Einführung in die BibTEX-Programmierung

Bernd Raichle

Herbsttagung 2005, Kiel

Gliederung

Ein erstes einfaches Beispiel

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.

Ein erstes einfaches Beispiel

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}
}
```

Ein erstes einfaches Beispiel
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}
```

Ein erstes einfaches Beispiel
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}
```

Ein erstes einfaches Beispiel

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$}
```

Ein erstes einfaches Beispiel

BibTEX-Style „simple.bst“

```
% 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.

Aufbau von BibTEX-Styles

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$`.

Aufbau von BibTEX-Styles
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$`.

Aufbau von BibTEX-Styles
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$`.

Aufbau von BibTEX-Styles
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$`.

Aufbau von BibTEX-Styles
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.

Merke: `ENTRY` und `MACRO` *vor* `READ`
`READ` *vor* `SORT`, `ITERATE`, `REVERSE`
Funktionen definieren *vor* deren Aufruf
Variablen deklarieren *vor* dem Zugriff

Aufbau von BibTEX-Styles
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 { ... }
```

Datentypen, Variablen, vordefinierte Funktionen

Datentypen

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

Zeichenketten-Konstanten

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

Zahl-Konstanten

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

Datentypen, Variablen, vordefinierte Funktionen
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$`

Datentypen, Variablen, vordefinierte Funktionen

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 `$`.

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* -

Konkatenation von Zeichenketten

- *string string* *

Prädikate auf Zeichenketten

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

Literatureintragsdaten

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

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)

Kontrollfluß

- *integer literal literal* `if$`
- *literal literal* `while$`
- `call.type$` (Indirekter Aufruf)

Spezielle Stack-Operationen

- *string* `quote$`
- *literal* `duplicate$`
- *literal* `pop$`
- *literal literal* `swap$`
- *literal* `top$` (Debugging)
- *literal* * `stack$` (Debugging)

Ausgabe bbl-Datei

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

Protokollausgabe

- *string* `warning$`

Sonstige

- `skip$` (No-Op)

UPN ist ungewohnt . . .
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.“)

UPN ist ungewohnt . . .
Infix-/Präfix-Notation anderer Programmiersprachen

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

Beispiele Infix-Notation

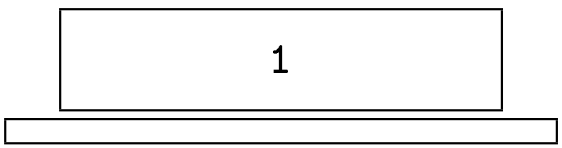
```
1 + 2
label := "a"
if 0 then print("gleich 0") else print("ungleich 0") endif
```

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"))
```

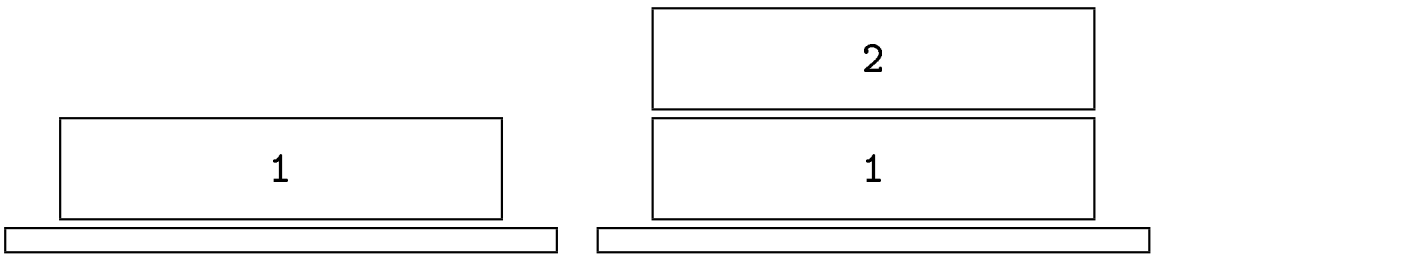
UPN ist ungewohnt . . .
UPN: Manipulation eines Stapels




```
#1 #2 +
```

UPN ist ungewohnt . . .

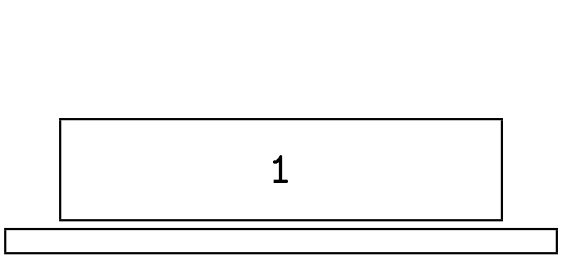
UPN: Manipulation eines Stapels

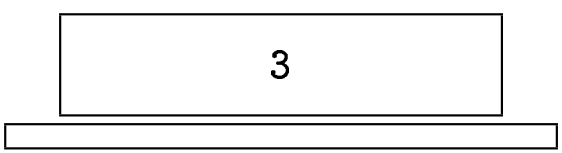



```
#1 #2 +
```

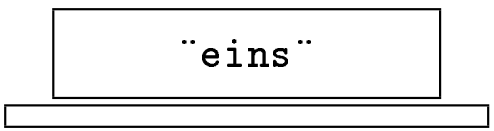
UPN ist ungewohnt . . .

UPN: Manipulation eines Stapels

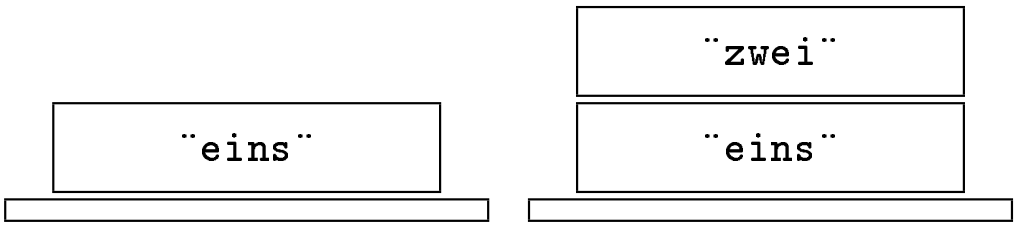





```
#1 #2 +
```

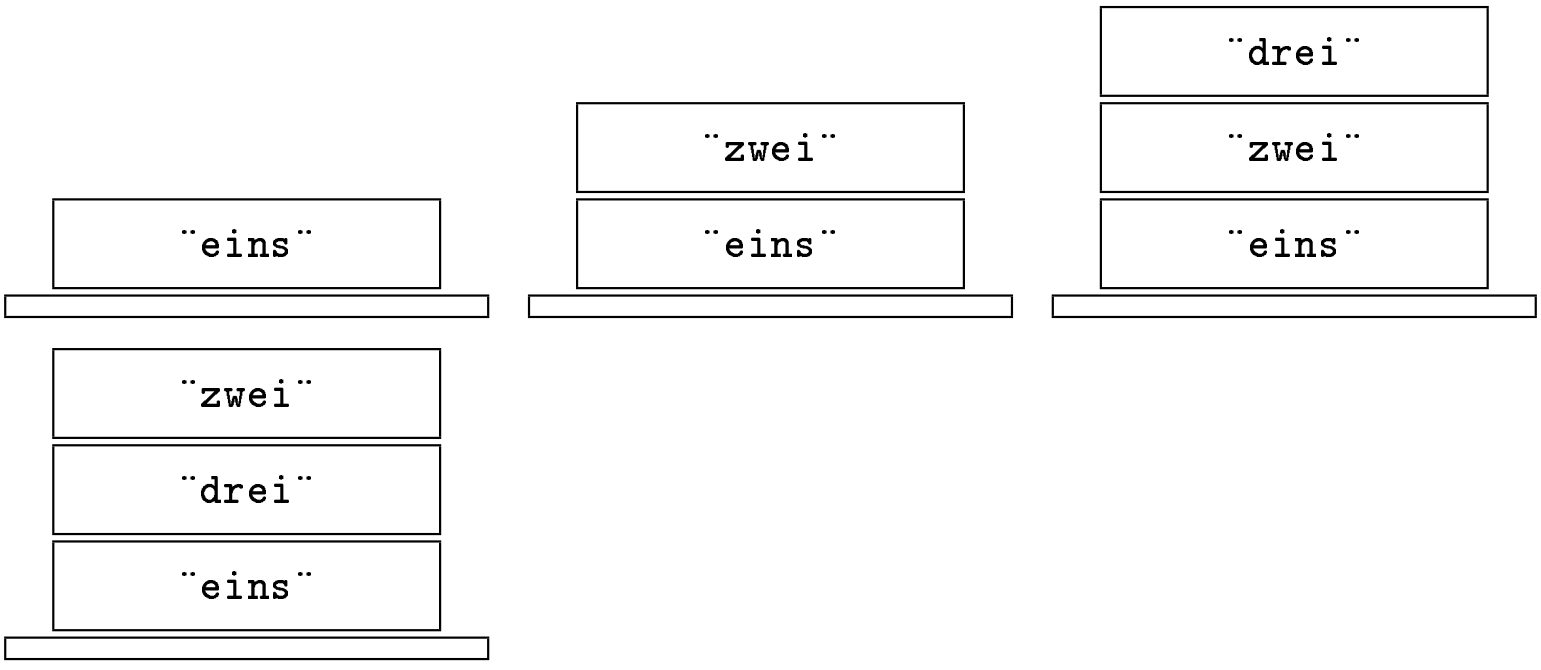



```
"eins" "zwei" "drei" swap$ * swap$ * pop$
```

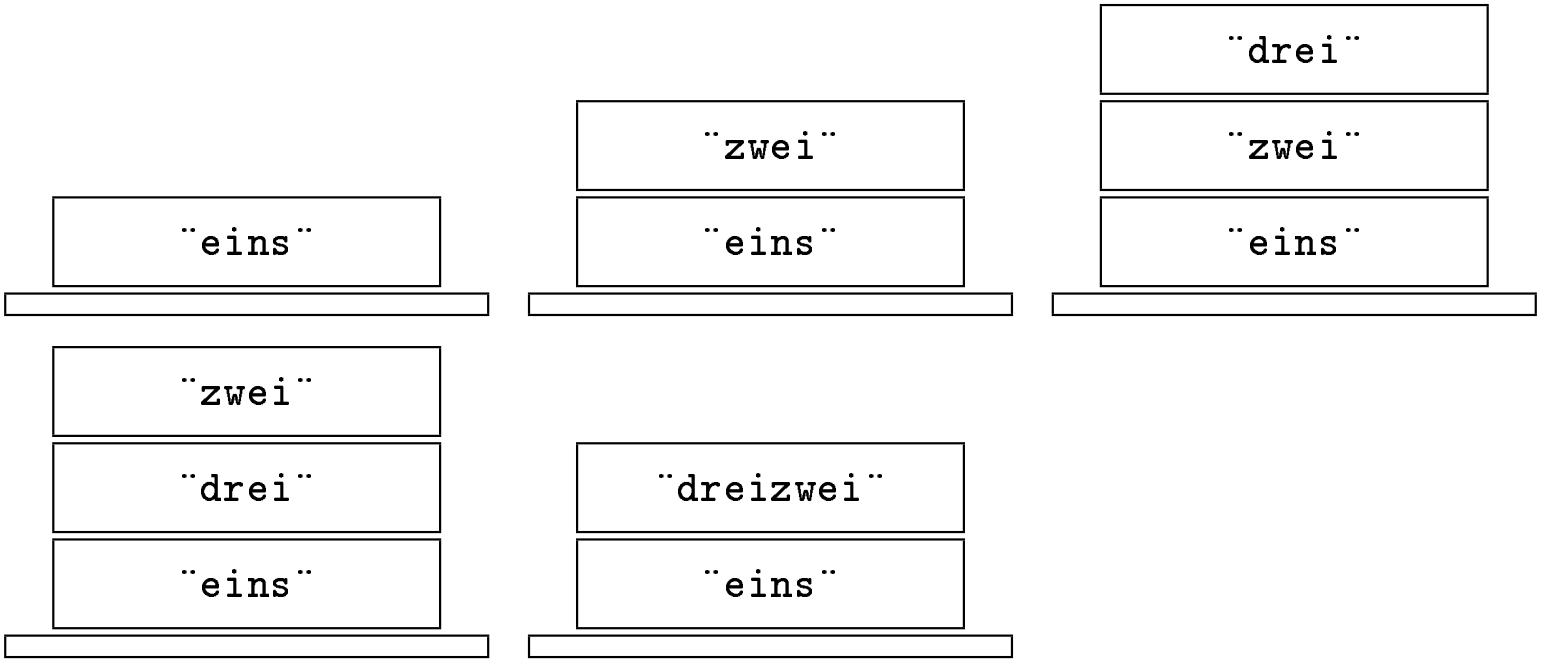



```
"eins" "zwei" "drei" swap$ * swap$ * pop$
```

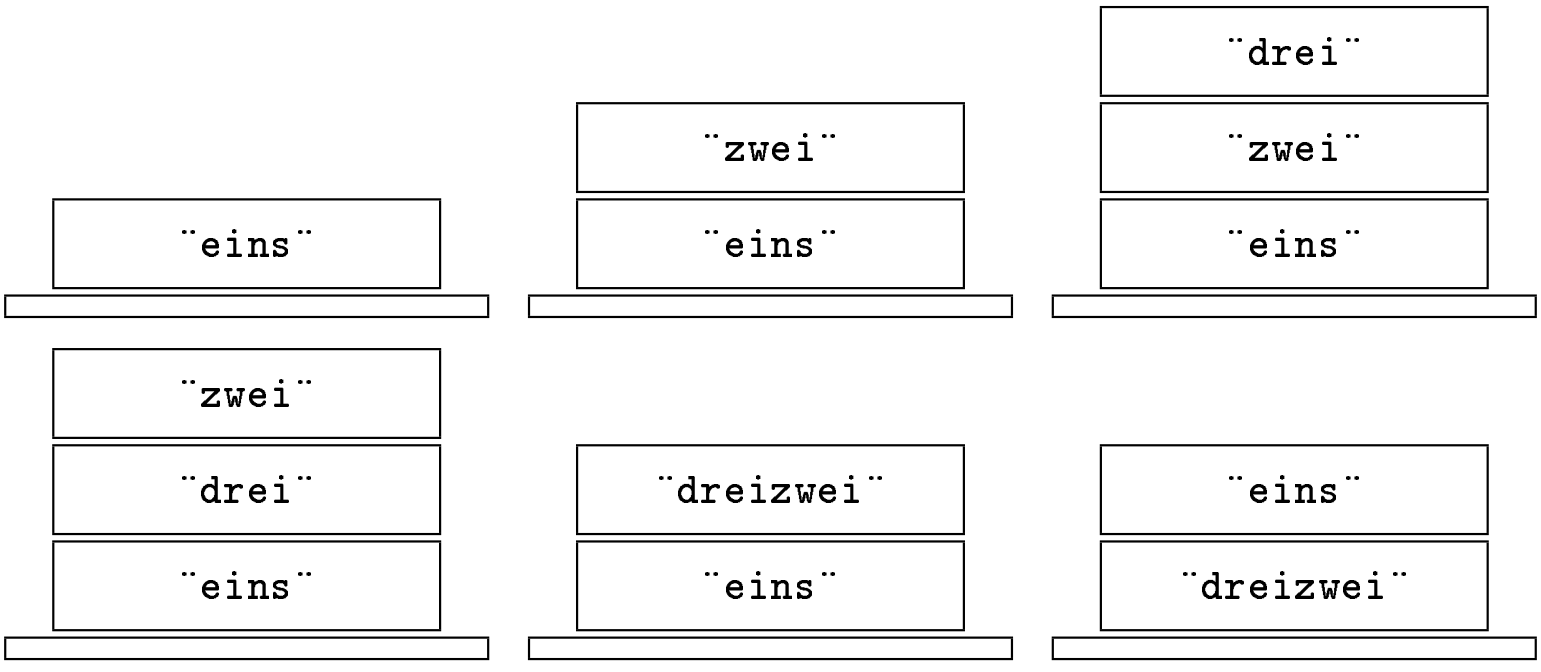

```
"eins" "zwei" "drei" swap$ * swap$ * pop$
```



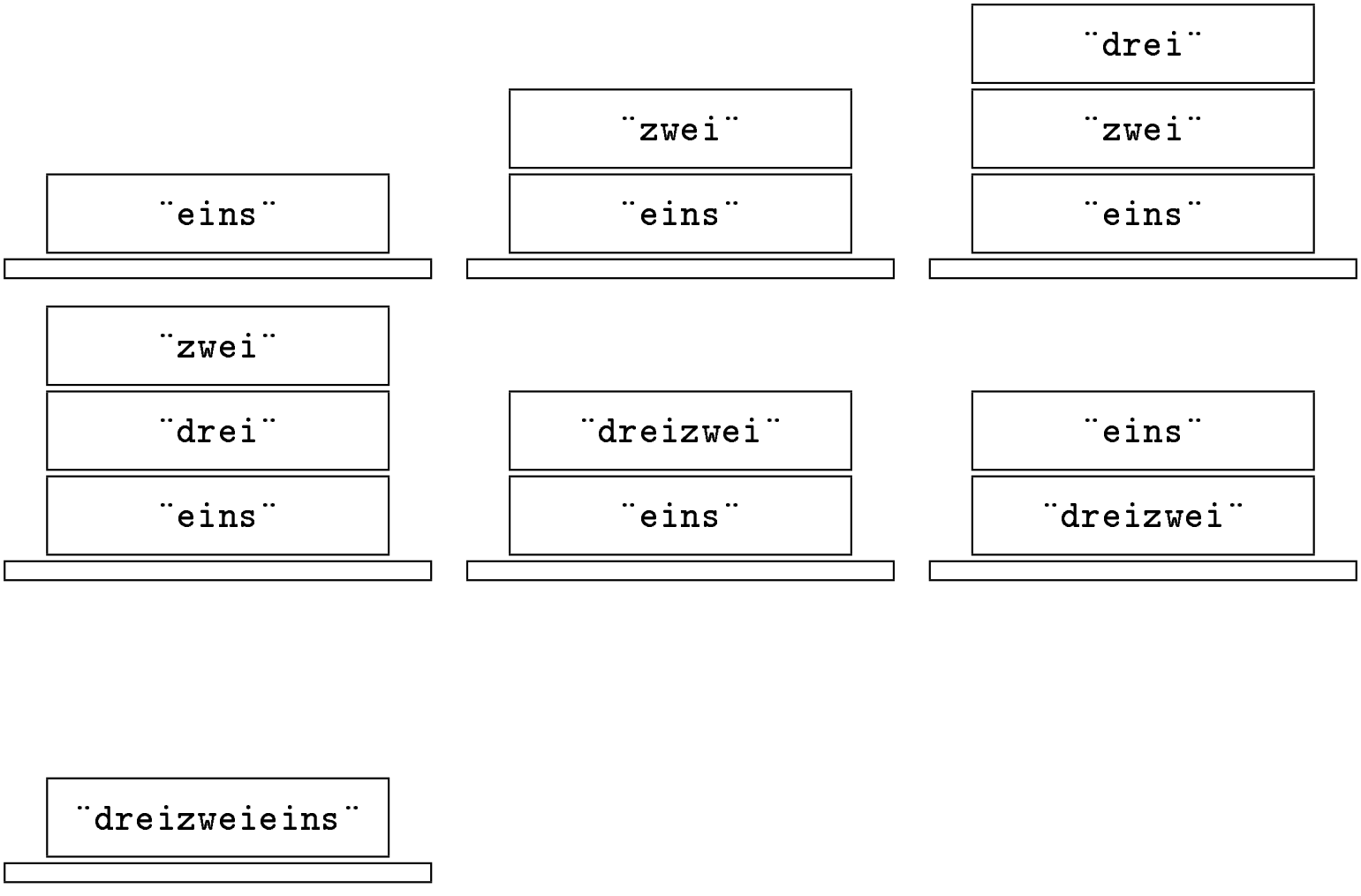

```
"eins" "zwei" "drei" swap$ * swap$ * pop$
```



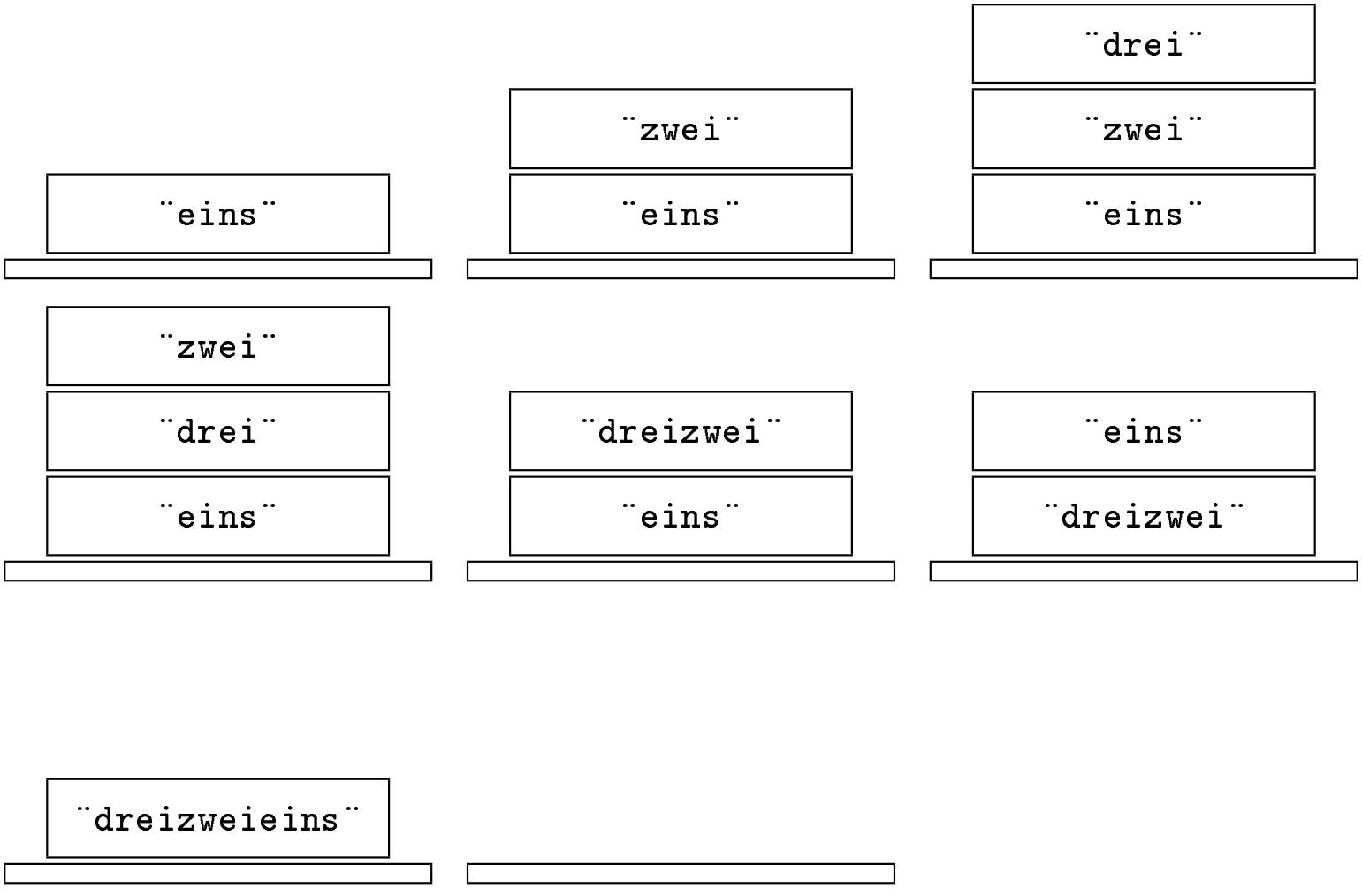

```
"eins" "zwei" "drei" swap$ * swap$ * pop$
```




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




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




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

UPN ist ungewohnt . . .

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

```
% 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}
```

Erweiterungen

#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$}
```

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.

Erweiterungen

#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$}
```

Erweiterungen

#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}                       %%%%% <<<<<<<<<<<<
```

Erweiterungen
#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}
```

Erweiterungen
#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)
```

Erweiterungen
#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}
```

Erweiterungen
#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.

Erweiterungen
#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“ aneinandergereiht 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.

Erweiterungen
#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
}

...
```

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$}
```

LATEX ist nicht alles: Generierung von XML

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

Danke für die Aufmerksamkeit!

Und viel Spaß bei eigenen Experimenten.

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

Oren Patashnik: Designing BibTEX Styles. Teil der BibTEX-Dokumentation, enthält Beschreibung der Programmiersprache von BibTEX-Style-Dateien, Datei „`btxhak.tex`“. 8. Februar 1988.