

# Compiler Design

- Lexical Analysis
- Syntactical Analysis
- Semantic Analysis
- Optimization
- Code Generation

# Compilation Process

- Start with source code
- Perform analysis
- Generate object code
- Execute code from start to finish on hardware or software interpreter
- Note that execution is not a part of the compilation process

# Interpretation

- Start with source code
- Analyze, generate code, and execute code line-by-line.

# Compilation vs Interpretation

- Compilation generates object code file. Interpretation does not.
- Compile high-level code (code requiring lots of analysis). Interpret low-level code (code requiring very little analysis).
- Compilation is consistent with static scoping. Interpretation is consistent with dynamic scoping.

# Compilation vs Interpretation (continued)

- Compilation is non-interactive.  
Interpretation is interactive.
- Compilation is used with imperative/  
procedural languages (FORTRAN, Ada,  
Pascal, C, C++, Java, etc.) Interpretation is  
used with declarative languages (Prolog)  
and functional languages (LISP, Scheme,  
Hope, Miranda, ML, FP, Haskell, etc.

# Lexical Analysis

if (a>b) x=1; else y =4;

if, else – reserved word tokens

(, ), ; – special character tokens

a, b, x, y – ID tokens

> – relop token

1, 4 – num tokens

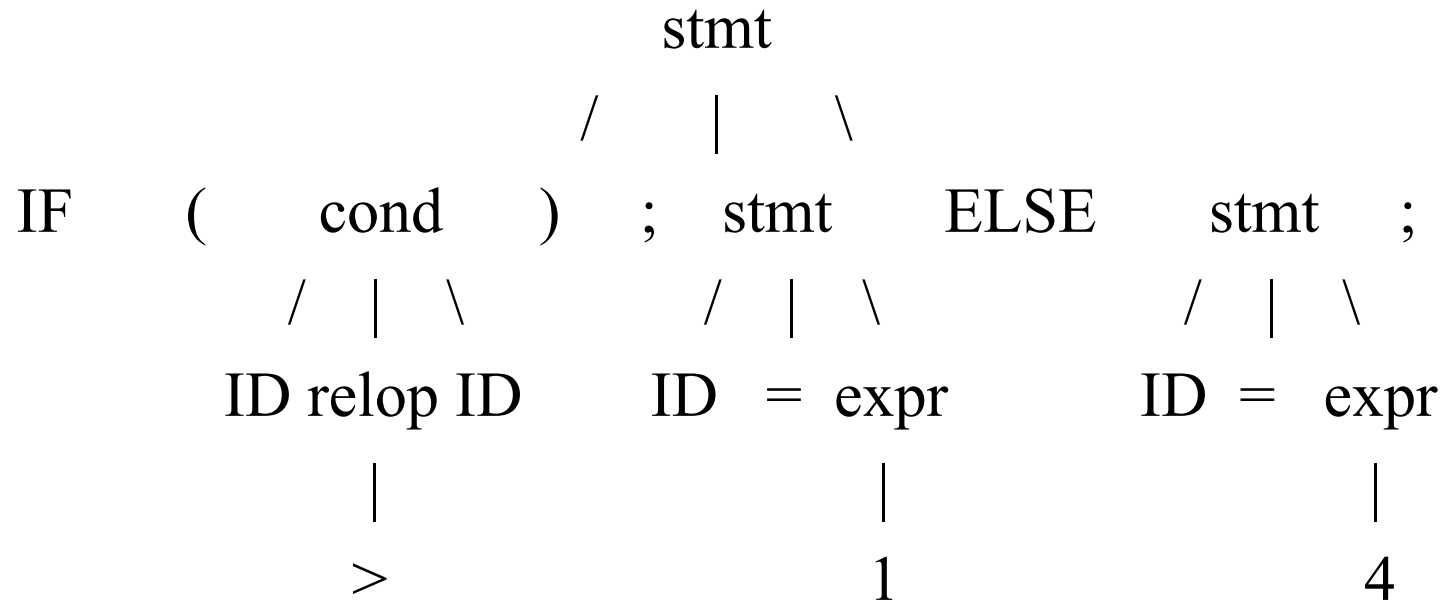
# Syntactical Analysis

stmt  $\rightarrow$  IF ( cond ) stmt  
| IF (cond) stmt ELSE stmt  
| ID = expr

cond  $\rightarrow$  ID RELOP ID

expr  $\rightarrow$  NUM | ID | ID OP ID

# Parse Tree for IF Statement



# Grammar Definition

Grammar  $G = (N, \Sigma, S, P)$

Where:      $N$  = set of non-terminals  
               $\Sigma$  = set of tokens/terminals  
               $S$  = start non-terminal  
               $P$  = set of productions

# Grammar Example

Grammar  $G_1 =$

$N = \{ \langle \text{sentence} \rangle, \langle \text{subject} \rangle, \langle \text{predicate} \rangle \}$

$\Sigma = \{ \text{HE}, \text{SHE}, \text{RAN}, \text{JUMPED} \}$

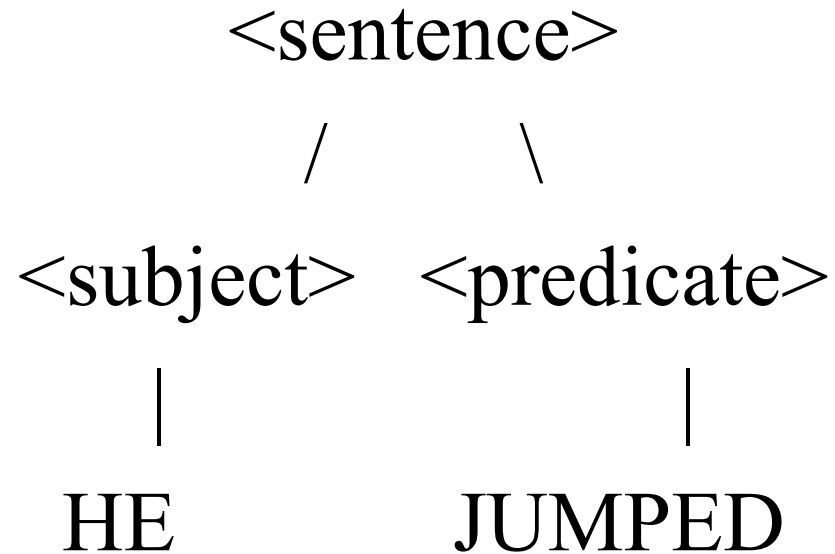
$S = \langle \text{sentence} \rangle$

$P = \langle \text{sentence} \rangle \rightarrow \langle \text{subject} \rangle \langle \text{predicate} \rangle$

$\langle \text{subject} \rangle \rightarrow \text{HE} \mid \text{SHE}$

$\langle \text{predicate} \rangle \rightarrow \text{RAN} \mid \text{JUMPED}$

# Parse Tree



# Language Definition

$$\text{Language } L = \{ s \mid S \overset{*}{\rightarrow} s \}$$

Where:

$S$  = Grammar Goal Symbol

$s$  = string of tokens/terminals

# Language Example

For Grammar  $G_1$

Language  $L_1 =$  HE RAN  
SHE RAN  
HE JUMPED  
SHE JUMPED

# Ambiguous Grammar

$G_2 =$

$$N = \{E\}$$

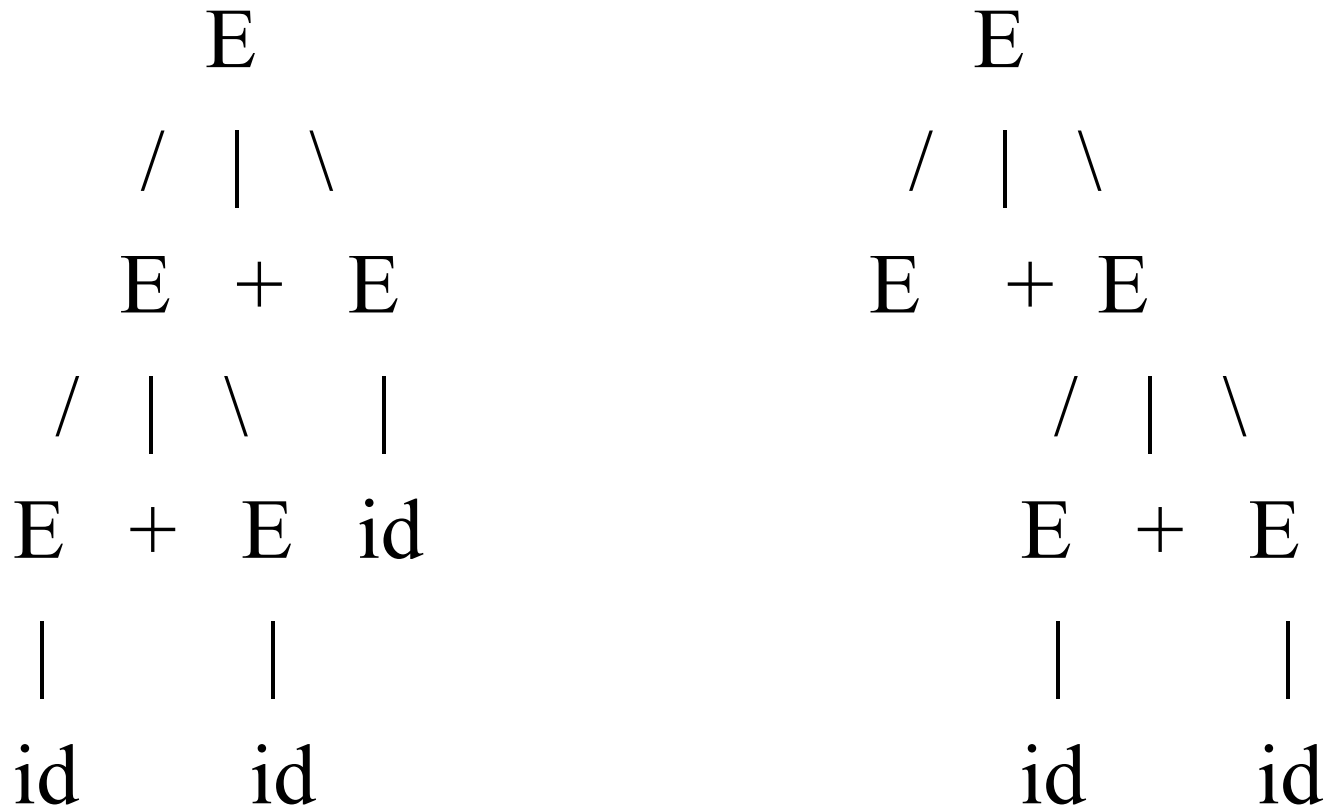
$$\Sigma = \{ID, (, ), +, *\}$$

$$S = E$$

$$P = E \rightarrow E + E \mid E * E \mid (E) \mid ID$$

# Parse Trees for $G_2$

Parse trees for:  $\text{id} + \text{id} + \text{id}$



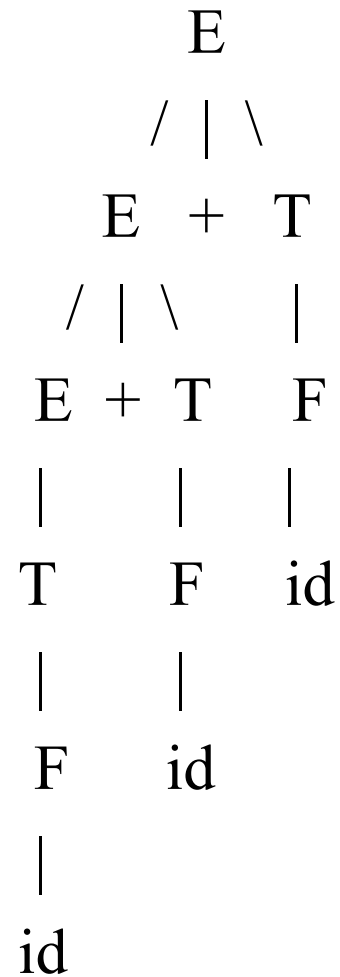
# Unambiguous Grammar

$$G_3 = E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow ( E ) \mid \text{id}$$

# Parse Tree for $G_3$



# Chomsky Grammar Hierarchy

- Type 0 – Unrestricted
- Type 1 – Context Sensitive
  - Production type:  $\beta A \delta \rightarrow \beta \alpha \delta$
  - Equivalent to: Spoken Languages
- Type 2 – Context Free
  - Production type:  $A \rightarrow \alpha$
  - Equivalent to: Parser, Push-Down Automata
- Type 3 – Regular
  - Production Type:  $A \rightarrow a B \mid b$  or  $A \rightarrow B a \mid b$
  - Equivalent to: Scanner, Finite Automata, Regular Expressions

# Context Sensitivities

Fire

Time flies like an arrow