Script **generated by TTT**

Title: Petter: Compiler Construction (30.04.2020) - 07: Scanner Implementation

Date: Wed Apr 22 16:31:42 CEST 2020

Duration: 18:50 min

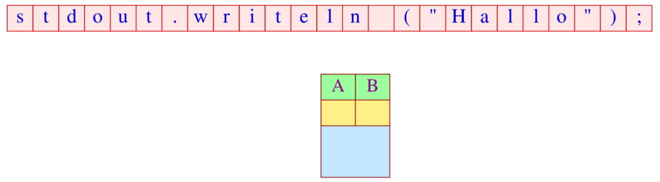
Pages: 5

Chapter 5: Scanner design

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.



Extension: States

- Now and then, it is handy to differentiate between particular scanner states.
- In different states, we want to recognize different token classes with different precedences.
- Depending on the consumed input, the scanner state can be changed

Example: Comments

Within a comment, identifiers, constants, comments, ... are ignored

Input (generalized):

a set of rules:

```
⟨state⟩  {  e_1      { action_1   yybegin(state_1); }
            e_2      { action_2   yybegin(state_2); }
               ...
            e_k      { action_k   yybegin(state_k); }
         }
```

- 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.