

# Hedef

- Veri tipleri Temel Veri Tipleri ve Kullanıcı tanımlı Bileşik Veri Tipleri olarak sınıflandırılacaktır.
- İlkel Veri Tipleri'nde sayısal, mantıksal, karakter tipler, karakter dizgi tipleri;
- Bileşik Veri Tipleri'nde ise diziler, record tipi, union tipi, set (küme) tipi, işaretçi (gösterge) tipi anlatılacaktır.
- Ayrıca, Kuvvetli Tipleme, Tip Uyumluluğu, Tip Denetimi ve Tip Dönüşümleri de bu bölümde incelenecektir.

# GİRİŞ

- Bir verinin bellekte nasıl tutulacağını, değerinin nasıl yorumlanacağını ve veri üzerinde hangi işlemlerin yapılabileceğini belirleyen bilgiye **veri tipi** denir.
- Bir programlama dilindeki veri tipleri, programlardaki ifade yeteneğini ve programların güvenilirliğini doğrudan etkiledikleri için, bir programlama dilinin değerlendirilmesinde önemli bir yer tutarlar.

# Temel Veri Tipleri-Bileşik veri tipleri

- Temel ve Bileşik veri tipleri arasındaki en önemli fark, **temel veri tiplerinin başka veri tiplerinden oluşmamasıdır.**
- Bileşik veri tipleri ise **kullanıcı tanımlı olup başka veri tiplerini de içermektedir.**

# Temel Veri Tipleri

- Başka veri tipleri aracılığıyla tanımlanmayan veri tiplerine **ilkel** (*primitive*) **veri tipleri** denir.
- Sayılar
- Karakter,
- Mantıksal,
- Karakter dizgi,
- Kullanıcı tanımlı sıralı tipler

# Sayısal Tipler

- İlkel sayısal veri tipleri; tamsayı, kayan noktalı ve onlu veri tipleridir.

TÜR İSMİ	UZUNLUK(byte) (DOS / UNIX)		SINIR DEĞERLERİ	
signed char	1		-128	127
unsigned char	1		0	255
signed short int	2		-32.768	32.767
unsigned short int	2		0	65.535
signed int	2	4	-32.768 - 2.147.483.648	32.767 2.147.483.647
unsigned int	2	4	0 0	65.535 4.294.967.296
long int	4		- 2.147.483.648	2.147.483.647
unsigned long int	4		0	4.294.967.296

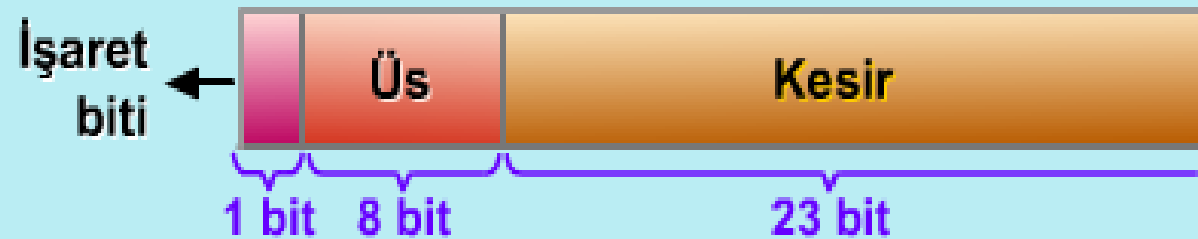
# Integer (Tamsayı)

- En bilinen temel 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 ve C programlama dillerinde, (*short int*, *int*, *long int* gibi) üç ayrı büyüklükte tamsayı tipi tanımlanmıştır. Ayrıca C'de işaretsiz tamsayı (*unsigned int*) veri tipi de bulunmaktadır.

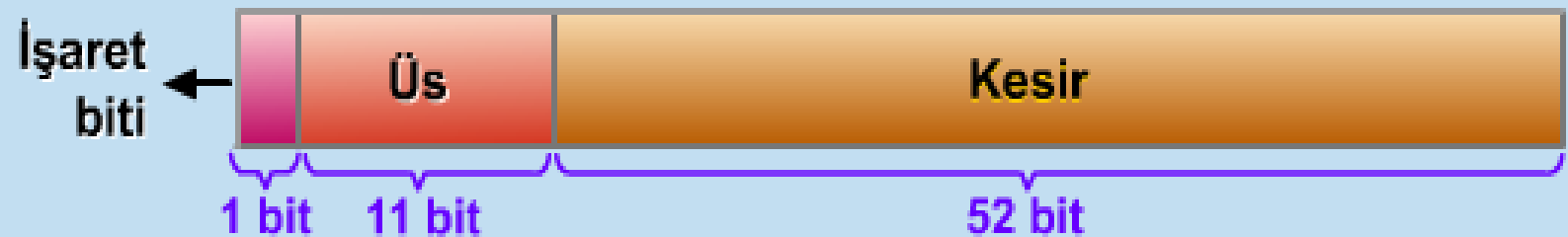
# Floating point (Kayan Noktalı)

- **Kayan noktalı** (*floating point*) veri tipleri, **gerçek 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.

### Gerçel Tip



### Çift Duyarlılık Tipi





# Decimal (Onlu)

- **Onlu** (*decimal*) veri tipi, ondalık noktanın sabit bir yerde bulunduğu sabit sayıda onlu basamak içeren bir veri tipidir. Bu veri tipi, az sayıda programlama dilinde (Örneğin; PL/I) 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.

# Boolean (Mantıksal)

- **Mantıksal**(*boolean*) veri tipi, ilk olarak ALGOL 60 tarafından tanıtılmış ve daha sonra çoğu programlama dilinde yer almıştır.
- Mantıksal veri tipi, sadece *doğru* (**true**) veya *yanlış* (**false**) şeklinde ifade edilen iki değer alabilir.
- Bir mantıksal değer bellekte bir ikili ile gösterilebilirse de, çoğu bilgisayarda bellekteki tek bir ikiliye etkin olarak erişim güç olduğu için, bir sekizlide saklanırlar.

# Mantıksal (devam)

- İlişkisel işlemciler, ve seçimli deyimler gibi programlamadaki birçok yapı, mantıksal tipte bir ifade üzerinde çalıştığı için mantıksal veri tipinin dilde yer almasının önemi büyüktür.
- ALGOL 60'dan sonraki çoğu dilde yer alan mantıksal veri tipinin yer almadığı bir programlama dili C dilidir. C'de ilişkisel işlemciler, ifadenin sonucu doğru ise 1, değilse 0 değeri döndürürler. C'de *if* deyimi, sıfır değeri için yanlış bölümünü, diğer durumlarda ise doğru bölümünü işler.

# Character (Karakter)

- **Karakter**(character) veri tipi, tek bir karakterlik bilgi saklayabilen ve bilgisayarlarda sayısal kodlamalar olarak saklanan bir veri tipidir.
- Karakter veri tipinde en yaygın olarak kullanılan kodlamalardan biri ASCII kodlamasıdır. ASCII kodlaması, 128 farklı karakteri göstermek için, 0..127 arasındaki tamsayı değerleri kullanır.
- ASCII kodlamasıyla bağlantılı olarak bazı programlama dilleri, karakter veri tipindeki değerlerle tamsayı tipi arasında ilişki kurarlar.
- C'de, *char* veri tipi, eş olarak kullanılabilir.

- C'de *char* ve *int* veri tipleri dönüşümlü olarak kullanılabilir.

Örnek

```
char ab;  
int sayı;  
ab = 'C' + 3;  
sayı = 'H'  
ab = sayı;
```

# Character String (Karakter Dizgi)

- Bir **karakter dizgi** (*character string*) 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.

# Karakter Dizgilerin Uzunlukları

Durağan Uzunluk	$ch1 + ch2 + ch3 + ch4$
	77, FORTRAN 90, COBOL, Pascal, Ada
Değişen Uzunluk (üst sınır var)	$ch1 + ch2 + ch3 + ..... + chn$
	PL/I, C, C++
Değişen Uzunluk (üst sınır yok)	$ch1 + ch2 + ch3 + ch4 + ..... + chn$
	Dinamik bellek yönetimi olan diller

# Kullanıcı Tanımlı Sıralı Tipler

- Bir **sıralı** (*ordinal*) **tip**, olası değerlerin pozitif tamsayılar kümesi ile ilişkilendirilebildiği veri tipidir.
- **Sayılama** (*enumeration*)
- **Altalan** (*subrange*) olmak üzere iki tür sıralı tip tanımlayabilir.

Gerçek dünya nesnelerini daha iyi modelleyebilen yeni tipler oluşturma olanağı sağlamaktır.



# Enumeration (Sayılama) Tipleri

- **Sayılama** (*enumeration*) **tipi**, gerçek hayattaki verilerin tamsayı (*integer*) veri tipine eşleştirilmesi için kullanılan veri tipidir.

# Sayılama (devam)

- Bir sayılama tip tanımı, parantezler arasında yazılmış belirli sayıdaki isimden oluşur.

**type** gunler **is** (Pazartesi, Salı, Çarsamba, Persembe, Cuma, Cumartesi, Pazar)

Bu tanımlamaya göre “Pazartesi” bilgisi 1 sayısı ile ve “Pazar” bilgisi 7 sayısı ile eşleştirilmiş olur. **ADA**

```
type renkbilgisi=(kırmızı, mavi, yeşil, sarı);  
var renk:renkbilgisi;  
...  
    renk:=mavi;  
if renk > red; .....
```

Typedef enum {soguk, ılık, sıcak} ısı;

**C**

**PASCAL**

Java dilinde sayılama veri tipi bulunmamaktadır.

Güvenilirlik, okunabilirlik

# Subrange (Altalan) Tipleri

- Bir sıralı tipin bir alt grubudur. Ana sınıfa uygulanabilen tüm işlemciler, altalan tiplerine de uygulanabilmektedir.

## 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.

Hem programlama dillerinin okunabilirliğine hem de güvenilirliğine olumlu katkı

# Diziler

- Bir **dizi** (*array*) homojen elemanlardan oluşur ve dizi elemanlarının yeri ilk elemana göre belirlenebilir.
- Programlarda dizilere olan başvurular, **dizi ismi ve indis değeri** şeklinde gerçekleştirilir.
- programlama dillerinde çoğunlukla (Pascal, C ve C++) İndislerin gösterimi için köşeli parentezler kullanılır.

```
double sonuc[15];
```

- ```
float x[100], y[100];
```

# Dizi İndisleri

- **İndisin Üst Sınırı:**
- İlk programlama dillerinin aksine, günümüzde popüler olan programlama dillerinde dizi indislerinin sayısı sınırlanmaz.
- 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. (`int mat[2][3];`)

# İndisin Alt Sınırı:

- C ve C++'da tüm indisler için alt sınır varsayılan olarak sıfır, FORTRAN 77 ve FORTRAN 90'da ise varsayılan olarak bir kabul edilir. Çoğu programlama dilinde ise, indislerin alt sınır değerleri dizinin tanımında belirtilmelidir.

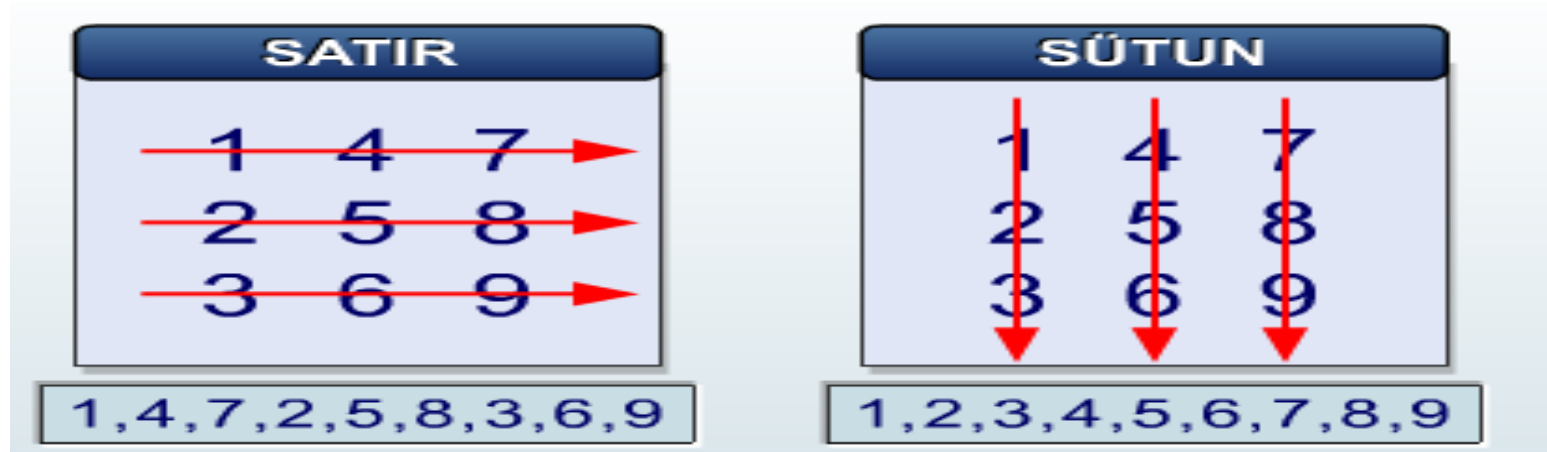
# Dizi Tiplerinin Gerçekleştirimi

- Alt indis sınırı 1 ve her bir elemanın sözcük uzunluğu c olan bir dizinin A[k]'nci elemanın adresi:

$$\text{Adres}(A[k]) = \text{adres}(A[1]) + (k-1) * c$$

$$\text{Adres}(A[k]) = \{\text{adres}(A[1]) - c\} + k * c$$

# Dizilerin Belleğe Eşleştirilmesi



İki boyutlu ( $i$  sıra ve  $j$  sütunu olan) ve her satırında  $n$  eleman bulunan  $A[i,j]$  isimli bir dizi için erişim fonksiyonu aşağıdaki gibi olmaktadır.

$$\text{Adres}([a(i,j)]) = \text{adres}[1,1] + ((i-1) * n) + (j-1) * c$$

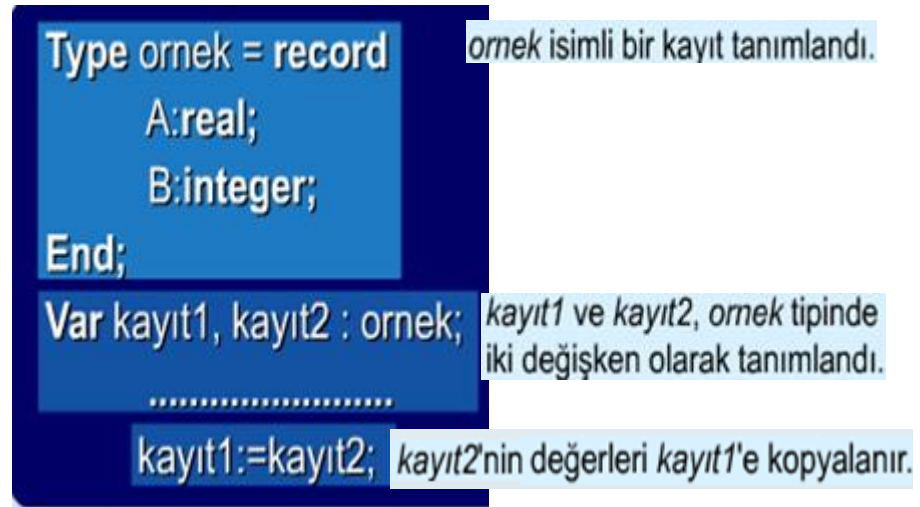


# Record (Kayıt) Tipi

- İlk olarak COBOL'da tanıtılmıştır.
- **Altalan** olarak isimlendirilen birden fazla ifadenin bulunduğu yapıdır. Kayıt veri tipi ile bir isim altında farklı tipte birden fazla alan tanımlanabilir.
- Yani dizilerde homojen elemanlar bulunurken kayıtlarda heterojen elemanlar vardır.
- Dizideki bir elemana indis numarası ile ulaşılırken kayıt veri tipinde alanlara her sahayı gösteren tanımlayıcılarla . (mus.ad) ulaşılmaktadır.
- Kayıt içerisinde tanımlanan her alanın birbirinden farklı isimleri vardır. İstenirse bu isimler bu tipin dışında farklı tanımlamalar için de kullanılabilir.

Pascal dilinde kayıt veri tipi aşağıdaki söz dizim ile tanımlanır.

- **KayıtTipAdı=RECORD**
- **AlanAdı1:TipAdı;**
- **AlanAdı2;TipAdı;**
- ...
- **End;**



```
personel=Record
Ad, Soyad:String[15];
Adres: String[15];
Telefon: String[12];
Maas: LongInt;
End;
```

Bu tipin alt alanlarına **personel.Ad** yazımı ile ulaşılabilir.

Burada Record tipine sahip değişkenlerin program içerisinde kullanımında değişken ismi ile altalan ismi arasına “.” Konulması zorunluluğu vardır. Her alt alan isiminden önce de değişken isminin yazılması programın uzamasına ve zaman kaybına sebep olmaktadır. Pascal bu olumsuz durumu **with** deyimi ile çözmektedir.

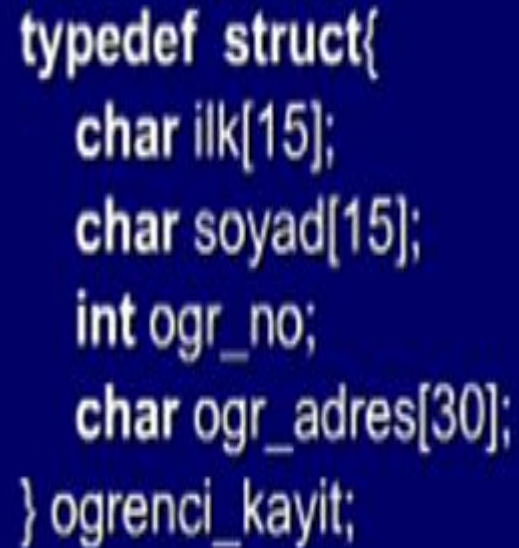
## PASCAL

```
Begin
With Index Do
Ad:='Mustafa'
Soyad:='Kara'
End;

with kayıt Do
Begin
İsim :=Index;
Meslek:='Doktor';
Telefon:='123456';
Adres:='Kayseri';
...
...

End.
```

## C



```
typedef struct{
    char ilk[15];
    char soyad[15];
    int ogr_no;
    char ogr_adres[30];
} ogrenci_kayit;
```

# Union (Bileşim) Tipi

Aynı bellek bölgesinin farklı değişkenler tarafından kullanılmasını sağlayan veri tipidir.

Bu veri tipindeki değişkenlere ortak değişkenler de denir.

Buradaki temel amaç farklı zamanlarda kullanılacak birden fazla değişken için ayrı ayrı yer ayırma zorunluluğunun ortadan kaldırılarak *belleğin iyi kullanılmasıdır.*

| İsim    |         |
|---------|---------|
| ad      | soyad   |
| 15 byte | 15 byte |

struct

| İsim    |
|---------|
| ad      |
| soyad   |
| 15 byte |

union

# Set (Küme)Tipi

**Var A:set of [1..2]**

tanımına göre A kümesi  
şu 4 kümeden biri olabilir:

**A= [], [1], [2], [1,2]**

**A + B** → Küme bileşimi

**A - B** → Küme farkı

**A \* B** → Küme kesişimi

## PASCAL'da Küme Tanımlaması

```
Type otomobiller = (tofas, renault, opel, ford, toyota, hyundai);  
      otokume = set of otomobiller;  
var kume1, kume2 : otokume;  
var arabam: otomobiller;
```

## PASCAL'da Küme Tipindeki Değişkenlerin Kullanımı

```
kume1:=[tofas, opel, ford];  
kume2:=[renault, opel, toyota, hyundai];
```

**If arabam in kume1 then begin**

Kodlar =**set of** [0..255];

Harf =**set of** ['A'..'Z'];

Rakam =**set of** [0..9];

kesişim \*

Birleşim +

fark işlemler -

mantıksal operatörler (=, <> , <= , >=, in)

# Pointer (İşaretçi) Tipi

- **İşaretçi** tipi, belirli bir veriyi içermek yerine başka bir veriye başvuru amacıyla kullanılır. Bir gösterge tipi sadece bellek adreslerinden oluşan değerler ve boş(*null*) değerini içerebilen bir tiptir.
- Gösterge tipindeki değerler, gösterdikleri veriden bağımsız olarak sabit bir büyüklükte dirler ve genellikle tek bir bellek yerine sığarlar.
- Bu yüzden işaretçilerin kullanımıyla bellek kullanımı ve yönetimi daha etkin hale gelmektedir. Liste, ikili ağaç ve dizi gibi veri yapıları işaretçilerle daha kolay kullanılabilir.

*int \*pdp; pdp isimli değişken \* karakteri ile işaretçi değişken olarak tanımlanmıştır*

```
#include <stdio.h>
```

```
void main()
```

```
{ int a=4, b=7; //int tipinde a ve b değişkenleri tanımlanmış ve 4 ve 7 değerleri atanmıştır.
```

```
    int *pa,*pb; // pa ve pb adında, yine int tipinde iki işaretçi değişken tanımlanmıştır
```

```
    printf("a=%d , b=%d\n",a,b);// a = 4 , b = 7 olarak ekrana yazdırır.
```

```
    pa=&a;           //a değişkeninin adresini pa işaretçi değişkenine ata
```

```
    pb=&b;           //b değişkeninin adresini pb işaretçi değişkenine ata
```

```
    *pa=*pa+10;      // pa'nin işaret ettikleri değere 10 değerini ekler
```

```
    *pb=*pb+10;      // pb'nin işaret ettikleri değere 10 değerini ekler
```

```
    printf("a=%d, b=%d",a,b);    }
```

Burada *\*pa=\*pa+10; \*pb=\*pb+10;* ifadelerinde *pa* ve *pb* nin işaret ettikleri değerlere sırasıyla 10 değerini ekler. **pa** işaretçi değişkeni **a** değişkenine ve **pb** işaretçi değişkeni ise **b** değişkenini işaret ettiğine göre aslında değeri artırılan değişkenler **a** ve **b** dir.

Bu programın ekran çıktısı

a = 4 , b = 7

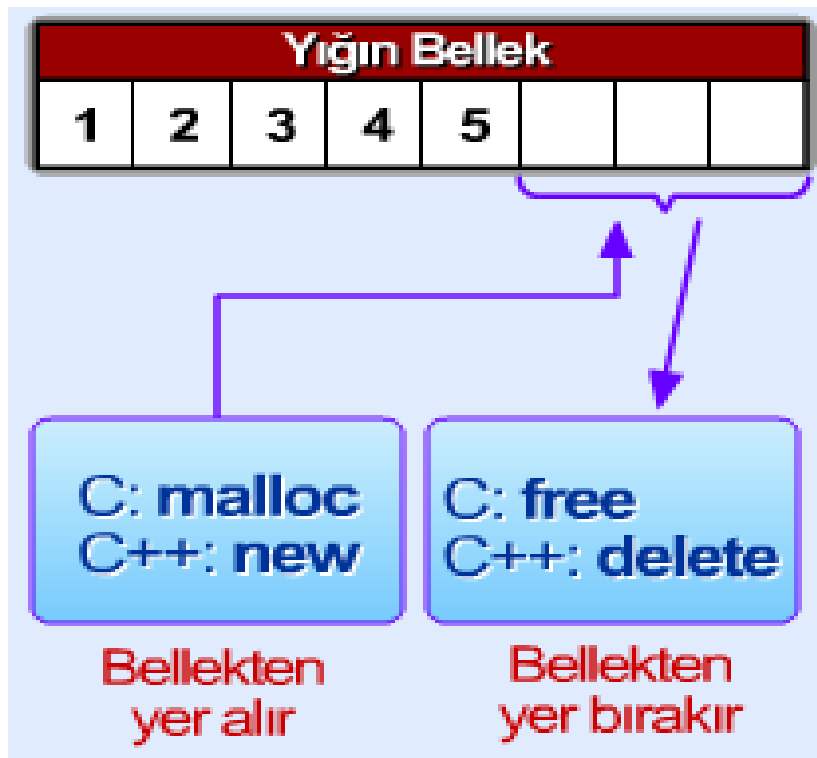
a = 14 , b = 17 olur.

# İşaretçi Veri Tipinin Sorunları

- Bir işaretçi değişkenini gösterebileceği veri tipi kısıtlanmazsa güvenlik sorunu oluşabilir. (gösterge kullanımlarında durağan tip denetimi yapılamadığı için) PL/I
- Bir işaretçi değişkeninin gösterdiği adreste geçerli bir veri olmayabilir.(Pascal, C, C++)
- GÜVENİLİRLİK tehlikeye girebilir.
- C'de bir gösterge değişken tanımlanırken, adresini tutabileceği değişken tipi de belirtilmelidir (int \*a;)
  - Java pointer kullanımına izin vermez.



# Dinamik bellek yönetimi



## Örnek

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

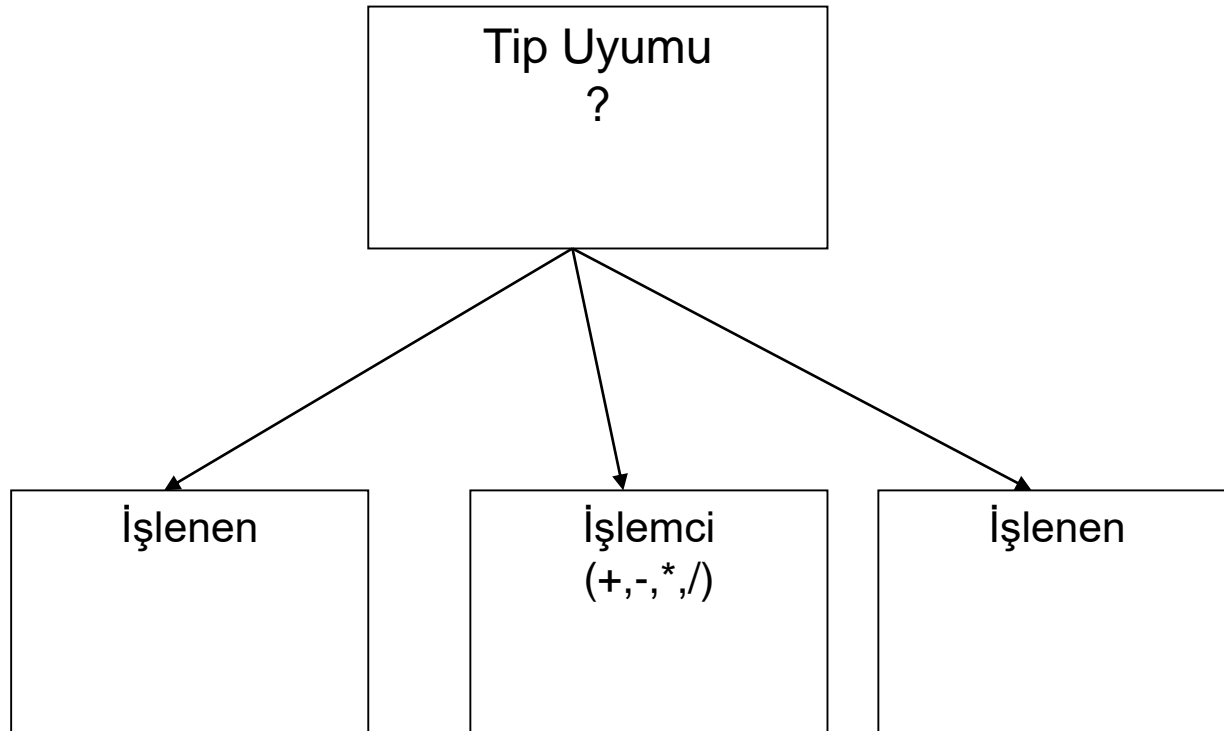
PASCAL: new, dispose

# KUVVETLİ TİPLEME

- **Kuvvetli tiplleme** (*strong typing*), farklı veri tiplerinin farklı soyutlamaları göstermeleri nedeniyle etkileşimlerinin kısıtlanmasıdır.
- Bütün tip hataları yakalanmalı
- Derleyici, her değişken ve her ifadenin tipinin belirlenebilmesi için kurallar içermeli.
- Zorunlu dönüşümler dışında eş olmayan tiplerin birbirlerine atanmasına ve altprogram çağırımlarında parametre aktarımlarına izin vermemeli.
- Pascal kuvvetli tipllemeyi destekler. C, C++ bu konuda zayıftır.

# TİP DENETİMİ

- Bir işlemcinin işlenenlerinin birbirleriyle uyumlu tipler olduğunun denetlenmesi, **tip denetimi** olarak adlandırılmaktadır.



# Tip Dönüşümleri

- Herhangi bir işlem birden fazla değişken, sabit ve operatör içerebilir. İşleme giren değişkenlerin farklı tiplerden olması durumunda sonucun hangi tipe olacağını işlem içerisinde değişken ve sabitler belirler. Böyle durumlarda işlem içerisinde hafızada en çok yer kaplayan ifadenin veri tipine göre sonucun tipi belirlenir.
- bir nesnenin kendi tipinin tüm değerlerini içeren bir tipe dönüşümü, genişleyen dönüşüm olmaktadır. Bir tamsayı değişkenin kayan noktalı tipe dönüşümü, genişleyen dönüşüm örneğidir.
- bir nesne, kendi tipindeki tüm değerleri içermeyen bir tipe dönüştürülüyorsa daralan dönüşüm gerçekleşmektedir. Örneğin, kayan noktalı tipten tamsayıya dönüşüm, daralan dönüşümdür.

```
#include <stdio.h>
void main()
{
    double x;
    x=4/3;
    printf("sonuç =%f\n",x);
}
```

```
#include <stdio.h>
void main()
{
    double x;
    x=4.0/3;
    printf("sonuç =%f \n",x);    // sonuç = 1.333333
    x=(double) 4/3;
    printf("sonuç =%f \n",x);    // sonuç = 1.333333
    x=(double) (4/3);
    printf("sonuç =%f \n",x);    // sonuç = 1.000000
}
```

Burada x değişkeni double olarak tanımlanmıştır. Double veri tipi kesirli ve çok büyük sayıları tutma özelliğine sahiptir.  $x=4/3$ ; işlemi ile 4 sayısı 3'e bölünmekte ve sonuç x değişkenine aktarılmaktadır ve printf komutu ile ekrana sonuç =1.000000 olarak yazdırılmaktadır. Bunun sebebi 4 ve 3 sabit bilgilerinin int veri tipinde birer bilgi olmasındandır.

Çoğu programlama dili daralan dönüşümlere izin vermezler.

## Güvenilirlik olumsuz etkileniyor.

- Hataların farkedilmesini engellenebilir.
- Zorunlu dönüşümün gerçekleştiği dillerde, tip denetimi kısıtlanmakta

Aşağıdaki programda  $a=b*d$  ifadesindeki  $d$ 'nin yanlışlıkla yazıldığını düşünelim

```
void main(){  
int a,b,c;  
float d;  
a=b*d;  
}
```

derleyici,  $b$  yi *float* tipine dönüştürecek ve çarpım işlemi *float* olarak gerçekleşecektir. Zorunlu dönüşüm, tip uyumsuzluğu hatasını engellemiştir.

## Dışsal (*explicit*) veya örtülü (*implicit*) Tip dönüşümleri

- derleyici tarafından gerçekleştirilen dönüşümler, **zorunlu dönüşüm** (*coercion*) olup bunlara örtülü tip dönüşümleri denir.
- Derleyici tasarımı sırasında, bir işlemcinin iki işleneni aynı tipte değilse, dönüşüme izin verilip verilmeyeceği belirlenmeli ve dönüşüm gerçekleşecekse bu bunun için gerekli olan kuralları belirlenmeli ve gerekli dönüşümü yapmalıdır.
- Pascal ve C gibi birçok programlama dilinde bir tamsayı ve bir gerçel sayı toplanmak istendiğinde, tamsayının değeri gerçel sayıya dönüştürülür ve işlem gerçel sayılar üzerinde yapılır

$$\begin{array}{ccc} a & + & b \\ \text{Reel} & & \text{int} \end{array} \rightarrow \text{COERCION} \rightarrow \begin{array}{ccc} a & + & b \\ \text{Reel} & & \text{Reel} \end{array}$$

# Dışsal Tip Dönüşümleri

- Bir operandtan önce değerin dönüştürülmesi istenen tip belirtiliyorsa buna dışsal tip dönüşümü denir.

**Toplam=(int)toplam2 + 100**

Bu ifadedeki toplam2 değişkeni önce int tipine dönüştürülür daha sonra 100 ile toplanır.



# Bölüm Özeti

## (veri tipi kavramı)

- Temel ve Bileşik Veri Tipi kavramları
- Sayısal, Mantıksal, Karakter, Karakter String ve Kullanıcı Tanımlı Sıralı Tipler;
- Diziler, Record (Kayıt) Tipi, Union (Bileşim) Tipi, Set (Küme)Tipi ile Pointer (işaretçi) Tipi
- Kuvvetli Tipleme, Tip Denetimi ve Tip Dönüşümleri kavramları incelenmiştir.