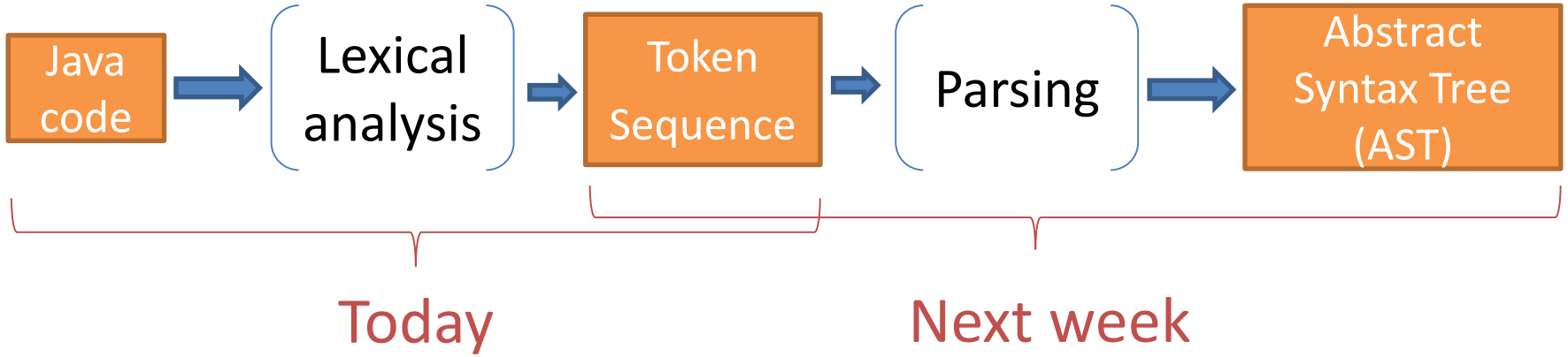
Compiler Construction
Winter 2020

Recitation 9:
Lexical Analysis

Yotam Feldman

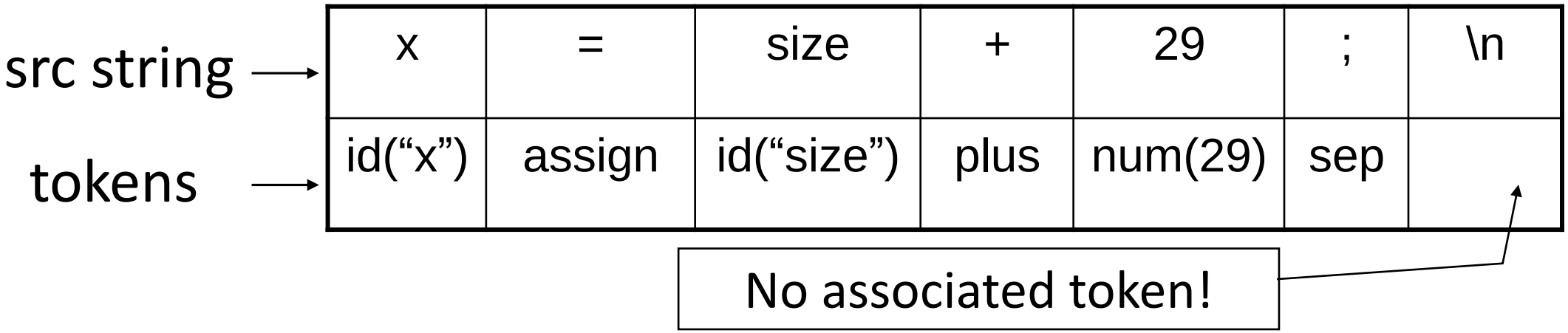
Based on slides by Guy Golan-Gueta
and the Technion compilers class' staff

Lexing & Parsing



Lexical Analysis

- Tokens are strings with an “identified meaning”



Lexical Analysis

- Tokens are strings with an “identified meaning”
- The **terminals** in the grammar

src string →	x	=	size	+	29	;	\n
tokens →	id(“x”)	assign	id(“size”)	plus	num(29)	sep	

$S \rightarrow$ id assign id plus num sep

$S \rightarrow$ ID WS ‘=’ WS ID WS ‘+’ WS NUM WS ‘;’ WS

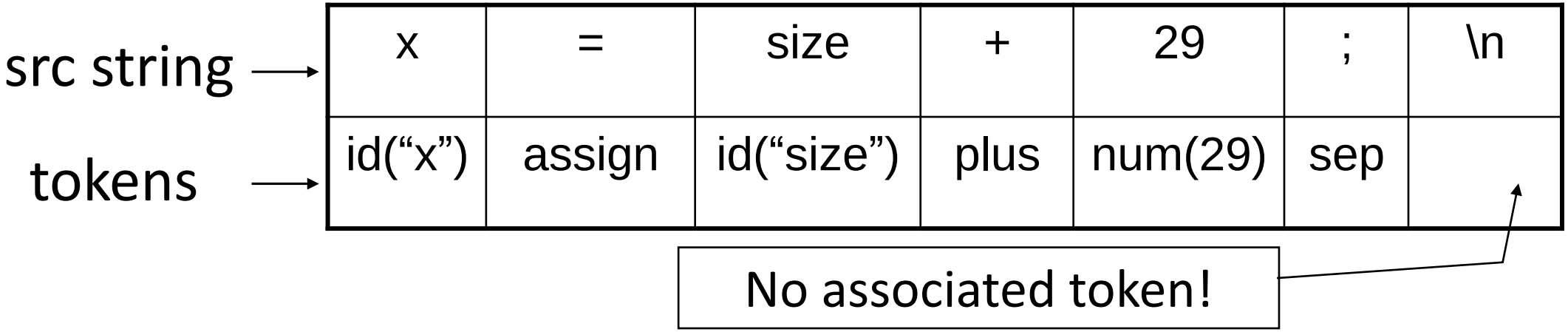
$WS \rightarrow$ ‘\n’ WS | ‘ ’ WS | ...

$ID \rightarrow$ ‘a’ ID | ‘b’ ID | ...

$NUM \rightarrow$ ‘1’ NUM0 | ‘2’ NUM0 | ...

Lexical Analysis

- Tokens are strings with an “identified meaning”
- The **terminals** in the grammar



- The compiler “**forgets**” the original strings

Manual Solution

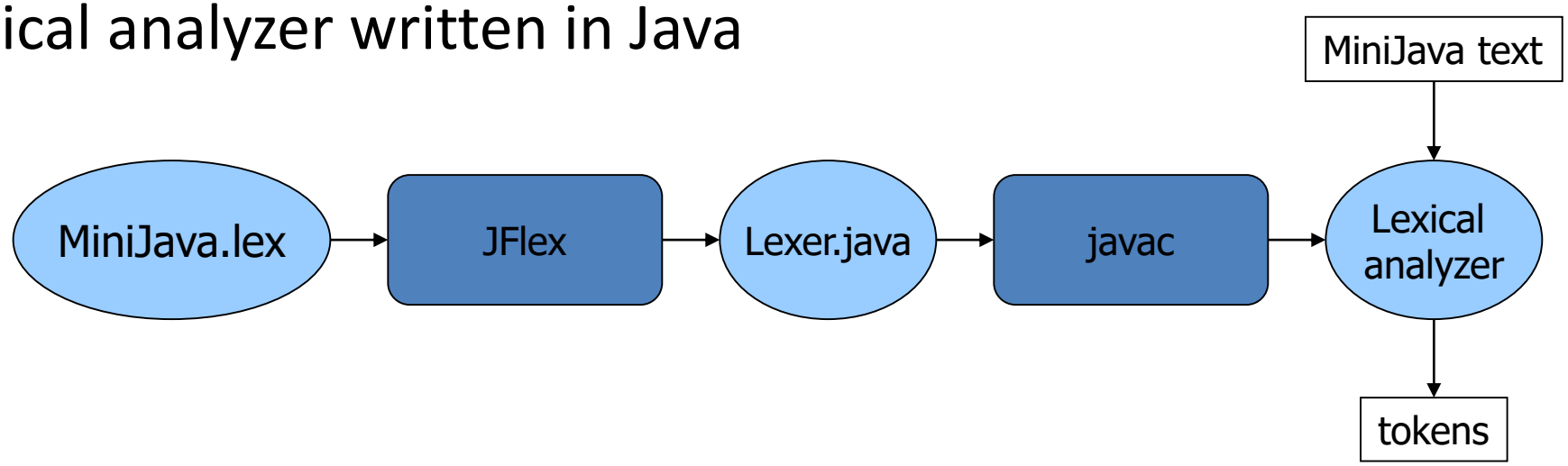
```
Token nextToken()
{
    char c ;
    while (true) {
        c = getchar();
        switch (c){
            case `  `: break;
            case `;`:  return Semicolon;
            case `+`:  c = getchar() ;
                       switch (c) {
                          case `+': return PlusPlus ;
                          case '=’  return PlusEqual;
                          default:  ungetc(c);
                                    return Plus;
        }                          ...
}
```

Hard to

- write,
- change,
- read,
- reuse,
- get correct

Lexical Analysis Generator

- Off the shelf lexical analysis generator
- Input
  - scanner specification file
- Output
  - Lexical analyzer written in Java



JFlex

- Simple
- Good for reuse
- Easy to understand
- Many developers and users debugged the generators

```
"+"             { return new symbol (sym.PLUS); }
"boolean"       { return new symbol (sym.BOOLEAN); }
“int"           { return new symbol (sym.INT); }
"null"          {return new symbol (sym.NULL);}
"while"         {return new symbol (sym.WHILE);}
"="             {return new symbol (sym.ASSIGN);}
…
```

JFlex Spec File

User code

Copied directly to Java file

%%

JFlex directives

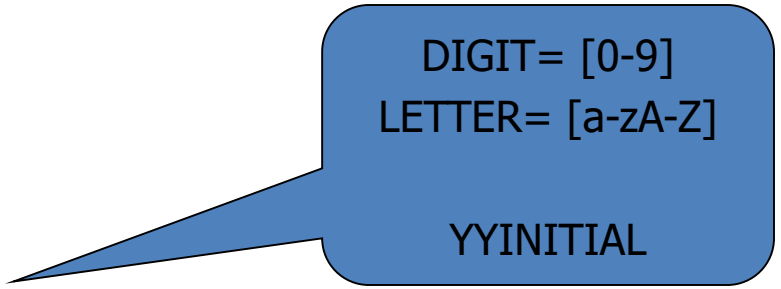
Define macros, state names

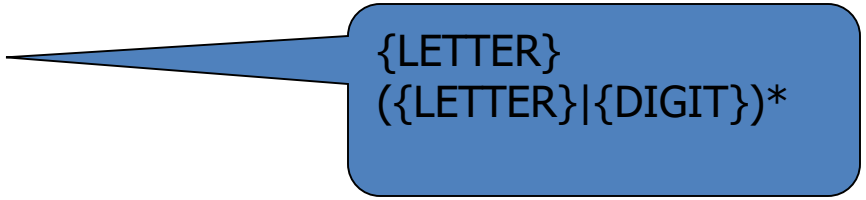
%%

Lexical analysis rules

How to break input to tokens

Action when token matched





Regular Expressions' Notation

.	any character except the newline	
"..."	string	"int"
a–b	range of characters	[0-9]
[...]	class of characters - any one character enclosed in brackets	[xyz] [a-zA-Z_]
[^…]	negated class – any one not enclosed in brackets	[^\t\r\n]
a|*b*	match *a* or *b*	
*	zero or more repetitions	.*
+	one or more repetitions	[0-9]+
?	zero or one repetitions	
(...)	grouping within regular expressions	
{name}	macro expansion	

Rules – Action

- Action
  - Java code
  - Can use special methods and vars
    - yyline
    - yytext()
  - Return a token when found

<YYINITIAL> {ID}
{return symbol(sym.ID, new String(yytext()));}

  - “Eat” chars for non tokens (whitespaces, comments)

Rules – “Eating”

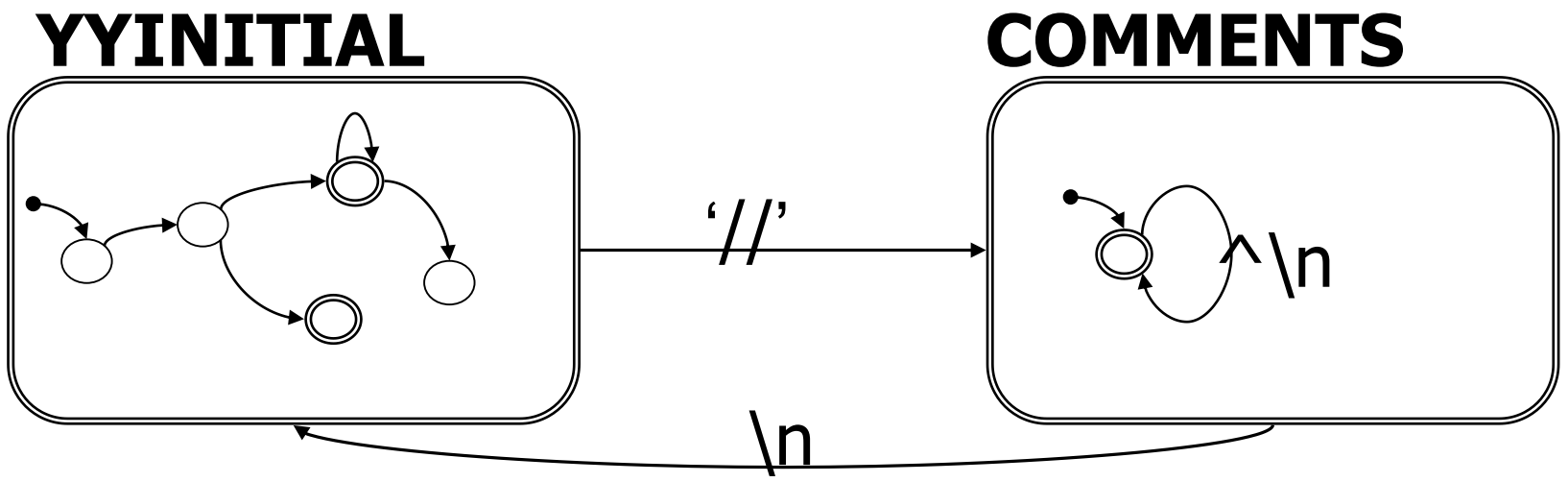
- Whitespaces:

<YYINITIAL> " "|"\t"|"\r"|"\n"
{ /* just skip what was found, do nothing */ }

- Single line comments?
- Multiline comments?

Rules – State

```
<YYINITIAL> "//" { yybegin(COMMENTS); }
<COMMENTS> [^\n] { }
<COMMENTS> [\n] { yybegin(YYINITIAL); }
```



Lines Count Example

```
    import java_cup.runtime.Symbol;
%%
    %cup
    %{
        private int lineCounter = 0;
    %}

    %eofval{
        System.out.println("line number=" + lineCounter);
        return new Symbol(sym.EOF);
    %eofval}

    NEWLINE=\n
%%
    <YYINITIAL>{NEWLINE} {
        lineCounter++;
    }
    <YYINITIAL>[^{NEWLINE}] { }
```

Lexer Conflicts

- When the same input matches several tokenizations.
- Example: Suppose that the following tokens are defined:
  - for: “for”
  - id: [a-z]+
- Conflicts:

Input	Possible tokenizations
abc	id(“a”) id(“ab”) id(“abc”)
ford	id(“f”) id(“fo”) id(“for”) id(“ford”) for
for	id(“for”) for

Solving Conflicts

- Maximal munch
  - Add more characters as long as still match some token
- Priorities
  - If still conflicted, the first matching rule in the lexer specification

- `for:` `"for"`
- `id:` `[a-z]+`

Input	Possible tokenizations
abc	id("a") id("ab") id("abc") ⬅
ford	id("f") id("fo") id("for") id("ford") ⬅ for
for	id("for") ⬅ for

Can such conflict resolving be performed directly in the parser?

Lexer Errors

- When the current input sequence cannot be matched to a token
- One form of syntactic error
  - More in the parser

Regex Debugger

- https://regex101.com/

Demo: Lexer for Simple Expressions

Summary

- Lexical analysis
- JFlex
- Maximal munch
- Rule precedence