Script **generated by TTT**

Title: Simon: Compilerbau (28.04.2014)

Date: Mon Apr 28 14:22:08 CEST 2014

Duration: 45:18 min

Pages: 38

Chapter 5:
Scanner design

Scanner design

Input (simplified): a set of rules:

$$\begin{array}{ll} e_1 & \{\ \texttt{action}_1\ \} \\ e_2 & \{\ \texttt{action}_2\ \} \\ \quad\ldots & \\ e_k & \{\ \texttt{action}_k\ \} \end{array}$$

Scanner design

Input (simplified): a set of rules:

$$\begin{array}{ll} e_1 & \{\ \texttt{action}_1\ \} \\ e_2 & \{\ \texttt{action}_2\ \} \\ \quad\ldots & \\ e_k & \{\ \texttt{action}_k\ \} \end{array}$$

Output: a program,

... reading a maximal prefix w from the input, that satisfies $e_1 \mid \ldots \mid e_k$;

... determining the minimal i , such that $w \in [\![e_i]\!]$;

... executing $\texttt{action}_i$ for w.

Implementation:

Idea:

- Create the DFA $\mathcal{P}(A_e) = (Q, \Sigma, \delta, q_0, F)$ for the expression $e = (e_1 \mid \ldots \mid e_k)$;
- Define the sets:

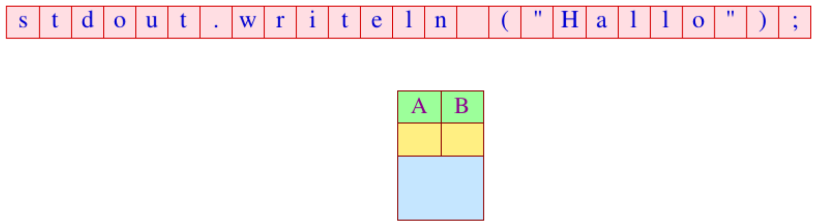
$$\begin{array}{lcl} \boxed{F_1} & = & \{q \in F \mid \boxed{q \cap \text{last}[e_1] \neq \emptyset}\} \\ \boxed{F_2} & = & \{q \in \boxed{(F \backslash F_1)} \mid q \cap \boxed{\text{last}[e_2]} \neq \emptyset\} \\ & \ldots & \\ F_k & = & \{q \in (F \backslash (F_1 \cup \ldots \cup F_{k-1})) \mid q \cap \text{last}[e_k] \neq \emptyset\} \end{array}$$

- For input w we find: $\delta^*(q_0, w) \in \boxed{F_i}$ iff the scanner must execute `action`$_{\boxed{i}}$ for w

Implementation:

Idea (cont'd):

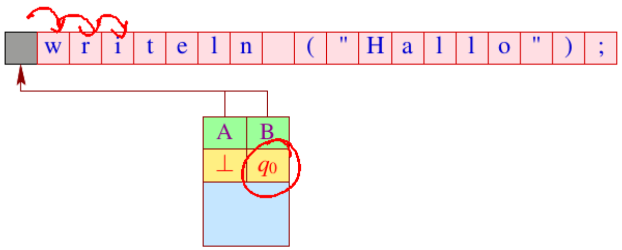
- The scanner manages two pointers $\langle A, B\rangle$ and the related states $\langle q_A, q_B\rangle$...
- Pointer A points to the last position in the input, after which a state $q_A \in F$ was reached;
- Pointer B tracks the current position.



Implementation:

Idea (cont'd):

- The scanner manages two pointers $\langle A, B\rangle$ and the related states $\langle q_A, q_B\rangle$...
- Pointer A points to the last position in the input, after which a state $q_A \in F$ was reached;
- Pointer B tracks the current position.

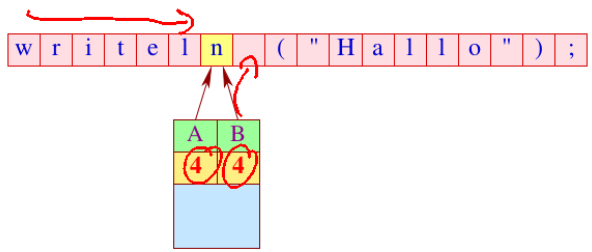


Implementation:

Idea (cont'd):

- The current state being $q_B = \emptyset$, we consume input up to position A and reset:

$$\begin{array}{lclclcl} B & := & A; & & A & := & \perp; \\ q_B & := & q_0; & & q_A & := & \perp \end{array}$$

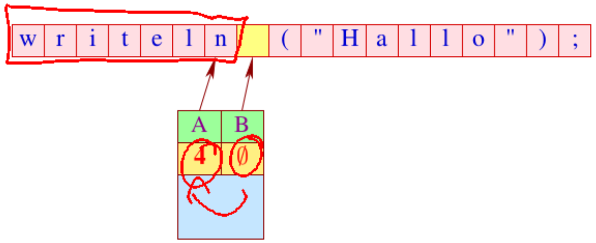


Implementation:

Idea (cont'd):

- The current state being $q_B = \emptyset$, we consume input up to position A and reset:

$$\begin{array}{lll} B & := & A; \quad A := \perp; \\ q_B & := & q_0; \quad q_A := \perp \end{array}$$



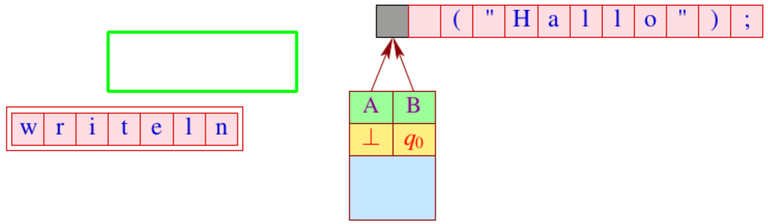
Implementation:

int x = 5;
/*
x = 42;
*/

Idea (cont'd):

- The current state being $q_B = \emptyset$, we consume input up to position A and reset:

$$\begin{array}{lll} B & := & A; \quad A := \perp; \\ q_B & := & q_0; \quad q_A := \perp \end{array}$$



Input (generalized): a set of rules:

```
⟨state⟩ { e1      { action1  yybegin(state1); }
          e2      { action2  yybegin(state2); }
             ...
          ek      { actionk  yybegin(statek); }
        }
```

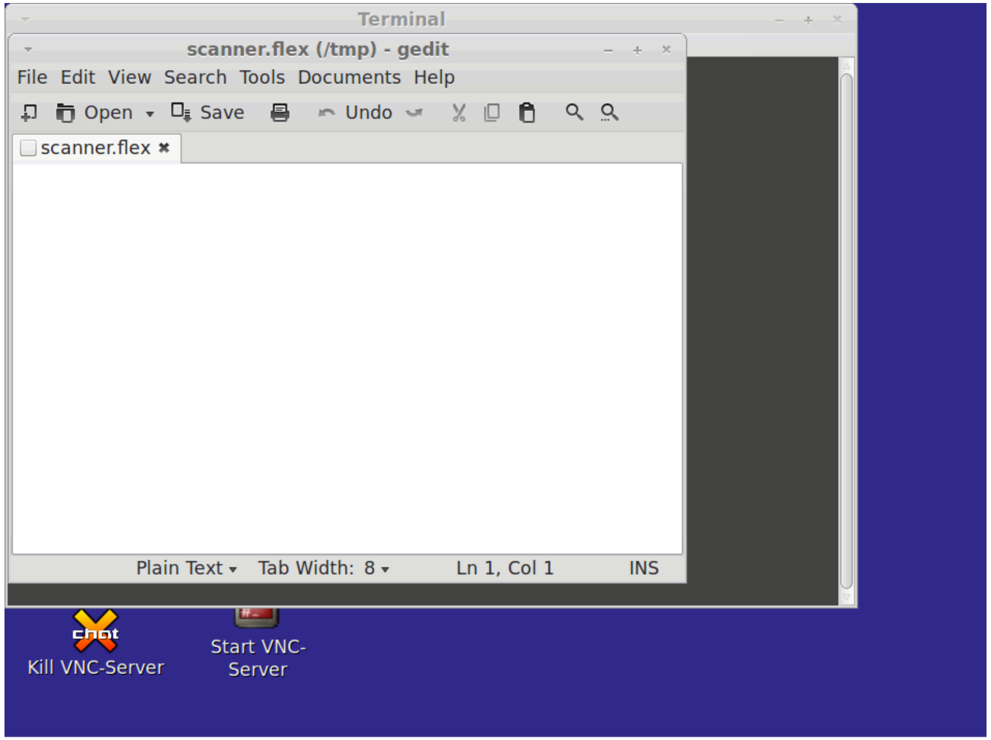
- The statement `yybegin (state_i);` resets the current state to `state_i`.
- The start state is called (e.g.flex JFlex) `YYINITIAL`.

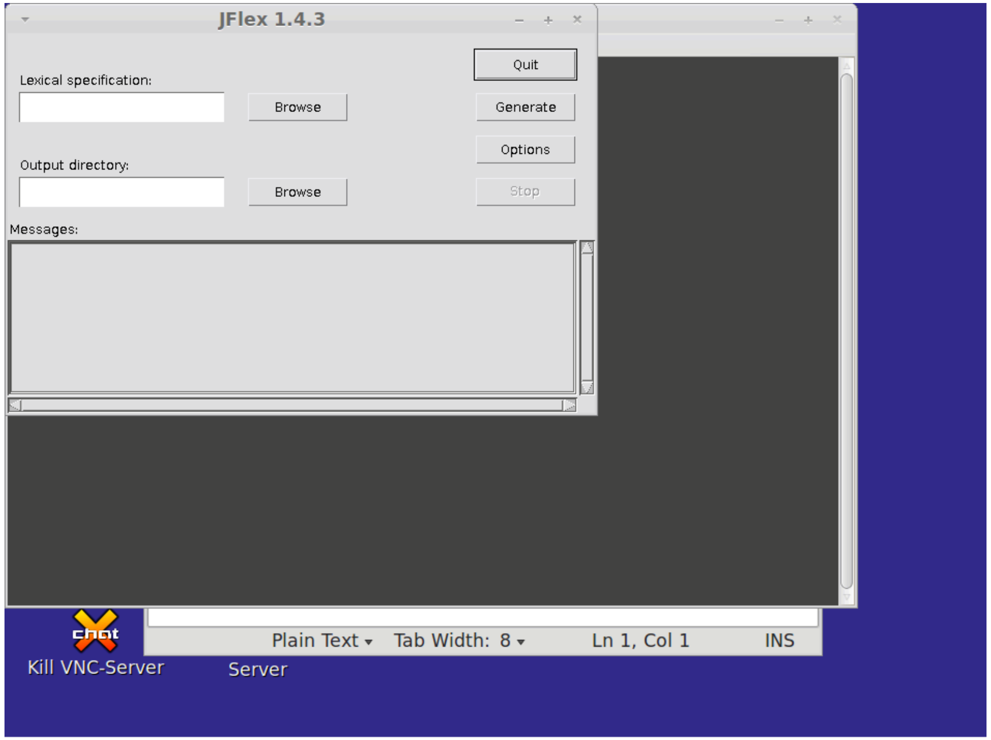
... for example:

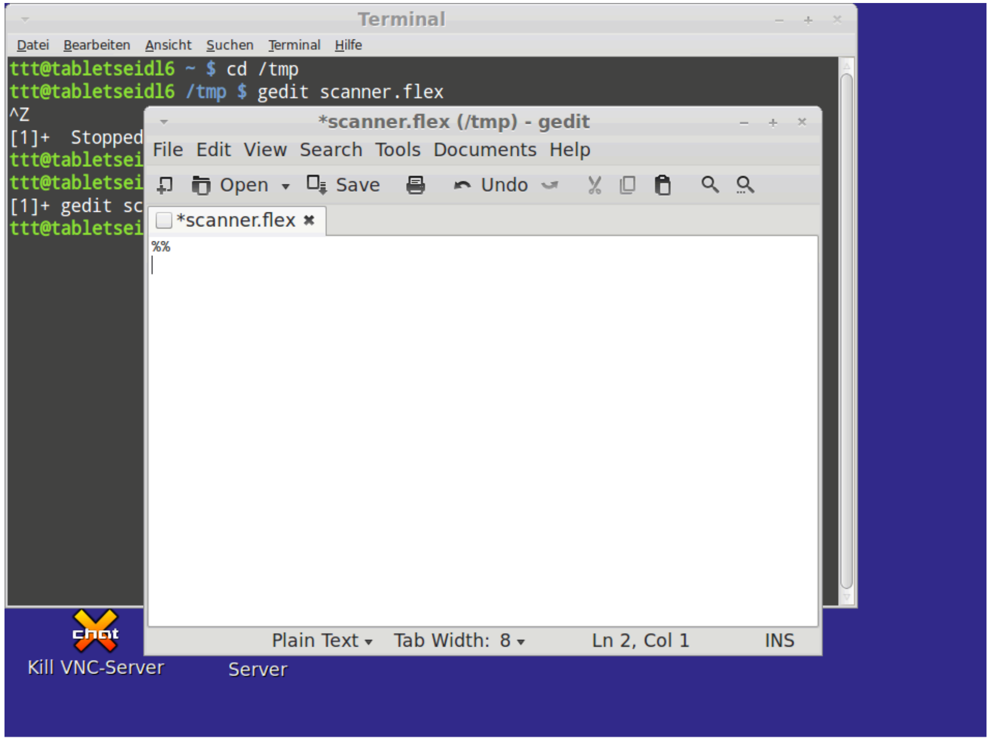
```
⟨YYINITIAL⟩    "/*"      { yybegin(COMMENT); }
⟨COMMENT⟩    { "*/"      { yybegin(YYINITIAL); }
               . | \n    { }
             }
```

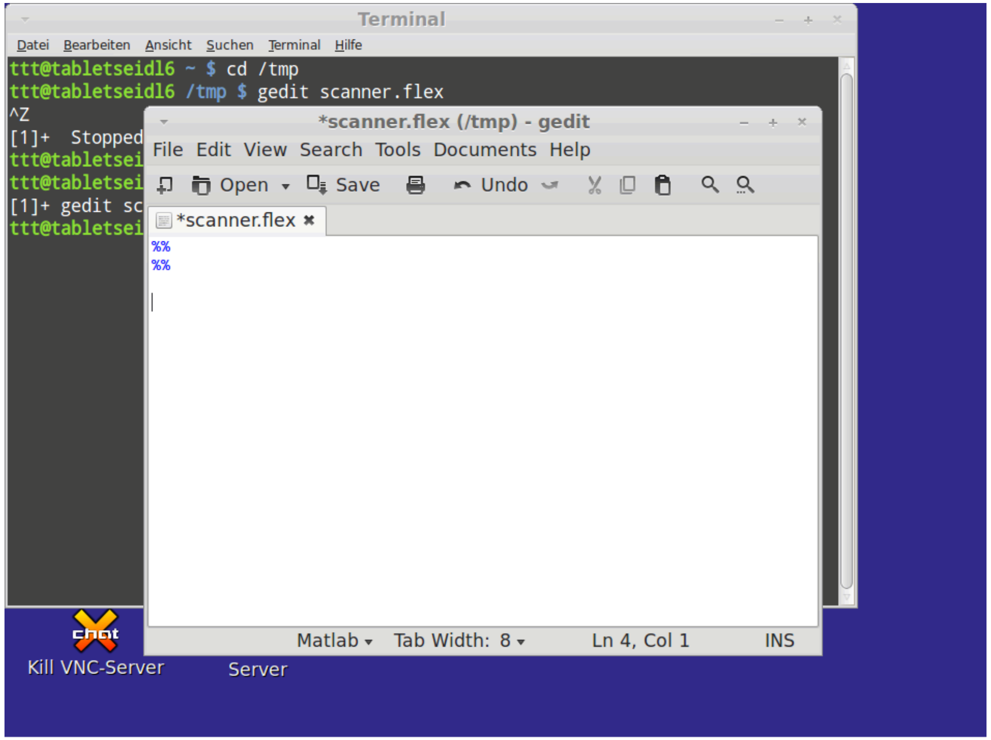

Remarks:

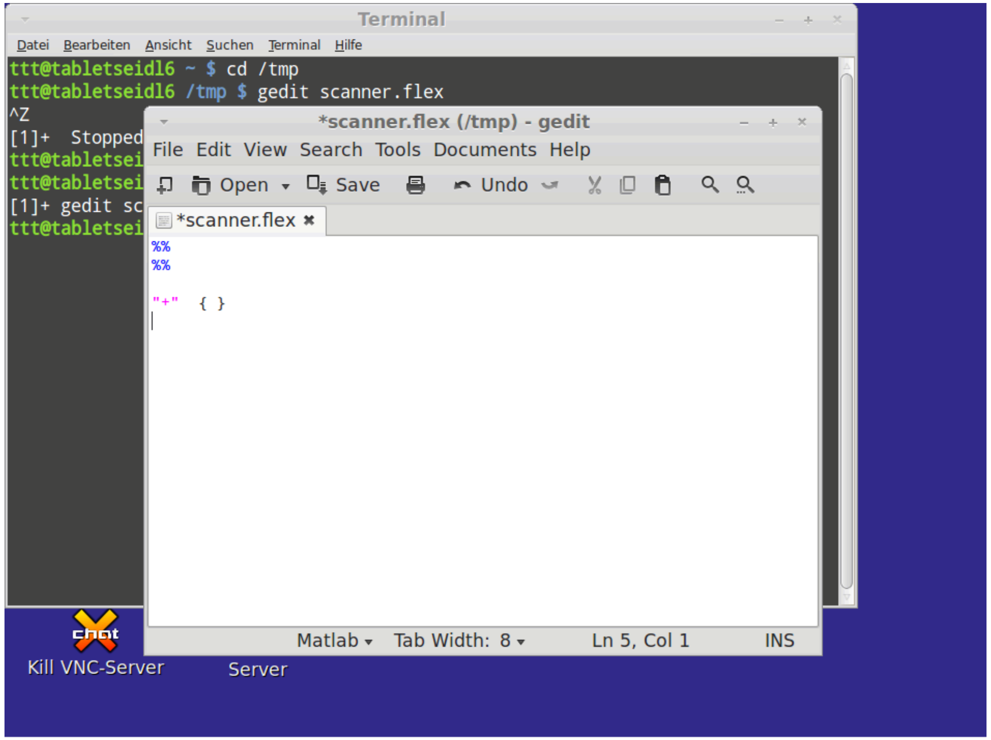
- "`.`" matches all characters different from "`\n`".
- For every state we generate the scanner respectively.
- Method `yybegin (STATE);` switches between different scanners.
- Comments might be directly implemented as (admittedly overly complex) token-class.
- Scanner-states are especially handy for implementing preprocessors, expanding special fragments in regular programs.

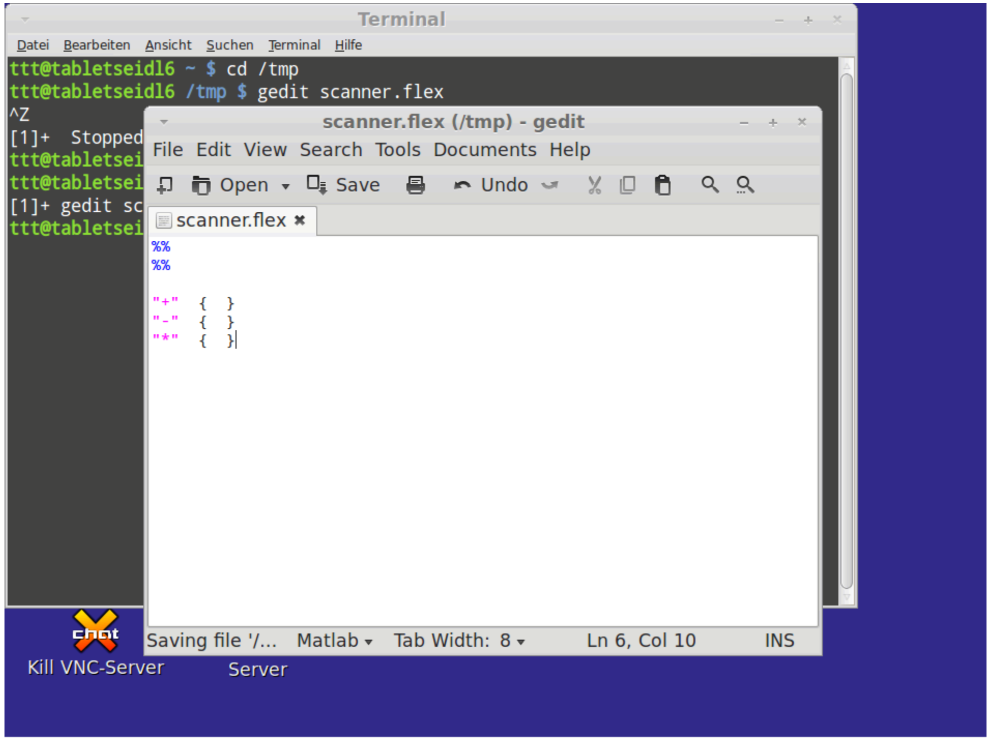

Terminal
scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
scanner.flex
Plain Text Tab Width: 8 Ln 1, Col 1 INS
Kill VNC-Server
Start VNC-Server

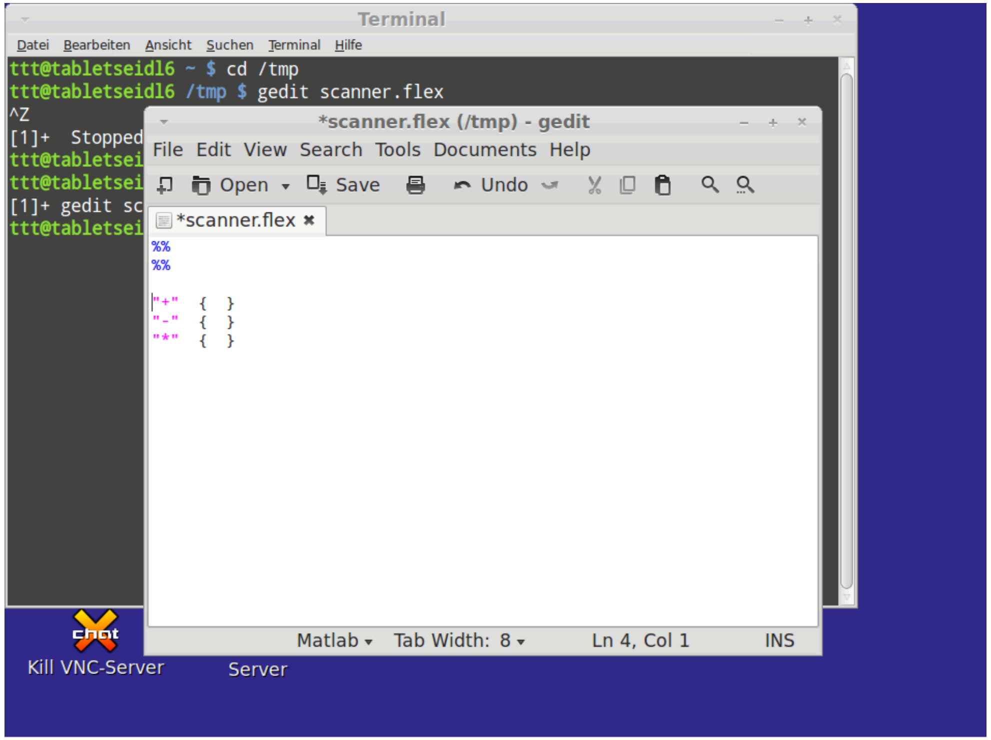

JFlex 1.4.3
Lexical specification:
Browse
Quit
Generate
Options
Output directory:
Browse
Stop
Messages:
Plain Text Tab Width: 8 Ln 1, Col 1 INS
Kill VNC-Server
Server

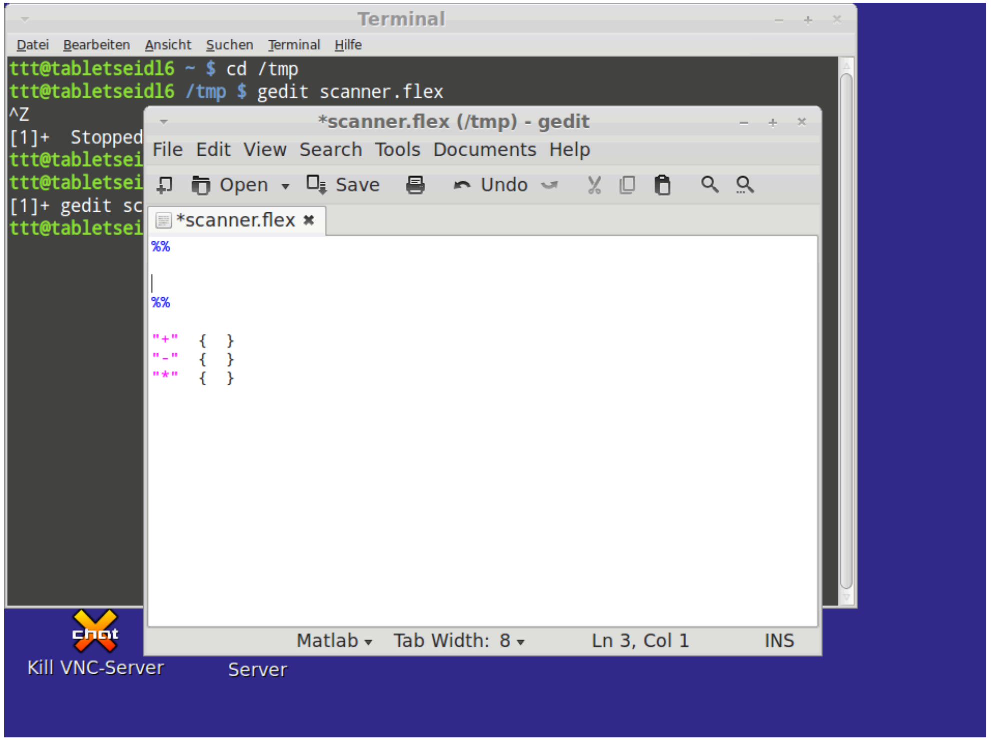

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
Plain Text Tab Width: 8 Ln 2, Col 1 INS
Kill VNC-Server
Server

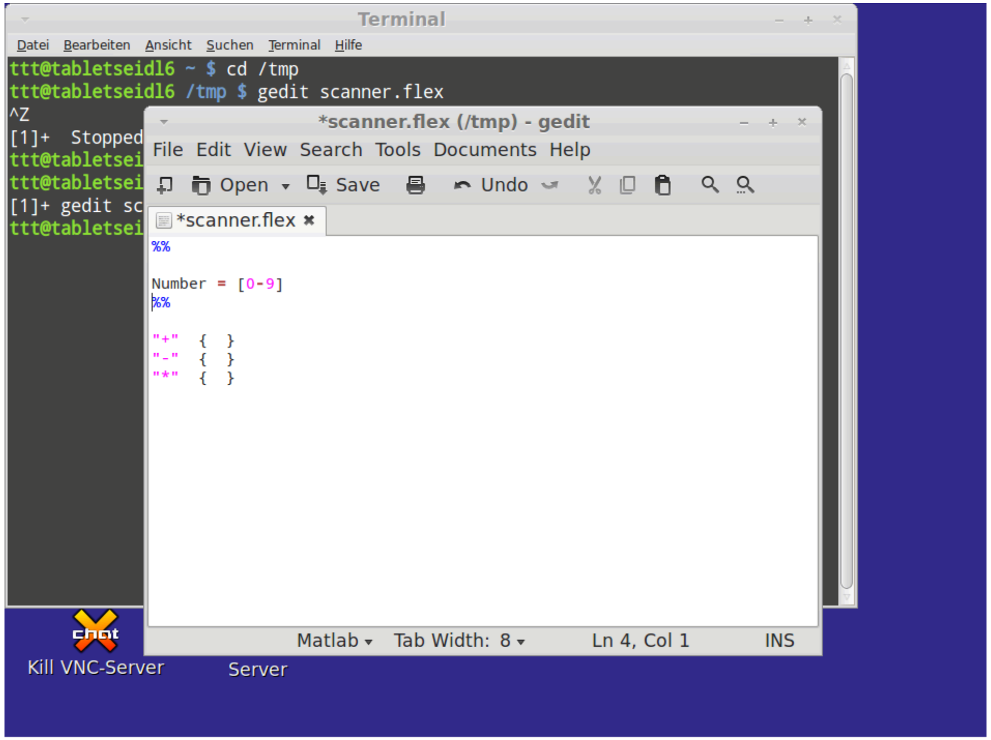

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
%%
Matlab Tab Width: 8 Ln 4, Col 1 INS
Kill VNC-Server
Server

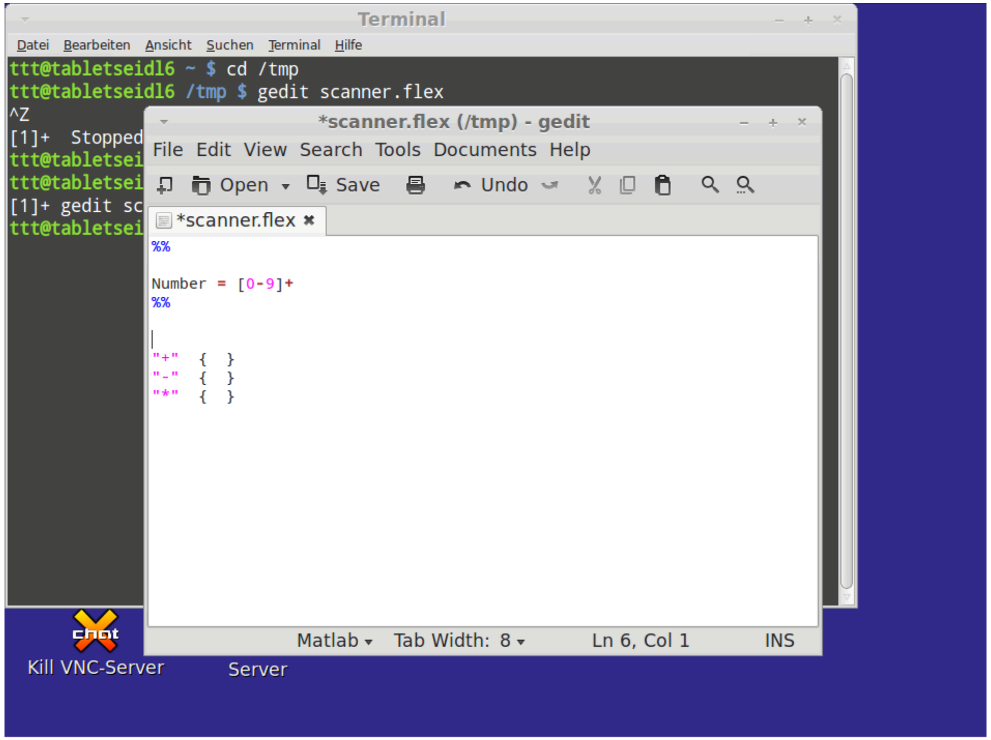

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
%%
"+" { }
Matlab Tab Width: 8 Ln 5, Col 1 INS
Kill VNC-Server
Server

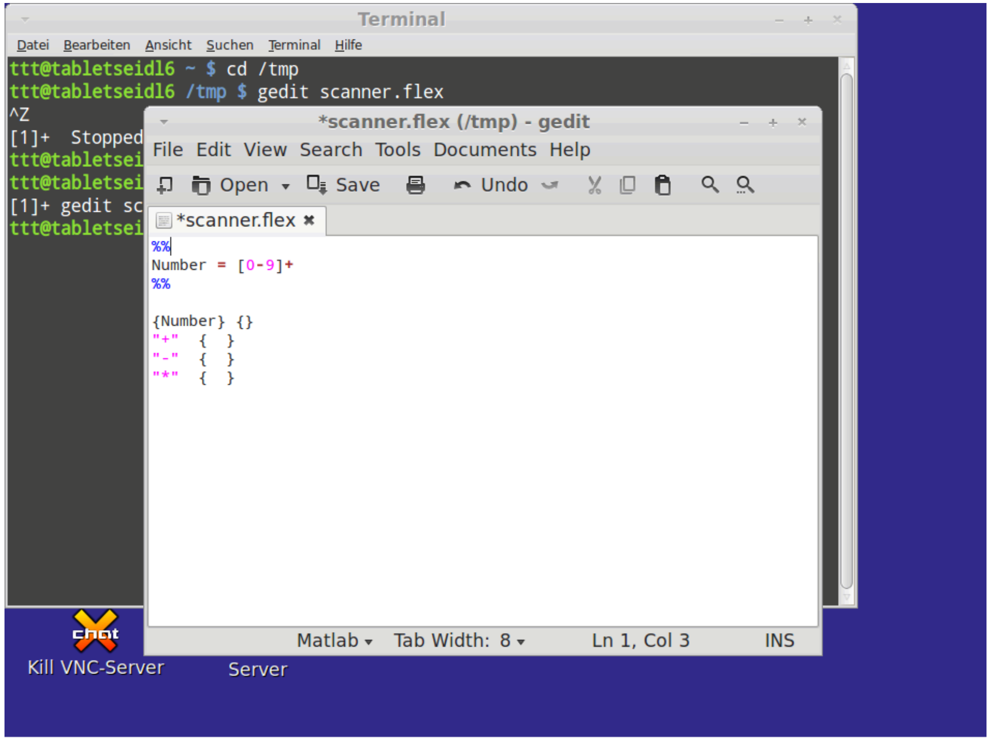

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
scanner.flex
%%
%%
"+" { }
"-" { }
"*" { }
Saving file '/... Matlab Tab Width: 8 Ln 6, Col 10 INS
Kill VNC-Server
Server

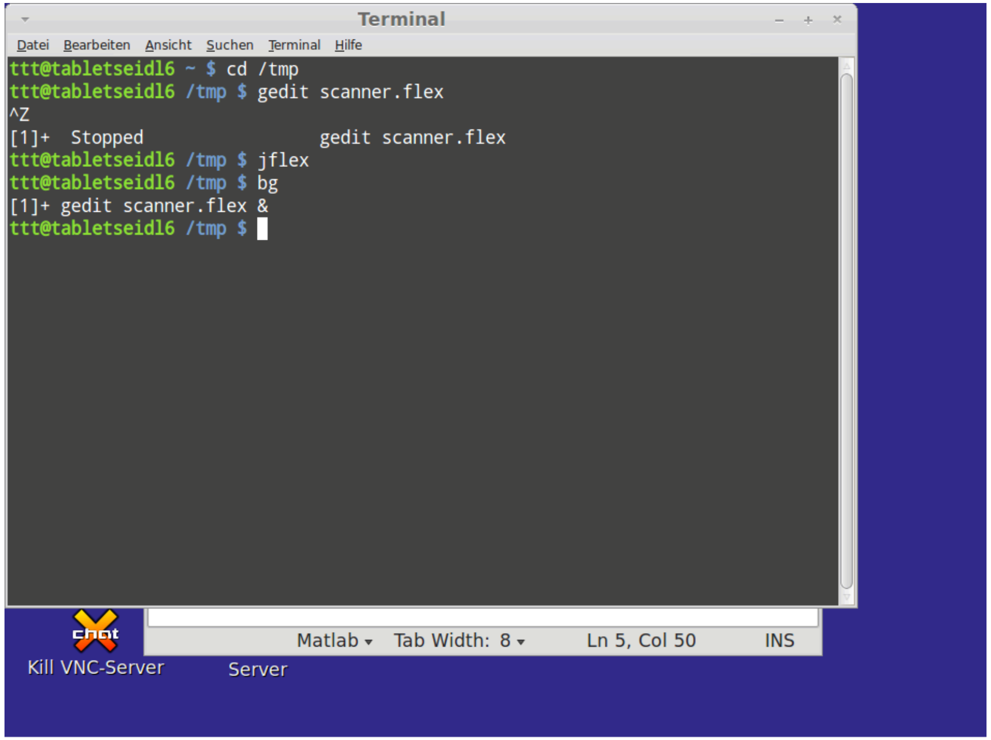

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
%%
"+" { }
"-" { }
"*" { }
Matlab Tab Width: 8 Ln 4, Col 1 INS
Kill VNC-Server
Server

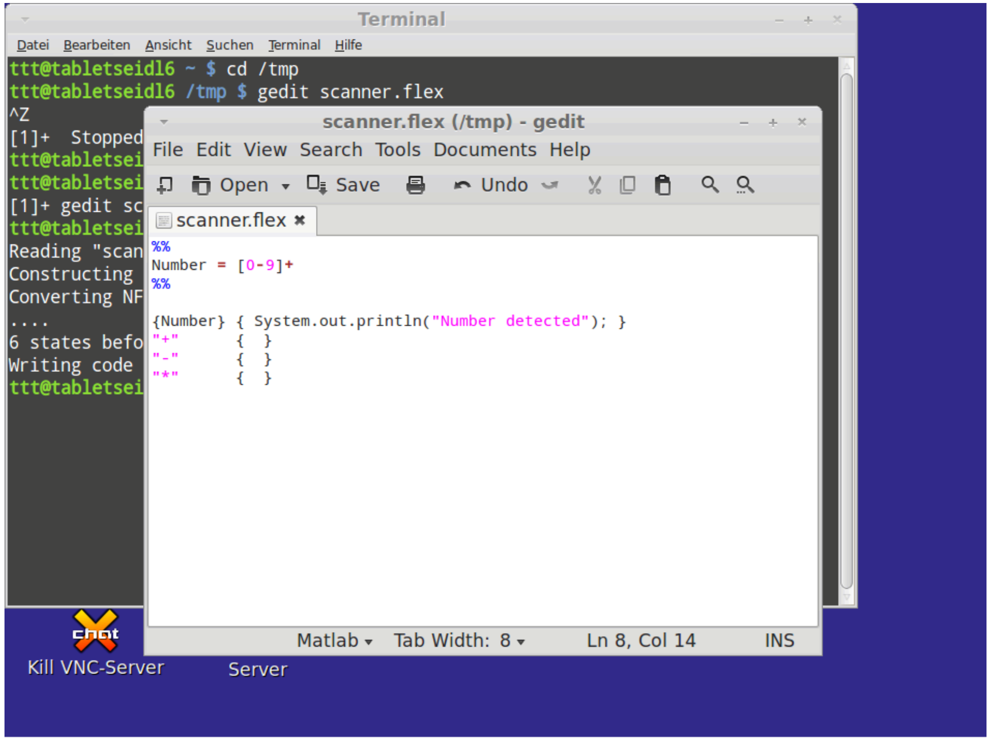

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
%%
"+" { }
"-" { }
"*" { }
Matlab Tab Width: 8 Ln 3, Col 1 INS
Kill VNC-Server
Server

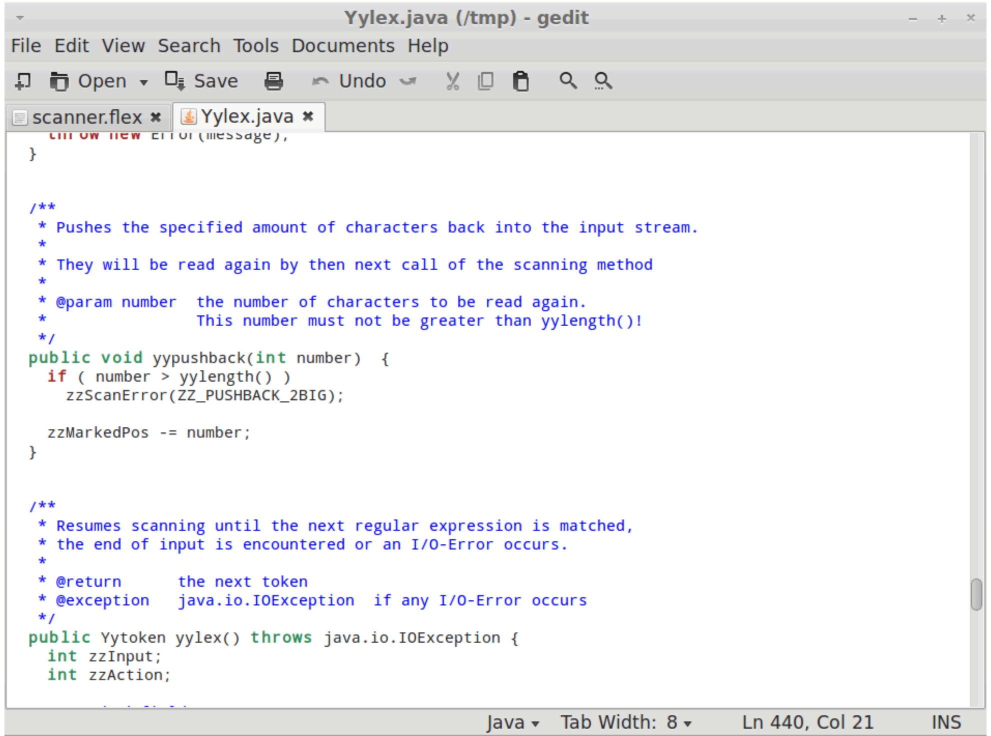

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
Number = [0-9]
%%
"+" { }
"-" { }
"*" { }
Matlab Tab Width: 8 Ln 4, Col 1 INS
Kill VNC-Server
Server

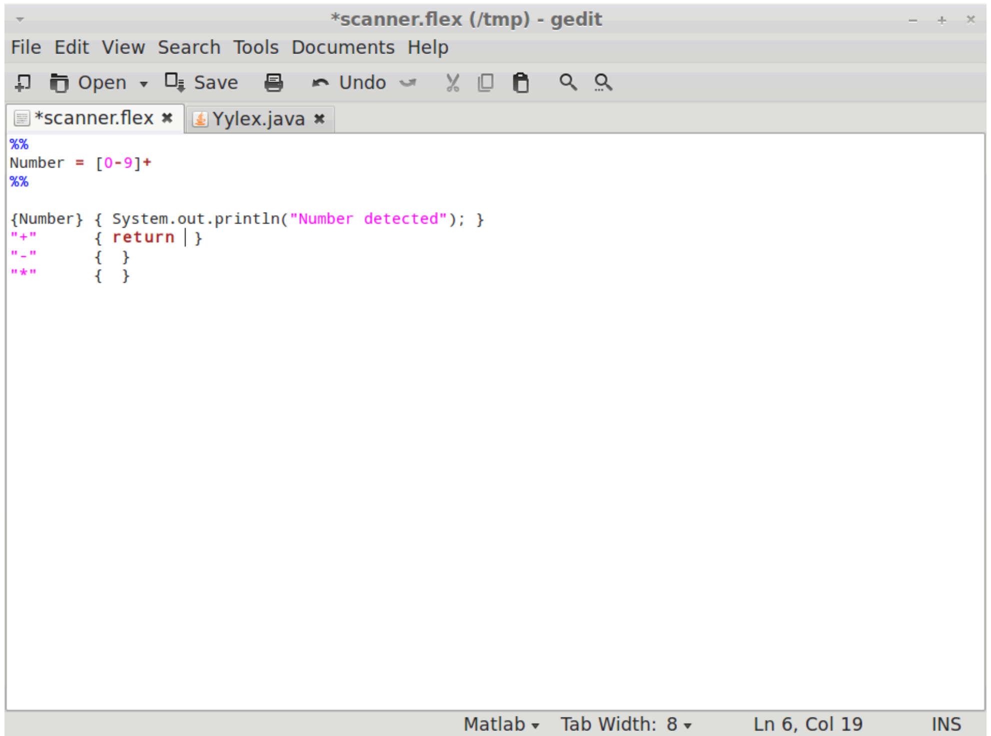

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
Number = [0-9]+
%%
"+" { }
"-" { }
"*" { }
Matlab Tab Width: 8 Ln 6, Col 1 INS
Kill VNC-Server
Server


Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped
[1]+ gedit sc
*scanner.flex (/tmp) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*scanner.flex
%%
Number = [0-9]+
%%
{Number} {}
"+" { }
"-" { }
"*" { }
Matlab Tab Width: 8 Ln 1, Col 3 INS
Kill VNC-Server
Server


Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ttt@tabletseidl6 ~ $ cd /tmp
ttt@tabletseidl6 /tmp $ gedit scanner.flex
^Z
[1]+ Stopped gedit scanner.flex
ttt@tabletseidl6 /tmp $ jflex
ttt@tabletseidl6 /tmp $ bg
[1]+ gedit scanner.flex &
ttt@tabletseidl6 /tmp $
Matlab Tab Width: 8 Ln 5, Col 50 INS
Kill VNC-Server
Server







Topic:

Syntactic Analysis

Syntactic Analysis

Token-Stream → Parser → Syntaxtree

- Syntactic analysis tries to integrate Tokens into larger program units.

Syntactic Analysis

Token-Stream → Parser → Syntaxtree

- Syntactic analysis tries to integrate Tokens into larger program units.
- Such units may possibliy be:
  → Expressions;
  → Statements;
  → Conditional branches;
  → loops; ...

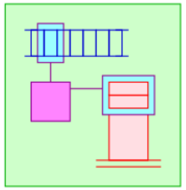
Discussion:

In general, parsers are not developed by hand, but generated from a specification:

Specification ➡ **Generator** ➡ Parser

Discussion:

In general, parsers are not developed by hand, but generated from a specification:

$E \to E\{op\}E$ ➡ **Generator** ➡ 

Specification of the hierarchical structure: contextfree grammars

Generated implementation: Pushdown automata + X

Chapter 1:
Basics of Contextfree Grammars

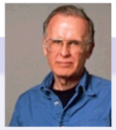
Basics: Context-free Grammars

- Programs of programming languages can have arbitrary numbers of tokens, but only finitely many Token-classes.
- This is why we choose the set of Token-classes to be the finite alphabet of terminals T.
- The nested structure of program components can be described elegantly via context-free grammars...

Basics: Context-free Grammars

- Programs of programming languages can have arbitrary numbers of tokens, but only finitely many Token-classes.
- This is why we choose the set of Token-classes to be the finite alphabet of terminals T.
- The nested structure of program components can be described elegantly via context-free grammars...


Noam Chomsky


John Backus

Definition:

A context-free grammar (CFG) is a 4-tuple $G = (N, T, P, S)$ with:

- N the set of nonterminals,
- T the set of terminals,
- P the set of productions or rules, and
- $S \in N$ the start symbol

Conventions

The rules of context-free grammars take the following form:

$$\boxed{A} \rightarrow \boxed{\alpha} \quad \text{with} \quad A \in N\,, \quad \alpha \in \boxed{(N \cup T)^*}$$

Conventions

The rules of context-free grammars take the following form:

$$A \to \alpha \quad \text{with} \quad A \in N\,, \quad \alpha \in (N \cup T)^*$$

... for example:

$$\begin{array}{|c|c|l} \boxed{S} & \boxed{\to} & \boxed{a\,S\,b} \\ \boxed{S} & \boxed{\to} & \boxed{\epsilon} \end{array}$$

Specified language: $\{a^n b^n \mid n \geq 0\}$

Conventions

The rules of context-free grammars take the following form:

$$A \to \alpha \quad \text{with} \quad A \in N\,, \quad \alpha \in (N \cup T)^*$$

... for example:

$$\begin{array}{lcl} S & \to & a\,S\,b \\ S & \to & \epsilon \end{array}$$

Specified language: $\{a^n b^n \mid n \geq 0\}$

Conventions:

In examples, we specify nonterminals and terminals in general implicitely:

- nonterminals are: $A, B, C, \ldots, \langle\text{exp}\rangle, \langle\text{stmt}\rangle, \ldots;$
- terminals are: $\boxed{a, b, c, \ldots, \text{int}, \text{name}, \ldots;}$

... further examples:

$$\begin{array}{lcl} S & \to & \langle\text{stmt}\rangle \\ \langle\text{stmt}\rangle & \to & \langle\text{if}\rangle \;\mid\; \langle\text{while}\rangle \;\mid\; \langle\text{rexp}\rangle; \\ \langle\text{if}\rangle & \to & \text{if}\ (\ \langle\text{rexp}\rangle\ )\ \langle\text{stmt}\rangle\ \text{else}\ \langle\text{stmt}\rangle \\ \langle\text{while}\rangle & \to & \text{while}\ (\ \langle\text{rexp}\rangle\ )\ \langle\text{stmt}\rangle \\ \langle\text{rexp}\rangle & \to & \text{int} \;\mid\; \langle\text{lexp}\rangle \;\mid\; \langle\text{lexp}\rangle = \langle\text{rexp}\rangle \;\mid\; \ldots \\ \langle\text{lexp}\rangle & \to & \text{name} \;\mid\; \ldots \end{array}$$