PROGRAMLAMA DİLLERİ

Step	Parse Stack	Look Ahead	Unscanned	Parser Action
0	*empty*	*id*	= B + C*2	Shift
1	*id*	=	B + C*2	Shift
2	*id* =	*id*	+ C*2	Shift
3	*id* = *id*	+	C*2	Reduce by Value ← *id*
4	*id* = Value	+	C*2	Reduce by Products ← Value
5	*id* = Products	+	C*2	Reduce by Sums ← Products
6	*id* = Sums	+	C*2	Shift
7	*id* = Sums +	*id*	*2	Shift
8	*id* = Sums + *id*	*	2	Reduce by Value ← *id*
9	*id* = Sums + Value	*	2	Reduce by Products ← Value
10	*id* = Sums + Products	*	2	Shift
11	*id* = Sums + Products *	*int*	*eof*	Shift
12	*id* = Sums + Products * *int*	*eof*		Reduce by Value ← *int*
13	*id* = Sums + Products * Value	*eof*		Reduce by Products ← Products * Value
14	*id* = Sums + Products	*eof*		Reduce by Sums ← Sums + Products
15	*id* = Sums	*eof*		Reduce by Assign ← *id* = Sums
16	Assign	*eof*		Done

Grammar

Assign ← *id* = Sums
Sums ← Sums + Products
Sums ← Products
Products ← Products * Value
Products ← Value
Value ← *int*
Value ← *id*

Konular

- Giriş
- İlkel (Primitive) Veri Tipleri
- Karakter (Character) String Tipleri
- Kullanıcı-tanımlı Sıra (Ordinal) Tipleri
- Dizi (Array) Tipleri
- Kayıt (Tutanak) (Record) Tipleri
- Bileşim (Union) Tipleri
- İşaretçi (Pointer) Tipleri

GİRİŞ

İlkel ve yapısal veri tipleri arasındaki en önemli fark, ilkel veri tiplerinin başka veri tiplerini içermesidir.

- Bir problemin çözümü esnasında bilgisayarda tutulan, CPU komutu olmayan her türlü bilgiye **veri** denir.
- Bir **veri tipi**, bir değerler kümesini ve bu değerler üzerindeki işlemleri tanımlar.
- Bir programlama dilindeki veri tipleri, programlardaki ifade yeteneğini ve programların güvenilirliğini doğrudan etkiledikleri için, bir programlama dilinin değerlendirilmesinde önemli bir yer tutarlar.
- İlkel (temel) ve yapısal veri tipleri arasındaki en önemli fark, ilkel veri tiplerinin başka veri tiplerini içermemesidir.

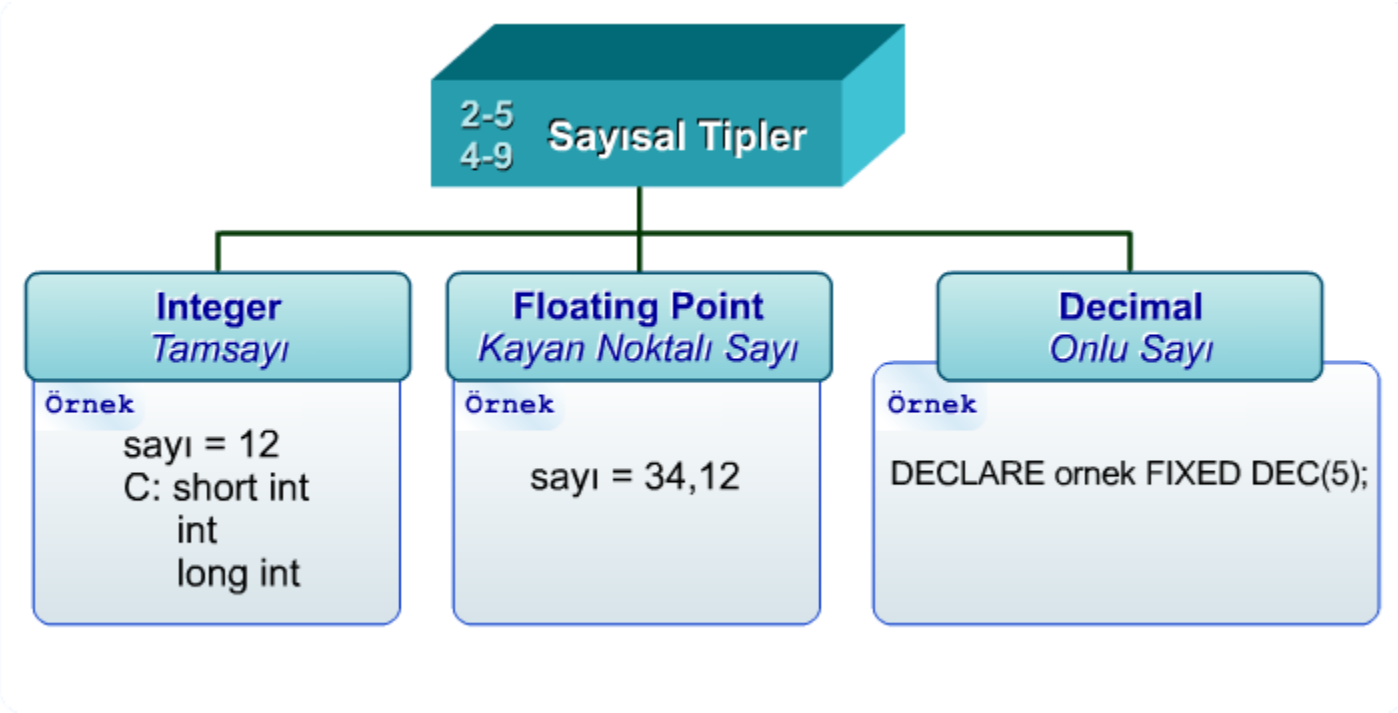
İLKEL VERİ TİPLERİ

- Başka veri tipleri aracılığıyla tanımlanmayan veri tiplerine **ilkel** (*primitive*) **veri tipleri** denir.

- Önceleri programlama dillerinde sadece sayısal ilkel veri tipleri tanımlanmışken, günümüzde popüler olan programlama dillerinde, karakter, mantıksal, karakter dizgi, kullanıcı tanımlı sıralı tipler gibi çeşitli ilkel veri tipleri bulunmaktadır.

Sayısal Tipler

- İlkel sayısal veri tipleri; tamsayı (integer), kayan noktalı (floating point) ve onlu (decimal) veri tipleridir.



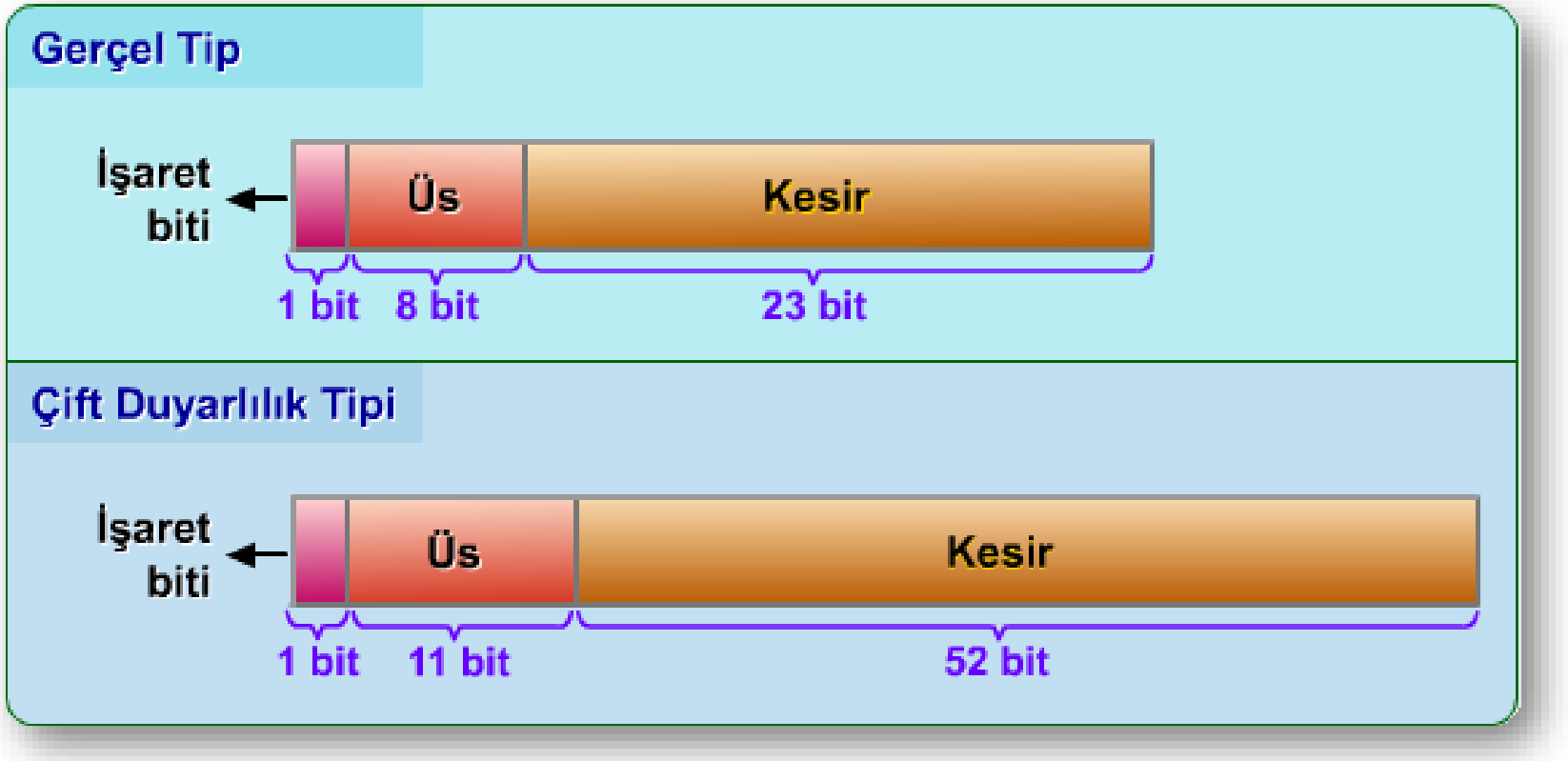
Sayısal Tipler

- **Integer (Tamsayı)**
- En bilinen ilkel veri tipi **tamsayı** (*integer*) dır.
- Bir tamsayı değer, bellekte en sol bit işaret biti olmak üzere bir dizi ikili (*bit*) ile gösterilir.
- Temel tamsayı veri tipine ek olarak Ada, C ve Java programlama dillerinde, (*short int, int, long int* gibi) üç ayrı büyüklükte tamsayı tipi tanımlanmıştır. Ayrıca C, C++, ve C#' ta işaretsiz tamsayı (*unsigned int*) veri tipi de bulunmaktadır.

Sayısal Tipler

- **Floating point (Kayan Noktalı)**
- **Kayan noktalı** (*floating point*) veri tipleri, gerçel sayıları modellerler.
- Kayan noktalı sayılar, kesirler ve üsler olarak iki bölümde ifade edilirler.
- Kayan noktalı tipler, duyarlılık (*precision*) ve alan (*range*) açısından tanımlanırlar. Duyarlılık, değerin kesir bölümünün tamlığıdır. Alan ise, kesirlerin ve üslerin birleşmesidir.
- Aşağıdaki şekilde görüldüğü gibi kayan noktalı veri tipi, **gerçel** (*real*) ve **çift-duyarlılık** (*double-precision*) olmak üzere iki tiple gösterilebilirler.

Sayısal Tipler



NaN : not a number

Sayısal Tipler

- **Decimal (Onlu)**
- **Onlu** (*decimal*) veri tipi, ondalık noktanın sabit bir yerde bulunduğu sabit sayıda onlu basamak içeren bir veri tipidir.
- Ticari uygulamalar için kullanılır (para)
  - COBOL temellidir.
  - C# dili decimal veri tipi sunar.
- Bu veri tipi, az sayıda programlama dilinde (Örneğin; PL/I, Cobol ve C#) tanımlanmıştır.
- Onlu veri tipi, onlu değerleri tam olarak saklayabilirse de, üsler bulunmadığı için gösterilebilecek değer alanı sınırlıdır. Her basamak için bir sekizli (byte) gerekli olması nedeniyle, belleği etkin olarak kullanmaz.

Sayısal Tipler

- **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.

Sayısal Tipler

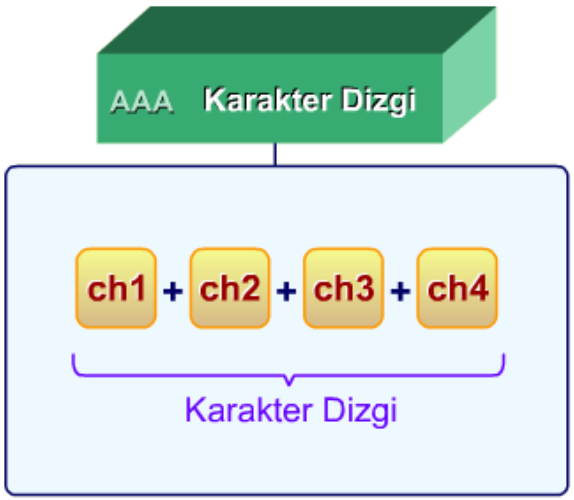
- **Boolean (İkili)**
- En basit veri tipidir.
- Değer aralığında yalnızca iki değer bulunmaktadır. Bunlar true (doğru) ve false (yanlış)'tır.
- Bitler olarak uygulanabilir, fakat çoğu zaman byte kullanılır.
  - Avantaj: Okunabilirlik

Character (Karakter)

- **Karakter** veri tipi, tek bir karakterlik bilgi saklayabilen ve bilgisayarlarda sayısal kodlamalar olarak saklanan bir veri tipidir.

  - **ASCII Kodlaması:** Karakter veri tipinde en yaygın olarak kullanılan kodlamalardan biri 8 bitlik ASCII kodlamasıdır. ASCII kodlaması, 128 farklı karakteri göstermek için, 0..127 arasındaki tamsayı değerleri kullanır.

Character String (Karakter Dizgi)

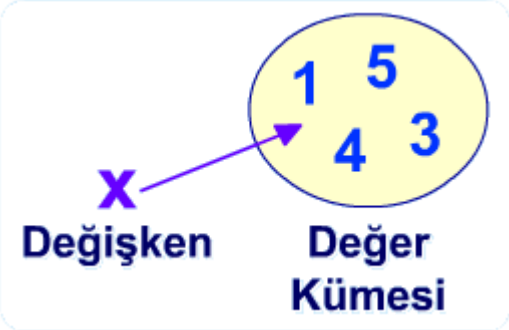


- Bir **karakter dizgi** veri tipinde, nesneler karakterler dizisi olarak bulunur.
- Karakter dizgi veri tipi bazı programlama dillerinde ilkel bir veri tipi olarak, bazılarında ise özel bir karakter dizisi olarak yer almıştır.
- FORTRAN77, FORTRAN90 ve BASIC'te karakter dizgiler ilkel bir veri tipidir ve karakter dizgilerin atanması, karşılaştırılması vb. işlemler için işlemciler sağlanmıştır.
- Pascal, C, C++ ve Ada'da ise karakter dizgi veri tipi, tek karakterlerden oluşan diziler şeklinde saklanır.

Character String (Karakter Dizgi)

- İşlemler:
  - Atama, tanımlama (char *str = “özellikler”;)
  - Karşılaştırma (=, >, strcmp, vs.)
  - Birleştirme
  - Alt dizgiye erişim (dizinin sonunu al, ilk 5 karakterini al)
  - Örüntü eşleme (A-Za-z][A-Za-z\d]+/)

Kullanıcı Tanımlı Sıralı Tipler



- Bir sıralı (ordinal) tip, olası değerlerin pozitif tamsayılar kümesi ile ilişkilendirilebildiği veri tipidir.
- Bir çok programlama dilinde kullanıcılar, **sayılama** (*enumeration*) ve **altalan** (*subrange*) olmak üzere iki tür sıralı tip tanımlayabilir.
- Bu tip tanımlarındaki amaç, programcılara modellenen gerçek dünya nesnelerine karşı gelebilecek yeni tipler oluşturma olanağı sağlamaktır.

Kullanıcı Tanımlı Sıralı Tipler

- **Enumeration (Sayılama) Tipleri**
  - **Sayılama** (*enumeration*) **tipi**, gerçek hayattaki verilerin tamsayı (*integer*) veri tipine eşleştirilmesi için kullanılan veri tipidir.
  - Bir sayılama tip tanımı, parantezler arasında yazılmış belirli sayıdaki isimden oluşur.
  - Sayılama tipi, değişkenleri tamsayılarla ilişkilendirildiği için, bu tipteki değişkenler dizi indisleri olarak, ***for*** döngü değişkenleri olarak kullanılabilir. Benzer şekilde, iki sayılama tipinde değişken veya sabit ilişkisel işlemcilerle, tanımlamadaki sıralarına göre karşılaştırılabilir.

Kullanıcı Tanımlı Sıralı Tipler

- **Enumeration (Sayılama) Tipleri**
  - Pascal'daki type deyiminin benzeri olarak düşünülebilen C'deki enum yapısı ile yeni bir tip yaratılmaz. Bu yapının kullanımındaki amaç, bir dizi define kullanımı yerine daha anlaşılır bir yaklaşım sunmaktır.
  - İsimlendirilmiş değerlerin hatırlanması kolay olduğu için, sayılama tipi kullanımı, programların okunabilirliğini ve güvenilirliğini artırır.

Örnek

Aşağıdaki C deyiminde, sıcaklığın üç değeri, sıralarıyla birlikte tanımlanmıştır.

```
typedef enum {dusuk, orta, yuksek} sıcaklık;
```

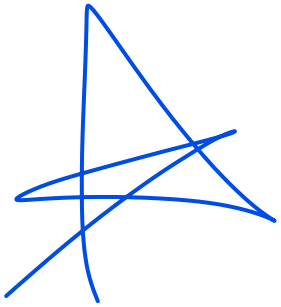
Kullanıcı Tanımlı Sıralı Tipler

- **Subrange (Altalan) Tipleri**
  - Bir **altalan** (*subrange*) **tipi**, bir sıralı tipin bir alt grubudur
  - Bir altalan tipinin tanımındaki değerler, daha önceden tanımlanmış veya dilde tanımlı olan (*built-in*) sıralı tiplerle ilişkilendirir. Böylece, yeni tanımlanan tip ile alt grubu olduğu ana sınıf arasında bağ kurulur. Altalan tipinin ana sınıfına uygulanabilen tüm işlemciler, altalan tiplerine de uygulanabilmektedir.

15..20, tamsayı tipinin bir alt grubudur.

Kullanıcı Tanımlı Sıralı Tipler

- **Subrange (Altalan) Tipleri**
  - Altalan tipleri ilk olarak Pascal'da tanıtılmıştır.
  - Altalan tipleri değer alanını sınırladıkları için, bir altalan tipindeki bir değişkenin hangi değerleri alabileceği belirlidir. Bu durum, hem programlamların okunabilirliğini hem de güvenilirliğini artırır.
    - `Type Day is (mon, tue, wed, thu, fri, sat, sun);`
    - `subtype Weekdays is Days range Mon..Fri;`
    - `subtype Index is Integer range 1..100;`
    - `Day1 : Days;`
    - `Day2 : Weekdays;`
    - `. . .`
    - `Day2 := Day1;`



Örnek Aşağıda iki altalan tipinin Pascal'da tanımı örneklenmiştir:

```
Type
   Buyukharfler ='A'..'Z';   → Buyukharfler, tek karakterler için dilde tanımlı olan
                               char tipinin altalanı olarak,
   Puanlar =50..100;         → Puanlar ise integer tipinin altalanı olarak tanımlanmıştır.
```

  - **Ada** – Altalan tipleri yeni tipler değildir. Sınırlandırılmış var olan tiplerdir. Pascal'daki gibi "case" içinde de kullanılabilir.
  - subtype POS_TYPE is INTEGER range 0..INTEGER'LAST;
  - subtype Index is Integer range 1..100;

Kullanıcı Tanımlı Sıralı Tipler

- **Subrange (Altalan) Tipleri Değerlendirmesi**
- Okunabilirliğe yardım
  - Okuyucuların kolayca görebileceği altalan değişkenlerini yalnızca belirli aralıkta saklayabiliriz.
- Güvenilirlik
  - Belirlenen değerler dışında altalan değişkene farklı değerler atamak hata olarak algılanır.

YAPISAL VERİ TİPLERİ

- **Yapısal** (*structured*) tipler ilkel tiplerden oluşur ve bellekte bir dizi yerleşimde saklanırlar. Diziler, kayıtlar ve göstergeler yapısal veri tiplerini oluşturmaktadır.

Diziler (Arrays)

- **Diziler (Arrays)**
- Bir dizi, homojen veri elemanlarının bir kümesidir, elemanlardan her biri kümedeki birinci elemana göre olan pozisyonuyla tanımlanır.

Dizi İndisleri

- İndisler, indis elemanları ile dizi elemanlarını birbirlerine eşler.
- **array_name (index_value_list) → an element**
- **İndis sözdizimi (syntax):**
  - FORTRAN, PL/I, Ada ( ) parantez kullanır
  - Birçok diğer diller [ ] kullanır
  - Genelde dizi indisi integer
  - Indeks aralık kontrolü (C, C++ aralık kontrolü yapmaz)

Dizi İndisleri

- **İndisin Üst Sınırı:**

- İlk programlama dillerinin aksine, günümüzde popüler olan programlama dillerinde dizi indislerinin sayısı sınırlanmaz. Böylece çok boyutlu diziler oluşturulabilir. Ancak, genellikle programlama açısından 3'ten fazla boyutu olan dizilere gereksinim duyulmaz.

- C'de ise diğer programlama dillerinden farklı bir tasarım vardır. C'de dizilerin tek indisi olabilir. Ancak, dizilerin elemanları diziler olabildiği için çok boyutlu diziler oluşturulabilir. Her boyut için ayrı bir köşeli parantez gerektirmesi dışında, C'deki dizilerin diğer dillerdeki dizilerden bir farkı yoktur.

Dizi İndisleri

C'de 4,5 büyüklüğünde bir tablo aşağıdaki şekilde tanımlanmaktadır.

int tablo [4][5];

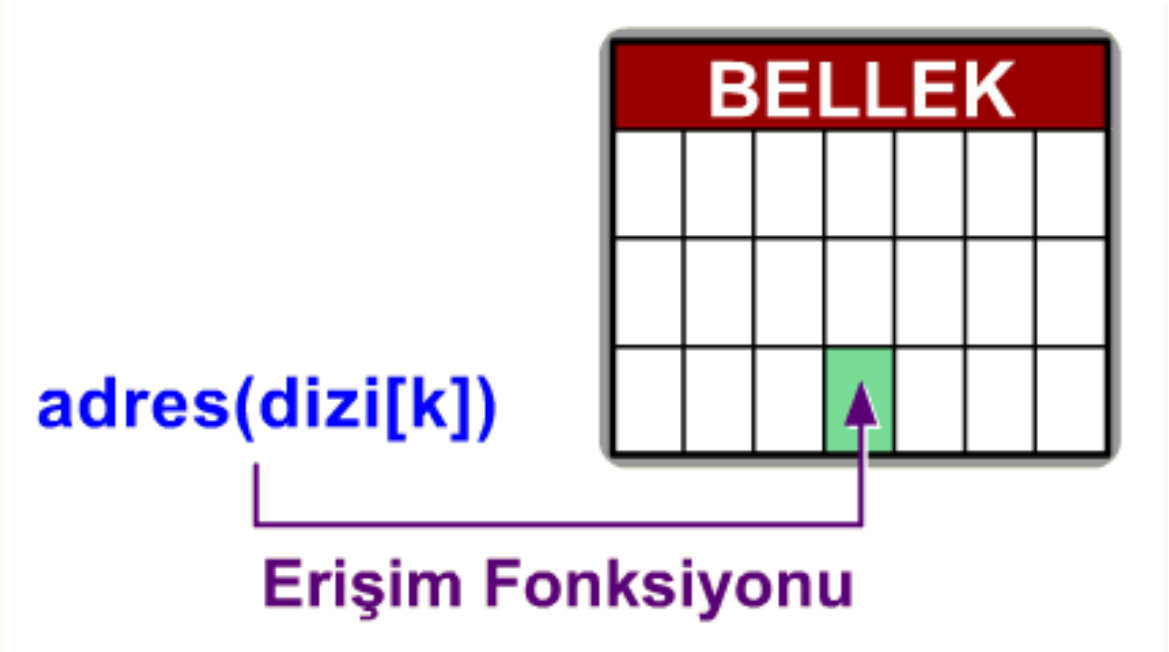
	1	2	3	4	5
1	1,1	1,2	1,3	1,4	1,5
2	2,1	2,2	2,3	2,4	2,5
3	3,1	3,2	3,3	3,4	3,5
4	4,1	4,2	4,3	4,4	4,5

Dizi İndisleri

- **İndisin Alt Sınırı:**
- İndisin alt sınırı, bazı dillerde varsayılan değerler alır. Örneğin, C ve C++'da tüm indisler için alt sınır varsayılan olarak sıfır (0), FORTRAN 77 ve FORTRAN 90'da ise varsayılan olarak bir (1) kabul edilir.
- Birçok programlama dilinde ise, indislerin alt sınır değerleri varsayılan olarak tanımlı değildir ve dizinin tanımında belirtilmelidir.

Dizi Tiplerinin Gerçekleştirimi

- Bir dizi için **erişim fonksiyonu**, dizinin taban adresini ve indis değerlerini, indis değerleriyle belirtilen bellek adreslerine eşleştirmektir.



Dizi Tiplerinin Gerçekleştirimi

- Bir dizi elemanının adresi iki bölümde hesaplanabilir.
- İlk bölüm, dizi tanımlanır tanımlanmaz hesaplanması mümkün olan bölüm, ikinci bölüm ise dizi indisinin değerine bağlı olduğu için çalışma zamanında hesaplanması gereken bölümdür.

Örnek

Tek boyutlu bir dizi için erişim fonksiyonu, alt indis sınırı 1 kabul edilerek, aşağıda belirtilmiştir:

```
adres(dizi[k]) = adres(dizi[1]) + (k-1) * eleman_uz
```

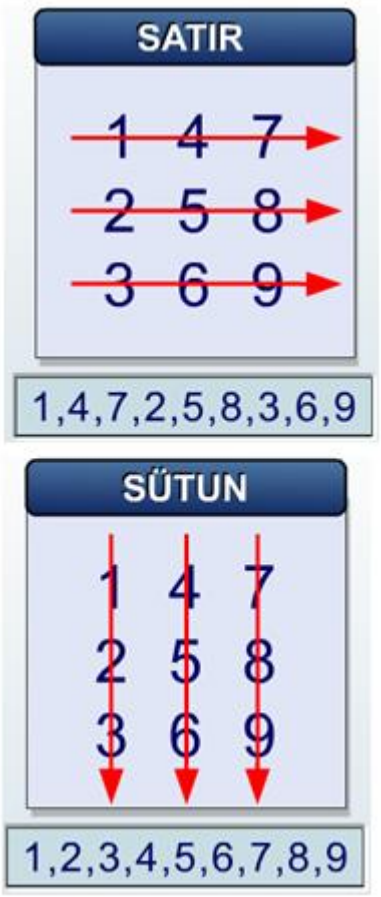
```
adres(dizi[k]) = adres(dizi[1]) + k * eleman_uz - eleman_uz
```

```
adres(dizi[k])=(adres(dizi[1])-eleman_uz)+(k* eleman_uz)
```

sabit **değişken**

Sadeleştirme sonucu elde edilen formülde ilk bölüm sabit olurken, ikinci bölüm değişken olmaktadır. Eğer eleman tipi ve dizi için bellek bağlaması durağan olarak gerçekleşiyorsa, sabit bölüm, çalışma zamanından önce hesaplanabilir.

Dizilerin Belleğe Eşleştirilmesi



- Bellek donanımı, doğrusal olan bir dizi sekizliden (byte) oluştuğu için, iki veya daha çok boyutlu diziler, tek boyutlu belleğe eşleştirilmelidir.
- Bu durum, çok boyutlu dizilerin gerçekleştirimini tek boyutlu dizilere göre daha karmaşıklaştırır. Çok boyutlu dizilerin tek boyuta eşleştirilmesi *satır tabanlı sıra* ve *sütun tabanlı sıra* olmak üzere iki şekilde yapılabilir.
- Satır tabanlı sırada, eğer dizi bir matris ise, satırlara göre saklanır.

İndis bağlama ve dizi kategorileri-

- İndis bağlanmaları ve bellekteki bağlanmalarına göre dizi kategorileri

- **1. Statik (Durağan):** İndis sınırları ve dizinin bellekteki bağlanmaları çalışma zamanında önce hesaplanır, statiktir.
  - FORTRAN 77, Ada'da bazı diziler. C'de statik diziler.
  - Avantaj: Yürütme verimi (bellekten yer alma, geri verme yok)

- **2. Sabit Yığın Dinamik (Fixed Stack Dynamic):** İndis sınırlarının bağlanmaları statik fakat dizinin belleğe bağlanması çalışma zamanında gerçekleşir (yer ayırma işlemi tanımlama zamanında yapılır).
  - Örneğin "statik" olmayan çoğu Java, C yerel değişkenleri
  - Avantaj: bellek verimi

İndis bağlama ve dizi kategorileri-

- **3. Sabit Yığın Dinamik (Fixed Heap-dynamic):**
- İndis ve bellek bağlanmaları dinamik (yer tahsisi heapten) olur, fakat bir kez belirlendikten sonra sabit. Bellek geri verilebilir.
  - FORTRAN 90
    - INTEGER, ALLOCATABLE, ARRAY (:,:) :: MAT

      (MAT'ı dinamik 2-boyutlu dizilim olarak tanımlar)
    - ALLOCATE (MAT (10,NUMBER_OF_COLS))

      (MAT'a 10 satır ve NUMBER_OF_COLS sütun ayırır)
    - DEALLOCATE MAT (MAT için ayrılmış belleği iade eder)
  - C, C++: malloc, free ...
  - C++: new, delete.
  - Avantaj: Esneklik – Dizi kullanılmaya başlanmadan önce boyutu bilinmek zorunda değildir

İndis bağlama ve dizi kategorileri

- **4. Yığın Dinamik (Heap-dynamic):** İndis aralığı ve yer ayırma (bellek bağlama) dinamiktir ve istenilen zamanda değiştirilebilir.
  - Avantaj: esneklik (diziler program çalışması sırasında büyüyebilir veya küçülebilir)
  - APL'de, Perl ve JavaScript, diziler (Arrays) ihtiyaca göre büyüyüp küçülebilir.
    - Perl örneği: @list = (1 , 3, 7, 10);
    - push(@list, 13 , 17); // -> (1 , 3, 7, 10 13 , 17)
    - @list = (); // belleği boşaltır ve iade eder.

Dizi Başlatma

- Dizi başlangıcı, genellikle dizi elemanlarının bellekte saklandığı sıra ile sıralanmış olarak konulan değerlerin listesidir.
- Bazı diller dizi başlangıcına izin verir.
  - C, C++, Java, C#
    - int list [] = {4, 5, 7, 83}
  - C ve C++ da karakter stringleri
    - char name [] = “freddie”;
  - C ve C++da pointerlar ile string dizileri
    - char *names [] = {“Bob”, “Jake”, “Joe”];
  - Java String nesneleri
    - String[] names = {“Bob”, “Jake”, “Joe”};
  - Python
    - List=[x ** 2 for x in range (12) if x%3==0]

Heterojen Dizi

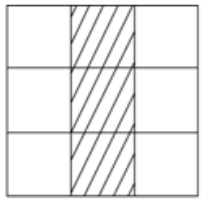
- Heterojen dizi, elemanları aynı tipten olması gerekmeyen dizilerdir.
- Perl, Python, JavaScript ve Ruby tarafından desteklenir.
- Diğer dillerde heterojen diziler yerine struct (yapılar) kullanılır, fakat yapılar heterojen diziyi tam manasıyla karşılayamazlar.

Dizi İşlemleri

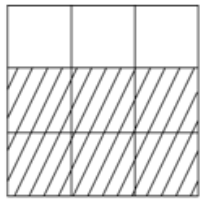
- APL vektörler ve matrisler için hem en güçlü dizi işleme işlemleri hem de tekli operatör desteği (örneğin, kolon elemanları tersine çevirmek için) sağlar
- Ada yalnızca dizilerde atama, birleştirme ve ilişkisel operatör işlemlerine izin verir.
- Pyton'nun dizi atamaları yalnızca referans değişikliği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

Kesitler

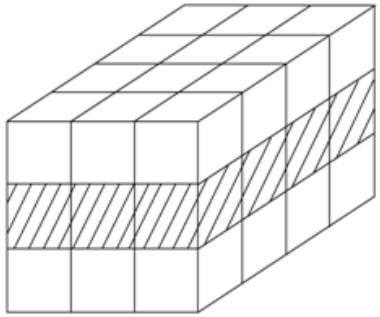
- Kesitler (slices)
  - Kesit (slice), bir dizininin bir kısım altyapısıdır (substructure) ; bir referanslama mekanizmasından fazla bir şey değildir
  - Kesitler (slices) sadece dizi işlemleri olan diller için kullanışlıdır.



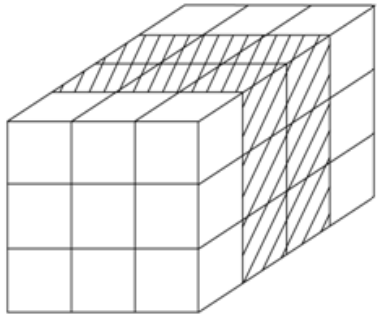
MAT (1:3, 2)



MAT (2:3, 1:3)



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



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

MAT = CUBE(:,:,2) // cube dizininin ikinci dilimini mat'a koyar.

6.3.1.9.1. 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.

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 başlayarak referanslandırır.

+ operatörünü kullanır ve del komutuyla silinir.

Demet (Tuple) Tipi

- 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)` ve `(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. Buradaki karmaşıklığı ortadan kaldırmak için veri listelerinin önüne ‘ işareti konur.

  `'(A B C)` veridir

Liste Tipleri

- 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

- 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 burada hd ve tl olarak adlandırılır.

Liste Tipleri

- 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 olabilir
  - Liste oluşturulması aşağıdaki gibidir.

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

Liste Tipleri

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

    `x = myList[1]` `x’` e `5.8` atar
  - 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

- 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.

Record (Kayıt) Tipi

- Bir sınıftaki öğrencilerin hem numara, hem ad-soyad hem de adreslerinin tutulması için 3 ayrı diziye gereksinim vardır.
- Kayıt tipi, programlardaki bu tür gereksinimleri karşılamak için tasarlanmıştır.
- **Kayıt** (*record*) **tipi**, her elemanın ismiyle ayırt edildiği heterojen veri elemanları topluluğudur. İlk olarak COBOL'da tanıtılan kayıt tipi, daha sonra çoğu programlama dilinde yer almıştır.

Record (Kayıt) Tipi

- Diziler ve kayıtlar arasındaki en temel fark, dizilerde homojen elemanların, kayıtlarda ise heterojen elemanların bulunmasıdır.
- Bunun sonucu olarak, kayıtlardaki sahalara indislerle değil, her sahayı gösteren tanımlayıcılarla ulaşılmaktadır. (örneğin; ogrenci.adsoyad gibi).
- Kayıt sahaları, dizilerde olduğu gibi homojen olmak zorunda olmadıkları için, saha tipleri seçiminde kayıtlar, dizilerden daha fazla esneklik sunarlar.
- Dizi elemanlarına erişim kayıt alanlarına erişimden daha yavaştır, çünkü altsimgeler (subscripts) dinamiktir (alan adları (field names) statiktir).

- Kayıtlardaki sahalara başvuru, çeşitli şekillerde gösterilebilir. Ortak nokta, erişilmek istenilen sahanın ve bunu içeren kaydın belirtilmesidir.
- COBOL düzey numaraları kullanır, diğerleri özyinelemeli olarak tanımlarlar.
- Aşağıdaki COBOL için verilen örnekte, *OGRENCI_KAYDI, OGRENCI_ADI* kaydından ve *OGRENCI_NO* ve *OGRENCI_ADRES* sahalarından oluşmaktadır. *01,02* ve *05* düzey numaraları olup kayıttaki hiyerarşik yapıyı göstermektedir.

```
01 OGRENCI_KAYDI.
   02 OGRENCI_ADI.
      05 ILK PICTURE IS X(15).
      05 SOYAD PICTURE IS X(15).
   02 OGRENCI_NO PICTURE IS 99999.
   02 OGRENCI_ADRES PICTURE IS X(30).
```

Union (Bileşim) Tipi

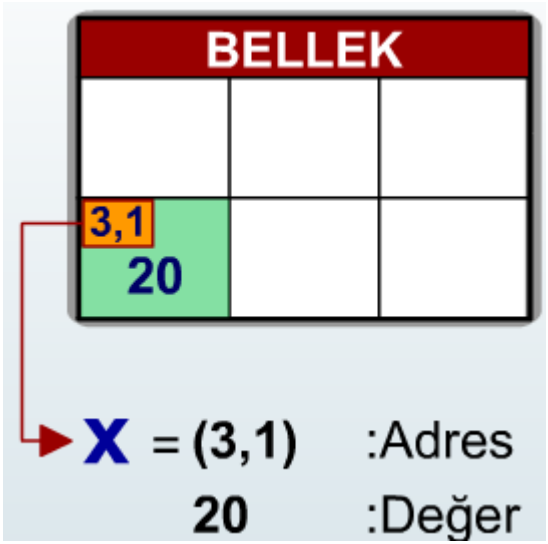
- Bir programın çalışması boyunca farklı değerler saklayabilen **bileşim** (*union*) tipi, kayıt tipinin özel bir şekli olarak görülebilir.
- Bir programda veya fonksiyonda değişkenlerin aynı bellek alanını paylaşması için ortaklık bildirimi bileşim (union) deyimi ile yapılır. Bu da belleğin daha verimli kullanılmasına imkan verir.
- Bileşim tipinin tasarımı ve bileşim tipinde tip denetimi, programlama dillerinde çeşitli biçimlerde işlenmiştir.

Union (Bileşim) Tipi

- C'de bileşim tipi, union ile yapılır. Örneğin:
  - union paylas{ double f;int i;char kr; };
- Yukarıdaki bildirim yapıldığında, değişkenler için bellekte bir yer ayrılmaz. Değişken bildirimi:
  - union paylas bir, iki;

  şeklinde yapılır.
- Üyelere erişmek aşağıdaki gibi olur:
  - bir.kr= 'A'; iki.f = 3.14; bir.i = 2000;

Pointer (Gösterge) Tipi

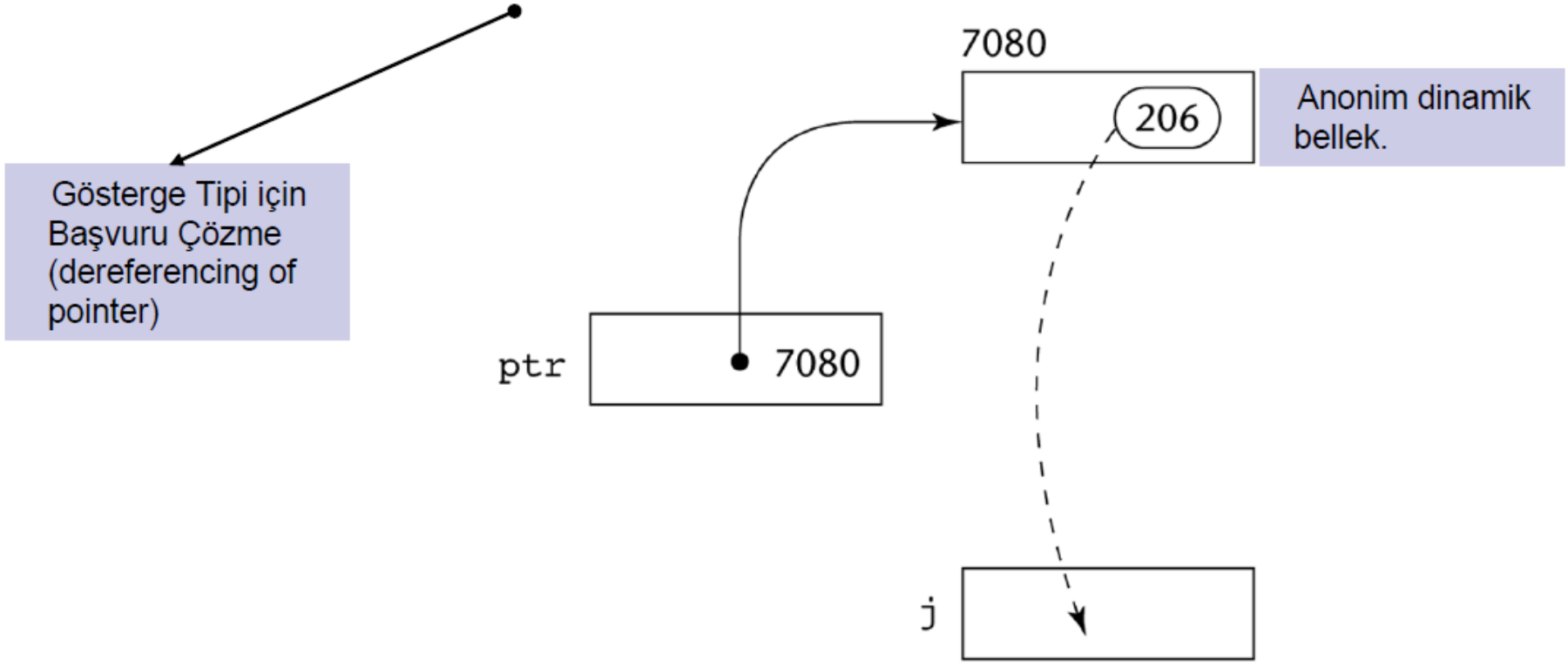


- **Gösterge** (*pointer*) tipi, belirli bir veriyi içermek yerine başka bir veriye başvuru amacıyla kullanılır. Bu nedenle, hem ilkel tiplerden hem de yapısal tiplerden farklı bir veri tipidir. Bir gösterge tipi sadece bellek adreslerinden oluşan değerler ve boş (*null*) değerini içerebilen bir tiptir.

- Gösterge tipindeki değerler, gösterdikleri veriden bağımsız olarak sabit bir büyüklüktedirler ve genellikle tek bir bellek yerine sığarlar.

Pointer (Gösterge) Tipi

Örnek: atama işlemi: `int *ptr; j = *ptr;`



Dangling Pointer (Sallanan Gösterge)

- Bir gösterge değişkenin gösterdiği adreste geçerli veri olmaması durumu, göstergenin serbest bırakılmış bir dinamik yığın değişkene işaret etmesi ile oluşur. Serbest bırakılmış bir bellek adresini gösteren değişkene **sallanan gösterge** (*dangling pointer)* denir. Bir göstericinin gösterdiği belleğin bir şekilde sisteme iade edildiği durumdur.

- **Pascal, C, C++ ve Java'da Sallanan Gösterge:**
- Pascal'da, yığın değiştirilebilir değişkenler ***new*** ile oluşturulur ve ***dispose*** ile yok edilirler. Pascal'da gösterge değişkenler, sadece değiştirilebilir yığın değişkenlere erişmek için kullanılırlar. Ancak Pascal'da *dispose* deyimi bir değişkeni gösteren tüm gösterge değişkenleri düzenlenmediği için sallanan göstergeler oluşabilir.

Dangling Pointer (Sallanan Gösterge)


```
int *a, *b;
...
a =malloc(sizeof (int));
b =a;
free(a);
```

- Benzer şekilde C ve C++'da da sallanan gösterge sorunu vardır. C'de sallanan gösterge oluşumu yandaki şekilde görülmektedir.
- Başka bir örnek verelim:
  - **int* f () { int fv = 42; return &fv; }**
- Aşağıdaki kod gösterici dönen f'i çağırır, sonucu bir gösterici değişkene koyar, sonra bunu değiştirmeye çalışır.
  - **int* p = f(); *p = 0;**
- Fakat göstericinin gösterdiği fv f'nin içinde tanımlıdır, f'nin yaşamı bitince onun bellekte kullandığı yer de iade edilmiştir, bu şekilde kullanılması beklenmedik sonuçlar doğurur.
- Java'da ise gösterge veri tipine dilde yer verilmeyerek, güvenilirlik problemlerinin önlenmesi amaçlanmıştır.

Bellek Sızıntısı

Kayıp yığın dinamik (Heap-Dynamic) değişkenler.

- Yığın dinamik değişken herhangi bir program göstericisi tarafından gösterilmemektedir.
- Örnek:
  a) Gösterici p1 önce bir yığın değişkeni gösterir:
  int *p1;
  ```
  p1 =(int *)malloc(sizeof(int));
  ```
  b) Daha sonra, yeni yaratılmış başka birini:
  ```
  p1 =(int *)malloc(sizeof(int));
  ```
- Bu şekildeki kayıplara bellek kaçağı veya sızıntısı denir.

C ve C++

- Dinamik bellek ve adres yönetimi için kullanılır.
- Açık başvuru çözme, adres kaldırma işlemleri.
- Alan tipinin sabitleştirilmesi şart değil (**`void *`**).
- **`void *`** - herşeyi gösterebilir ve tip kontrolü yapılabilir (başvuru çözme işlemi yapılamaz)
- Sınırlı olarak adres aritmetiği yapılır, örneğin:

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

`*(p+5)` ➔ `stuff[5]` ve `p[5]`

`*(p+i)` ➔ `stuff[i]` ve `p[i]`

(örtülü ölçekleme)