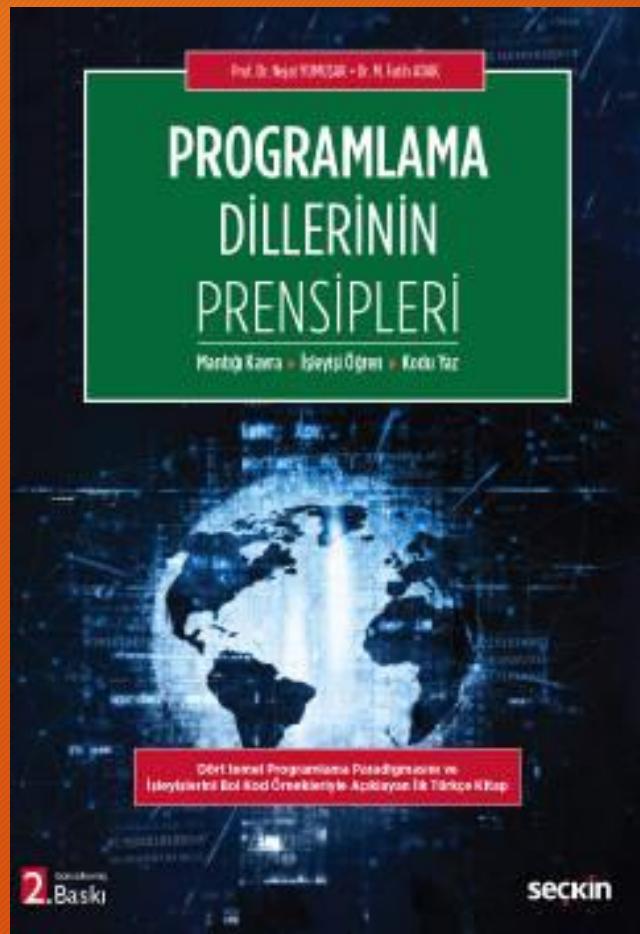


# Programlama Dillerinin Prensipleri

## Hafta 1 - Giriş

Dr. Öğr. Üyesi M. Fatih ADAK

# Ders Kitabı



<https://www.seckin.com.tr/kitap/989255263>

# Ders Akışı

Hafta	Konular
1	Giriş ve Programlama Kavramlarının Tanıtılması
2	Dillerin tarihçesi ve evrimi
3	Dil tanımlanması ve Dil çevrimi
4	Temel programlama kavramları, Veri tipleri ve yapıları
5	Bağlama Kavramları ve İsim Kapsamları
6	Yapısal programlama
7	Altprogramlar ve Modülasyon
8	Parametre aktarım yöntemleri
9	Nesne yönelimli programlama kavramları
10	Programlama dillerinde hata yakalama
11	Programlama dillerinde eşzamanlılık
12	Yorumlamalı Diller ve Python
13	Fonksiyonel programlama kavramları
14	Mantıksal programlama kavramları

# Değerlendirme Sistemi

- 1. Ödev (Java) : % 19.8
  - 2. Ödev (C) : % 19.8
  - Kısa Sınav (Test) : % 5.4
  - Ara Sınav (Klasik) : % 15
  - Final (Test-Klasik) : % 40
- 
- Ödevler bireyseldir.

# Ödev ve Kod Derslerinde Araçlar

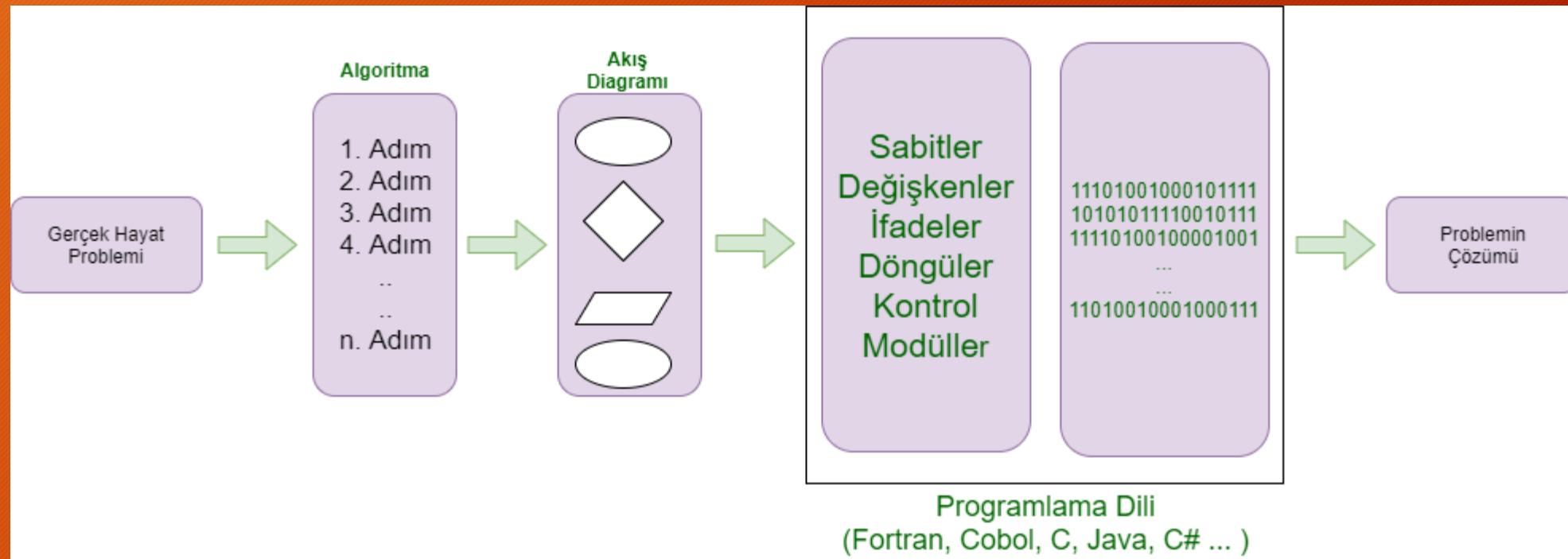
- Java için Eclipse 2021-12
  - <https://www.eclipse.org/downloads/>
- C Dili için
  - MinGW
  - <http://www.mingw.org/>
- Lisp Dili için
  - GNU Common Lisp 2.6.12
  - <https://www.gnu.org/software/gcl/>

# Bu Haftaki İçerik

- Programlama Dillerinin Sınıflandırılması
- Programlama Dili Paradigmaları
- Programlama Dillerinin Değerlendirme Ölçütleri
- C Dili
- Java Dili
- Lisp Dili
- Prolog

# Programlama Dili Nedir?

- Bir problemin çözümünün bilgisayardaki gerçekleştirimini ifade etmek üzere program oluşturmak için kullanılan araca denir.



# Yazılımın Uygulama Alanlarına Göre Gruplandırılması

- Bilimsel ve mühendislik yazılımları
- Mesleki yazılımlar
- Yapay zeka yazılımları
- Görüntüsel yazılımlar
- Sistem yazılımları

# Programlama Dilinin Amaçları

- Bir programlama dili makinalara talimat vermek için gerekli bir araçtır.
- Programcılar arasında iletişim için gerekli bir vasıtabır.
- Yüksek seviyeli tasarımları ifade etmek için gerekli bir araçtır.
- Algoritmaları göstermeye yarayan bir notasyondur.
- Genel kavramlar arasındaki yakınlıkları ifade etmeye yarayan bir yoldur.
- Çözümlerin ve çözüm yollarının test edilmesi için gerekli bir araçtır.
- Bilgisayarlı cihazları kontrol etmek için gerekli bir vasıtabır.

# Programlama Dillerinin Sınıflandırılması

- Seviyelerine Göre Sınıflandırma
- Uygulama Alanlarına Göre Sınıflandırma
- Tasarım Paradigmalarına Göre Sınıflandırma

# Programlama Dillerinin Seviyelerine Göre Sınıflandırma

- Makine Dili
- Alçak Seviyeli Diller
- Orta Seviyeli Diller
- Yüksek Seviyeli Diller
- Çok Yüksek Seviyeli Diller

# Makine Dili

- Bir bilgisayarın doğrudan anlayabildiği bir dildir.
- Bilgisayarın ana dili olarak kabul edilir.
- Makine dili taşınabilir değildir ve makineye özgü yazılması gereklidir.
- Makine dilinde kod yazmak çok zahmetli, çok zaman alıcı ve uğraştırıcıdır.

# Alçak Seviyeli Programlama Dili

- **Sembolik Dil (Assembly)**

- Makine dili kullanımının getirdiği problemleri ortadan kaldırmak üzere yapılan çalışmalarda
- Önce makine dilinin anlaşılma zorluğunu kısmen de olsa ortadan kaldırmak üzere sembolik dil geliştirilmiştir.
- Sembolik dilde 0 ve 1'ler yerine İngilizce ifadeler yer almaktaydı.
- Burada bellekten okuma yazma yerine çok daha hızlı olması açısından register'lar kullanılır.

Sembolik dilde ekrana Merhaba yazdırılması
mesaj db 'Merhaba', 0x0d, 0x0a, '\$' mov dx, mesaj mov ah, 9 int 0x21

# Orta Seviyeli Diller

- Sembolik diller bilgisayar kullanımını hızla arttırmıştır.
- Ancak çok basit işlemler için bile birçok komut gerekmektedir.
- Ayrıca sembolik diller her seferinde makine diline çevrilip öyle çalıştırılıyordu. Bu işlem program hızını 30 kat yavaşlatıyordu.
- Grace Hopper, bu problemin çözümü için derleyici fikrini ortaya attı.
- Program kodu bir kez derlenip makine diline çevrilecek ve bir daha bu işleme gerek kalmayacaktı.

# Orta Seviyeli Diller

- Ada, C gibi diller örnek verilebilir.
- Daha az kayıplla makine diline çevrilebildiğinden daha hızlı çalışır.
- Program yazmak yine zordur fakat sembolik dile göre oldukça kolaydır.

# Yüksek Seviyeli Diller

- Fortran ilk yüksek seviyeli dildir.
- Program yazmak daha kolay fakat orta seviyeli dillere göre program hızı daha yavaştır.
- Bu seviyedekiler 3. kuşak diller olarak kabul edilir.

# Çok Yüksek Seviyeli Diller

- Genellikle algoritmik yapı içermeyen görsel bir ortamda yazılan dillerdir.
- 4. Kuşak olarak isimlenidirlirler.
- Java, C#, Visual Basic, Access, Oracle Forms bu seviyeye örnek verilebilir.
- Program hızları makine dillerine göre oldukça yavaştır.

# Programlama Dillerini Uygulama Alanlarına Göre Sınıflandırma

- Bilimsel ve Mühendislik Uygulama Dilleri
  - Pascal, C, Fortran
- Veritabanı Dilleri
  - MSSQL, Oracle Forms, XBASE
- Genel Amaçlı Programlama Dilleri
  - Pascal, C, Basic, Java
- Yapay Zeka Dilleri
  - Prolog, Lisp
- Modelleme Yapmak Üzere Geliştirilen Simülasyon Dilleri
  - GPSS, Simula67

# Programlama Dillerini Uygulama Alanlarına Göre Sınıflandırma

- Makro Diller (Script Diller)
  - awk, Perl, Python, Tcl, Javascript
- Sistem Programlama Dilleri
  - C (UNIX işletim sisteminin %80'i C dili ile geri kalanı sembolik dil ile yazılmıştır.)
- Ticari Uygulamalara Yönelik Programlama Dilleri
  - Cobol

# Programlama Dillerini Tasarım Paradigmalarına Göre Sınıflandırma

- Emir Esaslı Programlama
- Nesneye Yönelik Programlama
- Fonksiyonel Programlama
- Mantık Esaslı Programlama

# Programlama Dillerini Tasarım Paradigmalarına Göre Sınıflandırma

- Emir Esaslı (Imperative) Programlama

Deyim 1
Deyim 2
.
.
.
Deyim n

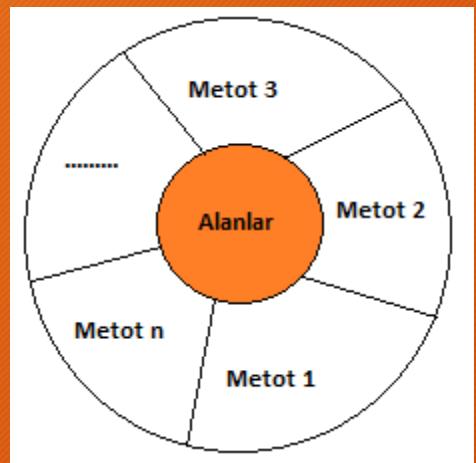
- Emir esaslı programlama dilleri işlem tabanlı olup, bir program bir dizi işlem olarak görülür.
- Bu diller yaygın olarak kullanılan ilk dil grubudur.
- C, Fortran, Pascal, Cobol örnek olarak verilebilir.

- Örneğin, atama işlemi bir deyimdir.

Farklı dillerde atama işlemleri
C dili x=13
Pascal dili x:=13
APL dili x<--13
Scheme veya Lisp dili (setq x 13)

# Programlama Dillerini Tasarım Paradigmalarına Göre Sınıflandırma

- Nesneye Yönelik Programlama



- Temeli simula67 programlama dilidir.
- Nesnelerin sınıfı ve alt sınıflara gruplanması, nesneye yönelik programlamanın temel noktasıdır.
- Karmaşık veri nesneleri ve bu veriler üzerinde çalışacak işlemler (metotlar) tasarılanır.



Nesneye yönelik programlamanın genel yapısı

# Programlama Dillerini Tasarım Paradigmalarına Göre Sınıflandırma

- Fonksiyonel Programlama

- Veriler ve sonucu elde etmek için veriye uygulanacak fonksiyonel dönüşümler bu paradigmın temelini oluşturur.
- Lisp, Scheme ve ML dilleri bu paradigmaya örnektir.



- Fonksiyonel programmanın temelini oluşturan parçalar



# Programlama Dillerini Tasarım Paradigmalarına Göre Sınıflandırma

- Mantık Esaslı Programlama

- Bir işin nasıl yapılacağının belirtilmesi yerine, ne yapılması istendiğinin belirtilmesi olarak görülür.
- Belirli bir koşulun varlığını kontrol ederek ve koşul sağlanıyorsa, uygun bir işlem gerçekleştirerek çalışırlar.
- Emir esaslı programlamaya benzer fakat deyimler sıralı olarak işlenmez.

Mantıksal paradigmayı destekleyen dillerin sözdizimi
Şart_1 → Hareket_1
Şart_2 → Hareket_2
Şart_3 → Hareket_3
...
...
...
Şart_n → Hareket_n

# Programlama Dillerinin Değerlendirme Ölçütleri

- Her programlama dili bir düşünce biçimini olduğundan binlerce programlama dili vardır denilebilir.
- Çözülecek problemin tipine ve uygulama alanına göre programlama dilleri arasında seçim yapmak için çeşitli değerlendirme ölçütlerine ihtiyaç duyulmaktadır.
- Her alan için en iyi olan bir programlama dili yoktur.

# Programlama Dillerinin Değerlendirme Ölçütleri

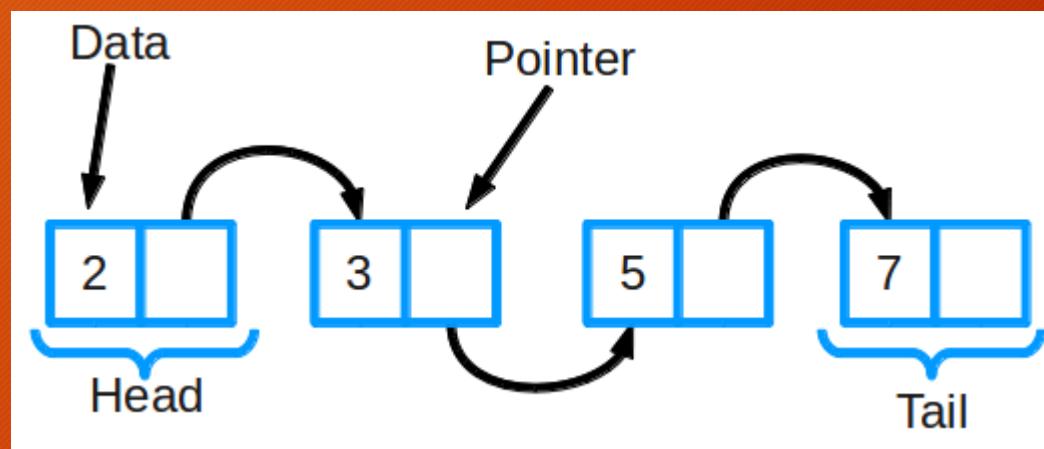
- İfade Gücü (Expression Power)
- Veri Türleri ve Yapıları (Data Types and Structures)
- Giriş/Çıkış Kolaylığı (Input/Output Facilities)
- Taşınabilirlik (Portability)
- Alt programlama Yeteneği (Modularity)
- Verimlilik (Efficiency)
- Okunabilirlik (Readability), Yazılabilirlik
- Esneklik (Flexibility)
- Öğrenme Kolaylığı (Pedagogy)
- Genel Amaçlılık (Generality)
- Yapısallık (Structrulness)
- Nesne yönelimlilik (Object Orientation)

# Programlama Dillerinin Değerlendirme Ölçütleri

- İfade Gücü (Expression Power)
  - Algoritmayı tasarlayan kişinin niyetlerini açık bir biçimde yansıtılmasına olanak tanımı
  - Günümüz popüler programlama dillerinin ifade gücü yüksektir.

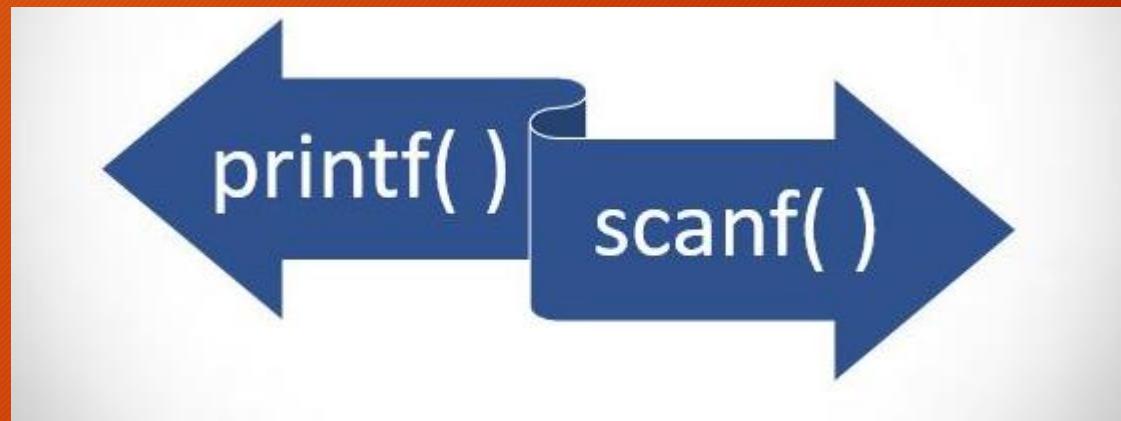
# Programlama Dillerinin Değerlendirme Ölçütleri

- Veri Türleri ve Yapıları (Data Types and Structures)
  - Çeşitli veri türlerini (tamsayı, gerçek sayı, karakter...) ve veri yapılarını (diziler, bağlı liste, kuyruk, yapılar vs.) destekleme yeteneğidir.



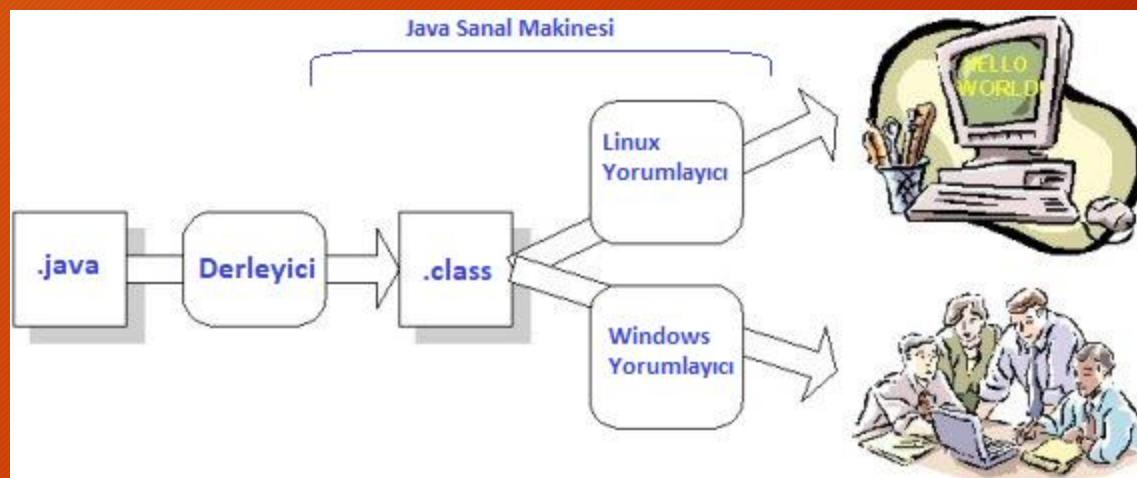
# Programlama Dillerinin Değerlendirme Ölçütleri

- Input Output Kolaylığı
  - Program yazmayı kolaylaştıran ve ifade gücünü arttıran bir özelliktir
  - Örneğin C dilinde bu ölçüt çok yüksek değildir.



# Programlama Dillerinin Değerlendirme Ölçütleri

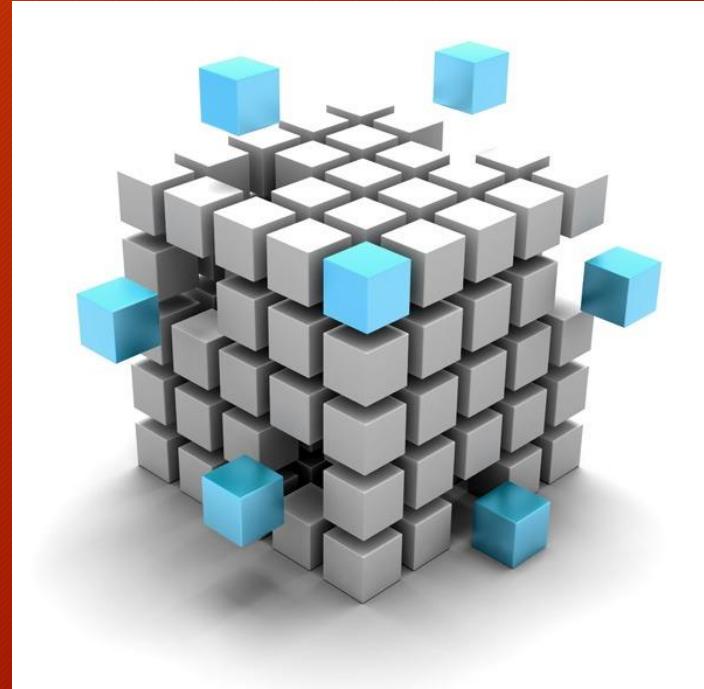
- Taşınabilirlik
  - Bu terim kaynak kod için kullanılır.
  - Bir sistemde yazılmış kaynak kodun bir başka sistemde de sorunsuz derlenip çalışmasıdır.
  - Dillerin seviyesi düştükçe taşınabilirlik azalır.
  - Hiçbir dil için mükemmel taşınabilirlik mümkün değildir.



# Programlama Dillerinin Değerlendirme Ölçütleri

- Alt Programlama Yeteneği

- Bir programı parçalar halinde yazmayı desteklemesidir.
- Yapısal Programlama tekniğinin vazgeçilmez bir parçasıdır.
- Yazılacak kodu oldukça azaltır.
- Program kodlarının anlaşılmasını kolaylaştırır.



# Programlama Dillerinin Değerlendirme Ölçütleri

- **Verimlilik**

- Amaç koda dönüştürülmüş programların hızlı çalışılmasına verimlilik denir.
- Verimlilik derleyici, dil seviyesi ve dilin genel yapısına bağlıdır.
- Çalışabilir kodun küçüklüğü ile çalışma hızı arasında doğrusal bir ilişki vardır.
- C programları hızlı çalışır ve az yer kaplar.

# Programlama Dillerinin Değerlendirme Ölçütleri

- Okunabilirlik
  - Kaynak kodun çabuk ve kolay bir biçimde algılanabilmesi anlamına gelir.
  - Okunabilirlik güncelleştirmeyi kolay kılar ve proje grubu halinde kodun üzerinde çalışılabilir mesine olanak sağlar.
  - En iyi program kodu **anlaşılamayan ama çok zekice yazılmış kod değildir.**
  - En kolay okunabilen ve anlaşılabilen kod en iyi koddur.

"Babaannene iki dakikada açıklayamayacağı tek bir satır kodu bile programına ekleme, hatta babaannen Ada Lovelace olsa bile."

Anonim



# Programlama Dillerinin Değerlendirme Ölçütleri

- Esneklik

- Esneklik programlama dilinin programcayı kısıtlamamasıdır.
- Esnek dillerde birçok işlem, programcı için serbest bırakılmıştır.
- Bu deneyimsiz programcılar için hata yapma riskini arttırmır.
- C esnek bir dil iken Java esnekliği çok kısıtlanmış bir dildir.

C dilinde karakter ve tamsayıların birbirine kolayca atanabilmesi

```
int main(){  
    int x=97;  
    char c=x;  
    char b='b';  
    int y=b;  
    printf("%c\n",c);  
    printf("%d\n",y);  
    return 0;  
}
```

# Programlama Dillerinin Değerlendirme Ölçütleri

- Öğrenme Kolaylığı
  - Programlama dillerinin seviyesi arttıkça öğrenme kolaylığı artar.
  - Yüksek seviyeli dillerin popüler olması öğrenme kolaylığına bağlıdır.
  - C dili öğrenmesi zor bir dil iken Java aksine basittir.

# Programlama Dillerinin Değerlendirme Ölçütleri

- Genel Amaçlılık

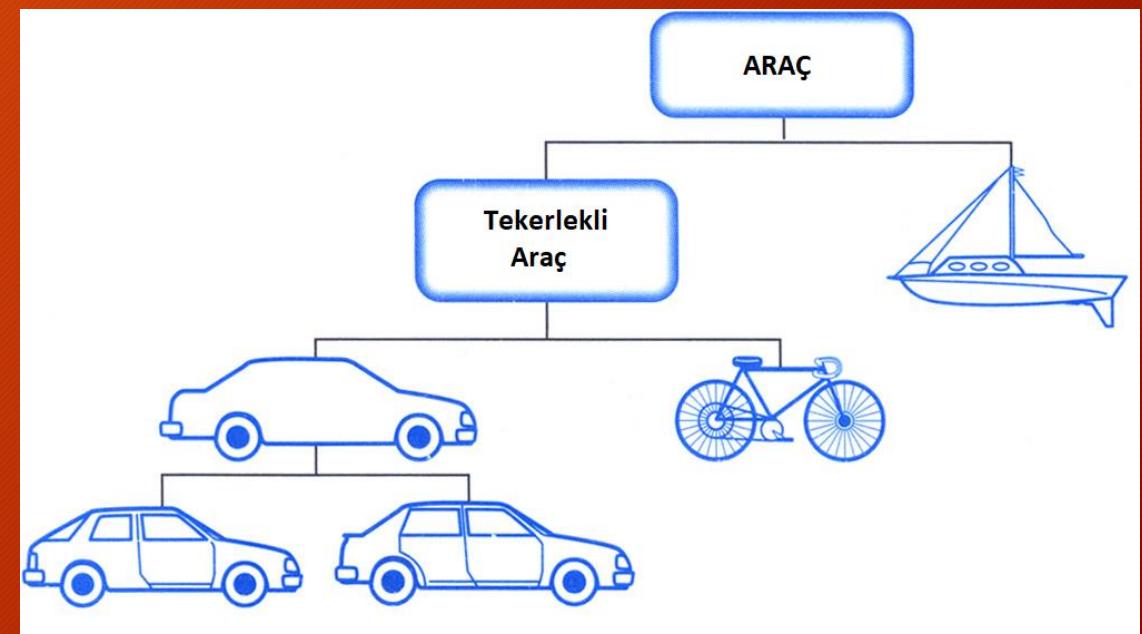
- Programlama dillerinin çok çeşitli uygulamalarda etkin olarak kullanılabilirliğidir.
- Cobol dili ticari uygulamalarda etkin bir dil iken mühendislik uygulamalarında tercih edilmez.
- Java dilinin genel amaçlılığı yüksektir.

# Programlama Dillerinin Değerlendirme Ölçütleri

- Yapısalılık
  - Yapısalılık bir programlama tekniğidir.
  - Bu teknigi kullanan dillerde bloklar halinde yazım ön plandadır.
  - Alt programlar yoğun olarak kullanılır.
  - C dili iyi bir örnektir.
  - Yapısal programlama 4 ana ilke üzerine kurulmuştur.
    - Böl ve Yönet
    - Veri Gizleme (lokal değişkenler)
    - Tek Giriş ve Çıkış
    - Döngüler ve Diğer Kontrol Yapıları

# Programlama Dillerinin Değerlendirme Ölçütleri

- Nesne Yönelimlilik
  - Veri + Program = Nesne
  - Büyük programların yazılması için tasarlanmış bir tekniktir.
  - Üç temel üzerine kurulmuştur.
    - Kapsülleme
    - Çok Biçimlilik
    - Kalıtım



# C Dili

- C dili ilk olarak Dennis Ritchie tarafından 1972 yılında Bell laboratuvarında geliştirilmiştir.
- C dili işletim sistemi dili olarak bilinir.
- C dili emir esaslı ve yapısal bir dildir.

Örnek bir C kodu

```
#include "stdio.h"
int main(){
    int yas;
    printf("Lutfen yasinizi girin:");
    scanf("%d",&yas);
    int dogumyili = 2017-yas;
    printf("Dogum Yiliniz:%d\n",dogumyili);
    return 0;
}
```

# make Dosyası

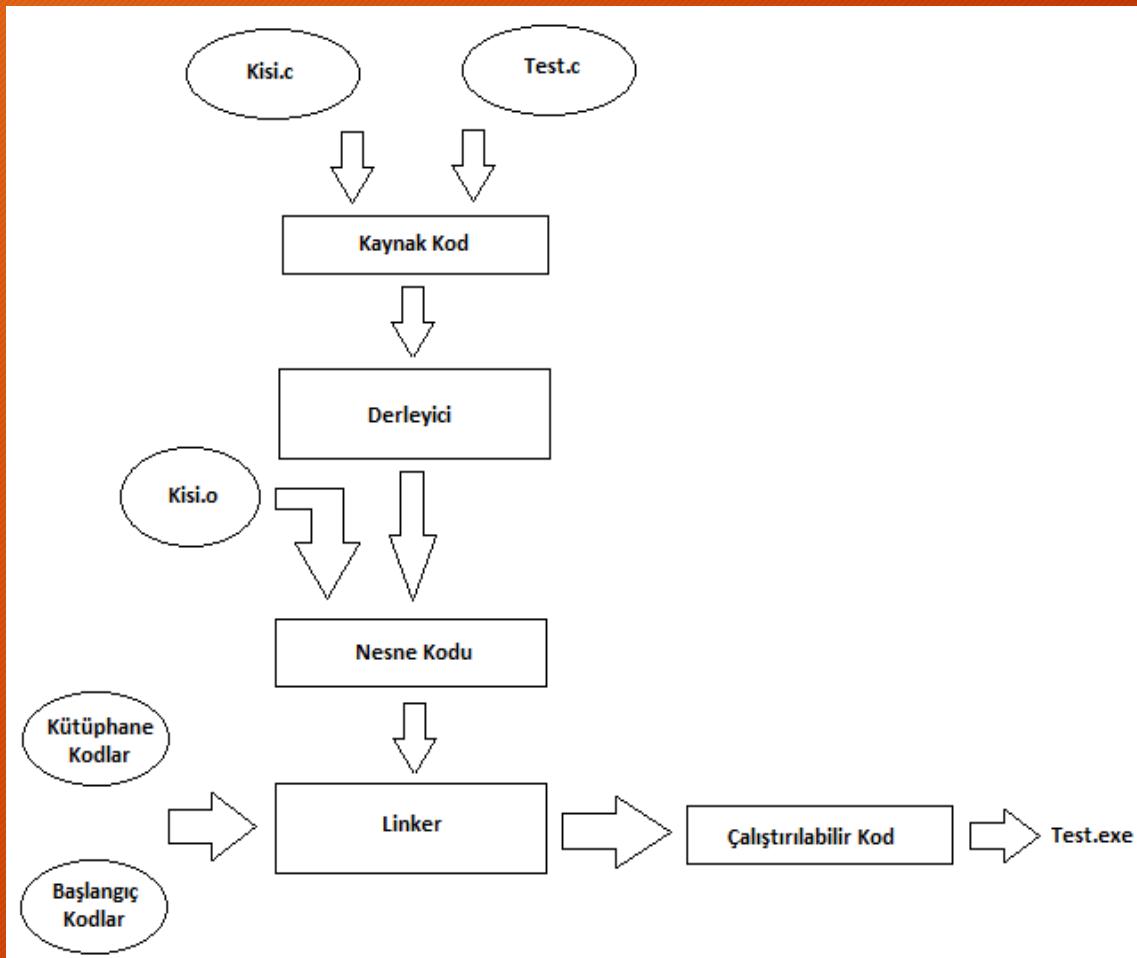
- C dilinde gelişmiş programların tasarlanmasıında birden fazla kaynak kod ve başlık dosyası kullanılabilmektektir. Bu durumda komutların tek tek el ile her seferinde komut satırına girilmesi zahmetli ve zaman alan bir iştir.
- Bu komutların bir dosyaya yazılıp dosyanın komut satırından çağrılması zamandan kazanç sağlayacaktır.
- Bu dosyanın ismine make dosyası (makefile) denilmektedir.

```
Örnek C kodunu derlemek için make dosyası
hepsi: derle calistir

derle:
    gcc -I ./include/ -o ./lib/Kisi.o -c ./src/Kisi.c
    gcc -I ./include/ -o ./bin/Test ./lib/Kisi.o ./src/Test.c

calistir:
    ./bin/Test
```

# C Dilinde Derlenme Süreci



# Java Dili

- İlk olarak Oak ismiyle tasarlanmış ve gömülü sistemler için kullanılmıştır.
- Daha sonraları Java ismini alıp internet uygulama geliştirme için kullanılmıştır.
- Genel bir programlama dili olan Java her platformda kullanılabilmektedir.
- Tamamen nesne yönelimlidir. En ufak program için yine Sınıf yazılmalıdır.

# Lisp Dili

- Sembolik veri işleme için tasarlanan Lisp dili fonksiyonel programlama dilidir.
- Daha çok yapay zeka çalışmalarında kullanılmıştır.
- İki temel veri yapısı içerir bunlar **Atom** ve **Liste** dir.

Ekrana merhaba yazan Lisp fonksiyonu

```
(Defun MerhabaYaz()
  "Merhaba"
)
```

# Prolog Dili

- 1970 yılında İngiliz ve Fransız ortaklığında geliştirilmiştir.
- Mantıksal programlama dili olan Prolog'ta bildirme esaslı bir yapı kullanılır.
- Prolog'u iki temel konsept oluşturur.
  - Olaylar
    - Doğru olan durumlardan oluşur.
  - Kurallar
    - Kuralları ifade etmek için önermelerden faydalанılır.
      - Atomik önermeler
      - Bileşik terimler

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 2. Baskı, Seçkin Yayıncılık, 2021
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016

# Programlama Dillerinin Prensipleri

## Hafta 2 - Programlama Dillerinin Tarihçesi ve Çeşitleri

Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Makine Dili
- Sembolik Diller
- Derleyicili Diller
- Yorumlayıcılı Diller
- Script Dilleri

# İlk Yıllar

- 1800'lü yıllarda Charles Babbage, programlanabilir bilgisayar fikrini ilk ortaya attı.
- Bunlar fiziksel hareketlerden ibaretti elektriksel sinyallere dönüşmesi ENIAC ile 1942 yılını bulacaktı.
- Bu yıllarda programlama denilince akla gelen, çözülecek probleme ilişkin bir devre tasarlamak anlamaktaydı.
- Bu yıllarda sadece Elektronik Müh. Programlama yapabilmekteydi.

# Makine Dili

- Sadece ikili sayı sisteminde oluşturmaktadır. (0-1)
- ENIAC sadece matematiksel işlemler yabilen (0-1) ile çalışan bir makine dilidir. Aynı zamanda ENIAC makinelerin adıdır.



# Sembolik Dil

- 1950'li yıllarda Sembolik dil (Assembly) geliştirildi.
- İkili kodlarla program yazmak oldukça zor olduğu için ikili kodlar sembollerle ifade edilmiştir.
- Komutların adlandırılması ve akılda kalması kolaylaştırılmıştır.
- Tamamen donanıma bağlı düşük düzeyli bir programlama dilidir.

# Sembolik Dil

- Burada bellekten okuma yazma yerine çok daha hızlı olması açısından register'lar kullanılır.
- Sembolik diller bilgisayar kullanımını hızla arttırmıştır.
- Ancak çok basit işlemler için bile birçok komut gerekmektedir.
- Ayrıca sembolik diller her seferinde makine diline çevrilip öyle çalıştırılıyordu. Bu işlem program hızını 30 kat yavaşlatıyordu.

The diagram illustrates the assembly language and its conversion to machine code. On the left, assembly code is shown in a box with a red arrow pointing to it labeled 'Assembler'. To the right is a table of swap operations, and below that is a box containing binary machine code.

Swap İşlemi		
muli	\$2, \$5, 4	
add	\$2, \$4, \$2	
lw	\$15, 0(\$2)	
lw	\$16, 4(\$2)	
sw	\$16, 0(\$2)	
sw	\$15, 4(\$2)	
jr	\$31	

```
000000001010000100000000000011000  
00000000100011100001100000100001  
10001100011000100000000000000000  
10001100111100100000000000000000  
10101100111100100000000000000000  
10101100011000100000000000000000  
00000011111000000000000000000000
```

# Derleyici Fikri

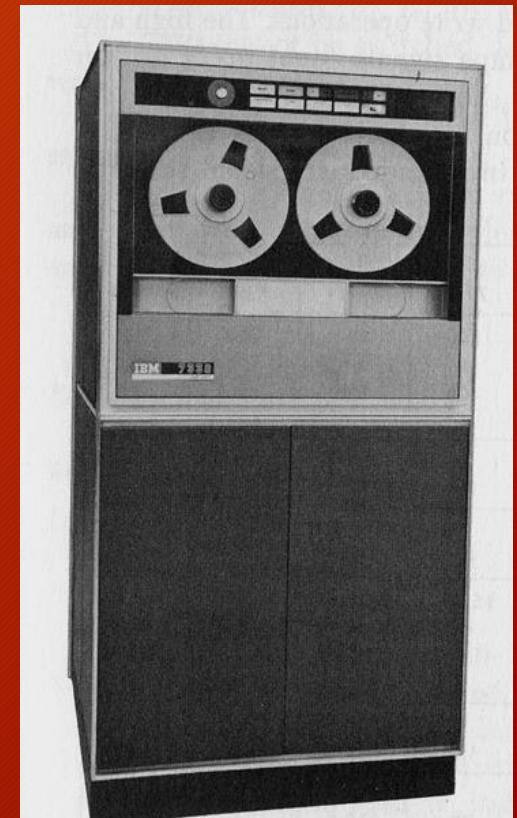
- Sembolik dil bir yorumlayıcıydı.
- Her seferinde makine diline çevrilip öyle çalıştırılıyordu.
- Grace Hopper, bu problemin çözümü için derleyici fikrini ortaya attı.
- Program kodu bir kez derlenip makine diline çevrilecek ve bir daha bu işleme gerek kalmayacaktı.

# Fortran Dili

- Fortran (FORmul TRANslating System) 1954 yılında IBM firmasında John Backus tarafından geliştirildi.
- Do deyimleri, G/Ç deyimleri ve atama deyimleri içeriyordu.
- Matematiksel denklemlerin çözümü amaçlanmıştı.
- Çalışma sırasında veri tipleme ifadeleri ve bellek tahsis'i yoktu.
- Fortran ifadeleri İngilizce kelimelerden oluşuyordu ve sembolik dile göre anlaşılması çok kolaydı.

# Fortran Dili

- Fortran diline bir derleyici yazılıp piyasaya sunulması 1957 yılını bulacaktır.
- Fortran derleyicisi bir teyp biriminde saklanıyordu.
- Fortran'ın daha sonra birçok sürümü yayınlanmıştır.



# Fortran'ın Sürümleri

- Fortran I
  - Taşınabilirlik yönünden oldukça kötü bir dildir.
- Fortran 66
  - Karakter türü verileri işlemeye çok kısıtlı. Yapısal programlama desteklenmiyor.
- Fortran 77
  - ANSI karakter türü verileri işleyebiliyor.
  - Yapısal programlamayı destekliyor.

# Fortran 90 ve 95

- Bu yıllarda C dili çok iddalı ve popüler olmaya başlamıştı.
- Dolayısıyla Fortran 90'da C'deki birçok özellik eklenmeye çalışıldı.
- Fortran 90'da pointer, özyineleme, bit düzeyinde işlem ve dizi yapıları daha kullanılabilir hale getirildi.
- Fortran 95'in ise temel hedefi taşınabilirliğin mükemmel hale getirilmesi olmuştur.
- Fortran 95'te nesneye yönelikプログラما özellikleri de eklenmeye çalışılmıştır.

# Fonksiyonel Paradigma

- Fonksiyonel paradigma ile ilk 1958 yılında tanışılmıştır.
- Fonksiyonel programlama dili olarak geliştirilen Lisp daha önce tanıtılan dillerden çok farklıydı.
- Lisp daha çok yapay zeka uygulamaları için kullanılmaktaydı.
- Atom ve Liste adı verilen iki veri yapısına sahipti.
- Lisp'in daha sonraki iki sürümü
  - Scheme (1970)
  - Common Lisp (1984)

# Algol Dili

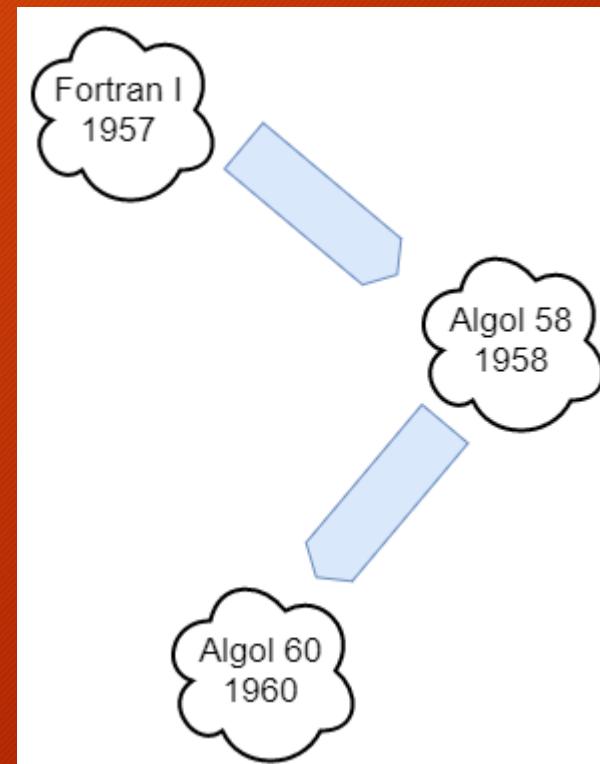
- 1958 yılında Avrupalı ve Amerikalı bir komisyonun Fortran I'den esinlenerek Zürih'teki çalışmaları sonucu yüksek seviyeli dil olan Algol (ALGOrithmic Language) geliştirmiştirlerdir.
- Bu dile ilk olarak Algol 58 ismi verilmiştir.
- Algol 60
  - Algol 58'e eklemeler yapılmıştır.
  - Daha matematiksel notasyonlara yakın ve kolay okunabilir bir dil olmuştur.
  - Makine diline kolaylıkla çevrilebilen bir dil olmuştur.

# Algol 60'ın Başarılı ve Eksik Yönleri

Başarılı Yönleri	Eksik Yönleri
Yapısal programlama tekniği benimsenmiştir	Aşırı esnek olmasından dolayı anlaşılabilirliği düşmüştür.
Pass by value ve pass by name desteklenmektedir.	Yürütmeye verimsizliğe götürecek esnek yapısı bulunmaktadır.
Özyineleme desteklenmektedir.	I/O ifadelerinde yetersizlik ya da zayıflık bulunmaktadır.
Stack-dinamik dizilere izin verilmiştir.	

# Algol Dilinin Soy Ağacı

- Bilim adamlarının eğitim ve araştırma aracı olarak Algol dilini kullanmasına rağmen IBM tarafından desteklenmemiştir.
- Çünkü IBM bu sırada Fortran’ı desteklemekteydi.
- O zamanlar için IBM'in bilişim sektörünün %80'ine sahip olduğu düşünüldüğünde bu durum Algol için bir dezavantaj olmuştur.

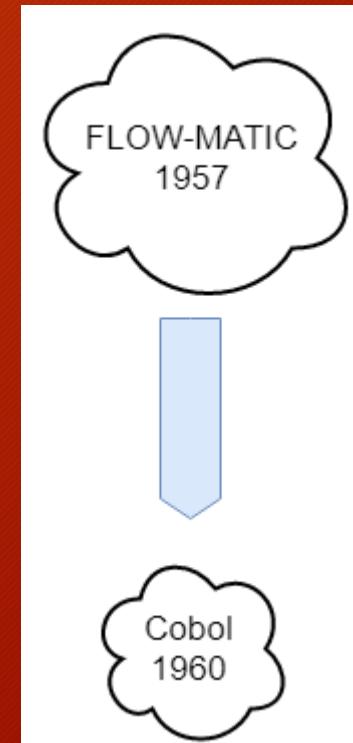


# Algol 68

- Algol 60'tan yaklaşık 8 sene sonra geliştirilmiştir.
- Kullanıcı tanımlı veri tiplerini destekleyen ilk programlama dilidir.
- Aynı zamanda dinamik dizi kavramına izin veren ilk dildir.
- Algol 68 öğrenilmesi oldukça zor olan bir gramer ve dil yapısına sahipti.

# Cobol

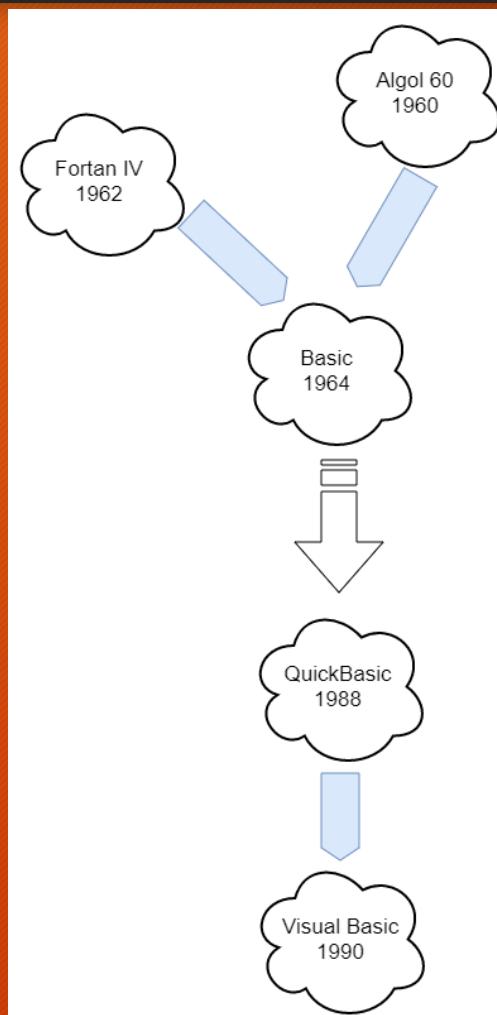
- 1959 yılının Mayıs ayında Cobol (Common Business Oriented Language) tanıtılmıştır.
- Bu dil savunma sektörünün çalışmaları sonucu ortaya çıkmıştır.
- İşletmelere yönelik bir dildir.
- Bilgisayar büyük miktarda bilgi giriş-çıkışının yapıldığı uygulamalar için geliştirilmiştir.
- Hiyerarsik veri yapıları ve yan anlamlı isimlerin görüldüğü ilk dildir.
- Fonksiyonlar desteklenmemektedir.
- 1990'lı yıllarda nesne yönelimli versiyonu üretilmiştir.



# Basic

- 1964 yılında tanıtılmıştır.
- Öğrenilmesi oldukça kolaydır.
- Dolayısıyla daha çok eğitim amaçlı kullanılmıştır.
- Uzaktaki bir bilgisayarla bağlantı kuran ilk programlama dilidir.
- İlk başlarda sadece 14 komuta sahipti ve tek veri tipi vardı.
  - LET, PRINT, GOTO, ...
  - number veri tipi (kayan noktalı ve tamsayı)

# Basic Soy Ağacı



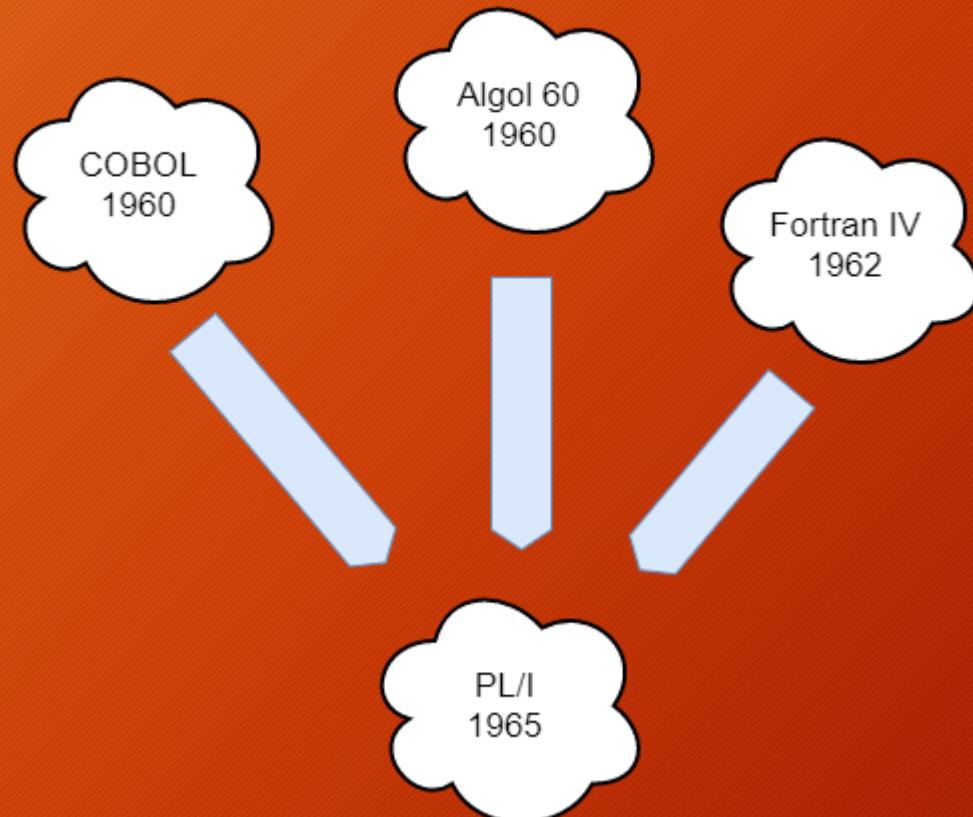
# Basic Dilinin Genel Özellikleri

- Kolay ve genel amaçlı bir dil
- Açık ve anlaşılır hata mesajlarına sahip
- Küçük boyutlu programları hızlı bir şekilde çalıştırabilir.
- Kullanımı için donanım bilgisine sahip olmaya gerek yok.
- Kullanıcıyı işletim sistemi ayrıntılarından dahi soyutlayabilmektedir.
- 1990 yılında nesne yönelimli uyarlama olan visual basic sunulmuştur.

# PL/I (Programming Language One)

- 1965 yılında bilimsel ve işletme problemlerine çözüm sağlayabilmek için geliştirilmiştir.
- Floating-point ve desimal veri tiplerini desteklemektedir.
- Eşzamanlı çalışan alt programlara izin vermektedir.
- Özyinelemeyi desteklemektedir.
- Pointer kullanımına izin vermektedir.
- Hafıza gereksinimi yüzünden karmaşık ve tasarım yönünden iyi değildir.

# PL/I Soy Ağacı



# Simula 67

- İlk olarak simülasyon için tasarlanmıştır.
- İlk nesne yönelimli dildir.
- Algol 60'ın genişletilmiş versiyonudur.
- Veri soyutlamasına imkan veren sınıfları desteklemektedir.

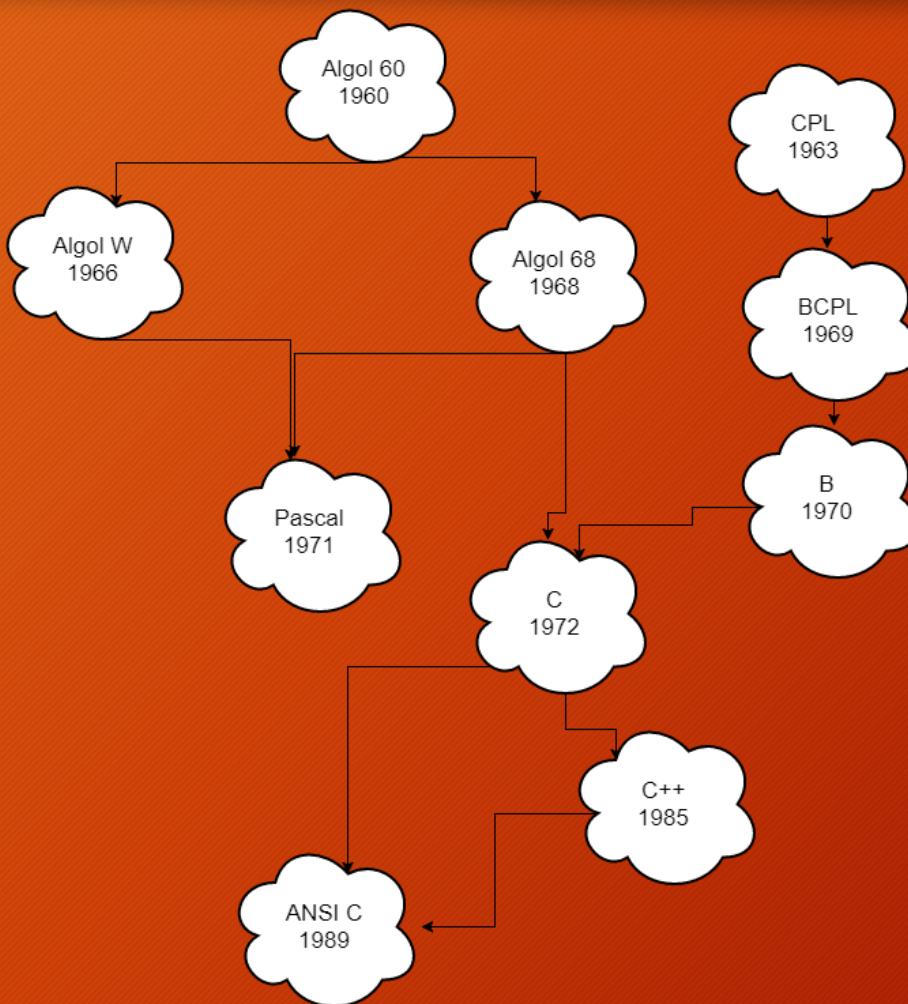
# Simula 67 Soy Ağacı



# ALGOL'un Torunları

- Pascal
  - 1971 yılında tanıtıldı.
  - Basit ve okunabilir bir dil olduğu için ilk önceleri eğitim dili olarak kullanıldı.
  - Fortran ve C diline göre güvenli bir dildir.
- C Programlama Dili
  - 1972 yılında tanıtıldı.
  - Aktif bellek erişimi, taşınabilir özelliği güçlü yanlarıdır.
  - Bir sistem programlama dilidir.
  - Yeterli derecede kontrol ifadelerine sahiptir.
  - Tip kontrolü yetersizdir.

# Pascal ve C Dillerinin Soy Ağacı



# Modula

- Güçlü bir tip ayırmı ve tip kontrolü mekanizmasına sahiptir.
- Dinamik dizi kullanılabilir.
- Daha sonra geliştirilen Modula 2 yazımı daha esnek bir dildir.
- Soyut veri tipini desteklemekte fakat kalıtım olmadığı için nesne yönelimli bir dil değildir.

# Prolog

- 1972 yılında geliştirilmiştir.
- Çok yüksek seviyeli programlama dilinin ilk örneğidir.
- Dil kurallar ve önermelerden oluşmaktadır.
- Yapay zeka alanlarında kullanılmıştır.

# Ada

- ABD savunma bakanlığını bir çalışması sonucu ortaya çıkmıştır.
- Programlama dilleri tarihindeki en geniş tasarım çalışmasıdır.
- Blok yapılı, nesne yönelimli ve eş zamanlılığı destekleyen bir dildir.
- Kalıtım ve çok biçimlilik özellikleri eklenerek Ada 95 sürümü tanıtılmıştır.

# Ada Dili Soy Ağacı



# Smalltalk

- 1970'li yıllarda Alan Kay ve grubu tarafından geliştirildi.
- Ana amacı okunması ve geliştirilmesi kolay bir dil tasarlamaktı.
- Tamamen nesne yönelimli bir programlama dilidir.



# C++

- Emir esaslı ve nesne yönelimli programlama paradigmasının birleşiminden oluşur.
- C++ dili C ve Smalltalk'ın birleşimidir.
- Çoklu sınıf kalıtımı desteklenmektedir.
- Operatör ve fonksiyon overloading desteklenmektedir.
- Şablon sınıf ve metot desteği vardır.
- Aktif bellek yönetimini destekler.
- Java dilinden daha az güvenli bir özelliğe sahiptir.

# Eiffel Dili

- Neredeyse bir C++ dili olarak görülebilir.
- Emir esaslı ve nesne yönelimli programlama paradigmalarını destekler.
- C++'tan daha küçük ve basittir.

# Delphi

- Emir esaslı ve nesne yönelimli programlama paradigması başarılı bir biçimde birleştirilmiştir.
- Pascal'dan esinlenerek geliştirilmiştir.
- Dizi elemanlarının kontrolü, pointer aritmetiği ve tip zorlamada C/C++'tan daha güvenlidir.
- Delphi daha iyi ve kolay yazılım geliştirmek için GUI sağlamaktadır.

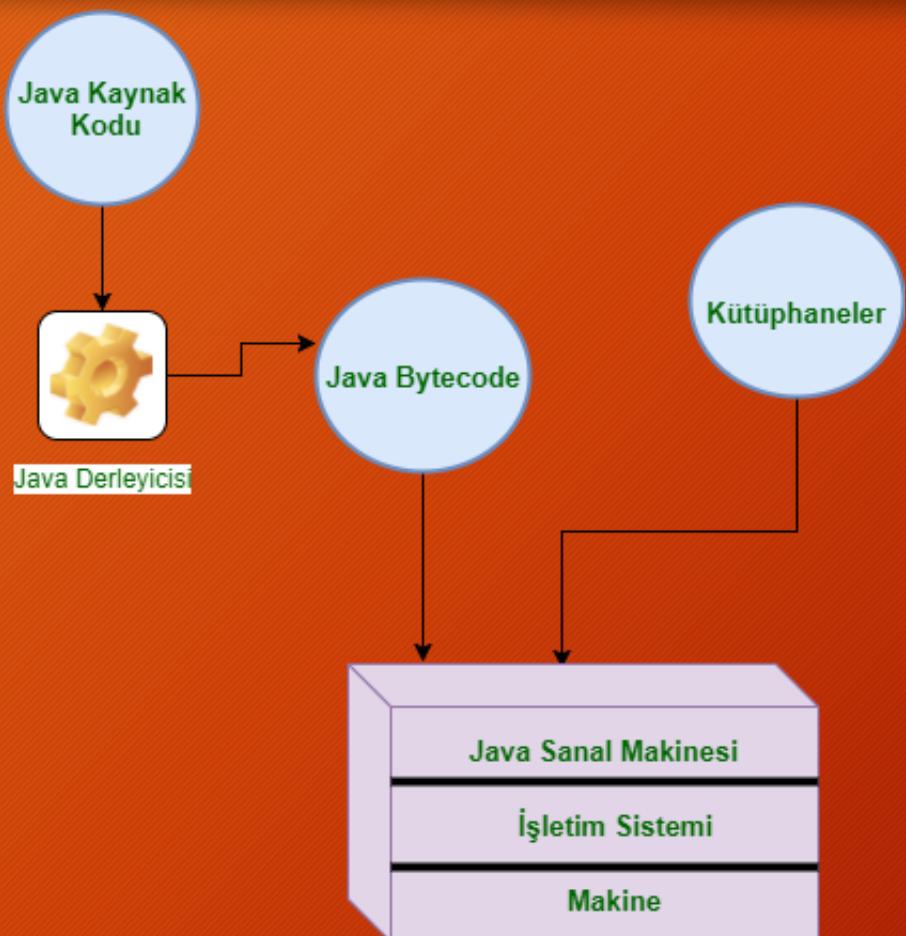
## Delphi'de dizi tanımı ve kullanımı

```
var
    sayilar : array[5..20] of string;
    i : Integer;
begin
    for i := 5 to 20 do
        sayilar[i] := IntToStr(i * 5);
end;
```

# Java

- C++ temel alınarak daha küçük, basit ve daha güvenilir bir programlama dili geliştirilmiştir.
- Tamamen taşınabilir ve nesneye yönelik bir programlama dilidir.
- Java dilinde her şey sınıflardan oluşur.
- Eski Java sürümlerinde şablon yapıları yoktur. Bu ihtiyaç Object sınıfı kullanılarak giderilebilmektedir.
- Java dilinde otomatik çöp toplayıcı mekanizması vardır.

# Java Sanal Makinesi



# Script Dilleri

- Bir dizi komutun bir dosya içerisinde konulması ve dosya çağrıldığında komutların çalıştırılması prensibine dayanır.
- İlk script dili David Korn tarafından geliştirilen ksh olarak gösterilebilir.
- Daha sonra bu dil awk ve Perl gibi dillerin doğmasına ön ayak olacaktır.
- Çalışma şekli olarak iki türlü script söylenebilir.
  - İstemci taraflı
  - Sunucu taraflı

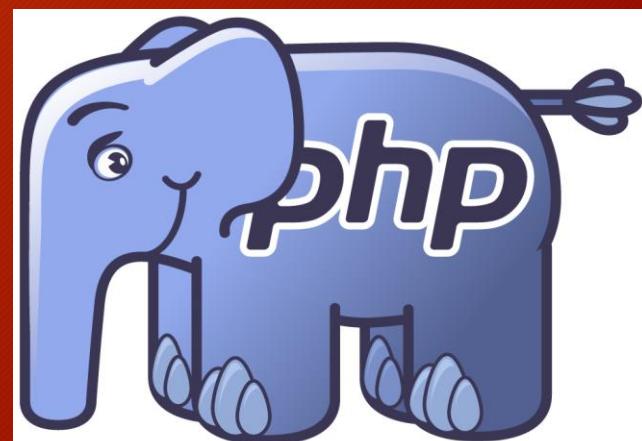
# Javascript

- İstemci taraflı script diline örnektir.
- Netscape ve Sun'ın ortak çalışması sonucu 1995'te tanıtılmıştır.
- Java'nın kuvvetli tip kontrolü bu dilde daha esnektir.



# Php

- Sunucu tabanlı bir script dilidir.
- Rasmus Lerdorf tarafından 1994 yılında geliştirilmiştir.
- Kodlar sunucu tarafında yorumlanır.
- Php dizi yapısı Javascript ve Perl dizi yapılarının bir bileşenidir.



# Python Dili

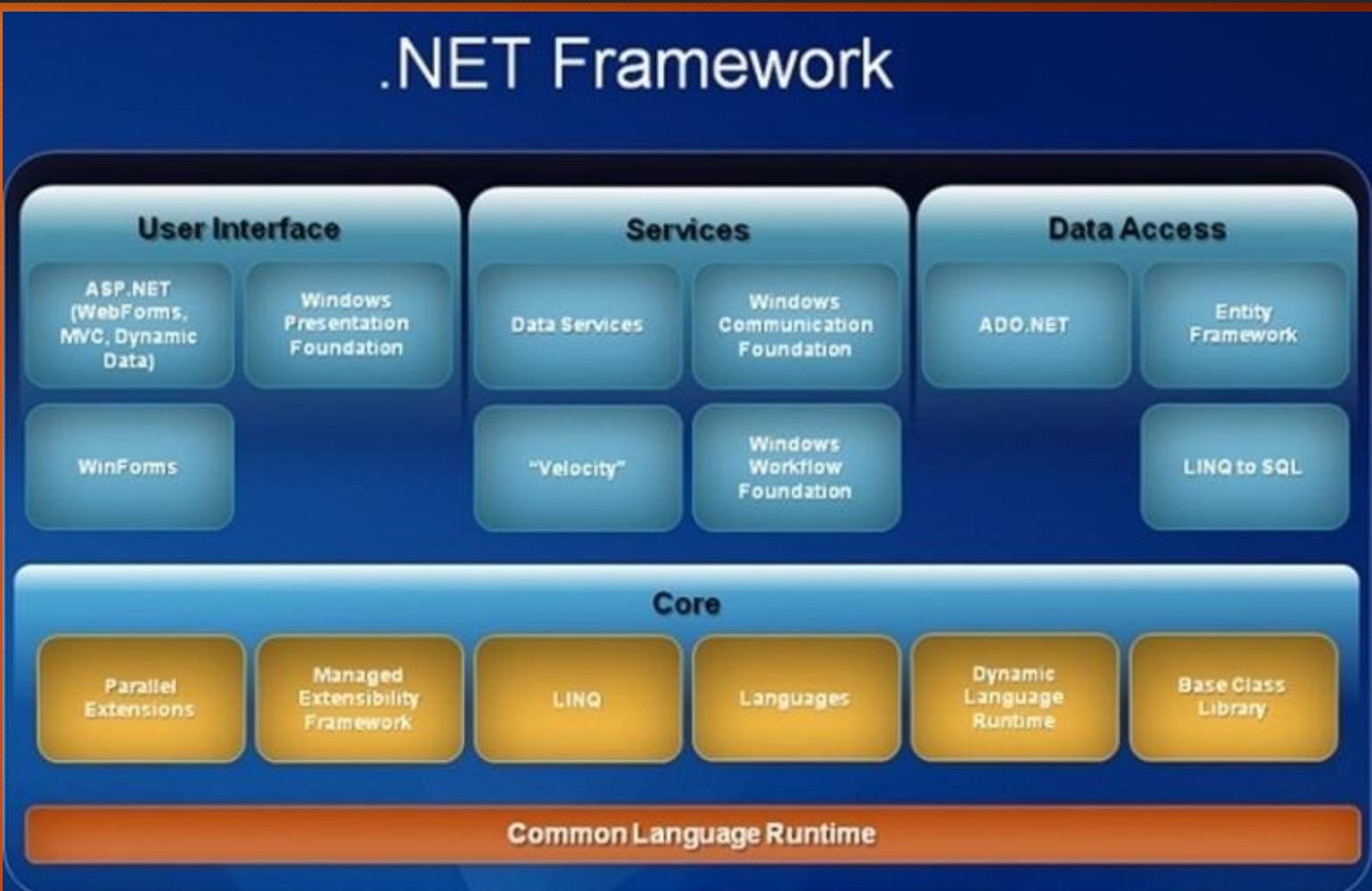
- 1991 yılında Guido van Rossum tarafından geliştirildi.
- Yüksek seviyeli ve genel amaçlı bir programlama dilidir.
- Başlarda Amoeba işletim sisteminde çalışacak şekilde tasarlanan python dili daha sonra her platformda çalışabilecek hale gelmiştir.
- Başarılı bir istisnai durum yönetimi olan python dili yorumlama şeklinde çalışır.
- Bir çok veri yapısı desteği yanında nesne yönelimli bir dildir.
- Devrim yaratan versiyonu 2000 yılında tanıtılan Python 2.0 olarak bilinir.
- Çöp toplayıcıya sahiptir.



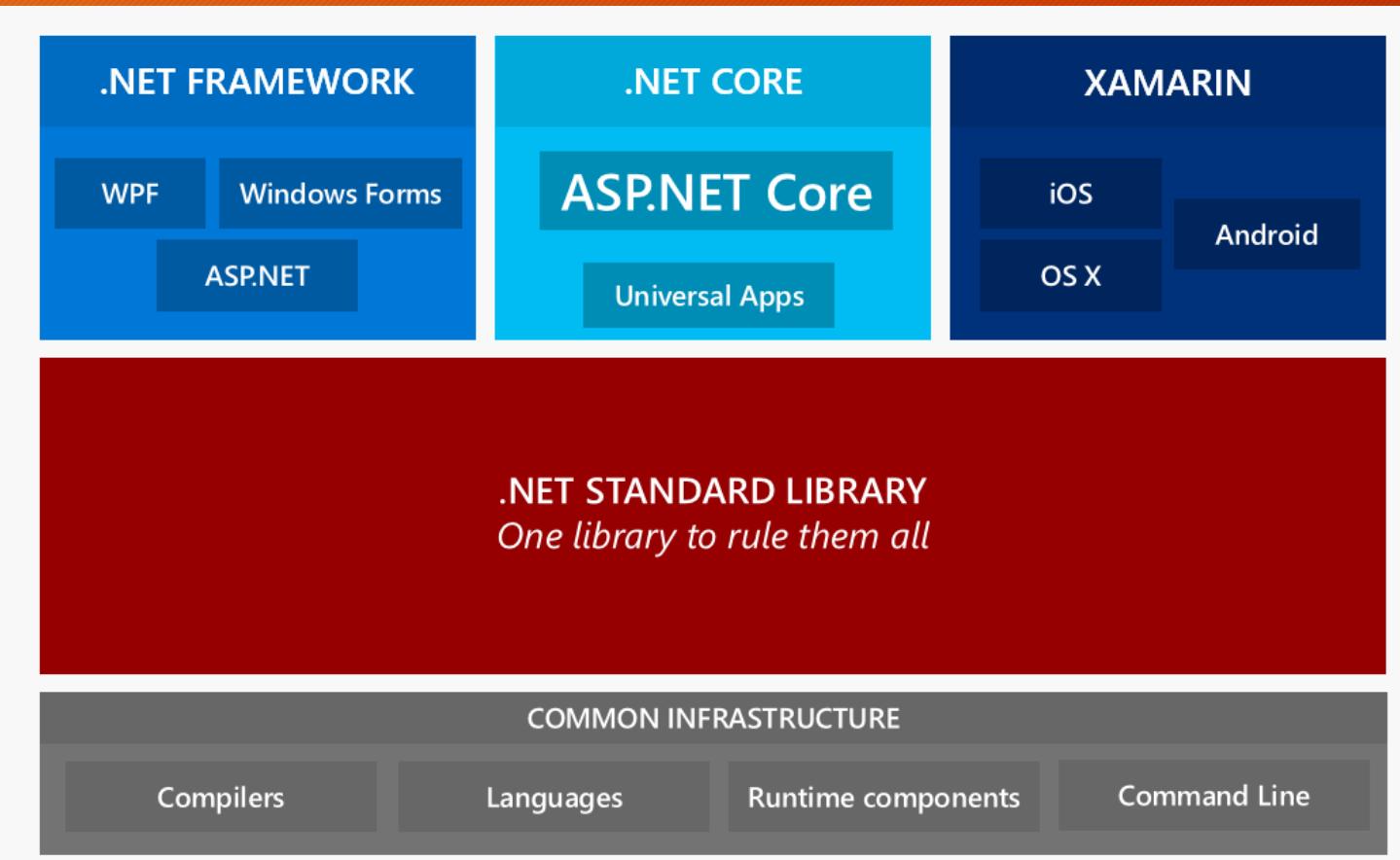
# C# Dili

- C ve C++ dil ailesinin ilk bileşen yönelimli (Component-oriented) dilidir.
- Common Language Runtime (CLR)
  - Bir yürütme motoru
  - Bir çöp toplayıcı
  - Anında derleme
  - Güvenlik sistemi
  - Zengin bir sınıf çerçevesi (.NET Framework)
- CLR birçok dil desteğine kadar hersey için tasarlanmıştır.
- Otomatik bellek yönetimi kullanır.

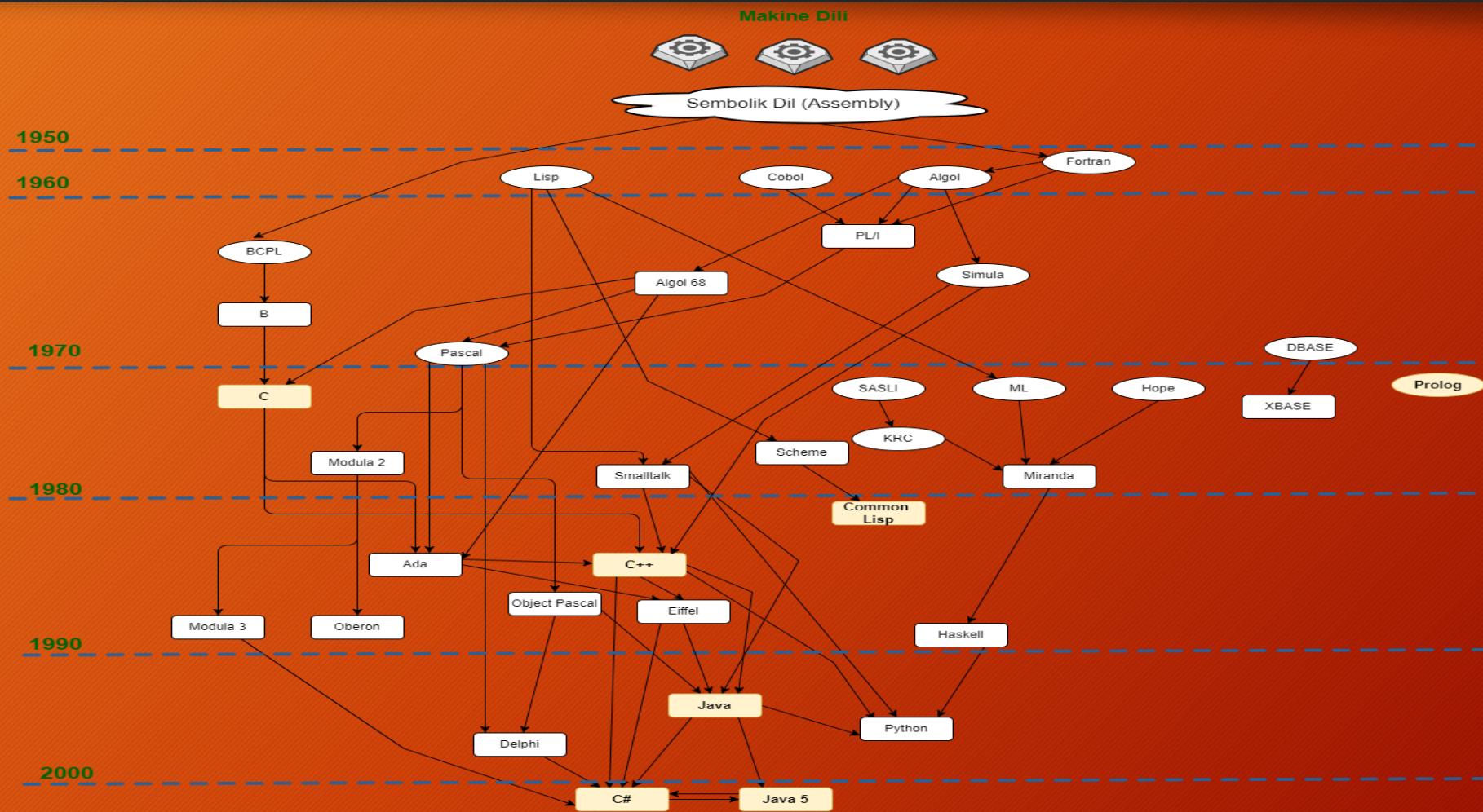
# .NET Framework



# .NET CORE



# Programlama Dillerinin Soy Ağacı



# 2022 Yılı Popüler Programlama Dilleri

Jan 2023	Jan 2022	Change	Programming Language	Ratings	Change
1	1		 Python	16.36%	+2.78%
2	2		 C	16.26%	+3.82%
3	4		 C++	12.91%	+4.62%
4	3		 Java	12.21%	+1.55%
5	5		 C#	5.73%	+0.05%
6	6		 Visual Basic	4.64%	-0.10%
7	7		 JavaScript	2.87%	+0.78%
8	9		 SQL	2.50%	+0.70%
9	8		 Assembly language	1.60%	-0.25%
10	11		 PHP	1.39%	-0.00%

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016

# Programlama Dillerinin Prensipleri

HAFTA 4

TEMEL PROGRAMLAMA KAVRAMLARI  
VERİ TIPLERİ VE YAPILARI

DR. OGR. UYESİ DENİZ BALTA

# Programlama dillerinin temel elemanları

- ▶ Bir veya daha çok bellek hücresinin soyutlaması değişkenler yolu ile sağlanmaktadır.
- ▶ Her değişken; isim, adres, değer , tip, yaşam süresi ve kapsam özelliklerine sahiptir.
- ▶ Bir programlama dilinde aynı isim farklı kısımlarda farklı adreslerle bağlanabilir. Böyle bir durumda her ikisi de farklı değişkenlermiş gibi işlem görecektir ve birbirinden bağımsız olacaktır.
- ▶ Yada bir çok tanımlayıcının aynı adresle ilişkilendirilmesi durumunda **örtüşme (aliasing)** durumu ortaya çıkar. Bu durumda programın okuyucusu farklı isimlerdeki değişkenlerin aynı bellek bölgesindeki değerlerini hatırlamak gereksinimi duyacaktır.
- ▶ Burada belleği daha iyi kullanmak amaçlanmıştır.

# Programlama dillerinin temel elemanları

- ▶ Değişkenin değeri belirli bir adreste belirli bir yönteme göre kodlanarak saklanacaktır ve bu değer çalışma sırasında değişebilir.
- ▶ Bir değişken programda kullanılmadan önce tanımlanmalı, yani bir isim verilmeli ve tip bildirimi yapılmış olmalıdır.
- ▶ Bir değişken tanımlandıktan ve tip bildirimi yapıldıktan sonra bir bellekle ilişkilendirilir.
- ▶ Değişkenin bu bellek birimi ile ilişkili kaldığı süreye değişkenin **yaşam süresi (life time)** denir. Tanımlanan değişken programın hangi deyimlerinde geçerli ise bu alanlara da **değişkenin kapsamı** denir.

# Değişkenlerin isim özelliği

- ▶ İsim değişkenlerin en temel özelliklerinden biridir.
- ▶ İsimler aynı zamanda etiketler, altprogramlar, formal parametreler ve diğer pek çok program yapılarının tanımlanmasında da kullanılmaktadır.
- ▶ Dil tanımlamalarında isimler yerine tanımlayıcı (identifier) terimi de kullanılmaktadır.

# Değişkenlerin isim özelliği

## ► İsimlerin maksimum uzunluğu:

Programlama Dili	İzin verilen Maksimum isim uzunluğu
FORTRAN I	maksimum 6
COBOL	30
FORTRAN 90, ANSI C	31
Ada	limit yoktur, ve hepsi anlamlıdır(significant)
Java	limit yoktur, ve hepsi anlamlıdır(significant)
ANSI C	31
C++	limit yoktur fakat konabilir

# Değişkenlerin isim özelliği

## ► Büyük küçük harf duyarlılığı :

- Çoğu programlama dilinde büyük/küçük harf farkı yoktur. Ama bazı dillerde bu ayrımlı önemlidir.
- Örneğin; C, C++, Java programlama dilleri isimlerde küçük-büyük harf duyarlığını uygulamaktadır.
- Büyük küçük harf duyarlığını uygulayan dillerde ortalamada $2$  değişkeni ile ORTALAMA $2$  değişkeni derleyicide farklı değerlendirecektir.
- Böyle bir dilde aynı isimmiş gibi görünen ama farklı anlamlı olan değişkenler olacağından okunabilirlik olumsuz etkilenecektir.
- Büyük küçük harf duyarlığını uygulamayan dillerde ise ortalamada $2$  değişkeni ile ORTALAMA $2$  değişkeni arasında fark yoktur.

# Değişkenlerin isim özelliği

- ▶ **Özel kelimeler:**
- ▶ Özel kelimeler, bir programlama dilindeki temel yapıların kullandığı kelimelerdir.

Anahtar kelimeler(Keywords):

- ▶ Bir anahtar sözcük(keyword) yalnızca belirli bir bağamlarda(kontekstler)(contexts) özel olan sözcüktür(word).

Örneğin Fortran dilinde:

Real VarName (Real arkasından bir ad(name) gelen bir veri tipidir(data type), bu yüzden Real bir anahtar sözcüktür (keyword))

Real = 3.4 (Real bir değişkendir(variable))

FORTRAN'da REAL TOPLAM yazımında tanımlama deyimi olarak kullanılmıştır.

Oysa REAL = 87.6 kullanımında bir değişken adıdır.

# Değişkenlerin isim özelliği

## Ayrılmış kelimeler(Reserved words)

- ▶ Programlama dillerinde bazı kelimeler tanımlayıcı olarak kullanılamaz.
- ▶ Örneğin C++ dilindeki do, for , while gibi ve PASCAL'da procedure, begin, end gibi kelimere ayrılmış kelime denir.

Örnek: C dili için isimlendirme kuralları:

İsimler, İngilizce büyük ve küçük harfler, rakamlar ve altçizgi işaretinden oluşabilir

İsmen ilk simgesi bir rakam olamaz.

İsmen en az ilk 31 simgesi anlamlıdır.

İsimlerde büyük-küçük harf ayımı vardır.

C dilinin sözcükleri ( int, main, void, return gibi) isimler olamaz.

Kitaplıklardan alınan fonksiyon isimleri ayrılmış kelime değildir.

Değişken ve fonksiyon isimleri küçük harflerle başlar.

Anlamlı isimler verilmek istendiğinde birden fazla sözcüğe gereksinim duyulursa ismi oluşturan iki sözcük bir altçizgi işaretiyile birleştirilir.

# Değişkenlerin tipi

- ▶ Bir değişkenin hangi aralıklarda değer alabileceği ve bu değişken üzerinde hangi işlemlerin yapılabileceği o değişkenin tipi ile temsil edilir.  
Örneğin tamsayı (integer) tipi, dile bağımlı olarak belirlenen en küçük ve en büyük değerler arasında tamsayılar içerebilir ve sayısal işlemlerde yer alabilir.

Veri tipleri temel veri tipleri ve türemiş tipler olarak incelenebilir.

- ▶ Temel veri tiplerini çoğu programlama dili tanımlamıştır ve bu tipler başka tiplerden oluşan basit yapıdadır. Tamsayı, mantıksal, karakter, karakter katarı ve kullanıcı tanımlı karakter katarı veri tiplerini temel veri tipleri gurubunda gösterebiliriz.
- ▶ Türemiş tipler, çeşitli veri tiplerinde olabilen bileşenlere sahiptir. Yapısal tipin elemanları, tipin bileşenlerini oluşturur ve her bileşenin, tip ve değer özellikleri bulunmaktadır. (Diziler, Record (kayıt) ve (gösterge) pointer örnek olarak verilebilir).

# Değişkenlerin tipi

Tablo Temel C++ değişken tipleri

Keyword	Numerik aralık		Ondalık kısım	Bellek alanı
	Alt sınır	Üst sınır		
char	-128	127	yok	1
short	-32,768	32,767	yok	2
int	-2,147,483,648	2,147,483,647	yok	4
long	-2,147,483,648	2,147,483,647	yok	4
float	$3.4 \times 10^{-38}$	$3.4 \times 10^{38}$	7	4
double	$1.7 \times 10^{-308}$	$1.7 \times 10^{308}$	15	8
long double	$3.4 \times 10^{-4932}$	$1.1 \times 10^{4932}$	19	10

# Sabitler

- ▶ Belirli bir tipteki bir değerin kodlanmış gösterimini içeren ancak programın çalıştırılması sırasında değiştirilemeyen bellek hücrelerine sabit denir.
- ▶ Eğer ilgili bellek hücresiyle ilişkilendirildiğinde bir değerle de ilişkilendiriliyorsa bu değişkene isimlendirilmiş sabit denir.
- ▶ Eğer sabit bir değer programda birçok kez tekrar ediliyorsa programın okunabilirliğini ve yazılabilirliğini artırmak için isimlendirilmiş sabit kullanılır (3.14159 değeri yerine pi ismi kullanılması).
- ▶ Ayrıca n elemanlı bir dizide, özellikle döngüsel yapıarda, birçok kez dizi sınırlına başvuru yapılır.

# Programlama dilleri işlemcileri

## GENEL ÖZELLİKLER

İşlenen sayısı:

- ▶ Bir işlemci sonuç üretmek için kaç operand gerektiriyorsa bu isimle sınıflandırılabilir.
- ▶ Bir işlemci, alabileceği işlenen sayısına göre tekli (unary), ikili(binary) ve üçlü (ternary) olabilir.
- ▶ "-" işlemcisi ve C'de bir değişkenin adresini gösteren "&" işlemcisi tekli işlemcilerdir. Bu işlemci için tek operand yeterlidir. "+" ise ikili işlemcilere bir örnektir.
- ▶ Koşullu yapıları üçlü işlemcilere örnek gösterebiliriz.

# Programlama dilleri işlemcileri Genel özelliklerine göre

## İşlemcinin yeri

- ▶ Çoğu işlemci işlenenlerin arasında yazılmakla birlikte, bazı işlemciler, işlenenlerinden önce veya sonra da yazılabılır.
- ▶ İşlemciler bir ifadede, işlenenlerden önce(prefix), işlenenler arasında (infix) ve işlenenlerden sonra (postfix) olmak üzere üç şekilde yer alabilirler.

# Programlama dilleri işlemcileri Genel özelliklerine göre

- ▶ İşlemcilerin öncelikleri, birden çok işlemcinin aynı anda yer aldığı ifadelerde eğer parantez kullanılmamışsa ifadenin hangi sırada işleneceğini belirler.
- ▶ Çoğu emir esaslı programlama dilinde ikili işlemciler infix formundadır. Fakat burada " $a+b*c$ " yada " $a+b+c$ " gibi bir ifadenin değerlendirilmesinde sorun oluşabileceğinden öncelik ve birleşme (associativity) kavramlarının açıklanması gerekmektedir.

## Öncelik

Programlama dili tasarılanırken her işlemci için önceden belirlenmiş bir öncelik düzeyi vardır.

Yüksek düzeyde önceliğe sahip bir işlemci, işlenenlerini daha düşük bir düzeydeki bir işlemciden önce alır.

# Programlama dilleri işlemcileri Genel özelliklerine göre

## Birleşme

- ▶ “ $a+b+c$ ” yada  $a^b^c$  gibi bir ifadenin değerlendirilmesinde olacak sorunlar için birleşme (associativity) özelliğine ilişkin kuralların belirlenmesi gereklidir.
- ▶ Eğer bir ifadede aynı öncelik düzeyinde iki yada daha fazla işlemci bulunuyorsa, hangi işlemcinin önce değerlendirileceği dilin birleşme (associativity) kuralları ile belirlenir.
- ▶ Programlama dili birleşme özelliği yönüyle, işlemcisini sağ veya sol birleşmeli (associative) olarak tanımlayabilir.
- ▶ Bir işlemcinin birden çok kez yer aldığı bir ifade, soldan sağa olarak grпланırsa, işlemci sol birleşmeli olarak adlandırılır. Eğer sağdan sola grпланırlarak değerlendirme yapılyorsa sağ birleşmeli olarak adlandırılır.

# Programlama dilleri işlemcileri Niteliklerine göre

## Sayısal İşlemciler

sembol	İşlev	formül	sonuç
*	Çarpma	$4*2$	8
/	Bölme ve tamsayı bölme	$64/4$	16
%	Modul veya kalan	$13\%6$	1
+	Toplama	$12+9$	21
-	Çıkarma	$80-15$	65

# Programlama dilleri işlemcileri Niteliklerine göre

## İlişkisel İşlemciler

Anlamı	C++	PASCAL	FORTRAN 77	ADA	C
Büyüktür	>	>	.GT.	>	>
Küçüktür	<	<	.LT.	<	<
Eşittir	==	=	.EQ.	=	==
Eşit değildir	!=	<>	.NE.	/=	!=
Büyük veya eşittir	>=	>=	.GE.	>=	>=
Küçüktür veya eşittir	<=	<=	.LE.	<=	<=

# Programlama dilleri işlemcileri Niteliklerine göre

## Mantıksal İşlemciler

- ▶ Eğer birden fazla koşul sınamacaksa bunların birleştirilip tek bir koşul durumuna getirilmesi gereklidir. Böyle durumlarda birden çok koşulun birleştirilmesi için mantıksal operatörler kullanılır.
- ▶ Mantıksal işlemciler, sadece mantıksal (Boolean) işlenenleri alarak mantıksal değerler ( 0:Yanlış yada 1:Doğru) oluştururlar.
- ▶ Mantıksal işlemciler, genellikle AND (ve), OR(veya), NOT (değil) ve XOR(özel veya) gibi işlemleri içerirler.
- ▶ Sayısal işlemciler gibi mantıksal işlemciler de hiyerarşik öncelik sırasında değerlendirilirler.
- ▶ Mantıksal işlemcilerde öncelik sırası, çoğu dilde NOT, AND ve OR biçimindedir.

# İşlemci yükleme

- ▶ İşlenenlerin sayısına ve tipine bağlı olarak İşlemcilerin anamları değişimdir. Buna **İşlemci yüklemesi (operator overloading)** adı verilir.
- ▶ Örneğin "+" işlemcisi tamsayı ve kayan-noktalı sayıarda toplama anlamıyla kullanılırken, bazı dillerde, karakter katarlarının birleştirilmesi için kullanılır.
- ▶ İşlemci “-“ de hem çıkarma işlemi için hem de bir sayının negatif olup olmadığını belirlemek için kullanılabilir.

# Atama deyimi

- ▶ Emir esaslı programlama dillerinde en temel sıralı işlem deyimi atama (assign) deyimidir.
- ▶ Atama deyimi ile sağ tarafın içeriğinin sol tarafa aktarılması amaçlanır.

<hedef\_değişken> <atama\_islemcisi> <ifade>

- ▶ Atama işaretinin sağ ve sol tarafındaki değişkenlerin tip uyumlu olması incelenmesi gereken bir problemdir (tip uyuşmazlığı, tip dönüşümleri, zorunlu tip dönüşümü).

# Atama deyimi

**Çoklu hedefli atama:**

PL/I:

Sum, total=0 deyimi ile hem sum değişkenine hem de total değişkenine 0 değeri atanacaktır.

C:

Sum=total=0 deyimiyle önce total değişkenine 0 değeri atanmakta ve daha sonra total değişkeninin değeri sum değişkenine atanmaktadır.

# Atama deyimi

## Koşullu Hedefler:

Bu ifadelerde atama bir koşula göre yapılır. C++ ve Java koşullu hedeflere izin vermektedir.

F ? count1:count2=0

ifadesinde F'de belirtilen koşul geçerli ise count1=0 aksi halde count2=0 olacaktır.

# Atama deyimi

## Bileşik atama:

- ▶ Bir atama işlemcisi ile bir ikili işlemci birleştirilerek bileşik atama işlemcilerini oluştururlar.
- ▶ C ve C++'da, çeşitli ikili işlemciler için bileşik atama işlemcileri vardır.
- ▶ C'de; "+=", "-=", "\*=", "/=", "%=" bileşik işlemcileri tanımlıdır.

Operator	örnek	Eşdeğer ifadesi
<code>+=</code>	<code>puan+=500;</code>	<code>puan = puan +500;</code>
<code>-=</code>	<code>c-=50;</code>	<code>c=c-50;</code>
<code>*=</code>	<code>maas*=1.2;</code>	<code>maas=maas*1.2;</code>
<code>/=</code>	<code>factor/=50;</code>	<code>factor=factor/.50;</code>
<code>%=</code>	<code>d%=7</code>	<code>d=d%7;</code>

# Atama deyimi

## Tekli atama işlemcileri

- ▶ Bu işlemciler yalnızca tek değişkenlere uygulanırlar.  
Örneğin “+”,  $i = +1$ ; deyiminde ve  $j = -i$  deyiminde tek operatör olarak kullanılmıştır.  
“+5” ve “-5” deyiminde Tek bir + sayının pozitif yada negatif olduğunu göstermektedir.
- ▶ Arttırma (++) ve azaltma (- -) operatörleri de tek bir değişken üzerinde işlem yaparlar ve tamsayı değişkenlerde kullanıldığında bir azaltma ve bir arttırma işlemini gerçekleştirirler.
- ▶ Eğer bu operatörler karakter değişkenleri ile kullanılıyorsa; örneğin C dilinde `k='A'` ise ve `k++` deyimi çalıştırılırsa `k='B'` olur.

# VERİ TİPLERİ VE YAPILARI

- ▶ Bir verinin bellekte nasıl tutulacağını, değerinin nasıl yorumlanacağını ve veri üzerinde hangi işlemlerin yapılabileceğini belirleyen bilgiye **veri tipi** denir.
- ▶ Veri tipi kavramının programlama dillerinin gelişiminde çok önemli bir yeri vardır.
- ▶ Programlama dillerinde veri tipleri **basit-temel (primitive)** ve **türetilmiş (user defined types)** veri tipi olarak iki gurupta incelenir.
- ▶ Bunların arasındaki temel fark, temel veri tiplerinin başka veri tiplerinde oluşmamasıdır.

# İlkel veri tipleri

- ▶ Bu veri tipleri, dilin tasarıminda kararlaştırılmış olup ve dilin kurallarına göre varlığı kesin olan türlerdir.
- ▶ Farklı programlama dillerindeki önceden tanımlanan veri türleri birbirlerinden farklı olabilir.

Örneğin C dilinde de önceden tanımlanmış 11 adet veri türü vardır.

Bu veri türlerinden 8 tanesi tamsayı türünden (Tamsayı veri türleri-integer types) verileri tutmak için, 3 tanesi gerçek sayı (floating types) türünden verileri tutmak için tasarlanmıştır.

# İlkel veri tipleri

- ▶ C, C++ ve C++ Builder derleyicilerinde kullanılan veri tipleri arasında bazı farklılıklar olmakla birlikte, kullanılan temel veri tipleri aşağıdaki tabloda verilmiştir.

Tip Adı	Uzunluk	Sınırlar	
		Alt Sınır	Üst Sınır
Enum	16 bit	-32,768	32,767
unsigned int	16 bit	0	65,535
short int	16 bit	-32,768	32,767
int	16 bit	-32,768	32,767
unsigned long	32 bit	0	4,294,967,295
Long	32 bit	-2,147,483,648	2,147,483,647
Float	32 bit	$3.4 \times 10^{-38}$	$3.4 \times 10^{38}$
Double	64 bit	$1.7 \times 10^{-308}$	$1.7 \times 10^{308}$
long double	80 bit	$3.4 \times 10^{-4932}$	$1.1 \times 10^{4932}$
unsigned char	8 bits	0	255
Char	8 bits	-128	127

# İlkel veri tipleri

## ► Sayısal (Numeric) Tipler

Temel sayısal veri tiplerini; tamsayı veri tipi, kayan noktalı veri tipi ve onlu veri tipi olarak inceleyebiliriz.

## ► Integer (Tamsayı)

**Tamsayı** (*integer*) veri tipinde bir tamsayı değer, bellekte en sol bit işaret biti olmak üzere bir dizi ikili (*bit*) ile gösterilir.

C dilinin toplam 4 ayrı tamsayı veri türü ve bunların da her birinin kendi içinde işaretli ve işaretsiz biçimini olmak üzere toplam 8 tamsayı türü vardır.

İşaretli (*signed*) tamsayı türlerinde pozitif ve negatif tam sayı değerleri tutulabilirken, işaretsiz (*unsigned*) veri türlerinde negatif tamsayı değerleri tutulamaz.

# İlkel veri tipleri

## ► Floating point (Kayan Noktalı)

Reel sayıları modellemek için **Kayan noktalı** (*floating point*) veri tipleri tanımlanır.

Kayan noktalı sayılar, kesirler ve üsler olarak iki bölümde ifade edilirler.

Kayan noktalı sayıların tanımlanmasında duyarlılık (*precision*) ve alan (*range*) terimleri kullanılır.

Burada duyarlılık, değerin kesir bölümünün tamlığını; alan ise, kesirlerin ve üslerin birleşmesini ifade eder.

# İlkel veri tipleri

## ► Decimal (Onlu)

Ticari işletmelerde ele alınan problemleri desteklemek için bilgisayarlar decimal veri tipini tanımlamışlardır.

Bu veri tipinde onlu değerler tam olarak saklanabilmekte, üsler bulunmadığı için sınırlı bir değer aralığını göstermektedir.

PL/I dilinde tanımlanmıştır. Desimal tipler bilgisayarda BCD (binary coded decimal) Her basamak için bir byte (sekiz bit) kullanıldığından bellek kullanımı bakımından etkin bir veri tipi değildir.

Javada desteklenmez fakat c# dilinde kullanılır. C# dilinde decimal türünde ondalık kismı double a göre çok daha hassastır.

# İlkel veri tipleri

## ► Karakter (Character) Tipi

Hesaplamalarda geçerli olmayan, sadece karakterlik bilgi saklayabilen ve bilgisayarda genel olarak ASCII kodlaması ile saklanan bir veri tipidir.

Bu veri tipi değerlerini (128 tane) ASCII tablosundan alır. Bunlardan 8 tanesi kontrol karakteri, 31 tanesi noktalama işaretleri, 26 tanesi yazılamayan karakterler, diğerleri ise (A\_Z, a\_z) harfler ve (0\_9) rakamlardan oluşmaktadır.

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

C++'da karakter veri tipi de *char* anahtar kelimesi ile tanımlanır. C++ derleyicisi 'a' gibi bir karakter sabiti ile karşılaşlığında bunu ASCII koduna çevirir.

Javada karakterler Unicode olarak saklanır ve bellekte 2 byte yer kaplar.

# İlkel veri tipleri

## ► Karakter Katarı

Bir karakter katarı (*character string*) veri tipi karakter dizisi olarak sunulur.

Temel veri tipi olarak tanımlanmamış dillerde (Pascal, C, C++ ve Ada) ise tek karakterli bir karakter dizisi olarak saklanmaktadır.

ADA dilinde STRING tipi vardır ve CHARACTER elemanlarından oluşan tek boyutlu bir dizidir.

ADA dilindeki NAME1:= NAME1 & NAME1 deyimi bu iki katarın bitiştilmesi anlamına gelmektedir. Eğer NAME1 katarı "Sakarya" ve NAME2 katarı "Üniversitesi" ise yukarıdaki deyim ADA dili için SakaryaÜniversitesi sonucunu verecektir.

# İlkel veri tipleri

## ► Karakter Katarı

JAVA programlama dilinde karakter katarları temel veri tipi olarak desteklenmektedir. String class ile desteklenen tiplerin değerleri sabittir ve StringBuffer class ile desteklenen değerler değiştirilebilir.

C dilinde String türü bulunmamaktadır ve yaklaşım tarzı karakter dizisi şeklindedir.

Diziler C'de ilk değerin adresini gösteren bir gösterici olarak tutulmaktadır.

# Türetilmiş veri tipleri

- ▶ Programlama dillerinin çoğu, önceden tanımlanmış veri türlerine ek olarak, programcının da yeni türler tanımlanmasına izin vermektedir.
- ▶ Programcının tanımlayacağı bir nesne için önceden tanımlanmış veri türleri yetersiz kalıyorsa, programcı kendi veri türünü yaratabilir.
- ▶ C dilinde de programcı yeni bir veri türünü derleyiciye tanıtabilir ve tanıttığı veri türünden nesneler tanımlayabilir.
- ▶ Türetilmiş (*structured*) tipler ilkel tiplerden oluşur ve bellekte bir dizi yerlesimde saklanırlar.
- ▶ Diziler, kayıtlar ve göstergeler türetilmiş veri tiplerini oluşturmaktadır.

# Türetilmiş veri tipleri

## ► Diziler

Dizilerde bir elemanın yeri ilk elemana göre belirlenebilir.

Dizilerin veri elemanları bir temel veri tipi, ya da daha önceden tanımlanmış bir veri tipidir.

Dizilerin en önemli özelliği, dizideki bir elemana, tanımlayıcı kullanmadan, elemanın dizideki konumunu belirten bir indis aracılığıyla ulaşılabilmesidir.

# Dizilerde adres polinomu

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

$$\text{Adres}(A[k]) = \text{Adres}(A[0]) + k * c$$

şeklinde tanımlanır.

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

$$\text{Adres}([a(i,j)]) = \text{adres}[0,0] + ((i-0)*n) + (j-0)*c$$

şeklinde tanımlanır.

# Record, Union, Küme türü

## ► Record (Kayıt) Tipi

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 ulaşılmaktadır

Kayıt içerisinde tanımlanan her alanın birbirinden farklı isimleri vardır. İstenirse bu isimler bu tipin dışında farklı tanımlamalar için de kullanılabilir.

# Record, Union, Küme türü

## ► Union (Ortaklık) 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.

Struct veri tipinde her değişken için bellekte ayrı ayrı yer ayrıılır.

Struct yapısında tüm alanlara farklı bilgilerin girilmesi mümkündür.

Ortaklık (union) yapısında ise ilgili bellekte tek bilgi tutulur.

# Record, Union, Küme türü

## ► Set (Küme) Tipi

Aynı tipte ve birbirleri ile ilgili bilgilerin oluşturduğu bir guruptur.

Küme veri tipi emir esaslı (*imperative*) dillerden sadece Pascal'da tanımlıdır.

Diğer programlama dillerinin desteklememesinin nedeni set bellekte Word boyutuna göre yer ayırrır.

► PASCAL dilinde type kısmından faydalananarak *Set of* anahtar kelimesi ile küme veri tipi oluşturulur.

Type

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

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

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

# Bellek yönetimi

- ▶ Bir programdaki değişkenler, sınıflar, metodlar bellekte tutuldukları yer bakımından 3 farklı bölge bulunmaktadır.
- ▶ Buna ek çalışan programın kodlarının bulunduğu ayrı bir kısımda derlenmiş kod olarak vardır.

Statik bellek bölgesi

Çalışma anı yığını

Heap bellek bölgesi

Derlenmiş kod

# İşaretçiler

- ▶ İşaretçiler değişkenlerin kendisini değil adresini gösterir.
- ▶ Bir **İşaretçi** bir adresdir ve **İşaretçi değişken** ise adreslerin saklandığı yerdır.
- ▶ İşaretçi tipindeki değerler, gösterdikleri veri ne olursa olsun sabit bir büyülüktedirler ve genellikle tek bir bellek yerine sıgarlar.
- ▶ Liste, ikili ağaç ve dizi gibi veri yapıları işaretçilerle daha kolay kullanılabilir.
- ▶ İşaretçilerle Bir fonksiyondan başka birine dizi ve stringlerin aktarılması daha güvenli yapılabilir.
- ▶ Ayrıca normalde bir fonksiyondan tek bir değer geri alınabilirken işaretçiler ile birden fazla değer almak mümkündür.
- ▶ Bir işaretçi değişken tanımı aşağıdaki gibi tanımlanabilir.

*int \*pdp;*

Burada *pdp isimli* değişken \* karakteri ile işaretçi değişken olarak tanımlanmıştır. *int* ise değişkenin veri tipini göstermektedir.

# İşaretçiler

- ▶ İşaretçi değişkenler belleğe dolaylı erişim için kullanılabilmektedir.
- ▶ Örneğin « **toplam** » isimli bir işaretçi değişkeninin 1950 değerini içersin ve 1950 olan bellek hücresinde 1250 değeri varsa,
  - ▶ “**toplam**” değişkenine normal başvuru 1950 değerini,
  - ▶ **dolaylı bir başvuru** ise 1250 değerini verecektir.

# İşaretçiler

## VOID GÖSTERİCİLER

- ▶ C dilinde, türü olmayan bir göstericidir.
- ▶ Dolayısıyla yeri geldiğinde bir tamayı gösterebileceği gibi yeri geldiğinde bir ondalık sayıyı da gösterebilir.
- ▶ Sadece göstericinin gösterdiği yer kullanılacağı zaman, derleyicinin o anki hangi tür olduğu bilmesi açısından dönüştürme işlemi uygulanmalıdır.

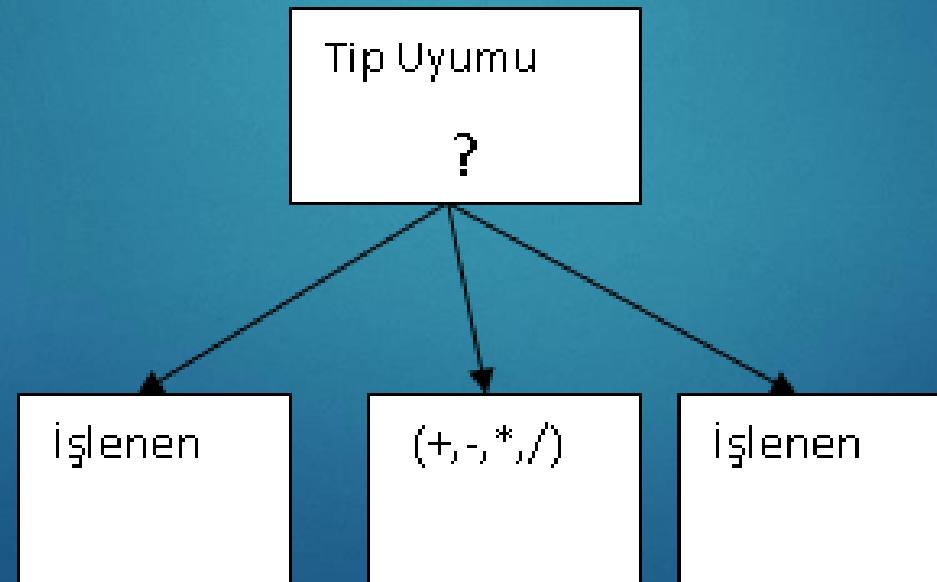
# İşaretçiler

## SALLANAN GÖSTERİCİLER

- ▶ Eğer bir gösterge tipi değişken serbest bırakılmış bir bellek adresini gösteriyorsa yani ilgili adreste geçerli veri bulunmuyorsa bu değişkene **sallanan işaretçi** (*dangling pointer*).
- ▶ Bir işaretçi tipinin tanımlayabileceği veri tipinde bir sınırlama olmaması halinde durağan tip denetimi yapılamadığı için, programların güvenilirliği azalmıştır.
  - ▶ İşaretçi değişkenin tanımlandığı ilk dil olan PL/I'da, gösterge değişkenin gösterebileceği elemanların veri tipi sınırlanmamıştır.
  - ▶ C'de ise bir işaretçi değişken tanımlanırken, adresini tutabileceği değişken tipi de belirtilmelidir. Bir `(int *a)` gösteriminde a, sadece int tipinde bir değişkene işaret eder.
- ▶ Java programlama dili işaretçi değişken kullanımına izin vermeyerek bu gibi sorunları önlemeye çalışmıştır.

# Tip denetimi

- ▶ Bir programlama dilinin güvenilirliğinde en önemli etkenlerden biri tip denetimidir.
- ▶ Bir işlemcinin işlenenleri birbirleriyle uyumlu tipler olmalıdır. Tip denetimi derleme zamanında ve çalışma zamanında yapılır.



# Tip dönüşümleri

- ▶ Herhangi bir işlem birden fazla değişken, sabit ve operatör içerebilir.
- ▶ İşleme giren değişken ve sabit ifadelerin farklı tiplerden olması doğaldır.
- ▶ Bu durumda sonucun hangi tipte olacağını işlem içerisinde değişken ve sabitler belirler.
- ▶ Böyle durumlarda işlem içerisinde hafızada en çok yer kaplayan ifadenin veri tipine göre sonucun tipi belirlenir.
- ▶ Ancak bazı durumlarda bu hatalı sonuçların oluşmasına sebep olur.

# Tip dönüşümleri

- ▶ Farklı tipteki sabit veya değişkenler bir ifade içerisinde toplandığında hepsi aynı tipe dönüştürülür.
- ▶ Dönüşümme işlemi en yüksek ifadenin tipine göre yapılır.
- ▶ İlk olarak bütün **char** ve **short int** değerler **int** veri tipine otomatik olarak yükseltilir.
- ▶ Bu adım tamamlandıktan sonra diğer tüm dönüşürmeler aşağıda verilen tip dönüşümme algoritmasına göre adım adım yapılır:

Eğer işlenen **long double** ise ikinci **long double**'a dönüştürülür

Değilse Eğer işlenen **double** ise ikinci **double**'a dönüştürülür

Değilse Eğer işlenen **float** ise ikinci **float**'a dönüştürülür

Değilse Eğer işlenen **unsigned long** ise ikinci **unsigned long**'a dönüştürülür

Değilse Eğer işlenen **long** ise ikinci **long**'a dönüştürülür

Değilse Eğer işlenen **unsigned int** ise ikinci **unsigned int**'a dönüştürülür.

# Tip dönüşümleri

- ▶ **Karışık tipli ifadelerde** (*mixed mode expression*) bir operatör farklı tiplerde operandlar aldığında eğer farklı operandlar için ayrıca operatör yoksa böyle ifadelerde tip dönüşümlerine ihtiyaç vardır.
- ▶ Bir tip dönüşümünde, bir nesne, kendi tipindeki tüm değerleri içermeyen bir tipe dönüştürülüyorsa bu tip dönüşümüne **daralan dönüşüm** denir. Bu dönüşümde hatalar oluşabileceğinden güvenli değildir. Kayan noktalı tipten tamsayıya dönüşüm, daralan dönüşümdür.
- ▶ Eğer bir değişkenin kendi tipinin tüm değerlerini içeren bir tipe dönüşümü gerçekleştiriyorsa bu dönüşüme genişleyen dönüşüm denir. Daha güvenli bir dönüşümdür. Tamsayı değişkenin kayan noktalı tipe dönüşümü, **genişleyen dönüşümdür**.

# Tip dönüşümleri

- ▶ Derleyici karışık tipli bir işlemciyle karşılaşırsa, düşük tipteki değişken yüksek tipteki değişkenin tipine dönüştürülür.
- ▶ Derleyici tarafından gerçekleştirilen ve zorunlu dönüşüm olarak adlandırılan bu tip dönüşümüne **örtülü (*implicit*) dönüşüm** denir.
- ▶ Bu tip dönüşümler derleyicinin tasarımlı sırasında belirlenir.

# Tip dönüşümleri

## Atamalarda tip dönüşümü

- ▶ Tip dönüşümü, bir tipteki değişkeni başka bir tipteki değişkene atama durumunda meydana gelir.
- ▶ Atama işleminde sağ taraftaki ifade sol taraftaki değişkenin tipine dönüştürülür.

```
#include <stdio.h>
void main()
{
    int x=63;
    float f=27.54;
    char c='A';
    c = x; // x değişkeni c'ye atanıyor
    x = f; // f değişkeni x'e atanıyor
    f = c; // c değişkeni f'ye atanıyor
    f = x; // x değişkeni f'ye atanıyor
    printf(" %d\n", x ); // sonuç : 27
    printf(" %f\n", f ); // sonuç : 27.0000
    printf(" %d\n", c ); // sonuç : 63 }
```

# Programlama Dillerinin Prensipleri

HAFTA 5

BAĞLAMA KAVRAMLARI VE İSİM KAPSAMLARI

DR. ÖĞR. ÜYESİ DENİZ BALTA

# Bağlama (Binding)

- ▶ Bir özellikle (isim, adres, değer vb) bir program elemanı (değişken, altprogram vb.) arasında ilişki kurulmasına **bağlama** (*binding*) denir.



- ▶ Bağlamanın gerçekleştiği zamana **bağlama zamanı** denir.

# Bağlama (Binding)

- ▶ Bu özelliklerin bağlanma zamanına göre ve bağlamanın durağan yada dinamik olmasına göre diller farklılık gösterir.
- ▶ Dilin tasarıımı, gerçekleştirmesi ve derlenmesi zamanında yapılan bağlamalara **statik bağlama** denir.
- ▶ Çalışma zamanında yapılan bağlamalara ise **dinamik bağlama** denir.



# Bağlama zamanı

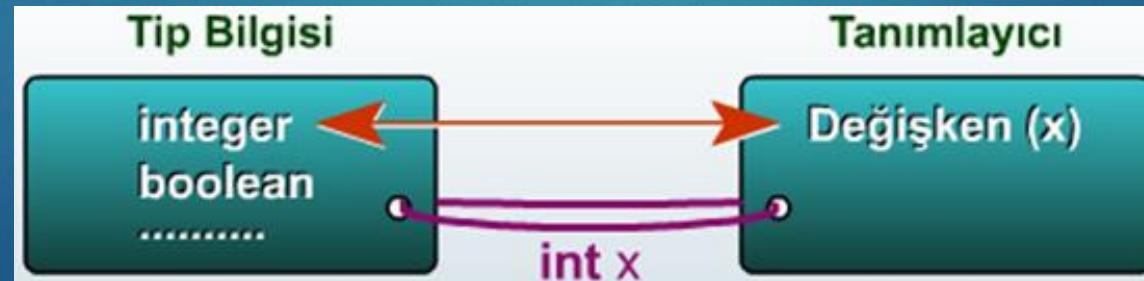
int hesap; ... hesap=hesap+10;

<b>Hesap için olası tipler</b>	Dilin tasarım zamanında
<b>Hesap değişkeninin tipi</b>	Dilin derlenmesi zamanında
<b>Hesap değişkeninin olası değerleri</b>	Derleyici tasarım zamanı
<b>Hesabın değeri</b>	Bu deyimin yürütülmesi zamanında
<b>+ işlemcisinin muhtemel anlamları</b>	Dilin tanımlanması zamanında
<b>+ işlemcisinin bu deyimdeki anlamı</b>	Derlenme süreci
<b>10 literalinin ara gösterimi</b>	Derleyici tasarıtı zamanında
<b>Hesap değişkeninin alacağı son değer</b>	Çalışma zamanında

- ▶ Bir programlama dilinin semantik olarak anlaşılmasında program elemanlarının özelliklerinin bağlama zamanlarının tam olarak anlaşılmaması şarttır.
- ▶ Örneğin bir alt programın ne yaptığının anlaşılmasında çağrıdaki gerçek parametrenin altprogramın tanımındaki formal parametreye nasıl bağlandığının anlaşılmasına şarttır.
- ▶ Bir değişenin o anki değerinin bilinmesi için belleğe ne zaman bağlılığının bilinmesi gereklidir.

# Tip bağlama

- ▶ Bir tanımlayıcı (id) bir tip bilgisi ile ilişkilendirilince o tiple bağlanmış olur.
- ▶ Bir programlama dilinde bir değişken kullanılmadan önce isimlendirilmeli, bir tip ile bağlanmalıdır.
- ▶ Böylece o değişkenin hangi değerleri alabileceği ve üzerinde hangi işlemlerin yapılabileceği belirlenmiş olur.
- ▶ Semantik anlam analizi için bu çok önemlidir.



# Statik tip bağlama

Durağan Tip Bağlama	Dinamik Tip Bağlama
Derleme Zamanında	bir değişkenin tipi çalışma zamanında, değişkenin bağlandığı değer ile belirleniyorsa
bir değişken, <i>integer</i> tipi ile bağlanmışsa	bir değişken, atama sembolünün sağ tarafında bulunan değerin, değişkenin veya ifadenin tipine bağlanır ve değişkenin tipi, çalışma zamanında değişkenin yeni değerler alması ile değiştirilir. A=1.5 A=14  Avantaj: Esneklik (örneğin sıralama )
FORTRAN, Pascal, C ve C++'da bir değişkenin tip bağlaması durağan olarak gerçekleşir ve çalışma süresince değiştirilemez.	APL, LISP, SMALLTALK, SNOBOL4
derleyici, tip hatalarını, program çalıştırılmadan önce yakalar.	Derleyicinin hata yakalama yeteneği zayıftır. Statik tip kontrolü yapılamaz Yorumlayıcı kullanıcılar

# Statik tip bağlama

## Örtülü (*implicit*) Tip Bağlama :

- ▶ Herhangi bir tanımlama deyiminin kullanılmaksızın, bazı varsayılan kurallar ile tip bağlaması yapılıyorsa buna **Örtülü tip bağlama** denir.
- ▶ Örneğin bir değişken isminin programda ilk kullanıldığı deyim ile ilişkili olarak tipini bağlanması örtülü tip bağlamadır.
- ▶ FORTRAN, PL/I, BASIC dilleri örtülü tanımlamalara sahiptir. FORTRAN'da bir değişenin ismi I,J, K, L, M, N harflerinden biri ile başlıyorsa bu değişken örtülü olarak INTEGER tipi ile aksi hallerde REAL tipi ile bağlanır.
- ▶ BASIC dilinde ise son karakteri \$ olan değişkenler char tipi ile bağlanırlar.

# Statik tip bağlama

## Örtülü (*implicit*) Tip Bağlama :

- ▶ Örtülü tanımlamalar, programlama dilinin güvenilirliğini tehlkeye atabilir.
- ▶ Çünkü bu örtülü tip bağlamaları bazı tip hatalarının ve programcı hatalarının derleyici tarafından yakalanmasına engel olabilir.
- ▶ Programcının tanımlamayı unuttuğu bir değişkene örtülü olarak tip atamasının yapılması fark edilemeyen hatalar oluşturabilir.
- ▶ Bu yüzden PL/I, BASIC, Perl ve FORTRAN gibi dillerde örtülü tanımlamalar bulunmasına karşın günümüzde çoğu programlama dili dışsal tanımlama yolunu tercih etmektedir.

# Statik tip bağlama

## Dışsal Tip Bağlama:

- ▶ Bu yöntemde tip bağlaması için **int**, **float** ve **char** gibi bir deyim kullanılır.
- ▶ Bir çok dil güvenlik nedeniyle bu yöntemi tercih eder.

```
public static void main (String [] args)
{
    String x;
    x="Mustafa";
}
```

# Dinamik tip bağlama

- ▶ Bir değişkenin tipi çalışma zamanında ve değişkenin bağlandığı değer ile belirleniyorsa, bu dil **dinamik tip bağlamalı** bir dildir.
- ▶ Burada bir bildirim deyimi kullanılarak tip bağlaması yapılmaz, bunun yerine bir atama deyiminde bu değişkene bir değer atandığı zaman tip bağlaması yapılır.
- ▶ İlgi değişken, atama sembolünün sağ tarafında bulunan değerin, değişkenin veya ifadenin tipine bağlanır. Eğer değişken çalışma zamanında yeni değerler alırsa değişkenin tipi değiştirilir.

# Dinamik tip bağlama

- ▶ Dinamik tip bağlamalı dillerde derleyicinin hata yakalama şansı statik tip bağlamalı dillere göre daha zayıftır.
- ▶ Çünkü bir atama operatörünün sağ ve sol taraflarında herhangi iki tip görünebilir.
- ▶ Bu durumda atama operatörünün sağındaki yanlış tip hata olarak algılanmaz ve sol tarafın tipi böylece yanlış bir tipe dönüşebilir.

# Dinamik tip bağlama

- ▶ Dinamik tip bağlamalı dillerde tip kontrolü çalışma zamanında yapılmak zorundadır.
- ▶ Bir değişkenin değeri için kullanılan bellek değişken boyutta olmalıdır.
- ▶ Çünkü farklı tipler farklı miktarda yer kaplamaktadır.
- ▶ Dolayısıyla dinamik tip bağlamalı dillerde statik tip kontrolü yapmak mümkün olmamaktadır.
- ▶ Dinamik tip bağlamalı dillerin gerçekleşmesinde bu yüzden yorumlayıcı kullanılmaktadır.
- ▶ Çünkü dinamik olarak değişen tipleri makine koduna çevirmek zor olacaktır.

# Dinamik tip bağlama

## Avantajları:

- ▶ Değişkenlere Dinamik olarak tip bağlanması, programlamaya esneklik sağlar.
- ▶ Dinamik tip bağlamalı bir dilde farklı tipteki değerlerin sıralanması mümkündür.
- ▶ Sıralama programındaki değişkenlerin tipleri çalışma zamanında belirlenebiliyorsa dinamik tip bağlamalı dil bir avantajdır.
- ▶ Durağan tip bağlamalı programlama diller sadece tek bir veri tipi için bir sıralama programı yazılabilir. Ayrıca bu veri tipi başlangıçta bilinmemelidir.

# Bellek bağlama

- ▶ Bir değişkenin ulaşılabilir bir bellek hücresi ile ilişkilendirilmesine **bellek yeri ataması** (*memory allocation*) denir.
- ▶ Değişkenin bu bellek hücresini iade etmesi ise **belleğin serbest bırakılması** (*deallocation*) olarak adlandırılır.
- ▶ Bir değişkenin bu bellek hücresi ile ilişkili kaldığı süreye ise (**bellek yeri ataması ile belleğin serbest bırakılması arası**) değişken için yaşam süresi (*lifetime*) denir.

# Bellek bağlama

## Program çalışma zamanı bellek düzeni:

- ▶ Programın derlenmiş hali yani derlenmiş kod parçası belleğin derlenmiş program kısmında saklanır.
- ▶ Program boyunca geçerli değişkenler (global değişkenler) statik (yığıt) bellek bölgesi kısmında tutulur.
- ▶ Yığın (Heap) bellek kısmı dinamik bellek değerleri için kullanılır. Dinamik bellek bölümü, gerektikçe büyüyebilir ve kullanılan bellek hücreleri iade edilebilir.

# Bellek bağlama

## Değişkenlerin bellek yeri bağlaması



**etkinlik (activation) kaydı**

aynı bellek bölümünün yeniden kullanılabilmesi

Doğrudan adresleme

Pascal-*dispose*

Java-otomatik

C'deki malloc fonksiyonu  
C++ 'daki new işlemcisi

# Bellek bağlama

## Değişkenlerin bellek yeri bağlaması

### Statik değişkenler:

- ▶ Statik değişkenler, programın yürütülmesi başlamadan bellek hücrelerine bağlanırlar ve bellek hücreleri ile programın çalışması sonlanıncaya kadar bağlı kalırlar.
- ▶ Derleme zamanında bu değişkenler için bellek ayrılması gerçekleşir.
- ▶ Dolaylı adresleme gerektiren değişken türlerine erişim yavaş iken Doğrudan adreslemeden dolayı Statik değişkenler verimlidir, erişim hızlıdır.
- ▶ Statik değişkenler esneklik kriterine olumsuz etki etmektedir.

# Bellek bağlama

## Değişkenlerin bellek yeri bağlaması

### Yığıt dinamik (stack-dynamic) değişkenler:

- ▶ Yığıt dinamik değişkenlerin bellek yeri bağlamaları kendilerine ilişkin tanımlama deyimleri çalıştığında gerçekleşir.
- ▶ Yığıt dinamik değişkenler için bellek yeri, çalışma zamanında bellekteki yığıt bellekten ayrıılır.
- ▶ Dolayısıyla Yığıt dinamik değişkenler için ne kadar belleğe ihtiyaç olduğu derleme zamanında hesaplanamaz.
- ▶ ALGOL 60 ve bu çizgideki diller yığıt dinamik değişkenleri tanımlamaktadır. FORTRAN77 ve FORTRAN90 yerel olarak yığıt dinamik değişkenlere izin vermektedir. Pascal, C ve C++'da, lokal değişkenler, varsayılan olarak yığıt\_dinamik değişkenlerdir.

# Bellek bağlama

## Değişkenlerin bellek yeri bağlaması

### Dışsal heap dinamik değişkenler:

- ▶ Dışsal yığın dinamik değişkenler için ne kadar bellek gerektiği önceden bilinmez.
- ▶ Dolayısıyla bellek yeri bağlaması çalışma zamanında gerçekleşir.
- ▶ Çalışma zamanında veriler oldukça belleğe atanır ve bellek yeri yığın bellekten alınır ve daha sonra yığın belleğe iade edilir.
- ▶ Bu verilere sadece işaretçi (pointer) değişkenler aracılığıyla ulaşılabilir. Bu değişkenlerin tip bağlaması derleme zamanında gerçekleşir.

# Bellek bağlama

## Değişkenlerin bellek yeri bağlaması

### Örtülü heap dinamik değişkenler:

- ▶ Örtülü değişkenler, sadece kendisine bir değer atandığı zaman belleğe bağlanırlar ve her yeni atamada tip ve bellek özellikleri yeniden belirlenebilir.
- ▶ Örtülü dinamik değişkenler kod yazımına esneklik kazandırmaktadır.
- ▶ Bunun yanında değişen tüm özelliklerin çalışma zamanında izlenmesi zorunluluğundan dolayı bir hız kaybı vardır.
- ▶ Ayrıca esneklikten dolayı derleyicilerin yakalayamayacağı hatalar olabilir.

# İsim kapsamları

- ▶ Programda tanımlanan bir isim için hangi komutların ve deyimlerin bu isme ulaşabileceği ve bu ismin geçerli ve etkin olduğu program alanına **isim kapsamı (name scope)** denir.
- ▶ İsim kapsam, ismin tanımlandığı noktadan başlar ve o programlama dilinin kabul ettiği isim kapsamı kurallarına bağlı olarak sonraki bir noktaya kadar devam eder.
- ▶ İsimler geçerli olduğu kapsam alanı için lokal değişkendir. Bir kapsamda onu saran kapsamalardan alınan isimler lokal olmayan değişkenlerdir.
- ▶ Ana program blokundaki bir isim ise global değişkendir.

# İsim kapsamları

## **Statik Kapsam Bağlama:**

- ▶ Statik kapsam bağlamada değişkenlerin kapsam alanları programın lexical – metinsel düzenine göre yapılır.
- ▶ Bir değişken ismi ile karşılaşıldığında değişkenin tanımı öncelikle bulunduğu blokta aranır.
- ▶ Eğer burada bu değişken bildirilmemişse programın lexical incelemesi ile fiziksel olarak kendisine en yakın blokta değişkene başvuru yapılır.
- ▶ Değişkenin bildirimini yaptığı ilk yerdeki değerlere göre işlemler yapılır. Ve ilk çağrım noktasına dönülür.

# İsim kapsamları

**Statik Kapsam Bağlama:**

**Statik kapsam bağlamanın değerlendirilmesi:**

- ▶ Programlama dillerinde Blok kavramı ile altprogramların ayrıştırılması kolaylaşmakla birlikte statik kapsam bağlamada, iç içe altprogramlarda fazla sayıda genel değişken kullanımına sebep olabilir.
- ▶ Programdaki genel değişkenler, gerekli olmasa bile, tüm altprogramlara görünür olacaklardır. Bu ise programlama dilinin güvenilirlik kriterine zarar verecektir.

# İsim kapsamları

## Dinamik Kapsam Bağlama:

- ▶ Eğer bir ismin kapsamı, altprogramların metinsel düzenine (lexical scope) değil de altprogramların çağrılmış sırasına göre çalışma zamanında belirleniyorsa bu bağlamaya **dinamik kapsam bağlama** olarak adlandırılır.
- ▶ Dinamik kapsam bağlamada bir değişken ismi çalışma zamanında aynı isimli yeni bir değişken bulunana kadar, kendisinden sonra çalıştırılan tüm deyimlerde geçerlidir.

# İsim kapsamları

**Dinamik Kapsam Bağlama:**

**Dinamik Kapsam Bağlamamanın Değerlendirilmesi:**

- ▶ Dinamik kapsam bağlama kurallarının uygulanması kolaydır.
- ▶ Fakat procedure'de bir değişkene yapılan başvuru, deyimin her çalışmasında farklı değişkenleri gösterebilir.
- ▶ Ayrıca bir değişkenin aldığı değer çalışma zamanında değişimekte ve kullanıcı bunu programı okuyarak kolayca izleyememektedir.
- ▶ Bu ise programın anlaşılabilirliğini azaltmakta ve dillerin okunabilirlik ölçütünü olumsuz etkilemektedir.

# Bloklar

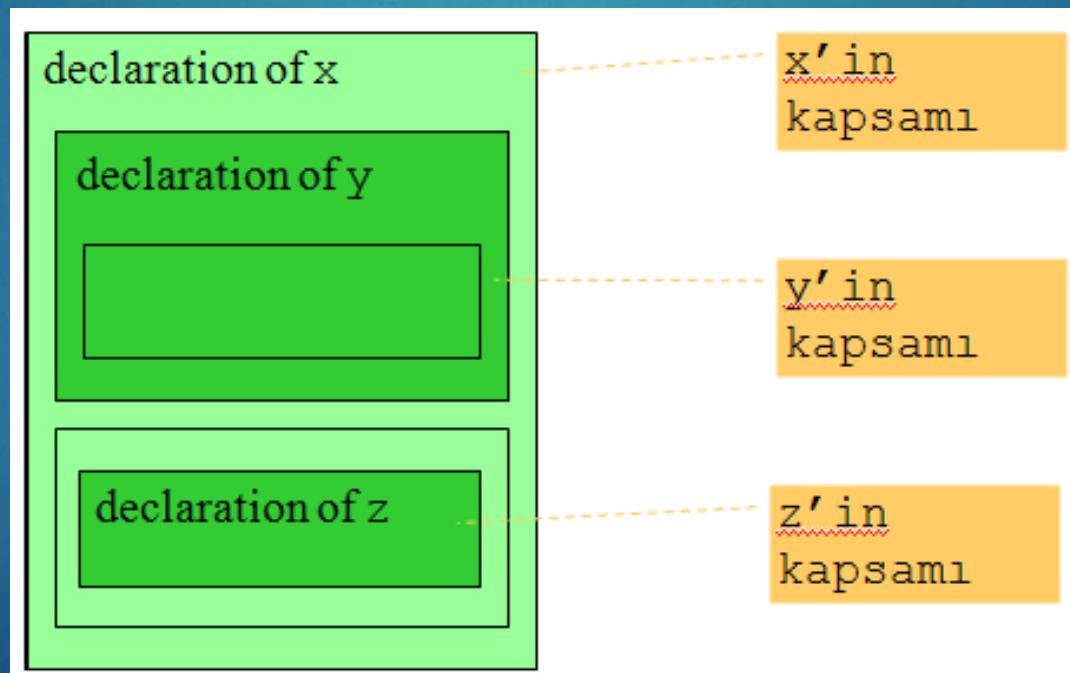
- ▶ Blok, herhangi bir bildirimin, bağlamanın kapsamının sınırlandırıldığı program bölgesidir.
- ▶ Bir program blokunda deyimler bir araya getirilir ve bu deyimlere özgü yerel değişkenler tanımlanır. Bir blok içerisinde tanımlanan değişkenlere bu bloğun **yerel değişkenleri** denir.
- ▶ O blok içinde görünebilen fakat orada bildirilmemiş değişkenlere ise o blok için **yerel olmayan değişkenler** denir.
- ▶ Her programlama dilinin kendine has blok yapısı vardır.
- ▶ Bir programlama dilinde altprogramlar içi içe yuvalanabiliyorsa bu dil **blok yapılı** bir dildir.

# Bloklar

- ▶ Pascal, altprogramların yuvalanmasına izin verdiği için blok yapılı bir dil olarak nitelenmesine karşın, altprogram olmayan bloklar, Pascal programlarında yer alamaz.
- ▶ C'de altprogramlar yuvalanamaz ve isimsiz bloklar bulunabilir.
- ▶ C ve C++'da "**{... }**" arasındaki birleşik (**compound**) deyimler bir bloktur ve blok içerisinde yeni kapsamlar tanımlanabilir.
- ▶ Bloklarda tanımlanmış değişkenler, yığıt dinamik değişkenlerdir ve bu blok çağrııldığı zaman bellek kullanırlar.

# Bloklar

- ▶ Modern programlama dilleri içiçe blok yapılarına izin vermektedir.
- ▶ Blok yapısı programlama dillerinde okunabilirliği ve ifade gücünü yükselten bir tekniktir.



# Programlama Dillerinin Prensipleri

HAFTA 6

YAPISAL PROGRAMLAMA

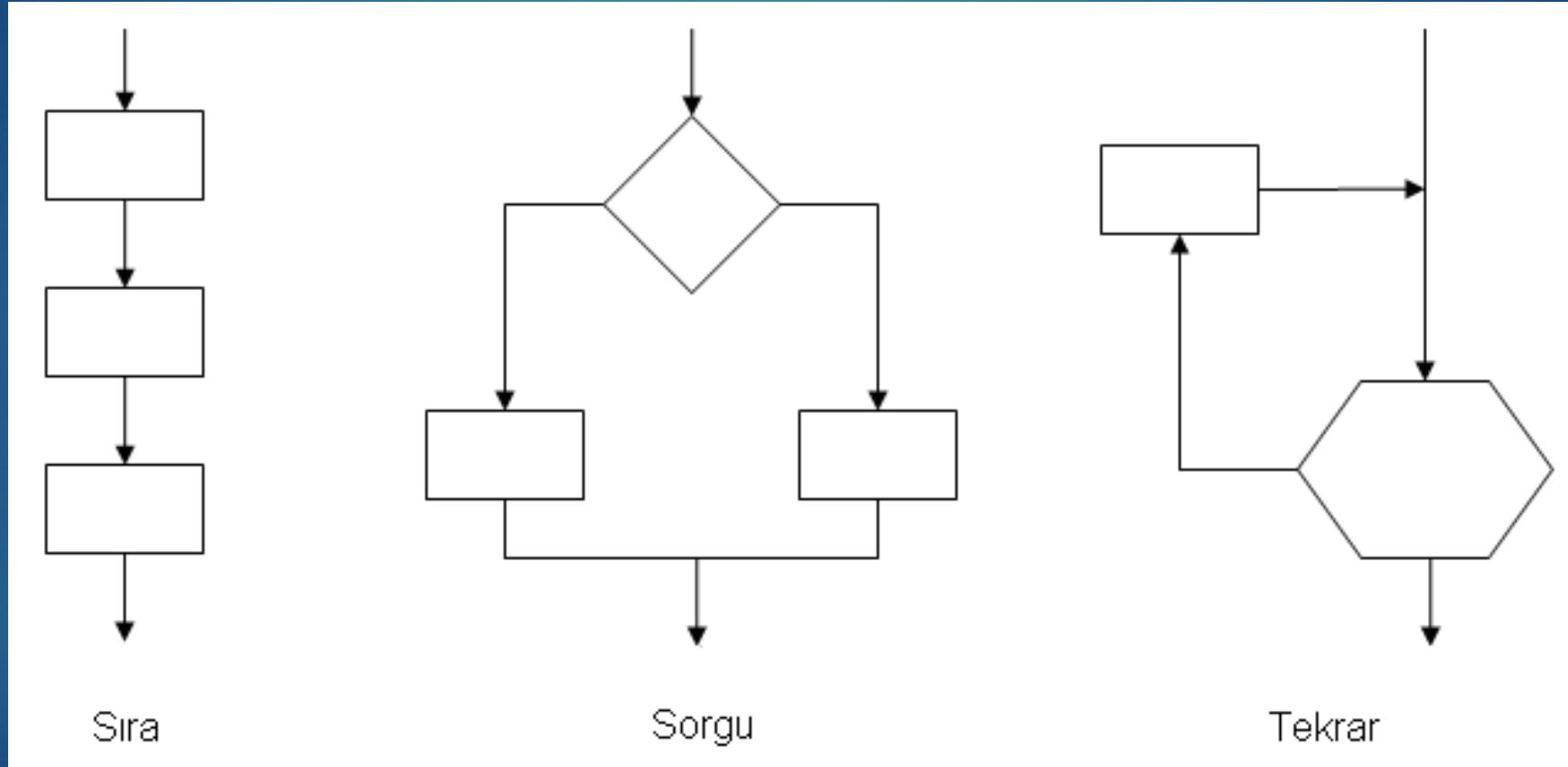
ALT PROGRAMLAR VE MODÜLASYON

DR. ÖĞR. ÜYESİ DENİZ BALTA

# Yapısal Programlama

- ▶ Yapısal programlama, program tasarımı ve yazılmasını kurallara bağlayan ve disiplin altına alan bir yaklaşımdır.
- ▶ Yapısal programlama tekniğinde bir programın kolay yazılması, okunabilir olması ve hatalardan daha kısa sürede ayıklanması amaçlanmaktadır.
- ▶ Özellikle 70'li yıllar yapısal programmanın ilke olarak yerleşmekte olduğu yillardır.

# Yapısal Programlama



# Sıralı Yapılar

- ▶ Sıralı yapılar bir programdaki bir yada birden fazla deyimin göründükleri sırada yapılmasını sağlar.
- ▶ Sıralı işlemlerde en temel işlem atama işlemidir.
- ▶ En basit akış şeması ifadeleri, bir dizi işlemin birbiri ardından sırasıyla yapılmasını şeklinde olan akış ifadeleridir.
- ▶ Bu tip akışlar oldukça yalın ve basittir.
- ▶ Bu tarz akışlar genelde bir problemin bir parçasını çözümlemek ve ifade etmek için kullanılır.
- ▶ Sorgu ve tekrar gerektirmeyen bazı basit ardışık problemler de bu tip akış kullanabilir.

# Seçimlik Yapılar

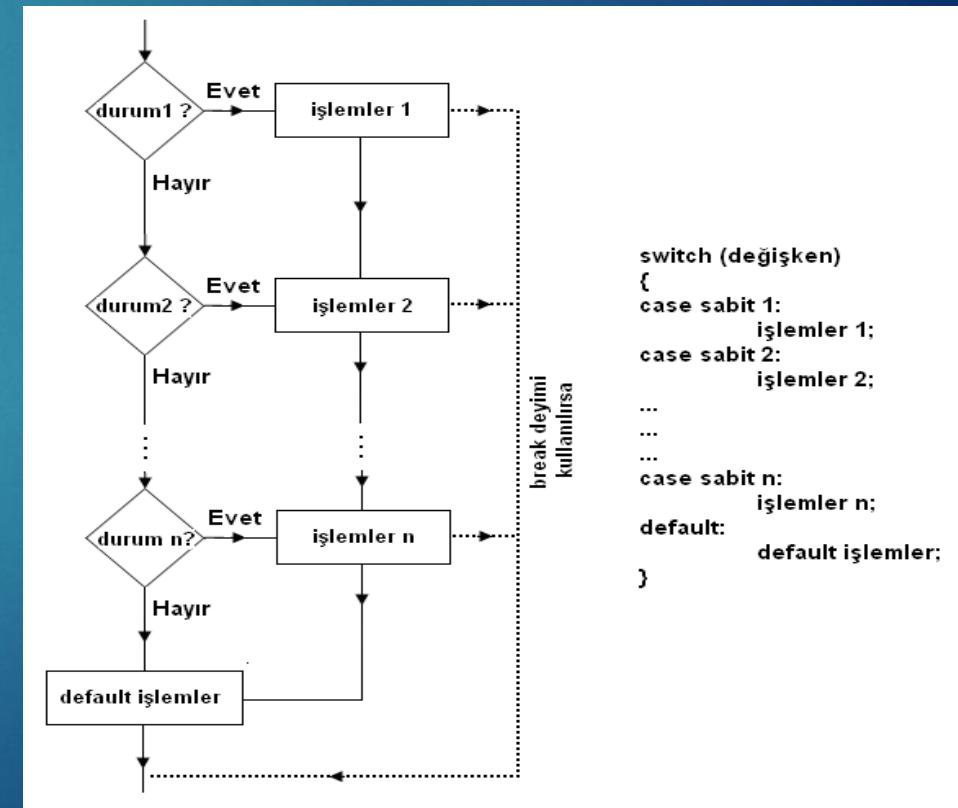
- ▶ Algoritma içerisinde verilen işlem adımları genel de sıralı adımlardan oluşur.
- ▶ Fakat bazı koşullarda bu işlem sıralarının değiştirilmesi ve diğer bir işlem sırasının seçilmesi gerekebilir.
- ▶ Soru (seçme) işlemi akış diyagramında baklava dilimi şeklindeki karşılaştırma simgesi ile ifade edilir.
- ▶ Simgenin içerisine koşul yazılır. Koşulun sonucuna göre iki yönden birisi seçilir.
- ▶ Program akışı, koşul olumlu ise “evet” olumsuz ise “hayır” olarak etiketlenen yöne dallanma yapar.

# Seçimlik Yapılar – İç içe (Nested) seçimlik yapılar

- ▶ İki-yollu seçim deyimleri içiçe yuvalandığında, else deyiminin hangi if deyimine ait olduğunu belirlenmesi güçleşir ve sallanan-else (dangling else) problemi oluşabilir.
- ▶ Bu problem dillerde, ikinci if-then yapısı bir deyimin gurubunun tek bir deyim gibi algılanmasını sağlayan birleşik deyim yapılması yoluyla çözülmüştür.
- ▶ Birleşik deyimler, ALGOL60, Pascal'da begin ... end yapısı, C'de ise { ... } yapısıyla sunulmaktadır.

# Seçimlik Yapılar- Çoklu seçim yapıları

- ▶ “switch” – “case” seçme yapısı bir değişkenin içeriğine bakarak programın akışını birçok seçenekten birisine yönlendiren bir karşılaştırma deyimidir.
- ▶ Değişkenin içeriği hangi sabit ile uyuşursa ona ait işlem kümesi ve arkasındaki bütün işlem kümeleri yürütülür.
- ▶ Ancak küme deyimleri arasında break kullanılırsa, daha sonraki tüm işlem kümeleri atlanarak “switch” bloğunun sonuna gidilir.



# Seçimlik Yapılar- Kısa devre değerlendirme

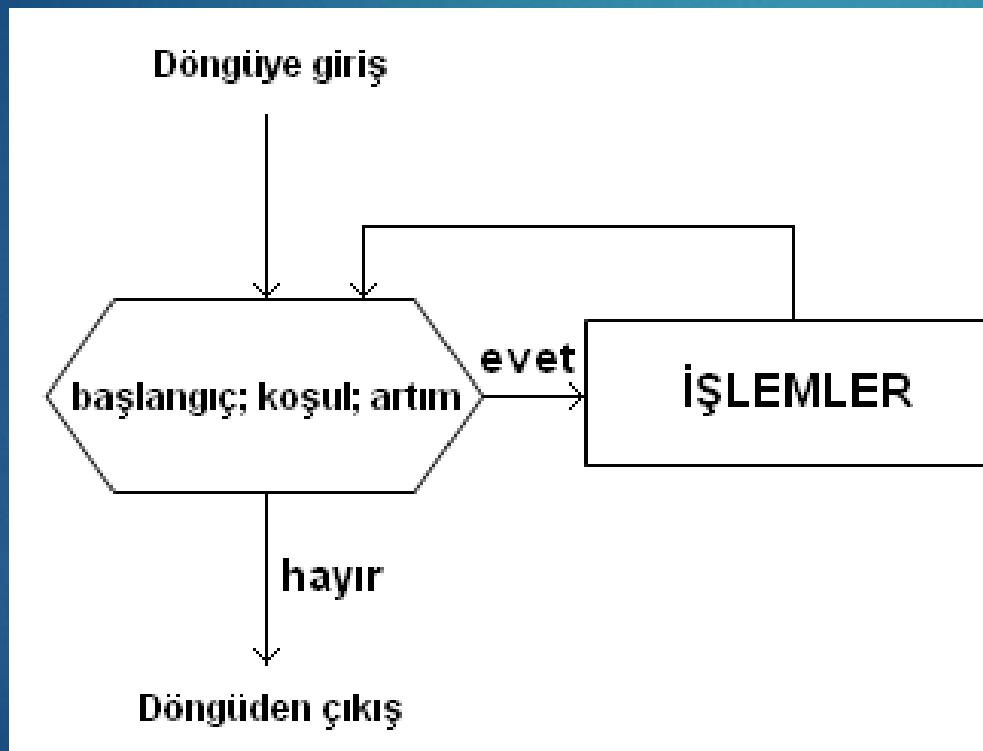
- ▶ Bir ifadenin sonucunun elde edilmesinde ifadede bulunan bütün operand yada operatörlerin değerlendirilmesine gerek yok ise burada kısa devre değerlendirme söz konusudur.
- ▶ Kısa devre değerlendirmede, ifadenin sonucunun elde edilmesi için tek bir bileşenin sonucu yeterli olabilir.
- ▶ Eğer sonucun belirlenmesinde bütün operand yada operatörlerin değerlendirilmesi gerekiyorsa buna **tam değerlendirme** denir. Tam değerlendirmede, ifadedeki her bileşen ayrı ayrı değerlendirilir.
- ▶ Bir programlama dilinin tam değerlendirme veya kısa devre değerlendirmeyi destekleyip desteklemediği dilin tasarımlı sırasında bağlanır.

# Tekrar Yapıları

- ▶ Uygulamalarda sıkılıkla döngü kurulması gerekmektedir.
- ▶ Bu döngü kurma işlemi, ya döngü deyimleriyle yada yapısal programlamada ilke olarak kullanılması istenilmeyen goto deyimiyle gerçekleştirilir.
- ▶ Döngü deyimleri uygulamalarda yazılan kod uzunluğunu azaltır.
- ▶ Tekrar yapıları temel olarak Sayaç kontrollü ve mantıksal kontrollü döngüler olmak üzere iki sınıfa ayıralabiliriz.

# Sayaç kontrollü döngü yapıları

## FOR Döngüsü:



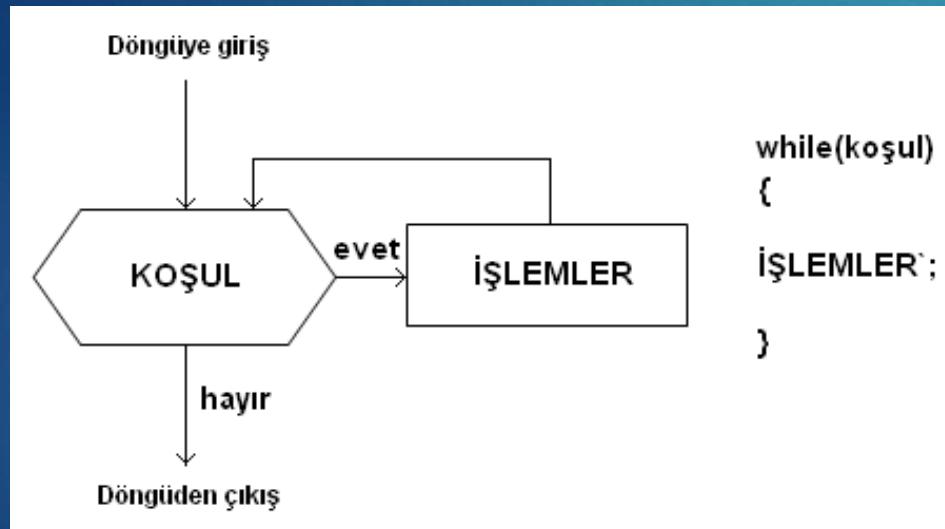
```
for(sayaç başlangıcı; koşul; artım)
{
    İŞLEMLER;
}
```

# Mantıksal Kontrollü Döngü yapıları

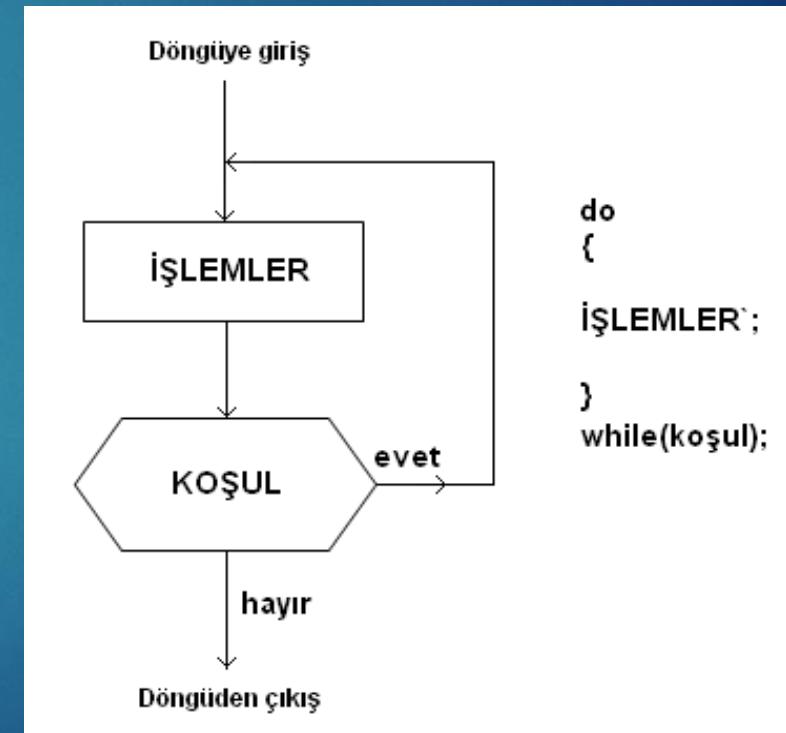
- ▶ Mantıksal kontrolü Tekrar Yapıları kendi içerisinde koşulu başta sınayan (pre-test), diğerı koşulu sonda sınayan (post-test) olmak üzere iki sınıfa ayrılır.
- ▶ Koşulu başa sınavması, daha çevrime girmeden döngü koşuluna bakılması ve koşul olumlu ise çevrime girilmesi, koşul olumsuz ise çevrime girilmeden sonraki adımlara geçilmesi anlamına gelir.
- ▶ Koşulun sonda sınanması ise, çevrim içerisinde kodun en az bir kere işletilmesi ve eğer koşul sağlanıyorsa çevrime devam edilmesi sağlanmıyor ise çevrimden çıkışması anlamına gelir.

# Mantıksal Kontrollü Döngü yapıları

## ► WHILE döngüsü



## DO-WHILE döngüsü



# Döngü kontrol mekanizmaları

- ▶ Bir program içinde deyimlerin akışı kullanıcı tarafından yönlendirilebilir.
- ▶ Dil döngünün bir tekrarının normalden önce tamamlanması için deyimler kullanılmasına izin verebilir.
- ▶ Bir döngünün tek başlangıç ve çıkış noktası olmalıdır. Oysa zaman zaman bir döngüden normalden önce çıkmak yada başka bir döngüye girmek gerekebilir.
- ▶ Java, C ve C++ dillerinde bir döngüden normalden önce çıkış için **break** deyimi, Modula-2, QuickBASIC ve Fortran 90 dillerinde ise ADA'da olduğu gibi **exit** deyimi kullanılır.
- ▶ C' de döngülerde akışı değiştirmek için ayrıca denetimi en içteki döngünün sınıma deyimine aktaran **continue** deyimi kullanılabilmektedir.

# Alt Programlar ve Modülasyon

- ▶ Çoğu problemin çözümünde genellikle uzun programlara ihtiyaç duyulmaktadır.
- ▶ Binlerce ifadeden oluşan bir programların yazılması ve anlaşılması oldukça zor bir işlemidir.
- ▶ Bu sebeple problemin daha kolay çözülebilen alt parçalarına (modül) ayrıştırılması işlemi yapısal programmanın temel yaklaşımlarındanandır.

# Alt program modül tanımı

- ▶ Küçük program parçacıkları olarak adlandırılabilen bu yapı alt program, modül, fonksiyon veya prosedüre şeklinde kendini gösterebilir.
- ▶ Altprogram kullanmanın faydaları aşağıdaki gibi sıralanabilir:
  - ▶ Belirli bir işi yapan program parçasının, birden çok yerde değişik veriler için ayrı ayrı yazılmasını önlerek program kodlarının gereksiz yere uzaması önlenmiş olur.
  - ▶ Düşük düzeydeki ayrıntıları, program mantığının ana akışından uzaklaştırıp ayrı modüllere yerleştirerek, programın tasarılanmasını kolaylaştırır ve okunabilirliğini artırır.
  - ▶ Büyük bir programın, daha küçük ve yazılması daha kolay fonksiyonel parçalara bölünür.
  - ▶ Program yazma işinin birden çok programcı tarafından paylaşılmasına olanak sağlar ve altprogramlar birden çok uygulama arasında paylaşılabilir.

# Fonksiyonlar

- ▶ Alt program olarak isimlendirilebilecek fonksiyon kendi kendine işlem görebilecek modülerliğe sahip bir işi gerçekleştirmek için yazılmış kod bloğudur.
- ▶ Altprogram (fonksiyon) çağrıldığı zaman formal parametreleri ve lokal değişkenleri ile birlikte çağrılır.
- ▶ Her altprogramın tek bir başlangıç noktası vardır.
- ▶ Bellek yeri ataması da çağrıldığı zaman gerçekleşir.
- ▶ Denetim çağrıldığı noktaya geçtiğinde ise bu bellek yeri boşaltılır.

# Fonksiyonlar

- ▶ Yığıt bellek, yerel değişkenler ve altprogramın çalışması süresince kullanılan parametreler gibi bir altprogram çağrımla ilişkili bilgiyi saklamak için kullanılır.
- ▶ Yığıt, LIFO mantığında çalışan bir veri yapısı olduğu için ve aynı anda tek bir altprogram etkin olabildiği için, altprogramların çalışma teknüğine uygundur.
- ▶ Etkin olan her altprogram için yığıt bellekte bir etkinlik kaydı (activation record) oluşturulur.
- ▶ Altprogramlarda kullanılan yerel değişkenler için her yordama ilişkin etkinlik kaydı kapsamında, yığıt bellekte bellek atanır.

# Fonksiyonlar

Program kodu Altprogram çağrılmamış deyimi Program kodu	
Altprogram Başlığı	Altprogram ismini, parametreler listesini (formal parametreler) ve varsa altprogramın döndürdüğü değer tipini bildirir.
Lokal değişkenler	Altprogramda geçerli değişkenler
Deyimler	Altprogramda tanımlı deyimler
Altprogram sonu	Altprogramın çalışması bittikten sonra kontrol çağrıldığı noktaya geçer.
Program kodu	
call prog1 (a,b,c)	call:çalıştırma deyimi prog1:altprogram ismi a,b,c: gerçek parametre

# Fonksiyonlarda çağrı kısmı

- ▶ Altprogramlar gerekli parametrelerle birlikte çağrılinca etkinleşir ve çalışması tamamlanıncaya kadar, diğer etkin olan programlar durdurulur.
- ▶ Eğer alt program bir fonksiyon ise fonksiyonun çalışması bitince, sonuç olarak tek bir değer üretilir ve bu değer, fonksiyonu çağrıran ifadeye döndürülerek fonksiyon çağrısının yerini alır.
- ▶ Böylece etkinlik yeniden ilk program birimine geçer.

# Fonksiyonlarda dönüş kısmı

- ▶ Bir alt program çağrıldığında ve dönüş kısmına geldiğinde bellekte yapılan işlemlere bakıldığında yerel değişkenler çıkarılır.
- ▶ İlgili parametreler karşılık gelen gerçek parametrelere aktarılır.
- ▶ Dönüş adresi yığıttan alınır ve bu adressteki komutlar çalıştırılır.
- ▶ Kontrol çağrılan noktadan devam eder.

# Prototip Tanımlama

- ▶ C ve C++ dilinde metod ve fonksiyonlar değişkenler gibidir.
- ▶ Çağrıldıkları yerden daha yukarıda tanımlanmış olmaları gereklidir.
- ▶ Bu tanımlama metodun tamamı olabileceği gibi sadece değişken tanımı gibi tanımlama yapılabilir.
- ▶ Bu tanımlama işlemine **prototip tanımlama** denir.
- ▶ Prototip tanımlamada metodun içeriği yazılmaz sadece parametre türleri ve dönüş türü yazılır.
- ▶ C dilinde, C++'tan farklı olarak dönüş türü int olan fonksiyonların prototipi tanımlanması zorunlu değildir.
- ▶ Derleyici bir fonksiyonu çağrııldığı yerden daha önce bulamaz ise onun dönüş türünün int olduğunu varsayıacak ve dosyada bu fonksiyonu arayacaktır.

# Parametre Aktarma Yöntemleri

- ▶ C dili için alt programın (metot ya da fonksiyon) veriye erişmesinin iki yolu vardır.
- ▶ Bunlardan biri lokal olmayan değişken tanımlayıp erişmek diğer ise parametreler.
- ▶ Lokal olmayan değişken yani global değişken tanımı önerilmeyen bir yöntemdir.
- ▶ Dolayısıyla en iyi yol parametre ile alt yordamların iletişime geçmesidir.
- ▶ Parametreler lokal değişkenlerdir ve çalışma anı yığınında tutulurlar. Fonksiyon çağrıları bittiğinde bellekten silinirler.

# Parametre Aktarma Yöntemleri

- ▶ Formal parametre, fonksiyon ya da altprogramın tanımlandığı yerdeki parametrelerdir.
- ▶ Gerçek parametre ise fonksiyonun çağrıldığı noktada karşılaşılan parametrelerdir.
- ▶ Fonksiyon çağrılığında bu gerçek parameterler ile formal parametreler arasında bir aktarım ilişkisi kurulacaktır.

# Parametre Aktarma Yöntemleri- Değer ile çağrıma – Pass by value

- ▶ Çağırın, çağrırlana değeri direkt olarak gönderir.
- ▶ Formal parametre, asıl parametre ile ilişkili olur.
- ▶ Sadece gerçek parametrelerden formal parametrelere doğru bir aktarım söz konusudur.
- ▶ Bütün dillerin desteklediği güvenli bir yöntemdir.
- ▶ Aktarım sırasında formal parametre için bellekte yer ayrıılır.

# Parametre Aktarma Yöntemleri- Referans ile çağrıma – Pass by reference

- ▶ C ve Java dilinde desteklenmeyen bu çağrıım şekli C++ dilinde desteklenmektedir.
- ▶ Aktarma işlemi sırasında formal parametre için ayrı bir bellek tahsisini yapılmaz.
- ▶ Kullanılan gerçek parametrenin belleğidir.

# Parametre Aktarma Yöntemleri- Gösterici (adres) ile çağrıma

- ▶ Bu çağrım türünde parametre türü bir göstericidir ve çağrıırken değer yerine adres gönderilir.
- ▶ Dolayısıyla adres ile çağrıma olarak ta isimlendirilir.
- ▶ Gerçek ve formal parametrelər ayrı ve birer gösterici olarak bellekte yer işgal eder.
  
- ▶ Referans ile gösterici ile çağrıma yöntemi karşılaştırıldığında referans ile çağrılmaya tercih edilmektedir.
- ▶ Çünkü formal parametrenin gösterdiği adresin değişme durumu olabilir. Bu nedenle referans ile çağrıma daha güvenlidir.

# Parametre Aktarma Yöntemleri- Sonuç ile çağrıma

- ▶ sonuç (out) parametresi alt programa (metot ya da fonksiyon) bir değer olarak gönderilmez bilakis fonksiyon içerisinde değer alıp gelir.
- ▶ Formal parametre lokal değişken gibi davranışır ve çağrıvana değer olarak döner.
- ▶ C++'ta referans çağrıma yöntemi kullanılarak yapılabilecek bu işlem C dilinde referans ile çağrıma desteklenmediği için C dilinde sonuç ile çağrıma yöntemi uygulanamaz .

# Parametre Aktarma Yöntemleri- isim ile çağrıma

- ▶ Popüler dillerde pek rastlanmayan aktarma yöntemidir.
- ▶ Algol dilinde destek vardır.
- ▶ Fonksiyon çağrılığında parametre olarak değer değil de fonksiyonun adı gider.

# Programlama Dillerinin Prensipleri

## Hafta 7 - Nesne Yönelimli Programlama

Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Nesnelerin Görünüşleri
- Nesneye Dayalı Düşünme
- Sınıf Hiyerarşisi
- this terimi
- Nitelikler
- İç içe Sınıflar
- Yıkıcı Metot
- Erişim Niteleyicileri
- Kalıtım
- Overload ve Override

# Tanım

- Nesneye dayalı programlama kompleks sistemleri yapılandırma ve yönetmeye imkan sağlamaktadır.
- Günlük hayatımızda kullandığımız şeylerin taklit edilmesi esasına dayanılır.
- Nesnelerin tanımlanmasını ve kullanımını destekleyen dillere nesne yönelimli dil denir.
- Tüm nesneye yönelik diller, programlama dilleri literatüründe sınıf ve alt sınıf kavramını ilk defa tanıtan SIMULA67 diline dayanmaktadır.

# Nesnelerin Görünüşleri

- Bir problemin çözümünde bir nesne herhangi bir varlık olarak ifade edilebilir.

## Harici Görünüş

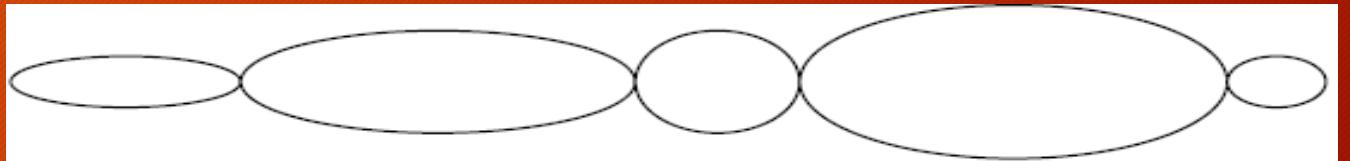
### Kargo Takip Sistemi

- Kargo
- Müşteri
- Nakliye

Bu kullanılabilecek varlıklarlardır. Bunlar nesnelerin harici görünüşüdür.

# Farklı Özelliklere Sahip Aynı Tür Nesneler

- Genişlik ve yükseklikleri farklı fakat aynı elips nesneleri



# Nesneye Dayalı Düşünme

- Object (nesne) : Veriler ve işlemler topluluğu
- Class (sınıf) : Nesneler kümesinin bir tanımı
- Subclass (altsınıf) : Bir sınıfın ilave özelliklere sahip birkümesi
- Instance (örnek) : Bir sınıfın bir nesnesi için kullanılan teknik bir terim
- Method (metod) : Bir işlemi yürüten bir altprogram içeriği
- Message (mesaj) : Bir metodu işlemek için bir istek, bir alt programın çağrılması

# Sınıf Hiyerarşisi

Shape - Şekil

Box - Kutu

Elipse - Elips

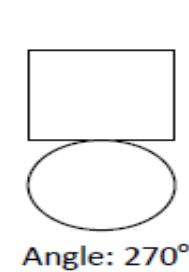
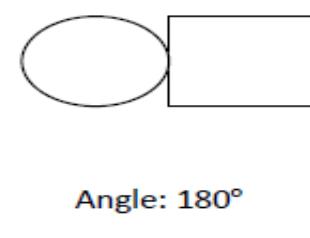
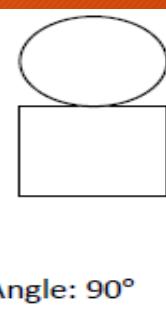
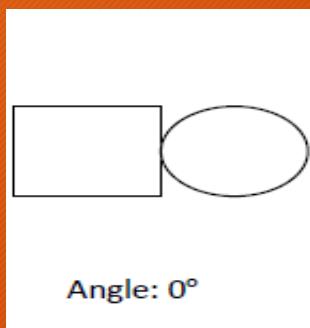
Circle - Daire

Line – Çizgi

Text

- Bir sınıfın içine girmiş sınıf alt sınıfıdır. Tersi süper sınıf veya üst sınıfıdır.
- Bir nesneye gelen bir mesaj, bir altprogramın çağrılmasına benzer.

# Nesne Türerilmesine Örnekler

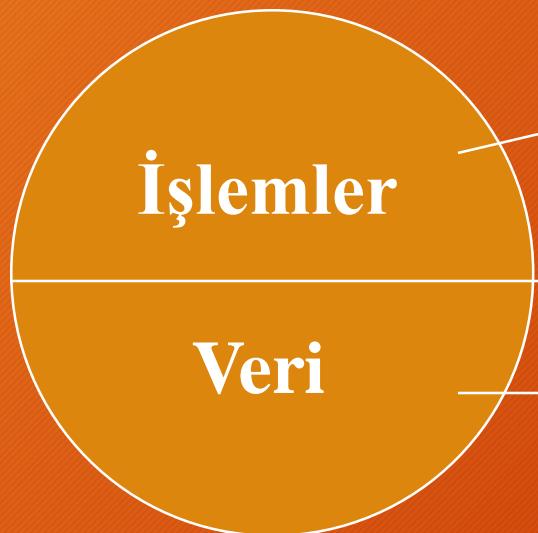


# Sınıf

- Kapsüllemeyi, veri gizlemeyi ve çok biçimliliği mümkün hale getiren yapı.
- Kullanıcı tanımlı bir türdür.
- Veri ve metotları içerir, dolayısıyla mesaj gönderilip cevap alınabilir.

# Nesne Terimi

**NESNE**



Üye Fonksiyonlar

Alt değişkenler ve private  
fonksiyonlar

# Nesnenin Oluşturulması

- Sınıftan türetilenek bir nesnede içerisinde bulunacak elemanlar ilk değerlerini almaları gereklidir.
- Bu durumda nesne oluşturma aşamasında bir metot çağrılmalıdır.
- Bu fonksiyon kurucu veya yapıcısı (constructor) fonksiyondur.

```
// Java  Ogrenci o1 = new Ogrenci("Ahmet Almaz",159);
class Ogrenci{
    private String isim;
    private int numara;

    public Ogrenci(String ism,int nmr){
        isim=ism;
        numara = nmr;
    }
}
```

```
//C++  Ogrenci o1("Ahmet Almaz",159);
class Ogrenci{
private:
    string isim;
    int numara;
public:
    Ogrenci(String ism,int nmr){
        isim=ism;
        numara = nmr;
    }
};
```

# this Terimi

- Sınıf hiyerarşisinde, o anda oluşturulan nesneyi ifade etmek için kullanılır.
- Bazı durumlarda kullanımı gereklidir.
- Ayrıca okunabilirliği artırmak için de kullanılmaktadır.

## this teriminin kullanımı

```
class Kisi{  
    public String isim;  
    ...  
  
    public Kisi(String isim){  
        isim=isim;  
        yas=0;  
        boy=20;  
        kilo=4;  
    }  
    ...  
}
```

```
// doğrusu  
public Kisi(String isim){  
    this.isim=isim;  
    yas=0;  
    boy=20;  
    kilo=4;  
}
```

# Nitelik Tanımı (Property)

- Nitelikler bazı programlama dillerinde desteklenmektedirler.
- Amaç sınıfındaki bir veya birden fazla alt alanı okuma ve yazma yönünden korumaya almak veya yönetebilmektir.
- Nitelik desteklemeyen programlama dillerinde bu metodlar yardımıyla yapılır.

```
class Kisi
{
    private double kilo;
    private double kalori;

    public Kisi()
    {
        kilo = 4;
        kalori = 0;
    }

    // C# dili
}

public double Kalori
{
    get
    {
        return kalori;
    }
    set
    {
        kalori = value;
        if (kalori > 1000)
            kilo += (kalori / 1000);
    }
}

public double Kilo
{
    get
    {
        return kilo;
    }
}
```

# Yıkıcı Metotlar

- Çöp toplayıcı araçlarına sahip diller için dikkat edilmesine gerek olmayan metotlardır.
- Fakat söz konusu C++ gibi bir dil ise Heap bellek bölgesinde oluşturulan nesnelerin belleğe geri iadesi gerekeceğinden yıkıcı metotların önemi büyütür.

# Yıkıcı Metotlar

- C++ dilinde programcı tarafından çağrılabilen yıkıcı metot java ve C#, Python gibi dillerde çöp toplayıcı tarafından çağrılır.

```
//Java  
public class Arac{  
    @Override  
    protected void finalize() throws Throwable {  
        // Çöp toplayıcı tarafından çağrılr.  
    }  
}
```

```
// C#  
class Arac : IDisposable  
{  
    ~Arac()  
    {  
        // Çöp toplayıcı tarafından  
    }  
  
    public void Dispose()  
    {  
        // Programcı çağrırlabilir  
        // Nesnenin yok edilmesini garantiyor  
    }  
}
```

```
// Python  
class Arac:  
    def __del__(self):  
        # Çöp toplayıcı tarafından
```

# Erişim Niteleyicileri

- Sınıflarda bilgi gizleme, dış kullanıma açma, kalıtım için kullanma gibi işlemler için erişim niteleyicileri kullanılır.

Varsayılan: Bazı dillerde

C++ (struct) public, C++(class) private

C# private

Java aynı pakette ise sınıf dışından erişilebilir.

private: Sadece sınıf içerisinde erişilebilir.

protected: Sınıf dışından sadece kalıtım yolu ile erişilebilir.

public: Sınıf içi ve dışında erişilebilir.

# Erişim Niteleyicileri

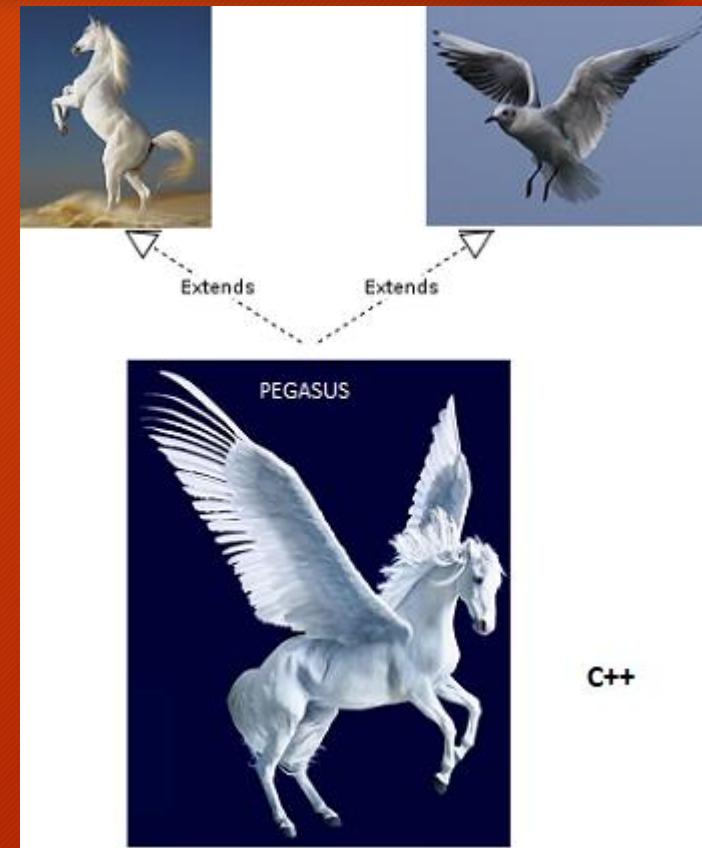
- Bilgi gizleme iki çeşit değişikliğe imkan sağlar.
  - Yürütme (implementation) Değişiklikleri
    - Eğer bir nesne ile olan bütün etkileşimler o nesnenin arayüzü üzerinden yapılıyorsa o zaman arayüzün arkasında gizlenen algoritmalar ve veri yapıları yürütme yardımıyla değiştirilebilir.
  - Kalıtım Değişiklikleri
    - Bir üst sınıfa olan bütün etkileşimler arayüzler üzerinden olursa program alt sınıflarının eklenmesi ile genişletilebilir.

# Kalıtım

- Birçok tür tamamen bağımsız olmayacağı ve bazı özellikleri başka türlere benzeyecektir.
- Bu durumda bu özelliklerin bir üst sınıfın alınması geliştirme sürecini hızlandırır.
- Kalıtım alınan sınıfı üst sınıf denir ve programlama dillerinde yukarıya mesaj gönderme şekli;
  - Java      super
  - C#        base
  - C++      Sınıf adı ile

# Kalıtım

- C++ gibi bazı dillerde çoklu sınıf kalıtımı desteklenir.
- Java ve C# dillerinde çoklu arayüz kalıtımı desteklenmektedir.
- Çok biçimlilik yine kalıtım ile desteklenmektedir.



# Çoklu Kalıtım ve Diamond Problemi

```
class Canli{
    public:
        double kilo;
};

class Kus : public Canli{
    public:
        Kus(){
            kilo=0.25;
        }
        void YemekYe(double kalori){
            kilo += (kalori/10000);
        }
        double Kilo(){
            return kilo;
        }
};
class At : public Canli{
    public:
        At(){
            kilo=100;
        }
        void YemekYe(double kalori){
            kilo += (kalori/1000);
        }
        double Kilo(){
            return kilo;
        }
};
class Pegasus : public At, public Kus{
    public:
        Pegasus(){
        }
};

int main(){
    Pegasus *p = new Pegasus();
    p->YemekYe(1500);
    cout<<p->Kilo();
    delete p;
    return 0;
}
```

# Overload (Aşırı Yükleme) ve Override (Ezme) Terimleri

- Overload bir metoda yeni anlamlar kazandırarak çok biçimliliği destekler.
- Override ise metodu yeniden tanımlamaktır. Dolayısıyla eski metodu ezmiş olacaktır.

## C# dilinde operatörlerin aşırı yüklenmesi

```
class Sayi
{
    public int deger;
    public Sayi(int dgr)
    {
        deger = dgr;
    }
    public static Sayi operator +(Sayi s1, Sayi s2)
    {
        return new Sayi(s1.deger+s2.deger);
    }
    public override string ToString()
    {
        return this.deger.ToString();
    }
}
```

## C#'ta ilkel türe benzer kullanım

```
static void Main(string[] args)
{
    Sayi x = new Sayi(10);
    Sayi y = new Sayi(5);
    Console.WriteLine(x+y);
    Console.ReadKey();
}

static void Main(string[] args)
{
    int x = 10, y = 5;
    Console.WriteLine(x+y);
    Console.ReadKey();
}
```

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016

# Programlama Dillerinin Prensipleri

## Hafta 8 - Nesne Yönelimli Programlama - 2

Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Arayüzler
- Çoklu Sınıf ve Arayüz Kalıtımıları
- namespace ve Paketler
- Soyut Sınıflar
- Kalıtım Hiyerarşisi
- Object Veri Türü
- Generic - Şablon Yapılar
- Friend Erişim Niteleyicisi

# Arayüz Tanımı

- Arayüz kullanımı aslında bir sözleşme imzalamaktır.
- Bir arayüzden kalıtım alan bir sınıf o arayüzü gerçekleştireceğini vadediyor demektir.
- Arayüzler yardımıyla kullanılabılırlik dış dünyaya rahatlıkla açılabilir.
- Java ve C#'ta arayüzler alan içeremez sadece public metot tanımı içerebilirler.
- Java ve C# dilinde direkt desteği bulunan arayüzler, C++ dilinde soyut sınıflar kullanılarak gerçekleştirilir.

```
public interface MuzikCalar {  
    public void Oynat(String dosyaTuru, String dosyaAdi);  
}
```

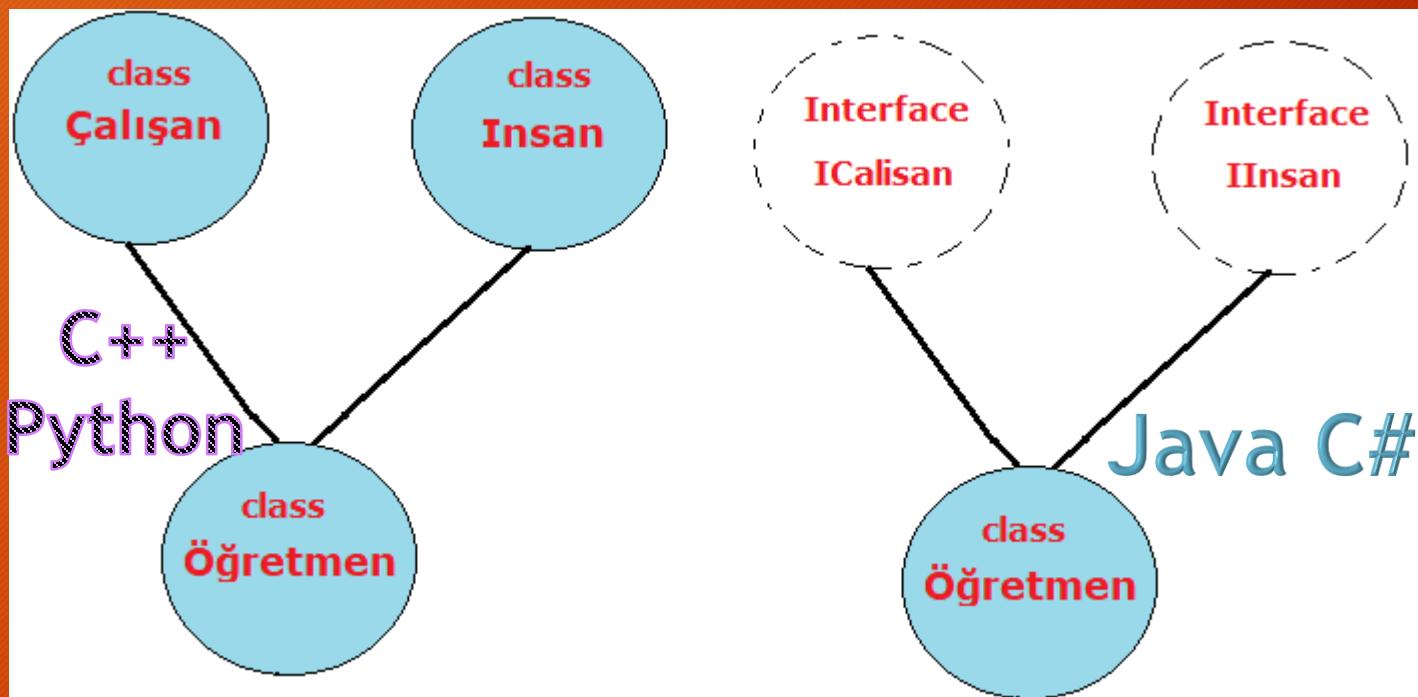
# Arayüzler

```
interface IEmail{
    public void setGonderen(String gonderen);
    public void setAlan(String alan);
    public void setMesaj(IIcerik mesaj);
}
interface IIcerik{
    public String StringOlarakIcerik();
}
class Email implements IEmail{
    public void setGonderen(String gonderen){ /*Göndereni ayarlar*/ }
    public void setAlan(String alan){ /*Alanı ayarlar*/ }
    public void setMesaj(IIcerik mesaj){ /*Mail mesajını ayarlar*/ }
}
```

```
class Html implements IIcerik{
    public String StringOlarakIcerik(){
        // HTML'i String olarak ifade et.
    }
}
```

# Arayüzler

- Çoklu sınıf kalıtımı izin verilmeyen dillerde çözüm olarak arayüz bulunmalıdır. Çoklu arayüz kalıtımı yapılmalıdır.



# Çoklu Sınıf Kalıtımı

```
class Sekil
{
    protected double genislik;
    protected double yukseklik;
    1 başvuru
    public Sekil(double genislik,double yukseklik)
    {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
    }
}
```

C#



base

```
class Daire : Sekil
{
    0 başvuru
    public Daire(double g,double y) : base(g,y)
    {
    }
}
```

```
class Insan{
    protected:
        double boy;
    public:
        Insan(double boy) { this->boy = boy; }
};

class Calisan{
    protected:
        int Id;
    public:
        Calisan(int Id){ this->Id = Id; }
};

class Ogretmen : public Insan, public Calisan{
    public:
        Ogretmen(double boy,int Id) : Insan(boy),Calisan(Id) {
        }
};

class Insan:
    ...
class Calisan:
    ...
class Ogretmen(Insan, Calisan):
    super()
```

## Python

Yazılan sıraya göre çağrıır  
Önce Insan sonra Calisan sınıfının  
Kurucusu çağrırlır.

# namespace (isim uzayları) ve Paketler

- Bir arayüzden farklı olarak isim uzayları kodu organize etmek, derli toplu bir halde bulunmasını sağlar. C++ ve C# dillerinde desteklenir.
- Java dilindekiavourites karşılığı paketlerdir ve klasör mantığı ile saklanır.

```
namespace A{
    double Hesapla() { return 0; }
}
namespace B{
    double Hesapla() { return 1; }
}
int main(){
    cout<<A::Hesapla()<<endl; 0 yazar
    cout<<B::Hesapla()<<endl; 1 yazar
    return 0;
}
```

C++

# Soyut Sınıflar

- İçerisinde tamamlanmamış alanlar içeren sınıflara denir.
- Arayüzden farkı tanımlanmış alanlar da içerebilirler.
- Tanımlanmamış alanlar kalıtım yolu ile tanımlanır.
- Eksik tanım içerdikleri için soyut sınıflardan nesne türetilemez.

# Soyut Sınıflar

## C++

```
class Canli{
    private:
        int yas;
    public:
        virtual void YemekYe() = 0;
        int Yas() const { return yas; }
};

class Kedi : public Canli{
    public:
        void YemekYe() {
        }
};


```

## C#

```
public abstract class Canli{
    public int yas { get; }

    public abstract void YemekYe();
}

public class Kedi : Canli
{
    public override void YemekYe()
    {
    }
}
```

# Kalıtım Hiyerarşisi

- Java ve C#



# Object Veri Türü

- Herhangi bir türü içinde barındırabilecek şekilde tasarlanmış veri türüdür.
- Boxing ve Unboxing kullanımlarında büyük önem arz eder.
- Java dilinde en üstteki sınıf Object sınıfıdır ve bu sınıfın diğer bütün sınıflar gizli olarak kalıtım alır.
- Java ve C# dillerinde Object sınıfı bulunurken, C++ dilinde böyle bir sınıf yoktur.

# Generic - Şablon Yapıları

- C# ve C++ dillerinde aktif olarak kullanılan şablon yapıları Java diline sonradan dahil olmuştur.
- Bir dilde Object sınıfı ile şablon yapıları taklit edilebilir.

```
public class Koleksiyon<Tur>
{
    public Tur[] Elemanlar;

    public Koleksiyon(Tur []Elemanlar)
    {
        this.Elemanlar = Elemanlar;
    }

    public String EnBuyukEnKucukBirlestir()
    {
        return Elemanlar.Max().ToString() + Elemanlar.Min().ToString();
    }
}
```



```
int[] tamsayilar = { 15,95,20,2,18,32};
Koleksiyon<int> koleksiyonsayilar = new Koleksiyon<int>(tamsayilar);
koleksiyonsayilar.EnBuyukEnKucukBirlestir(); 952 döner

char[] karakterler = { 'a', 'w', 'r', 't', 'k', 'p' };
Koleksiyon<char> koleksiyonkarakterler = new Koleksiyon<char>(karakterler);
koleksiyonkarakterler.EnBuyukEnKucukBirlestir(); wa döner
```

# Generic - Şablon Yapılar

```
template <typename Tur>
class Koleksiyon{
public:
    Tur *Elemanlar;
    int uzunluk;
    Koleksiyon(Tur *Elemanlar,int uzunluk){
        this->Elemanlar = Elemanlar;
        this->uzunluk = uzunluk;
    }
    string EnBuyukEnKucukBirlestir(){
        return toString(*max_element(Elemanlar,Elemanlar+uzunluk))+  
               toString(*min_element(Elemanlar,Elemanlar+uzunluk));
    }
    string toString(Tur t)
    {
        ostringstream ss;
        ss << t;
        return ss.str();
    }
};
```



# friend Erişim Niteleyicisi

```
class Sayi{  
    private:  
        double deger;  
        void setDeger(double dgr){  
            deger = dgr;  
        }  
        friend class Top;  
};  
class Top{  
    private:  
        Sayi *s;  
    public:  
        Top(){  
            s = new Sayi();  
            s->setDeger(100);  
        }  
        double getDeger() const{  
            return s->deger;  
        }  
};
```



- Java dilinde **friend** diye bir terim yoktur. Fakat bu işlem, Sınıfları aynı pakete yerleştirerek kısmen gerçekleştirilebilir.
- Aynı paketteki sınıflar birbirlerinin **protected** ve erişim niteleyicisi olmayan elemanlarına erişebilirler.

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016

# Programlama Dillerinin Prensipleri

## Hafta 9 - Nesne Yönelimli Programlama - 2

Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Arayüzler
- Çoklu Sınıf ve Arayüz Kalıtımıları
- namespace ve Paketler
- Soyut Sınıflar
- Kalıtım Hiyerarşisi
- Object Veri Türü
- Generic - Şablon Yapılar
- Friend Erişim Niteleyicisi

# Arayüz Tanımı

- Arayüz kullanımı aslında bir sözleşme imzalamaktır.
- Bir arayüzden kalıtım alan bir sınıf o arayüzü gerçekleştireceğini vadediyor demektir.
- Arayüzler yardımıyla kullanılabılırlik dış dünyaya rahatlıkla açılabilir.
- Java ve C#'ta arayüzler alan içeremez sadece public metot tanımı içerebilirler.
- Java ve C# dilinde direkt desteği bulunan arayüzler, C++ dilinde soyut sınıflar kullanılarak gerçekleştirilir.

```
public interface MuzikCalar {  
    public void Oynat(String dosyaTuru, String dosyaAdi);  
}
```

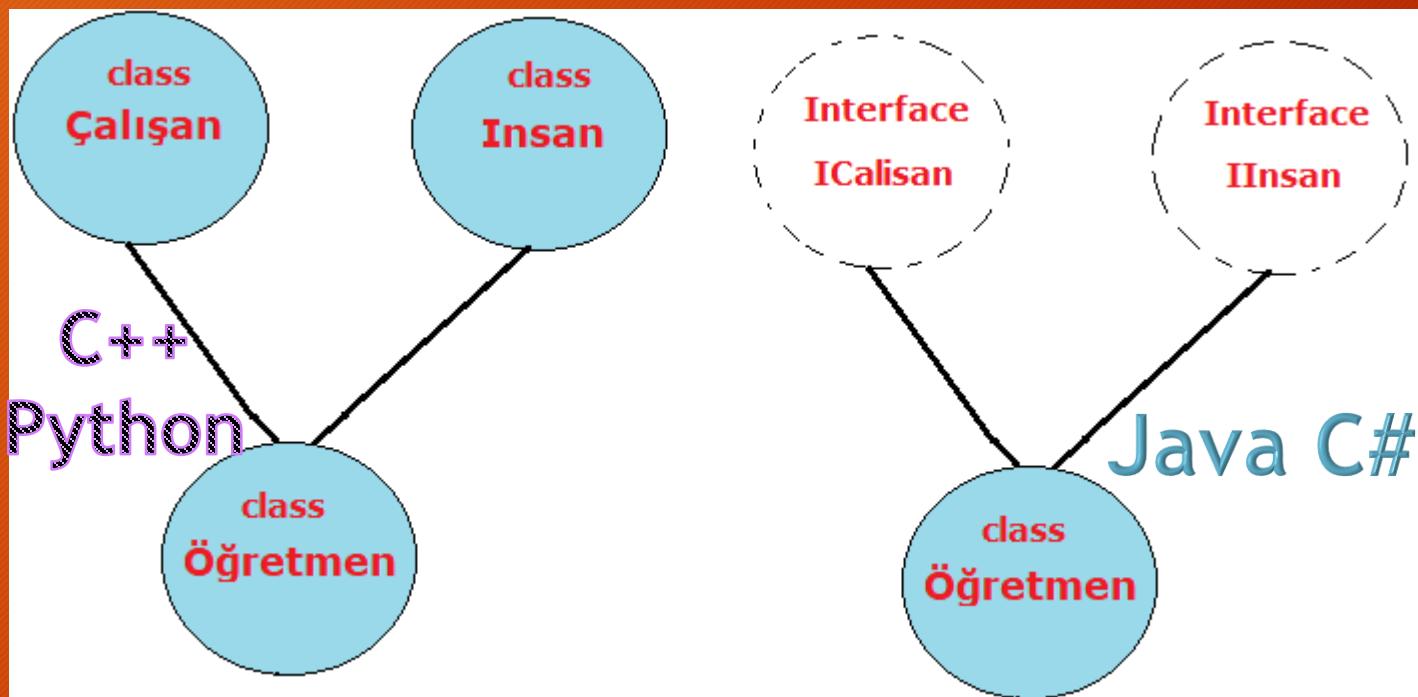
# Arayüzler

```
interface IEmail{
    public void setGonderen(String gonderen);
    public void setAlan(String alan);
    public void setMesaj(IIcerik mesaj);
}
interface IIcerik{
    public String StringOlarakIcerik();
}
class Email implements IEmail{
    public void setGonderen(String gonderen){ /*Göndereni ayarlar*/ }
    public void setAlan(String alan){ /*Alanı ayarlar*/ }
    public void setMesaj(IIcerik mesaj){ /*Mail mesajını ayarlar*/ }
}
```

```
class Html implements IIcerik{
    public String StringOlarakIcerik(){
        // HTML'i String olarak ifade et.
    }
}
```

# Arayüzler

- Çoklu sınıf kalıtımı izin verilmeyen dillerde çözüm olarak arayüz bulunmalıdır. Çoklu arayüz kalıtımı yapılmalıdır.



# Çoklu Sınıf Kalıtımı

```
class Sekil
{
    protected double genislik;
    protected double yukseklik;
    1 başvuru
    public Sekil(double genislik,double yukseklik)
    {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
    }
}
```

C#



base

```
class Daire : Sekil
{
    0 başvuru
    public Daire(double g,double y) : base(g,y)
    {
    }
}
```

```
class Insan{
    protected:
        double boy;
    public:
        Insan(double boy) { this->boy = boy; }
};

class Calisan{
    protected:
        int Id;
    public:
        Calisan(int Id){ this->Id = Id; }
};

class Ogretmen : public Insan, public Calisan{
    public:
        Ogretmen(double boy,int Id) : Insan(boy),Calisan(Id) {
        }
};

class Insan:
    ...
class Calisan:
    ...
class Ogretmen(Insan, Calisan):
    super()
```

## Python

Yazılan sıraya göre çağrıır  
Önce Insan sonra Calisan sınıfının  
Kurucusu çağrırlır.

# namespace (isim uzayları) ve Paketler

- Bir arayüzden farklı olarak isim uzayları kodu organize etmek, derli toplu bir halde bulunmasını sağlar. C++ ve C# dillerinde desteklenir.
- Java dilindekiavourites karşılığı paketlerdir ve klasör mantığı ile saklanır.

```
namespace A{
    double Hesapla() { return 0; }
}
namespace B{
    double Hesapla() { return 1; }
}
int main(){
    cout<<A::Hesapla()<<endl; 0 yazar
    cout<<B::Hesapla()<<endl; 1 yazar
    return 0;
}
```

C++

# Soyut Sınıflar

- İçerisinde tamamlanmamış alanlar içeren sınıflara denir.
- Arayüzden farkı tanımlanmış alanlar da içerebilirler.
- Tanımlanmamış alanlar kalıtım yolu ile tanımlanır.
- Eksik tanım içerdikleri için soyut sınıflardan nesne türetilemez.

# Soyut Sınıflar

## C++

```
class Canli{
    private:
        int yas;
    public:
        virtual void YemekYe() = 0;
        int Yas() const { return yas; }
};

class Kedi : public Canli{
    public:
        void YemekYe() {
        }
};


```

## C#

```
public abstract class Canli{

    public int yas { get; }

    public abstract void YemekYe();
}

public class Kedi : Canli
{
    public override void YemekYe()
    {
    }
}
```

# Kalıtım Hiyerarşisi

- Java ve C#



# Object Veri Türü

- Herhangi bir türü içinde barındırabilecek şekilde tasarlanmış veri türüdür.
- Boxing ve Unboxing kullanımlarında büyük önem arz eder.
- Java dilinde en üstteki sınıf Object sınıfıdır ve bu sınıfın diğer bütün sınıflar gizli olarak kalıtım alır.
- Java ve C# dillerinde Object sınıfı bulunurken, C++ dilinde böyle bir sınıf yoktur.

# Generic - Şablon Yapıları

- C# ve C++ dillerinde aktif olarak kullanılan şablon yapıları Java diline sonradan dahil olmuştur.
- Bir dilde Object sınıfı ile şablon yapıları taklit edilebilir.

```
public class Koleksiyon<Tur>
{
    public Tur[] Elemanlar;

    public Koleksiyon(Tur []Elemanlar)
    {
        this.Elemanlar = Elemanlar;
    }

    public String EnBuyukEnKucukBirlestir()
    {
        return Elemanlar.Max().ToString() + Elemanlar.Min().ToString();
    }
}
```



```
int[] tamsayilar = { 15,95,20,2,18,32};
Koleksiyon<int> koleksiyonsayilar = new Koleksiyon<int>(tamsayilar);
koleksiyonsayilar.EnBuyukEnKucukBirlestir(); 952 döner

char[] karakterler = { 'a', 'w', 'r', 't', 'k', 'p' };
Koleksiyon<char> koleksiyonkarakterler = new Koleksiyon<char>(karakterler);
koleksiyonkarakterler.EnBuyukEnKucukBirlestir(); wa döner
```

# Generic - Şablon Yapılar

```
template <typename Tur>
class Koleksiyon{
public:
    Tur *Elemanlar;
    int uzunluk;
    Koleksiyon(Tur *Elemanlar,int uzunluk) {
        this->Elemanlar = Elemanlar;
        this->uzunluk = uzunluk;
    }
    string EnBuyukEnKucukBirlestir() {
        return toString(*max_element(Elemanlar,Elemanlar+uzunluk))+  

            toString(*min_element(Elemanlar,Elemanlar+uzunluk));
    }
    string toString(Tur t)
    {
        ostringstream ss;
        ss << t;
        return ss.str();
    }
};
```



# friend Erişim Niteleyicisi

```
class Sayi{  
    private:  
        double deger;  
        void setDeger(double dgr){  
            deger = dgr;  
        }  
        friend class Top;  
};  
class Top{  
    private:  
        Sayi *s;  
    public:  
        Top(){  
            s = new Sayi();  
            s->setDeger(100);  
        }  
        double getDeger() const{  
            return s->deger;  
        }  
};
```



- Java dilinde **friend** diye bir terim yoktur. Fakat bu işlem, Sınıfları aynı pakete yerleştirerek kısmen gerçekleştirilebilir.
- Aynı paketteki sınıflar birbirlerinin **protected** ve erişim niteleyicisi olmayan elemanlarına erişebilirler.

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016

# Programlama Dillerinin Prensipleri

## Hafta 12 - Eş Zamanlı Programlama

Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Tanım
- Öncelik Grafları
- Eşzamanlık Şartları
- Fork ve Join Yapıları
- Parbegin ve Parenend Yapıları
- Programlama Dillerinde Eş Zamanlılığın Gerçekleştirimi
- İşlem Durumları
- İşlem Grafları
- Kritik Bölge
- Semaforlar

# Tanım

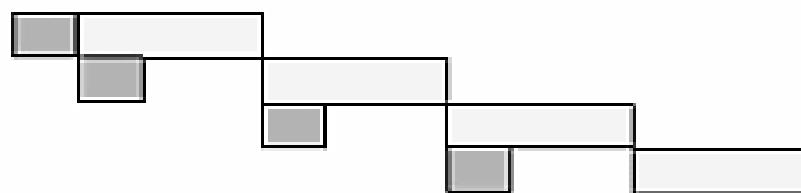
- Programlama dillerindeki eş zamanlılık kavramı ile bilgisayar donanımındaki paralel çalışma birbirinden bağımsız kavramlardır.
- Programlama dillerinde eş zamanlılık belli kavramlar kullanılarak gerçekleştirilebilir.
- Amaç programın hesaplama hızını arttırip verimi yükseltmektir.
- Bir programda eş zamanlılık kullanılmamışsa varsayılan olarak bir thread o programı yürütecektir.

# Donanımsal Paralellik

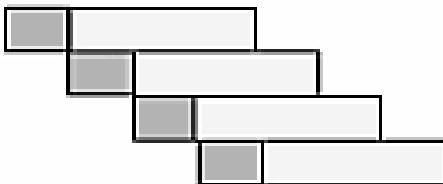
**Thread Yok**



**Çok Thread  
Tek İşlemci**

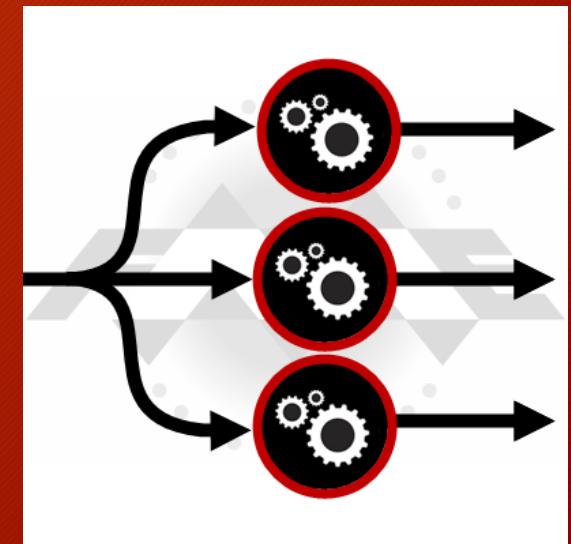


**Çok Thread  
Çok İşlemci**



# Eş Zamanlılık Türleri

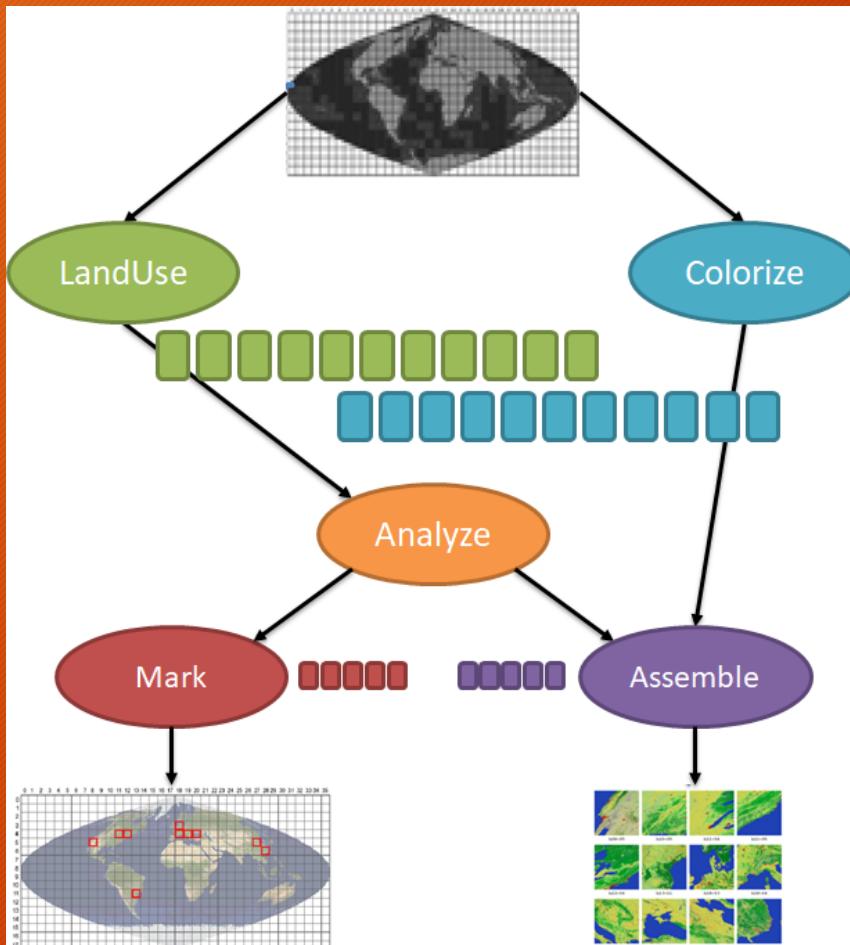
- Program içerisindeki eş zamanlılık 4 farklı şekilde gerçekleşebilir.
  - **Makine komutu düzeyinde** : 2 veya daha fazla makine komutunun paralel çalıştırılması
  - **Program kod satırı düzeyinde**: 2 veya daha fazla program kod satırının paralel çalıştırılması
  - **Birim seviyesinde** : 2 veya daha fazla fonksiyonun paralel çalıştırılması
  - **Program seviyesinde** : 2 veya daha fazla programın paralel çalıştırılması



# Programlama Dillerinde Gerçekleştirimi

- Bazı dillerde kütüphaneler yardımıyla gerçekleştirilir.
  - Örnek: OpenMP - C/C++ ve Fortran
- Diğer bazı diller bunu kendi içerisinde sağlayabilir.
- Bu şekilde ilk destek PL/I programlama dili ile başlamıştır.
- Daha sonra bunu, Ada95, Java, C#, Python ve Ruby takip etmiştir.

# Python Paralelleştirme Örneği

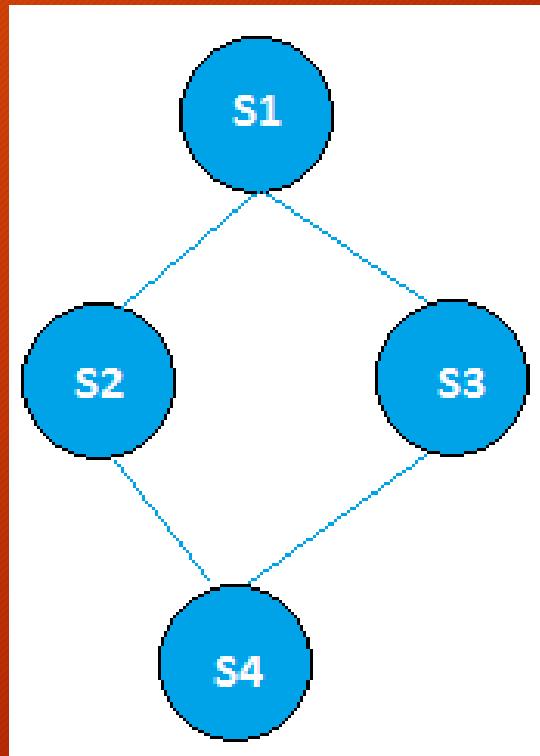


# Öncelik Grafları

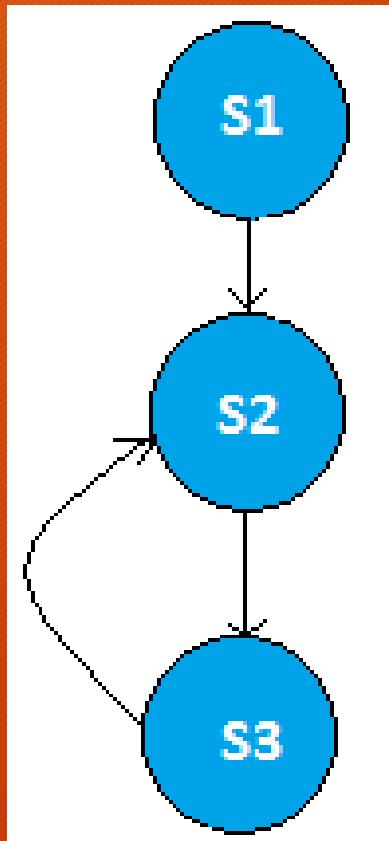
- Bağımlı ve bağımsız değişkenlerin tespit edilip hangi işlerin eş zamanlı çalıştırılabileceği belirlenir.
- Bir öncelik grafi hangi işlemlerin eş zamanlı hangi işlemlerin seri olarak çalıştırılması gerekeceğini gösterir.
- Hangi işlemlerin hangi işlemleri bekleyeceği görülebilir.

# Öncelik Grafları

- S2 ve S3 çalıştırılabilmesi için S1 ifadesi işlemini bitirmelidir.
- S2 ve S3 eş zamanlı çalıştırılabilir.
- S4 çalışabilmesi için hem S2 hem de S3 işlemini bitirmelidir.



# Hatalı Öncelik Grafi



# Eşzamanlı Şartları

$R(S_i) = \{a_1, a_2, \dots, a_n\}$ :  $S_i$  için “oku” kümesi.

$W(S_i) = \{b_1, b_2, \dots, b_n\}$ :  $S_i$  için “yaz” kümesi.

$S_1$  ve  $S_2$  eşzamanlı çalışması için aşağıdaki 3 şart gerçekleşmelidir.

1.  $R(S_1) \cap W(S_2) = \emptyset$
2.  $W(S_1) \cap R(S_2) = \emptyset$
3.  $W(S_1) \cap W(S_2) = \emptyset$

# Eşzamanlı Şartları

S1  $a := x + y;$        $R(S1) = \{x, y\}$

S2  $b := z + 1;$        $R(S2) = \{z\}$

S3  $c := a - b;$        $R(S3) = \{a, b\}$

$W(S1) = \{a\}$

$W(S2) = \{b\}$

$W(S3) = \{c\}$

S1 ve S2 deyimleri eş zamanlı olarak çalışabilir mi?

Koşul 1.  $R(S1) \cap W(S2) = \{x, y\} \cap \{b\} = \{\}$

Koşul 2.  $W(S1) \cap R(S2) = \{a\} \cap \{z\} = \{\}$

Koşul 3.  $W(S1) \cap W(S2) = \{a\} \cap \{b\} = \{\}$



S1 ve S3 deyimleri eş zamanlı olarak çalışabilir mi?

Koşul 1.  $R(S1) \cap W(S3) = \{x, y\} \cap \{c\} = \{\}$

Koşul 2.  $W(S1) \cap R(S3) = \{a\} \cap \{a, b\} = \{a\}$

Koşul 3.  $W(S1) \cap W(S3) = \{a\} \cap \{c\} = \{\}$



S2 ve S3 deyimleri eş zamanlı olarak çalışabilir mi?

Koşul 1.  $R(S2) \cap W(S3) = \{z\} \cap \{c\} = \{\}$

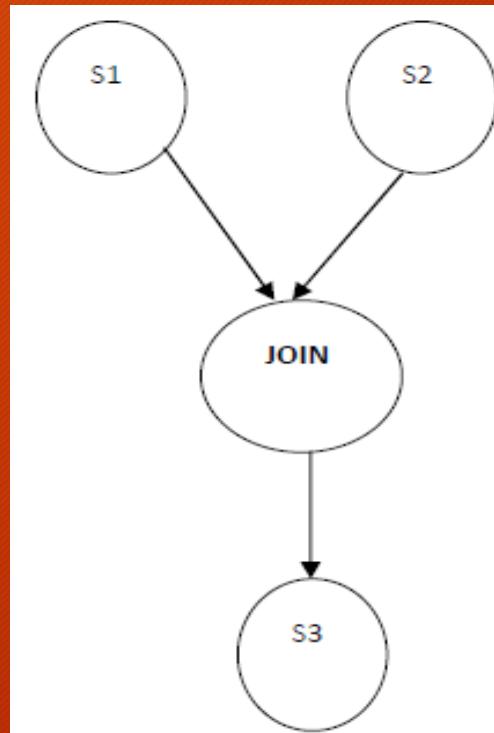
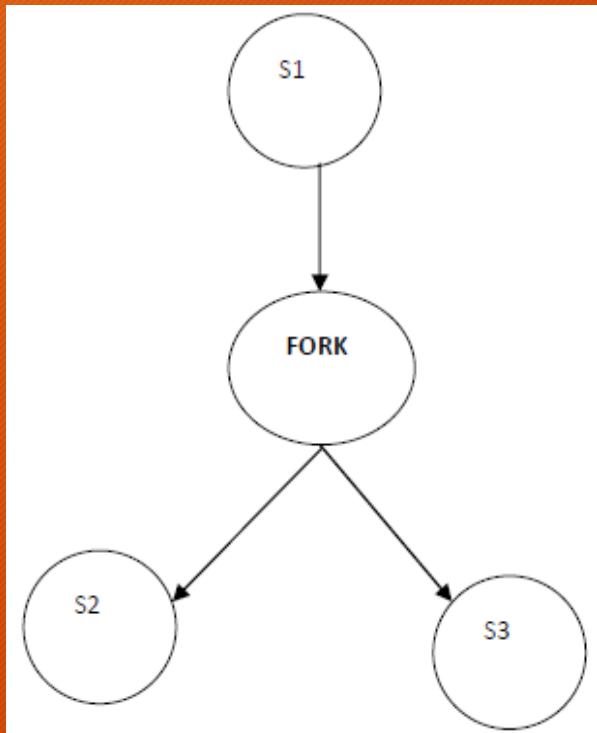
Koşul 2.  $W(S2) \cap R(S3) = \{b\} \cap \{a, b\} = \{b\}$

Koşul 3.  $W(S2) \cap W(S3) = \{b\} \cap \{c\} = \{\}$



# Fork ve Join Yapıları

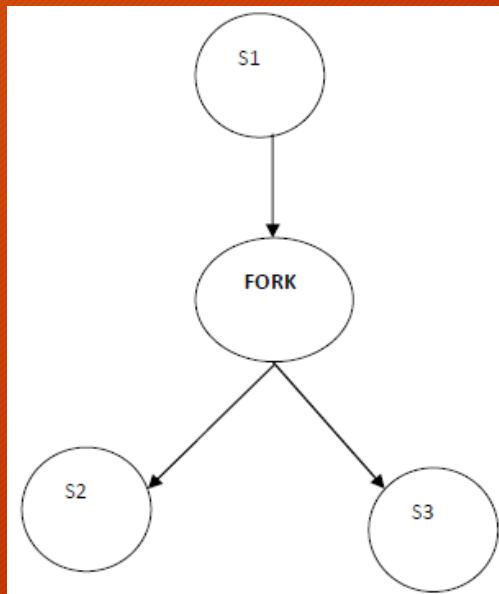
- Eş zamanlılığı tanımlayan ilk programlama dili notasyonlarından biridir.



# Fork Yapısı

- İki eş zamanlı eşleme üretir.

S1;  
**FORK L**  
S2;  
...  
...  
L:S3;



# Join Yapısı

Count:=2;

FORK L1;

...

...

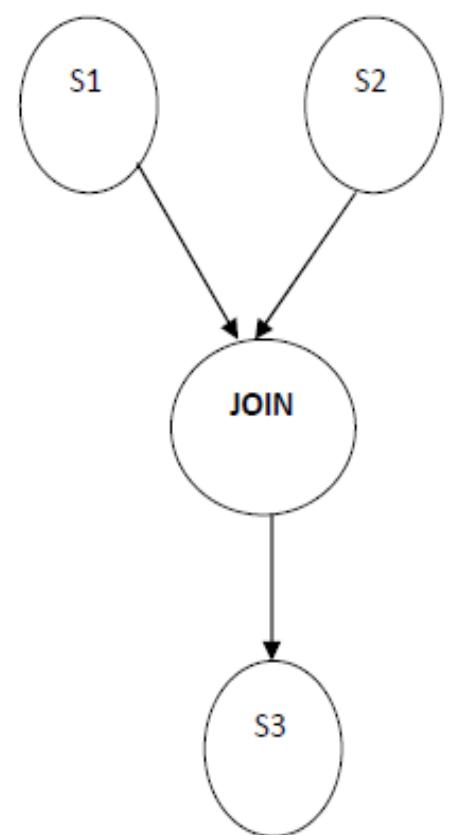
S1;

Go to L2;

L1:S2;

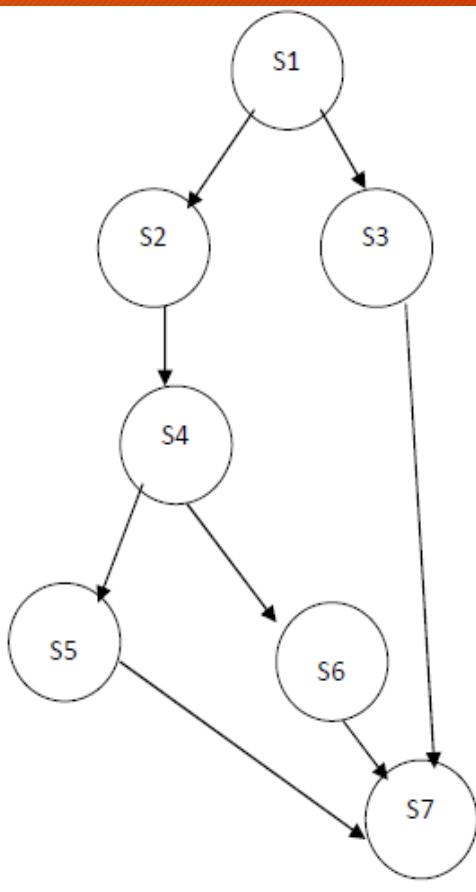
L2:JOIN count;

S3;



# Fork Join Örneği

```
S1  
Count:=3;  
FORK L1;  
S2;  
S4;  
FORK L2;  
S5;  
Goto L3;  
L2:S6;  
Goto L3;  
L1:S3;  
L3: JOIN count;  
S7;
```



# Parbegin ve Parenend Yapıları

Parbegin

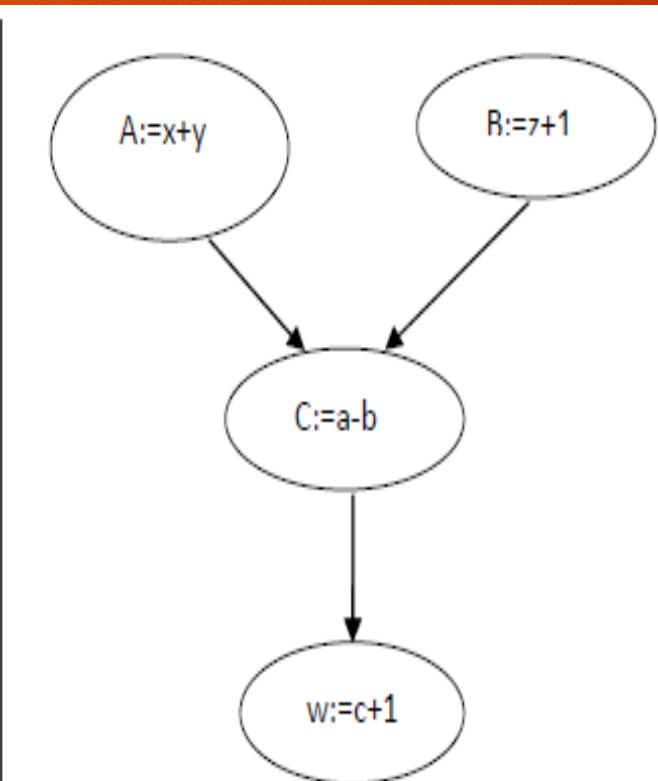
a:= x + y;

b:= z + 1;

parend;

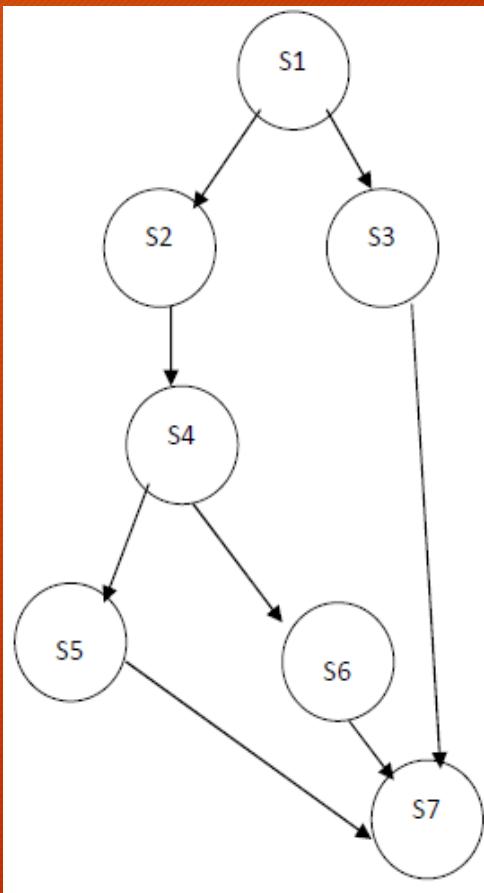
c:= a - b;

w:=c + 1;

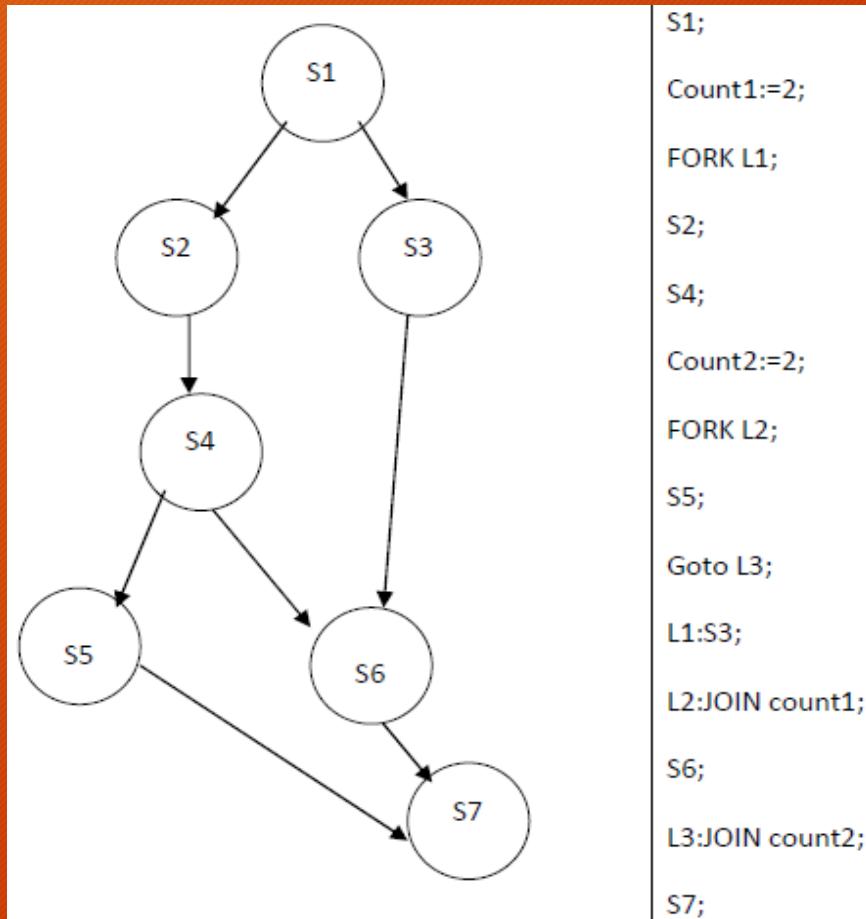


# Parbegin ve Parenend Yapıları

```
S1;  
Parbegin  
S3;  
Begin  
S2;  
S4;  
Parbegin  
S5;  
S6;  
Parenend;  
End;  
Parenend;  
S7;
```



# Fork Join ile Yapılıp Parbegin Paren'd ile Yapılamayan Örnek



# Fonksiyonel Dillerde Eş Zamanlılık

- Multi-LISP
  - 1985 Yılında tanıtılan bu dil program parçalarının eş zamanlı çalışmasına izin veriyordu.
  - pcall yapısı ile bu gerçekleştiriliyordu.

(fonk x y z)



Eş Zamanlılık olmayan normal bir fonksiyon çağrıması

(pcall fonk x y z)



Eş Zamanlılık içeren fonksiyon çağrıması

- pcall ile çağrılması x, y ve z parametrelerinin eş zamanlı çalışmasını sağlayacaktır. x y ve z parametreleri yine bir fonksiyon olabilir bu durumda bu fonksiyonlar eş zamanlı çalıştırılır.

# Thread Yield Metodu

- yield metodu geçici olarak diğer threadlere zaman verir.
- Thread.yield(); şeklinde kullanılır.
- O satıra gelen her thread bu komutu uygulayacaktır.

# Thread Sleep Metodu

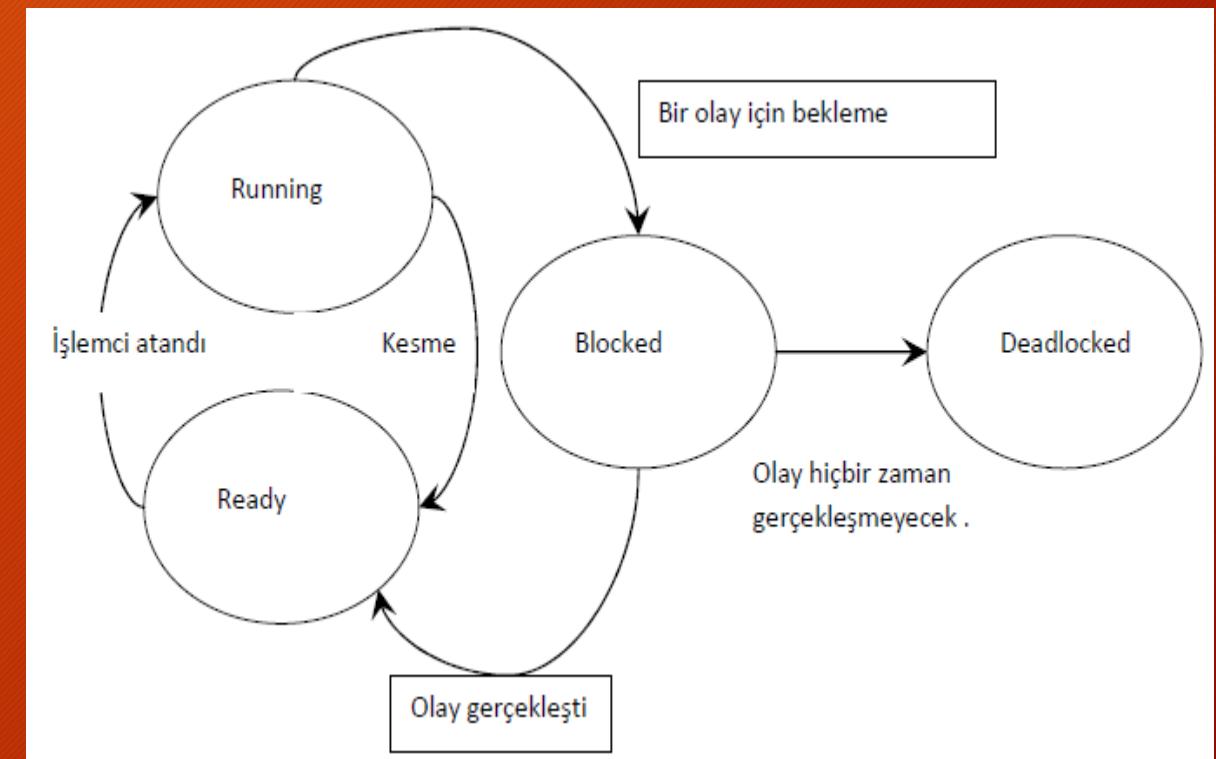
- Belirtilen süre boyunca thread uykuya geçer.
- Bu tepki süresinin uzun olduğu bazı durumlarda zorunlu olarak kullanılabilir.
- Thread.sleep(1); 1 milisaniye uykuya geçirir.
- Java dilinde sleep metodу yakalanması zorunlu bir hata fırlatma durumu olduğu için try catch bloklarında kullanımı zorunludur.

# Sleep ve Yield Metotları

- Bu metotların kullanımı için eş zamanlılık şart değildir.
- Programa atanan varsayılan thread bu komutlara denk geldiğinde işleyişi uygular.

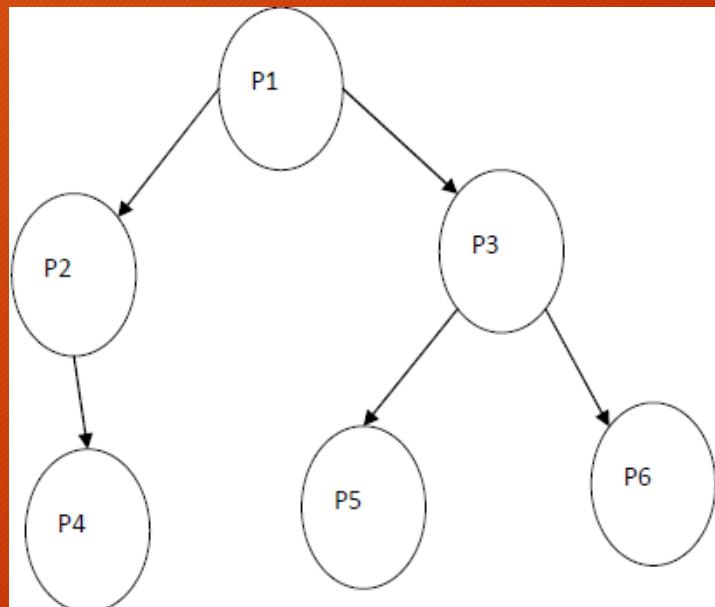
# İşlem Durumları

- Running: Komutlar işletiliyor.
- Blocked: Sistem bazı durumlar için bekletiliyor.
- Ready: İşlem bir işlemciye atanmak için hazır durumda bekletiliyor.
- Deadlock: İşlem hiç bir zaman gerçekleşmeyecek olayları bekliyor.



# İşlem Grafları

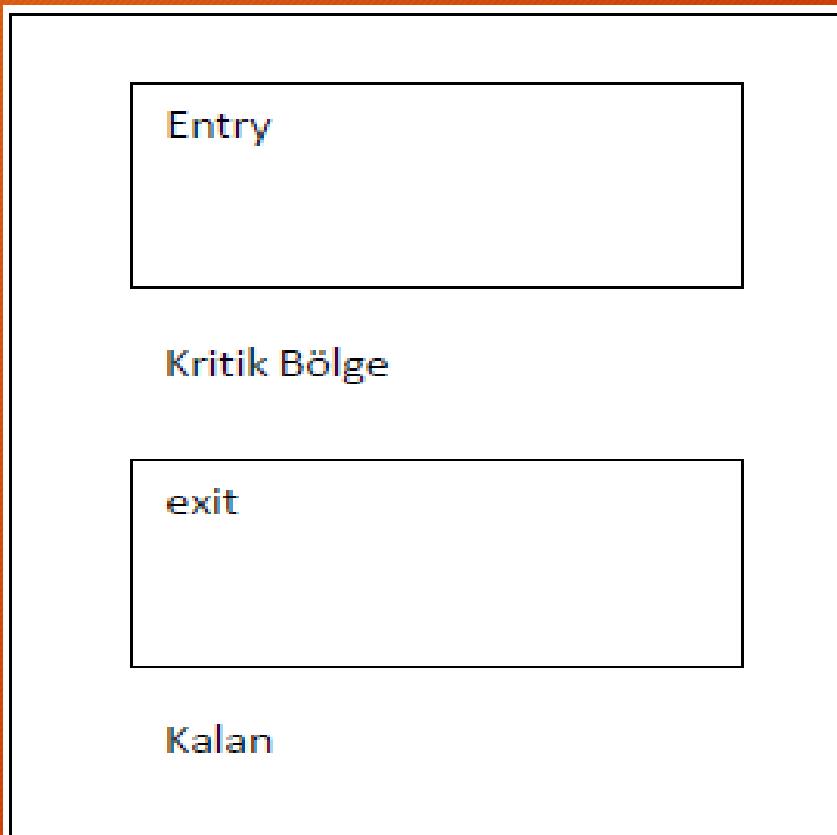
- Bir işlem grafi düğümleri işlemlere karşılık gelen yönlendirilmiş köklü bir graftır.
- $P_i$  düğümünden  $P_j$  düğümüne gelen ok işaretini  $P_i$ 'nin  $P_j$ 'yi oluşturduğunu ifade eder.



# Kritik Bölge

- Birlikte çalışan n adet işleminden {P1, P2, ..., Pn} oluşan bir sistem olduğu düşünülürse.
- Her bir işlem ortak değişkenleri okuyan bir tabloyu güncelleyen, bir dosyayı yazan vb. işlemleri içerebilir.
- Bu bölümlere kritik bölge ismi verilir.
- Bu bölgelere aynı anda sadece bir thread girmelidir.

# Kritik Bölgelenin Yapısı



# Kritik Bölge Gerçekleştirimi için Yaklaşımlar

## Örnek Algoritma 1

```
Repeat
    While turn <> i do skip;
        Kritik Bölge
        Turn=j;
        Kalan
    Until false;
```

### Analiz:

- Algoritma hangi işlemin kritik bölgесine girmesine izin verdigini hatırlar
- İşlemenin hangi aşamada olduğunu hatırlayamaz.

# Kritik Bölge Gerçekleştirimi için Yaklaşımlar

## Örnek Algoritma 2

Repeat

While flag[j] do skip;

Flag[i]=true;

Kritik Bölge

Flag[i]:=False;

Kalan

Until false;

### Analiz:

- Algoritma hangi işlemin kritik bölgесine girmesine izin verdigini hatırlar
- İşlemenin hangi aşamada olduğunu hatırlar.
- Fakat bir dizi kontrol edildiği için aynı anda birden fazla thread'in kritik bölgeye girme ihtimali vardır.

# Kritik Bölge Gerçekleştirimi için Yaklaşımlar

## Örnek Algoritma 3

```
Repeat
    flag[i]:=true;
    turn:=j;
    While (flag[j] and turn=j)do skip;

    Kritik Bölge
    Flag[i]:=False;

    Kalan
    Until false;
```

### Analiz:

- **Algoritma hangi işlemin kritik bölgesine girmesine izin verdığını hatırlar**
- **İşlemenin hangi aşamada olduğunu hatırlar.**
- **Birden fazla thread'in kritik bölgeye girmesine ihtimal yoktur.**

# Semaforlar

- Karşılıklı hariç bırakma (mutual exclusion) problemi için yapılan çözümleri daha komplex problemler için genelleştirmek kolay değildir.
- Bu zorluğun üstesinden gelebilmek için semaforlar olarak adlandırılan bir senkronizasyon aracı kullanılabilir.

# Semaforlar

Repeat

P(Mutex)

Kritik Bölge

V(Mutex)

Kalan

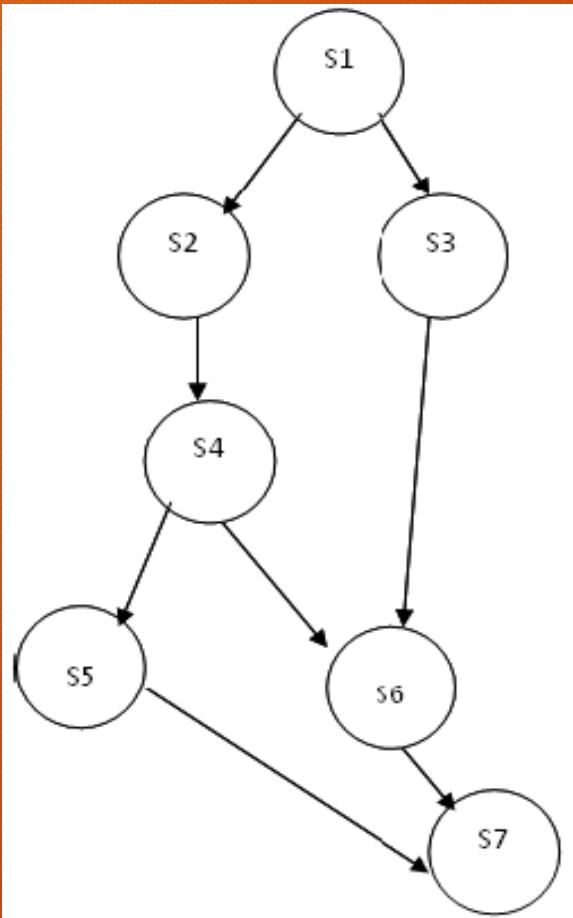
Until false;

S1;  
V(synch);  
P(synch);  
S2;

P1 işlemine konulur.

P2 işlemine konulur.

# Semaforların Örnek Üzerinde Kullanımı



```
Var a, b, c, d, e, f, g: semaphore;  
Begin  
Parbegin  
Begin S1; V(a); V(b) end;  
Begin P(a); S2; S4; V(c); V(d); end;  
Begin P(b); S3; V(e); end;  
Begin P( c); S5; V(f); end;  
Begin P(d); P(e); S6; V(g); end;  
Begin P(f); P(g); S7; end  
Parend;  
End;
```

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016

# Programlama Dillerinin Prensipleri

## Hafta 13 - Fonksiyonel Programlama

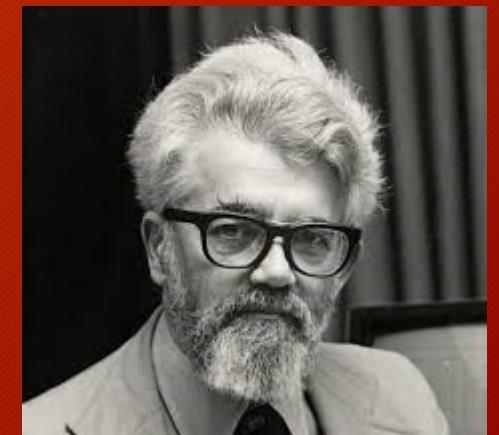
Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Tarihsel gelişim
- Tanım
- Neden fonksiyonel paradigma?
- Yaklaşım
- Soy ağacı
- Fonksiyonel dillerin yapısı
- Değişkenin rolü
- Haskell dili
- Lisp dili
  - Formlar
  - Veri türleri
- Fonksiyonel ile Emir Esaslı karşılaştırılması

# Tarihsel Gelişim

- Fonksiyonel tasarım ilk John McCarty tarafından 1956 yılında tanıtılmıştır.
- En güçlü temsilcisi Lisp dilidir. Bu isim güçlü liste işlemleri yapabilmesinden gelir.



John McCarthy ( 1927 - 2011 )

# Tanım

- Fonksiyonel dillerin tasarımı Matematiksel Fonksiyonlara dayalıdır ve değişkenler(variables), matematikte olduğu gibi gereklidir.
- Kullanıcıya yakın olan sağlam bir teorik temele sahiptir.
- Fonksiyonel programlamada , bir fonksiyon aynı parametreler verildiğinde daima aynı sonucu üretir (referential transparency).

# Neden Fonksiyonel Paradigma?

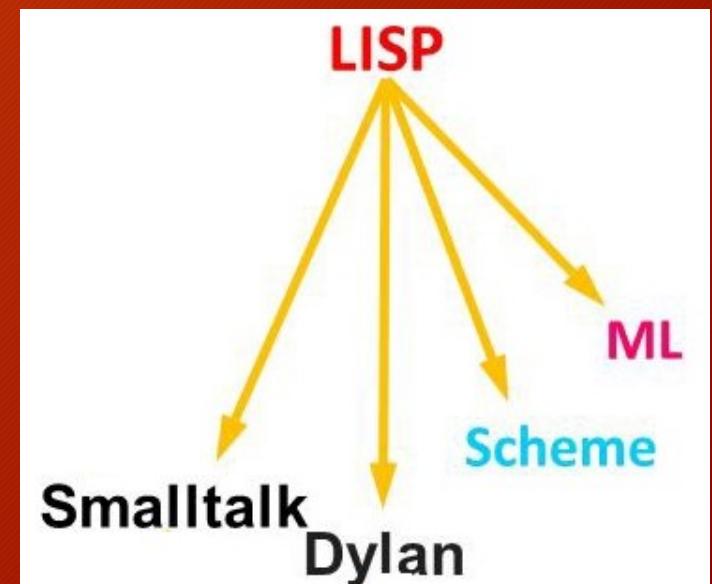
- Tarihsel süreçte emir esaslı tasarımdan sonra tanıtılmıştır.
- Emir esaslı dillerin tasarımı doğrudan doğruya von Neumann mimarisine dayanır.
- Bir emir esaslı dilde, işlemler yapılır ve sonuçlar daha sonra kullanım için değişkenlerde(variables) tutulur. Emir esaslı dillerde değişkenlerin yönetimi karmaşıklığa yol açar.

# Yaklaşım

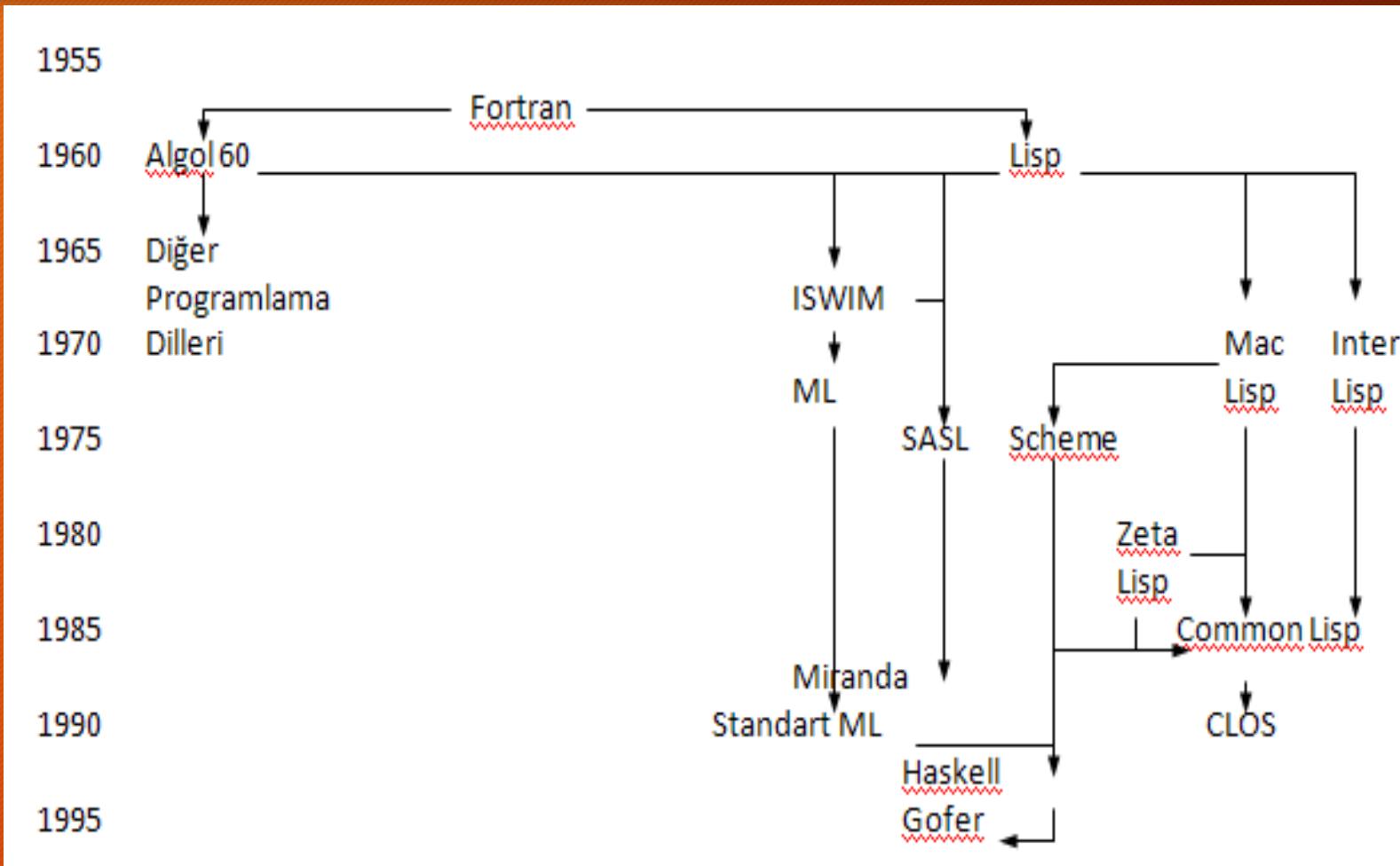
- Fonksiyonel dillerde problemin nasıl çözüleceğinden çok problemin ne olduğu önemlidir.
- For, if, while gibi denetim mekanizmaları makrolar halinde sunulur ve özyineleme ile gerçekleştirilir. Daha çok yapay zeka ve benzetim uygulamaları için uygun olabilir.
- Fonksiyon yaklaşımından dolayı matematik temeli oldukça sağlam olacağından optimize edilme (en iyileme) şansı çok yüksektir.

# Yaklaşım devam...

- Fonksiyonel programlama paradigması, Programlama dilini fonksiyon tanımının temel biçimleri üzerine oturtarak, algoritmaların ifadesi için basit ve açık bir ortam elde etmeyi amaçlamıştır.
- İlk örnek LISP dilidir ve onu ML, Scheme ve Haskell, dilleri izlemiştir.



# Soy Ağacı



# Fonksiyonel Dillerin Yapısı

- Sadece fonksiyonlar üzerine kurulmuş bir modeldir.
- Fonksiyonlar bir çok değer alır ve geriye sadece bir değer döndürürler.
- Fonksiyonlar başka fonksiyonları çağrıır ya da başka fonksiyonun parametresi olurlar.  
`Fonskiyon(..(fonksiyon2(fonksiyon1(veriler))))..)`
- Bu dillerde, alt yordamlar,fonksiyonlar (prosedürler) kullanılarak program daha alt parçalara bölünür.

# Fonksiyonel Dillerin Yapısı devam...

- Fonksiyonel diller Sembolik veri işleme amacı ile dizayn edilmiştir.
- Bu diller;
  - Türev ve integral hesaplamalarındaki
  - Elektrik devre teorisideki
  - Matematiksel mantık oyunlarındaki
  - Yapay zekanın diğer alanlarındaki
- sebolik hesaplamalarda kullanılmaktadır.
- Karmaşık hesaplamalar daha basit ifadeler cinsinden yazılarak kolaylıkla çözümlenebilir.

# Değişkenin Rolü

- Fonksiyonel olmayan tasarımlarda değişken, bir değeri tutan yer rolünü üstlenirken fonksiyonel tasarımda direkt değerin kendisidir.

$x = x + 1$  ifadesinde her  $x$  farklı bir değeri temsil eder.

$10 = 9 + 1$  deki gibi düşünülebilir.

# Haskell Dili

- Bağımlı ve bağımsız değişkenlerin tespit edilip hangi işlerin eş zamanlı çalıştırılabileceği belirlenir.
- Tam olarak fonksiyonel bir dildir. (değişkenler yoktur, atama ifadeleri yoktur, hiçbir çeşit yan etki yoktur).
- Tembel değerlendirme(lazy evaluation) kullanır (değer gerekmediği sürece hiçbir alt-ifadeyi değerlendirme)
- Liste kapsamları(list comprehensions), sonsuz listelerle çalışabilmeye izin verir.

# Lisp Dili Formları

- ANSI Common Lisp (cLisp)
  - Derleyici, yorumlayıcı, debugger içerir
- GNU Common Lisp (gcl)
  - Derleyici, yorumlayıcı içerir
- Allegro CL (Commercial Common Lisp Implementation)

# Lisp Dili Veri Türleri

- İki ana veri türünden oluşur.
  - Atom ve List
- Atom Veri Türü
  - String
  - Tam ve Ondalık sayılar
  - Karmaşık sayılar

# Fonksiyonel ile Emir Esaslı Tasarım Karşılaştırması

Emir Esaslı (imperative) diller	Fonksiyonel diller
Verimli çalışma	Verimsiz çalışma
Karmaşık semantik	Basit semantik
Karmaşık sentaks	Basit sentaks
Eş Zamanlılık (kullanıcı tanımlı)	Eş Zamanlılık (Otomatik)

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016

# Programlama Dillerinin Prensipleri

## Hafta 14 - Mantıksal Programlama

Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Tarihsel gelişim
- Tanım
- Prolog dili ve yapısı
- Prolog bellek yönetimi
- Mantıksal konsept
- Olaylar
- Kurallar
- Önermeler
- Prolog dilinde değişken ve sabit kavramı
- Eşitlik kavramı
- Durum oluşturma
- Kararlılık
- Geriye Zincir kuralı
- Detaylı Prolog örneği

# Tarihsel Gelişim

- Mantıksal programlamanın tek temsilcisi Prolog dilidir.
- 1970'li yıllarda Fransa, Aix-Marseille üniversitesinden Alain Colmerauer ve grubu tarafından tanıtılmıştır.
- Prolog Fransızca **Programmation en Logique** kelimesinden gelmektedir.
- Mantığın doğrudan doğruya bir bilgisayar dili olarak kullanılmasını sağlar.



Alain Colmerauer ( 1941 - 2017 )

# Tanım

- Mantıksal programlamada bilgisayarın belirli bir problemi çözebilmesi için programa problem ve çözüm yoluyla ilgili bilgi verilmesi gereklidir.
- Mantıksal programlamada döngü, alt program, seçmeli yapı gibi kontrol blokları yerine bildirme esaslı (declarative) bir yapı kullanılır.



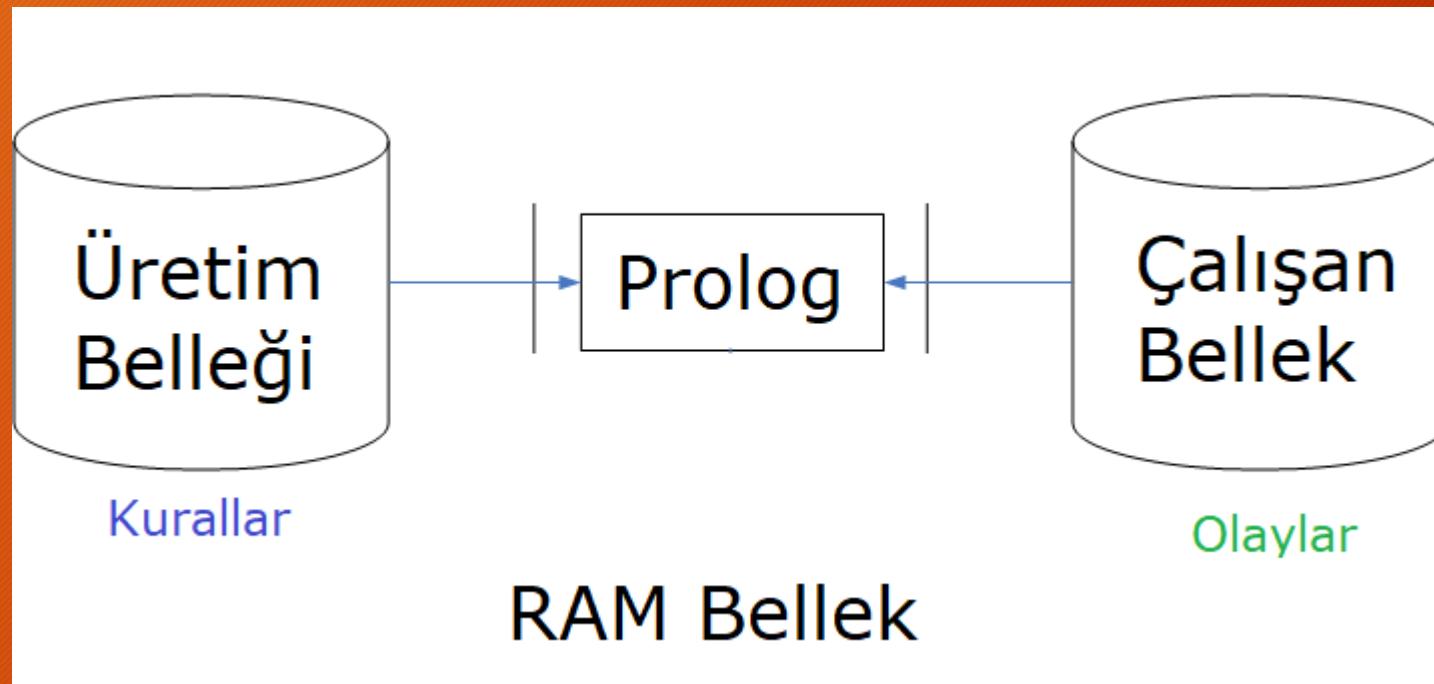
# Prolog Dili ve Yapısı

- Mantıksal programlama kural tabanlı sisteme benzer.
- Kurallar basit if kontrolleri ile gerçekleşir.
- Programın gövdesini olayların gerçekleşmesi oluşturur.
- Prolog birçok kusur içermesine rağmen büyük problemleri az programlama bilgisi ile çözmeyi sağlar.

# Prolog Bellek Yönetimi

- Kural tabanlı sistemlerde programlama dili, belleği iki bölgede yönetir.
  - Üretim Belleği
  - Çalışan Bellek

# Prolog Bellek Yönetimi



# Mantıksal Konsept

- Mantıksal programlamada bildirme esaslı yapı kullanıldığı için dili iki temel konsept oluşturur.
  - Olaylar
  - Kurallar



# Olaylar

- Doğru olan durumlar olayları oluşturur.
- Mantıksal programlama, bir durumun doğru olduğu ispat etmek için kullanılır.
- Kişi ve Yiyecek değişkenleri üzerindeki kısıtlamalar

Kişi	Yiyecek	
Ahmet	Döner	 Yemekyer(Ahmet,Döner)
Ayşe	Sarma	
Mehmet	Pilav	
Kadir	Köfte	
Hamza	Makarna	
Hüma	Balık	

# Kurallar

- Basit if yapıları kuralları oluşturur.
- Kurallar ifade edilmek için önermelerden faydalananır.
  - Atomik Önermeler
  - Bileşik Önermeler

# Önermeler

- Önermeler sonuçları doğru olanlar ya da ispat edilmesi gerekenlerdir.
- Yanlış olan durumlar önerme olarak eklenmez.
- Örneğin Eksi(X) ifadesi
  - X bir sayı ise onun negatif olduğunu söyleyen bir önermedir.
  - X çoğunluğu belirtiyorsa onu eksiltmeyi ifade edecktir.Her iki durumda da önerme doğrudur.

# Prolog Dilinde Değişken ve Sabit Kavramı

- Büyük harf veya alt çizgi ile başlayanlar değişkendir.
- Küçük harf ya da tek tırnak içinde olanlar sabitlerdir.

Değişken: Kişi, Pilav vb.

Sabit: yemekyer(Ahmet, Pilav)

# Eşitlik Kavramı

- Emir Esaslı ve Nesne Yönelimli dillerde eşitlik değişkenin içерdiği değerin aynı olma durumudur.
- Mantıksal Programlamada eşitlik her iki ifadenin aynı doğruluk tablosuna sahip olmaları şeklinde yorumlanır.

P	Q	$\neg P$	$P \rightarrow Q$	$\neg P \wedge (P \rightarrow Q)$
T	T	F	T	F
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

□

# Eşitlik Kavramı

- Mantıksal olmayan dillerde boolean türündeki iki değişken x ve y eşit olması için x doğru ise y doğru olmalı ya da x yanlış ise y yanlış olmalıdır.
- Mantıksal programlamada X doğru ise Y doğru önermesinde, X yanlış ise Y yanlış olmak zorunda değildir.

# Durum Oluşturma

- Murat'ın patronu Mehmet'tir.

patron(Mehmet,Murat)



$\forall a \text{ if } \text{çalışır}(a, X) \text{ then } \text{patron}(Mehmet, a)$

- Murat X şirketinde çalışmaktadır.

çalışır(Murat,X)

# Kararlılık

$\text{patron}(\text{Mehmet}, \text{Kemal}) = \text{patron}(\text{Mehmet}, \text{Kemal})$  ifadesi doğrudur. Ama Mantıksal programlama bunu yapmaz. Böyle bir çıkışında bulunamaz.

- Mantıksal programmanın karar verebilmesi için **Geriye Zincir Kuralı** olmalıdır.

if X then H

if H then P

if P then Y



Y doğru ise X doğrudur. Prolog bunu hesaplayabilir.

# GerİYE ZİNCİR KURALI

- Her kısıtlamanın 4 portu vardır.
- Başarısız portlar yinele portlarını geri besler.
- Çağrım başarılı olduqua bir sonraki kısıtlamaya ilerlenecektir.
- Başarısızlık ortaya çıkarsa geriye doğru hareket eder.



# Detaylı Prolog Örneği

```
kanişköpeği(Karabaş)  
avköpeği(İnceKuyruk)  
teriercinsi(Afacan)
```

köpek(X) ⊂ kanişköpeği(X)	// Kanişler bir köpektir
köpek(X) ⊂ avköpeği (X)	// Av köpekleri bir köpektir
köpek(X) ⊂ teriercinsi(X)	// Terier bir köpektir
küçük(X) ⊂ kanişköpeği(X)	// Kaniş köpekleri küçüktür
küçük(X) ⊂ teriercinsi(X)	// Terier cinsi küçüktür.
büyük(X) ⊂ avköpeği(X)	// Av köpekleri büyüktür.
evcil(X) ⊂ köpek(X)	// Köpekler evcildir
eviçievcil(X) ⊂ evcil(X) and küçük(X)	// Küçük evciller ev içi evcildir.
evdışievcil(X) ⊂ evcil(X) and büyük(X)	// Büyük evciller ev dışı evcildir.

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016