Tutorium: Einführung in die BibTEX-Programmierung

Bernd Raichle

DANTE 2002, Erlangen

Inhaltsverzeichnis

1 Ein erstes einfaches Beispiel

1.1 LaTeX-Dokument

Folgendes Dokument wird uns, mit leichten Abwandlungen, für einige Zeit lang durch die vorgestellten BibTeX-Style-Beispiele begleiten.

```
% simple.tex
\documentclass[a4paper]{article}
\newenvironment{book}[1]{Buch ``#1'':%
  \begin{description}%
    \renewcommand\author[1]{\item[Autor:] ##1.}%
    \renewcommand\title [1]{\item[Titel:] ##1.}%
}{%
  \end{description}
}
\begin{document}
\nocite{*}
\bibliography{simple}
\bibliographystyle{simple}
\end{document}
```

Es ist sehr einfach gehalten: Wichtig sind die letzten drei Zeilen vor dem `\end{document}`, in der die Literaturdatenbank namens `simple.bib` und der BibTeX-Style `simple.bst` deklariert werden. Durch `\nocite{*}` wird dafür gesorgt, daß alle in der Datenbank existierenden Einträge bearbeitet werden.

Die Umgebung `book` und die beiden Anweisungen `\author` und `\title` werden vom gleich vorgestellten Beispiel-BibTeX-Style als Markup-Anweisungen erzeugt.

1.2 Literaturdatenbank

Durch die ersten Beispiele wird uns folgende kleine Literaturdatenbank begleiten:

```
% simple.bib
@book{eins,
  author = {Bernd Raichle},
  title  = {Bib\TeX-Programmierung}
}

@book{zwei,
  author = {A.U. Thor},
  title  = {Das Buch}
}
```

1.3 Formatierung mit LaTeX

Der erste Formatierdurchlauf mit LaTeX ...

```
> latex simple
This is TeX, Version 3.14159 (Web2C 7.3.1)
(simple.tex
LaTeX2e <2000/06/01>
Babel <v3.6x> and hyphenation patterns for american, french, german,
ngerman, nohyphenation, loaded.
(/koblocal0/local/teTeX/share/texmf/tex/latex/base/article.cls
Document Class: article 2000/06/07 v1.4a Standard LaTeX document class
(/koblocal0/local/teTeX/share/texmf/tex/latex/base/size10.clo))
No file simple.aux.
No file simple.bbl.
(simple.aux) )
No pages of output.
Transcript written on simple.log.
```

... erzeugt folgende `.aux`-Datei:

```
% simple.aux
\relax
\citation{*}
\bibdata{simple}
\bibstyle{simple}
```

Wie man sieht, werden die Anweisungen `\bibliography`, in der man auch mehr als eine Literaturdatenbankdatei angeben kann, und `\bibliographystyle` in eine etwas andere Form in die `.aux`-Datei geschrieben. Außerdem findet sich für jedes im Dokument vorkommendes `\cite` und `\nocite` ebenso eine entsprechende Zeile in der `.aux`-Datei.

1.4 Generierung des Literaturverzeichnisses mit BibTeX

Der anschließende BibTeX-Lauf erzeugt ...

```
> bibtex simple
This is BibTeX, Version 0.99c (Web2C 7.3.1)
The top-level auxiliary file: simple.aux
The style file: simple.bst
Database file #1: simple.bib
```

... dann das formatierte Literaturverzeichnis:

```
% simple.bbl
\begin{book}{eins}
  \author{Bernd Raichle}
  \title{Bib\TeX-Programmierung}
\end{book}
\begin{book}{zwei}
  \author{A.U. Thor}
  \title{Das Buch}
\end{book}
```

Im Unterschied zu den Standard-BibTEX-Styles wählte ich hier bewußt eine andere Markup-Struktur. Üblicherweise sieht eine `bbl`-Datei wie folgt aus:

```
% standard-bst.bbl
\begin{thebibliography}{1}
\bibitem{eins} Bernd Raichle: \emph{Bib\TeX-Programmierung}.
\bibitem{zwei} A.U. Thor: \emph{Das Buch}.
\end{thebibliography}
```

BibTEX benötigt immer eine `.aux`-Datei, die *genau einen* Eintrag mit mindestens einer Literaturdatenbankdatei und *genau einen* Eintrag für den zu verwendenden BibTEX-Style enthalten muss. Außerdem wird *mindestens eine* Literaturreferenz benötigt. Die gelesene `.aux` wie auch den verwendeten BibTEX-Style als auch alle gelesenen Literaturdatenbankdateien werden im Protokoll des BibTEX-Laufs aufgeführt.

1.5 BibTEX-Style „simple.bst“

Mit diesem *vollständigen* BibTEX-Style, bestehend aus nur 14 nichtleeren Zeilen, wird das vorstehende Ergebnis erzeugt.

```
% simple.bst
ENTRY
  { author
    title
  }
  {}
  {}

FUNCTION {book}
{ "\begin{book}{" write$ cite$ write$ "}" write$ newline$
  "  \author{" author * "}" * write$ newline$
  "  \title{" title "}" * * write$ newline$
  "\end{book}" write$ newline$
}

READ
ITERATE {call.type$}
```

Dieses Beispiel zeigt den prinzipiellen Aufbei eines BibTEX-Styles:

1. Deklaration der benutzten Felder (`author`, `title`) in den Literaturdatenbanken,
2. Definition von Funktionen für jeden verwendeten Eintragstyp (`book`),
3. Einlesen der Literaturdatenbanken mit `READ`,
4. iteratives Bearbeiten jedes Eintrags durch Aufruf der zuvor definierten Funktion für den jeweiligen Eintragstyp.

2 Aufbau von BibTEX-Styles

2.1 Bestandteile eines BibTEX-Styles

Jeder BibTEX-Style hat folgende Bestandteile:

- Deklaration der verwendeten Felder und Integer-/String-Variablen, auf die im Programm zugegriffen wird, für jeden Literatureintrag (Anweisung: `ENTRY`).
- Deklaration aller global verwendeten Integer-/String-Variablen (Anweisungen: `STRING` und `INTEGERS`).
- Definition aller Makros, die in der Literaturdatenbank verwendet werden (Anweisung: `MACRO`).
- Definition aller Funktionen (Anweisung: `FUNCTION`).
- Anweisung `READ` zum Einlesen der Literaturdatenbank.
- Anweisung `EXECUTE` zum Aufruf einer Funktion.
- Anweisungen `ITERATE` und `REVERSE` zum Aufruf einer Funktion, die auf alle Literatureinträge in gegebener bzw. umgekehrter Reihenfolge angewandt wird.

– Anweisung `SORT` zum Sortieren der Literatureinträge anhand der String-Variable `sort.key$`.

Die Reihenfolge, in der diese Anweisungen stehen müssen, ist nicht ganz beliebig: Vor dem Einlesen mit `READ` müssen mit `ENTRY` die zu verwendeten Felder und mit `MACRO` alle in der Datenbank verwendeten Makros deklariert bzw. definiert werden. Weiters können die Anweisungen zur Manipulation von Einträgen (`SORT`, `ITERATE`, `REVERSE`) erst ausgeführt werden, nachdem mit `READ` die Einträge aus der Datenbank gelesen wurden. Schließlich können Funktionen erst aufgerufen und Variablen erst verwendet werden, wenn diese vorher definiert bzw. deklariert wurden.

2.2 Struktur eines BibTEX-Styles

Jeder BibTEX-Style hat folgende grobe Struktur:

1. Deklaration der verwendeten Literatureintragstypen und -felder, Lesen der Literatureinträge:

```
% Deklaration der Felder und Variablen
ENTRY
  { ... Felder ...
  }
  { ... Integer-Variablen ...}
  { ... String-Variablen ...}

INTEGERS { ... globale Integer-Variablen ... }
STRINGS { ... globale String-Variablen ... }

% Definition von Makros und Funktionen
MACRO { ... }
FUNCTION { ... }

% Initialisierung von Variablen
EXECUTE { ... }

% Einlesen der Literaturdatenbank
READ
```

2. Bearbeitung der Literatureinträge, Erzeugung des Literaturverzeichnisses:

```
% Deklaration weiterer Variablen
INTEGERS { ... globale Integer-Variablen ... }
STRINGS { ... globale String-Variablen ... }

% Definition weiterer Funktionen
FUNCTION { ... }

% Initialisierung von Variablen
EXECUTE { ... }


% Manipulation der Datenbankeintraege
ITERATE { ... }
REVERSE { ... }
SORT


% Initialisierung vor und Beginn der Ausgabe
EXEUTE { ... }

% Erzeugung der Ausgabe pro Eintrag
ITERATE {call.type$}

% Abschluss der Ausgabe
EXEUTE { ... }
```

Wie vorher schon beschrieben, enthält ein BibTEX-Style irgendwann die Anweisung `READ`, die die Einträge in allen angegebenen Literaturdatenbanken, also den Dateien mit Endung `.bib`, einliest. Vor dieser Anweisung müssen alle Eintragsfelder, auf die man im folenden zugreifen will, mit `ENTRY` deklariert werden. Ebenso müssen auch alle in den Datenbanken verwendeten Abkürzungen entweder in dieser mit `@String...` oder im BibTEX-Style mit der Anweisung `MACRO` definieren. Somit startet jede BibTEX-Style üblicherweise mit eben diesen Deklarationen und sonstigen Initialisierungen.

Im zweiten Teil eines BibTEX-Styles wird das Ergebnis erzeugt. Dort findet man irgendwann die Zeile `ITERATE{call.type$}`. Um für jeden Eintrag die gewünschte Formatierung festzulegen, müssen dazu vorher entsprechende Funktionen definiert werden. Diese Funktionen verwenden Unterfunktionen und greifen auf Variablen zur Zwischenspeicherung zu, diese vor Verwendung definiert werden. Wenn Einträge umsortiert werden sollen, so macht man dies auch zwischen Einlesen der Einträge und dem Erzeugen des Gesamtergebnisses. Somit findet man in diesem Abschnitt weitere Funktionsdefinitionen und Variablendeklarationen, Anweisungen wie `SORT`, `ITERATE` und `REVERSE`

3 Datentypen, Variablen, vordefinierte Funktionen

3.1 Datentypen

Als *Datentypen* gibt es nur ganze Zahlen (Integer) und Zeichenketten (Strings).

Zeichenketten-Konstanten

- `""`
- `"a"`
- `"ein Name"`, ...

Zahl-Konstanten

- `#0`
- `#1`, ...

3.2 Variablen

Variablen- und Funktionsnamen beginnen mit einem Buchstaben, gefolgt von Buchstaben, Ziffern und sonstigen darstellbaren Zeichen außer den folgenden zehn Zeichen:

```
" # % ' ( ) , = { }
```

Klein- und Großschreibung wird *nicht* unterschieden. Alle Variablen müssen *vorher* explizit als Zahl- oder Zeichenkettenvariable mit `INTEGERS`, `STRINGS` oder `ENTRY` deklariert werden.

Wert einer Variable

- `label`

Name einer Variable

- `'label`
- `"label" quote$`

3.3 Vordefinierte Funktionen

BibTEX stellt dem Programmierer 32 vordefinierte Funktionen zur Verfügung. Ein Programmierer kann mit der Anweisung `FUNCTION` eigene Funktionen definieren, die diese vordefinierte Funktionen verwenden, um komplexere Dinge zu bewerkstelligen.

Fast alle Funktionsnamen der internen Funktionen enden mit einem `$`. Um selbstdefinierte von den vordefinierte Funktionen leicht unterscheiden zu können, ist es daher ratsam, in diesen kein Dollarzeichen `$` zu verwenden.

Die vordefinierten Funktionen lassen sich grob in folgende Klassen einteilen:

Wertzuweisung

- *integer variable* `:=`
- *string variable* `:=`

Prädikate auf Zahlen

- *integer integer* `<`
- *integer integer* `>`
- *integer integer* `=`

Rechenoperationen auf Zahlen

- *integer integer* `+`
- *integer integer* `-`

Wandlung String, Char, Integer

- *string* `chr.to.int$`
- *integer* `int.to.chr$`
- *integer* `int.to.str$`

String-Information

- *string* `num.names$`
- *string* `text.length$`
- *string* `width$`

String-Manipulation

- *string integer integer* `substring$`
- *string integer* `text.prefix$`
- *string* `purify$`
- *string string* `change.case$`
- *string integer string* `format.name$`
- *string* `add.period$`

Vordefinierte Variablen

- `sort.key$` (String, pro Eintrag)
- `entry.max$` (Integer, maximale String-Länge)
- `global.max$` (Integer, maximale String-Länge)

Konkatenation von Zeichenketten

- *string string* `*`

Prädikate auf Zeichenketten

- *string* `empty$`
- *string* `missing$`
- *string string* `=`

Literatureintragsdaten

- `cite$`
- `type$`
- `preamble$`

Kontrollfluß

- *integer literal literal* `if$`
- *literal literal* `while$`
- `call.type$` (Indirekter Aufruf)
- `skip$` (No-Op)

Spezielle Stack-Operationen

- *literal* („Push“)
- *literal* `pop$`
- *literal* `duplicate$`
- *literal literal* `swap$`
- *literal* `top$` (Debugging)
- *literal* * `stack$` (Debugging)

Ausgabe bbl-Datei

- *string* `write$`
- `newline$`

Protokollausgabe

- *string* `warning$`

Sonstige

- *string* `quote$`

4 UPN ist ungewohnt ...

4.1 Postfix-Notation von BibTEX

Alle Anweisungen in einem BibTEX-Style werden in Postfix-Notation bzw. in der *umgekehrten polnischen Notation* (UPN) gegeben. Der prinzipielle Aufbau ist immer

$$operand_1 \ \ldots \ operand_n \ operator$$

Beispiele

```
#1 #2 +
"a" 'label :=
#0 { "gleich 0" warning$ } { "ungleich 0" warning$ } if$
```

Dies ist zuerst sehr ungewohnt, da die meisten Programmiersprache eine Infix- oder Präfix-Notation besitzen. Es gibt jedoch neben BibTEX noch weitere Programmiersprachen, wie beispielsweise PostScript, mit Postfix-Notation.

Merkregel: Ausführung der Anweisung in passiver Form formulieren (Beispiel: „Die Konstanten 1 und 2 werden addiert.“)

4.2 Infix-/Präfix-Notation anderer Programmiersprachen

Zum Vergleich habe ich hier einmal obige Zeilen in einer wohl etwas gewohnteren Form hingeschrieben.

Bei diesen Programmiersprachen stehen die Operatoren zwischen bzw. vor den Operanden.

Beispiele Infix-Notation

```
1 + 2
label := "a"
if 0 then print("gleich 0") else print("ungleich 0") endif
```

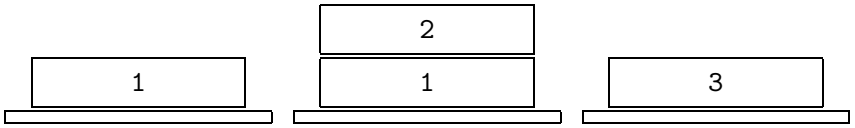
Beispiele Präfix-Notation

```
add(1, 2)
assign(label, "a")
if(0, print("gleich 0"), print("ungleich 0"))
```

```
(+ 1 2)
(setq label "a")
(if 0 (print "gleich 0") (print "ungleich 0"))
```

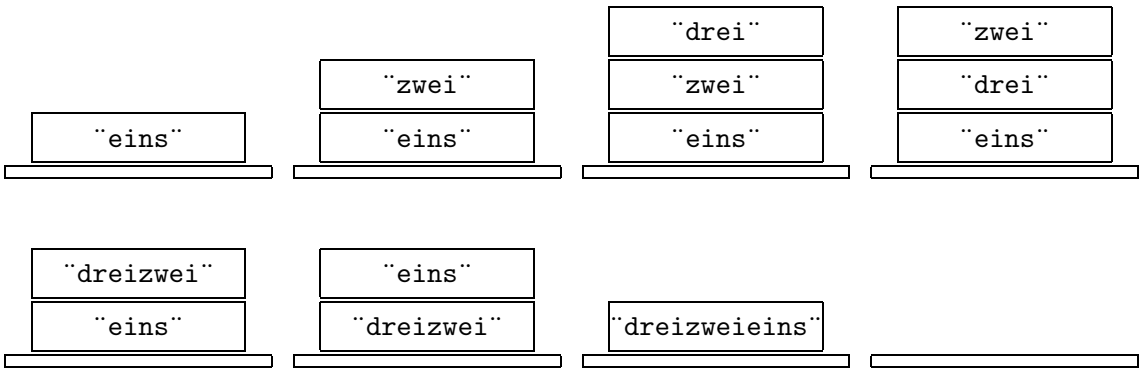
4.3 UPN: Manipulation eines Stapels

Wie Operationen in UPN ausgeführt werden, läßt sich am anschaulichsten mit einem Stapel beschreiben, auf den man Dinge legen und von dem man Dinge nehmen kann. Dabei ist wichtig, daß man dies nur zuoberst tun kann. Jede Operation mit Argumenten nimmt diese von unserem gedachten Stapel herunter, tut mit diesen etwas oder berechnet etwas und legt ein eventuell vorhandenes Ergebnis am Schluß wieder auf diesen Stapel.




```
#1 #2 +
```

Konstanten werden einfach auf den Stapel gelegt. Als erstes wird somit zuerst die Konstante 1 auf den Stapel gelegt, danach die Konstante 2 obenauf. Die Operation + erwartet zwei Argumente, dies sollten Zahlen sein, addiert diese und legt das Ergebnis wieder auf den Stapel ab.+ nimmt zuerst die 2, dann die 1 vom Stapel, addiert beide zu 3 und legt dieses Ergebnis auf den Stapel ab.




```
"eins" "zwei" "drei" swap$ * swap$ * pop$
```

Auch hier werden zuerst die drei String-Konstanten nacheinander auf den Stapel gelegt. Die Operation `swap$` erwartet nun zwei Argumente und legt diese vertauscht auf den Stapel zurück. Somit werden dadurch die beiden obersten Strings `"drei"` und `"zwei"` auf dem Stapel vertauscht. * ist eine Konkatenation zweier Strings, erwartet zwei Argumente, die Strings sein müssen, und legt den zusammengefügten String auf den Stapel zurück. Hiermit

wird aus den zuvor vertauschten Strings der neue String `"dreizwei"` erzeugt. Die nachfolgenden Operationen `*` und `swap$` vertauscht erneut die beiden obersten Einträge und fügt diese zum neuen String `"dreizweieins"` zusammen. `pop$` ist eine Operation mit einem Argument, die sonst nichts tut und auch nichts auf den Stapel zurücklegt. Diese Operation verwendet man, um das oberste Element auf dem Stapel zu entfernen, wenn man dieses nicht mehr verwenden kann. Dies tun wir auch in diesem Beispiel, um am Ende „aufzuräumen“ und einen leeren Stapel zu hinterlassen.

4.4 BibTEX-Style „simple.bst“ . . . nochmals betrachtet

Wenn wir mit diesen Grundlagen über Aufbau eines BibTEX-Styles und die Programmiersprache unser erstes Beispiel nochmals betrachten, so sollte uns dieses schon etwas vertrauter vorkommen.

```
% simple.bst
ENTRY
  { author
    title
  }
  {}
  {}

FUNCTION {book}
{ "\begin{book}{" write$ cite$ write$ "}" write$ newline$
  "  \author{" author * "}" * write$ newline$
  "  \title{" title "}" * * write$ newline$
  "\end{book}" write$ newline$
}

READ
ITERATE {call.type$}
```

```
% simple.bbl
\begin{book}{eins}
  \author{Bernd Raichle}
  \title{Bib\TeX-Programmierung}
\end{book}
\begin{book}{zwei}
  \author{A.U. Thor}
  \title{Das Buch}
\end{book}
```

Durch die Anweisung `ITERATE {call.type$}` wird für jeden Literatureintrag des Typs `book` die gleichnamige Funktion aufgerufen. Diese erzeugt pro Aufruf vier Zeilen in der `bbl`-Datei.

Auf die Felder `author` und `title` eines Literatureintrags kann einfach durch den gleichnamigen in `ENTRY` deklarierten Variablenname zugegriffen werden. Man erhält hierbei den

Wert des Feldes als Zeichenkette. Den Zitierschlüssel des momentan bearbeiteten Literatureintrags erhalte ich mit der Funktion `cite$` ebenso als Zeichenkette.

Wie man am Code erkennen kann, werden die vier Ergebniszeilen auf sehr unterschiedliche Weise zusammengesetzt: Während ich die einzelnen Teile der ersten Zeile durch mehrere `write$`-Anweisungen erzeuge, baue ich die einzelnen Zeichenketten der zweiten und dritten Zeile vor dem Herausschreiben durch Konkatenation mit `*` zu einer einzigen Zeichenkette zusammen.

Die Funktion `*` erwartet zwei Zeichenketten als Operanden und hat als Ergebnis die zusammengefügte Zeichenkette. Wie man im Vergleich der zweiten und dritten Zeile sieht, kann man mehrere Zeichenketten entweder paarweise sofort zusammenfügen oder die Zeichenketten zuerst hinschreiben und dann erst die notwendigen Konkatenationen ausführen.

5 Erweiterung #1: `format.feld`

```
% erw1.bst
ENTRY
  { author
    title
  }
  {}
  {}

FUNCTION {format.feld}                %%%%% <<<<<<<<<<<<<
{ duplicate$ empty$
    { pop$ pop$ }
    { swap$ "{" * swap$ * "}" * write$ newline$ }
  if$
}

FUNCTION {book}
{ "\begin{book}{" write$ cite$ write$ "}" write$ newline$
  "  \author" author format.feld      %%%%% <<<<<<<<<<<<<
  "  \title"  title  format.feld      %%%%% <<<<<<<<<<<<<
  "\end{book}" write$ newline$
}

READ
ITERATE {call.type$}
```

Bisher wurde die Formatierung für beide Felder durch die immer wieder gleiche Abfolge von vordefinierten Funktionen erzeugt. Diese fasse ich nun in einer eigenen Funktion `format.feld` zusammen, die ich als Unterfunktion verwende. Diese neue Funktion erweitere ich gleich: Ist ein Feld nicht vorhanden oder leer, so wird keine Ausgabezeile erzeugt.

Die Funktion `format.feld` erwartet zwei Operanden: das Feld (bzw. den Wert des Feldes) und die LaTeX-Anweisung, die in die `bbl`-Datei geschrieben werden soll. Zuerst wird der

Feldwert dupliziert und mit `empty$` getestet, ob das Duplikat leer ist. Die Funktion `if$` nimmt nun das Ergebnis dieses Tests und führt abhängig davon entweder die eine oder die andere Anweisungsfolge ab. War das Feld leer oder nicht vorhanden, so wird mit den beiden Anweisungen `pop$` der verbliebene Feldwert als auch die Zeichenkette mit der LaTeX-Anweisung entfernt. Ist das Feld nicht-leer, so wird mit `swap$` der verbliebende Feldwert und die Zeichenkette mit der LaTeX-Anweisung vertauscht, damit die öffnende Klammer angefügt werden kann. Anschließend wird diese neu entstandende Zeichenkette nochmals mit dem davorstehenden Feldwert vertauscht und dann wie gewohnt Feldwert und schließende Klammer zusammengefügt.

Alternativ kann man die Funktion `format.feld` auch besser wie folgt implementieren:

```
FUNCTION {format.feld}
{ duplicate$ empty$
    { pop$ pop$ }
    { "{" swap$ "}" * * * write$ newline$ }
  if$
}
```

Hierdurch werden zuerst die öffnende Klammer, der Feldwert und dann die schließende Klammer zu einer Zeichenkette verbunden, bevor diese dann mit der LaTeX-Anweisung zusammengefügt wird. Man beachte, daß hierbei zwischendurch andere Zeichenketten entstehen, da in einer anderen Reihenfolge zusammengefügt wird.

Diese Reihenfolge ergibt leichter erfaßbaren und verstehbaren Code und ist daher zu bevorzugen.

6 Erweiterung #2: `format.feld` nochmals verbessert

```
% erw2.bst
ENTRY
  { author
    title
  }
  {}
  {}

FUNCTION {format.feld}
{ duplicate$ empty$
    { pop$ pop$ }
    { "{" swap$ "}" * * * "  \" swap$ * write$ newline$ }
                        % ^^^^^^^^^^^^^--- <<<<<<<<<<<<<
  if$
}

FUNCTION {book}
{ "\begin{book}{" write$ cite$ write$ "}" write$ newline$
  "author" author format.feld        %%%%% <<<<<<<<<<<<<
  "title"  title  format.feld        %%%%% <<<<<<<<<<<<<
  "\end{book}" write$ newline$
}

READ
ITERATE {call.type$}
```

Die LaTeX-Anweisungen, die man über das zweite Argument der Funktion `format.feld` enthalten noch ähnliche Bestandteile. Daher bietet es sich an, auch diese in die Funktion zu verlagern.

7 Erweiterung #3: `begin.bib/end.bib`

```
% erw3.bst

...

FUNCTION {begin.bib}                    %%%%% <<<<<<<<<<<<<
{ "\begin{thebibliography}{9999}" write$ newline$
}

FUNCTION {end.bib}                      %%%%% <<<<<<<<<<<<<
{ "\end{thebibliography}" write$ newline$
  "\endinput" write$ newline$
}

READ
EXECUTE {begin.bib}                     %%%%% <<<<<<<<<<<<<
ITERATE {call.type$}
EXECUTE {end.bib}                       %%%%% <<<<<<<<<<<<<
```

Falls es noch nicht bemerkt wurde: Durch unseren bisherigen BibTEX-Style und die LATEX-Makros im Dokument wurde die einzelnen Literatureinträge nicht in Listenumgebung für Literaturverzeichnisse wie `thebibliography` gepackt, jeder Eintrag war im Dokument für sich separat. Zu Beginn und am Ende einer `bbl`-Datei kann etwas sehr leicht mit `EXECUTE`-Anweisungen eingefügt werden. Der obige BibTEX-Style fügt am Anfang und am Ende der `bbl`-Datei entsprechende Zeilen ein, um die Umgebung `thebibliography` zu beginnen bzw. zu beenden.

8 Erweiterung #4: `begin.entry/end.entry`

Die Zeichenketten, die zu Beginn und am Ende eines Literatureintrags in die `bbl`-Datei geschrieben werden, enthalten gleiche Anteile. Der Markup wird ebenso verallgemeinert.

```
% erw4.bst

...

FUNCTION {begin.entry}                %%%%% <<<<<<<<<<<<<
{ "\begin{entry}{" swap$ * "}{" * cite$ * "}" * write$ newline$
}

FUNCTION {end.entry}                  %%%%% <<<<<<<<<<<<<
{ "\end{entry}" write$ newline$
}

FUNCTION {book}
{ "Buch" begin.entry                  %%%%% <<<<<<<<<<<<<
  "author" author format.feld
  "title"  title  format.feld
  end.entry                           %%%%% <<<<<<<<<<<<<
}

...
```

Hier sind die notwendigen Änderungen an den LaTeX-Markup-Anweisungen:

```
% erw4.tex
\documentclass[a4paper]{article}
\newenvironment{entry}[2]{\bibitem[#1]{#2} ‘‘#2’’:%
  %             ^^^^^--^-----------^^------ <<<<<<<<<<
  \begin{description}%
    \renewcommand\author[1]{\item[Autor:] ##1.}%
    \renewcommand\title [1]{\item[Titel:] ##1.}%
}{%
  \end{description}
}
\begin{document}
\nocite{*}
\bibliography{simple}
\bibliographystyle{erw4}
\end{document}
```

9 Erweiterung #5: `default.type`

Unsere bisherige Literaturdatenbank wird nun erweitert:

1. Neues Feld namens `year`.
2. Neue Eintragstypen `report` etc.

```
% erw5.bib
@book{eins,
  author = {Bernd Raichle},
  title  = {Bib\TeX-Programmierung},
  year   = 2001
}

@book{zwei,
  author = {A.U. Thor},
  title  = {Das Buch},
  year   = {2000*B}
}

@report{drei,
  author = {Un Bekannt and Ohne Namen and Name Nslos},
  titel  = {Ein Bericht}
}
```

Eintragstypen, für die keine spezielle Funktion definiert wurde, bei denen man aber einen Fehlerabbruch durch BibTEX vermeiden will, kann man mit der speziellen Funktion `default.type` verarbeiten.

```
% erw5.bst

...

FUNCTION {default.type}
{ "??" begin.entry
  "author" author format.feld
  "title"  title  format.feld
  end.entry
}

...
```

BibTEX gibt für diese undefinierten Eintragstypen eine entsprechende Warnung aus:

```
% erw5.blg
This is BibTeX, Version 0.99c (Web2C 7.3.1)
The top-level auxiliary file: erw5.aux
The style file: erw5.bst
Database file #1: erw5.bib
Warning--entry type for "drei" isn't style-file defined
--line 13 of file erw5.bib
(There was 1 warning)
```

10 Erweiterung #6: Weiteres Feld `year`

Folgende Änderungen sind notwendig, um das neue Feld `year` ebenso zu verarbeiten:

```
% erw6.bst
ENTRY
  { author
    title
    year                              %%%%% <<<<<<<<<<<<<
  }
  {}
  {}
...
FUNCTION {format.author.title.year}  %%%%% <<<<<<<<<<<<<
{ "author" author format.feld
  "title"  title  format.feld
  "year"   year   format.feld
}

FUNCTION {book}
{ "Buch" begin.entry
  format.author.title.year           %%%%% <<<<<<<<<<<<<
  end.entry
}

FUNCTION {default.type}
{ "??" begin.entry
  format.author.title.year           %%%%% <<<<<<<<<<<<<
  end.entry
}
...
```

Parallel dazu werden die LaTeX-Markup-Anweisungen ergänzt:

```
% erw6.tex
\documentclass[a4paper]{article}
\newenvironment{entry}[2]{\bibitem[#1]{#2} ‘‘#2’’:%
  \begin{description}%
    \renewcommand\author[1]{\item[Autor:] ##1.}%
    \renewcommand\title [1]{\item[Titel:] ##1.}%
    \renewcommand\year  [1]{\item[Jahr:] ##1.}%
    \newcommand\sortkey [1]{\item[Sortierschl\"ussel:] ##1.}%
}{%
  \end{description}
}
\begin{document}
\nocite{*}
\bibliography{erw5}
\bibliographystyle{erw6}
\end{document}
```

11 Erweiterung #7: Sortierung nach Feld `year`

BibTEX verwendet zum Sortieren der eingelesenen Literatureinträge das für jeden Eintrag vordefinierte Feld `sort.key$` vom Typ Zeichenkette. Dieses Feld muß *vor* dem Sortierlauf durch einen geeigneten Sortierschlüssel besetzt werden:

```
% erw7sort.bst

...

FUNCTION {presort}                                %%%%% <<<<<<<<<<<<<
{ year
  duplicate$ empty$     % ist Jahr-Feld vorhanden und nicht-leer?
    { pop$ "" }
    'skip$
  if$
  " " *            % sicherstellen, dass mind. 1 Zeichen vorhanden
  purify$          % nur alphanumerischen Zeichen und Leerzeichen
  "l" change.case$ % alles in Kleinbuchstaben wandeln
  #1 entry.max$ substring$    % zu langen String abschneiden
  'sort.key$ :=
}

...

READ
ITERATE {presort}                                 %%%%% <<<<<<<<<<<<<
SORT                                              %%%%% <<<<<<<<<<<<<
EXECUTE {begin.bib}
ITERATE {call.type$}
EXECUTE {end.bib}
```

Um zum Experimentieren die errechneten Sortierschlüssel `sort.key$` sichtbar zu machen, kann man sich folgender Änderungen bedienen:

```
% erw7sort.bst
...


FUNCTION {book}
{ "Buch" begin.entry
  "sortkey" sort.key$ format.feld     %%%%% <<<<<<<<<<<<<
  format.author.title.year
  end.entry
}

FUNCTION {default.type}
{ "??" begin.entry
  "sortkey" sort.key$ format.feld     %%%%% <<<<<<<<<<<<<
  format.author.title.year
  end.entry
}

...
```

In der Datei `erw6.tex` der vorigen Erweiterung ist bereits LaTeX-Markup für den Sortierschlüssel definiert.

12 Erweiterung #8: Formatieren von Namen

```
% erw8names.bst

...

FUNCTION {format.names}            %%%%% <<<<<<<<<<<<<
{ duplicate$ empty$
    { pop$ pop$ }
    { format.namelist }
  if$
}

FUNCTION {format.author.title.year}
{ "author" author format.names     %%%%% <<<<<<<<<<<<<
  "title"  title  format.feld
  "year"   year   format.feld
}

...
```

Die neue Funktion `format.namelist` geht in einer Schleife über die Liste der Namen, die in der Zeichenkette des Literaturdatenbankeintrags mit dem Schlüsselwort „and“ aneinan-

dergereiht vorliegen.

`num.names$` liefert die Anzahl der Namen in einer Zeichenkette, `format.name$` greift den x-ten Namen heraus und formatiert ihn gemäß der Formatiervorgabe.

```
% erw8names.bst

...

STRINGS { namelist tag }            %%%%% <<<<<<<<<<<<<
INTEGERS { numnames }               %%%%% <<<<<<<<<<<<<

FUNCTION {format.namelist}          %%%%% <<<<<<<<<<<<<
{ 'namelist :=
  'tag :=
  namelist num.names$ 'numnames :=
  { numnames #0 > }
  { tag
    namelist numnames "{ff~}{vv~}{ll}{, jj}" format.name$
    format.feld
    numnames #1 - 'numnames :=
    }
  while$
}

...
```

Anmerkung: Zur Vereinfachung des vorgestellten Codes geht die gezeigte Funktion die Namensliste von hinten nach vorne durch.

13 Erweiterung #8a: Variante

Statt einer Variablen kann der Stapel häufig zur Speicherung verwendet werden, hier zur Speicherung des LaTeX-Markup-Tag „author“:

```
% erw9names.bst

...

STRINGS { namelist }
INTEGERS { numnames }

FUNCTION {format.namelist}
{ 'namelist :=
  namelist num.names$ 'numnames :=
  { numnames #0 > }
  { duplicate$   % LaTeX-Tag "..." verdoppeln
      namelist numnames "{ff~}{vv~}{ll}{, jj}" format.name$
      format.feld
    numnames #1 - 'numnames :=
    }
  while$
  pop$           % LaTeX-Tag "..." entfernen
}

...
```

14 LaTeX ist nicht alles: Generierung von XML

Man kann BibTeX nicht nur dazu verwenden, um eine `bbl`-Datei mit LaTeX-Umgebungen und -Anweisungen zu erzeugen. Folgendes Beispiel zeigt einen *un*vollständigen Ansatz, um aus einer Literaturdatenbank eine XML-Datei zu erzeugen.

```
% simplexml.bst
ENTRY
  { author
    title
  }
  {}
  {}

FUNCTION {book}
{ "<book>" write$ newline$
  "  <author>" author * "</author>" * write$ newline$
  "  <title>" title "</title>" * * write$ newline$
  "</book>" write$ newline$
}

READ
ITERATE {call.type$}
```

```
% simplexml.bbl
<book>
  <author>Bernd Raichle</author>
  <title>Bib\TeX-Programmierung</title>
</book>
<book>
  <author>A.U. Thor</author>
  <title>Das Buch</title>
</book>
```

Um aus diesem sehr einfachen Beispiel, wie man Einträge einer Literaturdatenbank mit XML-Markup versehen kann, eine vollständige Lösung zu machen, sind jedoch noch sehr viel mehr Dinge notwendig: Beispielsweise sollten Umlaute, andere Sonderzeichen als auch alle in der Literaturdatenbank verwendeten TEX-Eingabenotationen in ein entsprechendes XML-Äquivalent umgesetzt werden. BibTEX kennt leider keinerlei einfachen Textersetzungsanweisungen. Stattdessen muss man diese mit den vordefinierten Funktionen wie `substring$` und Schleifen selbst implementieren.

Um TEX-Markup in entsprechenden XML-Markup umzusetzen, ist etwas mehr Aufwand notwendig. Hier ein Beispiel, wie das TEX-Logo `\TeX` ersetzt werden könnte:

```
% stringmanip.bst

...

INTEGERS {len}
STRINGS {string newstring}

FUNCTION {substexlogo}
{
  'string :=
  "" 'newstring :=
  string text.length$ 'len :=  % Laenge des Strings
  { len #0 > }
    { string #1 #4 substring$
      "\TeX" =       % Vergleiche erste 4 Zeichen mit "\TeX"
        { newstring "&TeX;" *      'newstring :=
          string #5 len substring$ 'string :=
        }
        { newstring string #1 #1 substring$ * 'newstring :=
          string #2 len substring$ 'string :=
        }
      if$
      string text.length$ 'len :=  % Laenge des Strings neu berechnen
    }
  while$
  newstring
}
```

Die Funktion `substexlogo` zeigt eine mögliche Lösung, wie man in BibTEX nach einer Zeichenkette suchen und diese durch eine andere Zeichenkette ersetzen kann. Ich habe mich dabei auf die Zeichenkette „`\TeX`“, bestehend aus vier Zeichen, beschränkt, die durch das XML-Literal „`&TeX;`“ ersetzt wird.

Mit der Funktion `substring$` ist es leider nicht möglich, eine exakt vorher festlegbare Anzahl von Zeichen als Teilzeichenkette zu erhalten, da das Längen-Argument der Funktion nicht die Zahl der Zeichen, sondern die geforderte Zahl von aufeinanderfolgenden Zeichen, die ungleich dem Leerzeichen sind, angibt. Somit erhält man häufig eine längere Teilzeichenkette als Ergebnis. Daher habe ich obige Such- und Ersetze-Funktion so geschrieben, dass jeweils zeichenweise durch die Zeichenkette gewandert wird, dabei immer ein einzelnes Zeichen abgeschnitten und mit diesem eine neue Ergebniszeichenkette zusammengebaut wird.

Um die Funktion `substexlogo` perfekt zu machen, müsste man die Funktion noch dahingehend erweitern, dass die Ersetzung nur dann erfolgt, wenn nach der Zeichenkette „`\TeX`“ ein Zeichen folgt, das nicht ein Buchstabe ist.

```
% stringmanip.bst ... continued

FUNCTION {book}
{ "<book>" write$ newline$
  "  <author>" author substexlogo * "</author>" * write$ newline$
  "  <title>" title substexlogo "</title>" * * write$ newline$
  "</book>" write$ newline$
}

READ
ITERATE {call.type$}
```

... und hier sieht man das entstandene XML-Literaturverzeichnis:

```
% stringmanip.bbl
<book>
  <author>Bernd Raichle</author>
  <title>Bib&TeX;-Programmierung</title>
</book>
<book>
  <author>A.U. Thor</author>
  <title>Das Buch</title>
</book>
```

Literatur

[1] Oren Patashnik: BibTEXing. Dokumentation von BibTEX für Autoren, Datei „`btxdoc.tex`“. 8. Februar 1988.

[2] Oren Patashnik: Designing BibTEX Styles. Teil der BibTEX-Dokumentation,

enthält Beschreibung der Programmiersprache von BibTEX-Style-Dateien, Datei „`btxhak.tex`“. 8. Februar 1988.