Hoe met LaTeX een boek kan worden gemaakt

Rein Smedinga

Vakgroep Informatica
Rijksuniversiteit Groningen
Postbus 800
9700 AV Groningen

rein@cs.rug.nl

Abstract

Het creëren van een boek met LaTeX is geen vanzelfsprekende bezigheid. Onderstaand het relaas van zo'n poging: het boek-in-wording *Inleiding Informatica* dat in eerste instantie als dictaat bij het bijbehorende college is geschreven en nu in een boekversie is aangeboden aan Addison-Wesley.
In onderstaand betoog wordt voorbij gegaan aan het belangrijkste onderdeel van het schrijven van een boek: het schrijven van de tekst. We zullen het hier slechts hebben over de problemen en oplossingen voor wat betreft de layout.

1 De stijl

Addison-Wesley kent een aantal regels waaraan de layout van een boek moet voldoen. Ondanks het feit dat Addison-Wesley de hoofduitgever is voor wat betreft TeX-materiaal, kent de Nederlandstalige tak (waarvoor het boek bedoeld is) geen mogelijkheid tekst langs electronische wijze aan te bieden. Nodig is dus een camera-ready manuscript met een layout overeenkomstig de eisen en regels van de uitgever .

Het boek *Inleiding Informatica* is niet mijn eerste boek bij Addison-Wesley. Voorheen verscheen *Simulatie en Implementatie*. Voor dat boek heb ik de layout al eens moeten aanpassen aan de eisen van de uitgever. Dit heb ik gedaan door uit te gaan van de LaTeX-stijl `book` en deze aan te passen. Belangrijkste aanpassingen zijn de gewijzigde hoofdstuktitels (cursief, nummer erboven, rechts aangelijnd, zonder het woord hoofdstuk), geen extra lege regels tussen opsommingen, terwijl alle indentaties (nieuwe paragraaf, items in `itemize` en `enumerate`, formules) gelijk zijn. Verder dienden de sectie-titels te worden aangepast (font, ruimte eromheen e.d.), de titels van figuren links aangelijnd te zijn (niet gecentreerd) en waren bepaalde waarden opgegeven voor tekstbreedte en -hoogte. Eén en ander bleek redelijk eenvoudig in de files `book.sty` en `bk12.sty` aan te passen. Zo ontstonden `awbook.sty`, `awbk12.sty` en later ook `awbk11.sty` voor eigen gebruik bij dictaten.

Naast `awbook` zijn nog een aantal aanvullende `sty`-files vereist. We noemen `fleqn` (om formules links te beginnen in plaats van gecentreerd), `which` (zie verderop), `pict` (de file `pict.sty` bevat een groot aantal macros voor het tekenen van eindige automaten. Een aantal daarvan zijn ook nuttig voor tekeningen in dit boek), `idxans` (zie verderop), `inlmacros` (alle macros, die voor het boek nodig zijn, zoals vetgedrukte **en**, stelling en opgave-omgevingen e.d.) en `program2` (de zelfontworpen programma-omgeving waarmee PASCAL-programma's elegant kunnen worden gelayout, zie verderop).

2 De indeling

Zoals Lamport al aangeeft in zijn boek over LaTeX is het verstandig een boek op te delen in hoofdstukken en per hoofdstuk een file aan te maken. Hiervoor geschikt zijn de macros `\include` en `\includeonly`. De laatste heeft als parameter de naam van de file die verwerkt moet worden. Van alle ge-`\include`-de files wordt wel de betreffende `.aux` file gelezen, maar verwerking van de tekst vindt alleen plaats als de filenaam ook als parameter in `\includeonly` meegegeven is. Zo kan inderdaad netjes hoofdstuk voor hoofdstuk worden afgewerkt zonder problemen te hebben met paginanummering, cross-referenties e.d.

Het boek is dan ook onderverdeeld in een hoofdfile `dictaat.tex` (Het boek bestond al als dictaat voordat het een boek werd!) waarin de nodige macro-files worden aangeroepen en verder slechts aanroepen staan van `\include` (voor elk hoofdstuk één). De tekst

van de hoofdstukken is vervolgens terug te vinden in de files `ch00.tex` tot en met `ch18.tex`. Aangezien later hoofdstukken zijn toegevoegd bestaan ook de files `ch11b.tex` en `ch14b.tex`. Ook voorwoord, inhoud, index, e.d. hebben een eigen file, maar daarover later meer.

3 \which

Voor eerdere boeken (en artikelen met hoofdstukken) is al eens de macro `\which` ontworpen, die in grote trekken overeenkomt met de standaard macro `\includeonly` met dat verschil, dat het argument van de laatste interactief kan worden opgegeven. Bovendien worden alle files verwerkt, indien de vraag wordt beantwoord met een return. De macro `\which` is alsvolgt gedefinieerd:

```
\def\which{\typeout{%
Give file(s) to be processed.}
\typein[\file@name]{%
(without .tex, separated by commas,
return for all files)}
\ifx\@empty\file@name \else
\includeonly{\file@name} \fi}
```

De `\ifx` vergelijkt de ingevoerde filenaam met de lege macro en roept alleen `includeonly` aan indien de filenaam niet leeg is. Is de naam wel leeg, dan volgt geen aanroep van `\includeonly` en worden dus alle files verwerkt. `\include` heeft dan ongeveer dezelfde betekenis als `\input`.

4 \include

De macro `\include` heeft als grote bezwaar, dat bijvoorbeeld de index steeds terechtkomt in de file `dictaat.idx`. Als we een hoofdstuk verwerkt hebben en aan een nieuw hoofdstuk beginnen worden de oude gegevens in `dictaat.idx` overschreven door de gegevens uit het nieuwe hoofdstuk.

Om dit op te lossen is de macro `\include` vervangen door de macro `\Include` die per hoofdstukfile een `.idx` file aanmaakt. Bij verwerking van een volgend hoofdstuk blijven de gegevens van het vorige hoofdstuk dus netjes bewaard. Voor de index betekent dat dan dat (afgezien van sorteren e.d. van de index-items) we alleen nog maar de files `ch00.idx` tot en met `ch18.idx` achterelkaar behoeven te zetten en te verwerken. Bij gebruik van de oude `\include` moet de laatste keer het hele boek worden ge-LaTeX-ed om een lijst met index-items te verkrijgen voor het gehele boek.

Het boek bevat verder veel opgaven en het leek handig de antwoorden van de opgaven direct in de tekst erbij te kunnen typen en LaTeX ervoor te laten zorgen dat deze naar een aparte file worden geschreven om later in één keer weer te worden ingelezen. We hebben hiervoor gebruik gemaakt van de macro `\answer` zoals beschreven in het TeXbook van Knuth:

```
\newwrite\ans
\immediate\openout\ans=\jobname.ans
\outer\def\answer{\immediate\write\ans{}
\immediate\write\ans{\string\antwoord{\theopg}}
\copytoend}
\def\copytoend{\begingroup\setupcopy\copyans}
\def\setupcopy{\catcode‘\\=12 \catcode‘\{=12
\catcode‘\}=12 \catcode‘\$=12 \catcode‘\&=12
\catcode‘\#=12 \catcode‘\%=12 \catcode‘\~=12
\catcode‘\_=12 \catcode‘\^=12 \catcode‘\ =12
\catcode‘\ =12 \obeylines}
{\obeylines\gdef\copyans#1
{\def\next{#1}%
\ifx\next\empty\let\next=\endgroup%
\else\immediate\write\ans{\next}
\let\next=\copyans\fi\next}}
```

Voordeel van deze methode is verder dat de nummers van de opgaven (deze staan in een counter met de naam `opg`) direct beschikbaar zijn (als `\theopg`) en niet met behulp van `\label` en `\ref` opgeslagen behoeven te worden (bovendien spaart dit geheugenruimte en verkleint het de kans op de mededeling dat de capaciteit van TeX overgeschreden is).

Gebruik van de macro `\answer` eist de nodige zorgvuldigheid: alle tekst tot de eerstvolgende lege regel wordt naar de antwoordenfile weggeschreven.

Evenals bij de index willen we graag per hoofdstuk een antwoordenfile (met extentie `.ans`). De macro `\Include` ziet er dan alsvolgt uit:

```
\def\Include#1{\clearpage
\if@filesw \immediate \write \@mainaux
{\string \@input {#1.aux}}\fi
\@tempswatrue \if@partsw \@tempswafalse
\def \@tempb {#1}\@for
\@tempa :=\@partlist \do {\ifx \@tempa
\@tempb \@tempswatrue \fi }\fi
\if@tempswa \if@filesw
\let \@auxout =\@partaux
\immediate \openout \@partaux #1.aux
\immediate \write \@partaux {\relax }\fi
\immediate \closeout\@indexfile
\immediate \openout \@indexfile #1.idx
\immediate \closeout\ans
\immediate \openout \ans #1.ans
\@input {#1.tex}\clearpage
\@writeckpt {#1}\if@filesw
\immediate \closeout \@partaux \fi
\let \@auxout =\@mainaux \else
\@nameuse {cp@#1}\fi }
```

Het verwerken van één hoofstuk heeft nu tot gevolg dat voor dat hoofdstuk een file onstaat met de bijbehorende index-items en met de bij de opgaven uit dat hoofdstuk behorende antwoorden.

De macros `\Include` en `\answer` zijn uiteindelijk samen gezet in de file `idxans.sty`, die nu binnen onze vakgroep voor algemeen gebruik beschikbaar is.

Voor het afzonderlijk bekijken van de antwoorden van één hoofdstuk (en de index) is er een tweede mainfile gemaakt: `answer.tex`. Deze werkt weer gewoon met `\include` en geeft de mogelijkheid (m.b.v.

`\which`) de antwoorden en de index-items per hoofdstuk te verwerken. Vooral aan het begin is dit handig omdat dan niet alle antwoorden ge-LaTeX-ed behoeven te worden en steeds locaal per hoofdstuk kan worden gewerkt. De index-items worden in volgorde van optreden in de tekst genoteerd. Hiervoor is in `answer.tex` een macro `\itementry` gedefinieerd (alle indexitems komen standaard voorafgegaan door deze macro-aanroep in de `.idx`-file, dit doet LaTeX) die een item in de indexlijst oplevert. Deze indexverwerking gaat dus buiten `makeindex` (zie verderop) om.

5 De index

Voor het maken van de index kon gelukkig op het laatste moment gebruik worden gemaakt van het programma `makeindex`. Deze heeft een aantal handige faciliteiten, zoals het in de index opnemen van pagina-ranges waarin een term wordt uitgelegd:

```
\index{term|(}          % op beginpositie
\index{term|)}          % op eindpositie
```

het gebruiken van subitems en subsubitems:

```
\index{term!subterm!subsubterm}
```

het in de index plaatsen van verwijzigen naar een ander woord:

```
\index{term|see{other term}}
```

en het in andere stijlen (bold, cursief, in mathmode e.d.) opnemen van woorden zonder dat dit de sortering verstoort:

```
\index{en@{\bf en}}
```

Belangrijk is overigens al dat het programma de index-items sorteert en gelijke voorkomens bijelkaar veegt. Een heel plezierig programma. In ons geval is nu slechts nodig alle `.idx` files in één nieuwe file te zetten (met behulp van het UNIX commando `cat` bijvoorbeeld), `makeindex` hierop los te laten en het uiteindelijke resultaat nog eens door LaTeX te halen. Lange tijd is overigens besteed aan het opsporen van fouten in de index (vooral het terugzoeken van de bijbehorende macro-aanroep in de tekst) en het consistent krijgen van de index (geen woorden soms als item en soms als subitem opnemen, bijvoorbeeld). Klein probleem was verder nog dat de uiteindelijke file begint met `\begin{theindex}`. In LaTeX wordt standaard een index niet in de inhoud opgenomen. Dit kan wel door na `\begin{theindex}` een extra macro aan te roepen(nl. `\addcontentsline`), maar dat is in ons geval weer onmogelijk omdat we dan de file die `makeindex` oplevert steeds weer opnieuw eerst zelf zouden moeten editen. Een oplossing is een herdefinitie van de macro `\theindex` die uiteindelijk door `\begin{theindex}` wordt aangeroepen:

```
\let\oldindex\theindex
\def\theindex{\oldindex
\addcontentsline{toc}{chapter}{Index}}
\input{index.tex}
```

waarna de door `makeindex` geproduceerde file `index.tex` direct met het gewenste effect kan worden ingelezen. Voorts diende de verwijzing in de index aangepast te worden aan het Nederlands:

```
\def\see#1#2{{\it zie} #1}
       % #2 is pagenumber, to be omitted
```

6 Programma-omgeving

Oorspronkelijk stonden de grotere programma's in een `figure`-omgeving (één van de twee mogelijkheden zogenaamde zwevende tekst te krijgen. De ander is de `table`-omgeving), maar aardiger is het een speciale programma-omgeving te hebben: een zwevende tekst, waarin een programma gezet kan worden, die in de bijbehorende titel (de `caption`) ook netjes programma heet en niet figuur of zo. Hierin is LaTeX heel netjes, slechts een paar regels waren nodig om dit opgelost te krijgen:

```
\def\listofprogrammes{\@restonecolfalse\if@twocolumn\@restonecoltrue\onecolumn
 \fi\chapter*{\@PLijst}\@mrkr{\@PLijst}%
\@starttoc{lop}\if@restonecol\twocolumn
 \fi}
\def\l@programme{\@dottedtocline{1}{1.5em}{2.3em}}
\newcounter{programme}[chapter]  % new counter
\def\theprogramme{\thechapter.\@arabic\c@programme} % counter layout
\def\fps@programme{tbp}          %
\def\ftype@programme{3}          % figure has type 1, table type 2
\def\ext@programme{lop}          % listing in file .lop
\def\fnum@programme{\@Programme\ \theprogramme} % caption number layout
\def\programme{\@float{programme}} % the real definition
\let\endprogramme\end@float
\@namedef{programme*}{\@dblfloat{programme}} %idem for twocolumns
\@namedef{endprogramme*}{\end@dblfloat}
\def\@PLijst{Lijst van programma's} % title for list-chapter
```

Overigens moest hier gebruik worden gemaakt van de naam `programme` omdat `program` al een omgevingsnaam is (zie verderop).

7 Programma-layout

Er zijn in feite twee methoden om een (PASCAL-)programma(fragment) in de tekst op te nemen: een eenvoudige en een mooie. De eenvoudige manier is gebruik te maken van de `verbatim`-omgeving. De programma's verschijnen dan in teletype-font en letterlijk zoals ze zijn ingetikt. De mooie manier is gebruik te maken van vetgedrukte gereserveerde woorden, in math-mode geplaatste statements, in teletype geplaatste strings en alles volgens een strakke, van te voren vastgelegde layout. Mijn voorkeur ging al direct uit naar de tweede manier.

In eerste instantie verzon ik een groot aantal macro's om de gereserveerde woorden in boldface te krijgen zonder iedere keer dingen hoeven in te tikken zoals `{\bf program}`. Zo ontstonden de macro's `\PROGRAM` e.d. Verder maakte ik gebruik van de `tabbing`-omgeving om een nette layout te krijgen. Later bedacht ik, dat het verkrijgen van een nette layout wellicht meer geautomatiseerd kon door de `tabbing`-macros op te nemen in de gemaakte macros.

Aangezien een programma uit een aantal levels bestaat (elke procedure-aanroep creeërt een nieuw level en bij beëindiging van een procedure-body wordt weer teruggegaan naar het vorige level) was iets nodig als een stapelmechanisme om oude tab-settings te bewaren. Dit mechanisme is in TeX standaard aanwezig in de vorm van groups. Bij het verlaten van een binnengroup worden de waarden en definities van de omvattende group teruggezet. Hier heb ik dan ook driftig gebruik van gemaakt. Verder kan met behulp van `\+` en `\-` worden geregeld dan de eerste voorkomens van `\>` in een regel achterwege gelaten kunnen worden.

Op de een of andere manier heb ik gekozen voor een layout met de volgende eigenschappen:

1. locale declaraties en de body van een procedure zijn van een dieper niveau dan de procedureheading.
2. de programma-heading, de globale procedure-headings en het hoofdprogramma behoren tot het bovenste niveau.
3. de formele specificatie van een procedure (of functie) behoort tot hetzelfde niveau als de heading.
4. een dieper niveau springt in.
5. bij het herhalingsstatement staat de **do** onder de **while** indien het herhalingstatement een compound statement is. Evenzo voor **if**, **then** en **else** en vergelijkbare statements.
6. puntkomma's tussen statements in een compound-statement staan niet achteraan de regel, maar vooraan de volgende, onder de **b** van **begin**. De **e** van de afsluitende **end** staat hier weer onder.
7. declaraties over meerdere regels springen op volgende regels in.

Uiteindelijk resulteerde dit in een de verzameling macro's die is terug te vinden in de stijl `program2`.[1] Om niet onnodig $-tekens te hoeven tikken heb ik naast `tabbing`-omgeving een `mathtabbing`-omgeving gemaakt (`tabbing` in mathmode).

De uiteindelijke programma-stijl wordt op dit moment ook door andere leden van de vakgroep gebruikt. Een uitgebreide handleiding hiervoor is beschikbaar onder de naam `program.tex`.

Als illustratie een aantal fragmenten uit de `sty`-file. Het stapelmechanisme is geïmplementeerd door de macros `\@push` en `\@pop`. De teller `\@tabs` geeft het aantal extra tabposities in de nieuwe omgeving aan. De teller `\@etabs` geeft het aantal extra terug te springen tabposities aan bij verlaten van het level. Bijvoorbeeld: een **end** komt te staan onder de **begin**, dat is in het te verlaten level `\@tabs` tabposities terug. In de regel na deze **end** moet er nogeens `\@etabs` posities worden teruggesprongen (het compound statement sprong naar alle waarschijnlijkheid al een aantal tabposities in). De macro's `\@stopf` en `\@contf` komen uit de `tabbing`-omgeving. De macro `\@test` is een debug-faciliteit en drukt de waarden van een aantal tellers af.

```
% \@pushXX creates following level and defines \@tabs to be XX
\def\@push#1{%\@test@{push}
\@stopf\global\advance\@level by1\relax \begingroup
 \@tabs#1\relax \@etabs=0 \relax\@contf}
% \@addXX adds the value XX to \@tabs
\def\@add#1{\@stopf\advance\@tabs by#1\relax\@contf}
%
%\@extraXX sets extra \- to be done by reentering this level
\def\@extra#1{%\@test@{extra}
\@stopf\advance\@etabs by#1\relax\@contf}
%
% \@pop returns one level and does the needed number of \- and \<
% it does \-\< as many times as mentioned in \@tabs and \- as mentioned in
% \@etabs
\def\@pop{%\@test@{pop}
\ifnum\@level<0\@warning{Negative level in program environment}
\else\@stopf\global\advance\@level by-1 \global\@tmp=\@tabs \@contf%
```

[1] Er is ook een stijl `program` waarin de mathematische mode ontbreekt.

```
\@ifatmargin
 \ifcase\@tmp\or\<\-\or\<\<\-\-\or\<\<\<\-\-\-\or\<\<\<\<\-\-\-\-\fi
\else \sp@c % mathspace
 \ifcase\@tmp\or\-\or\-\-\or\-\-\-\or\-\-\-\-\fi
\fi \@stopf
\endgroup  \global\@tmp=\@etabs \@etabs=0 \@contf%
\ifcase\@tmp\or\-\or\-\-\or\-\-\-\or\-\-\-\-\fi \fi }
```

Als voorbeeld het gebruik van `\@pop` en `\@push`
voor de **begin** en **end**-symbolen:

```
\def\@plusb{\@stopf\advance\@Begins by1\@contf}
\def\@minusb{\@stopf\advance\@Begins by-1\@contf}
\def\@begin@#1{\enddecl\endheading#1\=\+\@push1\@plusb}
\def\@end@#1{\@minusb
          \ifrecordcase\@pop#1\@pop\@stopf\recordcasefalse\@contf
          \else\@pop#1\fi\@stopf\recordfalse\@contf
          \ifbody\ifnum\@Begins=0\ifnum\@level>0
             \@stopf\bodyfalse\@contf \@pop \fi\fi\fi}
\def\BEGIN{\@begin@{\mbox{\bf begin }}}
\def\END{\@end@{\mbox{\bf end }}}
```

Vooral de `\END` is lastig. Een **end** kan zowel de afsluiter zijn van een **begin** (waarbij het verschil maakt of dit een gewoon compound statement is of een body, in het laatste geval sluit de **end** tevens de procedure declaratie af), als van een **record** (eventueel een variant record), als van een **loop** (in ModPas). Zonodig is van conditionals gebruik gemaakt om het onderscheid te kunnen maken.

Andere overwonnen moeilijkheden in deze omgeving zijn de indendatie van procedurebodies en locale declaraties, terwijl de regels tussen de heading en de locale declaraties gebruikt kan worden voor een formele specificatie en niet ingesprongen worden, de verschillende betekenis van bijvoorbeeld `\VAR` als aankondiging van een variabele-declaratie en als aankondiging van een locale var-parameter (in het laatste geval dienen geen tabsettings plaats te vinden, in het eerste (meestal) wel), de juiste indentatie voor compoundstatements achter een **do** van een herhalingsstatement, achter een **then**, enzovoorts.

Naast de macros zoals `\PROGRAM` bestaan er ook macro's `\Program` die een gereserveerd woord vetgedrukt afdrukken: **program**-tekst krijg je dan door
`\Program-tekst`
te tikken.

8 Ergenissen

De grootste ergenis bij het gebruik van LaTeX is het veelvuldig optreden van de mededeling
`TeX capacity exceeded`
als gevolg van het gebruik van de `picture`-omgeving in combinatie met een redelijke hoeveelheid cross-referenties. Een aantal keren heb ik de mededeling (en het afbreken van de verwerking) kunnen voorkomen door een aantal `aux`-files tijdelijk weg te halen (de cross-referentie-lijst werd dan minder lang) en een aantal keren door de plaatjes uit de tekst te halen (d.i. nog slechts een lege `picture`-omgeving laten staan) en in een aparte file het werkelijke plaatje onder te brengen. De bijbehorende plaatjes moeten dan wel met de hand worden ingeplakt.

Helaas wordt in LaTeX voor elk label een macrodefinitie gemaakt en deze wordt op dezelfde plaats opgeslagen als de nog niet ge-`\output`-te ingelezen tekst. Zeker bij plaatjes in een zwevende tekst omgeving loop je dan het risico de maximale hoeveelheid te gebruiken ruimte te overschrijden (overigens een kwestie van het integer-bereik van de gebruikte PASCAL-implementatie, is mij eens verteld).

9 Conclusie

Wie een boek schrijft en het manuscript camera-ready aanlevert haalt zich dus veel werk op de hals. Gebruik van TeX (en vooral LaTeX) heeft dit werk sterk vereenvoudigd (veel is al mogelijk of eenvoudig te maken en met behulp van `makeindex` is nog nooit zo snel en volledig correct een index gemaakt) en ook weer sterk bemoeilijkt (omdat het zo mooi kan, wil je het ook zo mooi mogelijk, wordt je verschrikkelijk precies en ga je alle mogelijkheden benutten).

Overigens, een ieder die iets van bovenstaande wil gebruiken (of gedetailleerder wil bekijken) kan mij een email sturen.