

YAZILIM MÜHENDİSLİĞİNE GİRİŞ

HAFTA 8 VERİ SOYUTLAMA

Dr. Öğr. Üyesi Ebru Aydoğan
eaydogan@mehmetakif.edu.tr

Ders İçeriği: Hafta 8

- Giriş
- Temel Veri Yapıları
- Genel Kavramlar
- Veri Yapılarının Gerçekleştirilmesi
- Örnek Bir Durum Çalışması
- Özelleştirilmiş Veri Tipleri
- Sınıflar ve Nesneler

Giriş

- Sayı, karakter, kelime vb. türdeki verileri doğrudan bellek içerisinde kaydetmek için veri tipleri yeterlidir.
- Peki, bazı kavramsal yapıları doğrusal bir hafızada nasıl saklarız?
- Örneğin bir dizi veya matris belleğe nasıl yerleştirilebilir?
- Veri soyutlama (Veri yapıları) kullanılarak:
 - Ana amaç, veriye doğrudan bilgisayar ana bellek organizasyonu ile değil de soyut bir araç halinde gereksiz ayrıntılarla uğraşmaya gerek kalmadan ulaşabilmek.
- Bu bölümde veri yapılarını gerçekleştirmenin etkili yolları incelenecektir.
- Gerçekleştirme derken, bu veri türlerinin bir bilgisayarın bellek hücrelerine eşlenmesini kastedilmektedir.

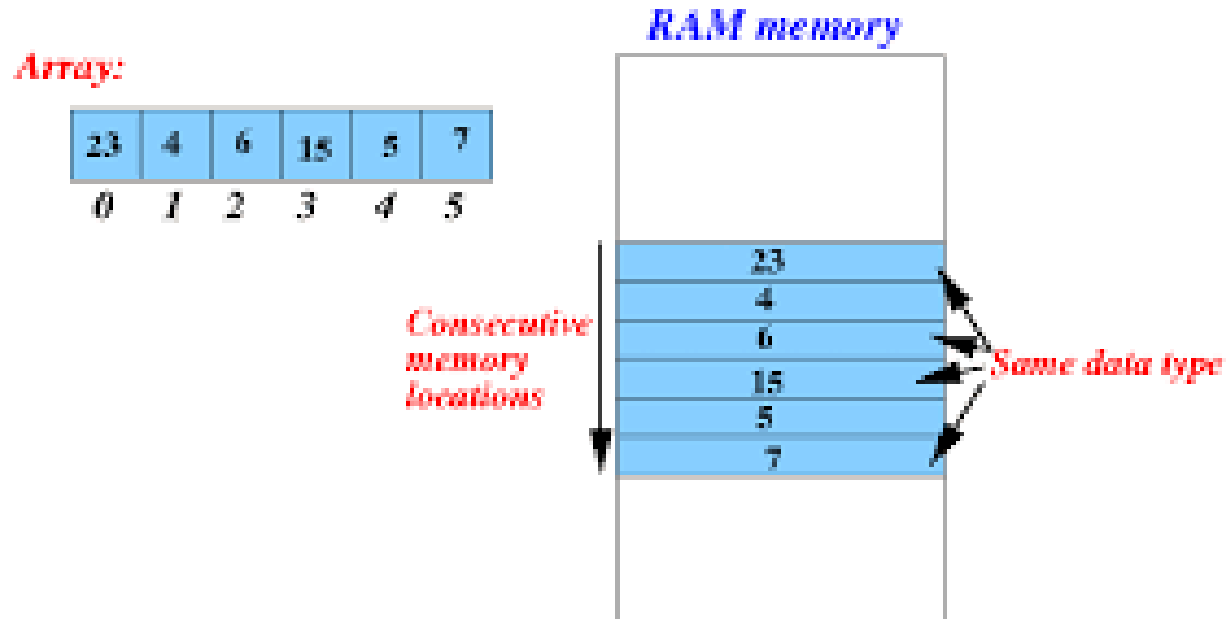
Temel Veri Yapıları

- Diziler (Arrays)
- Bileşik Veri Türleri (Aggregates)
- Listeler (Lists)
 - Yığın (Stack)
 - Kuyruk (Queue)
- Ağaç (Tree)

Temel Veri Yapıları

➤ Diziler (Arrays)

- Dizi aynı tipten girdilere yani elemanlara sahip dikdörtgen bir veri öbeğidir.
 - İki boyutlu bir dizi, satır ve sütunlardan oluşur.
 - Elemanların yerlerini belirlemek için indeks değerleri kullanılır.



Temel Veri Yapıları

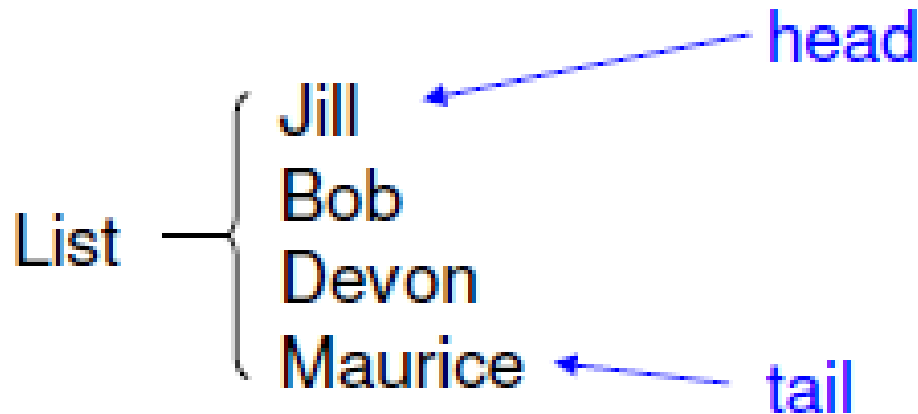
➤ Bileşik Veri Tipleri (Aggregates)

- Diziden farklı olarak, farklı tür veya boyutlarda olabilecek verilerden oluşan veri bloğuna birleşik tip denir.
- Bloktaki her bir veri öğesine **alan** denmektedir.
- Alanlara genellikle alan adları ile erişilir.

Temel Veri Yapıları

➤ Listeler (Lists)

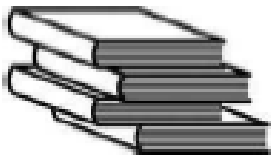
- Diğer bir temel veri yapısı ise listedir.
- Elemanları birbirine bağlı olarak düzenlenen veri koleksiyonlarına liste denir.
- Bir listenin başlangıç elemanına listenin başı (head) denir.
- Listenin son elemanına ise kuyruk (tail) denmektedir.



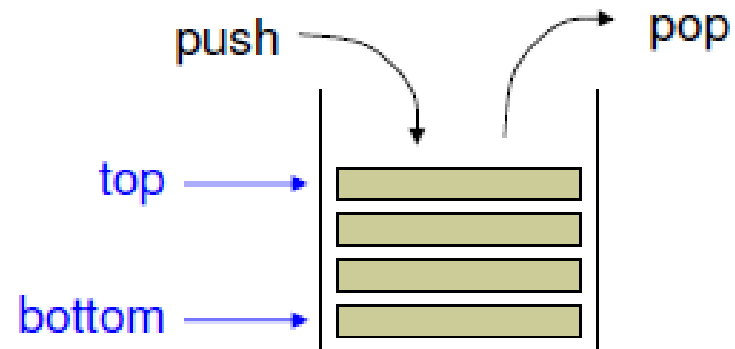
Temel Veri Yapıları

➤ Yığınlar (Stacks)

- Yığın, sadece baştaki girdilerin çıkarılıp yeni girdilerin başa eklenebildiği bir listedir.
- LIFO: Son giren ilk çıkar (Last In First Out) mantığı ile çalışır.
- Üst (Top): Listenin (yığın) başı
- Alt (Bottom) veya taban (base): Listenin kuyruğu
- Pop: Üstteki elemanı çıkarmak içindir
- Push: En üste bir giriş eklemek içindir



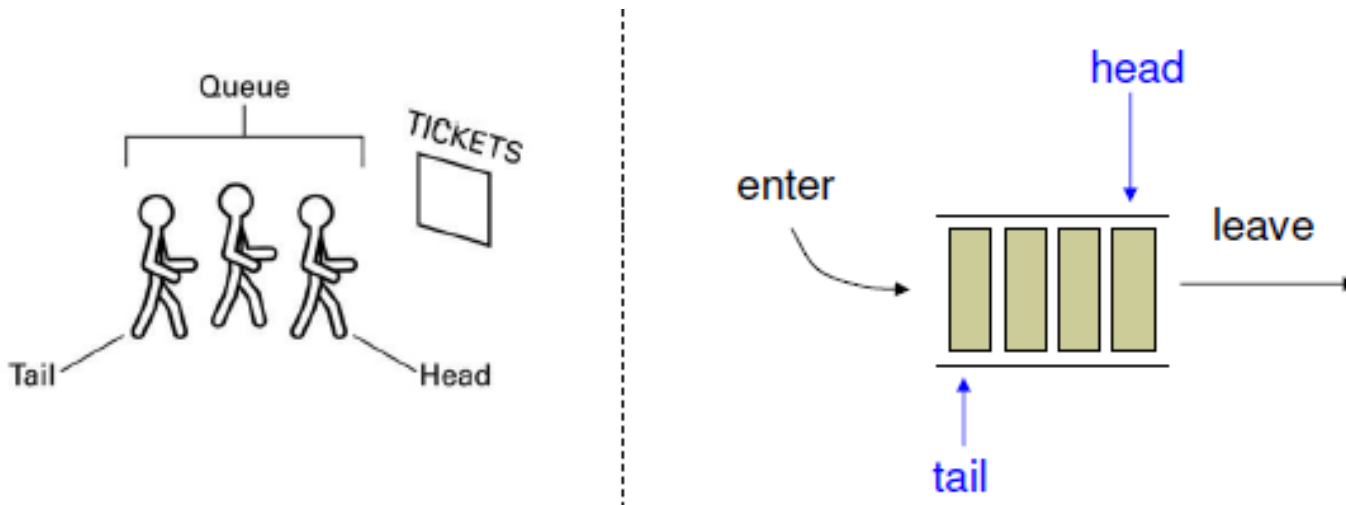
a stack of books



Temel Veri Yapıları

➤ Kuyruklar (Queues)

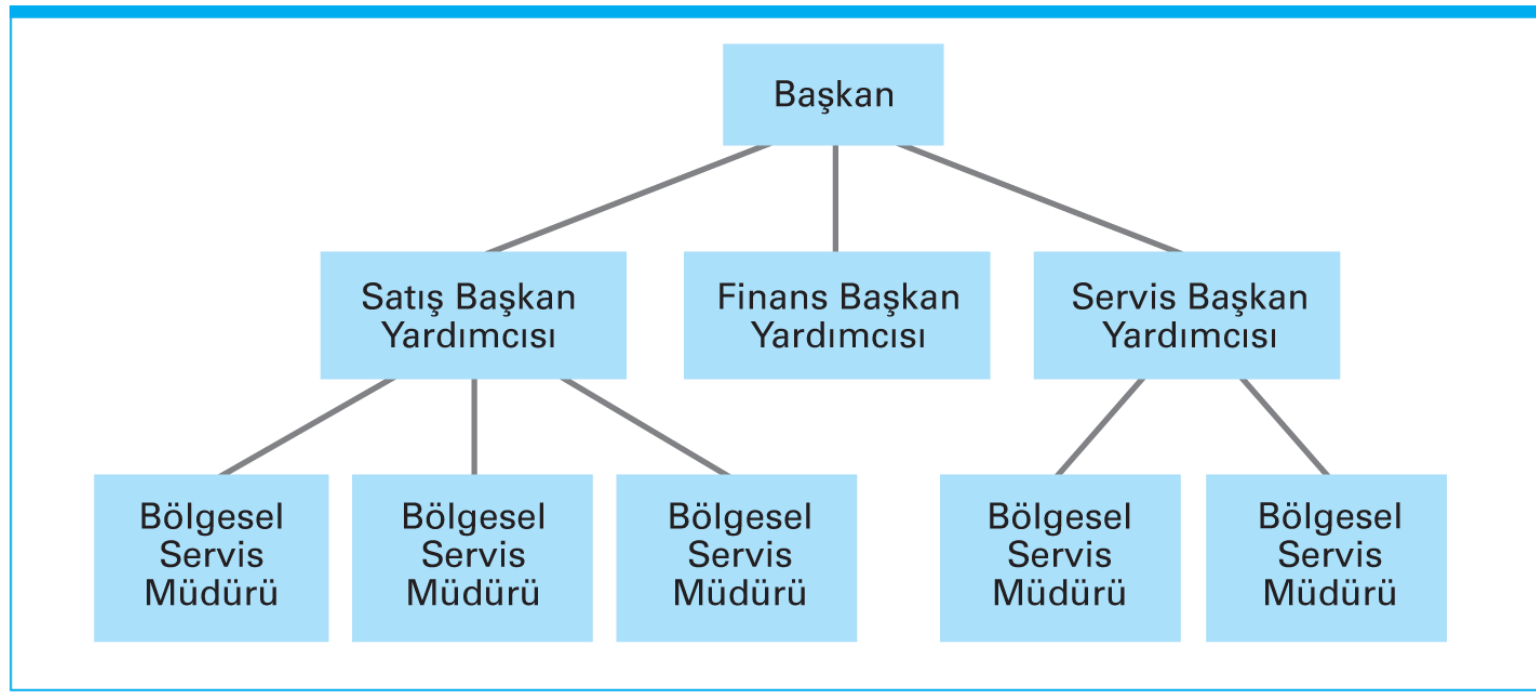
- Kuyruk ise, girdilerin sadece baştan çıkarılabildiği ve yeni girdilerin kuyruk kısmından eklenebildiği bir listedir.
- FIFO: İlk giren ilk çıkar (First In First Out) mantığı ile çalışır.
- enqueue: kuyruğa eleman ekleme içindir.
- dequeue: kuyruktan veri çıkışı içindir.



Temel Veri Yapıları

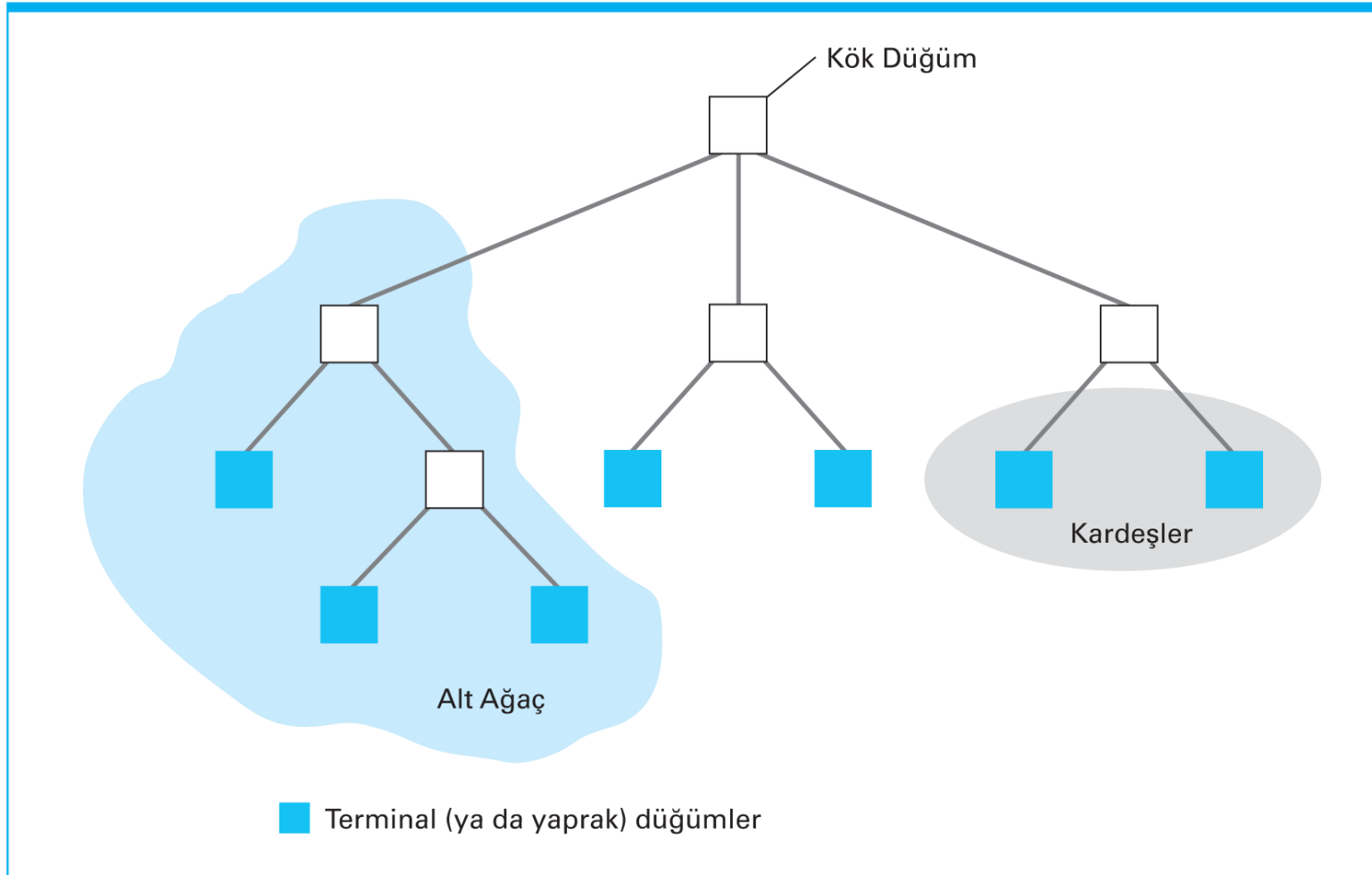
➤ Ağaçlar (Trees)

- Girdilerin bir şirketin sahip olduğuna benzer tipik bir hiyerarşik organizasyona sahip olduğu derlemeye bir ağaç denir.



Temel Veri Yapıları

➤ Ağaç Terimleri



Temel Veri Yapıları

➤ Ağaç Terimleri

- Ağaçtaki her girişe düğüm (node) adı verilir
- Kök (root), ağacın en üstündeki düğümdür
- Bir yaprak (veya terminal) ağacın en altındaki bir düğümdür
- Bir düğümün ebeveyni, belirtilen düğümün hemen üzerindeki düğümdür
- Bir düğümün çocuğu, belirtilen düğümün hemen altındaki düğümdür

Temel Veri Yapıları

➤ Ağaç Terimleri

- Bir düğümün ataları, düğümün ebeveyni, ebeveynin ebeveyni vb.'dir.
- Bir düğümün soyundan gelenler, düğümün çocukları, çocukların çocukları vb.'dir.
- Bir düğümün kardeşleri, o düğümle aynı ebeveyni paylaşan tüm düğümlerdir.
- İkili ağaç, her düğümün en fazla iki çocuğa sahip olduğu bir ağaçtır
- Bir ağacın derinliği, kökten yaprağa kadar olan en uzun yoldaki düğümlerin sayısıdır.

Genel Kavramlar

➤ Soyutlama

- Bir bilgisayarın ana belleği adreslenebilir hücreler olarak organize edilmektedir.
- Bu nedenle, diğer tüm yapıların benzetiminin yapılması gerekir.
- Kullanılan organizasyonlar ile kullanıcılar gerçek veri ortamının detaylarından yalıtılmakta ve bilgiye daha uygun bir formda ulaşmaları sağlanmaktadır.

Genel Kavramlar

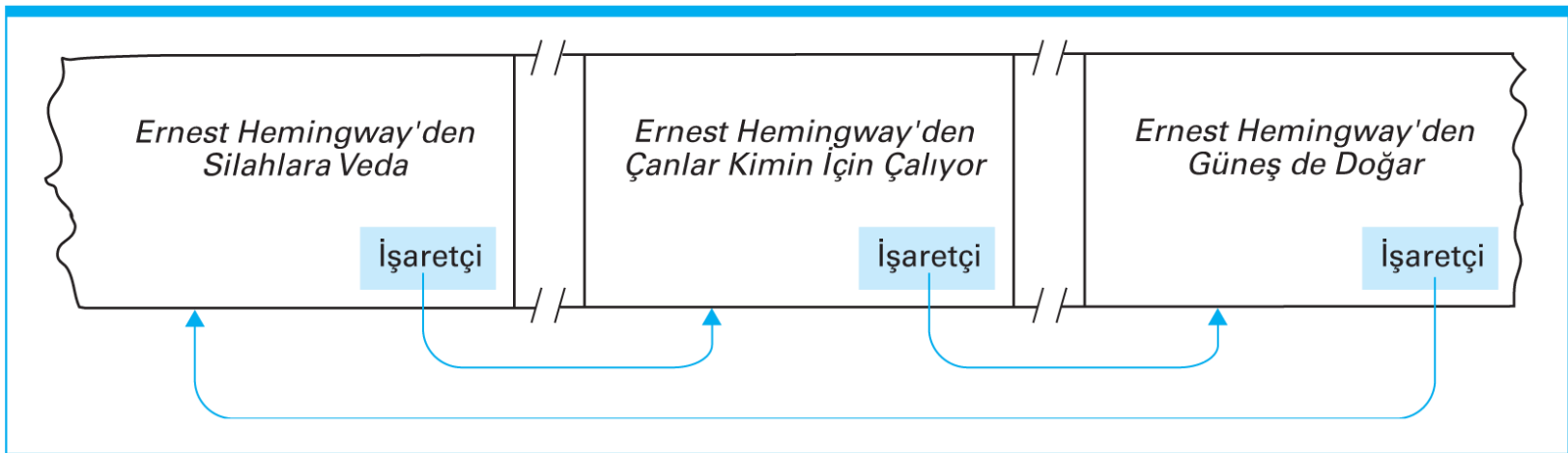
➤ Durağan Yapılara Karşı Devingen Yapılar

- Soyutlanmış veri yapıları oluşturmaktaki önemli bir ayrım ise yapının şeklinin ya da büyüklüğünün zamana bağlı olarak değişmesi yani durağan/static (static) ya da devingen/dinamik (dynamic) olarak benzetimde bulunulmasıdır.
- Statik veri yapısı:
 - Veri yapısının boyutu değişmez
- Dinamik veri yapısı:
 - Veri yapısının boyutu değişebilir

Genel Kavramlar

➤ İşaretçiler

- Makinenin ana belleğindeki belirli hücreler sayısal adresler ile çağırılırlar.
- Sayısal değerler olması sebebiyle, bu değerler de aynı zamanda bellek hücrelerinde kodlanabilir ve saklanabilir.
- Bir işaretçi bu tür kodlanmış adresleri içeren saklama alanlarına denmektedir.



Veri Yapılarının Gerçekleştirilmesi

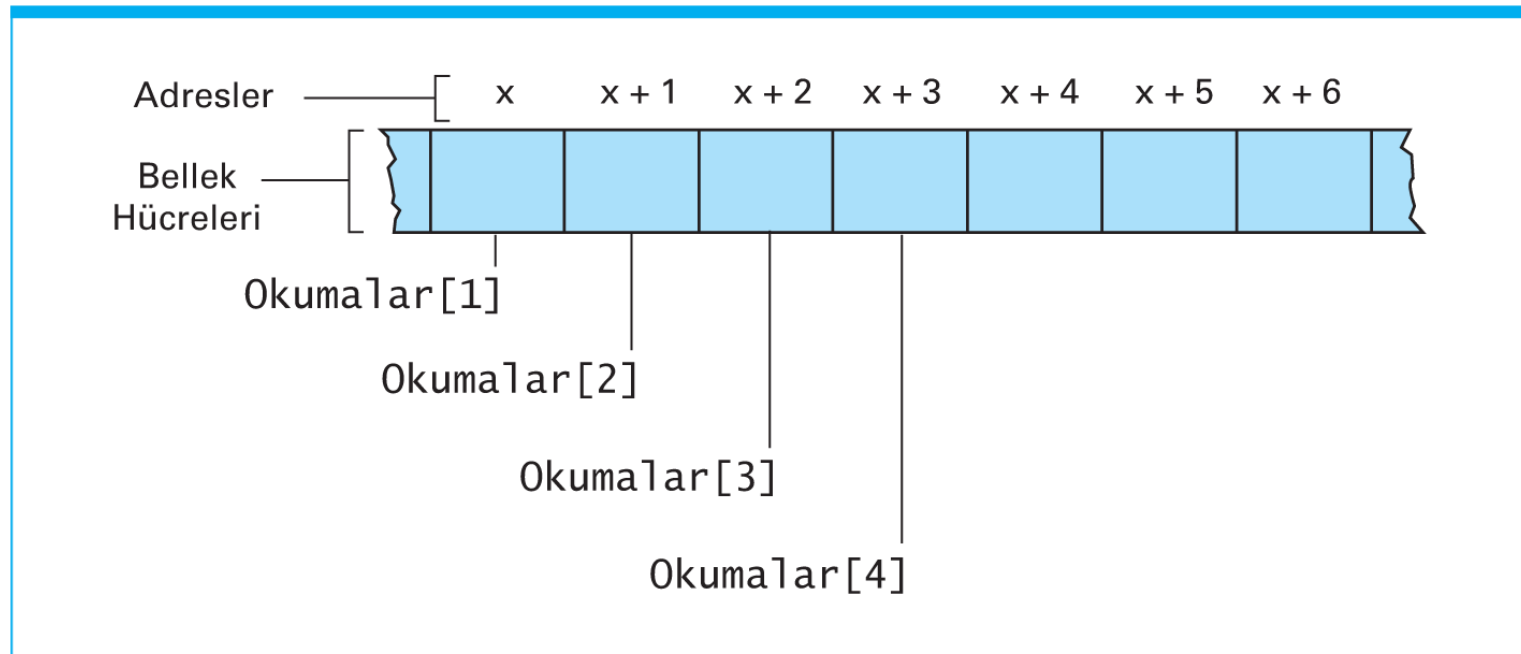
➤ Dizilerin Saklanması

- Homojen diziler genellikle bitişik bellek bloklarında saklanır
- 24 saatlik sıcaklık okumalarını saklamak istediğimizi varsayalım. Her bir okuma saklama alanı olarak bir bellek hücresine ihtiyaç duysun. Dahası, bu okumaları ardışığındaki (sıralamadaki) pozisyonlarına göre tanımlamak isteyelim.
- Böylece birinci ya da beşinci okumaya doğrudan ulaşabilmek isteyelim.
- Kısaca, sıralamayı bir boyutlu diziymiş gibi değiştirmek isteyelim.

Veri Yapılarının Gerçekleştirilmesi

➤ Dizilerin Saklanması

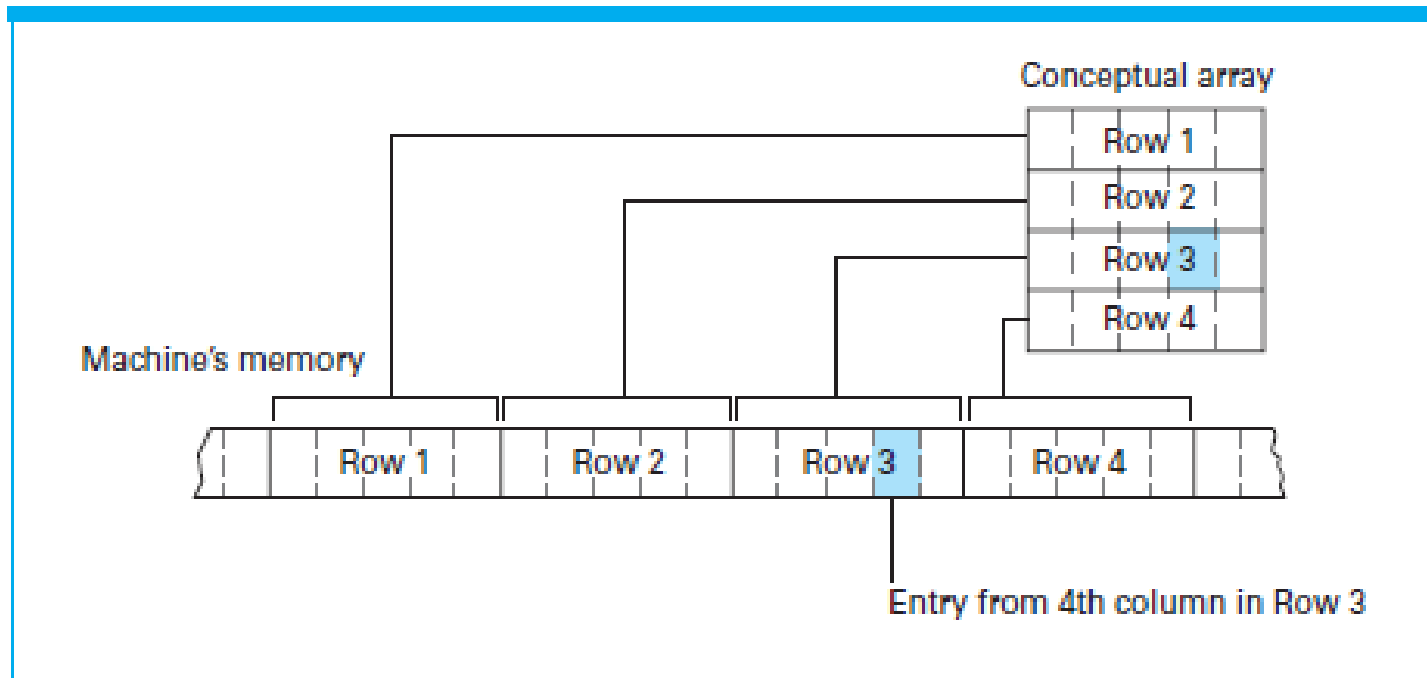
- x adresinden başlayarak bellekte saklanan sıcaklık okumaları dizisi



Veri Yapılarının Gerçekleştirilmesi

➤ İki Boyutlu Dizilerin Saklanması

- Bellek hücreleri tek boyutlu sıradadır, iki boyutlu bir diziyi depolamak için bazı eşlemelerin yapılması gerekir:
 - Veri girişleri sütun sütun saklanır
 - Veri girişleri satır satır saklanır



Veri Yapılarının Gerçekleştirilmesi

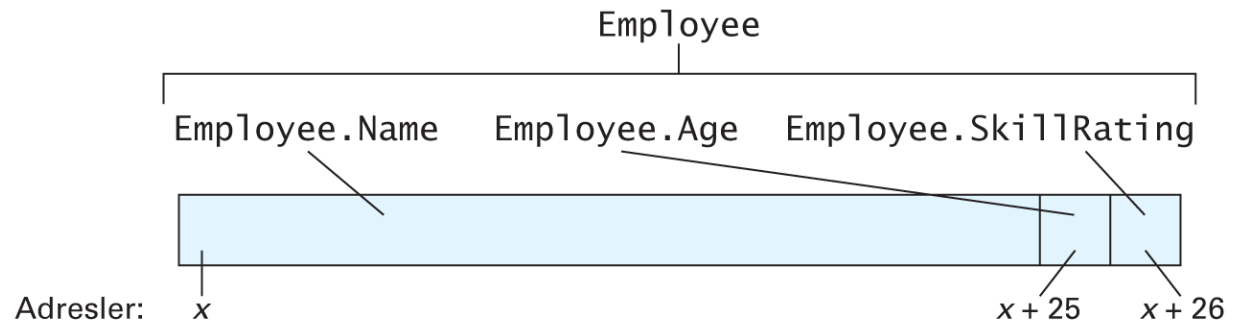
➤ Birleşiklerin Saklanması

- Heterojen diziler (yapılar), işaretçiler kullanılarak bitişik bloklar veya bitişik olmayan bloklar halinde saklanabilir.
- Üç adet alana sahip bir Employee isimli bir birleşik yapısı saklamak istediğimizi varsayalım: karakter dizisi tipindeki Name, tam sayı tipindeki Age ve ondalık tipindeki SkillRating.
- Eğer her bir alan için gerekli bellek hücresinin sayısı sabit ise, birleşigi ardışık hücre blokları içerisinde saklayabiliriz.

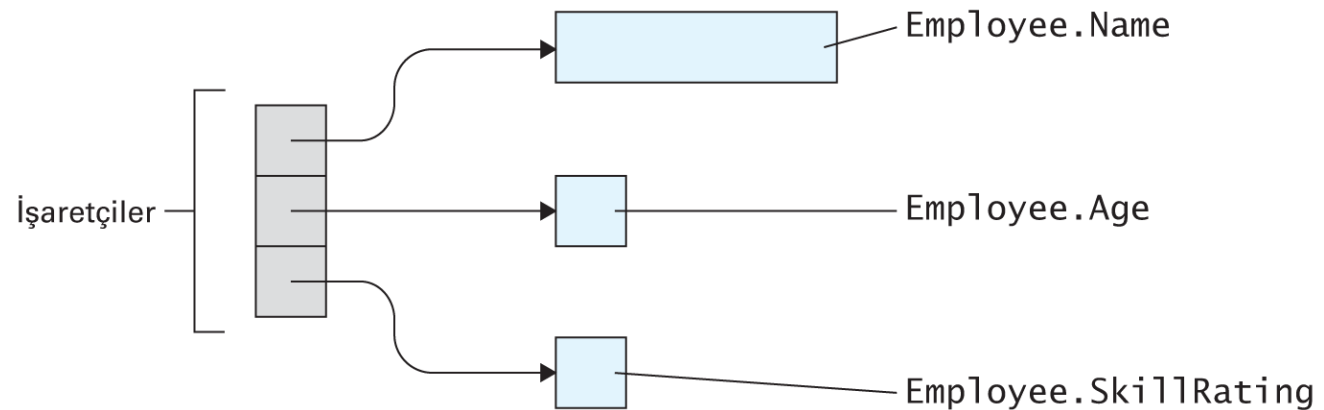
```
struct  
{  
    char Name[25];  
    int Age;  
    float SkillRating;  
} Employee;
```

Veri Yapılarının Gerçekleştirilmesi

- Employee
isimli birleşik
tipi saklamak



a. Bir bitişik blokta saklanan birleşik



b. Ayrı yerlerde saklanan birleşik alanları

Veri Yapılarının Gerçekleştirilmesi

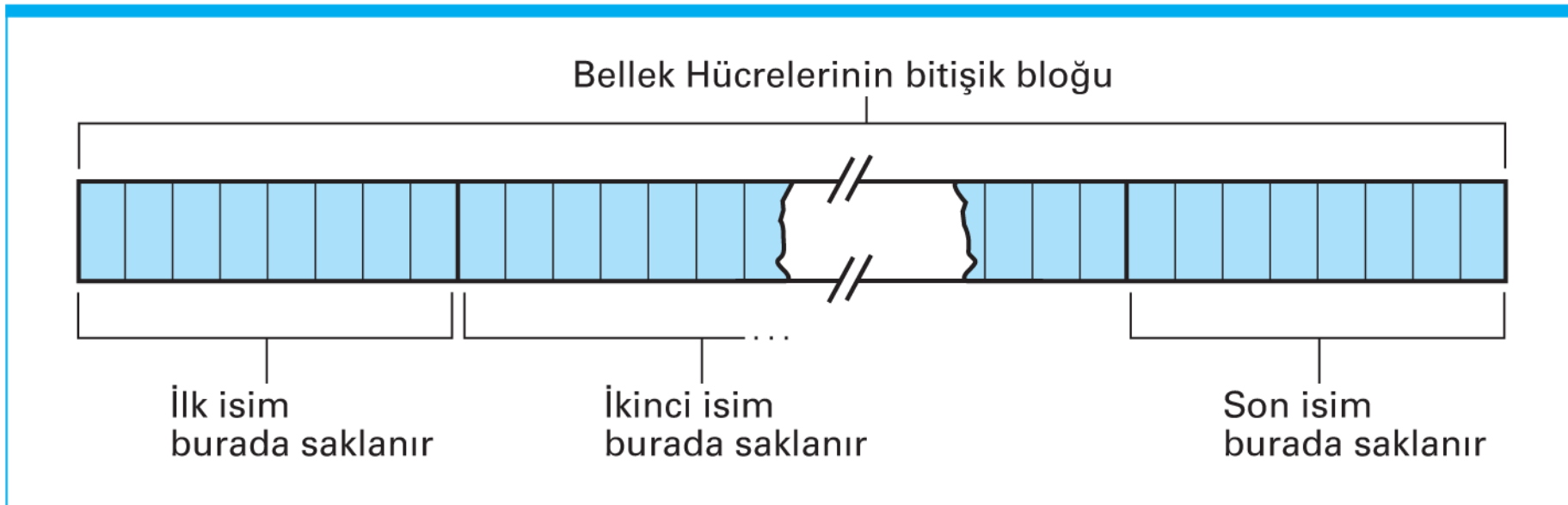
➤ Listelerin Saklanması

- Bir strateji tüm listeyi ardışık adreslerde tek bir bellek hücreleri bloğu içerisinde saklamaktır. Bu yöntem bitişik liste yöntemidir.
- Bir diğer yöntem ise, işaretçilerin kullanılması ve liste elemanlarının belleğin farklı alanlarında tutulmasıdır.
- Buna da bağlı liste denir.
 - Baş (head) işaretçisi: Listedeki ilk eleman için işaretçi
 - Boş (null): Listenin sonunu belirtmek için kullanılan "işaretçi olmayan" değer

Veri Yapılarının Gerçekleştirilmesi

➤ Listelerin Saklanması

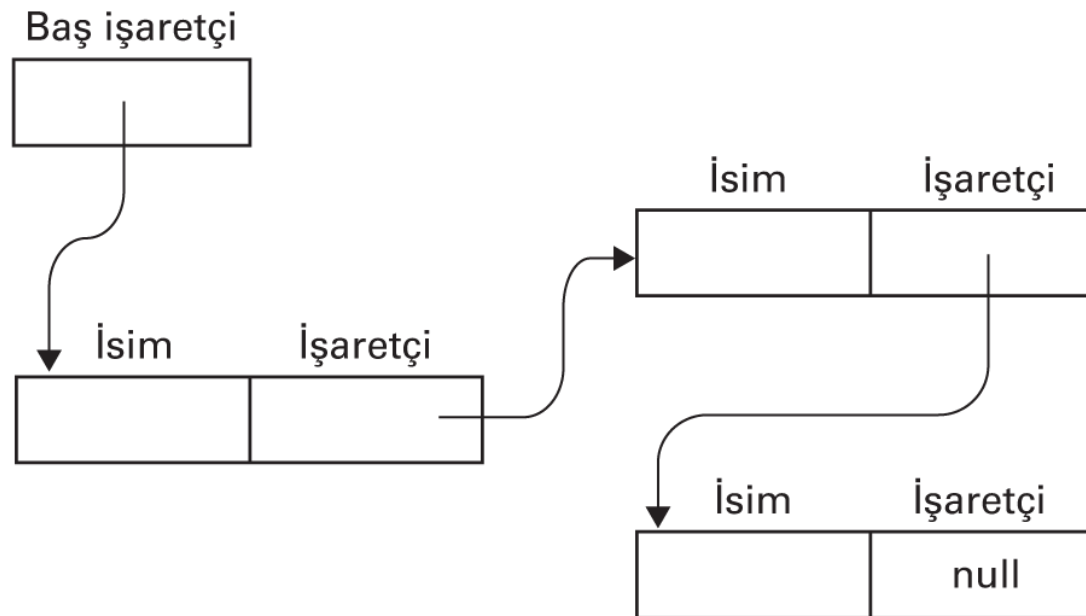
- Bellekte bitişik bir liste olarak saklanan isimler:



Veri Yapılarının Gerçekleştirilmesi

➤ Listelerin Saklanması

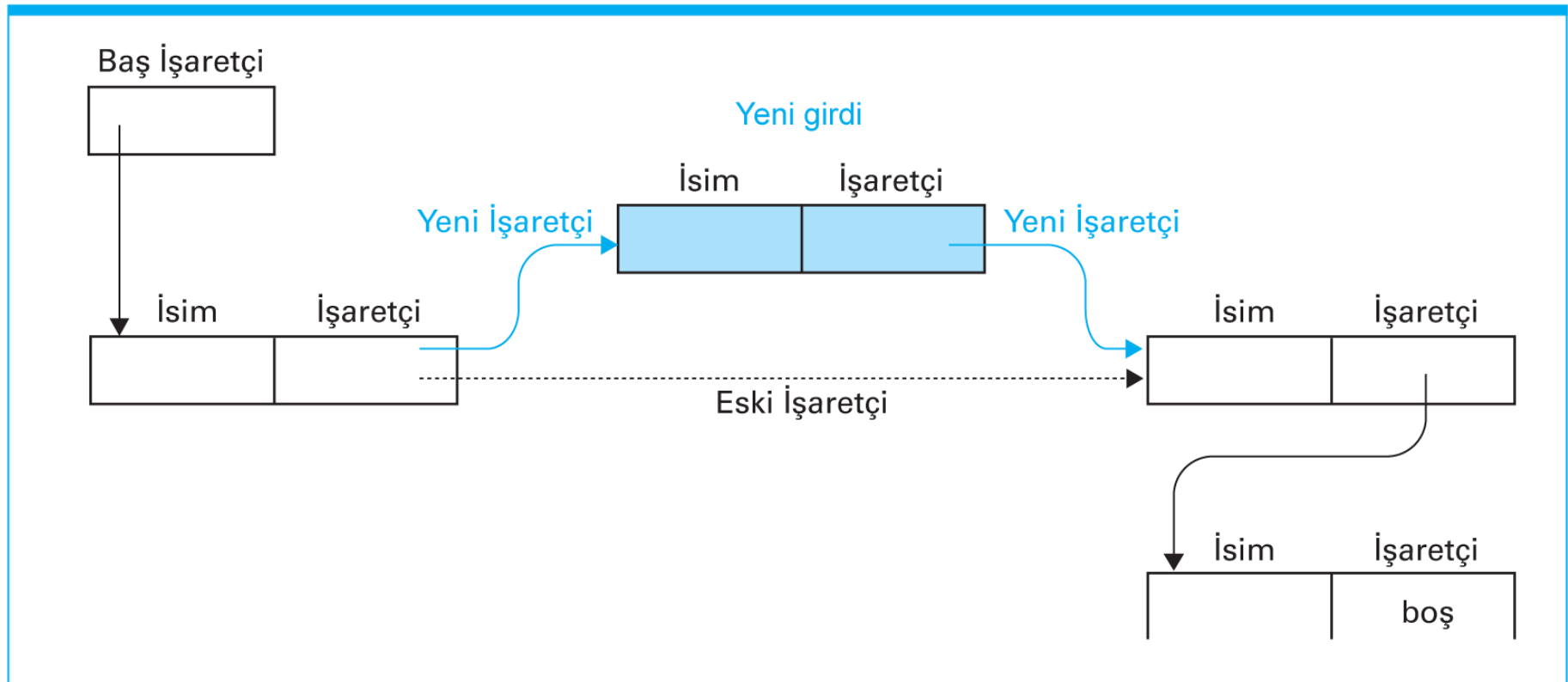
- Bir bağlı listenin yapısı



Veri Yapılarının Gerçekleştirilmesi

➤ Listelerin Saklanması

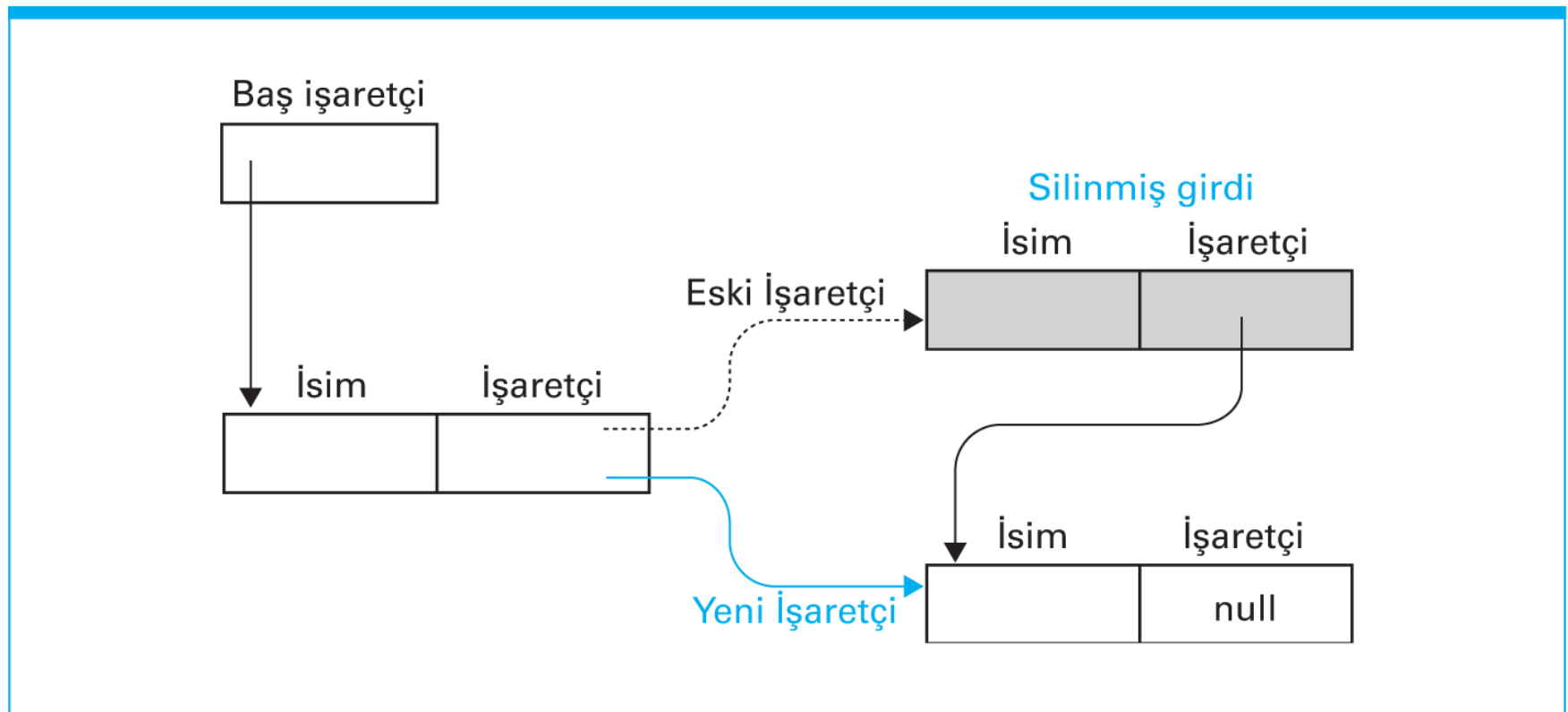
- Bağlı listeye bir girdi eklemek



Veri Yapılarının Gerçekleştirilmesi

➤ Listelerin Saklanması

- Bağlı listeden bir girdiyi silmek



Veri Yapılarının Gerçekleştirilmesi

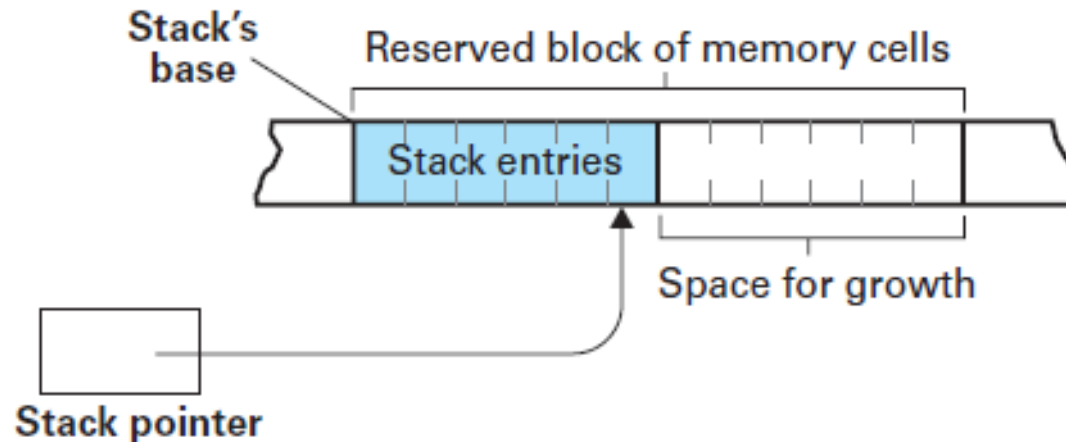
➤ Yığınların ve Kuyrukların Saklanması

- Yığın ve kuyrukları saklayabilmek için listedeki organizasyon yapıları kullanılır.
- Yığınlar genellikle bitişik listeler olarak saklanır.
- Kuyruklar için ise genellikle dairesel liste şeklinde bir yaklaşım seçilebilir.
 - İlk elemanın son elemanı takip ettiği kabul edilen bitişik bir blokta saklanır
 - Bir kuyruğun kendine ayrılmış depolama alanından dışarı sarkması engellenmiş olur.

Veri Yapılarının Gerçekleştirilmesi

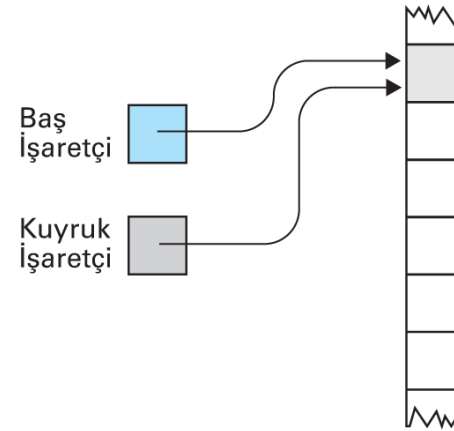
➤ Yığınların ve Kuyrukların Saklanması

- Bellekteki bir yığın

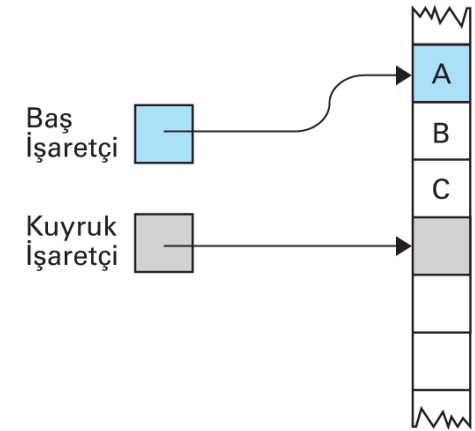


Veri Yapılarının Gerçekleştirilmesi

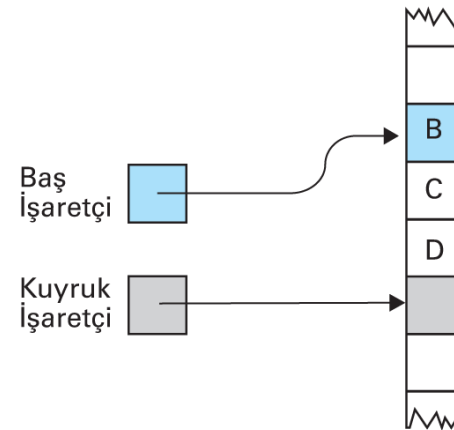
- Baş ve kuyruk işaretçileri ile bir kuyruk gerçekleştirimi



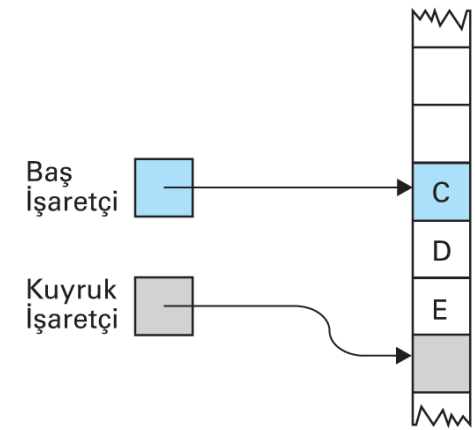
a. Boş Kuyruk



b. A, B ve C girdilerini ekledikten sonra



c. A'yı çıkarıp, D'yi ekledikten sonrası

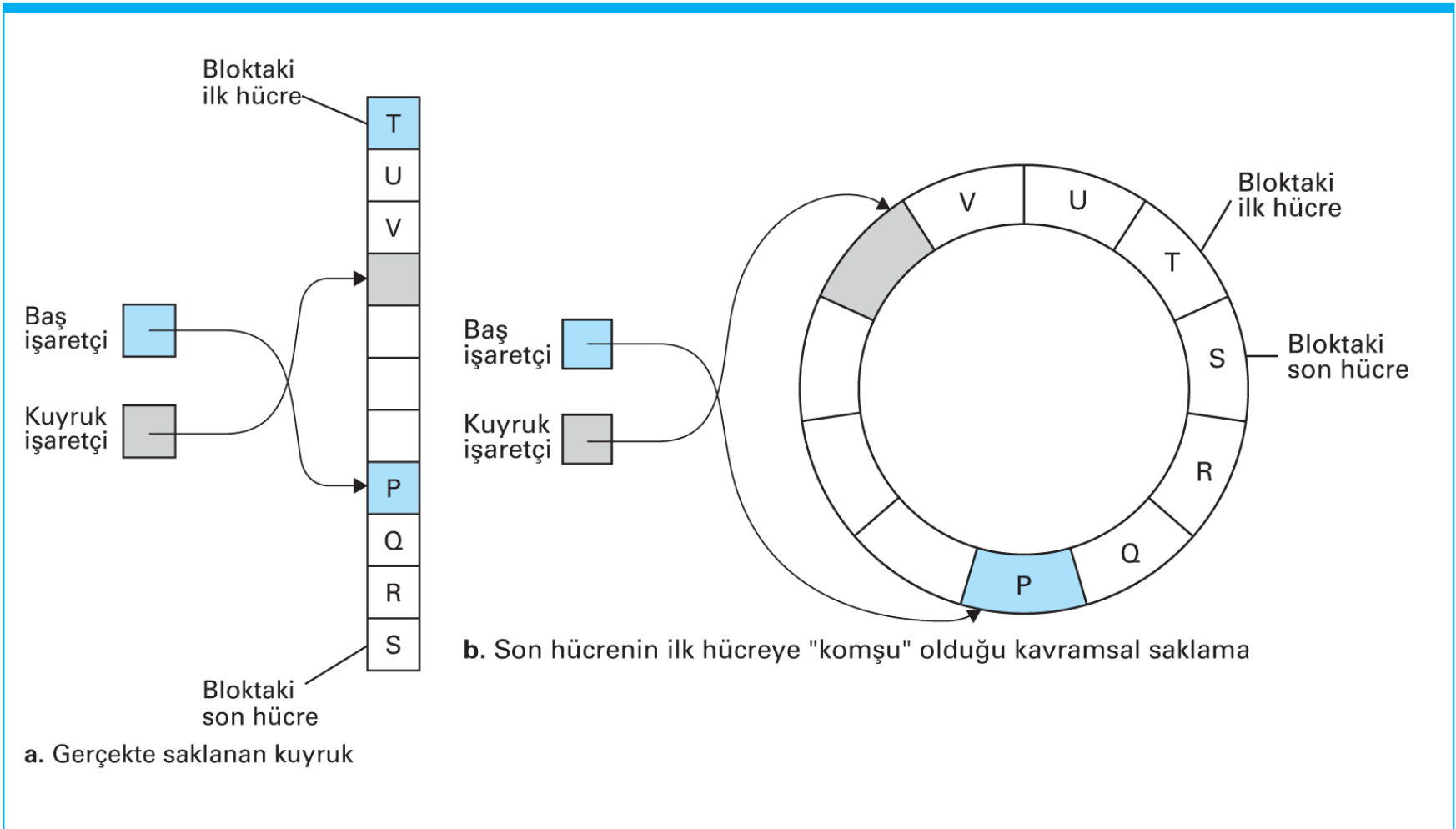


d. B'yi çıkarıp, E'yi ekledikten sonrası

Veri Yapılarının Gerçekleştirilmesi

➤ Yığınların ve Kuyrukların Saklanması

- P'den V'ye kadar olan harfleri içeren dairesel bir kuyruk



Veri Yapılarının Gerçekleştirilmesi

➤ İkili Ağaçların Saklanması

- Ağaç veri yapısının saklanma tekniklerinin tartışılması amacıyla, ilgimizi sadece her bir düğümü en fazla iki çocuğa sahip olan ikili ağaçlar ile sınırlandırıyoruz.
- İkili bir ağaçtaki düğüm yapısı



Veri Yapılarının Gerçekleştirilmesi

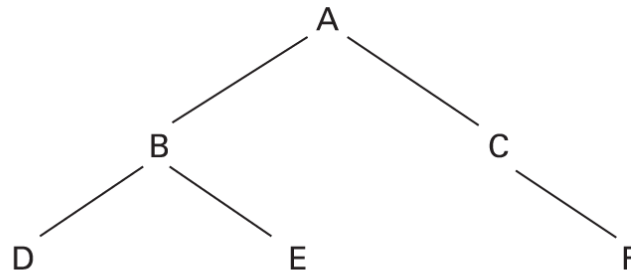
➤ İkili Ağaçların Saklanması

- Bağlantılı Yapı: Bağlı Liste
 - Her düğüm = veri hücresi + iki çocuk işaretçisi
 - Kök düğüme bir işaretçi ile ulaşılır.
- Bitişik Dizi Yapısı
 - $A[1]$ = kök düğüm
 - $A[2], A[3]$ = $A[1]$ 'in çocukları
 - $A[4], A[5], A[6], A[7]$ = $A[2]$ ve $A[3]$ 'ün çocukları

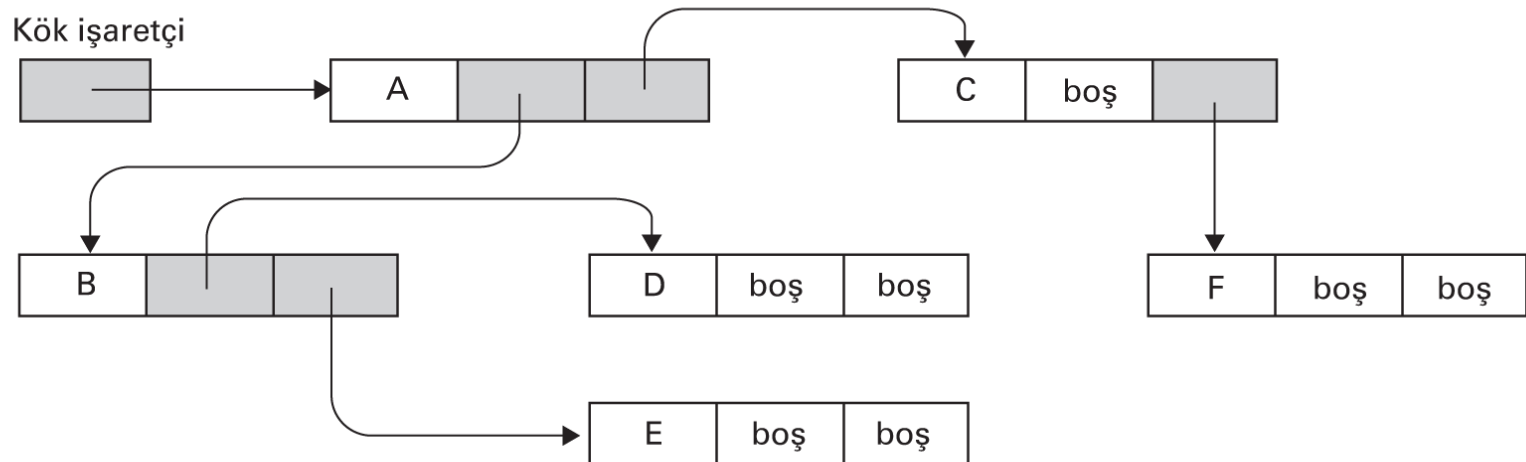
Veri Yapılarının Gerçekleştirilmesi

➤ Bağlı Liste ile ikili ağacın kavramsal ve gerçek organizasyonu

Kavramsal Ağaç



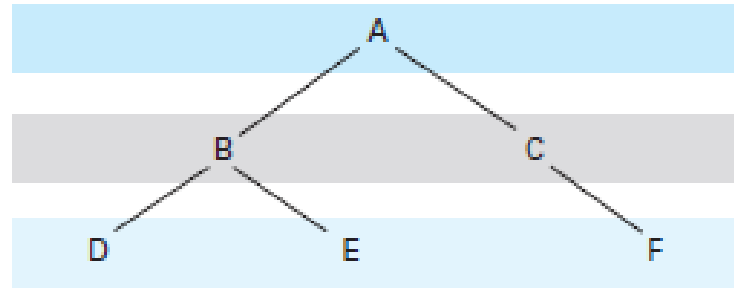
Gerçek Saklama Organizasyonu



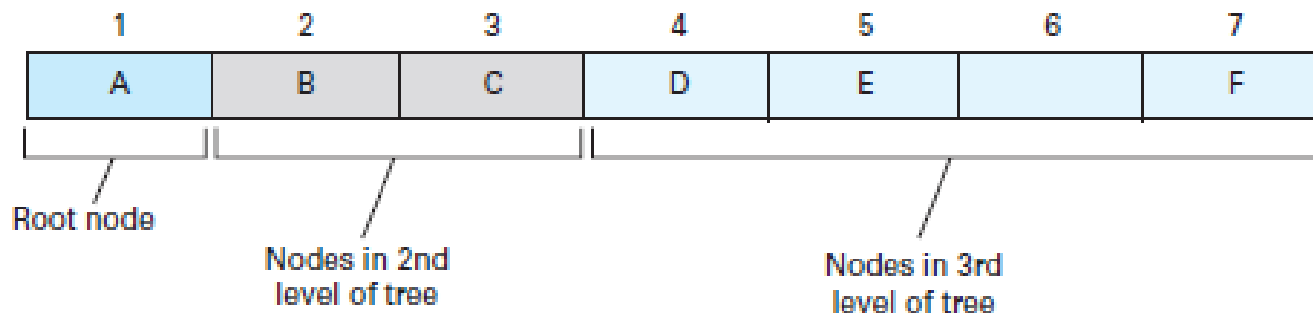
Veri Yapılarının Gerçekleştirilmesi

- Bitişik Dizi ile kavramsal formda gösterilen ve işaretçiler olmadan saklanan ağaç

Conceptual tree



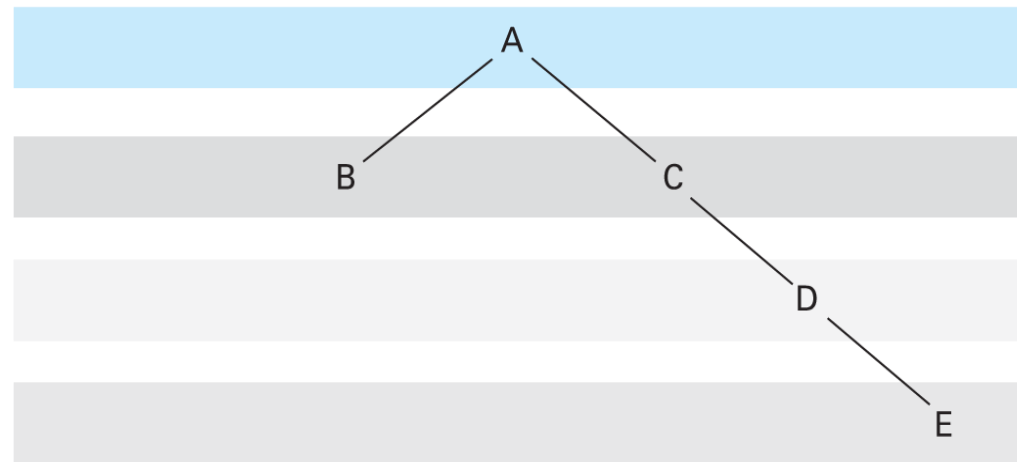
Actual storage organization



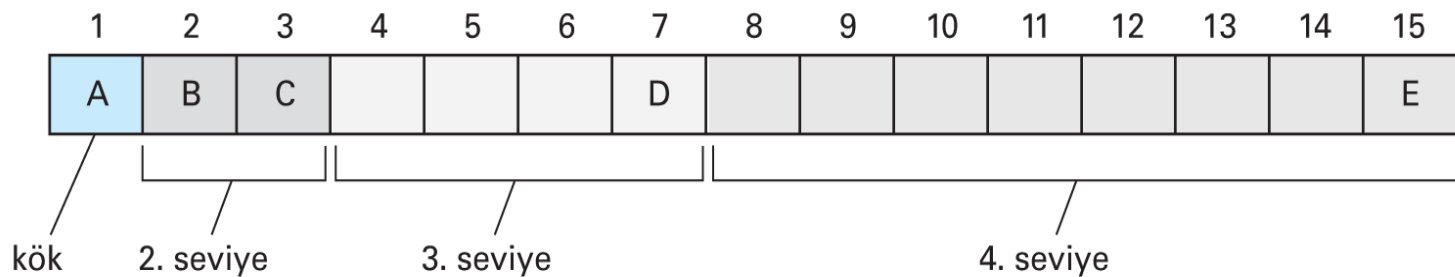
Veri Yapılarının Gerçekleştirilmesi

- Bitişik Dizi ile Kavramsal formda gösterilen ve işaretçiler olmadan saklanan seyrek ve dengesiz ağaç

Kavramsal ağaç



Gerçek saklama organizasyonu



Veri Yapılarının Gerçekleştirilmesi

➤ Veri Yapılarının Yönetilmesi

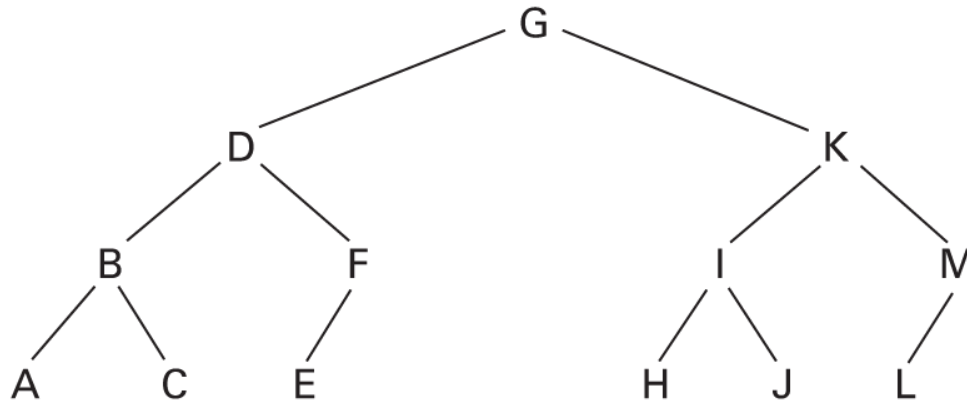
- Veri yapılarının aslında bilgisayarın belleğinde saklanma şeklinin, kullanıcının hayal ettiği kavramsal yapıyla aynı olmadığını gördük.
 - İki boyutlu bir dizi aslında iki boyutlu dikdörtgen blok olarak saklanmamakta ve bir liste ya da ağaç büyük bir bellek alanına yayılmış ufak parçalardan oluşabilmektedir.
- Bu nedenle, kullanıcının yapıya soyut bir araç olarak erişebilmesini sağlamak için kullanıcıyı gerçek depolama sisteminin karmaşıklığından korumamız gerekir.
- Yani, kullanıcı tarafı komutları gerçek saklama sistemine uygun olan adımlara çevrilmelidir.
- Bir veri yapısı yalnızca önceden tanımlanmış fonksiyonlar tarafından düzenlenerek tam bir soyut araç haline gelir.

Örnek Bir Durum Çalışması

- Bir isim listesinin alfabetik olarak saklanması işlemini düşünelim.
- Bu liste üzerinde gerçekleştirilecek işlemlerin aşağıdakiler olmasını tahminleriz:
 - Bir girdinin varlığını ara,
 - Listeyi alfabetik olarak yazdır ve
 - Yeni girdiyi ekle.

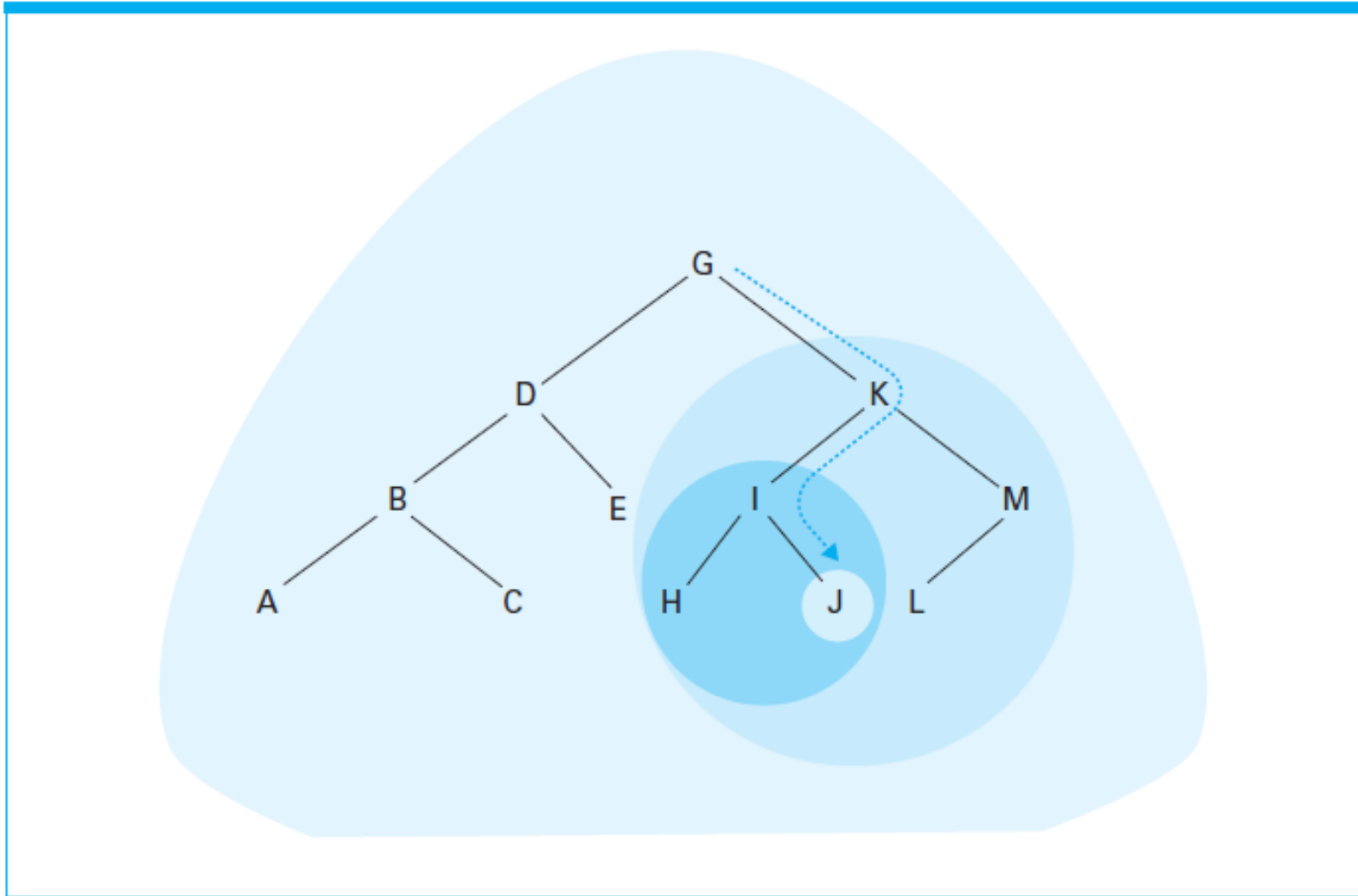
Örnek Bir Durum Çalışması

- Sıralı bir ağaç içerisinde yerleştirilen A'dan M'ye kadar harfler



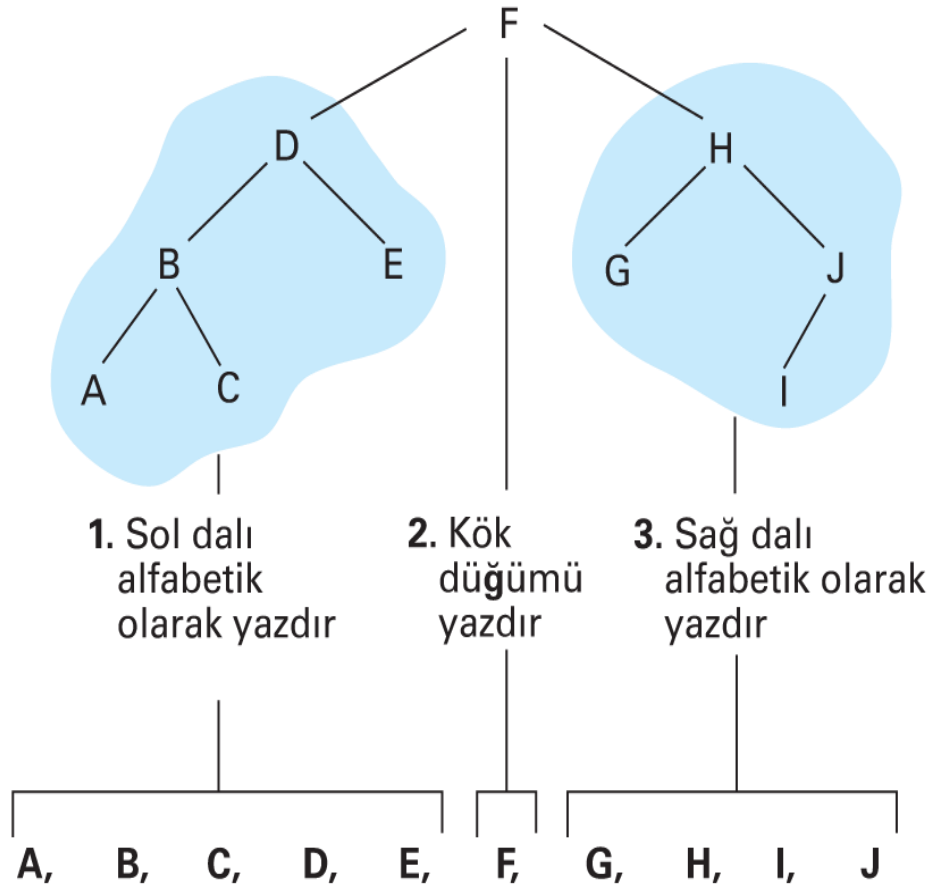
Örnek Bir Durum Çalışması

- İkili arama kullanarak J harfini ararken oluşan alt ağaçlar



Örnek Bir Durum Çalışması

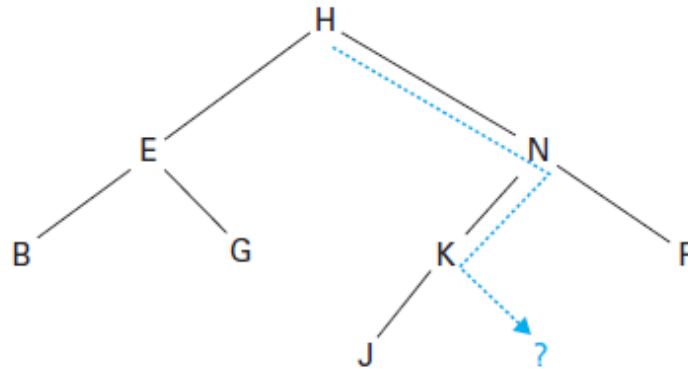
- Bir arama ağacının alfabetik olarak yazdırılması



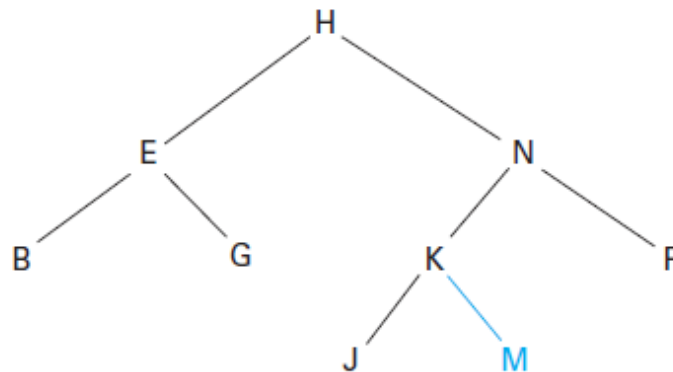
Örnek Bir Durum Çalışması

- M girdisinin aşağıdaki ağaç listesine eklenmesi

a. Search for the new entry until its absence is detected



b. This is the position in which the new entry should be attached



Özelleştirilmiş Veri Tipleri

➤ Kullanıcı-Tanımlı Veri Tipleri

- Programlama dilinde basit tiplerin dışındaki veri tipleri de destekleniyorsa bir algoritmayı ifade etmek daha kolaydır.
- Bu “ev yapımı” veri tiplerinin en basit örneği kullanıcı-tanımlı veri tipleri olarak bilinmektedir.
- Kullanıcı-tanımlı veri tipleri (User-defined data types) aslında basit tiplerin tek bir isim altında kümelenmesidir.

```
struct  
{  
    char    Name[25];  
    int     Age;  
    float   SkillRating;  
} Employee;
```

Özelleştirilmiş Veri Tipleri

➤ Soyut Veri Tipleri

- Bir soyut veri tipi (Abstract Data Type-ADT) hem veri (gösterim) hem de fonksiyonlar (davranış) içeren kullanıcı-tanımlı bir veri tipidir.
- ADT'lerin oluşturulmasını destekleyen programlama dilleri genellikle iki özellik sağlar:
 - ADT'yi tek birim olarak tanımlamak için sözdizimi: verileri ve işlevleri bir arada tutmaya yönelik, bakım ve hata ayıklamayı kolaylaştıran önemli bir organizasyon aracıdır.
 - ADT'nin iç yapısını onu kullanacak olan programın diğer bölümlerinden gizlemek için bir mekanizma: ADT dışındaki kodların, bu amaç için sağlanan işlevlerden geçmeden verilere erişmesini engelleyerek güvenilirlik sağlar.

Sınıflar ve Nesneler

- Her bir nesne başka nesneler tarafından aldığı mesajlara cevap verebilen bir varlıktır.
- Nesneler, sınıf olarak bilinen taslaklar tarafından tanımlanmaktadır.
- Birçok açıdan, bu sınıflar soyut veri tiplerinin örneklerine nesne denen tanımlarıdır ve ADT'lerin uzantısıdır.
- Ancak sınıflar ve soyut veri türleri arasında farklılıklar vardır:
 - Ayrı ayrı nesnelerin yapılandırıcılar ile özelleştirilmesine yani çok biçimliliğe olanak tanır.
 - Kalıtım ile benzer sınıflar arası özellik aktarımını destekler.
 - Kapsülleme ile koruma ve gizlilik sağlar.
 - Ayrıca, soyutlamaları uygun bir şekilde tanımlama ve kullanma yeteneği yaratır.

Özetle

