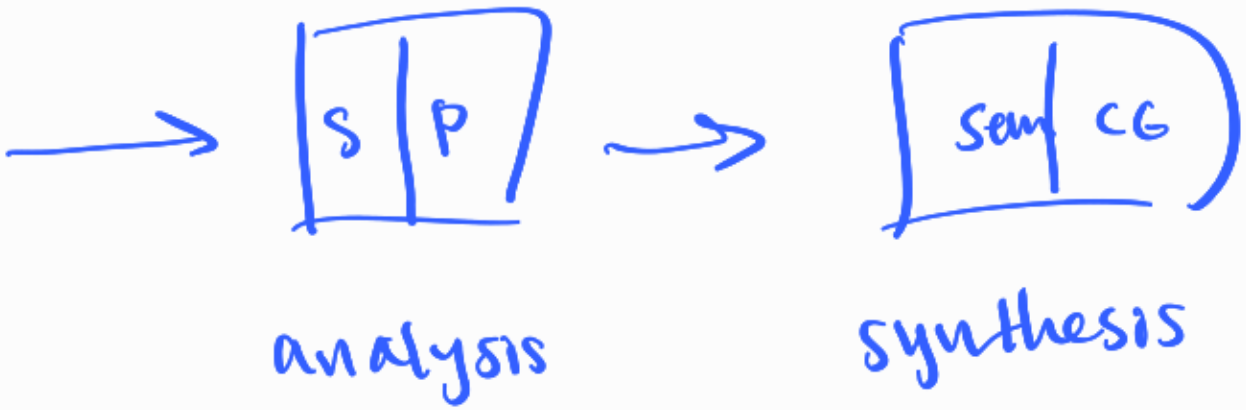
Compilers & Interpreters

341 Guest Lecture

3/12/2025

What is a compiler?



scanner

program:

```
// this is a comment!
if (a >= b) b = 2;
```

tokens:

IF LPAREN ID(a) GEQ
ID(b) RPAREN ID(b)
BECOMES INT(2) SEMI

Scanner – Mini-Idle

```
; converts a string to a list of tokens; uses special tokens lparen, rparen
; and comma for ( ) ,
(define (tokenize str)
  (let ([re "[a-zA-Z_][a-zA-Z0-9_.]*|[*][*]|==|<=|>=|!=|//|[-+*/%'(),<>=]|[^-+*/<>='(),^a-zA-Z_]+"])
    (define (is-string s)
      (and (>= (string-length s) 1) (equal? "\"" (substring s 0 1))))
    (define (expand s)
      (regexp-match* re s))
    (define (to-symbol s)
      (let ((n (string->number s)))
        (cond
         (n n)
         ((equal? s "(") 'lparen)
         ((equal? s ")") 'rparen)
         ((equal? s ",") 'comma)
         ((equal? s "True") #t)
         ((equal? s "False") #f)
         (else (string->symbol s)))))
    (define (process s)
      (if (is-string s)
          (list (substring s 1 (- (string-length s) 1)))
          (map to-symbol
               (foldr append '()
                      (map expand (regexp-split "[ \t\r]+" s))))))
    (foldr append '()
           (map process
                (regexp-match* "[^\"]+|\"[^\"]*\"" str)))))
```

iffy

Parser – Tiny Language Example

program ::= *statement* | *program statement*

statement ::= *assignStmt* | *ifStmt*

assignStmt ::= *id* = *expr* ;

ifStmt ::= if (*expr*) *statement*

expr ::= *id* | *int* | *expr* + *expr*

id ::= a | b | c | i | j | k | n | x | y | z

int ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

← production

nonterminal

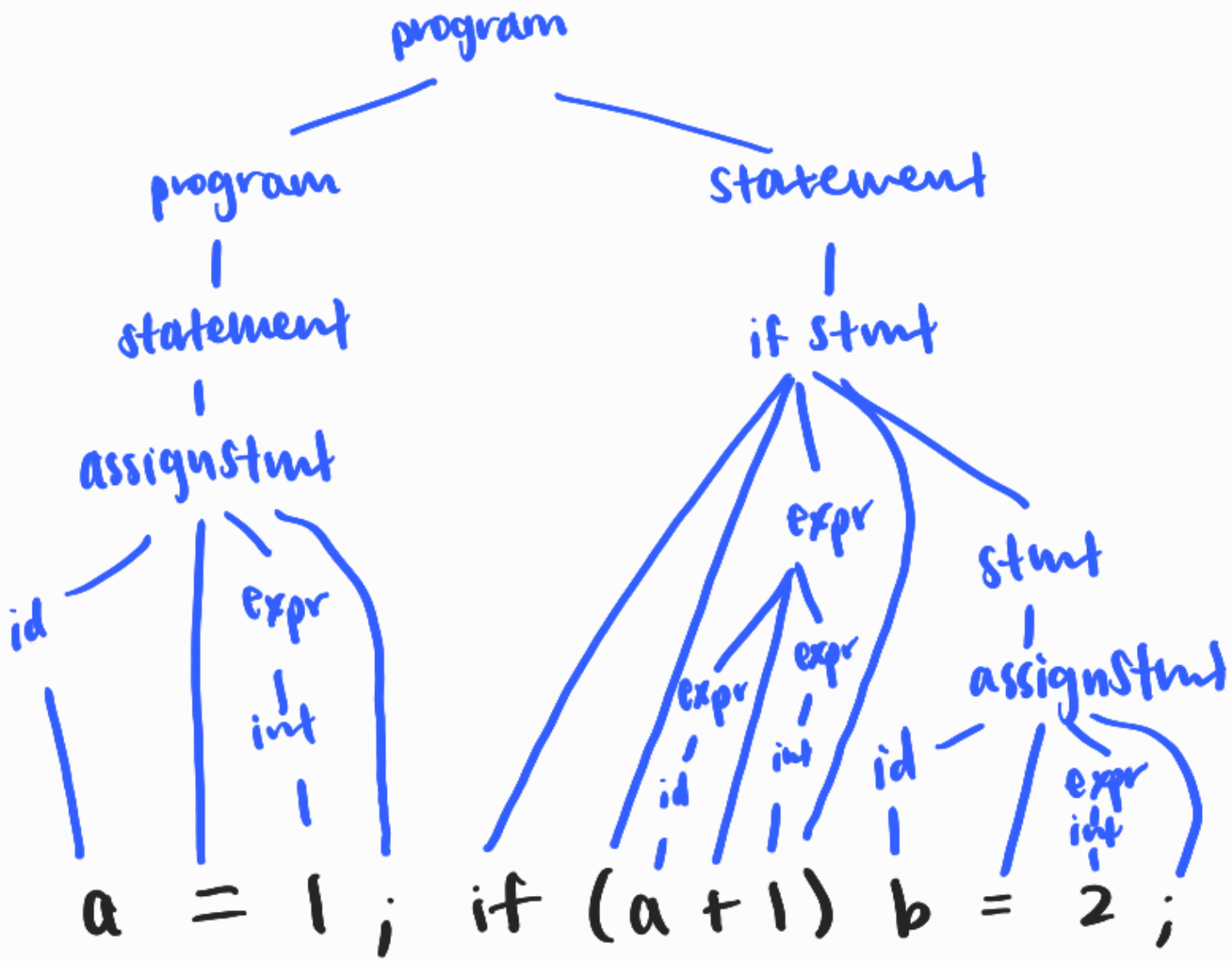
terminal

Parser – Mini-Idle Grammar

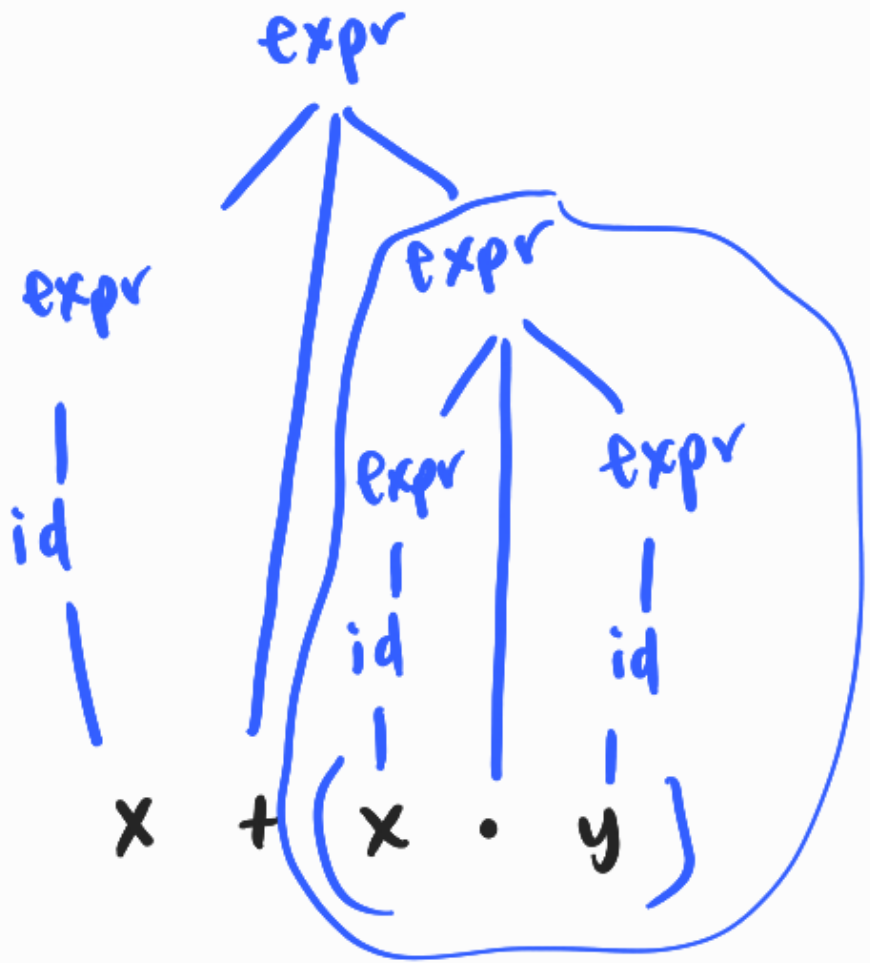
```
<expression> ::= <and> | <expression> "or" <and>
<and> ::= <negation> | <and> "and" <negation>
<negation> ::= <test> | "not" <negation>
<test> ::= <additive> | <additive> ("<" | ">" | "<=" | ">=" | "==" | "!=") <additive>
<additive> ::= <term> | <additive> ("+" | "-") <term>
<term> ::= <element> | <term> ("*" | "/" | "//" | %) <element>
<element> ::= <factor> | <factor> "**" <element> | ("-" | "+") <element>
<factor> ::= <unsigned number> | "(" <expression> ")" | <variable> |
             True | False | <f> "(" <expression> ")"
<f> ::= math.sin | math.cos | tan | math.log | math.exp | abs |
        float | math.ceil | math.floor | math.sqrt
```

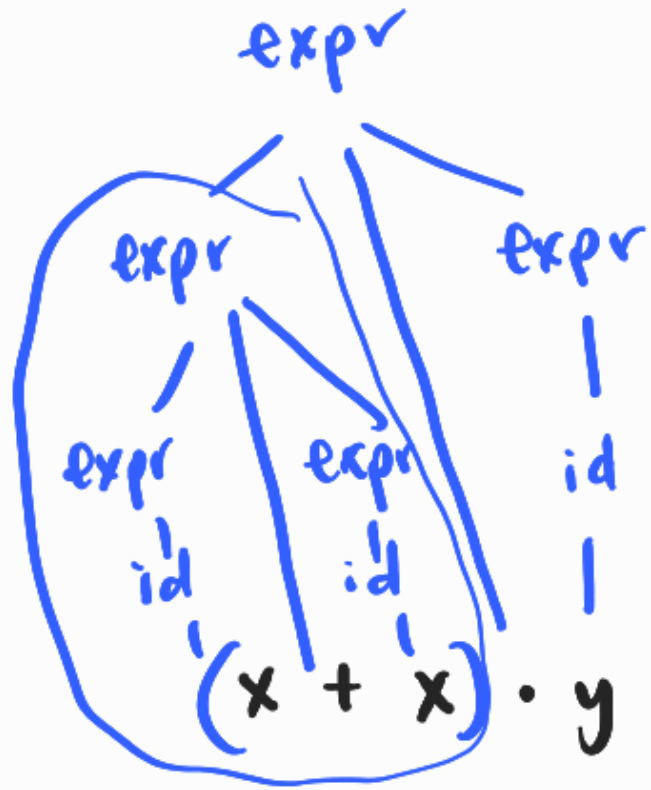
parser—Derivation Example

program ::= *statement* | *program statement*
statement ::= *assignStmt* | *ifStmt*
assignStmt ::= *id* = *expr* ;
ifStmt ::= if (*expr*) *statement*
expr ::= *id* | *int* | *expr* + *expr*
id ::= a | b | c | i | j | k | n | x | y | z
int ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9



Parser — Grammars & Ambiguity

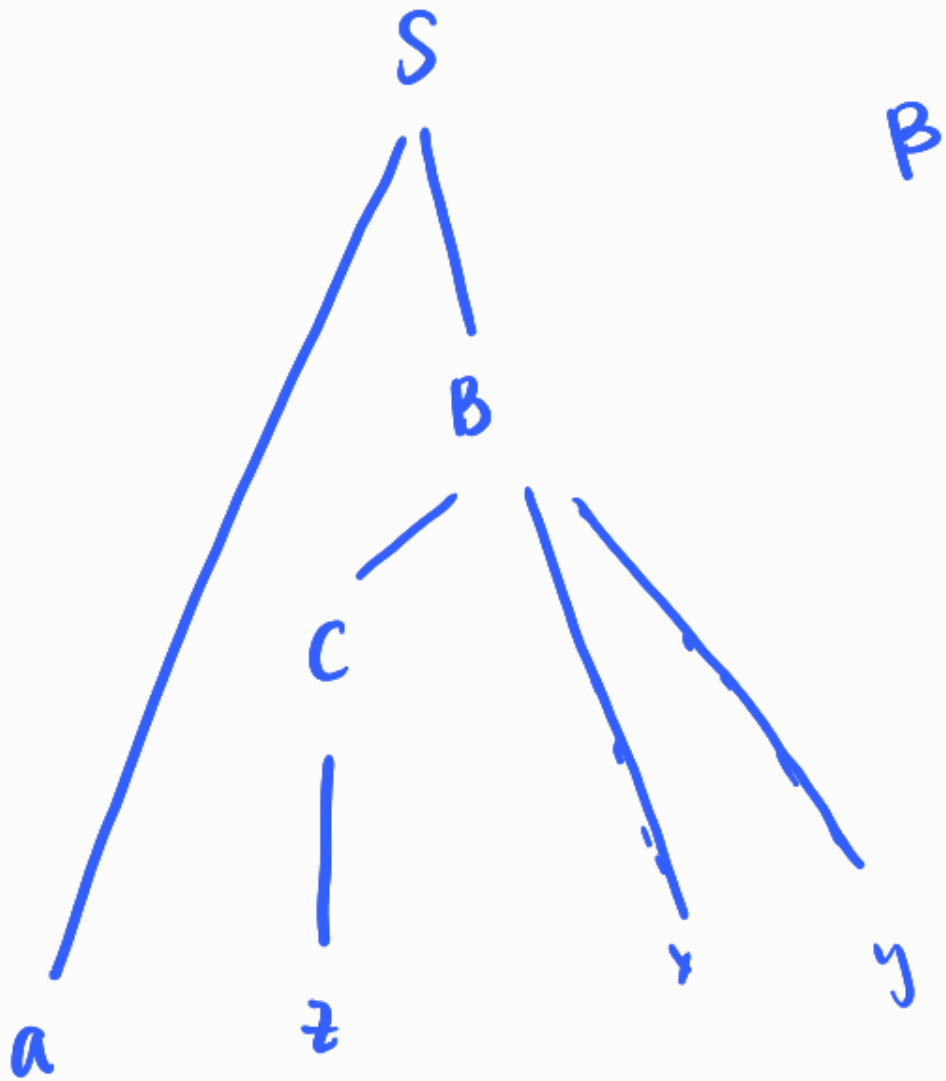




expr ::= expr + expr
| expr • expr
| id

id ::= x | y

Parser – Recursive Descent Example



B: C x y | z x

z

S ::= a B

B ::= C x y | y | C x x

C ::= z

Lookahead

y

Remaining

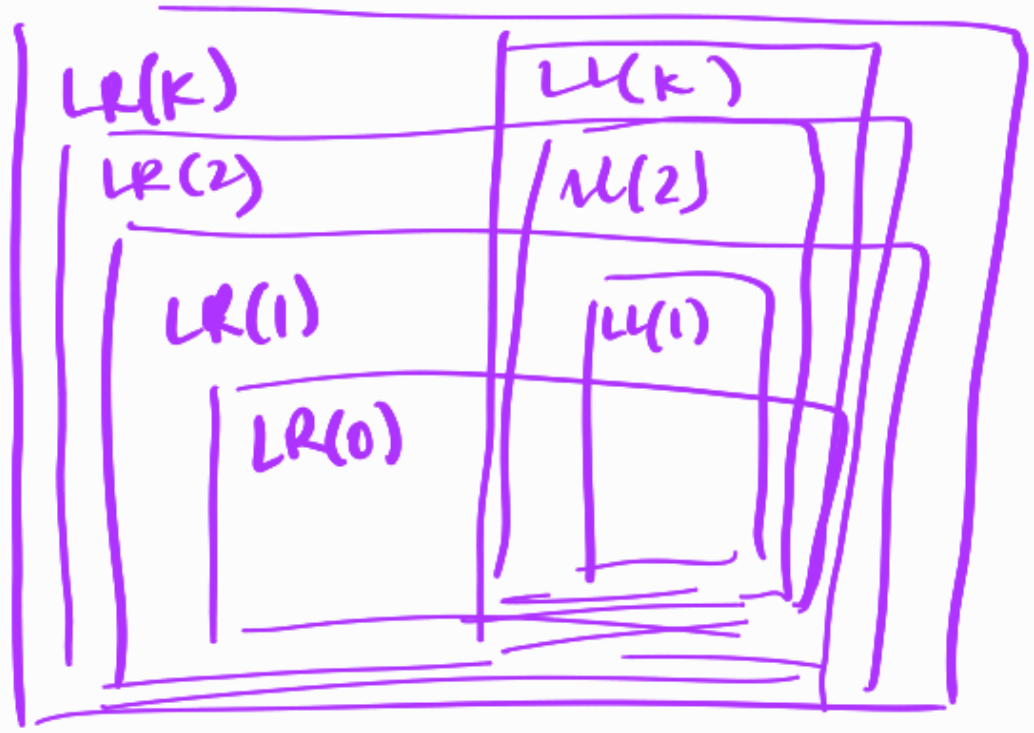
user input:

a z x y

Parser – Shift-Reduce Example

S ::= aABe
A ::= Abc | b
B ::= d
C ::= d

Stack	Input	Action
$	abbcde$	S
$a	bbcde$	S
$ab	bcde$	R
$aA	bcde$	S
$aAb	cde$	S
$aAbc	de$	R
$aA	de$	S
$aAd	e$	R
$aAB	e$	S
$aABe	$	R
$S	$	S
S		acc



Abstract Syntax Tree

program ::= *statement* | *program statement*

statement ::= *assignStmt* | *ifStmt*

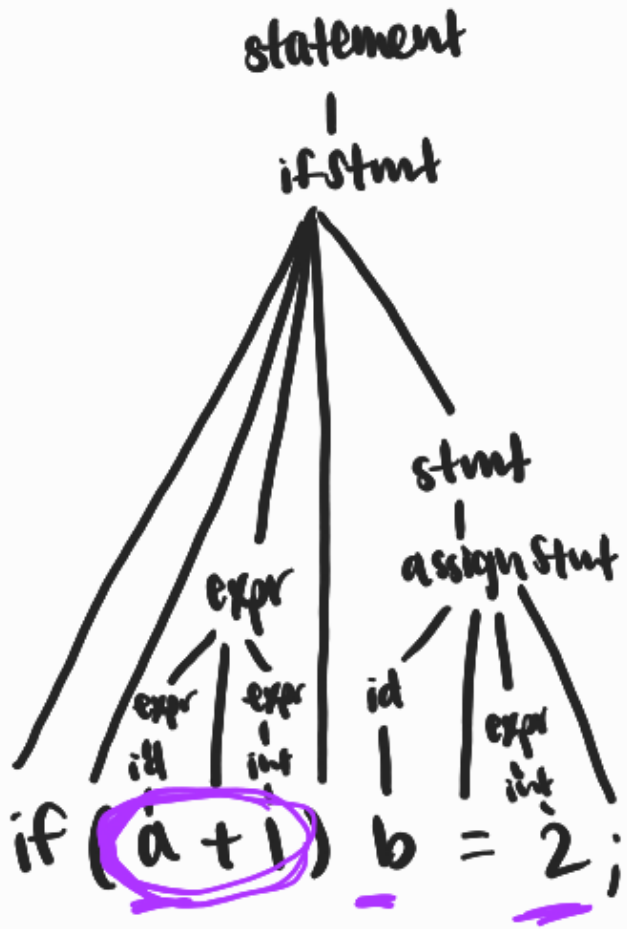
assignStmt ::= *id* = *expr* ;

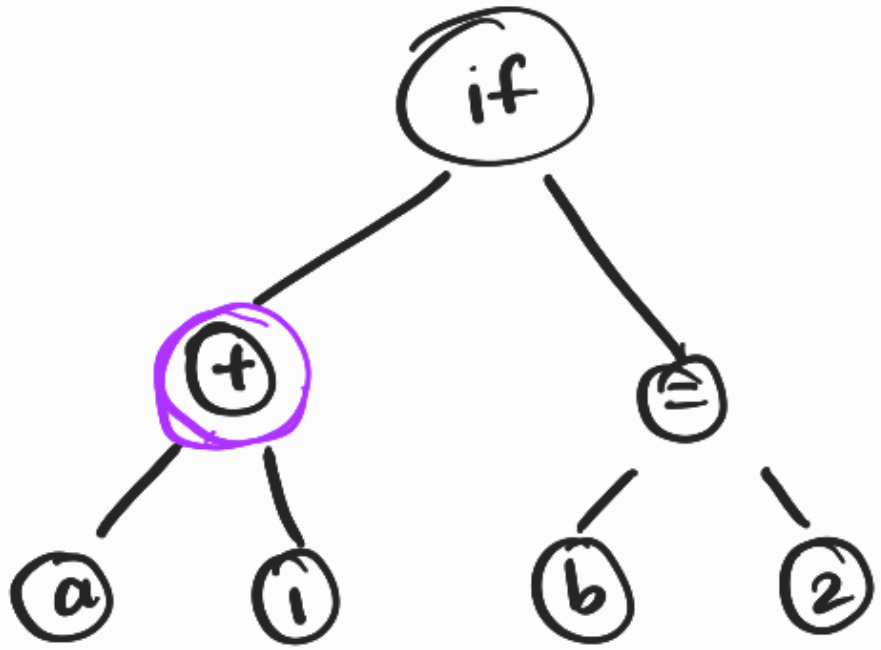
ifStmt ::= if (*expr*) *statement*

expr ::= *id* | *int* | *expr* + *expr*

id ::= a | b | c | i | j | k | n | x | y | z

int ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9





Optimizations

- peephole
- Local
- Intraprocedural
- Interprocedural

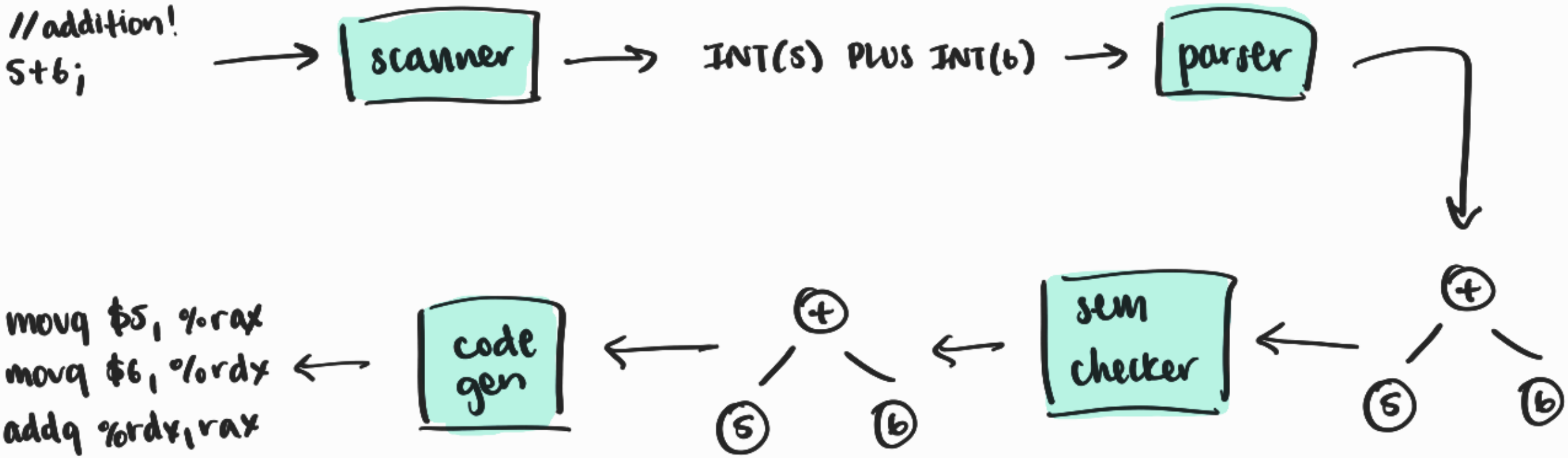
```
z = b[j]
for ( i++ )
    x += arr[i] + z
```

```
a = 3 + 4
```

```
z = 3 * x
a = z
```

```
b = z
```

Code Generation + Full Picture!



Thanks for coming!

- If you’re interested, look into CSE 401 (compilers!)
  - Several examples were taken from 401 course slides ☺