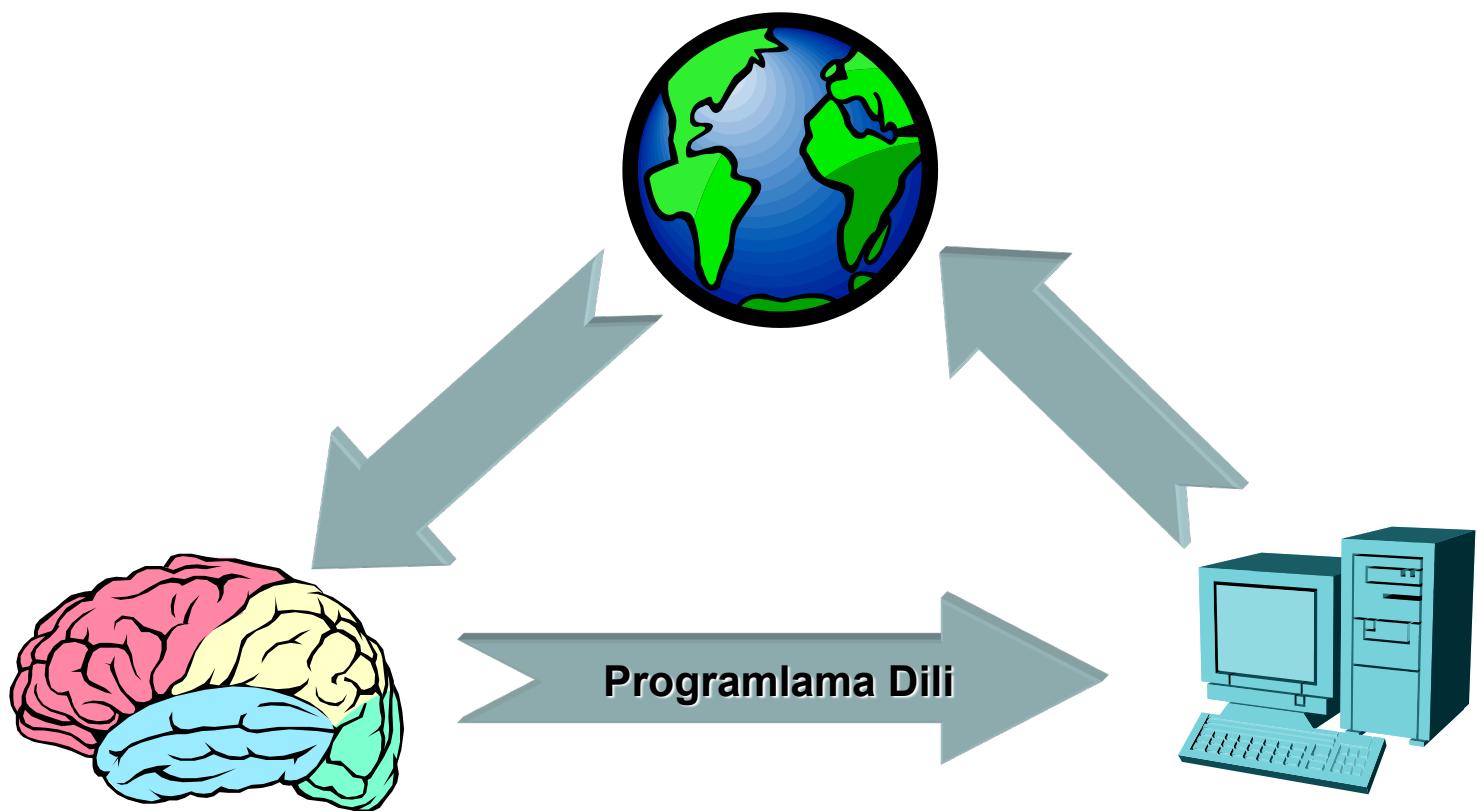


Bölüm Hedefi

- Programlama dillerinin tanımı,
- Dillerin kuşaklara ayrılması
- Programlama dillerinin sınıflandırılması,
- Programlama dilleri değerlendirilmesinde kullanılan ölçütler,
- Temel programlama paradigmaları

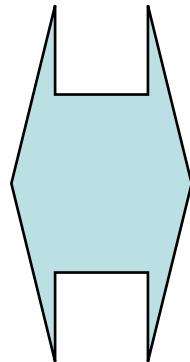
PROGRAMLAMA DİLİ NEDİR?

- **Programlama dili**, bir problemin çözümünün bilgisayardaki gerçekleştirimini ifade etmek amacıyla tasarlanmış ve o programlama dili için **hem insanlar hem de bilgisayarlar tarafından ortak olarak anlaşılacak** kurallar ve semboller dizisidir.



Programlama

```
int sum(int[] x) {  
    int sum = 0;  
    n = 0;  
    while (n < x.length) {  
        sum += x[n];  
    }  
    return sum;  
}
```



```
00101010101010  
10101011111010  
11101010101110  
00101010101010  
...  
.
```

```
program gcd(input, output);  
var i, j: integer;  
begin  
    read(i, j);  
    while i <> j do  
        if i > j then i := i - j;  
        else j := j - i;  
    writeln(i)  
end.
```

Compilation

```
27bdffd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c  
00401825 10820008 0064082a 10200003 00000000 10000002 00832023  
00641823 1483ffff 0064082a 0c1002b2 00000000 8fbf0014 27bd0020  
03e00008 00001025
```

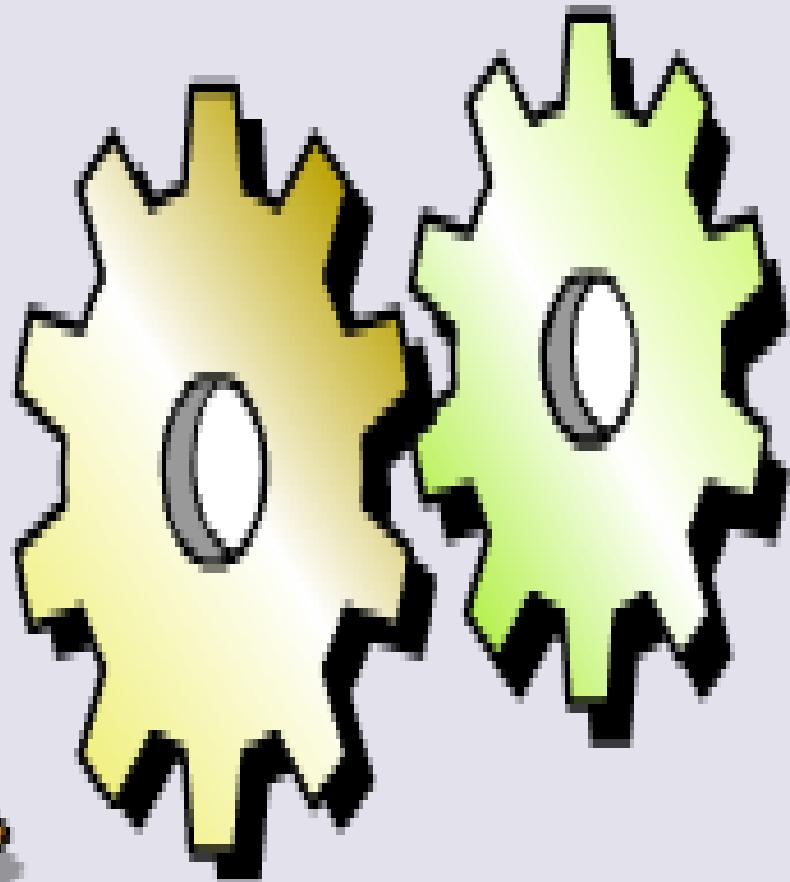
Makine Dili

- Bir programlama dilinin bilgisayar tarafından anlaşılması için, o dilin sözdiziminin ve anlamının makine diline çevrilmesi gereklidir.
- **Makine dili**, bir bilgisayarın doğrudan anladığı gösterim olup, bilgisayarların ana dili olarak nitelenebilir.

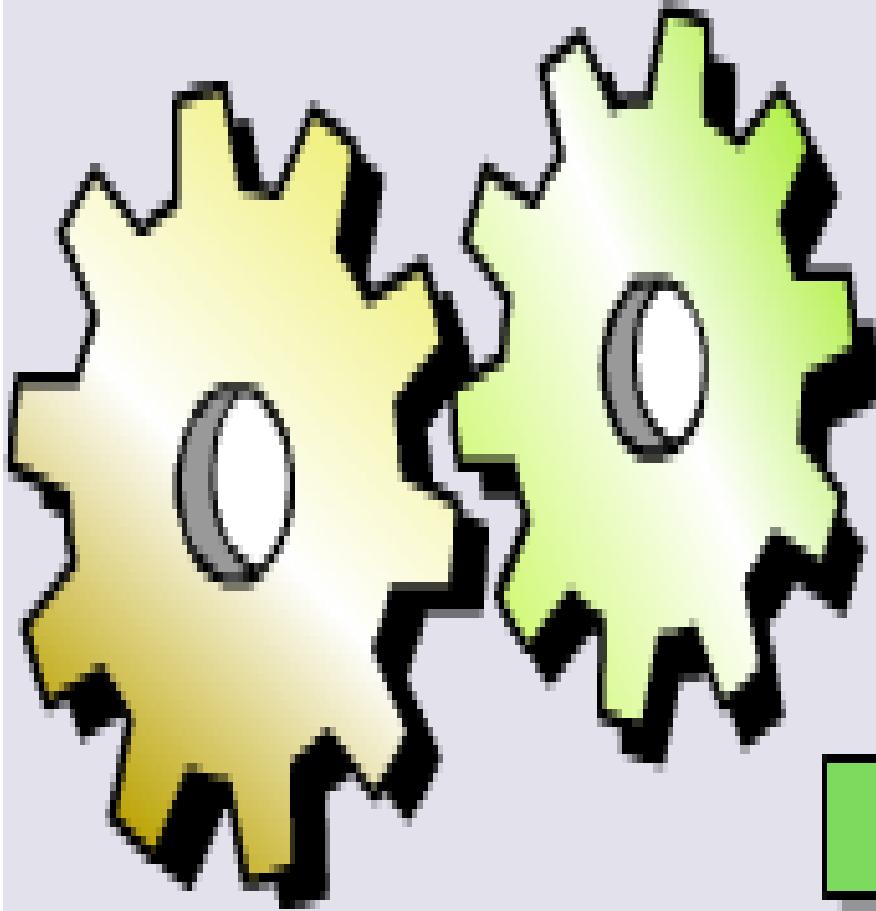


```
a function ()  
{> new array [  
]; math()  
m()_app()  
tary () < 0  
null)if else
```

PROGRAMLAMA DİLİ



DERLEME



DERLEME

00100101
10101010
01101101
10110101
10101101
01001110

MAKİNE DİLİ



Dillerin kuşaklara ayrılması

- Düşük düzeyli programlama dilleri
- Yüksek düzeyli programlama dilleri

Düşük düzeyli programlama dilleri

- Dilin özellikleri bilgisayar donanımına bağlıdır
- Makine dili ve Asembler dili
- Makine dili sadece 0 ve 1'lerden oluşur ama makinenin ana özelliklerini içerir.
- Asembly dilinde ise işlemler, değerler ve bellek yerlerinin yerini isimler ve bazı semboller almıştır. (ADD R1 24)

Yüksek düzeyli programlama dilleri

- Bu diller bilgisayar donanımına bağlı özellikler taşımaz
- Okunabilir bir gösterim vardır
- Donanımdan bağımsızdır.
- Program kütüphaneleri sağlanmaktadır.

PROGRAMLAMA DİLLERİNİN GELİŞİMİ

- Programlama dili tasarım ve gerçekleştirimleri, 1950'li yıllarda tanıtılan ilk yüksek düzeyli diller olan FORTRAN, COBOL ve LISP'den beri sürekli olarak gelişmiştir.
- Günümüzde hızla değişen bilgisayar teknolojileri, yeni gereksinimleri ortaya çıkarmaktadır. Bunun sonucu olarak, gelecekte de yeni programlama dillerinin geliştirilmesi kaçınılmazdır.

Örnekler

- FORTRAN
- COBOL
- ALGOL
- PASCAL
- BASIC
- ADA
- JAVA
- C. C++, C#
- PROLOG

Programlama Dillerinin Sınıflandırılması (Seviyesine göre)

- Çok Yüksek Seviyeli Programlama Dilleri (İnsana en yakın)
VISUAL BASIC, Access....(Dekleratif Diller)
- Yüksek Seviyeli Programlama Dilleri (PASCAL, COBOL)
- Orta Seviyeli Programlama Dilleri (C, ADA)
- Alçak Seviyeli Programlama Dilleri (Sembolik Makine Dilleri)
- Makine Dilleri (Bilgisayara en yakın)

Dillerdeki seviye yükseldikçe programcının işi daha kolay hale gelirken genel olarak esneklik ve verimlilik azalmaktadır.

Tablo 1.1 Programlama dilleri

Dil	Seviye	Programlama Kolaylığı	Göreli Hızı	Gücü ve Esnekliği
Basic	Yüksek	Kolay	Yavaş	Zayıf
Cobol	Yüksek	Kolay	Yavaş	Zayıf
Pascal	Yüksek	Kısmi Kolay	Yavaş	Zayıf
Fortran	Yüksek	Orta	Orta	Orta
C	Orta	Kısmi Zor	Hızlı	İyi
Assembly	Düşük	Zor	Çok Hızlı	Çok İyi

Programlama Dillerinin Sınıflandırılması

- Programlama dilleri, **her dil grubunu diğerlerinden ayırt eden özellik** olduğu kabul edilen bir özelliğe göre sınıflandırılırlar. Bu sınıflandırmalar birbirini dışlayan sınıflar oluşturmaz ve gruplar birbirleriyle çakışabilirler.

Uygulama Alanlarına Göre

- Sayısal Uygulamalar için programlama dilleri
- Ticari Uygulamalar için programlama dilleri
- Yapay Zeka Uygulamaları için programlama dilleri
- Sistem programlama için programlama dilleri

Sayısal Uygulamalara Yönelik Programlama Dilleri

- Bilgisayarların ilk olarak kullandıkları alan sayısal uygulamaların ağırlıklı olduğu bilimsel çalışmalar olmuştur. Bu nedenle ilk geliştirilen programlama dilleri, sayısal programlama özelliklerini vurgulamışlardır.

FORTRAN

ALGOL

PL/I

BASIC

APL

SIMULA 67

PASCAL

C

ADA

Ticari Uygulamalara Yönelik Programlama Dilleri

- Ticari uygulamalardaki veri işleme, sayısal hesaplamalardan sonra ilk olarak gelişen uygulama alanıdır.
- **COBOL** (COmmon Bussiness Oriented Language) :
- A.B.D. Savunma Bakanlığı'nın, birçok şirkete birlikte İngilizce'ye yakın ve ticari uygulamalara yönelik bir dilin geliştirilmesi çalışmalarını desteklemesi sonucu, 1959 yılında COBOL tanıtılmıştır.
- COBOL dili, özellikle yoğun miktarda veri işleme kolaylıklarını sağlayan deyimleri ve yapıları nedeniyle ticari uygulamalar alanında yazılım geliştirmek için popüler olmuştur.
- COBOL, hiyerarşik veri yapıları gibi birçok yeni kavram içeren ve özellikle raporlama açısından çeşitli olanaklar sağlayan bir dildir. 1961 ve 1962'de yenilenen dil, 1968'de standartlaştırılmış ve 1984'de tekrar yenilenmiştir.

Yapay Zeka Uygulamaları İçin Programlama Dilleri

- **LISP :** LISP 1950'li yılların sonunda, *liste işleme* amaçlı fonksiyonel bir dil olarak, John McCarthy tarafından IBM 704 bilgisayarları için geliştirilmiştir. LISP, diferansiyel ve integral hesaplamalarında, sayısal mantık ve yapay zekanın diğer alanlarında sembolik hesaplamalar için kullanılmıştır. Sonraki yıllarda, birçok kez yenilenen LISP'ten başka, Scheme (1975) ve ML (1988) de LISP'i izleyen yapay zeka alanındaki fonksiyonel dillere örnektir.
- **PROLOG:** Prolog, temel denetim yapısı sembolik mantık kavramlarına dayanan özel amaçlı bir dildir. 1972 yılında tanıtılmış olan Prolog'un temel uygulama alanı, doğal dil işlemedir. Günümüze kadar Prolog, veritabanlarından uzman sistemlere kadar çeşitli uygulama alanlarında kullanılmıştır.

Sistem Programlama Dilleri

Sistem programlama dillerine en tanınmış örnek C programlama dilidir. Sistem programlaması alanında etkinlik gereksinimi nedeniyle, birleştirici dilleri yaygın olarak kullanılmıştır. 1970'li yılların başında C dilinin geliştirilmesi ile sistem programlama alanında da diğer uygulama alanlarında olduğu gibi yüksek düzeyli programlama dillerinin kullanımı yaygınlaşmıştır.

Programlama Dillerinin Sınıflandırılması (Uygulama Alanlarına göre)

- Bilimsel ve Mühendislik Dilleri:Fortran, PASCAL, C, C++
- Veritabanı Programlama Dilleri:DBASE, PARADOX, FOXPRO, SQL
- Yapay Zeka Dilleri:LISP, PROLOG
- Genel Amaçlı Diller: C, PASCAL...
- Sistem Programlama Dilleri:C, Sembolik Makine Dilleri

Dil Değerlendirme Ölçütleri

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

İFADE GÜCÜ

- Algoritmayı tasarlayan kişinin niyetlerini açık bir biçimde yansıtılmasına olanak tanıyan bir dil, ifade gücü yüksek bir dildir.
- Bir matematikçi kendi alanındaki sembollerini kullanmak isteyebilir
- C ve PASCAL dillerinin ifade gücü yüksektir.

VERİ TÜRLERİ VE YAPILARI

- Çeşitli veri türleri (tamsayı, gerçek sayı, karakter...) ve veri yapılarını (dizi, kayıt, stack ve kuyruk..) destekleme yeteneği olmalıdır.
- C ve PASCAL veri yapısı bakımından oldukça zengin dillerdir.

GİRİŞ/ÇIKIŞ KOLAYLIĞI

- Sıralı, indexli ve rastgele dosyalara erişme, veritabanı kayıtlarını geri alma, güncelleştirme ve sorgulama yeteneği olarak tanımlanabilir.
- C dili bu bakımdan zayıftır.
- Genel olarak veritabanı programları bu bakımdan güçlündür.

TAŞINABİLİRLİK

- Taşınabilirlik terimi kaynak kod için kullanılır.
- Bir programlama dilinde yazılmış kaynak kodun başka sistemlerde de sorunsuz derlenerek çalışılmasına taşınabilirlik denir.
- Seviye düştükçe taşınabilirlik azalır. Taşınabilirliği en az olan diller makine dilleridir.
- C dili taşınabilirliği en yüksek dildir.
- BASIC derleyicileri arasında büyük farklılıklar olduğundan taşınabilirliği yüksek değildir.
- Hiçbir dil için mükemmel taşınabilirlik mümkün değildir

ALTPROGRAMLAMA YETENEĞİ

- Kaynak programların altprogramlara ayrılarak parçalanabilme özelliğidir.

Altprogram kullanmanın faydaları:

- Altprogram kodu kültür
- Algılamayı daha kolay hale getirir
- Test imkanlarını arttırmır
- Kaynak kodun güncelleştirilebilirliğini
- Ve yeniden kullanılabilirliğini (reusability)
- Birden fazla kişinin program için çalışabilirliğini sağlar

VERİMLİLİK

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

OKUNABİLİRLİK

- Kaynak kodun çabuk ve kuvvetli bir biçimde algılanabilmesi anlamına gelir.
- **Okunabilirlik** güncelleştirmeyi kolay kılar
- **Okunabilirlik** birçok kişinin ortak kodlar üzerinde birlikte çalışabilmesine imkan sağlar
- **OKUNABİLİRLİK HİÇBİRŞEYE FEDA EDİLMEMELİDİR**

ESNEKLİK

- Bir programlama dilinin programcayı kısıtlamamasına **esneklik** denir.
- **Esnek bir dilde** derleme hataları daha azdır.
- **Esnek bir dilde** birçok işlem hata riskine rağmen programcı için serbest bırakılmıştır.(İyi bir programcı bunu kullanabilir, deneyimsiz ise zararı olabilir)
- C **esnek bir dildir**. Karakter türü ile tamsayı türü karşılıklı olarak birbirine atanabilir.

ÖĞRENME KOLAYLIĞI

- Her dilin öğrenme zorluğu farklıdır.
- Yüksek seviyeli dillerin öğrenimi daha kolaydır.
- BASIC kolay öğrenilebilir
- C ise öğrenimi kolay olmayan bir dildir
- C++, C# öğrenimi kolay değildir ve önkoşullar gerektirebilir

GENELLİK

- Bir dilin çok çeşitli uygulamalarda etkin olarak kullanılabilirnesine **genellik** denir.
- C, PASCAL, BASIC genel dilleri iken
- COBOL, FOXPRO, CLIPPER genelliği olmayan dillerdir.

YAPISALLIK

- Yapısal programmanın etkin olarak kullanıldığı diller yapısal dillerdir.
- Burada bloklar halinde yazım ön plandadır.
- Yoğun olarak altprogram kullanılmaktadır.
- Program akışında atlamaların yapılması okumayı ve algılamayı zorlaştırbilir.
- Altprogramlar sayesinde soyutlama söz konusudur.

Nesne Yönelimlilik

- Veri+program=nesne yapısına uygun olarak çalışan ve programların nesnelerle yapıldığı dillerdir.
- C# tamamen nesne yönelimli bir dildir.
- C dilinin nesne yönelimli programlama tekniğini destekleyen uyarlaması C++ dilidir.

Dil Seçimini etkileyen Etkenler

Karakteristik	Okunabilirlik	Yazılabilirlik	Güvenilirlik
Sadelik(Simplicity)	X	X	X
Kontrol Yapısı	X	X	X
Veri tip ve yapısı	X	X	X
Syntax tasarım	X	X	X
Soyutlama desteği		X	X
İfade gücü		X	X
Type Checking			X
Exception handling			X
Restricted aliasing			X

Programlama Paradigmaları

- Bir **paradigma**, bir grubun konuya bakış biçimini ve metodunu tanımlayan kavramsal şemadır.
- Programlama paradigmaları, bir programcının problemlere çözüm üretmesini önemli derecede etkilerler.
- Farklı paradigmalar, farklı programlama stilleri getirir ve programcıların algoritmala bakış şeklini değiştirirler.

Programlama Paradigmaları

- Emir Esaslı programlama paradigma
- Bildirim Esaslı programlama paradigma
- Nesne yönelimli paradigma
- Mantık esaslı paradigma

Imperative Paradigmayı Destekleyen Diller

- *Emir Esaslı (Imperative) paradyigma'daki* programlama dilleri işlem tabanlı olup, bir program, bir dizi işlem olarak görülür.
- Programlardaki deyimler, birbirleri ile değişkenler aracılığı ile iletişim kurar.

- Imperative diller, yaygın olarak kullanılan ilk dil grubudur ve günümüzde de yoğun olarak kullanılmaktadır.
- Imperative programlama paradigması, C, FORTRAN, PL/I, Pascal, COBOL, Ada gibi birçok dil tarafından desteklenmektedir.

FORTRAN ve Pascal, imperative paradigmayı desteklerler.

Deyim -1

Deyim -2

Deyim -3

... DEYİM

Deyim -n

Bildirim Esaslı Diller

- Burada bilgisayara bir işlemi nasıl yapacağı bildirilir.
- Veriler ve sonucu elde etmek için *veriye uygulanacak fonksiyonel dönüşümler*, paradigmının temelini oluşturur.
- Excel bildirim esaslı bir dil olarak düşünülebilir.

Nesneye Yönelik Paradigmayı Destekleyen Diller

- Nesneye yönelik programlama paradigmalarının temeli SIMULA 67 programlama dilindedir. Nesnelerin sınıf ve alt sınıflara gruplanması, nesneye yönelik programlamanın temel noktasıdır.
- Smalltalk ve Eiffel gibi diller nesneye yönelik programlama paradigmalarını desteklerler.
SIMULA 67, Smalltalk , C++ ve Java

Mantık Paradigmayı Destekleyen Diller

- Mantık programlama paradigmاسında programlama, bir işin nasıl yapılacağının belirtilmesi yerine, ne yapılması istendiğinin belirtilmesi olarak görülür.
- Mantık programlama paradigmاسını destekleyen diller, belirli bir koşulun varlığını kontrol ederek ve koşul sağlanıyorsa, uygun bir işlem gerçekleştirerek çalışırlar.
- Bu modeldeki dillere en tanınmış örnek, Prolog programlama dilidir. Mantık tabanlı bir dilin çalışması imperative bir dilin çalışmasına benzemekle birlikte, deyimler sıralı olarak işlenmez.

- Bu dillerin sözdizimi genel olarak şu şekildedir:

koşul_1 -> hareket_1

koşul_2 -> hareket_2

.....

koşul_n -> hareket_n

Paradigma-Yönelik Diller

- Bu bağlamda, bazı diller paradigma bağımsız olup, birden çok paradigmayı destekleyebilirler.
- Örneğin, C++ hem imperative hem de nesneye yönelik programların geliştirilmesini destekler.

C nasıl bir dildir?

- C orta seviyeli bir dildir
- Sistem programlama dilidir
- Algoritmik bir dildir
- Taşınabilirliği en yüksek bir dildir
- İfade gücü yüksek bir dildir
- Okunabilirliği yüksektir
- Çok esnektir
- Verimli bir dildir
- Atomik bir dildir
- Tasarım özellikleri iyi (Güçlü) bir dildir
- Eğitimi zor bir dildir
- Yapısal bir dildir

Özeti

- Bu bölümde programlama dillerinin sınıflandırılması çeşitli yönleriyle incelenmiştir.
- Programlama dillerinin değerlendirilmesinde kullanılan kriterler açıklanmıştır.
- Emir esaslı, nesneye yönelik, fonksiyonel (bildirim esaslı) ve mantık **programlama paradigmaları** açıklanmıştır.

BSM208
PROGRAMLAMA DİLLERİNİN PRENSİPLERİ

**Programlama Dillerinin
Tarihi Gelişimi
(Hafta2)**

Programlama Dillerinin Tarihi

- 1940'ların sonu ve 1950'lerin başında;
 - Programlama dillerinde; Yavaşlık, yetersiz güvenilirlik, pahalılık, sınırlı bellek ve yazılım ve mutlak adresleme problemi bulunmaktadır.
 - Dizi indeksleme ve kayan noktalı aritmetik kavramları henüz desteklenemiyor.
 - Kayan noktalı aritmetik için donanım yetersizliğinden Interpretive sistemler tolere ediliyor ve kayan noktalı aritmetik için yazılımların yürütülmesi hala çok yavaş olmakta.

1946-1950

- İşletim sistemi kavramı yok
- Bilgisayarın yapısının çok iyi bilinmesi gerekiyor.
- Programlama=Çözülecek probleme ilişkin devre tasarlama
- Dolayısıyla sadece uzmanlar programlama yapabilir.
(Elektronik Müh.)

**BU DURUM BİLGİSAYARIN YAYGIN KULLANIMI KONUSUNDA
CİDDİ BİR PROBLEM OLUŞTURUYORDU**

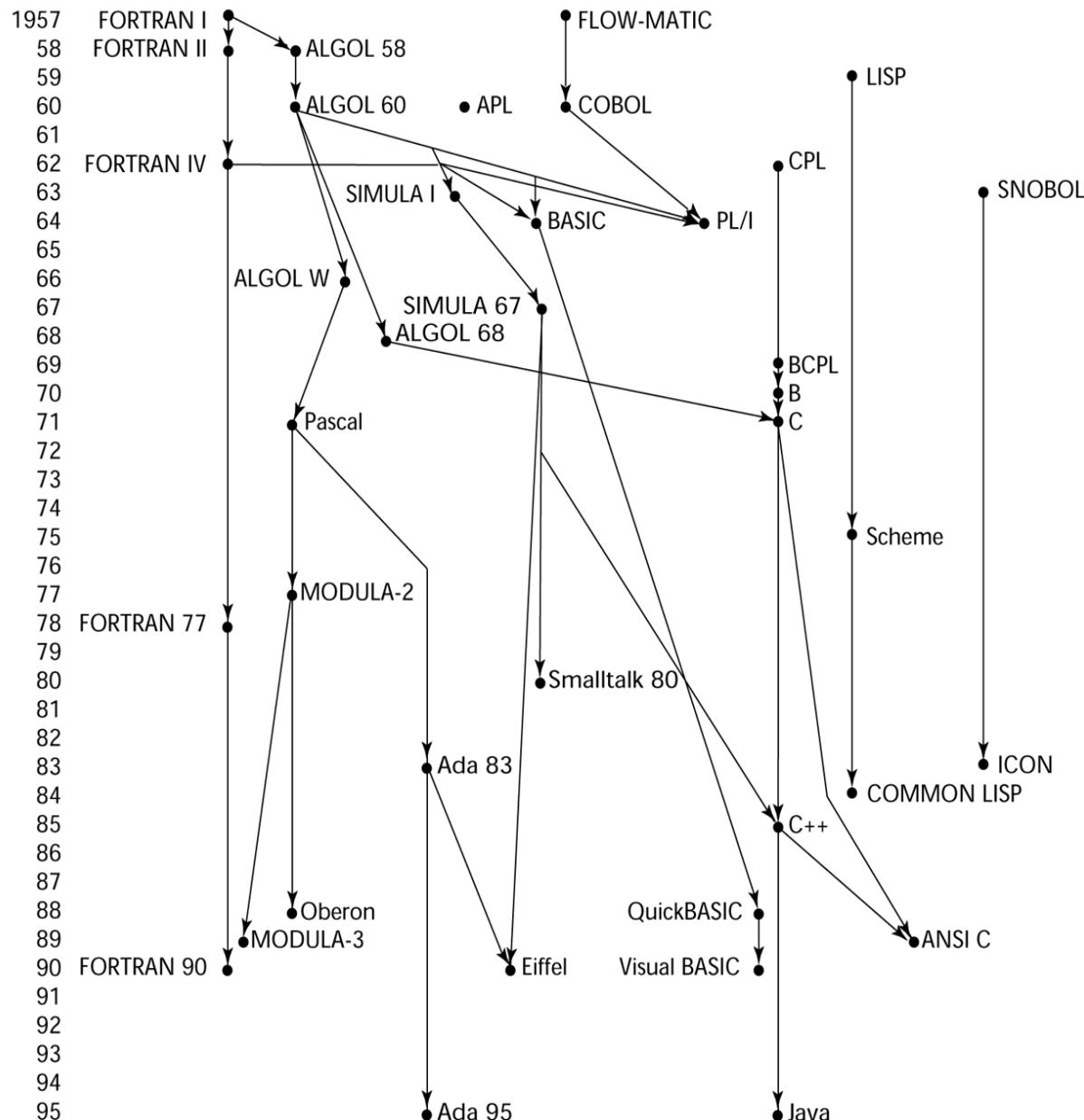
Çözüm Önerileri neydi?

- Günlük dilde kullanılan sözcüklere benzer sözcüklerden oluşan Bilgisayar dili oluşturmak
- Bu dilin anlaşılması ve öğrenilmesi kolay olmalı
- Matematiksel uygulamalar için uygun olmalı

YANI YÜKSEK SEVİYELİ BİR DİL OLMALI

Bu amaçlarla geliştirilen ilk dil FORTRAN idi

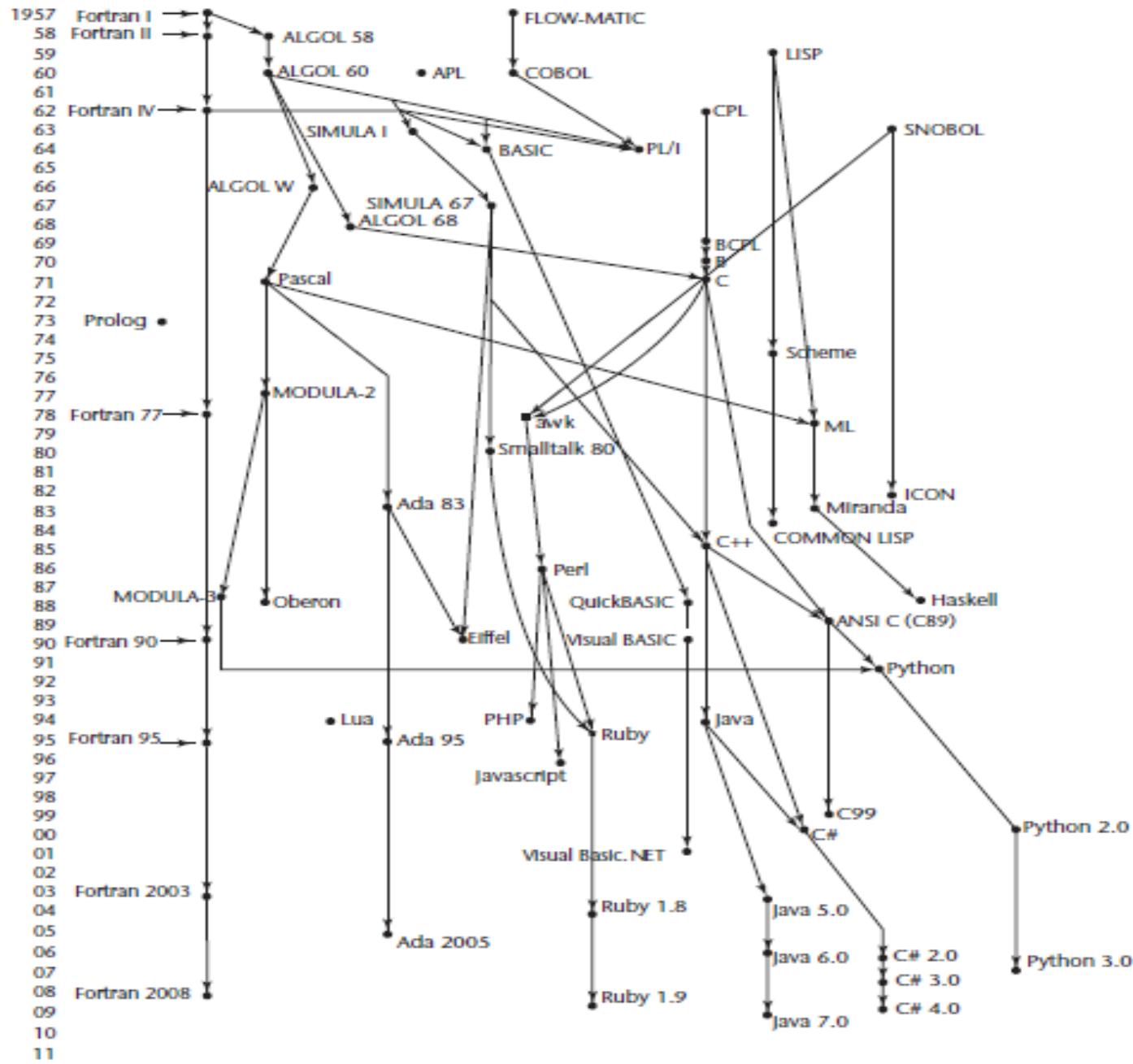
Yüksek Seviyeli Programlama Dillerinin Kronolojisi



Fortran

- FORmul TRANslating System
- 1954 yılında, IBM firmasında John Backus tarafından geliştirildi
- DO deyimleri, G/Ç deyimleri ve atama deyimleri vardı
- Matris, denklem sistemleri ve diff. Denklem sistemlerini kolaylaştıryordu
- 3 dala sahip **If** yapısı içeriyyordu:
- **If (arithmetic_expression) N1, N2, N3**
- iterative (loop) **Do** ifadesi şu biçimde kullanılıyordu:
Do N1 variable = first_value, last_value
- Çalışma sırasında veri tipleme ifadeleri ve bellek tahsis yoktu

Evolution of the Major Programming Languages



Fortran (devam)

- Fortran ingilizce sözcüklerden oluşuyordu
- Anlaşılması makine diline göre çok kolaydı
- Matematiksel uygulamalar için iyi özelliklere sahipti.
- Fakat;
- Fortran dilinin bilgisayar tarafından anlaşılması için DERLEYİCİ'ye ihtiyacı vardı.
- John Backus, 2 yılda, 51000 satırlık makine kodundan oluşan ve bir teyp biriminde saklanan Fortran derleyicisini üretmiştir.
- FORTRAN 1957 yılında ticari olarak kullanıma sunuldu
- Aynı kişi 1959 yılında, yüksek seviyeli bir dilin yazış kurallarını açıklama noktasında standart bir notasyon haline gelen **Backus Naur Form (BNF)**'yi bulmuştur.

FORTRAN ailesi

- FORTRAN I:Taşınabilirliği çok kötü
- FORTRAN 66:Karakter türü verileri işlemeye çok kısıtlı, yapısal programlama yok, rekürsif işlemler yapılamıyor.
- FORTRAN 77(American National Standards Institute-ANSI): Karakter türü verileri işleyebiliyor, Yapısal programlamayı destekliyor, DO çevrimine dışarıdan veri girilebiliyor. Taşınabilirlik daha iyi hale getiriliyor.
- FORTRAN 90:Pointer (bağlı liste gerçeklemesi, dinamik bellek yönetimi), rekürsif işlem yeteneği, bit düzeyinde işlem, Dizi yapıları daha iyi kullanılabiliyor, Altprogram yapıları daha esnek, isimler daha uzun yazılabiliyor (okunabilirlik)

FORTRAN Ailesi (devam)

- FORTRAN 95: Pointer ve yapılara varsayılan olarak ilk değer atanması ve taşınabilirliğin mükemmel hale getirilmesi temel hedef olmuştur.
- Nesneye dayalı programlama özellikleri de FORTRAN dil ailesine sunulmaya çalışılmaktadır.

UYGULAMA ALANLARI

- Bilimsel Hesaplamalar, Mühendislik Uygulamaları ve Sayısal Analiz alanlarında özellikle FORTRAN 90 etkin bir biçimde kullanılmaktadır.

Functional Programming: LISP (1958)

- LISP Dili yapay Zeka uygulamaları için, MC Carthy ve ekibi tarafından 1958 yılında geliştirilmiştir.
- Atomlar ve Listeler adı verilen iki veri yapısına sahiptir.
- Program kodu ve veriler tam olarak aynı formdadır:
- Örnek: (A B C D)
 - Bu Eğer bir veri ise , A, B, C ve D verilerini içeren bir listedir
 - Eğer bir kod ise o zaman A fonksiyonunun B, C ve D parametrelerine uygulanması olarak yorumlanır.
- İki LISP versiyonu Scheme (1970) ve COMMON LISP (1984)

LISP (devam)

- LISP esas olarak yorumlayıcı kullanan bir dildir. Derleyici kullanan versiyonları da vardır.
- Veri tiplemesi bakımından esnek bir dildir.
- Olağanüstü esneklik, ifade gücü ve **kodun aynı zamanda veri olarak da kullanılabilmesi** özelliği LISP'i yapay zaka uygulamalarında rakipsiz hale getirmiştir.
- Nesne Yönelimli Programlamayı destekler
- Veri tabanlarına erişim ve GUI olanağı vardır.
- Çok iyi belgelendirilmiş olup, COMMON LISP, ANSI tarafından standart hale getirilmiştir.

ALGOL

- ALGOrithmic Language
- FORTRAN I'den esinlenilmiştir
- İlk kez 1958 yılında Avrupalı ve Amerikalı bir komisyonun Zürih'teki çalışmaları sonucu oluşturulan yüksek seviyeli bir dildir. İlk isim ALGOL 58 olarak verilmiştir.

ALGOL 60

- GAMM(German Society for Applied Mathematics and Mechanics) tarafından ALGOL'58'e eklemeler yapılmıştır.
- Amaçlar:
 - Matematiksel notasyonlara yakın ve kolay okunabilen
 - Literatürdeki hesap süreçlerinde kullanılabilen
 - Makine diline kolaylıkla çevrilebilen bir dil olsun
- Makineden bağımsız, daha esnek ve daha güçlüğün dil olmuştur.
- Atama ifadesi olarak ilk evrensel kullanım bu dilde olmuştur.

variable := expression

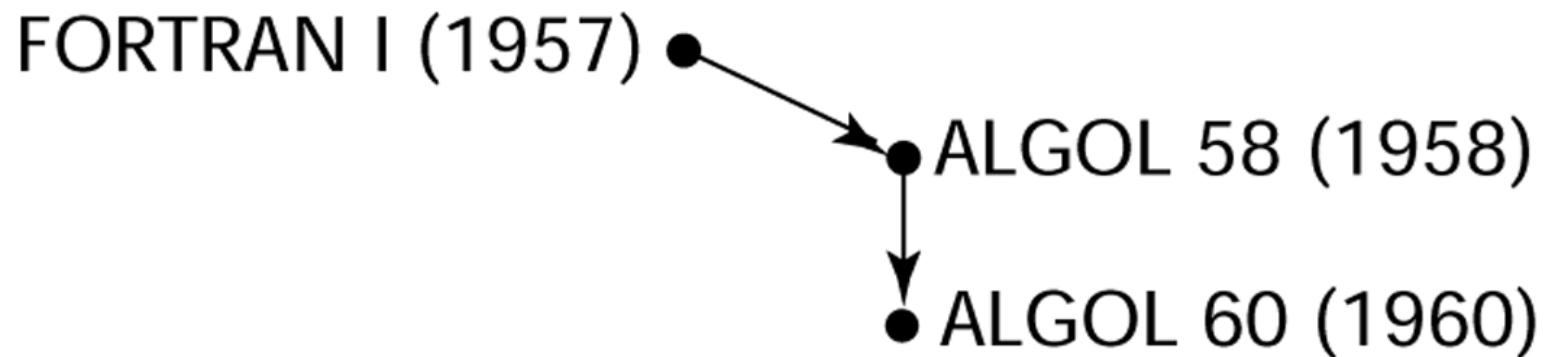
ALGOL 60: Başarılı yönler

- Blok yapısı kullanılmıştır
- Parametre aktarımı için, **pass by value ve pass by name** olmak üzere iki yöntem kullanılmıştır.
- **Rekürsiyon'a** izin verilmiştir.
- Stack-dynamic dizilere izin verilmiştir
- formal syntax tanımlı ilk dildir. Bu formal diller, Parsing teorisi ve derleyici tasarımı konuları için bir başlangıçtır.
- Rekürsiyon ve blok yapısını desteklemek için donanımda yiğita izin verildiğinden bilgisayar mimarisini etkilemiştir.

ALGOL 60: Eksik yönler

- Anlaşılmakta zorluk ve yürütmede (implementation) verimsizliğe götürürecek kadar esnek
- I/O ifadelerindeki yetersizlik yada zayıflıklar (eksiklik)
- 1960'lı yıllar için BNF(Backus Naur Form) kullanımı karmaşıklaşmış gibi gözükmeaktır.
- Avrupalı bilim adamları arasında sonderece popüler, eğitim ve araştırma aracı olarak kullanılır ve bilimsel makalelerde algoritmaları açıklamak in kullanılmasına rağmen;
- IBM tarafından desteklenmemiştir.
- Çünkü bu sırada IBM FORTRAN dilinde yazılmış zengin bir kütüphaneye sahiptir ve haliyle FORTRAN' desteklemektedir.
- O zamanlar IBM'in bilişim sektörünün %80'ine sahip olduğu düşünülürse, bu durum ALGOL60 için bir dezavantajdır.

ALGOL 60'ın şeceresi



COBOL: COmmon Business Oriented Language) **(1959-Mayıs)**

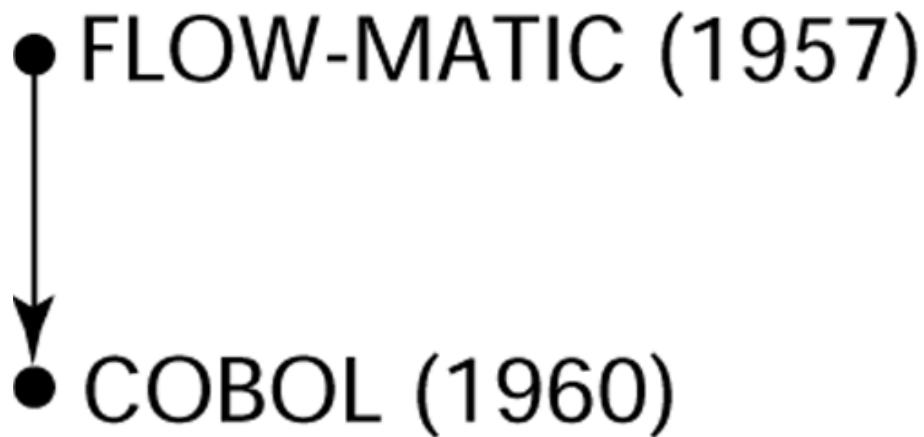
- Savunma bölümünün çalışmaları sonucu ortaya çıkmıştır.
- İngilizce sözcük yada cümle yapılarını kullanan ve İŞLETMELERe yönelik ortak bir dildir.
- Özellikle bilgisayara büyük miktarda bilgi giriş-cıkışının yapıldığı uygulamalar için geliştirilmiştir.(Stok kontrol, Bordro)
- Program içerisindeki dil ifadeleri ve fiziksel adresleri verilerin tanımlandığı ve çalıştırılabilir işlemlerin bulunduğu yer olarak ikiye ayrılır.
- Hiyerarşik veri yapıları ve yan anlamlı isimlerin (**connotative names**) görüldüğü ilk dillerdir.
- Fonksiyonları desteklememektedir.

COBOL (devam)

- COBOL içerisinde **rapor yazdırma**, **tablo içinde arama yapma**, ve kullanımı çok kolay olan **dosya sıralama rutinleri** bulunmaktadır.
- Yapısal özellikleri dolayısıyla veritabanı yönetim sistemleri ile birlikte kullanımı kolaydır.
- Tuğamiral Grace Murray Hopper, İlk önce yaklaşık 20 komuttan oluşan FLOW-MATIC'i geliştirdi ve COBOL'un temel modeli oldu.
- COBOL61-65-70-73, ANSI-COBOL versiyonları bulunmaktadır
- 1990'larda ise OOP versiyonu üretilmiştir.
- Üzerinde hala çalışmalar正在被执行中 ve Amerika hala büyük miktardaki verileri işlerken COBOL kullanmaktadır.

Figure 2.4

COBOL'un şeçeresi

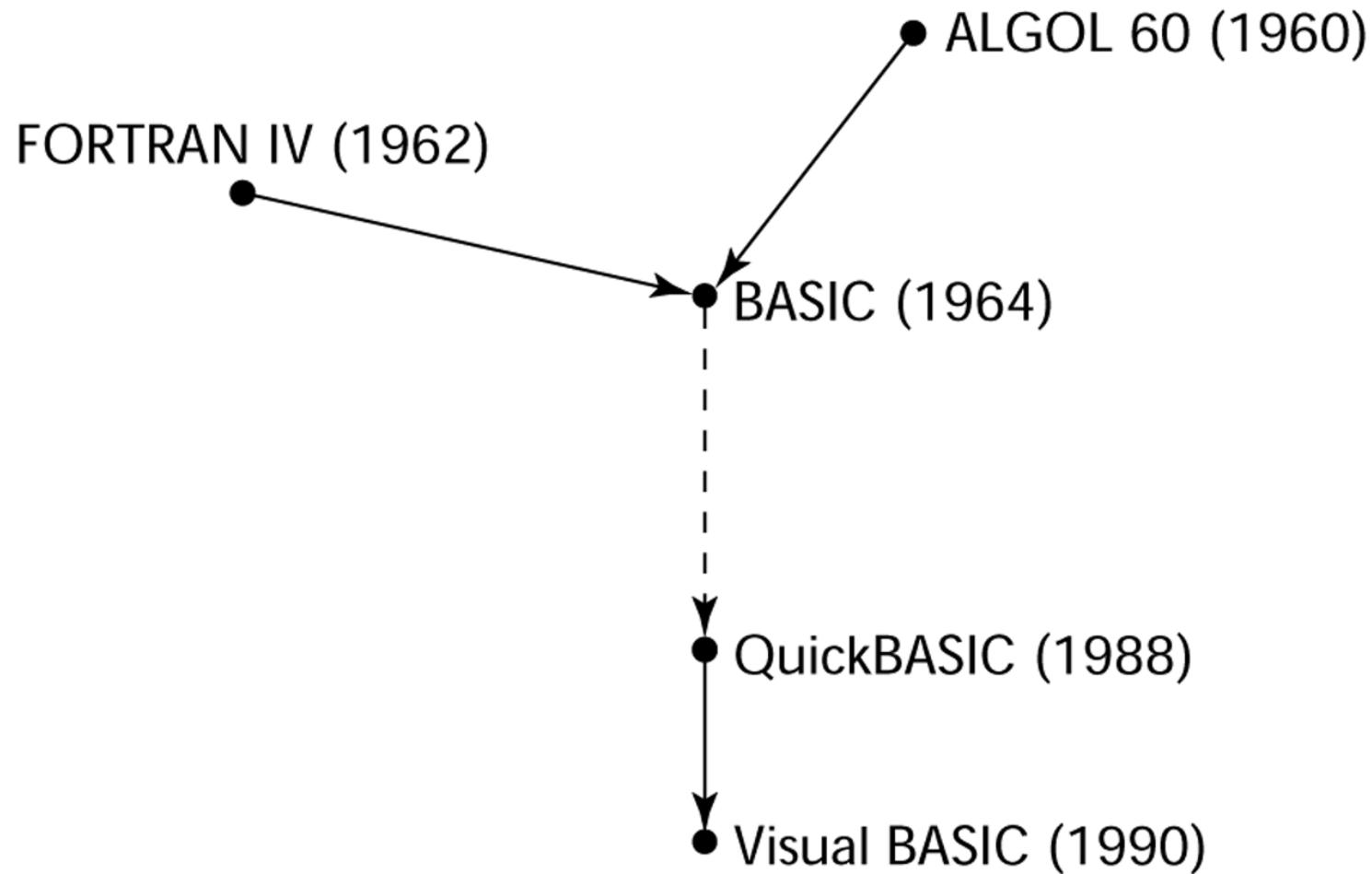


BASIC: İlk Zaman Paylaşımı Bilgisayar Sistemi (1964)

- Beginner's All-purpose Symbolic Instruction Code
- Öğrencilerin bilgisayara daha kolay erişimlerini sağlamak ve basit ve etkin bir programlama dili ile program yazabilme isteklerine cevap vermek için tasarlanmış bir dildir.
- The first PL used through terminals connecting to a remote computer
- Sadece 14 komuta (LET, PRINT, GOTO...) sahipti. Tek veri tipi (number= kayan noktalı ve tamsayı)
- BASIC, FORTRAN ve ALGOL'den bazı bileşenleri almıştır. FORTRAN'dan DO çevrimini, ALGOL'den ise "until" yerine "to"

- Kolay bir dil ve genel maksatlı, belirli bir alana bağlı değil
- Uzman kişilere de hitap edebiliyor
- Açık ve anlaşılır hata mesajlarına sahip, kullanıcı bilgisayarla etkileşimli çalışabiliyor
- Küçük boyutlu programları hızlı bir biçimde çalıştırabiliyor
- Kullanım için donanım bilgisine sahip olmaya gerek yok
- Kullanıcıyı işletim sistemi ayrıntılarından dahi koruyabiliyor
- Derleyici kullanıyor, programın tümü makine diline çevrildikten sonra icra ediliyor
- BASIC'in pekçok versiyonları olmuştur. 1989'da ise nesne yönelimli uyarlama olan Visual BASIC ve 1998'de VB6.0 sunulmuştur.

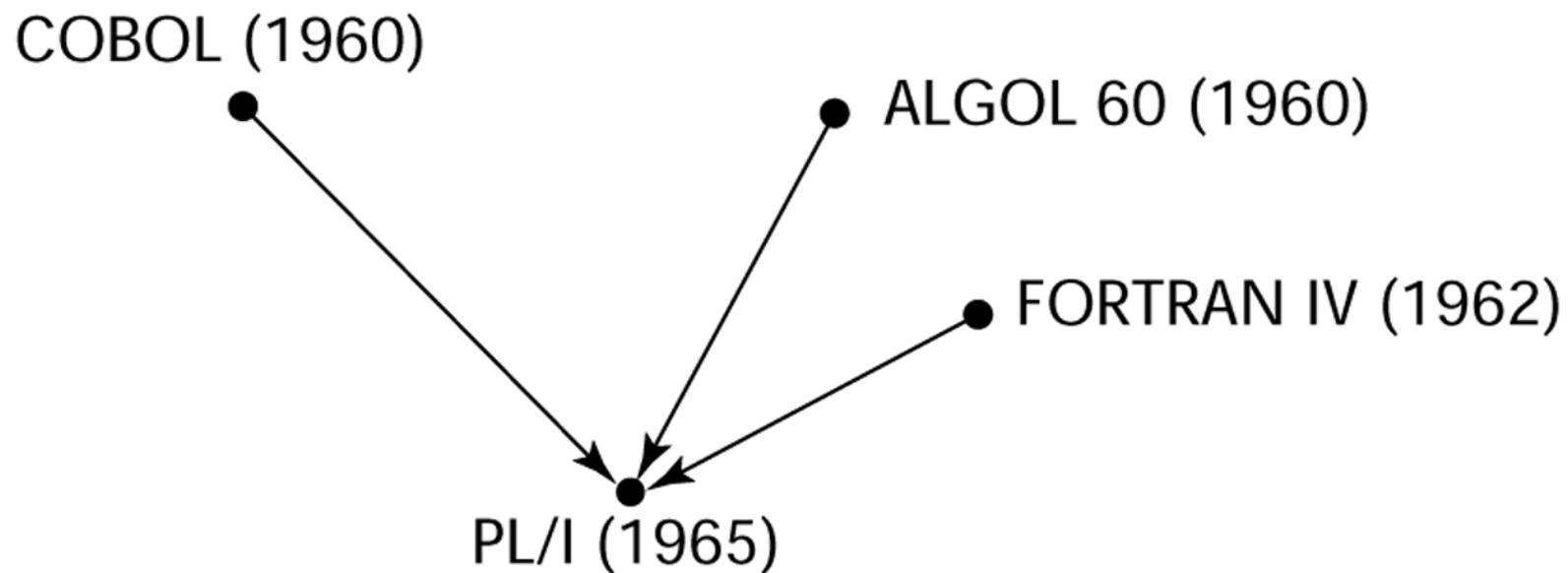
BASIC'in Şeceresi



PL/1 (1965)

- COBOL, Fortran IV ve ALGOL 60'tan sonra bilimsel ve işletme problemlerine çözüm sağlayabilmek için floating-point ve desima veritiplerini desteklemek üzere geliştirilmiş bir dildir.
- IBM ve SHARE grubunun ortak çalışmalarının bir sonucu ortaya çıkan bir üründür.
- İlk PL :
 - Eşzamanlı olarak çalışabilen altprogramlara izin verebilmektedir.
 - run-time hatalarından başka 23 farklı tip.
 - Rekürsif olarak kullanılabilecek prosedürleri desteklemektedir.
 - Pointer kullanımına izin vermektedir.
 - Bir matrisin bir satırına vektor olarak işaret mümkün
 - Hafıza gereksinimi yüzünden karmaşık olabilmektedir.
- Tasarım yönünden iyi değildir.

Figure 2.6
Genealogy of PL/I



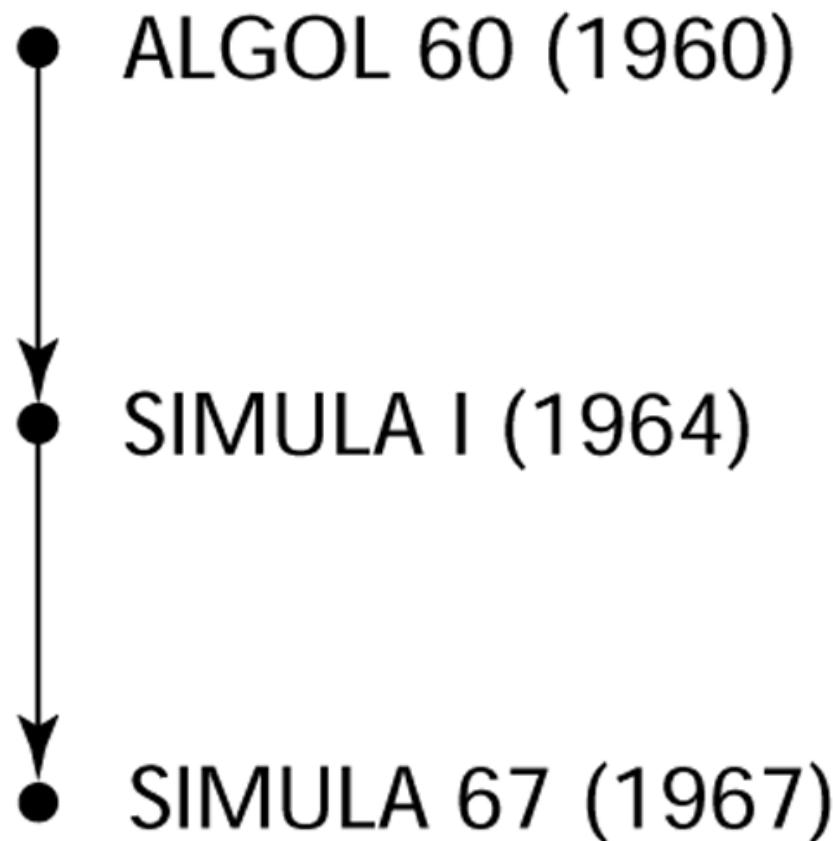
APL and SNOBOL: Dynamic Languages (1960s)

- APL (A Programming Language) dili tanımlayan bir bilgisayar mimarisi olarak IBM'de tasarlanmıştır.
- Dil ifadelerini oluşturan pek çok sayıda operatör vardır. Okuması oldukça zordur.
- SNOBOL (Snowball) metin işleyici (text processing) olarak Bell Lab'da özel olarak üretilmiştir.
- Yavaş bir dildir.
- Hem APL hem SNOBOL dinamik tipleme ve dinamik bellek tahsisini desteklemektedir.
- SNOBOL dili yorumlayıcı kullanan bir dildir.
- Veri ve değişkenler üzerinde tip bakımından bir sınırlama bulunmuyordu
- Bu dilin devamları: awk, icon, perl,

SIMULA 67: Beginnings of Data Abstraction (1962 ve 1964 arasında)

- İlk olarak simülasyon için tasarlanmış bir dildir.
- ALGOL60'ın genişletilmiş versiyonudur. (Blok yapısı ve kontrol ifadeleri buradan alınmıştır)
- Daha önce durdurulduğu yerden itibaren yeniden çalışmaya başlayan altprogramları desteklemektedir.
- Veri soyutlamasına imkan veren sınıf yapılarını desteklemektedir

SIMULA 67'in Şeceresi



ALGOL 68

- Üstünlükler
 - Kullanıcı tanımlı(user-defined) veri tiplerini destekleyen ilk PL'dir.
 - ALGOL68'de flex adı verilen dinamik dizi kavramına izin veren ilk PL'dir.
- Eksiklikler
 - Öğrenilmesi oldukça zor olan bir gramer ve dil yapısına sahiptir.
 - Sadece bilimsel uygulamalar hedeflenerek tasarlanmış bir dil

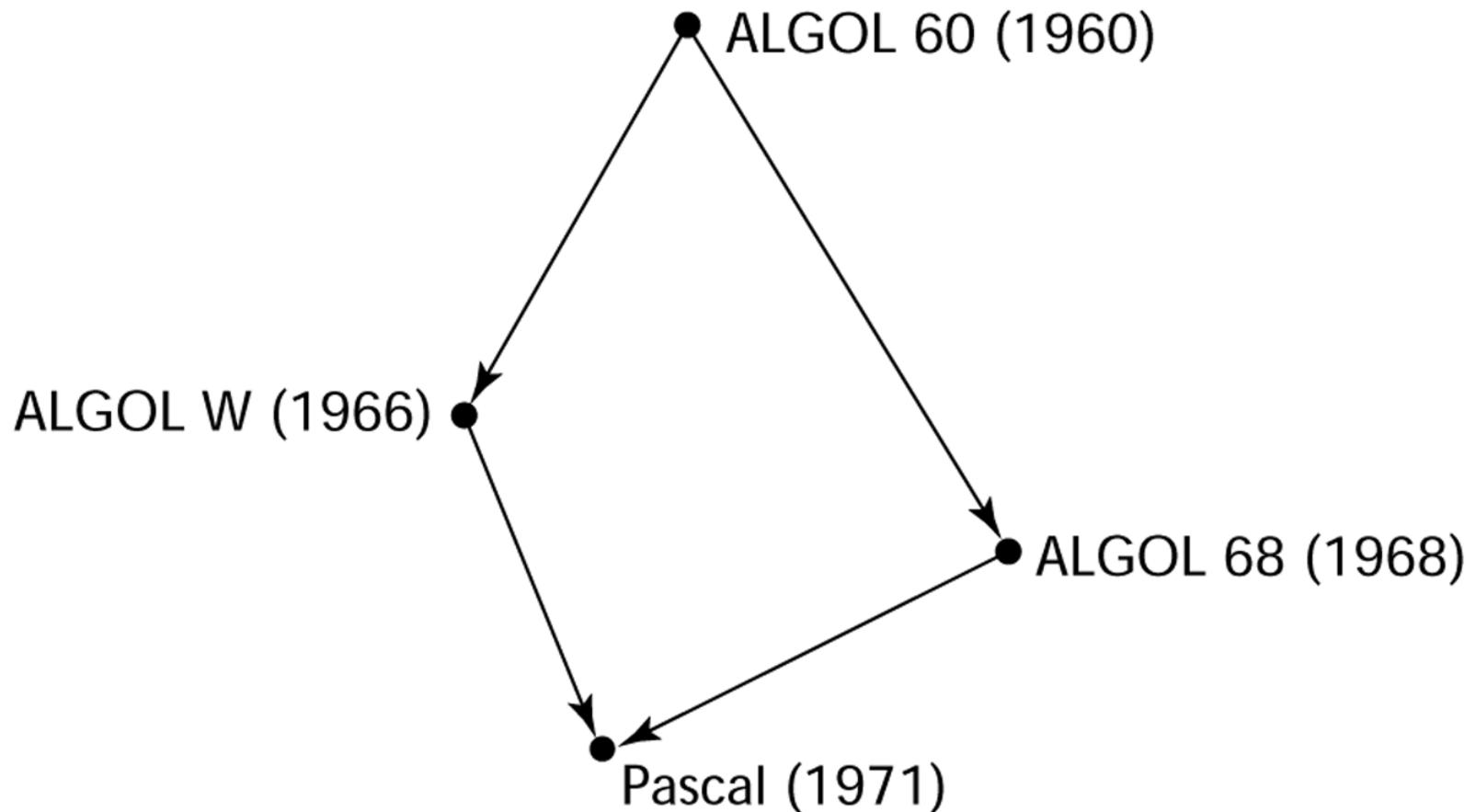
ALGOL 68'in şeceresi

- ALGOL 60 (1960)
- ALGOL 68 (1968)

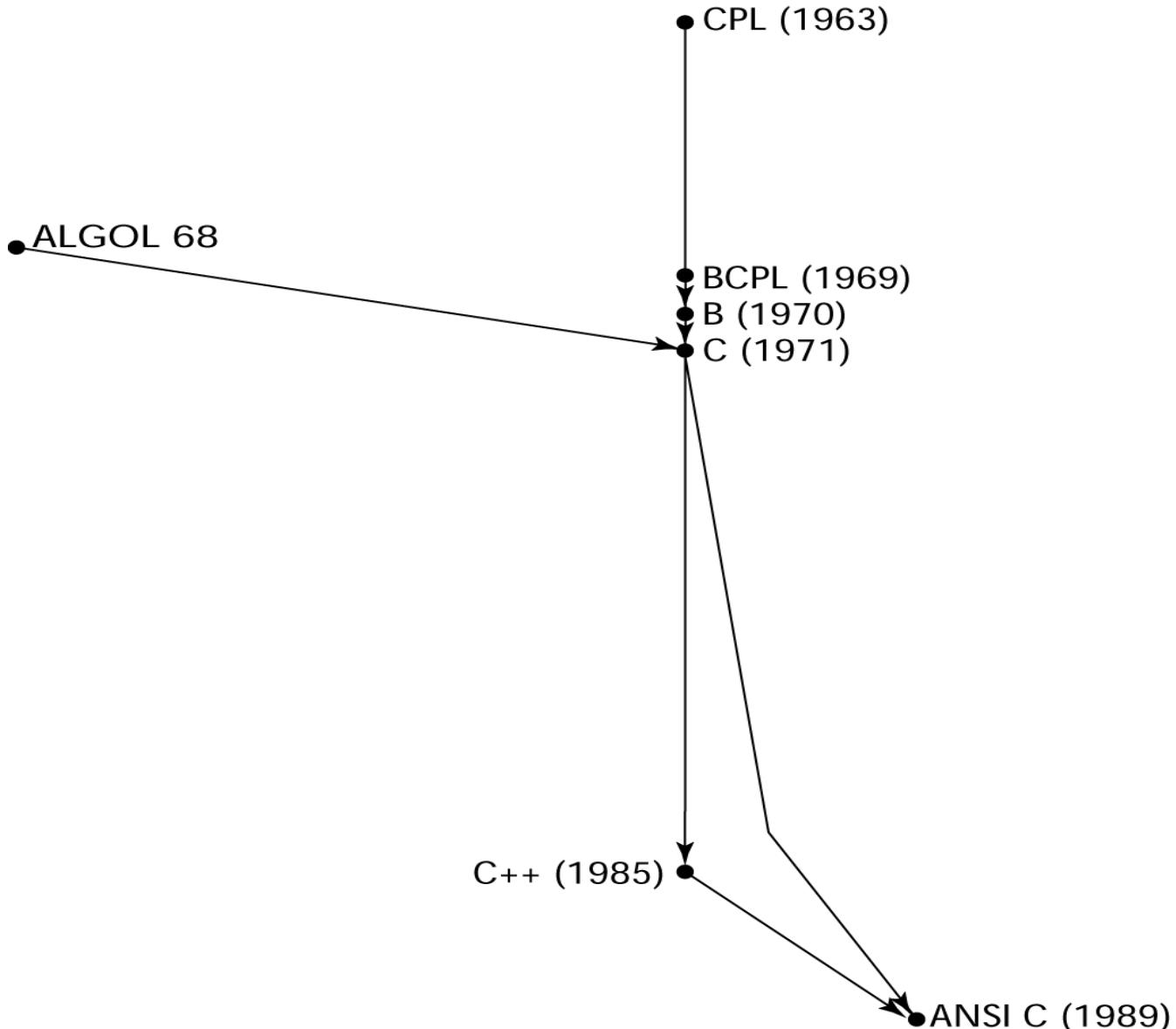
ALGOL’ün Torunları

- **Pascal (1971):**
 - İlk önceleri eğitim dili olarak kullanıldı
 - Basit ve okunabilir, yazılabilir (expressive)
 - Fortran and C ile karşılaştırıldığında emniyetli bir dildir.
- **C: Taşınabilir Sistem Programlama Dili (1972)**
 - Yeterli derecede kontrol ifadelerine ve veri yapısı olanaklarına sahip.
 - Tip kontrolü yetersiz.
 - Derleyicilerine çok kolay bir biçimde ulaşılabilir.

Pascal'ın şeceresi



C'nin Şeceresi



ALGOL’ün torunları(devam)

- **Modula:**
 - Yazılım Mühendisliği alanında önemli deneyimler sonucu ortaya çıkan ve güçlü bir tip ayımı ve tip kontrolü mekanizmasına sahip bir dildir.
 - Dinamik dizi kullanabilme (dinamik bellek yönetimi) ve eşzamanlılık özelliklerine sahiptir.
 - Lilith adı verilen yeni bir bilgisayar sistemi için tasarlanmıştır fakat kullanılmamıştır.
 - Modula 2 Pascal'a göre daha başarılı ve kullanıcıya esneklikler sunmakta. Modula 2'nin syntax'ı daha esnektir.
 - Gerçek uygulama yazılım sistemleri için geliştirilmiştir.
 - Soyut veri tipini destekliyor fakat miras alma (inheritance) özelliğini desteklemiyordu.Nesneye yönelik yok

ALGOL'ün torunları(devam)

- **Oberon:**
 - Modula-2'ye dayanmakta fakat ondan daha basit idi.
 - OOP'yi desteklemek için tipleme genişletilmişti.Tip ayrimı güçlündür. OOP'i tam olarak destekler.
 - Geleneksel veri türlerini ve kontrol yapılarının çoğunu içerir. Dinamik bellek yönetimi vardır.
 - Endüstriyel gücü yüksek bir dil idi. Uygulama geliştirme amacıyla geliştirilmiş bir dildir.
 - Veritabanı, ağ haberleşmesi, 3 boyutlu grafikler oluşturma ve nesne yönetimi için destek sağlayan zengin bir kütüphanesi vardır.
 - Soyut veri türlerini destekler.

Prolog: Programming Based on Logic (1972)

- Çok üst düzey programlama dilinin ilk örneği
- PRogramming LOGic
- Mantık yürütme ve ispatlama tekniklerini kullanır
- PROLOG programı, bir nesneler kümesi ile bu nesnelerle ilişkili hedeflere nasıl erişilebileceğini tanımlayan kurallar kümesinden oluşur.
- Prosedürel bir yaklaşım değildir.
- Dil kuralar ve gerçeklerden oluşur.
- Verimsiz kalmıştır.
- Kullanım alanı belirli tipte VTYS ve yapay zeka alanlarıyla sınırlıdır.

ADA:

Tarihin en geniş tasarım çalışması

- ABD savunma bakanlığının bir çalışması sonucu ortaya çıkmıştır. Gömülü sistemlerin (embedded systems) programlanması sağlayacak PL üretimi amaçlanmıştır. (1983). Gerçek zamanlı uygulamlar için
- Augusta Ada Byron'ın (1815-1851) ölümünden sonra isimlendirilmiştir.
- Blok yapılı, nsne yönelimli, genel amaçlı ve eşzamanlılığı destekleyen bir dildir.
- Büyük boyutlu yazılımlar için uygundur.

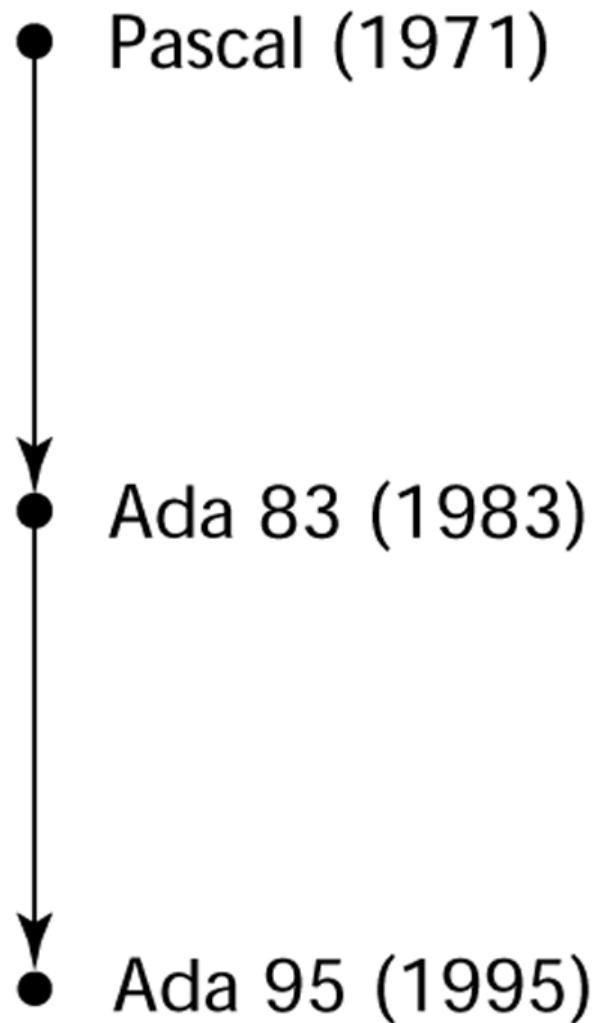
ADA (devam)

- Şablon (Generic) program birimleri sayesinde yazılımın yeniden kullanılabilmesini Buluşma yeri(rendezvous) mekanizmasının ilavesiyle eşzamanlı çalışmayı desteklemektedir. mechanism
- Çok geniş ve çok karmaşık bir dil. (Özellikle yazım kuralları)
- Çeşitli durumlara uygulanabilecek hazır şablonlara (template) sahiptir.
- İlk sıralar ADA derleyicileri kod üretmekte verimsiz idi.
- En çok gömülü sistemlerde başarılı olmuştur.
- Veri tipleri konusunda çok zengindir. Çok-iş işleme özelliğine sahiptir.

Ada 95

- ADA83'e kalıtım (**inheritance**), çok biçimlilik (**polymorphism**) gibi yeni eklemeler yapılarak ve **tip üretim mekanizmasını genişleterek** Ada 95 elde edilmiştir.
- Altprogramların dinamik kapsam bağlama kurallarına göre çağrılmaması mekanizması da ADA 95'in özelliklerine katılmıştır.

ADA'nın şeceresi

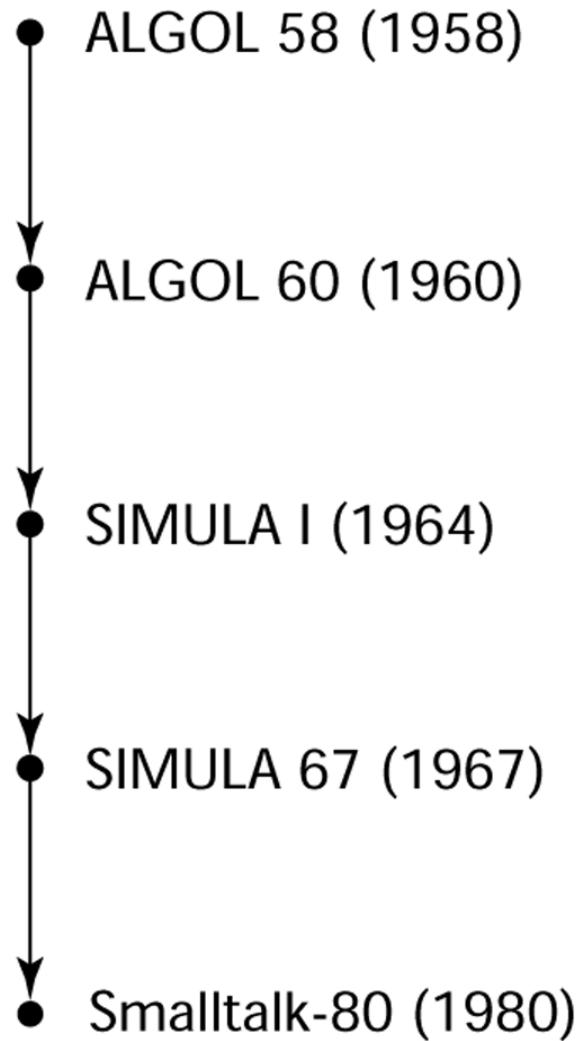


Smalltalk: Object-Oriented Programming

- OOP 3 temel karakteristiği: data abstraction, inheritance ve dynamic binding
- Smalltalk sadece bir PL değil, aynı zamanda yazılım geliştirme aracıdır.
- Program birimleri, verileri kapsülleyen nesneler ve yöntemlerden oluşur.
- Hesaplama bir nesneye bir mesaj göndererek yapılır

- İlk olarak Smalltalk-80 versiyonu Xerox dışında ticari amaçlı yazılımlarda kullanıldı. Halen yazılım geliştirimleri için yeni teknolojileri de içерerek kullanılmaktadır.
- **Smalltalk tamamen nesne yönelimli olan ve ticari ilk programlama dilidir.**
- Geliştirme ortamı nesne yönelimli bir alt yapısı olması nedeni ile çok gelişmiştir (tarayıcı, editör, debugger, açık kaynak kod).
- **Kaynak kodunun açık olması iyi bir eğitim ortamı haline gelmesini sağlamıştır. Diğer dillere göre çok basit bir sentaksı vardır ve Java gibi Geniş ve sürekli genişleyen bir programlama kütüphanesine sahiptir. Platform bağımsız bir dildir.**
- Bu dilde nesne yönelimli programmanın 3 temel karakteristiği olan veri soyutlama (data abstraction), kalıtım (inheritance) ve dinamik bağlama(dynamic binding) kavramlarının hepsi bulunmaktadır.

Smalltalk'ın şeceresi



C++: (Imperative + OO)

- C ve Smalltalk'ın birleşimi
- Çoklu miras alma desteklenmekte
- Fonksiyon ve NYP 'ya izin vermektedir.
- Operatör ve metodların üstüste bindirilmesi desteklenmekte
- Sınıflar ve metodlar template edilebilir.
- Kuvvetli tip ayrimı, **dinamik bellek yönetimi**, hazır şablonlara sahip olma ve çok biçimlilik (polymorphism) özellikleri vardır.
- C'de bulunan özelliklerin çoğu burada vardır.
- PL/1 gibi geniş ve kompleks.
- Ada ve Java'dan daha az güvenli

Eiffel (1992)

- Imperative ve OO özellikleri birleştiren Hybrid PL
- Soyut veri yapılarını, kalıtımı ve dinamik bildirimleri destekler. (OO)
- Altprogram ve çağrııcı (caller) arasındaki iletişim için bildirimler (assertions) kullanır.
- C++'dan daha küçük ve basittir. Fakat ifade edilebilirliği ve yazılabilirliği neredeyse eşittir.

Delphi:

- Emir esası ve OO PL'nin başarılı bir biçimde birleştirilmesidir.
- Pascal'dan türemiştir. Bu yüzden, dizi elemanlarının kontrolünde, pointer aritmetiği ve tip zorlamalarında C ve C++'tan daha emniyetlidir.
- C++ 'dan daha az komplextir.
- Kullanıcı tanımlı opertaörlere, generic altprogramlara ve parametrize edilmiş sınıflara izin vermez.
- Daha iyi ve daha kolay yazılım geliştirtme için bir Graphical User Interface (GUI) sağlamaktadır.

Java: Emir Esaslı bir NDPD (OOL)

- Daha küçük, daha basit ve daha güvenilir bir PL tasarlak üzere C++ tabanlı olarak geliştirilmiştir.
- Java, basit, taşınabilir ve nesneye yönelik özellikte bir didir.
- Miras alma, çok biçimlilik, kuvvetli tip kontrolü, eş zamanlılık kontrolü, dinamik olarak yüklenebilen kütüphaneler, diziler, string işlemleri ve standart kütüphane gibi özellikleri vardır.
- Bir Java programının temel yapısal bileşeni sınıfıdır. Bütün veri ve metodlar bir sınıf ile ilişkilidir. Global veri yada fonksiyon yoktur.
- Hem referans değişkenleri ile hem de ilkel tiplerle erişilebilen sınıflara sahiptir.
- C++'ta bulunan çoklu miras alma, opertaörlerin üstüste bindirilmesi, ve makro önişemcisi özellikleri Java'da yoktur.
- Java'da şablon yapıları yoktur. İhtiyaç da minimuma inmiştir.

Java (devam)

- Pointer yoktur. Fakat bütün nesne sınıfları object adlı kök sınıfından miras almaktadır.
- Records, union or enumeration tipler yoktur.
- Prosedürel programlamayı desteklemez.
- Sadece tekli miras almayı destekler fakat çok gelişmiş bir GUI'ya sahiptir.
- İplik (**threads**) yapısına (**bir pencere içerisinde yeni pencereler açılmasına izin var**) sahip olduğundan eşzamanlılığı yönetmek kolaydır.
- **Çöp toplama (Garbage collection)** nesneler için belleği eniyi kullanımı sağlar.
- Tip dönüşümü kuvvetlidir.

Java (devam)

Java tipik olarak platformdan bağımsız olarak byte kodları biçiminde derlenir. Daha sonra bu byte kodlar bir Java görüntü makinesi adı verilen bir Java **yorumlayıcısı** tarafından kullanılacağı kullanılacağı platformun makine dilindeki koduna çevrilir. Derlenmiş Java sınıflarının taşınabilirliğini garantileyen bir özellik vardır. Oda .class format adı verilen Java byte-kod dosya formatının kesin olarak tanımlanmış olmasıdır.

Java “applet” adı verilen başka sistemler içinde gömülü programların geliştirilmesi için de kullanılmaktadır. Bu appletler Internet Explorer yada Netscape gibi web tarayıcı programlar içerisinde kullanılabilir.

Java internet ve web programcılığında yaygın olarak kullanılmaktadır.

JavaScript: HTML-resident, client-side scripting language for the Web

- Netscape ve Sun Microsystems 'in ortak çalışmaları sonucu 1995'de üretilmiştir. Orijinal adı "LiveScript"tir.
- Dil bileşenlerinin içinde, web sayfalarının çeşitli kısımlarını ve özellikle HTML'i işleyecek ve kontrol edecek çeşitli olanaklar vardır.
- Javanın veri tipleri bakımından sert kısıtlamalarına karşılık, (Örn.Bütün değişken tipleri derleme zamanında belirlenir), Javascript bu açıdan toleranslıdır.(dynamik olarak tipleme)
- Java'nın HTML ile çalışma desteği zayıf iken, Javascript'in bu konudaki desteği tamdır.
- Javascript nesneye yönelik ve blok-yapısal özellikte bir dildir.

PHP: HTML-resident, server-side scripting language for the Web

- Personal Home Page (PHP), Rasmus Lerdorf tarafından 1994 yılında geliştirildi.
- Bir HTML dökümanı, okuyucu, hiper linkler boyunca diğer HTML dökümanlarına götürecek biçimde programlanabilir.
- Gömülü bir HTML belgesi browser tarafından istendiğinde PHP kodu Web server üzerinde yorumlanır.
- PHP'nin dizi yapısı JavaScript ve Perl dizi yapılarının bir birleşimidir.
- PHP ile form erişimi oldukça kolaydır.

C#

- C ve C++ dil ailesinin ilk bileşen yönelimli (Component-oriented) dilidir.
- C ve C++'dan derlenmiş, basit, modern, nesne yönelimli ve tür güvenli bir programlama dilidir.
- Yüksek başarımlı Common Language Runtime (CLR); bir yürütme motoru, bir çöp toplayıcı (garbage collection), anında derleme, bir güvenlik sistemi ve zengin bir sınıf çerçevesi (.NET Framework) içerir. CLR temelden, birden çok dil desteğine kadar hersey için tasarlanmıştır.
- CLR'ı hedef alan diller: Visual C#, Visual BASIC .NET, Managed C++, J#.NET ve Jscript.NET
- Common Language Specification, CLS, dil işlevselliliğinin yaygın bir düzeyini tanımlar. .NET Framework işlevselliliğine tam erişim ve ve diğer uyumlu dillerle zengin birlikte çalışabilirlik vardır.

C# (devam)

- C# otomatik bellek yönetimini kullanır. C# tür sistemi işaretçi türleri ve nesne adreslerinin doğrudan değiştirilmesine de izin verilir.
- C# tür sistemi bileşiktir. Her şey bir nesnedir. Kutulama ve kutuyu açma gibi kavramların yenilikçi kullanımı ile C#, her veri parçasının bir nesne olarak değerlendirilmesine olanak sağlayarak, değer türleri (value type) ve başvuru türleri (reference type) arasındaki açığı kapatır.

Bölümün Amacı

- Programlama dillerinde sözdizim ve anlam kavramları,
- Soyut sözdizim, metinsel sözdizim ve gramer kavramları,
- Ayrıştırma ağaçlarının ve türetmelerin oluşturulması,
- Sözdizim tanımlanması için kullanılan BNF metadili,
- Anlam tanımlamada ve dilin standartlaştırılması

GİRİŞ

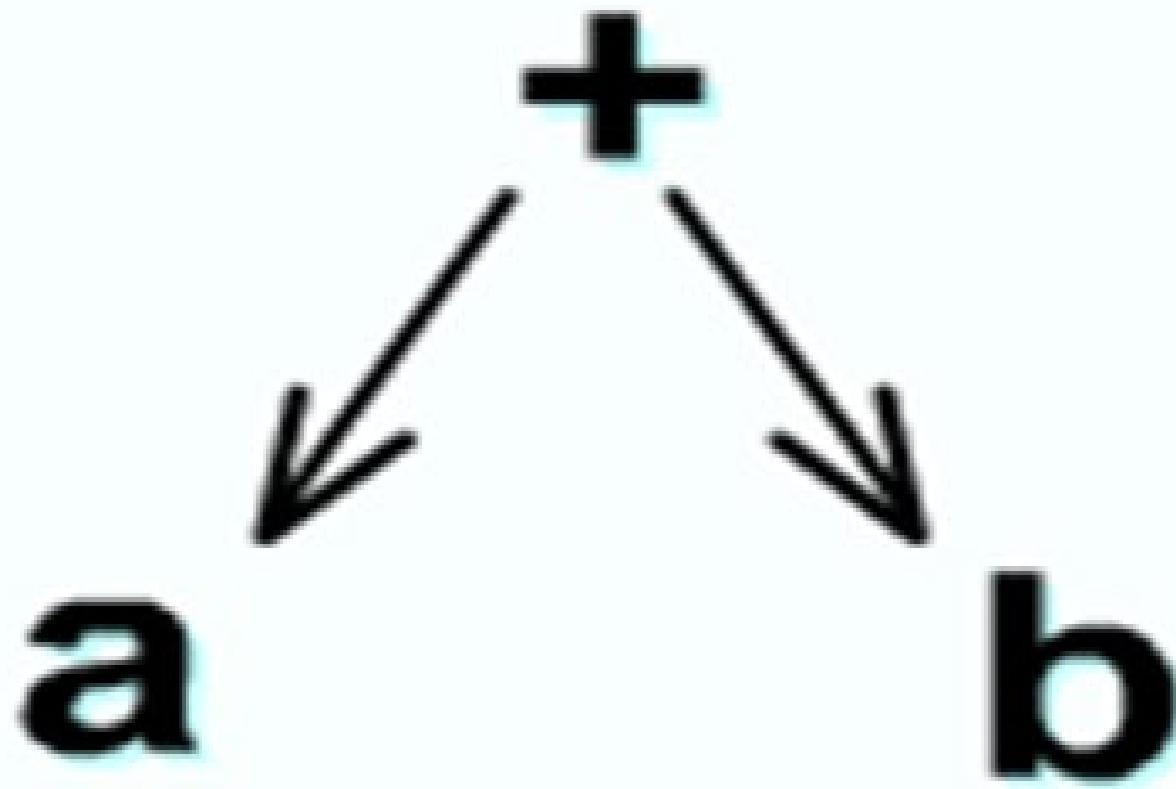
- Yazılan bir programın o dile ait olup olmadığını belirleyen kurallar, **sözdizim** (*syntax*) ve **anlam** (*semantics*) olarak ikiye ayrılabilir.
- Bir programın **begin** ile başlayıp **end** ile bitmesi, her deyimin sonunda noktalı virgül bulunması (sözdizim kuralları), bir değişkenin kullanılmadan önce tanımlanması (anlam kuralı) gibi.

- Bir dilin sözdizimini anlatmak amacıyla kullanılan bir araç vardır. BNF (Backus-Naur Form) **metadili** böyle bir araçtır.
- Anlam tanımlama için böyle bir dil yoktur.

SOYUT SÖZDİZİM

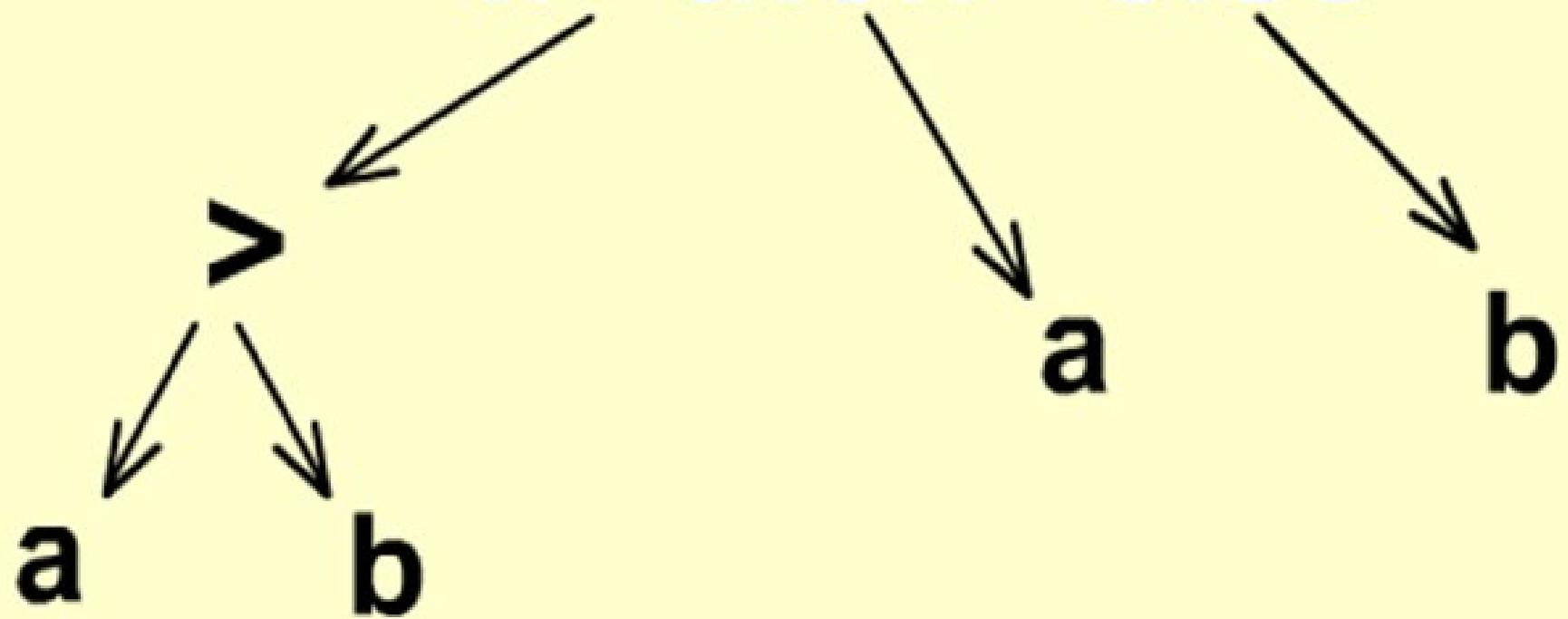
- Bir dilde bulunan her yapıdaki anlamlı bileşenler o dilin **soyut sözdizimi**, tanımlar.
- *+ab prefix* ifadesi,
- *a+b infix* ifadesi,
- *ab+ postfix* ifadelerindeki *+ işlemcisi* ve *a* ve *b* alt-ifadelerinden oluşan aynı anlamlı bileşenleri içermektedir.

Bu nedenle ağaç olarak üçünün de gösterimi yandaki şekildeki gibidir.



if $a > b$ then a else b

if - then - else



METİNSEL SÖZDİZİM

- Dil, bir alfabeeki karakter dizilerinden oluşurlar.
- Dildeki karakter dizileri **cümle** veya **deyim** oluşturur.
- **Sözdizim kuralları:** Hangi karakter dizilerinin o dilin alfabetesinden ve o dile ait olduğunu belirlerler.
- Diller sözdizimsel olarak basittir.

SÖZDİZİM (devam)

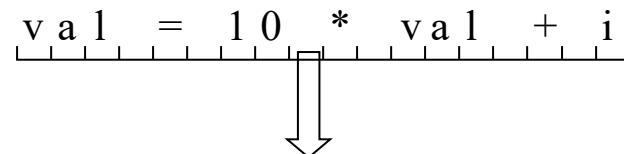
- **Lexeme**: Bir programlama dilindeki en düşük düzeyli sözdizimsel birimlerdir.
- Programlar, karakterler yerine *lexeme*'ler dizisi olarak düşünülebilir.
- **Token** : Bir dildeki *lexeme*'lerin gruplanması oluşan sınıflardır.

- Dilinin metinsel sözdizimi, *token*'lar dizisidir.
- Örneğin bir tanımlayıcı (identifier); *ortalama* veya *ogrenci* gibi *lexeme*'leri olabilen bir *token*'dır.
- Bir *token*'ın sadece tek bir olası *lexeme*'i olabilir.

Örneğin, çarpma_islemcisi denilen aritmetik işlemci "*" sembolü için, tek bir olası *lexeme* vardır.

- Boşluk (space), ara (tab) veya yeni satır karakterleri, *token*'lar arasına yerleştirilmesi programın anlamına etkisi yoktur.

character stream



lexical analysis (scanning)

token stream

A table representing the token stream. It consists of seven columns, each containing a token number, its type, and its value. The tokens are: 1 (ident) "val", 3 (assign) -, 2 (number) 10, 4 (times) *, 1 (ident) "val", 5 (plus) +, and 1 (ident) "i". Arrows point from the right side of the table to the rightmost column, labeled 'token number' above and 'token value' below.

1	3	2	4	1	5	1
(ident)	(assign)	(number)	(times)	(ident)	(plus)	(ident)
"val"	-	10	*	"val"	+	"i"

token number

token value

PROGRAMLAMA DİLLERİNDE GRAMER

- Bir programlama dilinin metinsel (somut) sözdizimini açıklamak için **Gramer**, kullanılır.
- Gramerler, anahtar kelimelerin (keywords) ve noktalama işaretlerinin yerleri; deyim listelerinin nasıl oluşturulacağını belirleyen bir dizi kuraldan oluşur.

character stream

v a l = 1 0 * v a l + i



lexical analysis (scanning)

token stream

1 (ident) "val"	3 (assign) -	2 (number) 10	4 (times) -	1 (ident) "val"	5 (plus) -	1 (ident) "i"
-----------------------	--------------------	---------------------	-------------------	-----------------------	------------------	---------------------

token number

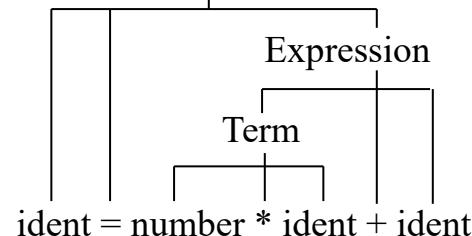
token value



syntax analysis (parsing)

Statement

syntax tree



Grammar Nedir?

Example Statement = "if" "(" Condition ")" Statement ["else" Statement].

4 bileşenden oluşur

terminal symbols atomik "if", ">=", ident, number, ...

nonterminal symbols Sözdizim değişkenleri Statement, Expr, Type, ...

productions Nonterminallerin çözümü Statement = Designator "=" Expr ";".
Designator = ident ["."] ident].
...

start symbol Başlangıç nonterminalı begin

Arithmetik İfadelerin Grameri

Productions

Expr = ["+" | "-"] Term { ("+" | "-") Term }.
Term = Factor { ("*" | "/") Factor }.
Factor = ident | number | "(" Expr ")".

Terminal symbols

simple TS: "+" , "-" , "*" , "/" , "(" , ")"
(just 1 instance)

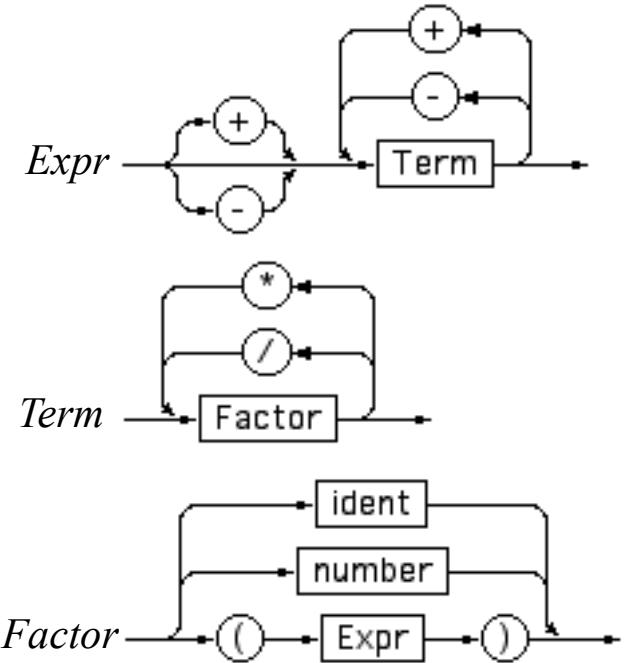
terminal classes: ident, number
(multiple instances)

Nonterminal symbols

Expr, Term, Factor

Start symbol

Expr



Backus-Naur Form (BNF)-CFG

□ CFG

$$\begin{aligned} \text{expression} &\rightarrow \text{identifier} \mid \text{number} \mid - \text{ expression} \\ &\quad \mid (\text{ expression }) \\ &\quad \mid \text{ expression } \text{ operator } \text{ expression} \\ \text{operator} &\rightarrow + \mid - \mid * \mid / \end{aligned}$$

□ BNF

$$\begin{aligned} \langle \text{expression} \rangle &\rightarrow \langle \text{identifier} \rangle \mid \langle \text{number} \rangle \mid - \\ &\quad \mid (\langle \text{expression} \rangle) \\ &\quad \mid \langle \text{expression} \rangle \langle \text{operator} \rangle \langle \text{expression} \rangle \\ \langle \text{operator} \rangle &\rightarrow + \mid - \mid * \mid / \end{aligned}$$

EBNF Notation

Extended Backus-Naur form

John Backus: developed the first Fortran compiler

Peter Naur: edited the Algol60 report

<i>symbol</i>	<i>meaning</i>	<i>examples</i>
string		"=", "while"
name		ident, Statement
=		A = b c d .
.		
	separates alternatives	a b c ⊢ a or b or c
(...)	groups alternatives	a (b c) ⊢ ab ac
[...]	optional part	[a] b ⊢ ab b
{...}	repetitive part	{ a } b ⊢ b ab aab ...

EBNF

<seçimlik_deyim> -> **If** (**<mantıksal>**) **<deyim>** [**else** **<deyim>**];

BNF

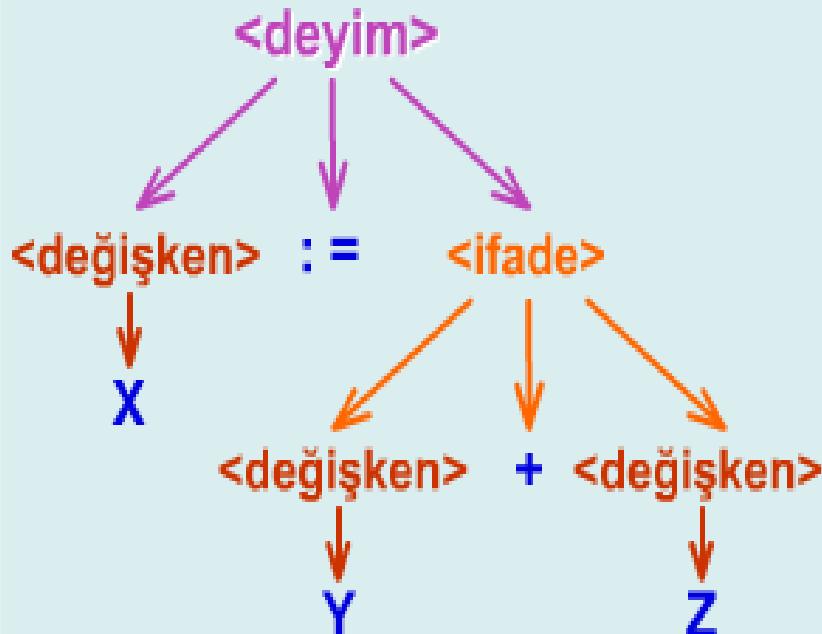
<seçimlik_deyim> -> **If** (**<mantıksal>**) **<deyim>**

<seçimlik_deyim> -> **If** (**<mantıksal>**) **<deyim>** **else** **<deyim>**;

Değiştirme (*alternation*) |

EBNF	<for_deyimi>->for<değişken> := <ifade> (to down to) <ifade> do <deyim>
BNF	<for_deyimi>->for<değişken> := <ifade> (to) <ifade> do <deyim>
	<for_deyimi>->for<değişken> := <ifade> (down to) <ifade> do <deyim>

Gramerler ve Türetimler



<program> -> begin <deyim_listesi> end

<deyim_listesi> -> <deyim>
| <deyim>; <deyim_listesi>

<deyim> -> <değişken> := <ifade>

<ifade> -> <değişken> + <değişken>
| <değişken>

<değişken> -> X | Y | Z

Örnek

- $\text{expression} \rightarrow \text{identifier} \mid \text{number} \mid - \text{expression}$
 - | (expression)
 - | $\text{expression operator expression}$
- $\text{operator} \rightarrow + \mid - \mid * \mid /$

slope * x + intercept ifadesini
turetelim.



slope * x + intercept

expression \Rightarrow *expression operator* *expression*

\Rightarrow *expression* *operator* intercept

\Rightarrow *expression* + intercept

\Rightarrow *expression operator* *expression* +
intercept

\Rightarrow *expression* *operator* x + intercept

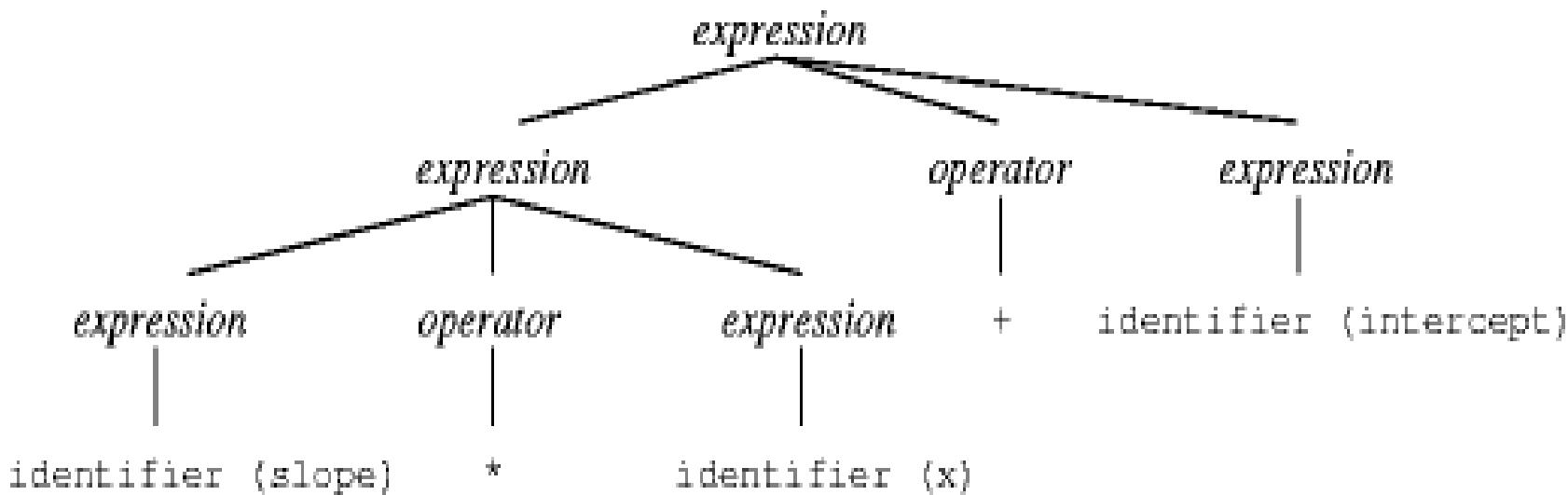
\Rightarrow *expression* * x + intercept

\Rightarrow slope * x + intercept

expression \Rightarrow^* slope * x + intercept

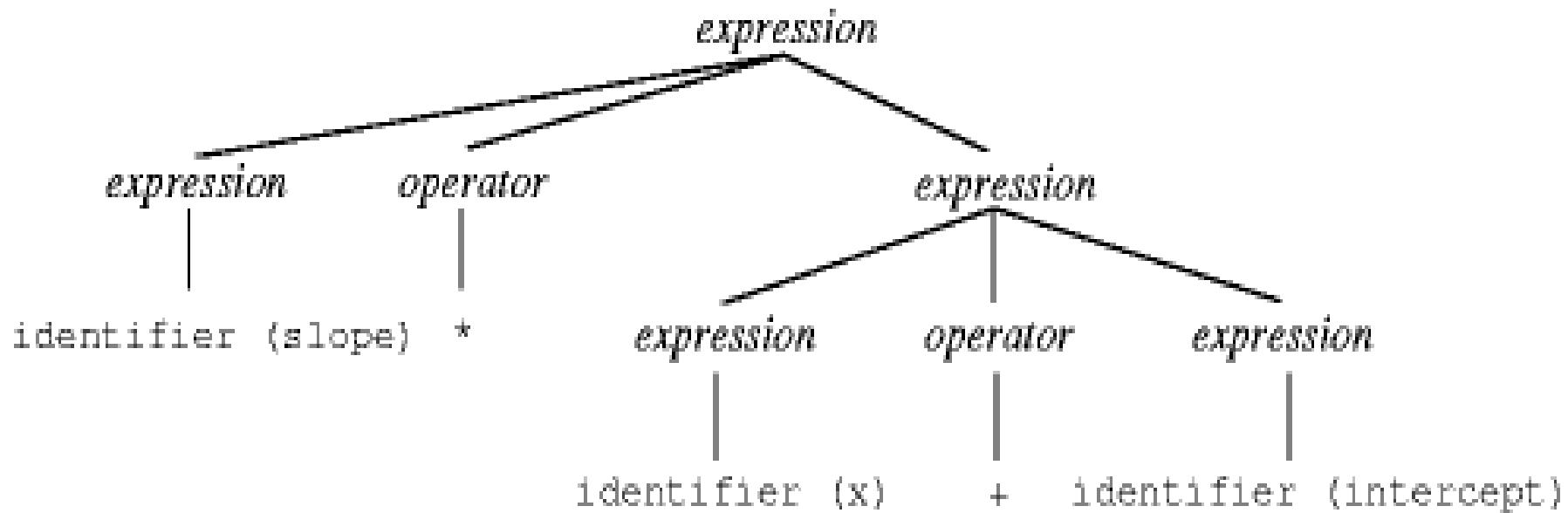
Ayrıştırma Ağacı(Parse Trees)

- Türetimin grafik gösterimi



Belirsiz Gramer

- Bir gramerde aynı ifade için alternatif ayrıştırma ağacı bulunuyorsa bu gramer belirsizdir (ambiguous)



Belirsizlik kaldırılmalıdır.

$$10 - 4 - 3 == (10 - 4) - 3$$

$$3 + 4 * 5 == 3 + (4 * 5)$$

(1) *expression* \rightarrow *term* | *expression add_op term*

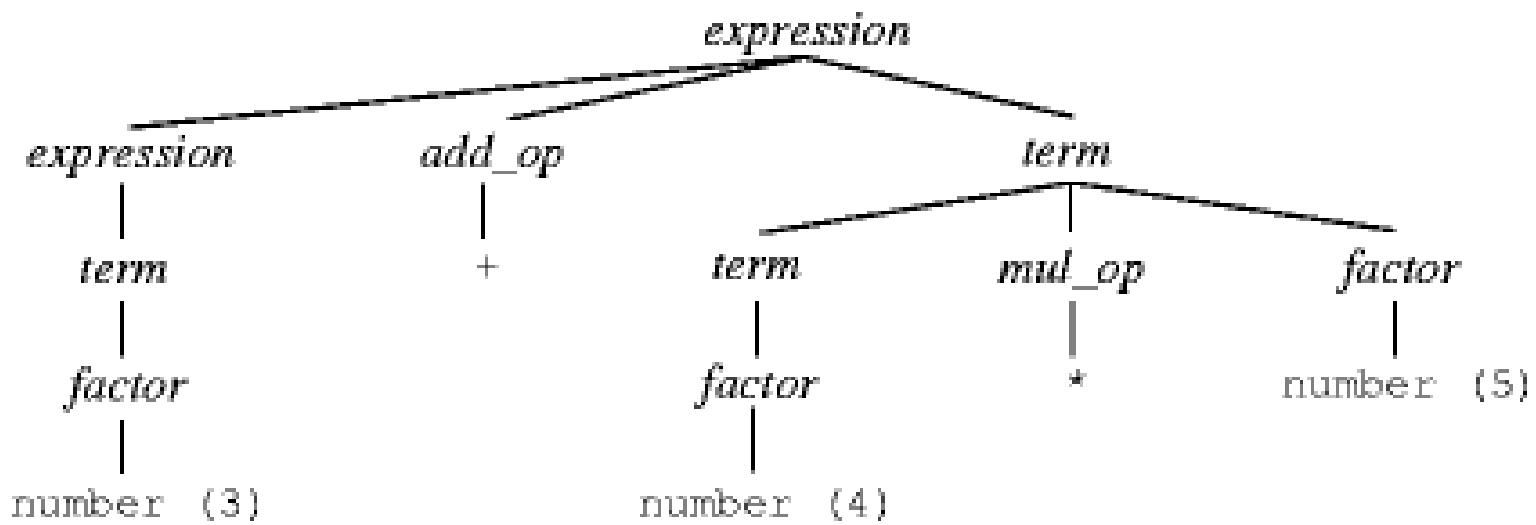
(2) *term* \rightarrow *factor* | *term mult_op factor*

(3) *factor* \rightarrow *identifier* | *number* | - *factor* | (*expression*)

(4) *add_op* \rightarrow + | -

(5) *mult_op* \rightarrow * | /

3 + 4 * 5 için Parse ağacı



Operator Priority

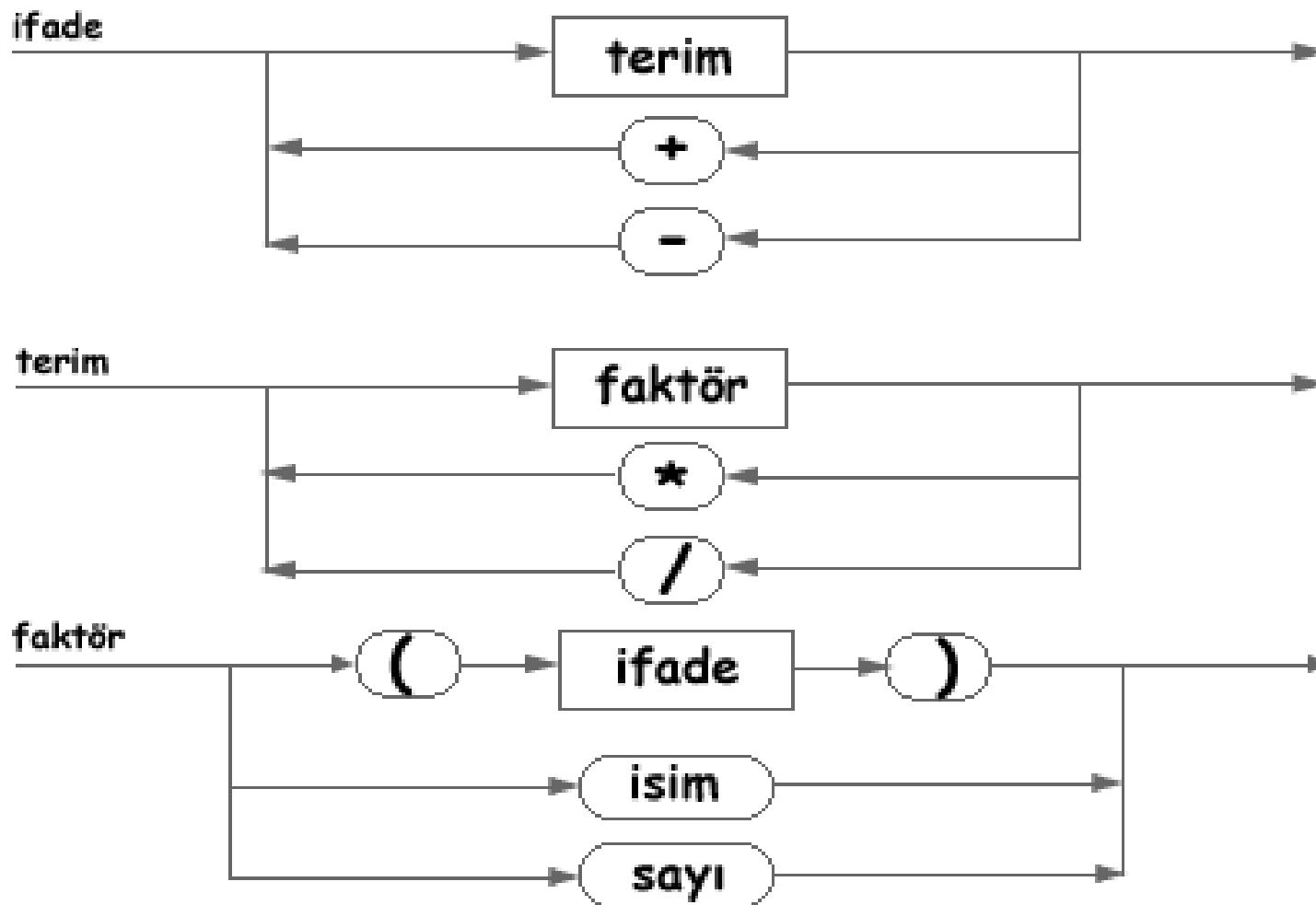
```
Expr  = [ "+" | "-" ] Term { ( "+" | "-" ) Term }.
Term  = Factor { ( "*" | "/" ) Factor }.
Factor = ident | number | "(" Expr ")".
```

input: - a * 3 + b / 4 - c

- ❑ - ident * number + ident / number - ident
- ❑ - Factor * Factor + Factor / Factor - Factor

Sözdizim Grafikleri

- BNF ve EBNF'teki kurallar, ayrıştırma ağacı dışında **sözdizim grafikleri** (*syntax graphs*) ile de gösterilebilir. Sözdizim grafikleri, ilk olarak Pascal'ın gramerini açıklamak için kullanılmıştır.



ANLAMSAL TANIMLANMA

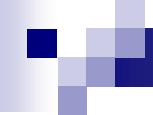
- Bir programlama dilinin **anlam (semantic) kuralları**, bir dilde sözdizimsel olarak geçerli olan herhangi bir programın anlamını belirler.

Anlam tanımlama

- Anlamsal tanımlama için varolan yöntemler oldukça karmaşıktır ve hiçbir yöntem sözdizim tanımlamak için kullanılan BNF metadili gibi yaygın kullanıma ulaşmamıştır.

Anlam tanımlama

- Anlam tanımlama için, dil yapılarının Türkçe gibi bir doğal dilde açıklanması sağlanır. Ancak doğal dil kullanılarak yapılan açıklamalar, açık ve kesin olmaz.



Durağan Anlam Kuralları

Dinamik anlam Kuralları

PROGRAMLAMA DİLLERİNİN STANDARTLAŞTIRILMASI

- Dilin sözdizimi ve anlamı tam ve açık olarak tanımlanmalı ve **dil standardı** oluşturulmalıdır.
- Bir dil standartı, dil tasarımcısı, dil tasarımını destekleyen kuruluş veya Amerika Birleşik Devletleri'nde *American National Standards Institute (ANSI)* yada *International Standards Organization (ISO)* tarafından gerçekleştirilebilir gibi bir kuruluş tarafından tanımlanabilir.

Bölüm Özeti

Metinsel Sözdizim
Gramer, BNF, EBNF
Türetim ve türetme ağaçları
Belirsizlik
Belirsizliğin kaldırılması

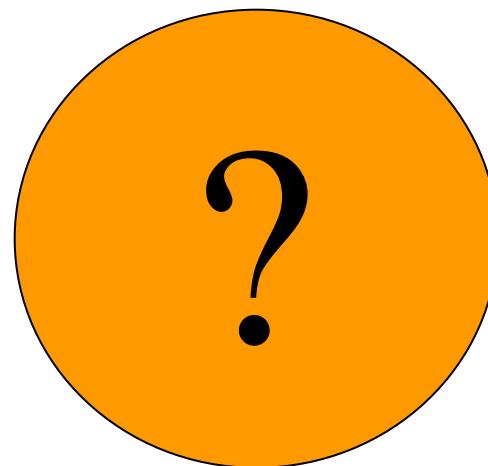
Kaynaklar:

- **Programming Languages: Concepts & Constructs, Second Edition, Ravi Sethi.**
- **Concept of Programming Languages, Fourth Edition, Robert W. Sebesta.**

DİL ÇEVİRİMİ

- Yüksek düzeyli bir dilde yazılmış bir program ancak makine diline çevrilerek bir bilgisayarda çalıştırılabilir.

High-level source code

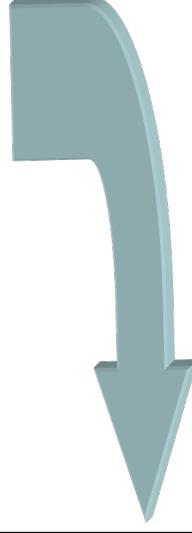


Low-level machine code

Yüksek düzeyli bir dilde yazılmış kaynak kodun makine diline (Hedef kod) dönüştürülmesi zorunluluğu **Dil Çevirici** yazılımların oluşturulmasına neden olmuştur.

```
program gcd(input, output);  
var i, j: integer;  
begin  
    read(i, j);  
    while i <> j do  
        if i > j then i := i - j;  
        else j := j - i;  
    writeln(i)  
end.
```

Compilation



```
27bdffd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c  
00401825 10820008 0064082a 10200003 00000000 10000002 00832023  
00641823 1483ffffa 0064082a 0c1002b2 00000000 8fbf0014 27bd0020  
03e00008 00001025
```

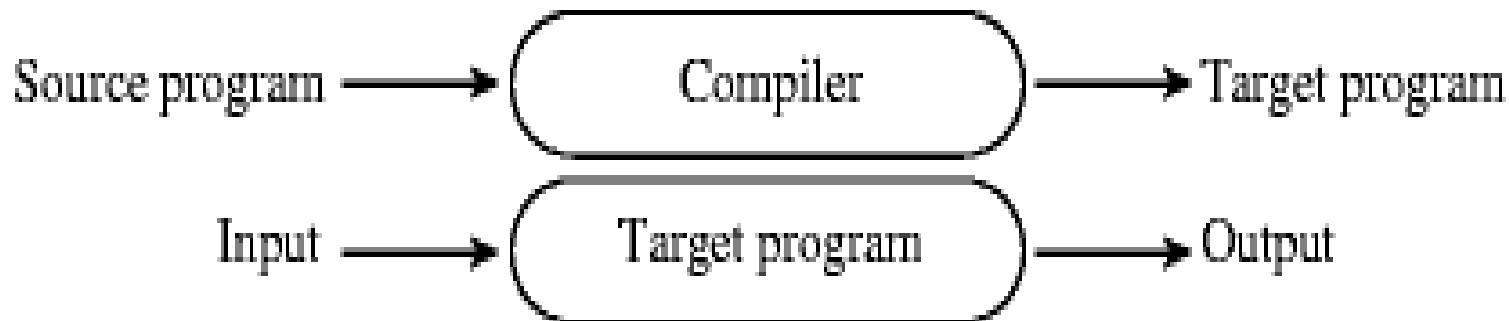
```
program gcd(input, output);
var i, j: integer;
begin
    read(i, j);
    while i <> j do
        if i > j then i := i - j;
        else j := j - i;
    writeln(i)
end.
```

DERLEYİCİ

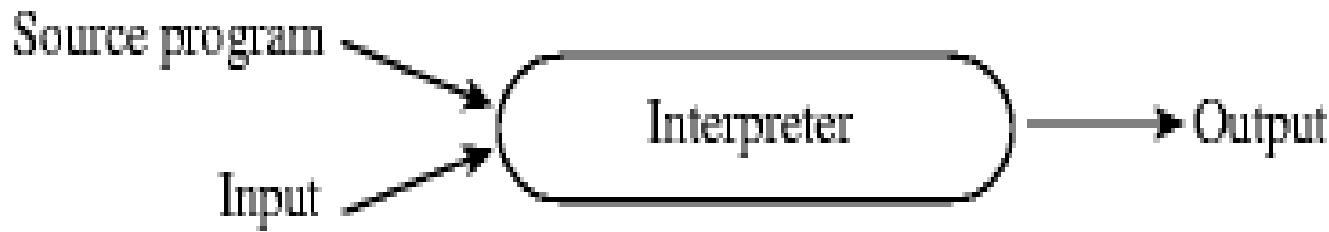
001010101010
10101011111010
11101010101110
001010101010
. . .

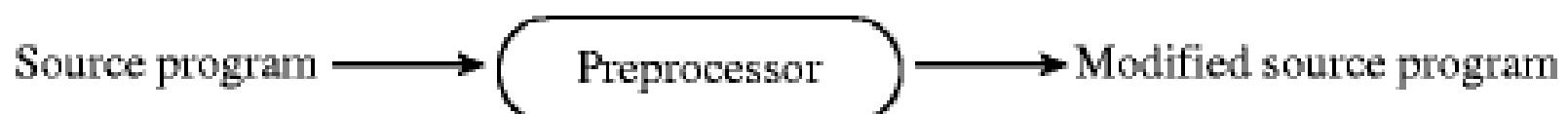


- Dönüşüm için **derleme** ve **yorumlama** olmak üzere iki temel yöntem vardır.
- Bir **yorumlayıcı**, bir programın her deyimini birer birer makine diline çevirerek çalıştırır.
- **Derleyici**, bir programlama dilinde yazılmış bir kaynak kod için o koda eşdeğer olan makine dilinde bir program oluşturur.

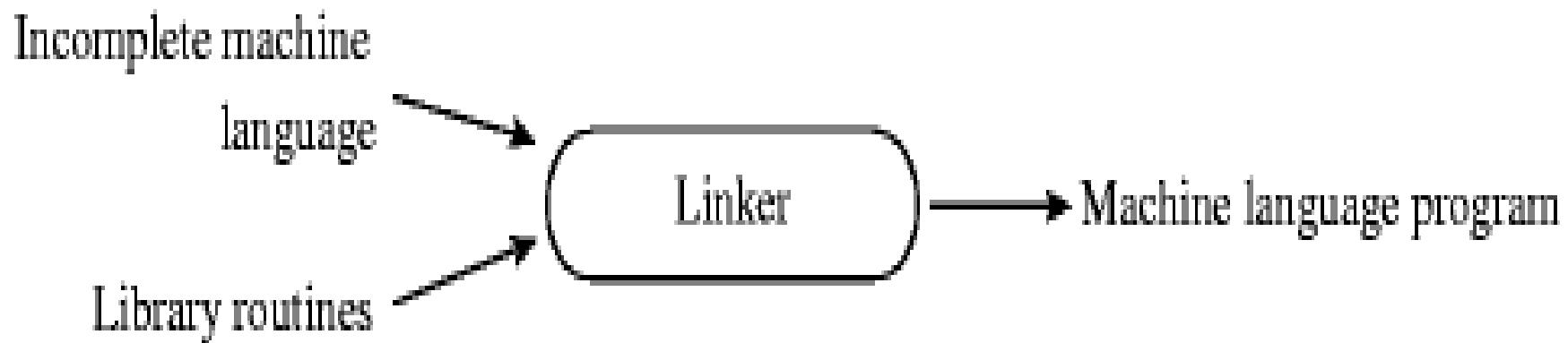


- Yorumlayıcı





```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
```



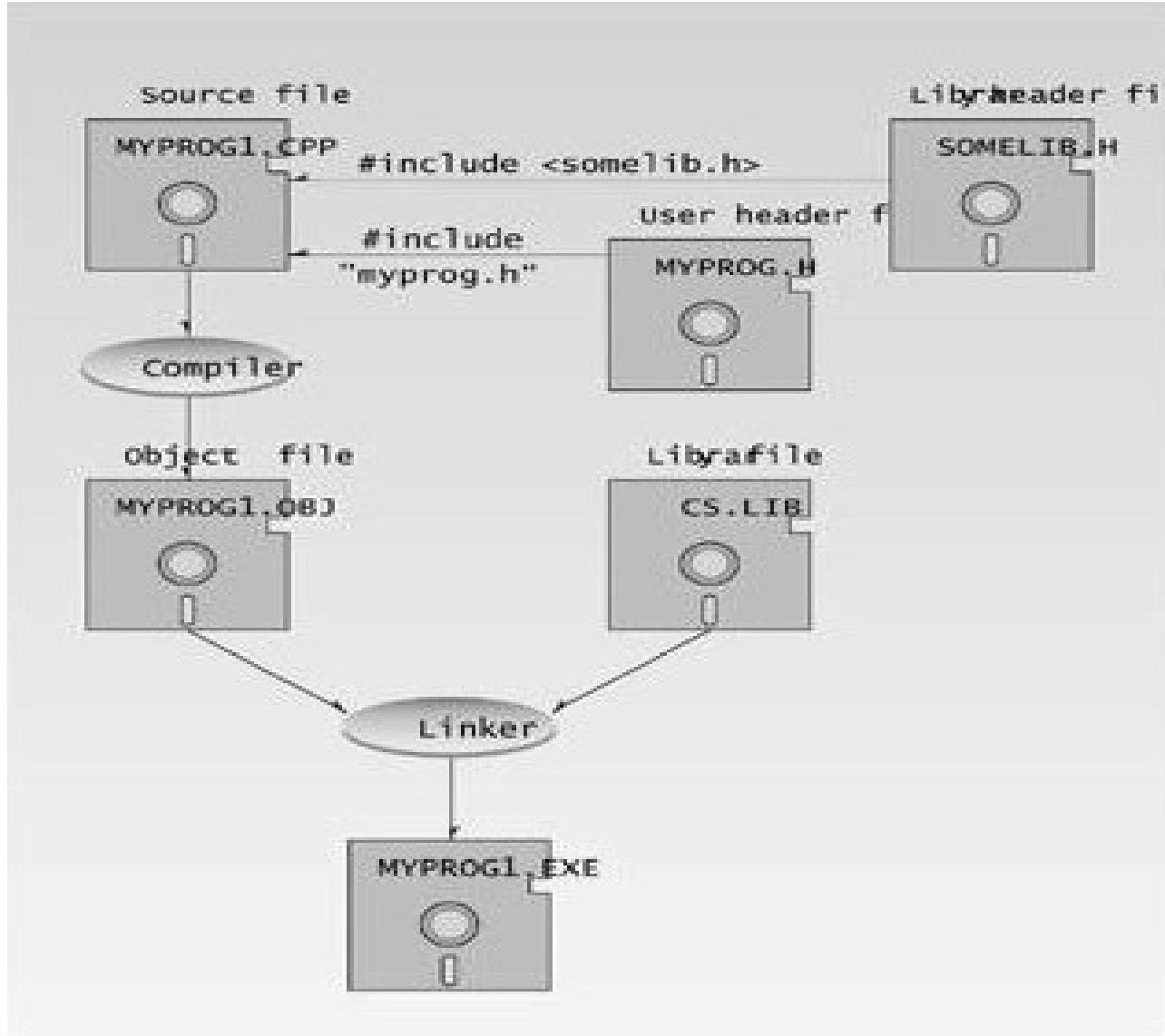


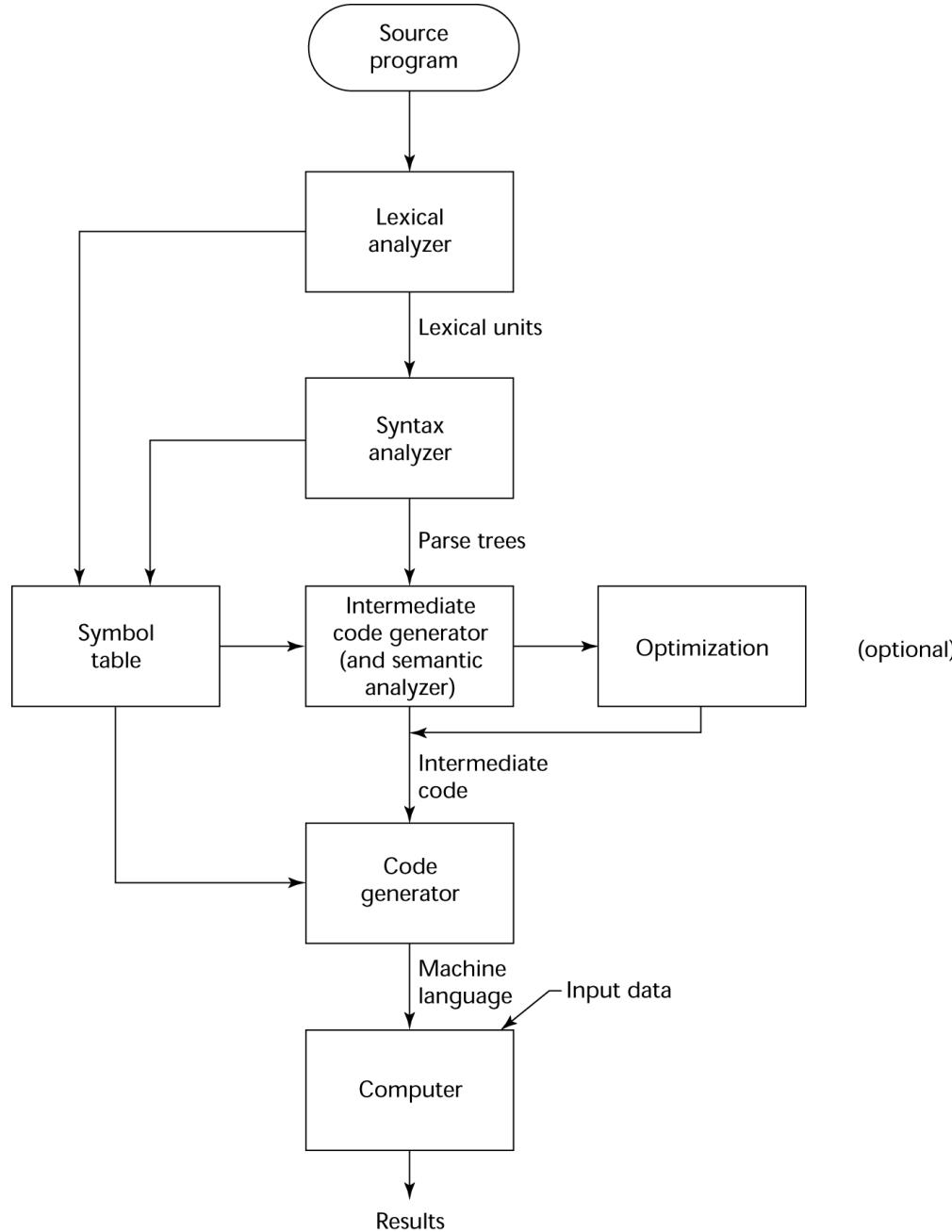
Figure 2.12 Header and library files.

Derleme Süreci

- Derleme sürecinin başlangıcında derleyiciye verilen yüksek düzeyli bir programlama dili deyimlerini içeren programa, **kaynak (source) program**, derleme sürecinin sonucunda oluşan makine dilindeki programa ise **amaç (object) program** adı verilir.

Derleme süreci

- Derleyicinin çalışması sırasında geçen zamana **derleme zamanı** (*compile time*) denir.
- Hedef programların çalışması sırasında geçen zamana **çalışma zamanı** (*run time*) adı verilir.



Sadeleştirilmiş Derleyici Yapısı

Source code

(Karakter Dizisi)
`if (b == 0) a = b;`

Metinsel

(Lexical) analiz

Token dizisi

Ayrıştırma

Soyut Sözdizim Ağacı

Ara Kod Üretimi

Ara Kod

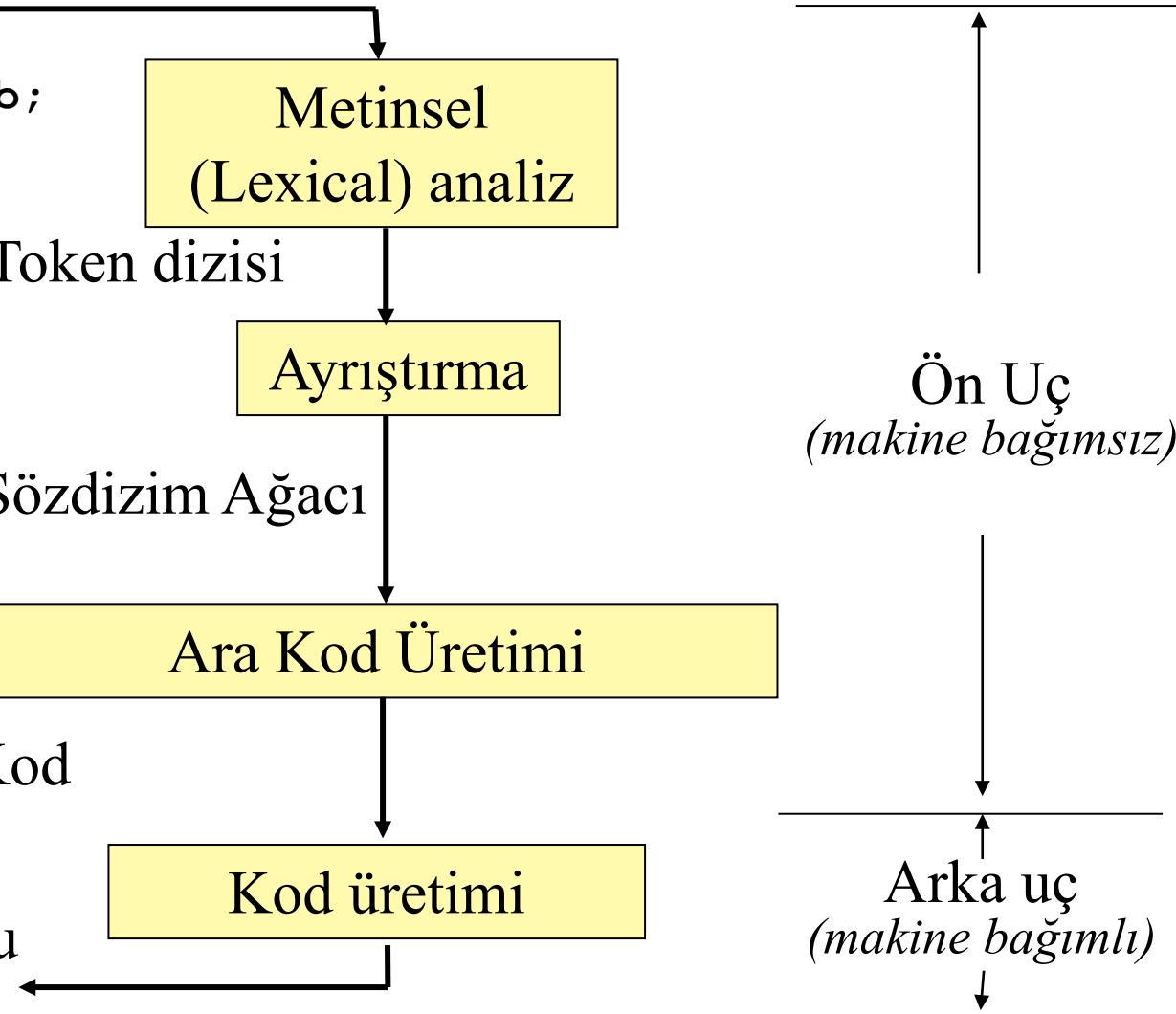
Kod üretimi

Assembly Kodu

`CMP CX,0
CMOVZ DX,CX`

Ön Uç
(makine bağımsız)

Arka uç
(makine bağımlı)

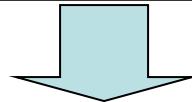


Metinsel Analiz-Scanner

- Derleme sürecindeki ilk ve en uzun süren aşamadır.
- Bir derleyicinin ön ucunda yer alan **metinsel çözümleyici** (*lexical analyzer*), bir kaynak programı bir dizi *token dizisine* çevirir.
- Regüler ifade aracı kullanır

Scanner

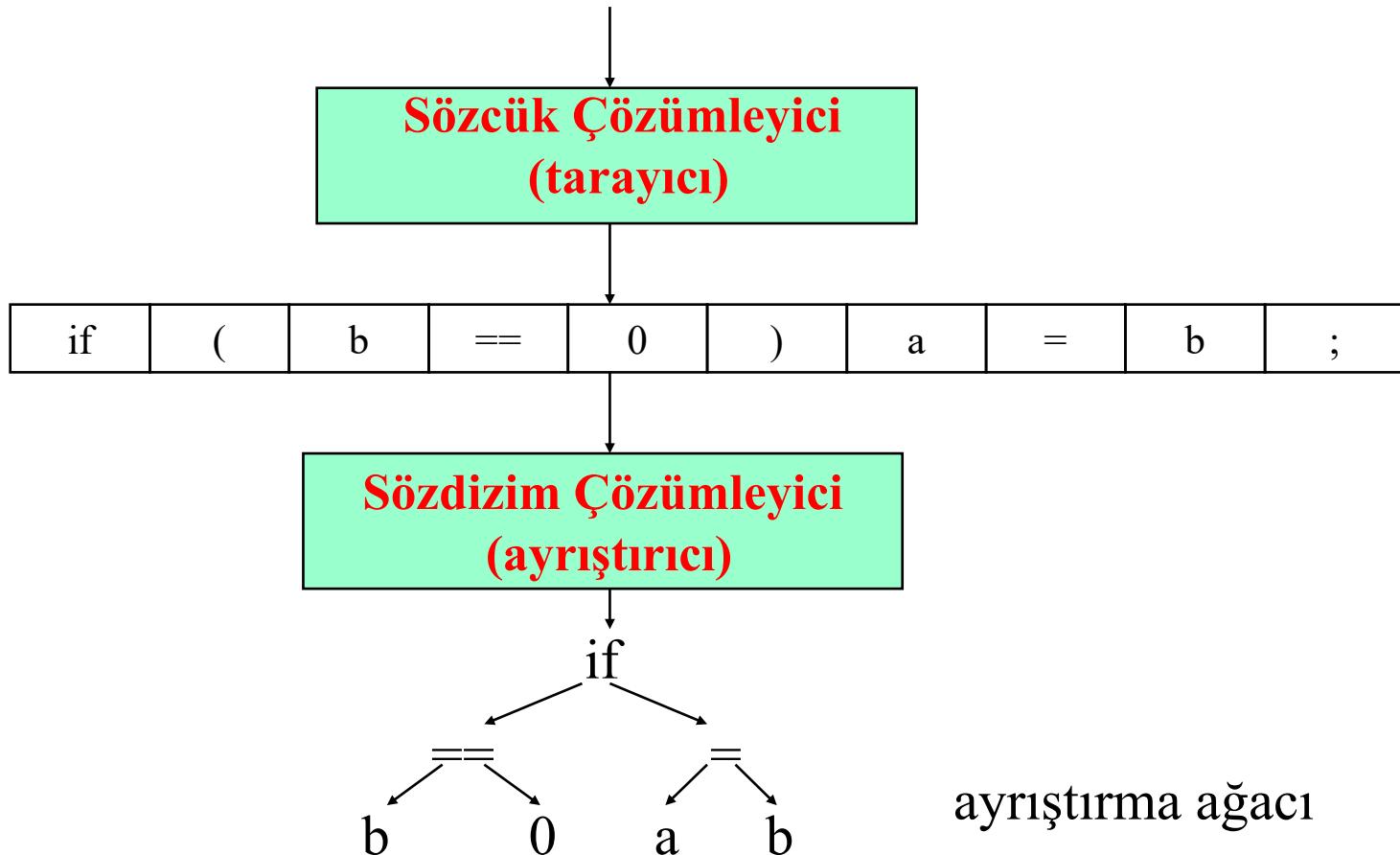
```
program gcd (input, output);
var i, j : integer;
begin
  read (i, j);
  while i <> j do
    if i > j then i := i - j else j := j - i;
  writeln (i)
end.
```



```
program gcd ( input , output ) ;
var i , j : integer ;
read ( i , j ) ;
i <> j do if i > j then i := i - j else j := j - i ;
writeln ( i )
) end .
```

Syntax Analiz

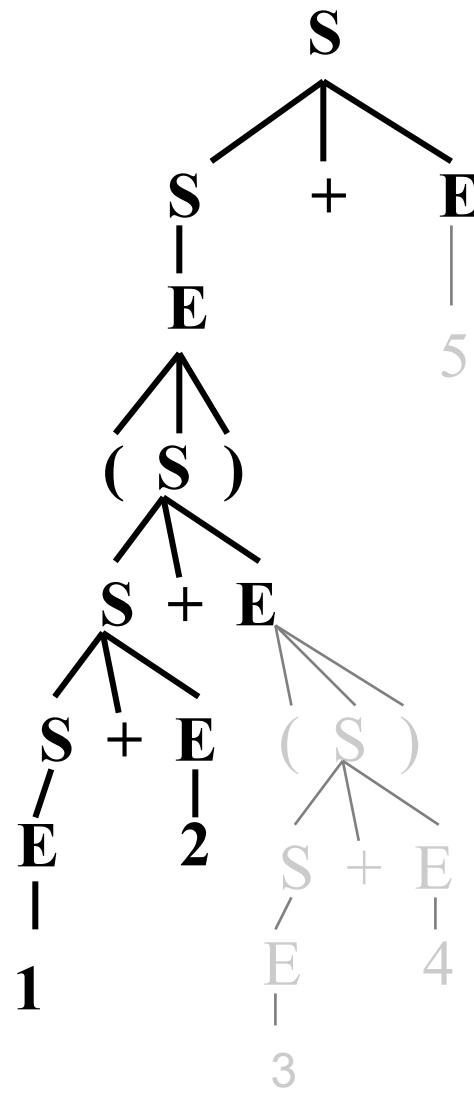
if (b == 0) a = b;



Ayrıştırma (Parser) yöntemleri

- Ayrıştırıcı bir sözcük katarının dilin gramer kuralları ile türetilebilirliğini inceler ve sözcük katarına ilişkin bir **ayrıştırma ağacı** oluşturur
- İki yol vardır.
 - Yukarıdan-aşağıya (top-down)
 - Aşağıdan-yukarıya (bottom-up)

Top-Down



Sembol Tablosu

- Derleme sürecinde programdaki her tanımlayıcı için bir eleman içeren **sembol tablosu** oluşturulur. Sembol tablosu, derleme sürecindeki çeşitli aşamalarda kullanılır ve güncellenir.
- Bir tanımlayıcı kaynak programda ilk kez bulunduğuanda, o tanımlayıcı için sembol tablosunda bir eleman oluşturulur. Aynı tanımlayıcının daha sonraki kullanımları için ilgili *token*, aynı sembol tablosu elemanına başvuru içerir.
- Metinsel çözümleme aşamasının sonunda, programdaki *token*'lar ve her *token*'ın özelliklerinin tutulduğu sembol tablosu elemanına işaret edilen göstergeleri içeren *token* dizisi oluşturulur

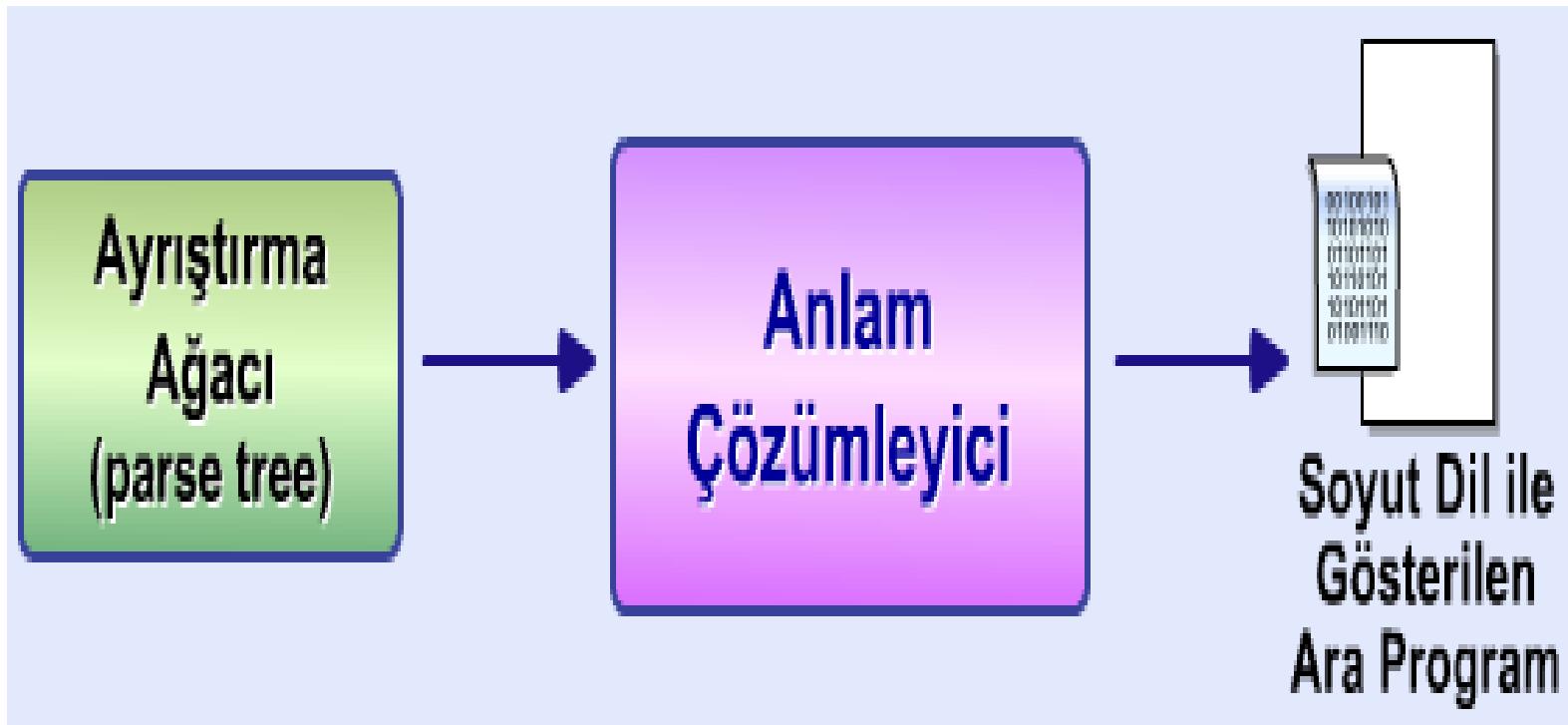
Sembol tablosu

Ortalama := toplam / 10		
Ortalama	(identifier, 1)	tanımlayıcı
:=	(atama,nil)	Atama işlemcisi
toplam	(identifier, 2)	Tanımlayıcı
/	(bolme,nil)	Bölme işlemcisi
10	(tamsayı, 3)	tamsayı

Anlam Çözümleme

- **Anlam çözümleme**, kaynak program için sözdizim çözümleme sırasında oluşturulmuş ayrıştırma ağacı kullanılarak, soyut bir programlama dilinde bir program oluşturulmasıdır.

Anlam Çözümleme



Soyut Dil

- Anlam çözümleme sonucunda üretilen kod için kullanılan ara diller, genel olarak, üst düzeyli bir birleştirici diline benzerler. Bu soyut dil, kaynak dilin veri türleri ve işlemleriyle uyumlu olacak şekilde tasarlanmış, hayali bir makine için bir makine dili olup, derleyicinin kaynak ve amaç dilleri arasında bir ara adım oluşturur.

Kod Oluşturma

- Soyut dilde ifade edilen kodu alınarak belirli bir bilgisayar için makine kodunu oluşturular.
- Derleyicinin ön ucu programlama diline bağlımlı, arka ucu ise bilgisayara bağlımlıdır.

Eniyileme (Optimizasyon)

- İsteğe bağlı olarak ara kod kısmında iyileştirmeler yapılabilir.
- Buradaki çalışma bir programın daha etkin olarak çalışacak bir eşdeğer programa dönüştürülmesi için yapılır.

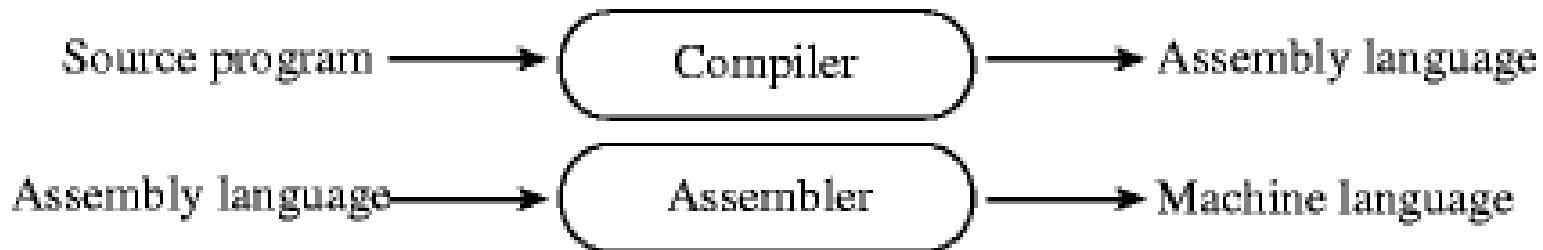
Dil Çevrim Yöntemlerinin Karşılaştırılması

Çevirici	Zaman	Hafıza	Hata Yakalama
Derleyici	+	-	-
Yorumlayıcı	-	+	+

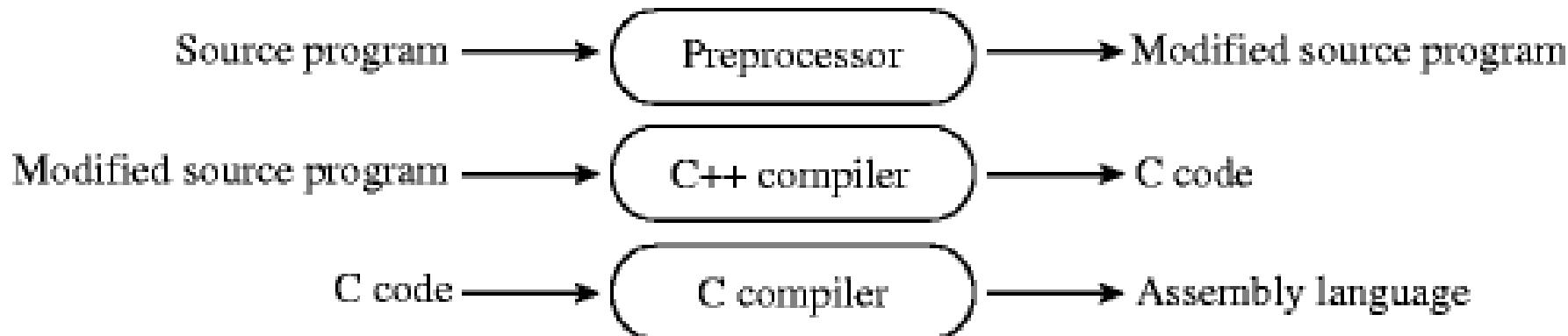
Taşınabilir Kod

- Dil çevriminde bütünüyle yorumlama ve bütünüyle derleme yöntemleri, iki uç durumdur. Bazı programlama dilleri, iki yöntemin birleştirilmesi ile gerçekleştirirler.
- Bir program, kaynak program üzerinde basit düzenlemeler yapılarak bir sanal makinenin daha sonra yorumlanacak olan makine kodu olarak nitelenebilen bir ara koda çevrilebilir. Bu çözüm, ağırlıklı olarak derlemeye dayanır ve farklı makinelerde çalıştırılabilen **taşınabilir kod** üretmek amacıyla kullanılabilir.

Taşınabilirlik



- Ara kod

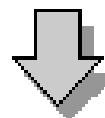


- Java dili bu tür dil çevrimini uygulamaktadır. Java programları, Java *bytecode*'u adı verilen bir koda dönüştürüldükten sonra yorumlanır.

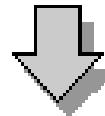
compile-time environment

Your program's source files

A.java B.java C.java



Java
compiler



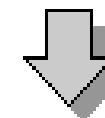
A.class B.class C.class

Your program's class files

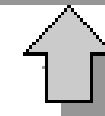
run-time environment

Your program's class files

A.class B.class C.class



Java
Virtual
Machine



Object.class String.class ...

Java API's class files

*Your
class files
move
locally
or through
a network*

Örnek

Örnek

read A

read B

sum := A + B

write sum

write sum / 2

Metinsel Analiz (Lexical Analysis)

- Tokens:

id = letter (letter | digit) * [except "read" and "write"]

literal = digit digit *

"`:=`", "`+`", "`-`", "`*`", "`/`", "`(`", "`)`"

`$$$` [end of file]

Sözdizim Analizi (Syntax Analysis)

- EBNF

```
<pgm>          -> <statement list> $$$  
<stmt list>   -> <stmt list> <stmt> | E  
<stmt>          -> id := <expr> | read <id> | write <expr>  
<expr>          -> <term> | <expr> <add op> <term>  
<term>          -> <factor> | <term> <mult op> <factor>  
<factor>        -> ( <expr> ) | id | literal  
<add op>        -> + | -  
<mult op>       -> * | /
```

Ara Kod Dönüşümü

- Intermediate code:

```
read
pop A
read
pop B
push A
push B
add
pop sum
push sum
write
push sum
push 2
div
write
```

Ara Kod Dönüşümü

- Target code:

```
.data
A:    .long 0
B:    .long 0
sum:   .long 0
.text
main:  jsr read
       movl d0,d1
       movl d1,A
       jsr read
       movl d0,d1
       movl d1,B
       movl A,d1
```

Kod üretimi

```
movl    B,d2  
addl    d1,d2  
movl    d1,sum  
movl    sum,d1  
movl    d1,d0  
jsr write  
movl    sum,d1  
movl    #2,d2  
divsl   d1,d2  
movl    d1,d0  
jsr write
```

Bölüm Özeti

- Dil çevrimi
- Dil çevriminin Aşamaları
- Dil çevrim Yöntemleri
- Dil çevrim yöntemlerinin karşılaştırılması

Programlama Dillerinin Temel Elemanları

Değişkenler

Sabitler

İşlemciler

İfadeler

Deyimler

Von Neumann mimarisi,
her bellek hücresinin özgün bir adres ile
tanımlandığı ana bellek kavramına
dayanmaktadır.

GİRİŞ (devam)

- Bir bellek hücresinin içeriği, bir değerin belirli bir yönteme göre kodlanmış gösterimidir.
- Bu içerik, programların çalışması sırasında okunabilir ve değiştirilebilir.
- *Imperative* programlama, von Neumann mimarisindeki bilgisayarlara uygun olarak
- programların işlem deyimleri ile bellekteki değerleri değiştirmesine dayanır.

Bir değişken, bir veya daha çok bellek hücresinin soyutlamasıdır.

DEĞİŞKEN ÖZELLİKLERİ

İSİM	okunabilirlik
ADRES	Aliasing
DEĞER	Bellekte belirli yönteme göre kodlanmış
TİP	Tip uyusuzluğu, tip dönüşümü
YAŞAM SÜRESİ	Bellekle ilişkili kaldığı süre
KAPSAM	Geçerli olduğu deyimler

İsimler

- İsimler, programlama dillerinde, değişkenlerin yanı sıra, **etiketler**, **altprogramlar**, **parametreler** gibi program elemanlarını tanımlamak için kullanılırlar.

- İsimleri tasarlamak için programlama dillerinde
 - EN FAZLA UZUNLUK
 - BÜYÜK-KÜÇÜK HARF DUYARLILIĞI
 - ÖZEL KELİMELER
- gibi farklı yaklaşımalar uygulanmaktadır

En Fazla Uzunluk

- Programlama dillerinde bir ismin en fazla kaç karakter uzunluğunda olabileceği konusunda farklı yaklaşımalar uygulanmıştır.

Önceleri programlama dillerinde bir isim için izin verilen karakter sayısı daha sınırlı iken, günümüzdeki yaklaşım, en fazla uzunluğu kullanışlı bir sayıyla sınırlamak ve çoklu isimler oluşturmak için altçizgi "_" karakterini kullanmaktadır

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

Küçük-Büyük Harf Duyarlılığı (Case Sensitivity)

- Birçok programlama dilinde, isimler için kullanılan küçük ve büyük harfler arasında ayrılmazken
- Bazı programlama dilleri (Örneğin; C, C++, Java) isimlerde küçük-büyük harf duyarlığını uygulamaktadır

- Bu durumda, aynı harflerden oluşmuş isimler derleyici tarafından farklı olarak algılanmaktadır
- *TOPLAM*, *toplam*, ve *ToPlaM*, üç ayrı değişkeni göstermektedir

Özel Kelimeler

- Özel kelimeler, bir programlama dilindeki temel yapılar tarafından kullanılan kelimeleri göstermektedir.
- A)Anahtar kelimeler(Keywords)
- B)Ayrılmış kelimeler(Reserved words)

Anahtar Kelime

- Bir anahtar kelime (*keyword*), bir programlama dilinin sadece belirli içeriklerde özel anlam taşıyan kelimelerini göstermektedir.
- Örneğin FORTRAN'da *REAL* kelimesi, bir deyimin başında yer alıp, bir isim tarafından izlenirse, o deyimin tanımlama deyimi olduğunu gösterir. (*REAL apple*)
- Eğer *REAL* kelimesi, atama işlemcisi "=" tarafından izlenirse, bir değişken ismi olarak görülür. *REAL = 10.05* gibi.
- Bu durum dilin okunabilirliğini azaltır.

Ayrılmış Kelime:

- Öte yandan, ayrılmış kelime (*reserved word*), bir programlama dilinde bir isim olarak kullanılamayacak özel kelimeleri göstermektedir.
- C++ dilindeki do, for , while gibi ve
- PASCAL'da procedure, begin, end gibi kelimere ayrılmış kelime denir.

Veri Tipi Kavramı

- Bir **veri tipi**, aynı işlemlerin tanımlı olduğu değerler kümesini göstermektedir. Bir değişkenin tipi, değişkenin tutabileceği değerleri ve o değerlere uygulanabilecek işlemleri gösterir. Örneğin; tamsayı (*integer*) tipi, dile bağımlı olarak belirlenen en küçük ve en büyük değerler arasında tamsayılar içerebilir ve sayısal işlemlerde yer alabilir.
- Veri tipleri, programlama dillerinde önemli gelişmelerin gerçekleştiği bir alan olmuş ve bunun sonucu olarak, programlama dillerinde çeşitli veri tipleri tanıtılmıştır.
- Tipler *basit tipler* ve bileşik tipler olarak gruplandırılabilir.

- Basit tipler
- İlkel tipler, çoğu programlama dilinde yer alan ve diğer tiplerden oluşmamış veri tiplerini göstermektedir.
- Tam sayı, Mantıksal, Karakter, Karakter katarı, Kullanıcı tanımlı tipler örnek olarak verilebilir.

- Bileşik Tipler
- Çeşitli veri tiplerinde olabilen bileşenlerden oluşmuştur.
 - Dizi, Kayıt, Pointer

Tablo Temel C++ değişken tipleri

	Numerik aralık			Bellek alanı
Keyword	Alt sınır	Üst sınır	Ondalık kısım	byte
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	3.4×10^{-38}	3.4×10^{38}	7	4
double	1.7×10^{-308}	1.7×10^{308}	15	8
long double	3.4×10^{-4932}	1.1×10^{4932}	19	10

SABİTLER

- Bir **sabit**, belirli bir tipteki bir değerin kodlanmış gösterimini içeren ancak programın çalıştırılması sırasında değiştirilemeyen bellek hücresına veya hücrelerine verilen isimdir.
- **isimlendirilmiş sabit**
- **Okunabilirliğe olumlu katkı**
- Pascal'da *const* ve C'de *#define* kullanılır

İŞLEMÇİLER

Genel Özellikler	İşlenen sayısı İşlemcinin yeri İşlem önceliği Birleşme Özelliği
İşlenenlere (Niteliğine) göre	Sayısal işlemciler İlişkisel işlemciler Mantıksal işlemciler

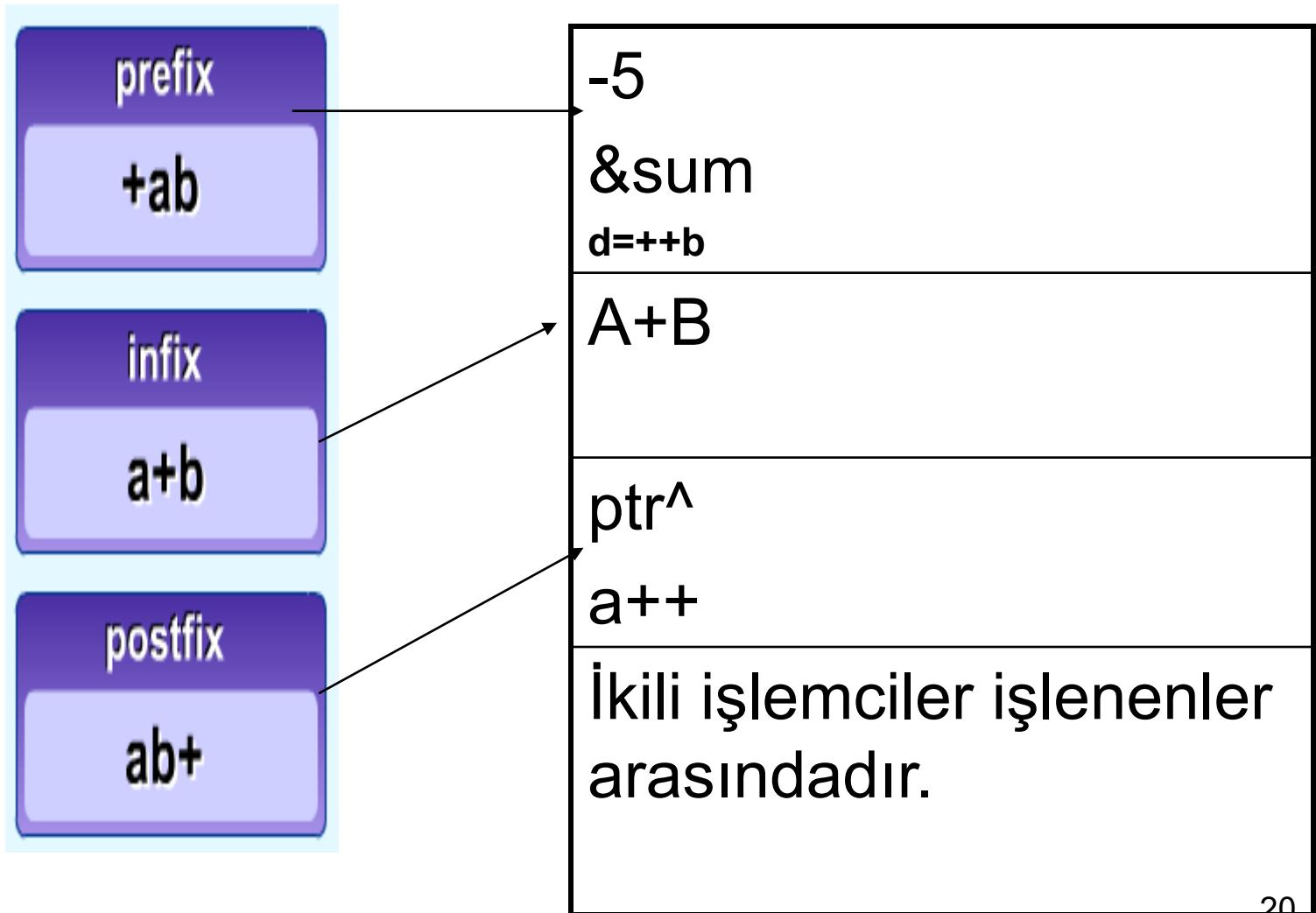
Tekli işlemler

- Yalnızca tek değişkenlere uygulanırlar..

$i = +1;$

$j = -i;$

İşlemcinin yeri (devam)



Öncelik

	FORTAN	PASCAL	C	ADA
Enyüksek öncelik	** (exponentation)	*, / , div , mod	++, -- (postfix)	** , abs
	* , /	+ , -	++, -- (prefix)	*, /, mod
	+ , -		Tekli (unary) +, -	Tekli(unary) +, -
			* , /, %	İkili(Binary)+, -
En düşük öncelik			İkili(Binary)+, -	

Associativity

Programlama Dili	Birleşme Kuralı	İşlemciler
FORTRAN	Sol Birleşmeli	$*, /, +, -$
	Sağ Birleşmeli	$**$
PASCAL	Sol Birleşmeli	Bütün işlemciler
	Sağ Birleşmeli	
C	Sol Birleşmeli	Postfix++, postfix--, *, /, %, ikili +, ikili-
	Sağ Birleşmeli	Prefix++, prefix--, tekli +, tekli-
C++	Sol Birleşmeli	$, /, %$, ikili +, ikili-
	Sağ Birleşmeli	$++, --$, tekli +, tekli-
ADA	Sol Birleşmeli	$**$ dışındakiler
	Sağ Birleşmeli	$**$ birleşme özelliği yok

NİTELİĞİNE GÖRE İŞLEMCİLER

SAYISAL İŞLEMCİLER

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

İLİŞKİSEL İŞLEMCİLER

Anlamı	C++	PASCAL
Büyüktür	>	>
Küçüktür	<	<
Eşittir	==	=
Eşit değildir	!=	<>
Büyük veya eşittir	>=	>=
Küçüktür veya eşittir	<=	<=

MANTIKSAL İŞLEMCİLER

C dili Mantıksal İşlemcileri		
&&	VE	AND
	VEYA	OR
!	DEĞİL	NOT



İşlemcilerin öncelikleri

- Sayısal ifadeler, ilişkisel ifadelerin işlenenleri olabileceği ve ilişkisel ifadeler de Boolean ifadelerin işlenenleri olabileceği için, üç işlemci grubunun kendi aralarında öncelikleri vardır.
- İlişkisel işlemcilerin önceliği, her zaman sayısal işlemcilerden düşüktür.
 - $X+20 \leq k^2$
- İlişkisel ifadeler ise mantıksal ifadeler için bir operand olabileceğinden ilişkisel ifadeler mantıksal ifadelerden önce yapılmalıdır.

ADA Programlama Dili

En yüksek	**, abs, not	(A<B) and (A>C) or X=0 Doğru
	*, /, mod, rem	
	+, - (tekli)	A<B and A>C or X=0 Yanlış
	+, -, & (ikili)	
	=, /=, >, <, <=, >=, in, not in	
Endüştük	AND, OR, XOR, AND THEN, OR ELSE	

İşlemci Yükleme

- İşlemcilerin anımlarının, işlenenlerin sayısına ve tipine bağlı olarak belirlenmesine **işlemci yüklemesi** (*operator overloading*) denir.
- "+", hem tamsayı hem de kayan-noktalı toplama için kullanılır ve bazı dillerde, sayısal işlemlere ek olarak karakter dizgilerin birleştirilmesi için de kullanılır.
- Okunabilirlik zayıflıyor.

İFADELER

- **Ifadeler**, yeni değerler oluşturmak için değerleri ve işlemcileri birleştirmeye yarayan sözdizimsel yapılardır.
- Bir ifade, bir sabit, bir değişken, bir değer döndüren bir fonksiyon çağrımı veya bir işlemciden oluşabilir.
 - SAYISAL İFADELER ($a+b*c$)
 - İLİŞKİSEL İFADELER ($a>=b$)
 - MANTIKSAL İFADELER ($(A>10) \text{ and } (C<2)$)

DEYİMLER

- **Deyimler**, bir programdaki işlemleri göstermek ve akışı yönlendirmek için kullanılan yapılardır.
- **Basit Deyimler**(atama deyimi)
- **Birleşik Deyimler** ((*if-then- else* ve *case* deyimleri ve *while* ve *for* deyimleri gibi

Atama İşlemi

- Programlama dillerinde atama sembolünün anlamı, **sağ taraftaki değerin sol taraftaki değişkene aktarılmasıdır.**
 - `=, :=, ==`

Atama deyimi

- <hedef_değişken> <atama_islemcisi> <ifade>

Sum=++count

Count=count+1
Sum=count

Sum, total=0

Sum=0 ve total=0

puan+=500;

Puan=Puan+500

Sum=count ++

Sum=count
Count=count+1

count ++

count+1

F ? count1:count2=0

F=1 ise count1=0
F=0 ise count2=0

Özet

- Temel programlama dili elemanları olarak **değişkenler, sabitler, işlemciler, ifadeler ve deyimler** incelenmiştir.
- İşlemcilerin sınıflandırılmış (sayısal işlemciler, ilişkisel işlemciler, mantıksal işlemciler) ve işlemci yükleme kavramı açıklanmıştır.
- Atama deyimi farklı sözdizimlerle örneklenmiştir.

Hedef

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

GİRİŞ

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

Temel Veri Tipleri-Bileşik veri tipleri

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

Temel Veri Tipleri

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

Sayısal Tipler

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

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

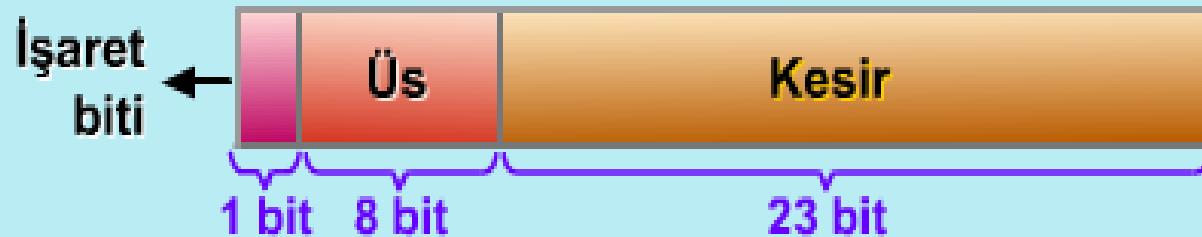
Integer (Tamsayı)

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

Floating point (Kayan Noktalı)

- Kayan noktalı (*floating point*) veri tipleri, **gerçel sayıları** modellerler.
- Kayan noktalı sayılar, kesirler ve üsler olarak iki bölümde ifade edilirler.
- Kayan noktalı tipler, duyarlılık (*precision*) ve alan (*range*) açısından tanımlanırlar.
- Duyarlılık, değerin kesir bölümünün tamlığıdır. Alan ise, kesirlerin ve üslerin birleşmesidir.

Gerçek Tip



Çift Duyarlılık Tipi



Decimal (Onlu)

- **Onlu** (*decimal*) veri tipi, ondalık noktanın sabit bir yerde bulunduğu sabit sayıda onlu basamak içeren bir veri tipidir. Bu veri tipi, az sayıda programlama dilinde (Örneğin; PL/I) tanımlanmıştır.
- Onlu veri tipi, onlu değerleri tam olarak saklayabilirse de, üsler bulunmadığı için gösterilebilecek değer alanı sınırlıdır. Her basamak için bir sekizli (byte) gereklili olması nedeniyle, belleği etkin olarak kullanmaz.

Boolean (Mantıksal)

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

Mantıksal (devam)

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

Character (Karakter)

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

Örnek

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

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

Character String (Karakter Dizgi)

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

Karakter Dizgilerin Uzunlukları

Durağan Uzunluk	$ch_1 + ch_2 + ch_3 + ch_4$
	77, FORTRAN 90, COBOL, Pascal, Ada
Değişen Uzunluk (üst sınır var)	$ch_1 + ch_2 + ch_3 + \dots + ch_n$
	PL/I, C, C++
Değişen Uzunluk (üst sınır yok)	$ch_1 + ch_2 + ch_3 + ch_4 + \dots$
	Dinamik bellek yönetimi olan diller

Kullanıcı Tanımlı Sıralı Tipler

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

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

Enumeration (Sayılama) Tipleri

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

Sayılama (devam)

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

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

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

```
type renkbilgisi=(kırmızı, mavi, yeşil, sarı);  
var renk:renkbilgisi;
```

```
...  
renk:=mavi;  
if renk > red; .....
```

Typedef enum {soguk, ılık, sıcak} ISI;

C

PASCAL

Java dilinde sayılama veri tipi bulunmamaktadır.

Güvenilirlik, okunabilirlik

Subrange (Altalan) Tipleri

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

Type

Buyukharfler ='A'..'Z'; → Buyukharfler, tek karakterler için dilde tanımlı olan char tipinin altalani olarak,

Puanlar =50..100; → Puanlar ise integer tipinin altalani olarak tanımlanmıştır.

Hem programlama dillerinin okunabilirliğine hem de güvenilirliğine olumlu katkı

Diziler

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

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

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

Dizi İndisleri

- **İndisin Üst Sınırı:**
- İlk programlama dillerinin aksine, günümüzde popüler olan programlama dillerinde dizi indislerinin sayısı sınırlanmaz.
- C'de ise diğer programlama dillerinden farklı bir tasarım vardır. C'de dizilerin tek indis olabilir. Ancak, dizilerin elemanları diziler olabileceği için çok boyutlu diziler oluşturulabilir. (int mat[2][3];)

İndisin Alt Sınırı:

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

Dizi Tiplerinin Gerçekleştirimi

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

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

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

Dizilerin Belleğe Eşleştirilmesi



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

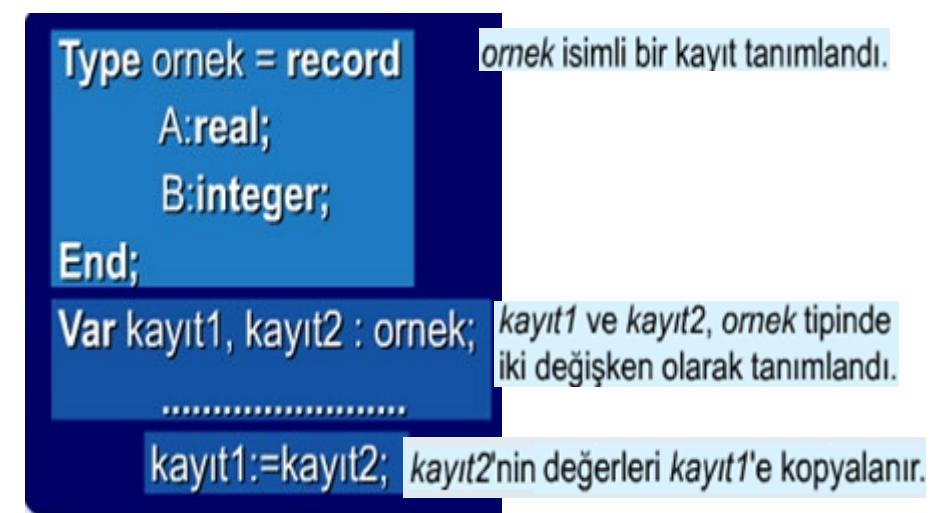
$$\text{Adres}([a(i,j)]) = \text{adres}[1,1] + ((i-1)*n) + (j-1)*c$$

Record (Kayıt) Tipi

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

Pascal dilinde kayıt veri tipi aşağıdaki söz dizimi ile tanımlanır.

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



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

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

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

PASCAL

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

  with kayit Do
  Begin
    Isim :=Index;
    Meslek:='Doktor';
    Telefon:='123456';
    Adres:='Kayseri';
    ...
    ...
  End.
```

C

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

Union (Bileşim) Tipi

Aynı bellek bölgesinin farklı değişkenler tarafından kullanılmasını sağlayan veri tipidir.

Bu veri tipindeki değişkenlere ortak değişkenler denir.

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

İsim	
ad	soyad
15 byte	15 byte

struct

İsim
ad
soyad
15 byte

union

Set (Küme) Tipi

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

tanımına göre A kümesi
şu 4 kümeden biri olabilir:

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

A + B → Küme bileşimi

A - B → Küme farkı

A * B → Küme kesişimi

PASCAL'da Küme Tanımlaması

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

PASCAL'da Küme Tipindeki Değişkenlerin Kullanımı

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

If arabam in kume1 then begin

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

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

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

kesişim *
Birleşim +
fark işlemler -
mantıksal operatörler (=, <> , <= , >=, in)

Pointer (İşaretçi) Tipi

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

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

```
#include <stdio.h>
void main()
{ int a=4, b=7; //int tipinde a ve b değişkenleri tanımlanmış ve 4 ve 7 değerleri atanmıştır.
  int *pa,*pb; // pa ve pb adında, yine int tipinde iki işaretçi değişken tanımlanmıştır
  printf("a=%d , b=%d\n",a,b);// a = 4 , b = 7 olarak ekrana yazdırır.
  pa=&a;           //a değişkeninin adresini pa işaretçi değişkenine ata
  pb=&b;           //b değişkeninin adresini pb işaretçi değişkenine ata
  *pa=*pa+10;      // pa'nin işaret ettiğleri değere 10 değerini ekler
  *pb=*pb+10;      // pb'nin işaret ettiğleri değere 10 değerini ekler
  printf("a=%d, b=%d",a,b);    }
```

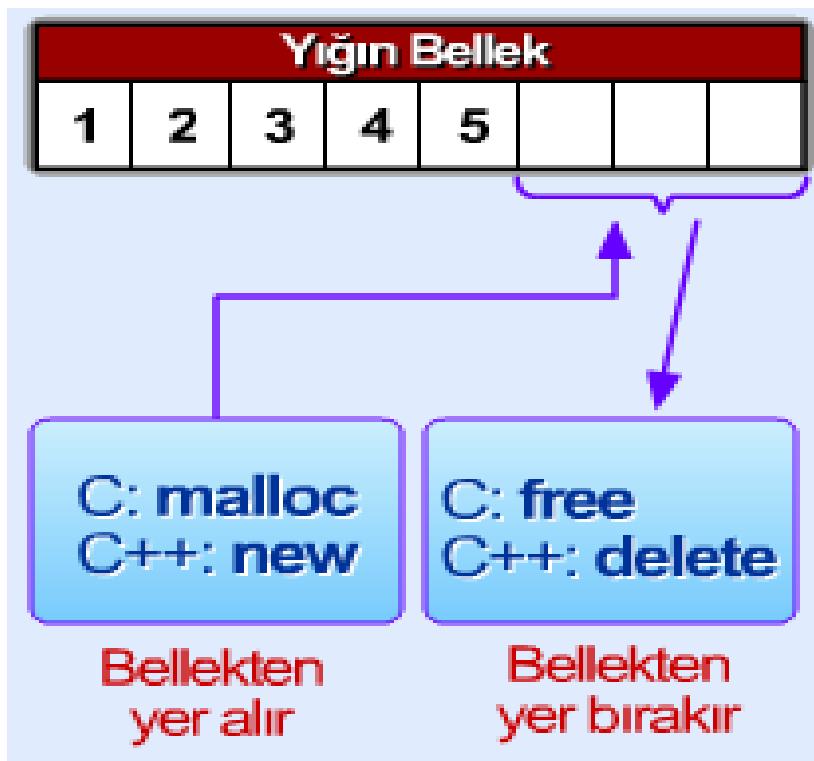
Burada ***pa=*pa+10; *pb=*pb+10;** ifadelerinde **pa** ve **pb** nin işaret ettiğleri değerlere sırasıyla 10 değerini ekler. **pa** işaretçi değişkeni **a** değişkenine ve **pb** işaretçi değişkeni ise **b** değişkenini işaret ettiğine göre aslında değeri artırılan değişkenler **a** ve **b** dir.

Bu programın ekran çıktısı
a = 4 , b = 7
a = 14 , b = 17 olur.

İşaretçi Veri Tipinin Sorunları

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

Dinamik bellek yönetimi



Örnek

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

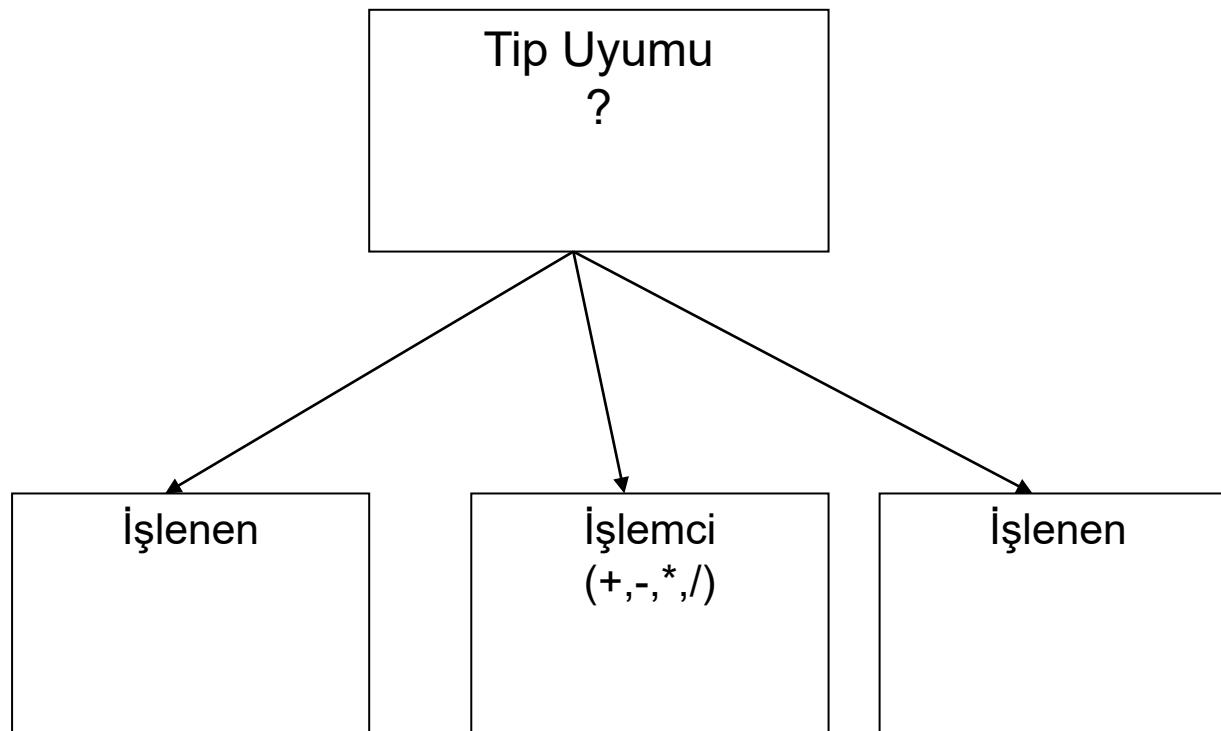
PASCAL: new, dispose

KUVVETLİ TİPLEME

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

TİP DENETİMİ

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



Tip Dönüşümleri

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

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

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

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

Çoğu programlama dili daralan dönüşümlere izin vermezler.

Güvenilirlik olumsuz etkileniyor.

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

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

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

derleyici, b yi *float* tipine dönüştürecek ve çarpım işlemi *float* olarak gerçekleşecektir. Zorunlu dönüşüm, tip uyusuzluğu hmasını engellemiştir.

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

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

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

Dışsal Tip Dönüşümleri

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

Toplam=(int)toplams2 + 100

Bu ifadedeki toplams2 değişkeni önce int tipine dönüştürülür daha sonra 100 ile toplanır.

Bölüm Özeti

(veri tipi kavramı)

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

Bağlama ve Kapsam Kavramları

İçerik

- Bağlama Kavramı
 - Bağlama zamanı,
 - Tip bağlama, Bellek bağlama
- İsim Kapsamları
 - Durağan Kapsam Bağlama
 - Dinamik Kapsam Bağlama konuları

Bağlama(Binding)

- Bir özellikle bir program elemanı arasında ilişki kurulmasına **bağlama (binding)** denir.



- Çeşitli programlama dilleri, özelliklerin program elemanlarına **bağlanma zamanı** ve bu özelliklerin **durağan (static)** veya **dinamik (dynamic)** olması açısından farklılıklar göstermektedir.

Bağlama Zamanı

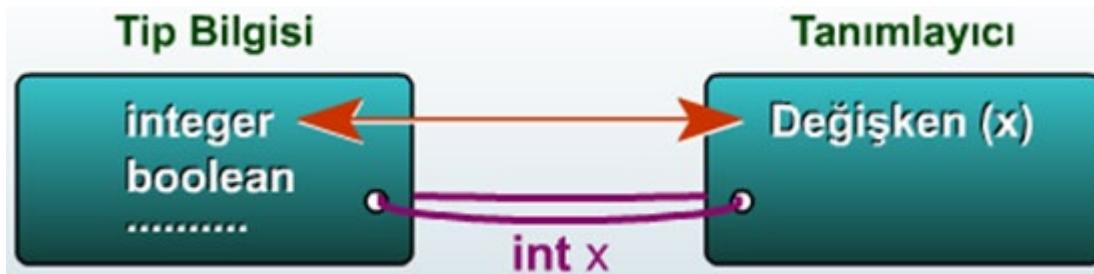
- Bir programlama dilinde çeşitli bağlamalar farklı zamanlarda gerçekleşebilir.



int hesap; ... hesap=hesap+10;	
Hesap için olası tipler	Dilin tasarım zamanında
Hesap değişkeninin tipi	Dilin derlenmesi zamanında
Hesap değişkeninin olası değerleri	Derleyici tasarım zamanı
Hesabın değeri	Bu deyimin yürütülmesi zamanında
+ işlemcisinin muhtemel anlamları	Dilin tanımlanması zamanında
+ işlemcisinin bu deyimdeki anlamı	Derlenme süreci
10 literalinin ara gösterimi	Derleyici tasarımları zamanında
Hesap değişkeninin alacağı son değer	Çalışma zamanında

Tip Bağlama

- Bir tanımlayıcı (id) bir tip bilgisi, 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.



Bağlama Zamanı

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

Örtülü (*implicit*)

ve

Dışsal (*explicit*)

Değişkene, programda yer alan bir tanımlama deyimi ile bir tip ile bağlanır.

Tanımlama deyimleri kullanılmaz ve değişkenlerin tipleri, varsayılan (*default*) kurallar ile belirlenir

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:son karakteri \$ ola değişkenler karakter tipi ile bağlanır.

Yazım yanlışlığı hataların derleme sırasında yakalanması engellenebilir.
Programlama dilinin güvenilirliğini azaltırlar.

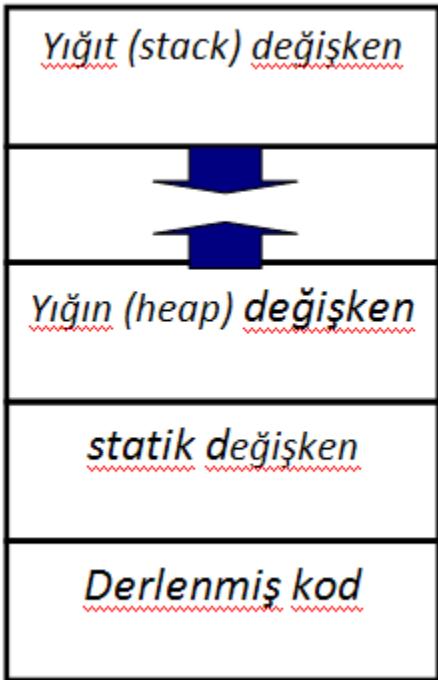
```
procedure D;  
var n: char;  
begin  
n:= "D";  
W;  
end;
```

```
{ int t=a;  
a=b;  
b=t;  
}
```

Örnek: PL/I, BASIC, PERL ve FORTRAN

Bellek Bağlama

- (*allocation*) (*deallocation*) lifetime



etkinlik (activation) kaydı

aynı bellek bölümünün
yeniden kullanılabilmesi

Doğrudan adresleme

Pascal-*dispose* Java-otomatik
C'deki malloc fonksiyonu
C++ 'daki new işlemcisi

Statik değişkenler, programın yürütülmesi başlamadan bellek hücrelerine bağlanırlar ve bellek hücreleri ile programın çalışması sonlanıncaya kadar bağlı kalırlar. FORTRAN I, II ve FORTRAN IV'de hepsi statik. C, C++ ve Java **static** anahtarını kullanır.

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.

Dışsal yığın dinamik değişkenlerin bellek yeri bağlanması çalışma zamanında gerçekleşir. Ne kadar bellek gerektiği önceden bilinmez. 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, bellek yeri bağlanması ise çalışma zamanında gerçekleşir.

	<i>Statik</i>	<i>Stack</i>	<i>Heap</i>
Ada		Lokal değişkenler, altprogram parametreleri	<i>implicit</i> : local değişkenler; <i>explicit</i> : new (garbage collection)
C	global değişkenler; statik local değişkenler	Lokal değişkenler, altprogram parametreleri	<i>explicit</i> : malloc ve free
C++	C ile aynı, static sınıf üyeleri	C ile aynı	<i>Explicit</i> : new ve delete
Java		Sadece ilkel tipli local değişkenler	<i>Implicit</i> : her sınıf(garbage collection)
Fortran 7	global değişkenler(bloklar), lokal değişkenler ve altprogram parametreleri (implementation dependent); SAVE , static bellek atamasını düzenler	Lokal değişkenler, altprogram parametreleri (implementation dependent)	
Pascal	global değişkenler(compiler bağımlı)	global değişkenler(compiler dependent), local değişkenler altprogram parametreleri	<i>Explicit</i> : new ve dispose

İSİM KAPSAMLARI (Name Scope)

- Belirli isim tanımlarının etkin olduğu bir program alanına **isim kapsamı** denir.



Statik isim kapsam

Dinamik Kapsam

Değişkenlerin kapsamları, programın metinsel düzeneğe göre, fiziksel yakınlığa göre, belirlenir.

Bir ismin kapsamının, altprogramların fiziksel yakınlıklarına göre değil, altprogramların çağrılmama sırasına göre çalışma zamanında belirlenmesi **dinamik kapsam bağlama** olarak adlandırılır.

ALGOL 60'ı izleyen çok sayıda dilde tanımlıdır. Altprogramlar iç içe yuvalanabilir. (C++ ve FORTRAN hariç)

LISP, APL dillerinin ilk sürümleri

1. Altprogramların yuvalanması sonucu gereğinden fazla genel değişken kullanımı olabilir.
2. Bir programda genel olarak tanımlanan değişkenler tüm altprogramlara görünebilir olacakları için güvenilirlik azalmaktadır.

1. Bir altprogramda bir değişkene yapılan başvuru, deyimin her çalışmasında farklı değişkenleri gösterebilir.
2. Programların anlaşılabilirliğini azaltmaktadır

ÖRNEK

statik kapsam bağlama kuralına göre

```
program L;
    var n: char;           {n, L' de bildirilmiş }

procedure W;
begin
    writeln(n)   {W de n 'ye başvuru var.}

end;

procedure D;
    var n: char;           {D de n tekrar bildirilmiş}

begin
    n:= "D";
    W;                    { D' deki W' i çağırdı}

end;

begin {L}
    n:= "L";
    W;                  {Ana program L' den W' u çağrıldı}
    D;
end.
```

L
L

Dinamik kapsam bağlama kuralına göre

L
D

Örnek: Aşağıdaki program parçasının çıkışını

- a) statik kapsam bağlama kurallarına göre
- b) Dinamik kapsam bağlama kurallarına göre bulunuz.

```
int x;

int main() {
    x = 2;
    f();
    g();
}

void f() {
    int x = 3;
    h();
}

void g() {
    int x = 4;
    h();
}

void h() {
    printf("%d\n",x);
}
```

Dinamik kapsam bağlamaya göre çıkış: 3 4

Statik kapsam bağlamaya göre çıkış: 2 2

Özet

- Bu hafta
- Programlama elemanlarıyla çeşitli özelliklerinin ilişkilendirilmesini sağlayan bağlama konusu, durağan-dinamik tip, bellek ve isim kapsamı bağlama yönüyle incelemiştir.
- Isim kapsamlarının durağan kapsam bağlama ve dinamik kapsam bağlama seçenekleri incelemiştir ve karşılaştırılmıştır.



YAPISAL PROGRAMLAMA



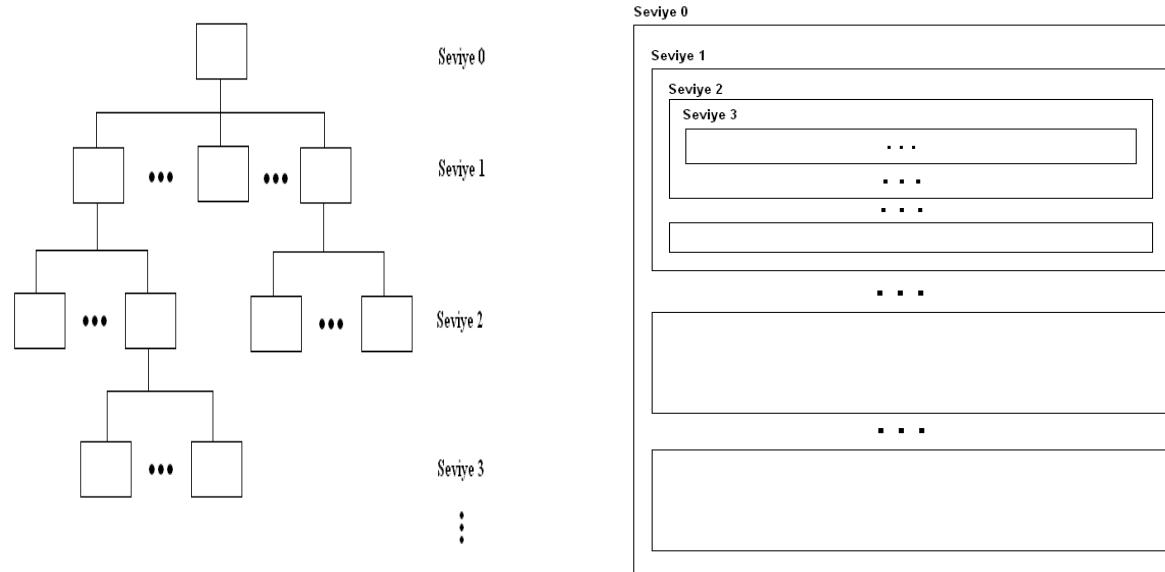
YAPISAL PROGRAMLAMA KAVRAMI

- Yapısal programlama, program tasarıımı ve yazılmasını kurallara bağlayan ve disiplin altına alan bir yaklaşımdır.
- Yapısal programlamada problem çözümü daha kolay alt problemlere (modül) bölünür. Her bir alt problem (modül) daha düşük seviyedeki alt seviyelere bölünür.



YAPISAL PROGRAMLAMA KAVRAMI

- Bu işlem, aşağıdaki şekilde de görülebileceği gibi her bir modülün kolaylıkla çözülebileceği seviyeye kadar devam eder.



- En üst seviyede çözümün ana mantığının sergilendiği ana modül yer alır. Alt seviyelere indikçe izlenecek adımlar ile ilgili ayrıntılar artar.



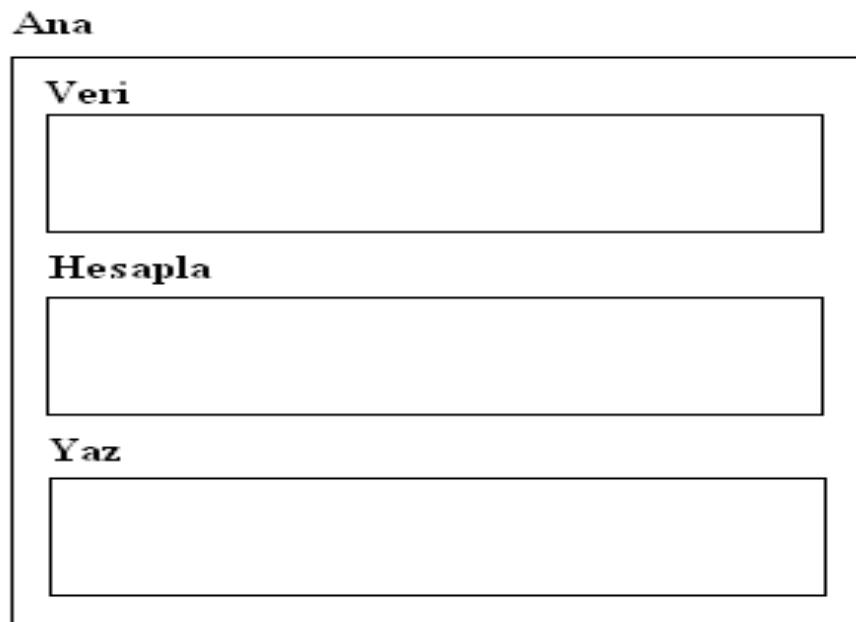
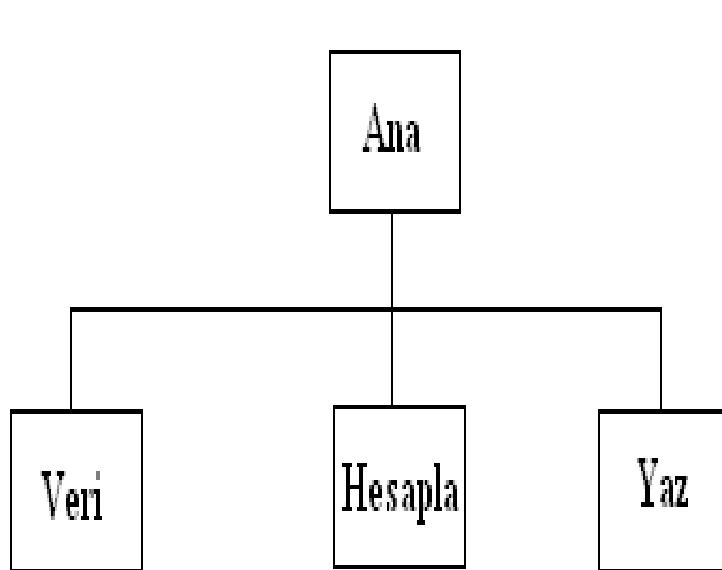
YAPISAL PROGRAMLAMA KAVRAMI

- Modüler program tasarımında her modül diğerlerinden bağımsız olmalıdır. Kontrol her modüle bir üst seviyedeki modülden geçmeli ve modül işlendikten sonra tekrar aynı modüle iletilmelidir.
- Modüllerin tanımlanmasında, (algoritma parçası olduğu için) sözde kod (pseudo-code) veya akış diyagramı kullanılır.
- Bir modülün çözümünde kullanılacak algoritma, sözde kod ile veya kullanılacak programlama dilinin yapısına uygun bir şekilde akış diyagramı ile ifade edilirse programa geçiş büyük ölçüde kolaylaşır.



YAPISAL PROGRAMLAMA KAVRAMI

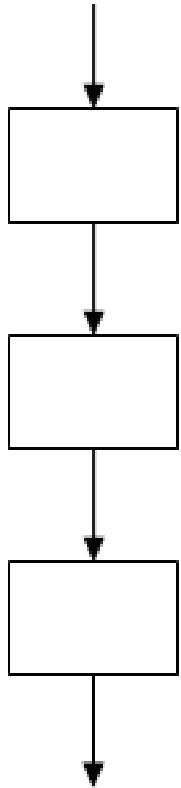
- Örnek 1:
- *Problem:* 1'den n' ye kadar olan tam sayıların toplamını bulmak.
- Problem çok kolay olmasına rağmen modüler programlamaya bir örnek olması açısından aşağıdaki şekilde bir tasarım düşünelim;



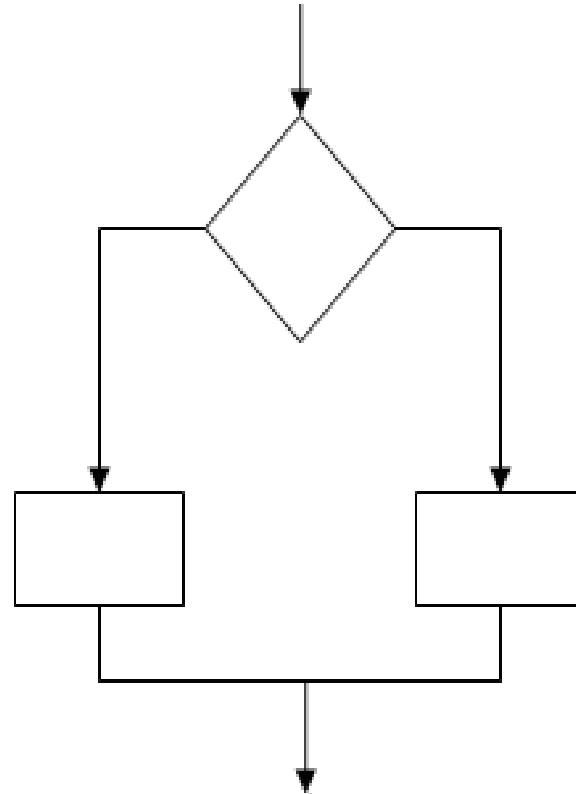


Yapısal Programlama (Devam)

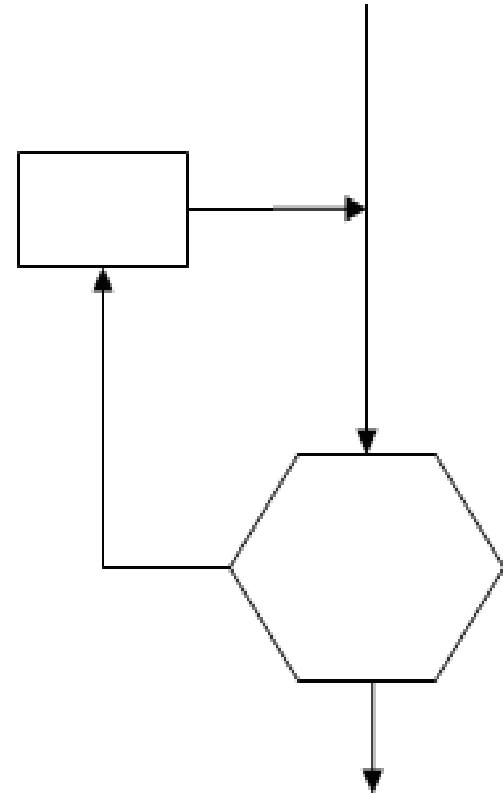
- Yapısal programlama, programlardaki akış denetimini aşağıdaki üç temel yapı ile sağlamaktadır.
 - Sıralı yapı
 - Sorğu (Seçimli) yapı
 - Tekrar (Yinelemeli) yapı



Sıra



Sorgu



Tekrar



Sıralı Yapı-Sorgu-Yinelemeli

- Bir programda yer alan iki veya daha fazla program deyimi gördükleri sırada çalıştırılır.
- Programdaki **iki veya daha fazla yol arasından biri seçilir.**
- Programdaki herhangi bir komutun **istenilen sayıda çalıştırılmasını** gerçekleştirmek mümkündür.

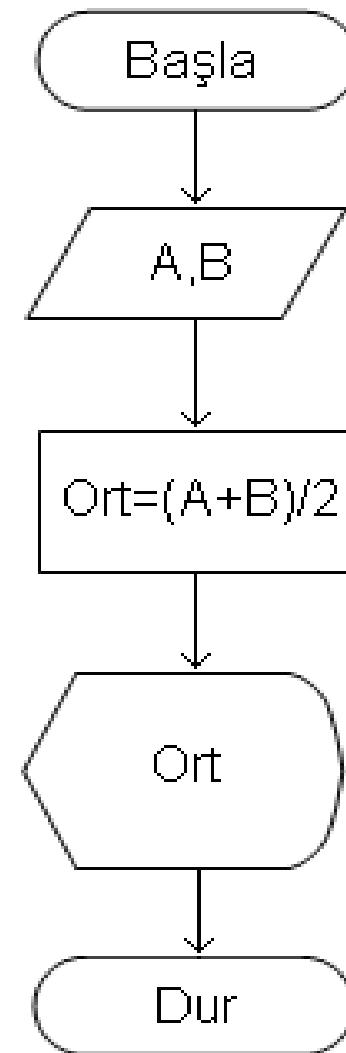


- Yapısal programlama, programlarda koşulsuz olarak akışı değiştiren *goto* gibi deyimlere yer vermez.
- Yapısal programlama, günümüzde yararlarını kanıtlamış bir programlama tekniğidir.



Sıra yapısı

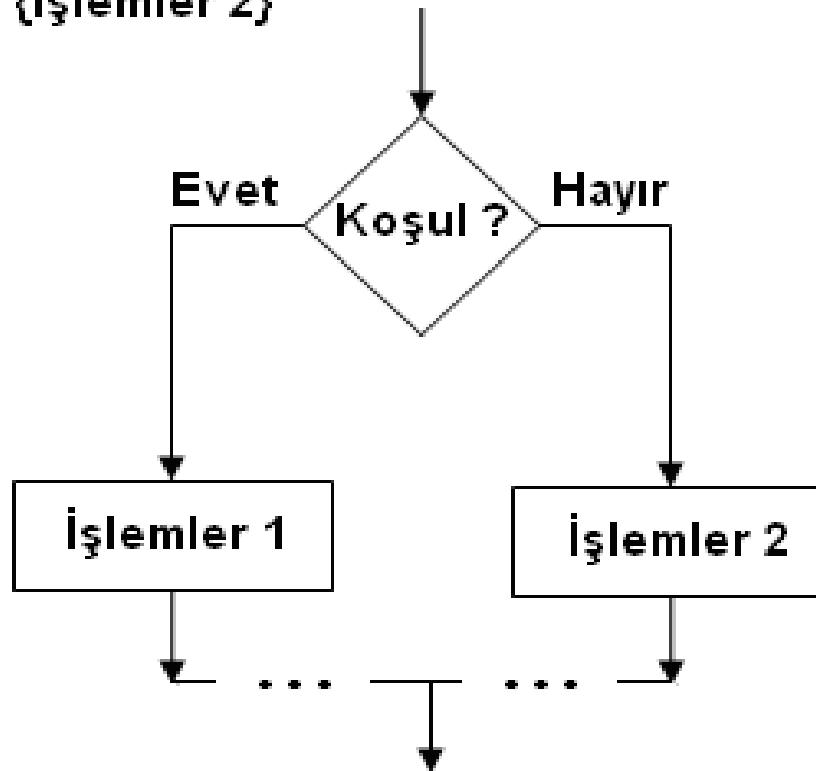
- Örneğin;
- Klavyeden girilen iki sayıyı okuyup aritmetik ortalamasını hesaplayan ve sonucu ekrana yazan bir programın akışı yandaki şekilde ifade edilebilir;



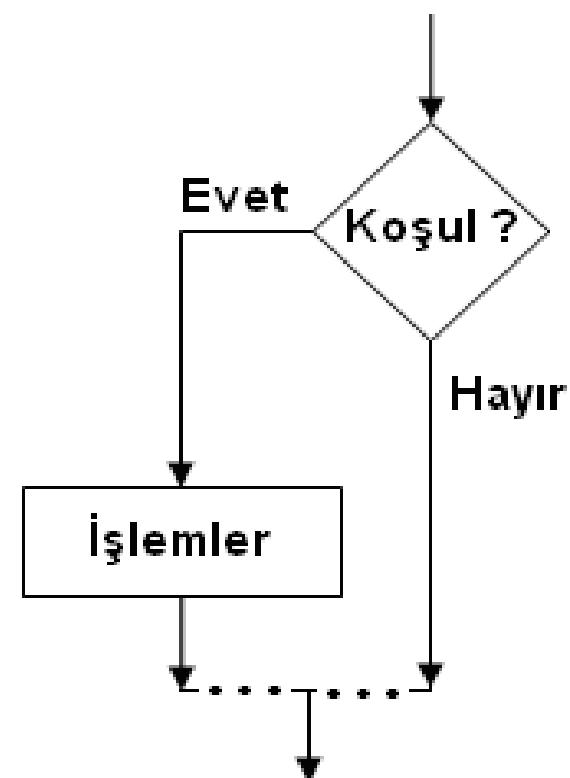


Seçim Yapıları

```
if( koşul )  
    {işlemler 1}  
else  
    {işlemler 2}
```



```
if( koşul )  
    {işlemler}
```





İçinde *if* ve sallanan-else (*dangling else*) problemi

```
if (a>b) then  
    if ( b>c)  
        then sonuc := 0  
else sonuc:=1
```



Birleşik Deyimler:

- Her *else* deyimi kendisine en yakın eşleşmemiş *then* deyimi ile eşleştirilmelidir.
- *İçinde if* deyimlerinde sallanan else probleminin çözümü için ikinci *if-then* yapısı birleşik deyim yapılmalıdır.
- Birleşik deyimler, bir dizi deyimin tek bir deyime soyutlanmasıını sağlarlar.
- Pascal'da *begin ... end* yapısı, C'de ise `{ ... }` yapısı kullanılmaktadır.



```
if (a>b){  
    if (c>d)  
        printf(...);  
}  
else  
    scanf(...);
```



If Deyimlerinin Sonlandırılması

- C ve Pascal'da *if* deyimi sözdizimi, *then* veya *else*'den sonra tek bir deyim yer almamasını, daha çok deyim bulunması durumunda ise birleşik deyim oluşturulmasını gerektirir.
- Bu sözdizimdeki eksiklik,örneğin C'de birleşik deyimlerin kapanışı için kullanılan "}" unutulsa bile, derleme sırasında sadece eksik "}" uyarısı verilmesidir.



if- then- else yapısının sonunu
belirten özel bir kelime

```
if .....then
      deyim1
else
      deyim2
end if
```



Kısa devre ve tam değerlendirme

- If ($x > 5$) AND ($y > 20$) then
- If ($x > 5$) OR ($y > 20$) then

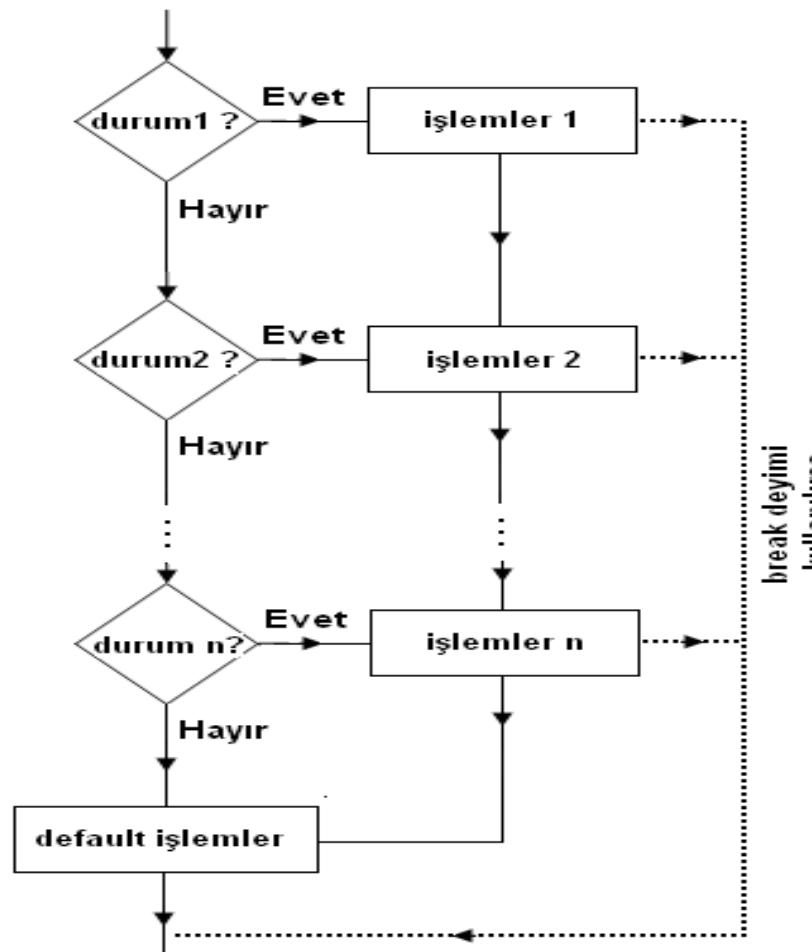


- Pascal ve C'nin çoğu gerçekleştirmi, kısa devre değerlendirmeyi uygulamaktadır.
- Ada'da ise ***and then*** ve ***or else*** olmak üzere kısa devre değerlendirme için ayrı işlemciler tanımlıdır.



Çoklu Seçim Deyimi

- Programdaki akışı belirleyen ikiden fazla yol varsa **çoklu seçim deyimi** kullanılır.



```
switch (değişken)
{
    case sabit 1:
        işlemler 1;
    case sabit 2:
        işlemler 2;
    ...
    ...
    case sabit n:
        işlemler n;
    default:
        default işlemler;
}
```



“switch - case” seçme yapısı

- Örnek: “switch” – “case” seçme yapısı için örnek bir C/C++ programı;

```
switch (a) {  
    case '1': printf("cok zayif \n"); break;  
    case '2': printf("zayif \n"); break;  
    case '3': printf("orta \n"); break;  
    case '4': printf("iyi \n"); break;  
    case '5': printf("pekiyi \n"); break;  
    default: printf("yanlis secim \n");  
}  
getch(); }
```



3.Yineleme Yapıları Döngü (loop)

- Bir programda yer alan bir komut yada komut gurubunun istenilen sayıda çalıştırılmasını sağlayan yapılara yineleme yapıları denir.
 - Döngü Çeşitleri
 - **sayaç denetimli döngüler**
 - **mantıksal denetimli döngüler**



a) Sayaç Denetimli Döngüler

- Bir sayaç denetimli döngüde, **döngü değişkeni** adı verilen bir değişken sayaç değerini gösterir.
- Bu sayaç değerinin **başlangıç değeri**, **bitiş değeri** ve ardışık iki değeri arasındaki farkı gösteren **adım büyüklüğü**, bir sayaç denetimli döngü gövdesinin kaç kez yinleneceğini belirler.
- Başlangıç değerinden başlayan döngü değişkeni, her yineleme için, adım büyüklüğü değerine göre artırılacak veya azaltılacaktır.



C

- C'de sayaç denetimli döngü tasarımının genel şekli aşağıdaki gibiidir.

```
for (ifade_1; ifade_2; ifade_3)
```

```
for (sayac=0; sayac>25;sayac++)  
    T=T+sayac;
```

- Birinci parametre döngü sayacına başlangıç değeri verilmesi,
- ikinci parametre koşulu ve
- üçüncü parametre her çevrimde sayacın nasıl artacağını/eksileceğini ifade eder.



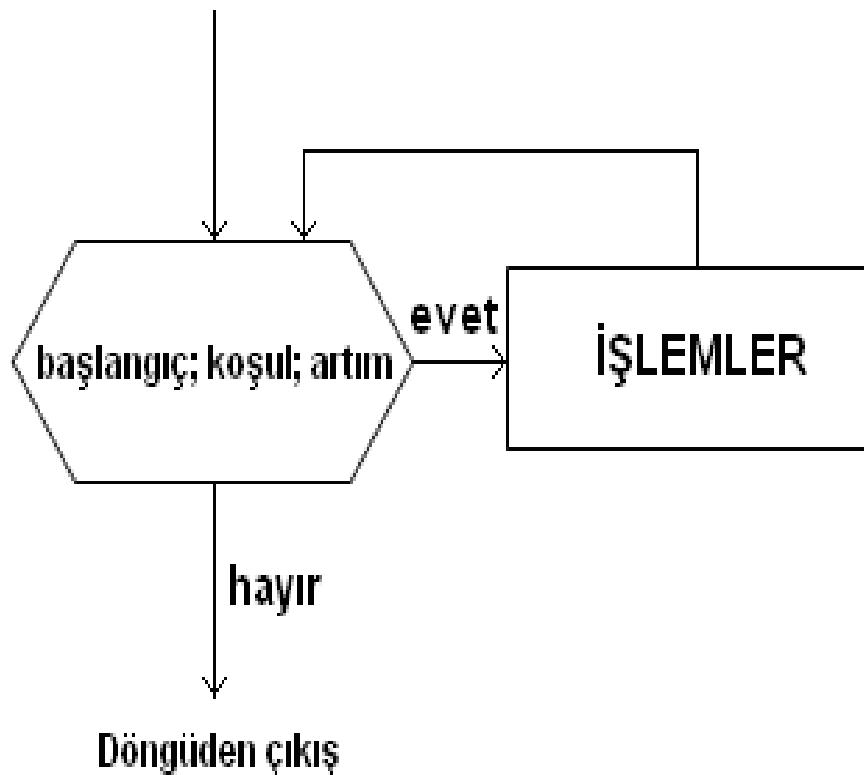
-
- *Örnek yazılım formatları:*
 - `for (k=1;k<50; k+=2)`
 - `for (k=5;k<=n; k++)`
 - `for (x=50;x>10;x--)`
 - `for (;x<10;x++)` /* başlangıç değeri daha önce atanmış olmalı */
 - `for (x=2;x<n;)` /* x döngü sayacı döngü içinde değiştirilmeli */



FOR Döngüsü

- Aşağıda verilen şekilde “for” döngü yapısı akış diyagramı olarak gösterilmekte ve genel yazılım formatı verilmektedir;

Döngüye giriş



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



b) Mantıksal denetimli döngüler

- **önce sıنانan** (*pretest*) döngü,
- **sonra sınanan** (*posttest*) döngü
- Döngü başı ve sonu arasındaki deyimlere döngü gövdesi denir.



C'de mantıksal denetimli döngüler için kullanılan yapı aşağıdaki şekilde görülmektedir.

```
while (mantıksal_ifade)
      deyim } önce sınanan
```

```
do
      deyim
      while (mantıksal_ifade) } sonra sınanan
```



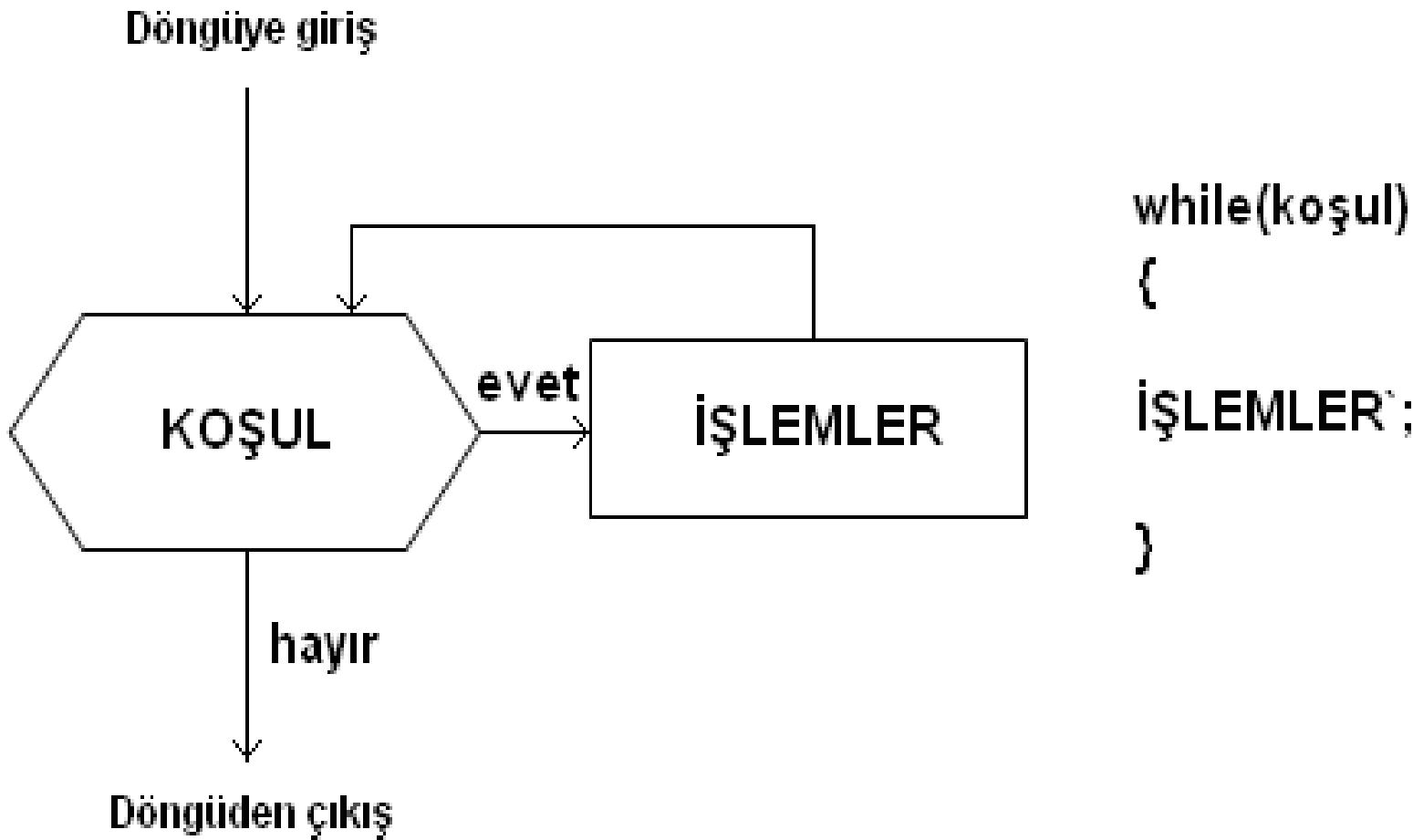
WHILE Döngüsü

- “while” döngüsü “for” döngüsü gibi aynı işlemleri birçok kez tekrarlamak için kullanılır. Bu döngüde de koşul sınaması çevrime girmeden yapılır.
- Koşul tek bir karşılaştırmadan oluşabileceği gibi birden çok koşulun mantıksal operatörler ile birleştirilmesi ile de oluşturulabilir.



WHILE Döngüsü

- Aşağıda verilen şekilde “while” döngü yapısı akış diyagramı olarak gösterilmekte ve genel yazılım formatı verilmektedir;





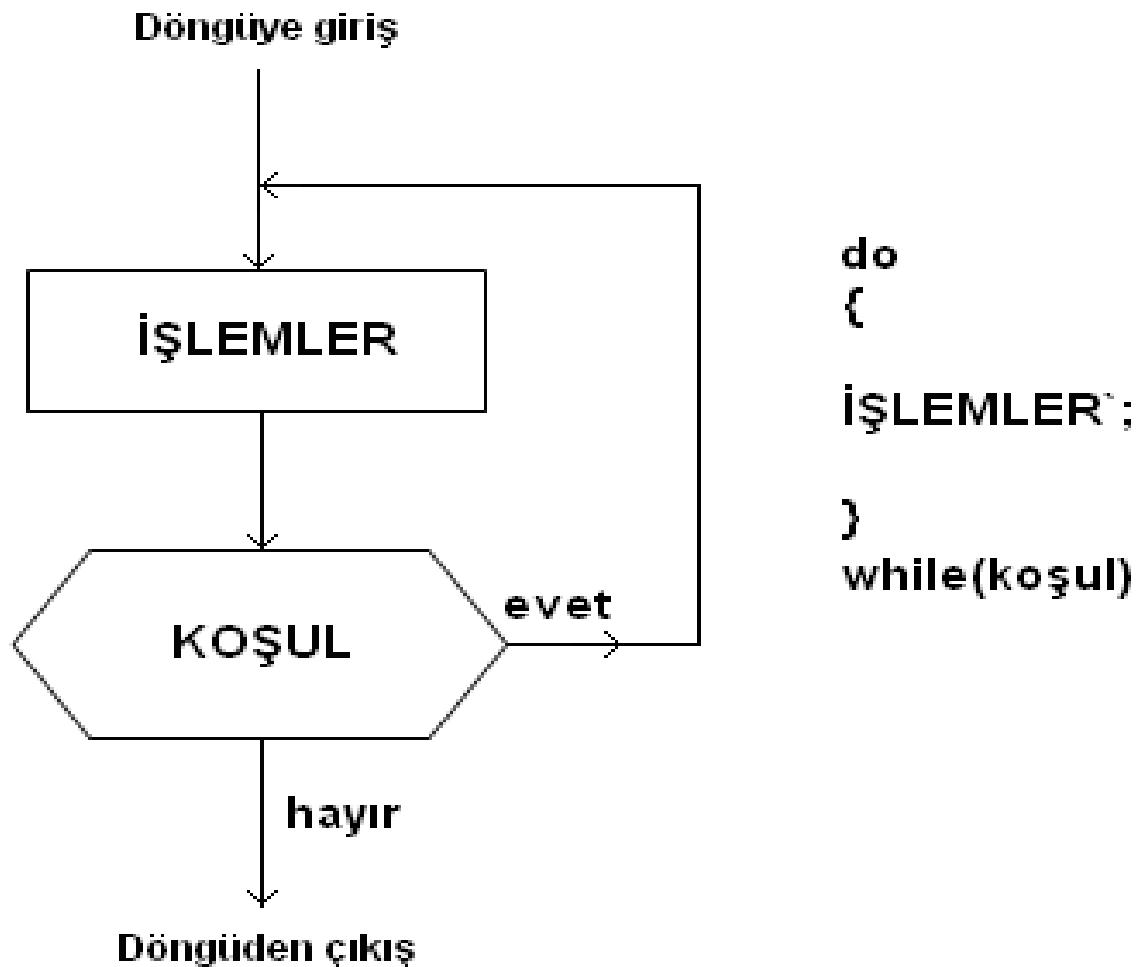
DO ... WHILE Döngüsü

- *do ... while*" döngüsü diğer döngüler gibi aynı işlemleri birçok kez tekrarlamak için kullanılır.
- Farklı olarak, bu döngüde koşul sınaması yapılmadan çevrime girilir ve işlem kümesi en az bir kere işletilir. Bu deyim yapısında da koşul sağlandığı sürece çevrim tekrarlanır.
- Koşul tek bir karşılaştırmadan oluşabileceği gibi birden çok koşulun mantıksal operatörler ile birleştirilmesi ile de oluşturulabilir.



DO ... WHILE Döngüsü

- Yanda verilen şekilde “do … while” döngü yapısı akış diyagramı olarak gösterilmekte ve genel yazılım formatı verilmektedir;





PASCAL

Repeat

.....

Until mantiksal_ifade

Ada

loop

.....

end loop



Akış Denetimini Değiştirme (Koşulsuz)

- Yapısal programlama için, bir program içinde deyimlerin akışı denetlenmelidir. Ancak akış denetiminin değiştirilmesini sağlayan deyimler, yapısal programlama ilkelerine uymayan yapılar içerebilirler.
- Programlama dillerinde yer alan *go to* deyimi ve döngülerden erken çıkış için veya döngünün bir geçişinin normalden önce tamamlanması için kullanılan deyimler, bu deyimlere örnek oluşturmaktadır.



Begin

.....
program kodu

Döngü

deyim

program kodu

.....

end.



Goto Deyimi

- **Goto** deyimi, bir programda akış denetimini koşulsuz olarak değiştirmeyi sağlayan deyimdir. Akışı yönetmek için güçlü bir deyim olmakla birlikte, akışı koşulsuz olarak değiştirme, bir programdaki deyimlerin sırasını rasgele olarak belirleyebildiği için, sorunlara yol açmaktadır.
- *goto* deyiminin programların okunabilirliğini ve güvenilirliğini azaltarak, bakım aşamasını ve programların etkin çalışmasını güçlendirdiğini göstermiştir. (Bunu ifade eden bir benzeştirmede *goto* deyiminin yer verildiği programlar, *spaghetti kodu* olarak nitelendirilmiştir.)



- Popüler diller *goto* deyimine yer vermeyi ancak kullanımını kısıtlamayı tercih etmektedir. (Pascal ve C gibi)





Exit, Break

Normalden önce çıkışı sağlar.

Continue, Cycle

Döngü içinde bir bölümün atlanmasını sağlar.



QuickBASIC'te *exit* deyimi

```
for j=1 to 5
    input miktar
    if miktar < 0 then exit for
    toplam = toplam + miktar
end
```





Ada (exit deyimi)

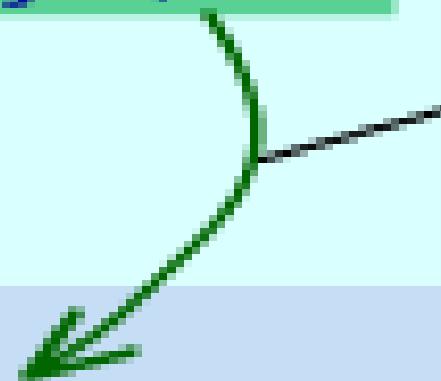
```
loop
```

```
....
```

```
exit when koşul;
```

```
....
```

```
end loop;
```





C' de *break* Deyimi:

- C'de sayıç denetimli veya mantıksal denetimli döngülerden normalden önce çıkış, *break* deyimi kullanılarak sağlanabilir. *break* deyimi çalıştırılır çalıştırılmaz, döngü dışına çıkışı sağlar.



C' de *continue* Deyimi:

- C' de döngülerde akışı değiştirmek için *break* deyimine ek olarak denetimi en içteki döngünün sınıma deyimine aktaran *continue* deyimi tanımlıdır. *continue* deyimi, döngüyü sona erdirmeden döngü gövdesinde bulunulan noktadan döngü kapanış deyimine kadar olan deyimlerin atlanması ve döngünün bir sonraki yinelemeye devam etmesini sağlar.



Özet

- Yapısal programlama, program tasarıımı ve yazılmasını kurallara bağlayan ve disiplin altına alan bir yaklaşımındır.

Yapısal programlama:

1. Programların anlaşılabilirliğini artırır.
2. Bir programın hatalarının ayıklanmasını kolaylaştırır.
3. Programın sınanmasını ve düzeltilme zamanını kısaltır.
4. Bir programın niteliği, güvenilirliği ve etkinliği artar.

Programlama Dillerinin Prensipleri

ALTPROGRAMLAR

Hedef

- Bu bölümde, programlarda sıkça kullanılan işlemlerin bir araya gruplanması ile oluşturulan altprogramlar incelenecaktır. Altprogramlar ile, bir programda birden çok kez gereksinim duyulan kod bölümleri programda bir kez yer alır ve gerekince yeniden kullanılır.
- Bölüm tamamlandığında,
- Altprogramların Genel Özellikleri,
- Parametre Aktarım Yöntemleri,
- Etkinlik Kayıtları ve
- Çeşitli Programlama Dillerinde Uygulanan Yaklaşımalar
- öğrenilmiş olacaktır.

ALTPROGRAMLAR

- Programlama dillerinde her bir modülün (alt parça) bağımsız olarak ele alınmasını sağlayan ve alt program adı verilen birimler tanımlanır.
- Altprogramlar büyük ve karmaşık problemin çözümünde yapısal programlamanın etkin kullanımı için gerekli temel taşlarındanandır. Altprogramlar bir kere yazılır ve program içerisinde bir yada daha çok yerde çağrılarak kullanılır.
- Altprogramlar ile işlev soyutlaması amaçlanır.

Neden Altprogram

- Program kodlarının gereksiz yere uzaması önlenir.
- Programın tasarlanması kolaylaştırır ve okunabilirliğini artırır.
- Büyük bir programın, daha küçük ve yazılması daha kolay fonksiyonel parçalara bölünür.
- Ekip çalışması
- Altprogramlar birden çok uygulama arasında paylaşılabilir.

Altprogramların Genel Yapısı

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

altprogram ismi
resmi parametreler
procedure ornek (a: integer, b: real)

- Bir altprogram bir fonksiyon ise, parametre profiline ek olarak fonksiyonun sonuç değerinin dönüş tipini de içerir ve bu bilgiye **altprogram protokolü** adı verilir.
- C programlama dilinde, altprogramların tanımlandığı deyimler yer alır. Bu deyimlerde altprogramların parametre tipleri belirtilir ve bu deyimlere **prototip** (*prototype*) adı verilir.
- Bir altprogram, bir **yordam** veya bir **fonksiyon** olabilir
- İki altprogram türü arasındaki fark, fonksiyonların çağrıran program birimine bir sonuç değeri döndürmelerinin şart olmasıdır.

procedure ortalama (parametreler);

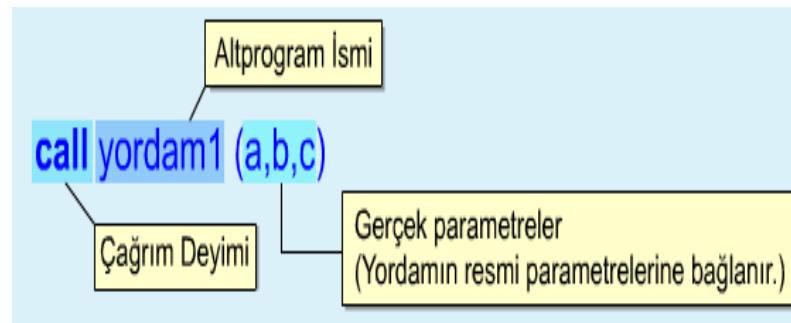
ortalama (parametreler);

Çeşitli dillerde Altprogramlar

- Yordamlar, Pascal ve Ada'da procedure anahtar kelimesi ile belirtilirler.
- C'de yordamların belirtilmesi için özel bir anahtar kelime kullanılmaz.
- Çoğu popüler *imperative* dilde hem fonksiyonlar hem de yordamlar bulunur.
- C ve C++'da sadece fonksiyonlar yer alırsa da, bu fonksiyonlar yordamların işlevlerini gerçekleştirebilirler.
- FORTRAN'da genel özellikleri diğer dillerdeki yordamlara benzer olmakla birlikte, yordamlara SUBROUTINE ismi verilir.

Parametreler

- **gerçek parametrelerin** ve **resmi parametrelerin** ilişkilendirilmesi için kullanılan iki yöntem:
- **konumsal** ve **anahtar kelime parametre** yöntemleridir.
- çağrılmış deyiminde, altprogramın ismine ek olarak, yordamın resmi parametreleriyle eşleştirilecek **gerçek (*actual*) parametreler** de yer alır.
-



- Bir yordam çağrımı gerçekleşince, etkin olan program birimindeki komutların çalışması durdurulur ve yordam etkin duruma geçer. Yordam gövdesindeki komutlar çalıştırıldıktan sonra etkinlik, yeniden çağrıının yapıldığı program birimine geçirilir ve yordam çağrı deyimini izleyen ilk deyim etkin olur

Konumsal Parametreler

- Eğer bir yordam çağrılarında gerçek parametreler ile resmi parametreler arasındaki bağlama, parametrelerin çağrım deyimindeki ve yordam başlığındaki konumuna göre yapılıyorsa, bu parametrelere **konumsal (positional) parametre** adı verilir. Birçok programlama dilinde uygulanan bu yöntem, parametre sayısı az olduğu zaman kullanılabilir.

- Aşağıda görüldüğü gibi bir çağrım deyimi ile etkin duruma geçen *ortalama* yordamında, *c* resmi parametresi, *a* gerçek parametresi ile; *d* resmi parametresi, *b* gerçek parametresi ile bağlılıyorsa, konumsal parametreler yaklaşımı uygulanmıştır.

```
call ortalama(a,b);
```

```
Procedure ortalama(c: integer; d:integer);
```

```
begin
```

```
...
```

```
end;
```

Resmi Parametre	Gerçek Parametre
<i>c</i>	\rightarrow <i>a</i>
<i>d</i>	\rightarrow <i>b</i>

Anahtar Kelime Parametre Yöntemi

- Gerçek ve resmi parametreler arasındaki bağlamayı konuma göre belirlemek yerine, her iki parametrenin de ismini belirterek gösterme, **anahtar kelime parametre** yöntemi olarak adlandırılır. Bu durumda çağrılmış deyiminde, hem resmi, hem de gerçek parametrelerin isimleri belirtilir.

Call (toplam→alttoplam, liste→dizi, gelir→kazanc);

Anahtar kelime parametre yöntemi, parametre sayısının çok olduğu durumda yararlı bir yöntemdir.

Ada ve FORTRAN 90'da hem anahtar kelime hem de konumsal parametreler yöntemi kullanılabilmektedir.

Fonksiyonlar

C

C'de geri dönüş değerinin tipi, fonksiyon adından önce yazılmalıdır.

Örnek

```
int topla(int say1, int say2);
```

Fonksiyon Adı

Geri Dönüş Tipi

Pascal

Pascal'da geri dönüş değerinin tipi tanımın en sonunda yazılmalıdır.

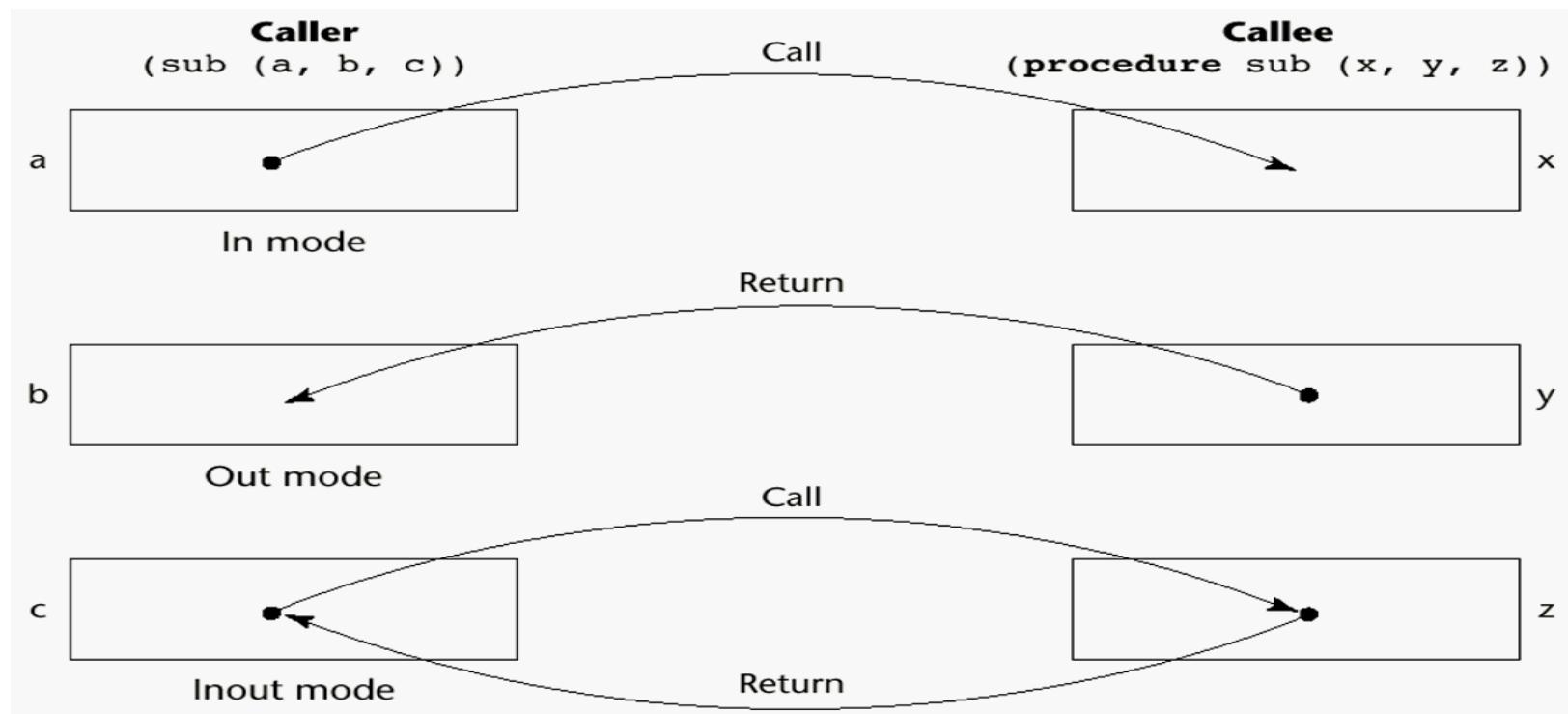
Örnek

```
function kare( say3: integer): integer ;
```

Geri Dönüş Tipi

Parametre Aktarım Yöntemleri

- Bir yordam etkin duruma getirilirken, çağrım deyimindeki gerçek parametreler ile yordamın resmi parametrelerine farklı değerler aktarılabilir.
Gerçek parametreler yordamlara aktarılacak değerleri gösterdikleri için, değişken, sabit veya ifade olabilir. Resmi parametreler ise bu değerleri tutacak bellek yerlerini gösterdikleri için değişken olmak durumundadır.



Parametre Aktarım Yöntemleri

ÇAĞIRMA YÖNTEMLERİ

Değer ile Çağırma

Sonuç ile Çağırma

Değer ve Sonuç ile Çağırma

Başvuru ile Çağırma

İsim ile Çağırma

} Gerçek değer fiziksel olarak kopyalanarak aktarılır.

} Verinin erişim yolu aktarılır.

Parametre Aktarım Yöntemleri (Devam)

- Değer ile çağrıma, sonuç ile çağrıma, değer ve sonuç ile çağrıma yöntemlerinde, gerçek ve formal parametreler arasındaki veri fiziksel olarak kopyalanarak aktarılmakta, başvuru ile çağrıma yönteminde ise veri yerine verinin adresi aktarılmaktadır.

Değer ile Çağırma (*Call by Value*)

- Bu yöntemde formal parametre, gerçek parametrenin değeriyle ilklendikten sonra, altprograma yerel bir değişken olarak değerlendirilir.

```
void f(int n)
{
    n++;
}
int main() {
    int x = 2;
    f(x);
    cout << x; }
```

Sadece gerçek parametreden formal parametreye değer geçışı olduğu için en güvenilir parametre aktarım yöntemidir

Programın çıktısı= 2 olacaktır.

Sonuç ile Çağırma (*Call by Result*)

- Bu yöntemde çağrılmış deyimi ile altprograma bir değer aktarılmazken, gerçek bir parametreye karşı gelen formal parametrenin değeri, altprogram sonunda, denetim yeniden çağrıran programa geçmeden önce, gerçek parametreyi gösteren değişkene aktarılır. (gerçek parametrenin değişken olmalı). Gerçek parametreye karşı gelen resmi parametre, altprogramın çalışması süresince yerel değişkendir.

```
procedure sub1(y: out Integer; z: out Integer) is
    . . .;
begin
    sub1(x, x);
```

Değer ve Sonuç ile Çağırma (*Call by Value Result*)

- Değer ile çağrıma ve sonuç ile çağrıma yöntemlerinin birleşimidir.
- Gerçek parametrenin değeri ile karşı gelen formal parametrenin değeri ilklenir ve sonra resmi parametre, altprogramın çalışması süresince yerel değişken gibi davranışır ve altprogram sona erdiğinde formal parametrenin değeri gerçek parametreye aktarılır.

```
int x=0;  
int main()  
{  
    f(x);  
    .....  
}  
void f(int a) {  
    x=3;  
    a++;}
```

x'in son değeri değer-sonuç aktarımına göre 1 olacaktır.

Parametreler için birden çok bellek yeri gerekmesi ve değer kopyalama işlemlerinin zaman almaktadır.

Başvuru ile Çağırma (*Call by Reference*)

- Başvuru ile çağrıma yöntemi de gerçek ve formal parametreler arasında iki yönlü veri aktarımı vardır.
- Altprograma verinin adresi aktarılır. Bu adres aracılığıyla altprogram, çağrıran program ile aynı bellek yerine erişebilir ve gerçek parametre, çağrıran program ve altprogram arasında ortak olarak kullanılır.

```
int x=0;  
int main()  
{  
    f(x);  
    .....  
}  
void f(int a) {  
    x=3;  
    a++;}
```

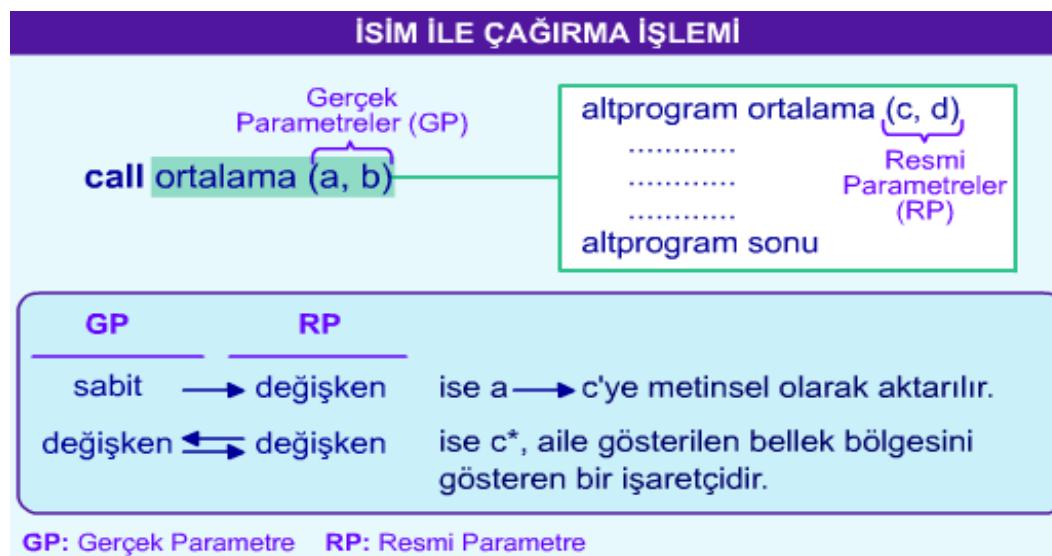
x'in son değeri değer-sonuç aktarımına göre 4 olacaktır.

Hem yer hem de zaman açısından etkindir. Fiziksel kopyalama yoktur.

Formal parametrelere erişim için bir **dolaylı erişim** gerekiyor. Sadece çağrıran programdan altprograma değer aktarılması isteniyorsa, bu yöntemde gerçek parametrenin bellekteki yerine altprogram tarafından ulaşılabilen için, değerinde **istenmeyen değişiklikler** oluşabilir.

İsim ile Çağırma (Call by Name)

- Altprogramda gerçek parametreye karşı gelen formal parametrenin bulunduğu her yere metinsel olarak gerçek parametre yerleştirilir.
- Eğer gerçek parametre bir sabit değerse, isim ile çağrıma yöntemi, değer ile çağrıma yöntemi ile aynı şekilde gerçekleşir.
- Eğer gerçek parametre bir değişkense, isim ile çağrıma yöntemi başvuru ile çağrıma yöntemi ile aynı şekilde gerçekleşir.
- İsim ile çağrıma yönteminin gerçekleştirilmesi güçtür ve kullanıldığı programların hem yazılmasını hem de okunmasını karmaşıklaştırabilir. Bu nedenle ALGOL 60 ile tanıtılan isim ile çağrıma yöntemi, günümüzde popüler olan programlama dillerinde uygulanmamaktadır.



İsim	Referans	Değer-Sonuç	Sonuç	Değer	Programlama Dili
					FORTTRAN IV
		X			Fortran 77
				seçimlik	ALGOL 60
		X			ALGOL W
	X				C++
				X	PASCAL, Modula2
	X	X	X	X	ADA
				X	Java

Özet

- Altprogramlar ve altprogramların genel özellikleri
- Parametreler,
- Parametre Aktarım Yöntemleri,
- Programlama Dillerinde Parametre Aktarımı,

Programlama Dillerinin Prensipleri

Altprogramların İşletimi

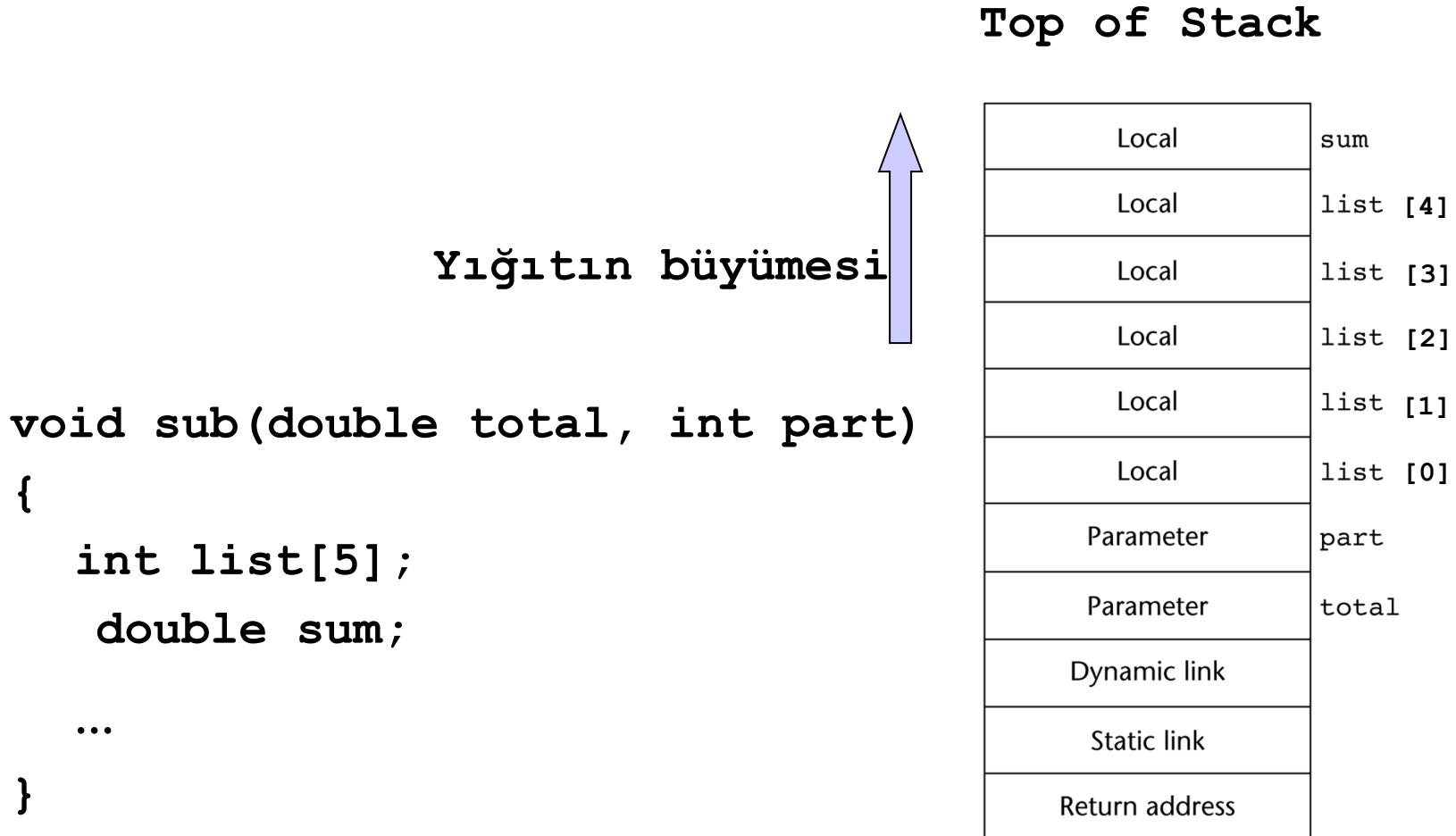
Etkinlik Kayıtları

- Bir altprogramdaki değerler ve değişkenler için bellek ataması denetim altprograma aktarıldığında yapılır.

- ALGOL60 ve izleyen dillerde, altprogramlar arasında parametre iletişimini çalışma zamanındaki belleğin yığıt bölümü aracılığıyla gerçekleştir.
- Yığıt bellek, yerel değişkenler ve formal parametreler, geri dönüş adresi gibi bilgileri saklamak için kullanılır.
- (*Last In First Out, LIFO*)

- 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.

C++ Fonksiyon Örneği



Aktivasyon Kaydı

```

void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}
void fun2(double r) {
    int s,t;
    ...
    fun1(s);
    ...
}
void fun3(int q) {
    ...
}
void main()
{
    double p;
    ...
    fun2(p);
    ...
}

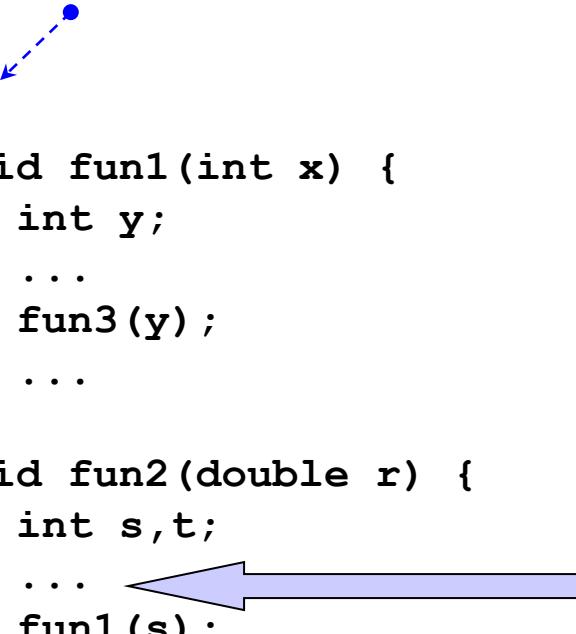
```



```

void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}
void fun2(double r) {
    int s,t;
    ...
    fun1(s);
    ...
}
void fun3(int q) {
    ...
}
void main()
{
    double p;
    ...
    fun2(p);
    ...
}

```



	YİĞİT TEPESİ	
	Yerel değişkenler	t
	Yerel değişkenler	s
Fun2 için aktivasyon kaydı	Formal parametre	r
	Dinmik bağ (link)	
	Main'e dönüş	
Main için aktivasyon kayıdı	Yerel değişkenler	p



```

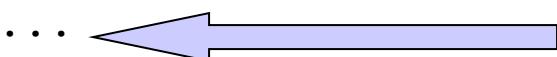
void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}

void fun2(double r) {
    int s,t;
    ...
    fun1(s);
    ...
}

void fun3(int q) {
    ...
}

void main()
{
    double p;
    ...
    fun2(p);
    ...
}

```



	YİĞITİN TEPESİ	
	Yerel değişken	y
Fun1 için aktivasyon kaydı	Formal parametre	x
	Dinamik link	
	Fun2'ye dönüş	
	Yerel değişkenler	t
	Yerel değişkenler	s
Fun2 için aktivasyon kaydı	Formal parametre	r
	Dinmik bağ (link)	
	Main'e dönüş	
Main için aktivasyonkayıdı	Yerel değişkenler	p



```

void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}
void fun2(double r) {
    int s,t;
    ...
    fun1(s);
    ...
}
void fun3(int q) {
    ...
}
void main()
{
    double p;
    ...
    fun2(p);
    ...
}

```

	YİĞİTİN TEPESİ	
Fun3 için aktivasyon kaydı	Formal parametre	q
	Dinamik link	
	Fun1'e dönüş	
Fun1 için aktivasyon kaydı	Yerel değişken	s
	Formal parametre	r
	Dinamik link	
	Fun2'ye dönüş	
Fun2 için aktivasyon kaydı	Yerel değişkenler	t
	Yerel değişkenler	s
	Formal parametre	r
	Dinamik bağ (link)	
	Main'e dönüş	
Main için aktivasyon kayıdı	Yerel değişkenler	p

The diagram illustrates the activation records for each function:

- fun3 activation record:** Contains "Formal parametre" (q). A blue arrow points from here to the "Dinamik link" entry in the fun1 activation record.
- fun1 activation record:** Contains "Yerel değişken" (s), "Formal parametre" (r), and "Dinamik link". A blue arrow points from here to the "Fun2'ye dönüş" entry in the fun2 activation record.
- fun2 activation record:** Contains "Yerel değişkenler" (t), "Yerel değişkenler" (s), "Formal parametre" (r), and "Dinamik bağ (link)". A blue arrow points from here to the "Main'e dönüş" entry in the main activation record.
- main activation record:** Contains "Yerel değişkenler" (p).

```

void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}
void fun2(double r) {
    int s,t;
    ...
    fun1(s);
    ...
}
void fun3(int q) {
    ...
}
void main()
{
    double p;
    ...
    fun2(p);
    ...
}

```

	YİĞITİN TEPESİ	
	Yerel değişken	y
Fun1 için aktivasyon kaydı	Formal parametre	x
	Dinamik link	
	Fun2'ye dönüş	
	Yerel değişkenler	t
	Yerel değişkenler	s
Fun2 için aktivasyon kaydı	Formal parametre	r
	Dinmik bağ (link)	
	Main'e dönüş	
Main için aktivasyon kayıdı	Yerel değişkenler	p

```

void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}
void fun2(double r) {
    int s,t;
    ...
    fun1(s);
    ...
}
void fun3(int q) {
    ...
}
void main()
{
    double p;
    ...
    fun2(p);
    ...
}

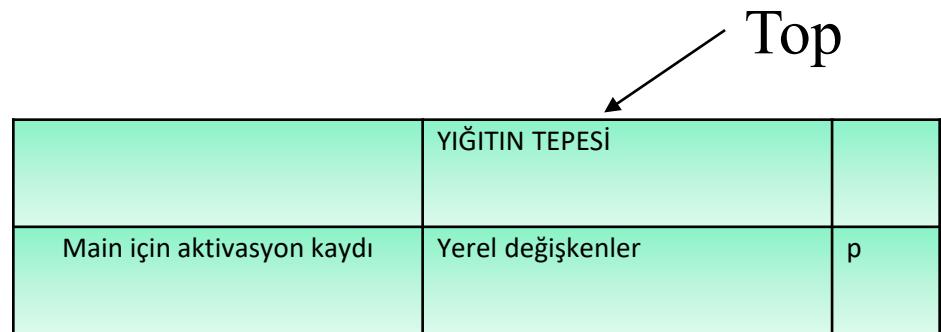
```

	YİĞITİN TEPESİ	
	Yerel değişkenler	t
	Yerel değişkenler	s
Fun2 için aktivasyon kaydı	Formal parametre	r
	Dinmik bağ (link)	
	Main'e dönüş	
Main için aktivasyon kayıdı	Yerel değişkenler	p

```

void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}
void fun2(double r) {
    int s,t;
    ...
    fun1(s);
    ...
}
void fun3(int q) {
    ...
}
void main()
{
    double p;
    ...
    fun2(p);
    ...
}

```



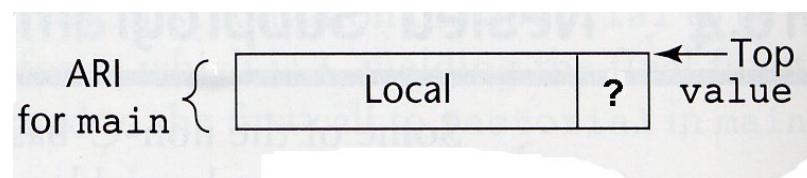
	YİĞİTİN TEPESİ	
Main için aktivasyon kaydı	Yerel değişkenler	p

```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```

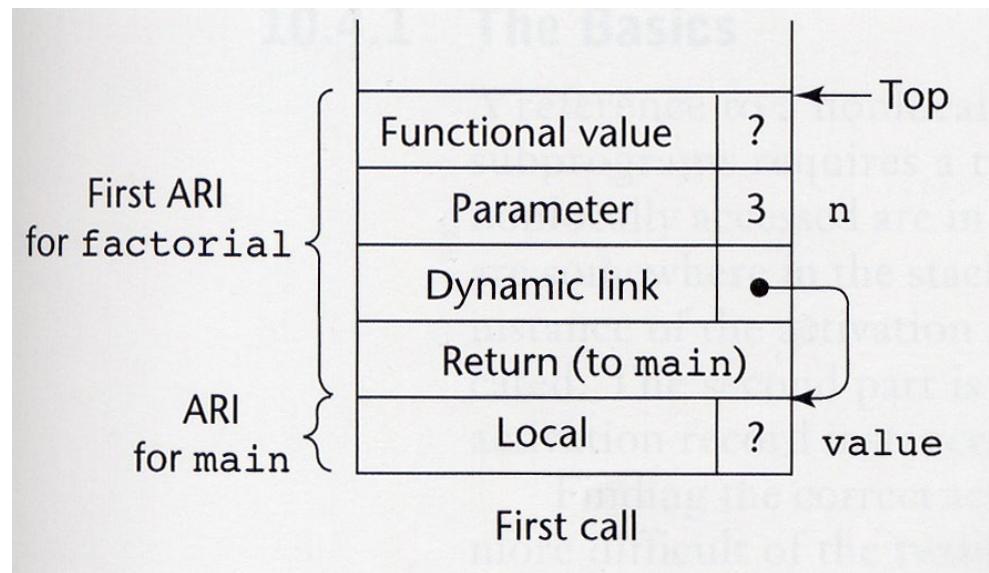


```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```

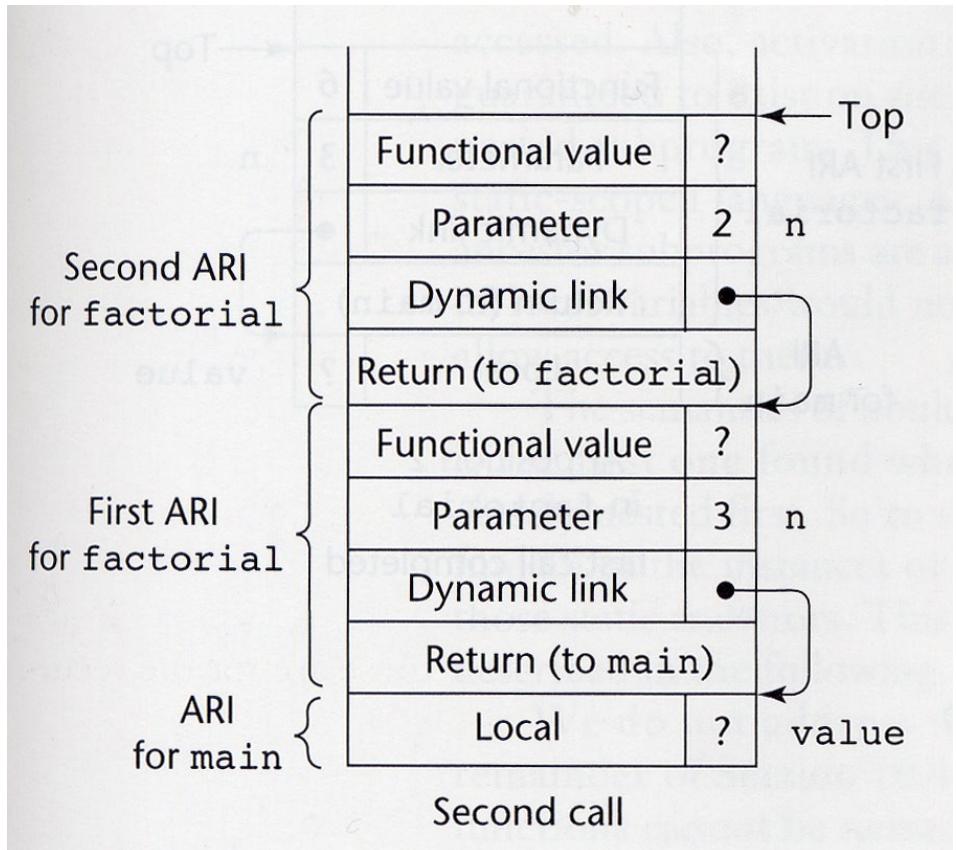


```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```

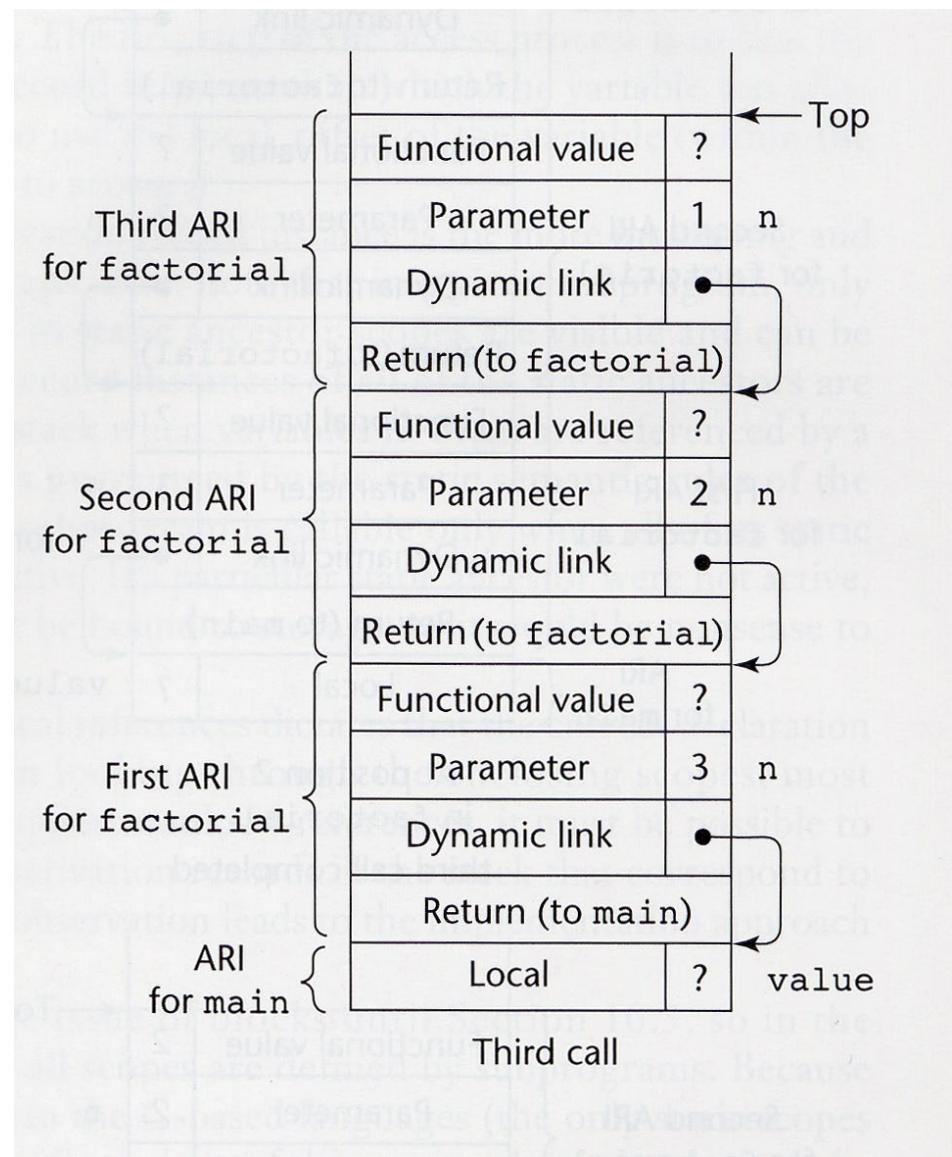


```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```

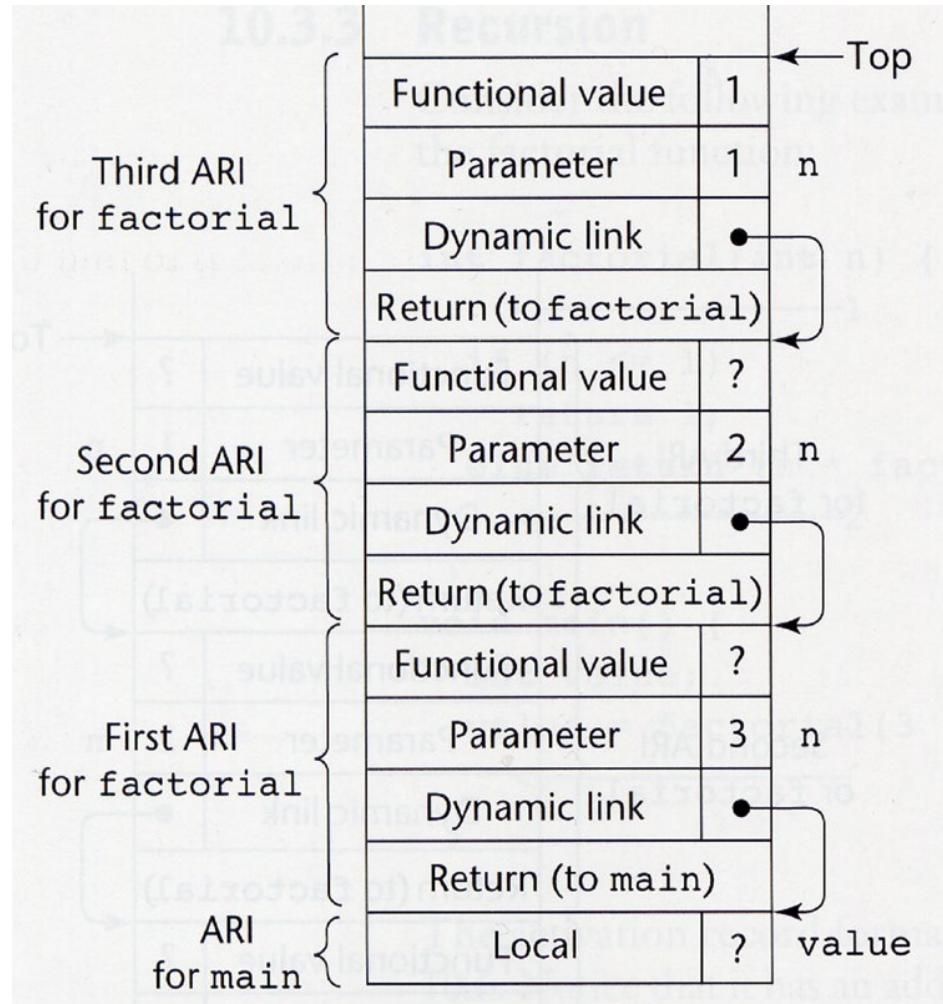


```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1; ←
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```

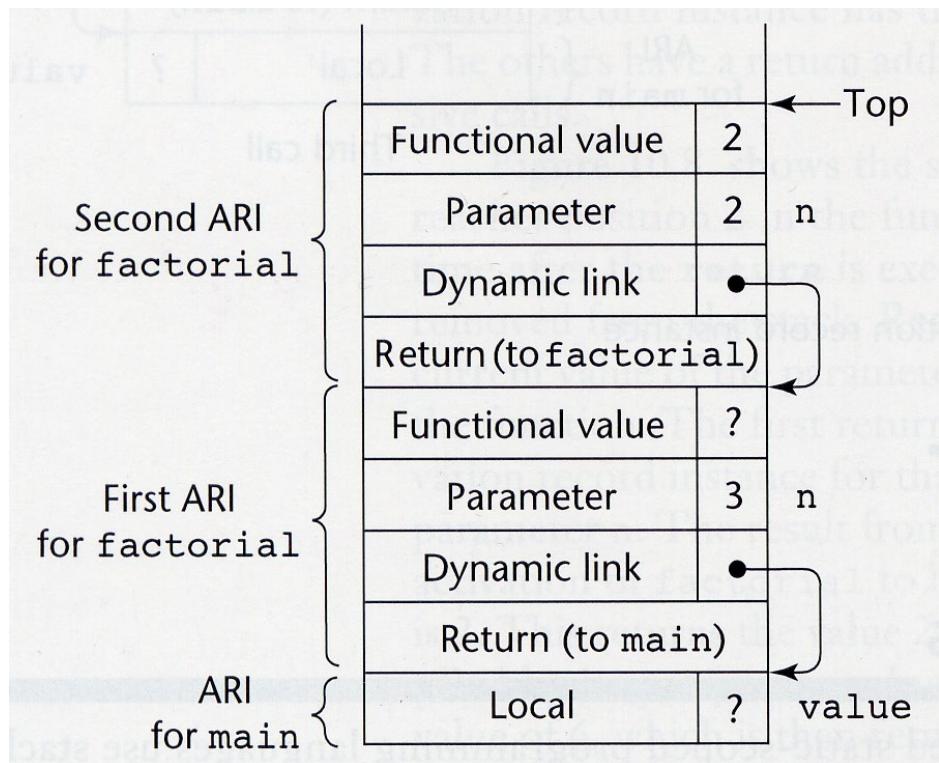


```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```

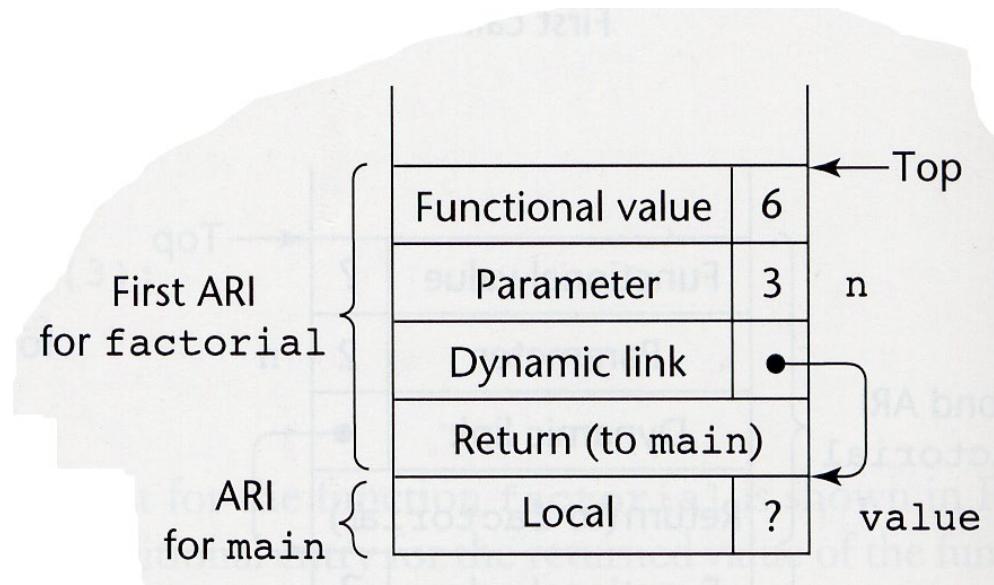


```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```

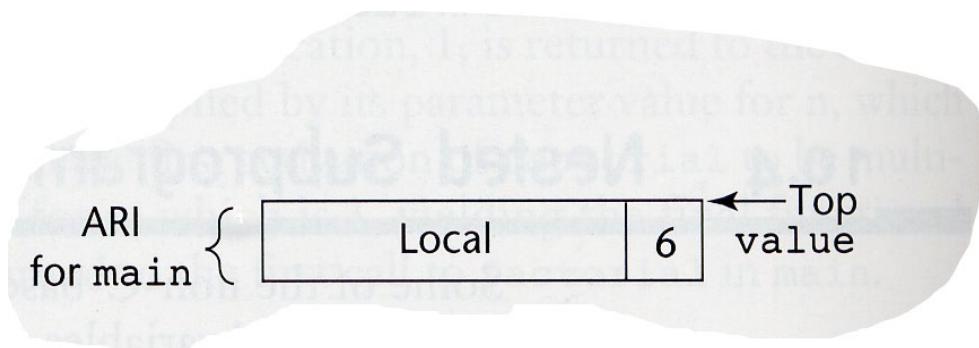


```

void factorial(int x) {
    ...
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return (n*factorial(n-1));
    }
}

void main()
{
    int value;
    ...
    value = factorial(3);
    ...
}

```



```

program MAIN_2;
var X : integer;
procedure BIGSUB;
var A, B, C : integer;

procedure SUB1;
var A, D : integer;
begin { SUB1 }
A := B + C; <-----1
end; { SUB1 }
procedure SUB2(X : integer);
var B, E : integer;

procedure SUB3;
var C, E : integer;
begin { SUB3 }
SUB1;
E := B + A; <-----2
end; { SUB3 }

begin { SUB2 }
SUB3;
A := D + E; <-----3
end; { SUB2 }

begin { BIGSUB }
SUB2(7);
end; { BIGSUB }

begin
BIGSUB;
end. { MAIN_2 }

```

MAIN_2 calls BIGSUB
BIGSUB calls SUB2
SUB2 calls SUB3
SUB3 calls SUB1

	YigitinTepesi	
	yerel	D
	Yerel	A
Aktivasyon kaydi (SUB1)	Dinamik link	
	Static link	
	Return (sub3)	
	Yerel	E
	Yerel	C
Aktivasyon kaydi (SUB3)	Dinamik link	
	Static link	
	Return (sub2)	
	Yerel	E
	Yerel	B
Aktivasyon kaydi (SUB2)	Parametre	7, X
	Dinamik link	
	Static link	
	Return (BIGSUB)	
	Yerel	C
	Yerel	B
	Yerel	A
Aktivasyon kaydi (BIGSUB)	Dinamik link	
	Static link	
	Return (main_2)	
Aktivasyon kaydi (main_2)	Yerel Değişken	x

Dynamic Kapsam

```

void sub3()
{
    int x, z;
    x = u + v;
    ...
}

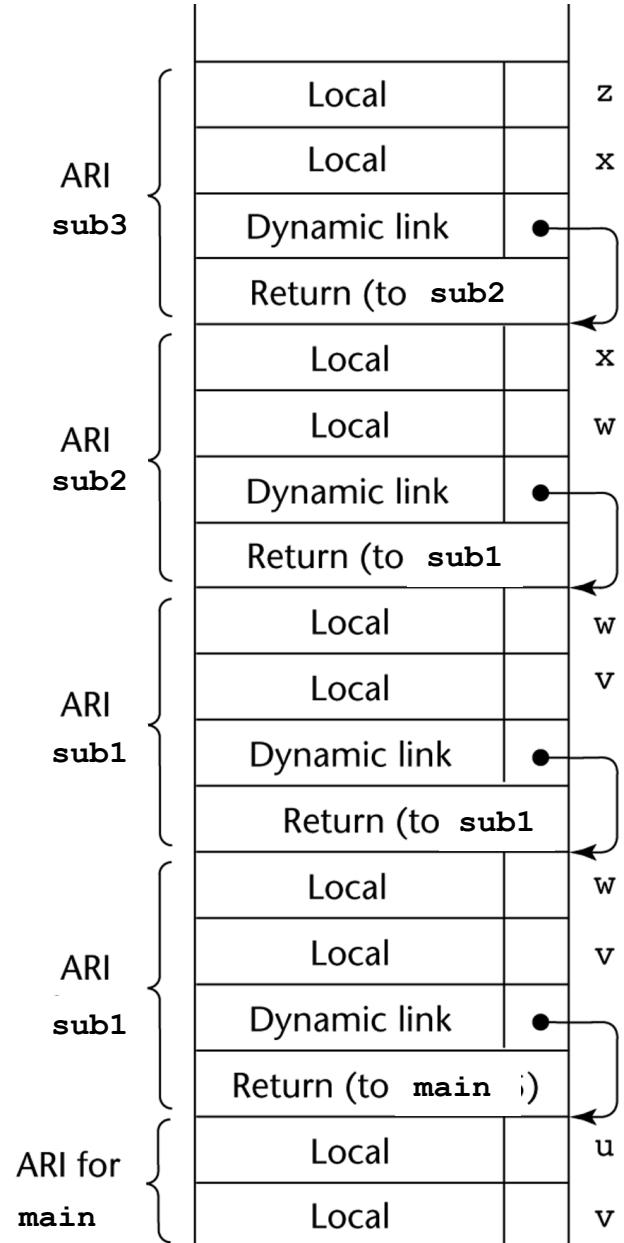
void sub2()
{
    int w, x;
    ...
}

void sub1()
{
    int v, w;
    ...
}

void main()
{
    int v, u;
    ...
}

```

main sub1'i
 sub1 sub1'i
 sub1 sub2'yi
 sub2 sub3'ü
 çağırırsın



ARI = activation record instance

Eşzamanlılık

(Concurrency)

- Programlama dillerindeki eş zamanlılık kavramı ile bilgisayar donanımındaki paralel çalışma birbirinden bağımsız kavamlardır.
- Eğer çalışma zamanında üst üste gelme durumu varsa donanım işlemlerinde paralellik oluşur.
- Bir programdaki işlemler eğer paralel olarak işlenebiliyorsa program eş zamanlıdır denilir.
- Eş zamanlılık kavramının karşıtı ise bilirli bir sıraya göre dizilmiş ardışıl işlemlerdir.

Öncelik grafları:

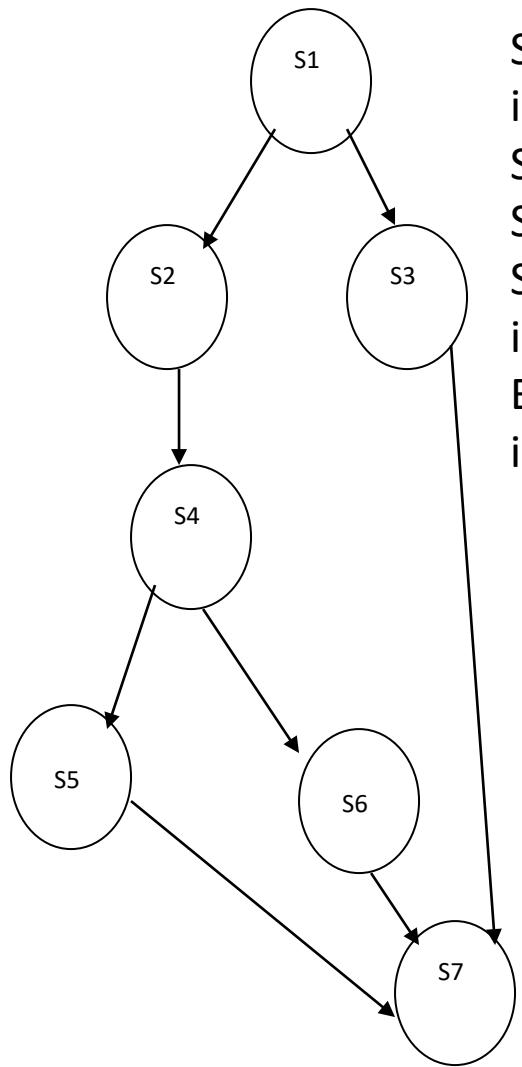
- **a:=x + y;**
- **b:= z + 1;**
- **c:= a - b;**
- **w:= c + 1;**

Burada $c := a - b$ yi hesaplamak için öncelikle a ve b 'ye değer atanması gerekmektedir.

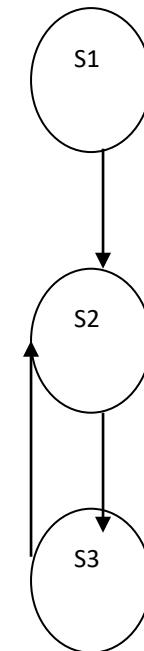
Benzer biçimde $w := c + 1$ ifadesinin sonucu da c 'nin hesaplanmasına bağlıdır.

Diğer taraftan $a := x + y$ ve $b := z + 1$ deyimleri birbirine bağlı değildir. Bu yüzden bu iki deyim birlikte çalıştırılabilir.

Buradan anlaşılıyor ki bir program parçasında değişik deyimler arasında bir öncelik sıralaması yapılabilir. Bu sıralamanın grafik olarak gösterimine öncelik grafi denir. Bir öncelik grafi, her bir düğümü ayrı bir deyimi ifade eden, döngüsel olmayan yönlendirilmiş bir graftır.



S2 ve S3 deyimleri, S1 tamamlandıktan sonra işletilebilir.
 S4, S2 tamamlandıktan sonra işletilebilir.
 S5 ve S6, S4 tamamlandıktan sonra işletilebilir.
 S7, sadece S5,S6 ve S3 tamamlandıktan sonra işletilebilir.
 Bu örnekte S3 deyimi S2, S4, S5 ve S6 deyimleri ile eş zamanlı olarak çalışabilir.



Bu grafta görüldüğü gibi, S3 sadece S2 tamamlandıktan sonra işletilebilir. S2 deyimi ise sadece S3 tamamlandıktan sonra işletilebilir. Burada açıkça görülmektedir ki bu iki kısıtlamanın her ikisi aynı anda giderilemez. Yani bir programın akışını ifade eden öncelik grafında döngü içermemelidir.

Eşzamanlılık Şartları:

- $1.R(S1) \cap W(S2) = \{\}$
- $2.W(S1) \cap R(S2) = \{\}$
- $3.W(S1) \cap W(S2) = \{\}$

S1 ve S2 deyimleri eş zamanlı olarak çalışabilir mi?

S1: $a := x + y$

S2: $b := z + 1$

S3: $c := a - b$

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

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

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

$R(S1) = \{x, y\}$

$R(S2) = \{z\}$

$R(S3) = \{a, b\}$

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

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

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

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?

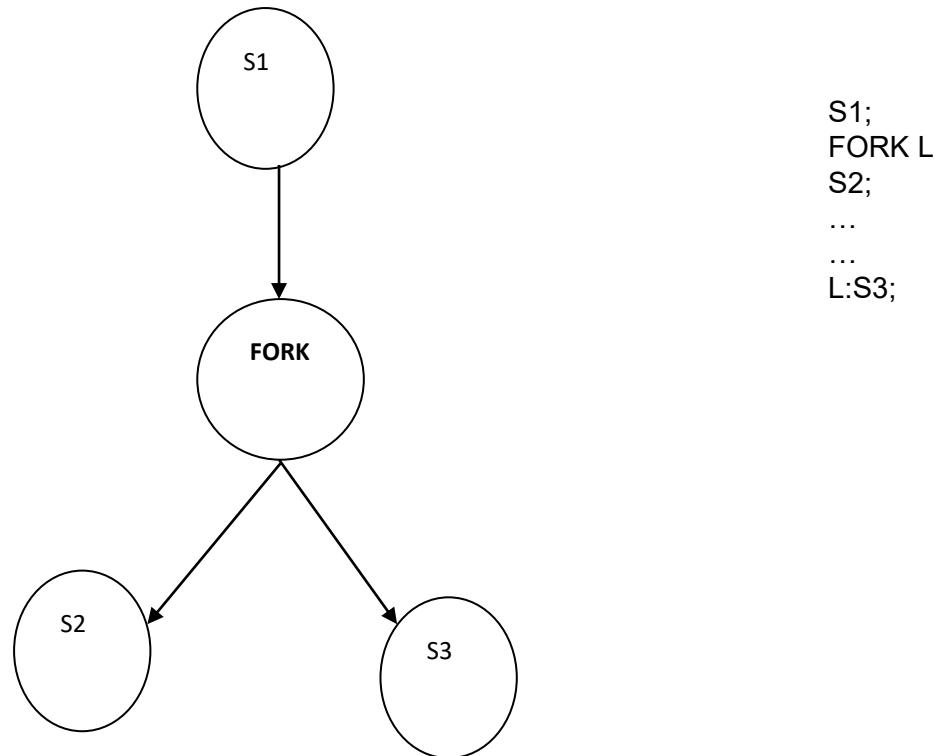
Koşul 1. $R(S2) \cap W(S3) = \{z\} \cap \{c\} = \{\}$

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:

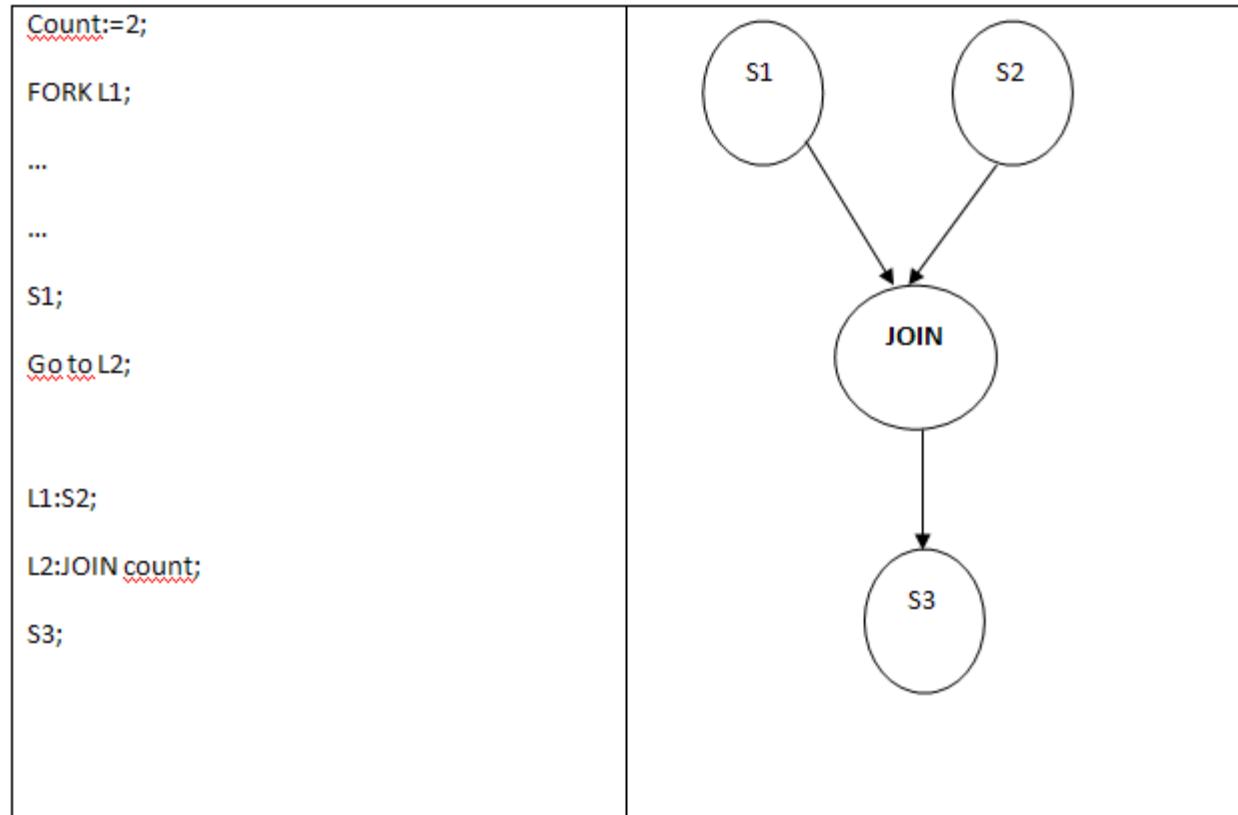
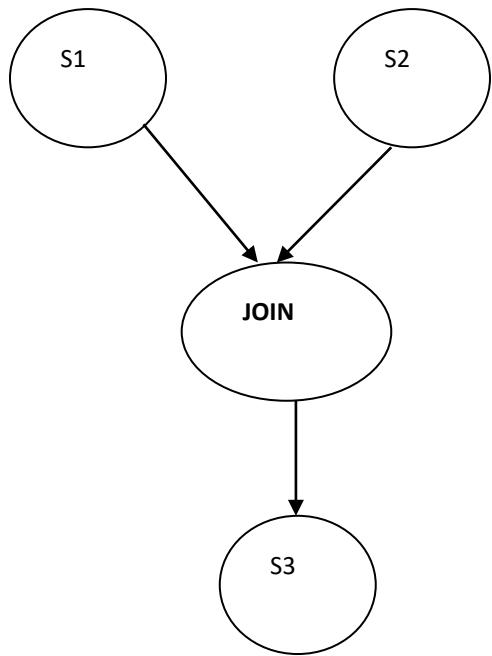
- FORK ve JOIN yapıları eş zamanlılığı tanımlayan ilk programlama dili notasyonlarından biridir. Aşağıdaki öncelik grafi bu komutlardan FORK yapısını ifade etmektedir.



Burada eş zamanlı işlemlerden birisi L etiketi ile gösterilen deyimlerden başlarken diğer FORK komutunu izleyen deyimlerin işlenmesi ile devam eder.

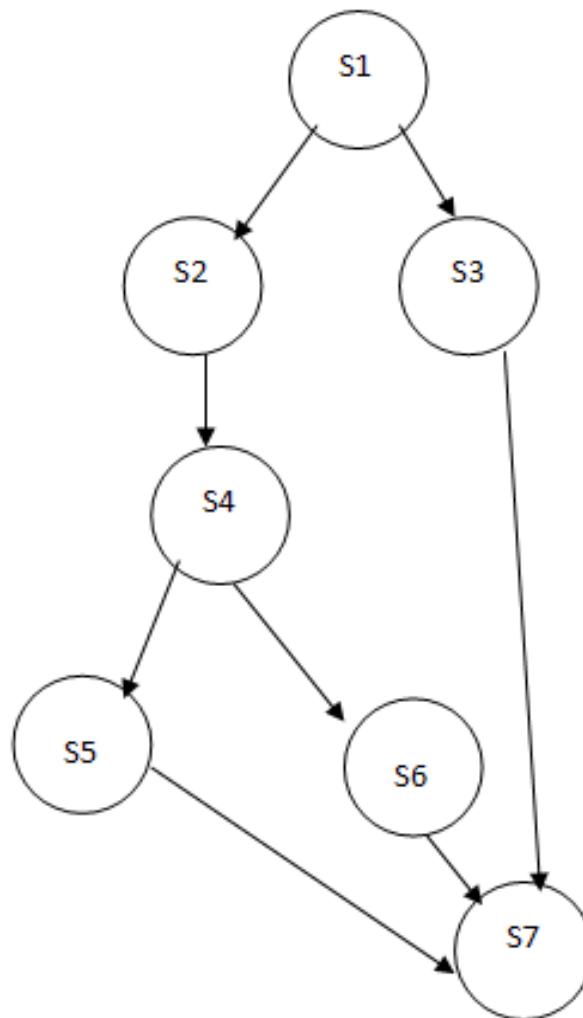
FORK L deyimi işletildiği zaman S3'de yeni bir hesaplama başlar. Bu yeni hesaplama S2'de devam eden eski hesaplama ile eş zamanlı olarak işletilir.

JOIN komutu iki eş zamanlı hesaplamayı tekrar birleştirir. JOIN komutunun öncelik grafi karşılığı aşağıda verilmiştir.



Örnek:

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



Parbegin-Parend eş zamanlılık deyimleri:

Parbegin

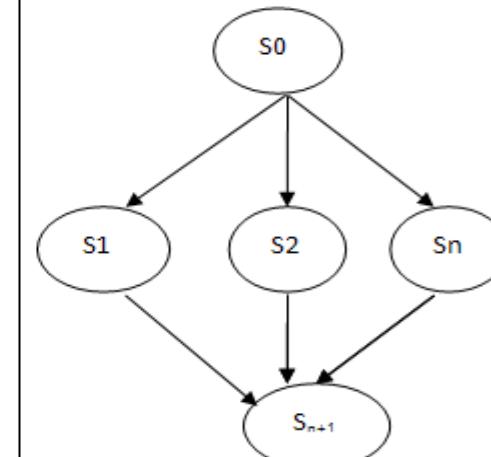
S1;

S2;

...

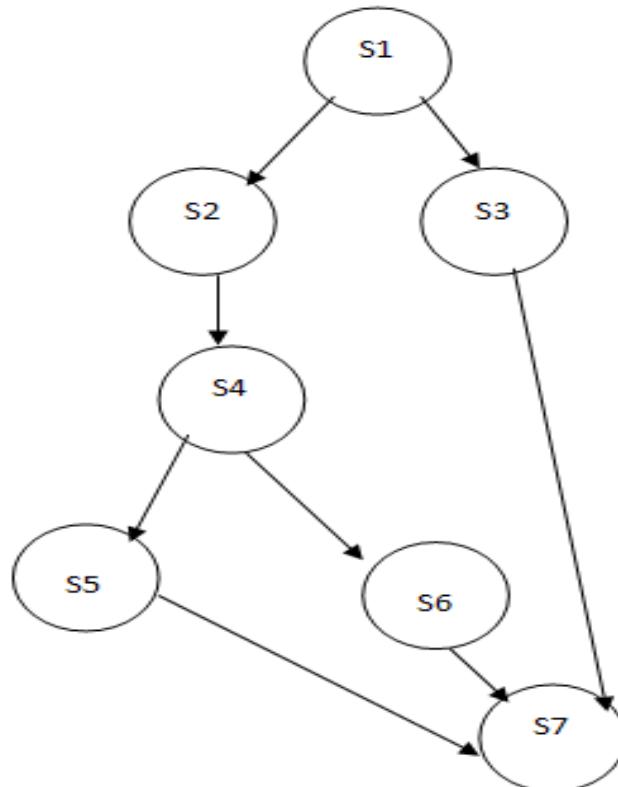
Sn;...;

Parend;

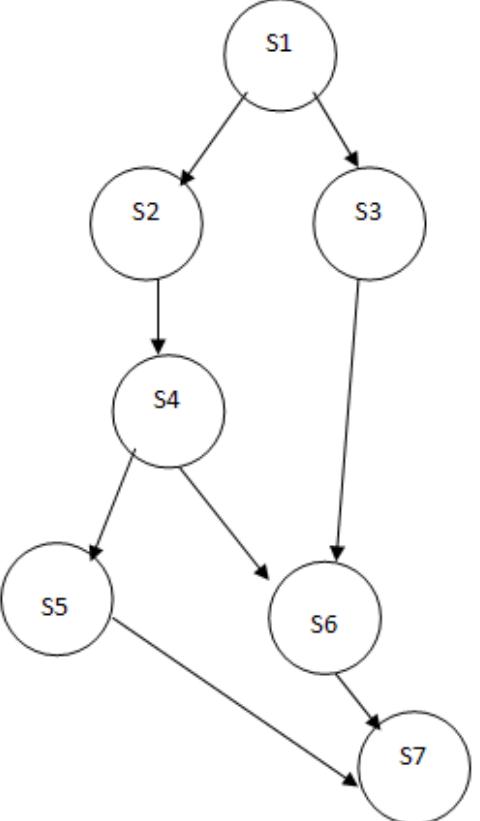


Örnek

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



Fork-Join/Parbegin-parend

	<pre>S1; Count1:=2; FORK L1; S2; S4; Count2:=2; FORK L2; S5; Goto L3; L1:S3; L2:JOIN count1; S6; L3:JOIN count2; S7;</pre>
--	--

Bu öncelik grafini sadece parbegin-parend yapısı kullanarak gerçekleştiremeyiz.

Nesne Yönelimli Yaklaşım

Nesne(object): verileri ve bu veriler üzerinde işlem yapan üye fonksiyonları birleştiren yapı.

Üye fonksiyonlar o nesnenin verilerine erişmeyi sağlayan tek yoldur.
Veriye doğrudan ulaşılmaz.

Verilerin paketlenmesi(encapsulation) : verilerin ve üye fonksiyonlarının tek bir çatı altında paketlenmesidir.

Veri gizliliği(data hiding) : Nesne içerisindeki veriler üye fonksiyonlarla erişildiğinden, veriye doğrudan erişilmediğinden veriler korunmuş olur buna veri gizliliği denir.

Nesne Yönelimli dillerin özellikleri

Nesneler: nesne yönelimli bir dilde programlama problemini çözmemiz gerektiğinde, problemin fonksiyonlara nasıl bölüneceği değil nesnelere nasıl bölüneceği düşünülmelidir.

Sınıflar : Bir dizi benzer nesnenin genel tanımıdır. Örneğin taşıt sınıfının nesneleri, kamyon, otobüs, minibüs vs dir.

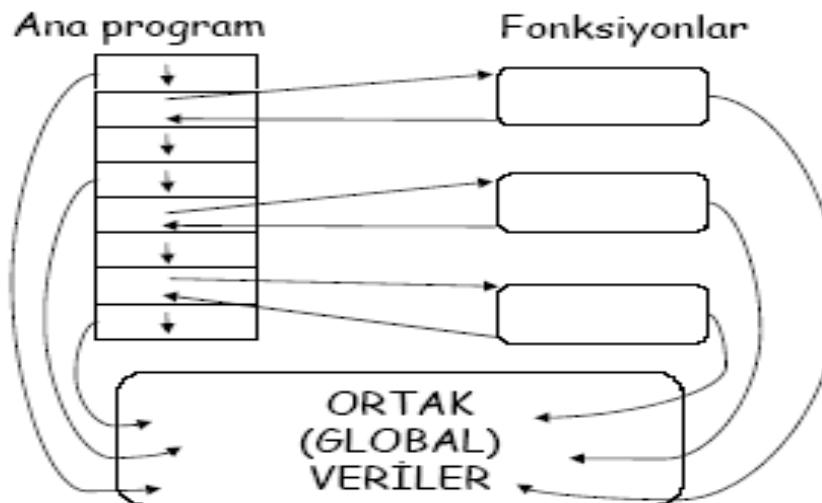
Kalıtım: Miras alma(inheritance) olayıdır. Örneğin taşıt sınıfının üyeleri araba, kamyon, otobüs vs dir. Bunlarda birer sınıf oluşturmaktadırlar örneğin araba sınıfında, çok çeşitli model ve markada araba nesneleri mevcuttur. Yani bir sınıfın, alt sınıflar türetilabilir. Alt sınıf üyesi olduğu sınıfın özelliklerini miras alır(taşır).

Yeniden kullanılabılırlik: bir sınıfı hatasız olarak oluşturuktan sonra, diğer programcılara kendi programlarında kullanmaları için dağıtılabılır.

Çok biçimlilik(polymorphism): operatörler ve fonksiyonlar işlevlerine bağlı olarak farklı şekillerde kullanılabilir.

Emir Esaslı(Procedural) Programlama Yöntemi

- Basic, Fortran, Pascal, C gibi programlama dillerinin desteklediği bu yöntemde öncelikle gerçeklenmek istenen sistemin yapması gereken iş belirlenir.
- Büyük boyutlu ve karmaşık işler, daha küçük ve basit işlevlere (fonksiyon) bölünerek gerçekleştirirler.



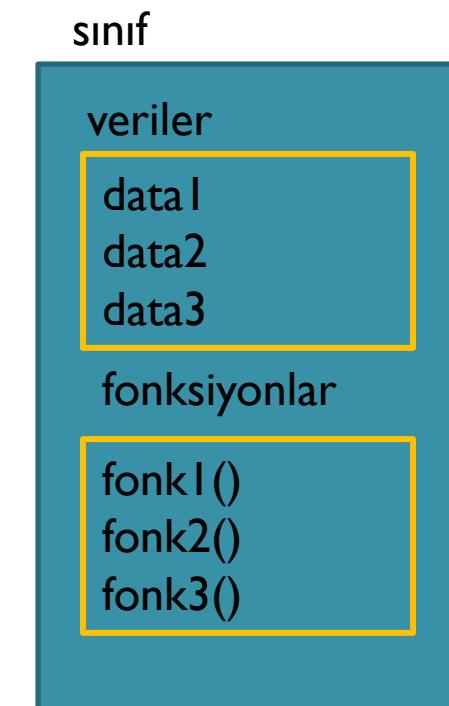
Emir Esaslı Programlama Yönteminin Değerlendirmesi

- “Böl ve yönet” prensibine dayanır. Amaç büyük programları küçük parçalara bölerek yazılım geliştirme işini kolaylaştırmaktır.
- Ancak yazılımların karmaşıklıkları sadece boyutlarından kaynaklanmaz. Küçük problemler de karmaşık olabilir.
- Gerçek dünyadaki sistemler sadece fonksiyonlardan oluşmaz. Dolayısıyla emir esaslı yaklaşımda karmaşık bir problemin gerçeğe yakın bir modelini bilgisayarda oluşturmak zordur.
- Tasarım aşamasında verilerden çok fonksiyonlara odaklanıldığından hatalar nedeniyle veri güvenliği tehlikeye girebilmektedir.
- Kullanıcılar kendi veri tiplerini çok güçlü biçimde tanımlayamazlar.
- Programın güncellenmesi zordur.
- İşleve dayalı yöntemi de kullanarak kaliteli programlar yazmak mümkün değildir. Ancak nesneye dayalı yöntem kaliteli programların oluşturulması için programcılara daha çok olanak sağlamaktadır ve yukarıda açıklanan sakıncaları önleyecek yapılarla sahiptir.

Nesneye Dayalı (Object-Oriented) Programlama Yöntemi

- Gerçek dünya nesnelerden oluşmaktadır.
- Çözülmek istenen problemi oluşturan nesneler, gerçek dünyadaki yapılarına benzer bir şekilde bilgisayarda modellenmelidir.
- Nesnelerin yapıları iki bölümden oluşmaktadır: **1. Nitelikler**(özellikler ya da durum bilgileri), **2. Davranışlar**(yetenekler)
- Tasarım yapılırken sistemin işlevi değil, sistemi oluşturan **veriler** esas alınır.
- Aşağıdaki elemanlar nesne olarak modellenebilir:
- İnsan kaynakları ile ilgili bir programda; memur, işveren, işçi, müdür, genel müdür.
- Grafik programında; nokta, çizgi, çember, silindir.
- Matematiksel işlemler yapan programda; karmaşık sayılar, matris.
- Kullanıcı arayüzü programında; pencere, menü, çerçeve.

Veri ve fonksiyonları tek bir bütün haline getirmek nesne yönelimli programlamanın temel yaklaşımıdır.



Sınıf ve nesneler

Bir nesne ile sınıf arasındaki ilişki tipki bir değişken ile veri tipi arasındaki ilişki gibidir. Bir nesne kendi sınıfının bir örneğidir.

Bir programda sınıf (**basitnesne**) önce tanımlanacak , main() fonksiyonu içerisinde sınıfa ait nesneler (**b1,b2**) üretilecektir.

Sınıf oluşturma class anahtar kelimesi ile başlar, ardından sınıfın adı gelir.Yapı tanımlamasında olduğu gibi sınıfın gövdesi küme parantezleri ile ayrılır ve sonuna (;) konur.

Private(özel): private olarak tanımlanan veri ve fonksiyonlara sadece, tanımlı oldukları sınıf içerisinde erişilebilir.

Public(genel): public olan veri ve fonksiyonlara tanımlı oldukları sınıfların dışından erişmek mümkündür.

Genellikle bir sınıfındaki fonksiyonlar public, veriler private tır.Veriler üzerinde kazara oynanmasın diye private olarak tanımlanırlar. Üzerinde işlem yapılan fonksiyonlar ise sınıfın dışından erişilebilsin diye public olarak tanımlanırlar.

Bu bir kural değildir.

Yöntemin Değerlendirmesi:

- Gerçek dünya nesnelerden oluştugundan bu yöntem ile sistemin daha gerçekçi bir modeli oluşturulabilir. Programın okunabilirliği güçlenir.
- Nesne modellerinin içindeki veriler sadece üye fonksiyonların erişebileceği şekilde düzenlenebilirler. Veri saklama (data hiding) adı verilen bu özellik sayesinde verilerin herhangi bir fonksiyon tarafından değiştirilmesi önlenir.
- Programcılar kendi veri tiplerini yaratabilirler.
- Bir nesne modeli oluşturuktan sonra bu modeli çeşitli şekillerde defalarca kullanmak mümkündür (reusability).
- Programları güncellemek daha kolaydır.
- Nesneye dayalı yöntem takım çalışmaları için uygundur.

NESNEYE YÖNELİK PROGRAMLAMA DİLLERİ

- Programlama dilleri literatüründe sınıf ve alt sınıf kavramını ilk olarak SIMULA67dili tanıtmıştır.
- Smalltalk, Eiffel, ADA95 ve Java dillerinde tüm veriler nesne şeklindedir. Yani bu diller tam nesne yönelimlidir.
- C++ ise hem emir esaslı paradigmayı hem de nesneye yönelik paradigmayı destekler.

Smalltalk

- SIMULA67'de tanıtılan fikirler, Smalltalk ile güçlenmiş ve Smalltalk ile nesne yönelimli dil popüler hale gelmiştir
- Smalltalk'ta bir program sadece kalıtım hiyerarşisi içinde düzenlenmiş birbirleriyle mesajlar ile etkileşen nesne sınıflarından oluşabilir. Smalltalk'ta tüm veriler nesnelerle gösterilmek zorunda olduğu için tam nesneye yönelik bir programlama dili olarak nitelendirilir.
- Smalltalk dilinde bütün bağlamlar dinamik olarak gerçekleşir.
- Smalltalk'ta sınıfların sadece tek üst sınıfı bulunabilir (tekli kalıtım modeli).

C++

- C diline sınıf tanımlama, sınıf türetme, *public/private* erişim kontrolü, *constructor/deconstructor* ve metod yükleme özelliklerinin ve çoklu kalıtım, soyut sınıflar, sanal metodlar eklenesiyle elde edilmiş halidir.
- *Constructor*, bir nesne yaratıldığında bir kez çalıştırılan özel bir metod olmaktadır. Benzer şekilde bir nesne yok edildiği zaman bir *destructor* metodу varsayılan olarak çağrılır.
- Bir sınıf için bir veya daha fazla *Constructor* metodу tanımlanabilir.
- C++'da bağlama genellikle durağan olarak gerçekleşir.
- Ancak, *virtual* metodlar ve göstergeler, dinamik bağlama etkisi vermek için kullanılır. Metodların dinamik olarak bağlanabilmesi için metod, üst sınıfta *virtual* olarak tanımlanmalı ve daha sonra türetilmiş sınıflarda yeniden tanımlanmalıdır.

Java

- Java, çeşitli elektronik aygıtlara yazılım geliştirmek için geliştirilmiş bir programlama dilidir.
 - Java'da gösterge tipi yoktur.
- Yorumlayıcıya dayalı gerçekleştiriminden dolayı Java taşınabilirdir. Bir Java kaynak kodundan bytecode adı verilen bir ara kod üretilir ve bytecode yorumlayıcısının bulunduğu her makina bu programı çalıştırabilir.
- Birçok Internet tarayıcısı (browser) Java programlarını doğrudan yükleyebilir ve çalıştırabilir. Bu özelliği nedeniyle Java ağ programlama dili olarak nitelenmektedir.
- Java'da sınıflar arasında tekli kalıtımı izin verilmiştir. Ancak çoklu kalıtımı desteklemek için ayrı arayüz modülleri sağlanmıştır.
- Java'da programlamada eszamanlılık (concurrency) önceden tanımlı olan thread sınıfı ile desteklenir.
- Dinamik bellek yönetimi için, otomatik bellek düzenlemeye (garbage collection) gerçekleştirilmektedir.

Java ve C++

- *public, protected* ve *private* tanımlayıcılar Java'da da geçerlidir. Java'da yeni nesne tiplerinin tanımlanması için sınıf yapısı vardır. Bir sınıfta, veri sahaları ve metodlar vardır. Bir sınıfın bir örneği, o nesneyi oluşturan sahaların kendine ilişkin kopyalarını içerir.
- Java'da da *constructor* ve *destructor* metodları her sınıf için tanımlanabilir. Metod yükleme constructor metodlarına da uygulanabilir.
- Java'da tüm sınıflar, *object* adlı kökten türetilmiş bir hiyerarşi ağacının düğümleridir. Her sınıf, *object* sınıfından türetilmiştir.
- Java, soyut sınıfların ve metodların tanımlanmasını destekler. Soyut bir metod, bir alt sınıf tarafından gerçekleştirilmelidir.
- Java'da tekli kalıtima izin verilir. Ama aynı zamanda alt sınıflamanın kısıtlanması da olasıdır. Java'da *public, private, protected* tanımlarına ek olarak, kalıtım süreci değiştirilebilmesi için *static, abstract* ve *final* tanımlayıcıları bulunur.

static, abstract ve final tanımlayıcıları

- *static* özelliğindeki bir veri sahası, bir sınıfın tüm örneklerinde paylaşılır. *static* bir metod, sınıfın bir örneği yaratılmadan da çağrılabılır ve *static* bir metod, bir alt sınıfta yeniden tanımlanamaz.
- *abstract* olarak tanımlanmış bir sınıf örneklenemez ve sadece üst sınıf olabilir. Bir *abstract* metod ise bir alt sınıf tarafından gerçekleştirilmek zorundadır.
- *final* olarak tanımlanmış bir sınıfın alt sınıfları tanımlanamaz ve *final* olarak tanımlanmış bir metod, herhangi bir alt sınıfta yeniden tanımlanamaz.

Özet

- Bu bölümde emir esaslı paradigma ile nesneye yönelik programlamanın temel özellikleri karşılaştırılmış ve
- Sınıf, Nesne, Kalıtım, Çokyapılılık gibi nesne yönelimli kavramlar tanıtılmıştır.
- Smalltalk, C++, Java gibi nesneye yönelik dillerin genel özellikleri anlatılmıştır.

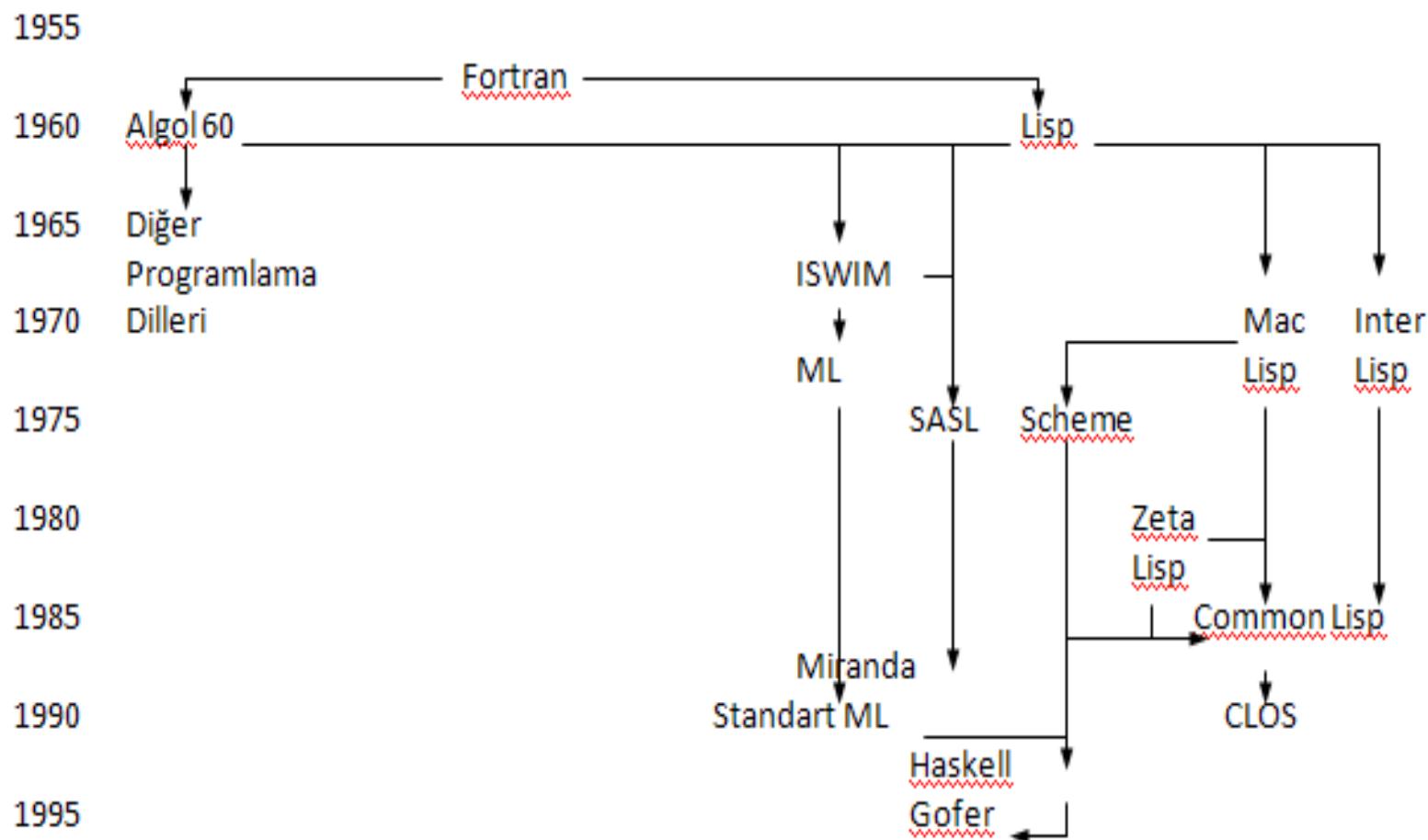
Fonksiyonel Programlama

LISP, ML, Haskel, Scheme

- Emir esaslı dillerin tasarıımı doğrudan doğruya von Neumann mimarisine dayanır.
- Bir imperative dilde, işlemler yapılır ve sonuçlar daha sonra kullanım için değişkenlerde(variables) tutulur. Emir esaslı dillerde değişkenlerin yönetimi karmaşıklığa yol açar.
- Fonksiyonel dillerin tasarıımı Matematiksel Fonksiyonlara dayalıdır ve değişkenler(variables), matematikte olduğu gibi gerekli değildir. Kullanıcıya da yakın olan sağlam bir teorik temele sahiptir.
- Fonksiyonel programlamada , bir fonksiyon aynı parametreler verildiğinde daima aynı sonucu üretir (referential transparency).

- 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.

- 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.



- 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 olur.
Fonskiyonn(..(fonksiyon2(fonksiyon1(veriler))))..)
- Bu dillerde, alt yordamlar,fonksiyonlar (prosedürler) kullanılarak program daha alt parçalara bölünür.

Örnek:

- factorial (n) = if (n=0) then 1 else (n * factorial(n-1))
- (define factorial
- (lambda (n)
- (if (zero ? n)
- 1
- (* n (- n 1))))))

- Lisp dili Sembolik veri işleme amacı ile dizayn edilmiştir. Bu dil türev ve integral hesaplamalarındaki, elektrik devre teorisideki , matematiksel mantık oyunlarındaki ve yapay zekanın diğer alanlarındaki sembolik hesaplamalarda kullanılmaktadır. Karmaşık hesaplamalar daha basit ifadeler cinsinden yazılarak kolaylıkla çözümlenebilir.

Haskell

- Kuvvetli tipli, statik kapsam bağlamalı ve tip yorumlamalı bir dildir.
- Tam olarak fonksiyonel bir dildir. (değişkenler yoktur, atama ifadeleri yoktur, hiçbir çeşit yan etki yoktur).
- Tembel değerlendirme(**lazy evaluation**) kullanır (değer gerekmemiş olduğu sürece hiçbir alt-ifadeyi değerlendirmeme)
- Liste kapsamları(**list comprehensions**), sonsuz listelerle çalışmaya izin verir

Örnekler

1. **fib 0 = 1**

fib 1 = 1

fib (n + 2) = fib (n + 1)
+ fib n

2.

fact n

| **n == 0 = 1**

| **n > 0 = n * fact (n - 1)**

3. Liste işlemleri

– Liste gösterimi:

– `directions = ["north", "south", "east", "west"]`

– Uzunluk(Length): #

#directions = 4

– .. operatorü ile aritmetik seriler

`[2, 4..10]` gösterimi `2, 4, 6, 8, 10` olarak değerlendirilir.

Örnekler devam

3. Liste işlemleri(devam)

- ++ ile zincirleme(Catenation)

[10, 30] ++ [50, 70] → [10, 30, 50, 70]

- :operatörü yoluyla

1: [3, 5, 7] → [1, 3, 5, 7]

```
product [] = 1
product (a:x) = a * product x

fact n = product [1..n]
```

Haskell örnekler (devam)

4. Liste kapsamları:küme gösterimi

```
[n * n | n ← [1..10]]
```

ilk 10 pozitif tamsayının karelerinden oluşan
bir liste tanımlar

```
factors n = [i | i ← [1..n `div`  
2],  
              n `mod` i == 0]
```

Bu fonksiyon verilen parametrenin bütün
çarpanlarını hesaplar

Haskell örnekleri

- Quicksort(Hızlı Sıralama):

```
sort [] = []
```

```
sort (a:x) = sort [b | b <- x; b
<= a]
```

```
++ [a] ++
```

```
sort [b | b <- x; b > a]
```

Haskell örnekleri (devam)

5. Tembel değerlendirme(Lazy evaluation)

```
positives = [0..]
```

```
squares = [n * n | n ← [0..]]
```

(sadece gerekli olanları hesapla)

```
member squares 16
```

True döndürür

Haskell örnekleri

- Üye(member) şu şekilde de yazılabilirdi:

```
member [] b = False
```

```
member (a:x) b=(a == b) || member x b
```

- Ancak, bu sadece kare(square) olan parametre tamkare olduğu zaman çalışacaktı; eğer değilse, sonsuza kadar üretmeye devam edecekti. Şu versiyon her zaman çalışır:

```
member2 (m:x) n
  | m < n= member2 x n
  | m == n      = True
  | otherwise    = False
```

Fonksiyonel Dillerin uygulamaları

- LISP yapay zeka(artificial intelligence) için kullanılır
 - Bilgi gösterimi
 - Makine öğrenmesi(Machine learning)
 - Doğal Dil İşleme(Natural language processing), Konuşma ve görmeyi modelleme

Fonksiyonel –Emir Esaslı

Emir Esaslı (imperative) diller	fonksiyonel diller:
Verimli çalışma	Verimsiz çalışma
Karmaşık semantik(semantics)	Basit semantik(semantics)
Karmaşık sentaks(syntax)	Basit sentaks(syntax)
Eşzamanlılık(kullanıcı tanımlı)	Eşzamanlılık (otomatik)