

YAZILIM MÜHENDİSLİĞİNE GİRİŞ

HAFTA 7 PROGRAMLAMA DİLLERİ

Dr. Öğr. Üyesi Ebru Aydoğan
eaydogan@mehmetakif.edu.tr

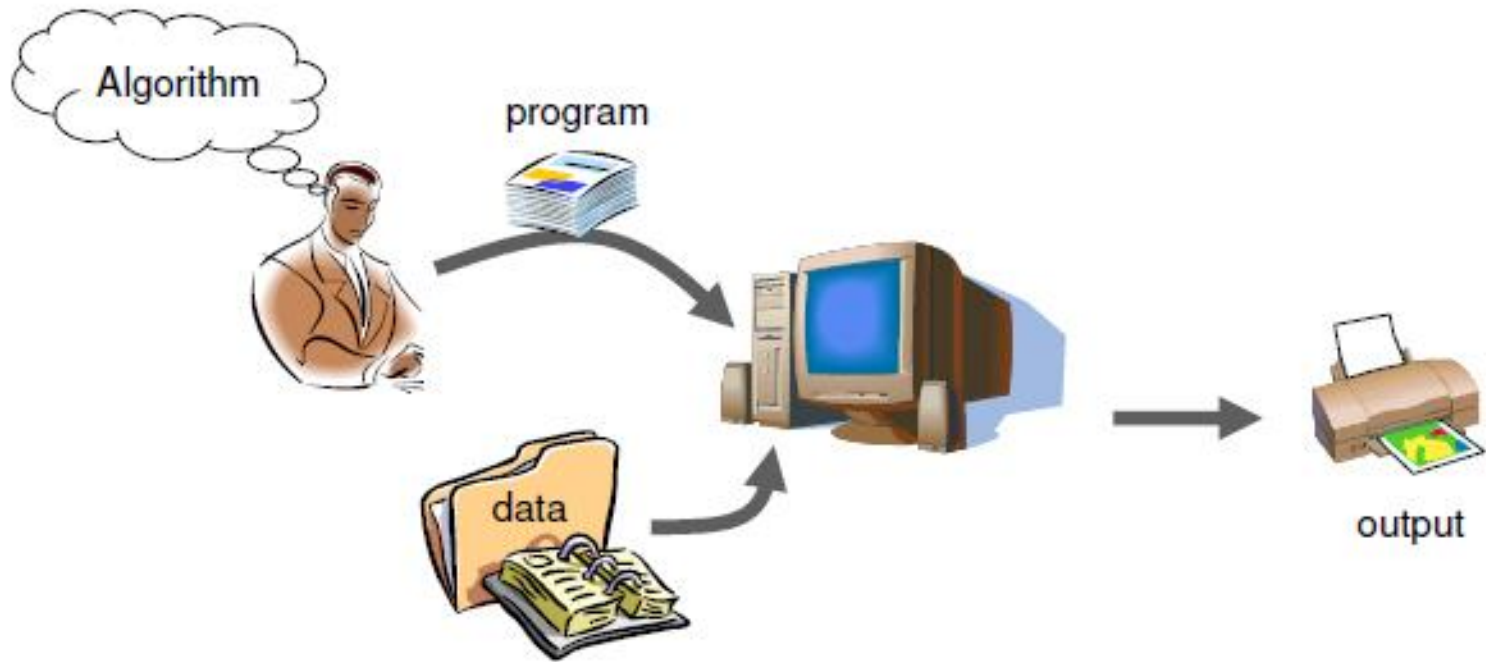
Ders İçeriği: Hafta 7

- Giriş
- Tarihsel Bakış
- Geleneksel Programlama Kavramları
- Yordamsal Birimler
- Dil Uygulaması
- Nesne Yönelimli Programlama
- Eş Zamanlı Eylemleri Programlama

Giriş

➤ Programlama Dili

- Algoritmayı bilgisayara tam olarak tanımlayabilen ve böylece algoritmanın çalıştırılabilmesini sağlayan bir dildir.



Giriş

- Eğer makine dili kullanmak zorunda olsaydık, bugün kullanılan bir çok yazılımı programlamak imkansız olacaktı.
- Programlama dilleri sayesinde, algoritmaların ifade edilebilmesi sağlanmıştır ve insanlar için daha kolay kabul edilebilen yapılar geliştirilmiştir.
- Programlama dilleri makine dili komutlarına çevrilerek bilgisayar tarafından kullanılır.

Giriş

- Bir programlama dilini tasarlarken iki uç nokta vardır:
 - İnsan dili kullanılabilir
 - Kesin tanımlı değil
 - Verimsiz (hem bilgisayar hem de insan için)
 - Kullanımı kolay
 - Hata ayıklamak zor
 - Makine kodu kullanılabilir
 - Hassas
 - Bilgisayarlar için verimli
 - Kullanımı ayrıntılı
 - Hata ayıklamak zor

Tarihsel Bakış

➤ İlk Nesiller

- Makine dili
 - Makine dilinde program yazmak, iş bitmeden önce bulunması ve düzeltilmesi (hata ayıklama olarak bilinir) gereken hatalara yol açtığından can sıkıcı bir iştir.
- Araştırmacılar sayısal form yerine daha akılda kalıcı olan sembolik bir formda kodları temsil edip işaretleme sistemleri geliştirerek programlama işlemini basitleştirdiler.

Tarihsel Bakış

➤ İlk Nesiller

- İkinci Nesil: Assembly Dili
- Makine kodlarının hatırlanması çok zor olduğundan, her işlemci üreticisi, işlem kodları için "hatırlanması kolay" bir ad tasarlar
 - Op-code için sembolik adlar
 - Yazmaçların her biri için isimlendirme
 - Program değişkenleri veya tanımlayıcılar: Programcı tarafından seçilen, hafıza konumları için tanımlayıcı isimlerdir.

Tarihsel Bakış

➤ İlk Nesiller

■ Assembly Dil Özellikleri

- Makine talimatları ile assembly talimatları arasında bire bir ilişki
- Programcı makine gibi düşünmeli
- Doğal olarak makineye bağlıdır
- Assembler adı verilen bir program tarafından makine diline dönüştürülür

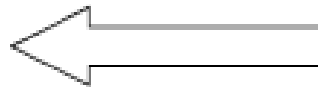
Tarihsel Bakış

➤ İlk Nesiller

Machine language program

```
156C
166D
5056
30CE
C000
```

assembler



Definition of mnemonics:

LD	means "load"
ADDI	means "Integer addition"
ST	means "store"
HLT	means "halt"
ORG	means "origin"
db	means "define byte"

Assembly language program

```
LD    R5, [Price]
LD    R6, [ShippingCharge]
ADDI  R0, R5 R6
ST    R0, [TotalCost]
HLT

;
ORG   6Ch
Price                db 25
ShippingCharge       db 5
TotalCost            db 00
```

Tarihsel Bakış

➤ İlk Nesiller

- Üçüncü Nesil: Üst düzey primitiveler kullanır
- Sözde kodlara benzer yapıdadır
- Makine bağımsızdır
- Örnekler: FORTRAN, COBOL
- Her primitive, bir dizi makine dili talimatına karşılık gelir.
- Compiler/derleyici adı verilen bir program tarafından makine diline dönüştürülür

Tarihsel Bakış

➤ Dil Çeviriciler (Dönüştürücü)

- Programlama dillerini birbirine dönüştüren farklı çeviriciler vardır:
 - Assembler: Assembly kodunu makine koduna bire bir eşleme sağlar.
 - Derleyici (Compiler): Yüksek seviyeli ifadeleri eşdeğer kısa makine kodlarına çevirir.
 - Yorumlayıcı (Interpreter): Yüksek seviyeli ifadelerin makine diline çevrimini ve yürütülmesini eş zamanlı gerçekleştiren dönüştürücülerdir.

➤ Biçimsel Diller

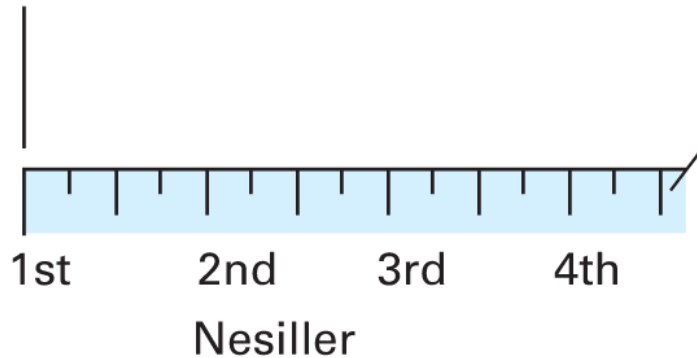
- Programlama dilleri, gramerlerle tam olarak tanımlanan yapay diller oldukları için “biçimsel dillerdir”
- Doğal (insan) dilleri gramerlerle kesin olarak tanımlanmaz; bunun yerine gramerler, dil kullanımını “özetlemek” için sonradan oluşturulur.
 - Esperanto, 1870'lerin sonlarında yapay olarak geliştirilen "resmi" bir insan dilidir.

Tarihsel Bakış

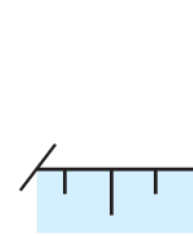
➤ Programlama Paradigmaları

- Programlama dillerinin nesilleri

Problemler, insanların makine özelliklerine uymak zorunda oldukları ortamlarda çözüldü



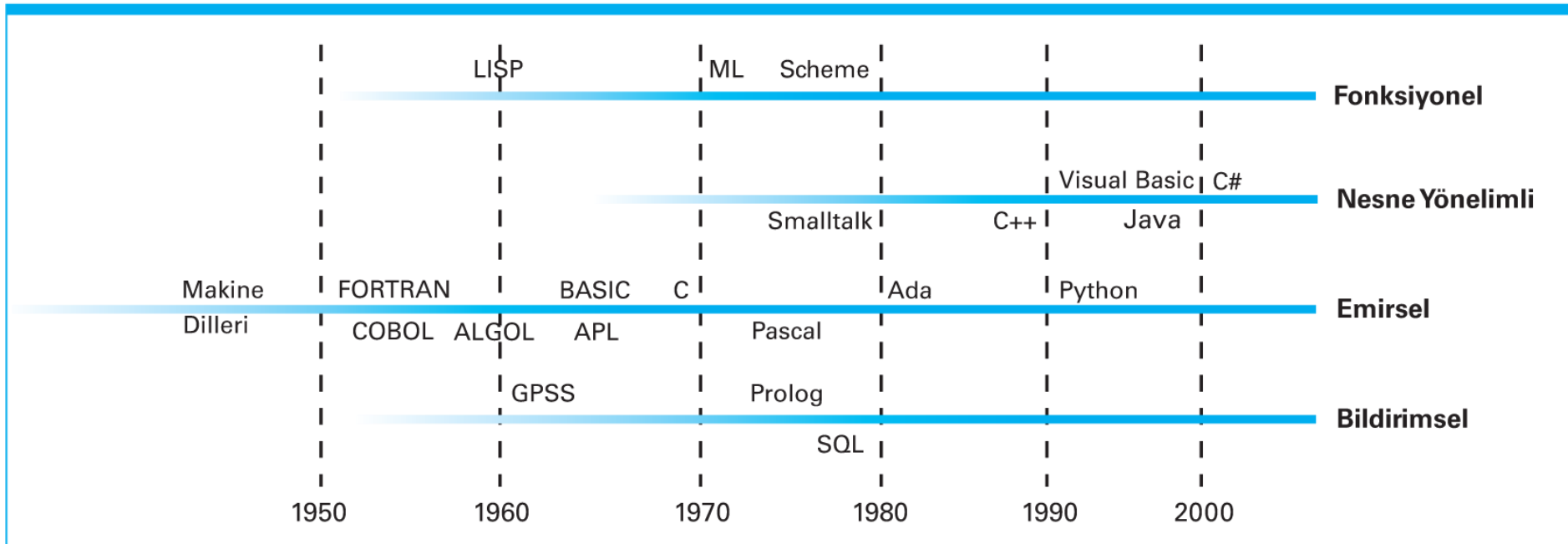
Problemler, makinelerin insan özelliklerine uymak zorunda oldukları ortamlarda çözüldü



Tarihsel Bakış

➤ Programlama Paradigmaları

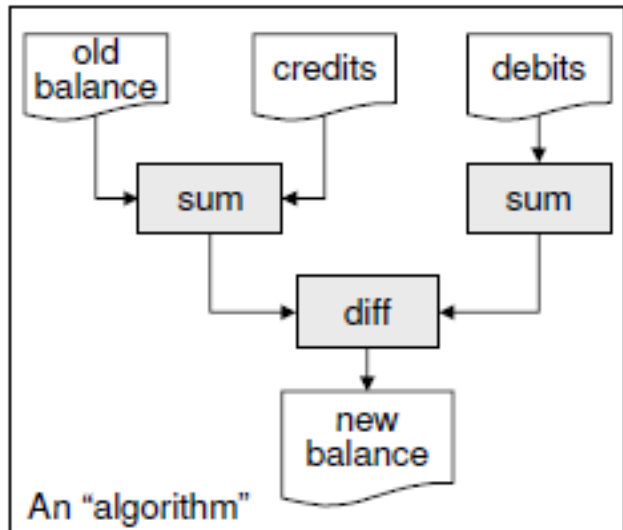
■ Programlama paradigmlarının evrimi



Tarihsel Bakış

➤ Programlama Paradigmaları

- Emirsel (Imperative) programlama dili
 - Bir program bir dizi komuttan oluşur
 - Programlamanın en eski yolu
- Fonksiyonel (Functional) programlama dili
 - Program, bir veri akışının (işlevsel birimlerin bağlantıları) açıklamasıdır



A program in LISP programming language:

```
(diff (sum old_balance credits) (sum debits))
```

Tarihsel Bakış

➤ Programlama Paradigmaları

- Bildirimsel (Declarative) programlama dili
 - Amaçlanan çözümü karşılayan koşulları açıklar; çözüme ulaşmak için gereken adımlar belirtilmemiş bir yorumlayıcıya bağlıdır
 - Yalnızca belirli bir sorun alanı için çalışır (örneğin, bilgiye dayalı çıkarım için)
- Nesne yönelimli (Object-oriented) programlama dili
 - “Veri merkezli” bir programlama dili
 - İşlemler verilere eklidir
 - Bir program, her biri o nesnenin izin verilen işlemlerinin bir listesiyle açıklanan nesnelerin bir listesinden oluşur.

Geleneksel Programlama Kavramları

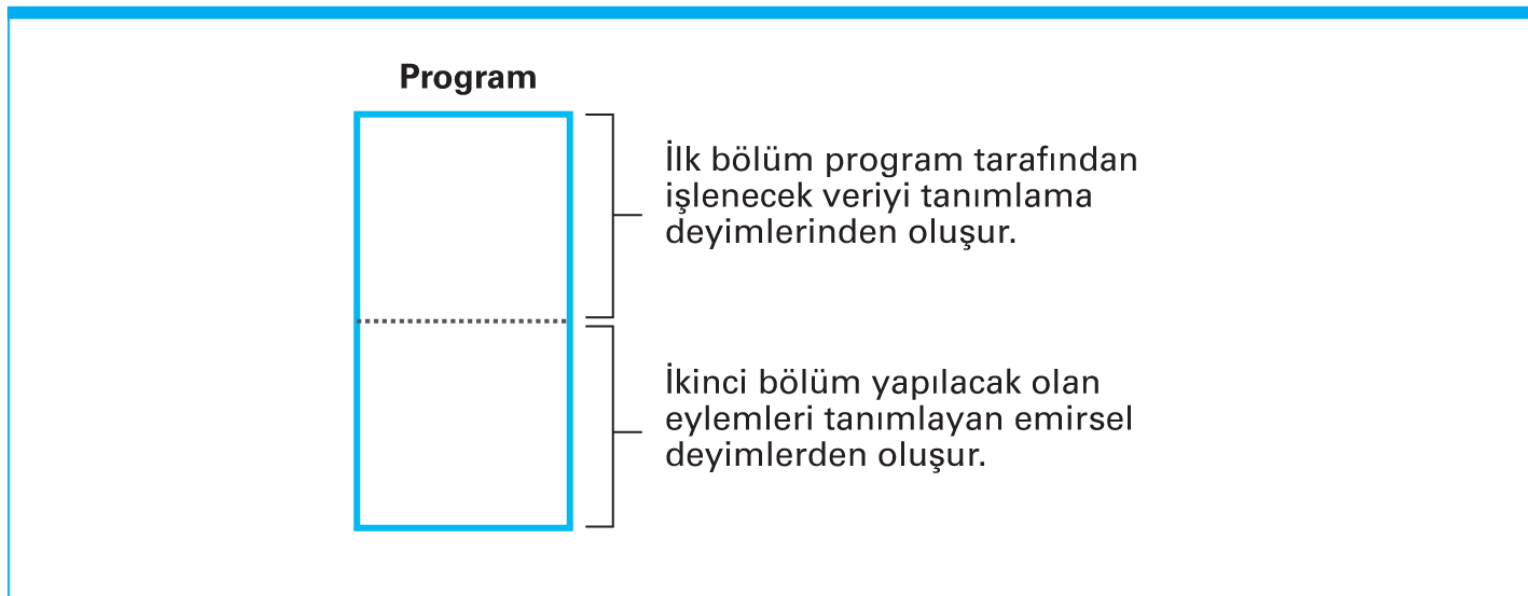
- Genelde emirsel veya nesne yönelimli bir program üç kategorideki deyimler topluluğundan oluşur.
 - Bildirimsel deyimler: Programda daha sonra kullanılacak olan özelleştirilmiş terminolojiyi tanımlar. (veri ögesi adlandırma vb.)
 - Emirsel deyimler: Arka plandaki algoritmanın adımlarını tanımlar.
 - Yorumlar: Programın gizli özelliklerini kullanıcıya açıklar.

Geleneksel Programlama Kavramları

- Amacımız belli bir dilin detaylarında takılıp kalmak değil, sadece yaygın dil özelliklerinin çeşitli gerçek dillerde nasıl belirtildiğini göstermektir.
- C, üçüncü nesil emirsel bir dildir.
- C++, C dilinin genişletilmiş hali olan bir nesne-yönelimli dildir.
- Java ve C#, C++'dan türetilen nesne-yönelimli dillerdir.

Geleneksel Programlama Kavramları

- Emirsel programlama, komutları bilgisayarlara ifade etmenin en sezgisel ve etkili yoludur.
- Tipik bir emirsel program biriminin yapısı



Geleneksel Programlama Kavramları

➤ Geleneksel Programlamanın Elemanları

- Geleneksel bir programlama dili aşağıdakilere yönelik deyimler sağlar:
 - Sabitler ve değişkenlerin ifadesi
 - Değişkenlere değer atama
 - Programın yürütme sırasını kontrol etme
 - Koşullu kontrol
 - Döngü kontrolü
 - Programı yorumlama
 - Fonksiyon birimlerinin çağırılması

Geleneksel Programlama Kavramları

➤ Değişkenler ve Veri Tipleri

- Değişken: Yüksek seviye programlama dillerinde ana bellekteki konumların sayısal adreslerinden ziyade tanımlayıcı isim olarak kullanılırlar.
- Veri Türü: Değişkenle ilişkili hafıza bölgesinde saklanacak olan verinin tipini tanımlar.
 - Integer: Tam sayılar
 - Real (float): Kesirli sayılar
 - Character: Semboller
 - Boolean: True/false (lojik)

Geleneksel Programlama Kavramları

➤ Değişkenler ve Veri Tipleri

- C, C++, C# ve Java için örnek değişken tanımlamaları:

```
float Length, Width;  
int Price, Total, Tax;  
char Symbol;  
int WeightLimit = 100;
```

Geleneksel Programlama Kavramları

➤ Veri Yapısı

- Veri tipine ek olarak, programdaki değişkenler çoğu zaman kavramsal bir yapı ya da veri topluluğu olan veri yapısı (data structure) ile ilişkilendirilebilir.
 - Dizi (array), Matris
 - Struct (birleşik tür yapı, kayıt)

```
int Scores[2][9];
```

⇒ array

```
Struct {  
    char Name[8];  
    int Age;  
    float SkillRating;  
} Employee;
```

⇒ structure (heterogeneous array)

Geleneksel Programlama Kavramları

➤ Veri Yapısı

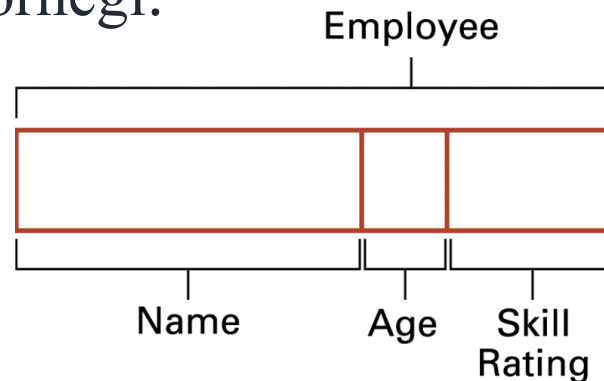
- İki satır ve dokuz sütunlu bir dizi örneği:

Scores

Scores (2, 4) in
FORTRAN where
indices start at one.

Scores [1] [3] in C
and its derivatives
where indices start
at zero.

- Yapı (struct) örneği:



Geleneksel Programlama Kavramları

➤ Atama Deyimleri

- En basit emirsel deyim bir değerin bir değişkene atanmasını (ya da daha açıkça, değişken ile belirtilen hafıza alanında depolanmasını) talep eden bir atama deyimidir.

- Örneğin,

Z = **X** + **y**;

deyimi, C, C++, C# ve Java'da X ve Y'nin toplamının Z değişkenine atanmasını ifade eder.

Geleneksel Programlama Kavramları

➤ Kontrol Deyimleri

- Programın çalışma gidişatını değiştiren deyimlerdir.
 - En basit olanı goto,
 - if-else ve do-while koşullu dallanmalar,
 - switch-case dallanma yapısı
 - for döngüsü

Geleneksel Programlama Kavramları

➤ Yorum Satırları (Comments)

- Bir program içindeki açıklayıcı ifadelerdir
- İnsanlar içindir, programcı programı okuduğunda neler olup bittiğini hızlıca kavrar
- Derleyici tarafından yok sayılır

Yordamsal Birimler

➤ Fonksiyonlar

- Diğer program birimleri tarafından şablon olarak kullanılabilecek bir işi yapmak için oluşturulmuş komutlar dizisine **fonksiyon** denir.
- Temel kavramlar için birçok terim kullanılmıştır:
 - Alt program, alt rutin, yordam, metot, fonksiyon...
- Birim, fonksiyonun başlığı ile başlar
- Yerel vs. Global değişkenler
- Biçimsel (formal) vs. Gerçek (actual) Parametreler
- Referans ile aktarma (passing by reference) vs. Değer ile aktarma (passing by value)

Yordamsal Birimler

➤ C dili ile örnek bir fonksiyon tanımlama

Starting the header with the term “void” is the way that a C programmer specifies that the program unit returns no value. We will learn about return values shortly.

The formal parameter list. Note that C, as with many programming languages, requires that the data type of each parameter be specified.

```
void ProjectPopulation (float GrowthRate)
```

```
{    int Year;           // This declares a local variable named Year.
```

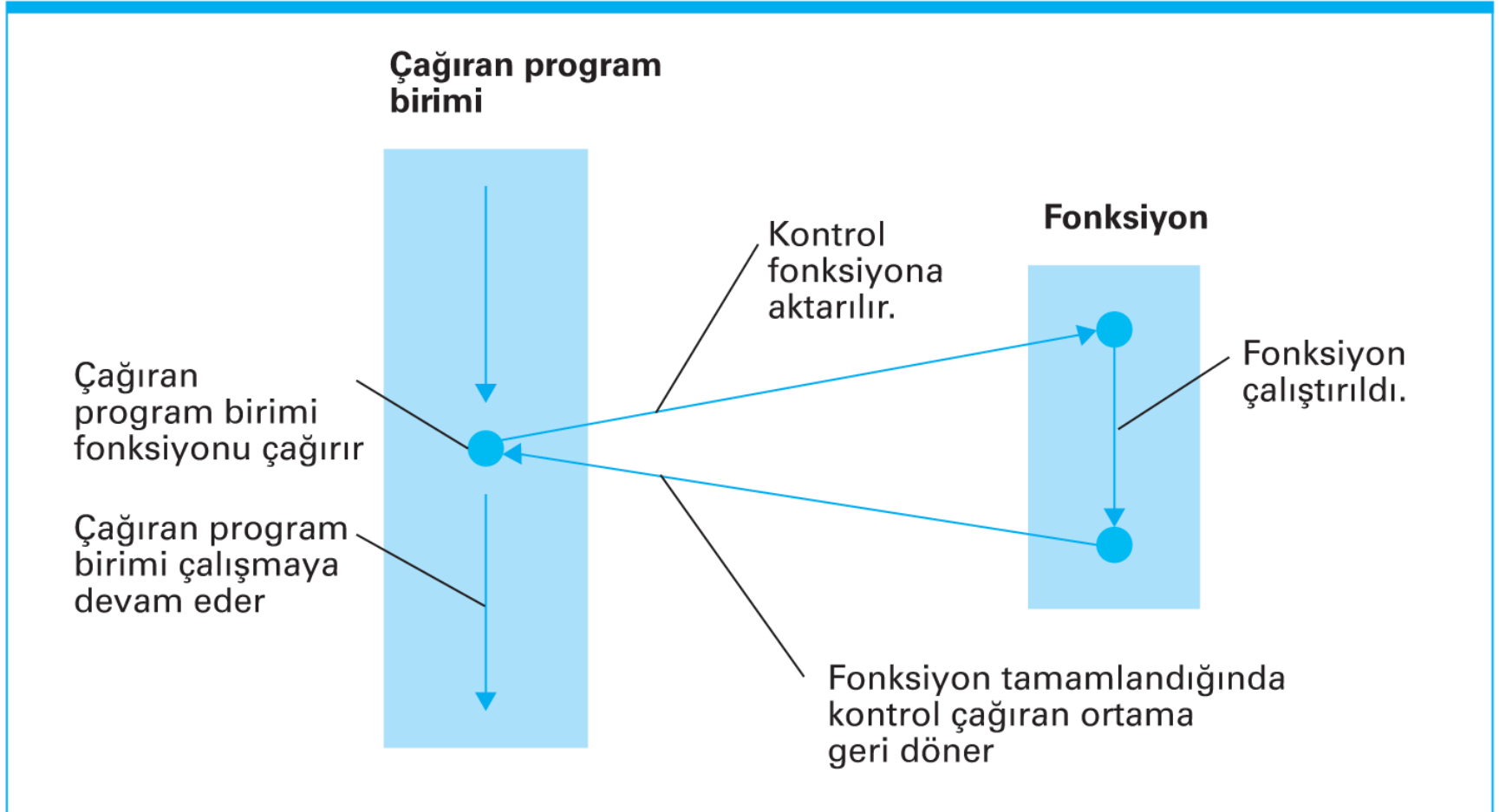
```
    Population[0] = 100.0;
    for (Year = 0; Year <= 10; Year++)
        Population[Year+1] = Population[Year] + (Population[Year] * GrowthRate);
```

```
}
```

These statements describe how the populations are to be computed and stored in the global array named Population.

Yordamsal Birimler

➤ Bir fonksiyon içeren program akışı



Yordamsal Birimler

➤ Parametreler

- Bir fonksiyonun üzerinde çalışacağı terimler, parametre olarak adlandırılır.
- Çağıran program biriminden çağrılan fonksiyona parametre aktarmanın birkaç yolu vardır:
 - Değere göre çağrı (passing by value)
 - Referansa göre çağrı (passing by reference)

Yordamsal Birimler

- Demo1 ve Demo2 isimli örnek fonksiyonlar C# ile yazılmış olsun:

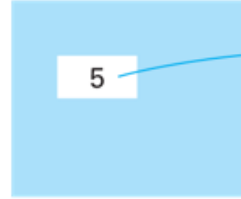
```
int Demo1(int Formal){  
    Formal=Formal+1;  
    return Formal;  
}  
  
void Demo2(int Formal) {  
    Formal=Formal+1;  
}  
  
public static void main(String[] args) {  
    int Actual = 5;  
    int sonuc = Demo1(Actual);  
    Demo2(&Actual);  
}
```

Yordamsal Birimler

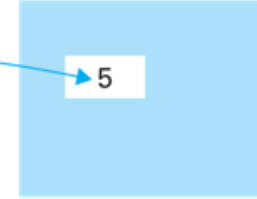
- Demo1
fonksiyonunun
çalıştırılması ve
parametreleri
değer ile aktarma

a. Fonksiyon çağırıldığı zaman verinin bir kopyası
fonksiyona verilir

Çağırıcı ortam



Fonksiyon ortamı



b. ve fonksiyon kopyayı işler.

Çağırıcı ortam



Fonksiyon ortamı



c. Böylece fonksiyon sonlandığında çağırıcı
ortam değişmemiş olur.

Çağırıcı ortam



Yordamsal Birimler

- Demo2
fonksiyonunun
çalıştırılması ve
parametreleri
referans ile
aktarma

- a. Fonksiyon çağırıldığında, resmi parametre gerçek parametreye yapılan bir referans haline gelir.



- b. Böylece, fonksiyon tarafından yönetilen değişiklikler gerçek parametreye yapılır.



- c. ve bu nedenle fonksiyon tamamlandığında korunmuş olur.



Dil Uygulaması

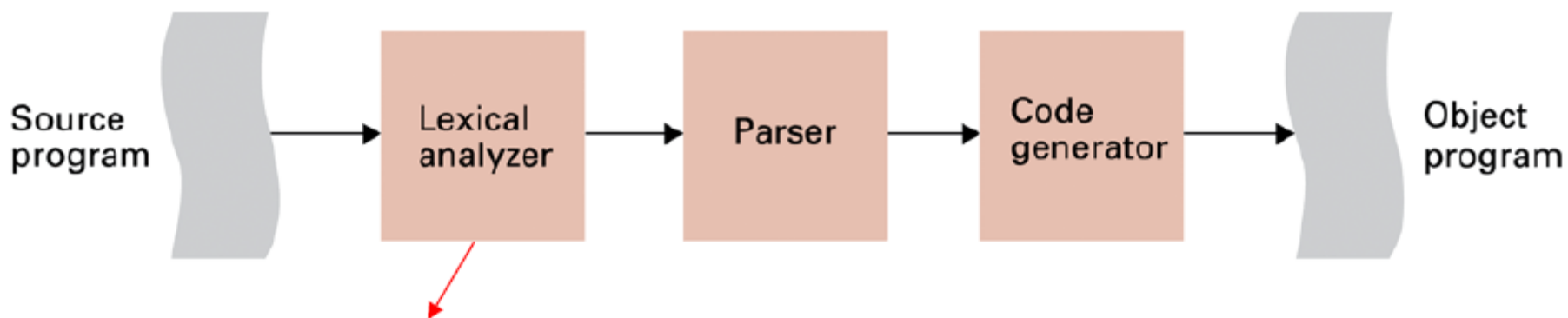
➤ Çeviri süreci

- Üst düzey bir dilde yazılmış bir programı makine tarafından yürütülebilir bir forma dönüştürme sürecidir.
- Orijinal yapıdaki programa kaynak program (source program), dönüştürülen programa ise amaç program (object program) denir.
- Dönüşüm süreci üç elemanın eylemlerinden oluşur:
 - Sözdizim analizcisi veya sözlüksel analiz (Lexical Analyzer), hangi sembol dizilerinin tek bir varlığı veya belirteci temsil ettiğini tanır.
 - Ayırıştırıcı (Parser), ayrıştırma ağaçları oluşturmak için sözdizimi diyagramlarını kullanarak belirteçleri ifadeler halinde gruplandırır.
 - Kod üretici (Code Generator), ifadeleri uygulamak için makine dilinde talimatlar oluşturur.

Dil Uygulaması

➤ Çeviri süreci

▪ Sözdizim Analizcisi (Lexical Analyzer)



Lexical analyzer converts alpha-numerical symbols in the source program to tokens; for example, if each token is specified by a 16-bit number, a lexical analyzer may perform the following conversion:

`position = x_coord + y_coord * 7` → 0003 1001 0001 1002 0002 1003 2001

The first byte specifies the type of token,
0 – variables, 1 – operators, 2 – constants

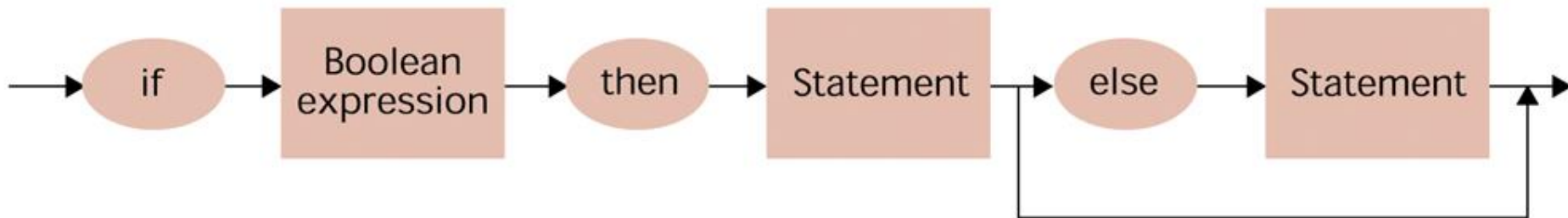
The remaining bytes compose an
index to the token value tables

Dil Uygulaması

➤ Çeviri süreci

▪ Ayrıştırıcı (Parser)

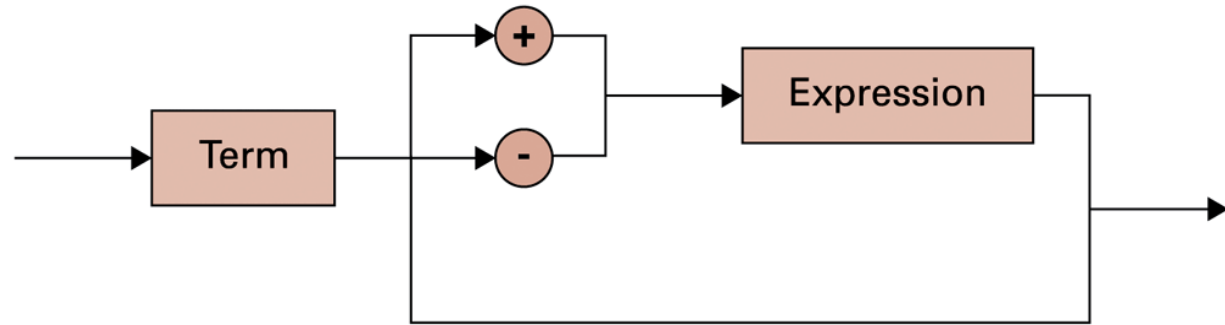
- Ayrıştırma işlemi, programlama dilinin sözdizimini tanımlayan bir dizi kurala dayanır.
 - Kurallara gramer denir
 - Kurallar sözdizimi diyagramlarıyla ifade edilebilir
- “if-then-else” ifadesinin sözdizimi diyagramı aşağıdaki gibidir:



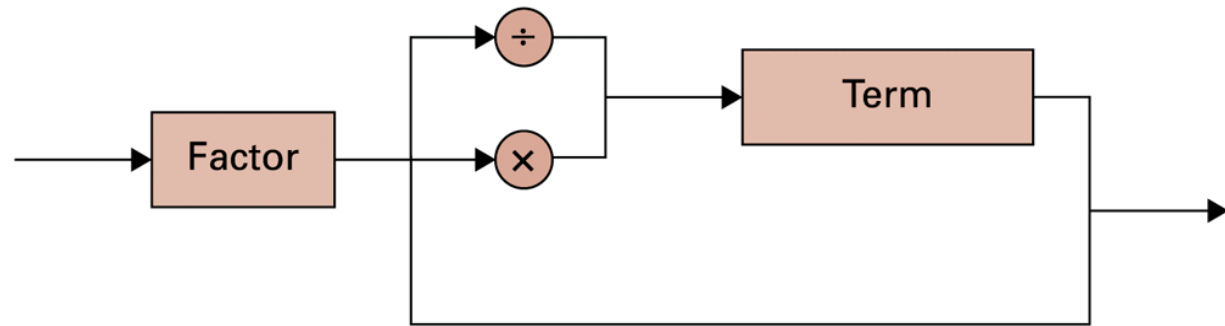
Dil Uygulaması

➤ Basit cebirsel
bir ifadeyi
tanımlayan
sözdizimi
şeması

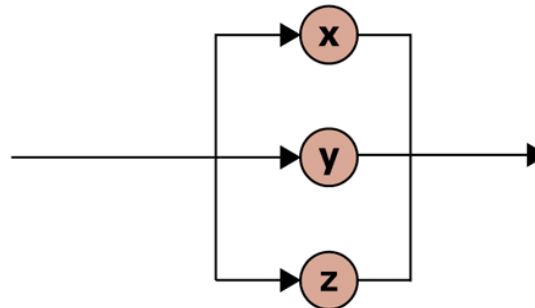
Expression



Term

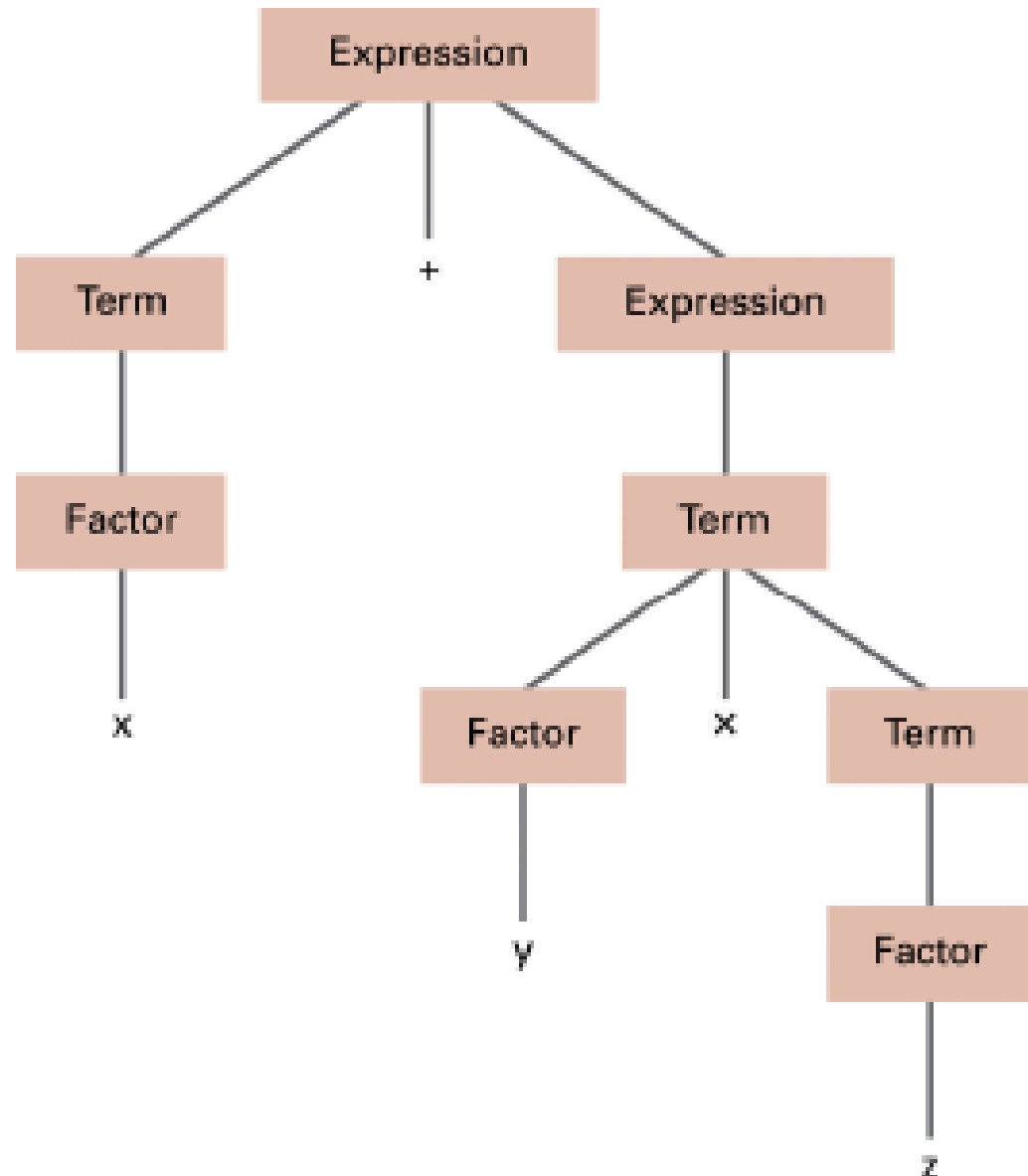


Factor



Dil Uygulaması

- Basit cebirsel bir ifadeyi tanımlayan sözdizimi şemasına göre $x + y * z$ için bir ayrıştırma ağacı



Dil Uygulaması

➤ Çeviri süreci

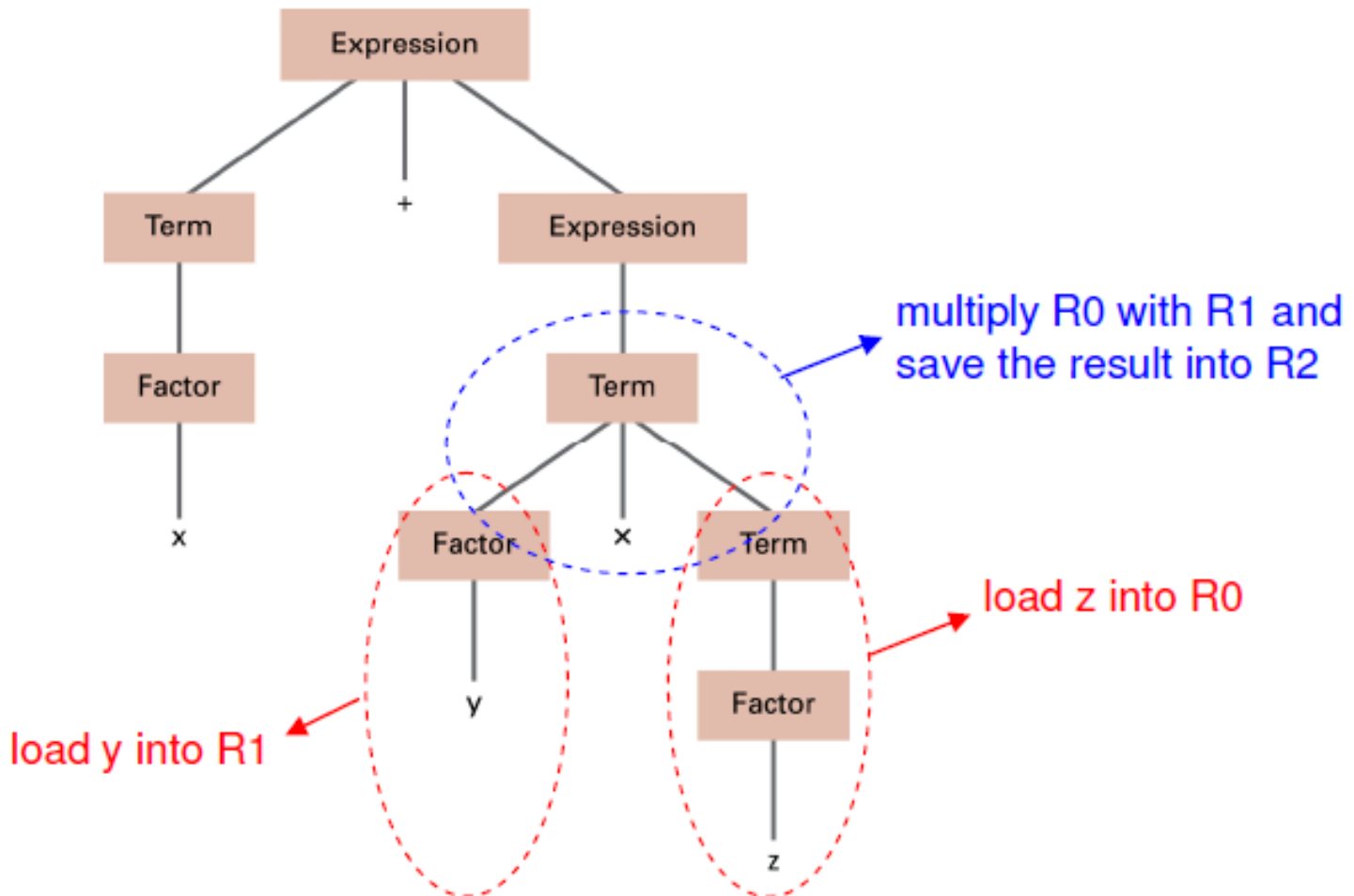
▪ Kod üretici (Code Generator) ve Optimizasyonu

- Ayrıştırma ağacı tamamlandıktan sonra, her alt ağaç veya düğüm için aşağıdan yukarıya makine kodları oluşturulmalıdır.
- Kod optimizasyonu, kod oluşturmanın en iyi yolunu bulmaya yönelik bir tekniktir

Dil Uygulaması

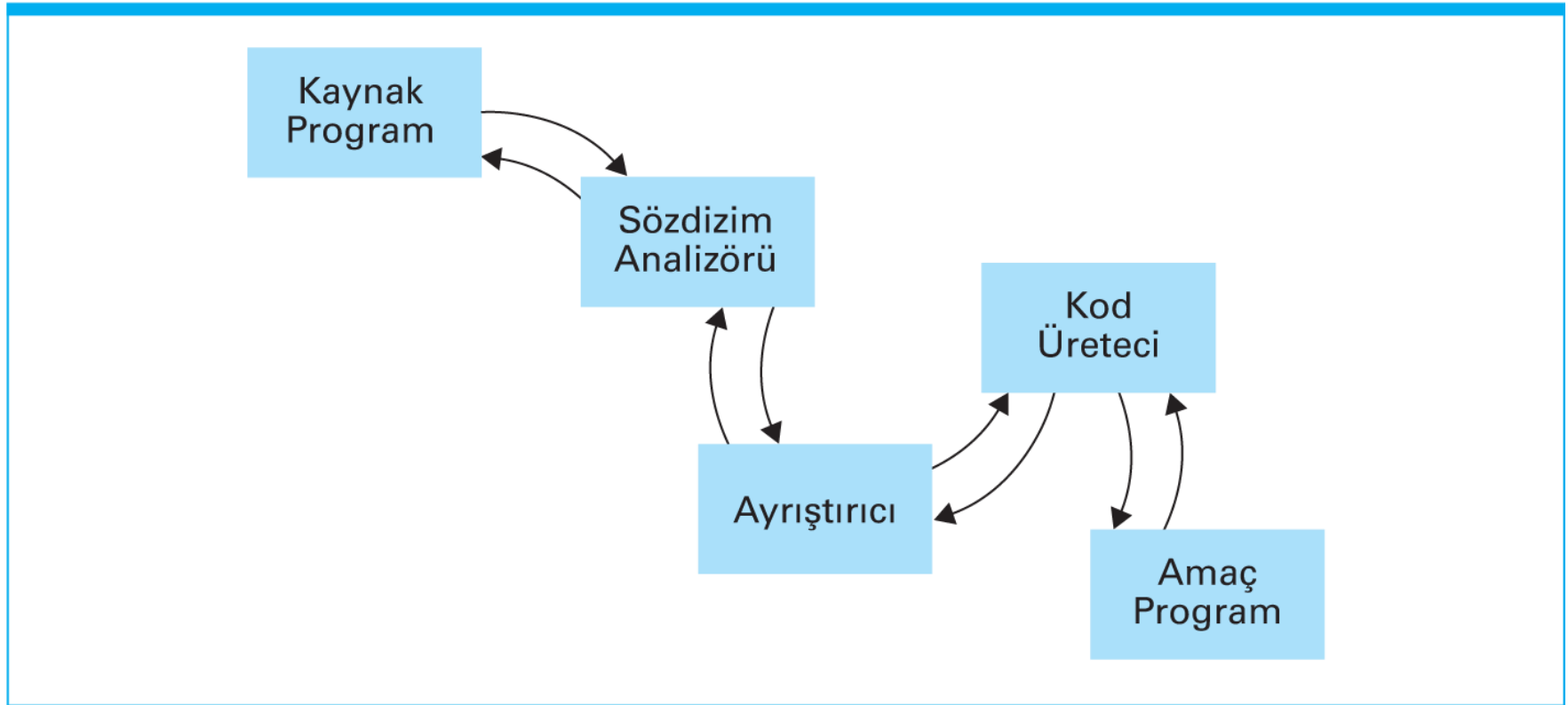
➤ Çeviri süreci

- Kod üretici (Code Generator) ve Optimizasyonu



Dil Uygulaması

- Çevrim işlemine nesne-yönelimli yaklaşım: Her bir nesnenin kendi işini yapması için etkileşimlidir.



Dil Uygulaması

➤ Yazılım Geliştirme Paketleri

- Yazılım geliştirme sürecinde kullanılan editör ve dönüştürücü gibi yazılım araçları tek bir entegre yazılım geliştirme sistemi olarak bir paket içinde gruplanmıştır.
- Programcı bu uygulama paketini kullanarak, program yazmak için bir editöre, makine diline kodları çeviren bir dönüştürücüye, arızalı programın çalışmasını takip ederek, programın nerede hataya yol açtığını bulmak için kullanılan çeşitli hata ayıklama araçlarına hazır erişim elde eder.

Nesne Yönelimli Programlama

➤ Sınıflar ve Nesneler

- Sınıf/Class: Nesne topluluğundan oluşan şablona denir.
- Nesne/Object: Hem verileri hem de çeşitli uyarılara nasıl tepki verileceğini tanımlayan fonksiyonlar içeren aktif program birimidir.
- Nesneye sınıfın bir örneği (instance) denir.

Nesne Yönelimli Programlama

➤ Sınıflar ve Nesneler

- Bir bilgisayar oyunundaki lazer silahını tanımlayan sınıf yapısı

```
class LaserClass
```

```
{ int RemainingPower = 100;
```

```
void turnRight ( )  
{ ... }
```

```
void turnLeft ( )  
{ ... }
```

```
void fire ( )  
{ ... }
```

```
}
```

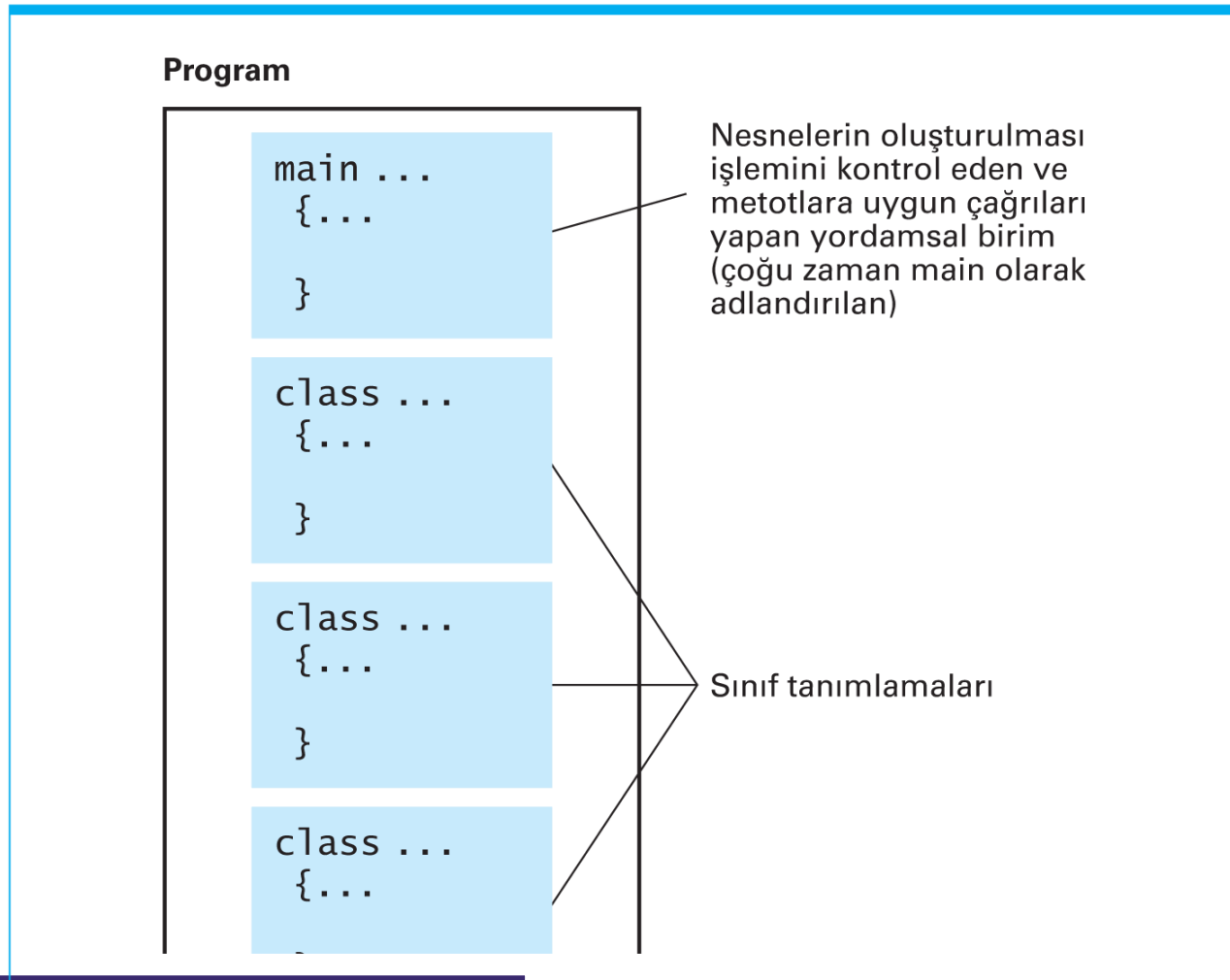
Bu türden olan her bir nesnenin içinde bulunan verinin tanımı

Bu türden bir nesnenin çeşitli mesajlara nasıl cevap vereceğini tanımlayan metotlar

Nesne Yönelimli Programlama

➤ Sınıflar ve Nesneler

- Tipik bir nesne-yönelimli program yapısı



Nesne Yönelimli Programlama

➤ Nesnenin Bileşenleri

- Instance Variable: Bir nesne içerisindeki değişkendir
 - Nesne içindeki bilgileri tutar
- Metot: Bir nesne içerisindeki prosedürdür
 - Nesnenin gerçekleştirebileceği eylemleri tanımlar
- Constructor/Yapıcı: Yeni bir nesne oluşturulduğu anda başlatmak için kullanılan özel metottur.

Nesne Yönelimli Programlama

➤ Nesnenin Bileşenleri

- Yapıcı fonksiyonlu bir sınıf

```
class LaserClass  
{ int RemainingPower;
```

Nesne oluşturulduğunda
yapıcı RemaininPower'a
bir değer atar

```
LaserClass (InitialPower)  
{ RemainingPower = InitialPower;  
}
```

```
void turnRight ( )  
{ ... }
```

```
void turnLeft ( )  
{ ... }
```

```
void fire ( )  
{ ... }
```

```
}
```

Nesne Yönelimli Programlama

➤ Kapsülleme, Kalıtım ve Çok Biçimlilik

- Kapsülleme (Encapsulation): Bir nesnenin dahili bileşenlerine erişimi kısıtlama yöntemidir.
 - Public ve Private erişim denetleyicileri kullanılır.
- Kalıtım (Inheritance): Benzer nesnelerin tanımlanmasında, bir sınıfın diğerinin özelliklerini kapsamasını ve kullanabilmesini sağlar.
- Çok Biçimlilik (Polymorphism): Yöntem çağrılarının çağrıyı alan nesne tarafından farklı şekillerde yorumlanabilmesidir.

Nesne Yönelimli Programlama

➤ Kapsülleme, Kalıtım ve Çok Biçimlilik

- Kapsülleme kullanan LaserClass sınıfı

Components in the class are designated public or private depending on whether they should be accessible from other program units.

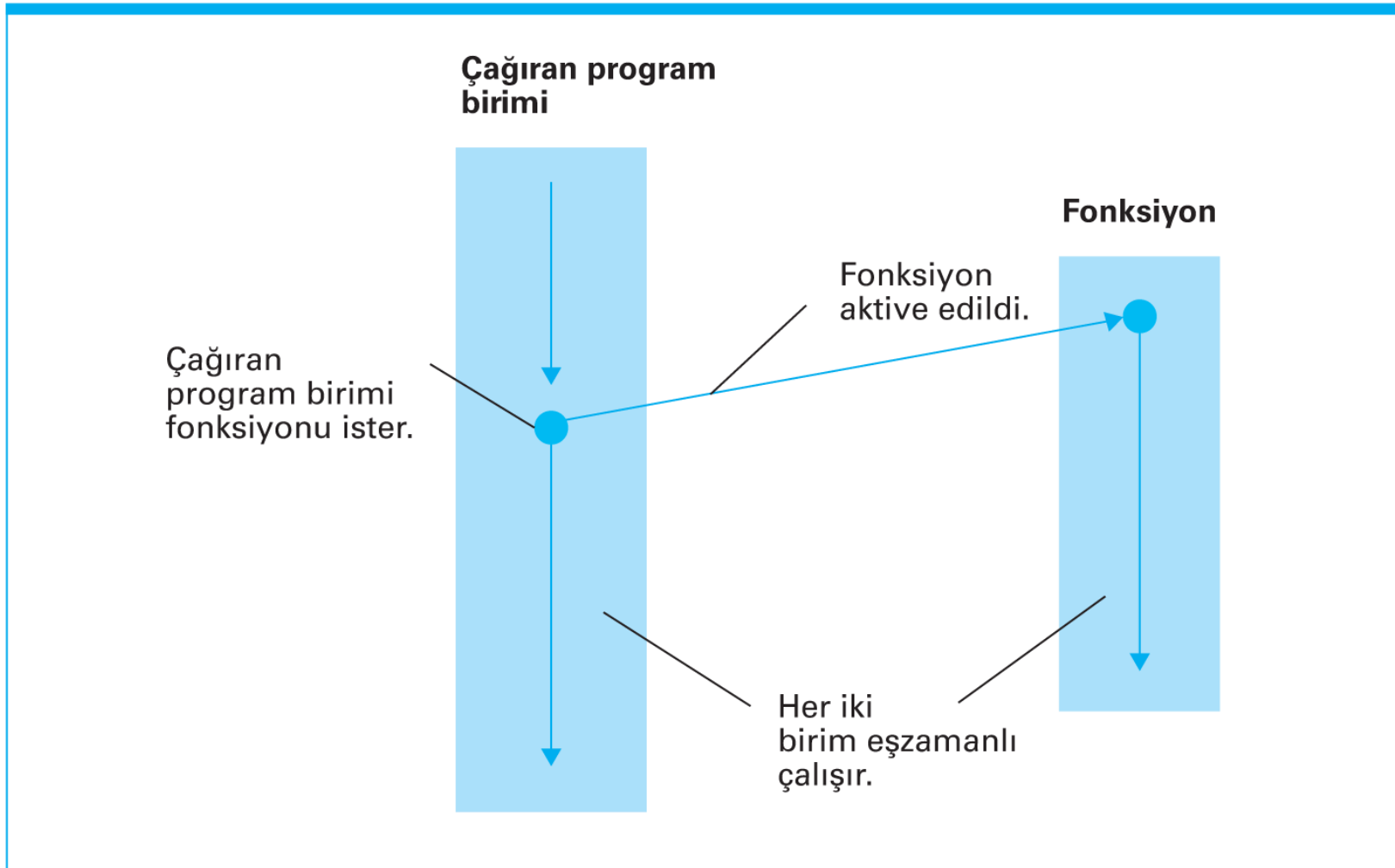
```
class LaserClass
{private int RemainingPower;
public LaserClass (InitialPower)
{RemainingPower = InitialPower;
}
public void turnRight ( )
{...}
public void turnLeft ( )
{...}
public void fire ( )
{...}
}
```

Eş Zamanlı Eylemleri Programlama

- Paralel (parallel or concurrent) işleme: Birçok prosesin eş zamanlı çalıştırılmasıdır.
- Gerçek eşzamanlı işleme, birden çok CPU gerektirir
- Tek bir CPU ile zaman paylaşımı kullanılarak simüle edilebilir

Eş Zamanlı Eylemleri Programlama

➤ İş parçacığı oluşturulması



Eş Zamanlı Eylemleri Programlama

➤ Veriye Erişimin Kontrolü

- Karşılıklı Dışlama (Mutual Exclusion): Verilere aynı anda yalnızca bir işlem tarafından erişilebilmesini garanti eden bir yöntemdir.
- Monitör: Kendine yapılan erişimleri kontrol etme yeteneği eklenmiş veri öğelerine monitör denir.