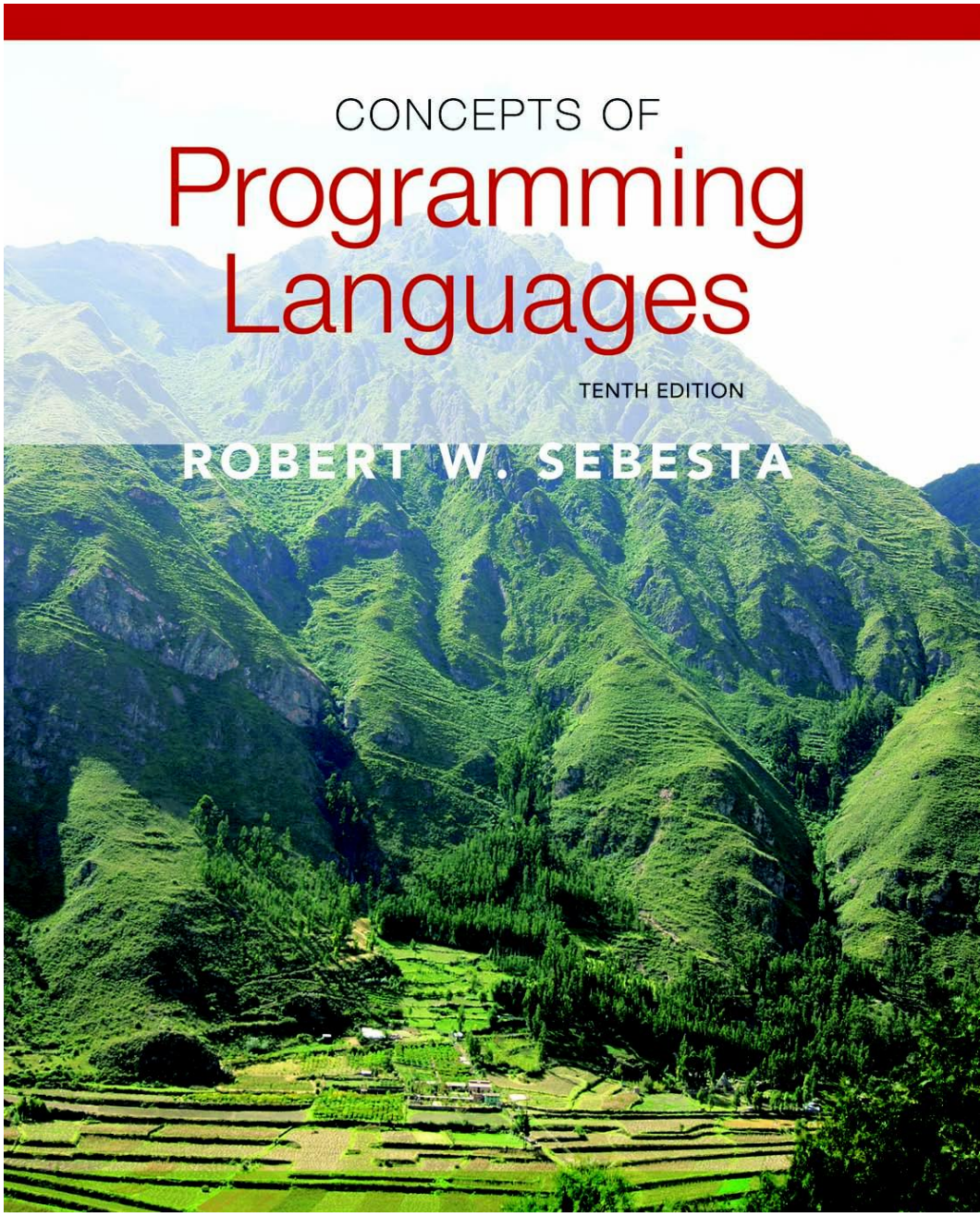
Bölüm 6

Veri Tipleri



6. Bölümün Başlıkları

- Giriş
- İlkel Veri Tipleri
- Karakter (String) Veri Tipleri
- Kullanıcı Tanımlı Sıra Tipleri
- Dizi Tipleri
- İlişkili Diziler
- Kayıt Tipleri
- Demet Tipler
- Liste Tipleri
- Birleşim Tipleri
- Pointer ve Referans Tipleri
- Tip Kontrolü
- Güçlü Tipleme
- Tip Eşitleme
- Teori ve Veri Tipleri

Giriş

- Bir veri tipi(data type) bir veri nesneleri(data objects) koleksiyonunu ve bu nesnelerin bir takım ön tanımlı işlemlerini (predefined operations) tanımlar.
- Bir tanımlayıcı bir değişken niteliklerinin topluluğudur.
- Bir nesne bir kullanıcı tanımlı (soyut veri) türünde bir örnek temsil eder.
- Bütün veri tipleri(data types) için bir tasarım meselesi:

  Hangi işlemler tanımlanmıştır ve nasıl belirlenir?

İlkel Veri Tipleri

- Neredeyse tüm programlama dilleri ilkel veri türleri kümesi sağlar.
- İlkel Veri Tipleri: Diğer veri tipleri cinsinden tanımlanmayan veri tipleridir.
- Bazı ilkel veri tipleri sadece donanım yansımalarıdır.
- Diğerleri bunların uygulanması için sadece küçük olmayan donanım desteği gerektirir.

İlkel Veri Tipleri: Integer

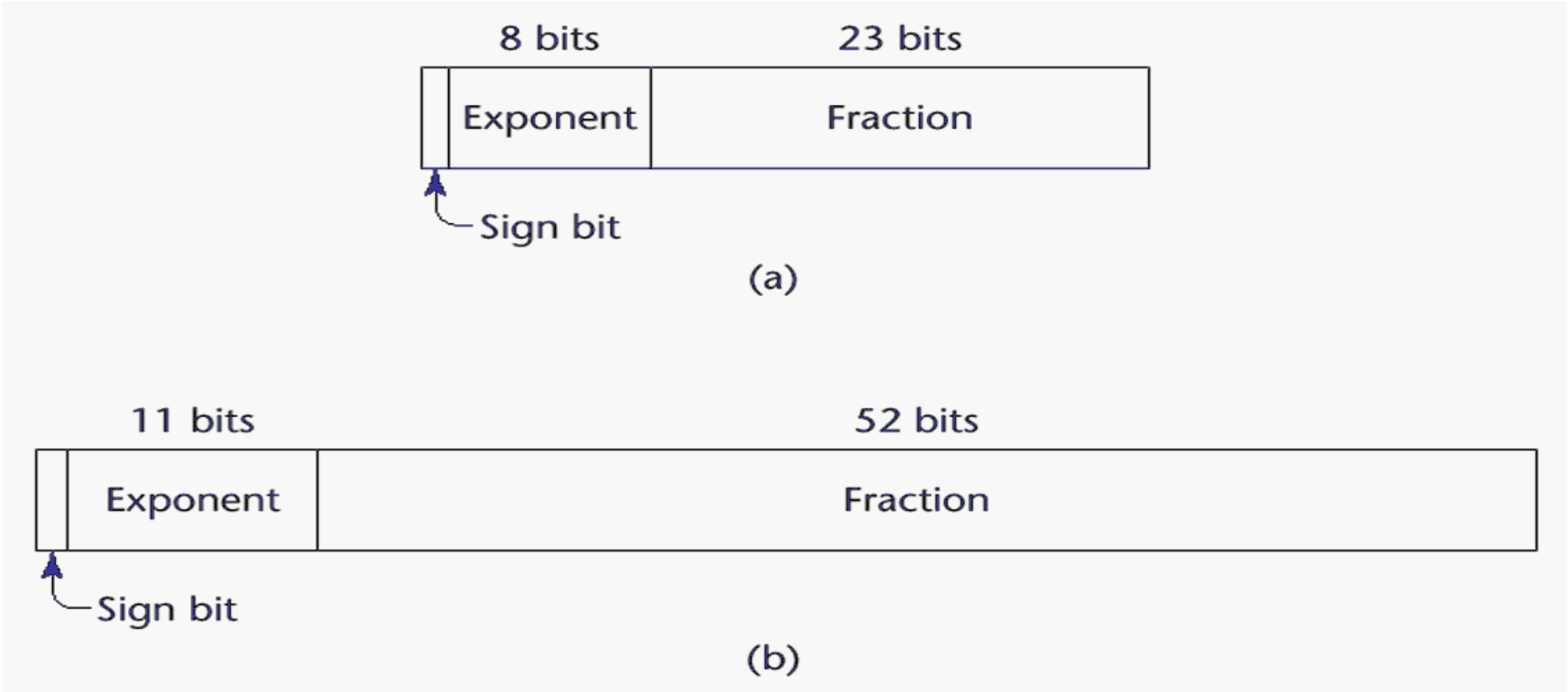
  - Genellikle her zaman donanımın(hardware) tam yansımasıdır, bu yüzden eşlenme(mapping) önemsizdir.
  - Bir dilde en çok sekiz farklı tamsayı (integer) tipi olabilir
- Java'nın integer tipi veri tipleri: **`byte, short, int, long`**

İlkel Veri Tipleri: Kayan Nokta

- Reel Sayıları yalnızca yaklaşım olarak modeller.
- Bilimsel kullanım için olan diller en az iki kayan nokta tipini destekler (Örneğin, **`float`** and **`double`**); Bazen daha fazla.
- Genellikle aynen donanım(hardware) gibidir, fakat her zaman değil
- IEEE 754 Kayan Nokta

Standartı

IEEE 754 Kayan Nokta Standartı



İlkel Veri Tipleri: Kompleks

- Bazı dilleri bu veri tipini destekler, Örneğin: C99, Fortran ve Python
- Her değer iki kısımdan oluşur, birisi reel değer diğer kısım ise imajiner değerdir.
- Python'daki formu aşağıdaki örnekte verilmiştir.:

  (7 + 3j), 7 sayının reel değeri, 3 ise imajiner değeridir.

İlkel Veri Tipleri: Ondalık(Decimal)

- Ticari uygulamalar için kullanılır(para)
  - COBOL temellidir.
    - C# dili decimal veri tipi sunar.
- Sabit ondalık sayıları muhafaza ederler. (BCD) (Ondalıklı sayıların ikilik kodlanması)
- *Avantaj*: Doğruluk
- *Dezavantaj*: Sınırlı aralık, belleği gereksiz harcama.

İlkel Veri Tipleri: Boolean

- En basit veri tipidir.
- Değer aralığında yalnızca iki değer bulunmaktadır. Bunlar true (doğru) ve false (yanlış)'tır.
- Bitler olarak uygulanabilir, fakat çoğu zaman byte kullanılır.
  - Avanaj: Okunabilirlik

İlkel Veri Tipleri: Karakter

- Sayısal kodlama olarak saklanırlar.
- En sık kullanılan kodlama: ASCII (8 bitlik kodlamadır.)
- 16 bitlik alternatif kodlama: Unicode (UCS-2)
  - Çoğu doğal dildeki karakterleri içerir.
  - Başlangıçta Java'da kullanıldı.
  - C# ve JavaScript'de Unicode'u destekleyen diller arasındadır.
- 32-bit Unicode (UCS-4)
  - 2003'te oluşturulmuştur ve fortran tarafından desteklenir

Karakter String Tipleri

- Değerler karakter (char) dizileridir. Örn: char dizi[10]=string deger;
- Tasarım Sorunları:
  1. Bu bir ilkel(primitive) tip midir yoksa sadece bir çeşit özel dizi midir (array)?
  2. Stringlerin uzunluğu statik mi yoksa dinamik mi olmaldır?

Karakter String Tip İşlemleri

- Tipik İşlemler:
  - Atama(Assignment) ve kopyalama
  - Karşılaştırma (Kıyaslama) (Comparison) (=, >, vs.)
  - Birleştirme(Catenation)
  - Altstring referansı (Substring reference)
  - Desen Eşleme (Pattern matching)

Belirli Dillerdeki Karakter String Tipleri

- C ve C++
  - İlkel değil
  - Char dizilerini ve işlemleri sağlayan fonksiyonların kütüphane fonksiyonları kullanır.
- SNOBOL4 (String işleme dili)
  - İlkel
  - Eşleştirme, ayrıntılı desenleme (örüntü tanıma) dahil bir çok işlemi yerine getirebilir.
- Fortran ve Python
  - Atama ve çeşitli operatörleri kullanan ilkel bir tiptir.
- Java
  - String class ile ilkel bir tiptir.
- Perl, JavaScript, Ruby ve PHP

  –Düzenli deyimler kullanılarak string tipinde desen eşleştirme sağlar.

Karakter String Uzunluk Seçenekleri

- Statik: COBOL,Fortran, Java'nın `String` sınıfı
- *Sınırlı Dinamik Uzunlık*: C and C++
  - Bu dillerde, uzunluğu temin etmekten ziyade string karakterlerin sonunu göstermek için özel bir karakter kullanılır.
- *Dinamik* : SNOBOL4, Perl, JavaScript
- Ada, tüm string uzunluk seçeneklerini destekler.

Karakter String Tip Değerlendirmesi

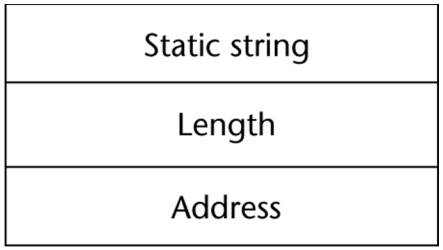
Yazılabilirliğe yardımcıdır

- Statik uzunluklu(Static length) bir ilkel(primitive) tip olarak, temin edilmesi ucuz--neden kullanmayalım ?
- Dinamik uzunluk(Dynamic length) iyidir, fakat masrafa değer mi?

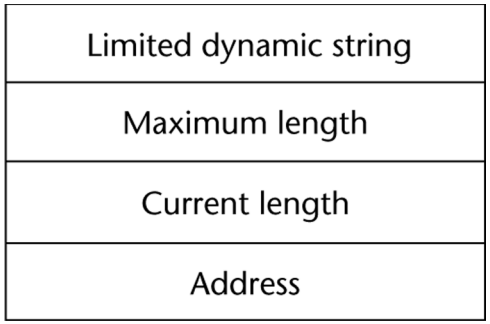
Karakter String Uygulaması

- İmplementasyon (Uygulama):
  - Statik Uzunluk(Static length): Derleme-süresi betimleyicisi (compile-time descriptor)
  - Sınırlı dinamik uzunluk(Limited dynamic length): Uzunluk için bir yürütme-süresi betimleyicisine (run-time descriptor) ihtiyaç duyabilir (fakat C ve C++ da değil)
  - Dinamik Uzunluk(Dynamic length) : Yürütme-süresi betimleyicisine(run-time descriptor) ihtiyaç duyar; atama/atamayı kaldırma (allocation/deallocation) en büyük uygulama problemidir.

Derleme ve Çalışma Zamanı Tanımlayıcıları



Statik stringler için derleme-süresi betimleyici



Sınırlı dinamik stringler için yürütme-süresi betimleyici

Kullanıcı Tanımlı Sıralı Tipler

- Bir sıralı tip(ordinal type), mümkün olan değerler(values) aralığının(range) pozitif tamsayılar(integers) kümesi ile kolayca ilişkilendirilebildiği tiptir.
- Java dilindeki ilkel tip örnekleri
  - **`integer`**
  - **`char`**
  - **`boolean`**

Enumeration(Sayım listesi) Tipleri

- Sabit olarak isimlendirilen tüm olası değerler, tanımda sağlanır.
- C# örneği

```
enum days {mon, tue, wed, thu, fri, sat, sun};
```

- Tasarım Problemi:

  – Bir sembolik sabitin(symbolic constant) birden fazla tip tanımlaması içinde yer almasına izin verilmeli midir? Eğer böyle ise o sabitin ortaya çıkmasının tipi nasıl kontrol edilir?

  – Sayım listesi değerleri tamsayıya zorlanır mı?

  – Diğer bir tip bir sayma tipine zorlanır mı?

Sayım Listesi Tip Değerlendirmesi

- Okunabilirliğe yardım, Örneğin bir sayı olarak bir renk koduna gerek yoktur.
- Güvenilirliğe yardım, Örneğin, derleyici aşağıdakileri kontrol edilebilir:
  - İşlemleri (Renk eklenmesine izin vermez)
  - Sayım listesi değişkeninin dışından bir değer atanmasına izin vermez
  - Ada, C# ve Java 5.0; C++'tan daha iyi sayım listesi desteği sağlar. Çünkü bu dillerdeki sayım listesi değişkenleri tamsayı tiplere zorlanmaz.

Altaralık (Subrange) Tipleri

- Sıralı bir tipteki bir sıralanmıi ardışık alt dizi.
  - Örnek: 12..18 tamsayı tipinin alt aralığıdır.
- Ada'nın tasarımı

```
Günler tipi (mon, tue, wed, thu, fri, sat, sun);
Günler tipinin altaralık tipi haftaiçi günler
  mon..fri;
  Index alt tipi aralığı 1..100 tamsayıdır.
Day1: Days;
Day2: Weekday;
Day2 := Day1;
```

Altaralık Tipleri Değerlendirmesi

- Okunabilirliğe yardım
  - Okuyucuların kolayca görebileceği altaralık değişkenlerini yalnızca belirli aralıkta saklayabiliriz.
- Güvenilirlik
  - Belirlenen değerler dışında altaralık değişkene farklı değerler atamak hata olarak algılanır.

Kullanıcı Tanımlı Sıralı Tiplerin Uygulanması

- Sıralı listeleme (Enumaration) tipleri, tamsayı olarak uygulanır.
- Altaralık tipleri, altaralık değişkenlerine atamaları sınırlamak için (derleyici tarafından) kod yerleştirilmiş ebeveyn(parent) tiplerdir.

Dizi Tipleri

- Bir dizi(array), homojen veri elemanların bir kümesidir, elemanlardan her biri kümedeki birinci elemana göre olan pozisyonuyla tanımlanır.

Dizi Tasarım Problemleri

- 1. Altsimgeler(indeks)(subscripts) için hangi tipler legaldir?
- 2. Eleman referansları aralığındaki altsimgeleme ifadeleri(subscripting expressions) kontrol edilmiş midir?
- 3. Altsimge aralıkları ne zaman bağlanır(bound)?
- 4. Ayırma (allocation) ne zaman olur?
- 5. Altsimgelerin(subscripts) maksimum sayısı nedir?
- 6. Dizi(array) nesneleri(objects) başlatılabilir mi (initialized)?
- 7. Herhangi bir çeşit kesite(slice) izin verilmiş midir?

Dizi İndeksleme

- İndeksleme(Dizin oluşturma)(Indexing) indislerden(indices) elementlere eşleştirme(mapping) yapmaktır

map(array_name, index_value_list) → an element

- İndeks Sentaksı
  - FORTRAN, PL/I, Ada parentezler kullanır
  - Diğer dillerin çoğu köşeli parantez (brackets) ([ ]) kullanır

Dizi İndeksleme (Altsimge) Tipleri

- FORTRAN, C,C#: Yalnızca integer veri tipini kullanır
- Ada: integer yada enumeration (Boolean ve char veri tiplerini içerir.)
- Java: Sadece integer veri tipi.
- Indeks aralık kontrolü
  - C, C++, Perl, ve Fortran özel aralık kontrolü yapmaz.
  - Java, ML, C# dillerinde özel aralık kontrolleri vardır.
  - Ada, Varsayılan aralığı gerektiren kontrolu yapar, ama bazen bu kontrol devre dışı olabilir.

İndis Bağlama ve Dizi Kategorileri

- Dizilerin(Arrays) Kategorileri(altsimge bağlamaya(subscript binding) ve belleğe(storage) bağlamaya dayalıdır)
  1. Statik – altsimgelerin(subscripts) aralığı(range) ve bellek bağlamaları(storage bindings) statiktir

     örn. FORTRAN 77, Ada 'daki bazı diziler(Arrays)
     - Avantaj: uygulama verimliliği(execution efficiency) (ayırma(atama)(allocation) veya serbest bırakma(atamayı kaldırma)(deallocation) yoktur)

İndis Bağlama ve Dizi Kategorileri (Devamı)

2. Sabit yığın dinamik (Fixed stack dynamic)– altsimgelerin (subscripts) aralığı(range) is statik olarak bağlanmıştır, fakat bellek(storage) işlenme zamanında bağlanır
   - örn. Çoğu Java lokalleri, ve `static` olmayan C lokalleri

     Avantaj: alan verimliliği(space efficiency)

İndis Bağlama ve Dizi Kategorileri (Devamı)

3. Yığın Dinamik(Stack-dynamic) – aralık (range) ve bellek(storage) dinamiktir, fakat sonra değişkenin(variable) ömrüne göre(lifetime) sabitlenir
   - örn. Ada bloklar tanımlar

```
declare
  STUFF : array (1..N) of FLOAT;
  begin
  ...
  end;
```

   - Avantaj: esneklik – dizi(array) kullanılmaya başlanmadan önce boyutu(size) bilinmek zorunda değildir.

İndis Bağlama ve Dizi Kategorileri (Devamı)

4. Heap-dynamic(Dinamik Öbekler) – altsimge(subscript) aralığı (range) ve bellek bağlamalar(storage bindings) dinamiktir ve sabitlenmiş değildir
   - Avantaj: esneklik(diziler programın çalıştırılması esnasında büyültülebilir veya küçültülebilir.)
   - APL'de, Perl, ve JavaScript, diziler(Arrays) ihtiyaca göre büyüyüp küçülebilir
   - Java ve C#'ta, bütün diziler(Arrays) birer nesnedir (heap-dynamic)

İndis Bağlama ve Dizi Kategorileri (Devamı)

- C ve C++'ta statik niteleyiciler içeren diziler statiktir.
- C ve C++'ta statik niteleyiciler içermeyen diziler sabit yığın (fixed stack) –dinamiktir.
- C ve C++ sabit öbek(fixed heap) ve dinamik dizileri destekler.
- C# 2. dizi sınıfı olan ArrayList'i destekler ve sabit öbeklerle dinamik diziler oluşturulabilir.
- Perl, JavaScript, Python, ve Ruby dinamik öbek dizilerini destekler(sağlar).

Dizi Oluşturma

- Bazı diller dizi oluşturma ve oluşturulan diziye bellek alanı ayırma işlemini aynı anda yapabilir.
  - C, C++, Java, C# örneği

```
int list [] = {4, 5, 7, 83}
```

  - C ve C++'ta karakter dizisi oluşturma

```
char name [] = "Asaf Varol";
```

  - C ve C++'ta pointerlar yardımıyla string dizisi oluşturma

```
char *names [] = {"Bob", "Jake", "Joe"];
```

  - Java'da string nesnesi oluşturma

```
String[] names = {"Bob", "Jake", "Joe"};
```

Heterojen Diziler

- Heterojen dizi, elemanları aynı tipten olması gerekmeyen dizilerdir.
- Perl, Python, JavaScript ve Ruby tarafından desteklenir.
- Diğer dillerde heterojen diziler yerine struct(yapılar) kullanılır, fakat yapılar heterojen diziyi tam manasıyla karşılayamazlar.

Dizi Oluşturma

- C-tabanlı diller
  - `int list [] = {1, 3, 5, 7}`
  - `char *names [] = {"Mike", "Fred", "Mary Lou"};`
- Ada
  - `List : array (1..5) of Integer :=`
    `(1 => 17, 3 => 34, others => 0);`
- Python
  - Liste Kapsamları

```
list = [x ** 2 for x in range(12) if x % 3 == 0]
puts [0, 9, 36, 81] in list
```

Dizi İşlemleri

- APL vektörler ve matrisler için hem en güçlü dizi işleme işlemleri hem de tekli operatör desteği (örneğin, kolon elemanları tersine çevirmek için) sağlar
- Ada yalnızca dizilerde atama,birleştirme ve ilişkisel operatör işlemlerine izin verir.
- Pyton'nun dizi atamaları yalnızca referans değişikliği işlemlerini yapmasına rağmen eleman üyelik sistemiyle Ada'nın sağladığı tüm işlemleri yapabilir.
- Ruby dizilerde atama,birleştirme ve ilişkisel operatör işlemlerine izin verir
- Fortran iki dizi arasındaki element işlemlerini destekler
  - Örneğin Fortrandaki + operatörü iki dizi çiftleri arasındaki elemanları toplar.

Dikdörtgen(Düzenli) ve Tırtıklı (Düzensiz) Diziler

- Bir dikdörtgen dizi satırları tüm unsurlarının aynı sayıda ve tüm sütun elemanlarının aynı sayıya sahip olduğu çok boyutlu dizidir.
- Tırtıklı matrisler ise her satırında aynı sayıda eleman bulunmayan dizilerdir.
  - Olası çoklu-boyutlu zaman dizileri aslında dizinler olarak görünür
  - C, C++, ve Java tırtıklı(düzensiz) dizileri destekler
- Fortran, Ada, ve C# dikdörtgen dizileri destekler (C# aynı zamanda tırtıklı dizileri de destekler)

Kesitler

- Kesitler(slices)
  - Kesit(slice) , bir dizininin bir kısım altyapısıdır (substructure) ; bir referanslama mekanizmasından fazla birşey değildir
  - Kesitler(slices) sadece dizi işlemleri olan diller için kullanışlıdır.

Kesit Örnekleri

- Python

```
vector = [2, 4, 6, 8, 10, 12, 14, 16]
mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

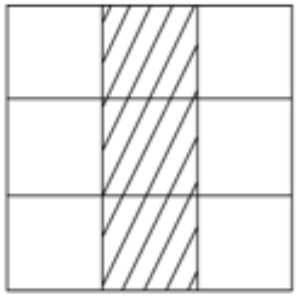
`vector (3:6)` dizinin 3 elemanını temsil eder.

`mat[0][0:2]` dizinin ilk satırındaki ilk elemandan 3. elemana kadar olanı gösterir.

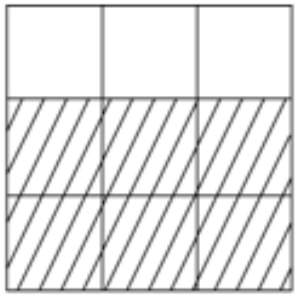
- Ruby slice metot olarak kesiti destekler.

`list.slice(2, 2)` örneğinde 2. elemandan 4. elemana kadar olanları kapsar.

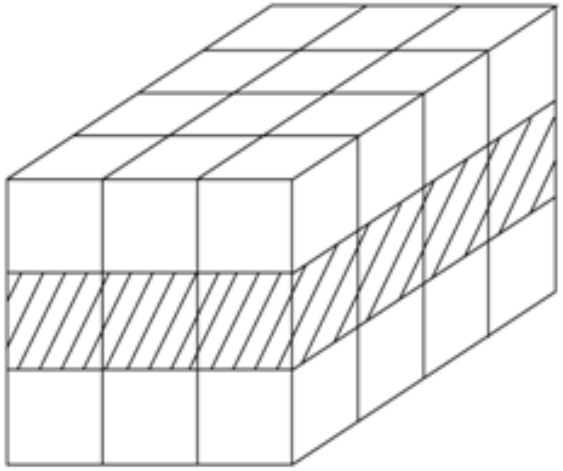
Kesitler



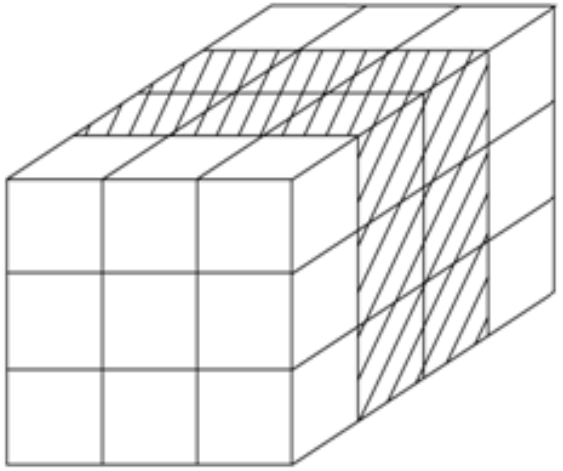
MAT (1:3, 2)



MAT (2:3, 1:3)



CUBE (2, 1:3, 1:4)

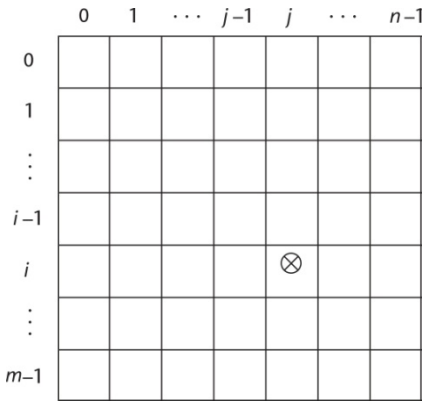


CUBE (1:3, 1:3, 2:3)

Dizi Uygulamaları

- Erişim fonksiyonları(Access function) altsimge(subscript) ifadelerini dizideki bir adrese eşler(map)
- Tek boyutlu diziler için erişim fonksiyonu:

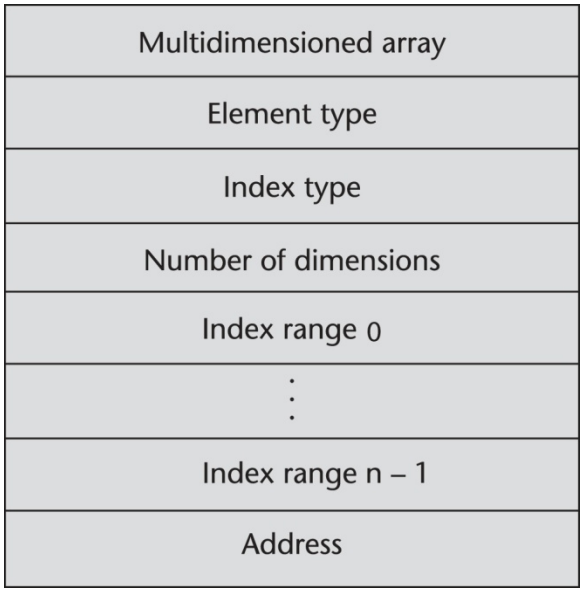
  address(list[k]) = address (list[lower_bound])
  + ((k-lower_bound) * element_size)



Çok Boyutlu Dizilere Erişim

- Yaygın olarak kullanılam metotlar:
  - Satıra göre sıralama – bir çok dilde kullanılan metottur
  - Sutüna göre sıralama – Fortran tarafından kullanılır
  - Çok boyutlu dizilerin

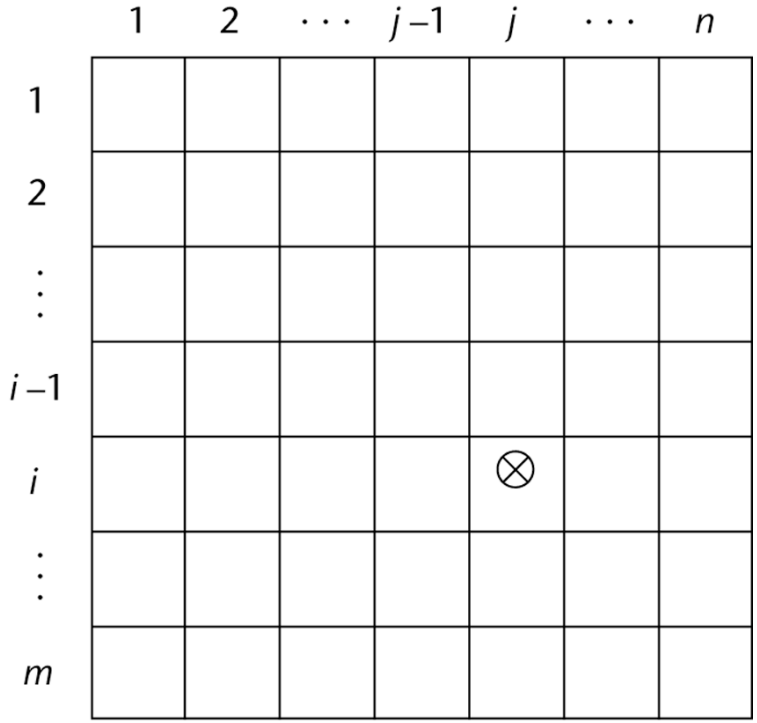
derleme süreleri yan taraftaki şekilde verilmiştir.



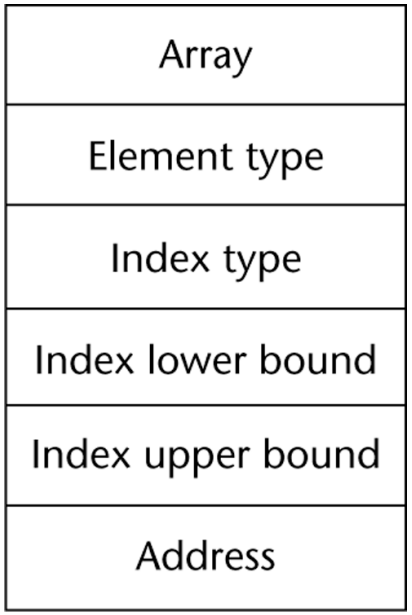
Çok Boyutlu Dizilerde Eleman Yerleştirme

- Genel format

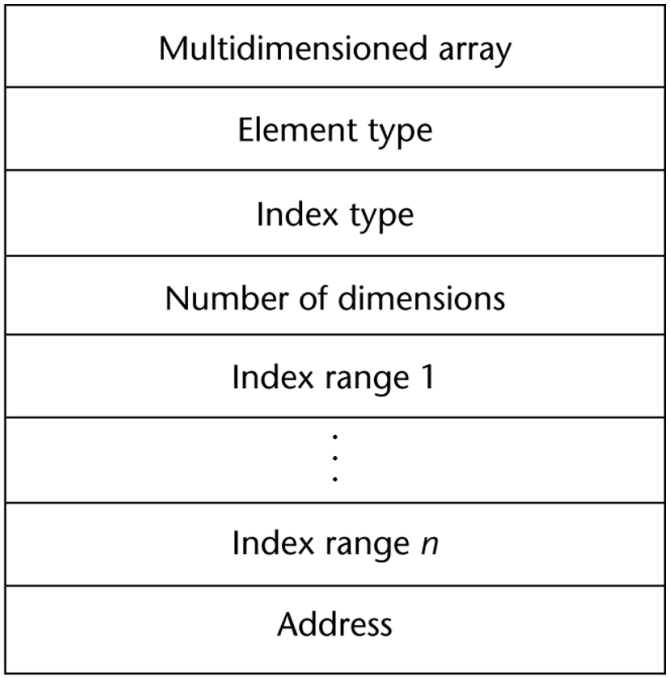
  Location (a[I,j]) = address of a [row_lb,col_lb] + (((I – row_lb) * n) + (j – col_lb)) * element_size



Derleme Süresi Betimleyiciler



Tek Boyutlu Dizi



Çok Boyutlu Dizi

İlişkili Diziler

- Bir ilişkili dizi(associative array), anahtar(key) adı verilen eşit sayıda değerlerle indekslenmiş veri elemanlarının sırasız bir koleksiyonudur.

- Tasarım Problemleri:
  1. Elemanlara referansın şekli nedir?
  2. Boyut statik midir yoksa dinamik mi?

Perl'de İlişkili Diziler

- İsimler% ile başlar; değişmezleri parantez tarafından ayrılmış

```
%hi_temps = ("Mon" => 77, "Tue" => 79, "Wed" =>
  65, …);
```

- İndisleme küme parantezi ve $ kullanılarak yapılır.

```
$hi_temps{"Wed"} = 83;
```

  - Elemanlar delete komutuyla silinir.

```
delete $hi_temps{"Tue"};
```

Kayıt Tipleri

- Bir kayıt ayrı eleman isimleri tarafından tanımlandığı bir veri elemanlarının heterojen toplamıdır.
- Tasarım problemleri:
  1. Referansların şekli nedir?
  2. Hangi birim işlemler tanımlanmıştır?

COBOL'da Kayıt Tanımlanması

- COBOL yuvalanmış kayıtları göstermek için seviye numaraları kullanır; diğerleri için özyinelemeli tanımı kullanabilirsiniz.
-
```
01 EMP-REC.
    02 EMP-NAME.
       05 FIRST PIC X(20).
       05 MID   PIC X(10).
       05 LAST  PIC X(20).
    02 HOURLY-RATE PIC 99V99.
```

Ada ile Kayıt Tanımlanması

- Kayıt yapıları ortagonol bir şekilde gösterilir.

```
type Emp_Rec_Type is record
    First: String (1..20);
    Mid: String (1..10);
    Last: String (1..20);
    Hourly_Rate: Float;
end record;
Emp_Rec: Emp_Rec_Type;
```

Kayıt Referansları

- Kayıt alanındaki referanslar
  1. COBOL
  field_name OF record_name_1 OF ... OF record_name_n
  2. Others (dot notation)
  record_name_1.record_name_2. ... record_name_n.field_name

- Kaliteli referanslar(references) bütün kayıt adlarını(record names) içermelidir

- Eliptik referanslar(Elliptical references), referans(reference) belirsiz olmadığı(unambiguous) sürece kayıt adlarının (record names) ihmal edilmesine izin verir

  FIRST, FIRST OF EMP-NAME, and FIRST of EMP-REC are elliptical references to the employee's first name

Kayıt İşlemleri

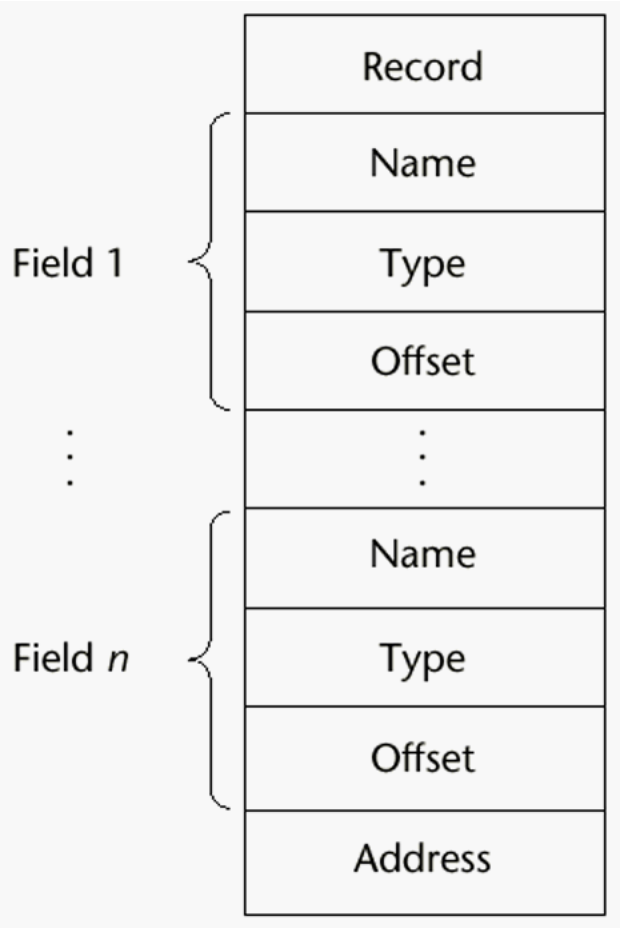
1. Atama(Assignment)
- Pascal, Ada, ve C tipleri özdeş(identical) ise izin verir
- Ada'da, sağ kısım(RHS) bir toplam sabit(aggregate constant) olabilir
2. Başlatma(Initialization)
- Ada'da izin verilmiştir, toplam sabit(aggregate constant) kullanarak
3. Kıyaslama(Comparison)
- Ada'da, = ve /=; bir operand toplam sabit(aggregate constant) olabilir
4. **`MOVE CORRESPONDING`**
- COBOL'de – kaynak kayıttaki(source record) bütün alanları(fields) hedef kayıttaki(destination record) aynı ada sahip alanlara(fields) taşır

Evaluation and Comparison to Arrays

1. Dizi(array) elemanlarına erişim kayıt alanlarına (record fields) erişimden daha yavaştır, çünkü altsimgeler(subscripts) dinamiktir (alan adları(field names) statiktir)
2. Dinamik altsimgeler(subscripts) kayıt alanına(record field) erişimle kullanılabilirdi, fakat o zaman tip kontrolüne(type checking) izin vermeyecekti ve çok daha yavaş olacaktı

Kayıt Tipinin Uygulanması

Kayıtların başlangıcına göre
Ofset adresi her alanı ile ilişkilidir.



Demet (Tuple) Tipi

- Demet veri tipi kayıt veri tipine benzeyen bir veri tipidir.
- Python,ML ,F#,C#(.Net 4.0 ile birlikte)'ta kullanılır.Fonksiyonlara birden fazla değer döndürür.
  - Python
    - Listelerle yakından ilişkili ama değiştirilemez
    - Demet oluşturma

      ```
      myTuple = (3, 5.8, 'apple')
      ```

      İndislerini 1'den başlayaraka referanslandırır.

      + operatörünü kullanır ve del komutuyla silinir.

Demet(Tuple) Tipi (Devamı)

- ML

```
val myTuple = (3, 5.8, 'apple');
```

  - Takipçilere erişim:

    `#1(myTuple)` demetin ilk elemanı

  - Yeni bir demet aşağıdaki gibi tanımlanır.

```
type intReal = int * real;
```

- F#

```
let tup = (3, 5, 7)
let a, b, c = tup
```

Liste Tipleri

- LISP ve Şema listeleri parantez ayracıyla kullanılırlar ve elemanlar arasına virgül konulmaz.

  `(A B C D)` and `(A (B C) D)`

- Veri ve kod aynı formdadır.

  Veri, `(A B C)`

  Kod, `(A B C)` bir fonksiyonun parametreleri

- Yorumlayıcı hangi listeye ihtiyaç duyacağını bilmelidir. Burdaki karmaşıklığı ortadan kaldırmak için veri listelerinin önüne ‘ işareti konur.

  `'(A B C)` is data

Liste Tipleri (devamı)

- Şema içerisindeki Liste operatörleri
  - `CAR` listesi ilk elemanını döndürürse

    `(CAR '(A B C))` returns `A`
  - `CDR` ilk elemanı söküldükten sonra kendi listesinde parametresi kalanı verir.

    `(CDR '(A B C))` returns `(B C)`
  - `CONS` Yeni bir liste yapmak için ikinci parametre, bir liste içine ilk parametre koyar.

    `(CONS 'A (B C))` returns `(A B C)`
  - `LIST` yeni bir liste döndürür.

    `(LIST 'A 'B '(C D))` returns `(A B (C D))`

Liste Tipleri (devamı)

- ML'de Liste Operatörleri
  - Listeler parantez içinde yazılır ve elemanları virgüllerle ayrılır.
  - Liste elemanları aynı veri tipinde olmalıdır.
  - `CONS` fonksiyonu ML dilinin binary operatörüdür, `::`

    `3 :: [5, 7, 9]` dönüşür `[3, 5, 7, 9]`
  - CAR ve CDL fonksiyonları burda hd ve tl olarak adlandırılır.

Liste Tipleri (devamı)

- F# Listeler
  - ML dilindeki liste yapısına benzer, yalnızca elemanların ayrılmasıyla hd ve tl metotları List sınıfının içinde yer alır
- Python Listeler
  - Liste veri tipi genelde python dizileri olarak sunulur
  - Genelde LISP,ML,F# ve Python listeleri birbirine benzer.
  - Listedeki elemanlar değişik veri tiplerinden olabilirle
  - Liste oluşturulması aşağıdaki gibidir.

```
myList = [3, 5.8, "grape"]
```

Liste Tipleri (continued)

- Python Listeler (devamı)
  - Liste indisi "0"dan başlar ve sonradan değiştirilebilir.

```
x = myList[1]
```
Sets `x` to `5.8`

  - Liste elemanları del komutuyla silinir.

```
del myList[1]
```

  - Liste anlamları – küme gösterimiyle temsil edilebilir.

```
[x * x for x in range(6) if x % 3 == 0]
```

`range(12)` creates `[0, 1, 2, 3, 4, 5, 6]`

Constructed list: `[0, 9, 36]`

Liste Tipleri (continued)

- Haskell'in Liste Anlamları
  - Orjinal

```
[n * n | n <- [1..10]]
```

- F#'in Liste Anlamları

```
let myArray = [|for i in 1 .. 5 -> [i * i) |]
```

- C# ve Java dilleri de listeleri destekler.Kendi dinamik koleksiyonlarında List ve ArrayList adında sınıfları vardır.

Birleşim Tipi

- Bir bileşim(union), değişkenlerinin(variable) yürütme süresi sırasında farklı zamanlarda farklı tipteki değerleri tutmasına izin verildiği tiptir
- Bileşimler(unions) için tasarim problemleri :
  1. Eğer varsa hangi tip tip kontrolü yapılmalıdır?
  2. Bileşimler(unions) kayıtlar(Records) ile entegre edilmeli midir?

Serbest Birleşimleri Ayırmak

Fortran, C ve C++ – serbest bileşimler(unions) (etiketler yoktur(tags))

- kayıtlarının(Records) kısımları yoktur
- Referanslarda tip kontrolü yoktur

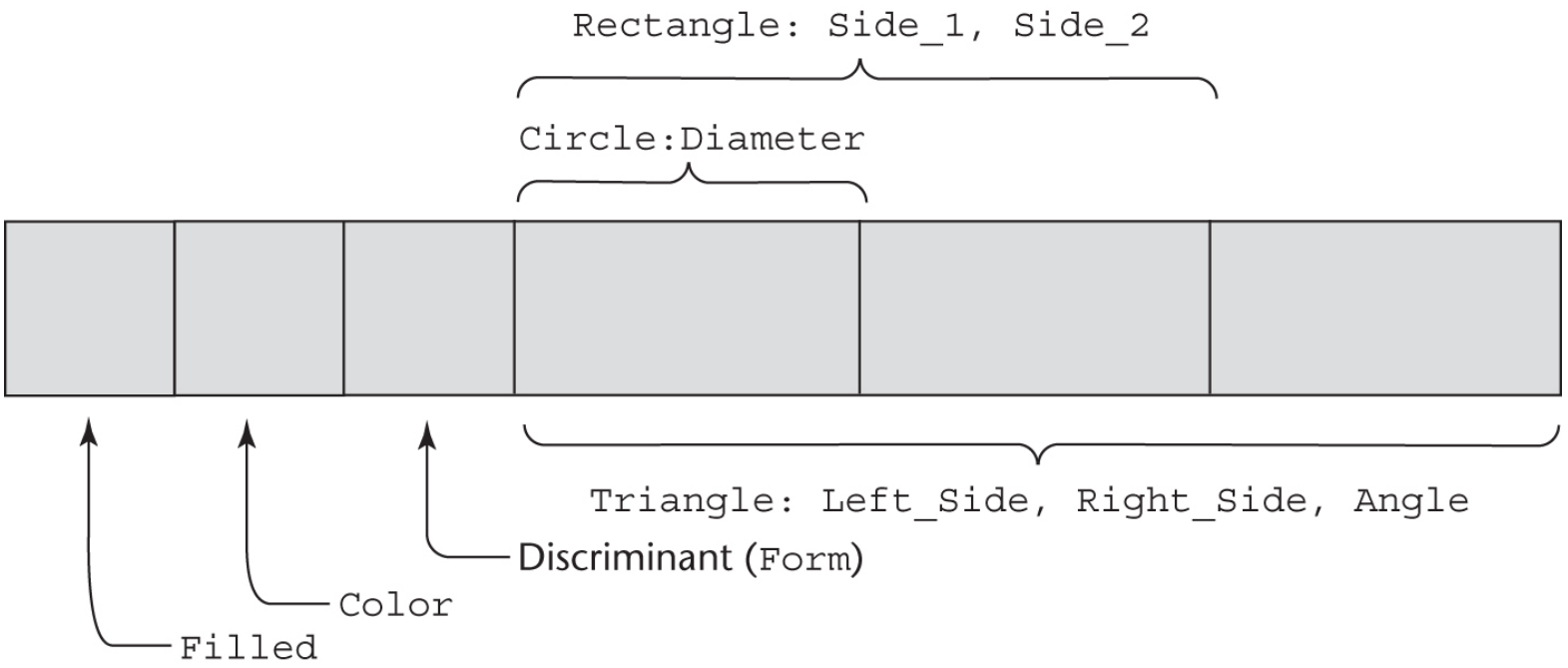
Java 'da kayıtlar(Records) da bileşimler(unions) de yoktur

- Değerlendirme – çoğu dilde güvensiz görünmektedir (Ada tek güvenilir dildir. Çünkü bileşim kontrolu yapar.)

Ada Birleşim Tipi

```
type Shape is (Circle, Triangle, Rectangle);
type Colors is (Red, Green, Blue);
type Figure (Form: Shape) is record
   Filled: Boolean;
   Color: Colors;
   case Form is
       when Circle => Diameter: Float;
       when Triangle =>
             Leftside, Rightside: Integer;
             Angle: Float;
       when Rectangle => Side1, Side2: Integer;
   end case;
end record;
```

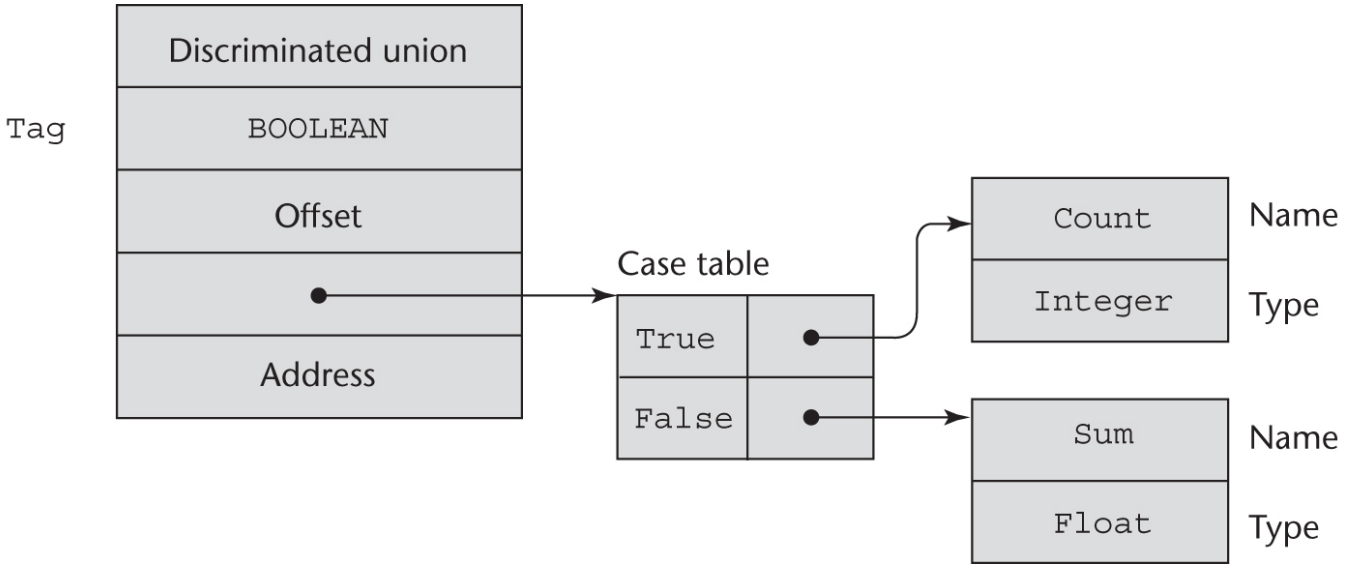
Ada Birleşim Tipi Örneği



Bir ayrılmış birleşimde 3 şeklin değeri gösterilmektedir.

Birleşimleri Uygulanması

```
type Node (Tag : Boolean) is
  record
    case Tag is
      when True => Count : Integer;
      when False => Sum : Float;
    end case;
  end record;
```



Birleşimlerin Değerlendirilmesi

- Serbest birleşimler güvenilir değildir.
  - Tip kontrolune izin vermezler.
- Java ve C# birleşimleri desteklemez.
  - Programlama dilinin güvenilirliğini artırmak için
- Ada'nın ayrık birleşimleri güvenilirdir.

Pointer ve referans Tipleri

- Bir işaretçi(pointer) tipi değişkeni oluşturan bellek adres aralığını tutan özel bir değerdir.
- Dolaylı adresleme gücünü sağlar
- Dinamik hafıza kullanmayı sağlayan bir yoldur.
- Bir işaretçi dinamik depolama olarak oluşturulan bir konuma ulaşmak için kullanılabilir. (Genellikle dinamik bir öbek)

Pointerların Tasarım Problemleri

- Pointer'in kapsamı ve ömrü nedir ?
- Dinamik öbek değişkenlerinin ömrğ nedir?
- Işaretçiler bu konuda ortaya koyabildikleri değer türü sınırlı mı?
- Pointerlar dinamik depolama için mi kullanılıyorlar yoksa dolaylı adresleme için mi ya da her ikisi için mi kullanılıyor?
- Kullandığın dil pointer tipini mi destekliyor yoksa referans tipini mi yoksa her ikisini mi destekliyor?

Pointer Operatörleri

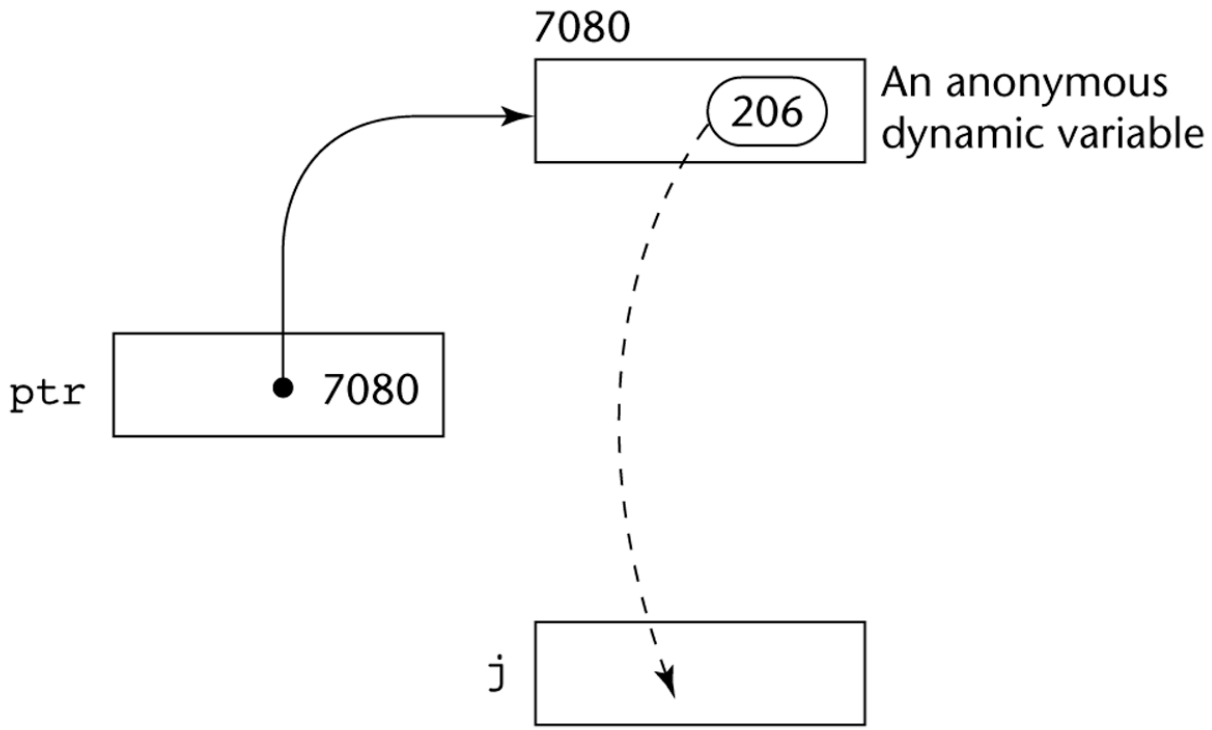
- İki temel işlemleri: atama ve başvurusu kaldırma.
- Atama bazı değişkenlerin değerini değiştirmekle birlikte bazı değişkenlerinde adres değerini değiştirir.
- Başvurusu işaretçi değeri ile temsil edilen bir konumda saklanan değeri verir.

  - C++ pointerlar için * operatorunu kullanır.

    ```
    j = *ptr
    ```

  J değerini ptr'nin gosterdiği adres değerine atadı.

Pointer Assignment Illustrated



Atama işlemi j = *ptr

7080 adresindeki 206 değerini j değişkenine atadı

Pointerlarla İlgili Problemler

1. Dangling(askıdaki-boşta kalan) İşaretçiler(Pointers) (tehlikeli)

- Bir işaretçi(pointer) serbest bırakılmış(deallocate) bir heap-dinamik değişkene işaret eder
- Bir tane oluşturmak (harici(explicit deallocation)):

a. Bir heap-dinamik değişken ayırma ve pointerı bunu göstermeye ayarlama

b. Birinci pointerın değerine ikinci bir pointerı atama

c. Birinci pointerı kullanarak heap-dinamik değişkeni serbest bırakma(deallocate)

Pointerlarla İlgili Problemler(devamı)

2. Kayıp(Lost) Heap-Dinamik Değişkenler (savurgan)

- Bir program işaretçisi(program pointer) tarafından artık referans edilmeyen heap-dinamik değişken(variable)
- Bir tane oluşturmak:

a. Pointer p1 yeni oluşturulmuş bir heap-dinamik değişkeni göstermeye ayarlanır

b. p1 daha sonra diğer bir yeni oluşturulmuş heap-dinamik değişkeni göstermeye ayarlanır

- Heap-dinamik değişkenleri kaybetme işlemine memory leakage(bellek sızıntısı) denir

Ada'da Pointer Problemleri

- Boşta kalan pointerlara izin vermez çünkü dinamik nesneler otomatikman boşta kalan pointerları yok eder.
- Kayıp dinamik değişken problemi hala çözülememiştir. (Belki `UNCHECKED_DEALLOCATION`)

C ve C++'ta Pointerlar

- Dinamik Bellek Yönetimi ve adresleme için kullanılır
- Tanım Kümesi Tipi(Domain type) sabit olmak zorunda değildir (`void *`)
- `void *` – herhengi bir tipe işaret edebilir ve tip kontrolü yapılabilir (dereference yapılamaz)
- Bu dillerde pointer kullanımı esnektir.

C ve C++'ta Pointer Aritmetiği

```
float stuff[100];
float *p;
p = stuff;
```

`*(p+5)` eşittir. `stuff[5]` ve `p[5]`

`*(p+i)` eşittir `stuff[i]` ve `p[i]`

Referans Tipi

- C + + formal parametreleri için öncelikle kullanılan bir başvuru türü denilen işaretçi türü özel bir tür içerir.
- Avantajı: Hem referans değerini hem de veri değerini verebilir.
- Java C++'in referans değerini uzatarak pointerların sadece referans değeri tutmasını sağlar.
  - Referanslar yerine adresleri olmaktan çok, nesnelere başvurular vardır.
- C# hem Java'nın nesne modelini hemde C++'nın referans modelini kullanmaktadır.

Pointerların Değerlendirilmesi

1. Askıda İşaretçiler(Dangling Pointers) ve Askıda Nesneler (dangling objects) problemlerdir, heap yönetiminde olduğu gibi
2. İşaretçiler(Pointers) goto'lar gibidir– bir değişkenin(variable) erişebileceği hücreler(cells) aralığını(range) genişletirler
3. İşaretçiler(Pointers) veya referanslar dinamik veri yapıları için gereklidir—bu yüzden onlar olmadan bir dil tasarlayamayız

Pointerlar

- Büyük bilgisayarlar basit değerler kullanır.
- Intel mikroişlemciler(microprocessors) kesim(segment) ve ofset(göreli-konum) (offset) kullanır.

Askıda Pointer Problemi

1. Tombstone: heap-dinamik değişkene işaretçilik yapan ekstra bir heap hücresi(cell)
   - Gerçek işaretçi değişkeni(actual pointer variable) sadece tombstone'lara işaret eder
   - heap-dinamik değişken serbest bırakıldığı zaman (deallocated), tombstone kalır fakat nil'e ayarlanır

.*Kilit ve Anahtar*: Pointerlar anahtar ve adres olmak üzere iki tip değeri temsil eder.

- Heap-dinamik değişkenleri tamsayı kilit değeri için değişken gözeler olarak temsil edilir.
- Yığın-dinamik değişken ayrılan zaman, kilit değeri oluşturulur ve kilit hücre ve işaretçi anahtar hücresine yerleştirilir.

Yığın(Öbek) Yönetimi

- Çok karmaşık çalışma zamanı
- Tek boyutlu hücreler veya değişken boyutlu hücreler
- Çöp verileri kurtarmak için kullanılan yaklaşımlar
  - Referans sayaçları (*istekli yaklaşım*): Kurtarma işlemi kademi olarak yapılır
  - İşaretle ve Süpür (*uyuşuk yaklaşım*): değişken alan listesi boş olduğunda kurtarma başlar.

Reference Counter

Referans Sayaçlar(Reference counters): her bir hücrede o anda o hücreyi gösteren işaretçilerin sayısını tutan bir sayaç(counter) sürdürmek

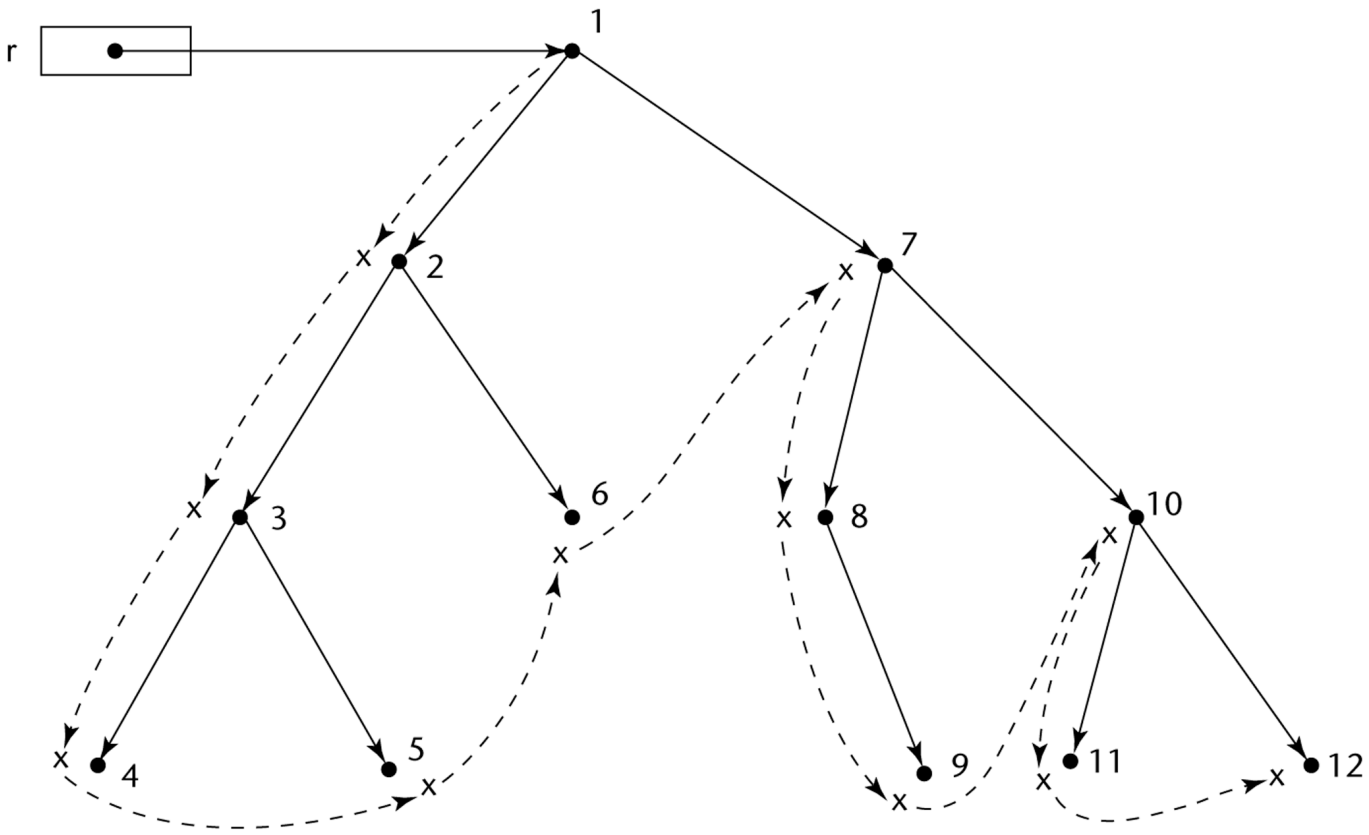
- Dezavantajlar: boş alan gerekir, yürütme zamanı(execution time) gerekir, dairesel olarak bağlanmış hücreler için komplikasyonlar.
- *Avantaj*: Uygulama yürütmedeki önemli gecikmeler engellenir

İşaretle-Süpür

Eldeki bütün hücreler ayrılmış(allocated) olana kadar ayrılır(allocate) ve bağlantı kesilir (disconnect); sonra bütün atık (garbage) toplanmaya başlanır

- Her heap hücresinin(cell) collection algorithm tarafından kullanılan ekstra bir biti vardır
- Bütün hücreler başlangıçta atığa ayarlanır
- Bütün işaretçiler(Pointers) heap içine kopya edilir, ve erişilebilir hücreler atık-değil olarak işaretlenir
- Bütün atık hücreler eldeki hücreler listesine geri döndürülür
- Dezavantajlar: en çok ihtiyaç duyduğunuz zaman, en kötü çalışır (program heap deki hücrelerin çoğuna ihtiyaç duyduğunda en çok zamanı alır)

İşaretleme Algoritması



Dashed lines show the order of node_marking

Değişken Boyutlu Hücreler

- Her yönüyle tek boyutlu hücrelerden daha zordur.
- Çoğu programlama dilinde olması gerekir.
- Eğer işaretle-süpür algoritması kullanılıyorsa ek problemler meydana gelir. Bunlar;
  - Tüm hücre göstergelerinin başlangıç ayarı zordur.
  - İşaretlenen işlem ziyaret edilmemişse problem büyür.
  - Kullanılabilir alan listesini bakımı yükü artar.

Tip Kontrolü

- Altprogramlar ve atamaları içerecek şekilde işlenen ve operatörler kavramını yaygınlaştırmak.
- *Tip denetleme bir operatorun işlenen tipin uyumunu sağlama faaliyetidir.*
- Uyumlu tipin üretiminde ve denetiminde derleyicinin ürettiği kod ve kurallar göz önüne alınır.
- Bu otomatik dönüşüm bir zorlama denir.Örn: double a=15; int k=a;
- Bir tip hatası desteklenmeyen tarzdaki bir tipin o veri tipi kümesinde yorumlanmaya çalışılmasıdır

Tip Kontrolü (devamı)

- Eğer tüm tip bağlayıcıları statikse, tip denetimi de statik olarak yapılır.
- Eğer tip bağlayıcıları dinamikse tip kontrolünün dinamik yapılması zorunludur.
- Tip hataları her zaman tespit edilirse programlama dilindeki tipler güçlüdür.
- Güçlü tiplerin avantajları: Hata algılama değişkenleri tip hatası sonucunu kolayca verir.

Güçlü Tipler

Dil Örnekleri:

- C ve C++ :parametre tür denetlemesi önlenebilir; birleşimler kontrol türü değildir.
- Ada'da (`UNCHECKED CONVERSION` kodu bir gözetleme deliğidir.)

  (Java and C# Ada'ya benzer)

Güçlü Tipler (devamı)

- Zorlama kuralları güçlü tiplerde güçlü efektler oluşturabilir veya onları zayıflatabilir. (C++ ve Ada)

- Java'da sadece C++'ın yarım atama kuralları olmasına rağmen,güçlü tiplerde Ada'dan daha az etkindir.

Tip Adı Eşitleme

- Tip Adı Eşitleme'nin manası ya aynı tipi yada aynı tip adını kullanarak değişkenleri birbirine eşitlemektir. Bu metot kullanıldığında iki değişkende eşdeğer tip var demektir.
- Uygulamada basit fakat hayli kısıtlayıcıdır:
  - Alt aralıklarda tanımlanan integer tipler örn. Notlar (1,100) integer tipine eşit değildir.
  - Örgün parametreleri, karşılık gelen gerçek parametreleri aynı türde olmalıdır.

Yapı Tipi Eşitleme

- Yapı tipi eşitleme, aynı yapıları varsa iki değişken eşdeğer tip olması anlamına gelir.
- Çok esnek fakat uygulaması çok zor.

Tip Eşitleme (devamı)

- Iki yapısal tip sorununu ele alalım:
  - Yapısal olarak aynı ama farklı alan adları kullanırsanız iki kayıt türünü eşitler misiniz?
  - İki dizi türünde, simgeler farklı olması dışında aynı ise bu diziler eşdeğer midir?

    (e.g. `[1..10]` and `[0..9]`)
  - İki sıralama tipi onların bileşenleri farklı yazıldığından eşdeğer olur mu?

Veri Tipleri Teorisi

- Tip teorisi matematik, mantık, bilgisayar bilimleri ve felsefe çalışmasını kapsayan geniş bir disiplinler arası alandır.
- Bilgisayar bilimlerinde tip teorisi iki ana dala ayrılmıştır.
  - Pratik – Ticari dillerdeki veri türleri
  - Soyut – İleri matematiksel hesaplamalar için kullanılır.
- Bir tip sistemi tipleri ayarlayan ve kuralları yöneten programları kullanır.

Veri Tipleri Teorisi (devamı)

- Bir tip sisteminde biçimsel model tipleri kümesi ve tip kuralları tanımlayabilirsiniz.
  - Tipleri belirlemek için dilbilgisi kuralları veya haritalar kullanılır.
  - Sonlu haritalama – model diziler ve fonksiyonlar
  - Kartezyan ürünler – model demetler ve kayıtlar
  - Birleşimleri ayarlama – Model birleşim tipleri
  - Altayarlar – model alttipler

Summary

- Veri tipleri, bir dilin kullanışlılığını belirleyen en büyük parçasıdır.
- Çoğu dilde zorunlu olarak yer alan ilkel veri türleri sayısal, karakter, ve Boolean türlerini içerir.
- Kullanıcı tanımlı numaralandırma ve alt aralık tipleri, programların okunabilirliği ve güvenilirliğini arttırır.
- Diziler ve kayıtlar birçok dilde bulunur.
- Pointerlar esnek adresleme ve dinamik bellek kontrolü yönetiminde kullanılan veri tipleridir.