460
1/23

Programming Languages
- Classes of languages
- OOP
- Interpreter vs. compiler
  how they work
- Write a simple language interpreter

Quizzes; miss more than 2 = -20% Final Grade

Textbook Concepts of Programming
Languages 11th edition
Robert Sebesta
Global Edition

Let Kooslesh know have book by Monday

Reading for Monday: Ch.3 up to
but not including 3.4

Syntax and Semantics of P.L.

```
int sum(std:: vector<int> &a){
        int total=0;
        for int i=0; i<a.length; i++)
            total += a[i];
        return total;
    }
```

Syntax: name operator expression semicolon
= syntactically correct assignment statement

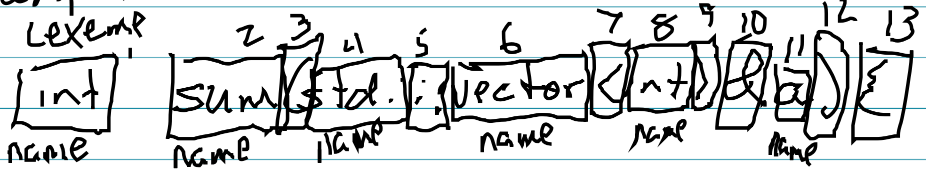
Semantics: total will be total + a[i]

Syntax determines format
Semantics describes what statement does

Compiled (c, c++). Take program build object code then run object file
Interpreted builds and runs in one step
① Lexica Analyzer ② Parser ③ evaluators (or generator)

Lexeme is an entity in the code

Example



Lexical Analyzer identifies the lexemes
It breaks apart the pieces without knowing what it means

~~Parser~~

token is a lexeme. May ~~take~~ group ~~of~~ lexemes together into an entity

~~Lexil~~ Parser takes lexemes and returns tokens. (lexeme and (if applicable) value

LexicalAnalyzer probably written as a class

parser identifies type of lexeme and identify values
It checks for syntax

Parser builds parse tree

Interpreter: traverses tree and executes it
Builds ~~symbl~~ symbol table
containing symbols (like variables)
with values

Syntax determined by CFG
Building Blocks of CFG
① Set of Terminals
② Set of Non-terminals
③ Set of production rules
④ Start Symbol (one of the non terminals)

$$\left.\underset{\text{LHS}}{\overset{\text{non terminal}}{S}} \to \overset{\text{terminal}}{a} \mid \underset{\text{RHS}}{aS}\ \right]\ \text{Defines Language}$$

Use CFG to generate statements or strings in the Language

Programming languages are all CFG

Take set of Production Rules and create a recognizer which takes tokens (terminals) follows rules backwards to see if it fits within language

READ TEXT CAREFULLY

Many times will have to read 2+ times

- ★ Generators
- ★ Recognizers

We will write recursive implementation of recognizer

Write CFG to generate
strings that contain letters
a & b

~~$S \to AB \mid SAS \mid SBS \mid \varepsilon$~~

~~$A \to a$~~

~~$B \to b$~~

$S \to SaS \mid SbS \mid \varepsilon$

$S \to ab \mid aSbS \mid$

$S \to \mid aTb \mid$

~~$T \to aTb \mid aT \mid Tb \mid \varepsilon$~~

$T \to aT \mid Tb \mid \varepsilon$

$$\boxed{\begin{array}{l} S \to aTb \\ T \to aT \mid Tb \mid \varepsilon \end{array}}$$

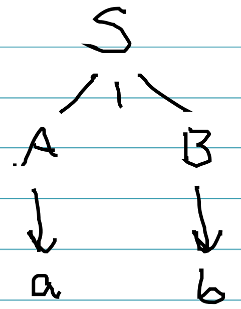
$S \to TT$

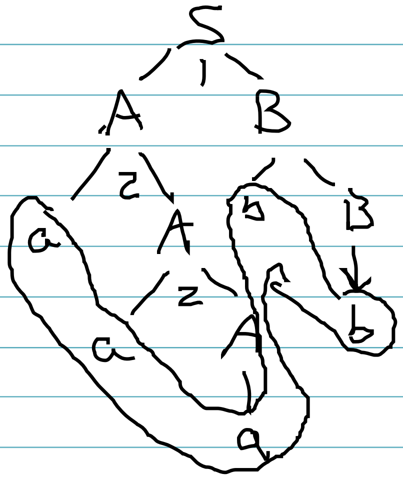
$T \to aT \mid Tb \mid a \mid b$

Derivation Tree

Leaves are non terminal

ab





a a a b b

int a, b, c ;

D → int V ;

V → <var> | <var>, V

<var> → a|b|c

~~ab~~ $a^n b^n$

~~S → aTb~~ | ab

① S → ab | aSb

② ~~S → () | (SP | (S) | (S)S~~ | S(S)

~~P →)~~

~~S → () | (SP | (S) | SS~~

~~P →)~~

In Class

S → aSb

S → aAb

A → aA

A → a

aaaaabb

$S \xrightarrow{1} aSb \xrightarrow{2} aaAbb$

$\xrightarrow{3} aaaAbb \xrightarrow{3} aaaaAbb$

$\xrightarrow{4} aaaaabb$

S → AB

A → a | aA

B → ab | aBb

Balanced parens

$$\boxed{S \to () \mid (S) \mid SS}$$

S → () | SS

Right Recursion appears on right AND is last non Term

1/21/19 460

- Quiz

$$S \to \overset{1}{(S)} \mid \overset{2}{()} \mid \overset{3}{SS}$$

Sentential Form

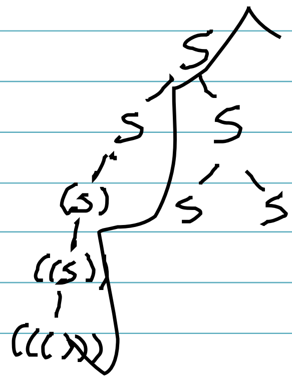
$$S \xrightarrow{3} SS \xrightarrow{3} SSS \xrightarrow{1} (S)SS \xrightarrow{1} ((S))SS \xrightarrow{2} ((()))SS$$

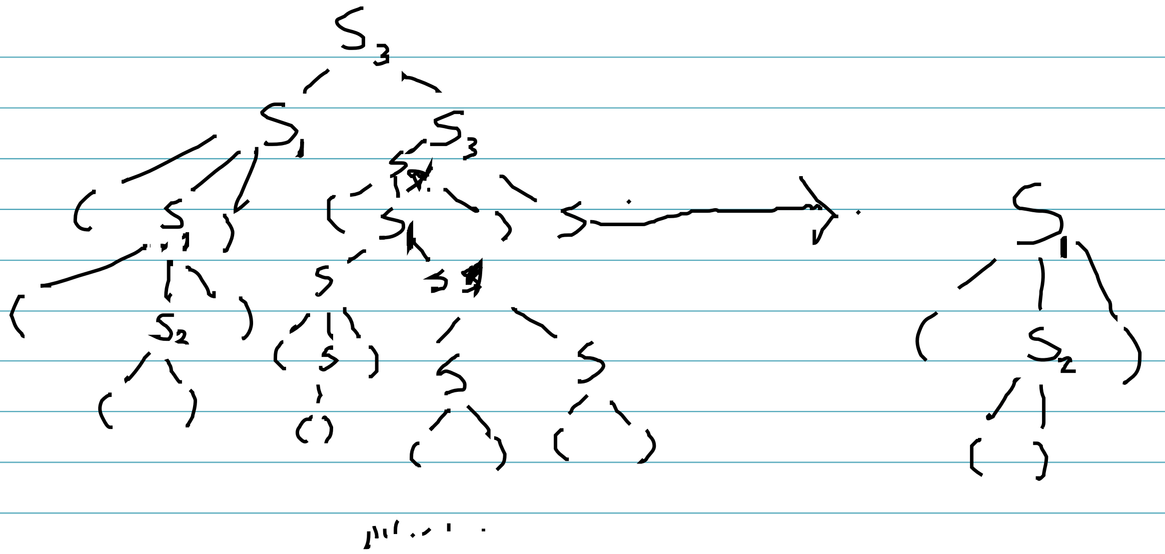
$$\xrightarrow{1} ((()))(S)S \xrightarrow{3} ((()))(SS)S \xrightarrow{3} ((()))(SSS)S$$

$$\xrightarrow{1} ((()))((S))SS)S \cancel{\xrightarrow{2} ((()))((())SS)S} \xrightarrow{2} ((()))((())S)S$$

$$\xrightarrow{2} ((()))((())()())S \xrightarrow{1} ((()))((())()())(S)$$

$$\xrightarrow{2} ((()))((())()())(())$$





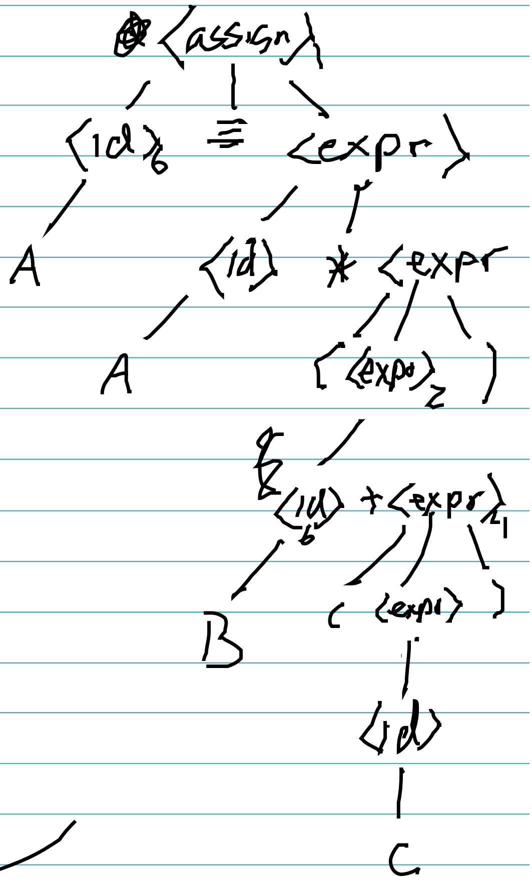
rule 3 inserted

expression tree : assignment statement

A = A * (B+(C))

<assign> → <id> = <expr>
<expr> → <id> + <expr>
<expr> → <id> * <expr>
<expr> → (<expr>)
<expr> → <id>

<id> → A|B|C



Use post order traversal of leaves

when writing production rules
must incorporate precedence enforcement

Rule of thumb
Higher precedence pushed deeper into tree

CFG w/precedence:

<assign> → <id> = <expr>

<expr> → <expr> + <term> | <term>

<term> → <term> * <factor> | <factor>

<factor> → (<expr>) | <id>

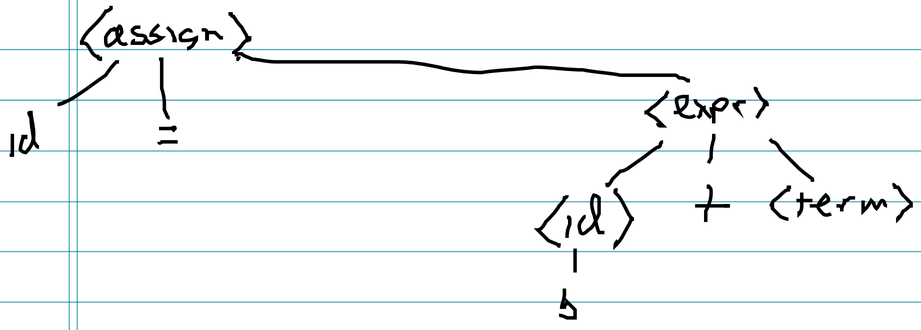
<id> → a variable

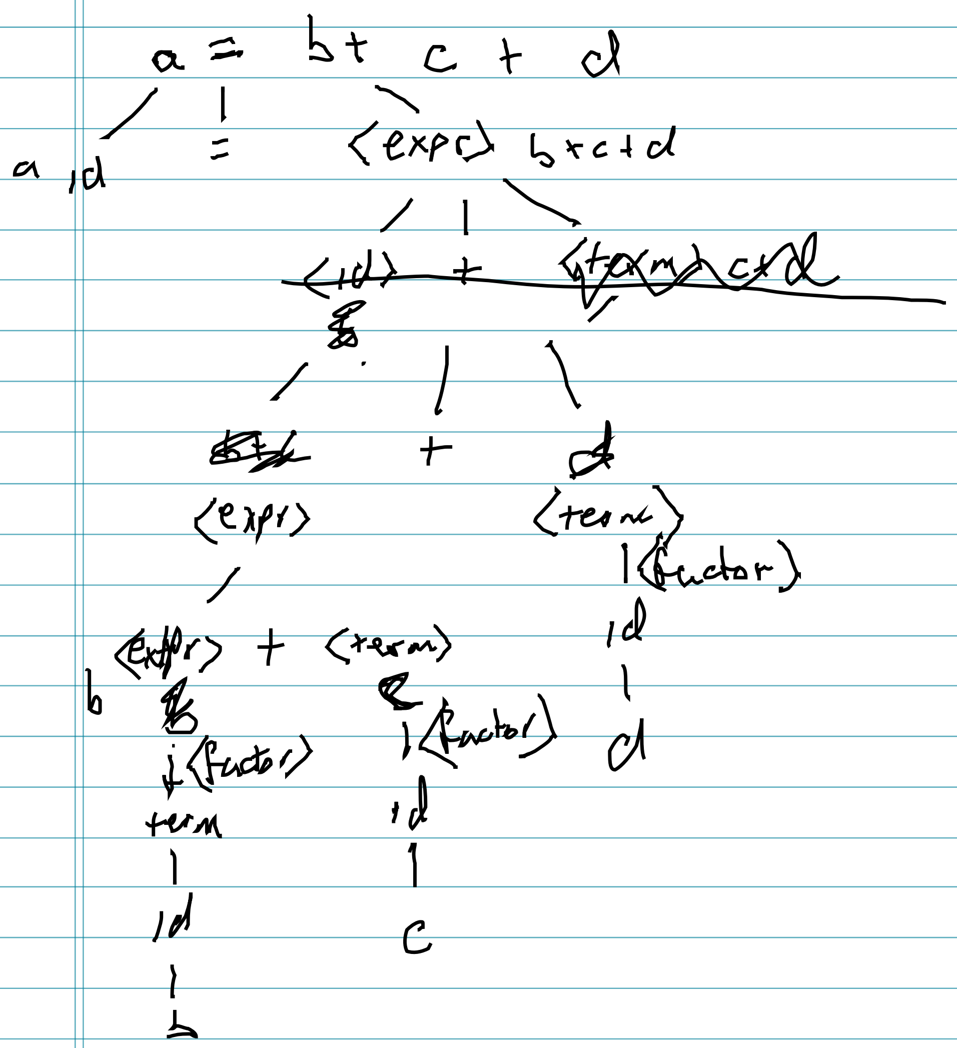
Things to observe

1. = is lowest precedence

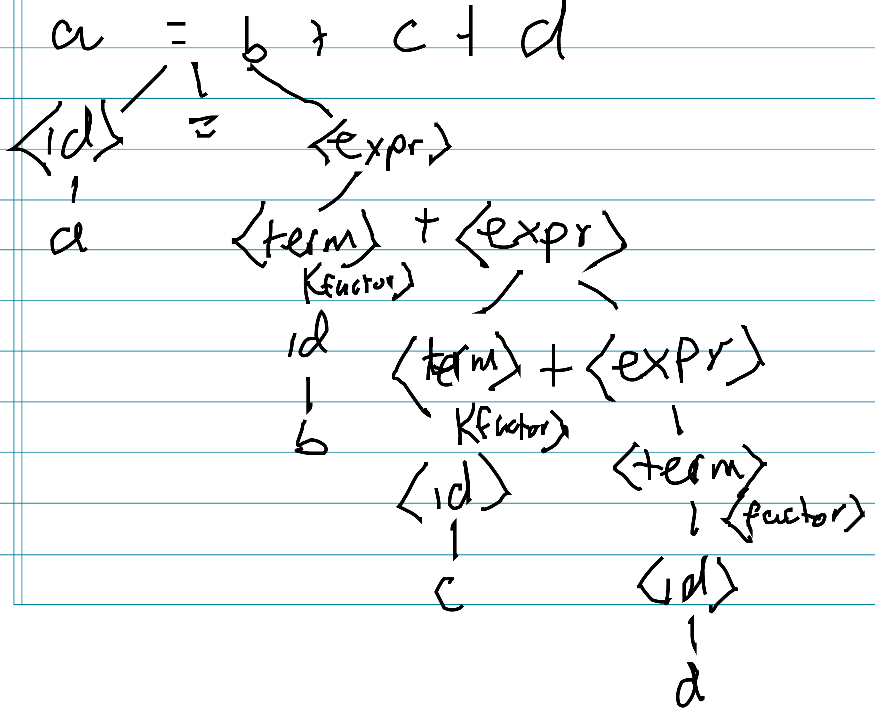
factor
↑
(<expr>)

$a = b + c * d - e * (f + d)$





switching term order reverses associativity
and/or ~~pre~~ depth



most programming languages ~~a~~ addition is left associative

int argc (how many arguments)
*argv[] array of pointers to arguments

in HPP
* - no Code!!

CPP
- Code goes here

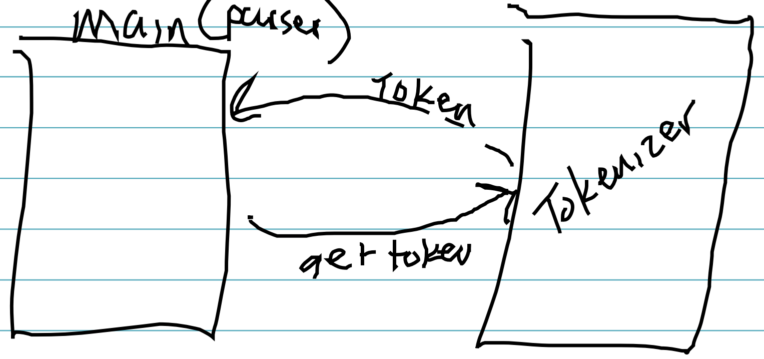
& = pass by reference (changes the value outside function)

will post tokenizer

---

* How many lines in input?
write this function

Components of Interpreter



2/4/19 460

work on writing tokenizer code

op → > | < | == | >= | <=

<Con> → && | || | !

~~S → <exp>~~

S → <boolExp>

<boolExp> → <boolExp> || <bool~~Ex~~Term> | <boolTerm> | ~~<boolExp>~~

<boolTerm> → <boolTerm> && <boolFactor> | ~~<boolTerm>~~

→ <boolFactor>

<boolFactor> → <boolFactor> ! <boolDeep>

→ <boolDeep>

<boolDeep> → (<boolDeep>) | <id>

<id> → variable