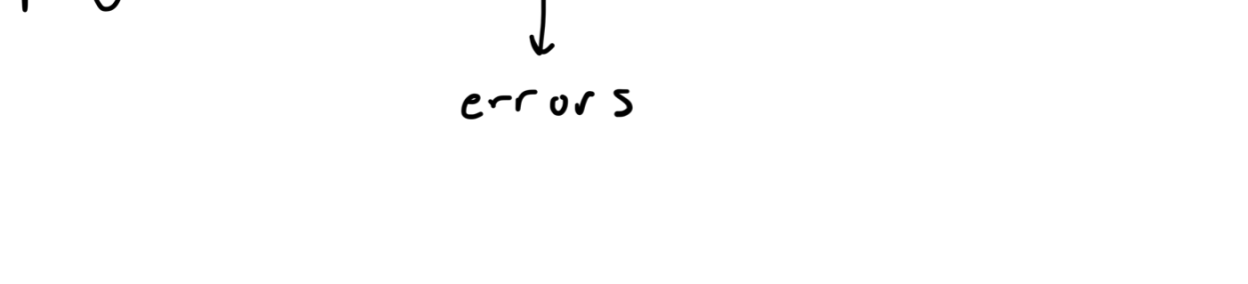


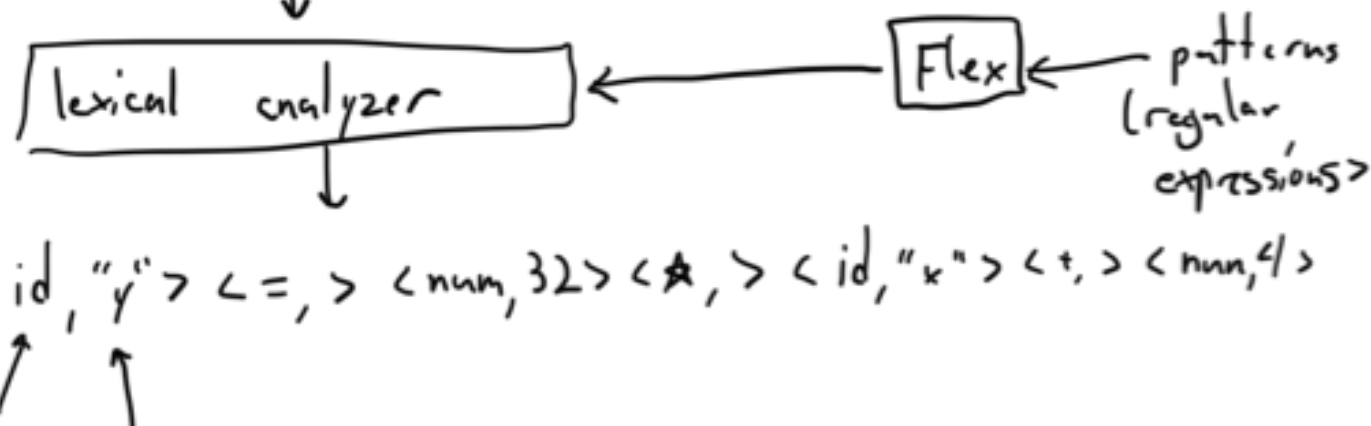
Compiler Design

Simply stated, a compiler is a program that reads a program written in one language - the source language - and translates it into an equivalent program in another language - the target language.

The diagram shows a 3D box on the left and a 2D rectangle on the right. The box is drawn with perspective, showing its front face, top, and right side. The rectangle is a simple 2D shape. Both are drawn with black lines on a white background.

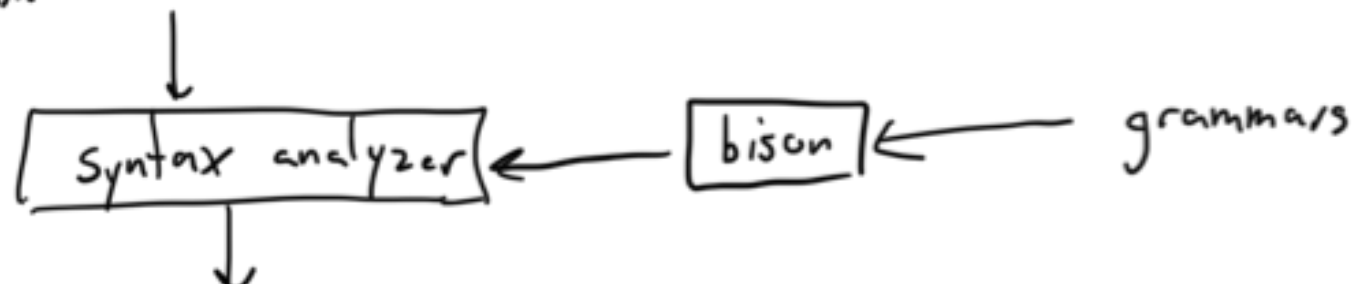


So wie bei:

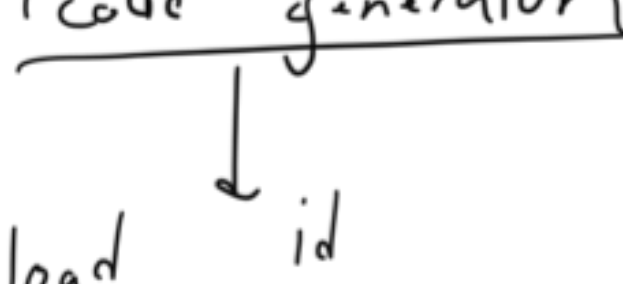
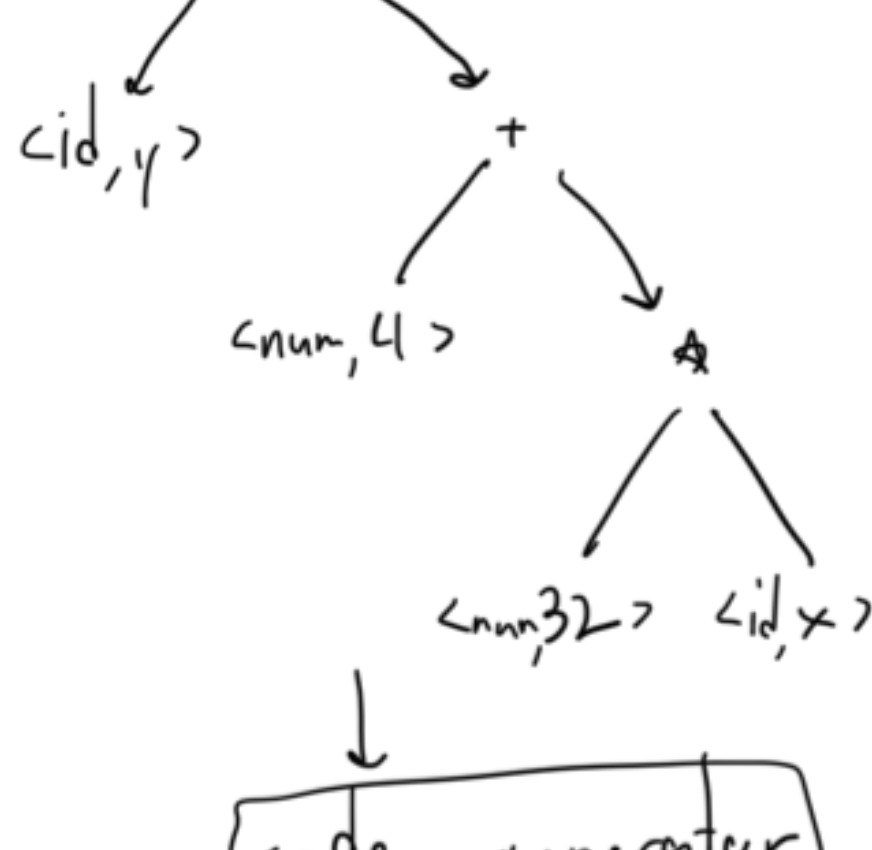
$$y = 32 + 2$$


tokens .

$\langle \text{id}, "y" \rangle \langle =, \rangle \langle \text{num}, 32 \rangle \langle *, \rangle \langle \text{id}, "x" \rangle \langle +, \rangle \langle \text{num}, 1 \rangle$

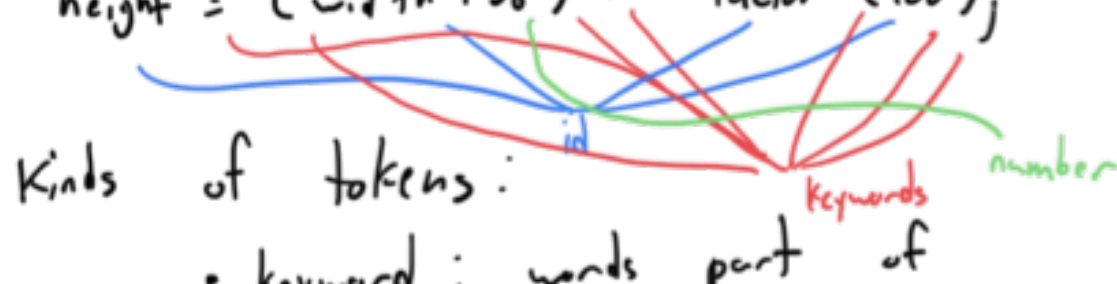


syntax tree



mul	32
add	4
store	4

6. d.



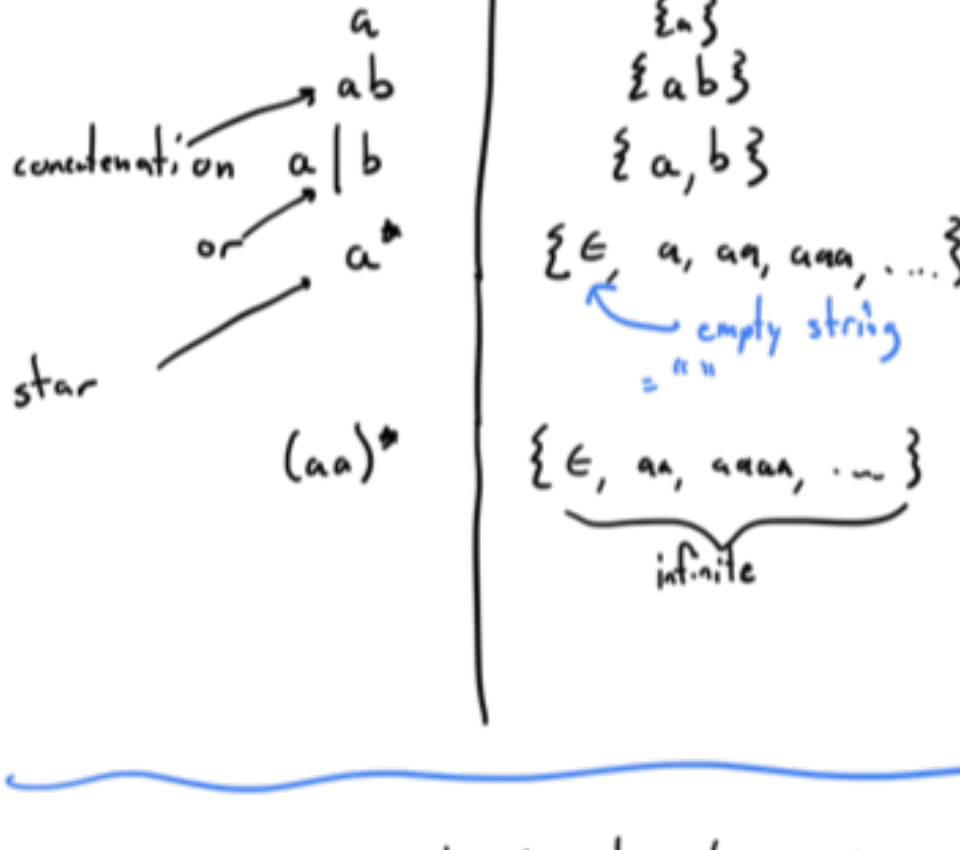
- keyword .
the language

- "while" or "class" or "true".
 - identifiers : names of variables, functions, etc defined by the programmer.
 - numbers : integers, floats, fractions
 - strings : literal character sequences
 - comments & whitespace
-
- Regular Expressions

regex is a string

denotes $L(s)$, a set of strings
derived from an alphabet Σ .
 $L(s)$ is the language of s

$\Sigma = \{a, b\}$	finite
reg^X	language



- and is one of the first
scanners. It was translated to
C to create Flex (fast
lexical analyzer).
- | input | match |
|-------|-------|
| | |

x	char 'x'
$[xyz]$	either x, y, or z
$[abj - oz]$	matches a, or b, or a letter j to o, or z.
$[ab][cd]$ concatenation	a or b, followed by c or d.
$[^A-Z]$	negation. Any char except an uppercase letter.
$[^A-Z\n]$	any char except an uppercase letter or a newline. <i>So, no newlines are matched</i>
r^*	matches regex r 0 or more times.
r^+	rr^*
$r?$	optional r
mo^*	$\{mo, moo, mooo, \dots\}$
$(mo)^*$	$\{\epsilon, moo, moomoo, \dots\}$
$r\{2,5\}$	2 to 5 r's
$r\{2,\}$	2 or more r's
$r\{4\}$	4 r's
$r s$	r or s
"foo"	literal "foo"
$\n$	newline
$\t$	tab
$foo bar^*$	$(foo) (ba(r^*))$
$foo bar)^*$	foo, bar, ϵ , barfoo, foofoo, foofoofoo, ...
$\$$	end of line before the $\n$ (different than $\n$)

```

graph LR
    A[flex.l] --> B[flex.i]
    B --> C[lex.yy.c]
  
```

