COMP3131/9102: Programming Languages and Compilers

Jingling Xue

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
`http://www.cse.unsw.edu.au/~cs3131`
`http://www.cse.unsw.edu.au/~cs9102`

The FAQs for the Course

- Why do you use the harmonic mean?
- Could you provide more test cases?
- Could you provide a binary reference implementation for each assignment beforehand?

ANSWERS: `http://www.cse.unsw.edu.au/~cs3131/Info/index.html`

Assignment 1 Feedback

- FAQs: your questions may have been answered there
- Whitespace serves to separate token but not all tokens need to be separated by whitespace (e.g., 1+2)
- getToken(): always returns the longest prefix (starting from the current char) that can be interpreted as a token

Tutorials

- Starting this week
- Questions and answers available in the subject home page
- The tutor: Dr. Samad Saadatmand Feyzrasa

Course Map

1. Lexical analysis $\Longrightarrow$ Assignment 1 $\surd$
   - crafting a scanner by hand $\surd$
   - regular expressions, NFA and DFA $\surd$ // More in Week 9
   - scanner generator (e.g.,, lex and JLex) // Introduced in Week 9
2. Context-free grammars
3. Syntactic analysis
   - abstract syntax trees (ASTs)
   - recursive-descent parsing and LL(k)
   - bottom-up parsing and LR(k)
   - Parser generators (e.g., yacc, JavaCC and JavaCUP)
4. Semantic analysis
   - symbol table
   - identification (i.e., binding)
   - type checking
5. Code generation
   - syntax-directed translation
   - Jasmin assembly language
   - Java Virtual Machines (JVMs)

Week 2 (1st Lecture): Context-Free Grammars, Languages and Parsing

1. The syntax and semantics of a programming language
2. Specify a language's syntax: CFG, BNF and EBNF
3. The parsing of a program in a language:
   - Construct leftmost and rightmost derivations
   - Construct parse trees
4. The structure of a grammar:
   - How language constructs are defined
   - The precedence and associativity of operators
   - Ambiguity
5. The Chomsky hierarchy

English: Syntax and Semantics

1. **John eats apples**
2. **Apples eat John**

- Syntax:
  - The form or structure of English sentences
  - No concern with the meaning of English sentences
  - Specified by the English grammar
- Semantics:
  - The meaning of English sentences
  - How is the English semantics defined?

Programming Languages: Syntax and Semantics

1. **i = i + 1;**
2. **if (door.isOpen()) System.out.println("hello");**

- Syntax:
  - The form or structure of programs
  - No concern with the meaning of programs
  - Specified by a context-free grammar (CFG)
- Semantics:
  - The meaning of programs
  - Specified by
    * operational, denotational or axiomatic semantics,
    * attribute grammars (Week 7), or
    * an informal English description as in C, Java and VC

Programming Languages: Syntax and Semantics

- Syntax:
  - The form or structure of a program and individual statements in the language
  - No concern with the meaning of a program
  - Specified by a CFG (universally used in compiler construction)
- Semantics:
  - Static Semantics: Context-sensitive restrictions enforced at compile-time
    * All identifiers declared before used
    * Assignment must be type-compatible
    * Operands must be type-compatible with operators
    * Methods called with the proper number of arguments
    * Assignment 4: context-sensitive handling for static semantics
  - Run-Time Semantics: What the program does or computes
    * The meaning of a program or what happens when it is executed.
    * Specified by code generation routines.

Static Semantics: Undeclared Variables

```
public  class Foo {
  public static void main(String argv[]) {
     i = 10;
  }
}
javac Foo.java
Foo.java:3: Undefined variable: i
     i = 10;
     ^
1 error
```

- Grammatical
- But has a semantic error: undeclared variable

Static Semantics: Assignment Incompatible

```
public  class Foo {
  public static void main(String argv[]) {
     int i;
     float f = 10;
     i = f;
  }
}
javac Foo.java
Foo.java:5: Incompatible type for =.
Explicit cast needed to convert float to int.
     i = f;
       ^
1 error
```

- Grammatical
- Semantic error: assignment incompatible

Static Semantics: Operands with Incompatible Types

```
public  class Foo {

  public static void main(String argv[]) {
      int i = 1 + main;
  }
}
javac Foo.java
Foo.java:4: Reference to method main in class Foo
as if it were a variable.
      int i = 1 + main;
                  ^
1 error
```

- Grammatical
- Semantic error: incompatible operand type

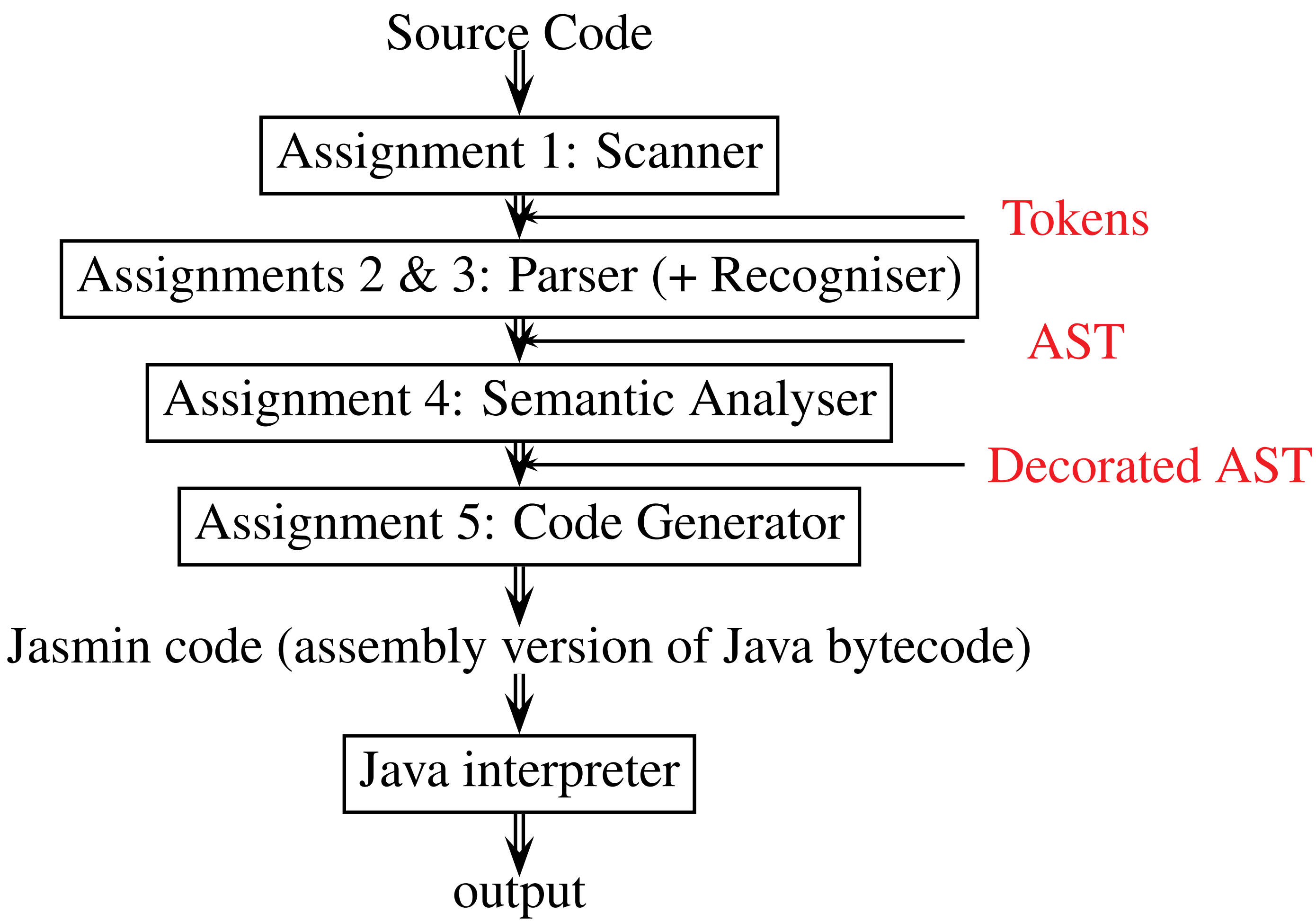
Static Semantics: Wrong Number of Arguments

```
public  class Foo {
  void sub(int i) { };

  public static void main(String argv[]) {
      (new Foo()).sub(1, 2);
  }
}
javac Foo.java
Foo.java:5: Wrong number of arguments in method.
      (new Foo()).sub(1, 2);
                     ^
1 error
```

- Grammatical
- Semantic error: wrong number of arguments

The VC Compiler: The Function of Each Component



- Semantic analyser considered a misnomer by many
- Contextual handler/analyser preferred in some compiler literature

Lecture 3: Context-Free Grammars, Languages and Parsing

1. The syntax and semantics of a programming language $\surd$
2. Specify a language's syntax: CFG, BNF and EBNF
3. The parsing of a program in a language:
   - Construct leftmost and rightmost derivations
   - Construct parse trees
4. The structure of a grammar:
   - How language constructs are defined
   - The precedence and associativity of operators
   - Ambiguity
5. The Chomsky hierarchy

Why Grammars at All?

- Give a precise and easy-to-understand syntactic specification of the language.
- New language constructs added easily.
- Facilitate programming language modifications and extensions
- Allow the meaning of the corresponding language to be defined in terms of the grammar (i.e., syntactical structure)
- Scanners and parsers constructed easily.
  - In 1950s, the first FORTRAN took 18 man-years
  - Now, a compiler written by a student in a semester
- Enable syntax-directed translation (Assignment 5)

CFG

- One type of grammar for specifying a language's syntax.
- Simple, widely used, and sufficient for most purposes.
- Not the most powerful syntax description tool. Powerful grammars like context-sensitive grammars and phrase-structure grammars too complex to be useful.
- Regular grammars (i.e., regular expressions) are less powerful

In this course, only CFGs and regular grammars required

Formal Definition of CFG

A grammar G is a quadruple (V_T, V_N, S, P), where

- V_T: a finite set of terminal symbols or tokens
- V_N: a finite set of nonterminal symbols ($V_T \cap V_N = \emptyset$)
- S: a unique start symbol ($S \in N$)
- P: a finite set of rules or productions of the form (A, α) where:
  - A is a nonterminal, and
  - α is a string of zero or more terminals and nonterminals

Note: zero means that $\alpha = \epsilon$ is possible

Backus-Naur Form (BNF)

- A notation for writing a CFG
- To recognise P. Naur's contributions as editor of the ALGOL60 report and J.W. Backus for applying the notation to the first FORTRAN compiler.
- Each production (A, α) is written as:

$$A \to \alpha$$

where the arrow $\to$ means "is defined to be", "can have the form of", "may be replaced with" or "derives"

- Can abbreviate the left to the right

$$\boxed{\begin{array}{ccc} A & \to & \alpha_1 \\ A & \to & \alpha_2 \\ & \vdots & \\ A & \to & \alpha_n \end{array}} \implies \boxed{\begin{array}{ccc} A & \to & \alpha_1 \\ & \mid & \alpha_2 \\ & \vdots & \\ & \mid & \alpha_n \end{array}}$$

where:

- $\alpha_1, \ldots, \alpha_n$ are the alternatives of A
- the vertical bar $\mid$ reads "or else"

CFG for micro-English

$$
\begin{array}{lll}
1 & \langle\text{sentence}\rangle & \rightarrow \langle\text{subject}\rangle\ \langle\text{predicate}\rangle \\
2 & \langle\text{subject}\rangle & \rightarrow \textbf{NOUN} \\
3 & & \mid\ \textbf{ARTICLE NOUN} \\
4 & \langle\text{predicate}\rangle & \rightarrow \textbf{VERB}\ \langle\text{object}\rangle \\
5 & \langle\text{object}\rangle & \rightarrow \textbf{NOUN} \\
6 & & \mid\ \textbf{ARTICLE NOUN}
\end{array}
$$

The four components of a CFG:

- V_N: set of nonterminals:
  - The symbol on the left-hand side of $\rightarrow$
  - The names of language constructs in the language.
- V_T: set of terminals or tokens:
  - The basic language units, parallel to the words in natural languages.
- S: $\langle$sentence$\rangle$, i.e., the left-hand side of the 1st production
- P: set of productions or rules of the form: $A \rightarrow X_1 X_2 \cdots X_n$.
  - A: a nonterminal
  - X_i: a terminal (can be ϵ) or nonterminal.

Derivations; Sentential Forms; Sentences; Languages

A grammar derives sentences by

1. beginning with the start symbol, and
2. repeatedly replacing a nonterminal by the right-hand side of a production with that nonterminal on the left-hand side, until there are no more nonterminals to replace.

- Such a sequence of replacements is called a derivation of the sentence being analysed
- The strings of terminals and nonterminals appearing in the various derivation steps are called sentential forms
- A sentence is a sentential form with terminals only
- The language: the set of all sentences thus derived

Verify if
"PETER PASSED THE TEST"
is a sentence?

The Three Derivations of PETER PASSED THE TEST

$$
\begin{array}{llll}
\langle\text{sentence}\rangle & \Longrightarrow & \langle\text{subject}\rangle\ \langle\text{predicate}\rangle & \text{by P1} \\
 & \Longrightarrow & \textsf{\textbf{NOUN}}\ \langle\text{predicate}\rangle & \text{by P2} \\
 & \Longrightarrow & \textsf{\textbf{NOUN VERB}}\ \langle\text{object}\rangle & \text{by P4} \\
 & \Longrightarrow & \textsf{\textbf{NOUN VERB ARTICLE NOUN}} & \text{by P6}
\end{array}
$$

$$
\begin{array}{llll}
\langle\text{sentence}\rangle & \Longrightarrow & \langle\text{subject}\rangle\ \langle\text{predicate}\rangle & \text{by P1} \\
 & \Longrightarrow & \langle\text{subject}\rangle\ \textsf{\textbf{VERB}}\ \langle\text{object}\rangle & \text{by P4} \\
 & \Longrightarrow & \langle\text{subject}\rangle\ \textsf{\textbf{VERB ARTICLE NOUN}} & \text{by P6} \\
 & \Longrightarrow & \textsf{\textbf{NOUN VERB ARTICLE NOUN}} & \text{by P2}
\end{array}
$$

$$
\begin{array}{llll}
\langle\text{sentence}\rangle & \Longrightarrow & \langle\text{subject}\rangle\ \langle\text{predicate}\rangle & \text{by P1} \\
 & \Longrightarrow & \langle\text{subject}\rangle\ \textsf{\textbf{VERB}}\ \langle\text{object}\rangle & \text{by P4} \\
 & \Longrightarrow & \textsf{\textbf{NOUN VERB}}\ \langle\text{object}\rangle & \text{by P2} \\
 & \Longrightarrow & \textsf{\textbf{NOUN VERB ARTICLE NOUN}} & \text{by P6}
\end{array}
$$

- Sentence: **NOUN VERB ARTICLE NOUN**
- Sentential forms: all the others

Leftmost and Rightmost Derivations

At each step in a derivation, two choices are made:

1. Which nonterminal to replace?
2. Which alternative to use for that nonterminal?

- Two types of useful derivations:
  - Leftmost derivation: always replace the leftmost nonterminal.
  - Rightmost derivation: always replace the rightmost nonterminal.

Leftmost and Rightmost Derivations

$$
\begin{aligned}
\langle\text{sentence}\rangle &\Longrightarrow_{\text{lm}} \langle\text{subject}\rangle\ \langle\text{predicate}\rangle && \text{by P1}\\
&\Longrightarrow_{\text{lm}} \textbf{NOUN}\ \langle\text{predicate}\rangle && \text{by P2}\\
&\Longrightarrow_{\text{lm}} \textbf{NOUN VERB}\ \langle\text{object}\rangle && \text{by P4}\\
&\Longrightarrow_{\text{lm}} \textbf{NOUN VERB ARTICLE NOUN} && \text{by P6}
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{sentence}\rangle &\Longrightarrow_{\text{rm}} \langle\text{subject}\rangle\ \langle\text{predicate}\rangle && \text{by P1}\\
&\Longrightarrow_{\text{rm}} \langle\text{subject}\rangle\ \textbf{VERB}\ \langle\text{object}\rangle && \text{by P4}\\
&\Longrightarrow_{\text{rm}} \langle\text{subject}\rangle\ \textbf{VERB ARTICLE NOUN} && \text{by P6}\\
&\Longrightarrow_{\text{rm}} \textbf{NOUN VERB ARTICLE NOUN} && \text{by P2}
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{sentence}\rangle &\Longrightarrow \langle\text{subject}\rangle\ \langle\text{predicate}\rangle && \text{by P1}\\
&\Longrightarrow \langle\text{subject}\rangle\ \textbf{VERB}\ \langle\text{object}\rangle && \text{by P4}\\
&\Longrightarrow \textbf{NOUN VERB}\ \langle\text{object}\rangle && \text{by P2}\\
&\Longrightarrow \textbf{NOUN VERB ARTICLE NOUN} && \text{by P6}
\end{aligned}
$$

neither

The Language Defined by a Grammar

- The language defined by a grammar: all the sentences derived from the grammar.
- The language defined by the micro-English grammar:

NOUN VERB NOUN

NOUN VERB ARTICLE NOUN

ARTICLE NOUN VERB NOUN

ARTICLE NOUN VERB ARTICLE NOUN

Conventions for Writing CFGs

- Start symbol:
  - The left side of the first production
  - The letter S, whenever it appears
- Nonterminals:
  - lower-case ⟨italic⟩ names such as ⟨sentence⟩ and ⟨expr⟩
  - capital letters like A, B, C
- Terminals:
  - **boldface** names such as **ID** and **INTLITERAL**
  - digits and operators such as 1 and + (sometimes in double quotes)
  - lower-case letters such as a, b, c
  - Usually anything non-italic
- Strings of terminals: lower-case letters late in the alphabet such as, $u, v, \cdots, z$
- Mixtures of nonterminals and terminals: lower-case Greek letters, such as $\alpha, \beta, \gamma, \cdots$

Some Formal Notations about Derivations

- $\Longrightarrow$: derivation in one step (one production used)
- $\Longrightarrow^+$: derivation in one or more steps
  - $\langle$sentence$\rangle \Longrightarrow^+ \langle$subject$\rangle$ **VERB** $\langle$object$\rangle$
  - $\langle$sentence$\rangle \Longrightarrow^+$ **NOUN VERB ARTICLE NOUN**
- $\Longrightarrow^*$: derivation in zero or more steps:

$$\begin{array}{ll} \langle\text{sentence}\rangle & \Longrightarrow^* \langle\text{sentence}\rangle \\ \langle\text{subject}\rangle\ \langle\text{predicate}\rangle & \Longrightarrow^* \langle\text{subject}\rangle\ \langle\text{predicate}\rangle \end{array}$$

- The language $L(G)$ defined by a grammar G:

$$L(G) = \{w \mid S \Longrightarrow^+ w\}$$

- The context-free language (CFL): the language generated by a CFG

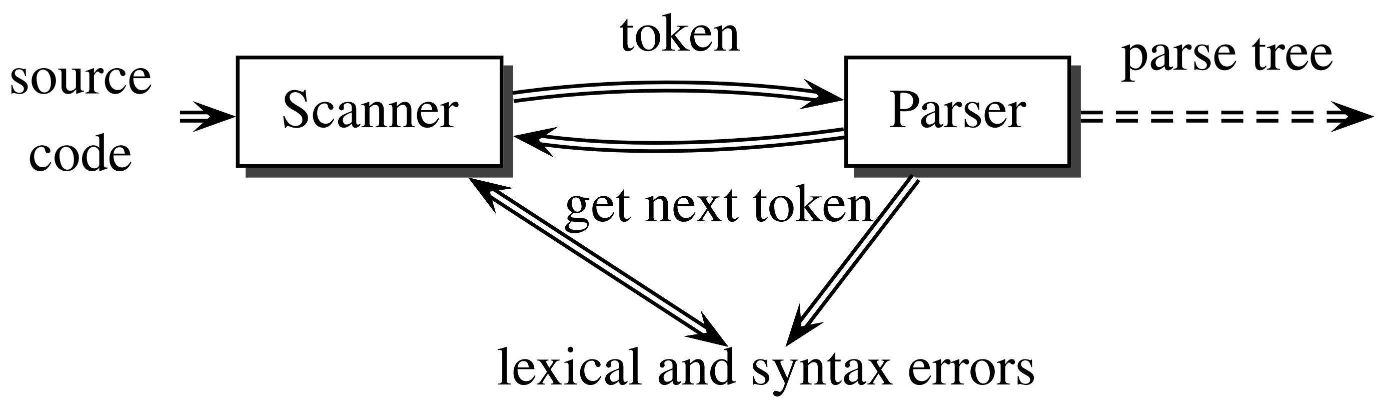
Lecture 3: Context-Free Grammars, Languages and Parsing

1. The syntax and semantics of a programming language $\surd$
2. Specify a language's syntax: CFG, BNF and EBNF $\surd$
3. The parsing of a program in a language:
   - Construct leftmost and rightmost derivations
   - Construct parse trees
4. The structure of a grammar:
   - How language constructs are defined
   - The precedence and associativity of operators
   - Ambiguity
5. The Chomsky hierarchy

The Parsing of a Sentence (or Program)

- Use syntactic rules to break a sentence into its component parts and analyse their relationship.
- The term parsing used in both linguistic and compiler theory.
- A parser is a program that uses a CFG to parse a sentence or a program (Assignment 3). In particular, it
  - constructs its leftmost or rightmost derivation, or
  - builds the parse tree for the sentence.
- A recogniser is a parser that checks only the syntax (without having to built the parse tree). It outputs YES if the program is legal and NO otherwise (Assignment 2).

The Role of the Parser



- Perform context-free syntactic analysis
- Construct a tree (an AST rather than a parse tree)
- Produce some meaningful error messages
- Attempt error recovery

Parsing: The Derivational View

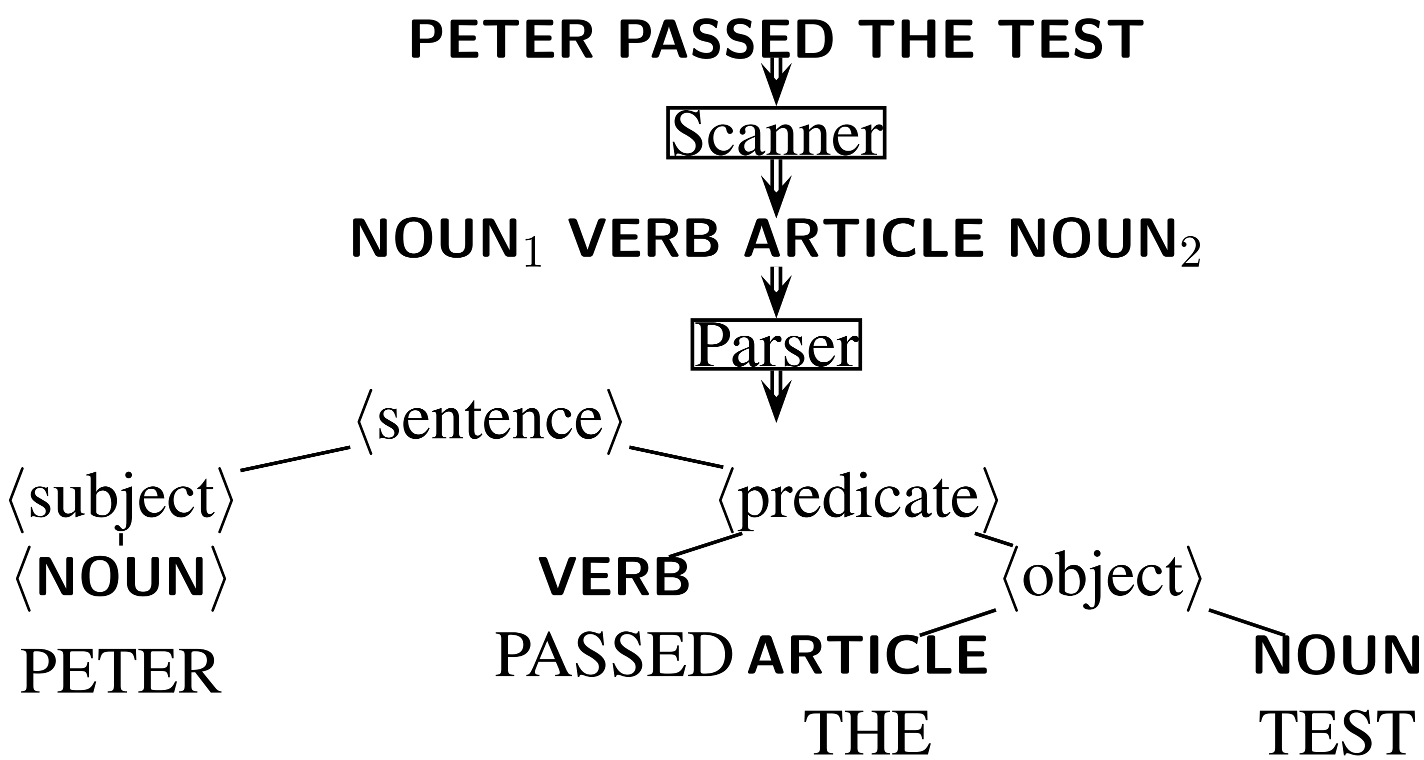
- Parsing: A process of constructing the leftmost or rightmost derivation of the sentence being analysed.
- **PETER PASSED THE TEST** $\stackrel{scanner}{\Longrightarrow}$ **NOUN**$_1$ **VERB ARTICLE NOUN**$_2$

$$
\begin{array}{rlr}
\langle \text{sentence} \rangle & \Longrightarrow_{\text{lm}} \langle \text{subject} \rangle\ \langle \text{predicate} \rangle & \text{by P1} \\
& \Longrightarrow_{\text{lm}} \textbf{NOUN}\ \langle \text{predicate} \rangle & \text{by P2} \\
& \Longrightarrow_{\text{lm}} \textbf{NOUN VERB}\ \langle \text{object} \rangle & \text{by P4} \\
& \Longrightarrow_{\text{lm}} \textbf{NOUN VERB ARTICLE NOUN} & \text{by P6}
\end{array}
$$

$$
\begin{array}{rlr}
\langle \text{sentence} \rangle & \Longrightarrow_{\text{rm}} \langle \text{subject} \rangle\ \langle \text{predicate} \rangle & \text{by P1} \\
& \Longrightarrow_{\text{rm}} \langle \text{subject} \rangle\ \textbf{VERB}\ \langle \text{object} \rangle & \text{by P4} \\
& \Longrightarrow_{\text{rm}} \langle \text{subject} \rangle\ \textbf{VERB ARTICLE NOUN} & \text{by P6} \\
& \Longrightarrow_{\text{rm}} \textbf{NOUN VERB ARTICLE NOUN} & \text{by P2}
\end{array}
$$

Parsing: Graphical Representation via Parse Trees

- Parsing: A process of constructing the parse tree for the sentence being analysed.



The Structure of Parse Trees

- The start symbol is always at the root of the tree.
- Nonterminals are always interior nodes.
- Terminals are always leaves in the tree.
- The sentence being analysed is the the leaves read from left to right.

Derivations v.s. Parse Trees

- The parsing of a sentence is to construct for the sentence
  - its leftmost or rightmost derivation, or
  - its parse tree
- The derivation and parse tree are two different views of the parsing of a sentence.
- The parse tree:
  - A graphical representation for a derivation.
  - The choice regarding to replacement order filtered out.

Summary So Far

- A language has two components: syntax and semantics.
  - Syntax: the form or structure of a program.
  - Semantics: the meaning of a program.
- A language's syntax is specified by a CFG.
- A CFG has four components.
- A BNF is a notation for writing a CFG.
- Parsing: discover a leftmost or rightmost derivation or build a parse tree
- Concepts
  - Sentential form
  - Sentence
  - Derivation: leftmost and rightmost
  - parse tree
  - Language and context-free language

Lecture 3: Context-Free Grammars, Languages and Parsing

1. The syntax and semantics of a programming language $\surd$
2. Specify a language's syntax: CFG, BNF and EBNF $\surd$
3. The parsing of a program in a language: $\surd$
   - Construct leftmost and rightmost derivations $\surd$
   - Construct parse trees $\surd$
4. The structure of a grammar:
   - How language constructs are defined
   - The precedence and associativity of operators
   - Ambiguity
5. The Chomsky hierarchy

Extended Backus-Naur Form (EBNF)

- EBNF = BNF + regular expressions
- (*something*)* means that the stuff inside can be repeated zero or more times:
- (*something*)$^+$ means that the stuff inside can be repeated one or more times:
- (*something*)? means that the stuff inside is optional
- The parentheses omitted if $\langle$something$\rangle$ is a single symbol
- More compact and readable than the BNF
- Convenient for writing recursive-descent parsers
- The VC grammar is given in the form of EBNF

An ENBF Example from the VC Grammar: Kleene Closure

- A VC program is a sequence of zero or more function/variable declarations
- The BNF productions:

$$
\begin{array}{lcl}
\textit{program} & \rightarrow & \textit{decl-list} \\
\textit{decl-list} & \rightarrow & \textit{decl-list func-decl} \\
 & | & \textit{decl-list var-decl} \\
 & | & \epsilon
\end{array}
$$

- The EBNF productions:

$$\textit{program} \rightarrow (\textit{func-decl} \mid \textit{var-decl})^*$$

An ENBF Example: Positive Closure

- A program is a sequence of one or more function/variable declarations
- The BNF productions:

$$\begin{array}{lcl} \textit{program} & \rightarrow & \textit{decl-list} \\ \textit{decl-list} & \rightarrow & \textit{decl-list func-decl} \\ & \mid & \textit{decl-list var-decl} \\ & \mid & \textit{func-decl} \\ & \mid & \textit{var-decl} \end{array}$$

- The EBNF productions:

$$\textit{program} \rightarrow (\textit{func-decl} \mid \textit{var-decl})^{+}$$

An ENBF Example from the VC Grammar: Optional Operator

- The if statement where the else-part is optional
- The BNF productions:

$$
\begin{array}{lcl}
\textit{stmt} & \rightarrow & \textsf{\textbf{IF}}\ \text{“(”}\ \textit{expr}\ \text{“)”}\ \textit{stmt} \\
 & | & \textsf{\textbf{IF}}\ \text{“(”}\ \textit{expr}\ \text{“)”}\ \textit{stmt}\ \textsf{\textbf{ELSE}}\ \textit{stmt} \\
 & | & \textsf{\textbf{other}}
\end{array}
$$

- The EBNF productions:

$$
\begin{array}{lcl}
\textit{stmt} & \rightarrow & \textsf{\textbf{IF}}\ \text{“(”}\ \textit{expr}\ \text{“)”}\ \textit{stmt}\ (\textsf{\textbf{ELSE}}\ \textit{stmt})? \\
 & | & \textsf{\textbf{other}}
\end{array}
$$

The Structure Of Grammars

- Top-Down Definition Of Language Constructs, as in VC:

```
program          -> ( func-decl | var-decl)*
func-decl        -> type identifier para-list compound-stmt
var-decl         -> type  init-declarator
type             -> void | boolean | int | float
compound-stmt    -> "{"  var-decl*  stmt*  "}"
stmt             -> compound-stmt
                 |   if-stmt
                     ...
                 |   expression-stmt
if-stmt          -> IF "("  expr ")"  stmt ( ELSE  stmt )?
expr-stmt        -> expr? ";"
expr             -> assignment-expr
assignment-expr  -> ...
```

- See the grammars for C (Kernighan and Ritchie's book) and Java (on-line)
- Bottom-Up Processing Of Language Constructs (Assignment 5). Roughly:
  - The deeper nodes in the parse tree processed first.
  - The deeper operators in the parse tree have higher precedence

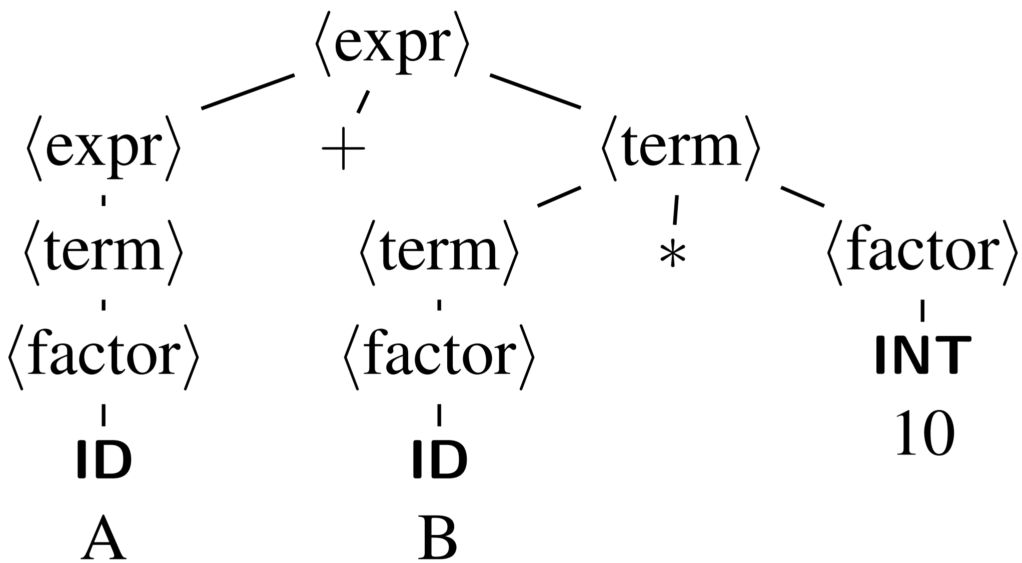
The Classic Expression Grammar

$$
\begin{array}{lrcl}
1 & \langle \text{expr} \rangle & \rightarrow & \langle \text{expr} \rangle + \langle \text{term} \rangle \\
2 & & \mid & \langle \text{expr} \rangle - \langle \text{term} \rangle \\
3 & & \mid & \langle \text{term} \rangle \\
4 & \langle \text{term} \rangle & \rightarrow & \langle \text{term} \rangle * \langle \text{factor} \rangle \\
5 & & \mid & \langle \text{term} \rangle \,/\, \langle \text{factor} \rangle \\
6 & & \mid & \langle \text{factor} \rangle \\
7 & \langle \text{factor} \rangle & \rightarrow & (\ \langle \text{expr} \rangle\ ) \\
8 & & \mid & \textbf{ID} \\
9 & & \mid & \textbf{INT}\ \text{// Note: integer numbers not the type}
\end{array}
$$

- Left-Recursive Productions: $A \rightarrow A\alpha$
- Right-Recursive Productions: $A \rightarrow \alpha A$

Operator Precedence: A + B ∗ 10

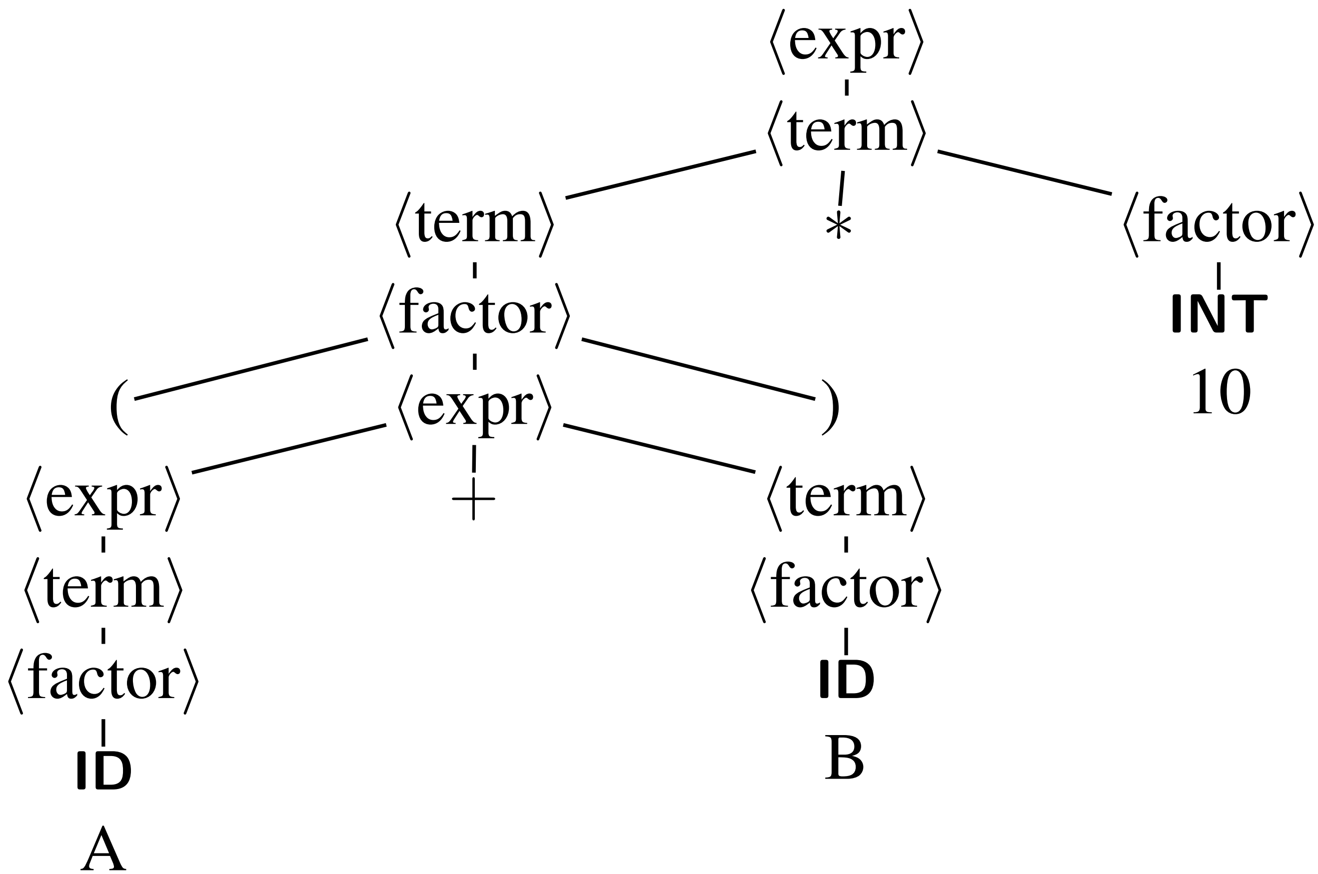
- Rules for binding operators to operands
- Higher precedence operators bind to their operands before lower precedence operators
- Higher precedence operators appear lower in the tree



- $A + B * 10$ evaluated as $A + (B * 10)$ as desired

Operator Precedence Changed by Parentheses: $(A + B) * 10$

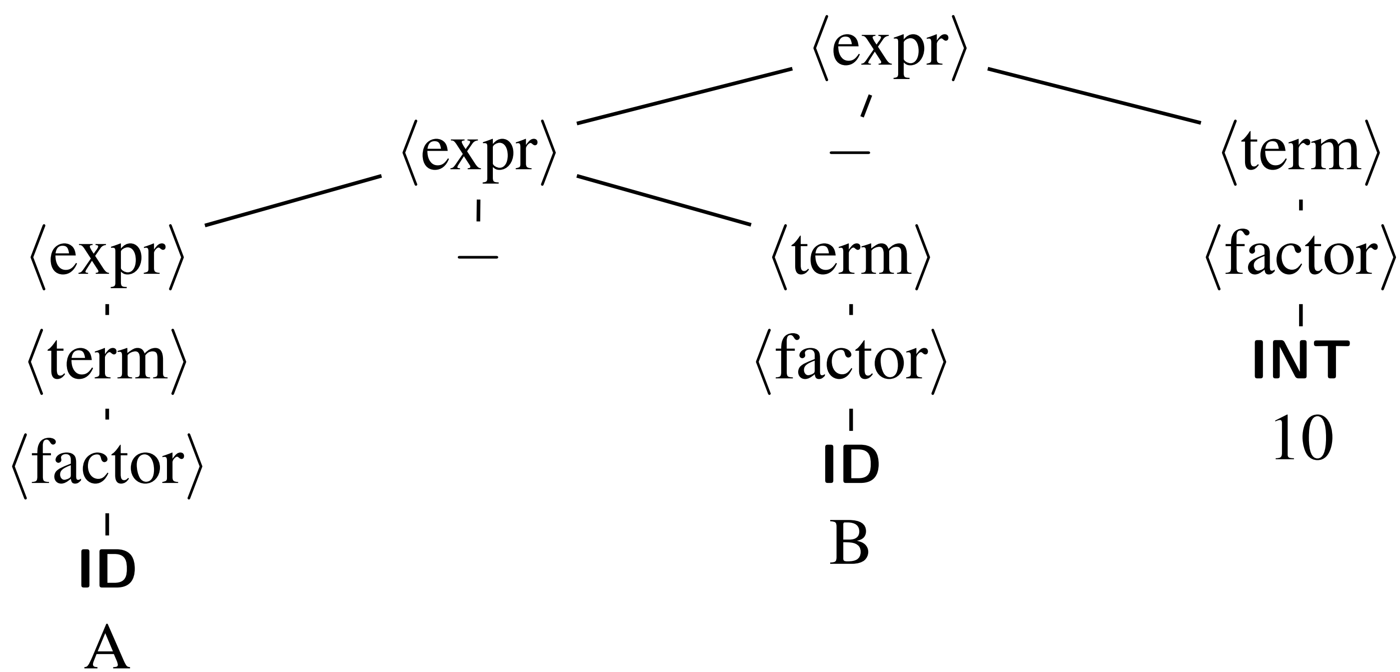
- $+$ appears lower than $*$ because of the use of (and):



- The addition will be evaluated first now

Operator Associativity: $A - B - 10$

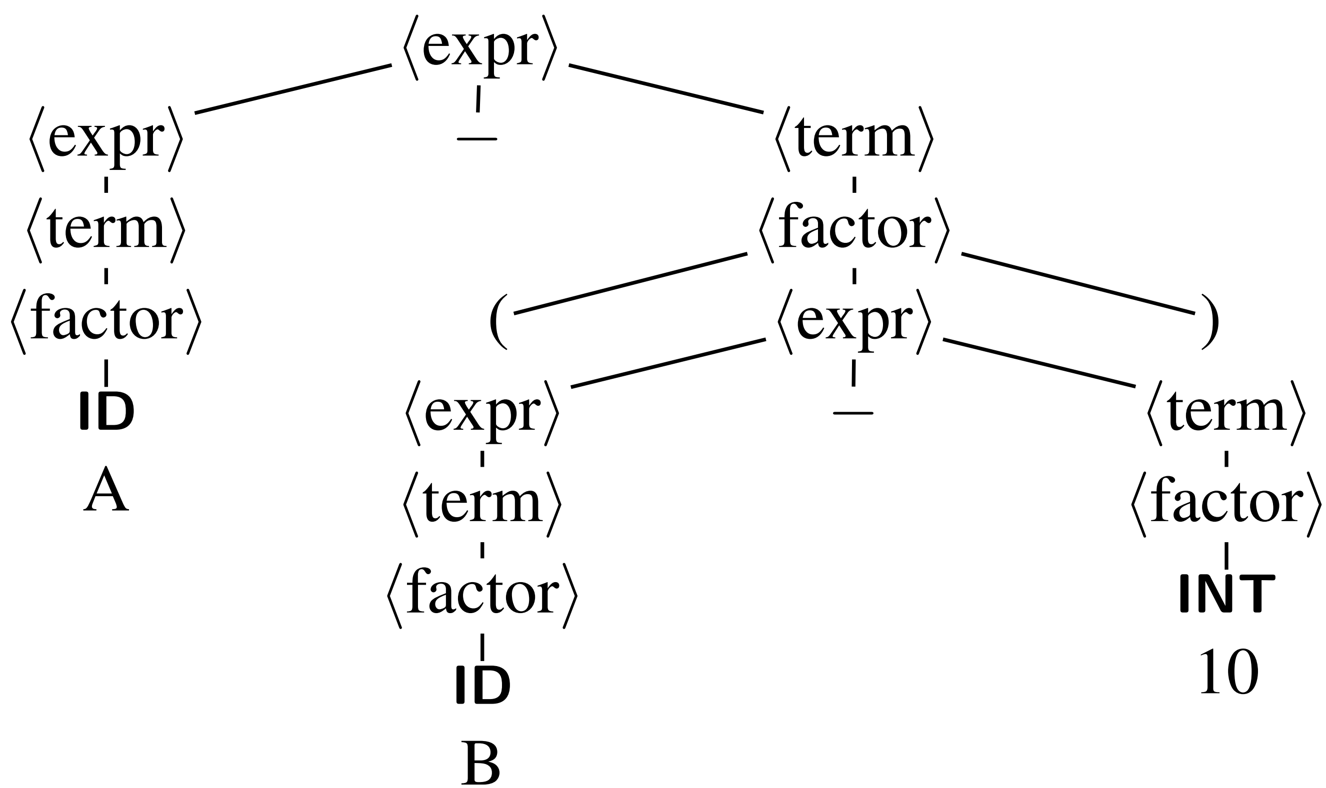
- Rules for grouping operators with equal precedence
- Given $\cdots - 1 - \ldots$, determines which $-$ takes the 1
- Left-recursive productions enforce left-associativity



- $A - B - 10$ evaluated as $(A - B) - C$ as desired

Operator Associativity Changed by Parentheses: $A - (B - 10)$

- The 2nd $-$ appears lower than the 1st $-$ in the tree:



- The 2nd subtraction will be evaluated first

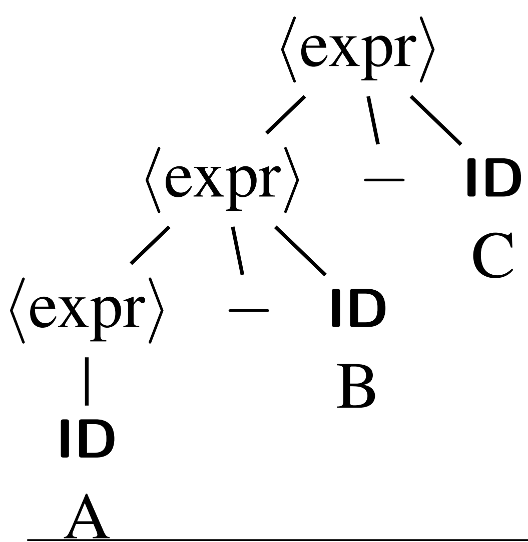
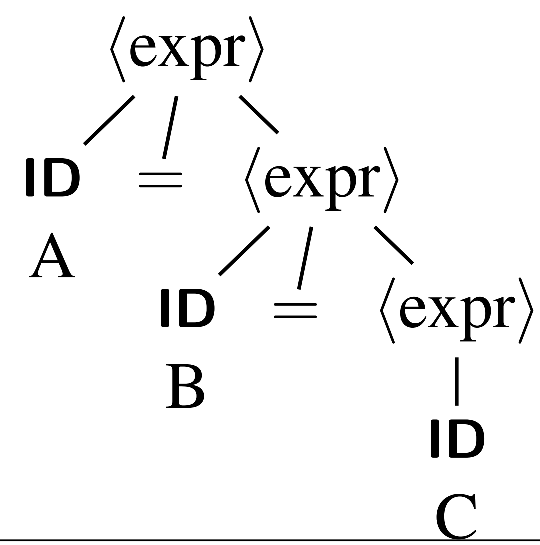
Operator Associativity: Summary

- A grammar consisting of left-recursive productions:

$$\langle\text{expr}\rangle \;\rightarrow\; \textbf{ID} \mid \langle\text{expr}\rangle - \textbf{ID}$$

- A grammar consisting of right-recursive productions:

$$\langle\text{expr}\rangle \;\rightarrow\; \textbf{ID} \mid \textbf{ID} = \langle\text{expr}\rangle$$

Parse tree of $A - B - C$	Parse tree of $A = B = C$
	

Precedence and Associativity Tables for Some Languages

C++: `en.cppreference.com/w/cpp/language/operator_precedence`

C: `https://en.cppreference.com/w/c/language/operator_precedence`

Java: `http://introcs.cs.princeton.edu/java/11precedence`

Python: `http://www.ibiblio.org/g2swap/byteofpython/read/operator-precedence.html`

Javascript: `http://www.scriptingmaster.com/javascript/operator-precedence.asp`

Quotes from Actual Medical Records

1. By the time he was admitted, his rapid heart had stopped, and he was feeling better.
2. Patient has chest pain if she lies on her left side for over a year.
3. On the second day the knee was better and on the third day it had completely disappeared.
4. The patient was tearful and crying constantly. She also appears to be depressed.
5. Discharge status: Alive but without permission. The patient will need disposition, and therefore we will get Dr. Blank to dispose of him.
6. Healthy appearing decrepit 69 year-old male, mentally alert but forgetful.
7. The patient refused an autopsy.

Ambiguous Grammars

- A grammar is ambiguous if it permits
  - more than one parse tree for a sentence,
    or in other words,
  - more than one leftmost derivation or more than one rightmost derivation for a sentence.
- An ambiguous expression grammar:

$$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \ \mid \textbf{ID} \ \mid \textbf{INT} \ \mid (\ \langle \text{expr} \rangle \ )$$

$$\langle \text{op} \rangle \quad \rightarrow \textbf{+} \mid - \mid * \mid \textbf{/}$$

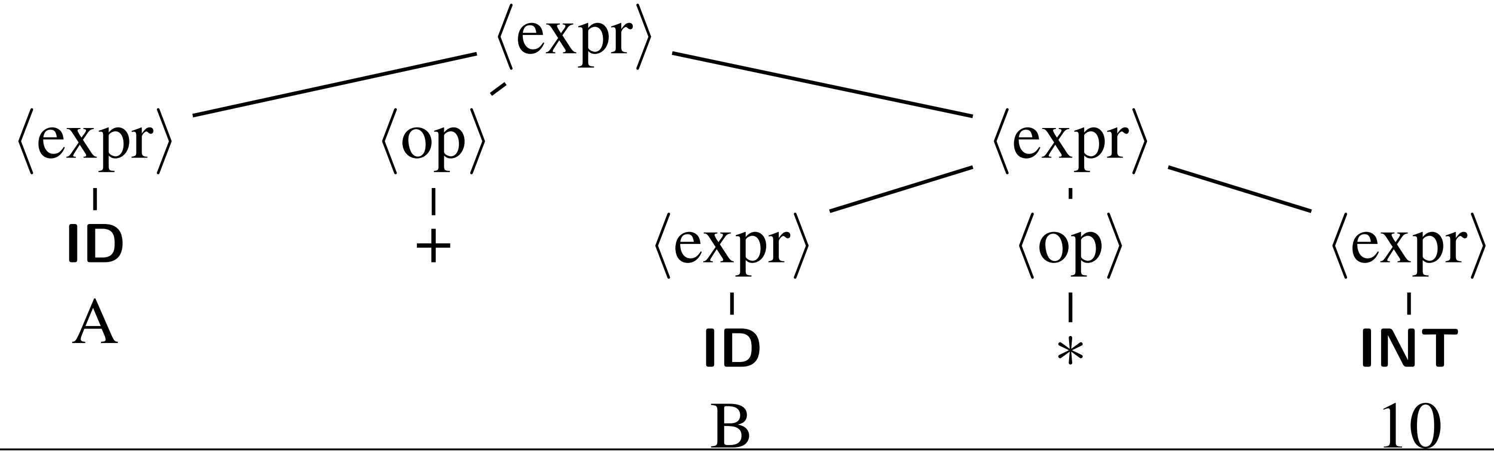
A + B $*$ 10: Two Distinct Leftmost Derivations

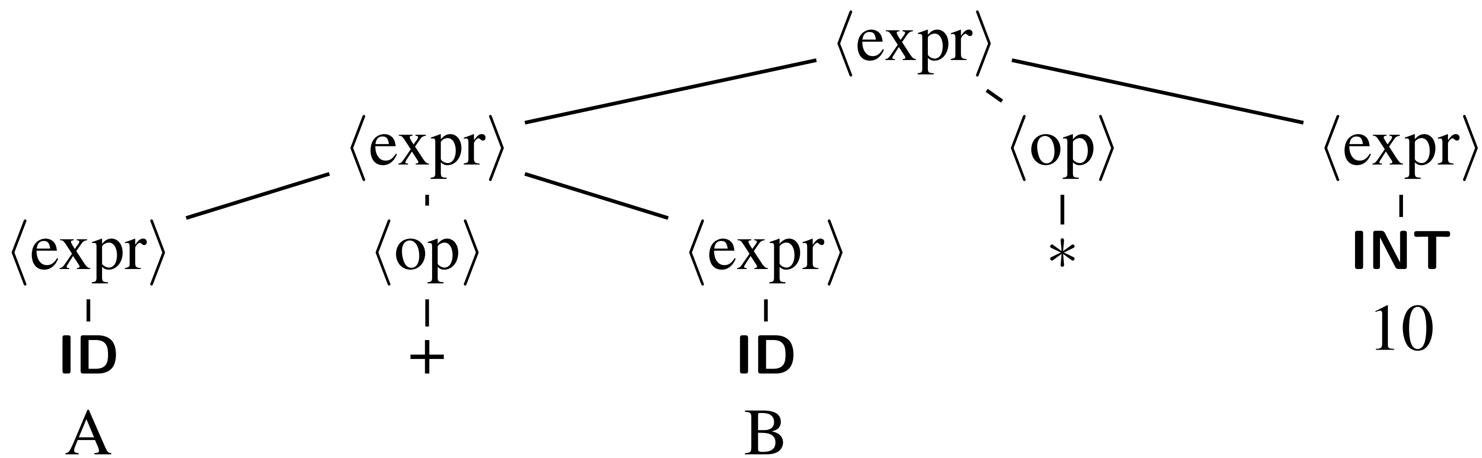
$$
\begin{aligned}
\langle \text{expr} \rangle &\Longrightarrow_{\text{lm}} \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \textbf{ID} \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \textbf{ID} * \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \textbf{ID} * \textbf{ID}
\end{aligned}
$$

$$
\begin{aligned}
\langle \text{expr} \rangle &\Longrightarrow_{\text{lm}} \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \textbf{ID} \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \textbf{ID} * \langle \text{expr} \rangle \\
&\Longrightarrow_{\text{lm}} \textbf{ID} + \textbf{ID} * \textbf{ID}
\end{aligned}
$$

Exercise: Find two distinct rightmost Derivations.

A + B $*$ 10: Two Distinct Parse Trees



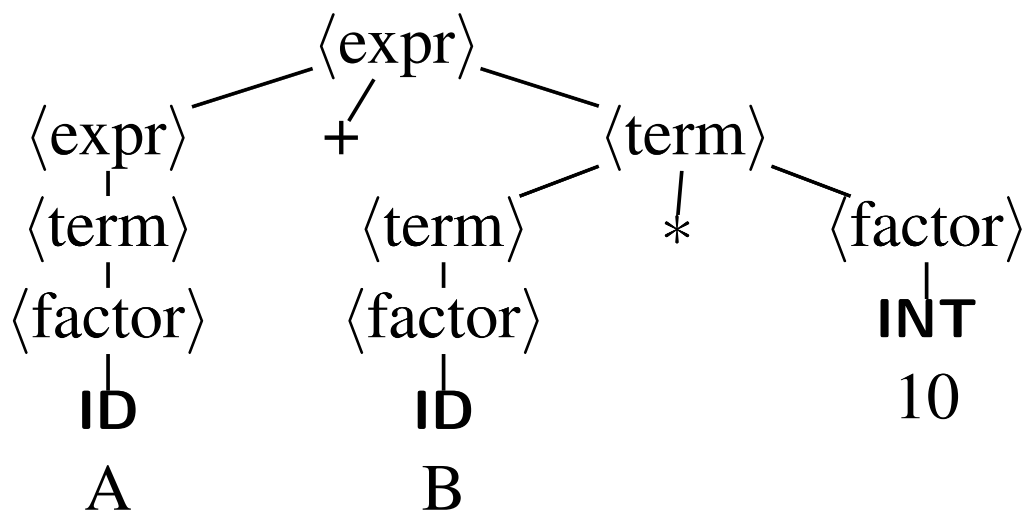


- The top tree means: $A + (B * 10)$
- The bottom tree means: $(A + B) * 10$

Coping With Ambiguous Grammars

- Method 1: Rewrite the grammar to make it unambiguous.

$$
\begin{aligned}
\langle\text{expr}\rangle &\rightarrow \langle\text{term}\rangle \mid \langle\text{expr}\rangle \; \textbf{+} \; \langle\text{term}\rangle \mid \langle\text{expr}\rangle - \langle\text{term}\rangle \\
\langle\text{term}\rangle &\rightarrow \langle\text{factor}\rangle \mid \langle\text{term}\rangle * \langle\text{factor}\rangle \mid \langle\text{term}\rangle \; \textbf{/} \; \langle\text{factor}\rangle \\
\langle\text{factor}\rangle &\rightarrow \textbf{ID} \mid \textbf{INT} \mid (\; \langle\text{expr}\rangle \;)
\end{aligned}
$$



- Un-ambiguous grammars preferred in practice

Coping With Ambiguous Grammars (Cont'd)

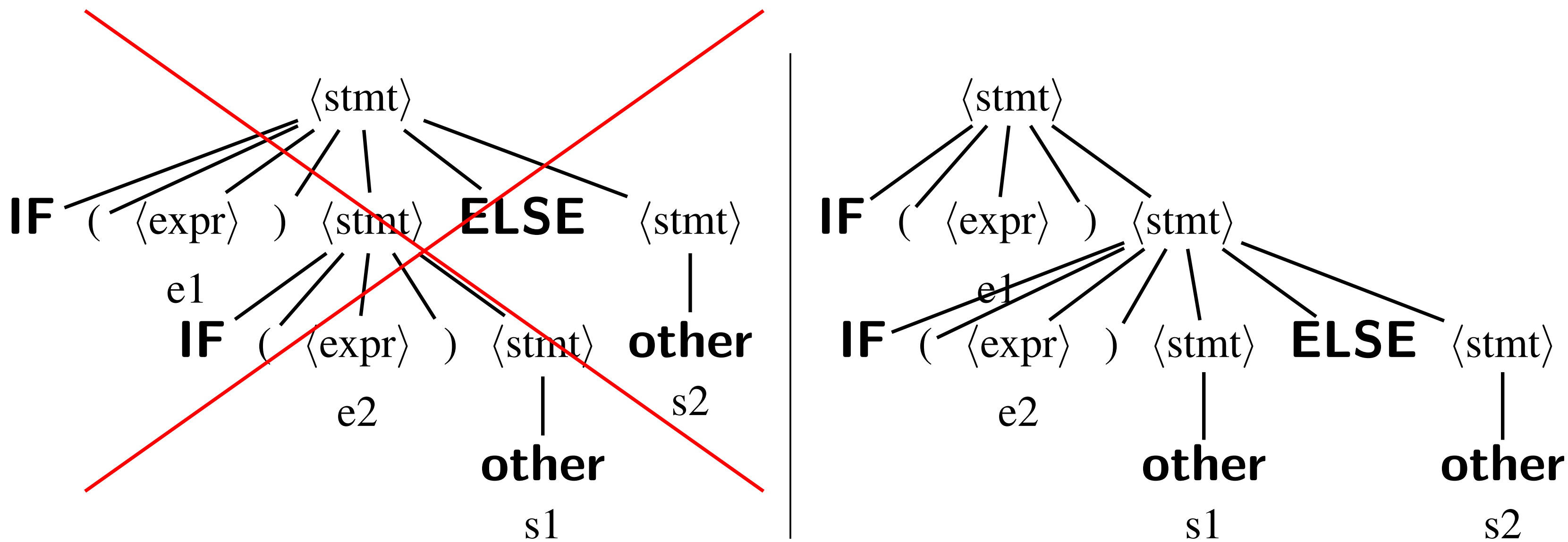
- Method 2: Use disambiguating rules to throw away undesirable parse trees, leaving only one tree for each sentence.
  - Rule 1: $*$ and / have higher precedence than + and $-$.
  - Rule 2: The operators of equal precedence associate to the left.
  - The desired parse tree: The one on the top of Slide 154.

The "Dangling-Else" Grammar

- The grammar

$$
\begin{array}{lcl}
\langle \text{stmt} \rangle & \rightarrow & \textbf{IF} \text{ ``('' } \langle \text{expr} \rangle \text{ ``)'' } \langle \text{stmt} \rangle \\
& | & \textbf{IF} \text{ ``('' } \langle \text{expr} \rangle \text{ ``)'' } \langle \text{stmt} \rangle \textbf{ ELSE } \langle \text{stmt} \rangle \\
& | & \textbf{other}
\end{array}
$$

- Two parse trees for **IF (e1) if (e2) s1 else s2**



- Match **else** with the closest previous unmatched **then**
- A parser disambiguates the two cases easily using this rule

Reading

- Pages 25 – 32, 40 – 43 and 159 – 175 of Red Dragon or Pages 39 – 52, $ 2.4.1 – 2.4.3 and 191 – 211 of Purple Dragon
- The VC Language Definition (Important for the next lecture)

Next Class:

- Top-Down Parsing (Assignment 2)
- Reading: Pages 44 – 56 and 176 – 195 of Red Dragon or Pages 60 – 76 and Pages 212 – 233 of Purple Dragon
- Assignment 2 spec
- Assignments 2 and 3 are one assignment – cannot do the latter if you do not do the former