Chapter 3
Context-free Grammars & Parsing

한양대학교 컴퓨터공학부

컴파일러

2014년 2학기

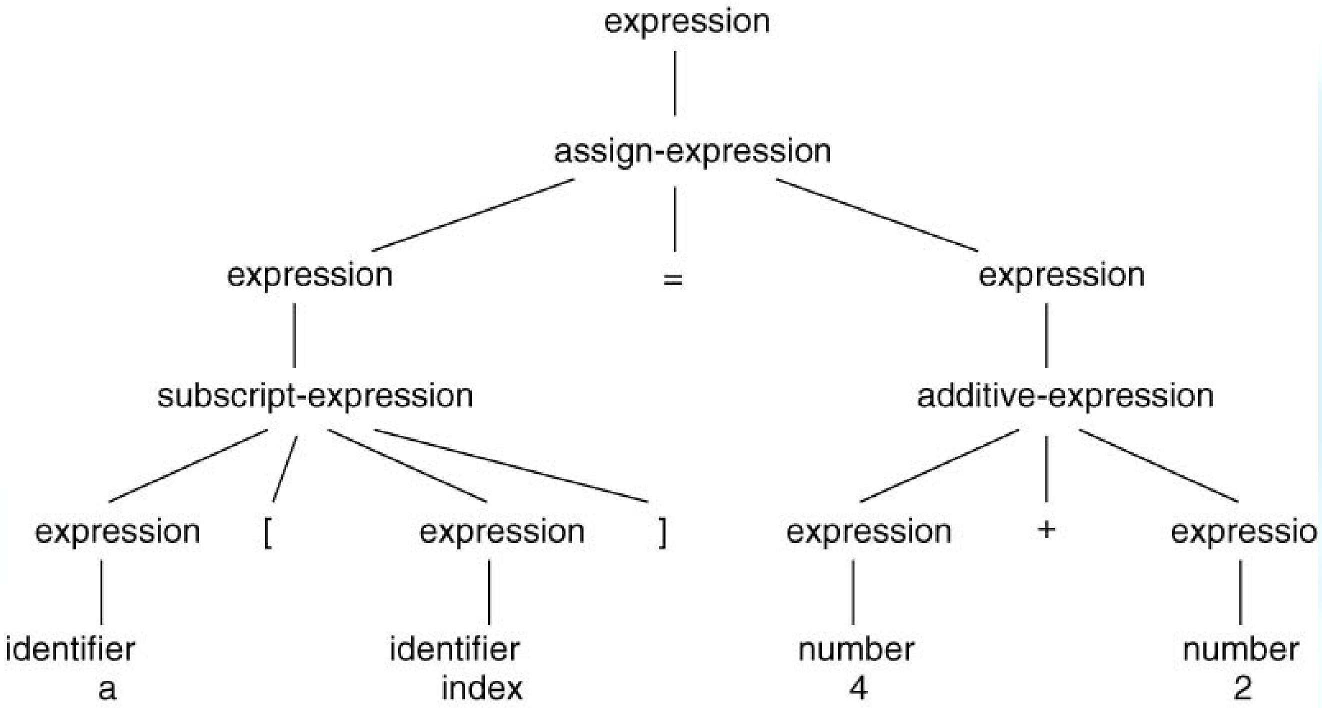
Introduction

- Parsing
  - task of determining the syntax, or structure, of a program
  - also called **syntax analysis**
- Syntax of a programming language
  - usually given by the **grammar rules** of a **context-free grammar**
- major difference between regular expressions and the rules of a context-free grammar
  - recursion

The parsing process

- sequences of tokens → parse tree or syntax tree

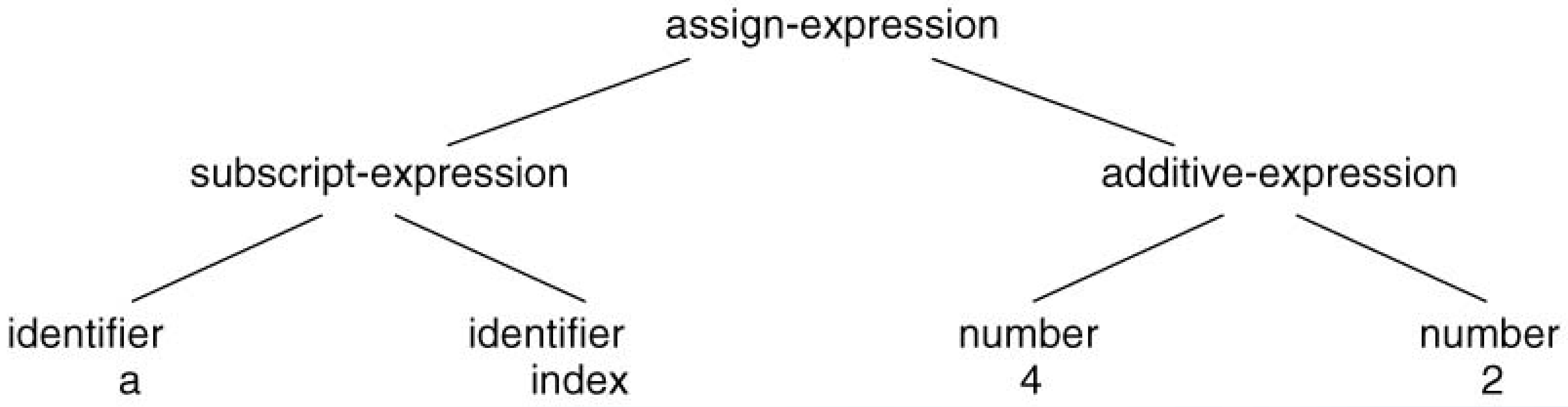
a / [/ index /] / = / 4 / + / 2



The parsing process

- sequences of tokens → parse tree or syntax tree

a / [/ index /] / = / 4 / + / 2



Context-free grammars

- Representing lexical structures
  - Regular expressions
- Representing syntactic structures
  - Context-free grammars

- A context-free grammar is a set of grammar rules.

Context-free grammars

- A context-free grammar rule in BNF (Backus Naur form) consists of a string of symbols.
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - The first symbol is a name for a structure.
  - The second symbol is "→"
  - which is followed by a string of symbols, each of which is either a token , a name, or |.

Context-free grammars

- Other form of grammar rules
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - <exp> ::= <exp> <op> <exp> | (<exp>) | NUMBER

- Equivalence of grammar rules
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - *exp* → *exp op exp*
    exp → (*exp*)
    exp → ***number***

Grammar and derivations

- Grammar: set of grammar rules
  - start symbol *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - *op* → **+** | **-** | *****
- A derivation is a sequence of replacements of structure names by choices on the right-hand sides of grammar rules.
  - (34 – 3) * 42
  - exp => *exp op exp*

⇒ *exp op* ***number***

⇒ *exp* ***** ***number***

⇒ **(***exp***)** ***** ***number***

⇒ **(***exp op exp***)** ***** ***number***

⇒ **(***exp op* ***number*** **)** ***** ***number***

⇒ **(***exp* **–** ***number*****)** ***** ***number***

⇒ **(*****number – number*****)*** ***number***

sentential forms

sentence

The language defined by a grammar

- The language defined by a grammar is the set of all strings of token symbols obtained by derivation.
  - L(G) = { s | *exp* =>* s }
- Nonterminals and terminals
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - *op* → **+** | **-** | *****

Grammar examples

- **Example 3.1**
  - $E \rightarrow (E) \mid a$
- **Example 3.2**
  - $E \rightarrow (E)$
- **Example 3.3**
  - $E \rightarrow E + a \mid a$

Grammar examples

- **Example 3.4**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement* | **if (** *exp* **)** *statement* **else** *statement*
  - *exp* → **0** | **1**

other

if (0) other

if (1) other else other

if (0) if (0) other

if (0) if (1) other else other

if (1) other else if (0) other else other

Grammar examples

- **Repetition**
  - $a+$
    - $A \rightarrow Aa \mid a$
      - $A \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow aaaa$ (left recursive)
    - $A \rightarrow aA \mid a$
      - $A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow aaaa$ (right recursive)
  - $A \rightarrow A\alpha|\beta$
      - $\beta\alpha^*$
  - $A \rightarrow \alpha A|\beta$
      - $\alpha^*\beta$

Grammar examples

- **Repetition**
  - $a*$
    - $A \rightarrow Aa \mid \varepsilon$
    - $A \rightarrow aA \mid \varepsilon$
- **Example 3.5**
  - $A \rightarrow (A)A \mid \varepsilon$

Grammar examples

- **Example 3.4**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement* | **if (** *exp* **)** *statement* **else** *statement*
  - *exp* → **0** | **1**

- **Example 3.6**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement else-part*
  - *else-part* → **else** *statement* | **ε**
  - *exp* → **0** | **1**

Grammar examples

- **Example 3.7**
  - *stmt-sequence* → *stmt* **;** *stmt-sequence* | *stmt*
  - *stmt* → **s**

{s, s;s, s;s;s, … }

(add *ε*)

- *stmt-sequence* → *stmt* **;** *stmt-sequence* | ***ε***
- *stmt* → **s**

{*ε* , s;, s;s;, s;s;s;, … }

(no ; in the end)

- *stmt-sequence* → *nonempty-stmt-sequence* | ***ε***
- *nonempty-stmt-sequence* → *stmt* **;** *nonempty-stmt-sequence* | *stmt*
- *stmt* → **s**

{*ε* , s, s;s, s;s;s, … }

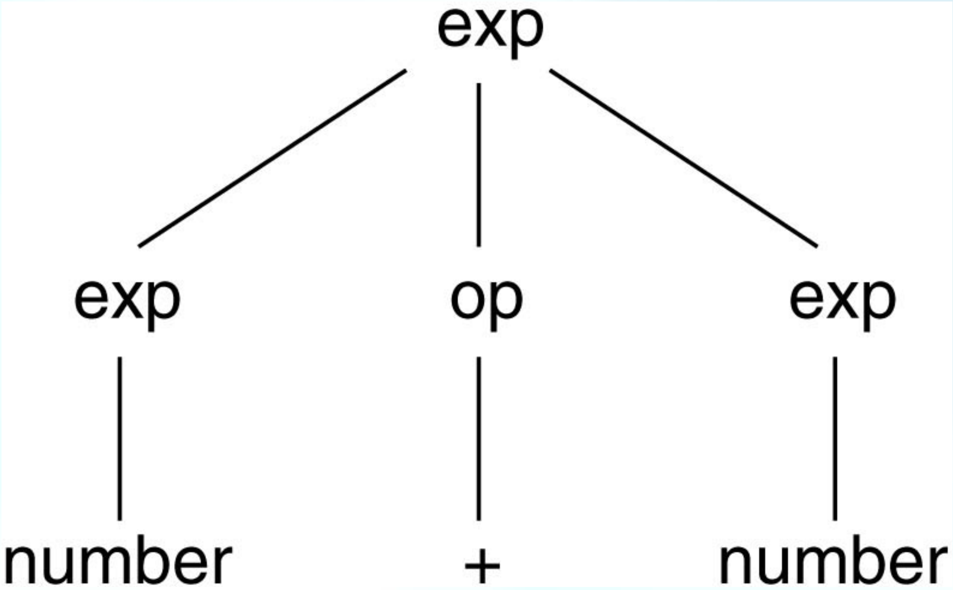
Parse trees

- Grammar
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - *op* → **+** | **-** | *****
- A derivation for ***number + number***
  - exp => *exp op exp*

    ⇒ ***number*** *op exp*

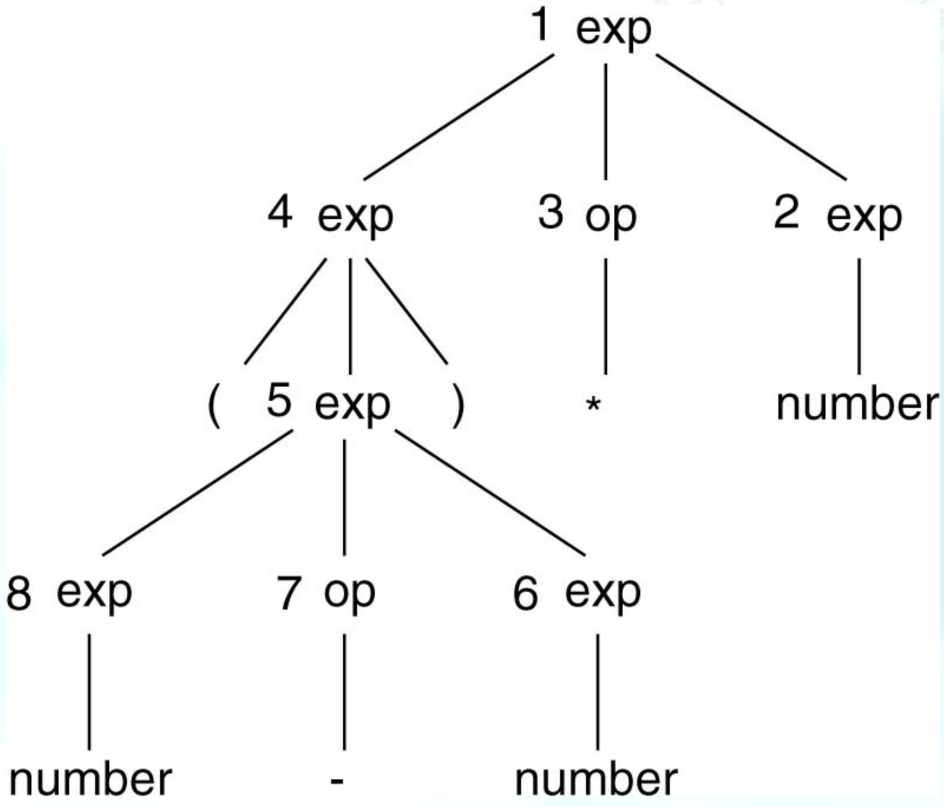
    ⇒ ***number* +** *exp*

    ⇒ ***number + number***



Parse trees

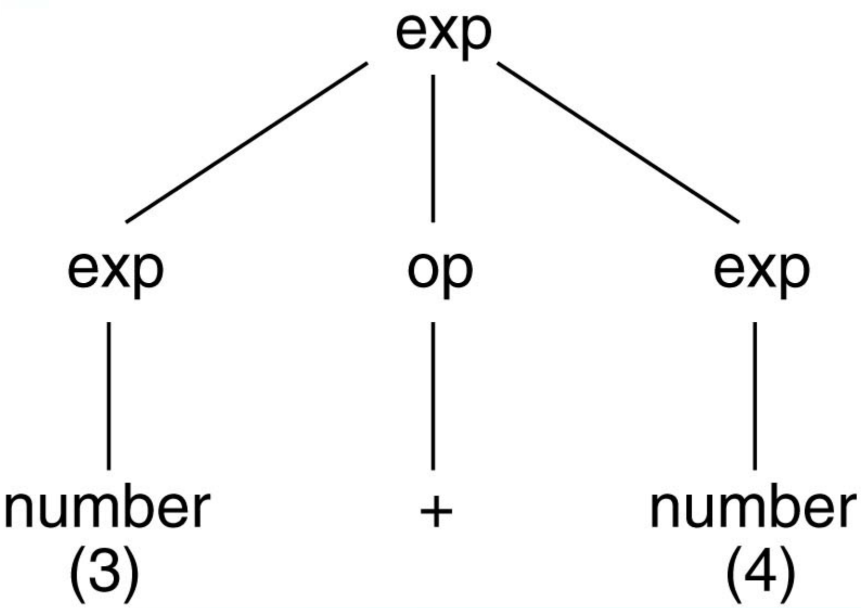
- Grammar
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - *op* → **+ | - | ***
- A parse tree for **(34-3)*42**



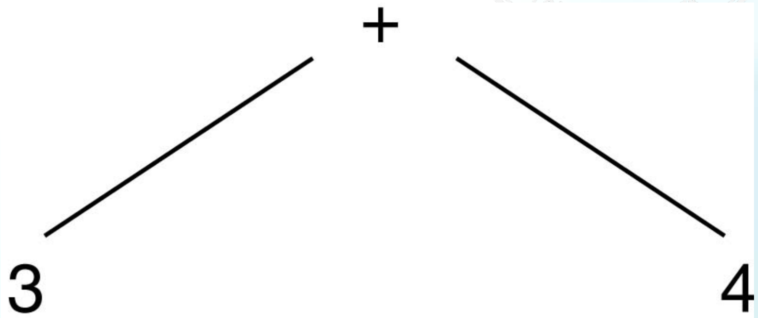
Abstract syntax trees

- **3+4**

Parse tree



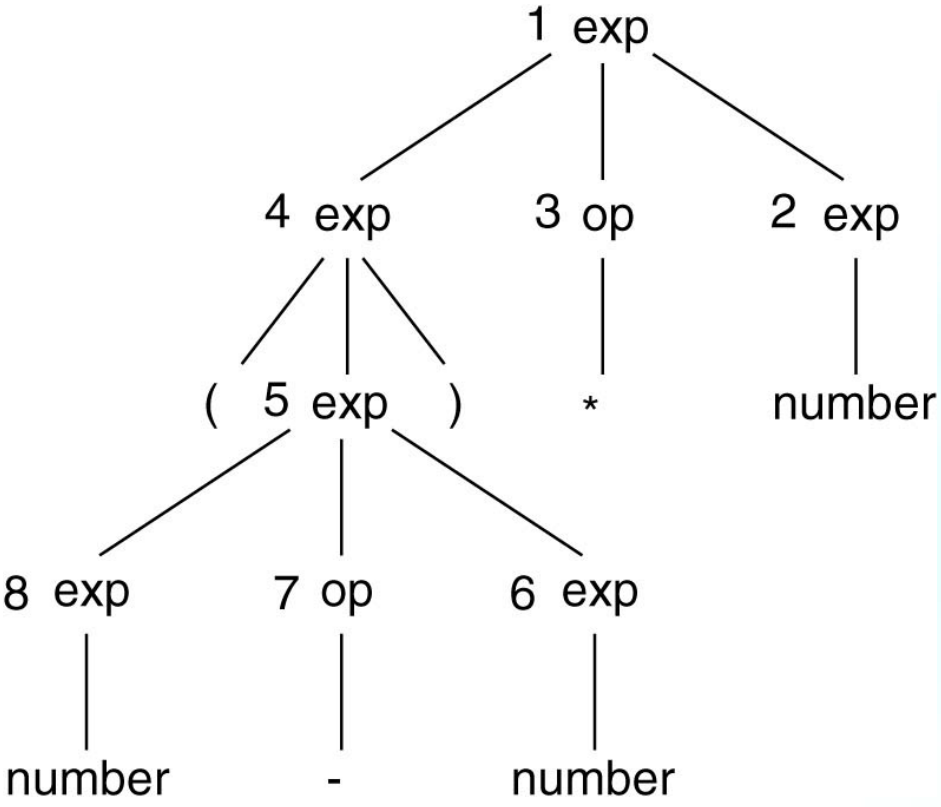
Abstract syntax tree



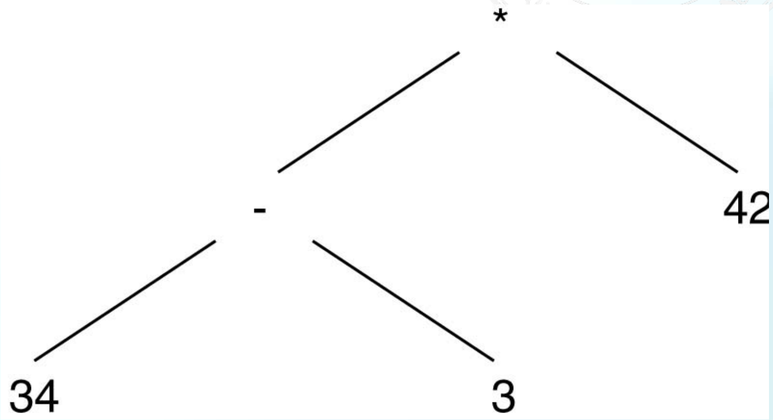
Abstract syntax trees

- **(34-3)*42**

Parse tree



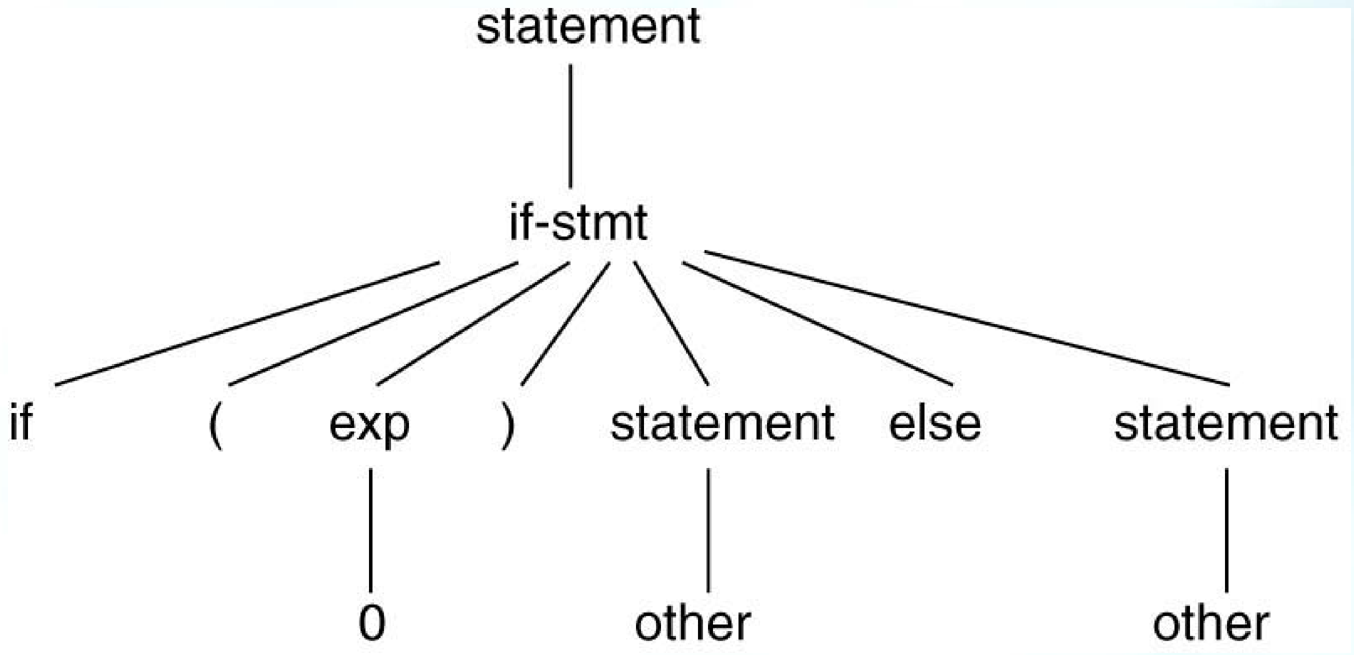
Abstract syntax tree



Abstract syntax trees

- **if (0) other else other**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement* | **if (** *exp* **)** *statement* **else** *statement*
  - *exp* → **0** | **1**

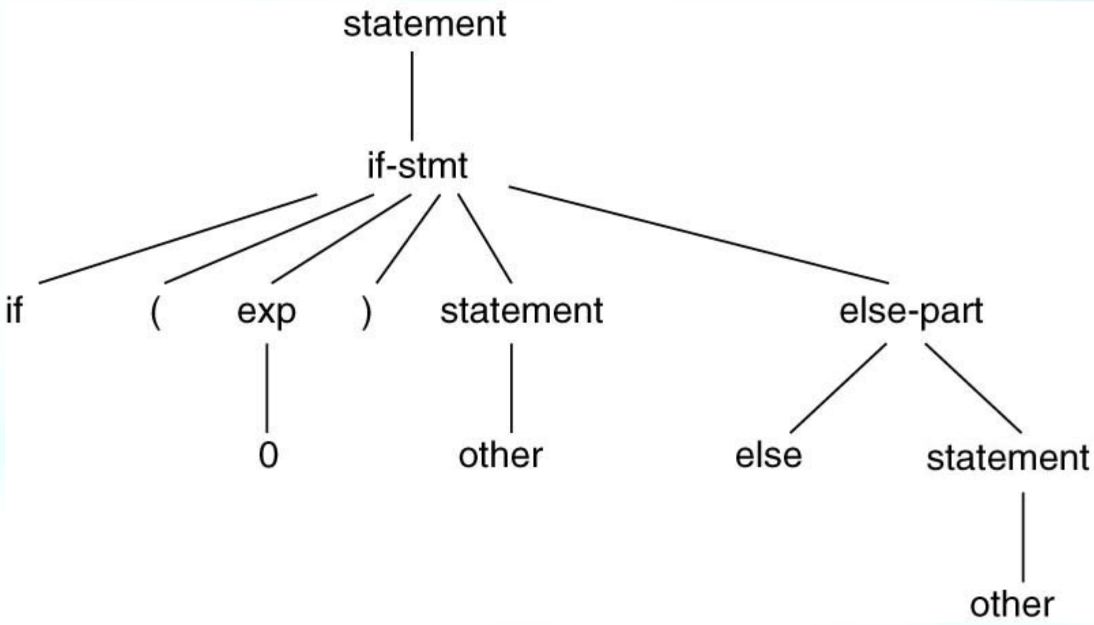
Parse tree



Abstract syntax trees

- **if (0) other else other**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement else-part*
  - *else-part* → **else** *statement* | **ε**
  - *exp* → **0** | **1**

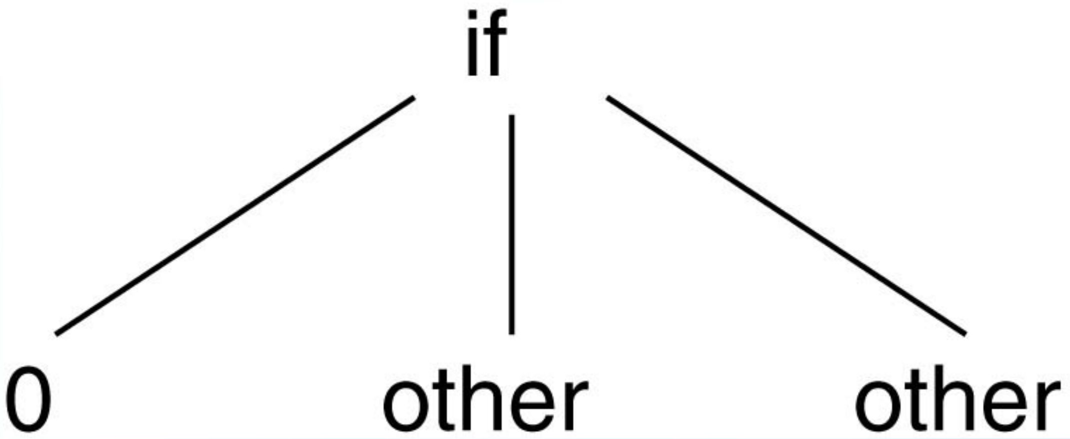
Parse tree



Abstract syntax trees

- **if (0) other else other**

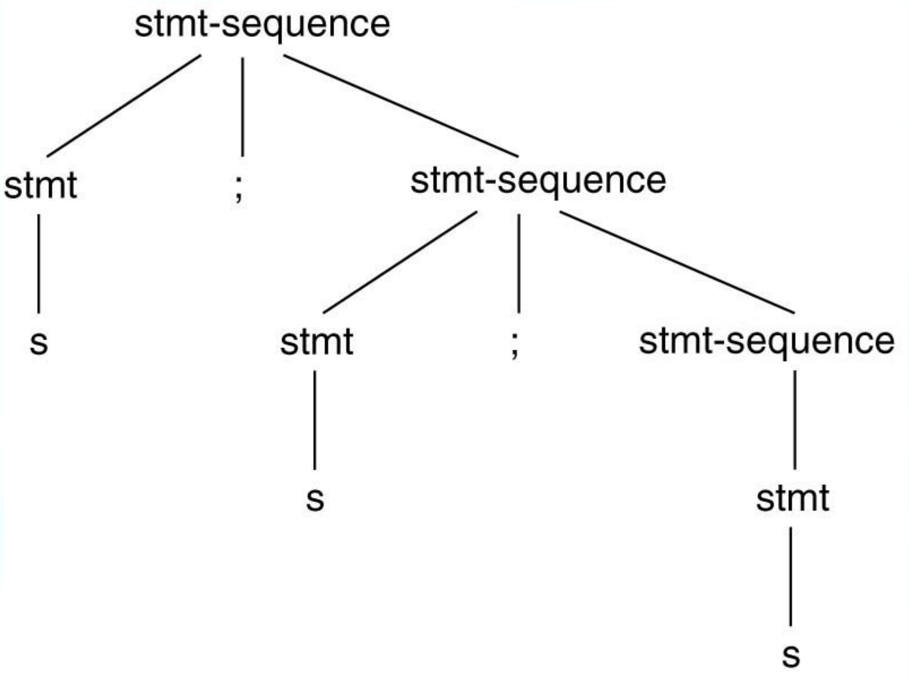
Abstract syntax tree



Abstract syntax trees

- **s; s; s**
  - *stmt-sequence* → *stmt* **;** *stmt-sequence* | *stmt*
  - *stmt* → **s**

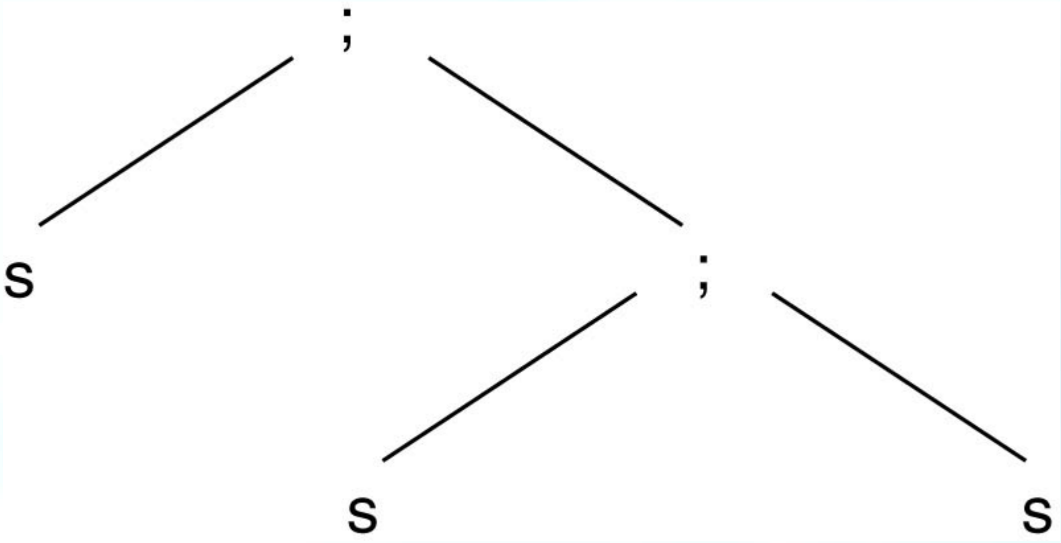
Parse tree



Abstract syntax trees

- **s; s; s**
  - *stmt-sequence* → *stmt* **;** *stmt-sequence* | *stmt*
  - *stmt* → **s**

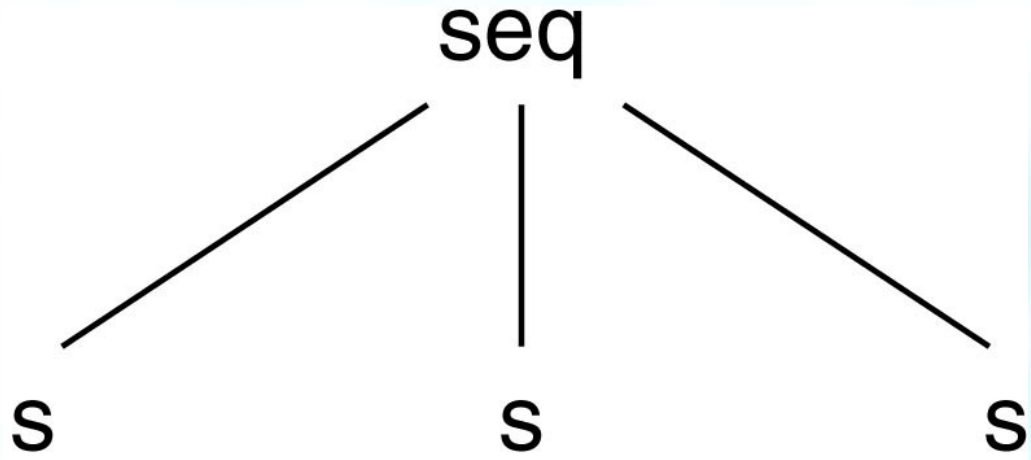
Abstract syntax tree



Abstract syntax trees

- **s; s; s**
  - *stmt-sequence* → *stmt* **;** *stmt-sequence* | *stmt*
  - *stmt* → **s**

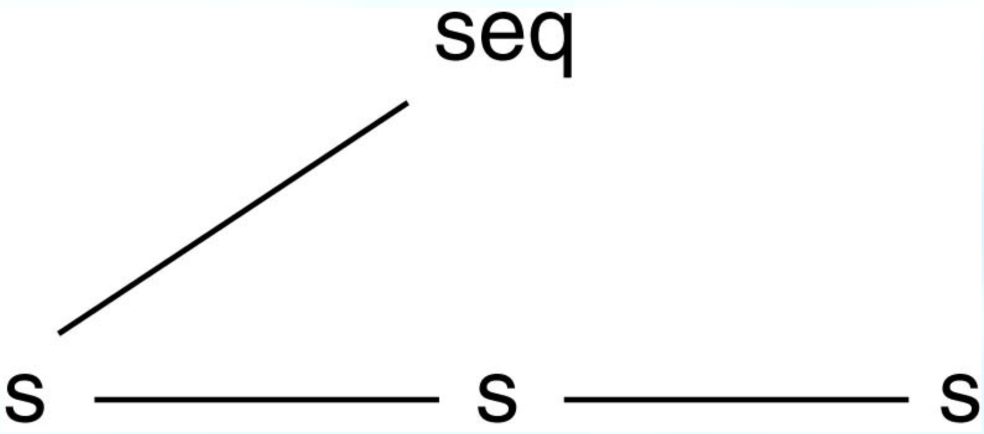
Abstract syntax tree



Abstract syntax trees

- **s; s; s**
  - *stmt-sequence* → *stmt* **;** *stmt-sequence* | *stmt*
  - *stmt* → **s**

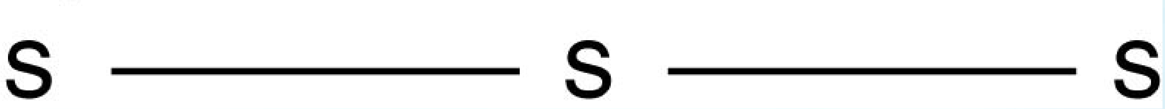
Abstract syntax tree



Abstract syntax trees

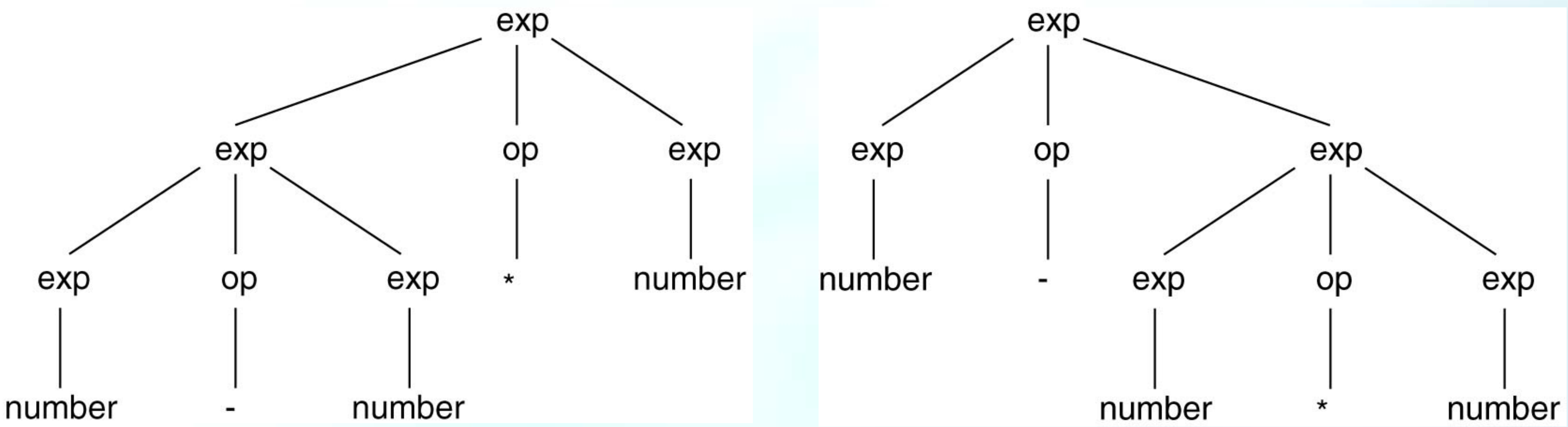
- **s; s; s**
  - *stmt-sequence* → *stmt* **;** *stmt-sequence* | *stmt*
  - *stmt* → **s**

Abstract syntax tree



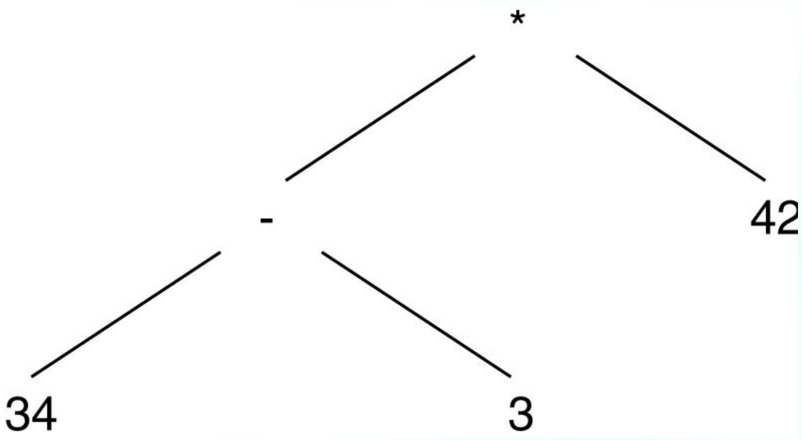
Ambiguity

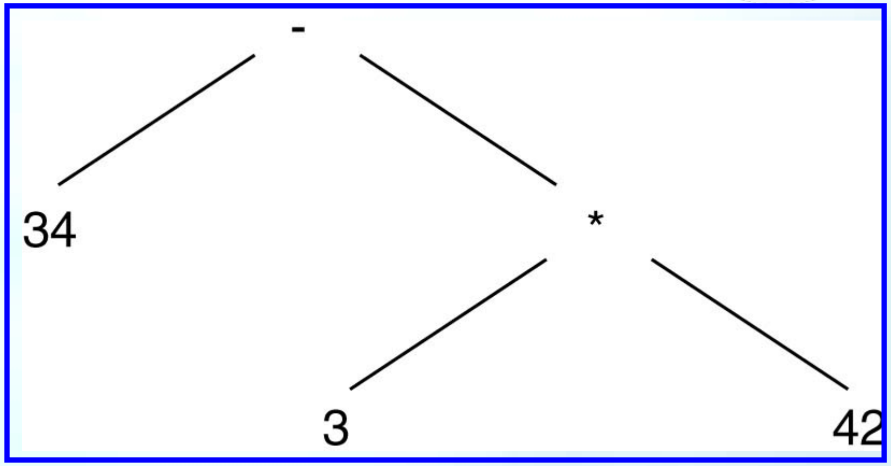
- ***number* – *number* * *number***
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - *op* → **+** | **-** | *****



Ambiguity

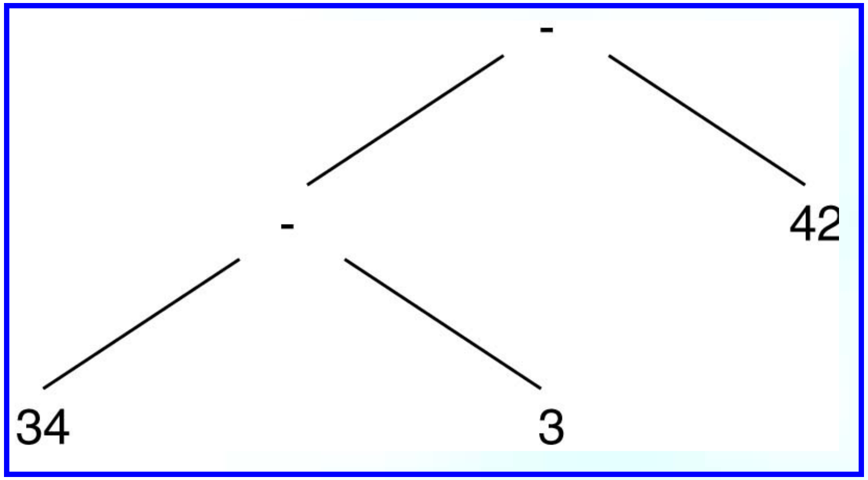
- **34 – 3 * 42 (precedence)**
  - *exp* → *exp op exp* | **(** *exp* **)** | ***number***
  - *op* → **+ | - | ***

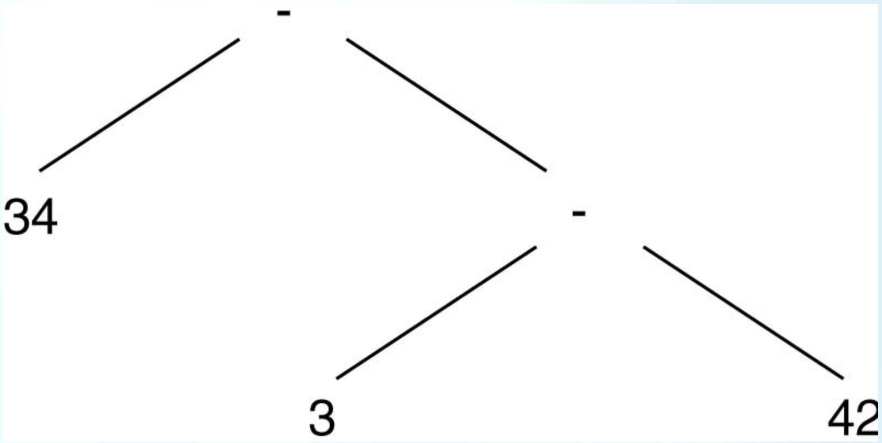




Ambiguity

- **34 – 3 – 42 (Associativity)**
  - *exp* → *exp op exp* | (*exp*) | ***number***
  - *op* → + | - | *





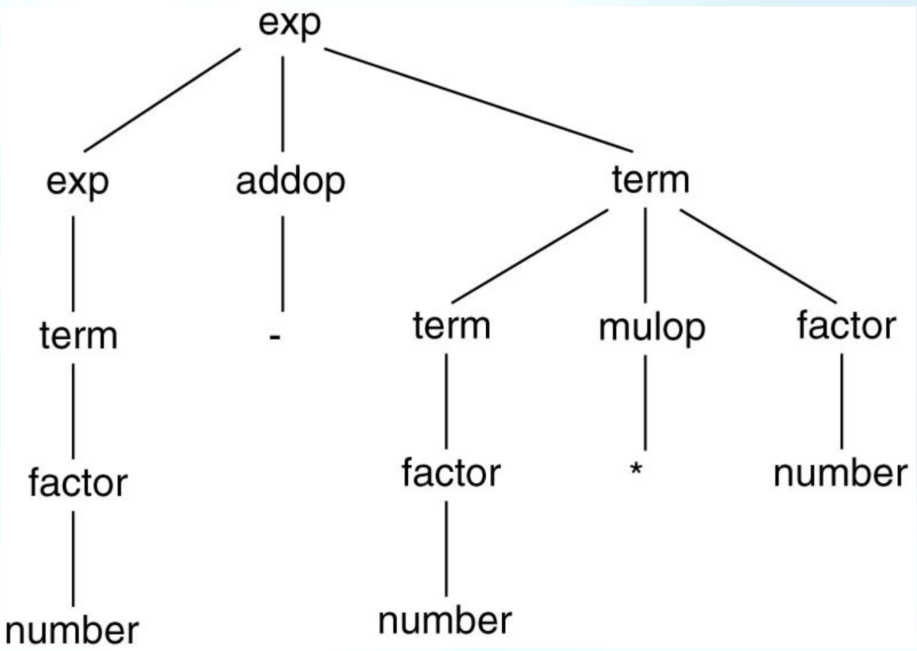
Ambiguity

- **Precedence cascade**
  - *exp* → *exp addop exp* | *term*
  - *addop* → **+** | **-**
  - *term* → *term mulop term* | *factor*
  - *mulop* → *
  - *factor* → (*exp*) | ***number***

- **Associativity (left)**
  - *exp* → *exp addop term* | *term*
  - *addop* → **+** | **-**
  - *term* → *term mulop factor* | *factor*
  - *mulop* → *
  - *factor* → (*exp*) | ***number***

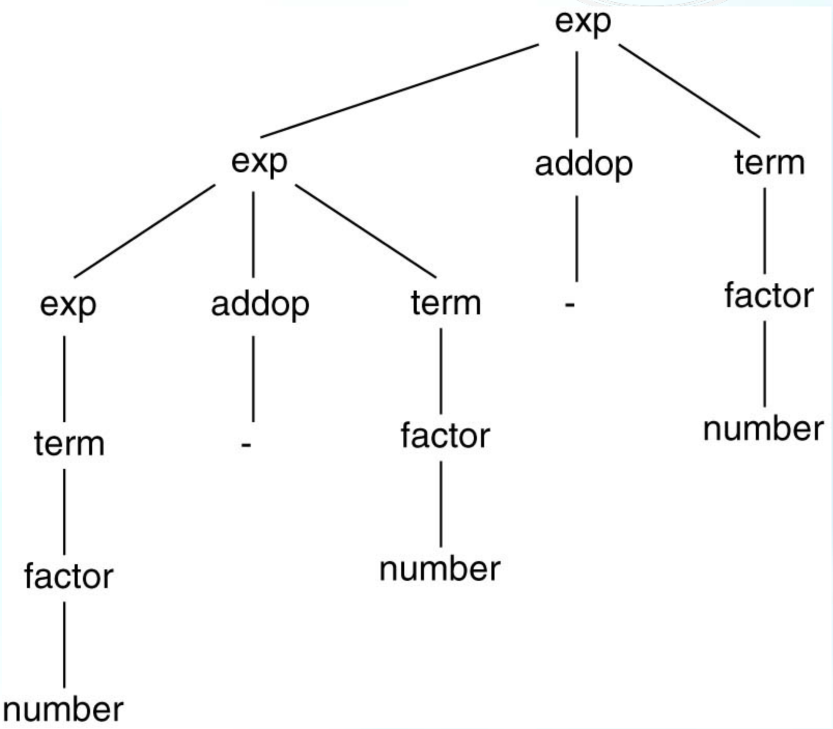
Ambiguity

- **34 – 3 * 42**
  - *exp* → *exp addop term* | *term*
  - *addop* → **+** | **-**
  - *term* → *term mulop factor* | *factor*
  - *mulop* → *
  - *factor* → (*exp*) | ***number***



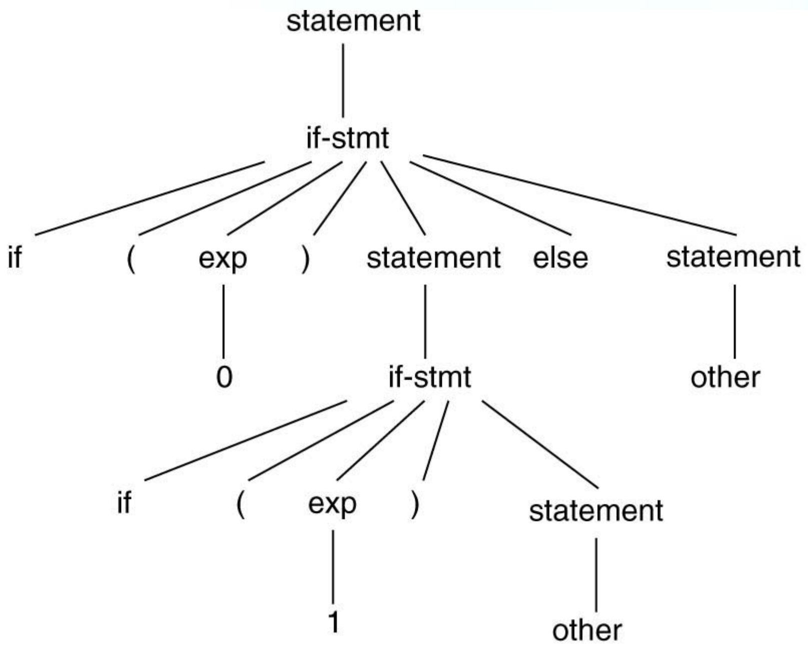
Ambiguity

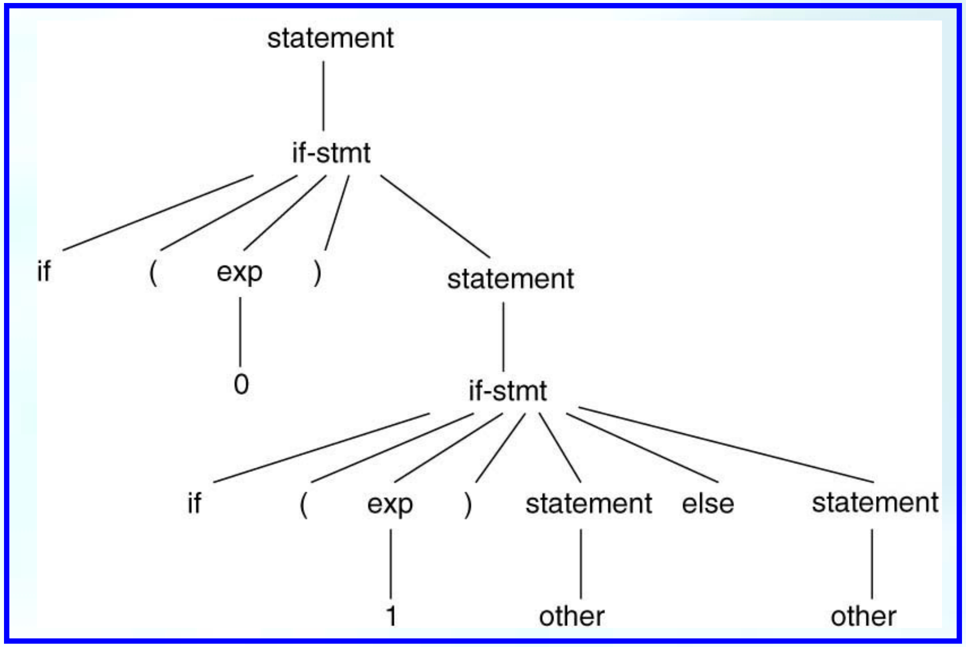
- **34 – 3 – 42**
  - *exp* → *exp addop term* | *term*
  - *addop* → **+ | -**
  - *term* → *term mulop factor* | *factor*
  - *mulop* → *
  - *factor* → (*exp*) | ***number***



Ambiguity

- **if (0) if (1) other else other (Dangling else)**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement* | **if (** *exp* **)** *statement* **else** *statement*
  - *exp* → **0** | **1**





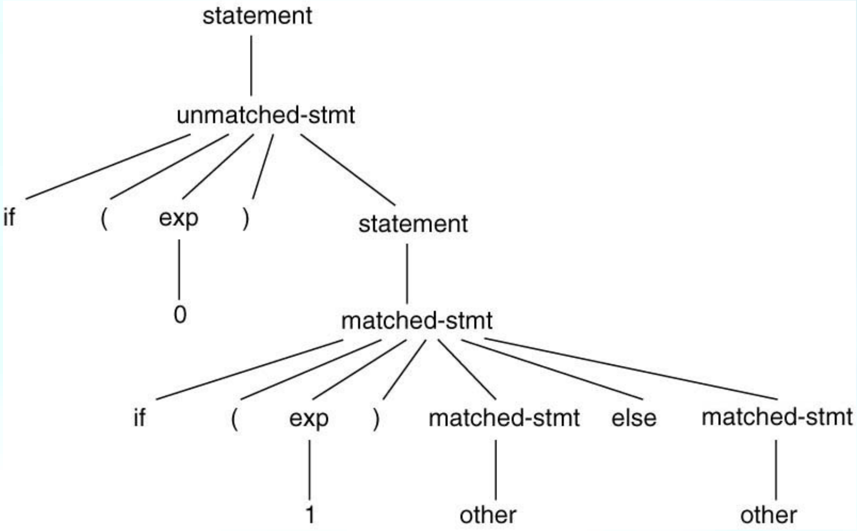
most closely nested rule

Ambiguity

- **A new grammar without the dangling else problem.**
  - *statement* → *matched-stmt* | *unmatched-stmt*
  - *matched-stmt* → **if (** *exp* **)** *matched-stmt* **else** *matched-stmt* | **other**
  - *unmatched-stmt* → **if (** *exp* **)** *statement*
    | **if (** *exp* **)** *matched-stmt*
    else *unmatched-stmt*
  - *exp* → **0 | 1**

Ambiguity

- **if (0) if (1) other else other**
  - *statement* → *matched-stmt* | *unmatched-stmt*
  - *matched-stmt* → **if (** *exp* **)** *matched-stmt* **else** *matched-stmt* | **other**
  - *unmatched-stmt* → **if (** *exp* **)** *statement*
    | **if (** *exp* **)** *matched-stmt* **else** *unmatched-stmt*
  - *exp* → **0** | **1**



Ambiguity

- **Other methods to solving dangling else problem**
  - Disambiguating rules
    - Bracketing keywords (p. 122)
      - *if-stmt* → **if** *condition* **then** *statement-sequence* **end if**
        | **if** *condition* **then** *statement-sequence* **else** *statement-sequence* **end if**
- Inessential ambiguity (p. 122)
  - Example
    - *stmt-sequence* → *stmt-sequence* **;** *stmt-sequence* | *stmt*
    - *stmt* → **s**
  - But still obtain unique syntax trees

EBNF

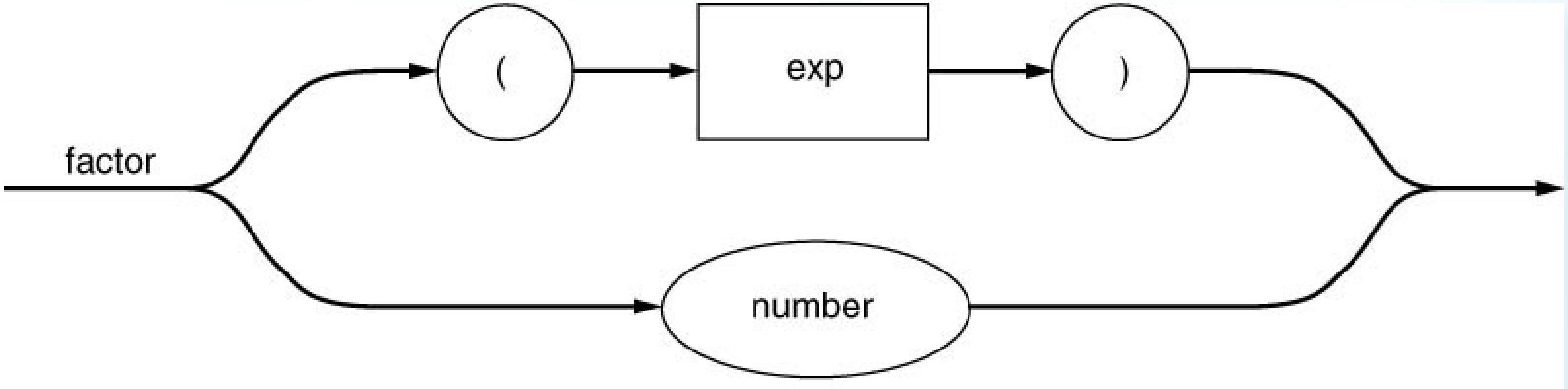
- {}
  - Repetition
  - $A \rightarrow A\alpha \mid \beta$ (left recursive) : $A \rightarrow \beta\{\alpha\}$
  - $A \rightarrow \alpha A \mid \beta$ (right recursive) : $A \rightarrow \{\alpha\}\beta$
  - p. 124
- []
  - Optional
  - *if-stmt* → **if (** *exp* **)** *statement* [**else** *statement*]
- ( )
  - Choice
  - *exp* → *exp* **("+" | "-" | "*")** *exp* | **"("** *exp* **")"** | ***number***

Syntax Diagrams

- Representing EBNF
  - Nonterminals
    - Square or rectangle boxes
  - Terminals
    - round or oval boxes

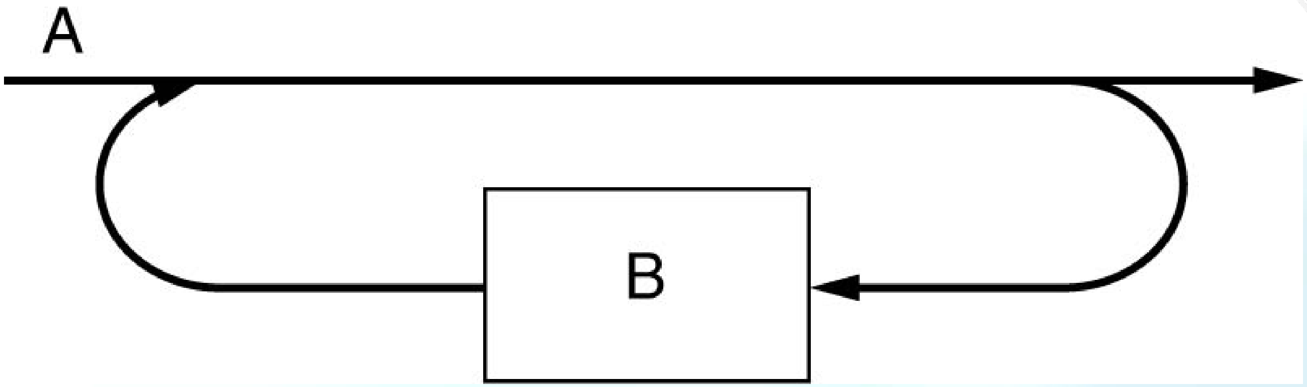
Syntax Diagrams

- *factor* → (*exp*) | ***number***



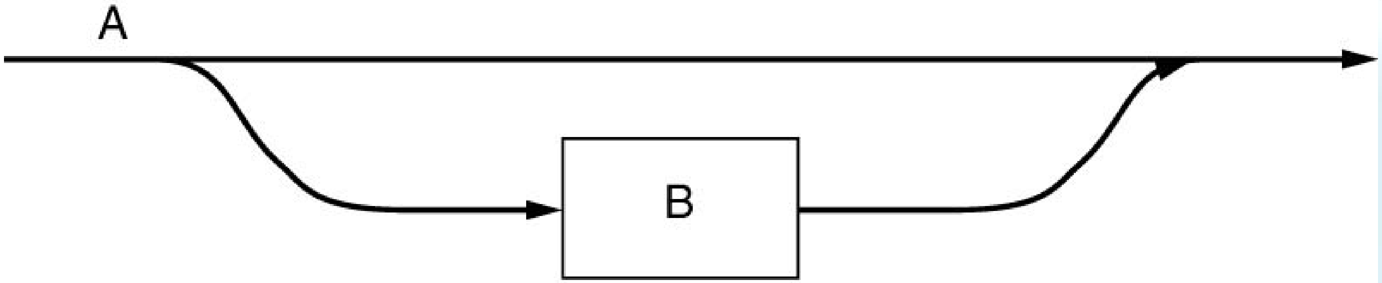
Syntax Diagrams

- $A \rightarrow \{\, B \,\}$



Syntax Diagrams

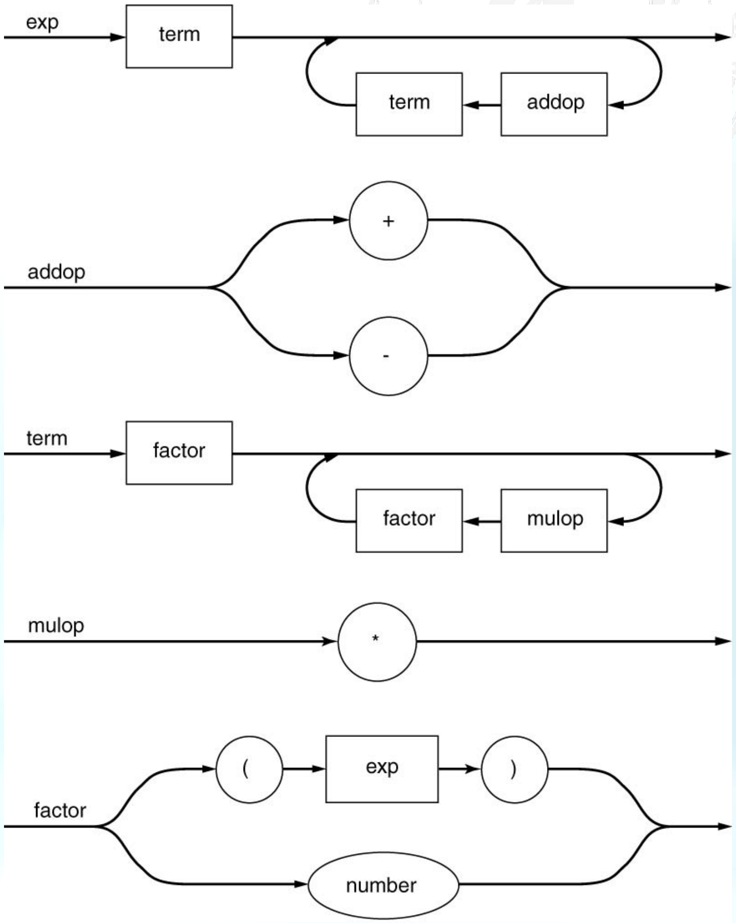
- $A \rightarrow [\, B \,]$



Syntax Diagrams

- **BNF**
  - *exp* → *exp addop term* | *term*
  - *addop* → **+** | **-**
  - *term* → *term mulop factor* | *factor*
  - *mulop* → *
  - *factor* → (*exp*) | ***number***

- **EBNF**
  - *exp* → *term* {*addop term*}
  - *addop* → **+** | **-**
  - *term* → *factor* {*mulop factor*}
  - *mulop* → *
  - *factor* → (*exp*) | ***number***



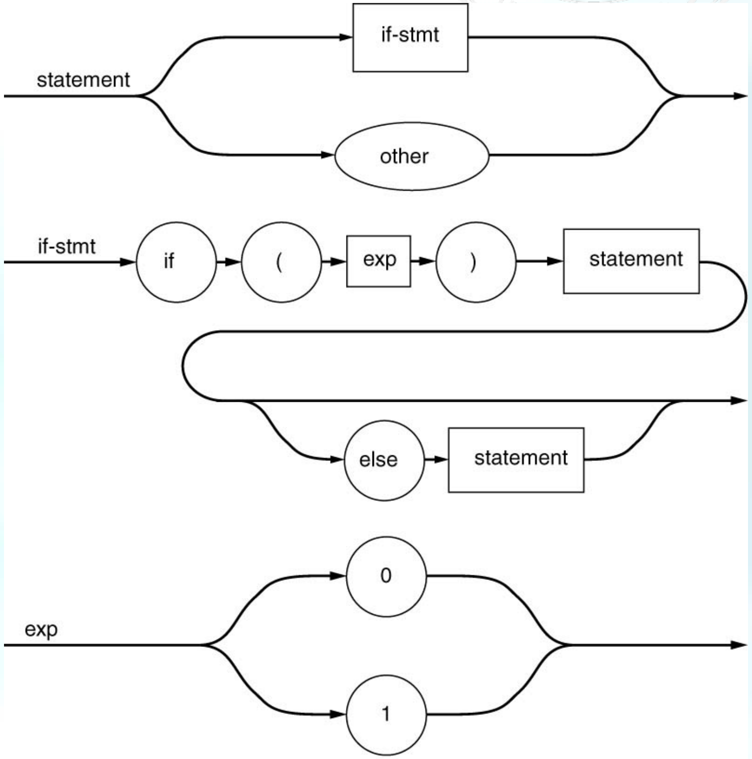
Syntax Diagrams

- **BNF**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement*
    | **if (** *exp* **)** *statement* **else** *statement*
  - *exp* → **0 | 1**

- **EBNF**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement* [**else** *statement*]
  - *exp* → **0 | 1**

Syntax Diagrams

- **EBNF**
  - *statement* → *if-stmt* | **other**
  - *if-stmt* → **if (** *exp* **)** *statement* [**else** *statement*]
  - *exp* → **0** | **1**



Syntax of the TINY language

program → *stmt-sequence*

stmt-sequence → *stmt-sequence* ; *statement* | *statement*

statement → *if-stmt* | *repeat-stmt* | *assign-stmt* | *read-stmt* | *write-stmt*

if-stmt → **if** *exp* **then** *stmt-sequence* **end**

| **if** *exp* **then** *stmt-sequence* **else** *stmt-sequence* **end**

repeat-stmt → ***repeat*** *stmt-sequence* **until** *exp*

assign-stmt → **identifier** **:=** *exp*

read-stmt → **read** **identifier**

write-stmt → **write** *exp*

exp → *simple-exp* *comparison-op* *simple-exp* | *simple-exp*

comparison-op → **<** | **=**

simple-exp → *simple-exp* *addop* *term* | *term*

addop → **+** | **-**

term → *term* *mulop* *factor* | *factor*

mulop → * | **/**

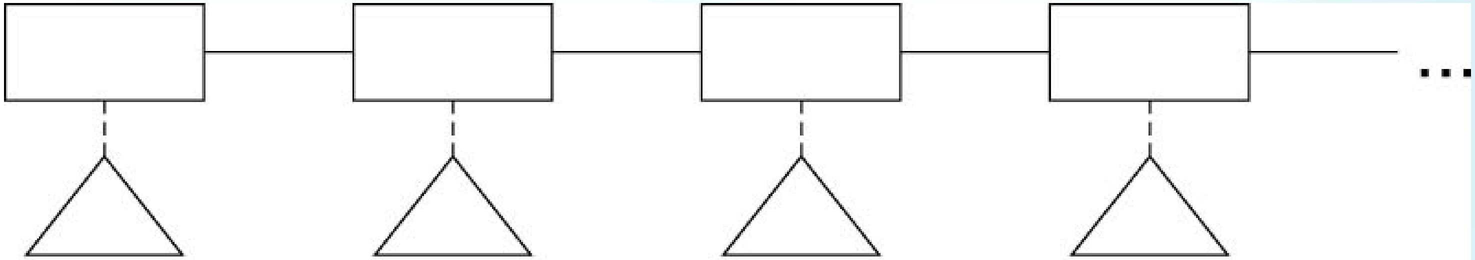
factor → **(**_exp_**)** | **number** | **identifier**

Syntax of the TINY language

- **A context-free grammar for TINY (Fig. 3.6)**
  - 1: program is a statement sequence.
  - 2: A statement sequence is a list of statement separated by ;.
  - 3: There are five kinds of statements.
  - 4-8: if, repeat, assign, read, and write statements.
  - 9-15: expressions
  - Precedence: ***, /** > **+,-** > **<, =**

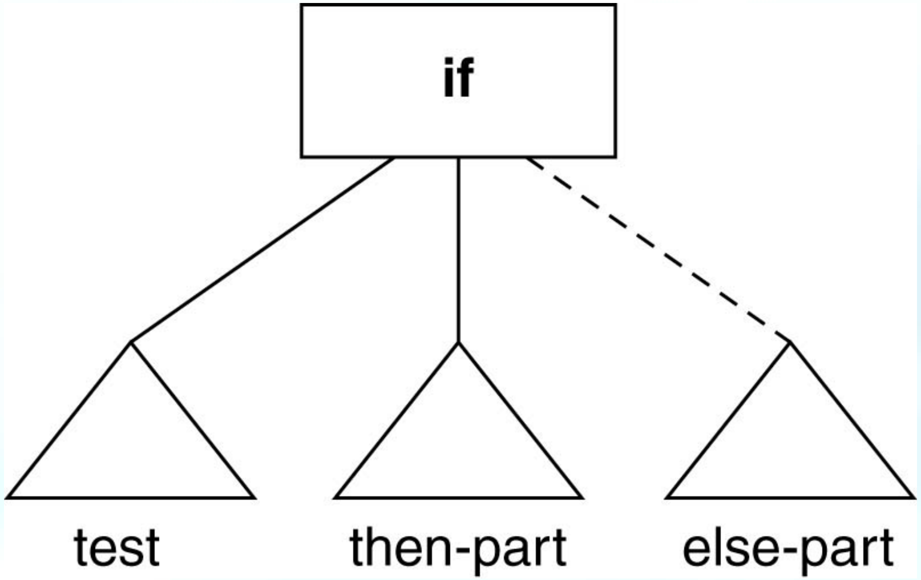
Syntax tree for the TINY

- **statement sequence**
  - *stmt-sequence* → *stmt-sequence* ; *statement* | *statement*



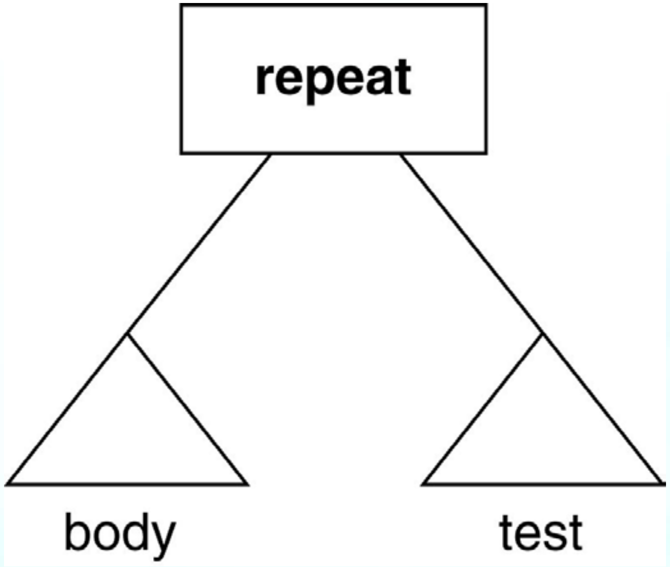
Syntax tree for the TINY

- **if statement**
  - *if-stmt* → **if** *exp* **then** *stmt-sequence* **end**
    | **if** *exp* **then** *stmt-sequence* **else** *stmt-sequence* **end**



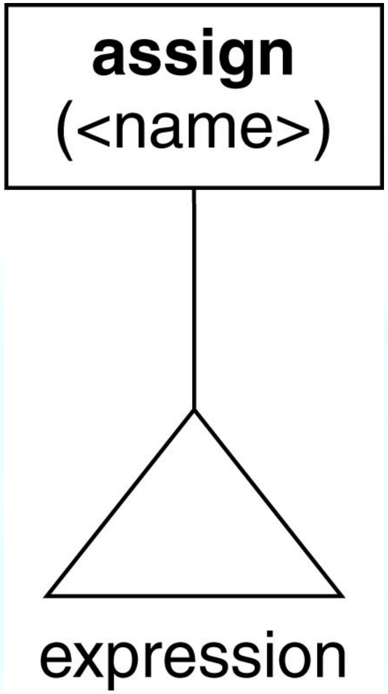
Syntax tree for the TINY

- **repeat statement**
  - *repeat-stmt* → **repeat** *stmt-sequence* **until** *exp*



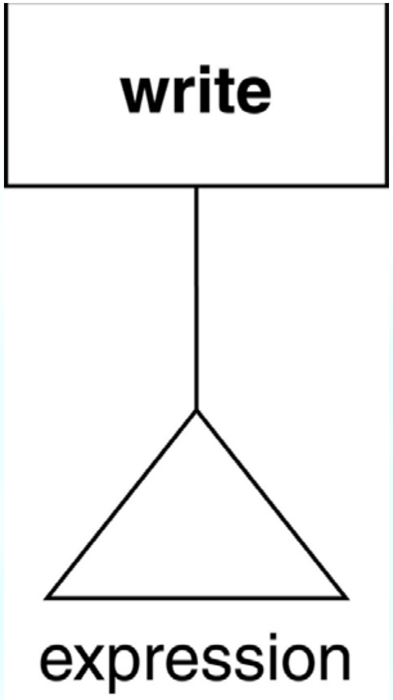
Syntax tree for the TINY

- **assign statement**
  - *assign-stmt* → ***identifier*** **:=** *exp*



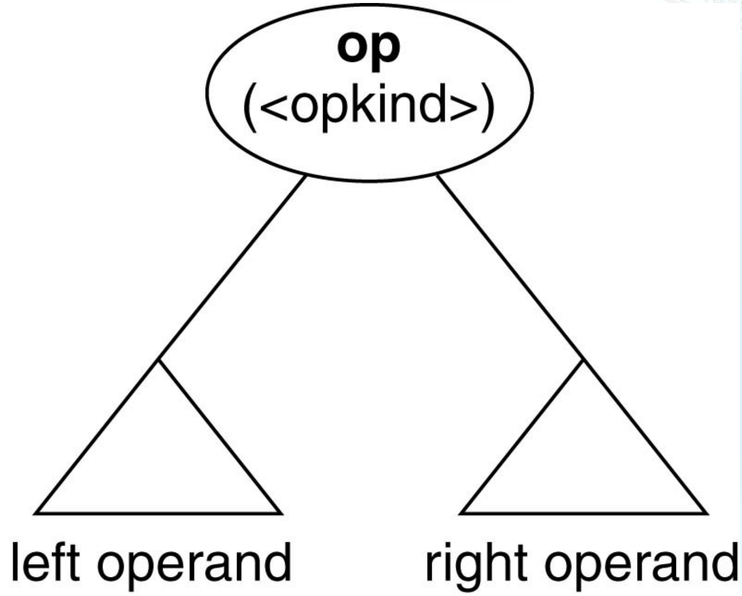
Syntax tree for the TINY

- **write statement**
  - *write-stmt* → **write** *exp*



Syntax tree for the TINY

- **operation**
  - *comparison-op* → < | =
  - *addop* → + | -
  - *mulop* → * | /



Syntax tree for the TINY

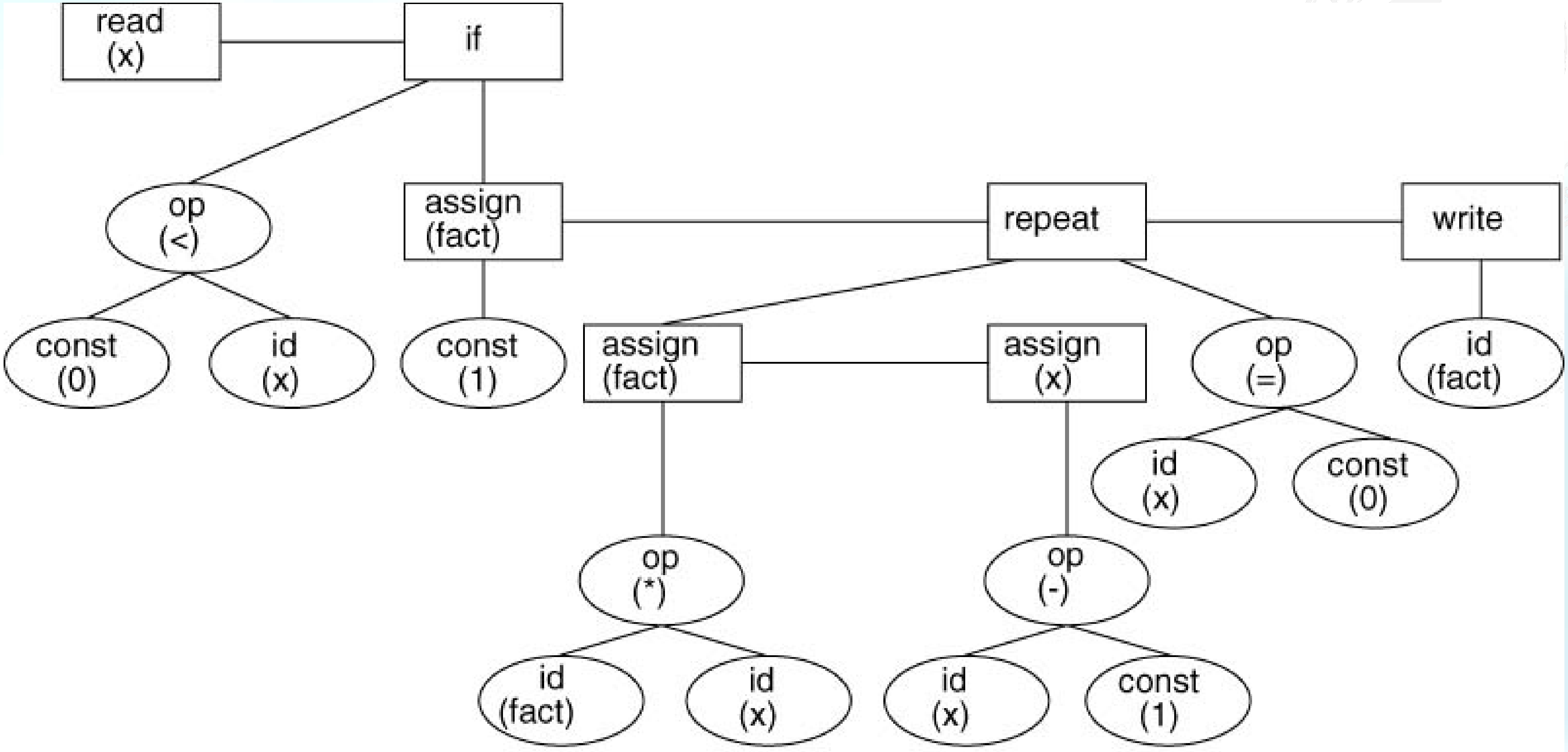
- **other statements?**
  - *read-stmt* → **read *identifier***
  - *addop* → **+ | -**
  - *mulop* → *** | /**
  - No syntax trees for grammar rules with only nonterminals on the right hand.

Syntax tree for the TINY

- Sample program in TINY language

```
{   Sample program
    in TINY language -
    computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

Syntax tree for the TINY


read
(x)
if
op
(<)
assign
(fact)
repeat
write
const
(0)
id
(x)
const
(1)
assign
(fact)
assign
(x)
op
(=)
id
(fact)
id
(x)
const
(0)
op
(*)
op
(-)
id
(fact)
id
(x)
id
(x)
const
(1)