BLM267

Bölüm 6: Bağlantılı Listeler

C Kullanarak Veri Yapıları, İkinci Baskı
Reema Thareja

- giriiş
- Tek Bağlantılı Listeler
- Dairesel Bağlantılı Listeler
- Çift Bağlantılı Listeler
- Dairesel Çift Bağlantılı Listeler
- Başlık Bağlantılı Listeler
- Çoklu Bağlantılı Listeler
- Bağlantılı Listelerin Uygulaması

giriiş

- Bir dizinin, elemanların ardışık bellek konumlarında depolandığı doğrusal bir veri öğeleri koleksiyonu olduğunu inceledik.
- Dizileri tanımlarken dizinin boyutunu belirtmemiz gerekir; bu, dizinin saklayabileceği eleman sayısını kısıtlayacaktır.
- Örneğin, bir diziyi int marks[10] olarak bildirirsek, dizi en fazla 10 veri öğesini depolayabilir, ancak bundan fazlası olamaz.
- Peki ya önceden eleman sayısından emin değilsek?
- Ayrıca, belleğin verimli kullanılabilmesi için elemanların ardışık konumlarda depolanması yerine rastgele herhangi bir konumda depolanması gerekir.
- Yani verimli programlar yazabilmek için, maksimum eleman sayısı ve depolama koşulu kısıtlamalarını ortadan kaldıran bir veri yapısının olması gerekir.

giriiş

- Bağlantılı liste bu kısıtlamalardan bağımsız bir veri yapısıdır.
- Bağlantılı liste, elemanlarını ardışık bellek konumlarında saklamaz ve kullanıcı, listeye istediği sayıda eleman ekleyebilir.
- Ancak bir dizinin aksine, bağlı liste verilere rastgele erişime izin vermez.
- Bağlantılı listedeki elemanlara yalnızca sıralı bir şekilde erişilebilir.
- Ancak bir dizi gibi, ekleme ve silme işlemleri listenin herhangi bir noktasında sabit bir sürede yapılabilir.

giriiş

- Temel terminolojiler
- Bağlantılı liste, basit bir ifadeyle, veri öğelerinin doğrusal bir koleksiyonudur.
- Bu veri elemanlarına düğüm adı verilir.
- Bağlantılı liste, diğer veri yapılarını uygulamak için kullanılabilen bir veri yapısıdır.
- Bu sayede yığınlar, kuyruklar ve bunların varyasyonları gibi veri yapılarının uygulanmasında yapı taşı görevi görür.
- Bağlantılı liste, her düğümün bir veya daha fazla veri alanı ve bir sonraki düğüme işaret eden bir işaretçi içerdiği bir tren veya düğüm dizisi olarak algılanabilir.

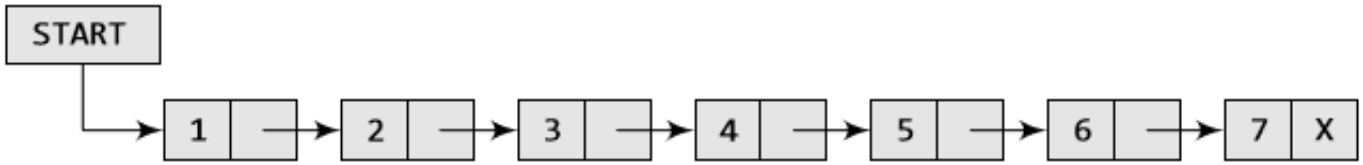


Figure 6.1 Simple linked list

giriiş

- Temel terminolojiler
- Şekil 6.1'de her düğümün bir tam sayı ve bir sonraki düğüme işaret eden bir işaretçi olmak üzere iki parçadan oluştuğu bir bağlı liste görebiliriz.
- Veri içeren düğümün sol kısmı basit bir veri türü, bir dizi veya bir yapı içerebilir.
- Düğümün sağ kısmı bir sonraki düğüme (veya sıradaki düğümün adresine) işaret eden bir işaretçi içerir.
- Son düğüme bağlı bir sonraki düğüm olmayacağından, NULL adı verilen özel bir değer depolayacaktır.
- Şekil 6.1'de NULL işaretçisi X ile temsil edilmektedir. Programlama yaparken NULL'u genellikle -1 olarak tanımlarız.
- Dolayısıyla, NULL işaretçisi listenin sonunu belirtir.
- Bağlı listede her düğüm, aynı tipteki başka bir düğüme işaret eden bir işaretçi içerdiğinden, buna öz-referanslı veri tipi de denir.

giriiş

- Temel terminolojiler
- Bağlantılı listeler, listedeki ilk düğümün adresini saklayan START adlı bir işaretçi değişkeni içerir.
- İlk düğümün adresini içeren START'ı kullanarak tüm listeyi dolaşabiliriz; ilk düğümün bir sonraki kısmı da kendisinden sonraki düğümün adresini depolar.
- Bu teknik kullanılarak listedeki her bir düğüm, bir düğüm zinciri oluşturacaktır.
- Eğer START = NULL ise, bağlı liste boştur ve düğüm içermez.
- C'de, aşağıdaki kodu kullanarak bağlı listeyi uygulayabiliriz:

```
yapı düğümü
{
   int veri;
   yapı düğümü *sonraki;
};
```

giriiş

- Temel terminolojiler
- Bağlı listenin hafızada nasıl tutulduğunu görelim.
- Bağlantılı listeyi oluşturabilmek için node adında iki alanı olan, DATA ve NEXT adında bir yapıya ihtiyacımız var.
- DATA bilgi kısmını depolayacak ve NEXT sıradaki bir sonraki düğümün adresini depolayacaktır. Şekil 6.2'yi düşünün.

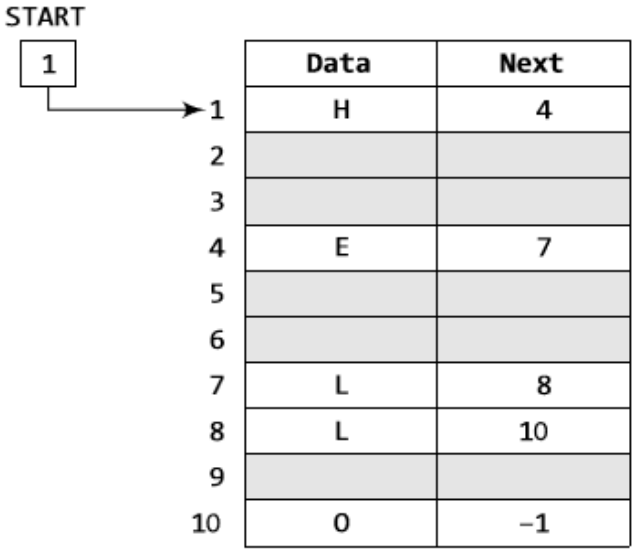


Figure 6.2 START pointing to the first element of the linked list in the memory

giriiş

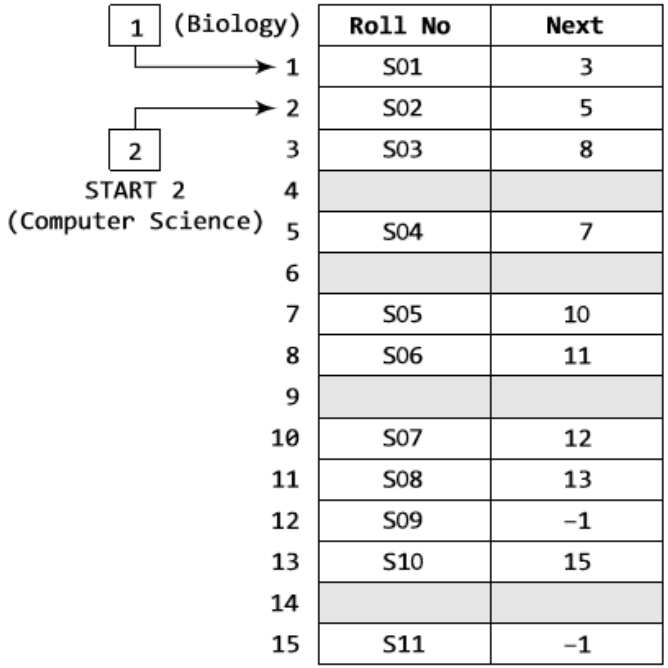
- Temel terminolojiler
- Şekilde START değişkeninin ilk düğümün adresini saklamak için kullanıldığını görebiliriz.
- Burada, bu örnekte, START = 1 olduğundan, ilk veri H olan adres 1'de saklanır.
- İlgili NEXT, bir sonraki düğümün adresi olan 4'ü depolar.
- Yani, bir sonraki veri öğesini almak için adres 4'e bakacağız. Adres 4'ten elde edilen ikinci veri öğesi E'dir.
- Tekrar bir sonraki node'a geçmek için ilgili NEXT'i görüyoruz.
- NEXT girişinden bir sonraki adres olan 7'yi alıyoruz ve veri olarak L'yi alıyoruz.
- Bağlantılı listenin sonunu ifade edeceği için NEXT girişinin –1 veya NULL içerdiği bir konuma ulaşana kadar bu prosedürü tekrarlıyoruz.
- DATA ve NEXT'i bu şekilde dolaştığımızda, yukarıdaki örnekteki bağlı listenin bir araya getirildiğinde HELLO kelimesini oluşturan karakterleri depoladığını görürüz.

giriiş

- Temel terminolojiler
- Şekil 6.2'nin 1 ile 10 arasında değişen bir bellek konumu yığınını gösterdiğine dikkat edin.
- Gölgeli kısım diğer uygulamalara ait verileri içermektedir.
- Bağlantılı bir listenin düğümlerinin ardışık bellek konumlarında olması gerekmediğini unutmayın. Örneğimizde, bağlantılı listenin düğümleri 1, 4, 7, 8 ve 10 adreslerinde depolanır.
- Bilgisayarın belleğinde iki bağlantılı listenin nasıl bir arada tutulduğunu görmek için başka bir örnek ele alalım.
- Örneğin, 11. Sınıf Fen Bilgisi grubundaki öğrencilerden Biyoloji ile Bilgisayar Bilimi arasında seçim yapmaları isteniyor.
- Şimdi her bir konu için bir tane olmak üzere iki adet bağlı liste tutacağız.
- Yani ilk bağlantılı listede Biyoloji bölümünü seçen tüm öğrencilerin sıra numaraları, ikinci listede ise Bilgisayar Bilimi bölümünü seçen öğrencilerin sıra numaraları yer alacaktır.

giriiş

- Temel terminolojiler
- Şimdi Şekil 6.3'e bakalım, bellekte aynı anda iki farklı bağlı liste tutuluyor.
- Listede gezinmede herhangi bir belirsizlik yoktur çünkü her liste, ilgili bağlı listelerin ilk düğümünün adresini veren ayrı bir START işaretçisi tutar.
- Geri kalan düğümlere NEXT'te saklanan değere bakılarak ulaşılır.
- Şekle bakıldığında Biyoloji bölümünü tercih eden öğrencilerin sıra numaralarının S01, S03, S06, S08, S10 ve S11 olduğu sonucuna varılabilir.
- Benzer şekilde Bilgisayar Bilimi'ni seçen öğrencilerin sıra numaraları S02, S04, S05, S07 ve S09'dur.



	Roll No	Next
1	S01	3
2	S02	5
3	S03	8
4		
5	S04	7
6		
7	S05	10
8	S06	11
9		
10	S07	12
11	S08	13
12	S09	-1
13	S10	15
14		
15	S11	-1

Figure 6.3 Two linked lists which are simultaneously

giriiş

- Temel terminolojiler
- Bir düğümün DATA kısmının yalnızca tek bir veri öğesi, bir dizi veya bir yapı içerebileceğini daha önce söylemiştik.
- Bellekte saklanan bağlı listede bir yapının nasıl tutulduğunu görmek için bir örnek ele alalım.
- Öğrencilerin sicil numarası, adı, toplamı ve notunun bağlantılı listeler kullanılarak saklandığı bir senaryoyu ele alalım.
- Şimdi, NEXT işaretçisinin verileri alfabetik olarak depolamak için nasıl kullanıldığını göreceğiz. Bu, Şekil 6.4'te gösterilmiştir.

giriiş

	Roll No	Name	Aggregate	Grade	Next
1	S01	Ram	78	Distinction	6
2	S02	Shyam	64	First division	14
3					
4	S03	Mohit	89	Outstanding	17
5					
6	S04	Rohit	77	Distinction	2
7	S05	Varun	86	Outstanding	10
8	S06	Karan	65	First division	12
9					
10	S07	Veena	54	Second division	−1
11	S08	Meera	67	First division	4
12	S09	Krish	45	Third division	13
13	S10	Kusum	91	Outstanding	11
14	S11	Silky	72	First division	7
15					
16					
17	S12	Monica	75	Distinction	1
18	S13	Ashish	63	First division	19
19	S14	Gaurav	61	First division	8

START → 18

Figure 6.4 Students' linked list

giriiş

- Bağlantılı Listeler ve Diziler
- Hem diziler hem de bağlı listeler veri elemanlarının doğrusal bir koleksiyonudur.
- Ancak bir dizinin aksine, bağlı liste düğümlerini ardışık bellek konumlarında depolamaz.
- Dizi ile bağlı liste arasındaki bir diğer fark ise bağlı listenin verilere rastgele erişime izin vermemesidir.
- Bağlantılı listedeki düğümlere yalnızca sıralı bir şekilde erişilebilir. Ancak bir dizi gibi, eklemeler ve silmeler listedeki herhangi bir noktada sabit bir sürede yapılabilir.
- Bağlı listenin dizilere göre bir diğer avantajı da listeye istediğimiz sayıda eleman ekleyebilmemizdir.
- Dizi söz konusu olduğunda bu mümkün değildir.
- Örneğin, bir diziyi int marks[20] olarak bildirirsek, dizi en fazla 20 veri öğesini depolayabilir. Bağlantılı liste durumunda böyle bir kısıtlama yoktur.
- Bu nedenle, bağlantılı listeler, bir sonraki düğümlerin adresini depolamak için gereken ekstra alan pahasına, ilgili verileri depolamanın ve ekleme, silme ve bilgi güncelleme gibi temel işlemleri gerçekleştirmen etkili bir yolunu sağlar.

C Kullanarak Veri Yapıları, İkinci Baskı
Reema Thareja

giriiş

- Bağlantılı Liste için bellek tahsisi ve tahsisin kaldırılması
- Bağlı listenin bellekte nasıl temsil edildiğini gördük.
- Hafızada var olan bir bağlı listeye bir düğüm eklemek istediğimizde, öncelikle hafızada boş alan bulup, daha sonra bu alanı bilgileri depolamak için kullanırız.
- Örneğin, Şekil 6.5'te gösterilen bağlantılı listeyi ele alalım.
- Bağlantılı liste, öğrencilerin sıra numaralarını, Biyoloji dersinden aldıkları notları ve son olarak sıradaki düğümün adresini saklayan SONRAKİ alanını içerir.
- Şimdi, sınıfa yeni bir öğrenci katılırsa ve diğer öğrencilerin girdiği aynı sınava girmesi istenirse, o zaman yeni öğrencinin notları da bağlantılı listeye kaydedilmelidir.
- Bunun için boş bir alan bulup bilgileri oraya kaydediyoruz.
- Şekil 6.5'te gri gölgeli kısım boş alanı göstermektedir ve bu neden 4 adet kullanılabilir bellek konumumuz bulunmaktadır.
- Verilerimizi depolamak için bunlardan herhangi birini kullanabiliriz. Bu, Şekil 6.5(a) ve (b)'de gösterilmiştir.

giriiş

- **Bağlantılı Liste için bellek tahsisi ve tahsisin kaldırılması**

START
1
(Biology)

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	–1

(a)

START
1
(Biology)

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4	S12	75	–1
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	4

(b)

Figure 6.5 (a) Students' linked list and (b) linked list after the insertion of new student's record

giriiş

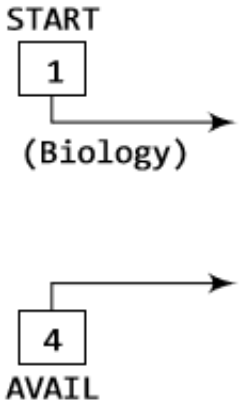
- Bağlantılı Liste için bellek tahsisi ve tahsisin kaldırılması
- Peki, şimdi soru şu; hafızanın hangi kısmı kullanılabilir durumda ve hangi kısmı meşgul?
- Bağlı listeden bir düğümü sildiğimizde, o düğümün işgal ettiği belleğin durumunu işgal edilmişten kullanılabilire kim değiştirir?
- Cevap işletim sistemidir.
- İşletim sisteminin tüm bunları nasıl yaptığının mekanizmasını tartışmak bu kitabın kapsamı dışındadır.
- Yani daha basit bir dille şunu söyleyebiliriz ki, bilgisayar bunu kullanıcı veya programcının hiçbir müdahalesine gerek kalmadan kendi kendine yapıyor.
- Bir programcı olarak sizin yapmanız gereken tek şey listedeki ekleme ve çıkarma işlemlerini yapacak kodlarla ilgilenmektir.
- Ancak bunun ardındaki temel kavrama kısaca değinelim.
- Bilgisayar tüm boş bellek hücrelerinin bir listesini tutar. Kullanılabilir alanın bu listesine boş havuz denir.

giriiş

- Bağlantılı Liste için bellek tahsisi ve tahsisin kaldırılması
- Her bağlı listenin, listenin ilk düğümünün adresini saklayan START adlı bir işaretçi değişkenine sahip olduğunu gördük.
- Benzer şekilde, boş havuz (tüm boş bellek hücrelerinin bağlı listesi) için, ilk boş alanın adresini depolayan AVAIL adlı bir işaretçi değişkenimiz var.
- Biyoloji dersindeki tüm öğrencilerin notlarını saklayan bağlantılı listenin bellek gösterimini yeniden ele alalım.
- Artık yeni bir öğrencinin kaydının eklenmesi gerektiğinde AVAIL'in işaret ettiği bellek adresi alınacak ve istenilen bilginin saklanması için kullanılacak.
- Ekleme işleminden sonra bir sonraki kullanılabilir boş alanın adresi AVAIL'de saklanacaktır.
- Örneğin, Şekil 6.6'da, yeni düğümün eklenmesi için ilk boş bellek alanı kullanıldığında, AVAIL adresi 6 olacak şekilde ayarlanacaktır.

giriiş

- **Bağlantılı Liste için bellek tahsisi ve tahsisin kaldırılması**



	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			6
5	S04	98	7
6			9
7	S05	55	8
8	S06	34	10
9			14
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			−1
15	S11	56	−1

Figure 6.6 Linked list with AVAIL and START pointers

giriiş

- Bağlantılı Liste için bellek tahsisi ve tahsisin kaldırılması
- Bu, mevcut bir bağlı listeye yeni bir düğüm eklemekle ilgiliydi.
- Şimdi bir node'u veya tüm bağlı listeyi silmeyi ele alacağız.
- Mevcut bir bağlı listeden belirli bir düğümü sildiğimizde veya tüm bağlı listeyi sildiğimizde, bu düğümün kapladığı alan, bellek alanına ihtiyaç duyan başka bir program tarafından yeniden kullanılabilmesi için boş havuza geri verilmelidir.
- İşletim sistemi, serbest bırakılan belleği serbest havuza ekleme görevini gerçekleştirir.
- İşletim sistemi, CPU'nun boşta olduğunu tespit ettiğinde veya programların bellek alanı azaldığında bu işlemi gerçekleştirir.
- İşletim sistemi tüm bellek hücrelerini tarar ve bazı programlar tarafından kullanılan hücreleri işaretler.
- Daha sonra kullanılmayan tüm hücreleri toplar ve adreslerini boş havuza ekler, böylece bu hücreler diğer programlar tarafından tekrar kullanılabilir.
- Bu işleme çöp toplama denir.

Tek Bağlantılı Listeler

- Tek bağlantılı liste, her düğümün bir miktar veri ve aynı veri türündeki bir sonraki düğüme işaret eden bir işaretçi içerdiği en basit bağlantılı liste türüdür.
- Düğümün bir sonraki düğüme işaret eden bir işaretçi içerdiğini söylediğimizde, düğümün bir sonraki düğümün adresini sırayla sakladığını kastediyoruz.
- Tek bağlantılı liste, verilerin yalnızca bir şekilde dolaşılmasına izin verir. Şekil 6.7 tek bağlantılı listeyi göstermektedir.

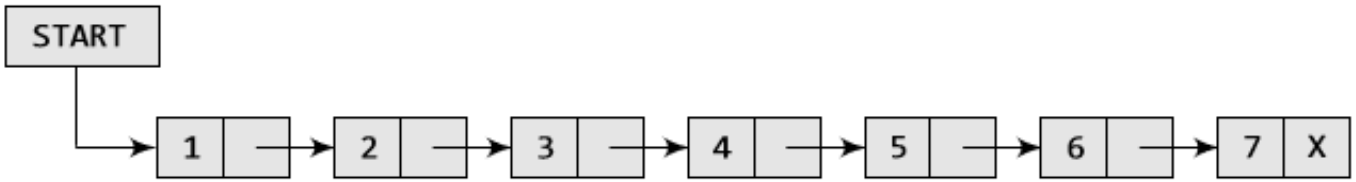


Figure 6.7 Singly linked list

Tek Bağlantılı Listeler

- Bağlantılı Bir Listeyi Gezinme
- Bağlantılı bir listede gezinmek, listedeki düğümlere erişerek üzerlerinde bazı işlemler yapmak anlamına gelir.
- Bağlantılı listelerin her zaman listenin ilk düğümünün adresini saklayan bir START işaretçi değişkeni içerdiğini unutmayın.
- Listenin sonu, son düğümün NEXT alanına NULL veya -1 yazılarak işaretlenir.
- Bağlantılı listeyi dolaşmak için, o anda erişilen düğümü işaret eden başka bir işaretçi değişkeni olan PTR'yi de kullanırız.
- Bağlantılı bir listeyi dolaşmak için algoritma Şekil 6.8'de gösterilmiştir.

Tek Bağlantılı Listeler

- Bağlantılı Bir Listeyi Gezinme
- Bu algoritmada ilk olarak PTR'yi START adresiyle başlatıyoruz.
- Yani artık PTR, bağlı listenin ilk düğümünü işaret ediyor.
- Daha sonra Adım 2'de, PTR son düğümü işleyene kadar, yani NULL ile karşılaşana kadar tekrarlanan bir while döngüsü yürütülür.
- 3. Adımda işlemi (örneğin, yazdırma) geçerli düğüme, yani PTR tarafından işaret edilen düğüme uygularız.
- 4. Adımda, PTR değişkenini NEXT alanında adresi saklanan düğüme işaret edecek şekilde ayarlayarak bir sonraki düğüme geçiyoruz.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:            Apply Process to PTR->DATA
Step 4:            SET PTR = PTR->NEXT
       [END OF LOOP]
Step 5: EXIT
```

Figure 6.8 Algorithm for traversing a linked list

Tek Bağlantılı Listeler

- Bağlantılı Bir Listeyi Gezinme
- Şimdi, bağlı listedeki düğüm sayısını sayan bir algoritma yazalım.
- Bunu yapmak için, listenin her bir düğümünü dolaşacağız ve her bir düğümü dolaşırken sayacı 1 artıracağız.
- NULL değerine ulaştığımızda, yani bağlı listenin tüm düğümleri dolaşıldığında, sayacın son değeri gösterilecektir.
- Şekil 6.9, bağlı bir listedeki düğüm sayısını yazdırmak için kullanılan algoritmayı göstermektedir.

```
Step 1: [INITIALIZE] SET COUNT = 0
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4:              SET COUNT = COUNT + 1
Step 5:              SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT
```

Figure 6.9 Algorithm to print the number of nodes in a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listede bir değeri arama
- Bağlantılı listede arama yapmak, bağlantılı listedeki belirli bir elemanı bulmak anlamına gelir.
- Daha önce de tartışıldığı gibi, bir bağlantılı liste, bilgi kısmı ve sonraki kısım olmak üzere iki bölüme ayrılmış düğümlerden oluşur.
- Yani arama, verilen değerin düğümün bilgi kısmında bulunup bulunmadığını bulmak anlamına gelir.
- Eğer mevcutsa algoritma, değeri içeren düğümün adresini döndürür.

Tek Bağlantılı Listeler

- Bağlantılı Listede bir değeri arama
- Şekil 6.10, bağlantılı bir listede arama yapmak için kullanılan algoritmayı göstermektedir.
- 1. Adımda, ilk düğümün adresini içeren START ile işaretçi değişkeni PTR'yi başlatıyoruz.
- Adım 2'de, aramanın yapıldığı her düğümün DATA'sını VAL ile karşılaştıracak bir while döngüsü yürütülür.
- Arama başarılı olursa, yani VAL bulunursa, o düğümün adresi POS'ta saklanır ve kontrol algoritmanın son ifadesine atlar.
- Ancak arama başarısız olursa POS, VAL'in bağlı listede bulunmadığını belirten NULL olarak ayarlanır.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:     IF VAL = PTR -> DATA
                SET POS = PTR
                Go To Step 5
            ELSE
                SET PTR = PTR -> NEXT
            [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

Figure 6.10 Algorithm to search a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listede bir değeri arama
- Şekil 6.10, bağlantılı bir listede arama yapmak için kullanılan algoritmayı göstermektedir.
- 1. Adımda, ilk düğümün adresini içeren START ile işaretçi değişkeni PTR'yi başlatıyoruz.
- Adım 2'de, aramanın yapıldığı her düğümün DATA'sını VAL ile karşılaştıracak bir while döngüsü yürütülür.
- Arama başarılı olursa, yani VAL bulunursa, o düğümün adresi POS'ta saklanır ve kontrol algoritmanın son ifadesine atlar.
- Ancak arama başarısız olursa POS, VAL'in bağlı listede bulunmadığını belirten NULL olarak ayarlanır.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:     IF VAL = PTR -> DATA
                SET POS = PTR
                Go To Step 5
            ELSE
                SET PTR = PTR -> NEXT
            [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

Figure 6.10 Algorithm to search a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listede bir değeri arama
- Şekil 6.11'de gösterilen bağlantılı listeyi ele alalım. Eğer VAL = 4 ise, algoritmanın akışı şekilde gösterildiği gibi açıklanabilir.

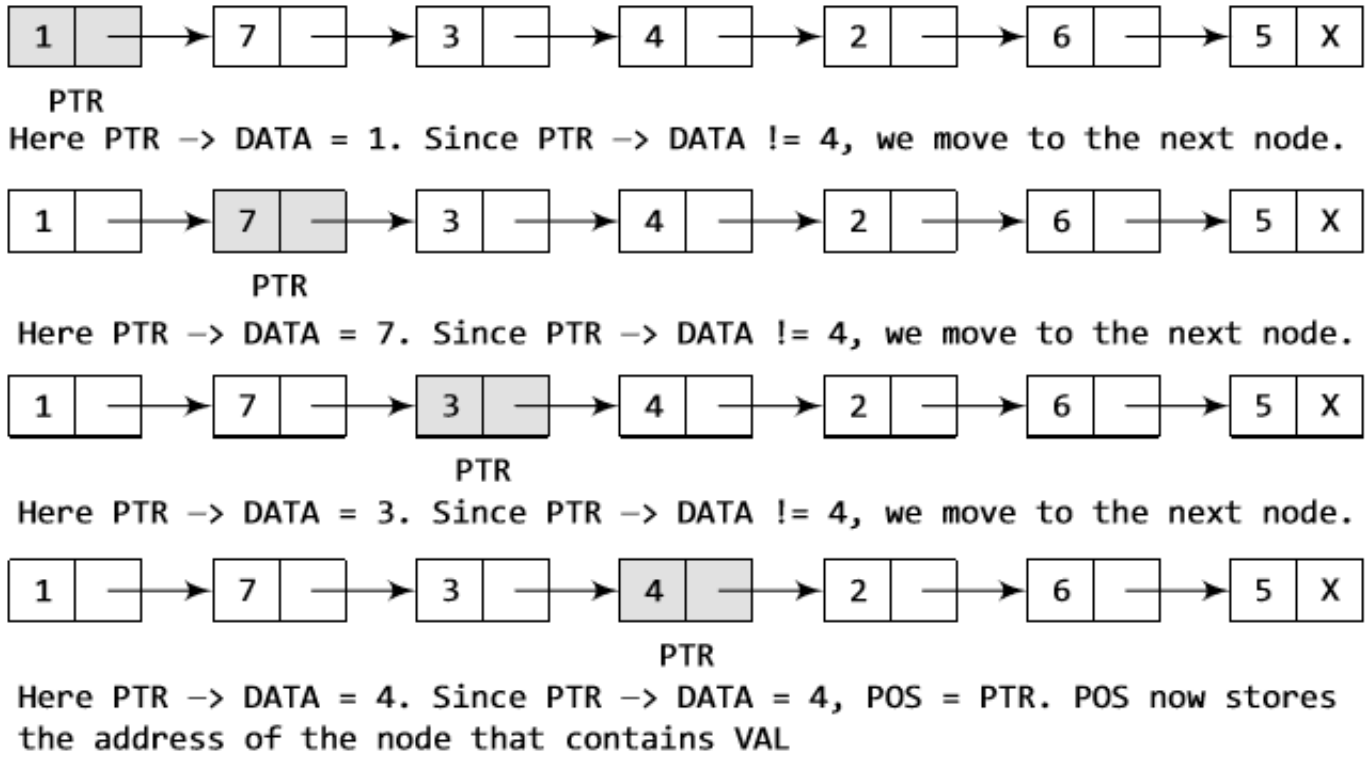


Figure 6.11 Searching a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listeye yeni bir düğüm ekleme
- Bu bölümde, var olan bir bağlı listeye yeni bir düğümün nasıl eklendiğini göreceğiz.
- Dört vakayı ele alacağız ve her vakada eklemenin nasıl yapıldığını göreceğiz.
- Durum 1: Yeni düğüm başa eklenir.
- Durum 2: Yeni düğüm sona eklenir.
- Durum 3: Yeni düğüm, belirli bir düğümden sonra eklenir.
- Durum 4: Yeni düğüm, belirli bir düğümden önce eklenir.
- Bu dört durumda eklemeleri gerçekleştirecek algoritmaları anlatmadan önce, OVERFLOW adı verilen önemli bir terimi ele alalım.
- Taşma, AVAIL = NULL olduğunda veya sistemde boş bellek hücresi bulunmadığında oluşan bir durumdur.
- Bu durum oluştuğunda programın buna uygun bir mesaj vermesi gerekir.

Tek Bağlantılı Listeler

- Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.12'de gösterilen bağlantılı listeyi ele alalım.
- Diyelim ki 9 numaralı veriye sahip yeni bir düğüm eklemek ve bunu listenin ilk düğümü olarak eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.

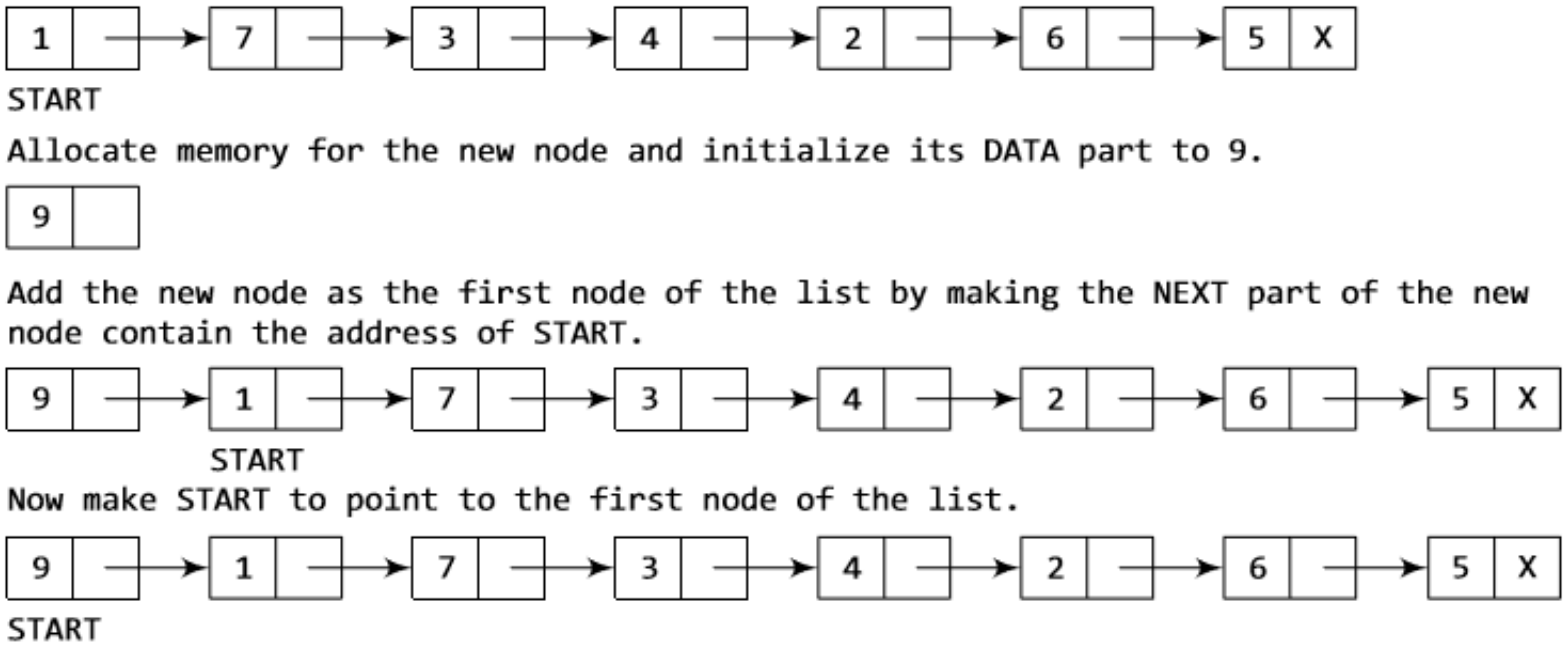


Figure 6.12 Inserting an element at the beginning of a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.13, bağlı listenin başına yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 1. Adımda öncelikle yeni düğüm için bellek olup olmadığını kontrol ediyoruz.
- Boş hafıza tükendiğinde OVERFLOW (TAŞMA) mesajı yazdırılır.
- Aksi takdirde eğer boş bir bellek hücresi varsa yeni düğüm için alan ayırırız.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

Figure 6.13 Algorithm to insert a new node at the beginning

Tek Bağlantılı Listeler

- Bağlantılı Listenin Başına Bir Düğüm Ekleme
- DATA kısmını verilen VAL ile ayarlayın ve NEXT kısmı START'ta saklanan listenin ilk düğümünün adresi ile başlatılır.
- Artık yeni düğüm listenin ilk düğümü olarak eklendiğinden, artık START düğümü olarak bilinecek, yani START işaretçi değişkeni artık NEW_NODE'un adresini tutacak.
- Aşağıdaki iki adımı dikkate alın:
- Adım 2: NEW_NODE = AVAIL olarak ayarlayın
- Adım 3: AVAIL = AVAIL -> NEXT olarak ayarlayın
- Bu adımlar yeni düğüm için bellek ayırır.
- C'de, kullanıcı adına otomatik olarak bellek ayırma işlemini yapan malloc(), alloc ve calloc() gibi fonksiyonlar vardır.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

Figure 6.13 Algorithm to insert a new node at the beginning

Tek Bağlantılı Listeler

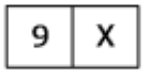
- Bağlantılı Listenin Sonuna Bir Düğüm Ekleme
- Şekil 6.14'te gösterilen bağlantılı listeyi ele alalım.
- Diyelim ki listenin son düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.
- Şekil 6.15, bağlı listenin sonuna yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 6. Adımda, PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani, PTR artık bağlı listenin ilk düğümüne işaret ediyor. While döngüsünde, son düğüme ulaşmak için bağlı listeyi dolaşıyoruz.
- Son düğüme ulaştığımızda, Adım 9'da, yeni düğümün adresini saklamak için son düğümün NEXT işaretçisini değiştiririz.
- Yeni düğümün NEXT alanının, bağlı listenin sonunu belirten NULL değerini içerdiğini unutmayın.

Tek Bağlantılı Listeler

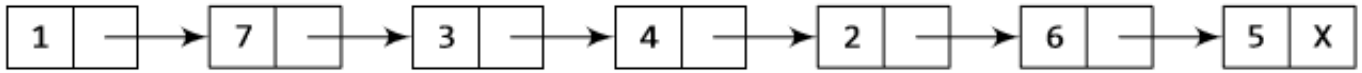
Bağlantılı Listenin Sonuna Bir Düğüm Ekleme

START

Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.

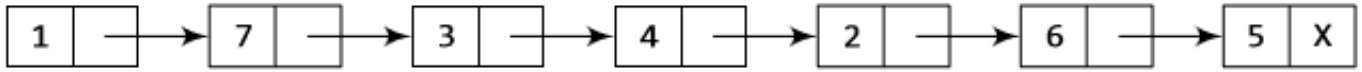


Take a pointer variable PTR which points to START.



START, PTR

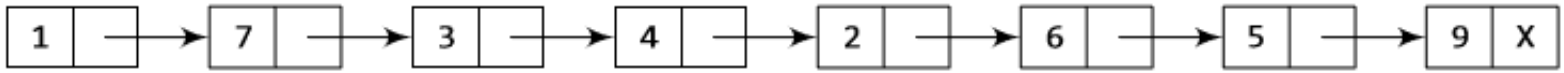
Move PTR so that it points to the last node of the list.



START

PTR

Add the new node after the node pointed by PTR. This is done by storing the address of the new node in the NEXT part of PTR.



START

PTR

Tek Bağlantılı Listeler

Bağlantılı Listenin Sonuna Bir Düğüm Ekleme

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET NEW_NODE->NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR->NEXT != NULL
Step 8:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 9: SET PTR->NEXT = NEW_NODE
Step 10: EXIT
```

Figure 6.15 Algorithm to insert a new node at the end

Tek Bağlantılı Listeler

- Bağlantılı Listede Verilen Bir Düğümden Sonra Bir Düğüm Ekleme
- Şekil 6.17'de gösterilen bağlantılı listeyi ele alalım.
- 3 numaralı veriyi içeren düğümün ardından 9 değerine sahip yeni bir düğüm eklemek istediğimizi varsayalım.
- Bağlı listede yapılacak değişiklikleri tartışmadan önce, Şekil 6.16'da gösterilen algoritmaya bakalım.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR->DATA
        != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
         [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

Figure 6.16 Algorithm to insert a new node after a node that has value NUM

Tek Bağlantılı Listeler

- Bağlantılı Listede Verilen Bir Düğümden Sonra Bir Düğüm Ekleme
- 5. Adımda, PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani PTR artık bağlı listenin ilk düğümünü işaret ediyor.
- Daha sonra PTR'den önceki düğümün adresini depolamak için kullanılacak olan PREPTR adlı başka bir işaretçi değişkeni alırız.
- Başlangıçta PREPTR, PTR olarak başlatılır.
- Yani artık PTR, PREPTR ve START'ın hepsi bağlı listenin ilk düğümüne işaret ediyor.
- While döngüsünde, bağlı listede dolaşarak değeri NUM olan düğüme ulaşırız.
- Yeni düğüm bu düğümden sonra ekleneceği için bu düğüme ulaşmamız gerekiyor.
- Bu düğüme ulaştığımızda, 10. ve 11. Adımlarda, NEXT işaretçilerini, istenen düğümden sonra yeni düğümün eklenmesini sağlayacak şekilde değiştiririz.

Tek Bağlantılı Listeler

- **Bağlantılı Listede Verilen Bir Düğümden Sonra Bir Düğüm Ekleme**

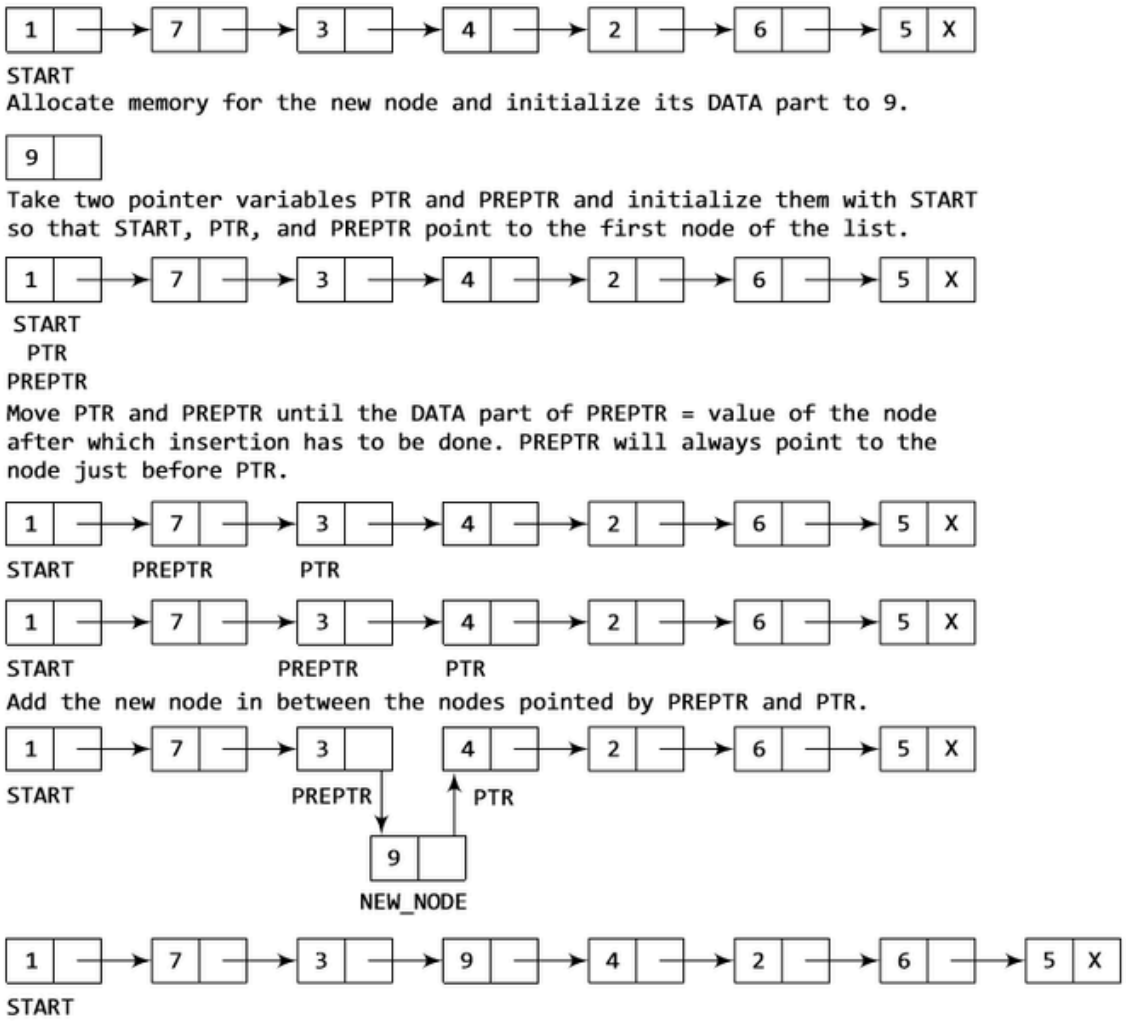


Figure 6.17 Inserting an element after a given node in a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listede Verilen Bir Düğümden Önce Bir Düğüm Ekleme
- Şekil 6.19'da gösterilen bağlantılı listeyi ele alalım.
- Diyelim ki 3 değerini içeren düğümden önce 9 değerine sahip yeni bir düğüm eklemek istiyoruz.
- Bağlı listede yapılacak değişiklikleri tartışmadan önce, Şekil 6.18'de gösterilen algoritmaya bakalım.
- 5. Adımda, PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani PTR artık bağlı listenin ilk düğümünü işaret ediyor.
- Daha sonra PREPTR adında bir işaretçi değişkeni daha alıp onu PTR ile başlatıyoruz.
- Yani artık PTR, PREPTR ve START'ın hepsi bağlı listenin ilk düğümüne işaret ediyor.

Tek Bağlantılı Listeler

- Bağlantılı Listede Verilen Bir Düğümden Önce Bir Düğüm Ekleme
- While döngüsünde, bağlı listede dolaşarak değeri NUM olan düğüme ulaşırız.
- Yeni düğüm bu düğümden önce ekleneceği için bu düğüme ulaşmamız gerekiyor.
- Bu düğüme ulaştığımızda, 10. ve 11. Adımlarda, NEXT işaretçilerini, yeni düğümün istenen düğümden önce eklenmesini sağlayacak şekilde değiştiririz.

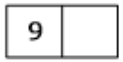
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

Figure 6.18 Algorithm to insert a new node before a node that has value NUM

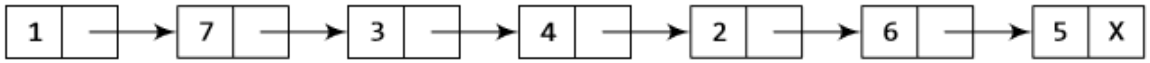
Tek Bağlantılı Listeler

- **Bağlantılı Listede Verilen Bir Düğümden Önce Bir Düğüm Ekleme**

START
Allocate memory for the new node and initialize its DATA part to 9.

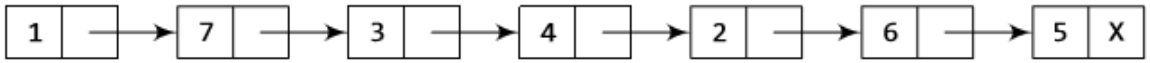


Initialize PREPTR and PTR to the START node.



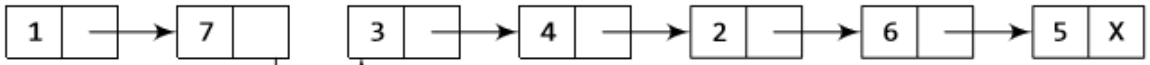
START
PTR
PREPTR

Move PTR and PREPTR until the DATA part of PTR = value of the node before which insertion has to be done. PREPTR will always point to the node just before PTR.

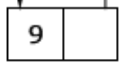


START PREPTR PTR

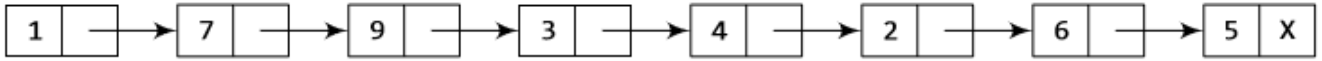
Insert the new node in between the nodes pointed by PREPTR and PTR.



START PREPTR PTR



NEW_NODE



START

Tek Bağlantılı Listeler

- Bağlantılı Listeden bir düğümü silme
- Bu bölümde, var olan bir bağlı listeden bir düğümün nasıl silineceğini ele alacağız.
- Üç durumu ele alacağız ve her durumda silme işleminin nasıl yapıldığını göreceğiz.
- Durum 1: İlk düğüm silinir.
- Durum 2: Son düğüm silinir.
- Durum 3: Belirli bir düğümden sonraki düğüm silinir.
- Bu üç durumdaki algoritmaları anlatmadan önce UNDERFLOW adı verilen önemli bir terimi ele alalım.
- Alt akış, boş olan bir bağlı listeden bir düğümü silmeye çalıştığımızda oluşan bir durumdur.
- Bu durum START = NULL olduğunda veya silinecek düğüm kalmadığında gerçekleşir.
- Bağlı listeden bir düğümü sildiğimizde aslında o düğümün kapladığı belleği boşaltmamız gerektiğini unutmayın.
- Bellek, diğer programların ve verilerin depolanması için kullanılabilmesi amacıyla boş havuza geri gönderilir.
- Silme durumu ne olursa olsun, AVAIL işaretçisini her zaman yakın zamanda boşaltılan adresi gösterecek şekilde değiştiririz.

Tek Bağlantılı Listeler

- Bağlantılı Listeden İlk Düğümü Silme
- Şekil 6.20'deki bağlantılı listeyi ele alalım.
- Listenin başından bir düğümü silmek istediğimizde, bağlı listede aşağıdaki değişiklikler yapılacaktır.

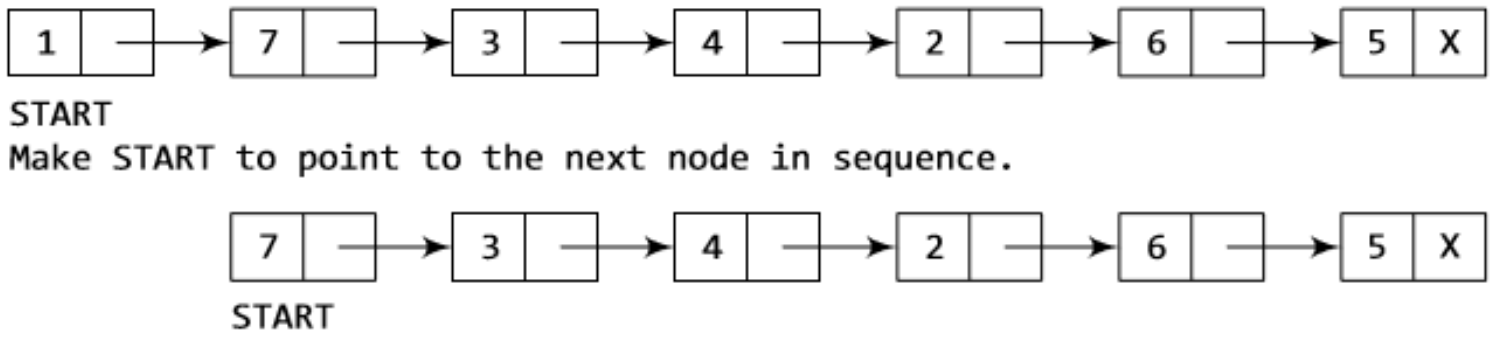


Figure 6.20 Deleting the first node of a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listeden İlk Düğümü Silme
- Şekil 6.21, bağlı listeden ilk düğümü silmek için kullanılan algoritmayı göstermektedir.
- 1. Adımda bağlı listenin var olup olmadığını kontrol ediyoruz.
- START = NULL ise listede düğüm olmadığı anlamına gelir ve kontrol algoritmanın son ifadesine aktarılır.
- Ancak, bağlı listede düğümler varsa, listenin ilk düğümünü işaret edecek şekilde ayarlanmış bir işaretçi değişkeni PTR kullanırız.
- Bunun için PTR'yi listenin ilk düğümünün adresini saklayan START ile başlatıyoruz.
- Adım 3'te START'ın sıradaki düğüme işaret etmesi sağlanır ve son olarak PTR'nin işaret ettiği düğümün (başlangıçta listenin ilk düğümü) işgal ettiği bellek serbest bırakılır ve boş havuza geri döndürülür.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

Figure 6.21 Algorithm to delete the first node

Tek Bağlantılı Listeler

- Bağlantılı Listeden Son Düğümü Silme
- Şekil 6.22'de gösterilen bağlantılı listeyi ele alalım.
- Bağlı listeden son düğümü silmek istediğimizi varsayalım, bu durumda bağlı listede aşağıdaki değişiklikler yapılacaktır.

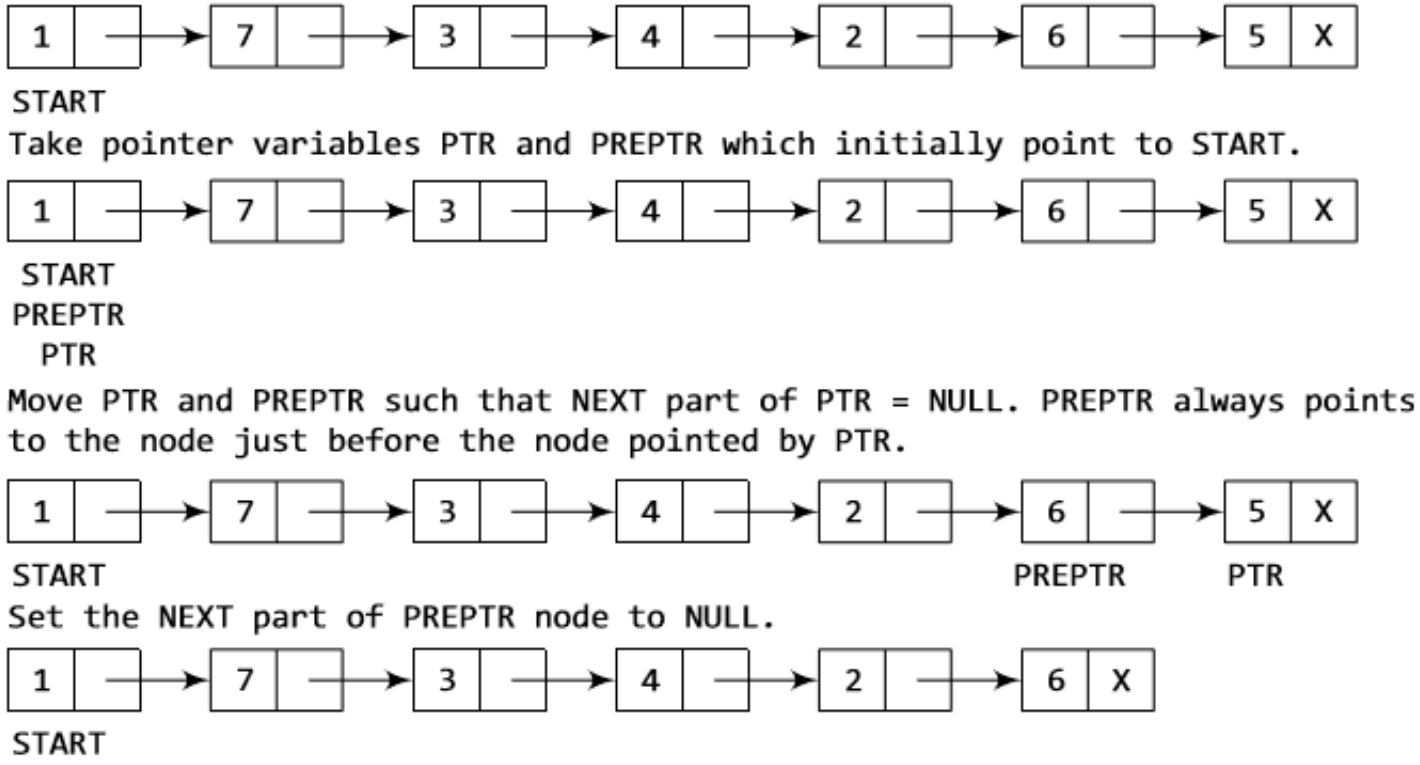


Figure 6.22 Deleting the last node of a linked list

Tek Bağlantılı Listeler

- Bağlantılı Listeden Son Düğümü Silme
- Şekil 6.23, bağlı listeden son düğümü silmek için kullanılan algoritmayı göstermektedir.
- 2. Adımda PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani, PTR artık bağlı listenin ilk düğümüne işaret ediyor. While döngüsünde, her zaman PTR'den önceki bir düğüme işaret edecek şekilde PREPTR adında başka bir işaretçi değişkeni alıyoruz.
- Son düğüme ve sondan ikinci düğüme ulaştığımızda, sondan ikinci düğümün NEXT işaretçisini NULL olarak ayarlarız, böylece bu artık bağlı listenin (yeni) son düğümü olur.
- Önceki son düğümün belleği serbest bırakılır ve serbest havuza geri döndürülür.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

Figure 6.23 Algorithm to delete the last node

Tek Bağlantılı Listeler

- Bağlantılı Listede Belirli Bir Düğümden Sonraki Düğümü Silme
- Şekil 6.24'te gösterilen bağlantılı listeyi ele alalım.
- Diyelim ki 4 veri değerini içeren düğümün ardından gelen düğümü silmek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.
- Şekil 6.25, bağlı listeden belirli bir düğümden sonraki düğümü silmek için kullanılan algoritmayı göstermektedir.
- 2. Adımda PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani, PTR artık bağlı listenin ilk düğümüne işaret ediyor. While döngüsünde, her zaman PTR'den önceki bir düğüme işaret edecek şekilde PREPTR adında başka bir işaretçi değişkeni alıyoruz.
- VAL'i içeren düğüme ve onu takip eden düğüme ulaştığımızda, VAL'i içeren düğümün next işaretçisini, onu takip eden düğümün next alanında bulunan adrese ayarlarız.
- Belirtilen düğümden sonraki düğümün belleği serbest bırakılır ve serbest havuza geri döndürülür.

Tek Bağlantılı Listeler

- **Bağlantılı Listede Belirli Bir Düğümden Sonraki Düğümü Silme**

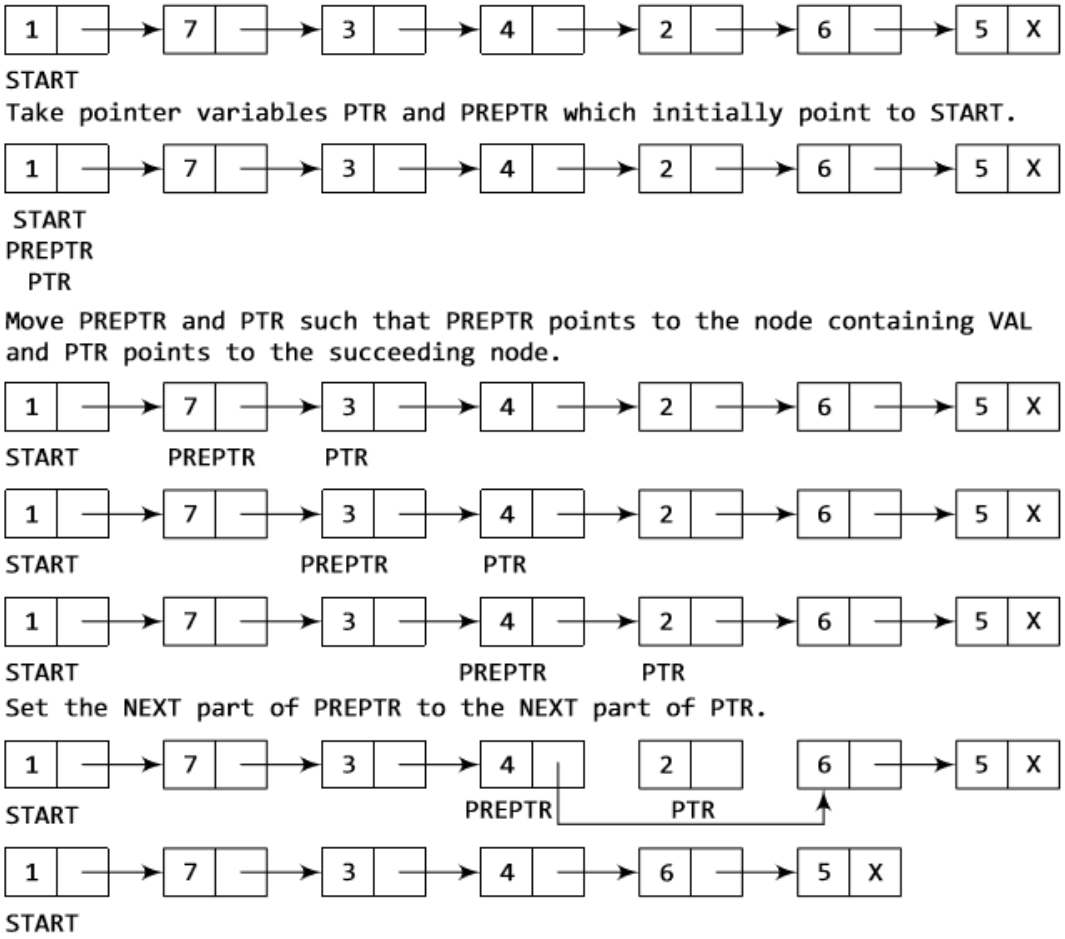


Figure 6.24 Deleting the node after a given node in a linked list

Tek Bağlantılı Listeler

- **Bağlantılı Listede Belirli Bir Düğümden Sonraki Düğümü Silme**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

Figure 6.25 Algorithm to delete the node after a given node

Tek Bağlantılı Listeler

```
struct node *create_ll(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter -1 to end");
        printf("\n Enter the data : ");
        scanf("%d", &num);
        while(num!=-1)
        {
                new_node = (struct node*)malloc(sizeof(struct node));
                new_node -> data=num;
                if(start==NULL)
                {
                        new_node -> next = NULL;
                        start = new_node;
                }
                else
                {
                        ptr=start;

                        while(ptr->next!=NULL)
                        ptr=ptr->next;
                        ptr->next = new_node;
                        new_node->next=NULL;
                }
                printf("\n Enter the data : ");
                scanf("%d", &num);
        }
        return start;
}
```

Tek Bağlantılı Listeler

```
struct node *display(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr != NULL)
        {
                printf("\t %d", ptr -> data);
                ptr = ptr -> next;
        }
        return start;
}
```

Tek Bağlantılı Listeler

```
struct node *insert_beg(struct node *start)
{
        struct node *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        new_node -> next = start;
        start = new_node;
        return start;
}
```

Tek Bağlantılı Listeler

```
struct node *insert_end(struct node *start)
{
        struct node *ptr, *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        new_node -> next = NULL;
        ptr = start;
        while(ptr -> next != NULL)
        ptr = ptr -> next;
        ptr -> next = new_node;
        return start;
}
```

Tek Bağlantılı Listeler

```
struct node *insert_after(struct node *start)
{
        struct node *new_node, *ptr, *preptr;
        int num, val;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        printf("\n Enter the value after which the data has to be inserted : ");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr = start;
        preptr = ptr;
        while(preptr -> data != val)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next=new_node;
        new_node -> next = ptr;
        return start;
}
```

Tek Bağlantılı Listeler

```
struct node *delete_beg(struct node *start)
{
        struct node *ptr;
        ptr = start;
        start = start -> next;
        free(ptr);
        return start;
}
```

```
{
        struct node *ptr, *preptr;
        ptr = start;
        while(ptr -> next != NULL)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next = NULL;
        free(ptr);
        return start;
```

Tek Bağlantılı Listeler

```
struct node *delete_after(struct node *start)
{
        struct node *ptr, *preptr;
        int val;
        printf("\n Enter the value after which the node has to deleted : ");
        scanf("%d", &val);
        ptr = start;
        preptr = ptr;
        while(preptr -> data != val)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next=ptr -> next;
        free(ptr);
        return start;
}
```

Dairesel Bağlantılı Listeler

- Dairesel bağlı listede son düğüm, listenin ilk düğümüne işaret eden bir işaretçi içerir.
- Tek bağlantılı dairesel bir liste olabileceği gibi çift bağlantılı dairesel bir liste de olabilir.
- Dairesel bağlı bir listede gezinirken, herhangi bir düğümden başlayıp, listeyi ileri veya geri herhangi bir yönde, başladığımız düğüme ulaşana kadar gezebiliriz.
- Bu nedenle dairesel bağlantılı listenin başlangıcı ve sonu yoktur. Şekil 6.26 dairesel bağlantılı listeyi göstermektedir.

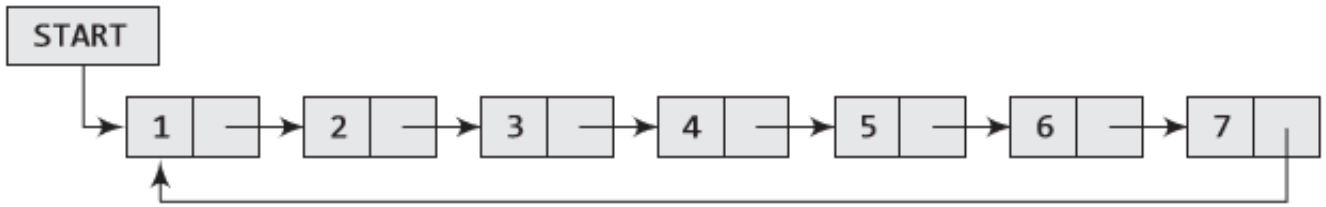


Figure 6.26 Circular linked list

Dairesel Bağlantılı Listeler

- Dairesel bağlı listenin tek dezavantajı yinelemenin karmaşık olmasıdır.
- Listenin hiçbir düğümünün NEXT kısmında NULL değer bulunmadığına dikkat edin.
- Dairesel bağlı listeler işletim sistemlerinde görev bakımı için yaygın olarak kullanılır.
- Şimdi dairesel bağlı listenin kullanıldığı bir örneği ele alacağız.
- İnternette gezinirken Geri ve İleri butonlarını kullanarak daha önce ziyaret ettiğimiz sayfalara geri dönebiliriz.
- Peki bu nasıl yapılır? Cevabı basit.
- Ziyaret edilen Web sayfalarının sırasını korumak için dairesel bağlantılı liste kullanılır. Bu dairesel bağlantılı listede ileri veya geri yönde gezinmek, Geri ve İleri düğmelerini kullanarak sayfaları tekrar ziyaret etmeye yardımcı olur.
- Aslında bu, dairesel yığın veya dairesel kuyruk kullanılarak yapılır.

Dairesel Bağlantılı Listeler

- Listenin ilk düğümünün adresini içeren SONRAKİ girdiyi bulana kadar listeyi dolaşabiliriz.
- Bu, bağlı listenin sonunu belirtir; yani, ilk düğümün adresini içeren düğüm aslında listenin son düğümüdür.
- DATA ve NEXT'i bu şekilde dolaştığımızda, Şekil 6.27'deki bağlı listenin, bir araya getirildiğinde HELLO kelimesini oluşturan karakterleri sakladığını göreceğiz.

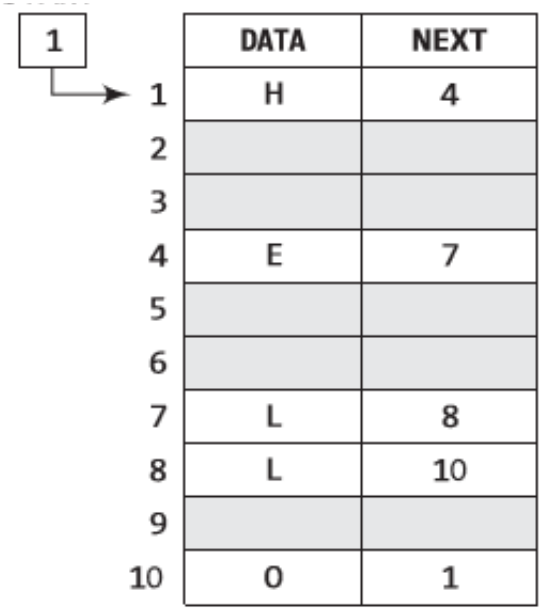
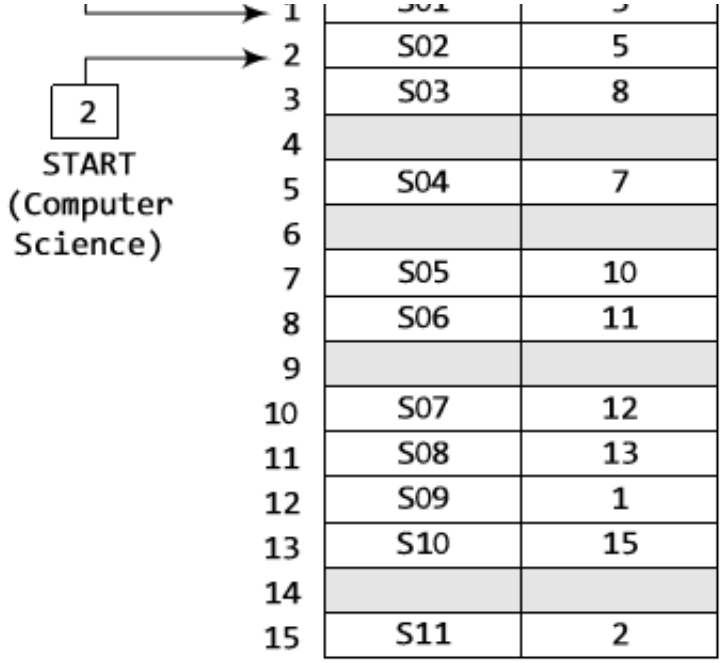


Figure 6.27 Memory representation

Dairesel Bağlantılı Listeler

- Şimdi Şekil 6.28'e bakalım. İki farklı bağlı liste aynı anda bellekte tutuluyor.
- Listede gezinmede herhangi bir belirsizlik yoktur çünkü her liste, ilgili bağlı listenin ilk düğümünün adresini veren ayrı bir BAŞLANGIÇ işaretçisi tutar.
- Geriye kalan düğümlere NEXT'te saklanan değere bakılarak ulaşılır.
- Şekle bakıldığında Biyoloji bölümünü tercih eden öğrencilerin sıra numaralarının S01, S03, S06, S08, S10 ve S11 olduğu sonucuna varılabilir.
- Benzer şekilde Bilgisayar Bilimi'ni seçen öğrencilerin sıra numaraları S02, S04, S05, S07 ve S09'dur.

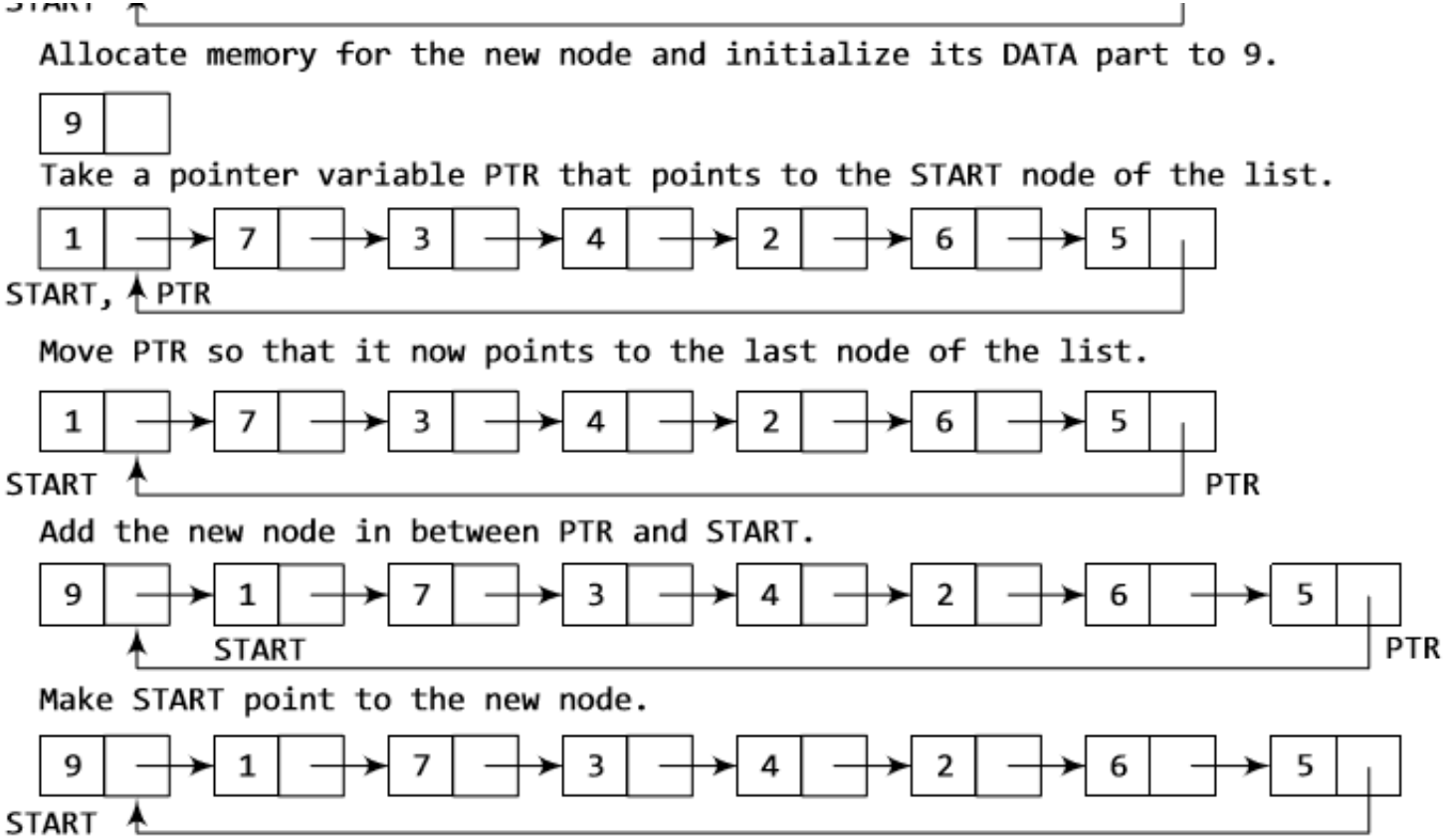


Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listeye yeni bir düğüm ekleme
- Bu bölümde, var olan bir bağlı listeye yeni bir düğümün nasıl eklendiğini göreceğiz.
- İki durumu ele alacağız ve her durumda eklemenin nasıl yapıldığını göreceğiz.
- Durum 1: Yeni düğüm dairesel bağlı listenin başına eklenir.
- Durum 2: Yeni düğüm dairesel bağlı listenin sonuna eklenir.

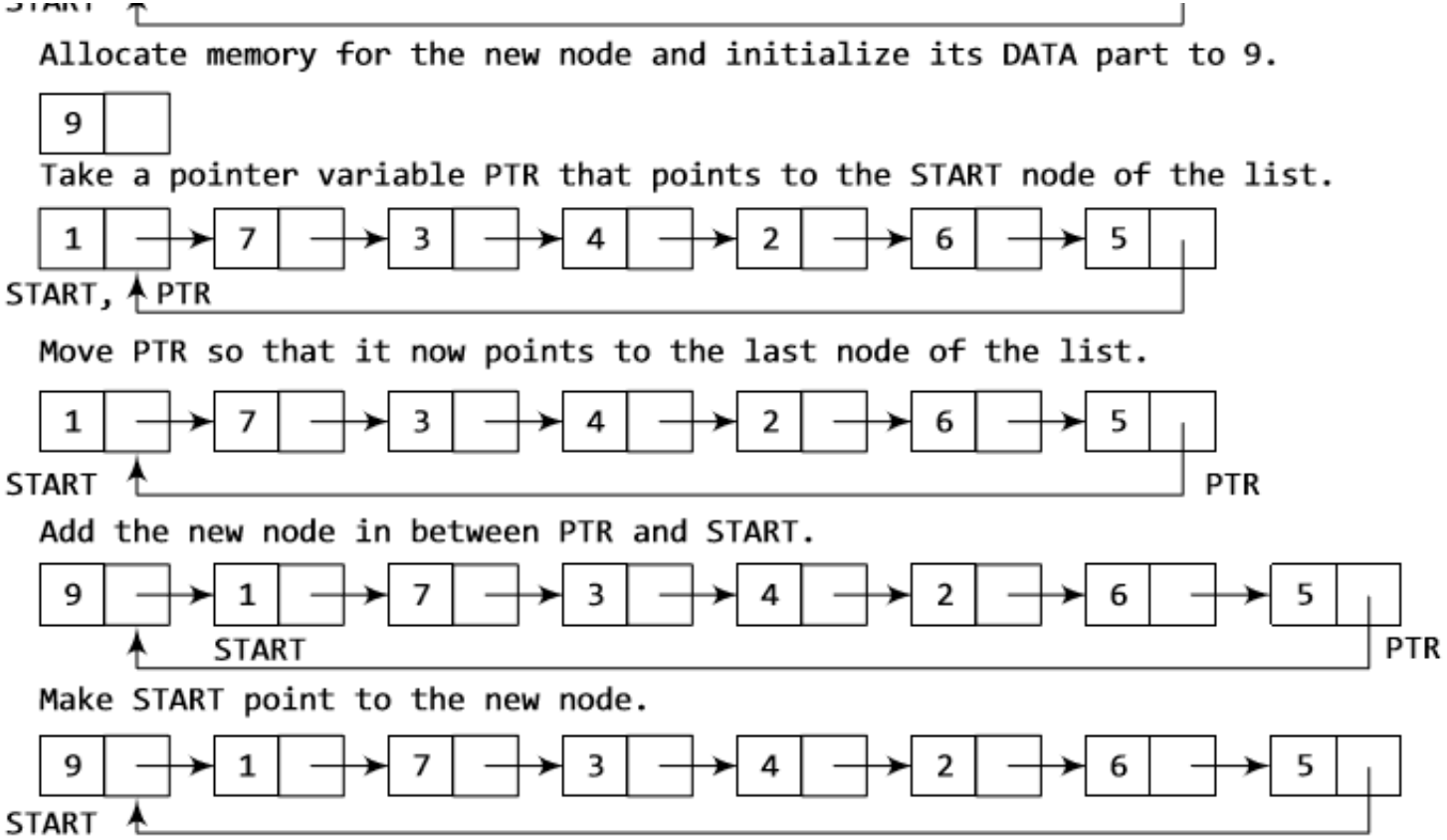
Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.29'da gösterilen bağlantılı listeyi ele alalım.
- Diyelim ki listenin ilk düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.



Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.29'da gösterilen bağlantılı listeyi ele alalım.
- Diyelim ki listenin ilk düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.



Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.30, bağlı listenin başına yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 1. Adımda öncelikle yeni düğüm için bellek olup olmadığını kontrol ediyoruz.
- Boş hafıza tükendiğinde OVERFLOW (TAŞMA) mesajı yazdırılır.
- Aksi takdirde eğer boş hafıza hücresi varsa yeni düğüm için alan ayırırız.
- DATA kısmını verilen VAL ile ayarlayın ve NEXT kısmı START'ta saklanan listenin ilk düğümünün adresi ile başlatılır.
- Artık yeni düğüm listenin ilk düğümü olarak eklendiğinden, artık START düğümü olarak bilinecek, yani START işaretçi değişkeni artık NEW_NODE'un adresini tutacak.
- Dairesel bağlı listeye bir düğüm eklerken listenin son düğümüne gitmek için while döngüsünü kullanmamız gerekir.
- Son düğüm START'a bir işaretçi içerdiğinden, onun NEXT alanı güncellenir; böylece eklemeden sonra artık START olarak bilinecek olan yeni düğüme işaret eder.

Dairesel Bağlantılı Listeler

- **Dairesel Bağlantılı Listenin Başına Bir Düğüm Ekleme**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = START
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

Figure 6.30 Algorithm to insert a new node at the beginning

Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listenin Sonuna Bir Düğüm Ekleme
- Şekil 6.31'de gösterilen bağlantılı listeyi ele alalım.
- Diyelim ki listenin son düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.

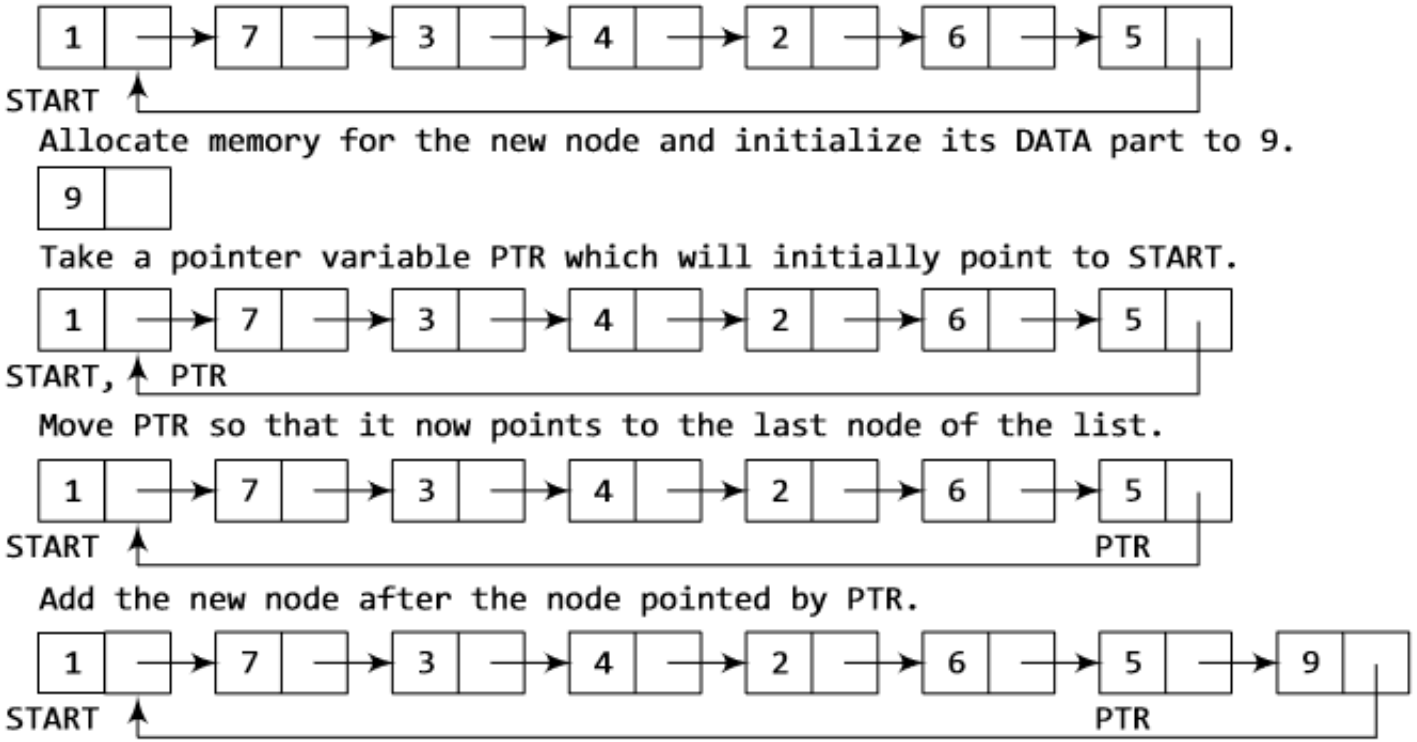


Figure 6.31 Inserting a new node at the end of a circular linked list

Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listenin Sonuna Bir Düğüm Ekleme
- Şekil 6.32 dairesel bağlı listenin sonuna yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 6. Adımda, PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani PTR artık bağlı listenin ilk düğümünü işaret ediyor.
- While döngüsünde bağlı listede dolaşarak son düğüme ulaşırız.
- Son düğüme ulaştığımızda, Adım 9'da, yeni düğümün adresini saklamak için son düğümün NEXT işaretçisini değiştiririz.
- Yeni düğümün NEXT alanının START ile gösterilen ilk düğümün adresini içerdiğini unutmayın.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

Figure 6.32 Algorithm to insert a new node at the end

Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listeden bir düğümün silinmesi
- Bu bölümde, var olan bir dairesel bağlı listeden bir düğümün nasıl silineceğini ele alacağız.
- İki durumu ele alacağız ve her durumda silme işleminin nasıl yapıldığını göreceğiz.
- Silme işlemlerinin geri kalan durumları tek bağlı listeler için verilenlerle aynıdır.
- Durum 1: İlk düğüm silinir.
- Durum 2: Son düğüm silinir.

Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listeden İlk Düğümü Silme
- Şekil 6.33'te gösterilen dairesel bağlantılı listeyi ele alalım.
- Listenin başından bir düğümü silmek istediğimizde, bağlı listede aşağıdaki değişiklikler yapılacaktır.

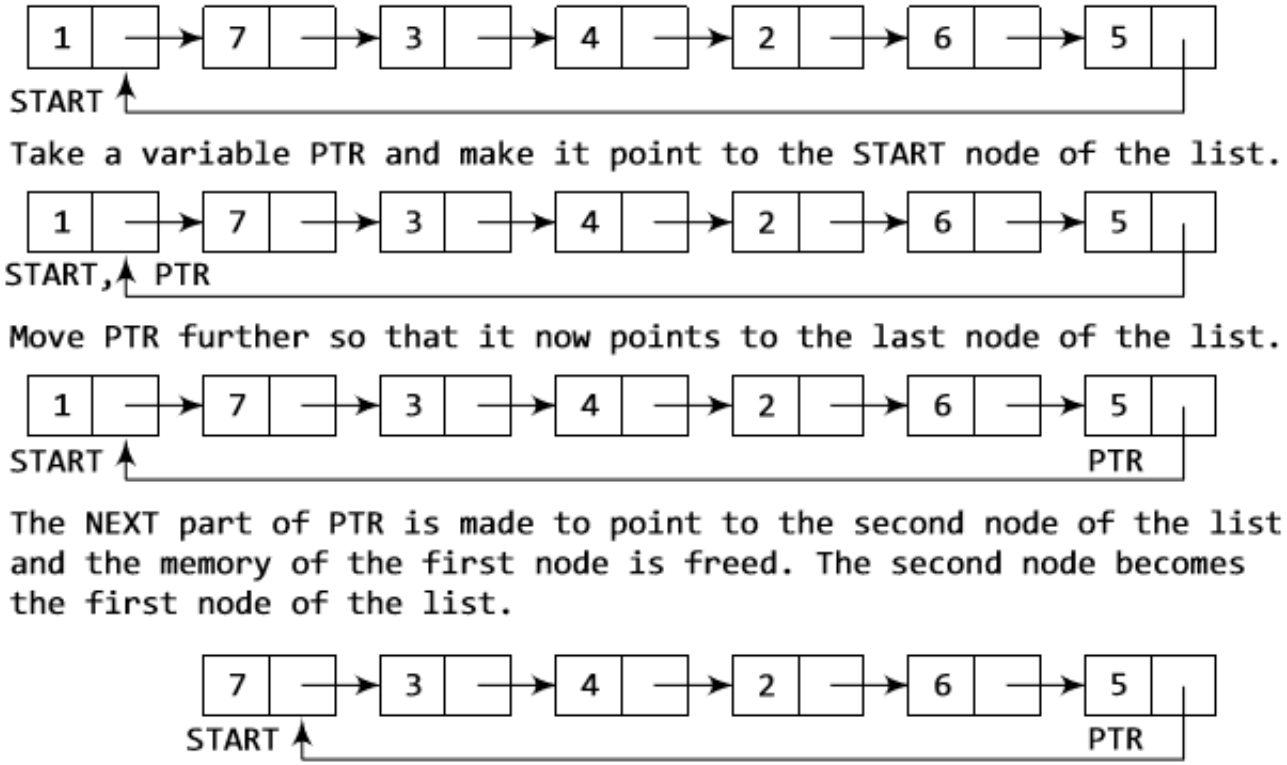


Figure 6.33 Deleting the first node from a circular linked list

Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listeden İlk Düğümü Silme
- Şekil 6.34 dairesel bağlı listeden ilk düğümü silmek için kullanılan algoritmayı göstermektedir.
- Algoritmanın 1. Adımında bağlı listenin var olup olmadığını kontrol ediyoruz.
- START = NULL ise listede düğüm olmadığı anlamına gelir ve kontrol algoritmanın son ifadesine aktarılır.
- Ancak, bağlı listede düğümler varsa, o zaman listeyi dolaşarak en son düğüme ulaşmak için kullanılacak olan PTR işaretçi değişkenini kullanırız.
- 5. Adımda son düğümün next işaretçisini dairesel bağlı listenin ikinci düğümüne işaret edecek şekilde değiştiriyoruz.
- 6. Adımda ilk düğümün işgal ettiği bellek serbest bırakılır.
- Son olarak Adım 7'de ikinci düğüm artık listenin ilk düğümü olur ve adresi START işaretçi değişkeninde saklanır.

Dairesel Bağlantılı Listeler

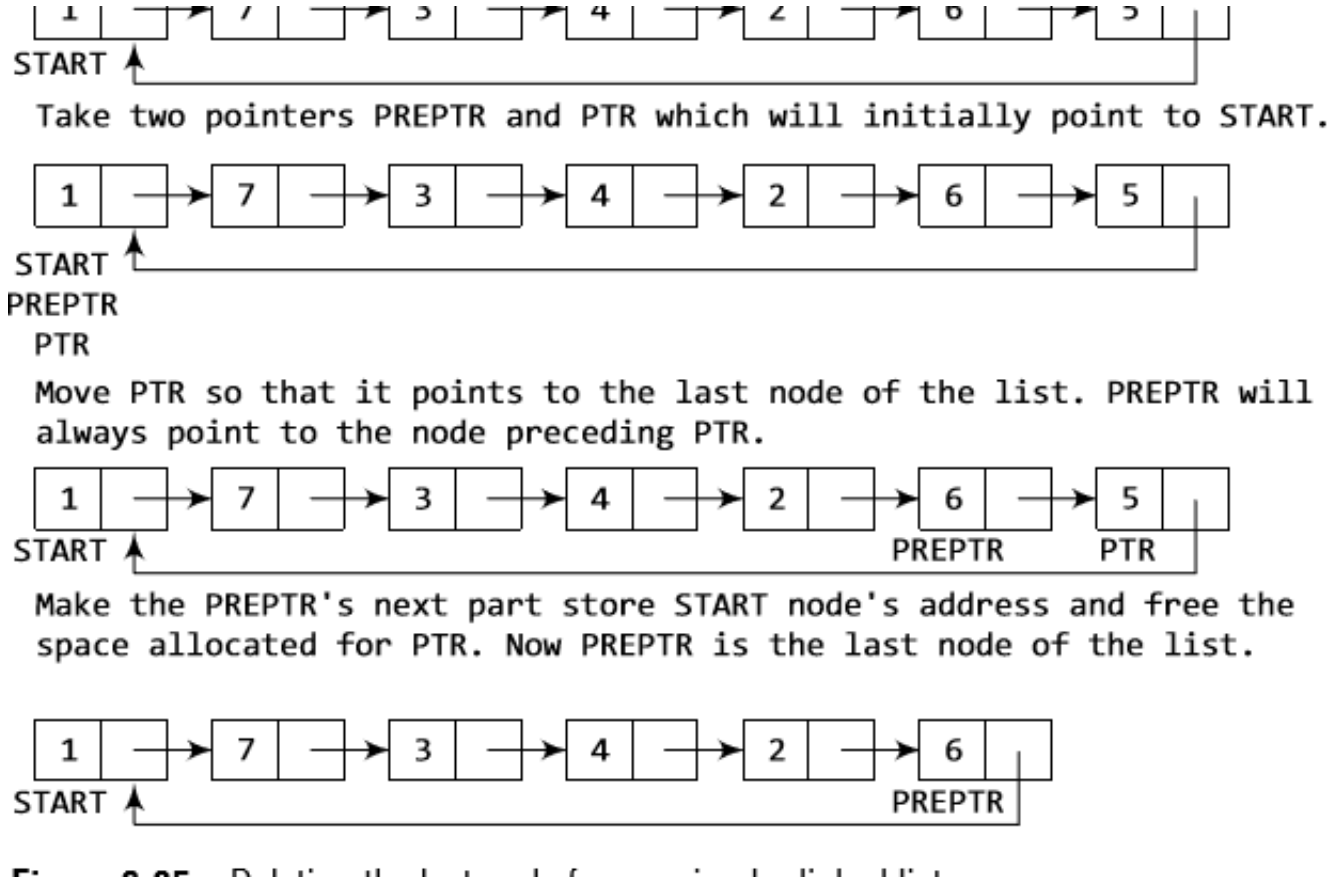
- **Dairesel Bağlantılı Listeden İlk Düğümü Silme**

```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
         [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: FREE START
Step 7: SET START = PTR -> NEXT
Step 8: EXIT
```

Figure 6.34 Algorithm to delete the first node

Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listeden Son Düğümü Silme
- Şekil 6.35'te gösterilen dairesel bağlantılı listeyi ele alalım.
- Bağlı listeden son düğümü silmek istediğimizi varsayalım, bu durumda bağlı listede aşağıdaki değişiklikler yapılacaktır.



Dairesel Bağlantılı Listeler

- Dairesel Bağlantılı Listeden Son Düğümü Silme
- Şekil 6.36 dairesel bağlı listeden son düğümü silmek için kullanılan algoritmayı göstermektedir.
- 2. Adımda PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani PTR artık bağlı listenin ilk düğümünü işaret ediyor.
- While döngüsünde PREPTR'nin her zaman PTR'den önceki bir düğüme işaret etmesini sağlayacak şekilde başka bir işaretçi değişkeni olan PREPTR alırız.
- Son düğüme ve sondan ikinci düğüme ulaştığımızda, sondan ikinci düğümün bir sonraki işaretçisini BAŞLAT olarak ayarlarız, böylece bu artık bağlı listenin (yeni) son düğümü olur.
- Önceki son düğümün belleği serbest bırakılır ve serbest havuza geri döndürülür.

Dairesel Bağlantılı Listeler

- **Dairesel Bağlantılı Listeden Son Düğümü Silme**

```
Step 1: IF START = NULL
                Write UNDERFLOW
                Go to Step 8
           [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START
Step 4:         SET PREPTR = PTR
Step 5:         SET PTR = PTR -> NEXT
           [END OF LOOP]
Step 6: SET PREPTR -> NEXT = START
Step 7: FREE PTR
Step 8: EXIT
```

Figure 6.36 Algorithm to delete the last node

Dairesel Bağlantılı Listeler

```
struct node *display(struct node *start)
{
        struct node *ptr;
        ptr=start;
        while(ptr -> next != start)
        {
                printf("\t %d", ptr -> data);
                ptr = ptr -> next;
        }
        printf("\t %d", ptr -> data);
        return start;
}
```

Dairesel Bağlantılı Listeler

```
ruct node *insert_beg(struct node *start)

      struct node *new_node, *ptr;
      int num;
      printf("\n Enter the data : ");
      scanf("%d", &num);
      new_node = (struct node *)malloc(sizeof(struct node)
      new_node -> data = num;
      ptr = start;
      while(ptr -> next != start)
                ptr = ptr -> next;
      ptr -> next = new_node;
      new_node -> next = start;
      start = new_node;
      return start;
```

Dairesel Bağlantılı Listeler

```
uct node *insert_end(struct node *start)

      struct node *ptr, *new_node;
      int num;
      printf("\n Enter the data : ");
      scanf("%d", &num);
      new_node = (struct node *)malloc(sizeof(struct node
      new_node -> data = num;
      ptr = start;
      while(ptr -> next != start)
                ptr = ptr -> next;
      ptr -> next = new_node;
      new_node -> next = start;
      return start;
```

Dairesel Bağlantılı Listeler

```
struct node *delete_end(struct node *start)
{
        struct node *ptr, *preptr;
        ptr = start;
        while(ptr->next != start)
        {
                preptr = ptr;
                ptr = ptr->next;
        }
        preptr->next = ptr->next;
        free(ptr);
        return start;
}
```

Dairesel Bağlantılı Listeler

```
uct node *delete_after(struct node *start)

      struct node *ptr, *preptr;
      int val;
      printf("\n Enter the value after which the node has to deleted : "
      scanf("%d", &val);
      ptr = start;
      preptr = ptr;
      while(preptr -> data != val)
      {
               preptr = ptr;
               ptr = ptr -> next;
      }
      preptr -> next = ptr -> next;
      if(ptr == start)
               start = preptr -> next;
      free(ptr);
      return start;
```

Dairesel Bağlantılı Listeler

```
uct node *delete_list(struct node *sta

      struct node *ptr;
      ptr = start;
      while(ptr ->next != start)
                  start = delete_end(start
      free(start);
      return start;
```

Çift Bağlantılı Listeler

- Çift yönlü bağlı liste veya iki yönlü bağlı liste, dizideki bir sonraki ve bir önceki düğüme işaretçi içeren, daha karmaşık bir bağlı liste türüdür.
- Bu nedenle, Şekil 6.37'de gösterildiği gibi üç bölümden oluşur: veriler, bir sonraki düğüme işaret eden bir işaretçi ve bir önceki düğüme işaret eden bir işaretçi.

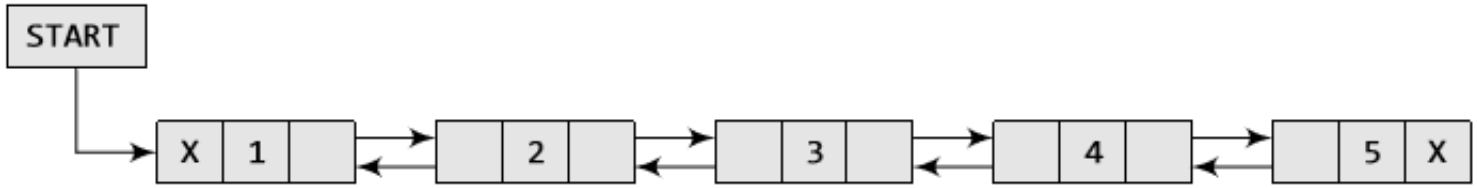


Figure 6.37 Doubly linked list

```
struct node
{
        struct node *prev;
        int data;
        struct node *next;
};
```

Çift Bağlantılı Listeler

- İlk düğümün PREV alanı ve son düğümün NEXT alanı NULL içerecektir.
- PREV alanı, listede geriye doğru dolaşabilmemizi sağlayan önceki düğümün adresini saklamak için kullanılır.
- Böylece, çift yönlü bağlı listenin düğüm başına daha fazla alan ve daha pahalı temel işlemler gerektirdiğini görüyoruz.
- Ancak, çift yönlü bağlı liste, her iki yönde (ileri ve geri) düğümlere işaretçiler sağladığı için listenin elemanlarını değiştirme kolaylığı sağlar.
- Çift bağlantılı liste kullanmanın en büyük avantajı aramayı iki kat daha verimli hale getirmesidir.
- Çift yönlü bağlı listenin hafızada nasıl tutulduğunu görelim.

START 1

	DATA	PREV	NEXT
1	H	−1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	−1

Figure 6.38 Memory representation of a doubly linked list

Çift Bağlantılı Listeler

- Şekilde, ilk düğümün adresini saklamak için bir START değişkeninin kullanıldığını görüyoruz.
- Bu örnekte START = 1 olduğundan ilk veri H olan 1 numaralı adreste saklanır.
- Bu ilk düğüm olduğundan, önceki bir düğümü yoktur ve bu nedenle PREV alanında NULL veya -1 depolar.
- NEXT girişinin –1 veya NULL içerdiği bir konuma ulaşana kadar listeyi dolaşacağız.
- Bu, bağlı listenin sonunu belirtir.
- DATA ve NEXT'i bu şekilde dolaştığımızda, yukarıdaki örnekteki bağlı listenin bir araya getirildiğinde HELLO kelimesini oluşturan karakterleri sakladığını göreceğiz.

Çift Bağlantılı Listeler

- Çift Bağlantılı Listeye yeni bir düğüm ekleme
- Bu bölümde, var olan çift yönlü bağlı listeye yeni bir düğümün nasıl eklendiğini ele alacağız.
- Dört vakayı ele alacağız ve her vakada eklemenin nasıl yapıldığını göreceğiz.
- Durum 1: Yeni düğüm başa eklenir.
- Durum 2: Yeni düğüm sona eklenir.
- Durum 3: Yeni düğüm, belirli bir düğümden sonra eklenir.
- Durum 4: Yeni düğüm, belirli bir düğümden önce eklenir.

Çift Bağlantılı Listeler

- Çift Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.39'da gösterilen çift bağlantılı listeyi ele alalım.
- Diyelim ki listenin ilk düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.

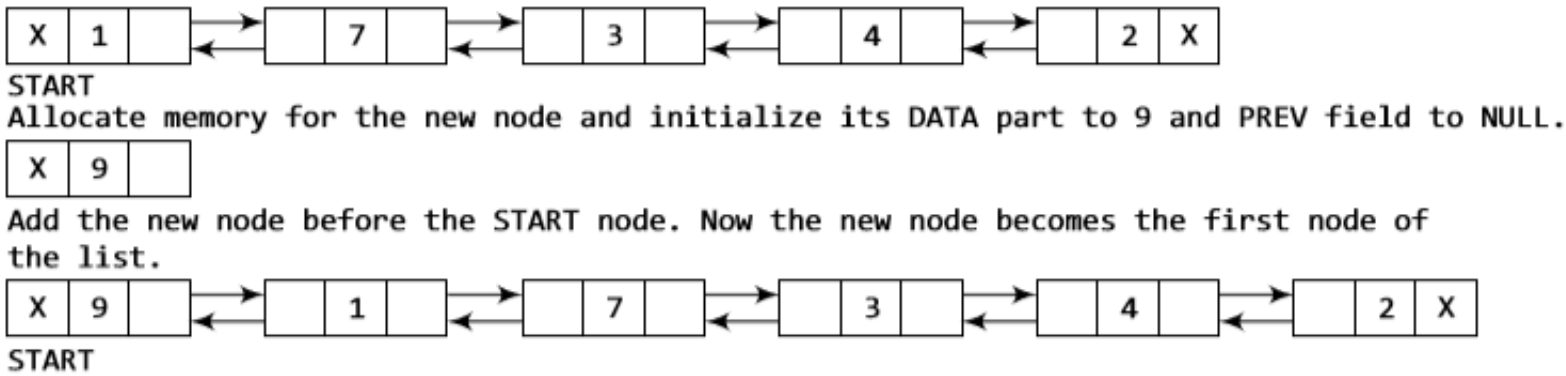


Figure 6.39 Inserting a new node at the beginning of a doubly linked list

Çift Bağlantılı Listeler

- Çift Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.40, çift yönlü bağlı listenin başına yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- Adım 1'de, öncelikle yeni düğüm için belleğin kullanılabilir olup olmadığını kontrol ederiz. Boş bellek tükendiyse, bir OVERFLOW mesajı yazdırılır.
- Aksi takdirde eğer boş hafıza hücresi varsa yeni düğüm için alan ayırırız.
- DATA kısmını verilen VAL ile ayarlayın ve NEXT kısmı START'ta saklanan listenin ilk düğümünün adresi ile başlatılır.
- Artık yeni düğüm listenin ilk düğümü olarak eklendiğinden, artık START düğümü olarak bilinecek, yani START işaretçi değişkeni artık NEW_NODE'un adresini tutacak.

```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

Figure 6.40 Algorithm to insert a new node at the beginning

Çift Bağlantılı Listeler

- Çift Bağlantılı Listenin Sonuna Bir Düğüm Ekleme
- Şekil 6.41'de gösterilen çift bağlantılı listeyi ele alalım.
- Diyelim ki listenin son düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.

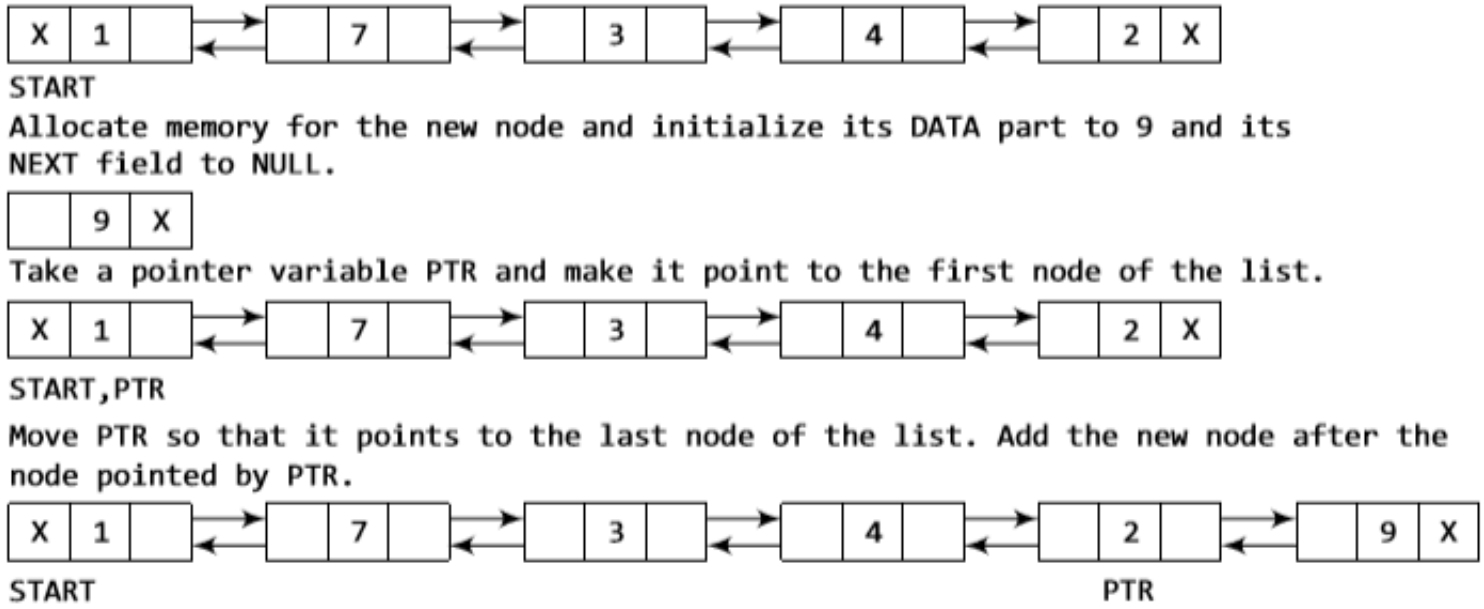


Figure 6.41 Inserting a new node at the end of a doubly linked list

Çift Bağlantılı Listeler

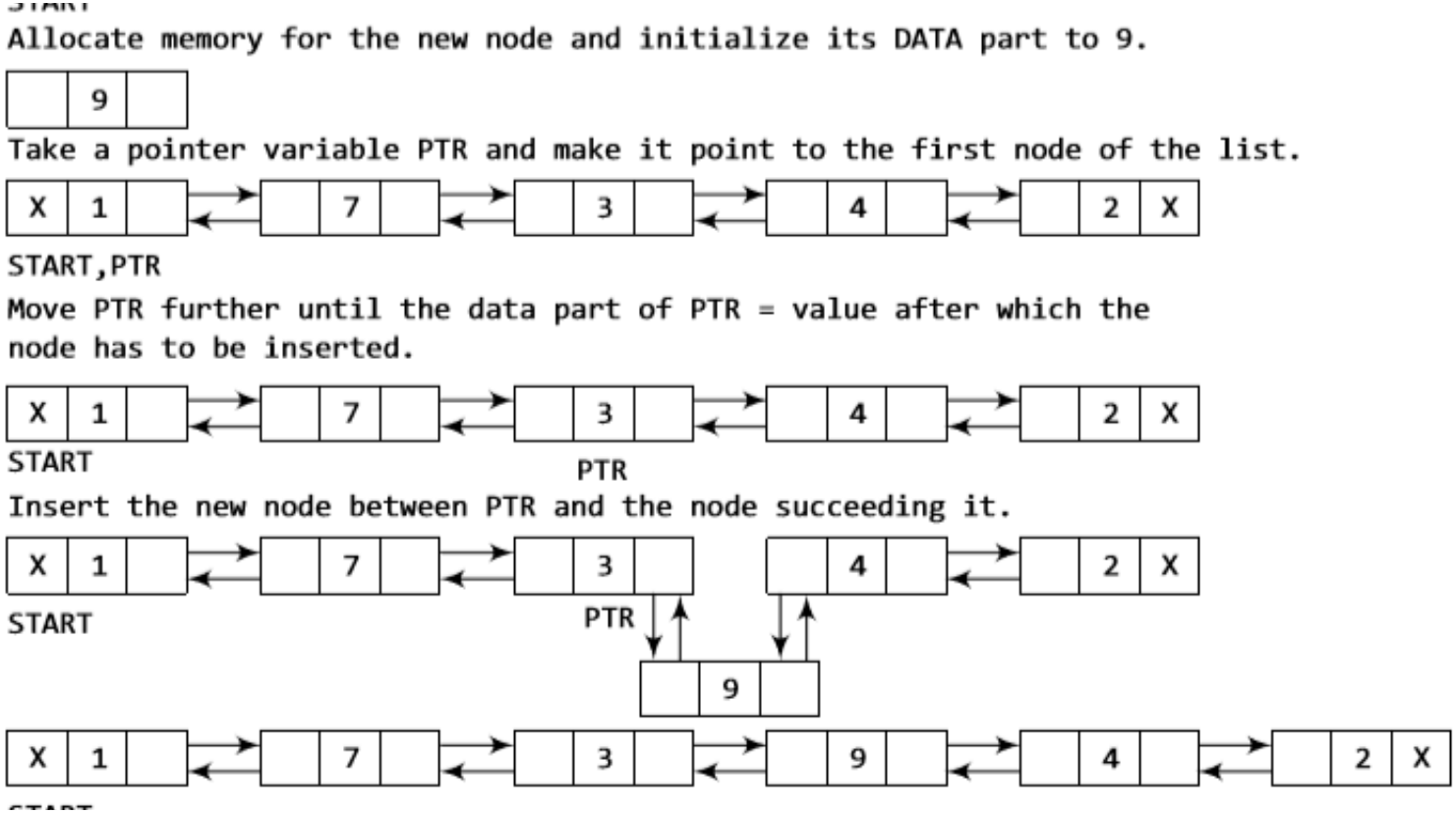
- Çift Bağlantılı Listenin Sonuna Bir Düğüm Ekleme
- Şekil 6.42, çift bağlantılı bir listenin sonuna yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 6. Adımda, bir işaretçi değişkeni PTR alırız ve onu START ile başlatırız. While döngüsünde, son düğüme ulaşmak için bağlı listeyi dolaşıyoruz.
- Son düğüme ulaştığımızda, Adım 9'da, yeni düğümün adresini saklamak için son düğümün NEXT işaretçisini değiştiririz.
- Yeni düğümün NEXT alanının, bağlı listenin sonunu belirten NULL değerini içerdiğini unutmayın.
- NEW_NODE'un PREV alanı, PTR'nin işaret ettiği düğümü (şimdi listenin sondan ikinci düğümü) i

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: EXIT
```

Figure 6.42 Algorithm to insert a new node at the end

Çift Bağlantılı Listeler

- Çift Bağlantılı Listede Verilen Bir Düğümden Sonra Bir Düğüm Ekleme
- Şekil 6.44'te gösterilen çift bağlantılı listeyi ele alalım.
- 3 değerini içeren düğümün ardından 9 değerine sahip yeni bir düğüm eklemek istediğimizi varsayalım.
- Bağlantılı listede yapılacak değişiklikleri tartışmadan önce, Şekil 6.43'te gösterilen algoritmaya bakalım.



Çift Bağlantılı Listeler

- Çift Bağlantılı Listede Verilen Bir Düğümden Sonra Bir Düğüm Ekleme
- Şekil 6.43, çift bağlantılı bir listedeki belirli bir düğümden sonra yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 5. Adımda bir PTR işaretçisi alıyoruz ve onu START ile başlatıyoruz.
- Yani PTR artık bağlı listenin ilk düğümünü işaret ediyor.
- While döngüsünde, bağlı listede dolaşarak değeri NUM olan düğüme ulaşırız.
- Yeni düğüm bu düğümden sonra ekleneceği için bu düğüme ulaşmamız gerekiyor.
- Bu düğüme ulaştığımızda NEXT ve PREV alanlarını, yeni düğümün istenilen düğümden sonra eklenmesini sağlayacak şekilde değiştiriyoruz.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET PTR -> NEXT = NEW_NODE
Step 11: SET PTR -> NEXT -> PREV = NEW_NODE
Step 12: EXIT
```

Figure 6.43 Algorithm to insert a new node after a given node

Çift Bağlantılı Listeler

- Çift Bağlantılı Bir Listede Verilen Bir Düğümden Önce Bir Düğüm Ekleme
- Şekil 6.46'da gösterilen çift bağlantılı listeyi ele alalım.
- Diyelim ki 3 değerini içeren düğümden önce 9 değerine sahip yeni bir düğüm eklemek istiyoruz.
- Bağlantılı listede yapılacak değişiklikleri tartışmadan önce, Şekil 6.45'te gösterilen algoritmaya bakalım.
- 1. Adımda öncelikle yeni düğüm için bellek olup olmadığını kontrol ediyoruz.
- 5. Adımda, PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani PTR artık bağlı listenin ilk düğümünü işaret ediyor.
- While döngüsünde, bağlı listede dolaşarak değeri NUM olan düğüme ulaşırız.
- Yeni düğüm bu düğümden önce ekleneceği için bu düğüme ulaşmamız gerekiyor.
- Bu düğüme ulaştığımızda NEXT ve PREV alanlarını, yeni düğümün istenilen düğümden önce eklenmesini sağlayacak şekilde değiştiriyoruz.

Çift Bağlantılı Listeler

- **Çift Bağlantılı Bir Listede Verilen Bir Düğümden Önce Bir Düğüm Ekleme**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR
Step 9: SET NEW_NODE -> PREV = PTR -> PREV
Step 10: SET PTR -> PREV = NEW_NODE
Step 11: SET PTR  -> PREV -> NEXT = NEW_NODE
Step 12: EXIT
```

Figure 6.45 Algorithm to insert a new node before a given node

Çift Bağlantılı Listeler

- **Çift Bağlantılı Bir Listede Verilen Bir Düğümden Önce Bir Düğüm Ekleme**

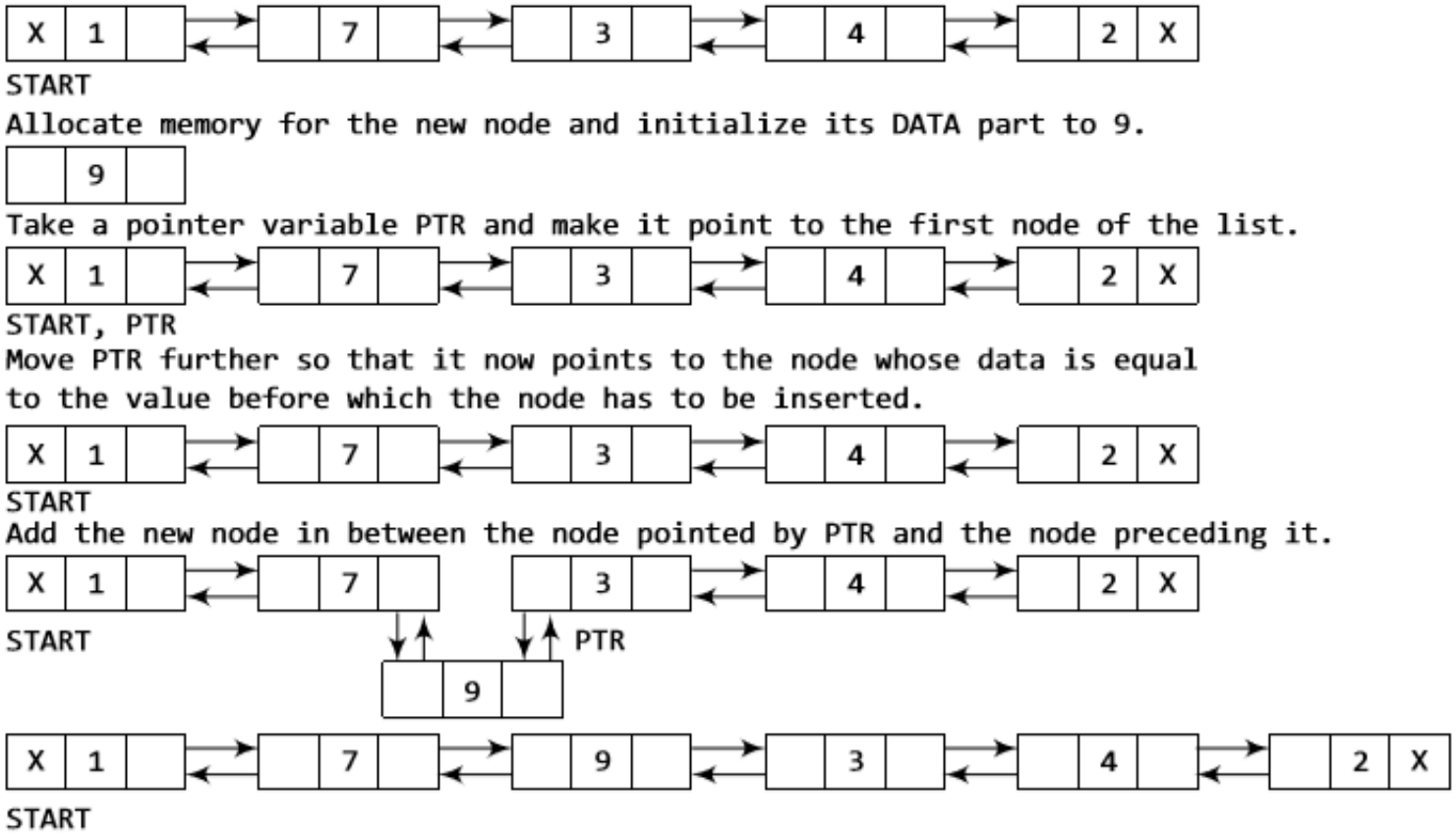


Figure 6.46 Inserting a new node before a given node in a doubly linked list

Çift Bağlantılı Listeler

- Çift Bağlantılı Listeden bir düğümü silme
- Bu bölümde, var olan bir çift yönlü bağlı listeden bir düğümün nasıl silineceğini göreceğiz.
- Dört durumu ele alacağız ve her durumda silme işleminin nasıl yapıldığını göreceğiz.
- Durum 1: İlk düğüm silinir.
- Durum 2: Son düğüm silinir.
- Durum 3: Belirli bir düğümden sonraki düğüm silinir.
- Durum 4: Belirli bir düğümden önceki düğüm silinir.

Çift Bağlantılı Listeler

- Çift Bağlantılı Bir Listeden İlk Düğümü Silme
- Şekil 6.47'de gösterilen çift bağlantılı listeyi ele alalım.
- Listenin başından bir düğümü silmek istediğimizde, bağlı listede aşağıdaki değişiklikler yapılacaktır.

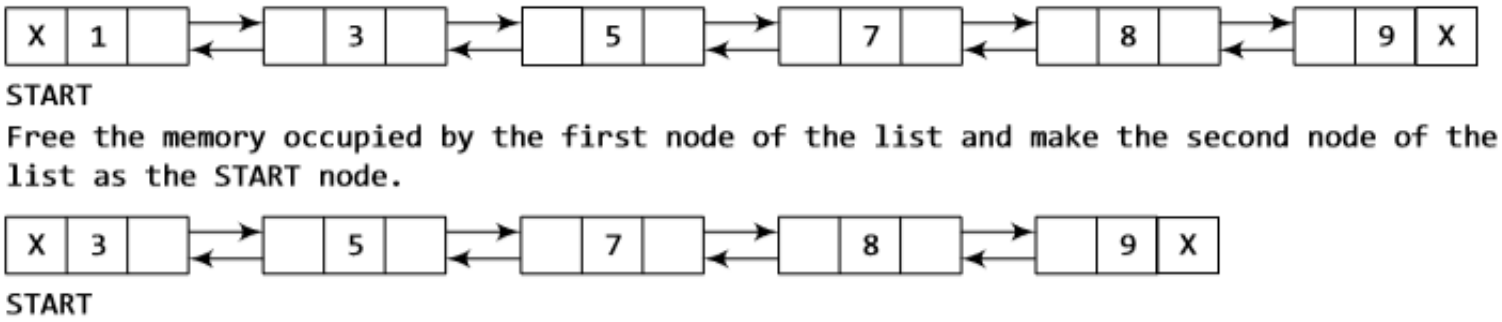


Figure 6.47 Deleting the first node from a doubly linked list

Çift Bağlantılı Listeler

- Çift Bağlantılı Bir Düğümden İlk Düğümü Silme
- Şekil 6.48, çift yönlü bağlı bir listenin ilk düğümünü silmek için kullanılan algoritmayı göstermektedir.
- Algoritmanın 1. Adımında bağlı listenin var olup olmadığını kontrol ediyoruz.
- START = NULL ise listede düğüm olmadığı anlamına gelir ve kontrol algoritmanın son ifadesine aktarılır.
- Ancak, bağlı listede düğümler varsa, listenin ilk düğümünü işaret edecek şekilde ayarlanmı geçici bir işaretçi değişkeni PTR kullanırız.
- Bunun için PTR'yi listenin ilk düğümünün adresini saklayan START ile başlatıyoruz.
- Adım 3'te START'ın sıradaki düğüme işaret etmesi sağlanır ve son olarak PTR'nin (başlangıçta listenin ilk düğümü) işgal ettiği bellek serbest bırakılır ve serbest havuza geri döndürülür.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: SET START -> PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

Figure 6.48 Algorithm to delete the first node

Çift Bağlantılı Listeler

- Çift Bağlantılı Bir Listeden Son Düğümü Silme
- Şekil 6.49'da gösterilen çift bağlantılı listeyi ele alalım.
- Bağlı listeden son düğümü silmek istediğimizi varsayalım, bu durumda bağlı listede aşağıdaki değişiklikler yapılacaktır.

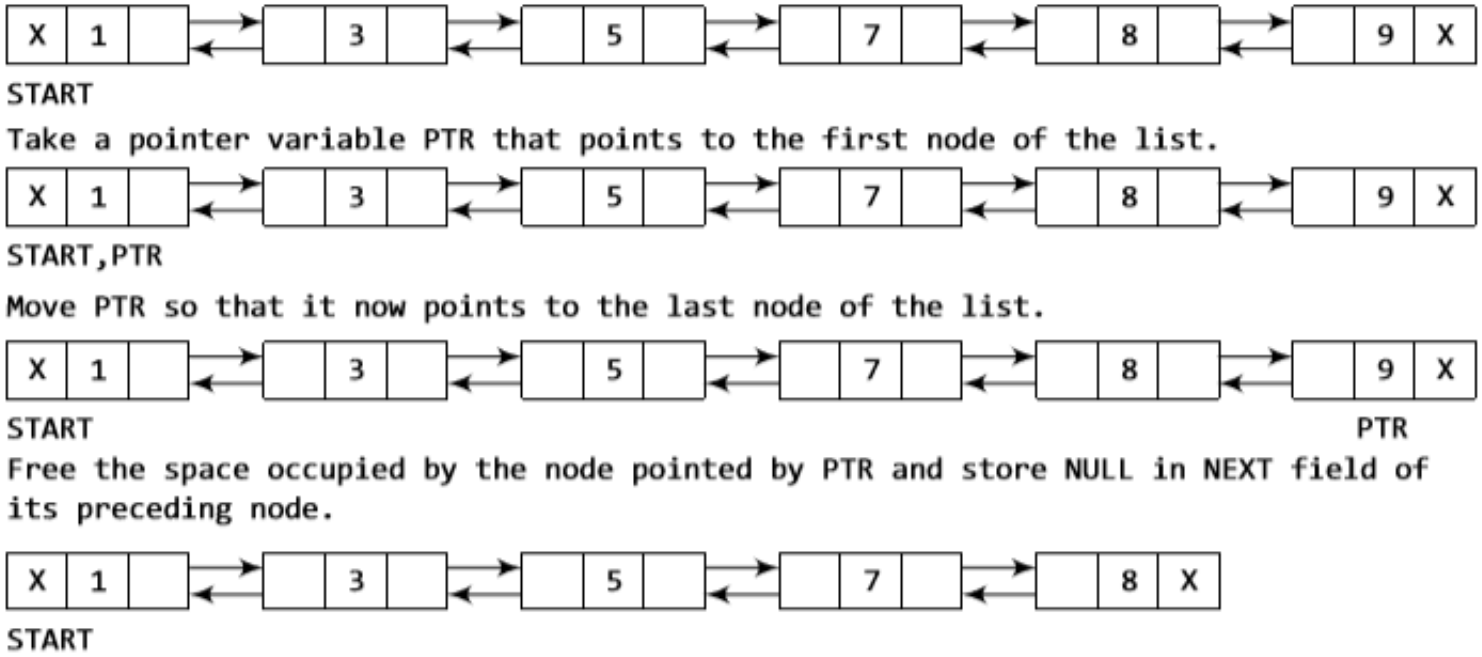


Figure 6.49 Deleting the last node from a doubly linked list

Çift Bağlantılı Listeler

- Çift Bağlantılı Bir Listeden Son Düğümü Silme
- Şekil 6.50, çift bağlantılı bir listenin son düğümünü silmek için kullanılan algoritmayı göstermektedir.
- 2. Adımda PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani PTR artık bağlı listenin ilk düğümünü işaret ediyor.
- While döngüsü, son düğüme ulaşmak için listeyi dolaşır.
- Son düğüme ulaştığımızda, son düğümün PREV alanından adresini alarak ikinci son düğüme de erişebiliriz.
- Son düğümü silmek için, ikinci son düğümün bir sonraki alanını NULL olarak ayarlamamız yeterlidir, böylece bu artık bağlı listenin (yeni) son düğümü olur.
- Önceki son düğümün belleği serbest bırakılır ve serbest havuza geri döndürülür.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != NULL
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

Figure 6.50 Algorithm to delete the last node

Çift Bağlantılı Listeler

- Çift Bağlantılı Listede Belirli Bir Düğümden Sonraki Düğümü Silme
- Şekil 6.51'de gösterilen çift bağlantılı listeyi ele alalım.
- Diyelim ki 4 veri değerini içeren düğümün ardından gelen düğümü silmek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.

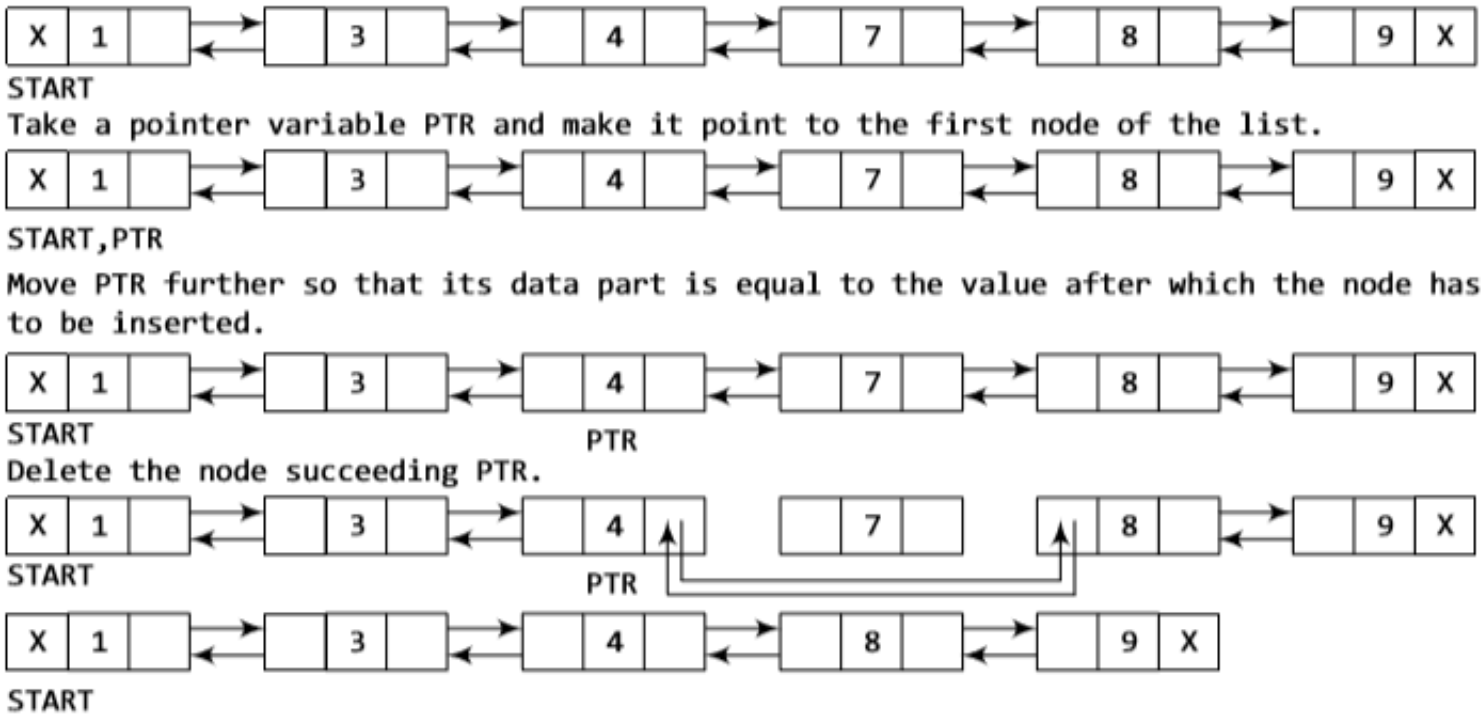


Figure 6.51 Deleting the node after a given node in a doubly linked list

Çift Bağlantılı Listeler

- Çift Bağlantılı Listede Belirli Bir Düğümden Sonraki Düğümü Silme
- Şekil 6.52, çift yönlü bağlı bir listenin belirli bir düğümünden sonraki düğümü silmek için kullanılan algoritmayı göstermektedir.
- 2. Adımda PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani PTR artık çift bağlantılı listenin ilk düğümünü işaret ediyor.
- While döngüsü, verilen düğüme ulaşmak için bağlı listeyi dolaşır.
- VAL'i içeren düğüme ulaştığımızda, onu takip eden düğüme NEXT alanında saklanan adresi kullanarak kolayca erişilebilir.
- Belirtilen düğümün NEXT alanı, sonraki düğümün NEXT alanındaki içerikleri içerecek şekilde ayarlanır.
- Son olarak, belirtilen düğümden sonraki düğümün belleği serbest bırakılır ve serbest hav[illegible]

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->DATA != NUM
Step 4:       SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR->NEXT
Step 6: SET PTR->NEXT = TEMP->NEXT
Step 7: SET TEMP->NEXT->PREV = PTR
Step 8: FREE TEMP
Step 9: EXIT
```

Figure 6.52 Algorithm to delete a node after a given node

Çift Bağlantılı Listeler

- Çift Bağlantılı Listede Belirli Bir Düğümden Önceki Düğümü Silme
- Şekil 6.53'te gösterilen çift bağlantılı listeyi ele alalım.
- Diyelim ki değeri 4 olan düğümden önceki düğümü silmek istiyoruz.
- Bağlı listede yapılacak değişiklikleri tartışmadan önce algoritmaya bakalım.

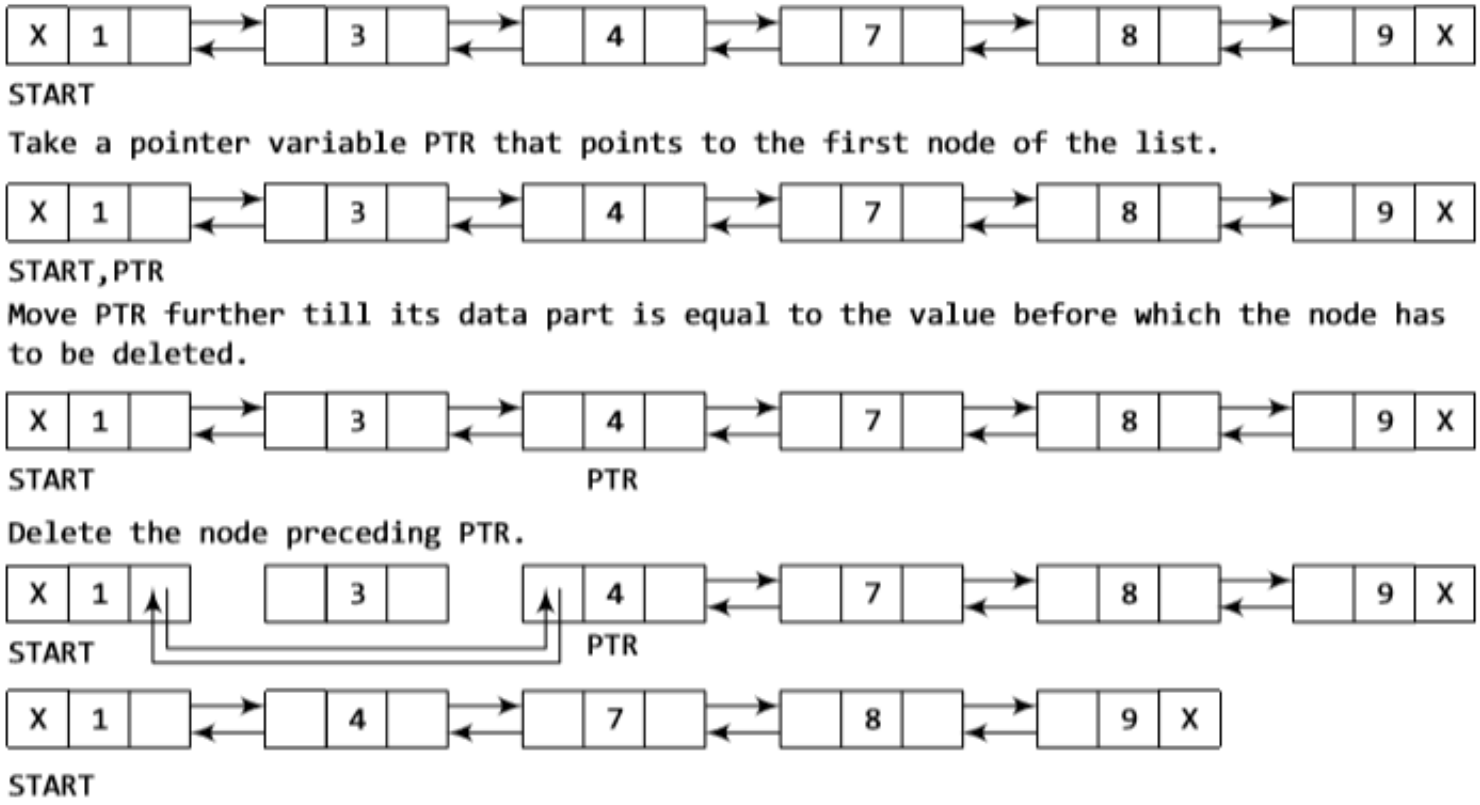


Figure 6.53 Deleting a node before a given node in a doubly linked list

Çift Bağlantılı Listeler

- Çift Bağlantılı Listede Belirli Bir Düğümden Önceki Düğümü Silme
- Şekil 6.54, çift yönlü bağlı bir listenin belirli bir düğümünden önceki düğümü silmek için kullanılan algoritmayı göstermektedir.
- Adım 2'de, bir işaretçi değişkeni PTR alırız ve onu START ile başlatırız. Yani, PTR artık bağlı listenin ilk düğümünü işaret eder.
- While döngüsü, bağlı listeyi dolaşarak istenilen düğüme ulaşır.
- VAL içeren düğüme ulaştığımızda, PTR'nin PREV alanı, PTR'den önce gelen düğümün öncesindeki düğümün adresini içerecek şekilde ayarlanır.
- PTR'den önceki düğümün belleği serbest bırakılır ve serbest havuza geri döndürülür. Bu nedenle, yalnızca o düğümün adresi verildiğinde sabit sayıda işlemde bir düğümü ekleyebileceğimizi veya silebileceğimizi görürüz.
- Aynı işlemin yapılabilmesi için önceki düğümün adresinin de gerekli olduğu tek bağlantılı listelerde bunun mümkün olmadığını unutmayın.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> PREV
Step 6: SET TEMP -> PREV -> NEXT = PTR
Step 7: SET PTR -> PREV = TEMP -> PREV
Step 8: FREE TEMP
Step 9: EXIT
```

Figure 6.54 Algorithm to delete a node before a given node

Çift Bağlantılı Listeler

```
struct node *create_ll(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter -1 to end");
        printf("\n Enter the data : ");
        scanf("%d", &num);
        while(num != -1)
        {
                if(start == NULL)
                {
                        new_node = (struct node*)malloc(sizeof(struct node));
                        new_node -> prev = NULL;
                        new_node -> data = num;
                        new_node -> next = NULL;
                        start = new_node;
                }
                else
                {
                        ptr=start;
                        new_node = (struct node*)malloc(sizeof(struct node));
                        new_node->data=num;
                        while(ptr->next!=NULL)
                                ptr = ptr->next;
                        ptr->next = new_node;
                        new_node->prev=ptr;
                        new_node->next=NULL;
                }
                printf("\n Enter the data : ");
                scanf("%d", &num);
        }
        return start;
}
```

Çift Bağlantılı Listeler

```
struct node *display(struct node *start)
{
        struct node *ptr;
        ptr=start;
        while(ptr!=NULL)
        {
                printf("\t %d", ptr -> data);
                ptr = ptr -> next;
        }
        return start;
}
```

Çift Bağlantılı Listeler

```
struct node *insert_end(struct node *start)
{
        struct node *ptr, *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr=start;
        while(ptr -> next != NULL)
                ptr = ptr -> next;
        ptr -> next = new_node;
        new_node -> prev = ptr;
        new_node -> next = NULL;
        return start;
}
```

Çift Bağlantılı Listeler

```
struct node *insert_before(struct node *start)
{
        struct node *new_node, *ptr;
        int num, val;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        printf("\n Enter the value before which the data has to be inserted:");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = num;
        ptr = start;
        while(ptr->data != val)
                ptr = ptr->next;
        new_node->next = ptr;
        new_node->prev = ptr->prev;
        ptr->prev->next = new_node;
        ptr->prev = new_node;
        return start;
}
```

Çift Bağlantılı Listeler

```
struct node *delete_beg(struct node *start)
{
        struct node *ptr;
        ptr = start;
        start = start -> next;
        start -> prev = NULL;
        free(ptr);
        return start;
}
struct node *delete_end(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr -> next != NULL)
                ptr = ptr -> next;
        ptr -> prev -> next = NULL;
        free(ptr);
        return start;
}
```

Çift Bağlantılı Listeler

```
truct node *delete_after(struct node *start)

        struct node *ptr, *temp;
        int val;
        printf("\n Enter the value after which the node has to deleted : "
        scanf("%d", &val);
        ptr = start;
        while(ptr->data != val)
                ptr = ptr->next;
        temp = ptr->next;
        ptr->next = temp->next;
        temp->next->prev = ptr;
        free(temp);
        return start;
```

Dairesel Çift Bağlantılı Listeler

- Dairesel çift yönlü bağlı liste veya dairesel iki yönlü bağlı liste, dizideki bir sonraki ve bir önceki düğüme işaretçi içeren, daha karmaşık bir bağlı liste türüdür.
- Çift bağlantılı liste ile dairesel çift bağlantılı liste arasındaki fark, tek bağlantılı liste ile dairesel bağlantılı liste arasındaki farka benzer.
- Dairesel çift bağlantılı liste, ilk düğümün önceki alanında ve son düğümün sonraki alanında NULL içermez.
- Bunun yerine, son düğümün bir sonraki alanı, listenin ilk düğümünün adresini, yani BAŞLAT'ı depolar.
- Benzer şekilde, ilk alanın önceki alanı son düğümün adresini depolar. Dairesel çift bağlantılı bir liste Şekil 6.55'te gösterilmiştir.

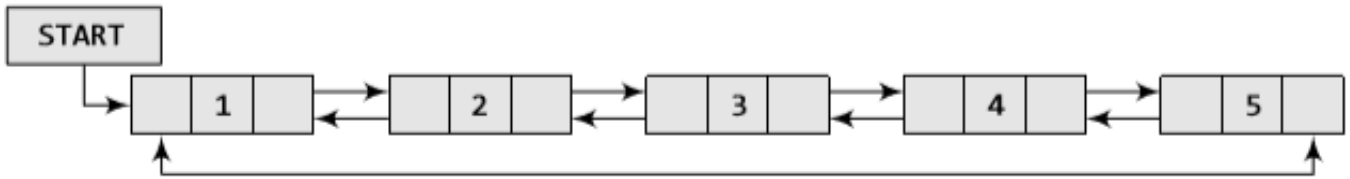


Figure 6.55 Circular doubly linked list

Dairesel Çift Bağlantılı Listeler

- Dairesel çift bağlı liste yapısında üç parçadan oluştuğu için düğüm başına daha fazla alan ve daha pahalı temel işlemler gerektirir.
- Ancak dairesel çift bağlantılı liste, her iki yönde (ileri ve geri) düğümlere işaretçiler sağladığı için listenin elemanlarını değiştirme kolaylığı sağlar
- Dairesel çift bağlantılı liste kullanmanın en büyük avantajı arama işlemini iki kat daha verimli hale getirmesidir.
- Dairesel çift bağlantılı bir listenin bellekte nasıl tutulduğunu görelim. Şekil 6.56'yı ele alalım.

START: 1

	DATA	PREV	Next
1	H	9	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	0	7	1

Figure 6.56 Memory representation of a circular doubly linked list

Dairesel Çift Bağlantılı Listeler

- Şekilde, ilk düğümün adresini saklamak için bir START değişkeninin kullanıldığını görüyoruz.
- Bu örnekte START = 1 olduğundan ilk veri H olan 1 numaralı adreste saklanır.
- Bu ilk düğüm olduğundan, listenin son düğümünün adresini bir önceki alanında saklar.
- İlgili NEXT, bir sonraki düğümün adresi olan 3'ü depolar.
- O halde bir sonraki veri öğesini almak için 3 numaralı adrese bakacağız.
- Önceki alan ilk düğümün adresini içerecektir.
- Adres 3'ten elde edilen ikinci veri elemanı E'dir.
- Listenin ilk elemanının adresinin NEXT girişinde saklandığı noktaya ulaşana kadar bu işlemi tekrarlıyoruz.
- Bu, bağlı listenin sonunu belirtir; yani, ilk düğümün adresini içeren düğüm aslında listenin son düğümüdür.

Dairesel Çift Bağlantılı Listeler

- Dairesel çift bağlantılı listeye yeni bir düğüm ekleme
- Bu bölümde, var olan dairesel çift bağlı listeye yeni bir düğümün nasıl eklendiğini göreceğiz.
- İki durumu ele alacağız ve her durumda eklemenin nasıl yapıldığını göreceğiz.
- Geri kalan durumlar çift bağlı listeler için verilenlere benzerdir.
- Durum 1: Yeni düğüm başa eklenir.
- Durum 2: Yeni düğüm sona eklenir.

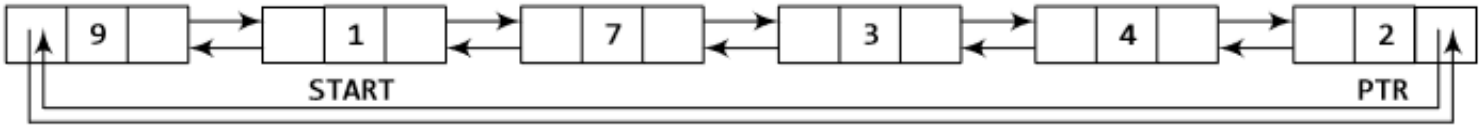
Dairesel Çift Bağlantılı Listeler

- Dairesel Çift Bağlantılı Listenin Başına Bir Düğüm Ekleme
- Şekil 6.57'de gösterilen dairesel çift bağlantılı listeyi ele alalım.
- Diyelim ki listenin ilk düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.

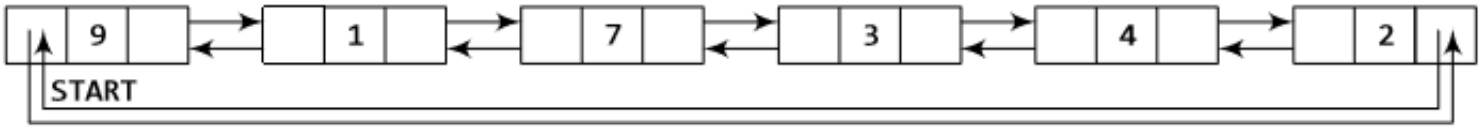
Allocate memory for the new node and initialize its DATA part to 9.

Take a pointer variable PTR that points to the first node of the list.

Move PTR so that it now points to the last node of the list. Insert the new node in between PTR and the START node.



START will now point to the new node.



Dairesel Çift Bağlantılı Listeler

- Şekil 6.58 dairesel çift bağlantılı listenin başına yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 1. Adımda öncelikle yeni düğüm için bellek olup olmadığını kontrol ediyoruz.
- Boş hafıza tükendiğinde OVERFLOW (TAŞMA) mesajı yazdırılır.
- Aksi takdirde, yeni düğüm için alan ayırırız. Veri kısmını verilen VAL ile ayarlayın ve bir sonraki kısmı, START'ta saklanan listenin ilk düğümünün adresiyle başlatılır.
- Artık yeni düğüm listenin ilk düğümü olarak eklendiğinden, artık START düğümü olarak bilinecek, yani START işaretçi değişkeni artık NEW_NODE'un adresini tutacak.
- Dairesel çift bağlantılı liste olduğundan NEW_NODE'un PREV alanı son düğümün adresini içerecek şekilde ayarlanır.

Dairesel Çift Bağlantılı Listeler

- **Dairesel Çift Bağlantılı Listenin Başına Bir Düğüm Ekleme**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 13
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET PTR -> NEXT = NEW_NODE
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET NEW_NODE -> NEXT = START
Step 11: SET START -> PREV = NEW_NODE
Step 12: SET START = NEW_NODE
Step 13: EXIT
```

Figure 6.58 Algorithm to insert a new node at the beginning

Dairesel Çift Bağlantılı Listeler

- Dairesel Çift Bağlantılı Listenin Sonuna Bir Düğüm Ekleme
- Şekil 6.59'da gösterilen dairesel çift bağlantılı listeyi ele alalım.
- Diyelim ki listenin son düğümüne 9 verisi olan yeni bir düğüm eklemek istiyoruz.
- Daha sonra bağlı listede aşağıdaki değişiklikler yapılacaktır.

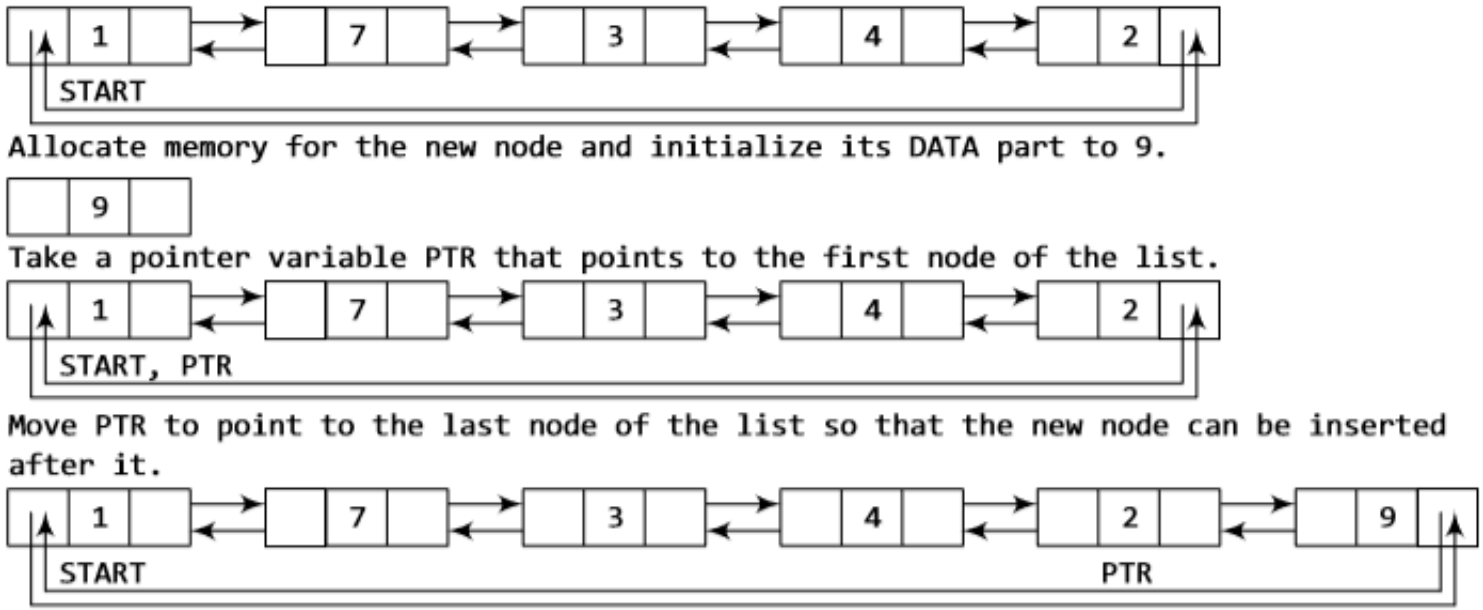


Figure 6.59 Inserting a new node at the end of a circular doubly linked list

Dairesel Çift Bağlantılı Listeler

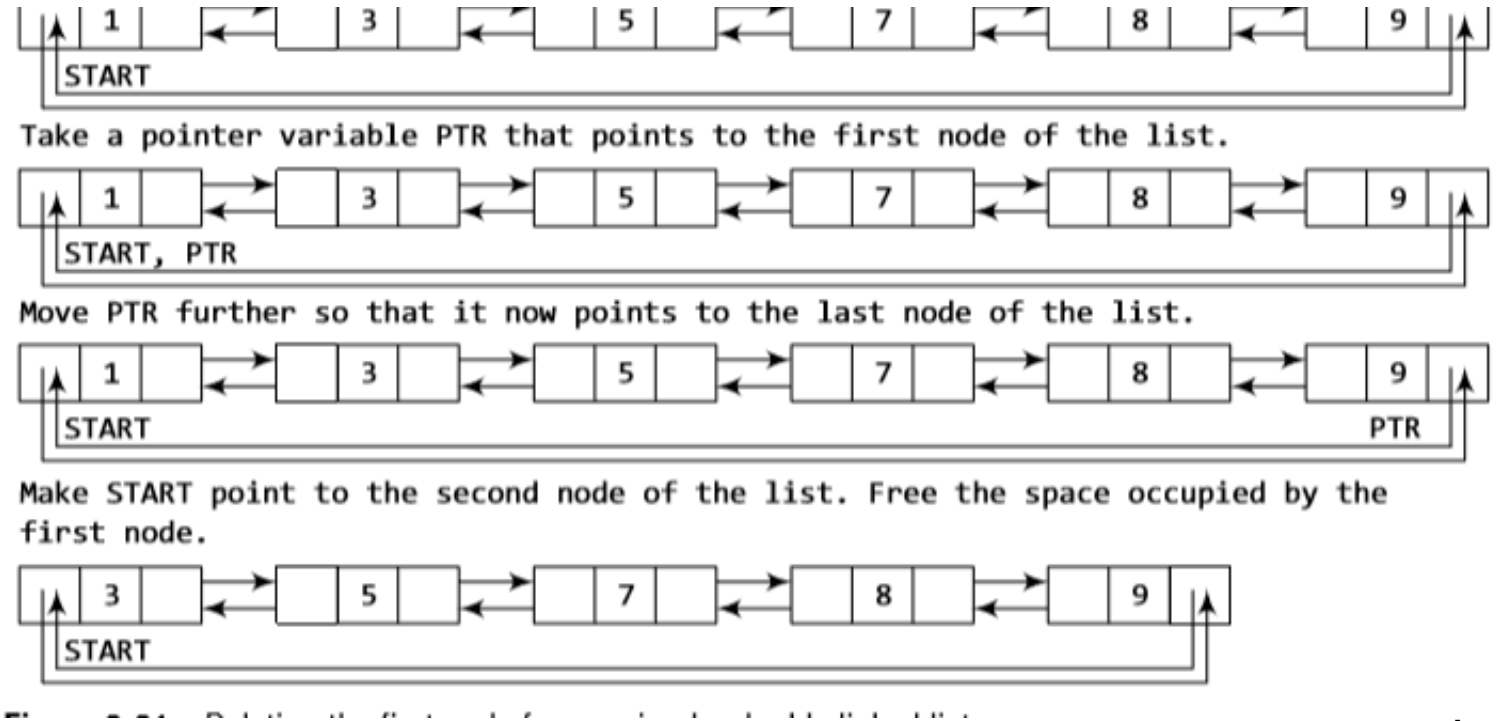
- Dairesel Çift Bağlantılı Listenin Sonuna Bir Düğüm Ekleme
- Şekil 6.60 dairesel çift bağlantılı listenin sonuna yeni bir düğüm eklemek için kullanılan algoritmayı göstermektedir.
- 6. Adımda, PTR işaretçi değişkenini alıp START ile başlatıyoruz.
- Yani, PTR artık bağlı listenin ilk düğümüne işaret ediyor. While döngüsünde, son düğüme ulaşmak için bağlı listeyi dolaşıyoruz.
- Son düğüme ulaştığımızda, Adım 9'da, yeni düğümün adresini saklamak için son düğümün NEXT işaretçisini değiştiririz.
- NEW_NODE'un PREV alanı, PTR'nin işaret ettiği düğümü (şimdi listenin sondan ikinci düğümü) işaret edecek şekilde ayarlanacaktır.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: SET START -> PREV = NEW_NODE
Step 12: EXIT
```

Figure 6.60 Algorithm to insert a new node at the end

Dairesel Çift Bağlantılı Listeler

- Dairesel Çift Bağlantılı Listeden İlk Düğümü Silme
- Şekil 6.61'de gösterilen dairesel çift bağlantılı listeyi ele alalım.
- Listenin başından bir düğümü silmek istediğimizde, bağlı listede aşağıdaki değişiklikler yapılacaktır.



Dairesel Çift Bağlantılı Listeler

- Dairesel Çift Bağlantılı Listeden Son Düğümün Silinmesi
- Şekil 6.63'te gösterilen dairesel çift bağlantılı listeyi ele alalım.
- Bağlı listeden son düğümü silmek istediğimizi varsayalım, bu durumda bağlı listede aşağıdaki değişiklikler yapılacaktır.

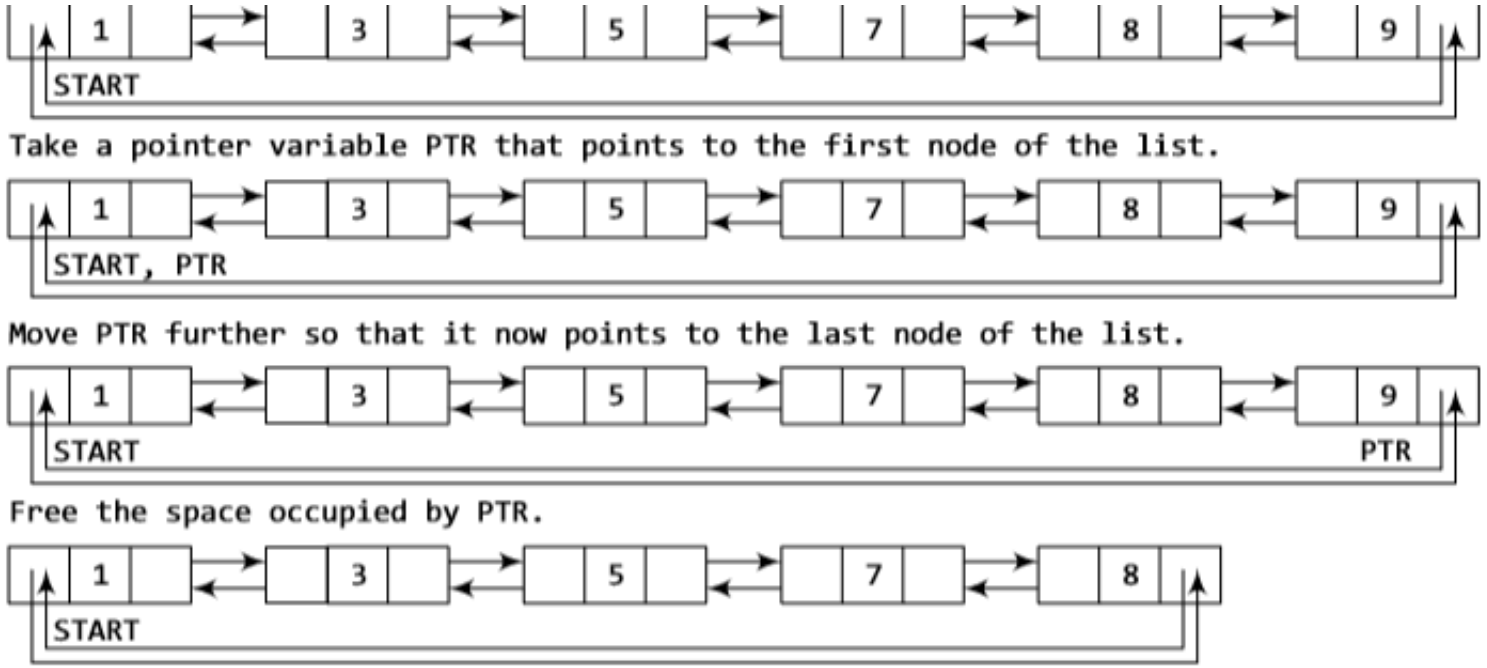


Figure 6.63 Deleting the last node from a circular doubly linked list

Dairesel Çift Bağlantılı Listeler

```
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter -1 to end");
        printf("\n Enter the data : ");
        scanf("%d", &num);
        while(num != -1)
        {
                if(start == NULL)
                {
                        new_node = (struct node*)malloc(sizeof(struct node));
                        new_node->prev = NULL;
                        new_node->data = num;
                        new_node->next = start;
                        start = new_node;
                }
                else
                {
                        new_node = (struct node*)malloc(sizeof(struct node));
                        new_node->data = num;
                        ptr = start;
                        while(ptr->next != start)
                                ptr = ptr->next;
                        new_node->prev = ptr;
                        ptr->next = new_node;
                        new_node->next = start;
                        start->prev = new_node;
                }
                printf("\n Enter the data : ");
                scanf("%d", &num);
        return start;
}
```

Dairesel Çift Bağlantılı Listeler

```
struct node *insert_beg(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = num;
        ptr = start;
        while(ptr->next != start)
                ptr = ptr->next;
        new_node->prev = ptr;
        ptr->next = new_node;
        new_node->next = start;
        start->prev = new_node;
        start = new_node;
        return start;
}
```

Dairesel Çift Bağlantılı Listeler

```
struct node *delete_node(struct node *start)
{
        struct node *ptr;
        int val;
        printf("\n Enter the value of the node which has to be deleted : ");
        scanf("%d", &val);
        ptr = start;
        if(ptr->data == val)
        {
                start = delete_beg(start);
                return start;
        }
        else
        {
                while(ptr->data != val)
                        ptr = ptr->next;
                ptr->prev->next = ptr->next;
                ptr->next->prev = ptr->prev;
                free(ptr);
                return start;
        }
}
```

Başlık Bağlantılı Listeler

- Başlık bağlantılı liste, listenin başında bir başlık düğümü bulunan özel bir bağlantılı liste türüdür.
- Yani, bir başlık bağlantılı listede START, listenin ilk düğümüne işaret etmeyecek, ancak START başlık düğümünün adresini içerecektir.
- Başlık bağlantılı listesinin iki çeşidi şunlardır:
  - Son düğümün bir sonraki alanına NULL depolayan topraklanmış başlık bağlantılı listesi.
  - Son düğümün bir sonraki alanında başlık düğümünün adresini depolayan dairesel başlık bağlantılı liste. Burada, başlık düğümü listenin sonunu belirtecektir.
- Her iki tür başlık bağlantılı listeyi gösteren Şekil 6.65'e bakın.

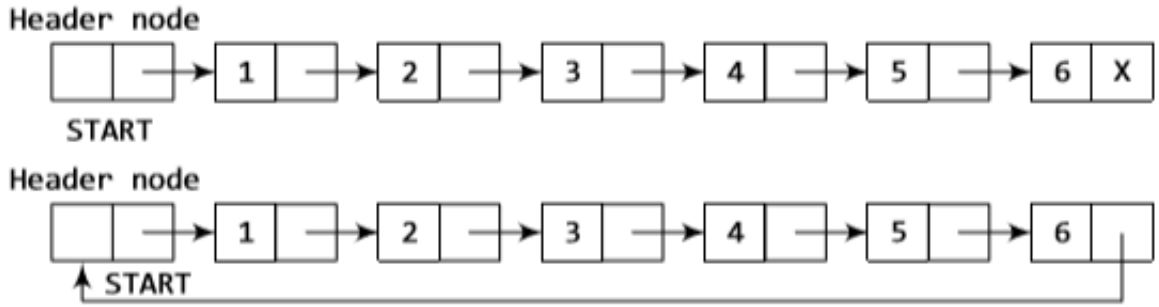


Figure 6.65 Header linked list

Başlık Bağlantılı Listeler

- Diğer bağlı listelerde olduğu gibi eğer START = NULL ise bu boş başlıklı bağlı listeyi gösterir.
- Grounded header bağlı listesinin hafızada nasıl saklandığını görelim.
- Topraklanmış başlıklı bağlı listede bir düğümün DATA ve NEXT olmak üzere iki alanı vardır.
- DATA alanı bilgi kısmını depolayacak ve NEXT alanı düğümün adresini sırayla depolayacaktır. Şekil 6.66'yı düşünün.

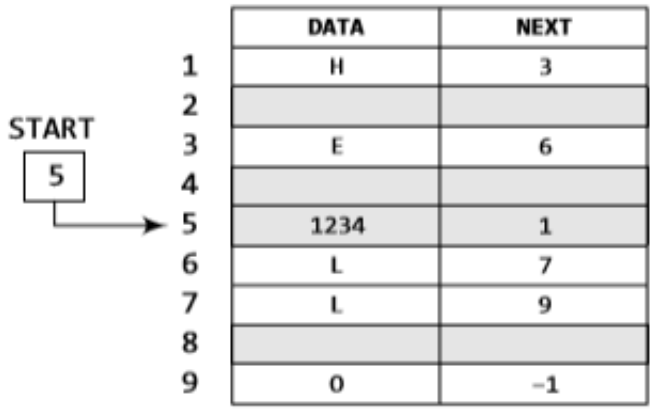


Figure 6.66 Memory representation of a header linked list

Başlık Bağlantılı Listeler

- START'ın başlık düğümünün adresini sakladığını unutmayın.
- Gölgeli satır bir başlık düğümünü belirtir.
- Başlık düğümünün NEXT alanı, listenin ilk düğümünün adresini depolar.
- Bu düğüm H'yi depolar.
- İlgili NEXT alanı bir sonraki düğümün adresini, yani 3'ü depolar.
- O halde bir sonraki veri öğesini almak için 3 numaralı adrese bakacağız.
- Dolayısıyla, ilk düğüme FIRST_NODE = START -> NEXT yazılarak erişilebileceğini, START = FIRST_NODE yazılarak erişilemeyeceğini görüyoruz.
- Bunun nedeni, START'ın başlık düğümüne, başlık düğümünün de başlık bağlı listesinin ilk düğümüne işaret etmesidir.

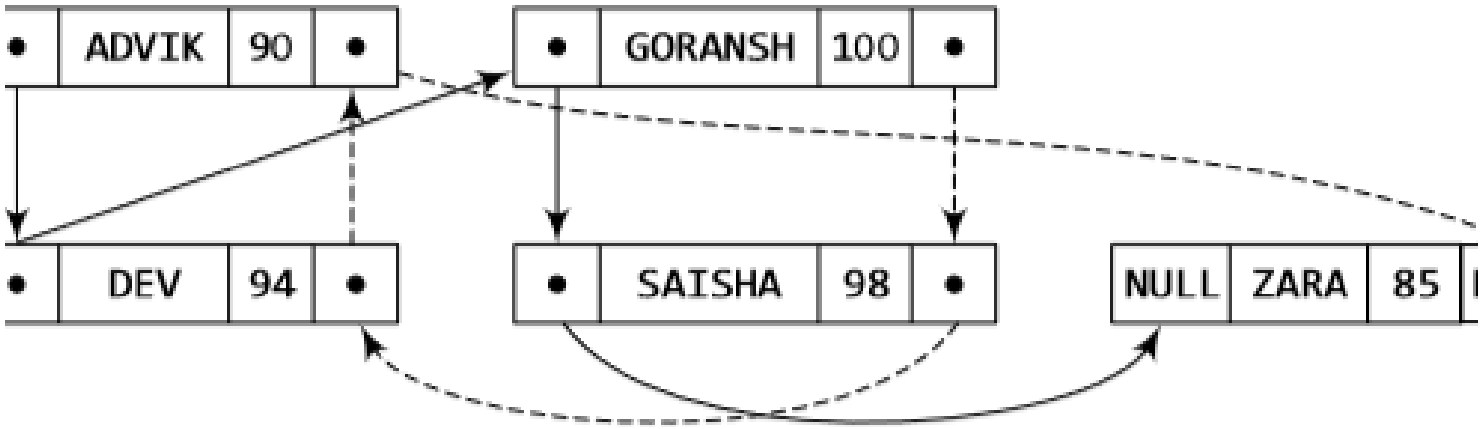
Çoklu Bağlantılı Listeler

- Çoklu bağlantılı bir listede, her düğüm diğer düğümlere n sayıda işaretçi gönderebilir.
- Çift bağlantılı liste, çoklu bağlantılı listelerin özel bir halidir.
- Ancak, çift bağlantılı listelerden farklı olarak, çok bağlantılı listedeki düğümlerin her işaretçi için tersi olabilir veya olmayabilir.
- Çift bağlantılı listeyi çoklu bağlantılı listeden iki şekilde ayırt edebiliriz:
- (a) Çift yönlü bağlı listenin tam olarak iki işaretçisi vardır.
- Bir işaretçi bir önceki düğümü, diğeri ise bir sonraki düğümü işaret eder.
- Ancak çoklu bağlantılı listedeki bir düğüm herhangi bir sayıda işaretçiye sahip olabilir.
- (b) Çift yönlü bağlı listede işaretçiler birbirinin tam tersidir, yani önceki bir düğümü işaret eden her işaretçi için sonraki düğümü işaret eden bir işaretçi vardır.
- Çoklu bağlantılı listeler için bu durum geçerli değildir.

Çoklu Bağlantılı Listeler

- Çoklu bağlantılı listeler genellikle bir eleman kümesinin birden fazla sırasını düzenlemek için kullanılır.
- Örneğin, bir sınıftaki öğrencilerin adlarını ve aldıkları notları saklayan bir bağlı listemiz varsa, listenin düğümlerini iki şekilde organize edebiliriz:
- (i) Düğümleri alfabetik olarak (isimlerine göre) düzenleyin
- (ii) Düğümleri, en yüksek notu alan öğrencinin bilgisinin diğerlerinden önce gelmesi için notların azalan sırasına göre düzenleyin.
- Şekil 6.71, öğrencilerin düğümlerinin yukarıda belirtilen her iki yolla da düzenlendiği çoklu bağlantılı bir listeyi göstermektedir.

Çoklu Bağlantılı Listeler



Multi-linked list that stores names alphabetically as well as according to c
order of marks

Bağlantılı Listelerin Uygulamaları

- Polinom gösterimi
- Bir polinomun bellekte bağlantılı liste kullanılarak nasıl temsil edildiğini görelim.
- $6x^3 + 9x^2 + 7x + 1$ polinomunu ele alalım.
- Bir polinomdaki her bir terim, bir katsayı ve bir kuvvet olmak üzere iki kısımdan oluşur.
- Burada 6, 9, 7 ve 1 sırasıyla kuvvetleri 3, 2, 1 ve 0 olan terimlerin katsayılarıdır.
- Bir polinomun her terimi, bağlı listenin bir düğümü olarak gösterilebilir.
- Şekil 6.74 yukarıdaki polinomun terimlerinin bağlantılı gösterimini göstermektedir.

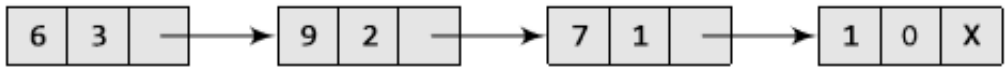


Figure 6.74 Linked representation of a polynomial

Bağlantılı Listelerin Uygulamaları

- **Polinom gösterimi**

```
        scanf("%d", &n);
        printf("\t Enter its coefficient : ");
        scanf("%d", &c);
        while(n != -1)
        {
                if(start==NULL)
                {
                        new_node = (struct node *)malloc(sizeof(struct node));
                        new_node->num = n;
                        new_node->coeff = c;
                        new_node->next = NULL;
                        start = new_node;
                }
                else
                {
                        ptr = start;
                        while(ptr->next != NULL)
                                ptr = ptr->next;
                        new_node = (struct node *)malloc(sizeof(struct node));
                        new_node->num = n;
                        new_node->coeff = c;
                        new_node->next = NULL;
                        ptr->next = new_node;
                }
                printf("\n Enter the number : ");
                scanf("%d", &n);
                if(n == -1)
                        break;
```