

CENG 218 Programlama Dilleri

Bölüm 1: Giriş

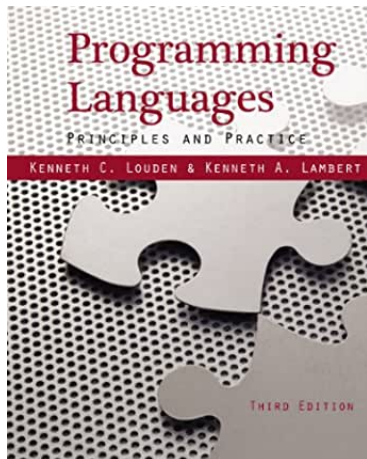
Öğr.Gör. Şevket Umut Çakır

Pamukkale Üniversitesi

Hafta 1

Ders Kitabı

- Louden, K. C., & Lambert, K. A. (2011). Programming Languages: principles and practices
- Kitap bağlantısı(EDS'yi kontrol edin)



Şekil: Ders kitabı



Üniteler

- 1 Giriş
- 2 Dil Tasarım Kriterleri
- 3 Fonksiyonel Programlama
- 4 Mantıksal Programlama
- 5 Nesne-yönelimli Programlama
- 6 Sözdizimi
- 7 Temel Semantik
- 8 Veri Tipleri
- 9 Kontrol I - İfadeler ve Komutlar
- 10 Kontrol II - Prosedürler ve Ortamlar
- 11 Soyut Veri Tipleri ve Modüller
- 12 Biçimsel Semantik
- 13 Paralel Programlama



Ders Saatleri ve Değerlendirme

- Ders Saatleri
 - ▶ Çarşamba 09:50 - 12:25 (Uzaktan, Canlı Ders Sistemi)
- Değerlendirme yöntemleri
 - ▶ Arasınava %40 (Çoktan seçmeli)
 - ▶ Dönem sonu sınavı %60 (Çoktan seçmeli)



Hedefler

- Nitelikleri(özellikleri), bağlamayı ve anlamsal işlevleri anlama
- Bildirimleri, blokları ve kapsamı anlama
- Bir sembol tablosunun nasıl oluşturulacağını öğrenme
- Ad çözümlemesini ve aşırı yüklemeyi anlama
- Tahsisi, yaşam sürelerini ve ortamı anlama



- Bilgisayarları nasıl programladığımız, hesaplama hakkındaki düşüncelerimizi etkiler ve bunun tersi de geçerlidir
- Programlama dillerinin temel ilkeleri ve kavramları, bilgisayar biliminin temel bilgi birikiminin bir parçasıdır
- Bu ilkelerin incelenmesi, programcılar ve bilgisayar bilimcileri için çok önemlidir
- Bu bölüm, programlama dillerinin temel kavramlarını tanıtmakta ve bazı temel kavramları özetlemektedir



Programlama Dilleri Zaman Çizelgesi

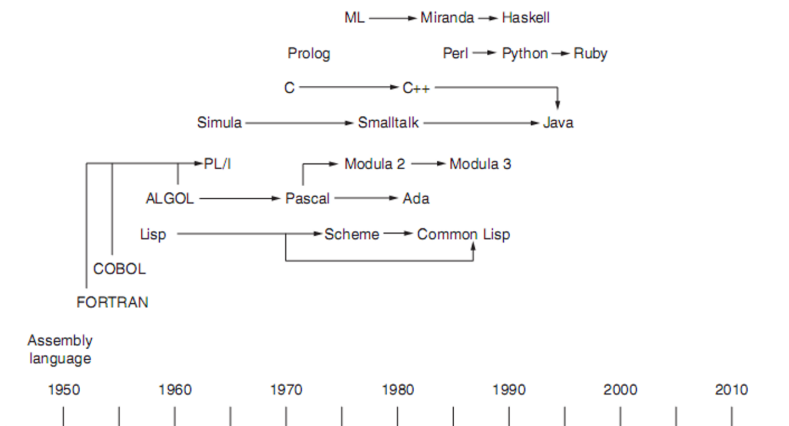


Figure 1.1 A programming language timeline



Programlama Dillerinin Kökenleri

- **Programlama dili:** genellikle "bir bilgisayara ne yapmasını istediğimizi iletmek için bir gösterim(notasyon)" olarak tanımlanır
- 1940'ların ortalarından önce, bilgisayar operatörleri, istenen görevleri yerine getirmek için bir bilgisayarın dahili kablolarını ayarlamak için anahtarlar(şalter) kullandılar
- Programlama dilleri, bilgisayar kullanıcılarının donanımı yeniden yapılandırmak zorunda kalmadan sorunları çözmelerine olanak tanıdı



Makine Dili ve İlk Depolanan Programlar

- **John von Neumann:** Bilgisayarların genel amaçlı işlemlerin küçük bir kümesi ile kalıcı olarak donatılması gerektiğini önerdi
- Operatörün daha spesifik sorunları çözmek için temel donanım işlemlerini düzenlemek için bir dizi ikili kod girmesine izin verir
- Operatörler, makine dili adı verilen bu kodları belleğe girmek için anahtarları çevirebilir



Makine Dili ve İlk Depolanan Programlar

```
0010001000000100
0010010000000100
0001011001000010
0011011000000011
1111000000100101
0000000000000101
0000000000000110
0000000000000000
```

Şekil: Bir makine dili programı



Makine Dili ve İlk Depolanan Programlar

- Her kod satırında 16 bit veya ikili rakam bulunur
 - ▶ Tek bir makine dili emrini(instruction) veya tek bir veri değerini temsil eder
- Program yürütme(icrası), kodun ilk satırıyla başlar
 - ▶ Kod bellekten alınır, kodu çözülür (yorumlanır) ve çalıştırılır
- Kontrol daha sonra sonraki kod satırına geçer ve durdurma emrine ulaşılan kadar devam eder
- **İşlem kodu(Opcode):** bir kod satırının ilk 4 biti
 - ▶ Gerçekleştirilecek işlemin türünü gösterir



Makine Dili ve İlk Depolanan Programlar

- Sonraki 12 bit, komutun işlenenleri(operand) için kod içerir
- Operand kodları, makine kaydedicilerinin sayısıdır veya diğer verilerin adresleriyle veya bellekteki talimatlarla ilgilidir
- Makine dili programlaması sıkıcıydı ve hataya açıktı



Assembly Dili, Sembolik Kodlar ve Yazılım Araçları

- **Assembly dili:** emir(instruction) kodları ve hafıza konumları için bir dizi anımsatıcı sembol
 - ▶ Örnek: **LD R1, FIRST**
- **Assembler:** sembolik assembly dili kodunu ikili makine koduna çeviren bir program
- **Yükleyici(Loader):** makine kodunu bilgisayar belleğine yükleyen bir program
- Giriş cihazları:
 - ▶ Keypunch makinesi
 - ▶ Kart okuyucu



Keypunch Makinesi



Şekil: Keypunch makinesi



Assembly Dili, Sembolik Kodlar ve Yazılım Araçları

Assembly Örneği

```
.ORIG X3000      ; İlk emirin adresi(onaltılık)
LD R1, FIRST     ; FIRST bellek alanındaki sayıyı R1 kaydedicisine yükle
LD R2, SECOND    ; SECOND bellek alanındaki sayıyı R2 kaydedicisine yükle
ADD R3, R2, R1   ; R1 ve R2'deki sayıları topla ve toplamı R3'e yaz

ST R3, SUM       ; R3'deki sayıyı SUM bellek alanına kopyala
HALT             ; Programı durdur
FIRST .FILL #5   ; FIRST konumu onluk 5 sayısını içerir
SECOND .FILL #6  ; SECOND konumu onluk 6 sayısını içerir
SUM .BLKW #1     ; SUM konumu (varsayılan olarak 0 içerir)
.END            ; Programın sonu
```

Şekil: İki sayıyı toplayan bir assembly dili programı



Assembly Dili, Sembolik Kodlar ve Yazılım Araçları

- Anımsatıcı semboller, ikili makine kodlarına göre bir gelişmeydi ancak yine de eksiklikleri vardı
- Geleneksel matematiksel gösterimin soyutlaması yok
- Her bilgisayar donanımı mimarisi türünün kendi makine dili talimat seti vardır ve kendi assembly dili lehçesini gerektirir
- Assembly dilleri ilk olarak 1950'lerde ortaya çıktı ve bugün hala düşük seviyeli sistem araçları veya el optimizasyonu için kullanılıyor



FORTRAN ve Cebirsel Gösterim

- **FORTAN:** FORmula TRANslation dili
 - ▶ 1950'lerin başında John Backus tarafından geliştirildi
 - ▶ Belirli bir makine türünün mimarisini yansıtıyordu
 - ▶ Daha sonraki yüksek seviyeli dillerin yapılandırılmış kontrol ifadeleri ve veri yapıları eksikti
- Cebirsel notasyon ve kayan noktalı sayıları desteklemesi nedeniyle bilim adamları ve mühendisler arasında popüler



ALGOL Ailesi: Yapısal Soyutlamalar ve Makine Bağımsızlığı

- **ALGOL:** ALGOritmic Language 1960'da yayınlandı
 - ▶ Bilgisayar bilimcilerinin dergilerde algoritmalar yayınlamaları için standart bir gösterim sağladı
 - ▶ Sıralama (başlangıç-bitiş blokları), döngüler (for döngüsü) ve seçim (if ve if-else ifadeleri) için yapılandırılmış kontrol ifadeleri dahil edildi
 - ▶ Farklı sayısal türler desteklenmekteydi
 - ▶ Dizi yapısını tanıttı
 - ▶ Öz yinelemeli prosedürler dahil prosedürler desteklenmekteydi



ALGOL Ailesi

- ALGOL, her donanım türüne sahip bir ALGOL derleyicisi gereksinimi ile makine bağımsızlığını elde etti
- **Derleyici(Compiler)**: programlama dili ifadelerini makine koduna çevirir
- ALGOL, resmi bir tarifname(formak specification) veya tanım alan ilk dildi
 - ▶ Hem programcılar hem de derleyici yazarlar için özelliklerini tanımlayan bir dilbilgisi içeriyordu



- Aşağıdakiler dahil çok sayıda üst düzey dil ALGOL'den türemiştir:
 - ▶ **Pascal**: 1970'lerde programlama öğretmek için kullanılan dil
 - ▶ **Ada**: ABD Savunma Bakanlığı'nın gömülü uygulamaları için



Von Neumann Mimarisi olmadan hesaplama

- Üst düzey diller hala bir makinenin von Neumann modelinin temelini oluşturan mimariyi yansıtıyordu
 - ▶ Programlar ve veriler için hafıza alanı saklanır
 - ▶ Bellekten alınan talimatları sıralı olarak yürüten ayrı merkezi işlem birimi
- İşlemci hızındaki gelişmeler ve programlama dillerinde artan soyutlama bilgi çağına yol açtı



Von Neumann Mimarisi olmadan hesaplama

- Dil soyutlaması ve donanım performansındaki ilerleme, ayrı engellerle karşılaştı:
 - ▶ Donanım, Moore Yasası tarafından öngörülen iyileştirmelerin sınırlarına ulaşmaya başladı ve çok çekirdekli yaklaşıma yol açtı
 - ▶ Büyük programların hata ayıklaması ve düzeltilmesi zordu
 - ▶ Tek işlemcili hesaplama modeli, paralel olarak çalışan birden çok CPU'nun yeni mimarisine kolayca eşleştirilemez



Von Neumann Mimarisi olmadan hesaplama

- Çözüm: Dillerin belirli bir donanım modeline dayanması gerekmez, ancak yalnızca problem çözme tarzlarına uygun hesaplama modellerini desteklemesi gerekir
- **Lambda hesabı**: matematikçi Alonzo Church tarafından geliştirilen hesaplamalı model
 - ▶ Özyinelemeli fonksiyonlar teorisine göre
- **Lisp**: fonksiyonel hesaplama modelini kullanan programlama dili



Von Neumann Mimarisi olmadan hesaplama

- Kendilerini paralel işlemeye uygun hale getiren von Neumann dışı hesaplama modellerinde modellenen diğer diller şunlardır:
 - ▶ Otomatik teoremi ispatlayan resmi bir mantık modeli
 - ▶ Nesnelerin mesaj geçişi yoluyla etkileşimini içeren bir model



Programlama Dillerinde Soyutlamalar

- İki tür programlama dili soyutlaması vardır:
 - ▶ Veri soyutlaması
 - ▶ Kontrol soyutlaması
- **Veri soyutlamaları:** insanlar için verilerin davranışını ve niteliklerini basitleştirir
 - ▶ Örnekler: sayılar, karakter dizeleri, arama ağaçları
- **Kontrol soyutlamaları:** kontrol aktarımının özelliklerini basitleştirir
 - ▶ Örnekler: döngüler, koşullu ifadeler, prosedür çağrıları



Programlama Dillerinde Soyutlamalar

- Soyutlamalar, seviyelere göre de kategorize edilebilir (soyutlamada bulunan veya gizlenen bilgi miktarının ölçüleri)
- **Temel soyutlamalar:** en yerelleştirilmiş makine bilgilerini toplar
- **Yapılandırılmış soyutlamalar:** bir programın yapısı hakkında ara bilgiler toplar
- **Birim soyutlamaları:** bir programda büyük ölçekli bilgiler toplar



Veri: Temel Soyutlamalar

- Temel veri soyutlaması:
 - ▶ Ortak veri değerlerinin dahili temsilini gizler
 - Değerler aynı zamanda "ilkel"(primitive) veya "atomik" olarak da adlandırılır çünkü programcı normal olarak dahili temsilin bileşen parçalarına veya bitlerine erişemez
- **Değişkenler(Variables)**: veri değerleri içeren bilgisayar bellek konumlarını gizlemek için sembolik adların kullanılması
- **Veri türleri**: veri değerlerinin türlerine verilen adlar
- **Bildirim(Declaration)**: bir değişkene bir isim ve bir veri türü verme işlemi
- Toplama ve çarpma gibi standart matematiksel işlemler



Veri: Yapılandırılmış Soyutlamalar

- Veri yapısı: ilgili veri değerlerini tek bir birimde toplar
 - ▶ Bileşen parçalarını gizler, ancak parçalardan inşa edilebilir ve parçalara erişilebilir ve değiştirilebilir
- Örnekler:
 - ▶ Çalışan kaydı ad, adres, telefon, maaş (farklı veri türleri) içerir
 - ▶ **Dizi**: aynı veri türüne sahip tek tek dizinlenmiş öğeler dizisi
 - ▶ Metin dosyası: harici bir depolama cihazına ve cihazından aktarım için bir dizi karakter



Veri: Birim Soyutlamaları

- Bilgi gizleme: bilgileri gizleyen yeni veri türlerini (veriler ve işlemler) tanımlama
- Birim soyutlaması: genellikle soyut bir veri türü kavramıyla ilişkilendirilir
 - ▶ Bir dizi veri değeri ve bu değerler üzerindeki işlemler
- Arayüzü(interface) gerçekleştirmeden(implementation) ayırır
 - ▶ **Arayüz(interface)**: kullanıcı tarafından kullanılabilen işlemler kümesi
 - ▶ **Gerçekleştirme(implementation)**: veri değerlerinin ve işlemlerinin dahili temsili



Veri: Birim Soyutlamaları

- Örnekler:
 - ▶ ML, Haskell ve Python'da **modül(module)**
 - ▶ Lisp, Ada ve Java'da **paket(package)**
 - ▶ Nesne yönelimli dillerde sınıf mekanizması
- Birim soyutlaması ayrıca **yeniden kullanılabilirlik(reusability)** sağlar
- Tipik olarak, bileşenler bir kitaplığa girilir ve kitaplık mekanizmalarının temeli haline gelir
 - ▶ Birlikte çalışabilirlik(interoperability), birimlerin kombinasyonuna izin verir
- **Uygulama programlama arayüzü (API):** kaynağın bileşenleri hakkında bilgi verir



Kontrol: Temel Soyutlamalar

- **Temel kontrol soyutlamaları:** birkaç makine emirlerini anlaşılması daha kolay soyut bir ifadede birleştiren ifadeler
- **Sözdizimsel şeker(Syntactic sugar):** karmaşık bir gösterimi daha basit, kısaltılmış bir gösterimle değiştirmenize izin veren bir mekanizma
 - ▶ Örnek: $x = x + 10$ yerine $x += 10$



Kontrol: Yapılandırılmış Soyutlamalar

- **Yapılandırılmış kontrol soyutlamaları:** bir programı, yürütölmelerini yöneten testlerin içine yerleştirilmiş talimat gruplarına ayırır
 - ▶ Sıralama, seçim ve yinelemenin birincil kontrol yapılarının mantığını ifade etmeye yardımcı olur
- **Dallanma(Branch)** emirleri: sonraki konum dışındaki bellek konumlarına seçim ve yinelemeyi destekleyen emirler



Kontrol: Yapısal Soyutlamalar

Assembly Örneği

```
LEA R1, LIST      ; Dizinin başlangıç adresini yükle
AND R2, R2, #0     ; Toplam değerini 0 olarak belirle
AND R3, R3, #0     ; Sayacı 10 olarak belirle
ADD R3, R3, #10    ; (geri sayım için)
WHILE LDR R4, R1, #0 ; Döngü başlangıcı: Mevcut dizi hücresinden veriyi
↳ yükle
BRZP INC          ; Eğer >= 0 ise sonraki iki adımı atla
NOT R4, R4        ; Eğer < 0 ise ikili tümleyeni kullanarak
↳ negatifini al
ADD R4, R4, #1    ;
INC ADD R2, R2, R4 ; Toplamı artır
ADD R1, R1, #1    ; Bir sonraki dizi hücresine geçmek için adresi
↳ artır
ADD R3, R3, #-1   ; Sayacı azalt
BRP WHILE        ; Eğer sayacı > 0 ise döngünün başına git
ST R2, SUM       ; Toplamı belleğe sakla(yaz)
```

Şekil: Assembly dilinde dizi-tabanlı bir döngü örneği



Kontrol: Yapısal Soyutlamalar

C++/Java Örneği

```
int sum = 0;
for (int i = 0; i < 10; i++) {
    int data = list[i];
    if (data < 0)
        data = -data;
    sum += data;
}
```

Şekil: C++ veya Java dilinde dizi-tabanlı
bir döngü örneği



Kontrol: Yapısal Soyutlamalar

- **Yineleyici(Iterator):** bir koleksiyonla ilişkili bir nesne (dizi, liste, küme veya ağaç gibi)
 - ▶ Bir koleksiyonda bir yineleyici açın, ardından yineleyicinin yöntemlerini kullanarak tüm öğeleri ziyaret edin
- Java'da yineleyici için sözdizimsel şekeri: geliştirilmiş for döngüsü

```
int sum = 0;
for (int data:list) {
    if (data < 0)
        data = -data;
    sum += data;
}
```



Kontrol: Yapısal Soyutlamalar

- **Prosedür(Procedure)** (veya **alt program(subprogram)** veya **alt rutin(subroutine)**): bir eylem dizisini, programdaki diğer noktalardan çağrılabilen tek bir eylemde gruplandırır.
 - ▶ **Prosedür bildirimi(declaration)**: bir prosedürü adlandırır ve bunu gerçekleştirilecek eylemlerle ilişkilendirir
 - ▶ **Çağrı(Invocation)** (veya **prosedür aktivasyonu(activation)**): prosedürü çağırma eylemi
 - ▶ **Parametreler**: çağırmadan çağırmaya değişebilen değerler
 - ▶ **Argümanlar(Arguments)** (veya **gerçek parametreler(actual parameters)**): parametreler için çağırın tarafından sağlanan değerler



Kontrol: Yapısal Soyutlamalar

Ada Prosedür Örneği

```
procedure gcd (u, v: in integer; x: out integer) is
    y, t, z: integer;
begin
    z := u;
    y := v;
    loop
        exit when y = 0;
        t := y;
        y := z mod y;
        z := t;
    end loop;
    x := z;
end gcd;
```

Şekil: Ada dilinde obeb(gcd/greatest common divisor) prosedürü



Kontrol: Yapısal Soyutlamalar

- **Çalışma zamanı ortamı(Runtime environment):** programın sistem gerçekleştirmesi(implementation)
 - ▶ Programın durumu ve prosedürlerin işleyiş şekli hakkında bilgi depolar
- **Fonskiyon:** bir prosedürle yakından ilgili
 - ▶ Çağırana bir değer veya sonuç verir/geri döndürür
 - ▶ Matematiksel soyutlamalara daha yakın karşılık gelecek şekilde yazılabilir
- **Özyineleme(Recursion):** soyutlama mekanizmasından daha fazla yararlanan bir mekanizma



Kontrol: Yapısal Soyutlamalar

Ada Fonksiyon Örneği

```
function gcd (u, v: in integer) return integer is
begin
    if v = 0 then
        return u;
    else
        return gcd(v, u mod v);
    end if;
end gcd;
```

Şekil: Ada dilinde obeb(gcd/greatest common divisor) fonksiyonu



Kontrol: Yapısal Soyutlamalar

- **Yüksek mertebeden fonksiyonlar(Higher-order functions):** diğer fonksiyonları argüman olarak alır ve değerler gibi fonksiyonları döndürebilir
- Örnek: `map` fonksiyonu
 - ▶ Bir fonksiyon ve koleksiyonu argüman olarak alır
 - ▶ Argüman fonksiyonunu, argüman koleksiyonundaki her öğeye uygular ve bir sonuç listesi döndürür

```
(map abs (list 33 -10 66 88 -4)) ; (33 10 66 88 4) döndürür
```



Kontrol: Birim Soyutlamaları

- **Birim(Unit)**: bir programın diğer bölümlerine mantıksal olarak ilgili hizmetler sağlayan bağımsız bir prosedürler koleksiyonu
 - ▶ Birim tarafından sağlanan hizmetlerin detaylarını bilmeye gerek kalmadan bir programın bir bütün olarak anlaşılmasını sağlar
- **İş parçacıkları(Threads)**: Java sistemi içinde ayrı ayrı yürütülen kontrol yolları
- **Süreçler(Processes)**: Java sistemi dışında çalışan diğer programlar
- **Görev(Task)**: Ada'da paralel yürütme mekanizması



Hesaplama Paradigmaları

- **Zorunlu(Imperative) dil:** üç özelliğe sahip bir dil
 - ▶ Emirlerin sıralı yürütülmesi
 - ▶ Bellek konumlarını temsil eden değişkenlerin kullanımı
 - ▶ Değişkenlerin değerlerini değiştirmek için atamanın kullanılması
- Programlama dilleri için bir paradigmayı (örüntü) temsil eder
- **von Neumann darboğazı(bottleneck):** bir programın bir dizi talimat olarak tanımlanması gerekliliği



Hesaplama Paradigmaları

- Fonksiyonel Paradigma:
 - ▶ Bir fonksiyonun Lambda hesabındaki(calculus) soyut gösterimine göre
- Mantıksal Paradigma:
 - ▶ Sembolik mantığa dayalı
- Hem fonksiyonel hem de mantıksal paradigmlar matematiksel temellere karşılık gelir
 - ▶ Bir programın doğru çalışıp çalışmayacağını belirlemeyi kolaylaştırır
- Nesne yönelimli paradigma:
 - ▶ Gerçek dünyadaki nesnelerin davranışlarını taklit edecek şekilde çalışan yeniden kullanılabilir kod



Dil Tanımı

- Resmi dil tanımı faydalar sağlar:
 - ▶ Programlar hakkında matematiksel mantık yürütmenize izin vermeye yardımcı olur
 - ▶ Makine veya uygulama bağımsızlığı için standardizasyonu teşvik eder
 - ▶ Program davranışını ve etkileşimini tanımlar
 - ▶ Bir dil tasarlandığında disiplini sağlar
- Dil tanımı genel olarak şunlara ayrılabilir:
 - ▶ **Syntax**: Sözdizimi veya yapı
 - ▶ **Semantics**: Anlambilim veya anlam



- **Dil sözdizimi(Language syntax):** doğal bir dilin gramerine benzer
- **Gramer:** dilin sözdiziminin biçimsel tanımı
- **Sözcük yapısı(Lexical structure):** dilin kelimelerinin yapısı
 - ▶ Doğal dillerde yazım yapmaya benzer
- **Semboller/Simgeler(Token):** dilin kelimeleri
 - ▶ Anahtar sözcükler, tanımlayıcılar, işlem sembolleri, özel noktalama işaretleri vb. içerir.



Dil Dözdizimi

Örnek: C dilinde `if` komutu

- Örnek: C dilinde `if` komutu

`if` komutu sözdizimi

Özellik: Bir if ifadesi, “if” kelimesinin ardından parantez içinde bir ifade, ardından bir ifade(statement), ardından “else” kelimesinden oluşan isteğe bağlı bir else ve başka bir ifadeden(statement) oluşur.

```
if (x<5)
    printf("x 5'ten küçüktür");
else
    printf("x 5'ten büyüktür")
```



Dil Semantiği

- **Semantik:** Bir dilin anlamı
 - ▶ Kodu çalıştırmanın etkilerini açıklar
 - ▶ Tüm bağlamlarda kapsamlı bir anlam tanımı sağlamak zordur
- Örnek: C dilinde if yapısı

if komutu anlamı

Bir if ifadesi, tüm yan etkileri de dahil olmak üzere bir aritmetik veya işaretçi türüne sahip olması gereken ifadesi tarafından çalıştırılır ve sonucu 0 olmazsa, ifadeyi izleyen ifade çalıştırılır. Eğer bir else kısmı varsa ve ifade 0 ise, “else” i izleyen ifade çalıştırılır.



- Semantiği tanımlamak için genel kabul görmüş resmi bir yöntem yok
- Birkaç notasyon sistemi geliştirilmiştir:
 - ▶ Operasyonel(Operational) semantik
 - ▶ Sözel(Denotational) semantik
 - ▶ Aksiyomatik(Axiomatic) semantik



- **Çevirmen(Translator):** diğer programları kabul eden ve bunları doğrudan çalıştıran veya yürütmeye uygun bir forma dönüştüren bir program
- İki ana çevirmen türü:
 - ▶ **Yorumlayıcı(Interpreter):** bir programı doğrudan yürütür
 - ▶ **Derleyici(Compiler):** yürütmeye uygun bir biçimde eşdeğer bir program üretir



Dil Çevirisi/Tercümesi

Derleme

- Derleme en az iki adım gerektirir
 - ▶ **Kaynak program(Source program)** derleyiciye girdidir
 - ▶ **Hedef program(Target program)** derleyiciden çıktıdır
- Hedef dil genellikle assembly dilidir, bu nedenle hedef program şöyle olmalıdır:
 - ▶ Bir **derleyici(assembler)** tarafından bir nesne(object) programına çevrilmeli
 - ▶ Diğer nesne programlarıyla **bağlantılı(linked)**
 - ▶ Uygun bellek konumlarına **yüklenmeli(loaded)**



Dil Çevirisi/Tercümesi

Derleme

- Hedef dil, bayt kodu olabilir (bir tür düşük seviyeli kod)
 - ▶ Bayt kodu daha sonra sanal makine adı verilen bir yorumlayıcı tarafından yürütülür
- **Sanal makine(Virtual machine)**: farklı donanım mimarileri için farklı yazılmıştır
 - ▶ Bayt kodu makineden bağımsızdır
 - ▶ Örnekler: Java, Python



Dil Çevirisi/Tercümesi

Derleme

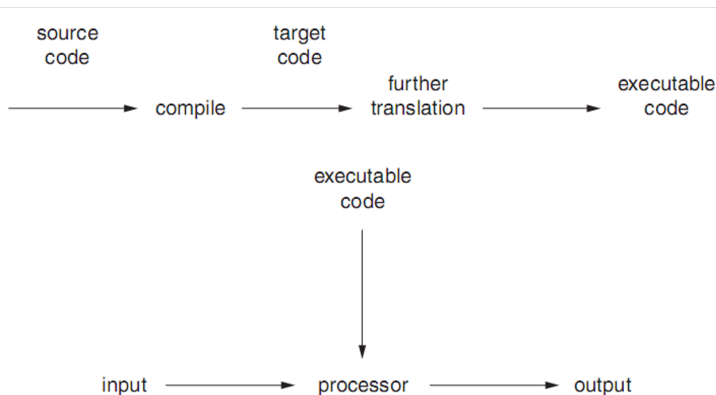


Figure 1.9 The compilation process



- Bir dilin, belirli bir yorumlayıcı veya derleyicinin davranışıyla tanımlanması mümkün
 - ▶ **Tanımlayıcı çevirmen(definitional translator)** olarak adlandırılır
 - ▶ Yaygın değil



Programlama Dillerinin Geleceği

- 1960'larda, bilgisayar bilimcileri tüm ihtiyaçları karşılayacak tek bir evrensel programlama dili istediler
- 1970'lerin sonunda ve 1980'lerin başında, kullanıcıların ihtiyaçları tanımlamasına ve ardından sistemi oluşturmaya olanak tanıyan spesifikasyon dilleri istediler.
 - ▶ Mantıksal programlama dillerinin yapmaya çalıştığı şey budur
- Programlama modası geçmiş değil
 - ▶ Ortaya çıkan yeni teknolojileri desteklemek için yeni diller ortaya çıkacak



Programlama Dillerinin Geleceği

- Haber gruplarındaki gönderi sayısına göre, 2000'den beri programlama dillerinin göreceli popülerliği:



















Mar 2009 (100d)	Feb 2003 (133 d)	Jan 2000 (365d)
news.tuwien.ac.at	news.individual.net	tele.dk
posts language	posts language	posts language
1 14110 python	59814 java	229034 java
2 13268 c	44242 c++	114769 basic
3 9554 c++	27054 c	113001 perl
4 9057 ruby	24438 python	102261 c++
5 9054 java	23590 perl	79139 javascript
6 5981 lisp	18993 javascript	70135 c

Figure 1.10 Popularity of programming languages (source: www.complang.tuwien.ac.at/anton/comp.lang-statistics/)



Programlama Dillerinin Geleceği

Tiobe Index(Şubat 2022)

Feb 2022	Feb 2021	Change	Programming Language	Ratings	Change
1	3	▲	 Python	15.33%	+4.47%
2	1	▼	 C	14.08%	-2.26%
3	2	▼	 Java	12.13%	+0.84%
4	4		 C++	8.01%	+1.13%
5	5		 C#	5.37%	+0.93%
6	6		 Visual Basic	5.23%	+0.90%
7	7		 JavaScript	1.83%	-0.45%
8	8		 PHP	1.79%	+0.04%
9	10	▲	 Assembly language	1.60%	-0.06%
10	9	▼	 SQL	1.55%	-0.18%
11	13	▲	 Go	1.23%	-0.05%
12	15	▲	 Swift	1.18%	+0.04%
13	11	▼	 R	1.11%	-0.45%
14	16	▲	 MATLAB	1.03%	-0.03%
15	17	▲	 Delphi/Object Pascal	0.90%	-0.12%
16	14	▼	 Ruby	0.89%	-0.35%
17	18	▲	 Classic Visual Basic	0.83%	-0.18%
18	20	▲	 Objective-C	0.81%	-0.08%
19	19		 Perl	0.79%	-0.13%
20	12	▼	 Groovy	0.74%	-0.76%