



Compilers

Lexical Analysis

1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Optimization
5. Code Generation

```
if (i == j)  
    Z = 0;  
else  
    Z = 1;
```

```
\tif ((i==j))\n\t\tz = 0;\n\telse\n\t\tz = 1;
```

- Token Class (or Class)

- In English:

Noun, verb, adjective, ...

- In a programming language:

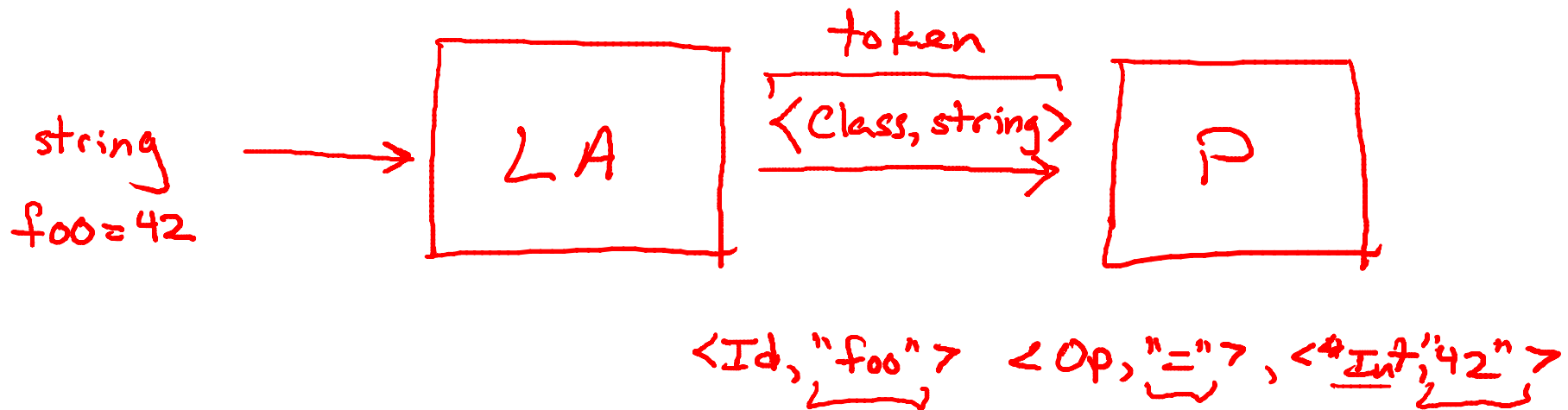
Identifier, Keywords, '(', ')', Numbers,
...

- Token classes correspond to sets of strings.
- Identifier:
 - *strings of letters or digits, starting with a letter*
- Integer: A1 Foo B17
 - *a non-empty string of digits* 0 12 001 00
- Keyword:
 - *“else” or “if” or “begin” or ...*
- Whitespace:
 - *a non-empty sequence of blanks, newlines, and tabs*

if --- → whitespace

- Classify program substrings according to role
- Communicate tokens to the parser

token class



Lexical Analysis

`\tif (i==j)\n\t\tz=0;\n\telse\n\t\tz=1;`

Diagram illustrating the lexical analysis of the code snippet above. The code is annotated with vertical lines and labels below it, indicating the classification of each token:

- `\`: w (whitespace)
- `t`: k (keyword)
- `i`: I (identifier)
- `=`: = (operator)
- `j`: I (identifier)
- `)`:) (operator)
- `\n`: w (whitespace)
- `\t`: w (whitespace)
- `\t`: w (whitespace)
- `z`: I (identifier)
- `=`: = (operator)
- `0`: = (operator)
`;`: ; (operator)- `\n`: w (whitespace)
- `e`: k (keyword)
- `l`: k (keyword)
- `s`: k (keyword)
- `e`: k (keyword)
- `\n`: w (whitespace)
- `\t`: w (whitespace)
- `\t`: w (whitespace)
- `z`: I (identifier)
- `=`: = (operator)
- `1`: = (operator)
- `:`: ; (operator)

Operator
Whitespace
Keywords
Identifiers
Numbers

(
)
;
=

For the code fragment below,
choose the correct number of tokens in
each class that appear in the code fragment

```
x = 0;\n\twhile (x < 10) {\n\t\tx++;\n}
```

- ☐ W = 9; K = 1; I = 3; N = 2; O = 9
- ☐ W = 11; K = 4; I = 0; N = 2; O = 9
- ☐ W = 9; K = 4; I = 0; N = 3; O = 9
- ☐ W = 11; K = 1; I = 3; N = 3; O = 9

W: Whitespace

K: Keyword

I: Identifier

N: Number

O: Other Tokens:

{ } () < ++ ; =

- An implementation must do two things:
 1. Recognize substrings corresponding to tokens
 - The lexemes
 2. Identify the token class of each lexeme

< token class, lexeme >
token