

# PROGRAMLAMA DİLLERİ

**Hafta 4**

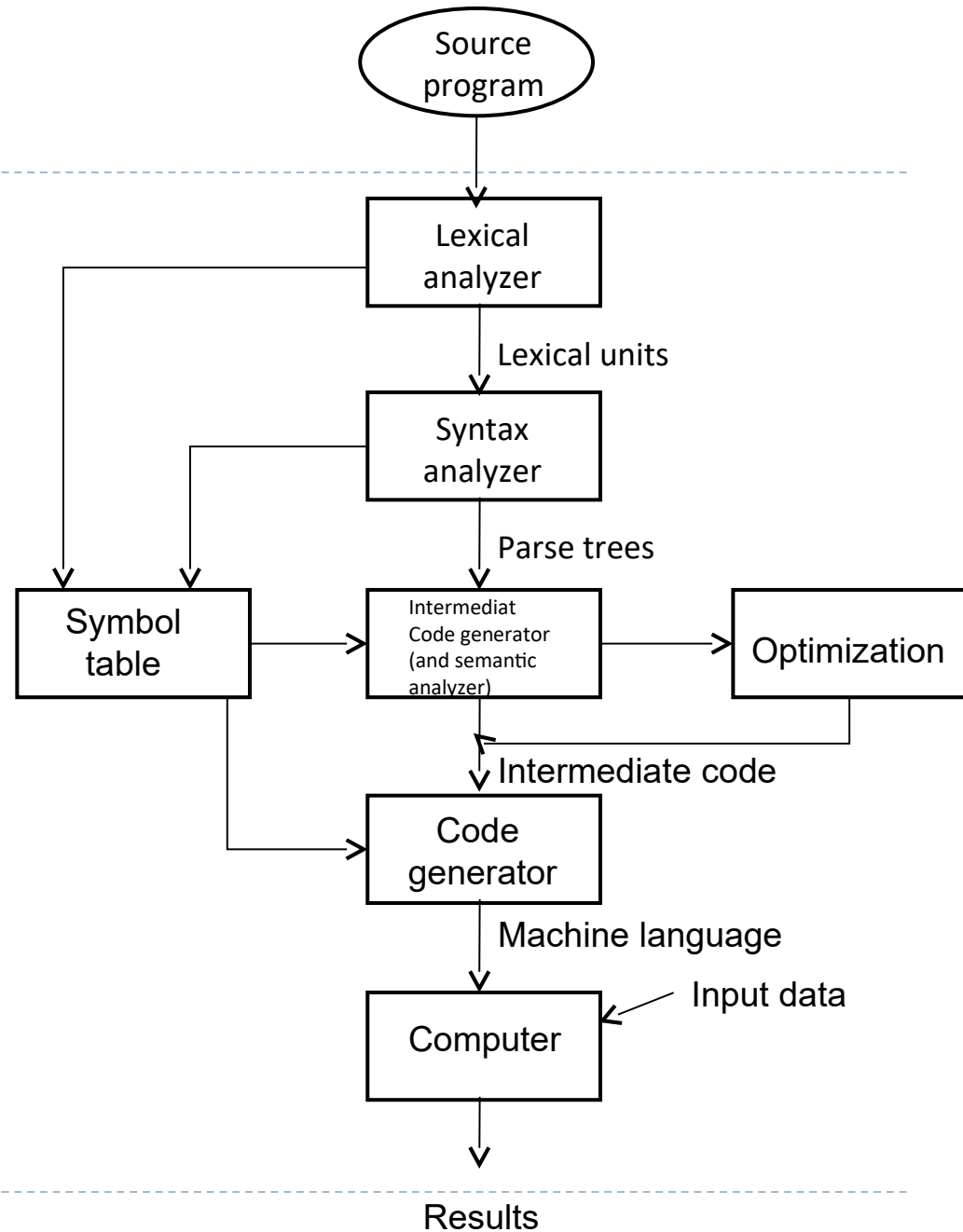
# Derleme işlemi

## ► 6 ana adım :

1. Lexical analyzer
2. Syntax analyzer
3. Semantic analyzer
4. Intermediate code generator
5. Code optimizer
6. Code generator

## ► İki ekstra adım :

- A. Symbol table manager
- B. Error handler



- 
- ▶ Her programlama dilindeki geçerli programları belirleyen bir dizi kural vardır. Bu kurallar **sentaks** (sözdizim, syntax) ve **semantik** (anlambilim, semantics) olarak ikiye ayrılır.
  - ▶ Her deyimin sonunda noktalı virgül bulunması sentaks kurallarına örnek oluştururken, bir değişkenin kullanılmadan önce tanımlanması bir semantik kuralı örneğidir.



# Sentaks (Sözdizimi) ve Semantik (Anlam)

---

- ▶ **Sentaks (Syntax):** İfadelerin (statements), deyimlerin (expressions), ve program birimlerinin biçimi veya yapısı
- ▶ **Semantik (Semantics):** Deyimlerin, ifadelerin, ve program birimlerinin anlamı
- ▶ Sentaks ve semantik bir dilin tanımını sağlar
  - ▶ Bir dil tanımının kullanıcıları
    - ▶ Diğer dil tasarımcıları
    - ▶ **Uygulamacılar (Implementers)**
    - ▶ Programcılar (Dilin kullanıcıları)



# Sentaks (Sözdizimi) ve Semantik (Anlam)

- Sentaks (Sözdizimi) ve Semantik (Anlam) bir dilin tanımı sağlar

Sözdizim (Syntax)	Anlam (Semantics)
Bir dilin sözdizim kuralları, bir deyimdeki her kelimenin nasıl yazılabileceğini belirler.	Bir dilin anlam kuralları ise, bir program çalıştırıldığında gerçekleşecek işlemleri tanımlar.



# Sentaks (Sözdizim) ve Semantik (Anlam)

- Sözdizim ve anlam arasındaki farkı, programlama dillerinden bağımsız olarak bir örnekle incelersek:
- Tarih gg.aa.yyyy şeklinde gösteriliyor olsun.

Sözdizim	Anlam	
10.06.2007	10 Haziran 2007	Türkiye
	6 Ekim 2007	ABD

- Ayrıca sözdizimindeki küçük farklar anlamda büyük farklılıklara neden olabilir. Bunlara dikkat etmek gerekir:
- ```
while (i<10)
{ a[i]= ++i;}
```

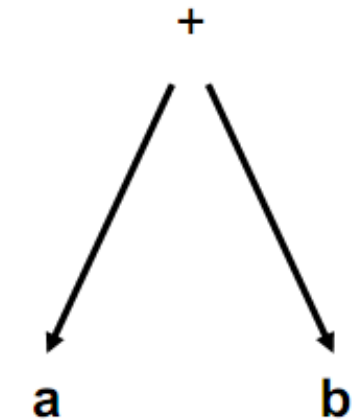
```
while (i<10)
{ a[i]= i++;}
```



# Soyut Sözdizim

- ▶ Bir dilin **soyut sözdizimi**, o dilde bulunan her yapıdaki anlamlı bileşenleri tanımlar.
- ▶ Örneğin;
  - ▶  $ab$  *prefix* ifadesi,
  - ▶  $a+b$  *infix* ifadesi,
  - ▶  $ab+$  *postfix* ifadesinde  
+ işlemcisi ve  $a$  ve  $b$  alt-ifadelerinden oluşan aynı anlamlı bileşenleri içermektedir. Bu nedenle ağaç olarak üçünün de gösterimi yandaki şekilde gibidir.

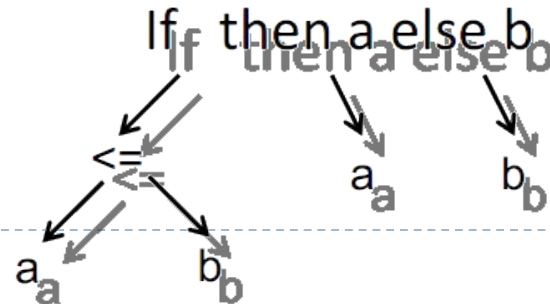
+a *prefix*  
 $a+b$  *infix*  
 $ab+$  *postfix için*



# Soyut Sözdizim

## Soyut Sözdizim Ağaçları

- ▶ Bir ifadedeki işlemci/işlenen yapısını gösteren ağaçlara **soyut sözdizim ağaçları** adı verilir.
- ▶ Soyut sözdizim ağaçları, bir ifadenin yazıldığı gösterimden bağımsız olarak sözdizimsel yapısını gösterebilmeleri nedeniyle bu şekilde isimlendirilirler.
- ▶ Soyut sözdizim ağaçları, uygun işlemcilerin geliştirilmesiyle diğer yapılar için de genişletilebilir.
- ▶ Örneğin
- ▶ **if a <= b then a else b**





# Metinsel Sözdizim

---

- ▶ Hem doğal diller hem de programlama dilleri, bir alfabedeki karakter dizilerinden oluşurlar.
- ▶ Bir dilin karakter dizilerine **cümle** veya **deyim** adı verilir.
- ▶ Bir dilin sözdizim kuralları, o dilin alfabesinden hangi karakter dizilerinin o dilde bulunduklarını belirler. En büyük ve en karmaşık programlama dili bile sözdizimsel olarak çok basittir.
- ▶ Bir programlama dilindeki en düşük düzeyli sözdizimsel birimlere **lexeme** adı verilir. Programlar, karakterler yerine *lexeme*'ler dizisi olarak düşünülebilir.
- ▶ Bir dildeki *lexeme*'lerin gruplanması ile dile ilişkin **token**'lar oluşturulur.



# Metinsel Sözdizim

```
puan = 4 * dogru + 10;
```

| Lexeme | Token             |
|--------|-------------------|
| puan   | Tanımlayıcı       |
| dogru  |                   |
| 4      | tamsayı_sabit     |
| 10     |                   |
| =      | eşit_işareti      |
| *      | çarpım_işlemcisi  |
| +      | toplama_işlemcisi |
| ;      | noktalı virgöl    |

- ▶ Bir programlama dilinin metinsel sözdizimi, *token*'lar ile tanımlanır. Örneğin bir tanımlayıcı; **toplam** veya **sonuc** gibi *lexeme*'leri olabilen bir *token*'dır.
- ▶ Bazı durumlarda, bir *token*'ın sadece tek bir olası *lexeme*'i vardır. Örneğin, toplama\_işlemcisi denilen aritmetik işlemci "+" sembolü için, tek bir olası *lexeme* vardır.
- ▶ Boşluk (*space*), ara (*tab*) veya yeni satır karakterleri, *token*'lar arasına yerleştirildiğinde bir programın anlamı değişmez.
- ▶ Yandaki örnekte, verilen C deyimi için *lexeme* ve *token*'lar listelenmiştir.

# Terminoloji

---

- ▶ Bir *cümle* (*sentence*) herhangi bir alfabede karakterlerden oluşan bir stringdir
- ▶ Bir *dil* (*language*) cümlelerden oluşan bir kümedir
- ▶ Bir *lexeme* bir dilin en alt seviyedeki sentaktik (*syntactic*) birimidir (örn., `*`, `sum`, `begin`)
- ▶ Bir *simge* (*token*) lexemelerin bir kategorisidir (örn., **tanıtıcı** (*identifier*))



---

```
x = (y+3.1) * z_5;
```

**Lexemes**

x

=

(

)

for

y

+

3.1

\*

z\_5

;

**Tokens**

identifier

equal\_sign

left\_paren

right\_paren

for

identifier

plus\_op

float\_literal

mult\_op

identifier

semi\_colon

---



*Karakter akışı*

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



Leksikal Analiz (Tarama)



*Token akışı*

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

token numarası

token değeri



Boşluklar? Sorun olur mu?

```
int x;  
cin >> x;  
if(x>5)  
    cout << "Hello";  
else  
    cout << "BOO";
```

```
int    x    ;  
cin    >>    x    ;  
if    (    x    >    5    )  
    cout    <<    "Hello"    ;  
else  
    cout    <<    "BOO"    ;
```

# Dillerin formal tanımları

---

## ▶ **Tanıyıcılar (Recognizers)**

- ▶ Bir tanıma aygıtı bir dilin girdi stringlerini okur ve girdi stringinin dile ait olup olmadığına karar verir
- ▶ Örnek: bir derleyicinin sentaks analizi kısmı

## ▶ **Üreteçler (Generators)**

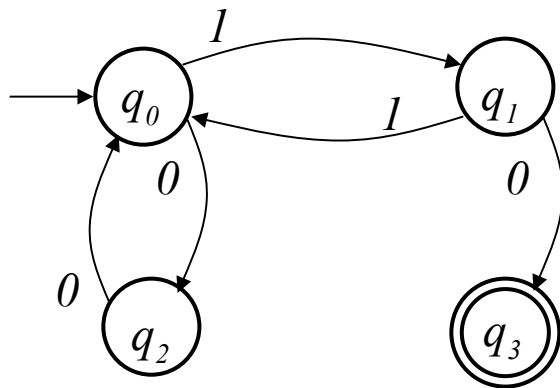
- ▶ Bir dilin cümlelerini üreten aygıttır
- ▶ Belli bir cümlenin sentaksının doğru olup olmadığı, üretecin yapısıyla karşılaştırılarak anlaşılabilir



# Dillerin formal tanımları

## ■ Dil Tanıyıcılar

- Verilen bir programın bir dilde olup olmadığına karar veren bir cihaz
- Mesela, bir derleyicinin syntax analizcisi sonlu otomat



*Sonlu otomatın geçiş diyagramı*

## ■ Dil Üreticiler

- Bir dilin cümlelerini üretmek için kullanılabilen cihaz
- Mesela, regular expressions, context-free grammars

$$((00)^* 1 (11)^*)^+ 0$$

*001110 → Kabul*

*111110 → Kabul*

*000110 → Red*

$$F = (Q, \Sigma, \delta, q_0, F)$$



# Sentaks tanımlamanın biçimsel metotları

---

- ▶ Bir ya da daha çok dilin sözdizimini anlatmak amacıyla kullanılan dile **metadil** adı verilir
- ▶ Programlama dillerinin sözdizimini anlatmak için BNF (Backus-Naur Form) adlı metadil kullanılacaktır. Öte yandan, anlam tanımlama için böyle bir dil bulunmamaktadır.
- ▶ **Backus-Naur Form ve İçerik Bağımsız (içerik-bağımsız) (Context-Free) Gramerler**
  - ▶ Programlama dili sentaksını tanımlayan en çok bilinen metottur.
- ▶ **(Genişletilmiş) Extended BNF**
  - ▶ BNF'un okunabilirliği ve yazılabilirliğini artırır
- ▶ Gramerler ve tanıyıcılar (recognizers)



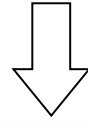
# İçerik Bağımsız (Context Free) Gramer

- ▶ **Gramer**, bir programlama dilinin metinsel (somut) sözdizimini açıklamak için kullanılan bir gösterimdir.
- ▶ Gramerler, anahtar kelimelerin ve noktalama işaretlerinin yerleri gibi metinsel ayrıntılar da dahil olmak üzere, bir dizi kuraldan oluşur.

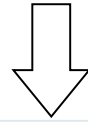


*Karakter akışı*

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



Leksikal Analiz (Tarama)



*Token akışı*

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

token numarası

token değeri

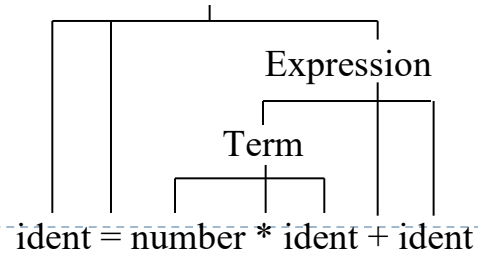


Sentaks Analiz



Statement

*Sentaks ağacı*



# Hatalar

---

- `int x$y;`
- `int 32xy;`
- `45b`
- `45ab`
- `x = x @ y,`

Sözlüksel (Lexical)  
Hatalar / Token Hataları?

- 
- `X = ;`
  - `Y = x +;`
  - `Z = [;`

Syntax Hataları



# BNF ve İçerik Bağımsız (Context-Free) Gramerler

---

- ▶ İçerik Bağımsız (Context-Free) Gramerler
  - ▶ Noam Chomsky tarafından 1950lerin ortalarında geliştirildi
  - ▶ Dil **üreteçleri** (generators), doğal dillerin sentaksını tanımlama amacındaydı
  - ▶ İçerik Bağımsız (Context-Free) diller adı verilen bir diller sınıfı tanımlandı
    - ▶ Bu dillerin özelliği  $A \rightarrow \gamma$  şeklinde gösterilmeleridir. Buradaki  $\gamma$  değeri uç birimler (terminals) ve uç birim olmayanlar (nonterminals) olabilmektedir. Bu diller aşağı sürüklemeli otomatlar (push down automata PDA) tarafından kabul edilen dillerdir ve hemen hemen bütün programlama dillerinin temelini oluşturmaktadırlar.



# Backus-Naur Form (BNF)

---

## ▶ Backus-Naur Form (1959)

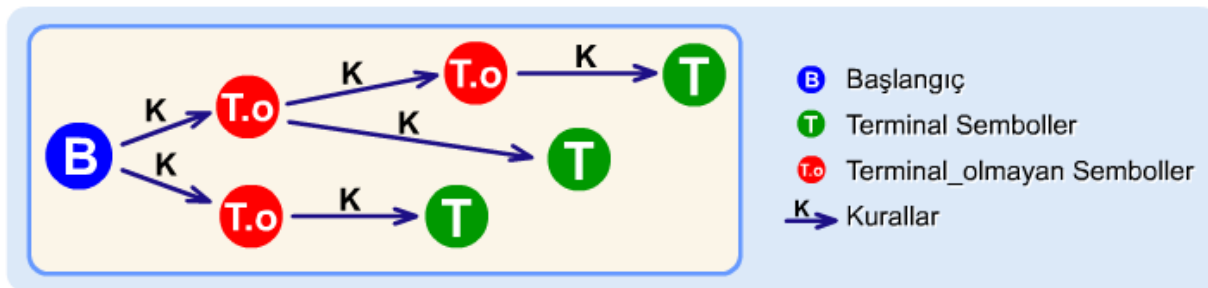
- ▶ John Backus tarafından Algol 58'i belirlemek için icat edildi
- ▶ Bu gösterim şekli, ALGOL60'ın tanımlanması için Peter Naur tarafından biraz değiştirilmiş ve yeni şekli **Backus-Naur (BNF)** formu olarak adlandırılmıştır
- ▶ BNF içerik-bağımsız (context-free) gramerlerin eşdeğeridir
- ▶ BNF başka bir dili tanımlamak için kullanılan bir *metadil*dir
- ▶ BNF'de, soyutlamalar sentaktik (syntactic) yapı sınıflarını temsil etmek için kullanılır--sentaktik değişkenler gibi davranırlar (*nonterminal semboller* adı da verilir)



# Backus-Naur Form (BNF) Temelleri

BNF'de açıklanan bir gramer, 4 bölümden oluşur:

1. Terminal Sembolleri (Atomik uç birimler-lexemeler ve simgeler (tokens))
2. Terminal Olmayan Semboller (Sözdizim değişkenleri)
3. Kurallar (Gramer, üretim, Terminal olmayan sembollerin çözümü)
4. Başlangıç Sembolü (Başlangıç terminal olmayan sembol)



# Backus-Naur Form (BNF)

---

- ▶ **1. Terminal Semboller:** Bir dilde geçerli olan yapıları oluşturmak için birleştirilen daha alt parçalara ayrılamayan (atomik) sembollerdir.  
Örnek: +, \*, -, %, if, >=, vb.
- ▶ **2. Terminal Olmayan Semboller:** Dilin kendisinde bulunmayan, ancak kurallar ile tanımlanan ara tanımları göstermek için kullanılan sembollerdir. BNF'de terminal olmayan semboller "<" ve ">" sembolleri arasında gösterilir ve kurallar ile tanımlanır.  
Örnek: <Statement>, <Expr>, <Type>





# Backus-Naur Form (BNF)

---

- ▶ **3. Kurallar :** Bir terminal olmayan sembolün bileşenlerinin tanımlanmasıdır. Her kuralın sol tarafında bir terminal olmayan daha sonra “:=” veya “→” sembolü ve sağ tarafında ise terminal veya terminal olmayanlardan oluşan bir dizi bileşen bulunur.

Örnek:

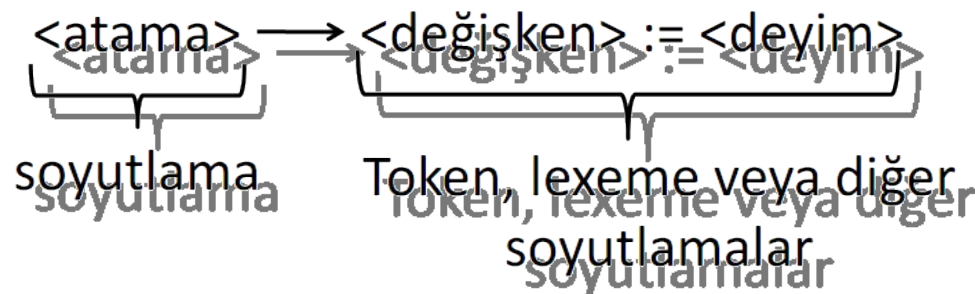
```
<ident list> → identifier | identifier,  
               <ident list>
```

```
<if_stmt> → if <logic_expr> then <stmt>
```



# Backus-Naur Form (BNF)

- ▶ BNF'deki kurallar, söz dizimsel yapıları göstermek için soyutlamalar (kurallar) olarak düşünülebilir.
- ▶ Örnek: Atama deyimi,  $\langle \text{atama} \rangle$  soyutlaması ile aşağıdaki gibi belirtilebilir:

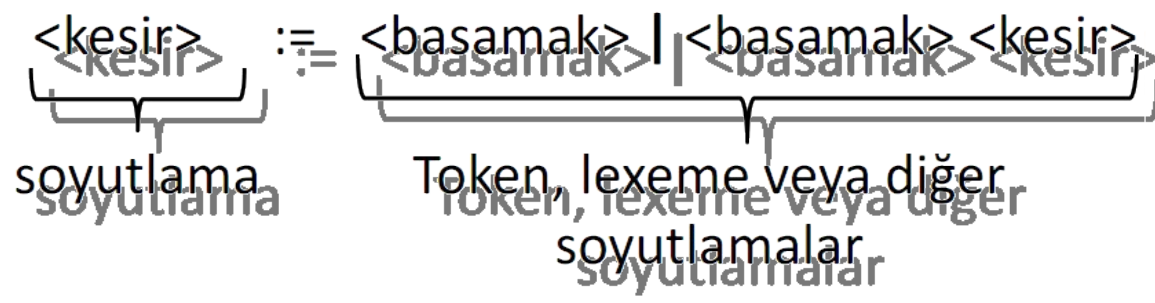


- ▶ Yukarıdaki soyutlama yapılmadan önce  $\langle \text{değişken} \rangle$  ve  $\langle \text{deyim} \rangle$  soyutlamalarının daha önceden yapılmış olması gerekmektedir.
- ▶ Bir gramer, kuralların boş olmayan sonlu bir kümesidir

# Backus-Naur Form (BNF)

- Bir soyutlama (veya kural) için birden çok tanımlama olabilir. Bu durumda bir soyutlama için geçerli olan kurallar “|” ile ayrılır. “|” sembolü veya anlamındadır.

► Örnek:



- Bir soyutlama (abstraction) (veya nonterminal sembol) birden fazla RHS'ye sahip olabilir

```
<stmt> → <single_stmt>
        | begin <stmt_list> end
```

# Backus-Naur Form (BNF)

---

## Özyinelemeli Kurallar:

- ▶ BNF'de bir kural tanımında sol tarafın sağ tarafta yer alması, kuralın özyinelemeli olması olarak açıklanır. Aşağıda görülen  $\langle \text{tanımlayıcı\_listesi} \rangle$ , özyinelemeli kurallara ve nonterminal sembol için birden çok kural olmasına örnektir.
- ▶  $\langle \text{tanımlayıcı\_listesi} \rangle := \text{tanımlayıcı} \mid \text{tanımlayıcı}, \langle \text{tanımlayıcı\_listesi} \rangle$
- ▶ **4. Başlangıç Sembolü**
- ▶ BNF'de dilin ana elemanını göstermek için, terminal olmayan sembollerden biri, başlangıç sembolü (amaç sembol) olarak tanımlanır.



# Grammerler ve Türetmeler

---

- ▶ BNF kullanılarak, bir dilde yer alan cümleler oluşturulabilir. Bu amaçla, başlangıç sembolünden başlayarak, dilin kurallarının sıra ile uygulanması gereklidir. **Bu şekilde cümle oluşturulmasına türetme (derivation) denir** ve BNF türetmeli bir yöntem olarak nitelendirilir.
- ▶ Bir türetme, başlangıç sembolüyle başlayan ve bir cümleyle (tüm terminal sembolleri) biten kuralların tekrarlamalı bir uygulamasıdır.



# Grammerler ve Türetmeler

- Örnek bir gramer :

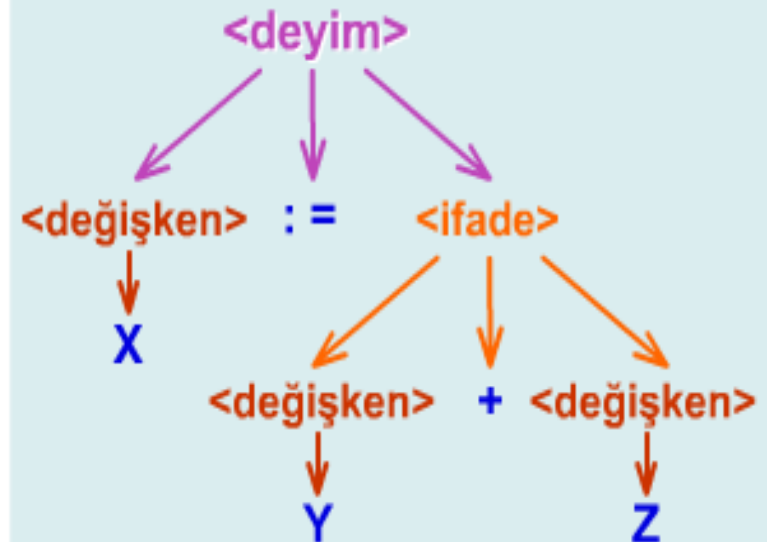
`<program> -> begin <deyim_listesi> end`

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

`<deyim>-> <değişken> :=<ifade>`

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

`<değişken> -> X | Y | Z`



# Grammerler ve Türetmeler

---

- ▶ Bu türetmedeki başlangıç sembolü *<program>* dır.
- ▶ Her cümle, bir önceki cümledeki terminal\_*olmayan*lardan birinin tanımının yerleştirilmesiyle türetilir.
- ▶ Bu türetmede, yeni bir satırda tanımlı yapılan terminal\_*olmayan*, her zaman bir önceki satırda yer alan en soldaki terminal\_*olmayan*dır.
- ▶ Bu sıra ile oluşturulan türetmelere **sola\_dayalı** türetme adı verilir. Türetme işlemi, sadece terminallerden veya *lexeme* lardan oluşan bir cümle oluşturulana kadar devam eder.
- ▶ Bir sola\_*dayalı*\_türetmede, terminal\_*olmayan*ları yerleştirmek için farklı sağ taraf kuralları seçerek, dildeki farklı cümleler oluşturulabilir. Bir türetme, sola\_*dayalı* türetmeye ek olarak, **sağa\_dayalı** olarak veya **ne sağa\_dayalı ne de sola\_dayalı** olarak oluşturulabilir.



# Grammerler ve Türetmeler

---

- Bu dildeki bir programın türetilmesi aşağıdaki örnek türetme üzerinde görülmektedir.

## Örnek

<program> -> begin <deyim\_listesi> end

-> begin <deyim>; end

-> begin <değişken> := <ifade>; end

->begin X := <ifade>; end

->begin X:=<değişken>+<değişken>; end

->begin X := Y + <değişken>; end

->begin X:=Y+Z; end



# Bir Gramer Örneği

---

$\langle \text{program} \rangle \rightarrow \langle \text{stmts} \rangle$

$\langle \text{stmts} \rangle \rightarrow \langle \text{stmt} \rangle \mid \langle \text{stmt} \rangle ; \langle \text{stmts} \rangle$

$\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$

$\langle \text{var} \rangle \rightarrow a \mid b \mid c \mid d$

$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle - \langle \text{term} \rangle$

$\langle \text{term} \rangle \rightarrow \langle \text{var} \rangle \mid \text{const}$



# Bir Türetme (derivation) Örneği

---

`a = b + const` ifadesi türetilebilir mi?

`<program> => <stmts> => <stmt>`

`=> <var> = <expr> => a = <expr>`

`=> a = <term> + <term>`

`=> a = <var> + <term>`

`=> a = b + <term>`

`=> a = b + const`



# Türetme (Derivation)

---

- ▶ Bir türetmede yar alan bütün sembol stringleri cümlesel biçimdedir (sentential form)
- ▶ Bir cümle (sentence) sadece terminal semboller içeren cümlesel bir biçimdir
- ▶ Bir ensol türetme (leftmost derivation), içindeki her bir cümlesel biçimdeki ensol nonterminalin genişletilmiş olmadığı türetmedir
- ▶ Bir türetme ensol (leftmost) veya ensağ (rightmost) dan her ikisi de olmayabilir



# Grammer ve türetme örneği

---

## Grammer

$a=b*(a+c)$

$\langle \text{assign} \rangle \rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$

$\langle \text{id} \rangle \rightarrow a \mid b \mid c$

$\langle \text{expr} \rangle \rightarrow \langle \text{id} \rangle + \langle \text{expr} \rangle$

$\mid \langle \text{id} \rangle * \langle \text{expr} \rangle$

$\mid (\langle \text{expr} \rangle)$

$\mid \text{id}$



# Grammer ve türetme örneği

---

## Sola dayalı türetme

`a=b* (a+c)`

```
<assign> => <id>=<expr>
=>a=<expr>
=>a=<id>*<expr>
=>a=b*<expr>
=>a=b* (<expr>)
=>a=b* (<id>+<expr>)
=>a=b* (a+<expr>)
=>a=b* (a+<id>)
=>a=b* (a+c)
```

### Grammer

`a=b* (a+c)`

`<assign> → <id>=<expr>`

`<id> → a | b | c`

`<expr> → <id> + <expr>`  
`| <id> * <expr>`  
`| (<expr>)`  
`| id`

---

$\langle \text{program} \rangle \rightarrow \mathbf{begin} \langle \text{stmt\_list} \rangle \mathbf{end}$

$\langle \text{stmt\_list} \rangle \rightarrow \langle \text{stmt} \rangle$

$\quad \quad \quad | \langle \text{stmt} \rangle ; \langle \text{stmt\_list} \rangle$

$\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$

$\langle \text{var} \rangle \rightarrow A \mid B \mid C$

$\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle$

$\quad \quad \quad | \langle \text{var} \rangle - \langle \text{var} \rangle$

$\quad \quad \quad | \langle \text{var} \rangle$



# En soldan türetme

$\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle$   
 $\langle \text{var} \rangle - \langle \text{var} \rangle$   
 $\langle \text{var} \rangle$

Bulunması istenen string ifade  
" begin  
A = A + C;  
B = B + C  
end "

left-most derivation

$\langle \text{program} \rangle \Rightarrow \text{begin} \langle \text{stmt-list} \rangle \text{end}$   
 $\Rightarrow \text{begin} \langle \text{stmt} \rangle; \langle \text{stmt-list} \rangle \text{end}$   
 $\Rightarrow \text{begin} \langle \text{var} \rangle = \langle \text{expression} \rangle; \langle \text{stmt-list} \rangle \text{end}$   
 $\Rightarrow \text{begin} A = \langle \text{expression} \rangle; \langle \text{stmt-list} \rangle \text{end}$   
 $\Rightarrow \text{begin} A = \langle \text{var} \rangle + \langle \text{var} \rangle; \langle \text{stmt-list} \rangle \text{end}$   
 $\Rightarrow \text{begin} A = A + \langle \text{var} \rangle; \langle \text{stmt-list} \rangle \text{end}$   
 $\Rightarrow \text{begin} A = A + C; \langle \text{stmt-list} \rangle \text{end}$   
 $\Rightarrow \text{begin} A = A + C; \langle \text{stmt} \rangle \text{end}$   
 $\Rightarrow \text{begin} A = A + C; \langle \text{var} \rangle = \langle \text{expression} \rangle \text{end}$   
 $\Rightarrow \text{begin} A = A + C; B = \langle \text{expression} \rangle \text{end}$

begin A = A + C; B =  $\langle \text{var} \rangle$  end  
begin A = A + C; B = C end ✓



# En sağdan türetme

right-most derivation

$\langle \text{program} \rangle \Rightarrow \text{begin } \langle \text{stmt-list} \rangle \text{ end}$

$\Rightarrow \text{begin } \langle \text{stmt} \rangle; \langle \text{stmt-list} \rangle \text{ end}$

$\Rightarrow \text{begin } \langle \text{stmt} \rangle; \langle \text{stmt} \rangle \text{ end}$

$\Rightarrow \text{begin } \langle \text{stmt} \rangle; \langle \text{var} \rangle = \langle \text{expression} \rangle \text{ end}$

$\Rightarrow \text{begin } \langle \text{stmt} \rangle; \langle \text{var} \rangle = \langle \text{var} \rangle \text{ end}$

$\Rightarrow \text{begin } \langle \text{stmt} \rangle; \langle \text{var} \rangle = C \text{ end}$

$\Rightarrow \text{begin } \langle \text{stmt} \rangle; B = C \text{ end}$

$\Rightarrow \text{begin } \langle \text{var} \rangle = \langle \text{expression} \rangle; B = C \text{ end}$

$\Rightarrow \text{begin } \langle \text{var} \rangle = \langle \text{var} \rangle + \langle \text{var} \rangle; B = C \text{ end}$

$\Rightarrow \text{begin } \langle \text{var} \rangle = \langle \text{var} \rangle + C; B = C \text{ end}$

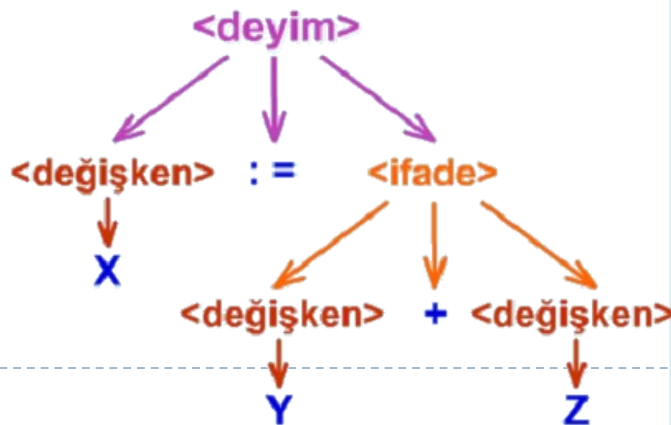
$\Rightarrow \text{begin } \langle \text{var} \rangle = B + C; B = C \text{ end}$

$\Rightarrow \text{begin } A = B + C; B = C \text{ end } \checkmark$



# Ayrıştırma Ağacı (Parse Tree)

- Gramerler, tanımladıkları dilin cümlelerinin hiyerarşik sözdizimsel yapısını tarif edebilirler. Bu hiyerarşik yapılara **ayrıştırma (parse) ağaçları** denir. Bir ayrıştırma ağacının en aşağıdaki düğümlerinde terminal semboller yer alır.
- Ayrıştırma ağacının diğer düğümleri, dil yapılarını gösteren terminal olmayanları içerir. **Ayrıştırma ağaçları ve türetmeler birbirleriyle ilişkili olup, birbirlerinden türetilirler.**
- Aşağıdaki şekilde yer alan ayrıştırma ağacı, “X := Y + Z”, deyiminin yapısını göstermektedir.



# Grammer ve türetme örneği

## Sola dayalı türetme

$a=b*(a+c)$

$\langle \text{assign} \rangle \Rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$

$\Rightarrow a = \langle \text{expr} \rangle$

$\Rightarrow a = \langle \text{id} \rangle * \langle \text{expr} \rangle$

$\Rightarrow a = b * \langle \text{expr} \rangle$

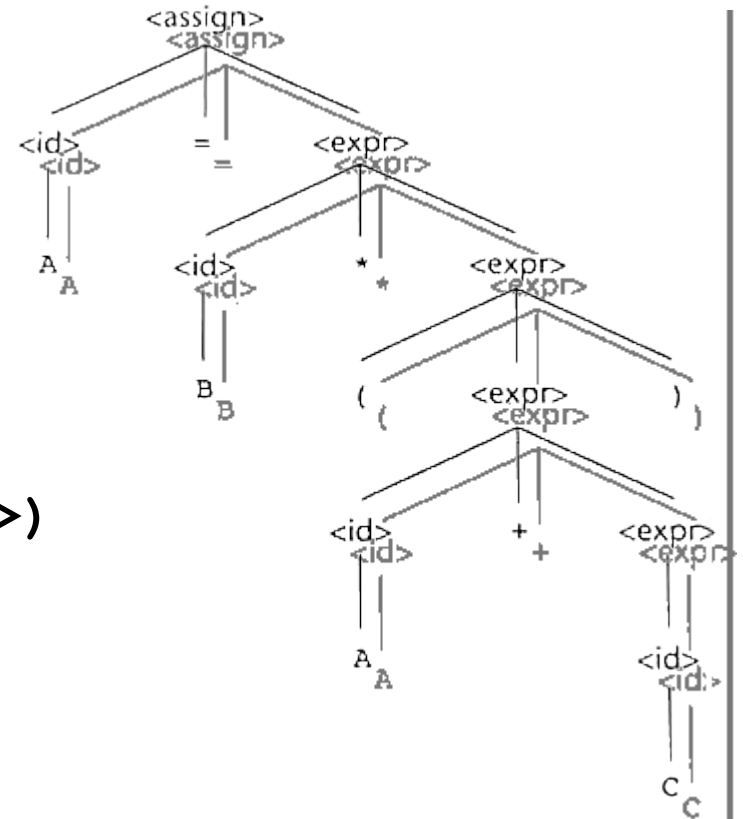
$\Rightarrow a = b * (\langle \text{expr} \rangle)$

$\Rightarrow a = b * (\langle \text{id} \rangle + \langle \text{expr} \rangle)$

$\Rightarrow a = b * (a + \langle \text{expr} \rangle)$

$\Rightarrow a = b * (a + \langle \text{id} \rangle)$

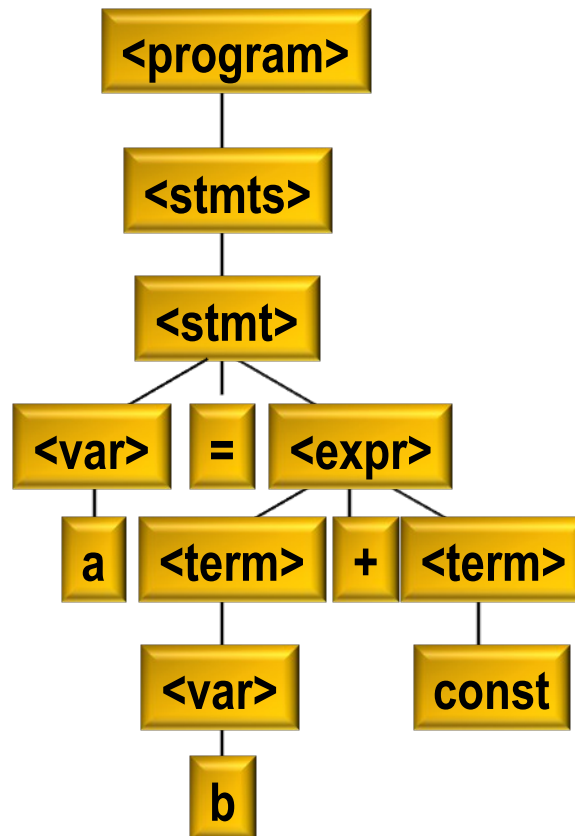
$\Rightarrow a = b * (a + c)$



# Ayrıştırma Ağacı (Parse Tree)

---

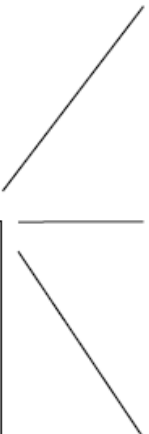
- Bir türetmenin (derivation) hiyerarşik gösterimi



---

**<program>**     $\Rightarrow$  **begin** <stmt\_list> **end**  
                   $\Rightarrow$  **begin** <stmt> ; <stmt\_list> **end**  
                   $\Rightarrow$  **begin** <var> := <expression>; <stmt\_list> **end**  
                   $\Rightarrow$  **begin** A := <expression>; <stmt\_list> **end**  
                   $\Rightarrow$  **begin** A := B; <stmt\_list> **end**  
                   $\Rightarrow$  **begin** A := B; <stmt> **end**  
                   $\Rightarrow$  **begin** A := B; <var> := <expression> **end**  
                   $\Rightarrow$  **begin** A := B; C := <expression> **end**  
                   $\Rightarrow$  **begin** A := B; C := <var><arith\_op><var> **end**  
                   $\Rightarrow$  **begin** A := B; C := A <arith\_op> <var> **end**  
                   $\Rightarrow$  **begin** A := B; C := A \* <var> **end**  
                   $\Rightarrow$  **begin** A := B; C := A \* B **end**

Each of these strings is called **sentential form**



If always the leftmost nonterminal is replaced, then it is called leftmost derivation.

---



# Grammerlerde Belirsizlik (Ambiguity)

---

- ▶ Bir gramer ancak ve ancak iki veya daha fazla farklı ayrıştırma ağacı olan bir cümlesel biçim (sentential form) üretiyorsa *belirsizdir*

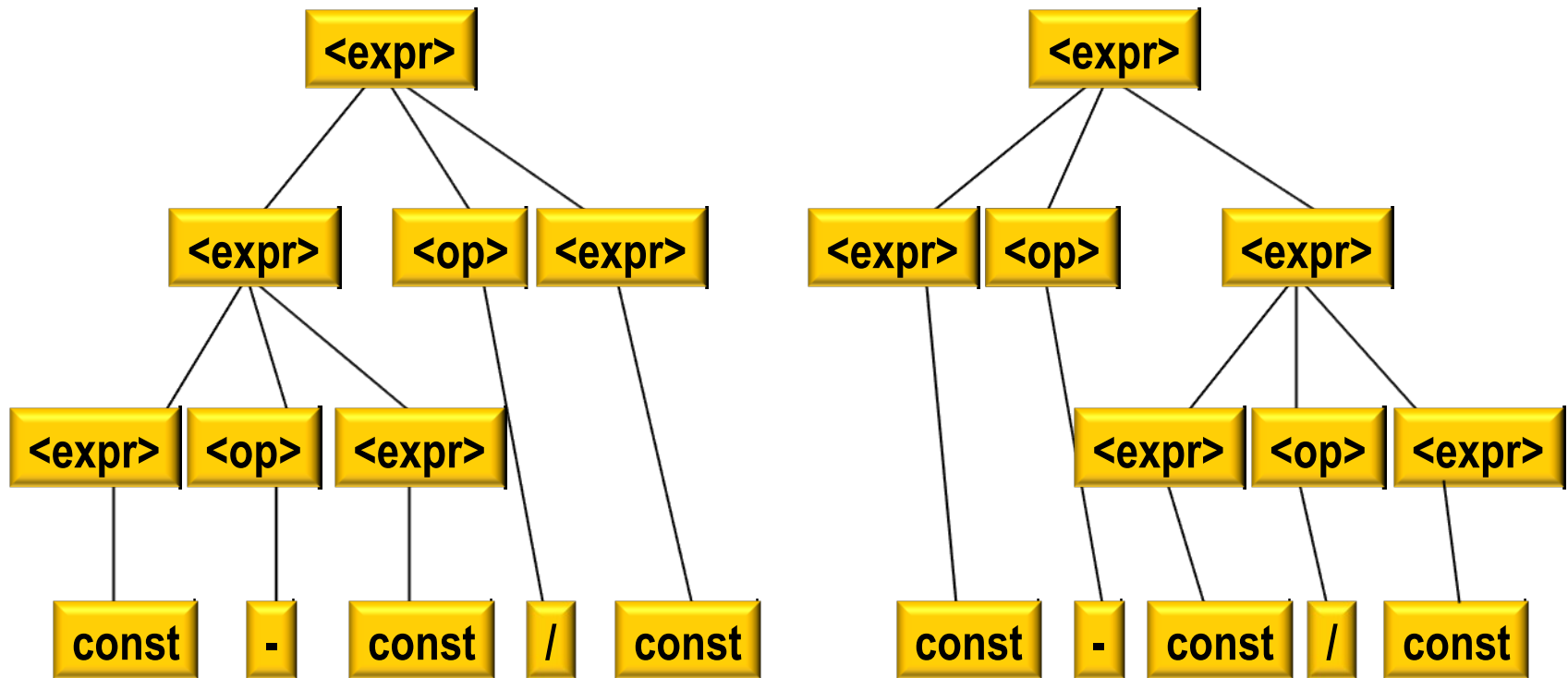


# Bir Belirsiz Deyim Grameri

$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \mid \text{const}$

$\langle \text{op} \rangle \rightarrow / \mid -$

**const – const / const**



# Kaynaklar

---

- ▶ Programlama Dillerinin Prensipleri, Prof. Dr. Nejat Yumuşak, Dr. Muhammed Fatih Adak, Seçkin Yayıncılık
- ▶ Sakarya Üniversitesi, Bilgisayar ve Bilişim Mühendisliği Programlama dilleri ve kavramları Ders Sunumları
- ▶ Concepts of Programming Languages (I I. ed.), Robert W. Sebesta sunumları
- ▶ Dr. Erkan Duman, Programlama Dilleri ders notları
- ▶ Harran Üniversitesi, Programlama dilleri ders sunumları

