){
    basaEkle(veril,veri2);
    return;
}
```

Düğüm listenin başına eklendikten ***return;*** ifadesi ile fonksiyonun sonlandırıldığına dikkat ediniz.

Bu kontrolden sonra tek yönlü bağlı liste üzerinde traversal işlemi uygulayarak düğümün araya ekleneceği pozisyonu bulmak gerekir. Bunun için fonksiyon gövdesinde yazılan ifadeler aşağıda gösterilmiştir:

```
struct bagliListe* temp = ilkDugum;
int sayac = 0;
while(temp != NULL){
    if(sayac== esira-1)
        break;
    temp = temp->sonrakiDugum;
    sayac++;
}
```

Traversal işleminden sonra fonksiyon gövdesinde bazı kontroller yapılmıştır. Örneğin traversal (dolaşma) işlemi sonunda (sayac + 1 != esira) ise liste üzerinde düğümün eklenebileceği ara pozisyon yoktur. Böyle bir durumda kullanıcıya mesaj verilip return ifadesi ile fonksiyonda çıkılır. Bu kontrol için fonksiyon gövdesinde yazılan satırlar aşağıda verilmiştir:

```
if(sayac + 1 != esira){
    printf("\n Boyle bir pozisyon yok...");
    return;
}
```

arayaEkle() fonksiyonu içerisinde yapılması gereken diğer bir kontrol (*temp->sonrakiDugum == NULL*) olduğu durumdur. Traversal (dolaşma) işlemi sonunda böyle bir koşul oluşmuş ise eklenecek düğüm listedeki

düğümlerin arasına değil listenin sonuna eklenecektir. Bu işlem içinde **SonaEkle(veri1,veri2)** fonksiyonu kullanılmıştır. arayaEkle() fonksiyonu içerisinde SonaEkle fonksiyonunun kullanılma şekli aşağıda gösterilmiştir.

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

arayaEkle() fonksiyonunda daha sonra araya eklenecek düğümün yerinde bulunan düğüm (*temp->sonrakiDugum*) yeni oluşturulan sonrakiGecici düğümüne atanmıştır. Çünkü o düğümün yerine arayaEklenecek düğümü gelmesi gerekmektedir. Bu işlem bir sonraki ifade ile (*temp->sonrakiDugum = arayaEklenecek;*) gerçekleştirilmiştir. Fonksiyonun son satırında (*arayaEklenecek->sonrakiDugum = sonrakiGecici;*) arayaEklenecek değerinin işaretçisi güncellenmiş ve tek yönlü bağlı listelerde araya eleman ekleme işlemi tamamlanmıştır.

Yukarıda açıklanan ve arayaEkle() fonksiyonunda asıl işlemleri (araya eleman ekleme işlemi) yapan satırlar aşağıda verilmiştir.

```
struct bagliListe* sonrakiGecici = temp->sonrakiDugum;
temp->sonrakiDugum = arayaEklenecek;
arayaEklenecek->sonrakiDugum = sonrakiGecici;
```

2.6. Tek Yönlü Bağlı Listelerde Eleman Silmek

2.6.1. Tek Yönlü Bağlı Listelerde Sondan Eleman Silmek

Tek yönlü bağlı listede listenin sonundan eleman silen fonksiyon aşağıda verilmiştir.

```
void sondanSil()
{
    struct bagliListe* temp =ilkDugum;
        while(temp->sonrakiDugum->sonrakiDugum!=NULL)
          temp=temp->sonrakiDugum;
        free(temp->sonrakiDugum);
        temp->sonrakiDugum=NULL;
}
```

Tek yönlü bağlı listelerde listenin sonundan eleman silmek için öncelikle listenin sondan bir önceki elemanını bulmamız gerekir. Ayrıca, bu yapacağımız silme işleminin dizilerdeki eleman silme işlemine benzemediğine, silme işleminin, silinecek düğümün hafızadan silinerek gerçekleştirileceğine dikkat ediniz.

Listenin sondan bir önceki elemanını bulmak için yine traversal (dolaşma) işlemi yapmamız gerekiyor ve sondan bir önceki düğümü bulabilmek için while döngüsünü (*temp->sonraki->sonraki!=NULL*) koşulu sağlandığı sürece dönecek. Döngüden çıkıldığında, liste üzerinde gelinen noktadan sonra gelen düğümün silinmesi işlemi yapılmalıdır. Silme işlemi için C programlama dilinde *free()* komutu kullanılır ve ifade *free(temp->sonraki);* şeklinde yazılır. Silme işleminden sonra temp değeri listenin son elemanı konumuna gelmiştir ve bu düğümün işaretçisi **NULL** olmalıdır. Bu *temp->sonraki=NULL;* ifadesi ile gerçekleştirilmiştir.

2.6.2. Tek Yönlü Bağlı Listelerde Baştan Eleman Silmek

Tek yönlü bağlı listede listenin başından eleman silen fonksiyon aşağıda verilmiştir.

```
void bastanSil(){
    if(ilkDugum == NULL){
        printf("\n Listede Silinecek Herhangi Bir eleman Yok  \n ");
        return;
        }
    if(ilkDugum->sonrakiDugum == NULL){
        ilkDugum = NULL;
        return;
    }
    struct bagliListe* ikinciSiradaki = ilkDugum->sonrakiDugum;
        free(ilkDugum);
        ilkDugum = ikinciSiradaki;
}
```

Liste boş olduğu için silinecek eleman yoktur

Listede sadece bir düğüm vardır ilkdugum=NULL ataması ile ilk duğum NULL olur

ikinci düğüm tutumuş ilk düğüm silinir ikinci düğüm ilk düğüm yapılır

2.6.3. Tek Yönlü Bağlı Listelerde Aradan Eleman Silmek

Tek yönlü bağlı listelerde, listenin düğümleri arasındaki bir düğümü silen fonksiyon aşağıda verilmiştir. Verilen bu fonksiyonun çalışabilmesi için, silinecek elemanın sırasının fonksiyona parametre olarak aktarılması gerekir. Bu yöntemden başka aradaki elemanları silmek için klavyeden girilecek elemana göre silme yöntemi de vardır. Bu yöntem için fonksiyon hazırlama işlemi öğrencilerimize bırakılmıştır.

```
void aradanSil(int esira){
    struct bagliListe* temp = ilkDugum; //Geçici bir düğüm oluşturulur
    int sayac = 0;
    if(esira == 0){
        bastanSil();
        return;
    }
    while(temp != NULL){
        if(sayac == esira - 1)
            break;
        temp = temp->sonrakiDugum;
        sayac++;
    }
    if(sayac + 1 != esira){
        printf("\n Girdiğiniz Sırada Silinecek Eleman Yok\n");
        return;
    }
    if(temp->sonrakiDugum == NULL){
        sondanSil();
    }
    struct bagliListe * geciciDugum = temp->sonrakiDugum->sonrakiDugum;
    //Yukarıdaki ifade ile temp'in sonrakiDugum 'ü degil ondan da sonraki dugum geciciDugum'de tutulur
    // Çünkü temp'ten sonra gelen düğüm silinecek ve temp silinecek düğümden sonra gelen düğüme bağlanacak.
    free(temp->sonrakiDugum); // free() komutu kullanılarak tempten sonra gelen duğüm siliniyor.
    temp->sonrakiDugum = geciciDugum; // temp geçici düğüme bağlanıyor.
}
```

// silinecek eleman 0. sırada ise bastanSil() fonksiyonu çağırılır.

//while döngüsü temp NULL olana //kadar döner. Bu sırada (sayac == //esira -1) olursa dönguden çıkılır.

//Döngü içerisinde travers işlemi yapılır ve sayaç arttırılır.

//Döngüden çıkıldığında bu eşitsizlik sözkonusu ise listede silenecek eleman yoktur. Fonksiyondan çıkılır.

Travers işlemi sonunda NULL değerine ulaşılmış ise silinecek eleman sondaki elemandır.

2.7. Tek Yönlü Bağlı Listenin Elemanlarını Saymak

Tek Yönlü Bağlı Listedeki elemanları saymak için *void* tipinde bir fonksiyon (elemanSay()) oluşturacağız. Fonksiyonun herhangi bir parametre almasına gerek yoktur. Buna ilave olarak fonksiyonun başında listede eleman olup olmadığını kontrol etmeye de gerek yok. Çünkü listede eleman bulunmuyorsa döngüye girilmeyecek ve sayaç sıfır olacaktır. Aşağıda Tek Yönlü Bağlı Listenin eleman sayısını ekrana yazdıran fonksiyon verilmiştir.

```
//Bağlı Listenin Elemanlarını Sayan Fonksiyon
void elemanSay() {
int sayac = 0;
 struct bagliListe* temp = ilkDugum;
  while(temp != NULL) {
  sayac++;
  temp = temp -> sonrakiDugum;
}
printf("\nBaglı Listede %d dugum var\n",sayac);
}
```

Bölüm Özeti

Bağlı Liste ve Tek Yönlü Bağlı Liste tanımını yapabilmek

Bağlı liste yapısı veri/verileri, diğer verilere erişim için gerekli olan adresi ya da bağı içeren düğümler topluluğudur şeklinde tanımlanabilir. Tek Yönlü bağlı liste ise veri alanına ve kendisinden sonra gelen elemana işaret eden bir adres (işaretçi) alanına sahip düğümler topluluğudur. Tanımdan da anlaşılacağı gibi Tek Yönlü bağlı listeler sadece kendisinden sonra gelen elemana işaret ederler, kendisinden önce gelen elemana işaret etmezler.

Bağlı listelerde, bir sonraki düğüme işaret eden işaretçiler kullanılarak liste elemanları üzerinde kolaylıkla listeye eleman ekleme, silme, arama, liste elemanlarını yazdırma gibi işlemler gerçekleştirilebilir.

Bağlı Listeler ve dizilerin benzerlik ve farklılıklarını açıklayabilmek

Bağlı listeler doğrusal olma özellikleri ile dizilere benzerler. Bu benzerliğin yanında dizilerle bağlı listelerin benzeşmeyen yönleri de vardır. Örneğin, dizilerde çalışma zamanı sırasında dizi boyuttu değiştirilemezken bağlı listelerde çalışma zamanında bilgisayar belleğinde yeni elemanlar için yer ayrılabilir veya kullanılmasına ihtiyaç kalmayan düğümler gerçek manada bellekten silinebilir. Bu bağlı listelerin dizilere göre en önemli farkıdır. Ayrıca, eleman ekleme ve silme işlemlerinin başarımı bağlı listelerde daha yüksektir. Diziler ve bağlı listeler arasında bu ve benzeri

farklardan dolayı diziler yerine çoğu kere bağlı listelerin kullanılması tercih edilir.

Bağlı Listelerin dizilere göre avantajlı ve dezavantajlı yönlerini açıklayabilmek

Yukarıda sözü konu edilen diziler ile bağlı listeler arasındaki farklar, bağlı listelerin hanesine avantaj olarak yazılmaktadır. Şöyle ki; bağlı listelerin boyutlarının derleme zamanından sonra da ihtiyaca göre arttırılıp eksiltilebilmesi, bağlı listeye yeni öğe eklemenin veya bağlı listeden eleman silmenin, dizilere eleman eklemek ve dizilerden eleman silmeye göre daha hızlı, daha kolay ve daha düşük maliyetle yapılabilmesi bağlı listelerin dizilere göre iki önemli avantajıdır.

Bu avantajlarının yanında bağlı listelerin dizilere göre dezavantajları da vardır. Bağlı listelerin dizilere göre birinci dezavantajı dizinin bir elemanına rasgele erişime izin verilirken, bağlı listelerin düğümlerine rasgele erişime izin verilmez. İkinci dezavantaj ise bağlı listelerin düğümlerinde bir sonraki elemana işaret eden işaretçi kullanılması zorunluluğudur. Bu programın çalışması sırasında kullanılan bellek miktarının artmasına neden olur.

Yazılımı projelerinin hazırlanması sırasında bağlı liste veya dizi kullanmak konusunda tercih yaparken yukarıda sayılan avantaj ve dezavantajları göz önüne almakta fayda vardır.

Tek Yönlü Bağlı Liste Oluşturabilmek

C programlama dilinde bir düğüm oluşturmadan önce, oluşturulacak düğümün yapısının tanımlanması gerekir. Bu da bir yapı tanımlayarak gerçekleştirilir. Daha sonra bu yapı temel alınarak ve malloc(), sizeof() fonksiyonları kullanılarak bellekte bağlı listenin düğümleri oluşturulur.