Kapitel 4

Einführung in den Scannergenerator Flex

Generatoren für die lexikalische Analyse

- Scannergeneratoren werden eingesetzt um die lexikalische Analyse eines Compilers möglichst kompakt und einfach zu spezifizieren (eine Domain Specific Language).
- Eingabe: Ein Zeichenstrom (ASCII, UTF-8, ...)
- Ausgabe lässt sich meist in der Beschreibungssprache spezifizieren. Mehrere Möglichkeiten zur Anbindung eines Parsers:
  - Push-Schnittstelle: Bei erkanntem Token wird eine Methode des Parsers aufgerufen
  - Pull-Schnittstelle: Der Parser ruft eine Methode im Lexer auf um das nächste Token zu bekommen. Meist am einfachsten zu programmieren.
  - Pufferung: Tokenstrom wird vor Syntaxanalyse teilweise oder komplett vorberechnet. Dies kann günstiger für den Cache sein.
- Typischerweise keine automatische Unterstützung für Symboltabellen (da oft sehr sprachspezifisch) und Fehlerbehandlung.

Verbreitete Generatoren

- *Lex/Flex*: Werkzeug zur Generierung von Scannern bekannt aus dem Unix-/C-Umfeld. Spezifikation von regulären Ausdrücken erzeugt Tabellengesteuerten Scanner.
- Flex ist eine Open-Source Implementierung von Lex.
- Zahlreiche Portierungen auf andere Programmiersprachen. Beispiele: *JLex* (Java), *CsLex* (C#), *Alex* (Haskell), ...
- Es gibt auch Scanner die nicht mit regulären Ausdrücken/endlichen Automaten arbeiten. Ein bekannter Vertreter ist *antlr* der $\mathrm{LL}(k)$ Grammatiken benutzt.

Vor- und Nachteile von Generatoren

Vorteile:

- Die Spezifikation von Scannern ist kompakt
  $\longrightarrow$ verständlich und gut wartbarer Code, weniger Fehler
- Automatische Konsistenzprüfung
- Kurze Entwicklungszyklen, schnelles Prototyping möglich
- Scannergeneratoren erzeugen sehr performanten Code.

Nachteile:

- Neue Sprache/Werkzeug muss erlernt werden
- Integration in Buildsystem notwendig
- Fehlersuche oft sehr mühsam da generierter Code für Menschen schlecht lesbar
- Fehlende Flexibilität; In der Praxis gibt es häufig kleinere Ausnahmen die man nur umständlich oder gar nicht mit Generatoren umsetzen kann.

Scanner für XML

XML-Dateien lassen sich in Tokens zerlegen. Beispiel:

```
<?xml version='1.0'?>
<!-- my personal books -->
<books>
    <book name='Goedel, Escher, Bach'/>
</books>
```

: Strings
: XML-Namen
: weitere Tokens

Scanner für XML

XML-Dateien lassen sich in Tokens zerlegen. Beispiel:

```
<?xml version='1.0'?>
<!-- my personal books -->
<books>
    <book name='Goedel, Escher, Bach'/>
</books>
```

: Strings
: XML-Namen
: weitere Tokens

Reguläre Ausdrücke

Spezifikation Regulärer Ausdrücke in Flex (Auszug):

`x`	Zeichen 'x' erkennen
`"xy"`	Zeichenkette xy erkennen (ohne Interpretation)
`\x`	Zeichen 'x' erkennen (ohne Interpretation)
`.`	Jedes Zeichen außer Zeilenumbruch erkennen
`[xyz]`	Zeichen 'x', 'y' oder 'z' erkennen
`[abj-oZ]`	Zeichen 'a', 'b', 'Z' oder Zeichen von 'j' bis 'o' erkennen
`[^A-Z]`	Alle Zeichen außer den Zeichen von 'A' bis 'Z' erkennen
`x|y`	*x* oder *y* erkennen
`(x)`	*x* erkennen (verändert die Bindung)
`x*`	0, 1 oder mehrere Vorkommen von *x* erkennen
`x+`	1 oder mehrere Vorkommen von *x* erkennen
`x?`	0 oder 1 Vorkommen von *x* erkennen
`{Name}`	Expansion der Definition *Name*
`\t,\n,\r`	Tabulator, Zeilenumbruch, Wagenrücklauf erkennen

Spezifikation

```
NameStartChar   [a-zA-Z]
NameChar        {NameStartChar}|[\-\.0-9]
Name            {NameStartChar}({NameChar})*
Comment         "<!--"([^\-]|"-"[^\-])*"-->"
String          \'[^\']*\'|\"[^\"]*\"
Token           "<?"|"</"|"<"|"/>"|">"|"?>"|"="
```

```
%%
{String}        printf("String: %s\n", yytext);
{Name}          printf("Name: %s\n", yytext);
{Comment}       /* skip */
{Token}         printf("Token: %s\n", yytext);
.|\n            /* skip */

%%
int main(void) {
   yylex();
   return 0;
}
```


Spezifikation

```
NameStartChar     [a-zA-Z]
NameChar          {NameStartChar}|[\-\.0-9]
Name              {NameStartChar}({NameChar})*
Comment           "<!--"([^\-]|"-"[^\-])*"-->"
String            \'[^\']*\'|\"[^\"]*\"
Token             "<?"|"</"|"<"|"/>"|">"|"?>"|"="


%%
{String}          printf("String: %s\n", yytext);
{Name}            printf("Name: %s\n", yytext);
{Comment}         /* skip */
{Token}           printf("Token: %s\n", yytext);
.|\n              /* skip */

%%
int main(void) {
   yylex();
   return 0;
}
```


Verwendung von Flex

```
$ flex -o scanner.c scanner.l
$ gcc scanner.c -lfl -o scanner
$ ./scanner < test.xml
Token: <?
Name: xml
Name: version
Token: =
String: "1.0"
Token: ?>
Token: <
Name: books
Token: >
Token: <
Name: book
Name: name
Token: =
String: 'Goedel, Escher, Bach'
Token: />
Token: </
Name: books
Token: >
```

test.xml:

```
<?xml version="1.0"?>
<!-- my personal books -->
<books>
  <book name=
    'Goedel, Escher, Bach'/>
</books>
```